test Documentation

Release test

test

Dec 06, 2017

Contents

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 1

Biopython

1.1 Tutorial and Cookbook

Source of the materials: Biopython Tutorial and Cookbook (adapted)

CHAPTER 2

Introduction

2.1 What is Biopython?

The Biopython Project is an international association of developers of freely available Python (http://www.python.org) tools for computational molecular biology. Python is an object oriented, interpreted, flexible language that is becoming increasingly popular for scientific computing. Python is easy to learn, has a very clear syntax and can easily be extended with modules written in C, C++ or FORTRAN.

The Biopython web site (http://www.biopython.org) provides an online resource for modules, scripts, and web links for developers of Python-based software for bioinformatics use and research. Basically, the goal of Biopython is to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and classes. Biopython features include parsers for various Bioinformatics file formats (BLAST, Clustalw, FASTA, Genbank,...), access to online services (NCBI, Expasy,...), interfaces to common and not-so-common programs (Clustalw, DSSP, MSMS...), a standard sequence class, various clustering modules, a KD tree data structure etc. and even documentation.

Basically, we just like to program in Python and want to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and scripts.

2.2 What can I find in the Biopython package

The main Biopython releases have lots of functionality, including:

- The ability to parse bioinformatics files into Python utilizable data structures, including support for the following formats:
- Blast output – both from standalone and WWW Blast
- Clustalw
- FASTA
- GenBank
- PubMed and Medline

- ExPASy files, like Enzyme and Prosite
- SCOP, including 'dom' and 'lin' files
- UniGene
- SwissProt
- Files in the supported formats can be iterated over record by record or indexed and accessed via a Dictionary interface.
- Code to deal with popular on-line bioinformatics destinations such as:
- NCBI – Blast, Entrez and PubMed services
- ExPASy – Swiss-Prot and Prosite entries, as well as Prosite searches
- Interfaces to common bioinformatics programs such as:
- Standalone Blast from NCBI
- Clustalw alignment program
- EMBOSS command line tools -A standard sequence class that deals with sequences, ids on sequences, and sequence features.
- Tools for performing common operations on sequences, such as translation, transcription and weight calculations.
- Code to perform classification of data using k Nearest Neighbors, Naive Bayes or Support Vector Machines.
- Code for dealing with alignments, including a standard way to create and deal with substitution matrices.
- Code making it easy to split up parallelizable tasks into separate processes.
- GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.
- Extensive documentation and help with using the modules, including this file, on-line wiki documentation, the web site, and the mailing list.
- Integration with BioSQL, a sequence database schema also supported by the BioPerl and BioJava projects.

We hope this gives you plenty of reasons to download and start using Biopython!

2.3 About these notebooks

These notebooks were prepared on Python 3 for Project Jupyter 4+ (formely IPython Notebook). Biopython should be installed and available (v1.66 or newer recommended).

You can check the basic installation and inspect the version by doing:

```
In [1]: import Bio
        print(Bio.__version__)
```

```
1.66
```

Source of the materials: Biopython cookbook (adapted)

CHAPTER 3

Quick Start

This section is designed to get you started quickly with Biopython, and to give a general overview of what is available and how to use it. All of the examples in this section assume that you have some general working knowledge of Python, and that you have successfully installed Biopython on your system. If you think you need to brush up on your Python, the main Python web site provides quite a bit of free documentation to get started with (http://www.python.org/doc/).

Since much biological work on the computer involves connecting with databases on the internet, some of the examples will also require a working internet connection in order to run.

Now that that is all out of the way, let's get into what we can do with Biopython.

3.1 General overview of what Biopython provides

As mentioned in the introduction, Biopython is a set of libraries to provide the ability to deal with "things" of interest to biologists working on the computer. In general this means that you will need to have at least some programming experience (in Python, of course!) or at least an interest in learning to program. Biopython's job is to make your job easier as a programmer by supplying reusable libraries so that you can focus on answering your specific question of interest, instead of focusing on the internals of parsing a particular file format (of course, if you want to help by writing a parser that doesn't exist and contributing it to Biopython, please go ahead!). So Biopython's job is to make you happy!

One thing to note about Biopython is that it often provides multiple ways of "doing the same thing." Things have improved in recent releases, but this can still be frustrating as in Python there should ideally be one right way to do something. However, this can also be a real benefit because it gives you lots of flexibility and control over the libraries. The tutorial helps to show you the common or easy ways to do things so that you can just make things work. To learn more about the alternative possibilities, look in the Cookbook, the Advanced section, the built in "docstrings" (via the Python help command, or the API documentation) or ultimately the code itself.

3.2 Working with sequences

We'll start with a quick introduction to the Biopython mechanisms for dealing with sequences, the Seq object, which we'll discuss in more detail later.

Most of the time when we think about sequences we have in my mind a string of letters like 'AGTACACTGGT'. You can create such Seq object with this sequence as follows:

```
In [1]: from Bio.Seq import Seq
        my_seq = Seq("AGTACACTGGT")
        my_seq
```

```
Out[1]: Seq('AGTACACTGGT', Alphabet())
```

```
In [2]: print(my_seq)
```

```
AGTACACTGGT
```

```
In [3]: my_seq.alphabet
```

```
Out[3]: Alphabet()
```

What we have here is a sequence object with a *generic* alphabet - reflecting the fact we have *not* specified if this is a DNA or protein sequence (okay, a protein with a lot of Alanines, Glycines, Cysteines and Threonines!).

In addition to having an alphabet, the Seq object differs from the Python string in the methods it supports. You can't do this with a plain string:

```
In [4]: my_seq.complement()
```

```
Out[4]: Seq('TCATGTGACCA', Alphabet())
```

```
In [5]: my_seq.reverse_complement()
```

```
Out[5]: Seq('ACCAGTGTACT', Alphabet())
```

The next most important class is the **SeqRecord** or Sequence Record. This holds a sequence (as a **Seq** object) with additional annotation including an identifier, name and description. The **Bio.SeqIO** module for reading and writing sequence file formats works with SeqRecord objects, which will be introduced below and covered in more detail later.

This covers the basic features and uses of the Biopython sequence class. Now that you've got some idea of what it is like to interact with the Biopython libraries, it's time to delve into the fun, fun world of dealing with biological file formats!

3.3 A usage example

Before we jump right into parsers and everything else to do with Biopython, let's set up an example to motivate everything we do and make life more interesting. After all, if there wasn't any biology in this tutorial, why would you want you read it?

Since I love plants, I think we're just going to have to have a plant based example (sorry to all the fans of other organisms out there!). Having just completed a recent trip to our local greenhouse, we've suddenly developed an incredible obsession with Lady Slipper Orchids.

Of course, orchids are not only beautiful to look at, they are also extremely interesting for people studying evolution and systematics. So let's suppose we're thinking about writing a funding proposal to do a molecular study of Lady Slipper evolution, and would like to see what kind of research has already been done and how we can add to that.

After a little bit of reading up we discover that the Lady Slipper Orchids are in the Orchidaceae family and the Cypripedioideae sub-family and are made up of 5 genera: *Cypripedium*, *Paphiopedilum*, *Phragmipedium*, *Selenipedium* and *Mexipedium*.

That gives us enough to get started delving for more information. So, let's look at how the Biopython tools can help us. We'll start with sequence parsing, but the orchids will be back later on as well - for example we'll search PubMed for papers about orchids and extract sequence data from GenBank, extract data from Swiss-Prot from certain orchid proteins and work with ClustalW multiple sequence alignments of orchid proteins.

3.4 Parsing sequence file formats

A large part of much bioinformatics work involves dealing with the many types of file formats designed to hold biological data. These files are loaded with interesting biological data, and a special challenge is parsing these files into a format so that you can manipulate them with some kind of programming language. However the task of parsing these files can be frustrated by the fact that the formats can change quite regularly, and that formats may contain small subtleties which can break even the most well designed parsers.

We are now going to briefly introduce the **Bio.SeqIO** module – you can find out later. We'll start with an online search for our friends, the lady slipper orchids. To keep this introduction simple, we're just using the NCBI website by hand. Let's just take a look through the nucleotide databases at NCBI, using an Entrez online search (http://www.ncbi.nlm.nih.gov:80/entrez/query.fcgi?db=Nucleotide) for everything mentioning the text *Cypripedioideae* (this is the subfamily of lady slipper orchids).

When this tutorial was originally written, this search gave us only 94 hits, which we saved as a FASTA formatted text file and as a GenBank formatted text file (files ls_orchid.fasta and ls_orchid.gbk, also included with the Biopython source code under docs/tutorial/examples/).

If you run the search today, you'll get hundreds of results! When following the tutorial, if you want to see the same list of genes, just download the two files above or copy them from docs/examples/ in the Biopython source code. Below we will look at how to do a search like this from within Python.

3.4.1 Simple FASTA parsing example

If you open the lady slipper orchids FASTA file ls_orchid.fasta in your favourite text editor, you'll see that the file starts like this:

> >gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTG-GAATAAACGATCGAGTG AATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCC-CGTGGTGACCCTGATTTGTTGTTGGG ...

It contains 94 records, each has a line starting with ">" (greater-than symbol) followed by the sequence on one or more lines. Now try this in Python (printing the first 5 records):

```
In [6]: from Bio import SeqIO
        for seq_record in list(SeqIO.parse("data/ls_orchid.fasta", "fasta"))[:5]:
            print(seq_record.id)
            print(repr(seq_record.seq))
            print(len(seq_record))
```

```
gi|2765658|emb|Z78533.1|CIZ78533
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', SingleLetterAlphabet())
740
gi|2765657|emb|Z78532.1|CCZ78532
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC', SingleLetterAlphabet())
753
gi|2765656|emb|Z78531.1|CFZ78531
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA', SingleLetterAlphabet())
748
gi|2765655|emb|Z78530.1|CMZ78530
```

```
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAAACAACAT...CAT', SingleLetterAlphabet())
744
gi|2765654|emb|Z78529.1|CLZ78529
Seq('ACGGCGAGCTGCCGAAGGACATTGTTGAGACAGCAGAATATACGATTGAGTGAA...AAA', SingleLetterAlphabet())
733
```

Notice that the FASTA format does not specify the alphabet, so **Bio.SeqIO** has defaulted to the rather generic **SingleLetterAlphabet()** rather than something DNA specific.

3.4.2 Simple GenBank parsing example

Now let's load the GenBank file ls_orchid.gbk instead - notice that the code to do this is almost identical to the snippet used above for the FASTA file - the only difference is we change the filename and the format string:

```
In [7]: from Bio import SeqIO
        for seq_record in list(SeqIO.parse("data/ls_orchid.gbk", "genbank"))[:5]:
            print(seq_record.id)
            print(repr(seq_record.seq))
            print(len(seq_record))
```

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
740
Z78532.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC', IUPACAmbiguousDNA())
753
Z78531.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA', IUPACAmbiguousDNA())
748
Z78530.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAAACAACAT...CAT', IUPACAmbiguousDNA())
744
Z78529.1
Seq('ACGGCGAGCTGCCGAAGGACATTGTTGAGACAGCAGAATATACGATTGAGTGAA...AAA', IUPACAmbiguousDNA())
733
```

This time **Bio.SeqIO** has been able to choose a sensible alphabet, IUPAC Ambiguous DNA. You'll also notice that a shorter string has been used as the **seq_record.id** in this case.

3.4.3 I love parsing – please don't stop talking about it!

Biopython has a lot of parsers, and each has its own little special niches based on the sequence format it is parsing and all of that.

While the most popular file formats have parsers integrated into *Bio.SeqIO* and/or *Bio.AlignIO*, for some of the rarer and unloved file formats there is either no parser at all, or an old parser which has not been linked in yet. Please also check the wiki pages http://biopython.org/wiki/SeqIO and http://biopython.org/wiki/AlignIO for the latest information, or ask on the mailing list. The wiki pages should include an up to date list of supported file types, and some additional examples.

The next place to look for information about specific parsers and how to do cool things with them is in the Cookbook. If you don't find the information you are looking for, please consider helping out your poor overworked documentors and submitting a cookbook entry about it! (once you figure out how to do it, that is!)

3.5 Connecting with biological databases

One of the very common things that you need to do in bioinformatics is extract information from biological databases. It can be quite tedious to access these databases manually, especially if you have a lot of repetitive work to do. Biopython attempts to save you time and energy by making some on-line databases available from Python scripts. Currently, Biopython has code to extract information from the following databases:

- Entrez (and PubMed) from the NCBI
- ExPASy
- SCOP

The code in these modules basically makes it easy to write Python code that interact with the CGI scripts on these pages, so that you can get results in an easy to deal with format. In some cases, the results can be tightly integrated with the Biopython parsers to make it even easier to extract information.

3.6 What to do next

Now that you've made it this far, you hopefully have a good understanding of the basics of Biopython and are ready to start using it for doing useful work. The best thing to do now is finish reading this tutorial, and then if you want start snooping around in the source code, and looking at the automatically generated documentation.

Once you get a picture of what you want to do, and what libraries in Biopython will do it, you should take a peak at the Cookbook (Chapter 18), which may have example code to do something similar to what you want to do.

If you know what you want to do, but can't figure out how to do it, please feel free to post questions to the main Biopython list (see http://biopython.org/wiki/Mailing_lists). This will not only help us answer your question, it will also allow us to improve the documentation so it can help the next person do what you want to do.

Enjoy the code!

Source of the materials: Biopython Tutorial and Cookbook (adapted)

CHAPTER 4

Sequence Objects

Biological sequences are arguably the central object in Bioinformatics, and in this chapter we'll introduce the Biopython mechanism for dealing with sequences, Seq object. Later we will introduce the related SeqRecord object, which combines the sequence information with any annotation.

Sequences are essentially strings of letters like AGTACACTGGT, which seems very natural since this is the most common way that sequences are seen in biological file formats.

There are two important differences between Seq objects and standard Python strings. First of all, they have different methods. Although the Seq object supports many of the same methods as a plain string, its translate() method differs by doing biological translation, and there are also additional biologically relevant methods like reverse_complement(). Secondly, the Seq object has an important attribute, alphabet, which is an object describing what the individual characters making up the sequence string ‘‘mean’‘, and how they should be interpreted. For example, is AGTACACTGGT a DNA sequence, or just a protein sequence that happens to be rich in Alanines, Glycines, Cysteines and Threonines?

```
In [1]: from Bio.Seq import Seq
        from Bio.Alphabet import IUPAC
        from Bio.Data import CodonTable
        from Bio.SeqUtils import GC
```

4.1 Sequences and Alphabets

The alphabet object is perhaps the important thing that makes the Seq object more than just a string. The currently available alphabets for Biopython are defined in the Bio.Alphabet module. We'll use the IUPAC alphabets (http://www.chem.qmw.ac.uk/iupac/) here to deal with some of our favorite objects: DNA, RNA and Proteins.

Bio.Alphabet.IUPAC provides basic definitions for proteins, DNA and RNA, but additionally provides the ability to extend and customize the basic definitions. For instance, for proteins, there is a basic IUPACProtein class, but there is an additional ExtendedIUPACProtein class providing for the additional elements `U'' (or`Sec” for selenocysteine) and `O'' (or`Pyl” for pyrrolysine), plus the ambiguous symbols `B'' (or`Asx” for asparagine or aspartic acid), `Z'' (or`Glx” for glutamine or glutamic acid), `J'' (or`Xle” for leucine isoleucine) and `X'' (or`Xxx” for an unknown amino acid). For DNA you've got choices of IUPACUnambiguousDNA, which provides for just the basic letters, IUPACAmbiguousDNA (which provides for ambiguity letters for every possible situation) and ExtendedIUPACDNA, which allows letters for modified bases. Similarly, RNA can be represented by IUPACAmbiguousRNA or IUPACUnambiguousRNA.

The advantages of having an alphabet class are two fold. First, this gives an idea of the type of information the Seq object contains. Secondly, this provides a means of constraining the information, as a means of type checking.

Now that we know what we are dealing with, let's look at how to utilize this class to do interesting work. You can create an ambiguous sequence with the default generic alphabet like this:

```
In [2]: my_seq = Seq("AGTACACTGGT")
        my_seq
```

```
Out[2]: Seq('AGTACACTGGT', Alphabet())
```

```
In [3]: my_seq.alphabet
```

```
Out[3]: Alphabet()
```

However, where possible you should specify the alphabet explicitly when creating your sequence objects - in this case an unambiguous DNA alphabet object:

```
In [4]: from Bio.Alphabet import IUPAC
        my_seq = Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
        my_seq
```

```
Out[4]: Seq('AGTACACTGGT', IUPACUnambiguousDNA())
```

```
In [5]: my_seq.alphabet
```

```
Out[5]: IUPACUnambiguousDNA()
```

Unless of course, this really is an amino acid sequence:

```
In [6]: my_prot = Seq("AGTACACTGGT", IUPAC.protein)
        my_prot
```

```
Out[6]: Seq('AGTACACTGGT', IUPACProtein())
```

```
In [7]: my_prot.alphabet
```

```
Out[7]: IUPACProtein()
```

4.2 Sequences act like strings

In many ways, we can deal with Seq objects as if they were normal Python strings, for example getting the length, or iterating over the elements:

```
In [8]: my_seq = Seq("GATCG", IUPAC.unambiguous_dna)
        for index, letter in enumerate(my_seq):
            print("%i %s" % (index, letter))
        print(len(my_seq))
```

```
0 G
1 A
2 T
3 C
4 G
5
```

You can access elements of the sequence in the same way as for strings (but remember, Python counts from zero!):

```
In [9]: print(my_seq[0]) #first letter
        print(my_seq[2]) #third letter
        print(my_seq[-1]) #last letter
```

```
G
T
G
```

The Seq object has a .count() method, just like a string. Note that this means that like a Python string, this gives a non-overlapping count:

```
In [10]: print("AAAA".count("AA"))
         print(Seq("AAAA").count("AA"))
```

```
2
2
```

For some biological uses, you may actually want an overlapping count (i.e. 3 in this trivial example). When searching for single letters, this makes no difference:

```
In [11]: my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPAC.unambiguous_dna)
         print(len(my_seq))
         print(my_seq.count("G"))
         print(100 * float(my_seq.count("G") + my_seq.count("C")) / len(my_seq))
```

```
32
9
46.875
```

While you could use the above snippet of code to calculate a GC%, note that the **Bio.SeqUtils** module has several GC functions already built. For example:

```
In [12]: my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPAC.unambiguous_dna)
         GC(my_seq)
```

```
Out[12]: 46.875
```

Note that using the **Bio.SeqUtils.GC()** function should automatically cope with mixed case sequences and the ambiguous nucleotide S which means G or C.

Also note that just like a normal Python string, the **Seq** object is in some ways ‘‘read-only‘‘. If you need to edit your sequence, for example simulating a point mutation, look at the Section sec:mutable-seq below which talks about the **MutableSeq** object.

4.3 Slicing a sequence

A more complicated example, let’s get a slice of the sequence:

```
In [13]: my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC", IUPAC.unambiguous_dna)
         my_seq[4:12]
```

```
Out[13]: Seq('GATGGGCC', IUPACUnambiguousDNA())
```

Two things are interesting to note. First, this follows the normal conventions for Python strings. So the first element of the sequence is 0 (which is normal for computer science, but not so normal for biology). When you do a slice the first item is included (i.e.~4 in this case) and the last is excluded (12 in this case), which is the way things work in Python, but of course not necessarily the way everyone in the world would expect. The main goal is to stay consistent with what Python does.

The second thing to notice is that the slice is performed on the sequence data string, but the new object produced is another **Seq** object which retains the alphabet information from the original **Seq** object.

Also like a Python string, you can do slices with a start, stop and *stride* (the step size, which defaults to one). For example, we can get the first, second and third codon positions of this DNA sequence:

```
In [14]: print(my_seq[0::3])
         print(my_seq[1::3])
         print(my_seq[2::3])
```

```
GCTGTAGTAAG
AGGCATGCATC
TAGCTAAGAC
```

Another stride trick you might have seen with a Python string is the use of a -1 stride to reverse the string. You can do this with a Seq object too:

```
In [15]: my_seq[::-1]
```

```
Out[15]: Seq('CGCTAAAAGCTAGGATATATCCGGGTAGCTAG', IUPACUnambiguousDNA())
```

4.4 Turning Seq objects into strings

If you really do just need a plain string, for example to write to a file, or insert into a database, then this is very easy to get:

```
In [16]: str(my_seq)
```

```
Out[16]: 'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

Since calling str() on a Seq object returns the full sequence as a string, you often don't actually have to do this conversion explicitly. Python does this automatically in the print function:

```
In [17]: print(my_seq)
```

```
GATCGATGGGCCTATATAGGATCGAAAATCGC
```

You can also use the Seq object directly with a %s placeholder when using the Python string formatting or interpolation operator (%):

```
In [18]: fasta_format_string = ">Name\n%s\n" % my_seq
         print(fasta_format_string)
```

```
>Name
GATCGATGGGCCTATATAGGATCGAAAATCGC
```

This line of code constructs a simple FASTA format record (without worrying about line wrapping). Later we will describe a neat way to get a FASTA formatted string from a SeqRecord object.

```
In [19]: my_seq.tostring()
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/Seq.py:343: BiopythonDeprecationWarning:
  BiopythonDeprecationWarning)
```

```
Out[19]: 'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

4.5 Concatenating or adding sequences

Naturally, you can in principle add any two Seq objects together - just like you can with Python strings to concatenate them. However, you can't add sequences with incompatible alphabets, such as a protein sequence and a DNA sequence:

```
In [20]: protein_seq = Seq("EVRNAK", IUPAC.protein)
         dna_seq = Seq("ACGT", IUPAC.unambiguous_dna)
         try:
             protein_seq + dna_seq
```

```
        except TypeError as e:
            print(e)
```

```
Incompatible alphabets IUPACProtein() and IUPACUnambiguousDNA()
```

If you *really* wanted to do this, you'd have to first give both sequences generic alphabets:

```
In [21]: from Bio.Alphabet import generic_alphabet
         protein_seq.alphabet = generic_alphabet
         dna_seq.alphabet = generic_alphabet
         protein_seq + dna_seq
```

```
Out[21]: Seq('EVRNAKACGT', Alphabet())
```

Here is an example of adding a generic nucleotide sequence to an unambiguous IUPAC DNA sequence, resulting in an ambiguous nucleotide sequence:

```
In [22]: from Bio.Alphabet import generic_nucleotide
         nuc_seq = Seq("GATCGATGC", generic_nucleotide)
         dna_seq = Seq("ACGT", IUPAC.unambiguous_dna)
         nuc_seq
```

```
Out[22]: Seq('GATCGATGC', NucleotideAlphabet())
```

```
In [23]: dna_seq
```

```
Out[23]: Seq('ACGT', IUPACUnambiguousDNA())
```

```
In [24]: nuc_seq + dna_seq
```

```
Out[24]: Seq('GATCGATGCACGT', NucleotideAlphabet())
```

You may often have many sequences to add together, which can be done with a for loop like this:

```
In [25]: from Bio.Alphabet import generic_dna
         list_of_seqs = [Seq("ACGT", generic_dna), Seq("AACC", generic_dna), Seq("GGTT", generic_dna
         concatenated = Seq("", generic_dna)
         for s in list_of_seqs:
              concatenated += s
         concatenated
```

```
Out[25]: Seq('ACGTAACCGGTT', DNAAlphabet())
```

Or, a more elegant approach is to the use built in **sum** function with its optional start value argument (which otherwise defaults to zero):

```
In [26]: list_of_seqs = [Seq("ACGT", generic_dna), Seq("AACC", generic_dna), Seq("GGTT", generic_dna
         sum(list_of_seqs, Seq("", generic_dna))
```

```
Out[26]: Seq('ACGTAACCGGTT', DNAAlphabet())
```

Unlike the Python string, the Biopython Seq does not (currently) have a .join method.

4.6 Changing case

Python strings have very useful upper and lower methods for changing the case. As of Biopython 1.53, the Seq object gained similar methods which are alphabet aware. For example,

```
In [27]: dna_seq = Seq("acgtACGT", generic_dna)
         print(dna_seq)
         print(dna_seq.upper())
         print(dna_seq.lower())
```

```
acgtACGT
ACGTACGT
acgtacgt
```

These are useful for doing case insensitive matching:

```
In [28]: print("GTAC" in dna_seq)
         print("GTAC" in dna_seq.upper())
```

```
False
True
```

Note that strictly speaking the IUPAC alphabets are for upper case sequences only, thus:

```
In [29]: dna_seq = Seq("ACGT", IUPAC.unambiguous_dna)
         dna_seq
```

```
Out[29]: Seq('ACGT', IUPACUnambiguousDNA())
```

```
In [30]: dna_seq.lower()
```

```
Out[30]: Seq('acgt', DNAAlphabet())
```

4.7 Nucleotide sequences and (reverse) complements

For nucleotide sequences, you can easily obtain the complement or reverse complement of a Seq object using its built-in methods:

```
In [31]: my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC", IUPAC.unambiguous_dna)
         print(my_seq)
         print(my_seq.complement())
         print(my_seq.reverse_complement())
```

```
GATCGATGGGCCTATATAGGATCGAAAATCGC
CTAGCTACCCGGATATATCCTAGCTTTTAGCG
GCGATTTTCGATCCTATATAGGCCCATCGATC
```

As mentioned earlier, an easy way to just reverse a Seq object (or a Python string) is slice it with -1 step:

```
In [32]: my_seq[::-1]
```

```
Out[32]: Seq('CGCTAAAAGCTAGGATATATCCGGGTAGCTAG', IUPACUnambiguousDNA())
```

In all of these operations, the alphabet property is maintained. This is very useful in case you accidentally end up trying to do something weird like take the (reverse)complement of a protein sequence:

```
In [33]: protein_seq = Seq("EVRNAK", IUPAC.protein)
         try:
             protein_seq.complement()
         except ValueError as e:
             print(e)
```

```
Proteins do not have complements!
```

4.8 Transcription

Before talking about transcription, I want to try and clarify the strand issue. Consider the following (made up) stretch of double stranded DNA which encodes a short peptide:

	DNA coding strand (Crick strand, strand +1)	
5’	ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG	3’
3’	TACCGGTAACATTACCCGGCGACTTTCCCACGGGCTATC	5’
	DNA template strand (Watson strand, strand -1)	
	Transcription	
5’	AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG}	3’
	Single stranded messenger RNA	

The actual biological transcription process works from the template strand, doing a reverse complement (TCAG -> CUGA) to give the mRNA. However, in Biopython and bioinformatics in general, we typically work directly with the coding strand because this means we can get the mRNA sequence just by switching T -> U.

Now let’s actually get down to doing a transcription in Biopython. First, let’s create Seq objects for the coding and template DNA strands:

```
In [34]: coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous_dna)
         print(coding_dna)
         template_dna = coding_dna.reverse_complement()
         print(template_dna)
```

```
ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG
CTATCGGGCACCCTTTCAGCGGCCCATTACAATGGCCAT
```

These should match the figure above - remember by convention nucleotide sequences are normally read from the 5’ to 3’ direction, while in the figure the template strand is shown reversed.

Now let’s transcribe the coding strand into the corresponding mRNA, using the Seq object’s built in transcribe method:

```
In [35]: messenger_rna = coding_dna.transcribe()
         messenger_rna
```

```
Out[35]: Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

As you can see, all this does is switch T $\rightarrow$ U, and adjust the alphabet.

If you do want to do a true biological transcription starting with the template strand, then this becomes a two-step process:

```
In [36]: template_dna.reverse_complement().transcribe()
```

```
Out[36]: Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

The Seq object also includes a back-transcription method for going from the mRNA to the coding strand of the DNA. Again, this is a simple U $\rightarrow$ T substitution and associated change of alphabet:

```
In [37]: messenger_rna.back_transcribe()
```

```
Out[37]: Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
```

4.9 Translation

Sticking with the same example discussed in the transcription section above, now let’s translate this mRNA into the corresponding protein sequence - again taking advantage of one of the Seq object’s biological methods:

```
In [38]: messenger_rna.translate()
```

```
Out[38]: Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

You can also translate directly from the coding strand DNA sequence:

```
In [39]: coding_dna.translate()
```

```
Out[39]: Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

You should notice in the above protein sequences that in addition to the end stop character, there is an internal stop as well. This was a deliberate choice of example, as it gives an excuse to talk about some optional arguments, including different translation tables (Genetic Codes).

The translation tables available in Biopython are based on those http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.c {from the NCBI} (see the next section of this tutorial). By default, translation will use the *standard* genetic code (NCBI table id 1). Suppose we are dealing with a mitochondrial sequence. We need to tell the translation function to use the relevant genetic code instead:

```
In [40]: coding_dna.translate(table="Vertebrate Mitochondrial")

Out[40]: Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
```

You can also specify the table using the NCBI table number which is shorter, and often included in the feature annotation of GenBank files:

```
In [41]: coding_dna.translate(table=2)

Out[41]: Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Now, you may want to translate the nucleotides up to the first in frame stop codon, and then stop (as happens in nature):

```
In [42]: coding_dna.translate()

Out[42]: Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))

In [43]: coding_dna.translate(to_stop=True)

Out[43]: Seq('MAIVMGR', IUPACProtein())

In [44]: coding_dna.translate(table=2)

Out[44]: Seq('MAIVMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))

In [45]: coding_dna.translate(table=2, to_stop=True)

Out[45]: Seq('MAIVMGRWKGAR', IUPACProtein())
```

Notice that when you use the to_stop argument, the stop codon itself is not translated - and the stop symbol is not included at the end of your protein sequence.

You can even specify the stop symbol if you don't like the default asterisk:

```
In [46]: coding_dna.translate(table=2, stop_symbol="@")

Out[46]: Seq('MAIVMGRWKGAR@', HasStopCodon(IUPACProtein(), '@'))
```

Now, suppose you have a complete coding sequence CDS, which is to say a nucleotide sequence (e.g. mRNA – after any splicing) which is a whole number of codons (i.e. the length is a multiple of three), commences with a start codon, ends with a stop codon, and has no internal in-frame stop codons. In general, given a complete CDS, the default translate method will do what you want (perhaps with the to_stop option). However, what if your sequence uses a non-standard start codon? This happens a lot in bacteria – for example the gene yaaX in *E. coli* K12:

```
In [47]: gene = Seq("GTGAAAAAGATGCAATCTATCGTACTCGCACTTTCCCTGGTTCTGGTCGCTCCCATGGCA" +
                    "GCACAGGCTGCGGAAATTACGTTAGTCCCGTCAGTAAAATTACAGATAGGCGATCGTGAT" +
                    "AATCGTGGCTATTACTGGGATGGAGGTCACTGGCGCGACCACGGCTGGTGGAAACAACAT" +
                    "TATGAATGGCGAGGCAATCGCTGGCACCTACACGGACCGCCGCCACCGCCGCGCCACCAT" +
                    "AAGAAAGCTCCTCATGATCATCACGGCGGTCATGGTCCAGGCAAACATCACCGCTAA",
                    generic_dna)
         gene.translate(table="Bacterial")

Out[47]: Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HR*', HasStopCodon(ExtendedIU

In [48]: gene.translate(table="Bacterial", to_stop=True)

Out[48]: Seq('VKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR', ExtendedIUPACProtein())
```

In the bacterial genetic code GTG is a valid start codon, and while it does *normally* encode Valine, if used as a start codon it should be translated as methionine. This happens if you tell Biopython your sequence is a complete CDS:

```
In [49]: gene.translate(table="Bacterial", cds=True)
```

```
Out[49]: Seq('MKKMQSIVLALSLVLVAPMAAQAAEITLVPSVKLQIGDRDNRGYYWDGGHWRDH...HHR', ExtendedIUPACProtein())
```

In addition to telling Biopython to translate an alternative start codon as methionine, using this option also makes sure your sequence really is a valid CDS (you'll get an exception if not).

4.10 Translation Tables

In the previous sections we talked about the Seq object translation method (and mentioned the equivalent function in the Bio.Seq module). Internally these use codon table objects derived from the NCBI information at ftp://ftp.ncbi.nlm.nih.gov/entrez/misc/data/gc.prt, also shown on http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi in a much more readable layout.

As before, let's just focus on two choices: the Standard translation table, and the translation table for Vertebrate Mitochondrial DNA.

```
In [50]: standard_table = CodonTable.unambiguous_dna_by_name["Standard"]
         mito_table = CodonTable.unambiguous_dna_by_name["Vertebrate Mitochondrial"]
```

Alternatively, these tables are labeled with ID numbers 1 and 2, respectively:

```
In [51]: standard_table = CodonTable.unambiguous_dna_by_id[1]
         mito_table = CodonTable.unambiguous_dna_by_id[2]
```

You can compare the actual tables visually by printing them:

```
In [52]: print(standard_table)
```

```
Table 1 Standard, SGC0

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I   | ACT T   | AAT N   | AGT S   | T
A | ATC I   | ACC T   | AAC N   | AGC S   | C
A | ATA I   | ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V   | GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

```
In [53]: print(mito_table)
```

```
Table 2 Vertebrate Mitochondrial, SGC1
```

```
  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA W   | A
T | TTG L   | TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L   | CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I(s)| ACT T   | AAT N   | AGT S   | T
A | ATC I(s)| ACC T   | AAC N   | AGC S   | C
A | ATA M(s)| ACA T   | AAA K   | AGA Stop| A
A | ATG M(s)| ACG T   | AAG K   | AGG Stop| G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V(s)| GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

You may find these following properties useful – for example if you are trying to do your own gene finding:

```
In [54]: print(mito_table.stop_codons)
         print(mito_table.start_codons)
         print(mito_table.forward_table["ACG"])
```

```
['TAA', 'TAG', 'AGA', 'AGG']
['ATT', 'ATC', 'ATA', 'ATG', 'GTG']
T
```

4.11 Comparing Seq objects

Sequence comparison is actually a very complicated topic, and there is no easy way to decide if two sequences are equal. The basic problem is the meaning of the letters in a sequence are context dependent - the letter ‘‘A” could be part of a DNA, RNA or protein sequence. Biopython uses alphabet objects as part of each Seq object to try and capture this information - so comparing two Seq objects means considering both the sequence strings *and* the alphabets.

For example, you might argue that the two DNA Seq objects Seq(“ACGT”, IUPAC.unambiguous_dna) and Seq(“ACGT”, IUPAC.ambiguous_dna) should be equal, even though they do have different alphabets. Depending on the context this could be important.

This gets worse – suppose you think Seq(“ACGT”, IUPAC.unambiguous_dna) and Seq(“ACGT”) (i.e. the default generic alphabet) should be equal. Then, logically, Seq(“ACGT”, IUPAC.protein) and Seq(“ACGT”) should also be equal. Now, in logic if $A = B$ and $B = C$, by transitivity we expect $A = C$. So for logical consistency we'd require Seq(“ACGT”, IUPAC.unambiguous_dna) and Seq(“ACGT”, IUPAC.protein) to be equal – which most people would agree is just not right. This transitivity problem would also have implications for using Seq objects as Python dictionary keys.

```
In [55]: seq1 = Seq("ACGT", IUPAC.unambiguous_dna)
         seq2 = Seq("ACGT", IUPAC.unambiguous_dna)
```

So, what does Biopython do? Well, the equality test is the default for Python objects – it tests to see if they are the same object in memory. This is a very strict test:

```
In [56]: print(seq1 == seq2)
         print(seq1 == seq1)
```

```
True
True
```

If you actually want to do this, you can be more explicit by using the Python id function,

```
In [57]: print(id(seq1) == id(seq2))
         print(id(seq1) == id(seq1))
```

```
False
True
```

Now, in every day use, your sequences will probably all have the same alphabet, or at least all be the same type of sequence (all DNA, all RNA, or all protein). What you probably want is to just compare the sequences as strings – so do this explicitly:

```
In [58]: print(str(seq1) == str(seq2))
         print(str(seq1) == str(seq1))
```

```
True
True
```

As an extension to this, while you can use a Python dictionary with **Seq** objects as keys, it is generally more useful to use the sequence a string for the key.

4.12 MutableSeq Objects

Just like the normal Python string, the **Seq** object is ''read only'', or in Python terminology, immutable. Apart from wanting the **Seq** object to act like a string, this is also a useful default since in many biological applications you want to ensure you are not changing your sequence data:

```
In [59]: my_seq = Seq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA", IUPAC.unambiguous_dna)
```

Observe what happens if you try to edit the sequence:

```
In [60]: try:
             my_seq[5] = "G"
         except Exception as e:
             print(e)
```

```
'Seq' object does not support item assignment
```

However, you can convert it into a mutable sequence (a **MutableSeq** object) and do pretty much anything you want with it

```
In [61]: mutable_seq = my_seq.tomutable()
         mutable_seq
```

```
Out[61]: MutableSeq('GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA', IUPACUnambiguousDNA())
```

Alternatively, you can create a **MutableSeq** object directly from a string:

```
In [62]: from Bio.Seq import MutableSeq
         mutable_seq = MutableSeq("GCCATTGTAATGGGCCGCTGAAAGGGTGCCCGA", IUPAC.unambiguous_dna)
```

Either way will give you a sequence object which can be changed:

```
In [63]: mutable_seq[5] = "C"
         print(mutable_seq)
         mutable_seq.remove("T")
         print(mutable_seq)
         mutable_seq.reverse()
         print(mutable_seq)
```

```
GCCATCGTAATGGGCCGCTGAAAGGGTGCCCGA
GCCACGTAATGGGCCGCTGAAAGGGTGCCCGA
AGCCCGTGGGAAAGTCGCCGGGTAATGCACCG
```

Do note that unlike the Seq object, the MutableSeq object's methods like reverse_complement() and reverse() act in-situ!

An important technical difference between mutable and immutable objects in Python means that you can't use a MutableSeq object as a dictionary key, but you can use a Python string or a Seq object in this way.

Once you have finished editing your a MutableSeq object, it's easy to get back to a read-only Seq object should you need to:

```
In [64]: new_seq = mutable_seq.toseq()
         new_seq
```

```
Out[64]: Seq('AGCCCGTGGGAAAGTCGCCGGGTAATGCACCG', IUPACUnambiguousDNA())
```

You can also get a string from a MutableSeq object just like from a Seq object.

4.13 UnknownSeq Objects

The UnknownSeq object is a subclass of the basic Seq object and its purpose is to represent a sequence where we know the length, but not the actual letters making it up. You could of course use a normal Seq object in this situation, but it wastes rather a lot of memory to hold a string of a million N`'' characters when you could just store a single letter`N" and the desired length as an integer.

```
In [65]: from Bio.Seq import UnknownSeq
         unk = UnknownSeq(20)
         unk
```

```
Out[65]: UnknownSeq(20, alphabet = Alphabet(), character = '?')
```

```
In [66]: print(unk)
         print(len(unk))
```

```
????????????????????
20
```

You can of course specify an alphabet, meaning for nucleotide sequences the letter defaults to 'N' and for proteins 'X', rather than just '?'.

```
In [67]: unk_dna = UnknownSeq(20, alphabet=IUPAC.ambiguous_dna)
         unk_dna
```

```
Out[67]: UnknownSeq(20, alphabet = IUPACAmbiguousDNA(), character = 'N')
```

```
In [68]: print(unk_dna)
```

```
NNNNNNNNNNNNNNNNNNNN
```

You can use all the usual **Seq** object methods too, note these give back memory saving **UnknownSeq** objects where appropriate as you might expect:

```
In [69]: unk_dna
```

```
Out[69]: UnknownSeq(20, alphabet = IUPACAmbiguousDNA(), character = 'N')
```

```
In [70]: unk_dna.complement()
```

```
Out[70]: UnknownSeq(20, alphabet = IUPACAmbiguousDNA(), character = 'N')
```

```
In [71]: unk_dna.reverse_complement()
```

```
Out[71]: UnknownSeq(20, alphabet = IUPACAmbiguousDNA(), character = 'N')
```

```
In [72]: unk_dna.transcribe()
```

```
Out[72]: UnknownSeq(20, alphabet = IUPACAmbiguousRNA(), character = 'N')
```

```
In [73]: unk_protein = unk_dna.translate()
         unk_protein
```

```
Out[73]: UnknownSeq(6, alphabet = ProteinAlphabet(), character = 'X')
```

```
In [74]: print(unk_protein)
         print(len(unk_protein))
```

```
XXXXXX
6
```

You may be able to find a use for the UnknownSeq object in your own code, but it is more likely that you will first come across them in a SeqRecord object created by Bio.SeqIO. Some sequence file formats don't always include the actual sequence, for example GenBank and EMBL files may include a list of features but for the sequence just present the contig information. Alternatively, the QUAL files used in sequencing work hold quality scores but they *never* contain a sequence – instead there is a partner FASTA file which *does* have the sequence.

4.14 Working with strings directly

To close this chapter, for those you who *really* don't want to use the sequence objects (or who prefer a functional programming style to an object orientated one), there are module level functions in Bio.Seq will accept plain Python strings, Seq objects (including UnknownSeq objects) or MutableSeq objects:

```
In [75]: from Bio.Seq import reverse_complement, transcribe, back_transcribe, translate
         my_string = "GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCTGCTGGTTAG"
         print(reverse_complement(my_string))
         print(transcribe(my_string))
         print(back_transcribe(my_string))
         print(translate(my_string))
```

```
CTAACCAGCAGCACGACCACCCTTCCAACGACCCATAACAGC
GCUGUUAUGGGUCGUUGGAAGGGUGGUCGUGCUGCUGGUUAG
GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCTGCTGGTTAG
AVMGRWKGGRAAG*
```

You are, however, encouraged to work with Seq objects by default.

Source of the materials: Biopython Tutorial and Cookbook (adapted)

CHAPTER 5

Sequence annotation objects

The previous notebook introduced the sequence classes. Immediately ‘‘above” the Seq class is the Sequence Record or SeqRecord class, defined in the Bio.SeqRecord module. This class allows higher level features such as identifiers and features (as SeqFeature objects) to be associated with the sequence, and is used throughout the sequence input/output interface Bio.SeqIO described fully in another notebook.

If you are only going to be working with simple data like FASTA files, you can probably skip this chapter for now. If on the other hand you are going to be using richly annotated sequence data, say from GenBank or EMBL files, this information is quite important.

While this chapter should cover most things to do with the **:raw-latex:‘\verb|SeqRecord|‘** and **:raw-latex:‘\verb|SeqFeature|‘** objects in this chapter, you may also want to read the SeqRecord wiki page (http://biopython.org/wiki/SeqRecord), and the built in documentation (also online – http://biopython.org/DIST/docs/api/Bio.SeqRecord.SeqRecord-class.html - SeqRecord and http://biopython.org/DIST/docs/api/Bio.SeqFeature.SeqFeature-class.html - SeqFeature):

```
In [1]: from Bio import SeqIO, SeqFeature
        from Bio.Alphabet import SingleLetterAlphabet, generic_protein
        from Bio.Alphabet.IUPAC import IUPACAmbiguousDNA
        from Bio.SeqFeature import FeatureLocation
        from Bio.SeqRecord import SeqRecord
```

5.1 The SeqRecord Object

The SeqRecord (Sequence Record) class is defined in the Bio.SeqRecord module. This class allows higher level features such as identifiers and features to be associated with a sequence, and is the basic data type for the Bio.SeqIO sequence input/output interface.

The SeqRecord class itself is quite simple, and offers the following information as attributes:

- **.seq** - The sequence itself, typically a Seq object.
- **.id** - The primary ID used to identify the sequence - a string. In most cases this is something like an accession number.

- **.name** - A ‘common’ name/id for the sequence - a string. In some cases this will be the same as the accession number, but it could also be a clone name. I think of this as being analogous to the LOCUS id in a GenBank record.
- **.description** - A human readable description or expressive name for the sequence - a string.
- **.letter_annotations** - Holds per-letter-annotations using a (restricted) dictionary of additional information about the letters in the sequence. The keys are the name of the information, and the information is contained in the value as a Python sequence (i.e. a list, tuple or string) with the same length as the sequence itself. This is often used for quality scores or secondary structure information (e.g. from Stockholm/PFAM alignment files).
- **.annotations** - A dictionary of additional information about the sequence. The keys are the name of the information, and the information is contained in the value. This allows the addition of more ‘unstructured’ information to the sequence.
- **.features** - A list of SeqFeature objects with more structured information about the features on a sequence (e.g. position of genes on a genome, or domains on a protein sequence).
- **.dbxrefs** - A list of database cross-references as strings.

5.2 Creating a SeqRecord

Using a SeqRecord object is not very complicated, since all of the information is presented as attributes of the class. Usually you won’t create a SeqRecord ‘by hand’, but instead use Bio.SeqIO to read in a sequence file for you and the examples below). However, creating SeqRecord can be quite simple.

5.2.1 SeqRecord objects from scratch

To create a SeqRecord at a minimum you just need a Seq object:

```
In [2]: from Bio.Seq import Seq
        from Bio.SeqRecord import SeqRecord
```

```
In [3]: simple_seq = Seq("GATC")
        print(simple_seq)
        simple_seq_r = SeqRecord(simple_seq)
        print(simple_seq_r)
```

```
GATC
ID: <unknown id>
Name: <unknown name>
Description: <unknown description>
Number of features: 0
Seq('GATC', Alphabet())
```

Additionally, you can also pass the id, name and description to the initialization function, but if not they will be set as strings indicating they are unknown, and can be modified subsequently:

```
In [4]: simple_seq_r.id
        simple_seq_r.id = "AC12345"
        simple_seq_r.description = "Made up sequence I wish I could write a paper about"
        print(simple_seq_r.description)
        simple_seq_r.seq
        print(simple_seq_r.seq)
```

```
Made up sequence I wish I could write a paper about
GATC
```

Including an identifier is very important if you want to output your SeqRecord to a file. You would normally include this when creating the object:

```
In [5]: simple_seq = Seq("GATC")
        print(simple_seq)
        simple_seq_r = SeqRecord(simple_seq, id="AC12345")
        print(simple_seq_r)
```

```
GATC
ID: AC12345
Name: <unknown name>
Description: <unknown description>
Number of features: 0
Seq('GATC', Alphabet())
```

As mentioned above, the SeqRecord has an dictionary attribute annotations. This is used for any miscellaneous annotations that doesn't fit under one of the other more specific attributes. Adding annotations is easy, and just involves dealing directly with the annotation dictionary:

```
In [6]: simple_seq_r.annotations["evidence"] = "None. I just made it up."
        print(simple_seq_r.annotations)
        print(simple_seq_r.annotations["evidence"])
```

```
{'evidence': 'None. I just made it up.'}
None. I just made it up.
```

Working with per-letter-annotations is similar, letter_annotations is a dictionary like attribute which will let you assign any Python sequence (i.e. a string, list or tuple) which has the same length as the sequence:

```
In [7]: simple_seq_r.letter_annotations["phred_quality"] = [40, 40, 38, 30]
        print(simple_seq_r.letter_annotations)
        print(simple_seq_r.letter_annotations["phred_quality"])
```

```
{'phred_quality': [40, 40, 38, 30]}
[40, 40, 38, 30]
```

The dbxrefs and features attributes are just Python lists, and should be used to store strings and SeqFeature objects (discussed later) respectively.

5.2.2 SeqRecord objects from FASTA files

This example uses a fairly large FASTA file containing the whole sequence for **:raw-latex:`\textit{Yersinia pestis biovar Microtus}`** str. 91001 plasmid pPCP1, originally downloaded from the NCBI. This file is included with the Biopython unit tests under the GenBank folder, or online (http://biopython.org/SRC/biopython/Tests/GenBank/NC_005816.fna) from our website.

The file starts like this - and you can check there is only one record present (i.e. only one line starting with a greater than symbol):

```
>gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus ... pPCP1, complete␣
↪sequence
TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGGGGGTAATCTGCTCTCC
...
```

In a previous notebook you will have seen the function Bio.SeqIO.parse used to loop over all the records in a file as SeqRecord objects. The Bio.SeqIO module has a sister function for use on files which contain just one record which we'll use here:

```
In [8]: record = SeqIO.read("data/NC_005816.fna", "fasta")
        print(record)
```

```
ID: gi|45478711|ref|NC_005816.1|
Name: gi|45478711|ref|NC_005816.1|
Description: gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, c
Number of features: 0
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', SingleLetterAlphabet())
```

Now, let's have a look at the key attributes of this SeqRecord individually - starting with the seq attribute which gives you a Seq object:

```
In [9]: record.seq
```

```
Out[9]: Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', SingleLetterAlphabet())
```

Here Bio.SeqIO has defaulted to a generic alphabet, rather than guessing that this is DNA. If you know in advance what kind of sequence your FASTA file contains, you can tell Bio.SeqIO which alphabet to use.

Next, the identifiers and description:

```
In [10]: print(record.id)
         print(record.name)
         print(record.description)
```

```
gi|45478711|ref|NC_005816.1|
gi|45478711|ref|NC_005816.1|
gi|45478711|ref|NC_005816.1| Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete seque
```

As you can see above, the first word of the FASTA record's title line (after removing the greater than symbol) is used for both the id and name attributes. The whole title line (after removing the greater than symbol) is used for the record description. This is deliberate, partly for backwards compatibility reasons, but it also makes sense if you have a FASTA file like this:

```
>Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1
TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGGGGGTAATCTGCTCTCC
...
```

Note that none of the other annotation attributes get populated when reading a FASTA file:

```
In [11]: print(record.dbxrefs)
         print(record.annotations)
         print(record.letter_annotations)
         print(record.features)
```

```
[]
{}
{}
[]
```

In this case our example FASTA file was from the NCBI, and they have a fairly well defined set of conventions for formatting their FASTA lines. This means it would be possible to parse this information and extract the GI number and accession for example. However, FASTA files from other sources vary, so this isn't possible in general.

5.2.3 SeqRecord objects from GenBank files

As in the previous example, we're going to look at the whole sequence for *Yersinia pestis biovar Microtus* str. 91001 plasmid pPCP1, originally downloaded from the NCBI, but this time as a GenBank file.

This file contains a single record (i.e. only one LOCUS line) and starts:

```
LOCUS       NC_005816               9609 bp    DNA     circular BCT 21-JUL-2008
DEFINITION  Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete
            sequence.
```

```
ACCESSION   NC_005816
VERSION     NC_005816.1  GI:45478711
PROJECT     GenomeProject:10638
...
```

Again, we'll use Bio.SeqIO to read this file in, and the code is almost identical to that for used above for the FASTA file:

```
In [12]: record = SeqIO.read("data/NC_005816.gb", "genbank")
         print(record)
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Database cross-references: Project:58037
Number of features: 41
/taxonomy=['Bacteria', 'Proteobacteria', 'Gammaproteobacteria', 'Enterobacteriales', 'Enterobacteriac
/comment=PROVISIONAL REFSEQ: This record has not yet been subject to final
NCBI review. The reference sequence was derived from AE017046.
COMPLETENESS: full length.
/accessions=['NC_005816']
/organism=Yersinia pestis biovar Microtus str. 91001
/sequence_version=1
/source=Yersinia pestis biovar Microtus str. 91001
/data_file_division=BCT
/keywords=['']
/date=21-JUL-2008
/gi=45478711
/references=[Reference(title='Genetics of metabolic variations between Yersinia pestis biovars and th
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', IUPACAmbiguousDNA())
```

You should be able to spot some differences already! But taking the attributes individually, the sequence string is the same as before, but this time Bio.SeqIO has been able to automatically assign a more specific alphabet:

```
In [13]: record.seq
```

```
Out[13]: Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', IUPACAmbiguousDNA())
```

The name comes from the LOCUS line, while the **:raw-latex:'\verb|id|'** includes the version suffix.

The description comes from the DEFINITION line:

```
In [14]: print(record.id)
         print(record.name)
         print(record.description)
```

```
NC_005816.1
NC_005816
Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
```

GenBank files don't have any per-letter annotations:

```
In [15]: record.letter_annotations
```

```
Out[15]: {}
```

Most of the annotations information gets recorded in the **:raw-latex:'\verb|annotations|'** dictionary, for example:

```
In [16]: print(len(record.annotations))
         print(record.annotations["source"])
```

```
11
Yersinia pestis biovar Microtus str. 91001
```

The dbxrefs list gets populated from any PROJECT or DBLINK lines:

```
In [17]: record.dbxrefs
```

```
Out[17]: ['Project:58037']
```

Finally, and perhaps most interestingly, all the entries in the features table (e.g. the genes or CDS features) get recorded as SeqFeature objects in the features list.

```
In [18]: len(record.features)
```

```
Out[18]: 41
```

5.3 Feature, location and position objects

5.3.1 SeqFeature objects

Sequence features are an essential part of describing a sequence. Once you get beyond the sequence itself, you need some way to organize and easily get at the more 'abstract' information that is known about the sequence. While it is probably impossible to develop a general sequence feature class that will cover everything, the Biopython SeqFeature class attempts to encapsulate as much of the information about the sequence as possible. The design is heavily based on the GenBank/EMBL feature tables, so if you understand how they look, you'll probably have an easier time grasping the structure of the Biopython classes.

The key idea about each SeqFeature object is to describe a region on a parent sequence, typically a SeqRecord object. That region is described with a location object, typically a range between two positions (see below).

The SeqFeature class has a number of attributes, so first we'll list them and their general features, and then later in the chapter work through examples to show how this applies to a real life example. The attributes of a SeqFeature are:

- **.type]** - This is a textual description of the type of feature (for instance, this will be something like 'CDS' or 'gene').
- **.location** - The location of the SeqFeature on the sequence that you are dealing with. The SeqFeature delegates much of its functionality to the location object, and includes a number of shortcut attributes for properties of the location:
- **.ref** - shorthand for .location.ref - any (different) reference sequence the location is referring to. Usually just None.
- **.ref_db** - shorthand for .location.ref_db - specifies the database any identifier in .ref refers to. Usually just None.
- **.strand** - shorthand for .location.strand - the strand on the sequence that the feature is located on. For double stranded nucleotide sequence this may either be 1 for the top strand, -1 for the bottom strand, 0 if the strand is important but is unknown, or None if it doesn't matter. This is None for proteins, or single stranded sequences.
- **.qualifiers** - This is a Python dictionary of additional information about the feature. The key is some kind of terse one-word description of what the information contained in the value is about, and the value is the actual information. For example, a common key for a qualifier might be 'evidence' and the value might be 'computational (non-experimental). This is just a way to let the person who is looking at the feature know that it has not be experimentally (i.e. in a wet lab) confirmed. Note that other the value will be a list of strings (even when there is only one string). This is a reflection of the feature tables in GenBank/EMBL files.
- **.sub_features** - This used to be used to represent features with complicated locations like 'joins' in GenBank/EMBL files. This has been deprecated with the introduction of the CompoundLocation object, and should now be ignored.

5.3.2 Positions and locations

The key idea about each SeqFeature object is to describe a region on a parent sequence, for which we use a location object, typically describing a range between two positions. Two try to clarify the terminology we're using:

- **position** - This refers to a single position on a sequence, which may be fuzzy or not. For instance, 5, 20, <100 and >200 are all positions.
- **location** - A location is region of sequence bounded by some positions. For instance 5..20 (i.e. 5 to 20) is a location.

I just mention this because sometimes I get confused between the two.

FeatureLocation object

Unless you work with eukaryotic genes, most SeqFeature locations are extremely simple - you just need start and end coordinates and a strand. That's essentially all the basic FeatureLocation object does.

In practise of course, things can be more complicated. First of all we have to handle compound locations made up of several regions. Secondly, the positions themselves may be fuzzy (inexact).

CompoundLocation object

Biopython 1.62 introduced the CompoundLocation as part of a restructuring of how complex locations made up of multiple regions are represented. The main usage is for handling 'join' locations in EMBL/GenBank files.

Fuzzy Positions

So far we've only used simple positions. One complication in dealing with feature locations comes in the positions themselves. In biology many times things aren't entirely certain (as much as us wet lab biologists try to make them certain!). For instance, you might do a dinucleotide priming experiment and discover that the start of mRNA transcript starts at one of two sites. This is very useful information, but the complication comes in how to represent this as a position. To help us deal with this, we have the concept of fuzzy positions. Basically there are several types of fuzzy positions, so we have five classes do deal with them:

- **ExactPosition** - As its name suggests, this class represents a position which is specified as exact along the sequence. This is represented as just a number, and you can get the position by looking at the position attribute of the object.
- **BeforePosition** - This class represents a fuzzy position that occurs prior to some specified site. In GenBank/EMBL notation, this is represented as something like <13, signifying that the real position is located somewhere less than 13. To get the specified upper boundary, look at the position attribute of the object.
- **AfterPosition** - Contrary to BeforePosition, this class represents a position that occurs after some specified site. This is represented in GenBank as >13, and like BeforePosition, you get the boundary number by looking at the position attribute of the object.
- **WithinPosition** - Occasionally used for GenBank/EMBL locations, this class models a position which occurs somewhere between two specified nucleotides. In GenBank/EMBL notation, this would be represented as (1.5), to represent that the position is somewhere within the range 1 to 5. To get the information in this class you have to look at two attributes. The position attribute specifies the lower boundary of the range we are looking at, so in our example case this would be one. The extension attribute specifies the range to the higher boundary, so in this case it would be 4. So object.position is the lower boundary and object.position + object.extension is the upper boundary.

- **OneOfPosition** - Occasionally used for GenBank/EMBL locations, this class deals with a position where several possible values exist, for instance you could use this if the start codon was unclear and there where two candidates for the start of the gene. Alternatively, that might be handled explicitly as two related gene features.
- **UnknownPosition** - This class deals with a position of unknown location. This is not used in GenBank/EMBL, but corresponds to the '?' feature coordinate used in UniProt.

Here's an example where we create a location with fuzzy end points:

```
In [19]: start_pos = SeqFeature.AfterPosition(5)
         end_pos = SeqFeature.BetweenPosition(9, left=8, right=9)
         my_location = SeqFeature.FeatureLocation(start_pos, end_pos)
```

Note that the details of some of the fuzzy-locations changed in Biopython 1.59, in particular for BetweenPosition and WithinPosition you must now make it explicit which integer position should be used for slicing etc. For a start position this is generally the lower (left) value, while for an end position this would generally be the higher (right) value.

If you print out a FeatureLocation object, you can get a nice representation of the information:

```
In [20]: print(my_location)
```

```
[>5:(8^9)]
```

We can access the fuzzy start and end positions using the start and end attributes of the location:

```
In [21]: print(my_location.start)
         print(my_location.start)
         print(my_location.end)
         print(my_location.end)
```

```
>5
>5
(8^9)
(8^9)
```

If you don't want to deal with fuzzy positions and just want numbers, they are actually subclasses of integers so should work like integers:

```
In [22]: print(int(my_location.start))
         print(int(my_location.end))
```

```
5
9
```

For compatibility with older versions of Biopython you can ask for the **:raw-latex:'\verb|nofuzzy_start|'** and **:raw-latex:'\verb|nofuzzy_end|'** attributes of the location which are plain integers:

```
In [23]: print(my_location.nofuzzy_start)
         print(my_location.nofuzzy_end)
```

```
5
9
```

Notice that this just gives you back the position attributes of the fuzzy locations.

Similarly, to make it easy to create a position without worrying about fuzzy positions, you can just pass in numbers to the **:raw-latex:'\verb|FeaturePosition|'** constructors, and you'll get back out **:raw-latex:'\verb|ExactPosition|'** objects:

```
In [24]: exact_location = SeqFeature.FeatureLocation(5, 9)
         print(exact_location)
         print(exact_location.start)
         print(int(exact_location.start))
         print(exact_location.nofuzzy_start)
```

```
[5:9]
5
5
5
```

That is most of the nitty gritty about dealing with fuzzy positions in Biopython. It has been designed so that dealing with fuzziness is not that much more complicated than dealing with exact positions, and hopefully you find that true!

Location testing

You can use the Python keyword in with a SeqFeature or location object to see if the base/residue for a parent coordinate is within the feature/location or not.

For example, suppose you have a SNP of interest and you want to know which features this SNP is within, and lets suppose this SNP is at index 4350 (Python counting!). Here is a simple brute force solution where we just check all the features one by one in a loop:

```
In [25]: my_snp = 4350
         record = SeqIO.read("data/NC_005816.gb", "genbank")
         for feature in record.features:
            if my_snp in feature:
                print("%s %s" % (feature.type, feature.qualifiers.get('db_xref')))
```

```
source ['taxon:229193']
gene ['GeneID:2767712']
CDS ['GI:45478716', 'GeneID:2767712']
```

Note that gene and CDS features from GenBank or EMBL files defined with joins are the union of the exons – they do not cover any introns.

5.3.3 Sequence described by a feature or location

A SeqFeature or location object doesn't directly contain a sequence, instead the location describes how to get this from the parent sequence. For example consider a (short) gene sequence with location 5:18 on the reverse strand, which in GenBank/EMBL notation using 1-based counting would be complement(6..18), like this:

```
In [26]: example_parent = Seq("ACCGAGACGGCAAAGGCTAGCATAGGTATGAGACTTCCTTCCTGCCAGTGCTGAGGAACTGGGAGCCTAC
         example_feature = SeqFeature.SeqFeature(FeatureLocation(5, 18), type="gene", strand=-1)
```

You could take the parent sequence, slice it to extract 5:18, and then take the reverse complement. If you are using Biopython 1.59 or later, the feature location's start and end are integer like so this works:

```
In [27]: feature_seq = example_parent[example_feature.location.start:example_feature.location.end].re
         print(feature_seq)
```

```
AGCCTTTGCCGTC
```

This is a simple example so this isn't too bad – however once you have to deal with compound features (joins) this is rather messy. Instead, the SeqFeature object has an extract method to take care of all this:

```
In [28]: feature_seq = example_feature.extract(example_parent)
         print(feature_seq)
```

```
AGCCTTTGCCGTC
```

The length of a SeqFeature or location matches that of the region of sequence it describes.

```
In [29]: print(example_feature.extract(example_parent))
         print(len(example_feature.extract(example_parent)))
         print(len(example_feature))
         print(len(example_feature.location))
```

```
AGCCTTTGCCGTC
13
13
13
```

For simple FeatureLocation objects the length is just the difference between the start and end positions. However, for a CompoundLocation the length is the sum of the constituent regions.

5.4 References

Another common annotation related to a sequence is a reference to a journal or other published work dealing with the sequence. We have a fairly simple way of representing a Reference in Biopython – we have a Bio.SeqFeature.Reference class that stores the relevant information about a reference as attributes of an object.

The attributes include things that you would expect to see in a reference like journal, title and authors. Additionally, it also can hold the medline_id and pubmed_id and a comment about the reference. These are all accessed simply as attributes of the object.

A reference also has a location object so that it can specify a particular location on the sequence that the reference refers to. For instance, you might have a journal that is dealing with a particular gene located on a BAC, and want to specify that it only refers to this position exactly. The location is a potentially fuzzy location.

Any reference objects are stored as a list in the SeqRecord object's annotations dictionary under the key 'references'. That's all there is too it. References are meant to be easy to deal with, and hopefully general enough to cover lots of usage cases.

5.5 The format method

The format method of the SeqRecord class gives a string containing your record formatted using one of the output file formats supported by Bio.SeqIO, such as FASTA:

```
In [30]: record = SeqRecord(Seq("MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD" \
                               +"GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK" \
                               +"NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM" \
                               +"SSAC", generic_protein),
                            id="gi|14150838|gb|AAK54648.1|AF376133_1",
                            description="chalcone synthase [Cucumis sativus]")
         print(record.format("fasta"))
```

```
>gi|14150838|gb|AAK54648.1|AF376133_1 chalcone synthase [Cucumis sativus]
MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD
GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK
NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM
SSAC
```

This format method takes a single mandatory argument, a lower case string which is supported by Bio.SeqIO as an output format. However, some of the file formats Bio.SeqIO can write to *require* more than one record (typically the case for multiple sequence alignment formats), and thus won't work via this format method.

5.6 Slicing a SeqRecord

You can slice a SeqRecord, to give you a new SeqRecord covering just part of the sequence. What is important here is that any per-letter annotations are also sliced, and any features which fall completely within the new sequence are preserved (with their locations adjusted).

For example, taking the same GenBank file used earlier:

```
In [31]: record = SeqIO.read("data/NC_005816.gb", "genbank")
         print(record)
         print(len(record))
         print(len(record.features))
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Database cross-references: Project:58037
Number of features: 41
/taxonomy=['Bacteria', 'Proteobacteria', 'Gammaproteobacteria', 'Enterobacteriales', 'Enterobacteria
/comment=PROVISIONAL REFSEQ: This record has not yet been subject to final
NCBI review. The reference sequence was derived from AE017046.
COMPLETENESS: full length.
/accessions=['NC_005816']
/organism=Yersinia pestis biovar Microtus str. 91001
/sequence_version=1
/source=Yersinia pestis biovar Microtus str. 91001
/data_file_division=BCT
/keywords=['']
/date=21-JUL-2008
/gi=45478711
/references=[Reference(title='Genetics of metabolic variations between Yersinia pestis biovars and th
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', IUPACAmbiguousDNA())
9609
41
```

For this example we're going to focus in on the pim gene, YP_pPCP05. If you have a look at the GenBank file directly you'll find this gene/CDS has location string 4343..4780, or in Python counting 4342:4780. From looking at the file you can work out that these are the twelfth and thirteenth entries in the file, so in Python zero-based counting they are entries 11 and 12 in the features list:

```
In [32]: print(record.features[20])
         print(record.features[21])
```

```
type: gene
location: [4342:4780](+)
qualifiers:
    Key: db_xref, Value: ['GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']

type: CDS
location: [4342:4780](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']
    Key: note, Value: ['similar to many previously sequenced pesticin immunity protein entries of Yer
    Key: product, Value: ['pesticin immunity protein']
    Key: protein_id, Value: ['NP_995571.1']
    Key: transl_table, Value: ['11']
    Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSHGIYGKQTTFKQTEFTNIKSNTK
```

Let's slice this parent record from 4300 to 4800 (enough to include the pim gene/CDS), and see how many features we get:

```
In [33]: sub_record = record[4300:4800]
         print(sub_record)
         sub_record
         print(sub_record)
         print(len(sub_record))
         print(len(sub_record.features))
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Number of features: 2
Seq('ATAAATAGATTATTCCAAATAATTTATTTATGTAAGAACAGGATGGGAGGGGGA...TTA', IUPACAmbiguousDNA())
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Number of features: 2
Seq('ATAAATAGATTATTCCAAATAATTTATTTATGTAAGAACAGGATGGGAGGGGGA...TTA', IUPACAmbiguousDNA())
500
2
```

Our sub-record just has two features, the gene and CDS entries for YP_pPCP05:

```
In [34]: print(sub_record.features[0])
         print(sub_record.features[1])
```

```
type: gene
location: [42:480](+)
qualifiers:
    Key: db_xref, Value: ['GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']

type: CDS
location: [42:480](+)
qualifiers:
    Key: codon_start, Value: ['1']
    Key: db_xref, Value: ['GI:45478716', 'GeneID:2767712']
    Key: gene, Value: ['pim']
    Key: locus_tag, Value: ['YP_pPCP05']
    Key: note, Value: ['similar to many previously sequenced pesticin immunity protein entries of Yer
    Key: product, Value: ['pesticin immunity protein']
    Key: protein_id, Value: ['NP_995571.1']
    Key: transl_table, Value: ['11']
    Key: translation, Value: ['MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNSLSHGIYGKQTTFKQTEFTNIKSNTK
```

Notice that their locations have been adjusted to reflect the new parent sequence!

While Biopython has done something sensible and hopefully intuitive with the features (and any per-letter annotation), for the other annotation it is impossible to know if this still applies to the sub-sequence or not. To avoid guessing, the annotations and dbxrefs are omitted from the sub-record, and it is up to you to transfer any relevant information as appropriate.

```
In [35]: print(sub_record.annotations)
         print(sub_record.dbxrefs)
```

```
{}
[]
```

The same point could be made about the record id, name and description, but for practicality these are preserved:

```
In [36]: print(sub_record.id)
         print(sub_record.name)
         print(sub_record.description)
```

```
NC_005816.1
NC_005816
Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
```

This illustrates the problem nicely though, our new sub-record is *not* the complete sequence of the plasmid, so the description is wrong! Let's fix this and then view the sub-record as a reduced GenBank file using the format method described above:

```
In [37]: sub_record.description = "Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, partial
         print(sub_record.format("genbank"))
```

```
LOCUS       NC_005816                500 bp    DNA              UNK 01-JAN-1980
DEFINITION  Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, partial.
ACCESSION   NC_005816
VERSION     NC_005816.1
KEYWORDS    .
SOURCE      .
  ORGANISM  .
            .
FEATURES             Location/Qualifiers
     gene            43..480
                     /db_xref="GeneID:2767712"
                     /gene="pim"
                     /locus_tag="YP_pPCP05"
     CDS             43..480
                     /codon_start=1
                     /db_xref="GI:45478716"
                     /db_xref="GeneID:2767712"
                     /gene="pim"
                     /locus_tag="YP_pPCP05"
                     /note="similar to many previously sequenced pesticin
                     immunity protein entries of Yersinia pestis plasmid pPCP,
                     e.g. gi| 16082683|,ref|NP_395230.1| (NC_003132) ,
                     gi|1200166|emb|CAA90861.1| (Z54145 ) , gi|1488655|
                     emb|CAA63439.1| (X92856) , gi|2996219|gb|AAC62543.1|
                     (AF053945) , and gi|5763814|emb|CAB531 67.1| (AL109969)"
                     /product="pesticin immunity protein"
                     /protein_id="NP_995571.1"
                     /transl_table=11
                     /translation="MGGGMISKLFCLALIFLSSSGLAEKNTYTAKDILQNLELNTFGNS
                     LSHGIYGKQTTFKQTEFTNIKSNTKKHIALINKDNSWMISLKILGIKRDEYTVCFEDFS
                     LIRPPTYVAIHPLLIKKVKSGNFIVVKEIKKSIPGCTVYYH"
ORIGIN
        1 ataaatagat tattccaaat aatttattta tgtaagaaca ggatgggagg gggaatgatc
       61 tcaaagttat tttgcttggc tctcatattt ttatcatcaa gtggccttgc agaaaaaaac
      121 acatatacag caaaagacat cttgcaaaac ctagaattaa atacctttgg caattcattg
      181 tctcatggca tctatgggaa acagacaacc ttcaagcaaa ccgagtttac aaatattaaa
      241 agcaacacca aaaaacacat tgcacttatc aataaagaca actcatggat gatatcatta
      301 aaaatactag gaattaagag agatgagtat actgtctgtt ttgaagattt ctctctaata
      361 agaccgccaa catatgtagc catacatcct ctacttataa aaaaagtaaa atctggaaac
      421 tttatagtag tgaaagaaat aaagaaatct atccctggtt gcactgtata ttatcattaa
      481 tagcaagccc ctcattatta
//
```

5.7 Adding SeqRecord objects

You can add SeqRecord objects together, giving a new SeqRecord. What is important here is that any common per-letter annotations are also added, all the features are preserved (with their locations adjusted), and any other common annotation is also kept (like the id, name and description).

For an example with per-letter annotation, we'll use the first record in a FASTQ file.

```
In [38]: record = next(SeqIO.parse("data/example.fastq", "fastq"))
         print(len(record))
         print(record.seq)
         print(record.letter_annotations["phred_quality"])
```

```
25
CCCTTCTTGTCTTCAGCGTTTCTCC
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26, 26, 26, 26, 23, 23]
```

Let's suppose this was Roche 454 data, and that from other information you think the TTT should be only TT. We can make a new edited record by first slicing the SeqRecord before and after the 'extra' third T:

```
In [39]: left = record[:20]
         print(left.seq)
         print(left.letter_annotations["phred_quality"])
         right = record[21:]
         print(right.seq)
         print(right.letter_annotations["phred_quality"])
```

```
CCCTTCTTGTCTTCAGCGTT
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26]
CTCC
[26, 26, 23, 23]
```

Now add the two parts together:

```
In [40]: edited = left + right
         print(len(edited))
         print(edited.seq)
         print(edited.letter_annotations["phred_quality"])
```

```
24
CCCTTCTTGTCTTCAGCGTTCTCC
[26, 26, 18, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 22, 26, 26, 26, 26, 26, 26, 23, 23]
```

Easy and intuitive? We hope so! You can make this shorter with just:

```
In [41]: edited = record[:20] + record[21:]
```

Now, for an example with features, we'll use a GenBank file. Suppose you have a circular genome:

```
In [42]: record = SeqIO.read("data/NC_005816.gb", "genbank")
         print(record)
         print(len(record))
         print(len(record.features))
         print(record.dbxrefs)
         print(record.annotations.keys())
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Database cross-references: Project:58037
Number of features: 41
/taxonomy=['Bacteria', 'Proteobacteria', 'Gammaproteobacteria', 'Enterobacteriales', 'Enterobacteria
/comment=PROVISIONAL REFSEQ: This record has not yet been subject to final
```

```
NCBI review. The reference sequence was derived from AE017046.
COMPLETENESS: full length.
/accessions=['NC_005816']
/organism=Yersinia pestis biovar Microtus str. 91001
/sequence_version=1
/source=Yersinia pestis biovar Microtus str. 91001
/data_file_division=BCT
/keywords=['']
/date=21-JUL-2008
/gi=45478711
/references=[Reference(title='Genetics of metabolic variations between Yersinia pestis biovars and th
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG', IUPACAmbiguousDNA())
9609
41
['Project:58037']
dict_keys(['taxonomy', 'comment', 'accessions', 'organism', 'sequence_version', 'source', 'data_file_
```

You can shift the origin like this:

```
In [43]: shifted = record[2000:] + record[:2000]
         print(shifted)
         print(len(shifted))
```

```
ID: NC_005816.1
Name: NC_005816
Description: Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, complete sequence.
Number of features: 40
Seq('GATACGCAGTCATATTTTTTACACAATTCTCTAATCCCGACAAGGTCGTAGGTC...GGA', IUPACAmbiguousDNA())
9609
```

Note that this isn't perfect in that some annotation like the database cross references and one of the features (the source feature) have been lost:

```
In [44]: print(len(shifted.features))
         print(shifted.dbxrefs)
         print(shifted.annotations.keys())
```

```
40
[]
dict_keys([])
```

This is because the SeqRecord slicing step is cautious in what annotation it preserves (erroneously propagating annotation can cause major problems). If you want to keep the database cross references or the annotations dictionary, this must be done explicitly:

```
In [45]: shifted.dbxrefs = record.dbxrefs[:]
         shifted.annotations = record.annotations.copy()
         print(shifted.dbxrefs)
         print(shifted.annotations.keys())
```

```
['Project:58037']
dict_keys(['accessions', 'source', 'data_file_division', 'comment', 'gi', 'references', 'taxonomy',
```

Also note that in an example like this, you should probably change the record identifiers since the NCBI references refer to the *original* unmodified sequence.

5.8 Reverse-complementing SeqRecord objects

One of the new features in Biopython 1.57 was the SeqRecord object's reverse_complement method. This tries to balance easy of use with worries about what to do with the annotation in the reverse complemented record.

For the sequence, this uses the Seq object's reverse complement method. Any features are transferred with the location and strand recalculated. Likewise any per-letter-annotation is also copied but reversed (which makes sense for typical examples like quality scores). However, transfer of most annotation is problematical.

For instance, if the record ID was an accession, that accession should not really apply to the reverse complemented sequence, and transferring the identifier by default could easily cause subtle data corruption in downstream analysis. Therefore by default, the SeqRecord's id, name, description, annotations and database cross references are all *not* transferred by default.

The SeqRecord object's reverse_complement method takes a number of optional arguments corresponding to properties of the record. Setting these arguments to True means copy the old values, while False means drop the old values and use the default value. You can alternatively provide the new desired value instead.

Consider this example record:

```
In [46]: record = SeqIO.read("data/NC_005816.gb", "genbank")
         print("%s %i %i %i %i" % (record.id, len(record), len(record.features), len(record.dbxrefs),
```

```
NC_005816.1 9609 41 1 11
```

Here we take the reverse complement and specify a new identifier - but notice how most of the annotation is dropped (but not the features):

```
In [47]: rc = record.reverse_complement(id="TESTING")
         print("%s %i %i %i %i" % (rc.id, len(rc), len(rc.features), len(rc.dbxrefs), len(rc.annotati
```

```
TESTING 9609 41 0 0
```

Source of the materials: Biopython Tutorial and Cookbook (adapted)

CHAPTER 6

Sequence Input/Output

In this noteboo we'll discuss in more detail the Bio.SeqIO module, which was briefly introduced before. This aims to provide a simple interface for working with assorted sequence file formats in a uniform way. See also the Bio.SeqIO wiki page (http://biopython.org/wiki/SeqIO), and the built in documentation (also http://biopython.org/DIST/docs/api/Bio.SeqIO-module.html :

```
In [1]: import gzip
        from io import StringIO

        from Bio import Entrez
        from Bio import ExPASy
        from Bio import SeqIO
        from Bio.Alphabet import generic_protein
        from Bio.Seq import Seq
        from Bio.SeqRecord import SeqRecord
        from Bio.SeqUtils.CheckSum import seguid
```

The catch is that you have to work with SeqRecord objects plus annotation like an identifier and description.

6.1 Parsing or Reading Sequences

The workhorse function Bio.SeqIO.parse() is used to read in sequence data as SeqRecord objects. This function expects two arguments:

- The first argument is a *handle* to read the data from, or a filename. A handle is typically a file opened for reading, but could be the output from a command line program, or data downloaded from the internet.
- The second argument is a lower case string specifying sequence format – we don't try and guess the file format for you! See http://biopython.org/wiki/SeqIO for a full listing of supported formats.

There is an optional argument alphabet to specify the alphabet to be used. This is useful for file formats like FASTA where otherwise Bio.SeqIO will default to a generic alphabet.

The Bio.SeqIO.parse() function returns an *iterator* which gives SeqRecord objects. Iterators are typically used in a for loop as shown below.

Sometimes you'll find yourself dealing with files which contain only a single record. For this situation use the function Bio.SeqIO.read() which takes the same arguments. Provided there is one and only one record in the file, this is returned as a SeqRecord object. Otherwise an exception is raised.

6.2 Reading Sequence Files

In general Bio.SeqIO.parse() is used to read in sequence files as SeqRecord objects, and is typically used with a for loop like this:

```
In [2]: # we show the first 3 only
        for i, seq_record in enumerate(SeqIO.parse("data/ls_orchid.fasta", "fasta")):
            print(seq_record.id)
            print(repr(seq_record.seq))
            print(len(seq_record))
            if i == 2:
                break
```

```
gi|2765658|emb|Z78533.1|CIZ78533
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', SingleLetterAlphabet())
740
gi|2765657|emb|Z78532.1|CCZ78532
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC', SingleLetterAlphabet())
753
gi|2765656|emb|Z78531.1|CFZ78531
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA', SingleLetterAlphabet())
748
```

The above example is repeated from a previous notebook, and will load the orchid DNA sequences in the FASTA format file. If instead you wanted to load a GenBank format file then all you need to do is change the filename and the format string:

```
In [3]: #we show the frist 3
        for i, seq_record in enumerate(SeqIO.parse("data/ls_orchid.gbk", "genbank")):
            print(seq_record.id)
            print(seq_record.seq)
            print(len(seq_record))
            if i == 2:
                break
```

```
Z78533.1
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTGAATCCGGAGGACCGGTGTACTCAGCTCACC
740
Z78532.1
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTGAATCTGGAGGACCTGTGGTAACTCAGCTCG
753
Z78531.1
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAGAACATACGATCGAGTGAATCCGGAGGACCCGTGGTTACACGGCTCA
748
```

Similarly, if you wanted to read in a file in another file format, then assuming Bio.SeqIO.parse() supports it you would just need to change the format string as appropriate, for example 'swiss' for SwissProt files or 'embl' for EMBL text files. There is a full listing on the wiki page (http://biopython.org/wiki/SeqIO) and in the built in documentation (also **:raw-latex:'\http'**://biopython.org/DIST/docs/api/Bio.SeqIO-module.html).

Another very common way to use a Python iterator is within a list comprehension (or a generator expression). For example, if all you wanted to extract from the file was a list of the record identifiers we can easily do this with the following list comprehension:

```
In [4]: [seq_record.id for seq_record in SeqIO.parse("data/ls_orchid.gbk", "genbank")][:10]  # ten o
```

```
Out[4]: ['Z78533.1',
          'Z78532.1',
          'Z78531.1',
          'Z78530.1',
          'Z78529.1',
          'Z78527.1',
          'Z78526.1',
          'Z78525.1',
          'Z78524.1',
          'Z78523.1']
```

6.2.1 Iterating over the records in a sequence file

In the above examples, we have usually used a for loop to iterate over all the records one by one. You can use the for loop with all sorts of Python objects (including lists, tuples and strings) which support the iteration interface.

The object returned by Bio.SeqIO is actually an iterator which returns SeqRecord objects. You get to see each record in turn, but once and only once. The plus point is that an iterator can save you memory when dealing with large files.

Instead of using a for loop, can also use the next() function on an iterator to step through the entries, like this:

```
In [5]: record_iterator = SeqIO.parse("data/ls_orchid.fasta", "fasta")

        first_record = next(record_iterator)
        print(first_record.id)
        print(first_record.description)

        second_record = next(record_iterator)
        print(second_record.id)
        print(second_record.description)
```

```
gi|2765658|emb|Z78533.1|CIZ78533
gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
gi|2765657|emb|Z78532.1|CCZ78532
gi|2765657|emb|Z78532.1|CCZ78532 C.californicum 5.8S rRNA gene and ITS1 and ITS2 DNA
```

Note that if you try to use next() and there are no more results, you'll get the special StopIteration exception.

One special case to consider is when your sequence files have multiple records, but you only want the first one. In this situation the following code is very concise:

```
In [6]: next(SeqIO.parse("data/ls_orchid.gbk", "genbank"))
```

```
Out[6]: SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbigu
```

A word of warning here – using the next() function like this will silently ignore any additional records in the file. If your files have *one and only one* record, like some of the online examples later in this chapter, or a GenBank file for a single chromosome, then use the new Bio.SeqIO.read() function instead. This will check there are no extra unexpected records present.

6.2.2 Getting a list of the records in a sequence file

In the previous section we talked about the fact that Bio.SeqIO.parse() gives you a SeqRecord iterator, and that you get the records one by one. Very often you need to be able to access the records in any order. The Python list data type is perfect for this, and we can turn the record iterator into a list of SeqRecord objects using the built-in Python function list() like so:

```
In [7]: records = list(SeqIO.parse("data/ls_orchid.gbk", "genbank"))
```

```
        print("Found %i records" % len(records))

        print("The last record")
        last_record = records[-1] #using Python's list tricks
        print(last_record.id)
        print(repr(last_record.seq))
        print(len(last_record))

        print("The first record")
        first_record = records[0] #remember, Python counts from zero
        print(first_record.id)
        print(repr(first_record.seq))
        print(len(first_record))
```

```
Found 94 records
The last record
Z78439.1
Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())
592
The first record
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
740
```

You can of course still use a for loop with a list of SeqRecord objects. Using a list is much more flexible than an iterator (for example, you can determine the number of records from the length of the list), but does need more memory because it will hold all the records in memory at once.

6.2.3 Extracting data

The SeqRecord object and its annotation structures are described more fully in in another notebook. As an example of how annotations are stored, we'll look at the output from parsing the first record in the orchid GenBank file.

```
In [8]: record_iterator = SeqIO.parse("data/ls_orchid.gbk", "genbank")
        first_record = next(record_iterator)
        print(first_record)
```

```
ID: Z78533.1
Name: Z78533
Description: C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA.
Number of features: 5
/organism=Cypripedium irapeanum
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta', 'Tracheophyta', 'Spermatophyt
/sequence_version=1
/keywords=['5.8S ribosomal RNA', '5.8S rRNA gene', 'internal transcribed spacer', 'ITS1', 'ITS2']
/date=30-NOV-2006
/accessions=['Z78533']
/source=Cypripedium irapeanum
/references=[Reference(title='Phylogenetics of the slipper orchids (Cypripedioideae: Orchidaceae): nu
/gi=2765658
/data_file_division=PLN
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())
```

This gives a human readable summary of most of the annotation data for the SeqRecord. For this example we're going to use the .annotations attribute which is just a Python dictionary. The contents of this annotations dictionary were shown when we printed the record above. You can also print them out directly:

print(first_record.annotations)

Like any Python dictionary, you can easily get a list of the keys:

print(first_record.annotations.keys())

or values:

```
print(first_record.annotations.values())
```

In general, the annotation values are strings, or lists of strings. One special case is any references in the file get stored as reference objects.

Suppose you wanted to extract a list of the species from the GenBank file. The information we want, *Cypripedium irapeanum*, is held in the annotations dictionary under 'source' and 'organism', which we can access like this:

```
In [9]: print(first_record.annotations["source"])
        print(first_record.annotations["organism"])
```

```
Cypripedium irapeanum
Cypripedium irapeanum
```

In general, 'organism' is used for the scientific name (in Latin, e.g. *Arabidopsis thaliana*), while 'source' will often be the common name (e.g. thale cress). In this example, as is often the case, the two fields are identical.

Now let's go through all the records, building up a list of the species each orchid sequence is from:

```
In [10]: all_species = []
         for seq_record in SeqIO.parse("data/ls_orchid.gbk", "genbank"):
             all_species.append(seq_record.annotations["organism"])
         print(all_species[:10])  # we print only 10
```

```
['Cypripedium irapeanum', 'Cypripedium californicum', 'Cypripedium fasciculatum', 'Cypripedium marga
```

Another way of writing this code is to use a list comprehension:

```
In [11]: all_species = [seq_record.annotations["organism"] for seq_record in \
                        SeqIO.parse("data/ls_orchid.gbk", "genbank")]
         print(all_species[:10])
```

```
['Cypripedium irapeanum', 'Cypripedium californicum', 'Cypripedium fasciculatum', 'Cypripedium marga
```

Great. That was pretty easy because GenBank files are annotated in a standardised way.

Now, let's suppose you wanted to extract a list of the species from a FASTA file, rather than the GenBank file. The bad news is you will have to write some code to extract the data you want from the record's description line - if the information is in the file in the first place! Our example FASTA format file starts like this:

```
>gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTG
AATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGG
...
```

You can check by hand, but for every record the species name is in the description line as the second word. This means if we break up each record's .description at the spaces, then the species is there as field number one (field zero is the record identifier). That means we can do this:

```
In [12]: all_species = []
         for seq_record in SeqIO.parse("data/ls_orchid.fasta", "fasta"):
             all_species.append(seq_record.description.split()[1])
         print(all_species[:10])
```

```
['C.irapeanum', 'C.californicum', 'C.fasciculatum', 'C.margaritaceum', 'C.lichiangense', 'C.yatabean
```

The concise alternative using list comprehensions would be:

```
In [13]: all_species == [seq_record.description.split()[1] for seq_record in \
                          SeqIO.parse("data/ls_orchid.fasta", "fasta")]
         print(all_species[:10])
```

```
['C.irapeanum', 'C.californicum', 'C.fasciculatum', 'C.margaritaceum', 'C.lichiangense', 'C.yatabeanu
```

In general, extracting information from the FASTA description line is not very nice. If you can get your sequences in a well annotated file format like GenBank or EMBL, then this sort of annotation information is much easier to deal with.

6.3 Parsing sequences from compressed files

In the previous section, we looked at parsing sequence data from a file. Instead of using a filename, you can give Bio.SeqIO a handle, and in this section we'll use handles to parse sequence from compressed files.

As you'll have seen above, we can use Bio.SeqIO.read() or Bio.SeqIO.parse() with a filename - for instance this quick example calculates the total length of the sequences in a multiple record GenBank file using a generator expression:

```
In [14]: print(sum(len(r) for r in SeqIO.parse("data/ls_orchid.gbk", "gb")))
```

```
67518
```

Here we use a file handle instead, using the **:raw-latex:‘\verb|with|‘** statement to close the handle automatically:

```
In [15]: with open("data/ls_orchid.gbk") as handle:
             print(sum(len(r) for r in SeqIO.parse(handle, "gb")))
```

```
67518
```

Or, the old fashioned way where you manually close the handle:

```
In [16]: handle = open("data/ls_orchid.gbk")
         print(sum(len(r) for r in SeqIO.parse(handle, "gb")))
         handle.close()
```

```
67518
```

Now, suppose we have a gzip compressed file instead? These are very commonly used on Linux. We can use Python's gzip module to open the compressed file for reading - which gives us a handle object:

```
In [17]: handle = gzip.open("data/ls_orchid.gbk.gz", "rt")
         print(sum(len(r) for r in SeqIO.parse(handle, "gb")))
         handle.close()
```

```
67518
```

There is a gzip (GNU Zip) variant called BGZF (Blocked GNU Zip Format), which can be treated like an ordinary gzip file for reading, but has advantages for random access later which we'll talk about later.

6.4 Parsing sequences from the net

In the previous sections, we looked at parsing sequence data from a file (using a filename or handle), and from compressed files (using a handle). Here we'll use Bio.SeqIO with another type of handle, a network connection, to download and parse sequences from the internet.

Note that just because you *can* download sequence data and parse it into a SeqRecord object in one go doesn't mean this is a good idea. In general, you should probably download sequences *once* and save them to a file for reuse.

6.4.1 Parsing GenBank records from the net

Let’s just connect to the NCBI and get a few *Opuntia* (prickly-pear) sequences from GenBank using their GI numbers.

First of all, let’s fetch just one record. If you don’t care about the annotations and features downloading a FASTA file is a good choice as these are compact. Now remember, when you expect the handle to contain one and only one record, use the Bio.SeqIO.read() function:

```
In [18]: Entrez.email = "A.N.Other@example.com"
         handle = Entrez.efetch(db="nucleotide", rettype="fasta", retmode="text", id="6273291")
         seq_record = SeqIO.read(handle, "fasta")
         handle.close()
         print("%s with %i features" % (seq_record.id, len(seq_record.features)))
```

```
gi|6273291|gb|AF191665.1|AF191665 with 0 features
```

The NCBI will also let you ask for the file in other formats, in particular as a GenBank file. Until Easter 2009, the Entrez EFetch API let you use ‘‘genbank” as the return type, however the NCBI now insist on using the official return types of ‘gb’ (or ‘gp’ for proteins) as described on http://www.ncbi.nlm.nih.gov/entrez/query/static/efetchseq_help.html.

As a result we support ‘gb’ as an alias for ‘genbank’ in Bio.SeqIO.

```
In [19]: Entrez.email = "A.N.Other@example.com"
         handle = Entrez.efetch(db="nucleotide", rettype="gb", retmode="text", id="6273291")
         seq_record = SeqIO.read(handle, "gb") #using "gb" as an alias for "genbank"
         handle.close()
         print("%s with %i features" % (seq_record.id, len(seq_record.features)))
```

```
AF191665.1 with 3 features
```

Notice this time we have three features.

Now let’s fetch several records. This time the handle contains multiple records, so we must use the **:raw-latex:‘\verb|Bio.SeqIO.parse()|‘** function:

```
In [20]: Entrez.email = "A.N.Other@example.com"
         handle = Entrez.efetch(db="nucleotide", rettype="gb", retmode="text",
                                id="6273291,6273290,6273289")
         for seq_record in SeqIO.parse(handle, "gb"):
             print (seq_record.id, seq_record.description[:50] + "...")
             print ("Sequence length %i," % len(seq_record))
             print ("%i features," % len(seq_record.features))
             print ("from: %s" % seq_record.annotations["source"])
         handle.close()
```

```
AF191665.1 Opuntia marenae rpl16 gene; chloroplast gene for c...
Sequence length 902,
3 features,
from: chloroplast Grusonia marenae
AF191664.1 Opuntia clavata rpl16 gene; chloroplast gene for c...
Sequence length 899,
3 features,
from: chloroplast Grusonia clavata
AF191663.1 Opuntia bradtiana rpl16 gene; chloroplast gene for...
Sequence length 899,
3 features,
from: chloroplast Grusonia bradtiana
```

6.4.2 Parsing SwissProt sequences from the net

Now let's use a handle to download a SwissProt file from ExPASy (SwissProt will be covered in detail in another notebook). As mentioned above, when you expect the handle to contain one and only one record, use the Bio.SeqIO.read() function:

```
In [21]: handle = ExPASy.get_sprot_raw("O23729")
         seq_record = SeqIO.read(handle, "swiss")
         handle.close()
         print(seq_record.id)
         print(seq_record.name)
         print(seq_record.description)
         print(repr(seq_record.seq))
         print("Length %i" % len(seq_record))
         print(seq_record.annotations["keywords"])
```

```
O23729
CHS3_BROFI
RecName: Full=Chalcone synthase 3; EC=2.3.1.74; AltName: Full=Naringenin-chalcone synthase 3;
Seq('MAPAMEEIRQAQRAEGPAAVLAIGTSTPPNALYQADYPDYYFRITKSEHLTELK...GAE', ProteinAlphabet())
Length 394
['Acyltransferase', 'Flavonoid biosynthesis', 'Transferase']
```

6.5 Sequence files as dictionaries

We're now going to introduce three related functions in the **:raw-latex:'\verb|Bio.SeqIO|'** module which allow dictionary like random access to a multi-sequence file. There is a trade off here between flexibility and memory usage. In summary:

- **Bio.SeqIO.to_dict()** is the most flexible but also the most memory demanding option. This is basically a helper function to build a normal Python dictionary with each entry held as a SeqRecord object in memory, allowing you to modify the records.
- **Bio.SeqIO.index()** is a useful middle ground, acting like a read only dictionary and parsing sequences into SeqRecord objects on demand.
- **Bio.SeqIO.index_db()** also acts like a read only dictionary but stores the identifiers and file offsets in a file on disk (as an SQLite3 database), meaning it has very low memory requirements, but will be a little bit slower.

6.5.1 Sequence files as Dictionaries – In memory

The next thing that we'll do with our ubiquitous orchid files is to show how to index them and access them like a database using the Python dictionary data type. This is very useful for moderately large files where you only need to access certain elements of the file, and makes for a nice quick 'n dirty database. For dealing with larger files where memory becomes a problem, see below.

You can use the function Bio.SeqIO.to_dict() to make a SeqRecord dictionary (in memory). By default this will use each record's identifier (i.e. the .id attribute) as the key. Let's try this using our GenBank file:

```
In [22]: orchid_dict = SeqIO.to_dict(SeqIO.parse("data/ls_orchid.gbk", "genbank"))
```

There is just one required argument for Bio.SeqIO.to_dict(), a list or generator giving SeqRecord objects. Here we have just used the output from the SeqIO.parse function. As the name suggests, this returns a Python dictionary.

Since this variable orchid_dict is an ordinary Python dictionary, we can look at all of the keys we have available:

```
In [23]: print(len(orchid_dict))
         print(orchid_dict.keys())
```

```
94
dict_keys(['Z78459.1', 'Z78496.1', 'Z78501.1', 'Z78443.1', 'Z78514.1', 'Z78452.1', 'Z78442.1', 'Z784
```

If you really want to, you can even look at all the records at once:

```
In [24]: list(orchid_dict.values())[:5]  # Ok not all at once...
```

```
Out[24]: [SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...TTT', IUPACAmb
          SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCGCAT...AGC', IUPACAmb
          SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACCGCAA...AGA', IUPACAmb
          SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...AGG', IUPACAmb
          SeqRecord(seq=Seq('CGTAACAAGGTTTCCGTAGGTGGACCTTCGGGAGGATCATTTTTGAAGCCCCCA...CTA', IUPACAmb
```

We can access a single **:raw-latex:‘\verb|SeqRecord|‘** object via the keys and manipulate the object as normal:

```
In [25]: seq_record = orchid_dict["Z78475.1"]
         print(seq_record.description)
         print(repr(seq_record.seq))
```

```
P.supardii 5.8S rRNA gene and ITS1 and ITS2 DNA.
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACAT...GGT', IUPACAmbiguousDNA())
```

So, it is very easy to create an in memory ‘database’ of our GenBank records. Next we’ll try this for the FASTA file instead.

Note that those of you with prior Python experience should all be able to construct a dictionary like this ‘by hand’. However, typical dictionary construction methods will not deal with the case of repeated keys very nicely. Using the Bio.SeqIO.to_dict() will explicitly check for duplicate keys, and raise an exception if any are found.

Specifying the dictionary keys

Using the same code as above, but for the FASTA file instead:

```
In [26]: orchid_dict = SeqIO.to_dict(SeqIO.parse("data/ls_orchid.fasta", "fasta"))
         print(list(orchid_dict.keys())[:5])
```

```
['gi|2765611|emb|Z78486.1|PBZ78486', 'gi|2765630|emb|Z78505.1|PSZ78505', 'gi|2765584|emb|Z78459.1|PD
```

You should recognise these strings from when we parsed the FASTA file earlier. Suppose you would rather have something else as the keys - like the accession numbers. This brings us nicely to SeqIO.to_dict()’s optional argument key_function, which lets you define what to use as the dictionary key for your records.

First you must write your own function to return the key you want (as a string) when given a SeqRecord object. In general, the details of function will depend on the sort of input records you are dealing with. But for our orchids, we can just split up the record’s identifier using the ‘pipe’ character (the vertical line) and return the fourth entry (field three):

```
In [27]: def get_accession(record):
             """"Given a SeqRecord, return the accession number as a string.

             e.g. "gi|2765613|emb|Z78488.1|PTZ78488" -> "Z78488.1"
             """
             parts = record.id.split("|")
             assert len(parts) == 5 and parts[0] == "gi" and parts[2] == "emb"
             return parts[3]
```

Then we can give this function to the SeqIO.to_dict() function to use in building the dictionary:

```
In [28]: orchid_dict = SeqIO.to_dict(SeqIO.parse("data/ls_orchid.fasta", "fasta"), key_function=get_a
         print(orchid_dict.keys())
```

```
dict_keys(['Z78459.1', 'Z78496.1', 'Z78501.1', 'Z78443.1', 'Z78514.1', 'Z78452.1', 'Z78442.1', 'Z784
```

Finally, as desired, the new dictionary keys:

Indexing a dictionary using the SEGUID checksum

To give another example of working with dictionaries of SeqRecord objects, we'll use the SEGUID checksum function. This is a relatively recent checksum, and collisions should be very rare (i.e. two different sequences with the same checksum), an improvement on the CRC64 checksum.

Once again, working with the orchids GenBank file:

```
In [29]: for i, record in enumerate(SeqIO.parse("data/ls_orchid.gbk", "genbank")):
             print(record.id, seguid(record.seq))
             if i == 4:  # OK, 5 is enough!
                 break
```

```
Z78533.1 JUEoWn6DPhgZ9nAyowsgtoD9TTo
Z78532.1 MN/s0q9zDoCVEEc+k/IFwCNF2pY
Z78531.1 xN45pACrTnmBH8a8Y9cWSgoLrwE
Z78530.1 yMhI5UUQfFOPcoJXb9B19XUyYlY
Z78529.1 s1Pnjq9zoSHoI/CG9jQr4GyeMZY
```

Now, recall the Bio.SeqIO.to_dict() function's key_function argument expects a function which turns a SeqRecord into a string. We can't use the seguid() function directly because it expects to be given a Seq object (or a string). However, we can use Python's lambda feature to create a 'one off' function to give to Bio.SeqIO.to_dict() instead:

```
In [30]: seguid_dict = SeqIO.to_dict(SeqIO.parse("data/ls_orchid.gbk", "genbank"),
                                     lambda rec : seguid(rec.seq))
         record = seguid_dict["MN/s0q9zDoCVEEc+k/IFwCNF2pY"]
         print(record.id)
         print(record.description)
```

```
Z78532.1
C.californicum 5.8S rRNA gene and ITS1 and ITS2 DNA.
```

That should have retrieved the record Z78532.1, the second entry in the file.

6.5.2 Sequence files as Dictionaries – Indexed files

As the previous couple of examples tried to illustrate, using Bio.SeqIO.to_dict() is very flexible. However, because it holds everything in memory, the size of file you can work with is limited by your computer's RAM. In general, this will only work on small to medium files.

For larger files you should consider Bio.SeqIO.index(), which works a little differently. Although it still returns a dictionary like object, this does *not* keep *everything* in memory. Instead, it just records where each record is within the file – when you ask for a particular record, it then parses it on demand.

As an example, let's use the same GenBank file as before:

```
In [31]: orchid_dict = SeqIO.index("data/ls_orchid.gbk", "genbank")
         print(len(orchid_dict))
         print(orchid_dict.keys())
         seq_record = orchid_dict["Z78475.1"]
         print(seq_record.description)
         print(seq_record.seq)
```

```
94
<dict_keyiterator object at 0x7f92241b0db8>
P.supardii 5.8S rRNA gene and ITS1 and ITS2 DNA.
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCAGTTTACTTTGGTCACCC
```

Note that Bio.SeqIO.index() won't take a handle, but only a filename. There are good reasons for this, but it is a little technical. The second argument is the file format (a lower case string as used in the other Bio.SeqIO functions). You can use many other simple file formats, including FASTA and FASTQ files. However, alignment formats like PHYLIP or Clustal are not supported. Finally as an optional argument you can supply an alphabet, or a key function.

Here is the same example using the FASTA file - all we change is the filename and the format name:

```
In [32]: orchid_dict = SeqIO.index("data/ls_orchid.fasta", "fasta")
         print(len(orchid_dict))
         print(list(orchid_dict.keys()))
```

```
94
['gi|2765611|emb|Z78486.1|PBZ78486', 'gi|2765630|emb|Z78505.1|PSZ78505', 'gi|2765584|emb|Z78459.1|PDZ
```

6.5.3 Specifying the dictionary keys

Suppose you want to use the same keys as before? You'll need to write a tiny function to map from the FASTA identifier (as a string) to the key you want:

```
In [33]: def get_acc(identifier):
             """"Given a SeqRecord identifier string, return the accession number as a string.

             e.g. "gi|2765613|emb|Z78488.1|PTZ78488" -> "Z78488.1"
             """
             parts = identifier.split("|")
             assert len(parts) == 5 and parts[0] == "gi" and parts[2] == "emb"
             return parts[3]
```

Then we can give this function to the Bio.SeqIO.index() function to use in building the dictionary:

```
In [ ]: orchid_dict = SeqIO.index("data/ls_orchid.fasta", "fasta", key_function=get_acc)
        print(list(orchid_dict.keys()))
```

```
['Z78459.1', 'Z78496.1', 'Z78501.1', 'Z78443.1', 'Z78514.1', 'Z78452.1', 'Z78442.1', 'Z78439.1', 'Z78
```

Getting the raw data for a record

The dictionary-like object from Bio.SeqIO.index() gives you each entry as a SeqRecord object. However, it is sometimes useful to be able to get the original raw data straight from the file. For this use the get_raw() method which takes a single argument (the record identifier) and returns a string (extracted from the file without modification).

A motivating example is extracting a subset of a records from a large file where either Bio.SeqIO.write() does not (yet) support the output file format (e.g. the plain text SwissProt file format) or where you need to preserve the text exactly (e.g. GenBank or EMBL output from Biopython does not yet preserve every last bit of annotation).

Let's suppose you have download the whole of UniProt in the plain text SwissPort file format from their FTP site (ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.dat.gz - Careful big download) and uncompressed it as the file uniprot_sprot.dat, and you want to extract just a few records from it:

[We should find a smaller example here, Tiago]

```
In [ ]: #Use this to download the file
        !wget -c ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/u
        !gzip -d data/uniprot_sprot.dat.gz
```

```
--2015-12-25 07:24:10--  ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/co
           => 'data/uniprot_sprot.dat.gz'
Resolving ftp.uniprot.org (ftp.uniprot.org)... 141.161.180.197
Connecting to ftp.uniprot.org (ftp.uniprot.org)|141.161.180.197|:21...
```

```
In [ ]: uniprot = SeqIO.index("data/uniprot_sprot.dat", "swiss")
        handle = open("data/selected.dat", "w")
        for acc in ["P33487", "P19801", "P13689", "Q8JZQ5", "Q9TRC7"]:
            handle.write(uniprot.get_raw(acc).decode("utf-8"))
        handle.close()
```

There is a longer example in sorting section using the SeqIO.index() function to sort a large sequence file (without loading everything into memory at once).

6.5.4 Sequence files as Dictionaries – Database indexed files

Biopython 1.57 introduced an alternative, Bio.SeqIO.index_db(), which can work on even extremely large files since it stores the record information as a file on disk (using an SQLite3 database) rather than in memory. Also, you can index multiple files together (providing all the record identifiers are unique).

The Bio.SeqIO.index() function takes three required arguments:

- Index filename, we suggest using something ending .idx. This index file is actually an SQLite3 database.
- List of sequence filenames to index (or a single filename)
- File format (lower case string as used in the rest of the SeqIO module).

As an example, consider the GenBank flat file releases from the NCBI FTP site, ftp://ftp.ncbi.nih.gov/genbank/, which are gzip compressed GenBank files. As of GenBank release 182, there are 16 files making up the viral sequences, gbvrl1.seq, ..., gbvrl16.seq, containing in total almost one million records. You can index them like this:

```
In [ ]: #this will download the files - Currently there are more than 16, but we will do only 4
        import os
        for i in range(1, 5):
            os.system('wget ftp://ftp.ncbi.nih.gov/genbank/gbvrl%i.seq.gz -O data/gbvrl%i.seq.gz' %
            os.system('gzip -d data/gbvrl%i.seq.gz' % i)
```

```
In [ ]: files = ["data/gbvrl%i.seq" % i for i in range(1, 5)]
        gb_vrl = SeqIO.index_db("data/gbvrl.idx", files, "genbank")
        print("%i sequences indexed" % len(gb_vrl))
```

That takes about two minutes to run on my machine. If you rerun it then the index file (here gbvrl.idx) is reloaded in under a second. You can use the index as a read only Python dictionary - without having to worry about which file the sequence comes from, e.g.

```
In [ ]: print(next(gb_vrl.keys()))
```

```
In [ ]: print(gb_vrl["AB000048.1"].description)
```

Getting the raw data for a record

Just as with the Bio.SeqIO.index() function, the dictionary like object also lets you get at the raw text of each record:

```
In [ ]: print(gb_vrl.get_raw("AB000048.1"))
```

6.5.5 Indexing compressed files

Very often when you are indexing a sequence file it can be quite large - so you may want to compress it on disk. Unfortunately efficient random access is difficult with the more common file formats like gzip and bzip2. In this setting, BGZF (Blocked GNU Zip Format) can be very helpful. This is a variant of gzip (and can be decompressed using standard gzip tools) popularised by the BAM file format, http://samtools.sourceforge.net/, and http://samtools.sourceforge.net/tabix.shtml.

To create a BGZF compressed file you can use the command line tool bgzip which comes with samtools. In our examples we use a filename extension *.bgz, so they can be distinguished from normal gzipped files (named *.gz). You can also use the Bio.bgzf module to read and write BGZF files from within Python.

The Bio.SeqIO.index() and Bio.SeqIO.index_db() can both be used with BGZF compressed files. For example, if you started with an uncompressed GenBank file:

```
In [ ]: orchid_dict = SeqIO.index("data/ls_orchid.gbk", "genbank")
        print(len(orchid_dict))
```

You could compress this (while keeping the original file) at the command line using the following command:

```
$ bgzip -c ls_orchid.gbk > ls_orchid.gbk.bgz
```

You can use the compressed file in exactly the same way:

```
In [ ]: orchid_dict = SeqIO.index("data/ls_orchid.gbk.bgz", "genbank")
        print(len(orchid_dict))
```

or

```
In [ ]: orchid_dict = SeqIO.index_db("data/ls_orchid.gbk.bgz.idx", "data/ls_orchid.gbk.bgz", "genban
        print(len(orchid_dict))
```

The SeqIO indexing automatically detects the BGZF compression. Note that you can't use the same index file for the uncompressed and compressed files.

6.5.6 Discussion

So, which of these methods should you use and why? It depends on what you are trying to do (and how much data you are dealing with). However, in general picking Bio.SeqIO.index() is a good starting point. If you are dealing with millions of records, multiple files, or repeated analyses, then look at Bio.SeqIO.index_db().

Reasons to choose Bio.SeqIO.to_dict() over either Bio.SeqIO.index() or Bio.SeqIO.index_db() boil down to a need for flexibility despite its high memory needs. The advantage of storing the SeqRecord objects in memory is they can be changed, added to, or removed at will. In addition to the downside of high memory consumption, indexing can also take longer because all the records must be fully parsed.

Both Bio.SeqIO.index() and Bio.SeqIO.index_db() only parse records on demand. When indexing, they scan the file once looking for the start of each record and do as little work as possible to extract the identifier.

Reasons to choose Bio.SeqIO.index() over Bio.SeqIO.index_db() include:

- Faster to build the index (more noticeable in simple file formats)
- Slightly faster access as SeqRecord objects (but the difference is only really noticeable for simple to parse file formats).
- Can use any immutable Python object as the dictionary keys (e.g. a tuple of strings, or a frozen set) not just strings.
- Don't need to worry about the index database being out of date if the sequence file being indexed has changed.

Reasons to choose Bio.SeqIO.index_db() over Bio.SeqIO.index() include:

- Not memory limited - this is already important with files from second generation sequencing where 10s of millions of sequences are common, and using Bio.SeqIO.index() can require more than 4GB of RAM and therefore a 64bit version of Python.
- Because the index is kept on disk, it can be reused. Although building the index database file takes longer, if you have a script which will be rerun on the same datafiles in future, this could save time in the long run.
- Indexing multiple files together

- The get_raw() method can be much faster, since for most file formats the length of each record is stored as well as its offset.

6.6 Writing sequence files

We've talked about using Bio.SeqIO.parse() for sequence input (reading files), and now we'll look at Bio.SeqIO.write() which is for sequence output (writing files). This is a function taking three arguments: some SeqRecord objects, a handle or filename to write to, and a sequence format.

Here is an example, where we start by creating a few SeqRecord objects the hard way (by hand, rather than by loading them from a file):

```
In [ ]: rec1 = SeqRecord(Seq("MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD" \
                             +"GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK" \
                             +"NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM" \
                             +"SSAC", generic_protein),
                          id="gi|14150838|gb|AAK54648.1|AF376133_1",
                          description="chalcone synthase [Cucumis sativus]")

        rec2 = SeqRecord(Seq("YPDYYFRITNREHKAELKEKFQRMCDKSMIKKRYMYLTEEILKENPSMCEYMAPSLDARQ" \
                             +"DMVVVEIPKLGKEAAVKAIKEWGQ", generic_protein),
                          id="gi|13919613|gb|AAK33142.1|",
                          description="chalcone synthase [Fragaria vesca subsp. bracteata]")

        rec3 = SeqRecord(Seq("MVTVEEFRRAQCAEGPATVMAIGTATPSNCVDQSTYPDYYFRITNSEHKVELKEKFKRMC" \
                             +"EKSMIKKRYMHLTEEILKENPNICAYMAPSLDARQDIVVVEVPKLGKEAAQKAIKEWGQP" \
                             +"KSKITHLVFCTTSGVDMPGCDYQLTKLLGLRPSVKRFMMYQQGCFAGGTVLRMAKDLAEN" \
                             +"NKGARVLVVCSEITAVTFRGPNDTHLDSLVGQALFGDGAAAVIIGSDPIPEVERPLFELV" \
                             +"SAAQTLLPDSEGAIDGHLREVGLTFHLLKDVPGLISKNIEKSLVEAFQPLGISDWNSLFW" \
                             +"IAHPGGPAILDQVELKLGLKQEKLKATRKVLSNYGNMSSACVLFILDEMRKASAKEGLGT" \
                             +"TGEGLEWGVLFGFGPGLTVETVVLHSVAT", generic_protein),
                          id="gi|13925890|gb|AAK49457.1|",
                          description="chalcone synthase [Nicotiana tabacum]")

        my_records = [rec1, rec2, rec3]
```

Now we have a list of SeqRecord objects, we'll write them to a FASTA format file:

```
In [ ]: SeqIO.write(my_records, "data/my_example.faa", "fasta")
```

And if you open this file in your favourite text editor it should look like this:

```
>gi|14150838|gb|AAK54648.1|AF376133_1 chalcone synthase [Cucumis sativus]
MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD
GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK
NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM
SSAC
>gi|13919613|gb|AAK33142.1| chalcone synthase [Fragaria vesca subsp. bracteata]
YPDYYFRITNREHKAELKEKFQRMCDKSMIKKRYMYLTEEILKENPSMCEYMAPSLDARQ
DMVVVEIPKLGKEAAVKAIKEWGQ
>gi|13925890|gb|AAK49457.1| chalcone synthase [Nicotiana tabacum]
MVTVEEFRRAQCAEGPATVMAIGTATPSNCVDQSTYPDYYFRITNSEHKVELKEKFKRMC
EKSMIKKRYMHLTEEILKENPNICAYMAPSLDARQDIVVVEVPKLGKEAAQKAIKEWGQP
KSKITHLVFCTTSGVDMPGCDYQLTKLLGLRPSVKRFMMYQQGCFAGGTVLRMAKDLAEN
NKGARVLVVCSEITAVTFRGPNDTHLDSLVGQALFGDGAAAVIIGSDPIPEVERPLFELV
SAAQTLLPDSEGAIDGHLREVGLTFHLLKDVPGLISKNIEKSLVEAFQPLGISDWNSLFW
IAHPGGPAILDQVELKLGLKQEKLKATRKVLSNYGNMSSACVLFILDEMRKASAKEGLGT
TGEGLEWGVLFGFGPGLTVETVVLHSVAT
```

Suppose you wanted to know how many records the Bio.SeqIO.write() function wrote to the handle? If your records were in a list you could just use len(my_records), however you can't do that when your records come from a generator/iterator. TheBio.SeqIO.write() function returns the number of SeqRecord objects written to the file.

Note - If you tell the Bio.SeqIO.write() function to write to a file that already exists, the old file will be overwritten without any warning.

6.6.1 Round trips

Some people like their parsers to be 'round-tripable', meaning if you read in a file and write it back out again it is unchanged. This requires that the parser must extract enough information to reproduce the original file *exactly*. Bio.SeqIO does *not* aim to do this.

As a trivial example, any line wrapping of the sequence data in FASTA files is allowed. An identical SeqRecord would be given from parsing the following two examples which differ only in their line breaks:

```
>YAL068C-7235.2170 Putative promoter sequence
TACGAGAATAATTTCTCATCATCCAGCTTTAACACAAAATTCGCACAGTTTTCGTTAAGA
GAACTTAACATTTTCTTATGACGTAAATGAAGTTTATATATAAATTTCCTTTTTATTGGA

>YAL068C-7235.2170 Putative promoter sequence
TACGAGAATAATTTCTCATCATCCAGCTTTAACACAAAATTCGCA
CAGTTTTCGTTAAGAGAACTTAACATTTTCTTATGACGTAAATGA
AGTTTATATATAAATTTCCTTTTTATTGGA
```

To make a round-tripable FASTA parser you would need to keep track of where the sequence line breaks occurred, and this extra information is usually pointless. Instead Biopython uses a default line wrapping of 60 characters on output. The same problem with white space applies in many other file formats too. Another issue in some cases is that Biopython does not (yet) preserve every last bit of annotation (e.g. GenBank and EMBL).

Occasionally preserving the original layout (with any quirks it may have) is important. See the section about raw access about the get_raw() method of the Bio.SeqIO.index() dictionary-like object for one potential solution.

6.6.2 Converting between sequence file formats

In previous example we used a list of SeqRecord objects as input to the Bio.SeqIO.write() function, but it will also accept a SeqRecord iterator like we get from Bio.SeqIO.parse() - this lets us do file conversion by combining these two functions.

For this example we'll read in the GenBank format file and write it out in FASTA format:

```
In [ ]: records = SeqIO.parse("data/ls_orchid.gbk", "genbank")
        count = SeqIO.write(records, "data/my_example.fasta", "fasta")
        print("Converted %i records" % count)
```

Still, that is a little bit complicated. So, because file conversion is such a common task, there is a helper function letting you replace that with just:

```
In [ ]: count = SeqIO.convert("data/ls_orchid.gbk", "genbank", "data/my_example.fasta", "fasta")
        print("Converted %i records" % count)
```

The Bio.SeqIO.convert() function will take handles *or* filenames. Watch out though - if the output file already exists, it will overwrite it! To find out more, see the built in help:

```
In [ ]: help(SeqIO.convert)
```

In principle, just by changing the filenames and the format names, this code could be used to convert between any file formats available in Biopython. However, writing some formats requires information (e.g. quality scores) which other files formats don't contain. For example, while you can turn a FASTQ file into a FASTA file, you can't do the reverse.

Finally, as an added incentive for using the Bio.SeqIO.convert() function (on top of the fact your code will be shorter), doing it this way may also be faster! The reason for this is the convert function can take advantage of several file format specific optimisations and tricks.

6.6.3 Converting a file of sequences to their reverse complements

Suppose you had a file of nucleotide sequences, and you wanted to turn it into a file containing their reverse complement sequences. This time a little bit of work is required to transform the SeqRecord objects we get from our input file into something suitable for saving to our output file.

To start with, we'll use Bio.SeqIO.parse() to load some nucleotide sequences from a file, then print out their reverse complements using the Seq object's built in .reverse_complement() method:

```
In [ ]: for i, record in enumerate(SeqIO.parse("data/ls_orchid.gbk", "genbank")):
            print(record.id)
            print(record.seq.reverse_complement())
            if i == 2:  # 3 is enough
                break
```

Now, if we want to save these reverse complements to a file, we'll need to make SeqRecord objects. We can use the SeqRecord object's built in .reverse_complement() method but we must decide how to name our new records.

This is an excellent place to demonstrate the power of list comprehensions which make a list in memory:

```
In [ ]: records = [rec.reverse_complement(id="rc_"+rec.id, description = "reverse complement") \
                   for rec in SeqIO.parse("data/ls_orchid.fasta", "fasta")]
        len(records)
```

Now list comprehensions have a nice trick up their sleeves, you can add a conditional statement:

```
In [ ]: records = [rec.reverse_complement(id="rc_"+rec.id, description = "reverse complement") \
                   for rec in SeqIO.parse("data/ls_orchid.fasta", "fasta") if len(rec)<700]
        len(records)
```

That would create an in memory list of reverse complement records where the sequence length was under 700 base pairs. However, we can do exactly the same with a generator expression - but with the advantage that this does not create a list of all the records in memory at once:

```
In [ ]: records = (rec.reverse_complement(id="rc_"+rec.id, description = "reverse complement") \
                  for rec in SeqIO.parse("data/ls_orchid.fasta", "fasta") if len(rec)<700)
```

As a complete example:

```
In [ ]: records = (rec.reverse_complement(id="rc_"+rec.id, description = "reverse complement") \
                   for rec in SeqIO.parse("data/ls_orchid.fasta", "fasta") if len(rec)<700)
        SeqIO.write(records, "data/rev_comp.fasta", "fasta")
```

6.6.4 Getting your SeqRecord objects as formatted strings

Suppose that you don't really want to write your records to a file or handle - instead you want a string containing the records in a particular file format. The Bio.SeqIO interface is based on handles, but Python has a useful built in module which provides a string based handle.

For an example of how you might use this, let's load in a bunch of SeqRecord objects from our orchids GenBank file, and create a string containing the records in FASTA format:

```
In [ ]: records = SeqIO.parse("data/ls_orchid.gbk", "genbank")
        out_handle = StringIO()
        SeqIO.write(records, out_handle, "fasta")
        fasta_data = out_handle.getvalue()
        print(fasta_data)
```

This isn't entirely straightforward the first time you see it! On the bright side, for the special case where you would like a string containing a *single* record in a particular file format, use the the SeqRecord class' format() method.

Note that although we don't encourage it, you *can* use the format() method to write to a file, for example something like this:

```
In [ ]: out_handle = open("data/ls_orchid_long.tab", "w")
        for record in SeqIO.parse("data/ls_orchid.gbk", "genbank"):
            if len(record) > 100:
                out_handle.write(record.format("tab"))
        out_handle.close()
```

While this style of code will work for a simple sequential file format like FASTA or the simple tab separated format used here, it will *not* work for more complex or interlaced file formats. This is why we still recommend using Bio.SeqIO.write(), as in the following example:

```
In [ ]: records = (rec for rec in SeqIO.parse("data/ls_orchid.gbk", "genbank") if len(rec) > 100)
        SeqIO.write(records, "data/ls_orchid.tab", "tab")
```

Making a single call to SeqIO.write(...) is also much quicker than multiple calls to the SeqRecord.format(...) method.

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 7

Multiple Sequence Alignment objects

This chapter is about Multiple Sequence Alignments, by which we mean a collection of multiple sequences which have been aligned together – usually with the insertion of gap characters, and addition of leading or trailing gaps – such that all the sequence strings are the same length. Such an alignment can be regarded as a matrix of letters, where each row is held as a `SeqRecord` object internally.

We will introduce the `MultipleSeqAlignment` object which holds this kind of data, and the `Bio.AlignIO` module for reading and writing them as various file formats (following the design of the `Bio.SeqIO` module from the previous chapter). Note that both `Bio.SeqIO` and `Bio.AlignIO` can read and write sequence alignment files. The appropriate choice will depend largely on what you want to do with the data.

The final part of this chapter is about our command line wrappers for common multiple sequence alignment tools like ClustalW and MUSCLE.

7.1 Parsing or Reading Sequence Alignments

We have two functions for reading in sequence alignments, `Bio.AlignIO.read()` and `Bio.AlignIO.parse()` which following the convention introduced in `Bio.SeqIO` are for files containing one or multiple alignments respectively.

Using `Bio.AlignIO.parse()` will return an *iterator* which gives `MultipleSeqAlignment` objects. Iterators are typically used in a for loop. Examples of situations where you will have multiple different alignments include resampled alignments from the PHYLIP tool `seqboot`, or multiple pairwise alignments from the EMBOSS tools `water` or `needle`, or Bill Pearson's FASTA tools.

However, in many situations you will be dealing with files which contain only a single alignment. In this case, you should use the `Bio.AlignIO.read()` function which returns a single `MultipleSeqAlignment` object.

Both functions expect two mandatory arguments:

1. The first argument is a *handle* to read the data from, typically an open file (see Section [sec:appendix-handles]), or a filename.
2. The second argument is a lower case string specifying the alignment format. As in `Bio.SeqIO` we don't try and guess the file format for you! See http://biopython.org/wiki/AlignIO for a full listing of supported formats.

There is also an optional `seq_count` argument which is discussed in Section [sec:AlignIO-count-argument] below for dealing with ambiguous file formats which may contain more than one alignment.

A further optional `alphabet` argument allowing you to specify the expected alphabet. This can be useful as many alignment file formats do not explicitly label the sequences as RNA, DNA or protein – which means `Bio.AlignIO` will default to using a generic alphabet.

7.1.1 Single Alignments

As an example, consider the following annotation rich protein alignment in the PFAM or Stockholm file format:

```
# STOCKHOLM 1.0
#=GS COATB_BPIKE/30-81  AC P03620.1
#=GS COATB_BPIKE/30-81  DR PDB; 1ifl ; 1-52;
#=GS Q9T0Q8_BPIKE/1-52  AC Q9T0Q8.1
#=GS COATB_BPI22/32-83  AC P15416.1
#=GS COATB_BPM13/24-72  AC P69541.1
#=GS COATB_BPM13/24-72  DR PDB; 2cpb ; 1-49;
#=GS COATB_BPM13/24-72  DR PDB; 2cps ; 1-49;
#=GS COATB_BPZJ2/1-49   AC P03618.1
#=GS Q9T0Q9_BPFD/1-49   AC Q9T0Q9.1
#=GS Q9T0Q9_BPFD/1-49   DR PDB; 1nh4 A; 1-49;
#=GS COATB_BPIF1/22-73  AC P03619.2
#=GS COATB_BPIF1/22-73  DR PDB; 1ifk ; 1-50;
COATB_BPIKE/30-81             AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
#=GR COATB_BPIKE/30-81  SS    -HHHHHHHHHHHHHH--HHHHHHHH--HHHHHHHHHHHHHHHHHHHHH----
Q9T0Q8_BPIKE/1-52             AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
COATB_BPI22/32-83             DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
COATB_BPM13/24-72             AEGDDP...AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR COATB_BPM13/24-72  SS    ---S-T...CHCHHHHCCCCTCCCTTCHHHHHHHHHHHHHHHHHHHHCTT--
COATB_BPZJ2/1-49              AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA
Q9T0Q9_BPFD/1-49              AEGDDP...AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
#=GR Q9T0Q9_BPFD/1-49   SS    ------...-HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH--
COATB_BPIF1/22-73             FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA
#=GR COATB_BPIF1/22-73  SS    XX-HHHH--HHHHHH--HHHHHHH--HHHHHHHHHHHHHHHHHHHHHHH---
#=GC SS_cons                  XHHHHHHHHHHHHHHHCHHHHHHHHCHHHHHHHHHHHHHHHHHHHHHHHC--
#=GC seq_cons                 AEssss...AptAhDSLpspAT-hIu.sWshVsslVsAsluIKLFKKFsSKA
//
```

This is the seed alignment for the Phage_Coat_Gp8 (PF05371) PFAM entry, downloaded from a now out of date release of PFAM from http://pfam.sanger.ac.uk/. We can load this file as follows (assuming it has been saved to disk as "PF05371_seed.sth" in the current working directory):

```
In [1]: from Bio import AlignIO
        alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
```

This code will print out a summary of the alignment:

```
In [2]: print(alignment)
```

```
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRL...SKA COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

You'll notice in the above output the sequences have been truncated. We could instead write our own code to format this as we please by iterating over the rows as `SeqRecord` objects:

```
In [3]: from Bio import AlignIO
        alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
        print("Alignment length %i" % alignment.get_alignment_length())
```

```
Alignment length 52
```

```
In [4]: for record in alignment:
            print("%s - %s" % (record.seq, record.id))
```

```
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA - COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA - Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA - COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA - COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA - COATB_BPIF1/22-73
```

You could also use the alignment object's `format` method to show it in a particular file format – see Section [sec:alignment-format-method] for details.

Did you notice in the raw file above that several of the sequences include database cross-references to the PDB and the associated known secondary structure? Try this:

```
In [5]: for record in alignment:
            if record.dbxrefs:
                print("%s %s" % (record.id, record.dbxrefs))
```

```
COATB_BPIKE/30-81 ['PDB; 1ifl ; 1-52;']
COATB_BPM13/24-72 ['PDB; 2cpb ; 1-49;', 'PDB; 2cps ; 1-49;']
Q9T0Q9_BPFD/1-49 ['PDB; 1nh4 A; 1-49;']
COATB_BPIF1/22-73 ['PDB; 1ifk ; 1-50;']
```

To have a look at all the sequence annotation, try this:

```
In [6]: for record in alignment:
            print(record)
```

```
ID: COATB_BPIKE/30-81
Name: COATB_BPIKE
Description: COATB_BPIKE/30-81
Database cross-references: PDB; 1ifl ; 1-52;
Number of features: 0
/start=30
/accession=P03620.1
/end=81
Per letter annotation for: secondary_structure
Seq('AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA', SingleLetterAlphabet())
ID: Q9T0Q8_BPIKE/1-52
Name: Q9T0Q8_BPIKE
Description: Q9T0Q8_BPIKE/1-52
Number of features: 0
/start=1
/accession=Q9T0Q8.1
/end=52
Seq('AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA', SingleLetterAlphabet())
ID: COATB_BPI22/32-83
Name: COATB_BPI22
Description: COATB_BPI22/32-83
Number of features: 0
/start=32
```

```
/accession=P15416.1
/end=83
Seq('DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA', SingleLetterAlphabet())
ID: COATB_BPM13/24-72
Name: COATB_BPM13
Description: COATB_BPM13/24-72
Database cross-references: PDB; 2cpb ; 1-49;, PDB; 2cps ; 1-49;
Number of features: 0
/start=24
/accession=P69541.1
/end=72
Per letter annotation for: secondary_structure
Seq('AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA', SingleLetterAlphabet())
ID: COATB_BPZJ2/1-49
Name: COATB_BPZJ2
Description: COATB_BPZJ2/1-49
Number of features: 0
/start=1
/accession=P03618.1
/end=49
Seq('AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA', SingleLetterAlphabet())
ID: Q9T0Q9_BPFD/1-49
Name: Q9T0Q9_BPFD
Description: Q9T0Q9_BPFD/1-49
Database cross-references: PDB; 1nh4 A; 1-49;
Number of features: 0
/start=1
/accession=Q9T0Q9.1
/end=49
Per letter annotation for: secondary_structure
Seq('AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA', SingleLetterAlphabet())
ID: COATB_BPIF1/22-73
Name: COATB_BPIF1
Description: COATB_BPIF1/22-73
Database cross-references: PDB; 1ifk ; 1-50;
Number of features: 0
/start=22
/accession=P03619.2
/end=73
Per letter annotation for: secondary_structure
Seq('FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA', SingleLetterAlphabet())
```

Sanger provide a nice web interface at http://pfam.sanger.ac.uk/family?acc=PF05371 which will actually let you download this alignment in several other formats. This is what the file looks like in the FASTA file format:

```
>COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA
>Q9T0Q8_BPIKE/1-52
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA
>COATB_BPI22/32-83
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA
>COATB_BPM13/24-72
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
>COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA
>Q9T0Q9_BPFD/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA
>COATB_BPIF1/22-73
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA
```

Note the website should have an option about showing gaps as periods (dots) or dashes, we've shown dashes above. Assuming you download and save this as file "PF05371_seed.faa" then you can load it with almost exactly the same code:

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.faa", "fasta")
print(alignment)
```

All that has changed in this code is the filename and the format string. You'll get the same output as before, the sequences and record identifiers are the same. However, as you should expect, if you check each `SeqRecord` there is no annotation nor database cross-references because these are not included in the FASTA file format.

Note that rather than using the Sanger website, you could have used `Bio.AlignIO` to convert the original Stockholm format file into a FASTA file yourself (see below).

With any supported file format, you can load an alignment in exactly the same way just by changing the format string. For example, use "phylip" for PHYLIP files, "nexus" for NEXUS files or "emboss" for the alignments output by the EMBOSS tools. There is a full listing on the wiki page (http://biopython.org/wiki/AlignIO) and in the built in documentation (also online):

```
In [7]: from Bio import AlignIO
        help(AlignIO)
```

```
Help on package Bio.AlignIO in Bio:

NAME
    Bio.AlignIO - Multiple sequence alignment input/output as alignment objects.

DESCRIPTION
    The Bio.AlignIO interface is deliberately very similar to Bio.SeqIO, and in
    fact the two are connected internally.  Both modules use the same set of file
    format names (lower case strings).  From the user's perspective, you can read
    in a PHYLIP file containing one or more alignments using Bio.AlignIO, or you
    can read in the sequences within these alignmenta using Bio.SeqIO.

    Bio.AlignIO is also documented at http://biopython.org/wiki/AlignIO and by
    a whole chapter in our tutorial:

    * `HTML Tutorial`_
    * `PDF Tutorial`_

    .. _`HTML Tutorial`: http://biopython.org/DIST/docs/tutorial/Tutorial.html
    .. _`PDF Tutorial`: http://biopython.org/DIST/docs/tutorial/Tutorial.pdf

    Input
    -----
    For the typical special case when your file or handle contains one and only
    one alignment, use the function Bio.AlignIO.read().  This takes an input file
    handle (or in recent versions of Biopython a filename as a string), format
    string and optional number of sequences per alignment.  It will return a single
    MultipleSeqAlignment object (or raise an exception if there isn't just one
    alignment):

    >>> from Bio import AlignIO
    >>> align = AlignIO.read("Phylip/interlaced.phy", "phylip")
    >>> print(align)
    SingleLetterAlphabet() alignment with 3 rows and 384 columns
    -----MKVILLFVLAVFTVFVSS---------------RGIPPE...I-- CYS1_DICDI
    MAHARVLLLALAVLATAAVAVASSSSFADSNPIRPVTDRAASTL...VAA ALEU_HORVU
    ------MWATLPLLCAGAWLLGV--------PVCGAAELSVNSL...PLV CATH_HUMAN
```

```
For the general case, when the handle could contain any number of alignments,
use the function Bio.AlignIO.parse(...) which takes the same arguments, but
returns an iterator giving MultipleSeqAlignment objects (typically used in a
for loop). If you want random access to the alignments by number, turn this
into a list:

>>> from Bio import AlignIO
>>> alignments = list(AlignIO.parse("Emboss/needle.txt", "emboss"))
>>> print(alignments[2])
SingleLetterAlphabet() alignment with 2 rows and 120 columns
-KILIVDDQYGIRILLNEVFNKEGYQTFQAANGLQALDIVTKER...--- ref_rec
LHIVVVDDDPGTCVYIESVFAELGHTCKSFVRPEAAEEYILTHP...HKE gi|94967506|receiver

Most alignment file formats can be concatenated so as to hold as many
different multiple sequence alignments as possible.  One common example
is the output of the tool seqboot in the PHLYIP suite.  Sometimes there
can be a file header and footer, as seen in the EMBOSS alignment output.

Output
------
Use the function Bio.AlignIO.write(...), which takes a complete set of
Alignment objects (either as a list, or an iterator), an output file handle
(or filename in recent versions of Biopython) and of course the file format::

  from Bio import AlignIO
  alignments = ...
  count = SeqIO.write(alignments, "example.faa", "fasta")

If using a handle make sure to close it to flush the data to the disk::

  from Bio import AlignIO
  alignments = ...
  with open("example.faa", "w") as handle:
    count = SeqIO.write(alignments, handle, "fasta")

In general, you are expected to call this function once (with all your
alignments) and then close the file handle.  However, for file formats
like PHYLIP where multiple alignments are stored sequentially (with no file
header and footer), then multiple calls to the write function should work as
expected when using handles.

If you are using a filename, the repeated calls to the write functions will
overwrite the existing file each time.

Conversion
----------
The Bio.AlignIO.convert(...) function allows an easy interface for simple
alignnment file format conversions. Additionally, it may use file format
specific optimisations so this should be the fastest way too.

In general however, you can combine the Bio.AlignIO.parse(...) function with
the Bio.AlignIO.write(...) function for sequence file conversion. Using
generator expressions provides a memory efficient way to perform filtering or
other extra operations as part of the process.

File Formats
------------
When specifying the file format, use lowercase strings.  The same format
```

```
    names are also used in Bio.SeqIO and include the following:

      - clustal -   Output from Clustal W or X, see also the module Bio.Clustalw
        which can be used to run the command line tool from Biopython.
      - emboss    - EMBOSS tools' "pairs" and "simple" alignment formats.
      - fasta     - The generic sequence file format where each record starts with
        an identifer line starting with a ">" character, followed by
        lines of sequence.
      - fasta-m10 - For the pairswise alignments output by Bill Pearson's FASTA
        tools when used with the -m 10 command line option for machine
        readable output.
      - ig        - The IntelliGenetics file format, apparently the same as the
        MASE alignment format.
      - nexus     - Output from NEXUS, see also the module Bio.Nexus which can also
        read any phylogenetic trees in these files.
      - phylip    - Interlaced PHYLIP, as used by the PHLIP tools.
      - phylip-sequential - Sequential PHYLIP.
      - phylip-relaxed - PHYLIP like format allowing longer names.
      - stockholm - A richly annotated alignment file format used by PFAM.

    Note that while Bio.AlignIO can read all the above file formats, it cannot
    write to all of them.

    You can also use any file format supported by Bio.SeqIO, such as "fasta" or
    "ig" (which are listed above), PROVIDED the sequences in your file are all the
    same length.

PACKAGE CONTENTS
    ClustalIO
    EmbossIO
    FastaIO
    Interfaces
    NexusIO
    PhylipIO
    StockholmIO

FUNCTIONS
    convert(in_file, in_format, out_file, out_format, alphabet=None)
        Convert between two alignment files, returns number of alignments.

            - in_file - an input handle or filename
            - in_format - input file format, lower case string
            - output - an output handle or filename
            - out_file - output file format, lower case string
            - alphabet - optional alphabet to assume

        **NOTE** - If you provide an output filename, it will be opened which will
        overwrite any existing file without warning. This may happen if even the
        conversion is aborted (e.g. an invalid out_format name is given).

    parse(handle, format, seq_count=None, alphabet=None)
        Iterate over an alignment file as MultipleSeqAlignment objects.

        Arguments:
          - handle    - handle to the file, or the filename as a string
            (note older versions of Biopython only took a handle).
          - format    - string describing the file format.
          - alphabet  - optional Alphabet object, useful when the sequence type
            cannot be automatically inferred from the file itself
            (e.g. fasta, phylip, clustal)
          - seq_count - Optional integer, number of sequences expected in each
            alignment.  Recommended for fasta format files.

        If you have the file name in a string 'filename', use:

        >>> from Bio import AlignIO
        >>> filename = "Emboss/needle.txt"
        >>> format = "emboss"
        >>> for alignment in AlignIO.parse(filename, format):
        ...     print("Alignment of length %i" % alignment.get_alignment_length())
        Alignment of length 124
        Alignment of length 119
        Alignment of length 120
        Alignment of length 118
        Alignment of length 125

        If you have a string 'data' containing the file contents, use::

          from Bio import AlignIO
          from StringIO import StringIO
          my_iterator = AlignIO.parse(StringIO(data), format)

        Use the Bio.AlignIO.read() function when you expect a single record only.
```

```
read(handle, format, seq_count=None, alphabet=None)
    Turns an alignment file into a single MultipleSeqAlignment object.

    Arguments:
      - handle    - handle to the file, or the filename as a string
        (note older versions of Biopython only took a handle).
      - format    - string describing the file format.
      - alphabet  - optional Alphabet object, useful when the sequence type
        cannot be automatically inferred from the file itself
        (e.g. fasta, phylip, clustal)
      - seq_count - Optional integer, number of sequences expected in each
        alignment.  Recommended for fasta format files.

    If the handle contains no alignments, or more than one alignment,
    an exception is raised.  For example, using a PFAM/Stockholm file
    containing one alignment:

    >>> from Bio import AlignIO
    >>> filename = "Clustalw/protein.aln"
    >>> format = "clustal"
    >>> alignment = AlignIO.read(filename, format)
    >>> print("Alignment of length %i" % alignment.get_alignment_length())
    Alignment of length 411

    If however you want the first alignment from a file containing
    multiple alignments this function would raise an exception.

    >>> from Bio import AlignIO
    >>> filename = "Emboss/needle.txt"
    >>> format = "emboss"
    >>> alignment = AlignIO.read(filename, format)
    Traceback (most recent call last):
        ...
    ValueError: More than one record found in handle
```

```
        Instead use:

        >>> from Bio import AlignIO
        >>> filename = "Emboss/needle.txt"
        >>> format = "emboss"
        >>> alignment = next(AlignIO.parse(filename, format))
        >>> print("First alignment has length %i" % alignment.get_alignment_length())
        First alignment has length 124

        You must use the Bio.AlignIO.parse() function if you want to read multiple
        records from the handle.

    write(alignments, handle, format)
        Write complete set of alignments to a file.

        Arguments:
          - alignments - A list (or iterator) of Alignment objects (ideally the
            new MultipleSeqAlignment objects), or (if using Biopython
            1.54 or later) a single alignment object.
          - handle    - File handle object to write to, or filename as string
            (note older versions of Biopython only took a handle).
          - format    - lower case string describing the file format to write.

        You should close the handle after calling this function.

        Returns the number of alignments written (as an integer).

DATA
    __docformat__ = 'restructuredtext en'
    print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0)...

FILE
    /home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py
```

7.1.2 Multiple Alignments

The previous section focused on reading files containing a single alignment. In general however, files can contain more than one alignment, and to read these files we must use the `Bio.AlignIO.parse()` function.

Suppose you have a small alignment in PHYLIP format:

```
5    6
Alpha     AACAAC
Beta      AACCCC
Gamma     ACCAAC
Delta     CCACCA
Epsilon   CCAAAC
```

If you wanted to bootstrap a phylogenetic tree using the PHYLIP tools, one of the steps would be to create a set of many resampled alignments using the tool `bootseq`. This would give output something like this, which has been abbreviated for conciseness:

```
5     6
Alpha     AAACCA
```

```
Beta      AAACCC
Gamma     ACCCCA
Delta     CCCAAC
Epsilon   CCCAAA
    5     6
Alpha     AAACAA
Beta      AAACCC
Gamma     ACCCAA
Delta     CCCACC
Epsilon   CCCAAA
    5     6
Alpha     AAAAAC
Beta      AAACCC
Gamma     AACAAC
Delta     CCCCCA
Epsilon   CCCAAC
...
    5     6
Alpha     AAAACC
Beta      ACCCCC
Gamma     AAAACC
Delta     CCCCAA
Epsilon   CAAACC
```

If you wanted to read this in using `Bio.AlignIO` you could use:

```
In [8]: from Bio import AlignIO
        alignments = AlignIO.parse("data/resampled.phy", "phylip")
        for alignment in alignments:
            print(alignment)
            print("")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-8-13e6344aed5e> in <module>()
      1 from Bio import AlignIO
      2 alignments = AlignIO.parse("data/resampled.phy", "phylip")
----> 3 for alignment in alignments:
      4     print(alignment)
      5     print("")

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py in parse(handle, forr
    348         raise TypeError("Need integer for seq_count (sequences per alignment)")
    349
--> 350     with as_handle(handle, 'rU') as fp:
    351         # Map the file format to a sequence iterator:
    352         if format in _FormatToIterator:

/home/tiago_antao/miniconda/lib/python3.5/contextlib.py in __enter__(self)
     57     def __enter__(self):
     58         try:
---> 59             return next(self.gen)
     60         except StopIteration:
     61             raise RuntimeError("generator didn't yield") from None

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/File.py in as_handle(handleish, mode, **l
     88                 yield fp
     89         else:
---> 90             with open(handleish, mode, **kwargs) as fp:
     91                 yield fp
     92     else:

FileNotFoundError: [Errno 2] No such file or directory: 'data/resampled.phy'
```

As with the function `Bio.SeqIO.parse()`, using `Bio.AlignIO.parse()` returns an iterator. If you want to keep all the alignments in memory at once, which will allow you to access them in any order, then turn the iterator into a list:

```
In [10]: from Bio import AlignIO
         alignments = list(AlignIO.parse("data/resampled.phy", "phylip"))
         last_align = alignments[-1]
         first_align = alignments[0]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-10-0ef98bef9604> in <module>()
      1 from Bio import AlignIO
----> 2 alignments = list(AlignIO.parse("data/resampled.phy", "phylip"))
      3 last_align = alignments[-1]
      4 first_align = alignments[0]

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py in parse(handle, for
    348         raise TypeError("Need integer for seq_count (sequences per alignment)")
    349 
--> 350     with as_handle(handle, 'rU') as fp:
    351         # Map the file format to a sequence iterator:
    352         if format in _FormatToIterator:

/home/tiago_antao/miniconda/lib/python3.5/contextlib.py in __enter__(self)
     57     def __enter__(self):
     58         try:
---> 59             return next(self.gen)
     60         except StopIteration:
     61             raise RuntimeError("generator didn't yield") from None

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/File.py in as_handle(handleish, mode, **
     88                 yield fp
     89         else:
---> 90             with open(handleish, mode, **kwargs) as fp:
     91                 yield fp
     92     else:

FileNotFoundError: [Errno 2] No such file or directory: 'data/resampled.phy'
```

7.1.3 Ambiguous Alignments

Many alignment file formats can explicitly store more than one alignment, and the division between each alignment is clear. However, when a general sequence file format has been used there is no such block structure. The most common such situation is when alignments have been saved in the FASTA file format. For example consider the following:

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
```

```
ACTACGACTAGCTCAGG--
>Beta
ACTACCGCTAGCTCAGAAG
>Gamma
ACTACGGCTAGCACAGAAG
```

This could be a single alignment containing six sequences (with repeated identifiers). Or, judging from the identifiers, this is probably two different alignments each with three sequences, which happen to all have the same length.

What about this next example?

```
>Alpha
ACTACGACTAGCTCAG--G
>Beta
ACTACCGCTAGCTCAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Gamma
ACTACGGCTAGCACAGAAG
>Alpha
ACTACGACTAGCTCAGG--
>Delta
ACTACGGCTAGCACAGAAG
```

Again, this could be a single alignment with six sequences. However this time based on the identifiers we might guess this is three pairwise alignments which by chance have all got the same lengths.

This final example is similar:

```
>Alpha
ACTACGACTAGCTCAG--G
>XXX
ACTACCGCTAGCTCAGAAG
>Alpha
ACTACGACTAGCTCAGG
>YYY
ACTACGGCAAGCACAGG
>Alpha
--ACTACGAC--TAGCTCAGG
>ZZZ
GGACTACGACAATAGCTCAGG
```

In this third example, because of the differing lengths, this cannot be treated as a single alignment containing all six records. However, it could be three pairwise alignments.

Clearly trying to store more than one alignment in a FASTA file is not ideal. However, if you are forced to deal with these as input files `Bio.AlignIO` can cope with the most common situation where all the alignments have the same number of records. One example of this is a collection of pairwise alignments, which can be produced by the EMBOSS tools `needle` and `water` – although in this situation, `Bio.AlignIO` should be able to understand their native output using “emboss” as the format string.

To interpret these FASTA examples as several separate alignments, we can use `Bio.AlignIO.parse()` with the optional `seq_count` argument which specifies how many sequences are expected in each alignment (in these examples, 3, 2 and 2 respectively). For example, using the third example as the input data:

```
In [11]: for alignment in AlignIO.parse(handle, "fasta", seq_count=2):
             print("Alignment length %i" % alignment.get_alignment_length())
             for record in alignment:
                 print("%s - %s" % (record.seq, record.id))
```

```
        print("")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-6e1f5efd5fb4> in <module>()
----> 1 for alignment in AlignIO.parse(handle, "fasta", seq_count=2):
      2     print("Alignment length %i" % alignment.get_alignment_length())
      3     for record in alignment:
      4         print("%s - %s" % (record.seq, record.id))
      5     print("")

NameError: name 'handle' is not defined
```

Using `Bio.AlignIO.read()` or `Bio.AlignIO.parse()` without the `seq_count` argument would give a single alignment containing all six records for the first two examples. For the third example, an exception would be raised because the lengths differ preventing them being turned into a single alignment.

If the file format itself has a block structure allowing `Bio.AlignIO` to determine the number of sequences in each alignment directly, then the `seq_count` argument is not needed. If it is supplied, and doesn't agree with the file contents, an error is raised.

Note that this optional `seq_count` argument assumes each alignment in the file has the same number of sequences. Hypothetically you may come across stranger situations, for example a FASTA file containing several alignments each with a different number of sequences – although I would love to hear of a real world example of this. Assuming you cannot get the data in a nicer file format, there is no straight forward way to deal with this using `Bio.AlignIO`. In this case, you could consider reading in the sequences themselves using `Bio.SeqIO` and batching them together to create the alignments as appropriate.

7.2 Writing Alignments

We've talked about using `Bio.AlignIO.read()` and `Bio.AlignIO.parse()` for alignment input (reading files), and now we'll look at `Bio.AlignIO.write()` which is for alignment output (writing files). This is a function taking three arguments: some `MultipleSeqAlignment` objects (or for backwards compatibility the obsolete `Alignment` objects), a handle or filename to write to, and a sequence format.

Here is an example, where we start by creating a few `MultipleSeqAlignment` objects the hard way (by hand, rather than by loading them from a file). Note we create some `SeqRecord` objects to construct the alignment from.

```
In [9]: from Bio.Alphabet import generic_dna
        from Bio.Seq import Seq
        from Bio.SeqRecord import SeqRecord
        from Bio.Align import MultipleSeqAlignment

        align1 = MultipleSeqAlignment([
                     SeqRecord(Seq("ACTGCTAGCTAG", generic_dna), id="Alpha"),
                     SeqRecord(Seq("ACT-CTAGCTAG", generic_dna), id="Beta"),
                     SeqRecord(Seq("ACTGCTAGDTAG", generic_dna), id="Gamma"),
                 ])

        align2 = MultipleSeqAlignment([
                     SeqRecord(Seq("GTCAGC-AG", generic_dna), id="Delta"),
                     SeqRecord(Seq("GACAGCTAG", generic_dna), id="Epsilon"),
                     SeqRecord(Seq("GTCAGCTAG", generic_dna), id="Zeta"),
                 ])

        align3 = MultipleSeqAlignment([
                     SeqRecord(Seq("ACTAGTACAGCTG", generic_dna), id="Eta"),
                     SeqRecord(Seq("ACTAGTACAGCT-", generic_dna), id="Theta"),
                     SeqRecord(Seq("-CTACTACAGGTG", generic_dna), id="Iota"),
                 ])

        my_alignments = [align1, align2, align3]
```

Now we have a list of `Alignment` objects, we'll write them to a PHYLIP format file:

```
In [10]: from Bio import AlignIO
         AlignIO.write(my_alignments, "my_example.phy", "phylip")
```

```
Out[10]: 3
```

And if you open this file in your favourite text editor it should look like this:

```
3 12
Alpha      ACTGCTAGCT AG
Beta       ACT-CTAGCT AG
Gamma      ACTGCTAGDT AG
 3 9
Delta      GTCAGC-AG
Epislon    GACAGCTAG
Zeta       GTCAGCTAG
 3 13
Eta        ACTAGTACAG CTG
Theta      ACTAGTACAG CT-
Iota       -CTACTACAG GTG
```

Its more common to want to load an existing alignment, and save that, perhaps after some simple manipulation like removing certain rows or columns.

Suppose you wanted to know how many alignments the `Bio.AlignIO.write()` function wrote to the handle? If your alignments were in a list like the example above, you could just use `len(my_alignments)`, however you can't do that when your records come from a generator/iterator. Therefore the `Bio.AlignIO.write()` function returns the number of alignments written to the file.

Note - If you tell the `Bio.AlignIO.write()` function to write to a file that already exists, the old file will be overwritten without any warning.

7.2.1 Converting between sequence alignment file formats

Converting between sequence alignment file formats with `Bio.AlignIO` works in the same way as converting between sequence file formats with `Bio.SeqIO` (Section [sec:SeqIO-conversion]). We load generally the alignment(s) using `Bio.AlignIO.parse()` and then save them using the `Bio.AlignIO.write()` – or just use the `Bio.AlignIO.convert()` helper function.

For this example, we'll load the PFAM/Stockholm format file used earlier and save it as a Clustal W format file:

```
In [11]: from Bio import AlignIO
         count = AlignIO.convert("data/PF05371_seed.sth", "stockholm", "PF05371_seed.aln", "clustal"
         print("Converted %i alignments" % count)
```

```
Converted 1 alignments
```

Or, using `Bio.AlignIO.parse()` and `Bio.AlignIO.write()`:

```
In [12]: from Bio import AlignIO
         alignments = AlignIO.parse("data/PF05371_seed.sth", "stockholm")
         count = AlignIO.write(alignments, "PF05371_seed.aln", "clustal")
         print("Converted %i alignments" % count)
```

```
Converted 1 alignments
```

The `Bio.AlignIO.write()` function expects to be given multiple alignment objects. In the example above we gave it the alignment iterator returned by `Bio.AlignIO.parse()`.

In this case, we know there is only one alignment in the file so we could have used `Bio.AlignIO.read()` instead, but notice we have to pass this alignment to `Bio.AlignIO.write()` as a single element list:

```
In [13]: from Bio import AlignIO
         alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
         AlignIO.write([alignment], "PF05371_seed.aln", "clustal")
```

```
Out[13]: 1
```

Either way, you should end up with the same new Clustal W format file "PF05371_seed.aln" with the following content:

```
CLUSTAL X (1.81) multiple sequence alignment


COATB_BPIKE/30-81                     AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSS
Q9T0Q8_BPIKE/1-52                     AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVS
COATB_BPI22/32-83                     DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSS
COATB_BPM13/24-72                     AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTS
COATB_BPZJ2/1-49                      AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFAS
Q9T0Q9_BPFD/1-49                      AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTS
COATB_BPIF1/22-73                     FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVS

COATB_BPIKE/30-81                     KA
Q9T0Q8_BPIKE/1-52                     RA
COATB_BPI22/32-83                     KA
COATB_BPM13/24-72                     KA
COATB_BPZJ2/1-49                      KA
Q9T0Q9_BPFD/1-49                      KA
COATB_BPIF1/22-73                     RA
```

Alternatively, you could make a PHYLIP format file which we'll name "PF05371_seed.phy":

```
In [14]: from Bio import AlignIO
         AlignIO.convert("data/PF05371_seed.sth", "stockholm", "PF05371_seed.phy", "phylip")
```

```
Out[14]: 1
```

This time the output looks like this:

```
7 52
COATB_BPIK AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS
Q9T0Q8_BPI AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS
COATB_BPI2 DGTSTATSYA TEAMNSLKTQ ATDLIDQTWP VVTSVAVAGL AIRLFKKFSS
COATB_BPM1 AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPZJ AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
Q9T0Q9_BPF AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
COATB_BPIF FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS

           KA
           RA
           KA
           KA
           KA
           KA
           RA
```

One of the big handicaps of the original PHYLIP alignment file format is that the sequence identifiers are strictly truncated at ten characters. In this example, as you can see the resulting names are still unique - but they are not very readable. As a result, a more relaxed variant of the original PHYLIP format is now quite widely used:

```
In [15]: from Bio import AlignIO
         AlignIO.convert("data/PF05371_seed.sth", "stockholm", "PF05371_seed.phy", "phylip-relaxed")
```

```
Out[15]: 1
```

This time the output looks like this, using a longer indentation to allow all the identifers to be given in full:

```
::
```

7 52 COATB_BPIKE/30-81 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS Q9T0Q8_BPIKE/1-52 AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS COATB_BPI22/32-83 DGTSTATSYA TEAMNSLKTQ AT-DLIDQTWP VVTSVAVAGL AIRLFKKFSS COATB_BPM13/24-72 AEGDDP—A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS COATB_BPZJ2/1-49 AEGDDP—A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS Q9T0Q9_BPFD/1-49 AEGDDP—A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS COATB_BPIF1/22-73 FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS

KA RA KA KA KA KA RA

If you have to work with the original strict PHYLIP format, then you may need to compress the identifers somehow – or assign your own names or numbering system. This following bit of code manipulates the record identifiers before saving the output:

```
In [16]: from Bio import AlignIO
         alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
         name_mapping = {}
         for i, record in enumerate(alignment):
             name_mapping[i] = record.id
             record.id = "seq%i" % i
         print(name_mapping)

         AlignIO.write([alignment], "PF05371_seed.phy", "phylip")
```

```
{0: 'COATB_BPIKE/30-81', 1: 'Q9T0Q8_BPIKE/1-52', 2: 'COATB_BPI22/32-83', 3: 'COATB_BPM13/24-72', 4:
```

```
Out[16]: 1
```

Here is the new (strict) PHYLIP format output:

```
7 52
seq0       AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIRLFKKFSS
seq1       AEPNAATNYA TEAMDSLKTQ AIDLISQTWP VVTTVVVAGL VIKLFKKFVS
seq2       DGTSTATSYA TEAMNSLKTQ ATDLIDQTWP VVTSVAVAGL AIRLFKKFSS
seq3       AEGDDP---A KAAFNSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
seq4       AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFAS
seq5       AEGDDP---A KAAFDSLQAS ATEYIGYAWA MVVVIVGATI GIKLFKKFTS
seq6       FAADDATSQA KAAFDSLTAQ ATEMSGYAWA LVVLVVGATV GIKLFKKFVS

           KA
           RA
           KA
           KA
           KA
           KA
           RA
```

In general, because of the identifier limitation, working with *strict* PHYLIP file formats shouldn't be your first choice. Using the PFAM/Stockholm format on the other hand allows you to record a lot of additional annotation too.

7.2.2 Getting your alignment objects as formatted strings

The `Bio.AlignIO` interface is based on handles, which means if you want to get your alignment(s) into a string in a particular file format you need to do a little bit more work (see below). However, you will probably prefer to take advantage of the alignment object's `format()` method. This takes a single mandatory argument, a lower case string which is supported by `Bio.AlignIO` as an output format. For example:

```
from Bio import AlignIO
alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
print(alignment.format("clustal"))
```

As described in Section [sec:SeqRecord-format], the `SeqRecord` object has a similar method using output formats supported by `Bio.SeqIO`.

Internally the `format()` method is using the `StringIO` string based handle and calling `Bio.AlignIO.write()`. You can do this in your own code if for example you are using an older version of Biopython:

```
from Bio import AlignIO
from StringIO import StringIO

alignments = AlignIO.parse("PF05371_seed.sth", "stockholm")

out_handle = StringIO()
AlignIO.write(alignments, out_handle, "clustal")
clustal_data = out_handle.getvalue()

print(clustal_data)
```

7.3 Manipulating Alignments

Now that we've covered loading and saving alignments, we'll look at what else you can do with them.

7.3.1 Slicing alignments

First of all, in some senses the alignment objects act like a Python `list` of `SeqRecord` objects (the rows). With this model in mind hopefully the actions of `len()` (the number of rows) and iteration (each row as a `SeqRecord`) make sense:

```
In [17]: from Bio import AlignIO
         alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
         print("Number of rows: %i" % len(alignment))
```

```
Number of rows: 7
```

```
In [18]: for record in alignment:
             print("%s - %s" % (record.seq, record.id))
```

```
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA - COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA - Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA - COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA - COATB_BPZJ2/1-49
```

```
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA - Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA - COATB_BPIF1/22-73
```

You can also use the list-like `append` and `extend` methods to add more rows to the alignment (as `SeqRecord` objects). Keeping the list metaphor in mind, simple slicing of the alignment should also make sense - it selects some of the rows giving back another alignment object:

```
In [19]: print(alignment)
```

```
SingleLetterAlphabet() alignment with 7 rows and 52 columns
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRL...SKA COATB_BPIKE/30-81
AEPNAATNYATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKL...SRA Q9T0Q8_BPIKE/1-52
DGTSTATSYATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRL...SKA COATB_BPI22/32-83
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

```
In [20]: print(alignment[3:7])
```

```
SingleLetterAlphabet() alignment with 4 rows and 52 columns
AEGDDP---AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPM13/24-72
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA COATB_BPZJ2/1-49
AEGDDP---AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKL...SKA Q9T0Q9_BPFD/1-49
FAADDATSQAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKL...SRA COATB_BPIF1/22-73
```

What if you wanted to select by column? Those of you who have used the NumPy matrix or array objects won't be surprised at this - you use a double index.

```
In [21]: print(alignment[2, 6])
```

```
T
```

Using two integer indices pulls out a single letter, short hand for this:

```
In [22]: print(alignment[2].seq[6])
```

```
T
```

You can pull out a single column as a string like this:

```
In [23]: print(alignment[:, 6])
```

```
TTT---T
```

You can also select a range of columns. For example, to pick out those same three rows we extracted earlier, but take just their first six columns:

```
In [24]: print(alignment[3:6, :6])
```

```
SingleLetterAlphabet() alignment with 3 rows and 6 columns
AEGDDP COATB_BPM13/24-72
AEGDDP COATB_BPZJ2/1-49
AEGDDP Q9T0Q9_BPFD/1-49
```

Leaving the first index as `:` means take all the rows:

```
In [25]: print(alignment[:, :6])
```

```
SingleLetterAlphabet() alignment with 7 rows and 6 columns
AEPNAA COATB_BPIKE/30-81
AEPNAA Q9T0Q8_BPIKE/1-52
DGTSTA COATB_BPI22/32-83
AEGDDP COATB_BPM13/24-72
AEGDDP COATB_BPZJ2/1-49
AEGDDP Q9T0Q9_BPFD/1-49
FAADDA COATB_BPIF1/22-73
```

This brings us to a neat way to remove a section. Notice columns 7, 8 and 9 which are gaps in three of the seven sequences:

```
In [26]: print(alignment[:, 6:9])
```

```
SingleLetterAlphabet() alignment with 7 rows and 3 columns
TNY COATB_BPIKE/30-81
TNY Q9T0Q8_BPIKE/1-52
TSY COATB_BPI22/32-83
--- COATB_BPM13/24-72
--- COATB_BPZJ2/1-49
--- Q9T0Q9_BPFD/1-49
TSQ COATB_BPIF1/22-73
```

Again, you can slice to get everything after the ninth column:

```
In [27]: print(alignment[:, 9:])
```

```
SingleLetterAlphabet() alignment with 7 rows and 43 columns
ATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
ATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
ATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
AKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
AKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
```

Now, the interesting thing is that addition of alignment objects works by column. This lets you do this as a way to remove a block of columns:

```
In [28]: edited = alignment[:, :6] + alignment[:, 9:]
         print(edited)
```

```
SingleLetterAlphabet() alignment with 7 rows and 49 columns
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
DGTSTAATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
AEGDDPAKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
FAADDAAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
```

Another common use of alignment addition would be to combine alignments for several different genes into a meta-alignment. Watch out though - the identifiers need to match up (see Section [sec:SeqRecord-addition] for how adding `SeqRecord` objects works). You may find it helpful to first sort the alignment rows alphabetically by id:

```
In [29]: edited.sort()
         print(edited)
```

```
SingleLetterAlphabet() alignment with 7 rows and 49 columns
DGTSTAATEAMNSLKTQATDLIDQTWPVVTSVAVAGLAIRLFKKFSSKA COATB_BPI22/32-83
FAADDAAKAAFDSLTAQATEMSGYAWALVVLVVGATVGIKLFKKFVSRA COATB_BPIF1/22-73
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIRLFKKFSSKA COATB_BPIKE/30-81
AEGDDPAKAAFNSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA COATB_BPM13/24-72
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFASKA COATB_BPZJ2/1-49
AEPNAAATEAMDSLKTQAIDLISQTWPVVTTVVVAGLVIKLFKKFVSRA Q9T0Q8_BPIKE/1-52
AEGDDPAKAAFDSLQASATEYIGYAWAMVVVIVGATIGIKLFKKFTSKA Q9T0Q9_BPFD/1-49
```

Note that you can only add two alignments together if they have the same number of rows.

7.3.2 Alignments as arrays

Depending on what you are doing, it can be more useful to turn the alignment object into an array of letters – and you can do this with NumPy:

```
In [30]: import numpy as np
         from Bio import AlignIO
         alignment = AlignIO.read("data/PF05371_seed.sth", "stockholm")
         align_array = np.array([list(rec) for rec in alignment], np.character)
         print("Array shape %i by %i" % align_array.shape)
```

```
Array shape 7 by 52
```

If you will be working heavily with the columns, you can tell NumPy to store the array by column (as in Fortran) rather then its default of by row (as in C):

```
In [31]: align_array = np.array([list(rec) for rec in alignment], np.character, order="F")
```

Note that this leaves the original Biopython alignment object and the NumPy array in memory as separate objects - editing one will not update the other!

7.4 Alignment Tools

There are *lots* of algorithms out there for aligning sequences, both pairwise alignments and multiple sequence alignments. These calculations are relatively slow, and you generally wouldn't want to write such an algorithm in Python. Instead, you can use Biopython to invoke a command line tool on your behalf. Normally you would:

1. Prepare an input file of your unaligned sequences, typically this will be a FASTA file which you might create using `Bio.SeqIO` (see Chapter [chapter:Bio.SeqIO]).
2. Call the command line tool to process this input file, typically via one of Biopython's command line wrappers (which we'll discuss here).
3. Read the output from the tool, i.e. your aligned sequences, typically using `Bio.AlignIO` (see earlier in this chapter).

All the command line wrappers we're going to talk about in this chapter follow the same style. You create a command line object specifying the options (e.g. the input filename and the output filename), then invoke this command line via a Python operating system call (e.g. using the `subprocess` module).

Most of these wrappers are defined in the `Bio.Align.Applications` module:

```
In [32]: import Bio.Align.Applications
         dir(Bio.Align.Applications)
```

```
Out[32]: ['ClustalOmegaCommandline',
          'ClustalwCommandline',
          'DialignCommandline',
          'MSAProbsCommandline',
          'MafftCommandline',
          'MuscleCommandline',
          'PrankCommandline',
          'ProbconsCommandline',
          'TCoffeeCommandline',
          '_ClustalOmega',
          '_Clustalw',
          '_Dialign',
          '_MSAProbs',
          '_Mafft',
          '_Muscle',
          '_Prank',
          '_Probcons',
          '_TCoffee',
          '__all__',
          '__builtins__',
          '__cached__',
          '__doc__',
          '__docformat__',
          '__file__',
          '__loader__',
          '__name__',
          '__package__',
          '__path__',
          '__spec__']
```

(Ignore the entries starting with an underscore – these have special meaning in Python.) The module `Bio.Emboss.Applications` has wrappers for some of the EMBOSS suite, including `needle` and `water`, which are described below in Section [seq:emboss-needle-water], and wrappers for the EMBOSS packaged versions of the PHYLIP tools (which EMBOSS refer to as one of their EMBASSY packages - third party tools with an EMBOSS style interface). We won't explore all these alignment tools here in the section, just a sample, but the same principles apply.

7.4.1 ClustalW

ClustalW is a popular command line tool for multiple sequence alignment (there is also a graphical interface called ClustalX). Biopython's `Bio.Align.Applications` module has a wrapper for this alignment tool (and several others).

Before trying to use ClustalW from within Python, you should first try running the ClustalW tool yourself by hand at the command line, to familiarise yourself the other options. You'll find the Biopython wrapper is very faithful to the actual command line API:

```
In [33]: from Bio.Align.Applications import ClustalwCommandline
         help(ClustalwCommandline)
```

```
Help on class ClustalwCommandline in module Bio.Align.Applications._Clustalw:

class ClustalwCommandline(Bio.Application.AbstractCommandline)
 |  Command line wrapper for clustalw (version one or two).
 |
 |  http://www.clustal.org/
 |
 |  Example:
 |  --------
 |
 |  >>> from Bio.Align.Applications import ClustalwCommandline
 |  >>> in_file = "unaligned.fasta"
 |  >>> clustalw_cline = ClustalwCommandline("clustalw2", infile=in_file)
 |  >>> print(clustalw_cline)
 |  clustalw2 -infile=unaligned.fasta
 |
 |  You would typically run the command line with clustalw_cline() or via
 |  the Python subprocess module, as described in the Biopython tutorial.
 |
 |  Citation:
 |  ---------
 |
 |  Larkin MA, Blackshields G, Brown NP, Chenna R, McGettigan PA,
 |  McWilliam H, Valentin F, Wallace IM, Wilm A, Lopez R, Thompson JD,
 |  Gibson TJ, Higgins DG. (2007). Clustal W and Clustal X version 2.0.
 |  Bioinformatics, 23, 2947-2948.
 |
 |  Last checked against versions: 1.83 and 2.1
 |
 |  Method resolution order:
 |      ClustalwCommandline
 |      Bio.Application.AbstractCommandline
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, cmd='clustalw', **kwargs)
 |      Create a new instance of a command line wrapper object.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.Application.AbstractCommandline:
 |
 |  __call__(self, stdin=None, stdout=True, stderr=True, cwd=None, env=None)
 |      Executes the command, waits for it to finish, and returns output.
 |
 |      Runs the command line tool and waits for it to finish. If it returns
 |      a non-zero error level, an exception is raised. Otherwise two strings
 |      are returned containing stdout and stderr.
 |
 |      The optional stdin argument should be a string of data which will be
 |      passed to the tool as standard input.
 |
 |      The optional stdout and stderr argument may be filenames (string),
 |      but otherwise are treated as a booleans, and control if the output
 |      should be captured as strings (True, default), or ignored by sending
 |      it to /dev/null to avoid wasting memory (False). If sent to a file
 |      or ignored, then empty string(s) are returned.
 |
 |      The optional cwd argument is a string giving the working directory
 |      to run the command from. See Python's subprocess module documentation
 |      for more details.
 |
 |      The optional env argument is a dictionary setting the environment
 |      variables to be used in the new process. By default the current
 |      process' environment variables are used. See Python's subprocess
 |      module documentation for more details.
 |
 |      Default example usage::
 |
 |          from Bio.Emboss.Applications import WaterCommandline
 |          water_cmd = WaterCommandline(gapopen=10, gapextend=0.5,
 |                                       stdout=True, auto=True,
 |                                       asequence="a.fasta", bsequence="b.fasta")
 |          print("About to run: %s" % water_cmd)
 |          std_output, err_output = water_cmd()
 |
 |      This functionality is similar to subprocess.check_output() added in
 |      Python 2.7. In general if you require more control over running the
 |      command, use subprocess directly.
 |
 |      As of Biopython 1.56, when the program called returns a non-zero error
 |      level, a custom ApplicationError exception is raised. This includes
 |      any stdout and stderr strings captured as attributes of the exception
 |      object, since they may be useful for diagnosing what went wrong.
 |
 |  __repr__(self)
 |      Return a representation of the command line object for debugging.
 |
 |      e.g.
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
 |      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
 |      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
 |      >>> cline.outfile = "temp_water.txt"
 |      >>> print(cline)
 |      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
 |      >>> cline
 |      WaterCommandline(cmd='water', outfile='temp_water.txt', asequence='asis:ACCCGGGCGCGGT', bsequ
 |
 |  __setattr__(self, name, value)
 |      Set attribute name to value (PRIVATE).
 |
 |      This code implements a workaround for a user interface issue.
 |      Without this __setattr__ attribute-based assignment of parameters
 |      will silently accept invalid parameters, leading to known instances
 |      of the user assuming that parameters for the application are set,
 |      when they are not.
 |
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5, stdout=True)
 |      >>> cline.asequence = "a.fasta"
 |      >>> cline.bsequence = "b.fasta"
 |      >>> cline.csequence = "c.fasta"
 |      Traceback (most recent call last):
 |      ...
 |      ValueError: Option name csequence was not found.
 |      >>> print(cline)
 |      water -stdout -asequence=a.fasta -bsequence=b.fasta -gapopen=10 -gapextend=0.5
 |
 |      This workaround uses a whitelist of object attributes, and sets the
 |      object attribute list as normal, for these.  Other attributes are
 |      assumed to be parameters, and passed to the self.set_parameter method
 |      for validation and assignment.
 |
 |  __str__(self)
 |      Make the commandline string with the currently set options.
 |
 |      e.g.
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
 |      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
 |      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
 |      >>> cline.outfile = "temp_water.txt"
 |      >>> print(cline)
 |      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
 |      >>> str(cline)
 |      'water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -g
 |
 |  set_parameter(self, name, value=None)
 |      Set a commandline option for a program (OBSOLETE).
 |
 |      Every parameter is available via a property and as a named
```

```
 |      keyword when creating the instance. Using either of these is
 |      preferred to this legacy set_parameter method which is now
 |      OBSOLETE, and likely to be DEPRECATED and later REMOVED in
 |      future releases.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.Application.AbstractCommandline:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Bio.Application.AbstractCommandline:
 |
 |  parameters = None
```

For the most basic usage, all you need is to have a FASTA input file, such as opuntia.fasta (available online or in the Doc/examples subdirectory of the Biopython source code). This is a small FASTA file containing seven prickly-pear DNA sequences (from the cactus family *Opuntia*).

By default ClustalW will generate an alignment and guide tree file with names based on the input FASTA file, in this case `opuntia.aln` and `opuntia.dnd`, but you can override this or make it explicit:

```
In [34]: from Bio.Align.Applications import ClustalwCommandline
         cline = ClustalwCommandline("clustalw2", infile="data/opuntia.fasta")
         print(cline)
```

```
clustalw2 -infile=data/opuntia.fasta
```

Notice here we have given the executable name as `clustalw2`, indicating we have version two installed, which has a different filename to version one (`clustalw`, the default). Fortunately both versions support the same set of arguments at the command line (and indeed, should be functionally identical).

You may find that even though you have ClustalW installed, the above command doesn't work – you may get a message about "command not found" (especially on Windows). This indicated that the ClustalW executable is not on your PATH (an environment variable, a list of directories to be searched). You can either update your PATH setting to include the location of your copy of ClustalW tools (how you do this will depend on your OS), or simply type in the full path of the tool. For example:

```
In [35]: import os
         from Bio.Align.Applications import ClustalwCommandline
         clustalw_exe = r"C:\Program Files\new clustal\clustalw2.exe"
         clustalw_cline = ClustalwCommandline(clustalw_exe, infile="data/opuntia.fasta")
```

```
In [37]: assert os.path.isfile(clustalw_exe), "Clustal W executable missing"
         stdout, stderr = clustalw_cline()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-37-2132fefeb91f> in <module>()
----> 1 assert os.path.isfile(clustalw_exe), "Clustal W executable missing"
      2 stdout, stderr = clustalw_cline()

AssertionError: Clustal W executable missing
```

Remember, in Python strings `\n` and `\t` are by default interpreted as a new line and a tab – which is why we're put a letter "r" at the start for a raw string that isn't translated in this way. This is generally good practice when specifying a

Windows style file name.

Internally this uses the `subprocess` module which is now the recommended way to run another program in Python. This replaces older options like the `os.system()` and the `os.popen*` functions.

Now, at this point it helps to know about how command line tools “work”. When you run a tool at the command line, it will often print text output directly to screen. This text can be captured or redirected, via two “pipes”, called standard output (the normal results) and standard error (for error messages and debug messages). There is also standard input, which is any text fed into the tool. These names get shortened to stdin, stdout and stderr. When the tool finishes, it has a return code (an integer), which by convention is zero for success.

When you run the command line tool like this via the Biopython wrapper, it will wait for it to finish, and check the return code. If this is non zero (indicating an error), an exception is raised. The wrapper then returns two strings, stdout and stderr.

In the case of ClustalW, when run at the command line all the important output is written directly to the output files. Everything normally printed to screen while you wait (via stdout or stderr) is boring and can be ignored (assuming it worked).

What we care about are the two output files, the alignment and the guide tree. We didn’t tell ClustalW what filenames to use, but it defaults to picking names based on the input file. In this case the output should be in the file `opuntia.aln`. You should be able to work out how to read in the alignment using `Bio.AlignIO` by now:

```
In [38]: from Bio import AlignIO
         align = AlignIO.read("data/opuntia.aln", "clustal")
         print(align)
```

```
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF191
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF191
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF191
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273291|gb|AF191665.1|AF191
```

In case you are interested (and this is an aside from the main thrust of this chapter), the `opuntia.dnd` file ClustalW creates is just a standard Newick tree file, and `Bio.Phylo` can parse these:

```
In [39]: from Bio import Phylo
         tree = Phylo.read("data/opuntia.dnd", "newick")
         Phylo.draw_ascii(tree)
```

```
                             _______________ gi|6273291|gb|AF191665.1|AF191665
  __________________________|
 |                          |   ______ gi|6273290|gb|AF191664.1|AF191664
 |                          |__|
 |                             |_____ gi|6273289|gb|AF191663.1|AF191663
 |
_|_________________ gi|6273287|gb|AF191661.1|AF191661
 |
 |__________ gi|6273286|gb|AF191660.1|AF191660
 |
 |    __ gi|6273285|gb|AF191659.1|AF191659
 |___|
     | gi|6273284|gb|AF191658.1|AF191658
```

Chapter [sec:Phylo] covers Biopython’s support for phylogenetic trees in more depth.

7.4.2 MUSCLE

MUSCLE is a more recent multiple sequence alignment tool than ClustalW, and Biopython also has a wrapper for it under the `Bio.Align.Applications` module. As before, we recommend you try using MUSCLE from the command line before trying it from within Python, as the Biopython wrapper is very faithful to the actual command line API:

```
In [40]: from Bio.Align.Applications import MuscleCommandline
         help(MuscleCommandline)
```

```
Help on class MuscleCommandline in module Bio.Align.Applications._Muscle:

class MuscleCommandline(Bio.Application.AbstractCommandline)
 |  Command line wrapper for the multiple alignment program MUSCLE.
 |
 |  http://www.drive5.com/muscle/
 |
 |  Example:
 |  --------
 |
 |  >>> from Bio.Align.Applications import MuscleCommandline
 |  >>> muscle_exe = r"C:\Program Files\Aligments\muscle3.8.31_i86win32.exe"
 |  >>> in_file = r"C:\My Documents\unaligned.fasta"
 |  >>> out_file = r"C:\My Documents\aligned.fasta"
 |  >>> muscle_cline = MuscleCommandline(muscle_exe, input=in_file, out=out_file)
 |  >>> print(muscle_cline)
 |  "C:\Program Files\Aligments\muscle3.8.31_i86win32.exe" -in "C:\My Documents\unaligned.fasta" -out
 |
 |  You would typically run the command line with muscle_cline() or via
 |  the Python subprocess module, as described in the Biopython tutorial.
 |
 |  Citations:
 |  ----------
 |
 |  Edgar, Robert C. (2004), MUSCLE: multiple sequence alignment with high
 |  accuracy and high throughput, Nucleic Acids Research 32(5), 1792-97.
 |
 |  Edgar, R.C. (2004) MUSCLE: a multiple sequence alignment method with
 |  reduced time and space complexity. BMC Bioinformatics 5(1): 113.
 |
 |  Last checked against version: 3.7, briefly against 3.8
 |
 |  Method resolution order:
 |      MuscleCommandline
 |      Bio.Application.AbstractCommandline
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, cmd='muscle', **kwargs)
 |      Create a new instance of a command line wrapper object.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.Application.AbstractCommandline:
 |
 |  __call__(self, stdin=None, stdout=True, stderr=True, cwd=None, env=None)
 |      Executes the command, waits for it to finish, and returns output.
 |
 |      Runs the command line tool and waits for it to finish. If it returns
 |      a non-zero error level, an exception is raised. Otherwise two strings
 |      are returned containing stdout and stderr.
 |
 |      The optional stdin argument should be a string of data which will be
 |      passed to the tool as standard input.
 |
 |      The optional stdout and stderr argument may be filenames (string),
 |      but otherwise are treated as a booleans, and control if the output
 |      should be captured as strings (True, default), or ignored by sending
 |      it to /dev/null to avoid wasting memory (False). If sent to a file
 |      or ignored, then empty string(s) are returned.
 |
 |      The optional cwd argument is a string giving the working directory
 |      to run the command from. See Python's subprocess module documentation
 |      for more details.
 |
 |      The optional env argument is a dictionary setting the environment
 |      variables to be used in the new process. By default the current
 |      process' environment variables are used. See Python's subprocess
 |      module documentation for more details.
 |
 |      Default example usage::
 |
 |          from Bio.Emboss.Applications import WaterCommandline
 |          water_cmd = WaterCommandline(gapopen=10, gapextend=0.5,
 |                                       stdout=True, auto=True,
 |                                       asequence="a.fasta", bsequence="b.fasta")
 |          print("About to run: %s" % water_cmd)
 |          std_output, err_output = water_cmd()
 |
 |      This functionality is similar to subprocess.check_output() added in
 |      Python 2.7. In general if you require more control over running the
 |      command, use subprocess directly.
 |
 |      As of Biopython 1.56, when the program called returns a non-zero error
 |      level, a custom ApplicationError exception is raised. This includes
 |      any stdout and stderr strings captured as attributes of the exception
 |      object, since they may be useful for diagnosing what went wrong.
 |
 |  __repr__(self)
 |      Return a representation of the command line object for debugging.
 |
 |      e.g.
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
 |      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
 |      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
 |      >>> cline.outfile = "temp_water.txt"
 |      >>> print(cline)
 |      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
 |      >>> cline
 |      WaterCommandline(cmd='water', outfile='temp_water.txt', asequence='asis:ACCCGGGCGCGGT', bsequ
 |
 |  __setattr__(self, name, value)
 |      Set attribute name to value (PRIVATE).
 |
 |      This code implements a workaround for a user interface issue.
 |      Without this __setattr__ attribute-based assignment of parameters
 |      will silently accept invalid parameters, leading to known instances
```

```
 |      of the user assuming that parameters for the application are set,
 |      when they are not.
 |
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5, stdout=True)
 |      >>> cline.asequence = "a.fasta"
 |      >>> cline.bsequence = "b.fasta"
 |      >>> cline.csequence = "c.fasta"
 |      Traceback (most recent call last):
 |      ...
 |      ValueError: Option name csequence was not found.
 |      >>> print(cline)
 |      water -stdout -asequence=a.fasta -bsequence=b.fasta -gapopen=10 -gapextend=0.5
 |
 |      This workaround uses a whitelist of object attributes, and sets the
 |      object attribute list as normal, for these.  Other attributes are
 |      assumed to be parameters, and passed to the self.set_parameter method
 |      for validation and assignment.
 |
 |  __str__(self)
 |      Make the commandline string with the currently set options.
 |
 |      e.g.
 |      >>> from Bio.Emboss.Applications import WaterCommandline
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
 |      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
 |      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
 |      >>> cline.outfile = "temp_water.txt"
 |      >>> print(cline)
 |      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
 |      >>> str(cline)
 |      'water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -g
 |
 |  set_parameter(self, name, value=None)
 |      Set a commandline option for a program (OBSOLETE).
 |
 |      Every parameter is available via a property and as a named
 |      keyword when creating the instance. Using either of these is
 |      preferred to this legacy set_parameter method which is now
 |      OBSOLETE, and likely to be DEPRECATED and later REMOVED in
 |      future releases.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.Application.AbstractCommandline:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Bio.Application.AbstractCommandline:
 |
 |  parameters = None
```

For the most basic usage, all you need is to have a FASTA input file, such as opuntia.fasta (available online or in the Doc/examples subdirectory of the Biopython source code). You can then tell MUSCLE to read in this FASTA file, and write the alignment to an output file:

```
In [41]: from Bio.Align.Applications import MuscleCommandline
         cline = MuscleCommandline(input="data/opuntia.fasta", out="opuntia.txt")
         print(cline)
```

```
muscle -in data/opuntia.fasta -out opuntia.txt
```

Note that MUSCLE uses "-in" and "-out" but in Biopython we have to use "input" and "out" as the keyword arguments or property names. This is because "in" is a reserved word in Python.

By default MUSCLE will output the alignment as a FASTA file (using gapped sequences). The `Bio.AlignIO` module should be able to read this alignment using `format=fasta`. You can also ask for ClustalW-like output:

```
In [42]: from Bio.Align.Applications import MuscleCommandline
         cline = MuscleCommandline(input="data/opuntia.fasta", out="opuntia.aln", clw=True)
         print(cline)
```

```
muscle -in data/opuntia.fasta -out opuntia.aln -clw
```

Or, strict ClustalW output where the original ClustalW header line is used for maximum compatibility:

```
In [43]: from Bio.Align.Applications import MuscleCommandline
         cline = MuscleCommandline(input="data/opuntia.fasta", out="opuntia.aln", clwstrict=True)
         print(cline)
```

```
muscle -in data/opuntia.fasta -out opuntia.aln -clwstrict
```

The `Bio.AlignIO` module should be able to read these alignments using `format=clustal`.

MUSCLE can also output in GCG MSF format (using the `msf` argument), but Biopython can't currently parse that, or using HTML which would give a human readable web page (not suitable for parsing).

You can also set the other optional parameters, for example the maximum number of iterations. See the built in help for details.

You would then run MUSCLE command line string as described above for ClustalW, and parse the output using `Bio.AlignIO` to get an alignment object.

7.4.3 MUSCLE using stdout

Using a MUSCLE command line as in the examples above will write the alignment to a file. This means there will be no important information written to the standard out (stdout) or standard error (stderr) handles. However, by default MUSCLE will write the alignment to standard output (stdout). We can take advantage of this to avoid having a temporary output file! For example:

```
In [44]: from Bio.Align.Applications import MuscleCommandline
         muscle_cline = MuscleCommandline(input="data/opuntia.fasta")
         print(muscle_cline)
```

```
muscle -in data/opuntia.fasta
```

If we run this via the wrapper, we get back the output as a string. In order to parse this we can use `StringIO` to turn it into a handle. Remember that MUSCLE defaults to using FASTA as the output format:

```
In [45]: from Bio.Align.Applications import MuscleCommandline
         muscle_cline = MuscleCommandline(input="data/opuntia.fasta")
         stdout, stderr = muscle_cline()
         from io import StringIO
         from Bio import AlignIO
         align = AlignIO.read(StringIO(stdout), "fasta")
         print(align)
```

```
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF191663
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273291|gb|AF191665.1|AF191665
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF191664
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF191661
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF191660
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF191659
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF191658
```

The above approach is fairly simple, but if you are dealing with very large output text the fact that all of stdout and stderr is loaded into memory as a string can be a potential drawback. Using the `subprocess` module we can work directly with handles instead:

```
In [49]: import subprocess
         import sys
         from Bio.Align.Applications import MuscleCommandline
         muscle_cline = MuscleCommandline(input="data/opuntia.fasta")
         child = subprocess.Popen(str(muscle_cline),
             stdout=subprocess.PIPE, stderr=subprocess.PIPE,
             shell=(sys.platform != "win32"),
             universal_newlines=True)
         from Bio import AlignIO
         align = AlignIO.read(child.stdout, "fasta")
         print(align)
```

```
SingleLetterAlphabet() alignment with 7 rows and 906 columns
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF191663
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273291|gb|AF191665.1|AF191665
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF191664
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF191661
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF191660
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF191659
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF191658
```

7.4.4 MUSCLE using stdin and stdout

We don't actually *need* to have our FASTA input sequences prepared in a file, because by default MUSCLE will read in the input sequence from standard input! Note this is a bit more advanced and fiddly, so don't bother with this technique unless you need to.

First, we'll need some unaligned sequences in memory as `SeqRecord` objects. For this demonstration I'm going to use a filtered version of the original FASTA file (using a generator expression), taking just six of the seven sequences:

```
In [73]: from Bio import SeqIO
         records = (r for r in SeqIO.parse("data/opuntia.fasta", "fasta") if len(r) < 900)
```

Then we create the MUSCLE command line, leaving the input and output to their defaults (stdin and stdout). I'm also going to ask for strict ClustalW format as for the output.

```
In [90]: from Bio.Align.Applications import MuscleCommandline
         muscle_cline = MuscleCommandline(clwstrict=True)
         print(muscle_cline)
```

```
muscle -clwstrict
```

Now for the fiddly bits using the `subprocess` module, stdin and stdout:

```
In [91]: import subprocess
         import sys
         child = subprocess.Popen(str(cline),
             stdin=subprocess.PIPE, stdout=subprocess.PIPE,
             stderr=subprocess.PIPE, universal_newlines=True,
             shell=(sys.platform != "win32"))
```

That should start MUSCLE, but it will be sitting waiting for its FASTA input sequences, which we must supply via its stdin handle:

```
In [92]: SeqIO.write(records, child.stdin, "fasta")

---------------------------------------------------------------------------
BrokenPipeError                           Traceback (most recent call last)
<ipython-input-92-11690e1b720b> in <module>()
----> 1 SeqIO.write(records, child.stdin, "fasta")

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/__init__.py in write(sequences, han
    481         if format in _FormatToWriter:
    482             writer_class = _FormatToWriter[format]
--> 483             count = writer_class(fp).write_file(sequences)
    484         elif format in AlignIO._FormatToWriter:
    485             # Try and turn all the records into a single alignment,

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/Interfaces.py in write_file(self, r
    209         """
    210         self.write_header()
--> 211         count = self.write_records(records)
    212         self.write_footer()
    213         return count

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/Interfaces.py in write_records(self
    194         count = 0
    195         for record in records:
--> 196             self.write_record(record)
    197             count += 1
    198         # Mark as true, even if there where no records

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/FastaIO.py in write_record(self, re
    209         if self.wrap:
    210             for i in range(0, len(data), self.wrap):
--> 211                 self.handle.write(data[i:i + self.wrap] + "\n")
    212         else:
    213             self.handle.write(data + "\n")

BrokenPipeError: [Errno 32] Broken pipe
```

```
In [93]: child.stdin.close()

---------------------------------------------------------------------------
BrokenPipeError                           Traceback (most recent call last)
BrokenPipeError: [Errno 32] Broken pipe

During handling of the above exception, another exception occurred:

BrokenPipeError                           Traceback (most recent call last)
<ipython-input-93-f5a7c02c7fa3> in <module>()
----> 1 child.stdin.close()

BrokenPipeError: [Errno 32] Broken pipe
```

After writing the six sequences to the handle, MUSCLE will still be waiting to see if that is all the FASTA sequences or not – so we must signal that this is all the input data by closing the handle. At that point MUSCLE should start to run, and we can ask for the output:

```
In [94]: from Bio import AlignIO
         align = AlignIO.read(child.stdout, "clustal")
         print(align)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-94-240021347a83> in <module>()
      1 from Bio import AlignIO
----> 2 align = AlignIO.read(child.stdout, "clustal")
      3 print(align)

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py in read(handle, forma
    429         first = None
    430     if first is None:
--> 431         raise ValueError("No records found in handle")
    432     try:
    433         second = next(iterator)

ValueError: No records found in handle
```

Wow! There we are with a new alignment of just the six records, without having created a temporary FASTA input file, or a temporary alignment output file. However, a word of caution: Dealing with errors with this style of calling external programs is much more complicated. It also becomes far harder to diagnose problems, because you can't try running MUSCLE manually outside of Biopython (because you don't have the input file to supply). There can also be subtle cross platform issues (e.g. Windows versus Linux, Python 2 versus Python 3), and how you run your script can have an impact (e.g. at the command line, from IDLE or an IDE, or as a GUI script). These are all generic Python issues though, and not specific to Biopython.

If you find working directly with `subprocess` like this scary, there is an alternative. If you execute the tool with `muscle_cline()` you can supply any standard input as a big string, `muscle_cline(stdin=...)`. So, provided your data isn't very big, you can prepare the FASTA input in memory as a string using `StringIO` (see Section [sec:appendix-handles]):

```
In [95]: from Bio import SeqIO
         records = (r for r in SeqIO.parse("data/opuntia.fasta", "fasta") if len(r) < 900)
         handle = StringIO()
         SeqIO.write(records, handle, "fasta")
```

```
Out[95]: 6
```

```
In [96]: data = handle.getvalue()
```

You can then run the tool and parse the alignment as follows:

```
In [97]: stdout, stderr = muscle_cline(stdin=data)
         from Bio import AlignIO
         align = AlignIO.read(StringIO(stdout), "clustal")
         print(align)
```

```
SingleLetterAlphabet() alignment with 6 rows and 900 columns
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF19166
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF19166
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273287|gb|AF191661.1|AF19166
TATACATAAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273286|gb|AF191660.1|AF19166
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273285|gb|AF191659.1|AF19165
TATACATTAAAGAAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273284|gb|AF191658.1|AF19165
```

You might find this easier, but it does require more memory (RAM) for the strings used for the input FASTA and output Clustal formatted data.

7.4.5 EMBOSS needle and water

The EMBOSS suite includes the `water` and `needle` tools for Smith-Waterman algorithm local alignment, and Needleman-Wunsch global alignment. The tools share the same style interface, so switching between the two is trivial – we'll just use `needle` here.

Suppose you want to do a global pairwise alignment between two sequences, prepared in FASTA format as follows:

```
>HBA_HUMAN
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG
KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTP
AVHASLDKFLASVSTVLTSKYR
```

in a file `alpha.fasta`, and secondly in a file `beta.fasta`:

```
>HBB_HUMAN
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFG
KEFTPPVQAAYQKVVAGVANALAHKYH
```

Let's start by creating a complete `needle` command line object in one go:

```
In [98]: from Bio.Emboss.Applications import NeedleCommandline
         needle_cline = NeedleCommandline(asequence="data/alpha.faa", bsequence="data/beta.faa",
             gapopen=10, gapextend=0.5, outfile="needle.txt")
         print(needle_cline)
```

```
needle -outfile=needle.txt -asequence=data/alpha.faa -bsequence=data/beta.faa -gapopen=10 -gapextend=
```

Why not try running this by hand at the command prompt? You should see it does a pairwise comparison and records the output in the file `needle.txt` (in the default EMBOSS alignment file format).

Even if you have EMBOSS installed, running this command may not work – you might get a message about "command not found" (especially on Windows). This probably means that the EMBOSS tools are not on your PATH environment variable. You can either update your PATH setting, or simply tell Biopython the full path to the tool, for example:

```
In [99]: from Bio.Emboss.Applications import NeedleCommandline
         needle_cline = NeedleCommandline(r"C:\EMBOSS\needle.exe",
             asequence="data/alpha.faa", bsequence="data/beta.faa",
             gapopen=10, gapextend=0.5, outfile="needle.txt")
```

Remember in Python that for a default string `\n` or `\t` means a new line or a tab – which is why we're put a letter "r" at the start for a raw string.

At this point it might help to try running the EMBOSS tools yourself by hand at the command line, to familiarise yourself the other options and compare them to the Biopython help text:

```
In [100]: from Bio.Emboss.Applications import NeedleCommandline
          help(NeedleCommandline)
```

```
Help on class NeedleCommandline in module Bio.Emboss.Applications:

class NeedleCommandline(_EmbossCommandLine)
 |  Commandline object for the needle program from EMBOSS.
 |
 |  Method resolution order:
 |      NeedleCommandline
 |      _EmbossCommandLine
 |      _EmbossMinimalCommandLine
 |      Bio.Application.AbstractCommandline
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, cmd='needle', **kwargs)
 |      Create a new instance of a command line wrapper object.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  aformat
 |      Display output in a different specified output format
 |
 |      This controls the addition of the -aformat parameter and its associated value.  Set this prop
 |
 |  asequence
 |      First sequence to align
 |
 |      This controls the addition of the -asequence parameter and its associated value.  Set this pr
 |
 |  auto
 |      Turn off prompts.
 |
 |                         Automatic mode disables prompting, so we recommend you set
 |                         this argument all the time when calling an EMBOSS tool from
 |                         Biopython.
 |
 |
 |      This property controls the addition of the -auto switch, treat this property as a boolean.
 |
 |  brief
 |      Display brief identity and similarity
 |
 |      This property controls the addition of the -brief switch, treat this property as a boolean.
 |
 |  bsequence
 |      Second sequence to align
 |
 |      This controls the addition of the -bsequence parameter and its associated value.  Set this pr
 |
 |  datafile
 |      Matrix file
 |
 |      This controls the addition of the -datafile parameter and its associated value.  Set this pro
 |
 |  debug
 |      Write debug output to program.dbg.
 |
 |      This property controls the addition of the -debug switch, treat this property as a boolean.
 |
 |  die
 |      Report dying program messages.
 |
 |      This property controls the addition of the -die switch, treat this property as a boolean.
 |
 |  endextend
 |      The score added to the end gap penality for each base or residue in the end gap.
 |
 |      This controls the addition of the -endextend parameter and its associated value.  Set this pr
 |
 |  endopen
```

```
 |      The score taken away when an end gap is created.
 |
 |      This controls the addition of the -endopen parameter and its associated value.  Set this prop
 |
 |  endweight
 |      Apply And gap penalties
 |
 |      This controls the addition of the -endweight parameter and its associated value.  Set this pr
 |
 |  error
 |      Report errors.
 |
 |      This property controls the addition of the -error switch, treat this property as a boolean.
 |
 |  filter
 |      Read standard input, write standard output.
 |
 |      This property controls the addition of the -filter switch, treat this property as a boolean.
 |
 |  gapextend
 |      Gap extension penalty
 |
 |      This controls the addition of the -gapextend parameter and its associated value.  Set this pr
 |
 |  gapopen
 |      Gap open penalty
 |
 |      This controls the addition of the -gapopen parameter and its associated value.  Set this prop
 |
 |  help
 |      Report command line options.
 |
 |                          More information on associated and general qualifiers can
 |                          be found with -help -verbose
 |
 |
 |      This property controls the addition of the -help switch, treat this property as a boolean.
 |
 |  nobrief
 |      Display extended identity and similarity
 |
 |      This property controls the addition of the -nobrief switch, treat this property as a boolean
 |
 |  options
 |      Prompt for standard and additional values.
 |
 |                          If you are calling an EMBOSS tool from within Biopython,
 |                          we DO NOT recommend using this option.
 |
 |
 |      This property controls the addition of the -options switch, treat this property as a boolean
 |
 |  outfile
 |      Output filename
 |
 |      This controls the addition of the -outfile parameter and its associated value.  Set this prop
 |
 |  similarity
 |      Display percent identity and similarity
```

```
|
|     This controls the addition of the -similarity parameter and its associated value.  Set this p
|
|  snucleotide
|      Sequences are nucleotide (boolean)
|
|      This controls the addition of the -snucleotide parameter and its associated value.  Set this
|
|  sprotein
|      Sequences are protein (boolean)
|
|      This controls the addition of the -sprotein parameter and its associated value.  Set this pro
|
|  stdout
|      Write standard output.
|
|      This property controls the addition of the -stdout switch, treat this property as a boolean.
|
|  verbose
|      Report some/full command line options
|
|      This property controls the addition of the -verbose switch, treat this property as a boolean
|
|  warning
|      Report warnings.
|
|      This property controls the addition of the -warning switch, treat this property as a boolean
|
|  ----------------------------------------------------------------------
|  Methods inherited from Bio.Application.AbstractCommandline:
|
|  __call__(self, stdin=None, stdout=True, stderr=True, cwd=None, env=None)
|      Executes the command, waits for it to finish, and returns output.
|
|      Runs the command line tool and waits for it to finish. If it returns
|      a non-zero error level, an exception is raised. Otherwise two strings
|      are returned containing stdout and stderr.
|
|      The optional stdin argument should be a string of data which will be
|      passed to the tool as standard input.
|
|      The optional stdout and stderr argument may be filenames (string),
|      but otherwise are treated as a booleans, and control if the output
|      should be captured as strings (True, default), or ignored by sending
|      it to /dev/null to avoid wasting memory (False). If sent to a file
|      or ignored, then empty string(s) are returned.
|
|      The optional cwd argument is a string giving the working directory
|      to run the command from. See Python's subprocess module documentation
|      for more details.
|
|      The optional env argument is a dictionary setting the environment
|      variables to be used in the new process. By default the current
|      process' environment variables are used. See Python's subprocess
|      module documentation for more details.
|
|      Default example usage::
|
|          from Bio.Emboss.Applications import WaterCommandline
|          water_cmd = WaterCommandline(gapopen=10, gapextend=0.5,
|                                       stdout=True, auto=True,
|                                       asequence="a.fasta", bsequence="b.fasta")
|          print("About to run: %s" % water_cmd)
|          std_output, err_output = water_cmd()
|
|      This functionality is similar to subprocess.check_output() added in
|      Python 2.7. In general if you require more control over running the
|      command, use subprocess directly.
|
|      As of Biopython 1.56, when the program called returns a non-zero error
|      level, a custom ApplicationError exception is raised. This includes
|      any stdout and stderr strings captured as attributes of the exception
|      object, since they may be useful for diagnosing what went wrong.
|
|  __repr__(self)
|      Return a representation of the command line object for debugging.
|
|      e.g.
|      >>> from Bio.Emboss.Applications import WaterCommandline
|      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
|      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
|      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
|      >>> cline.outfile = "temp_water.txt"
|      >>> print(cline)
|      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
|      >>> cline
|      WaterCommandline(cmd='water', outfile='temp_water.txt', asequence='asis:ACCCGGGCGCGGT', bsequ
|
|  __setattr__(self, name, value)
|      Set attribute name to value (PRIVATE).
|
|      This code implements a workaround for a user interface issue.
|      Without this __setattr__ attribute-based assignment of parameters
|      will silently accept invalid parameters, leading to known instances
|      of the user assuming that parameters for the application are set,
|      when they are not.
|
|      >>> from Bio.Emboss.Applications import WaterCommandline
|      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5, stdout=True)
|      >>> cline.asequence = "a.fasta"
|      >>> cline.bsequence = "b.fasta"
|      >>> cline.csequence = "c.fasta"
|      Traceback (most recent call last):
|      ...
|      ValueError: Option name csequence was not found.
|      >>> print(cline)
|      water -stdout -asequence=a.fasta -bsequence=b.fasta -gapopen=10 -gapextend=0.5
|
|      This workaround uses a whitelist of object attributes, and sets the
|      object attribute list as normal, for these.  Other attributes are
|      assumed to be parameters, and passed to the self.set_parameter method
|      for validation and assignment.
|
|  __str__(self)
|      Make the commandline string with the currently set options.
|
|      e.g.
|      >>> from Bio.Emboss.Applications import WaterCommandline
```

```
 |      >>> cline = WaterCommandline(gapopen=10, gapextend=0.5)
 |      >>> cline.asequence = "asis:ACCCGGGCGCGGT"
 |      >>> cline.bsequence = "asis:ACCCGAGCGCGGT"
 |      >>> cline.outfile = "temp_water.txt"
 |      >>> print(cline)
 |      water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -ga
 |      >>> str(cline)
 |      'water -outfile=temp_water.txt -asequence=asis:ACCCGGGCGCGGT -bsequence=asis:ACCCGAGCGCGGT -g
 |
 |  set_parameter(self, name, value=None)
 |      Set a commandline option for a program (OBSOLETE).
 |
 |      Every parameter is available via a property and as a named
 |      keyword when creating the instance. Using either of these is
 |      preferred to this legacy set_parameter method which is now
 |      OBSOLETE, and likely to be DEPRECATED and later REMOVED in
 |      future releases.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.Application.AbstractCommandline:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Bio.Application.AbstractCommandline:
 |
 |  parameters = None
```

Note that you can also specify (or change or look at) the settings like this:

```
In [101]: from Bio.Emboss.Applications import NeedleCommandline
          needle_cline = NeedleCommandline()
          needle_cline.asequence="data/alpha.faa"
          needle_cline.bsequence="data/beta.faa"
          needle_cline.gapopen=10
          needle_cline.gapextend=0.5
          needle_cline.outfile="needle.txt"
          print(needle_cline)
```

```
needle -outfile=needle.txt -asequence=data/alpha.faa -bsequence=data/beta.faa -gapopen=10 -gapextend=
```

```
In [102]: print(needle_cline.outfile)
```

```
needle.txt
```

Next we want to use Python to run this command for us. As explained above, for full control, we recommend you use the built in Python `subprocess` module, but for simple usage the wrapper object usually suffices:

```
In [103]: stdout, stderr = needle_cline()
          print(stdout + stderr)
```

```
---------------------------------------------------------------------------
ApplicationError                          Traceback (most recent call last)
<ipython-input-103-c1c0502efb05> in <module>()
----> 1 stdout, stderr = needle_cline()
      2 print(stdout + stderr)
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/Application/__init__.py in __call__(self,
    515         if return_code:
    516             raise ApplicationError(return_code, str(self),
--> 517                                    stdout_str, stderr_str)    518         return stdout_str,
    519

ApplicationError: Non-zero return code 127 from 'needle -outfile=needle.txt -asequence=data/alpha.faa
```

Next we can load the output file with `Bio.AlignIO` as discussed earlier in this chapter, as the `emboss` format:

```
In [104]: from Bio import AlignIO
          align = AlignIO.read("needle.txt", "emboss")
          print(align)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-104-cb0b4e27470d> in <module>()
      1 from Bio import AlignIO
----> 2 align = AlignIO.read("needle.txt", "emboss")
      3 print(align)

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py in read(handle, forma
    425     iterator = parse(handle, format, seq_count, alphabet)
    426     try:
--> 427         first = next(iterator)
    428     except StopIteration:
    429         first = None

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/AlignIO/__init__.py in parse(handle, form
    348         raise TypeError("Need integer for seq_count (sequences per alignment)")
    349
--> 350     with as_handle(handle, 'rU') as fp:
    351         # Map the file format to a sequence iterator:
    352         if format in _FormatToIterator:

/home/tiago_antao/miniconda/lib/python3.5/contextlib.py in __enter__(self)
     57     def __enter__(self):
     58         try:
---> 59             return next(self.gen)
     60         except StopIteration:
     61             raise RuntimeError("generator didn't yield") from None

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/File.py in as_handle(handleish, mode, **k
     88                 yield fp
     89         else:
---> 90             with open(handleish, mode, **kwargs) as fp:
     91                 yield fp
     92     else:

FileNotFoundError: [Errno 2] No such file or directory: 'needle.txt'
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 8

BLAST

Hey, everybody loves BLAST right? I mean, geez, how can it get any easier to do comparisons between one of your sequences and every other sequence in the known world? But, of course, this section isn't about how cool BLAST is, since we already know that. It is about the problem with BLAST – it can be really difficult to deal with the volume of data generated by large runs, and to automate BLAST runs in general.

Fortunately, the Biopython folks know this only too well, so they've developed lots of tools for dealing with BLAST and making things much easier. This section details how to use these tools and do useful things with them.

Dealing with BLAST can be split up into two steps, both of which can be done from within Biopython. Firstly, running BLAST for your query sequence(s), and getting some output. Secondly, parsing the BLAST output in Python for further analysis.

Your first introduction to running BLAST was probably via the NCBI web-service. In fact, there are lots of ways you can run BLAST, which can be categorised in several ways. The most important distinction is running BLAST locally (on your own machine), and running BLAST remotely (on another machine, typically the NCBI servers). We're going to start this chapter by invoking the NCBI online BLAST service from within a Python script.

NOTE: The following Chapter [chapter:searchio] describes `Bio.SearchIO`, an *experimental* module in Biopython. We intend this to ultimately replace the older `Bio.Blast` module, as it provides a more general framework handling other related sequence searching tools as well. However, until that is declared stable, for production code please continue to use the `Bio.Blast` module for dealing with NCBI BLAST.

8.1 Running BLAST over the Internet

We use the function `qblast()` in the `Bio.Blast.NCBIWWW` module to call the online version of BLAST. This has three non-optional arguments:

- The first argument is the blast program to use for the search, as a lower case string. The options and descriptions of the programs are available at http://www.ncbi.nlm.nih.gov/BLAST/blast_program.shtml. Currently `qblast` only works with blastn, blastp, blastx, tblast and tblastx.
- The second argument specifies the databases to search against. Again, the options for this are available on the NCBI web pages at http://www.ncbi.nlm.nih.gov/BLAST/blast_databases.shtml.
- The third argument is a string containing your query sequence. This can either be the sequence itself, the sequence in fasta format, or an identifier like a GI number.

The `qblast` function also take a number of other option arguments which are basically analogous to the different parameters you can set on the BLAST web page. We'll just highlight a few of them here:

- The `qblast` function can return the BLAST results in various formats, which you can choose with the optional `format_type` keyword: `"HTML"`, `"Text"`, `"ASN.1"`, or `"XML"`. The default is `"XML"`, as that is the format expected by the parser, described in section [sec:parsing-blast] below.
- The argument `expect` sets the expectation or e-value threshold.

For more about the optional BLAST arguments, we refer you to the NCBI's own documentation, or that built into Biopython:

```
In [1]: from Bio.Blast import NCBIWWW
        help(NCBIWWW.qblast)
```

```
Help on function qblast in module Bio.Blast.NCBIWWW:

qblast(program, database, sequence, auto_format=None, composition_based_statistics=None, db_genetic_c
    Do a BLAST search using the QBLAST server at NCBI.

    Supports all parameters of the qblast API for Put and Get.
    Some useful parameters:

     - program        blastn, blastp, blastx, tblastn, or tblastx (lower case)
     - database       Which database to search against (e.g. "nr").
     - sequence       The sequence to search.
     - ncbi_gi        TRUE/FALSE whether to give 'gi' identifier.
     - descriptions   Number of descriptions to show.  Def 500.
     - alignments     Number of alignments to show.  Def 500.
     - expect         An expect value cutoff.  Def 10.0.
     - matrix_name    Specify an alt. matrix (PAM30, PAM70, BLOSUM80, BLOSUM45).
     - filter         "none" turns off filtering.  Default no filtering
     - format_type    "HTML", "Text", "ASN.1", or "XML".  Def. "XML".
     - entrez_query   Entrez query to limit Blast search
     - hitlist_size   Number of hits to return. Default 50
     - megablast      TRUE/FALSE whether to use MEga BLAST algorithm (blastn only)
     - service        plain, psi, phi, rpsblast, megablast (lower case)

    This function does no checking of the validity of the parameters
    and passes the values to the server as is.  More help is available at:
    http://www.ncbi.nlm.nih.gov/BLAST/Doc/urlapi.html
```

Note that the default settings on the NCBI BLAST website are not quite the same as the defaults on QBLAST. If you get different results, you'll need to check the parameters (e.g., the expectation value threshold and the gap values).

For example, if you have a nucleotide sequence you want to search against the nucleotide database (nt) using BLASTN, and you know the GI number of your query sequence, you can use:

```
In [2]: from Bio.Blast import NCBIWWW
        result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

```
---------------------------------------------------------------------------
gaierror                                  Traceback (most recent call last)
/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in do_open(self, http_class, req, **http_
   1239             try:
-> 1240                 h.request(req.get_method(), req.selector, req.data, headers)
   1241             except OSError as err: # timeout error

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in request(self, method, url, body, headers)
   1082         """Send a complete request to the server."""
-> 1083         self._send_request(method, url, body, headers)
   1084

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _send_request(self, method, url, body, he
   1127             body = body.encode('iso-8859-1')
-> 1128         self.endheaders(body)
   1129

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in endheaders(self, message_body)
   1078             raise CannotSendHeader()
-> 1079         self._send_output(message_body)
   1080

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _send_output(self, message_body)
    910
--> 911         self.send(msg)
    912         if message_body is not None:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in send(self, data)
    853             if self.auto_open:
--> 854                 self.connect()
    855             else:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in connect(self)
    825         self.sock = self._create_connection(
--> 826             (self.host,self.port), self.timeout, self.source_address)    827         self.so

/home/tiago_antao/miniconda/lib/python3.5/socket.py in create_connection(address, timeout, source_ad
    692     err = None
--> 693     for res in getaddrinfo(host, port, 0, SOCK_STREAM):
    694         af, socktype, proto, canonname, sa = res

/home/tiago_antao/miniconda/lib/python3.5/socket.py in getaddrinfo(host, port, family, type, proto, f
    731     addrlist = []
--> 732     for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
    733         af, socktype, proto, canonname, sa = res

gaierror: [Errno -3] Temporary failure in name resolution

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
<ipython-input-2-f79554f7040a> in <module>()
      1 from Bio.Blast import NCBIWWW
----> 2 result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/Blast/NCBIWWW.py in qblast(program, datab
    164                                  message,
    165                                  {"User-Agent": "BiopythonClient"})
--> 166         handle = _urlopen(request)
    167         results = _as_string(handle.read())
    168 

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in urlopen(url, data, timeout, cafile, ca
    160     else:
    161         opener = _opener
--> 162     return opener.open(url, data, timeout)
    163 
    164 def install_opener(opener):

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    463             req = meth(req)
    464 
--> 465         response = self._open(req, data)
    466 
    467         # post-process response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _open(self, req, data)
    481         protocol = req.type
    482         result = self._call_chain(self.handle_open, protocol, protocol +
--> 483                                   '_open', req)    484         if result:
    485             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_open(self, req)
   1266 
   1267     def http_open(self, req):
-> 1268         return self.do_open(http.client.HTTPConnection, req)
   1269 
   1270     http_request = AbstractHTTPHandler.do_request_

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in do_open(self, http_class, req, **http_
   1240                 h.request(req.get_method(), req.selector, req.data, headers)
   1241             except OSError as err: # timeout error
-> 1242                 raise URLError(err)
   1243             r = h.getresponse()
   1244         except:

URLError: <urlopen error [Errno -3] Temporary failure in name resolution>
```

Alternatively, if we have our query sequence already in a FASTA formatted file, we just need to open the file and read in this record as a string, and use that as the query argument:

```
In [ ]: from Bio.Blast import NCBIWWW
        fasta_string = open("data/m_cold.fasta").read()
        result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
```

We could also have read in the FASTA file as a `SeqRecord` and then supplied just the sequence itself:

```
In [ ]: from Bio.Blast import NCBIWWW
        from Bio import SeqIO
        record = SeqIO.read("data/m_cold.fasta", format="fasta")
        result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

Supplying just the sequence means that BLAST will assign an identifier for your sequence automatically. You might prefer to use the `SeqRecord` object's format method to make a FASTA string (which will include the existing identifier):

```
In [ ]: from Bio.Blast import NCBIWWW
        from Bio import SeqIO
        record = SeqIO.read("data/m_cold.fasta", format="fasta")
        result_handle = NCBIWWW.qblast("blastn", "nt", record.format("fasta"))
```

This approach makes more sense if you have your sequence(s) in a non-FASTA file format which you can extract using `Bio.SeqIO` (see Chapter *5 - Sequence Input and Output*.)

Whatever arguments you give the `qblast()` function, you should get back your results in a handle object (by default in XML format). The next step would be to parse the XML output into Python objects representing the search results (Section [sec:parsing-blast]), but you might want to save a local copy of the output file first. I find this especially useful when debugging my code that extracts info from the BLAST results (because re-running the online search is slow and wastes the NCBI computer time).

8.2 Saving blast output

We need to be a bit careful since we can use `result_handle.read()` to read the BLAST output only once – calling `result_handle.read()` again returns an empty string.

```
In [ ]: with open("my_blast.xml", "w") as save_to:
            save_to.write(result_handle.read())
            result_handle.close()
```

After doing this, the results are in the file `my_blast.xml` and the original handle has had all its data extracted (so we closed it). However, the `parse` function of the BLAST parser (described in [sec:parsing-blast]) takes a file-handle-like object, so we can just open the saved file for input:

```
In [ ]: with open("my_blast.xml") as result_handle:
            print(result_handle)
```

Now that we've got the BLAST results back into a handle again, we are ready to do something with them, so this leads us right into the parsing section (see Section [sec:parsing-blast] below). You may want to jump ahead to that now ....

8.3 Running BLAST locally

8.3.1 Introduction

Running BLAST locally (as opposed to over the internet, see Section [sec:running-www-blast]) has at least major two advantages:

- Local BLAST may be faster than BLAST over the internet;
- Local BLAST allows you to make your own database to search for sequences against.

Dealing with proprietary or unpublished sequence data can be another reason to run BLAST locally. You may not be allowed to redistribute the sequences, so submitting them to the NCBI as a BLAST query would not be an option.

Unfortunately, there are some major drawbacks too – installing all the bits and getting it setup right takes some effort:

- Local BLAST requires command line tools to be installed.
- Local BLAST requires (large) BLAST databases to be setup (and potentially kept up to date).

To further confuse matters there are several different BLAST packages available, and there are also other tools which can produce imitation BLAST output files, such as BLAT.

8.3.2 Standalone NCBI BLAST+

The "new" NCBI BLAST+ suite was released in 2009. This replaces the old NCBI "legacy" BLAST package (see below).

This section will show briefly how to use these tools from within Python. If you have already read or tried the alignment tool examples in Section [sec:alignment-tools] this should all seem quite straightforward. First, we construct a command line string (as you would type in at the command line prompt if running standalone BLAST by hand). Then we can execute this command from within Python.

For example, taking a FASTA file of gene nucleotide sequences, you might want to run a BLASTX (translation) search against the non-redundant (NR) protein database. Assuming you (or your systems administrator) has downloaded and installed the NR database, you might run:

```
blastx -query opuntia.fasta -db nr -out opuntia.xml -evalue 0.001 -outfmt 5
```

This should run BLASTX against the NR database, using an expectation cut-off value of 0.001 and produce XML output to the specified file (which we can then parse). On my computer this takes about six minutes - a good reason to save the output to a file so you can repeat any analysis as needed.

From within Biopython we can use the NCBI BLASTX wrapper from the `Bio.Blast.Applications` module to build the command line string, and run it:

```
In [ ]: from Bio.Blast.Applications import NcbiblastxCommandline
        help(NcbiblastxCommandline)
```

```
In [ ]: blastx_cline = NcbiblastxCommandline(query="data/opuntia.fasta", db="nr", evalue=0.001,
        outfmt=5, out="opuntia.xml")
        blastx_cline
```

```
In [ ]: print(blastx_cline)
```

TODO: Need to add protein database [nr] or change example

```
In [ ]: # stdout, stderr = blastx_cline()
```

In this example there shouldn't be any output from BLASTX to the terminal, so stdout and stderr should be empty. You may want to check the output file `opuntia.xml` has been created.

As you may recall from earlier examples in the tutorial, the `opuntia.fasta` contains seven sequences, so the BLAST XML output should contain multiple results. Therefore use `Bio.Blast.NCBIXML.parse()` to parse it as described below in Section [sec:parsing-blast].

8.3.3 Other versions of BLAST

NCBI BLAST+ (written in C++) was first released in 2009 as a replacement for the original NCBI "legacy" BLAST (written in C) which is no longer being updated. There were a lot of changes – the old version had a single core command line tool `blastall` which covered multiple different BLAST search types (which are now separate commands in BLAST+), and all the command line options were renamed. Biopython's wrappers for the NCBI "legacy" BLAST tools have been deprecated and will be removed in a future release. To try to avoid confusion, we do not cover calling these old tools from Biopython in this tutorial.

You may also come across Washington University BLAST (WU-BLAST), and its successor, Advanced Biocomputing BLAST (AB-BLAST, released in 2009, not free/open source). These packages include the command line tools `wu-blastall` and `ab-blastall`, which mimicked `blastall` from the NCBI "legacy" BLAST suite. Biopython does not currently provide wrappers for calling these tools, but should be able to parse any NCBI compatible output from them.

8.4 Parsing BLAST output

As mentioned above, BLAST can generate output in various formats, such as XML, HTML, and plain text. Originally, Biopython had parsers for BLAST plain text and HTML output, as these were the only output formats offered at the time. Unfortunately, the BLAST output in these formats kept changing, each time breaking the Biopython parsers. Our HTML BLAST parser has been removed, but the plain text BLAST parser is still available (see Section [sec:parsing-blast-deprecated]). Use it at your own risk, it may or may not work, depending on which BLAST version you're using.

As keeping up with changes in BLAST became a hopeless endeavor, especially with users running different BLAST versions, we now recommend to parse the output in XML format, which can be generated by recent versions of BLAST. Not only is the XML output more stable than the plain text and HTML output, it is also much easier to parse automatically, making Biopython a whole lot more stable.

You can get BLAST output in XML format in various ways. For the parser, it doesn't matter how the output was generated, as long as it is in the XML format.

- You can use Biopython to run BLAST over the internet, as described in section [sec:running-www-blast].
- You can use Biopython to run BLAST locally, as described in section [sec:running-local-blast].
- You can do the BLAST search yourself on the NCBI site through your web browser, and then save the results. You need to choose XML as the format in which to receive the results, and save the final BLAST page you get (you know, the one with all of the interesting results!) to a file.
- You can also run BLAST locally without using Biopython, and save the output in a file. Again, you need to choose XML as the format in which to receive the results.

The important point is that you do not have to use Biopython scripts to fetch the data in order to be able to parse it. Doing things in one of these ways, you then need to get a handle to the results. In Python, a handle is just a nice general way of describing input to any info source so that the info can be retrieved using `read()` and `readline()` functions (see Section sec:appendix-handles).

If you followed the code above for interacting with BLAST through a script, then you already have `result_handle`, the handle to the BLAST results. For example, using a GI number to do an online search:

```
In [ ]: from Bio.Blast import NCBIWWW
        result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

If instead you ran BLAST some other way, and have the BLAST output (in XML format) in the file `my_blast.xml`, all you need to do is to open the file for reading:

```
In [ ]: result_handle = open("my_blast.xml", 'r')
```

Now that we've got a handle, we are ready to parse the output. The code to parse it is really quite small. If you expect a single BLAST result (i.e., you used a single query):

```
In [ ]: from Bio.Blast import NCBIXML
        blast_record = NCBIXML.read(result_handle)
```

or, if you have lots of results (i.e., multiple query sequences):

```
In [ ]: from Bio.Blast import NCBIXML
        blast_records = NCBIXML.parse(result_handle)
```

Just like `Bio.SeqIO` and `Bio.AlignIO` (see Chapters [chapter:Bio.SeqIO] and [chapter:Bio.AlignIO]), we have a pair of input functions, `read` and `parse`, where `read` is for when you have exactly one object, and `parse` is an iterator for when you can have lots of objects – but instead of getting `SeqRecord` or `MultipleSeqAlignment` objects, we get BLAST record objects.

To be able to handle the situation where the BLAST file may be huge, containing thousands of results, `NCBIXML.parse()` returns an iterator. In plain English, an iterator allows you to step through the BLAST output, retrieving BLAST records one by one for each BLAST search result:

TODO: should use an example with more than one record

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)
        blast_record = next(blast_records)
        print(blast_record.database_sequences)
        # # ... do something with blast_record
```

Or, you can use a `for`-loop:

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)
        for blast_record in blast_records:
            print(blast_record.database_sequences)
            # Do something with blast_record
```

Note though that you can step through the BLAST records only once. Usually, from each BLAST record you would save the information that you are interested in. If you want to save all returned BLAST records, you can convert the iterator into a list:

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)
        blast_records = list(blast_records)
        blast_records
```

Now you can access each BLAST record in the list with an index as usual. If your BLAST file is huge though, you may run into memory problems trying to save them all in a list.

Usually, you'll be running one BLAST search at a time. Then, all you need to do is to pick up the first (and only) BLAST record in `blast_records`:

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)
        blast_record = next(blast_records)
```

or more elegantly:

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)
```

I guess by now you're wondering what is in a BLAST record.

8.5 The BLAST record class

A BLAST Record contains everything you might ever want to extract from the BLAST output. Right now we'll just show an example of how to get some info out of the BLAST report, but if you want something in particular that is not described here, look at the info on the record class in detail, and take a gander into the code or automatically generated documentation – the docstrings have lots of good info about what is stored in each piece of information.

To continue with our example, let's just print out some summary info about all hits in our blast report greater than a particular threshold. The following code does this:

```
In [ ]: E_VALUE_THRESH = 0.04
```

```
In [ ]: from Bio.Blast import NCBIXML
        result_handle = open("my_blast.xml", 'r')
        blast_records = NCBIXML.parse(result_handle)

        for alignment in blast_record.alignments:
            for hsp in alignment.hsps:
                if hsp.expect < E_VALUE_THRESH:
                    print('****Alignment****')
                    print('sequence:', alignment.title)
                    print('length:', alignment.length)
                    print('e value:', hsp.expect)
                    print(hsp.query[0:75] + '...')
                    print(hsp.match[0:75] + '...')
                    print(hsp.sbjct[0:75] + '...')
```

Basically, you can do anything you want to with the info in the BLAST report once you have parsed it. This will, of course, depend on what you want to use it for, but hopefully this helps you get started on doing what you need to do!

An important consideration for extracting information from a BLAST report is the type of objects that the information is stored in. In Biopython, the parsers return `Record` objects, either `Blast` or `PSIBlast` depending on what you are parsing. These objects are defined in `Bio.Blast.Record` and are quite complete.

Here are my attempts at UML class diagrams for the `Blast` and `PSIBlast` record classes. If you are good at UML and see mistakes/improvements that can be made, please let me know. The Blast class diagram is shown in Figure

The PSIBlast record object is similar, but has support for the rounds that are used in the iteration steps of PSIBlast. The class diagram for PSIBlast is shown in Figure.

8.6 Deprecated BLAST parsers

8.7 Bio.Blast.NCBIStandalone

The three functions for calling the "legacy" NCBI BLAST command line tools blastall, blastpgp and rpsblast were declared obsolete in Biopython Release 1.53, deprecated in Release 1.61, and removed in Release 1.64. Please use the BLAST+ wrappers in Bio.Blast.Applications instead.

The remainder of this module is a parser for the plain text BLAST output, which was declared obsolete in Release 1.54, and deprecated in Release 1.63.

For some time now, both the NCBI and Biopython have encouraged people to parse the XML output instead, however Bio.SearchIO will initially attempt to support plain text BLAST output.

Older versions of Biopython had parsers for BLAST output in plain text or HTML format. Over the years, we discovered that it is very hard to maintain these parsers in working order. Basically, any small change to the BLAST output in newly released BLAST versions tends to cause the plain text and HTML parsers to break. We therefore recommend parsing BLAST output in XML format, as described in section [sec:parsing-blast].

Depending on which BLAST versions or programs you're using, our plain text BLAST parser may or may not work. Use it at your own risk!

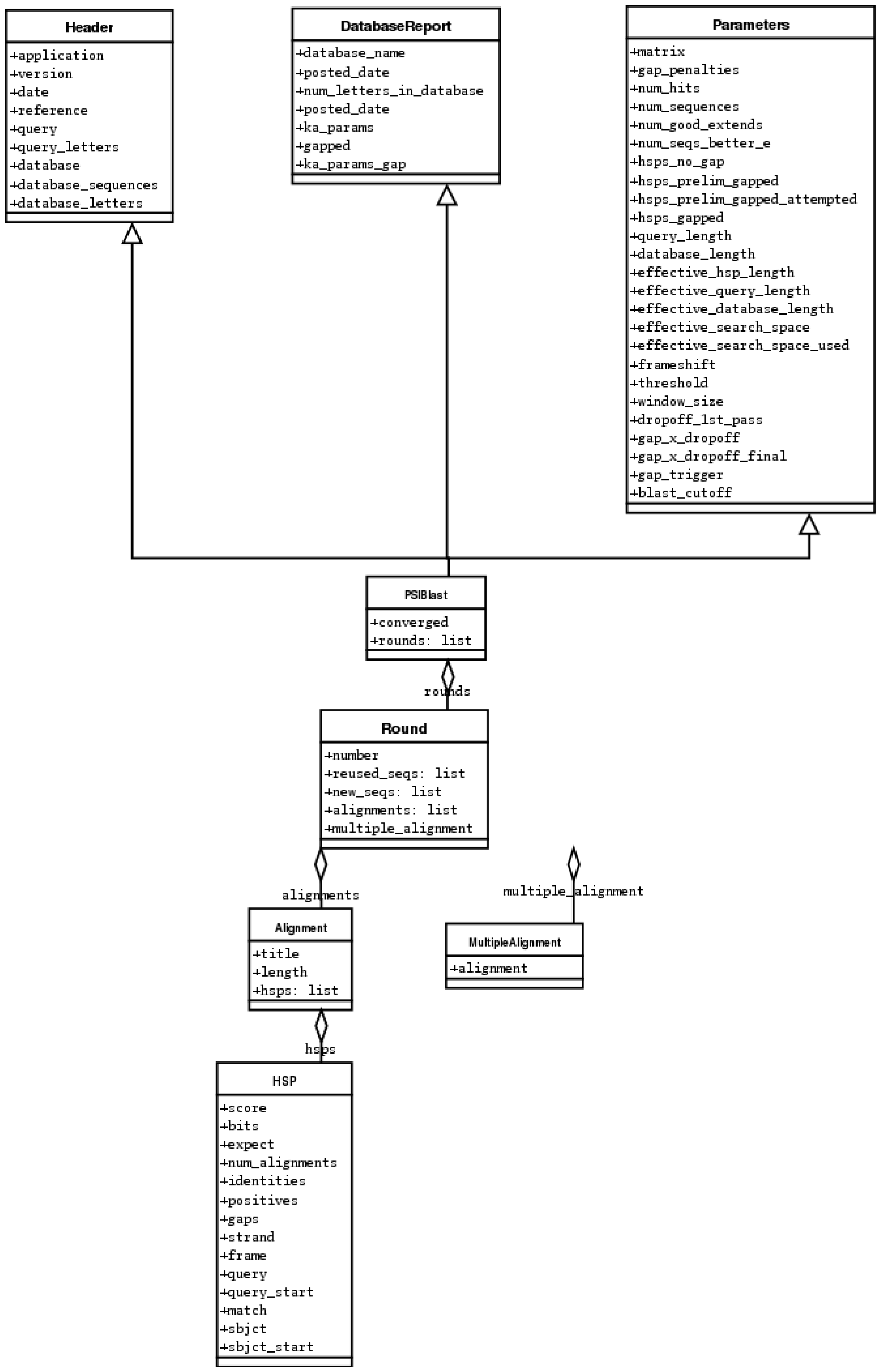


Fig. 8.1: Class diagram for the PSIBlast Record class.

8.7.1 Parsing plain-text BLAST output

The plain text BLAST parser is located in `Bio.Blast.NCBIStandalone`.

As with the XML parser, we need to have a handle object that we can pass to the parser. The handle must implement the `readline()` method and do this properly. The common ways to get such a handle are to either use the provided `blastall` or `blastpgp` functions to run the local blast, or to run a local blast via the command line, and then do something like the following:

```
In [ ]: # result_handle = open("my_file_of_blast_output.txt", 'r')
```

Well, now that we've got a handle (which we'll call `result_handle`), we are ready to parse it. This can be done with the following code:

```
In [ ]: # from Bio.Blast import NCBIStandalone
        # blast_parser = NCBIStandalone.BlastParser()
        # blast_record = blast_parser.parse(result_handle)
```

This will parse the BLAST report into a Blast Record class (either a Blast or a PSIBlast record, depending on what you are parsing) so that you can extract the information from it. In our case, let's just print out a quick summary of all of the alignments greater than some threshold value.

```
In [ ]: E_VALUE_THRESH = 0.04
        for alignment in blast_record.alignments:
            for hsp in alignment.hsps:
                if hsp.expect < E_VALUE_THRESH:
                    print('****Alignment****')
                    print('sequence:', alignment.title)
                    print('length:', alignment.length)
                    print('e value:', hsp.expect)
                    print(hsp.query[0:75] + '...')
                    print(hsp.match[0:75] + '...')
                    print(hsp.sbjct[0:75] + '...')
```

If you also read the section [sec:parsing-blast] on parsing BLAST XML output, you'll notice that the above code is identical to what is found in that section. Once you parse something into a record class you can deal with it independent of the format of the original BLAST info you were parsing. Pretty snazzy!

Sure, parsing one record is great, but I've got a BLAST file with tons of records – how can I parse them all? Well, fear not, the answer lies in the very next section.

8.7.2 Parsing a plain-text BLAST file full of BLAST runs

We can do this using the blast iterator. To set up an iterator, we first set up a parser, to parse our blast reports in Blast Record objects:

```
In [ ]: # from Bio.Blast import NCBIStandalone
        # blast_parser = NCBIStandalone.BlastParser()
```

Then we will assume we have a handle to a bunch of blast records, which we'll call `result_handle`. Getting a handle is described in full detail above in the blast parsing sections.

Now that we've got a parser and a handle, we are ready to set up the iterator with the following command:

```
In [ ]: # blast_iterator = NCBIStandalone.Iterator(result_handle, blast_parser)
```

The second option, the parser, is optional. If we don't supply a parser, then the iterator will just return the raw BLAST reports one at a time.

Now that we've got an iterator, we start retrieving blast records (generated by our parser) using `next()`:

```
In [ ]: # blast_record = next(blast_iterator)
```

Each call to next will return a new record that we can deal with. Now we can iterate through these records and generate our old favorite, a nice little blast report:

```
In [ ]: # for blast_record in blast_iterator:
        #     E_VALUE_THRESH = 0.04
        #     for alignment in blast_record.alignments:
        #         for hsp in alignment.hsps:
        #             if hsp.expect < E_VALUE_THRESH:
        #                 print('****Alignment****')
        #                 print('sequence:', alignment.title)
        #                 print('length:', alignment.length)
        #                 print('e value:', hsp.expect)
        #                 if len(hsp.query) > 75:
        #                     dots = '...'
        #             else:
        #                 dots = ''
        #                 print(hsp.query[0:75] + dots)
        #                 print(hsp.match[0:75] + dots)
        #                 print(hsp.sbjct[0:75] + dots)
```

The iterator allows you to deal with huge blast records without any memory problems, since things are read in one at a time. I have parsed tremendously huge files without any problems using this.

8.7.3 Finding a bad record somewhere in a huge plain-text BLAST file

One really ugly problem that happens to me is that I'll be parsing a huge blast file for a while, and the parser will bomb out with a ValueError. This is a serious problem, since you can't tell if the ValueError is due to a parser problem, or a problem with the BLAST. To make it even worse, you have no idea where the parse failed, so you can't just ignore the error, since this could be ignoring an important data point.

We used to have to make a little script to get around this problem, but the `Bio.Blast` module now includes a `BlastErrorParser` which really helps make this easier. The `BlastErrorParser` works very similar to the regular `BlastParser`, but it adds an extra layer of work by catching ValueErrors that are generated by the parser, and attempting to diagnose the errors.

Let's take a look at using this parser – first we define the file we are going to parse and the file to write the problem reports to:

```
In [ ]: # import os
        # blast_file = os.path.join(os.getcwd(), "blast_out", "big_blast.out")
        # error_file = os.path.join(os.getcwd(), "blast_out", "big_blast.problems")
```

Now we want to get a `BlastErrorParser`:

```
In [ ]: # from Bio.Blast import NCBIStandalone
        # error_handle = open(error_file, "w")
        # blast_error_parser = NCBIStandalone.BlastErrorParser(error_handle)
```

Notice that the parser take an optional argument of a handle. If a handle is passed, then the parser will write any blast records which generate a ValueError to this handle. Otherwise, these records will not be recorded.

Now we can use the `BlastErrorParser` just like a regular blast parser. Specifically, we might want to make an iterator that goes through our blast records one at a time and parses them with the error parser:

```
In [ ]: # result_handle = open(blast_file)
        # iterator = NCBIStandalone.Iterator(result_handle, blast_error_parser)
```

We can read these records one a time, but now we can catch and deal with errors that are due to problems with Blast (and not with the parser itself):

```
In [ ]: # try:
        # next_record = next(iterator)
        # except NCBIStandalone.LowQualityBlastError as info:
        # print("LowQualityBlastError detected in id %s" % info[1])
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 9

BLAST and other sequence search tools (*experimental code*)

WARNING: This chapter of the Tutorial describes an *experimental* module in Biopython. It is being included in Biopython and documented here in the tutorial in a pre-final state to allow a period of feedback and refinement before we declare it stable. Until then the details will probably change, and any scripts using the current `Bio.SearchIO` would need to be updated. Please keep this in mind! For stable code working with NCBI BLAST, please continue to use Bio.Blast described in the preceding Chapter [chapter:blast].

Biological sequence identification is an integral part of bioinformatics. Several tools are available for this, each with their own algorithms and approaches, such as BLAST (arguably the most popular), FASTA, HMMER, and many more. In general, these tools usually use your sequence to search a database of potential matches. With the growing number of known sequences (hence the growing number of potential matches), interpreting the results becomes increasingly hard as there could be hundreds or even thousands of potential matches. Naturally, manual interpretation of these searches' results is out of the question. Moreover, you often need to work with several sequence search tools, each with its own statistics, conventions, and output format. Imagine how daunting it would be when you need to work with multiple sequences using multiple search tools.

We know this too well ourselves, which is why we created the `Bio.SearchIO` submodule in Biopython. `Bio.SearchIO` allows you to extract information from your search results in a convenient way, while also dealing with the different standards and conventions used by different search tools. The name `SearchIO` is a homage to BioPerl's module of the same name.

In this chapter, we'll go through the main features of `Bio.SearchIO` to show what it can do for you. We'll use two popular search tools along the way: BLAST and BLAT. They are used merely for illustrative purposes, and you should be able to adapt the workflow to any other search tools supported by `Bio.SearchIO` in a breeze. You're very welcome to follow along with the search output files we'll be using. The BLAST output file can be downloaded here, and the BLAT output file here. Both output files were generated using this sequence:

```
>mystery_seq
CCCTCTACAGGGAAGCGCTTTCTGTTGTCTGAAAGAAAAGAAAGTGCTTCCTTTTAGAGGG
```

The BLAST result is an XML file generated using `blastn` against the NCBI `refseq_rna` database. For BLAT, the sequence database was the February 2009 `hg19` human genome draft and the output format is PSL.

We'll start from an introduction to the `Bio.SearchIO` object model. The model is the representation of your search results, thus it is core to `Bio.SearchIO` itself. After that, we'll check out the main functions in `Bio.SearchIO` that you may often use.

Now that we're all set, let's go to the first step: introducing the core object model.

9.1 The SearchIO object model

Despite the wildly differing output styles among many sequence search tools, it turns out that their underlying concept is similar:

- The output file may contain results from one or more search queries.
- In each search query, you will see one or more hits from the given search database.
- In each database hit, you will see one or more regions containing the actual sequence alignment between your query sequence and the database sequence.
- Some programs like BLAT or Exonerate may further split these regions into several alignment fragments (or blocks in BLAT and possibly exons in exonerate). This is not something you always see, as programs like BLAST and HMMER do not do this.

Realizing this generality, we decided use it as base for creating the `Bio.SearchIO` object model. The object model consists of a nested hierarchy of Python objects, each one representing one concept outlined above. These objects are:

- `QueryResult`, to represent a single search query.
- `Hit`, to represent a single database hit. `Hit` objects are contained within `QueryResult` and in each `QueryResult` there is zero or more `Hit` objects.
- `HSP` (short for high-scoring pair), to represent region(s) of significant alignments between query and hit sequences. `HSP` objects are contained within `Hit` objects and each `Hit` has one or more `HSP` objects.
- `HSPFragment`, to represent a single contiguous alignment between query and hit sequences. `HSPFragment` objects are contained within `HSP` objects. Most sequence search tools like BLAST and HMMER unify `HSP` and `HSPFragment` objects as each `HSP` will only have a single `HSPFragment`. However there are tools like BLAT and Exonerate that produce `HSP` containing multiple `HSPFragment`. Don't worry if this seems a tad confusing now, we'll elaborate more on these two objects later on.

These four objects are the ones you will interact with when you use `Bio.SearchIO`. They are created using one of the main `Bio.SearchIO` methods: `read`, `parse`, `index`, or `index_db`. The details of these methods are provided in later sections. For this section, we'll only be using read and parse. These functions behave similarly to their `Bio.SeqIO` and `Bio.AlignIO` counterparts:

- `read` is used for search output files with a single query and returns a `QueryResult` object
- `parse` is used for search output files with multiple queries and returns a generator that yields `QueryResult` objects

With that settled, let's start probing each `Bio.SearchIO` object, beginning with `QueryResult`.

9.1.1 QueryResult

The QueryResult object represents a single search query and contains zero or more Hit objects. Let's see what it looks like using the BLAST file we have:

```
In [1]: from Bio import SearchIO
        blast_qresult = SearchIO.read('data/my_blast.xml', 'blast-xml')
        print(blast_qresult)
```

```
Program: blastn (2.3.0+)
  Query: gi|8332116|gb|BE037100.1|BE037100 (1111)
         MP14H09 MP Mesembryanthemum crystallinum cDNA 5' similar to cold ac...
```

```
 Target: nt
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0      1  gi|731339628|ref|XM_010682658.1|  PREDICTED: Beta vulga...
            1      1  gi|568824607|ref|XM_006466626.1|  PREDICTED: Citrus sin...
            2      1  gi|568824605|ref|XM_006466625.1|  PREDICTED: Citrus sin...
            3      1  gi|568824603|ref|XM_006466624.1|  PREDICTED: Citrus sin...
            4      1  gi|568824601|ref|XM_006466623.1|  PREDICTED: Citrus sin...
            5      1  gi|568824599|ref|XM_006466622.1|  PREDICTED: Citrus sin...
            6      1  gi|567866318|ref|XM_006425719.1|  Citrus clementina hyp...
            7      1  gi|567866316|ref|XM_006425718.1|  Citrus clementina hyp...
            8      1  gi|567866314|ref|XM_006425717.1|  Citrus clementina hyp...
            9      1  gi|567866312|ref|XM_006425716.1|  Citrus clementina hyp...
           10      1  gi|590704208|ref|XM_007047033.1|  Theobroma cacao Cold-...
           11      1  gi|590704205|ref|XM_007047032.1|  Theobroma cacao Cold-...
           12      1  gi|694428700|ref|XM_009343631.1|  PREDICTED: Pyrus x br...
           13      1  gi|694402986|ref|XM_009378191.1|  PREDICTED: Pyrus x br...
           14      1  gi|743838297|ref|XM_011027373.1|  PREDICTED: Populus eu...
           15      1  gi|743838293|ref|XM_011027372.1|  PREDICTED: Populus eu...
           16      1  gi|595807351|ref|XM_007202530.1|  Prunus persica hypoth...
           17      1  gi|566180892|ref|XM_006380679.1|  Populus trichocarpa c...
           18      1  gi|764593175|ref|XM_004300526.2|  PREDICTED: Fragaria v...
           19      1  gi|731440276|ref|XM_002274845.3|  PREDICTED: Vitis vini...
           20      1  gi|349709091|emb|FQ378501.1|  Vitis vinifera clone SS0A...
           21      1  gi|719997221|ref|XM_010256725.1|  PREDICTED: Nelumbo nu...
           22      1  gi|645272858|ref|XM_008243375.1|  PREDICTED: Prunus mum...
           23      1  gi|645272856|ref|XM_008243374.1|  PREDICTED: Prunus mum...
           24      1  gi|848856318|ref|XM_013000712.1|  PREDICTED: Erythranth...
           25      1  gi|255562758|ref|XM_002522339.1|  Ricinus communis COR4...
           26      1  gi|658006068|ref|XM_008339966.1|  PREDICTED: Malus x do...
           27      1  gi|802641059|ref|XM_012223735.1|  PREDICTED: Jatropha c...
           28      1  gi|802641054|ref|XM_012223734.1|  PREDICTED: Jatropha c...
           29      1  gi|823184330|ref|XM_012633708.1|  PREDICTED: Gossypium ...
           ~~~
           47      1  gi|729314350|ref|XM_010532905.1|  PREDICTED: Tarenaya h...
           48      1  gi|731383573|ref|XM_002284686.2|  PREDICTED: Vitis vini...
           49      1  gi|255762732|gb|GQ370517.1|  Salvia miltiorrhiza cold a...
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SearchIO/__init__.py:211: BiopythonExper
  BiopythonExperimentalWarning)
```

We've just begun to scratch the surface of the object model, but you can see that there's already some useful information. By invoking `print` on the `QueryResult` object, you can see:

- The program name and version (blastn version 2.2.27+)
- The query ID, description, and its sequence length (ID is 42291, description is 'mystery_seq', and it is 61 nucleotides long)
- The target database to search against (refseq_rna)
- A quick overview of the resulting hits. For our query sequence, there are 100 potential hits (numbered 0–99 in the table). For each hit, we can also see how many HSPs it contains, its ID, and a snippet of its description. Notice here that `Bio.SearchIO` truncates the hit table overview, by showing only hits numbered 0–29, and then 97–99.

Now let's check our BLAT results using the same procedure as above:

```
In [2]: blat_qresult = SearchIO.read('data/my_blat.psl', 'blat-psl')
        print(blat_qresult)
```

```
Program: blat (<unknown version>)
  Query: mystery_seq (61)
         <unknown description>
 Target: <unknown target>
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0     17  chr19  <unknown description>
```

You'll immediately notice that there are some differences. Some of these are caused by the way PSL format stores its details, as you'll see. The rest are caused by the genuine program and target database differences between our BLAST and BLAT searches:

- The program name and version. `Bio.SearchIO` knows that the program is BLAT, but in the output file there is no information regarding the program version so it defaults to '<unknown version>'.
- The query ID, description, and its sequence length. Notice here that these details are slightly different from the ones we saw in BLAST. The ID is 'mystery_seq' instead of 42991, there is no known description, but the query length is still 61. This is actually a difference introduced by the file formats themselves. BLAST sometimes creates its own query IDs and uses your original ID as the sequence description.
- The target database is not known, as it is not stated in the BLAT output file.
- And finally, the list of hits we have is completely different. Here, we see that our query sequence only hits the 'chr19' database entry, but in it we see 17 HSP regions. This should not be surprising however, given that we are using a different program, each with its own target database.

All the details you saw when invoking the `print` method can be accessed individually using Python's object attribute access notation (a.k.a. the dot notation). There are also other format-specific attributes that you can access using the same method.

```
In [3]: print("%s %s" % (blast_qresult.program, blast_qresult.version))
```

```
blastn 2.3.0+
```

```
In [4]: print("%s %s" % (blat_qresult.program, blat_qresult.version))
```

```
blat <unknown version>
```

```
In [5]: blast_qresult.param_evalue_threshold    # blast-xml specific
```

```
Out[5]: 10.0
```

For a complete list of accessible attributes, you can check each format-specific documentation. Here are the ones for BLAST and for BLAT.

Having looked at using `print` on `QueryResult` objects, let's drill down deeper. What exactly is a `QueryResult`? In terms of Python objects, `QueryResult` is a hybrid between a list and a dictionary. In other words, it is a container object with all the convenient features of lists and dictionaries.

Like Python lists and dictionaries, `QueryResult` objects are iterable. Each iteration returns a `Hit` object:

```
In [6]: for hit in blast_qresult:
            hit
```

To check how many items (hits) a `QueryResult` has, you can simply invoke Python's `len` method:

```
In [7]: len(blast_qresult)
```

```
Out[7]: 50
```

```
In [8]: len(blat_qresult)
```

```
Out[8]: 1
```

Like Python lists, you can retrieve items (hits) from a `QueryResult` using the slice notation:

```
In [9]: blast_qresult[0]        # retrieves the top hit
```

```
Out[9]: Hit(id='gi|731339628|ref|XM_010682658.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 hs
```

```
In [10]: blast_qresult[-1]       # retrieves the last hit
```

```
Out[10]: Hit(id='gi|255762732|gb|GQ370517.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 hsps)
```

To retrieve multiple hits, you can slice `QueryResult` objects using the slice notation as well. In this case, the slice will return a new `QueryResult` object containing only the sliced hits:

```
In [11]: blast_slice = blast_qresult[:3]     # slices the first three hits
         print(blast_slice)
```

```
Program: blastn (2.3.0+)
  Query: gi|8332116|gb|BE037100.1|BE037100 (1111)
         MP14H09 MP Mesembryanthemum crystallinum cDNA 5' similar to cold ac...
 Target: nt
   Hits: ----  -----  ----------------------------------------------------------
            #  # HSP  ID + description
         ----  -----  ----------------------------------------------------------
            0      1  gi|731339628|ref|XM_010682658.1|  PREDICTED: Beta vulga...
            1      1  gi|568824607|ref|XM_006466626.1|  PREDICTED: Citrus sin...
            2      1  gi|568824605|ref|XM_006466625.1|  PREDICTED: Citrus sin...
```

Like Python dictionaries, you can also retrieve hits using the hit's ID. This is particularly useful if you know a given hit ID exists within a search query results:

```
In [16]: blast_qresult['gi|731339628|ref|XM_010682658.1|']
```

```
Out[16]: Hit(id='gi|731339628|ref|XM_010682658.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 h
```

You can also get a full list of `Hit` objects using `hits` and a full list of `Hit` IDs using `hit_keys`:

```
In [17]: blast_qresult.hits
```

```
Out[17]: [Hit(id='gi|731339628|ref|XM_010682658.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|568824607|ref|XM_006466626.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|568824605|ref|XM_006466625.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|568824603|ref|XM_006466624.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|568824601|ref|XM_006466623.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|568824599|ref|XM_006466622.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|567866318|ref|XM_006425719.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|567866316|ref|XM_006425718.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|567866314|ref|XM_006425717.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|567866312|ref|XM_006425716.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|590704208|ref|XM_007047033.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|590704205|ref|XM_007047032.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|694428700|ref|XM_009343631.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|694402986|ref|XM_009378191.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|743838297|ref|XM_011027373.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|743838293|ref|XM_011027372.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|595807351|ref|XM_007202530.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|566180892|ref|XM_006380679.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|764593175|ref|XM_004300526.2|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|731440276|ref|XM_002274845.3|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|349709091|emb|FQ378501.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 hsps
          Hit(id='gi|719997221|ref|XM_010256725.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|645272858|ref|XM_008243375.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|645272856|ref|XM_008243374.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|848856318|ref|XM_013000712.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|255562758|ref|XM_002522339.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|658006068|ref|XM_008339966.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|802641059|ref|XM_012223735.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|802641054|ref|XM_012223734.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|823184330|ref|XM_012633708.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|697116717|ref|XM_009613983.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|698554552|ref|XM_009771948.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|565355903|ref|XM_006344753.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|659079297|ref|XM_008441961.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|659079295|ref|XM_008441960.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|828326432|ref|XM_004509143.2|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|729300104|ref|XM_010559563.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|922336892|ref|XM_013590982.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|955375440|ref|XM_003548859.3|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|778718146|ref|XM_011659511.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|778718144|ref|XM_004141977.2|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|955306249|ref|XM_003519866.3|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|747102191|ref|XM_011100949.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|723677217|ref|XM_004233368.2|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|703126872|ref|XM_010105390.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|658037046|ref|XM_008355863.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|225311746|dbj|AK326681.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 hsps
          Hit(id='gi|729314350|ref|XM_010532905.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|731383573|ref|XM_002284686.2|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1
          Hit(id='gi|255762732|gb|GQ370517.1|', query_id='gi|8332116|gb|BE037100.1|BE037100', 1 hsps)
```

```
In [18]: blast_qresult.hit_keys
```

```
Out[18]: ['gi|731339628|ref|XM_010682658.1|',
          'gi|568824607|ref|XM_006466626.1|',
          'gi|568824605|ref|XM_006466625.1|',
          'gi|568824603|ref|XM_006466624.1|',
          'gi|568824601|ref|XM_006466623.1|',
          'gi|568824599|ref|XM_006466622.1|',
          'gi|567866318|ref|XM_006425719.1|',
          'gi|567866316|ref|XM_006425718.1|',
          'gi|567866314|ref|XM_006425717.1|',
          'gi|567866312|ref|XM_006425716.1|',
          'gi|590704208|ref|XM_007047033.1|',
          'gi|590704205|ref|XM_007047032.1|',
          'gi|694428700|ref|XM_009343631.1|',
          'gi|694402986|ref|XM_009378191.1|',
          'gi|743838297|ref|XM_011027373.1|',
          'gi|743838293|ref|XM_011027372.1|',
          'gi|595807351|ref|XM_007202530.1|',
          'gi|566180892|ref|XM_006380679.1|',
          'gi|764593175|ref|XM_004300526.2|',
          'gi|731440276|ref|XM_002274845.3|',
          'gi|349709091|emb|FQ378501.1|',
          'gi|719997221|ref|XM_010256725.1|',
          'gi|645272858|ref|XM_008243375.1|',
          'gi|645272856|ref|XM_008243374.1|',
          'gi|848856318|ref|XM_013000712.1|',
          'gi|255562758|ref|XM_002522339.1|',
          'gi|658006068|ref|XM_008339966.1|',
          'gi|802641059|ref|XM_012223735.1|',
          'gi|802641054|ref|XM_012223734.1|',
          'gi|823184330|ref|XM_012633708.1|',
          'gi|697116717|ref|XM_009613983.1|',
          'gi|698554552|ref|XM_009771948.1|',
          'gi|565355903|ref|XM_006344753.1|',
          'gi|659079297|ref|XM_008441961.1|',
          'gi|659079295|ref|XM_008441960.1|',
          'gi|828326432|ref|XM_004509143.2|',
          'gi|729300104|ref|XM_010559563.1|',
          'gi|922336892|ref|XM_013590982.1|',
          'gi|955375440|ref|XM_003548859.3|',
          'gi|778718146|ref|XM_011659511.1|',
          'gi|778718144|ref|XM_004141977.2|',
          'gi|955306249|ref|XM_003519866.3|',
          'gi|747102191|ref|XM_011100949.1|',
          'gi|723677217|ref|XM_004233368.2|',
          'gi|703126872|ref|XM_010105390.1|',
          'gi|658037046|ref|XM_008355863.1|',
          'gi|225311746|dbj|AK326681.1|',
          'gi|729314350|ref|XM_010532905.1|',
          'gi|731383573|ref|XM_002284686.2|',
          'gi|255762732|gb|GQ370517.1|']
```

What if you just want to check whether a particular hit is present in the query results? You can do a simple Python membership test using the `in` keyword:

```
In [19]: 'gi|262205317|ref|NR_030195.1|' in blast_qresult
```

```
Out[19]: False
```

```
In [21]: 'gi|731339628|ref|XM_010682658.1|' in blast_qresult
```

```
Out[21]: True
```

Sometimes, knowing whether a hit is present is not enough; you also want to know the rank of the hit. Here, the `index` method comes to the rescue:

```
In [23]: blast_qresult.index('gi|595807351|ref|XM_007202530.1|')
```

```
Out[23]: 16
```

Remember that we're using Python's indexing style here, which is zero-based. This means our hit above is ranked at no. 23, not 22.

Also, note that the hit rank you see here is based on the native hit ordering present in the original search output file. Different search tools may order these hits based on different criteria.

If the native hit ordering doesn't suit your taste, you can use the `sort` method of the `QueryResult` object. It is very similar to Python's `list.sort` method, with the addition of an option to create a new sorted `QueryResult` object or not.

Here is an example of using `QueryResult.sort` to sort the hits based on each hit's full sequence length. For this particular sort, we'll set the `in_place` flag to `False` so that sorting will return a new `QueryResult` object and leave our initial object unsorted. We'll also set the `reverse` flag to `True` so that we sort in descending order.

```
In [24]: for hit in blast_qresult[:5]:   # id and sequence length of the first five hits
             print("%s %i" % (hit.id, hit.seq_len))
```

```
gi|731339628|ref|XM_010682658.1| 847
gi|568824607|ref|XM_006466626.1| 878
gi|568824605|ref|XM_006466625.1| 911
gi|568824603|ref|XM_006466624.1| 894
gi|568824601|ref|XM_006466623.1| 922
```

```
In [25]: sort_key = lambda hit: hit.seq_len
         sorted_qresult = blast_qresult.sort(key=sort_key, reverse=True, in_place=False)
         for hit in sorted_qresult[:5]:
             print("%s %i" % (hit.id, hit.seq_len))
```

```
gi|955306249|ref|XM_003519866.3| 1388
gi|659079295|ref|XM_008441960.1| 1245
gi|567866316|ref|XM_006425718.1| 1202
```

```
gi|566180892|ref|XM_006380679.1| 1182
gi|565355903|ref|XM_006344753.1| 1153
```

The advantage of having the `in_place` flag here is that we're preserving the native ordering, so we may use it again later. You should note that this is not the default behavior of `QueryResult.sort`, however, which is why we needed to set the `in_place` flag to `True` explicitly.

At this point, you've known enough about `QueryResult` objects to make it work for you. But before we go on to the next object in the `Bio.SearchIO` model, let's take a look at two more sets of methods that could make it even easier to work with `QueryResult` objects: the `filter` and `map` methods.

If you're familiar with Python's list comprehensions, generator expressions or the built in `filter` and `map` functions, you'll know how useful they are for working with list-like objects (if you're not, check them out!). You can use these built in methods to manipulate `QueryResult` objects, but you'll end up with regular Python lists and lose the ability to do more interesting manipulations.

That's why, `QueryResult` objects provide its own flavor of `filter` and `map` methods. Analogous to `filter`, there are `hit_filter` and `hsp_filter` methods. As their name implies, these methods filter its `QueryResult` object either on its `Hit` objects or `HSP` objects. Similarly, analogous to `map`, `QueryResult` objects also provide the `hit_map` and `hsp_map` methods. These methods apply a given function to all hits or HSPs in a `QueryResult` object, respectively.

Let's see these methods in action, beginning with `hit_filter`. This method accepts a callback function that checks whether a given `Hit` object passes the condition you set or not. In other words, the function must accept as its argument a single `Hit` object and returns `True` or `False`.

Here is an example of using `hit_filter` to filter out `Hit` objects that only have one HSP:

```
In [26]: filter_func = lambda hit: len(hit.hsps) > 1     # the callback function
         len(blast_qresult)      # no. of hits before filtering
```

```
Out[26]: 50
```

```
In [27]: filtered_qresult = blast_qresult.hit_filter(filter_func)
         len(filtered_qresult)   # no. of hits after filtering
```

```
Out[27]: 0
```

```
In [28]: for hit in filtered_qresult[:5]:    # quick check for the hit lengths
             print("%s %i" % (hit.id, len(hit.hsps)))
```

`hsp_filter` works the same as `hit_filter`, only instead of looking at the `Hit` objects, it performs filtering on the `HSP` objects in each hits.

As for the `map` methods, they too accept a callback function as their arguments. However, instead of returning `True` or `False`, the callback function must return the modified `Hit` or `HSP` object (depending on whether you're using `hit_map` or `hsp_map`).

Let's see an example where we're using `hit_map` to rename the hit IDs:

```
In [29]: def map_func(hit):
             hit.id = hit.id.split('|')[3]   # renames 'gi|301171322|ref|NR_035857.1|' to 'NR_035857
             return hit

         mapped_qresult = blast_qresult.hit_map(map_func)
         for hit in mapped_qresult[:5]:
             print(hit.id)
```

```
XM_010682658.1
XM_006466626.1
XM_006466625.1
XM_006466624.1
XM_006466623.1
```

Again, `hsp_map` works the same as `hit_map`, but on `HSP` objects instead of `Hit` objects.

9.1.2 Hit

`Hit` objects represent all query results from a single database entry. They are the second-level container in the `Bio.SearchIO` object hierarchy. You've seen that they are contained by `QueryResult` objects, but they themselves contain `HSP` objects.

Let's see what they look like, beginning with our BLAST search:

```
In [30]: from Bio import SearchIO
         blast_qresult = SearchIO.read('data/my_blast.xml', 'blast-xml')
         blast_hit = blast_qresult[3]    # fourth hit from the query result
         print(blast_hit)
```

```
Query: gi|8332116|gb|BE037100.1|BE037100
       MP14H09 MP Mesembryanthemum crystallinum cDNA 5' similar to cold accl...
  Hit: gi|568824603|ref|XM_006466624.1| (894)
       PREDICTED: Citrus sinensis cold-regulated 413 plasma membrane protein...
 HSPs: ----  --------  ---------  ------  ---------------  ---------------------
          #   E-value  Bit score    Span      Query range              Hit range
       ----  --------  ---------  ------  ---------------  ---------------------
          0   6.9e-99     372.78     596         [63:655]              [108:697]
```

You see that we've got the essentials covered here:

- The query ID and description is present. A hit is always tied to a query, so we want to keep track of the originating query as well. These values can be accessed from a hit using the `query_id` and `query_description` attributes.
- We also have the unique hit ID, description, and full sequence lengths. They can be accessed using `id`, `description`, and `seq_len`, respectively.
- Finally, there's a table containing quick information about the HSPs this hit contains. In each row, we've got the important HSP details listed: the HSP index, its e-value, its bit score, its span (the alignment length including gaps), its query coordinates, and its hit coordinates.

Now let's contrast this with the BLAT search. Remember that in the BLAT search we had one hit with 17 HSPs.

```
In [31]: blat_qresult = SearchIO.read('data/my_blat.psl', 'blat-psl')
         blat_hit = blat_qresult[0]      # the only hit
         print(blat_hit)
```

```
Query: mystery_seq
       <unknown description>
  Hit: chr19 (59128983)
       <unknown description>
 HSPs: ----  --------  ---------  ------  ---------------  ---------------------
          #   E-value  Bit score    Span      Query range              Hit range
       ----  --------  ---------  ------  ---------------  ---------------------
          0         ?          ?       ?           [0:61]    [54204480:54204541]
          1         ?          ?       ?           [0:61]    [54233104:54264463]
          2         ?          ?       ?           [0:61]    [54254477:54260071]
          3         ?          ?       ?           [1:61]    [54210720:54210780]
          4         ?          ?       ?           [0:60]    [54198476:54198536]
          5         ?          ?       ?           [0:61]    [54265610:54265671]
          6         ?          ?       ?           [0:61]    [54238143:54240175]
          7         ?          ?       ?           [0:60]    [54189735:54189795]
          8         ?          ?       ?           [0:61]    [54185425:54185486]
          9         ?          ?       ?           [0:60]    [54197657:54197717]
         10         ?          ?       ?           [0:61]    [54255662:54255723]
```

```
           11          ?           ?        ?             [0:61]    [54201651:54201712]
           12          ?           ?        ?             [8:60]    [54206009:54206061]
           13          ?           ?        ?            [10:61]    [54178987:54179038]
           14          ?           ?        ?             [8:61]    [54212018:54212071]
           15          ?           ?        ?             [8:51]    [54234278:54234321]
           16          ?           ?        ?             [8:61]    [54238143:54238196]
```

Here, we've got a similar level of detail as with the BLAST hit we saw earlier. There are some differences worth explaining, though:

- The e-value and bit score column values. As BLAT HSPs do not have e-values and bit scores, the display defaults to '?'.
- What about the span column? The span values is meant to display the complete alignment length, which consists of all residues and any gaps that may be present. The PSL format do not have this information readily available and `Bio.SearchIO` does not attempt to try guess what it is, so we get a '?' similar to the e-value and bit score columns.

In terms of Python objects, `Hit` behaves almost the same as Python lists, but contain `HSP` objects exclusively. If you're familiar with lists, you should encounter no difficulties working with the `Hit` object.

Just like Python lists, `Hit` objects are iterable, and each iteration returns one `HSP` object it contains:

```
In [32]: for hsp in blast_hit:
             hsp
```

You can invoke `len` on a `Hit` to see how many `HSP` objects it has:

```
In [33]: len(blast_hit)

Out[33]: 1

In [34]: len(blat_hit)

Out[34]: 17
```

You can use the slice notation on `Hit` objects, whether to retrieve single `HSP` or multiple `HSP` objects. Like `QueryResult`, if you slice for multiple `HSP`, a new `Hit` object will be returned containing only the sliced `HSP` objects:

```
In [35]: blat_hit[0]                 # retrieve single items

Out[35]: HSP(hit_id='chr19', query_id='mystery_seq', 1 fragments)

In [36]: sliced_hit = blat_hit[4:9]  # retrieve multiple items
         len(sliced_hit)

Out[36]: 5

In [37]: print(sliced_hit)

Query: mystery_seq
       <unknown description>
  Hit: chr19 (59128983)
       <unknown description>
 HSPs: ----  --------  ---------  ------  ---------------  ---------------------
          #   E-value  Bit score    Span      Query range              Hit range
       ----  --------  ---------  ------  ---------------  ---------------------
          0         ?          ?       ?           [0:60]    [54198476:54198536]
          1         ?          ?       ?           [0:61]    [54265610:54265671]
          2         ?          ?       ?           [0:61]    [54238143:54240175]
          3         ?          ?       ?           [0:60]    [54189735:54189795]
          4         ?          ?       ?           [0:61]    [54185425:54185486]
```

You can also sort the `HSP` inside a `Hit`, using the exact same arguments like the sort method you saw in the `QueryResult` object.

Finally, there are also the `filter` and `map` methods you can use on `Hit` objects. Unlike in the `QueryResult` object, `Hit` objects only have one variant of `filter` (`Hit.filter`) and one variant of `map` (`Hit.map`). Both of `Hit.filter` and `Hit.map` work on the `HSP` objects a `Hit` has.

9.1.3 HSP

`HSP` (high-scoring pair) represents region(s) in the hit sequence that contains significant alignment(s) to the query sequence. It contains the actual match between your query sequence and a database entry. As this match is determined by the sequence search tool's algorithms, the `HSP` object contains the bulk of the statistics computed by the search tool. This also makes the distinction between `HSP` objects from different search tools more apparent compared to the differences you've seen in `QueryResult` or `Hit` objects.

Let's see some examples from our BLAST and BLAT searches. We'll look at the BLAST HSP first:

```
In [38]: from Bio import SearchIO
         blast_qresult = SearchIO.read('data/my_blast.xml', 'blast-xml')
         blast_hsp = blast_qresult[0][0]    # first hit, first hsp
         print(blast_hsp)
```

```
      Query: gi|8332116|gb|BE037100.1|BE037100 MP14H09 MP Mesembryanthemum cr...
        Hit: gi|731339628|ref|XM_010682658.1| PREDICTED: Beta vulgaris subsp...
Query range: [86:679] (1)
  Hit range: [80:677] (1)
Quick stats: evalue 1.2e-108; bitscore 405.24
  Fragments: 1 (597 columns)
     Query - TTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGATTCCGATATCAATGA~~~TGTAG
             |||||||||||||||| ||| ||||  |||||||| |||| ||||  ||||| |||||~~~|||||
       Hit - TTGGCCATGAAAACTGAGCAAATGGCGTTGGCTAATTTGATAGATTATGATATGAATGA~~~TGTAG
```

Just like `QueryResult` and `Hit`, invoking `print` on an `HSP` shows its general details:

- There are the query and hit IDs and descriptions. We need these to identify our `HSP`.
- We've also got the matching range of the query and hit sequences. The slice notation we're using here is an indication that the range is displayed using Python's indexing style (zero-based, half open). The number inside the parenthesis denotes the strand. In this case, both sequences have the plus strand.
- Some quick statistics are available: the e-value and bitscore.
- There is information about the HSP fragments. Ignore this for now; it will be explained later on.
- And finally, we have the query and hit sequence alignment itself.

These details can be accessed on their own using the dot notation, just like in `QueryResult` and `Hit`:

```
In [39]: blast_hsp.query_range
```

```
Out[39]: (86, 679)
```

```
In [40]: blast_hsp.evalue
```

```
Out[40]: 1.1684e-108
```

They're not the only attributes available, though. `HSP` objects come with a default set of properties that makes it easy to probe their various details. Here are some examples:

```
In [41]: blast_hsp.hit_start         # start coordinate of the hit sequence
```

```
Out[41]: 80
```

```
In [42]: def map_func(hit):
             hit.id = hit.id.split('|')[3]   # renames 'gi|301171322|ref|NR_035857.1|' to 'NR_035857
             return hitblast_hsp.query_span        # how many residues in the query sequence
```

```
In [43]: blast_hsp.aln_span          # how long the alignment is
```

```
Out[43]: 597
```

Check out the `HSP` documentation for a full list of these predefined properties.

Furthermore, each sequence search tool usually computes its own statistics / details for its `HSP` objects. For example, an XML BLAST search also outputs the number of gaps and identical residues. These attributes can be accessed like so:

```
In [44]: blast_hsp.gap_num       # number of gaps
```

```
Out[44]: 4
```

```
In [45]: blast_hsp.ident_num     # number of identical residues
```

```
Out[45]: 449
```

These details are format-specific; they may not be present in other formats. To see which details are available for a given sequence search tool, you should check the format's documentation in `Bio.SearchIO`. Alternatively, you may also use `.__dict__.keys()` for a quick list of what's available:

```
In [46]: blast_hsp.__dict__.keys()
```

```
Out[46]: dict_keys(['_items', 'pos_num', 'gap_num', 'bitscore', 'bitscore_raw', 'ident_num', 'evalue
```

Finally, you may have noticed that the `query` and `hit` attributes of our HSP are not just regular strings:

```
In [47]: blast_hsp.query
```

```
Out[47]: SeqRecord(seq=Seq('TTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGATTCCGATATC...TAG', DNAAlphabe
```

```
In [48]: blast_hsp.hit
```

```
Out[48]: SeqRecord(seq=Seq('TTGGCCATGAAAACTGAGCAAATGGCGTTGGCTAATTTGATAGATTATGATATG...TAG', DNAAlphabe
```

They are `SeqRecord` objects you saw earlier in Section [chapter:SeqRecord]! This means that you can do all sorts of interesting things you can do with `SeqRecord` objects on `HSP.query` and/or `HSP.hit`.

It should not surprise you now that the `HSP` object has an `alignment` property which is a `MultipleSeqAlignment` object:

```
In [49]: print(blast_hsp.aln)
```

```
DNAAlphabet() alignment with 2 rows and 597 columns
TTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGA...TAG gi|8332116|gb|BE037100.1|BE037100
TTGGCCATGAAAACTGAGCAAATGGCGTTGGCTAATTTGATAGA...TAG gi|731339628|ref|XM_010682658.1|
```

Having probed the BLAST HSP, let's now take a look at HSPs from our BLAT results for a different kind of HSP. As usual, we'll begin by invoking `print` on it:

```
In [50]: blat_qresult = SearchIO.read('data/my_blat.psl', 'blat-psl')
         blat_hsp = blat_qresult[0][0]       # first hit, first hsp
         print(blat_hsp)
```

```
      Query: mystery_seq <unknown description>
        Hit: chr19 <unknown description>
Query range: [0:61] (1)
  Hit range: [54204480:54204541] (1)
Quick stats: evalue ?; bitscore ?
  Fragments: 1 (? columns)
```

Some of the outputs you may have already guessed. We have the query and hit IDs and descriptions and the sequence coordinates. Values for evalue and bitscore is '?' as BLAT HSPs do not have these attributes. But The biggest difference here is that you don't see any sequence alignments displayed. If you look closer, PSL formats themselves do not have any hit or query sequences, so `Bio.SearchIO` won't create any sequence or alignment objects. What happens if you try to access `HSP.query`, `HSP.hit`, or `HSP.aln`? You'll get the default values for these attributes, which is `None`:

```
In [51]: blat_hsp.hit is None

Out[51]: True

In [52]: blat_hsp.query is None

Out[52]: True

In [53]: blat_hsp.aln is None

Out[53]: True
```

This does not affect other attributes, though. For example, you can still access the length of the query or hit alignment. Despite not displaying any attributes, the PSL format still have this information so `Bio.SearchIO` can extract them:

```
In [54]: blat_hsp.query_span     # length of query match

Out[54]: 61

In [55]: blat_hsp.hit_span       # length of hit match

Out[55]: 61
```

Other format-specific attributes are still present as well:

```
In [56]: blat_hsp.score          # PSL score

Out[56]: 61

In [57]: blat_hsp.mismatch_num   # the mismatch column

Out[57]: 0
```

So far so good? Things get more interesting when you look at another 'variant' of HSP present in our BLAT results. You might recall that in BLAT searches, sometimes we get our results separated into 'blocks'. These blocks are essentially alignment fragments that may have some intervening sequence between them.

Let's take a look at a BLAT HSP that contains multiple blocks to see how `Bio.SearchIO` deals with this:

```
In [58]: blat_hsp2 = blat_qresult[0][1]      # first hit, second hsp
         print(blat_hsp2)

      Query: mystery_seq <unknown description>
        Hit: chr19 <unknown description>
Query range: [0:61] (1)
  Hit range: [54233104:54264463] (1)
Quick stats: evalue ?; bitscore ?
  Fragments: ---  --------------  ----------------------  ----------------------
               #            Span             Query range               Hit range
             ---  --------------  ----------------------  ----------------------
               0               ?                  [0:18]     [54233104:54233122]
               1               ?                 [18:61]     [54264420:54264463]
```

What's happening here? We still some essential details covered: the IDs and descriptions, the coordinates, and the quick statistics are similar to what you've seen before. But the fragments detail is all different. Instead of showing 'Fragments: 1', we now have a table with two data rows.

This is how `Bio.SearchIO` deals with HSPs having multiple fragments. As mentioned before, an HSP alignment may be separated by intervening sequences into fragments. The intervening sequences are not part of the query-hit match, so they should not be considered part of query nor hit sequence. However, they do affect how we deal with sequence coordinates, so we can't ignore them.

Take a look at the hit coordinate of the HSP above. In the `Hit range:` field, we see that the coordinate is `[54233104:54264463]`. But looking at the table rows, we see that not the entire region spanned by this coordinate matches our query. Specifically, the intervening region spans from `54233122` to `54264420`.

Why then, is the query coordinates seem to be contiguous, you ask? This is perfectly fine. In this case it means that the query match is contiguous (no intervening regions), while the hit match is not.

All these attributes are accessible from the HSP directly, by the way:

```
In [59]: blat_hsp2.hit_range         # hit start and end coordinates of the entire HSP

Out[59]: (54233104, 54264463)

In [60]: blat_hsp2.hit_range_all     # hit start and end coordinates of each fragment

Out[60]: [(54233104, 54233122), (54264420, 54264463)]

In [61]: blat_hsp2.hit_span          # hit span of the entire HSP

Out[61]: 31359

In [62]: blat_hsp2.hit_span_all      # hit span of each fragment

Out[62]: [18, 43]

In [63]: blat_hsp2.hit_inter_ranges  # start and end coordinates of intervening regions in the hit se

Out[63]: [(54233122, 54264420)]

In [64]: blat_hsp2.hit_inter_spans   # span of intervening regions in the hit sequence

Out[64]: [31298]
```

Most of these attributes are not readily available from the PSL file we have, but `Bio.SearchIO` calculates them for you on the fly when you parse the PSL file. All it needs are the start and end coordinates of each fragment.

What about the `query`, `hit`, and `aln` attributes? If the HSP has multiple fragments, you won't be able to use these attributes as they only fetch single `SeqRecord` or `MultipleSeqAlignment` objects. However, you can use their `*_all` counterparts: `query_all`, `hit_all`, and `aln_all`. These properties will return a list containing `SeqRecord` or `MultipleSeqAlignment` objects from each of the HSP fragment. There are other attributes that behave similarly, i.e. they only work for HSPs with one fragment. Check out the `HSP` documentation for a full list.

Finally, to check whether you have multiple fragments or not, you can use the `is_fragmented` property like so:

```
In [65]: blat_hsp2.is_fragmented     # BLAT HSP with 2 fragments

Out[65]: True

In [66]: blat_hsp.is_fragmented      # BLAT HSP from earlier, with one fragment

Out[66]: False
```

Before we move on, you should also know that we can use the slice notation on `HSP` objects, just like `QueryResult` or `Hit` objects. When you use this notation, you'll get an `HSPFragment` object in return, the last component of the object model.

9.1.4 HSPFragment

`HSPFragment` represents a single, contiguous match between the query and hit sequences. You could consider it the core of the object model and search result, since it is the presence of these fragments that determine whether your search have results or not.

In most cases, you don't have to deal with `HSPFragment` objects directly since not that many sequence search tools fragment their HSPs. When you do have to deal with them, what you should remember is that `HSPFragment` objects were written with to be as compact as possible. In most cases, they only contain attributes directly related to sequences: strands, reading frames, alphabets, coordinates, the sequences themselves, and their IDs and descriptions.

These attributes are readily shown when you invoke `print` on an `HSPFragment`. Here's an example, taken from our BLAST search:

```
In [67]: from Bio import SearchIO
         blast_qresult = SearchIO.read('data/my_blast.xml', 'blast-xml')
         blast_frag = blast_qresult[0][0][0]    # first hit, first hsp, first fragment
         print(blast_frag)
```

```
      Query: gi|8332116|gb|BE037100.1|BE037100 MP14H09 MP Mesembryanthemum cr...
        Hit: gi|731339628|ref|XM_010682658.1| PREDICTED: Beta vulgaris subsp...
Query range: [86:679] (1)
  Hit range: [80:677] (1)
  Fragments: 1 (597 columns)
     Query - TTGGCCATGAAAACTGATCAATTGGCCGTGGCTAATATGATCGATTCCGATATCAATGA~~~TGTAG
             |||||||||||||||| ||| ||||  |||||||| |||| ||||  ||||| |||||~~~|||||
       Hit - TTGGCCATGAAAACTGAGCAAATGGCGTTGGCTAATTTGATAGATTATGATATGAATGA~~~TGTAG
```

At this level, the BLAT fragment looks quite similar to the BLAST fragment, save for the query and hit sequences which are not present:

```
In [68]: blat_qresult = SearchIO.read('data/my_blat.psl', 'blat-psl')
         blat_frag = blat_qresult[0][0][0]    # first hit, first hsp, first fragment
         print(blat_frag)
```

```
      Query: mystery_seq <unknown description>
        Hit: chr19 <unknown description>
Query range: [0:61] (1)
  Hit range: [54204480:54204541] (1)
  Fragments: 1 (? columns)
```

In all cases, these attributes are accessible using our favorite dot notation. Some examples:

```
In [69]: blast_frag.query_start      # query start coordinate
```

```
Out[69]: 86
```

```
In [70]: blast_frag.hit_strand       # hit sequence strand
```

```
Out[70]: 1
```

```
In [71]: blast_frag.hit              # hit sequence, as a SeqRecord object
```

```
Out[71]: SeqRecord(seq=Seq('TTGGCCATGAAAACTGAGCAAATGGCGTTGGCTAATTTGATAGATTATGATATG...TAG', DNAAlphabe
```

9.2 A note about standards and conventions

Before we move on to the main functions, there is something you ought to know about the standards `Bio.SearchIO` uses. If you've worked with multiple sequence search tools, you might have had to deal with the many different ways each program deals with things like sequence coordinates. It might not have been a pleasant experience as these search tools usually have their own standards. For example, one tools might use one-based coordinates, while the other uses zero-based coordinates. Or, one program might reverse the start and end coordinates if the strand is minus, while others don't. In short, these often creates unnecessary mess must be dealt with.

We realize this problem ourselves and we intend to address it in `Bio.SearchIO`. After all, one of the goals of `Bio.SearchIO` is to create a common, easy to use interface to deal with various search output files. This means creating standards that extend beyond the object model you just saw.

Now, you might complain, "Not another standard!". Well, eventually we have to choose one convention or the other, so this is necessary. Plus, we're not creating something entirely new here; just adopting a standard we think is best for a Python programmer (it is Biopython, after all).

There are three implicit standards that you can expect when working with `Bio.SearchIO`:

- The first one pertains to sequence coordinates. In `Bio.SearchIO`, all sequence coordinates follows Python's coordinate style: zero-based and half open. For example, if in a BLAST XML output file the start and end coordinates of an HSP are 10 and 28, they would become 9 and 28 in `Bio.SearchIO`. The start coordinate becomes 9 because Python indices start from zero, while the end coordinate remains 28 as Python slices omit the last item in an interval.

- The second is on sequence coordinate orders. In `Bio.SearchIO`, start coordinates are always less than or equal to end coordinates. This isn't always the case with all sequence search tools, as some of them have larger start coordinates when the sequence strand is minus.
- The last one is on strand and reading frame values. For strands, there are only four valid choices: `1` (plus strand), `-1` (minus strand), `0` (protein sequences), and `None` (no strand). For reading frames, the valid choices are integers from `-3` to `3` and `None`.

Note that these standards only exist in `Bio.SearchIO` objects. If you write `Bio.SearchIO` objects into an output format, `Bio.SearchIO` will use the format's standard for the output. It does not force its standard over to your output file.

9.3 Reading search output files

There are two functions you can use for reading search output files into `Bio.SearchIO` objects: `read` and `parse`. They're essentially similar to `read` and `parse` functions in other submodules like `Bio.SeqIO` or `Bio.AlignIO`. In both cases, you need to supply the search output file name and the file format name, both as Python strings. You can check the documentation for a list of format names `Bio.SearchIO` recognizes.

`Bio.SearchIO.read` is used for reading search output files with only one query and returns a `QueryResult` object. You've seen `read` used in our previous examples. What you haven't seen is that `read` may also accept additional keyword arguments, depending on the file format.

Here are some examples. In the first one, we use `read` just like previously to read a BLAST tabular output file. In the second one, we use a keyword argument to modify so it parses the BLAST tabular variant with comments in it:

```
In [73]: from Bio import SearchIO
         qresult = SearchIO.read('data/tab_2226_tblastn_003.txt', 'blast-tab')
         qresult
```

```
Out[73]: QueryResult(id='gi|16080617|ref|NP_391444.1|', 3 hits)
```

```
In [75]: qresult2 = SearchIO.read('data/tab_2226_tblastn_007.txt', 'blast-tab', comments=True)
         qresult2
```

```
Out[75]: QueryResult(id='gi|16080617|ref|NP_391444.1|', 3 hits)
```

These keyword arguments differs among file formats. Check the format documentation to see if it has keyword arguments that modifies its parser's behavior.

As for the `Bio.SearchIO.parse`, it is used for reading search output files with any number of queries. The function returns a generator object that yields a `QueryResult` object in each iteration. Like `Bio.SearchIO.read`, it also accepts format-specific keyword arguments:

```
In [77]: from Bio import SearchIO
         qresults = SearchIO.parse('data/tab_2226_tblastn_001.txt', 'blast-tab')
         for qresult in qresults:
             print(qresult.id)
```

```
gi|16080617|ref|NP_391444.1|
gi|11464971:4-101
```

```
In [78]: qresults2 = SearchIO.parse('data/tab_2226_tblastn_005.txt', 'blast-tab', comments=True)
         for qresult in qresults2:
             print(qresult.id)
```

```
random_s00
gi|16080617|ref|NP_391444.1|
gi|11464971:4-101
```

9.4 Dealing with large search output files with indexing

Sometimes, you're handed a search output file containing hundreds or thousands of queries that you need to parse. You can of course use `Bio.SearchIO.parse` for this file, but that would be grossly inefficient if you need to access only a few of the queries. This is because `parse` will parse all queries it sees before it fetches your query of interest.

In this case, the ideal choice would be to index the file using `Bio.SearchIO.index` or `Bio.SearchIO.index_db`. If the names sound familiar, it's because you've seen them before in Section [sec:SeqIO-index]. These functions also behave similarly to their `Bio.SeqIO` counterparts, with the addition of format-specific keyword arguments.

Here are some examples. You can use `index` with just the filename and format name:

```
In [79]: from Bio import SearchIO
         idx = SearchIO.index('data/tab_2226_tblastn_001.txt', 'blast-tab')
         sorted(idx.keys())

Out[79]: ['gi|11464971:4-101', 'gi|16080617|ref|NP_391444.1|']

In [80]: idx['gi|16080617|ref|NP_391444.1|']

Out[80]: QueryResult(id='gi|16080617|ref|NP_391444.1|', 3 hits)

In [81]: idx.close()
```

Or also with the format-specific keyword argument:

```
In [82]: idx = SearchIO.index('data/tab_2226_tblastn_005.txt', 'blast-tab', comments=True)
         sorted(idx.keys())

Out[82]: ['gi|11464971:4-101', 'gi|16080617|ref|NP_391444.1|', 'random_s00']

In [83]: idx['gi|16080617|ref|NP_391444.1|']

Out[83]: QueryResult(id='gi|16080617|ref|NP_391444.1|', 3 hits)

In [84]: idx.close()
```

Or with the `key_function` argument, as in `Bio.SeqIO`:

```
In [85]: key_function = lambda id: id.upper()    # capitalizes the keys
         idx = SearchIO.index('data/tab_2226_tblastn_001.txt', 'blast-tab', key_function=key_functio
         sorted(idx.keys())

Out[85]: ['GI|11464971:4-101', 'GI|16080617|REF|NP_391444.1|']

In [86]: idx['GI|16080617|REF|NP_391444.1|']

Out[86]: QueryResult(id='gi|16080617|ref|NP_391444.1|', 3 hits)

In [87]: idx.close()
```

`Bio.SearchIO.index_db` works like as `index`, only it writes the query offsets into an SQLite database file.

9.5 Writing and converting search output files

It is occasionally useful to be able to manipulate search results from an output file and write it again to a new file. `Bio.SearchIO` provides a `write` function that lets you do exactly this. It takes as its arguments an iterable returning

`QueryResult` objects, the output filename to write to, the format name to write to, and optionally some format-specific keyword arguments. It returns a four-item tuple, which denotes the number or `QueryResult`, `Hit`, `HSP`, and `HSPFragment` objects that were written.

```
In [88]: from Bio import SearchIO
         qresults = SearchIO.parse('data/mirna.xml', 'blast-xml')     # read XML file
         SearchIO.write(qresults, 'results.tab', 'blast-tab')    # write to tabular file
```

```
Out[88]: (3, 239, 277, 277)
```

You should note different file formats require different attributes of the `QueryResult`, `Hit`, `HSP` and `HSPFragment` objects. If these attributes are not present, writing won't work. In other words, you can't always write to the output format that you want. For example, if you read a BLAST XML file, you wouldn't be able to write the results to a PSL file as PSL files require attributes not calculated by BLAST (e.g. the number of repeat matches). You can always set these attributes manually, if you really want to write to PSL, though.

Like `read`, `parse`, `index`, and `index_db`, `write` also accepts format-specific keyword arguments. Check out the documentation for a complete list of formats `Bio.SearchIO` can write to and their arguments.

Finally, `Bio.SearchIO` also provides a `convert` function, which is simply a shortcut for `Bio.SearchIO.parse` and `Bio.SearchIO.write`. Using the convert function, our example above would be:

```
In [89]: from Bio import SearchIO
         SearchIO.convert('data/mirna.xml', 'blast-xml', 'results.tab', 'blast-tab')
```

```
Out[89]: (3, 239, 277, 277)
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 10

Accessing NCBI's Entrez databases

Entrez (http://www.ncbi.nlm.nih.gov/Entrez) is a data retrieval system that provides users access to NCBI's databases such as PubMed, GenBank, GEO, and many others. You can access Entrez from a web browser to manually enter queries, or you can use Biopython's `Bio.Entrez` module for programmatic access to Entrez. The latter allows you for example to search PubMed or download GenBank records from within a Python script.

The `Bio.Entrez` module makes use of the Entrez Programming Utilities (also known as EUtils), consisting of eight tools that are described in detail on NCBI's page at http://www.ncbi.nlm.nih.gov/entrez/utils/. Each of these tools corresponds to one Python function in the `Bio.Entrez` module, as described in the sections below. This module makes sure that the correct URL is used for the queries, and that not more than one request is made every three seconds, as required by NCBI.

The output returned by the Entrez Programming Utilities is typically in XML format. To parse such output, you have several options:

1. Use `Bio.Entrez`'s parser to parse the XML output into a Python object;
2. Use the DOM (Document Object Model) parser in Python's standard library;
3. Use the SAX (Simple API for XML) parser in Python's standard library;
4. Read the XML output as raw text, and parse it by string searching and manipulation.

For the DOM and SAX parsers, see the Python documentation. The parser in `Bio.Entrez` is discussed below.

NCBI uses DTD (Document Type Definition) files to describe the structure of the information contained in XML files. Most of the DTD files used by NCBI are included in the Biopython distribution. The `Bio.Entrez` parser makes use of the DTD files when parsing an XML file returned by NCBI Entrez.

Occasionally, you may find that the DTD file associated with a specific XML file is missing in the Biopython distribution. In particular, this may happen when NCBI updates its DTD files. If this happens, `Entrez.read` will show a warning message with the name and URL of the missing DTD file. The parser will proceed to access the missing DTD file through the internet, allowing the parsing of the XML file to continue. However, the parser is much faster if the DTD file is available locally. For this purpose, please download the DTD file from the URL in the warning message and place it in the directory `...site-packages/Bio/Entrez/DTDs`, containing the other DTD files. If you don't have write access to this directory, you can also place the DTD file in `~/.biopython/Bio/Entrez/DTDs`, where ~ represents your home directory. Since this directory is read before the directory `...site-packages/Bio/Entrez/DTDs`, you can also put newer versions of DTD files there if the ones in `...site-packages/Bio/Entrez/DTDs` become outdated. Alternatively, if you installed Biopython from source, you can add the DTD file to the source code's `Bio/Entrez/DTDs` directory, and reinstall Biopython. This will install the new DTD file in the correct location together with the other DTD files.

The Entrez Programming Utilities can also generate output in other formats, such as the Fasta or GenBank file formats for sequence databases, or the MedLine format for the literature database, discussed in Section *Specialized parsers*.

10.1 Entrez Guidelines

Before using Biopython to access the NCBI's online resources (via `Bio.Entrez` or some of the other modules), please read the NCBI's Entrez User Requirements. If the NCBI finds you are abusing their systems, they can and will ban your access!

To paraphrase:

- For any series of more than 100 requests, do this at weekends or outside USA peak times. This is up to you to obey.
- Use the http://eutils.ncbi.nlm.nih.gov address, not the standard NCBI Web address. Biopython uses this web address.
- Make no more than three requests every seconds (relaxed from at most one request every three seconds in early 2009). This is automatically enforced by Biopython.
- Use the optional email parameter so the NCBI can contact you if there is a problem. You can either explicitly set this as a parameter with each call to Entrez (e.g. include email="A.N.Other@example.com" in the argument list), or you can set a global email address:

```
In [1]: from Bio import Entrez
        Entrez.email = "A.N.Other@example.com"
```

Bio.Entrez will then use this email address with each call to Entrez. The example.com address is a reserved domain name specifically for documentation (RFC 2606). Please DO NOT use a random email – it's better not to give an email at all. The email parameter will be mandatory from June 1, 2010. In case of excessive usage, NCBI will attempt to contact a user at the e-mail address provided prior to blocking access to the E-utilities.

If you are using Biopython within some larger software suite, use the tool parameter to specify this. You can either explicitly set the tool name as a parameter with each call to Entrez (e.g. include tool="MyLocalScript" in the argument list), or you can set a global tool name:

```
In [2]: from Bio import Entrez
        Entrez.tool = "MyLocalScript"
```

```
The tool parameter will default to Biopython.
```

- For large queries, the NCBI also recommend using their session history feature (the WebEnv session cookie string, see Section *History and WebEnv*). This is only slightly more complicated.

In conclusion, be sensible with your usage levels. If you plan to download lots of data, consider other options. For example, if you want easy access to all the human genes, consider fetching each chromosome by FTP as a GenBank file, and importing these into your own BioSQL database (see Section [sec:BioSQL]).

10.2 EInfo: Obtaining information about the Entrez databases

- einfo source

EInfo provides field index term counts, last update, and available links for each of NCBI's databases. In addition, you can use EInfo to obtain a list of all database names accessible through the Entrez utilities. The variable `result` now contains a list of databases in XML format:

```
In [3]: from Bio import Entrez
        Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
        handle = Entrez.einfo()
        result = handle.read()
        print(result)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eInfoResult PUBLIC "-//NLM//DTD einfo 20130322//EN" "http://eutils.ncbi.nlm.nih.gov/eutils/
<eInfoResult>
<DbList>

        <DbName>pubmed</DbName>
        <DbName>protein</DbName>
        <DbName>nuccore</DbName>
        <DbName>nucleotide</DbName>
        <DbName>nucgss</DbName>
        <DbName>nucest</DbName>
        <DbName>structure</DbName>
        <DbName>genome</DbName>
        <DbName>annotinfo</DbName>
        <DbName>assembly</DbName>
        <DbName>bioproject</DbName>
        <DbName>biosample</DbName>
        <DbName>blastdbinfo</DbName>
        <DbName>books</DbName>
        <DbName>cdd</DbName>
        <DbName>clinvar</DbName>
        <DbName>clone</DbName>
        <DbName>gap</DbName>
        <DbName>gapplus</DbName>
        <DbName>grasp</DbName>
        <DbName>dbvar</DbName>
        <DbName>epigenomics</DbName>
        <DbName>gene</DbName>
        <DbName>gds</DbName>
        <DbName>geoprofiles</DbName>
        <DbName>homologene</DbName>
        <DbName>medgen</DbName>
        <DbName>mesh</DbName>
        <DbName>ncbisearch</DbName>
        <DbName>nlmcatalog</DbName>
        <DbName>omim</DbName>
        <DbName>orgtrack</DbName>
        <DbName>pmc</DbName>
        <DbName>popset</DbName>
        <DbName>probe</DbName>
        <DbName>proteinclusters</DbName>
        <DbName>pcassay</DbName>
        <DbName>biosystems</DbName>
        <DbName>pccompound</DbName>
        <DbName>pcsubstance</DbName>
        <DbName>pubmedhealth</DbName>
        <DbName>seqannot</DbName>
        <DbName>snp</DbName>
        <DbName>sra</DbName>
        <DbName>taxonomy</DbName>
        <DbName>unigene</DbName>
        <DbName>gencoll</DbName>
        <DbName>gtr</DbName>
</DbList>

</eInfoResult>
```

Since this is a fairly simple XML file, we could extract the information it contains simply by string searching. Using `Bio.Entrez`'s parser instead, we can directly parse this XML file into a Python object:

```
In [4]: from Bio import Entrez
        handle = Entrez.einfo()
        record = Entrez.read(handle)
```

Now `record` is a dictionary with exactly one key:

```
In [5]: record.keys()
```

```
Out[5]: dict_keys(['DbList'])
```

The values stored in this key is the list of database names shown in the XML above:

```
In [6]: record["DbList"]
```

```
Out[6]: ['pubmed', 'protein', 'nuccore', 'nucleotide', 'nucgss', 'nucest', 'structure', 'genome', 'ar
```

For each of these databases, we can use EInfo again to obtain more information:

```
In [7]: from Bio import Entrez
        handle = Entrez.einfo(db="pubmed")
        record = Entrez.read(handle)
        record["DbInfo"]["Description"]
```

```
Out[7]: 'PubMed bibliographic record'
```

```
In [8]: record['DbInfo'].keys()
```

```
Out[8]: dict_keys(['LastUpdate', 'Count', 'DbName', 'Description', 'MenuName', 'FieldList', 'DbBuild
```

```
In [9]: handle = Entrez.einfo(db="pubmed")
        record = Entrez.read(handle)
        record["DbInfo"]["Description"]
```

```
Out[9]: 'PubMed bibliographic record'
```

```
In [10]: record["DbInfo"]["Count"]
```

```
Out[10]: '25641704'
```

```
In [11]: record["DbInfo"]["LastUpdate"]
```

```
Out[11]: '2016/01/12 18:56'
```

Try `record["DbInfo"].keys()` for other information stored in this record. One of the most useful is a list of possible search fields for use with ESearch:

```
In [12]: for field in record["DbInfo"]["FieldList"]:
             print("%(Name)s, %(FullName)s, %(Description)s" % field)
```

```
ALL, All Fields, All terms from all searchable fields
UID, UID, Unique number assigned to publication
FILT, Filter, Limits the records
TITL, Title, Words in title of publication
WORD, Text Word, Free text associated with publication
MESH, MeSH Terms, Medical Subject Headings assigned to publication
MAJR, MeSH Major Topic, MeSH terms of major importance to publication
AUTH, Author, Author(s) of publication
JOUR, Journal, Journal abbreviation of publication
AFFL, Affiliation, Author's institutional affiliation and address
ECNO, EC/RN Number, EC number for enzyme or CAS registry number
SUBS, Supplementary Concept, CAS chemical name or MEDLINE Substance Name
PDAT, Date - Publication, Date of publication
EDAT, Date - Entrez, Date publication first accessible through Entrez
VOL, Volume, Volume number of publication
PAGE, Pagination, Page number(s) of publication
PTYP, Publication Type, Type of publication (e.g., review)
LANG, Language, Language of publication
ISS, Issue, Issue number of publication
SUBH, MeSH Subheading, Additional specificity for MeSH term
SI, Secondary Source ID, Cross-reference from publication to other databases
MHDA, Date - MeSH, Date publication was indexed with MeSH terms
TIAB, Title/Abstract, Free text associated with Abstract/Title
OTRM, Other Term, Other terms associated with publication
INVR, Investigator, Investigator
COLN, Author - Corporate, Corporate Author of publication
CNTY, Place of Publication, Country of publication
PAPX, Pharmacological Action, MeSH pharmacological action pre-explosions
GRNT, Grant Number, NIH Grant Numbers
MDAT, Date - Modification, Date of last modification
CDAT, Date - Completion, Date of completion
PID, Publisher ID, Publisher ID
FAUT, Author - First, First Author of publication
FULL, Author - Full, Full Author Name(s) of publication
FINV, Investigator - Full, Full name of investigator
TT, Transliterated Title, Words in transliterated title of publication
LAUT, Author - Last, Last Author of publication
PPDT, Print Publication Date, Date of print publication
EPDT, Electronic Publication Date, Date of Electronic publication
LID, Location ID, ELocation ID
CRDT, Date - Create, Date publication first accessible through Entrez
BOOK, Book, ID of the book that contains the document
ED, Editor, Section's Editor
ISBN, ISBN, ISBN
PUBN, Publisher, Publisher's name
AUCL, Author Cluster ID, Author Cluster ID
EID, Extended PMID, Extended PMID
DSO, DSO, Additional text from the summary
AUID, Author - Identifier, Author Identifier
PS, Subject - Personal Name, Personal Name as Subject
```

That's a long list, but indirectly this tells you that for the PubMed database, you can do things like `Jones[AUTH]` to search the author field, or `Sanger[AFFL]` to restrict to authors at the Sanger Centre. This can be very handy - especially if you are not so familiar with a particular database.

10.3 ESearch: Searching the Entrez databases

To search any of these databases, we use `Bio.Entrez.esearch()`. For example, let's search in PubMed for publications related to Biopython:

```
In [13]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.esearch(db="pubmed", term="biopython")
         record = Entrez.read(handle)
         record["IdList"]
```

```
Out[13]: ['24929426', '24497503', '24267035', '24194598', '23842806', '23157543', '22909249', '22399
```

```
In [14]: record
```

```
Out[14]: {'TranslationStack': [{'Explode': 'N', 'Term': 'biopython[All Fields]', 'Field': 'All Fields
```

In this output, you see seven PubMed IDs (including 19304878 which is the PMID for the Biopython application), which can be retrieved by EFetch (see section *EFetch: Downloading full records from Entrez*).

You can also use ESearch to search GenBank. Here we'll do a quick search for the *matK* gene in *Cypripedioideae* orchids (see Section [sec:entrez-einfo] about EInfo for one way to find out which fields you can search in each Entrez database):

```
In [15]: handle = Entrez.esearch(db="nucleotide", term="Cypripedioideae[Orgn] AND matK[Gene]")
         record = Entrez.read(handle)
         record["Count"]
```

```
Out[15]: '334'
```

```
In [16]: record["IdList"]
```

```
Out[16]: ['844174433', '937957673', '694174838', '944541375', '575524123', '575524121', '575524119',
```

Each of the IDs (126789333, 37222967, 37222966, . . .) is a GenBank identifier. See section *EFetch: Downloading full records from Entrez* for information on how to actually download these GenBank records.

Note that instead of a species name like `Cypripedioideae[Orgn]`, you can restrict the search using an NCBI taxon identifier, here this would be `txid158330[Orgn]`. This isn't currently documented on the ESearch help page - the NCBI explained this in reply to an email query. You can often deduce the search term formatting by playing with the Entrez web interface. For example, including `complete[prop]` in a genome search restricts to just completed genomes.

As a final example, let's get a list of computational journal titles:

```
In [17]: # nlmcatalog
         # handle = Entrez.esearch(db="nlmcatalog", term="computational")
         # record = Entrez.read(handle)
         # record["Count"]
         handle = Entrez.esearch(db="nlmcatalog", term="biopython[Journal]", RetMax='20')
         record = Entrez.read(handle)
         print("{} computational Journals found".format(record["Count"]))
         print("The first 20 are\n{}".format(record['IdList']))
```

```
0 computational Journals found
The first 20 are
[]
```

Again, we could use EFetch to obtain more information for each of these journal IDs.

ESearch has many useful options — see the ESearch help page for more information.

10.4 EPost: Uploading a list of identifiers

EPost uploads a list of UIs for use in subsequent search strategies; see the EPost help page for more information. It is available from Biopython through the `Bio.Entrez.epost()` function.

To give an example of when this is useful, suppose you have a long list of IDs you want to download using EFetch (maybe sequences, maybe citations – anything). When you make a request with EFetch your list of IDs, the database etc, are all turned into a long URL sent to the server. If your list of IDs is long, this URL gets long, and long URLs can break (e.g. some proxies don't cope well).

Instead, you can break this up into two steps, first uploading the list of IDs using EPost (this uses an "HTML post" internally, rather than an "HTML get", getting round the long URL problem). With the history support, you can then refer to this long list of IDs, and download the associated data with EFetch.

Let's look at a simple example to see how EPost works – uploading some PubMed identifiers:

```
In [18]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         id_list = ["19304878", "18606172", "16403221", "16377612", "14871861", "14630660"]
         print(Entrez.epost("pubmed", id=",".join(id_list)).read())
```

```
<?xml version="1.0"?>
<!DOCTYPE ePostResult PUBLIC "-//NLM//DTD ePostResult, 11 May 2002//EN" "http://www.ncbi.nlm.nih.gov/
<ePostResult>
        <QueryKey>1</QueryKey>
        <WebEnv>NCID_1_242547791_130.14.22.215_9001_1452651583_567658845_0MetA0_S_MegaStore_F_1</WebE
</ePostResult>
```

The returned XML includes two important strings, `QueryKey` and `WebEnv` which together define your history session. You would extract these values for use with another Entrez call such as EFetch:

```
In [19]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         id_list = ["19304878", "18606172", "16403221", "16377612", "14871861", "14630660"]
         search_results = Entrez.read(Entrez.epost("pubmed", id=",".join(id_list)))
         webenv = search_results["WebEnv"]
         query_key = search_results["QueryKey"]
```

Section *History and WebEnv* shows how to use the history feature.

10.5 ESummary: Retrieving summaries from primary IDs

ESummary retrieves document summaries from a list of primary IDs (see the ESummary help page for more information). In Biopython, ESummary is available as `Bio.Entrez.esummary()`. Using the search result above, we can for example find out more about the journal with ID 30367:

```
In [20]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.esummary(db="nlmcatalog", term="[journal]", id="101660833")
         record = Entrez.read(handle)
         info = record[0]['TitleMainList'][0]
         print("Journal info\nid: {}\nTitle: {}".format(record[0]["Id"], info["Title"]))
```

```
Journal info
id: 101660833
Title: IEEE transactions on computational imaging.
```

10.6 EFetch: Downloading full records from Entrez

EFetch is what you use when you want to retrieve a full record from Entrez. This covers several possible databases, as described on the main EFetch Help page.

For most of their databases, the NCBI support several different file formats. Requesting a specific file format from Entrez using `Bio.Entrez.efetch()` requires specifying the `rettype` and/or `retmode` optional arguments. The different combinations are described for each database type on the pages linked to on NCBI efetch webpage (e.g. literature, sequences and taxonomy).

One common usage is downloading sequences in the FASTA or GenBank/GenPept plain text formats (which can then be parsed with `Bio.SeqIO`, see Sections [sec:SeqIO_GenBank_Online] and *EFetch: Downloading full records from Entrez*). From the *Cypripedioideae* example above, we can download GenBank record 186972394 using `Bio.Entrez.efetch`:

```
In [21]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.efetch(db="nucleotide", id="186972394", rettype="gb", retmode="text")
         print(handle.read())
```

```
LOCUS       EU490707                1302 bp    DNA     linear   PLN 15-JAN-2009
DEFINITION  Selenipedium aequinoctiale maturase K (matK) gene, partial cds;
            chloroplast.
ACCESSION   EU490707
VERSION     EU490707.1  GI:186972394
KEYWORDS    .
SOURCE      chloroplast Selenipedium aequinoctiale
  ORGANISM  Selenipedium aequinoctiale
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; Liliopsida; Asparagales; Orchidaceae;
            Cypripedioideae; Selenipedium.
REFERENCE   1  (bases 1 to 1302)
  AUTHORS   Neubig,K.M., Whitten,W.M., Carlsward,B.S., Blanco,M.A., Endara,L.,
            Williams,N.H. and Moore,M.
  TITLE     Phylogenetic utility of ycf1 in orchids: a plastid gene more
            variable than matK
  JOURNAL   Plant Syst. Evol. 277 (1-2), 75-84 (2009)
REFERENCE   2  (bases 1 to 1302)
  AUTHORS   Neubig,K.M., Whitten,W.M., Carlsward,B.S., Blanco,M.A.,
            Endara,C.L., Williams,N.H. and Moore,M.J.
  TITLE     Direct Submission
  JOURNAL   Submitted (14-FEB-2008) Department of Botany, University of
            Florida, 220 Bartram Hall, Gainesville, FL 32611-8526, USA
FEATURES             Location/Qualifiers
     source          1..1302
                     /organism="Selenipedium aequinoctiale"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /specimen_voucher="FLAS:Blanco 2475"
                     /db_xref="taxon:256374"
     gene            <1..>1302
                     /gene="matK"
     CDS             <1..>1302
                     /gene="matK"
                     /codon_start=1
                     /transl_table=11
                     /product="maturase K"
                     /protein_id="ACC99456.1"
                     /db_xref="GI:186972395"
                     /translation="IFYEPVEIFGYDNKSSLVLVKRLITRMYQQNFLISSVNDSNQKG
                     FWGHKHFFSSHFSSQMVSEGFGVILEIPFSSQLVSSLEEKKIPKYQNLRSIHSIFPFL
                     EDKFLHLNYVSDLLIPHPIHLEILVQILQCRIKDVPSLHLLRLLFHEYHNLNSLITSK
                     KFIYAFSKRKKRFLWLLYNSYVYECEYLFQFLRKQSSYLRSTSSGVFLERTHLYVKIE
                     HLLVVCCNSFQRILCFLKDPFMHYVRYQGKAILASKGTLILMKKWKFHLVNFWQSYFH
                     FWSQPYRIHIKQLSNYSFSFLGYFSSVLENHLVVRNQMLENSFIINLLTKKFDTIAPV
                     ISLIGSLSKAQFCTVLGHPISKPIWTDFSDSDILDRFCRICRNLCRYHSGSSKKQVLY
                     RIKYILRLSCARTLARKHKSTVRTFMRRLGSGLLEEFFMEEE"
ORIGIN
        1 atttttttacg aacctgtgga aatttttggt tatgacaata aatctagttt agtacttgtg
       61 aaacgtttaa ttactcgaat gtatcaacag aattttttga tttcttcggt taatgattct
      121 aaccaaaaag gattttgggg gcacaagcat ttttttttctt ctcatttttc ttctcaaatg
      181 gtatcagaag gttttggagt cattctggaa attccattct cgtcgcaatt agtatcttct
      241 cttgaagaaa aaaaaatacc aaaatatcag aatttacgat ctattcattc aatatttccc
      301 tttttagaag acaaattttt acatttgaat tatgtgtcag atctactaat accccatccc
      361 atccatctgg aaatcttggt tcaaatcctt caatgccgga tcaaggatgt tccttctttg
      421 catttattgc gattgctttt ccacgaatat cataatttga atagtctcat tacttcaaag
      481 aaattcattt acgccttttc aaaaagaaag aaaagattcc tttggttact atataattct
      541 tatgtatatg aatgcgaata tctattccag tttcttcgta aacagtcttc ttatttacga
      601 tcaacatctt ctggagtctt tcttgagcga acacatttat atgtaaaaat agaacatctt
      661 ctagtagtgt gttgtaattc ttttcagagg atcctatgct ttctcaagga tcctttcatg
      721 cattatgttc gatatcaagg aaaagcaatt ctggcttcaa agggaactct tattctgatg
      781 aagaaatgga aatttcatct tgtgaatttt tggcaatctt attttcactt ttggtctcaa
      841 ccgtatagga ttcatataaa gcaattatcc aactattcct tctcttttct ggggtatttt
      901 tcaagtgtac tagaaaatca tttggtagta agaaatcaaa tgctagagaa ttcatttata
      961 ataaatcttc tgactaagaa attcgatacc atagccccag ttatttctct tattggatca
     1021 ttgtcgaaag ctcaattttg tactgtattg ggtcatccta ttagtaaacc gatctggacc
     1081 gatttctcgg attctgatat tcttgatcga ttttgccgga tatgtagaaa tctttgtcgt
     1141 tatcacagcg gatcctcaaa aaaacaggtt ttgtatcgta taaaatatat acttcgactt
     1201 tcgtgtgcta gaactttggc acggaaacat aaaagtacag tacgcacttt tatgcgaaga
     1261 ttaggttcgg gattattaga agaattcttt atggaagaag aa
//
```

The arguments `rettype="gb"` and `retmode="text"` let us download this record in the GenBank format.

Note that until Easter 2009, the Entrez EFetch API let you use "genbank" as the return type, however the NCBI now insist on using the official return types of "gb" or "gbwithparts" (or "gp" for proteins) as described on online. Also not that until Feb 2012, the Entrez EFetch API would default to returning plain text files, but now defaults to XML.

Alternatively, you could for example use `rettype="fasta"` to get the Fasta-format; see the EFetch Sequences Help page for other options. Remember – the available formats depend on which database you are downloading from - see the main EFetch Help page.

If you fetch the record in one of the formats accepted by `Bio.SeqIO` (see Chapter [chapter:Bio.SeqIO]), you could directly parse it into a `SeqRecord`:

```
In [22]: from Bio import Entrez, SeqIO
         handle = Entrez.efetch(db="nucleotide", id="186972394", rettype="gb", retmode="text")
         record = SeqIO.read(handle, "genbank")
         handle.close()
         print(record)
```

```
ID: EU490707.1
Name: EU490707
Description: Selenipedium aequinoctiale maturase K (matK) gene, partial cds; chloroplast.
Number of features: 3
/gi=186972394
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta', 'Tracheophyta', 'Spermatophyt
/date=15-JAN-2009
```

```
/references=[Reference(title='Phylogenetic utility of ycf1 in orchids: a plastid gene more variable t
/organism=Selenipedium aequinoctiale
/sequence_version=1
/accessions=['EU490707']
/data_file_division=PLN
/source=chloroplast Selenipedium aequinoctiale
/keywords=['']
Seq('ATTTTTTACGAACCTGTGGAAATTTTTGGTTATGACAATAAATCTAGTTTAGTA...GAA', IUPACAmbiguousDNA())
```

Note that a more typical use would be to save the sequence data to a local file, and *then* parse it with `Bio.SeqIO`. This can save you having to re-download the same file repeatedly while working on your script, and places less load on the NCBI's servers. For example:

```
In [23]: import os
         from Bio import SeqIO
         from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         filename = "gi_186972394.gbk"
         if not os.path.isfile(filename):
             # Downloading...
             with Entrez.efetch(db="nucleotide",id="186972394",rettype="gb", retmode="text") as net_h
                 with open(filename, "w") as out_handle:
                     out_handle.write(net_handle.read())
                 print("Saved")

         print("Parsing...")
         record = SeqIO.read(filename, "genbank")
         print(record)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-23-d6d046a39755> in <module>()
      6 if not os.path.isfile(filename):
      7     # Downloading...
----> 8     with Entrez.efetch(db="nucleotide",id="186972394",rettype="gb", retmode="text") as net_ha
      9         with open(filename, "w") as out_handle:
     10             out_handle.write(net_handle.read())

AttributeError: __exit__
```

To get the output in XML format, which you can parse using the `Bio.Entrez.read()` function, use `retmode="xml"`:

```
In [24]: from Bio import Entrez
         handle = Entrez.efetch(db="nucleotide", id="186972394", retmode="xml")
         record = Entrez.read(handle)
         handle.close()
         record[0]["GBSeq_definition"]
```

```
Out[24]: 'Selenipedium aequinoctiale maturase K (matK) gene, partial cds; chloroplast'
```

```
In [25]: record[0]["GBSeq_source"]
```

```
Out[25]: 'chloroplast Selenipedium aequinoctiale'
```

So, that dealt with sequences. For examples of parsing file formats specific to the other databases (e.g. the `MEDLINE` format used in PubMed), see Section *Specialized parsers*.

If you want to perform a search with `Bio.Entrez.esearch()`, and then download the records with `Bio.Entrez.efetch()`, you should use the WebEnv history feature – see Section *History and WebEnv*.

10.7 ELink: Searching for related items in NCBI Entrez

ELink, available from Biopython as `Bio.Entrez.elink()`, can be used to find related items in the NCBI Entrez databases. For example, you can us this to find nucleotide entries for an entry in the gene database, and other cool stuff.

Let's use ELink to find articles related to the Biopython application note published in *Bioinformatics* in 2009. The PubMed ID of this article is 19304878:

```
In [26]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"
         pmid = "19304878"
         record = Entrez.read(Entrez.elink(dbfrom="pubmed", id=pmid))
         print(record[0].keys())
         print('The record is from the {} database.'.format(record[0]["DbFrom"]))
         print('The IdList is {}.'.format(record[0]["IdList"]))
```

```
dict_keys(['ERROR', 'LinkSetDbHistory', 'IdList', 'LinkSetDb', 'DbFrom'])
The record is from the pubmed database.
The IdList is ['19304878'].
```

The `record` variable consists of a Python list, one for each database in which we searched. Since we specified only one PubMed ID to search for, `record` contains only one item. This item is a dictionary containing information about our search term, as well as all the related items that were found:

The `"LinkSetDb"` key contains the search results, stored as a list consisting of one item for each target database. In our search results, we only find hits in the PubMed database (although sub-divided into categories):

```
In [27]: print('There are {} search results'.format(len(record[0]["LinkSetDb"])))
         for linksetdb in record[0]["LinkSetDb"]:
             print(linksetdb["DbTo"], linksetdb["LinkName"], len(linksetdb["Link"]))
```

```
There are 8 search results
pubmed pubmed_pubmed 224
pubmed pubmed_pubmed_alsoviewed 3
pubmed pubmed_pubmed_citedin 276
pubmed pubmed_pubmed_combined 6
pubmed pubmed_pubmed_five 6
pubmed pubmed_pubmed_refs 17
pubmed pubmed_pubmed_reviews 8
pubmed pubmed_pubmed_reviews_five 6
```

The actual search results are stored as under the `"Link"` key. In total, 110 items were found under standard search. Let's now at the first search result:

```
In [28]: record[0]["LinkSetDb"][0]["Link"][0]
```

```
Out[28]: {'Id': '19304878'}
```

This is the article we searched for, which doesn't help us much, so let's look at the second search result:

```
In [29]: record[0]["LinkSetDb"][0]["Link"][1]
```

```
Out[29]: {'Id': '14630660'}
```

This paper, with PubMed ID 14630660, is about the Biopython PDB parser.

We can use a loop to print out all PubMed IDs:

```
In [30]: for link in record[0]["LinkSetDb"][0]["Link"]:
             print(link["Id"])
```

```
19304878
14630660
```

```
22909249
20739307
23023984
18689808
20733063
19628504
18251993
15572471
20823319
20847218
25273102
12368254
23842806
22399473
22302572
18238804
24064416
22332238
20421198
15723693
15096277
14512356
16377612
15130828
18593718
25236461
20973958
12230038
22368248
23633579
20591906
21352538
21216774
22815363
17237069
17101041
22581176
19181685
16257987
17441614
18227118
17316423
17384428
16539535
22824207
24463182
22877863
14751976
16899494
17237072
23479348
25661541
16569235
20537149
20375454
19106120
20334363
20439314
19460889
```

```
15969769
23456039
15059834
24885957
23292976
14871861
15383216
22276101
19698094
17483505
17291351
15260898
20472540
17586821
16741236
17121776
18442177
23493324
21798964
16371163
19958528
22788675
17586553
20022974
22500002
21949271
16188925
21210984
21210978
16922600
20542914
21737439
22084254
25677125
12401134
22942023
11524374
19336443
17483515
25414366
22556367
16796559
16539540
22396485
22253821
19773334
22595207
23071651
24564380
23846743
22039207
21385461
17537750
15980476
23666736
21715385
19578173
23699471
22908215
```

```
25481009
23355290
22565567
25706687
22813356
25697819
24903418
22942017
24894501
23742983
23956303
24929426
21685053
24078714
24574118
22494792
24951946
23418185
25189778
23220574
23422340
19648141
24068901
24600386
23957210
23242262
22024252
25414364
25378466
23367449
21500218
24961236
25344496
24930138
23987304
22171336
26079347
23348786
24431986
25600941
20375445
20616382
21210977
23435069
23516352
24045775
25494900
24618462
22238272
25217575
15461798
23628689
21984743
25295002
23574738
25328913
22936991
22844241
23396756
```

```
26587054
21362187
23786315
21458441
24258321
25150250
23765606
24795618
22645166
25126069
22954632
25091065
22369160
25505088
22943297
21803787
23633576
24359023
19225577
16766564
19055766
24995036
25591752
19607723
23652425
18328109
20924230
17281649
16729046
25974373
25653001
22479120
24924300
21716279
25024921
24834575
24870127
25942442
25433467
26153621
24086295
25110777
21253560
25132841
25926788
```

Now that was nice, but personally I am often more interested to find out if a paper has been cited. Well, ELink can do that too – at least for journals in Pubmed Central (see Section [sec:elink-citations]).

For help on ELink, see the ELink help page. There is an entire sub-page just for the link names, describing how different databases can be cross referenced.

10.8 EGQuery: Global Query - counts for search terms

EGQuery provides counts for a search term in each of the Entrez databases (i.e. a global query). This is particularly useful to find out how many items your search terms would find in each database without actually performing lots of separate searches with ESearch (see the example in [subsec:entrez_example_genbank] below).

In this example, we use `Bio.Entrez.egquery()` to obtain the counts for “Biopython”:

```
In [31]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.egquery(term="biopython")
         record = Entrez.read(handle)
         for row in record["eGQueryResult"]:
             print(row["DbName"], row["Count"])
```

```
pubmed 21
pmc 560
mesh 0
books 2
pubmedhealth 2
omim 0
ncbisearch 0
nuccore 0
nucgss 0
nucest 0
protein 0
genome 0
structure 0
taxonomy 0
snp 0
dbvar 0
epigenomics 0
gene 0
sra 0
biosystems 0
unigene 0
cdd 0
clone 0
popset 0
geoprofiles 0
gds 16
homologene 0
pccompound 0
pcsubstance 0
pcassay 0
nlmcatalog 0
probe 0
gap 0
proteinclusters 0
bioproject 0
biosample 0
```

See the EGQuery help page for more information.

10.9 ESpell: Obtaining spelling suggestions

ESpell retrieves spelling suggestions. In this example, we use `Bio.Entrez.espell()` to obtain the correct spelling of Biopython:

```
In [32]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.espell(term="biopythooon")
         record = Entrez.read(handle)
         record["Query"]
```

```
Out[32]: 'biopythooon'

In [33]: record["CorrectedQuery"]

Out[33]: 'biopython'
```

See the ESpell help page for more information. The main use of this is for GUI tools to provide automatic suggestions for search terms.

10.10 Parsing huge Entrez XML files

The `Entrez.read` function reads the entire XML file returned by Entrez into a single Python object, which is kept in memory. To parse Entrez XML files too large to fit in memory, you can use the function `Entrez.parse`. This is a generator function that reads records in the XML file one by one. This function is only useful if the XML file reflects a Python list object (in other words, if `Entrez.read` on a computer with infinite memory resources would return a Python list).

For example, you can download the entire Entrez Gene database for a given organism as a file from NCBI's ftp site. These files can be very large. As an example, on September 4, 2009, the file `Homo_sapiens.ags.gz`, containing the Entrez Gene database for human, had a size of 116576 kB. This file, which is in the `ASN` format, can be converted into an XML file using NCBI's `gene2xml` program (see NCBI's ftp site for more information):

```
gene2xml -b T -i Homo_sapiens.ags -o Homo_sapiens.xml
```

The resulting XML file has a size of 6.1 GB. Attempting `Entrez.read` on this file will result in a `MemoryError` on many computers.

The XML file `Homo_sapiens.xml` consists of a list of Entrez gene records, each corresponding to one Entrez gene in human. `Entrez.parse` retrieves these gene records one by one. You can then print out or store the relevant information in each record by iterating over the records. For example, this script iterates over the Entrez gene records and prints out the gene numbers and names for all current genes:

TODO: need alternate example, download option or ...

```python
from Bio import Entrez
handle = open("Homo_sapiens.xml")
records = Entrez.parse(handle)
```

```python
for record in records:
    status = record['Entrezgene_track-info']['Gene-track']['Gene-track_status']
    if status.attributes['value']=='discontinued':
        continue
    geneid = record['Entrezgene_track-info']['Gene-track']['Gene-track_geneid']
    genename = record['Entrezgene_gene']['Gene-ref']['Gene-ref_locus']
    print(geneid, genename)
```

This will print:

```
1 A1BG
2 A2M
3 A2MP
8 AA
9 NAT1
10 NAT2
11 AACP
12 SERPINA3
13 AADAC
```

```
14 AAMP
15 AANAT
16 AARS
17 AAVS1
...
```

10.11 Handling errors

Three things can go wrong when parsing an XML file:

- The file may not be an XML file to begin with;
- The file may end prematurely or otherwise be corrupted;
- The file may be correct XML, but contain items that are not represented in the associated DTD.

The first case occurs if, for example, you try to parse a Fasta file as if it were an XML file:

```
In [34]: from Bio import Entrez
         from Bio.Entrez.Parser import NotXMLError
         handle = open("data/NC_005816.fna", 'rb') # a Fasta file
         try:
             record = Entrez.read(handle)
         except NotXMLError as e:
             print('We are expecting to get NotXMLError')
             print(e)
```

```
We are expecting to get NotXMLError
Failed to parse the XML data (syntax error: line 1, column 0). Please make sure that the input data a
```

Here, the parser didn't find the `<?xml ...` tag with which an XML file is supposed to start, and therefore decides (correctly) that the file is not an XML file.

When your file is in the XML format but is corrupted (for example, by ending prematurely), the parser will raise a CorruptedXMLError. Here is an example of an XML file that ends prematurely:

```
<?xml version="1.0"?>
<!DOCTYPE eInfoResult PUBLIC "-//NLM//DTD eInfoResult, 11 May 2002//EN" "http://www.
↪ncbi.nlm.nih.gov/entrez/query/DTD/eInfo_020511.dtd">
<eInfoResult>
<DbList>
        <DbName>pubmed</DbName>
        <DbName>protein</DbName>
        <DbName>nucleotide</DbName>
        <DbName>nuccore</DbName>
        <DbName>nucgss</DbName>
        <DbName>nucest</DbName>
        <DbName>structure</DbName>
        <DbName>genome</DbName>
        <DbName>books</DbName>
        <DbName>cancerchromosomes</DbName>
        <DbName>cdd</DbName>
```

which will generate the following traceback:

```
---------------------------------------------------------------------------
ExpatError                                Traceback (most recent call last)
/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/Parser.
↪py in read(self, handle)
```

```
    214         try:
--> 215             self.parser.ParseFile(handle)
    216         except expat.ExpatError as e:

ExpatError: syntax error: line 1, column 0

During handling of the above exception, another exception occurred:

NotXMLError                               Traceback (most recent call last)
<ipython-input-63-ac0523d72453> in <module>()
----> 1 Entrez.read(handle)

/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/__init__
↪.py in read(handle, validate)
    419     from .Parser import DataHandler
    420     handler = DataHandler(validate)
--> 421     record = handler.read(handle)
    422     return record
    423

/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/Parser.
↪py in read(self, handle)
    223                 # We have not seen the initial <!xml declaration, so probably
    224                 # the input data is not in XML format.
--> 225                 raise NotXMLError(e)
    226         try:
    227             return self.object

NotXMLError: Failed to parse the XML data (syntax error: line 1, column 0). Please␣
↪make sure that the input data are in XML format.
```

Note that the error message tells you at what point in the XML file the error was detected.

The third type of error occurs if the XML file contains tags that do not have a description in the corresponding DTD file. This is an example of such an XML file:

```
<?xml version="1.0"?>
<!DOCTYPE eInfoResult PUBLIC "-//NLM//DTD eInfoResult, 11 May 2002//EN" "http://www.
↪ncbi.nlm.nih.gov/entrez/query/DTD/eInfo_020511.dtd">
<eInfoResult>
        <DbInfo>
        <DbName>pubmed</DbName>
        <MenuName>PubMed</MenuName>
        <Description>PubMed bibliographic record</Description>
        <Count>20161961</Count>
        <LastUpdate>2010/09/10 04:52</LastUpdate>
        <FieldList>
                <Field>
...
                </Field>
        </FieldList>
        <DocsumList>
                <Docsum>
                        <DsName>PubDate</DsName>
                        <DsType>4</DsType>
                        <DsTypeName>string</DsTypeName>
                </Docsum>
                <Docsum>
```

```
                        <DsName>EPubDate</DsName>
...
        </DbInfo>
</eInfoResult>
```

In this file, for some reason the tag `<DocsumList>` (and several others) are not listed in the DTD file `eInfo_020511.dtd`, which is specified on the second line as the DTD for this XML file. By default, the parser will stop and raise a ValidationError if it cannot find some tag in the DTD:

```
from Bio import Entrez
handle = open("data/einfo3.xml", 'rb')
record = Entrez.read(handle)
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
<ipython-input-65-cfb96ec3d2ca> in <module>()
      1 from Bio import Entrez
      2 handle = open("data/einfo3.xml", 'rb')
----> 3 record = Entrez.read(handle)

/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/__init__
↪.py in read(handle, validate)
    419     from .Parser import DataHandler
    420     handler = DataHandler(validate)
--> 421     record = handler.read(handle)
    422     return record
    423

/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/Parser.
↪py in read(self, handle)
    213             raise IOError("Can't parse a closed handle")
    214         try:
--> 215             self.parser.ParseFile(handle)
    216         except expat.ExpatError as e:
    217             if self.parser.StartElementHandler:

-------src-dir--------/Python-3.5.1/Modules/pyexpat.c in StartElement()

/Users/vincentdavis/anaconda/envs/py35/lib/python3.5/site-packages/Bio/Entrez/Parser.
↪py in startElementHandler(self, name, attrs)
    348             # Element not found in DTD
    349             if self.validating:
--> 350                 raise ValidationError(name)
    351             else:
    352                 # this will not be stored in the record

ValidationError: Failed to find tag 'DocsumList' in the DTD. To skip all tags that␣
↪are not represented in the DTD, please call Bio.Entrez.read or Bio.Entrez.parse␣
↪with validate=False.
```

Optionally, you can instruct the parser to skip such tags instead of raising a ValidationError. This is done by calling `Entrez.read` or `Entrez.parse` with the argument `validate` equal to False:

```
In [35]: from Bio import Entrez
         handle = open("data/einfo3.xml", 'rb')
         record = Entrez.read(handle, validate=False)
```

Of course, the information contained in the XML tags that are not in the DTD are not present in the record returned by Entrez.read.

10.12 Specialized parsers

The Bio.Entrez.read() function can parse most (if not all) XML output returned by Entrez. Entrez typically allows you to retrieve records in other formats, which may have some advantages compared to the XML format in terms of readability (or download size).

To request a specific file format from Entrez using Bio.Entrez.efetch() requires specifying the rettype and/or retmode optional arguments. The different combinations are described for each database type on the NCBI efetch webpage.

One obvious case is you may prefer to download sequences in the FASTA or GenBank/GenPept plain text formats (which can then be parsed with Bio.SeqIO, see Sections [sec:SeqIO_GenBank_Online] and *EFetch: Downloading full records from Entrez*). For the literature databases, Biopython contains a parser for the MEDLINE format used in PubMed.

10.12.1 Parsing Medline records

You can find the Medline parser in Bio.Medline. Suppose we want to parse the file pubmed_result1.txt, containing one Medline record. You can find this file in Biopython's Tests\Medline directory. The file looks like this:

```
PMID- 12230038
OWN - NLM
STAT- MEDLINE
DA  - 20020916
DCOM- 20030606
LR  - 20041117
PUBM- Print
IS  - 1467-5463 (Print)
VI  - 3
IP  - 3
DP  - 2002 Sep
TI  - The Bio* toolkits--a brief overview.
PG  - 296-302
AB  - Bioinformatics research is often difficult to do with commercial software. The
      Open Source BioPerl, BioPython and Biojava projects provide toolkits with
...
```

We first open the file and then parse it:

```
In [36]: from Bio import Medline
         with open("data/pubmed_result1.txt") as handle:
             record = Medline.read(handle)
```

The record now contains the Medline record as a Python dictionary:

```
In [37]: record["PMID"]
```

```
Out[37]: '12230038'
```

```
In [38]: record["AB"]
```

```
Out[38]: 'Bioinformatics research is often difficult to do with commercial software. The Open Source
```

The key names used in a Medline record can be rather obscure; use

```
In [39]: help(record)
```

```
Help on Record in module Bio.Medline object:

class Record(builtins.dict)
 |  A dictionary holding information from a Medline record.
 |
 |  All data are stored under the mnemonic appearing in the Medline
 |  file. These mnemonics have the following interpretations:
 |
 |  ========= ==============================
 |  Mnemonic  Description
 |  --------- ------------------------------
 |  AB        Abstract
 |  CI        Copyright Information
 |  AD        Affiliation
 |  IRAD      Investigator Affiliation
 |  AID       Article Identifier
 |  AU        Author
 |  FAU       Full Author
 |  CN        Corporate Author
 |  DCOM      Date Completed
 |  DA        Date Created
 |  LR        Date Last Revised
 |  DEP       Date of Electronic Publication
 |  DP        Date of Publication
 |  EDAT      Entrez Date
 |  GS        Gene Symbol
 |  GN        General Note
 |  GR        Grant Number
 |  IR        Investigator Name
 |  FIR       Full Investigator Name
 |  IS        ISSN
 |  IP        Issue
 |  TA        Journal Title Abbreviation
 |  JT        Journal Title
 |  LA        Language
 |  LID       Location Identifier
 |  MID       Manuscript Identifier
 |  MHDA      MeSH Date
 |  MH        MeSH Terms
 |  JID       NLM Unique ID
 |  RF        Number of References
 |  OAB       Other Abstract
 |  OCI       Other Copyright Information
 |  OID       Other ID
 |  OT        Other Term
 |  OTO       Other Term Owner
 |  OWN       Owner
 |  PG        Pagination
 |  PS        Personal Name as Subject
 |  FPS       Full Personal Name as Subject
 |  PL        Place of Publication
 |  PHST      Publication History Status
 |  PST       Publication Status
 |  PT        Publication Type
 |  PUBM      Publishing Model
 |  PMC       PubMed Central Identifier
 |  PMID      PubMed Unique Identifier
 |  RN        Registry Number/EC Number
 |  NM        Substance Name
 |  SI        Secondary Source ID
 |  SO        Source
 |  SFM       Space Flight Mission
 |  STAT      Status
 |  SB        Subset
 |  TI        Title
 |  TT        Transliterated Title
 |  VI        Volume
 |  CON       Comment on
 |  CIN       Comment in
 |  EIN       Erratum in
 |  EFR       Erratum for
 |  CRI       Corrected and Republished in
 |  CRF       Corrected and Republished from
 |  PRIN      Partial retraction in
 |  PROF      Partial retraction of
 |  RPI       Republished in
 |  RPF       Republished from
 |  RIN       Retraction in
 |  ROF       Retraction of
 |  UIN       Update in
 |  UOF       Update of
 |  SPIN      Summary for patients in
 |  ORI       Original report in
 |  ========= ==============================
 |
 |  Method resolution order:
 |      Record
 |      builtins.dict
 |      builtins.object
 |
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.dict:
 |
 |  __contains__(self, key, /)
 |      True if D has a key k, else False.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
```

```
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(...)
|      D.__sizeof__() -> size of D in memory, in bytes
|
|  clear(...)
|      D.clear() -> None.  Remove all items from D.
|
|  copy(...)
|      D.copy() -> a shallow copy of D
|
|  fromkeys(iterable, value=None, /) from builtins.type
|      Returns a new dict with keys from iterable and values equal to value.
|
|  get(...)
|      D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None.
|
|  items(...)
|      D.items() -> a set-like object providing a view on D's items
|
|  keys(...)
|      D.keys() -> a set-like object providing a view on D's keys
|
|  pop(...)
|      D.pop(k[,d]) -> v, remove specified key and return the corresponding value.
|      If key is not found, d is returned if given, otherwise KeyError is raised
|
|  popitem(...)
|      D.popitem() -> (k, v), remove and return some (key, value) pair as a
|      2-tuple; but raise KeyError if D is empty.
```

```
 |
 |  setdefault(...)
 |      D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D
 |
 |  update(...)
 |      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
 |      If E is present and has a .keys() method, then does:  for k in E: D[k] = E[k]
 |      If E is present and lacks a .keys() method, then does:  for k, v in E: D[k] = v
 |      In either case, this is followed by: for k in F:  D[k] = F[k]
 |
 |  values(...)
 |      D.values() -> an object providing a view on D's values
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from builtins.dict:
 |
 |  __hash__ = None
```

for a brief summary.

To parse a file containing multiple Medline records, you can use the `parse` function instead:

```
In [40]: from Bio import Medline
         with open("data/pubmed_result2.txt") as handle:
             for record in Medline.parse(handle):
                 print(record["TI"])
```

```
A high level interface to SCOP and ASTRAL implemented in python.
GenomeDiagram: a python package for the visualization of large-scale genomic data.
Open source clustering software.
PDB file parser and structure class implemented in Python.
```

Instead of parsing Medline records stored in files, you can also parse Medline records downloaded by `Bio.Entrez.efetch`. For example, let's look at all Medline records in PubMed related to Biopython:

```
In [41]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.esearch(db="pubmed", term="biopython")
         record = Entrez.read(handle)
         record["IdList"]
```

```
Out[41]: ['24929426', '24497503', '24267035', '24194598', '23842806', '23157543', '22909249', '22399
```

We now use `Bio.Entrez.efetch` to download these Medline records:

```
In [42]: idlist = record["IdList"]
         handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline", retmode="text")
```

Here, we specify `rettype="medline", retmode="text"` to obtain the Medline records in plain-text Medline format. Now we use `Bio.Medline` to parse these records:

```
In [43]: from Bio import Medline
         records = Medline.parse(handle)
         for record in records:
             print(record["AU"])
```

```
['Waldmann J', 'Gerken J', 'Hankeln W', 'Schweer T', 'Glockner FO']
['Mielke CJ', 'Mandarino LJ', 'Dinu V']
['Gajda MJ']
['Mathelier A', 'Zhao X', 'Zhang AW', 'Parcy F', 'Worsley-Hunt R', 'Arenillas DJ', 'Buchman S', 'Che
['Morales HF', 'Giovambattista G']
['Baldwin S', 'Revanna R', 'Thomson S', 'Pither-Joyce M', 'Wright K', 'Crowhurst R', 'Fiers M', 'Che
['Talevich E', 'Invergo BM', 'Cock PJ', 'Chapman BA']
```

```
['Prins P', 'Goto N', 'Yates A', 'Gautier L', 'Willis S', 'Fields C', 'Katayama T']
['Schmitt T', 'Messina DN', 'Schreiber F', 'Sonnhammer EL']
['Antao T']
['Cock PJ', 'Fields CJ', 'Goto N', 'Heuer ML', 'Rice PM']
['Jankun-Kelly TJ', 'Lindeman AD', 'Bridges SM']
['Korhonen J', 'Martinmaki P', 'Pizzi C', 'Rastas P', 'Ukkonen E']
['Cock PJ', 'Antao T', 'Chang JT', 'Chapman BA', 'Cox CJ', 'Dalke A', 'Friedberg I', 'Hamelryck T',
['Munteanu CR', 'Gonzalez-Diaz H', 'Magalhaes AL']
['Faircloth BC']
['Casbon JA', 'Crooks GE', 'Saqi MA']
['Pritchard L', 'White JA', 'Birch PR', 'Toth IK']
['de Hoon MJ', 'Imoto S', 'Nolan J', 'Miyano S']
['Hamelryck T', 'Manderick B']
```

For comparison, here we show an example using the XML format:

```
In [44]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.esearch(db="pubmed", term="biopython")
         record = Entrez.read(handle)
         idlist = record["IdList"]
         handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline", retmode="xml")
         records = Entrez.read(handle)
         for record in records:
             print(record["MedlineCitation"]["Article"]["ArticleTitle"])
```

```
FastaValidator: an open-source Java library to parse and validate FASTA formatted sequences.
AMASS: a database for investigating protein structures.
HPDB-Haskell library for processing atomic biomolecular structures in Protein Data Bank format.
JASPAR 2014: an extensively expanded and updated open-access database of transcription factor binding
BioSmalltalk: a pure object system and library for bioinformatics.
A toolkit for bulk PCR-based marker design from next-generation sequence data: application for develo
Bio.Phylo: a unified toolkit for processing, analyzing and visualizing phylogenetic trees in Biopytho
Sharing programming resources between Bio* projects through remote procedure call and native call sta
Letter to the editor: SeqXML and OrthoXML: standards for sequence and orthology information.
interPopula: a Python API to access the HapMap Project dataset.
The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variant
Exploratory visual analysis of conserved domains on multiple sequence alignments.
MOODS: fast search for position weight matrix matches in DNA sequences.
Biopython: freely available Python tools for computational molecular biology and bioinformatics.
Enzymes/non-enzymes classification model complexity based on composition, sequence, 3D and topologica
msatcommander: detection of microsatellite repeat arrays and automated, locus-specific primer design
A high level interface to SCOP and ASTRAL implemented in python.
GenomeDiagram: a python package for the visualization of large-scale genomic data.
Open source clustering software.
PDB file parser and structure class implemented in Python.
```

Note that in both of these examples, for simplicity we have naively combined ESearch and EFetch. In this situation, the NCBI would expect you to use their history feature, as illustrated in Section *History and WebEnv*.

10.12.2 Parsing GEO records

GEO (Gene Expression Omnibus) is a data repository of high-throughput gene expression and hybridization array data. The `Bio.Geo` module can be used to parse GEO-formatted data.

The following code fragment shows how to parse the example GEO file `GSE16.txt` into a record and print the record:

```
In [45]: from Bio import Geo
         handle = open("data/GSE16.txt")
```

```
        records = Geo.parse(handle)
        for record in records:
            print(record)
```

```
GEO Type: SAMPLE
GEO Id: GSM804
Sample_author: Antoine,M,Snijders

Sample_author: Norma,,Nowak

Sample_author: Richard,,Segraves

Sample_author: Stephanie,,Blackwood

Sample_author: Nils,,Brown

Sample_author: Jeffery,,Conroy

Sample_author: Greg,,Hamilton

Sample_author: Anna,K,Hindle

Sample_author: Bing,,Huey

Sample_author: Karen,,Kimura

Sample_author: Sindy,,Law

Sample_author: Ken,,Myambo

Sample_author: Joel,,Palmer

Sample_author: Bauke,,Ylstra

Sample_author: Jingzhu,P,Yue

Sample_author: Joe,W,Gray

Sample_author: Ajay,N,Jain

Sample_author: Daniel,,Pinkel

Sample_author: Donna,G,Albertson

Sample_description: Coriell Cell Repositories cell line <a h
ref="http://locus.umdnj.edu/nigms/nigms_cgi/display.cgi?GM05296">GM05296</a>.

Sample_description: Fibroblast cell line derived from a 1 mo
nth old female with multiple congenital malformations, dysmorphic features, intr
auterine growth retardation, heart murmur, cleft palate, equinovarus deformity,
microcephaly, coloboma of right iris, clinodactyly, reduced RBC catalase activit
y, and 1 copy of catalase gene.

Sample_description: Chromosome abnormalities are present.

Sample_description: Karyotype is 46,XX,-11,+der(11)inv ins(1
1;10)(11pter> 11p13::10q21>10q24::11p13>11qter)mat

Sample_organism: Homo sapiens
```

```
Sample_platform_id: GPL28

Sample_pubmed_id: 11687795

Sample_series_id: GSE16

Sample_status: Public on Feb 12 2002

Sample_submission_date: Jan 17 2002

Sample_submitter_city: San Francisco,CA,94143,USA

Sample_submitter_department: Comprehensive Cancer Center

Sample_submitter_email: albertson@cc.ucsf.edu

Sample_submitter_institute: University of California San Francisco

Sample_submitter_name: Donna,G,Albertson

Sample_submitter_phone: 415 502-8463

Sample_target_source1: Cell line GM05296

Sample_target_source2: normal male reference genomic DNA

Sample_title: CGH_Albertson_GM05296-001218

Sample_type: dual channel genomic

Column Header Definitions
    ID_REF: Unique row identifier, genome position o
    rder

    LINEAR_RATIO: Mean of replicate Cy3/Cy5 ratios

    LOG2STDDEV: Standard deviation of VALUE

    NO_REPLICATES: Number of replicate spot measurements

    VALUE: aka LOG2RATIO, mean of log base 2 of LIN
    EAR_RATIO

0: ID_REF      VALUE   LINEAR_RATIO   LOG2STDDEV      NO_REPLICATES
1: 1            1.047765        0.011853        3
2: 2                           0
3: 3    0.008824        1.006135        0.00143 3
4: 4    -0.000894       0.99938 0.001454        3
5: 5    0.075875        1.054   0.003077        3
6: 6    0.017303        1.012066        0.005876        2
7: 7    -0.006766       0.995321        0.013881        3
8: 8    0.020755        1.014491        0.005506        3
9: 9    -0.094938       0.936313        0.012662        3
10: 10  -0.054527       0.96291 0.01073 3
11: 11  -0.025057       0.982782        0.003855        3
12: 12                         0
13: 13  0.108454        1.078072        0.005196        3
14: 14  0.078633        1.056017        0.009165        3
```

```
15: 15  0.098571        1.070712        0.007834        3
16: 16  0.044048        1.031003        0.013651        3
17: 17  0.018039        1.012582        0.005471        3
18: 18  -0.088807       0.9403  0.010571        3
19: 19  0.016349        1.011397        0.007113        3
20: 20  0.030977        1.021704        0.016798        3
```

You can search the "gds" database (GEO datasets) with ESearch:

```
In [46]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are
         handle = Entrez.esearch(db="gds", term="GSE16")
         record = Entrez.read(handle)
         record["Count"]
```

```
Out[46]: '27'
```

```
In [47]: record["IdList"]
```

```
Out[47]: ['200000016', '100000028', '300000818', '300000817', '300000816', '300000815', '300000814',
```

From the Entrez website, UID "200000016" is GDS16 while the other hit "100000028" is for the associated platform, GPL28. Unfortunately, at the time of writing the NCBI don't seem to support downloading GEO files using Entrez (not as XML, nor in the *Simple Omnibus Format in Text* (SOFT) format).

However, it is actually pretty straight forward to download the GEO files by FTP or HTTP from http://ftp.ncbi.nih.gov/pub/geo/ instead. In this case you might want http://ftp.ncbi.nih.gov/pub/geo/DATA/SOFT/by_series/GSE16/GSE16_family.soft.gz (a compressed file, see the Python module gzip).

10.12.3 Parsing UniGene records

UniGene is an NCBI database of the transcriptome, with each UniGene record showing the set of transcripts that are associated with a particular gene in a specific organism. A typical UniGene record looks like this:

```
ID          Hs.2
TITLE       N-acetyltransferase 2 (arylamine N-acetyltransferase)
GENE        NAT2
CYTOBAND    8p22
GENE_ID     10
LOCUSLINK   10
HOMOL       YES
EXPRESS      bone| connective tissue| intestine| liver| liver tumor| normal| soft␣
↪tissue/muscle tissue tumor| adult
RESTR_EXPR   adult
CHROMOSOME  8
STS         ACC=PMC310725P3 UNISTS=272646
STS         ACC=WIAF-2120 UNISTS=44576
STS         ACC=G59899 UNISTS=137181
...
STS         ACC=GDB:187676 UNISTS=155563
PROTSIM     ORG=10090; PROTGI=6754794; PROTID=NP_035004.1; PCT=76.55; ALN=288
PROTSIM     ORG=9796; PROTGI=149742490; PROTID=XP_001487907.1; PCT=79.66; ALN=288
PROTSIM     ORG=9986; PROTGI=126722851; PROTID=NP_001075655.1; PCT=76.90; ALN=288
...
PROTSIM     ORG=9598; PROTGI=114619004; PROTID=XP_519631.2; PCT=98.28; ALN=288

SCOUNT      38
SEQUENCE    ACC=BC067218.1; NID=g45501306; PID=g45501307; SEQTYPE=mRNA
SEQUENCE    ACC=NM_000015.2; NID=g116295259; PID=g116295260; SEQTYPE=mRNA
SEQUENCE    ACC=D90042.1; NID=g219415; PID=g219416; SEQTYPE=mRNA
SEQUENCE    ACC=D90040.1; NID=g219411; PID=g219412; SEQTYPE=mRNA
SEQUENCE    ACC=BC015878.1; NID=g16198419; PID=g16198420; SEQTYPE=mRNA
SEQUENCE    ACC=CR407631.1; NID=g47115198; PID=g47115199; SEQTYPE=mRNA
SEQUENCE    ACC=BG569293.1; NID=g13576946; CLONE=IMAGE:4722596; END=5'; LID=6989;␣
↪SEQTYPE=EST; TRACE=44157214
...
SEQUENCE    ACC=AU099534.1; NID=g13550663; CLONE=HSI08034; END=5'; LID=8800;␣
↪SEQTYPE=EST
//
```

This particular record shows the set of transcripts (shown in the `SEQUENCE` lines) that originate from the human gene NAT2, encoding en N-acetyltransferase. The `PROTSIM` lines show proteins with significant similarity to NAT2, whereas the `STS` lines show the corresponding sequence-tagged sites in the genome.

To parse UniGene files, use the `Bio.UniGene` module:

TODO: Need a working example

```
In [48]: # from Bio import UniGene
         # input = open("data/myunigenefile.data")
         # record = UniGene.read(input)
```

The `record` returned by `UniGene.read` is a Python object with attributes corresponding to the fields in the UniGene record. For example,

```
In [49]: # record.ID
```

```
In [50]: # record.title
```

The `EXPRESS` and `RESTR_EXPR` lines are stored as Python lists of strings:

```
['bone', 'connective tissue', 'intestine', 'liver', 'liver tumor', 'normal', 'soft␣
↪tissue/muscle tissue tumor', 'adult']
```

Specialized objects are returned for the `STS`, `PROTSIM`, and `SEQUENCE` lines, storing the keys shown in each line as attributes:

```
In [51]: # record.sts[0].acc
```

```
In [52]: # record.sts[0].unists
```

and similarly for the `PROTSIM` and `SEQUENCE` lines.

To parse a file containing more than one UniGene record, use the `parse` function in `Bio.UniGene`:

TODO: Need a working example

```
In [53]: # from Bio import UniGene
         # input = open("unigenerecords.data")
         # records = UniGene.parse(input)
         # for record in records:
         #     print(record.ID)
```

10.13 Using a proxy

Normally you won't have to worry about using a proxy, but if this is an issue on your network here is how to deal with it. Internally, `Bio.Entrez` uses the standard Python library `urllib` for accessing the NCBI servers. This will check an environment variable called `http_proxy` to configure any simple proxy automatically. Unfortunately this module does not support the use of proxies which require authentication.

You may choose to set the `http_proxy` environment variable once (how you do this will depend on your operating system). Alternatively you can set this within Python at the start of your script, for example:

```
import os
os.environ["http_proxy"] = "http://proxyhost.example.com:8080"
```

See the urllib documentation for more details.

10.14 Examples

10.14.1 PubMed and Medline

If you are in the medical field or interested in human issues (and many times even if you are not!), PubMed (http://www.ncbi.nlm.nih.gov/PubMed/) is an excellent source of all kinds of goodies. So like other things, we'd like to be able to grab information from it and use it in Python scripts.

In this example, we will query PubMed for all articles having to do with orchids (see section [sec:orchids] for our motivation). We first check how many of such articles there are:

```
In [54]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.egquery(term="orchid")
         record = Entrez.read(handle)
         for row in record["eGQueryResult"]:
             if row["DbName"]=="pubmed":
                 print(row["Count"])
```

```
1376
```

Now we use the `Bio.Entrez.efetch` function to download the PubMed IDs of these 463 articles:

```
In [55]: handle = Entrez.esearch(db="pubmed", term="orchid", retmax=463)
         record = Entrez.read(handle)
         idlist = record["IdList"]
         print("The first 10 Id's containing all of the PubMed IDs of articles related to orchids:\n
```

```
The first 10 Id's containing all of the PubMed IDs of articles related to orchids:
 ['26752741', '26743923', '26738548', '26732875', '26732614', '26724929', '26715121', '26713612', '2
```

Now that we've got them, we obviously want to get the corresponding Medline records and extract the information from them. Here, we'll download the Medline records in the Medline flat-file format, and use the `Bio.Medline` module to parse them:

```
In [56]: from Bio import Medline
         handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline")
```

```
In [57]: records = Medline.parse(handle)
```

NOTE - We've just done a separate search and fetch here, the NCBI much prefer you to take advantage of their history support in this situation. See Section *History and WebEnv*.

Keep in mind that `records` is an iterator, so you can iterate through the records only once. If you want to save the records, you can convert them to a list:

```
In [58]: records = list(records)
```

Let's now iterate over the records to print out some information about each record:

```
In [59]: for record in records:
             print("title:", record.get("TI", "?"))
             print("authors:", record.get("AU", "?"))
```

```
            print("source:", record.get("SO", "?"))
            print("")
```

```
title: Promise and Challenge of DNA Barcoding in Venus Slipper (Paphiopedilum).
authors: ['Guo YY', 'Huang LQ', 'Liu ZJ', 'Wang XQ']
source: PLoS One. 2016 Jan 11;11(1):e0146880. doi: 10.1371/journal.pone.0146880. eCollection 2016.

title: In vitro profiling of anti-MRSA activity of thymoquinone against selected type and clinical st
authors: ['Hariharan P', 'Paul-Satyaseela M', 'Gnanamani A']
source: Lett Appl Microbiol. 2016 Jan 7. doi: 10.1111/lam.12544.

title: Low glutathione redox state couples with a decreased ascorbate redox ratio to accelerate flowe
authors: ['Chin DC', 'Hsieh CC', 'Lin HY', 'Yeh KW']
source: Plant Cell Physiol. 2016 Jan 6. pii: pcv206.

title: Proteomic and morphometric study of the in vitro interaction between Oncidium sphacelatum Lind
authors: ['Lopez-Chavez MY', 'Guillen-Navarro K', 'Bertolini V', 'Encarnacion S', 'Hernandez-Ortiz M
source: Mycorrhiza. 2016 Jan 6.

title: A transcriptome-wide, organ-specific regulatory map of Dendrobium officinale, an important tra
authors: ['Meng Y', 'Yu D', 'Xue J', 'Lu J', 'Feng S', 'Shen C', 'Wang H']
source: Sci Rep. 2016 Jan 6;6:18864. doi: 10.1038/srep18864.

title: Methods for genetic transformation in Dendrobium.
authors: ['Teixeira da Silva JA', 'Dobranszki J', 'Cardoso JC', 'Chandler SF', 'Zeng S']
source: Plant Cell Rep. 2016 Jan 2.

title: Sebacina vermifera: a unique root symbiont with vast agronomic potential.
authors: ['Ray P', 'Craven KD']
source: World J Microbiol Biotechnol. 2016 Jan;32(1):16. doi: 10.1007/s11274-015-1970-7. Epub 2015 De

title: Cuticular Hydrocarbons of Orchid Bees Males: Interspecific and Chemotaxonomy Variation.
authors: ['Dos Santos AB', 'do Nascimento FS']
source: PLoS One. 2015 Dec 29;10(12):e0145070. doi: 10.1371/journal.pone.0145070. eCollection 2015.

title: Sex and the Catasetinae (Darwin's favourite orchids).
authors: ['Perez-Escobar OA', 'Gottschling M', 'Whitten WM', 'Salazar G', 'Gerlach G']
source: Mol Phylogenet Evol. 2015 Dec 17. pii: S1055-7903(15)00372-3. doi: 10.1016/j.ympev.2015.11.01

title: Comparative Transcriptome Analysis of Genes Involved in GA-GID1-DELLA Regulatory Module in Sym
authors: ['Liu SS', 'Chen J', 'Li SC', 'Zeng X', 'Meng ZX', 'Guo SX']
source: Int J Mol Sci. 2015 Dec 18;16(12):30190-203. doi: 10.3390/ijms161226224.

title: Dual Drug Loaded Nanoliposomal Chemotherapy: A Promising Strategy for Treatment of Head and Ne
authors: ['Mohan A', 'Narayanan S', 'Balasubramanian G', 'Sethuraman S', 'Krishnan UM']
source: Eur J Pharm Biopharm. 2015 Dec 9. pii: S0939-6411(15)00489-0. doi: 10.1016/j.ejpb.2015.11.01

title: Orally available stilbene derivatives as potent HDAC inhibitors with antiproliferative activit
authors: ['Kachhadia V', 'Rajagopal S', 'Ponpandian T', 'Vignesh R', 'Anandhan K', 'Prabhu D', 'Rajen
source: Eur J Med Chem. 2015 Nov 19;108:274-286. doi: 10.1016/j.ejmech.2015.11.014.

title: Parapheromones for Thynnine Wasps.
authors: ['Bohman B', 'Karton A', 'Dixon RC', 'Barrow RA', 'Peakall R']
source: J Chem Ecol. 2015 Dec 14.

title: Interaction networks and the use of floral resources by male orchid bees (Hymenoptera: Apidae
authors: ['Ospina-Torres R', 'Montoya-Pfeiffer PM', 'Parra-H A', 'Solarte V', 'Tupac Otero J']
source: Rev Biol Trop. 2015 Sep;63(3):647-58.
```

```
title: A taste of pineapple evolution through genome sequencing.
authors: ['Xu Q', 'Liu ZJ']
source: Nat Genet. 2015 Dec 1;47(12):1374-6. doi: 10.1038/ng.3450.

title: Somatic Embryogenesis in Two Orchid Genera (Cymbidium, Dendrobium).
authors: ['Teixeira da Silva JA', 'Winarto B']
source: Methods Mol Biol. 2016;1359:371-86. doi: 10.1007/978-1-4939-3061-6_18.

title: Tumors of the Testis: Morphologic Features and Molecular Alterations.
authors: ['Howitt BE', 'Berney DM']
source: Surg Pathol Clin. 2015 Dec;8(4):687-716. doi: 10.1016/j.path.2015.07.007.

title: Scent emission profiles from Darwin's orchid - Angraecum sesquipedale: Investigation of the al
authors: ['Nielsen LJ', 'Moller BL']
source: Phytochemistry. 2015 Dec;120:3-18. doi: 10.1016/j.phytochem.2015.10.004. Epub 2015 Oct 22.

title: Two common species dominate the species-rich Euglossine bee fauna of an Atlantic Rainforest re
authors: ['Oliveira R', 'Pinto CE', 'Schlindwein C']
source: Braz J Biol. 2015 Nov;75(4 Suppl 1):1-8. doi: 10.1590/1519-6984.18513. Epub 2015 Nov 24.

title: Variation in the Abundance of Neotropical Bees in an Unpredictable Seasonal Environment.
authors: ['Knoll FR']
source: Neotrop Entomol. 2015 Nov 23.

title: Digital Gene Expression Analysis Based on De Novo Transcriptome Assembly Reveals New Genes Ass
authors: ['Yang F', 'Zhu G']
source: PLoS One. 2015 Nov 18;10(11):e0142434. doi: 10.1371/journal.pone.0142434. eCollection 2015.

title: LONG-TERM CONSERVATION OF PROTOCORMS OF Brassavola nodosa (L) LIND. (ORCHIDACEAE): EFFECT OF A
authors: ['Mata-Rosas M', 'Lastre-Puertos E']
source: Cryo Letters. 2015 Sep-Oct;36(5):289-98.

title: Cuticular Hydrocarbons as Potential Close Range Recognition Cues in Orchid Bees.
authors: ['Pokorny T', 'Ramirez SR', 'Weber MG', 'Eltz T']
source: J Chem Ecol. 2015 Dec;41(12):1080-94. doi: 10.1007/s10886-015-0647-x. Epub 2015 Nov 14.

title: Bilobate leaves of Bauhinia (Leguminosae, Caesalpinioideae, Cercideae) from the middle Miocene
authors: ['Lin Y', 'Wong WO', 'Shi G', 'Shen S', 'Li Z']
source: BMC Evol Biol. 2015 Nov 16;15(1):252. doi: 10.1186/s12862-015-0540-9.

title: Functional Significance of Labellum Pattern Variation in a Sexually Deceptive Orchid (Ophrys h
authors: ['Stejskal K', 'Streinzer M', 'Dyer A', 'Paulus HF', 'Spaethe J']
source: PLoS One. 2015 Nov 16;10(11):e0142971. doi: 10.1371/journal.pone.0142971. eCollection 2015.

title: Centralization of cleft care in the UK. Part 6: a tale of two studies.
authors: ['Ness AR', 'Wills AK', 'Waylen A', 'Al-Ghatam R', 'Jones TE', 'Preston R', 'Ireland AJ', 'P
source: Orthod Craniofac Res. 2015 Nov;18 Suppl 2:56-62. doi: 10.1111/ocr.12111.

title: The Cleft Care UK study. Part 4: perceptual speech outcomes.
authors: ['Sell D', 'Mildinhall S', 'Albery L', 'Wills AK', 'Sandy JR', 'Ness AR']
source: Orthod Craniofac Res. 2015 Nov;18 Suppl 2:36-46. doi: 10.1111/ocr.12112.

title: A cross-sectional survey of 5-year-old children with non-syndromic unilateral cleft lip and pa
authors: ['Persson M', 'Sandy JR', 'Waylen A', 'Wills AK', 'Al-Ghatam R', 'Ireland AJ', 'Hall AJ', 'H
source: Orthod Craniofac Res. 2015 Nov;18 Suppl 2:1-13. doi: 10.1111/ocr.12104.

title: Seven New Complete Plastome Sequences Reveal Rampant Independent Loss of the ndh Gene Family a
authors: ['Kim HT', 'Kim JS', 'Moore MJ', 'Neubig KM', 'Williams NH', 'Whitten WM', 'Kim JH']
```

```
source: PLoS One. 2015 Nov 11;10(11):e0142215. doi: 10.1371/journal.pone.0142215. eCollection 2015.

title: Orchid Species Richness along Elevational and Environmental Gradients in Yunnan, China.
authors: ['Zhang SB', 'Chen WY', 'Huang JL', 'Bi YF', 'Yang XF']
source: PLoS One. 2015 Nov 10;10(11):e0142621. doi: 10.1371/journal.pone.0142621. eCollection 2015.

title: Severe outbreeding and inbreeding depression maintain mating system differentiation in Epipact
authors: ['Brys R', 'Jacquemyn H']
source: J Evol Biol. 2015 Nov 9. doi: 10.1111/jeb.12787.

title: Simultaneous detection of Cymbidium mosaic virus and Odontoglossum ringspot virus in orchids u
authors: ['Kim SM', 'Choi SH']
source: Virus Genes. 2015 Dec;51(3):417-22. doi: 10.1007/s11262-015-1258-x. Epub 2015 Nov 5.

title: Analysis of the TCP genes expressed in the inflorescence of the orchid Orchis italica.
authors: ['De Paolo S', 'Gaudio L', 'Aceto S']
source: Sci Rep. 2015 Nov 4;5:16265. doi: 10.1038/srep16265.

title: RNA-Seq SSRs of Moth Orchid and Screening for Molecular Markers across Genus Phalaenopsis (Orc
authors: ['Tsai CC', 'Shih HC', 'Wang HV', 'Lin YS', 'Chang CH', 'Chiang YC', 'Chou CH']
source: PLoS One. 2015 Nov 2;10(11):e0141761. doi: 10.1371/journal.pone.0141761. eCollection 2015.

title: Mapping Adolescent Cancer Services: How Do Young People, Their Families, and Staff Describe Sp
authors: ['Vindrola-Padros C', 'Taylor RM', 'Lea S', 'Hooker L', 'Pearce S', 'Whelan J', 'Gibson F']
source: Cancer Nurs. 2015 Oct 28.

title: A putative miR172-targeted CeAPETALA2-like gene is involved in floral patterning regulation of
authors: ['Yang FX', 'Zhu GF', 'Wang Z', 'Liu HL', 'Huang D']
source: Genet Mol Res. 2015 Oct 5;14(4):12049-61. doi: 10.4238/2015.October.5.18.

title: Functional Characterization of PhapLEAFY, a FLORICAULA/LEAFY Ortholog in Phalaenopsis aphrodit
authors: ['Jang S']
source: Plant Cell Physiol. 2015 Nov;56(11):2234-47. doi: 10.1093/pcp/pcv130. Epub 2015 Oct 22.

title: Evolutionary history of PEPC genes in green plants: Implications for the evolution of CAM in o
authors: ['Deng H', 'Zhang LS', 'Zhang GQ', 'Zheng BQ', 'Liu ZJ', 'Wang Y']
source: Mol Phylogenet Evol. 2016 Jan;94(Pt B):559-64. doi: 10.1016/j.ympev.2015.10.007. Epub 2015 Oc

title: Three new alkaloids and three new phenolic glycosides from Liparis odorata.
authors: ['Jiang P', 'Liu H', 'Xu X', 'Liu B', 'Zhang D', 'Lai X', 'Zhu G', 'Xu P', 'Li B']
source: Fitoterapia. 2015 Dec;107:63-8. doi: 10.1016/j.fitote.2015.10.003. Epub 2015 Oct 19.

title: Microsatellite-based genetic diversity patterns in disjunct populations of a rare orchid.
authors: ['Pandey M', 'Richards M', 'Sharma J']
source: Genetica. 2015 Oct 20.

title: Effects of fusaric acid treatment on the protocorm-like bodies of Dendrobium sonia-28.
authors: ['Dehgahi R', 'Zakaria L', 'Mohamad A', 'Joniyas A', 'Subramaniam S']
source: Protoplasma. 2015 Oct 15.

title: Genes are information, so information theory is coming to the aid of evolutionary biology.
authors: ['Sherwin WB']
source: Mol Ecol Resour. 2015 Nov;15(6):1259-61. doi: 10.1111/1755-0998.12458.

title: A Comprehensive Review of the Cosmeceutical Benefits of Vanda Species (Orchidaceae).
authors: ['Hadi H', 'Razali SN', 'Awadh AI']
source: Nat Prod Commun. 2015 Aug;10(8):1483-8.

title: The C-Terminal Sequence and PI motif of the Orchid (Oncidium Gower Ramsey) PISTILLATA (PI) Ort
```

```
authors: ['Mao WT', 'Hsu HF', 'Hsu WH', 'Li JY', 'Lee YI', 'Yang CH']
source: Plant Cell Physiol. 2015 Nov;56(11):2079-99. doi: 10.1093/pcp/pcv129. Epub 2015 Sep 30.

title: Alternative translation initiation codons for the plastid maturase MatK: unraveling the pseudo
authors: ['Barthet MM', 'Moukarzel K', 'Smith KN', 'Patel J', 'Hilu KW']
source: BMC Evol Biol. 2015 Sep 29;15:210. doi: 10.1186/s12862-015-0491-1.

title: A dual functional probe for "turn-on" fluorescence response of Pb(2+) and colorimetric detecti
authors: ['Li M', 'Jiang XJ', 'Wu HH', 'Lu HL', 'Li HY', 'Xu H', 'Zang SQ', 'Mak TC']
source: Dalton Trans. 2015 Oct 21;44(39):17326-34. doi: 10.1039/c5dt02731d. Epub 2015 Sep 21.

title: Mining from transcriptomes: 315 single-copy orthologous genes concatenated for the phylogeneti
authors: ['Deng H', 'Zhang GQ', 'Lin M', 'Wang Y', 'Liu ZJ']
source: Ecol Evol. 2015 Sep;5(17):3800-7. doi: 10.1002/ece3.1642. Epub 2015 Aug 20.

title: Orchid conservation in the biodiversity hotspot of southwestern China.
authors: ['Liu Q', 'Chen J', 'Corlett RT', 'Fan X', 'Yu D', 'Yang H', 'Gao J']
source: Conserv Biol. 2015 Sep 15. doi: 10.1111/cobi.12584.

title: Biochemical characterization of embryogenic calli of Vanilla planifolia in response to two yea
authors: ['Kodja H', 'Noirot M', 'Khoyratty SS', 'Limbada H', 'Verpoorte R', 'Palama TL']
source: Plant Physiol Biochem. 2015 Nov;96:337-44. doi: 10.1016/j.plaphy.2015.08.017. Epub 2015 Aug 2

title: Vanda roxburghii: an experimental evaluation of antinociceptive properties of a traditional ep
authors: ['Uddin MJ', 'Rahman MM', 'Abdullah-Al-Mamun M', 'Sadik G']
source: BMC Complement Altern Med. 2015 Sep 3;15:305. doi: 10.1186/s12906-015-0833-y.

title: Rapid evolution of chemosensory receptor genes in a pair of sibling species of orchid bees (Ap
authors: ['Brand P', 'Ramirez SR', 'Leese F', 'Quezada-Euan JJ', 'Tollrian R', 'Eltz T']
source: BMC Evol Biol. 2015 Aug 28;15:176. doi: 10.1186/s12862-015-0451-9.

title: Changes in Orchid Bee Communities Across Forest-Agroecosystem Boundaries in Brazilian Atlantic
authors: ['Aguiar WM', 'Sofia SH', 'Melo GA', 'Gaglianone MC']
source: Environ Entomol. 2015 Dec;44(6):1465-71. doi: 10.1093/ee/nvv130. Epub 2015 Aug 11.

title: Orchid conservation: making the links.
authors: ['Fay MF', 'Pailler T', 'Dixon KW']
source: Ann Bot. 2015 Sep;116(3):377-9. doi: 10.1093/aob/mcv142.

title: Orchid phylogenomics and multiple drivers of their extraordinary diversification.
authors: ['Givnish TJ', 'Spalink D', 'Ames M', 'Lyon SP', 'Hunter SJ', 'Zuluaga A', 'Iles WJ', 'Cleme
source: Proc Biol Sci. 2015 Sep 7;282(1814). doi: 10.1098/rspb.2015.1553.

title: Photoprotection related to xanthophyll cycle pigments in epiphytic orchids acclimated at diffe
authors: ['de la Rosa-Manzano E', 'Andrade JL', 'Garcia-Mendoza E', 'Zotz G', 'Reyes-Garcia C']
source: Planta. 2015 Dec;242(6):1425-38. doi: 10.1007/s00425-015-2383-4. Epub 2015 Aug 25.

title: Mycorrhizal fungi isolated from native terrestrial orchids of pristine regions in Cordoba (Arg
authors: ['Fernandez Di Pardo A', 'Chiocchio VM', 'Barrera V', 'Colombo RP', 'Martinez AE', 'Gasoni L
source: Rev Biol Trop. 2015 Mar;63(1):275-83.

title: Orchid-pollinator interactions and potential vulnerability to biological invasion.
authors: ['Chupp AD', 'Battaglia LL', 'Schauber EM', 'Sipes SD']
source: AoB Plants. 2015 Aug 17;7. pii: plv099. doi: 10.1093/aobpla/plv099.

title: Germination and seedling establishment in orchids: a complex of requirements.
authors: ['Rasmussen HN', 'Dixon KW', 'Jersakova J', 'Tesitelova T']
source: Ann Bot. 2015 Sep;116(3):391-402. doi: 10.1093/aob/mcv087. Epub 2015 Aug 12.
```

```
title: Capsule formation and asymbiotic seed germination in some hybrids of Phalaenopsis, influenced
authors: ['Balilashaki K', 'Gantait S', 'Naderi R', 'Vahedi M']
source: Physiol Mol Biol Plants. 2015 Jul;21(3):341-7. doi: 10.1007/s12298-015-0309-z. Epub 2015 Jul

title: dsRNA silencing of an R2R3-MYB transcription factor affects flower cell shape in a Dendrobium
authors: ['Lau SE', 'Schwarzacher T', 'Othman RY', 'Harikrishna JA']
source: BMC Plant Biol. 2015 Aug 11;15:194. doi: 10.1186/s12870-015-0577-3.

title: Clinical and echocardiographic characteristics for differentiating between transthyretin-relat
authors: ['Mori M', 'An Y', 'Katayama O', 'Kitagawa T', 'Sasaki Y', 'Onaka T', 'Yonezawa A', 'Murata
source: Ann Hematol. 2015 Nov;94(11):1885-90. doi: 10.1007/s00277-015-2466-0. Epub 2015 Aug 8.

title: First record of the orchid bee genus Eufriesea Cockerell (Hymenoptera: Apidae: Euglossini) in
authors: ['Griswold T', 'Herndon JD', 'Gonzalez VH']
source: Zootaxa. 2015 May 15;3957(3):342-6. doi: 10.11646/zootaxa.3957.3.7.

title: Thuniopsis: A New Orchid Genus and Phylogeny of the Tribe Arethuseae (Orchidaceae).
authors: ['Li L', 'Ye DP', 'Niu M', 'Yan HF', 'Wen TL', 'Li SJ']
source: PLoS One. 2015 Aug 5;10(8):e0132777. doi: 10.1371/journal.pone.0132777. eCollection 2015.

title: Additive effects of pollinators and herbivores result in both conflicting and reinforcing sele
authors: ['Sletvold N', 'Moritz KK', 'Agren J']
source: Ecology. 2015 Jan;96(1):214-21.

title: Terrestrial orchids in a tropical forest: best sites for abundance differ from those for repro
authors: ['Whitman M', 'Ackerman JD']
source: Ecology. 2015 Mar;96(3):693-704.

title: Spiranthes sinensis Suppresses Production of Pro-Inflammatory Mediators by Down-Regulating the
authors: ['Shie PH', 'Huang SS', 'Deng JS', 'Huang GJ']
source: Am J Chin Med. 2015;43(5):969-89. doi: 10.1142/S0192415X15500561. Epub 2015 Jul 30.

title: Phosphodiesterase inhibitor, pentoxifylline enhances anticancer activity of histone deacetylas
authors: ['Nidhyanandan S', 'Boreddy TS', 'Chandrasekhar KB', 'Reddy ND', 'Kulkarni NM', 'Narayanan S
source: Eur J Pharmacol. 2015 Oct 5;764:508-19. doi: 10.1016/j.ejphar.2015.07.048. Epub 2015 Jul 21.

title: The effects of smoke derivatives on in vitro seed germination and development of the leopard o
authors: ['Papenfus HB', 'Naidoo D', 'Posta M', 'Finnie JF', 'Van Staden J']
source: Plant Biol (Stuttg). 2015 Jul 23. doi: 10.1111/plb.12374.

title: DhEFL2, 3 and 4, the three EARLY FLOWERING4-like genes in a Doritaenopsis hybrid regulate flor
authors: ['Chen W', 'Qin Q', 'Zhang C', 'Zheng Y', 'Wang C', 'Zhou M', 'Cui Y']
source: Plant Cell Rep. 2015 Dec;34(12):2027-41. doi: 10.1007/s00299-015-1848-z. Epub 2015 Jul 24.

title: Spatial variation in pollinator-mediated selection on phenology, floral display and spur lengt
authors: ['Chapurlat E', 'Agren J', 'Sletvold N']
source: New Phytol. 2015 Dec;208(4):1264-75. doi: 10.1111/nph.13555. Epub 2015 Jul 15.

title: Building the Evidence for Nursing Practice: Learning from a Structured Review of SIOP Abstract
authors: ['Gibson F', 'Vindrola-Padros C', 'Hinds P', 'Nolbris MJ', 'Kelly D', 'Kelly P', 'Ruccione K
source: Pediatr Blood Cancer. 2015 Dec;62(12):2172-6. doi: 10.1002/pbc.25652. Epub 2015 Jul 14.

title: Genetic variability within and among populations of an invasive, exotic orchid.
authors: ['Ueno S', 'Rodrigues JF', 'Alves-Pereira A', 'Pansarin ER', 'Veasey EA']
source: AoB Plants. 2015 Jul 10;7. pii: plv077. doi: 10.1093/aobpla/plv077.

title: Hydrolysis of clavulanate by Mycobacterium tuberculosis beta-lactamase BlaC harboring a canoni
authors: ['Soroka D', 'Li de la Sierra-Gallay I', 'Dubee V', 'Triboulet S', 'van Tilbeurgh H', 'Compa
source: Antimicrob Agents Chemother. 2015 Sep;59(9):5714-20. doi: 10.1128/AAC.00598-15. Epub 2015 Jul
```

```
title: Experimental fertilization increases amino acid content in floral nectar, fruit set and degree
authors: ['Gijbels P', 'Ceulemans T', 'Van den Ende W', 'Honnay O']
source: Oecologia. 2015 Nov;179(3):785-95. doi: 10.1007/s00442-015-3381-8. Epub 2015 Jul 7.

title: Seasonal cycles, phylogenetic assembly, and functional diversity of orchid bee communities.
authors: ['Ramirez SR', 'Hernandez C', 'Link A', 'Lopez-Uribe MM']
source: Ecol Evol. 2015 May;5(9):1896-907. doi: 10.1002/ece3.1466. Epub 2015 Apr 13.

title: Migration of nonylphenol from food-grade plastic is toxic to the coral reef fish species Pseud
authors: ['Hamlin HJ', 'Marciano K', 'Downs CA']
source: Chemosphere. 2015 Nov;139:223-8. doi: 10.1016/j.chemosphere.2015.06.032. Epub 2015 Jun 29.

title: Responses to simulated nitrogen deposition by the neotropical epiphytic orchid Laelia speciosa
authors: ['Diaz-Alvarez EA', 'Lindig-Cisneros R', 'de la Barrera E']
source: PeerJ. 2015 Jun 23;3:e1021. doi: 10.7717/peerj.1021. eCollection 2015.

title: Applicability of ISSR and DAMD markers for phyto-molecular characterization and association wi
authors: ['Bhattacharyya P', 'Kumaria S', 'Tandon P']
source: Phytochemistry. 2015 Sep;117:306-16. doi: 10.1016/j.phytochem.2015.06.022. Epub 2015 Jun 27.

title: The importance of associations with saprotrophic non-Rhizoctonia fungi among fully mycoheterot
authors: ['Lee YI', 'Yang CK', 'Gebauer G']
source: Ann Bot. 2015 Sep;116(3):423-35. doi: 10.1093/aob/mcv085. Epub 2015 Jun 25.

title: Continent-wide distribution in mycorrhizal fungi: implications for the biogeography of special
authors: ['Davis BJ', 'Phillips RD', 'Wright M', 'Linde CC', 'Dixon KW']
source: Ann Bot. 2015 Sep;116(3):413-21. doi: 10.1093/aob/mcv084. Epub 2015 Jun 22.

title: Dynamic distribution and the role of abscisic acid during seed development of a lady's slipper
authors: ['Lee YI', 'Chung MC', 'Yeung EC', 'Lee N']
source: Ann Bot. 2015 Sep;116(3):403-11. doi: 10.1093/aob/mcv079. Epub 2015 Jun 22.

title: Characterization of microsatellite loci for an Australian epiphytic orchid, Dendrobium calami
authors: ['Trapnell DW', 'Beasley RR', 'Lance SL', 'Field AR', 'Jones KL']
source: Appl Plant Sci. 2015 Jun 5;3(6). pii: apps.1500016. doi: 10.3732/apps.1500016. eCollection 20

title: Factors affecting reproductive success in three entomophilous orchid species in Hungary.
authors: ['Vojtko AE', 'Sonkoly J', 'Lukacs BA', 'Molnar V A']
source: Acta Biol Hung. 2015 Jun;66(2):231-41. doi: 10.1556/018.66.2015.2.9.

title: Pollination by sexual deception promotes outcrossing and mate diversity in self-compatible clo
authors: ['Whitehead MR', 'Linde CC', 'Peakall R']
source: J Evol Biol. 2015 Aug;28(8):1526-41. doi: 10.1111/jeb.12673. Epub 2015 Jul 3.

title: Adding Biotic Interactions into Paleodistribution Models: A Host-Cleptoparasite Complex of Neo
authors: ['Silva DP', 'Varela S', 'Nemesio A', 'De Marco P Jr']
source: PLoS One. 2015 Jun 12;10(6):e0129890. doi: 10.1371/journal.pone.0129890. eCollection 2015.

title: Mapping of the Interaction Between Agrobacterium tumefaciens and Vanda Kasem's Delight Orchid
authors: ['Gnasekaran P', 'Subramaniam S']
source: Indian J Microbiol. 2015 Sep;55(3):285-91. doi: 10.1007/s12088-015-0519-7. Epub 2015 Feb 25.

title: Spatial asymmetries in connectivity influence colonization-extinction dynamics.
authors: ['Acevedo MA', 'Fletcher RJ Jr', 'Tremblay RL', 'Melendez-Ackerman EJ']
source: Oecologia. 2015 Oct;179(2):415-24. doi: 10.1007/s00442-015-3361-z. Epub 2015 Jun 10.

title: Dendrobium micropropagation: a review.
authors: ['da Silva JA', 'Cardoso JC', 'Dobranszki J', 'Zeng S']
```

```
source: Plant Cell Rep. 2015 May;34(5):671-704.

title: Crystal structure of 2-(4-fluoro-3-methyl-phen-yl)-5-{[(naphthalen-1-yl)-oxy]meth-yl}-1,3,4-ox
authors: ['Govindhan M', 'Subramanian K', 'Viswanathan V', 'Velmurugan D']
source: Acta Crystallogr E Crystallogr Commun. 2015 Mar 11;71(Pt 4):o229-30. doi: 10.1107/S205698901

title: Cymbidium chlorotic mosaic virus, a new sobemovirus isolated from a spring orchid (Cymbidium g
authors: ['Kondo H', 'Takemoto S', 'Maruyama K', 'Chiba S', 'Andika IB', 'Suzuki N']
source: Arch Virol. 2015 Aug;160(8):2099-104. doi: 10.1007/s00705-015-2460-9. Epub 2015 May 31.

title: Effect of pesticide exposure on immunological, hematological and biochemical parameters in tha
authors: ['Aroonvilairat S', 'Kespichayawattana W', 'Sornprachum T', 'Chaisuriya P', 'Siwadune T', 'I
source: Int J Environ Res Public Health. 2015 May 27;12(6):5846-61. doi: 10.3390/ijerph120605846.

title: Phylogeny and classification of the East Asian Amitostigma alliance (Orchidaceae: Orchideae) b
authors: ['Tang Y', 'Yukawa T', 'Bateman RM', 'Jiang H', 'Peng H']
source: BMC Evol Biol. 2015 May 26;15:96. doi: 10.1186/s12862-015-0376-3.

title: Combinations of beta-Lactam Antibiotics Currently in Clinical Trials Are Efficacious in a DHP-
authors: ['Rullas J', 'Dhar N', 'McKinney JD', 'Garcia-Perez A', 'Lelievre J', 'Diacon AH', 'Hugonnet
source: Antimicrob Agents Chemother. 2015 Aug;59(8):4997-9. doi: 10.1128/AAC.01063-15. Epub 2015 May

title: A de novo floral transcriptome reveals clues into Phalaenopsis orchid flower development.
authors: ['Huang JZ', 'Lin CP', 'Cheng TC', 'Chang BC', 'Cheng SY', 'Chen YW', 'Lee CY', 'Chin SW',
source: PLoS One. 2015 May 13;10(5):e0123474. doi: 10.1371/journal.pone.0123474. eCollection 2015.

title: Mycorrhizal diversity, seed germination and long-term changes in population size across nine p
authors: ['Jacquemyn H', 'Waud M', 'Merckx VS', 'Lievens B', 'Brys R']
source: Mol Ecol. 2015 Jul;24(13):3269-80. doi: 10.1111/mec.13236. Epub 2015 Jun 5.

title: Potential osteogenic activity of ethanolic extract and oxoflavidin isolated from Pholidota art
authors: ['Sharma C', 'Dixit M', 'Singh R', 'Agrawal M', 'Mansoori MN', 'Kureel J', 'Singh D', 'Narer
source: J Ethnopharmacol. 2015 Jul 21;170:57-65. doi: 10.1016/j.jep.2015.04.045. Epub 2015 May 8.

title: Transitions between self-compatibility and self-incompatibility and the evolution of reproduct
authors: ['Pinheiro F', 'Cafasso D', 'Cozzolino S', 'Scopece G']
source: Ann Bot. 2015 Sep;116(3):457-67. doi: 10.1093/aob/mcv057. Epub 2015 May 7.

title: A new species of Cnemaspis (Sauria: Gekkonidae) from Northern Karnataka, India.
authors: ['Srinivasulu C', 'Kumar GC', 'Srinivasulu B']
source: Zootaxa. 2015 Apr 14;3947(1):85-98. doi: 10.11646/zootaxa.3947.1.5.

title: Visual profile of students in integrated schools in Malawi.
authors: ['Kaphle D', 'Marasini S', 'Kalua K', 'Reading A', 'Naidoo KS']
source: Clin Exp Optom. 2015 Jul;98(4):370-4. doi: 10.1111/cxo.12269. Epub 2015 May 5.

title: A direct assessment of realized seed and pollen flow within and between two isolated populatio
authors: ['Helsen K', 'Meekers T', 'Vranckx G', 'Roldan-Ruiz I', 'Vandepitte K', 'Honnay O']
source: Plant Biol (Stuttg). 2015 May 4. doi: 10.1111/plb.12342.

title: Challenges of flow-cytometric estimation of nuclear genome size in orchids, a plant group with
authors: ['Travnicek P', 'Ponert J', 'Urfus T', 'Jersakova J', 'Vrana J', 'Hribova E', 'Dolezel J',
source: Cytometry A. 2015 Oct;87(10):958-66. doi: 10.1002/cyto.a.22681. Epub 2015 Apr 30.

title: Transcriptome-wide analysis of the MADS-box gene family in the orchid Erycina pusilla.
authors: ['Lin CS', 'Hsu CT', 'Liao C', 'Chang WJ', 'Chou ML', 'Huang YT', 'Chen JJ', 'Ko SS', 'Chan
source: Plant Biotechnol J. 2015 Apr 28. doi: 10.1111/pbi.12383.

title: An informational diversity framework, illustrated with sexually deceptive orchids in early sta
```

```
authors: ['Smouse PE', 'Whitehead MR', 'Peakall R']
source: Mol Ecol Resour. 2015 Nov;15(6):1375-84. doi: 10.1111/1755-0998.12422. Epub 2015 May 20.

title: A new myco-heterotrophic genus, Yunorchis, and the molecular phylogenetic relationships of the
authors: ['Zhang GQ', 'Li MH', 'Su YY', 'Chen LJ', 'Lan SR', 'Liu ZJ']
source: PLoS One. 2015 Apr 22;10(4):e0123382. doi: 10.1371/journal.pone.0123382. eCollection 2015.

title: Phylogenetic placement and taxonomy of the genus Hederorkis (Orchidaceae).
authors: ['Mytnik-Ejsmont J', 'Szlachetko DL', 'Baranow P', 'Jolliffe K', 'Gorniak M']
source: PLoS One. 2015 Apr 22;10(4):e0122306. doi: 10.1371/journal.pone.0122306. eCollection 2015.

title: Species distribution modelling for conservation of an endangered endemic orchid.
authors: ['Wang HH', 'Wonkka CL', 'Treglia ML', 'Grant WE', 'Smeins FE', 'Rogers WE']
source: AoB Plants. 2015 Apr 21;7. pii: plv039. doi: 10.1093/aobpla/plv039.

title: Floral miniaturisation and autogamy in boreal-arctic plants are epitomised by Iceland's most r
authors: ['Bateman RM', 'Sramko G', 'Rudall PJ']
source: PeerJ. 2015 Apr 14;3:e894. doi: 10.7717/peerj.894. eCollection 2015.

title: Highly diversified fungi are associated with the achlorophyllous orchid Gastrodia flavilabella
authors: ['Liu T', 'Li CM', 'Han YL', 'Chiang TY', 'Chiang YC', 'Sung HM']
source: BMC Genomics. 2015 Mar 14;16:185. doi: 10.1186/s12864-015-1422-7.

title: Floral nectary anatomy and ultrastructure in mycoheterotrophic plant, Epipogium aphyllum Sw.
authors: ['Swieczkowska E', 'Kowalkowska AK']
source: ScientificWorldJournal. 2015;2015:201702. doi: 10.1155/2015/201702. Epub 2015 Mar 25.

title: The complete chloroplast genome sequence of Anoectochilus roxburghii.
authors: ['Yu CW', 'Lian Q', 'Wu KC', 'Yu SH', 'Xie LY', 'Wu ZJ']
source: Mitochondrial DNA. 2015 Apr 13:1-2.

title: Effects of droplet-vitrification cryopreservation based on physiological and antioxidant enzym
authors: ['Rahmah S', 'Ahmad Mubbarakh S', 'Soo Ping K', 'Subramaniam S']
source: ScientificWorldJournal. 2015;2015:961793. doi: 10.1155/2015/961793. Epub 2015 Mar 11.

title: Pollinator behaviour on a food-deceptive orchid Calypso bulbosa and coflowering species.
authors: ['Tuomi J', 'Lamsa J', 'Wannas L', 'Abeli T', 'Jakalaniemi A']
source: ScientificWorldJournal. 2015;2015:482161. doi: 10.1155/2015/482161. Epub 2015 Mar 12.

title: Reticulate evolution and sea-level fluctuations together drove species diversification of slip
authors: ['Guo YY', 'Luo YB', 'Liu ZJ', 'Wang XQ']
source: Mol Ecol. 2015 Jun;24(11):2838-55. doi: 10.1111/mec.13189. Epub 2015 May 7.

title: Crystal structure of 2-{[(naphthalen-1-yl)oxy]meth-yl}-5-(2,4,5-tri-fluoro-phen-yl)-1,3,4-oxa-
authors: ['Govindhan M', 'Subramanian K', 'Viswanathan V', 'Velmurugan D']
source: Acta Crystallogr E Crystallogr Commun. 2015 Feb 21;71(Pt 3):o190-1. doi: 10.1107/S20569890150

title: The effect of mealybug Pseudococcus longispinus (Targioni Tozzetti) infestation of different d
authors: ['Kot I', 'Kmiec K', 'Gorska-Drabik E', 'Golan K', 'Rubinowska K', 'Lagowska B']
source: Bull Entomol Res. 2015 Jun;105(3):373-80. doi: 10.1017/S000748531500022X. Epub 2015 Apr 1.

title: The Genome of Dendrobium officinale Illuminates the Biology of the Important Traditional Chine
authors: ['Yan L', 'Wang X', 'Liu H', 'Tian Y', 'Lian J', 'Yang R', 'Hao S', 'Wang X', 'Yang S', 'Li
source: Mol Plant. 2015 Jun;8(6):922-34. doi: 10.1016/j.molp.2014.12.011. Epub 2014 Dec 24.

title: Recurrent polymorphic mating type variation in Madagascan species (Orchidaceae) exemplifies a
authors: ['Gamisch A', 'Fischer GA', 'Comes HP']
source: Bot J Linn Soc. 2014 Jun;175(2):242-258. Epub 2014 May 20.
```

```
title: When stable-stage equilibrium is unlikely: integrating transient population dynamics improves
authors: ['Tremblay RL', 'Raventos J', 'Ackerman JD']
source: Ann Bot. 2015 Sep;116(3):381-90. doi: 10.1093/aob/mcv031. Epub 2015 Mar 26.

title: Antibiotic susceptibility pattern of Enterobacteriaceae and non-fermenter Gram-negative clinic
authors: ['Hariharan P', 'Bharani T', 'Franklyne JS', 'Biswas P', 'Solanki SS', 'Paul-Satyaseela M']
source: J Nat Sci Biol Med. 2015 Jan-Jun;6(1):198-201. doi: 10.4103/0976-9668.149121.

title: Pollination system and the effect of inflorescence size on fruit set in the deceptive orchid C
authors: ['Suetsugu K', 'Naito RS', 'Fukushima S', 'Kawakita A', 'Kato M']
source: J Plant Res. 2015 Jul;128(4):585-94. doi: 10.1007/s10265-015-0716-9. Epub 2015 Mar 24.

title: Polysaccharide hydrogel combined with mesenchymal stem cells promotes the healing of corneal a
authors: ['Ke Y', 'Wu Y', 'Cui X', 'Liu X', 'Yu M', 'Yang C', 'Li X']
source: PLoS One. 2015 Mar 19;10(3):e0119725. doi: 10.1371/journal.pone.0119725. eCollection 2015.

title: Ex situ germination as a method for seed viability assessment in a peatland orchid, Platanther
authors: ['Lemay MA', 'De Vriendt L', 'Pellerin S', 'Poulin M']
source: Am J Bot. 2015 Mar;102(3):390-5. doi: 10.3732/ajb.1400441. Epub 2015 Mar 1.

title: Preliminary findings on identification of mycorrhizal fungi from diverse orchids in the Centra
authors: ['Yokoya K', 'Zettler LW', 'Kendon JP', 'Bidartondo MI', 'Stice AL', 'Skarha S', 'Corey LL',
source: Mycorrhiza. 2015 Nov;25(8):611-25. doi: 10.1007/s00572-015-0635-6. Epub 2015 Mar 14.

title: Pollination biology in the dioecious orchid Catasetum uncatum: How does floral scent influence
authors: ['Milet-Pinheiro P', 'Navarro DM', 'Dotterl S', 'Carvalho AT', 'Pinto CE', 'Ayasse M', 'Schl
source: Phytochemistry. 2015 Aug;116:149-61. doi: 10.1016/j.phytochem.2015.02.027. Epub 2015 Mar 11.

title: The location and translocation of ndh genes of chloroplast origin in the Orchidaceae family.
authors: ['Lin CS', 'Chen JJ', 'Huang YT', 'Chan MT', 'Daniell H', 'Chang WJ', 'Hsu CT', 'Liao DC',
source: Sci Rep. 2015 Mar 12;5:9040. doi: 10.1038/srep09040.

title: Genetic structure is associated with phenotypic divergence in floral traits and reproductive i
authors: ['Leles B', 'Chaves AV', 'Russo P', 'Batista JA', 'Lovato MB']
source: PLoS One. 2015 Mar 10;10(3):e0120645. doi: 10.1371/journal.pone.0120645. eCollection 2015.

title: A molecular phylogeny of Aeridinae (Orchidaceae: Epidendroideae) inferred from multiple nuclea
authors: ['Zou LH', 'Huang JX', 'Zhang GQ', 'Liu ZJ', 'Zhuang XY']
source: Mol Phylogenet Evol. 2015 Apr;85:247-54. doi: 10.1016/j.ympev.2015.02.014. Epub 2015 Feb 26.

title: Corrigendum: The genome sequence of the orchid Phalaenopsis equestris.
authors: ['Cai J', 'Liu X', 'Vanneste K', 'Proost S', 'Tsai WC', 'Liu KW', 'Chen LJ', 'He Y', 'Xu Q',
source: Nat Genet. 2015 Mar;47(3):304. doi: 10.1038/ng0315-304a.

title: Convergent losses of decay mechanisms and rapid turnover of symbiosis genes in mycorrhizal mut
authors: ['Kohler A', 'Kuo A', 'Nagy LG', 'Morin E', 'Barry KW', 'Buscot F', 'Canback B', 'Choi C',
source: Nat Genet. 2015 Apr;47(4):410-5. doi: 10.1038/ng.3223. Epub 2015 Feb 23.

title: Chemical and morphological filters in a specialized floral mimicry system.
authors: ['Martos F', 'Cariou ML', 'Pailler T', 'Fournel J', 'Bytebier B', 'Johnson SD']
source: New Phytol. 2015 Jul;207(1):225-34. doi: 10.1111/nph.13350. Epub 2015 Feb 20.

title: Modeling the two-locus architecture of divergent pollinator adaptation: how variation in SAD p
authors: ['Xu S', 'Schluter PM']
source: Ecol Evol. 2015 Jan;5(2):493-502. doi: 10.1002/ece3.1378. Epub 2015 Jan 4.

title: Pollination ecology of two species of Elleanthus (Orchidaceae): novel mechanisms and underlyin
authors: ['Nunes CE', 'Amorim FW', 'Mayer JL', 'Sazima M']
source: Plant Biol (Stuttg). 2015 Feb 11. doi: 10.1111/plb.12312.
```

```
title: Sequential decarboxylative azide-alkyne cycloaddition and dehydrogenative coupling reactions:
authors: ['Bharathimohan K', 'Ponpandian T', 'Ahamed AJ', 'Bhuvanesh N']
source: Beilstein J Org Chem. 2014 Dec 17;10:3031-7. doi: 10.3762/bjoc.10.321. eCollection 2014.

title: Are tetraploids more successful? Floral signals, reproductive success and floral isolation in
authors: ['Gross K', 'Schiestl FP']
source: Ann Bot. 2015 Feb;115(2):263-73. doi: 10.1093/aob/mcu244.

title: Setting the pace of life: membrane composition of flight muscle varies with metabolic rate of
authors: ['Rodriguez E', 'Weber JM', 'Page B', 'Roubik DW', 'Suarez RK', 'Darveau CA']
source: Proc Biol Sci. 2015 Mar 7;282(1802). pii: 20142232. doi: 10.1098/rspb.2014.2232.

title: Mycorrhizal ecology and evolution: the past, the present, and the future.
authors: ['van der Heijden MG', 'Martin FM', 'Selosse MA', 'Sanders IR']
source: New Phytol. 2015 Mar;205(4):1406-23. doi: 10.1111/nph.13288. Epub 2015 Feb 2.

title: The orchid-bee fauna (Hymenoptera: Apidae) of a forest remnant in the southern portion of the
authors: ['Santos Junior JE', 'Ferrari RR', 'Nemesio A']
source: Braz J Biol. 2014 Aug;74(3 Suppl 1):S184-90. doi: 10.1590/1519-6984.25712.

title: Is the "Centro de Endemismo Pernambuco" a biodiversity hotspot for orchid bees?
authors: ['Nemesio A', 'Santos Junior JE']
source: Braz J Biol. 2014 Aug;74(3 Suppl 1):S78-92. doi: 10.1590/1519-6984.26412.

title: Sampling a biodiversity hotspot: the orchid-bee fauna (Hymenoptera: Apidae) of Tarapoto, nort
authors: ['Nemesio A', 'Rasmussen C']
source: Braz J Biol. 2014 Aug;74(3 Suppl 1):S33-44. doi: 10.1590/1519-6984.20412.

title: Mismatch in the distribution of floral ecotypes and pollinators: insights into the evolution o
authors: ['Phillips RD', 'Bohman B', 'Anthony JM', 'Krauss SL', 'Dixon KW', 'Peakall R']
source: J Evol Biol. 2015 Mar;28(3):601-12. doi: 10.1111/jeb.12593. Epub 2015 Feb 20.

title: Mycorrhizal networks and coexistence in species-rich orchid communities.
authors: ['Jacquemyn H', 'Brys R', 'Waud M', 'Busschaert P', 'Lievens B']
source: New Phytol. 2015 May;206(3):1127-34. doi: 10.1111/nph.13281. Epub 2015 Jan 23.

title: Two widespread green Neottia species (Orchidaceae) show mycorrhizal preference for Sebacinale
authors: ['Tesitelova T', 'Kotilinek M', 'Jersakova J', 'Joly FX', 'Kosnar J', 'Tatarenko I', 'Selos
source: Mol Ecol. 2015 Mar;24(5):1122-34. doi: 10.1111/mec.13088. Epub 2015 Feb 16.

title: Genetic stability and phytochemical analysis of the in vitro regenerated plants of Dendrobium
authors: ['Bhattacharyya P', 'Kumaria S', 'Diengdoh R', 'Tandon P']
source: Meta Gene. 2014 Jul 15;2:489-504. doi: 10.1016/j.mgene.2014.06.003. eCollection 2014 Dec.

title: Complete chloroplast genome of the orchid Cattleya crispata (Orchidaceae:Laeliinae), a Neotro
authors: ['da Rocha Perini V', 'Leles B', 'Furtado C', 'Prosdocimi F']
source: Mitochondrial DNA. 2015 Jan 20:1-3.

title: RNA/DNA co-analysis from human skin and contact traces--results of a sixth collaborative EDNA
authors: ['Haas C', 'Hanson E', 'Banemann R', 'Bento AM', 'Berti A', 'Carracedo A', 'Courts C', 'De
source: Forensic Sci Int Genet. 2015 May;16:139-47. doi: 10.1016/j.fsigen.2015.01.002. Epub 2015 Jan

title: Ethylene and pollination decrease transcript abundance of an ethylene receptor gene in Dendro
authors: ['Thongkum M', 'Burns P', 'Bhunchoth A', 'Warin N', 'Chatchawankanphanich O', 'van Doorn WG
source: J Plant Physiol. 2015 Mar 15;176:96-100. doi: 10.1016/j.jplph.2014.12.008. Epub 2014 Dec 18.

title: Pollen limitation and the contribution of autonomous selfing to fruit and seed set in a rewar
authors: ['Jacquemyn H', 'Brys R']
```

```
source: Am J Bot. 2015 Jan;102(1):67-72. doi: 10.3732/ajb.1400449. Epub 2014 Dec 22.

title: In vitro propagation of Paphiopedilum orchids.
authors: ['Zeng S', 'Huang W', 'Wu K', 'Zhang J', 'Teixeira da Silva JA', 'Duan J']
source: Crit Rev Biotechnol. 2015 Sep 11:1-14.

title: Vanillin-bioconversion and bioengineering of the most popular plant flavor and its de novo bio
authors: ['Gallage NJ', 'Moller BL']
source: Mol Plant. 2015 Jan;8(1):40-57. doi: 10.1016/j.molp.2014.11.008. Epub 2014 Dec 11.

title: Crystallographic investigations of select cathinones: emerging illicit street drugs known as
authors: ['Wood MR', 'Lalancette RA', 'Bernal I']
source: Acta Crystallogr C Struct Chem. 2015 Jan;71(Pt 1):32-8. doi: 10.1107/S2053229614025637. Epub

title: A genome to unveil the mysteries of orchids.
authors: ['Albert VA', 'Carretero-Paulet L']
source: Nat Genet. 2015 Jan;47(1):3-4. doi: 10.1038/ng.3179.

title: Temporal patterns of orchid mycorrhizal fungi in meadows and forests as revealed by 454 pyrose
authors: ['Oja J', 'Kohout P', 'Tedersoo L', 'Kull T', 'Koljalg U']
source: New Phytol. 2015 Mar;205(4):1608-18. doi: 10.1111/nph.13223. Epub 2014 Dec 24.

title: Interests shape how adolescents pay attention: the interaction of motivation and top-down atte
authors: ['Banerjee S', 'Frey HP', 'Molholm S', 'Foxe JJ']
source: Eur J Neurosci. 2015 Mar;41(6):818-34. doi: 10.1111/ejn.12810. Epub 2014 Dec 26.

title: Are carbon and nitrogen exchange between fungi and the orchid Goodyera repens affected by irra
authors: ['Liebel HT', 'Bidartondo MI', 'Gebauer G']
source: Ann Bot. 2015 Feb;115(2):251-61. doi: 10.1093/aob/mcu240. Epub 2014 Dec 22.

title: Taxonomic notes and distribution extension of Durga Das's leaf-nosed bat Hipposiderosdurgadas
authors: ['Kaur H', 'Chelmala S', 'Srinivasulu B', 'Shah TA', 'Devender G', 'Srinivasulu A']
source: Biodivers Data J. 2014 Nov 20;(2):e4127. doi: 10.3897/BDJ.2.e4127. eCollection 2014.

title: Characterization and expression analysis of somatic embryogenesis receptor-like kinase genes
authors: ['Huang YW', 'Tsai YJ', 'Chen FC']
source: Genet Mol Res. 2014 Dec 18;13(4):10690-703. doi: 10.4238/2014.December.18.11.

title: [Effects of different fungi on symbiotic seed germination of two Dendrobium species].
authors: ['Zi XM', 'Gao JY']
source: Zhongguo Zhong Yao Za Zhi. 2014 Sep;39(17):3238-44.

title: Conditioned Medium Reconditions Hippocampal Neurons against Kainic Acid Induced Excitotoxicity
authors: ['Bevinahal PK', 'Venugopal C', 'Yencharla HC', 'Chandanala S', 'Trichur RR', 'Talakad SN',
source: J Toxicol. 2014;2014:194967. doi: 10.1155/2014/194967. Epub 2014 Nov 23.

title: Histone acetylation accompanied with promoter sequences displaying differential expression pro
authors: ['Hsu CC', 'Wu PS', 'Chen TC', 'Yu CW', 'Tsai WC', 'Wu K', 'Wu WL', 'Chen WH', 'Chen HH']
source: PLoS One. 2014 Dec 11;9(12):e106033. doi: 10.1371/journal.pone.0106033. eCollection 2014.

title: The treatment of displaced intra-articular distal radius fractures in elderly patients.
authors: ['Bartl C', 'Stengel D', 'Bruckner T', 'Gebhard F']
source: Dtsch Arztebl Int. 2014 Nov 14;111(46):779-87. doi: 10.3238/arztebl.2014.0779.

title: "Double-trick" visual and chemical mimicry by the juvenile orchid mantis hymenopus coronatus
authors: ['Mizuno T', 'Yamaguchi S', 'Yamamoto I', 'Yamaoka R', 'Akino T']
source: Zoolog Sci. 2014 Dec;31(12):795-801. doi: 10.2108/zs140126.

title: Role of auxin in orchid development.
```

```
authors: ['Novak SD', 'Luna LJ', 'Gamage RN']
source: Plant Signal Behav. 2014;9(10):e972277. doi: 10.4161/psb.32169.

title: Orchid mating: the anther steps onto the stigma.
authors: ['Chen LJ', 'Liu ZJ']
source: Plant Signal Behav. 2014;9(11):e976484. doi: 10.4161/15592324.2014.976484.

title: Plant and fungal gene expression in mycorrhizal protocorms of the orchid Serapias vomeracea co
authors: ['Balestrini R', 'Nerva L', 'Sillo F', 'Girlanda M', 'Perotto S']
source: Plant Signal Behav. 2014;9(11):e977707. doi: 10.4161/15592324.2014.977707.

title: Development of Cymbidium ensifolium genic-SSR markers and their utility in genetic diversity a
authors: ['Li X', 'Jin F', 'Jin L', 'Jackson A', 'Huang C', 'Li K', 'Shu X']
source: BMC Genet. 2014 Dec 5;15:124. doi: 10.1186/s12863-014-0124-5.

title: Antinociceptive and cytotoxic activities of an epiphytic medicinal orchid: Vanda tessellata Ro
authors: ['Chowdhury MA', 'Rahman MM', 'Chowdhury MR', 'Uddin MJ', 'Sayeed MA', 'Hossain MA']
source: BMC Complement Altern Med. 2014 Dec 3;14:464. doi: 10.1186/1472-6882-14-464.

title: Climate change: bees and orchids lose touch.
authors: ['Willmer P']
source: Curr Biol. 2014 Dec 1;24(23):R1133-5. doi: 10.1016/j.cub.2014.10.061. Epub 2014 Dec 1.

title: Mudskipper genomes provide insights into the terrestrial adaptation of amphibious fishes.
authors: ['You X', 'Bian C', 'Zan Q', 'Xu X', 'Liu X', 'Chen J', 'Wang J', 'Qiu Y', 'Li W', 'Zhang X
source: Nat Commun. 2014 Dec 2;5:5594. doi: 10.1038/ncomms6594.

title: Traditional uses of medicinal plants in gastrointestinal disorders in Nepal.
authors: ['Rokaya MB', 'Uprety Y', 'Poudel RC', 'Timsina B', 'Munzbergova Z', 'Asselin H', 'Tiwari A
source: J Ethnopharmacol. 2014 Dec 2;158 Pt A:221-9. doi: 10.1016/j.jep.2014.10.014. Epub 2014 Oct 18

title: Potential disruption of pollination in a sexually deceptive orchid by climatic change.
authors: ['Robbirt KM', 'Roberts DL', 'Hutchings MJ', 'Davy AJ']
source: Curr Biol. 2014 Dec 1;24(23):2845-9. doi: 10.1016/j.cub.2014.10.033. Epub 2014 Nov 6.

title: CLL2-1, a chemical derivative of orchid 1,4-phenanthrenequinones, inhibits human platelet aggr
authors: ['Liao CY', 'Lee CL', 'Wang HC', 'Liang SS', 'Kung PH', 'Wu YC', 'Chang FR', 'Wu CC']
source: Free Radic Biol Med. 2015 Jan;78:101-10. doi: 10.1016/j.freeradbiomed.2014.10.512. Epub 2014

title: Individualizing hospital care for children and young people with learning disabilities: it's t
authors: ['Oulton K', 'Sell D', 'Kerry S', 'Gibson F']
source: J Pediatr Nurs. 2015 Jan-Feb;30(1):78-86. doi: 10.1016/j.pedn.2014.10.006. Epub 2014 Oct 23.

title: Ethanolic extract of Coelogyne cristata Lindley (Orchidaceae) and its compound coelogin promot
authors: ['Sharma C', 'Mansoori MN', 'Dixit M', 'Shukla P', 'Kumari T', 'Bhandari SP', 'Narender T',
source: Phytomedicine. 2014 Oct 15;21(12):1702-7. doi: 10.1016/j.phymed.2014.08.008. Epub 2014 Sep 16

title: Virus resistance in orchids.
authors: ['Koh KW', 'Lu HC', 'Chan MT']
source: Plant Sci. 2014 Nov;228:26-38. doi: 10.1016/j.plantsci.2014.04.015. Epub 2014 Apr 28.

title: In vitro regeneration and ploidy level analysis of Eulophia ochreata Lindl.
authors: ['Shriram V', 'Nanekar V', 'Kumar V', 'Kavi Kishor PB']
source: Indian J Exp Biol. 2014 Nov;52(11):1112-21.

title: A novel animal model of metabolic syndrome with non-alcoholic fatty liver disease and skin in
authors: ['Kulkarni NM', 'Jaji MS', 'Shetty P', 'Kurhe YV', 'Chaudhary S', 'Vijaykant G', 'Raghul J',
source: Pharm Biol. 2015 Aug;53(8):1110-7. doi: 10.3109/13880209.2014.960944. Epub 2014 Nov 28.
```

```
title: Identification and Molecular Characterization of Nuclear Citrus leprosis virus, a Member of th
authors: ['Roy A', 'Stone AL', 'Shao J', 'Otero-Colina G', 'Wei G', 'Choudhary N', 'Achor D', 'Levy L
source: Phytopathology. 2015 Apr;105(4):564-75. doi: 10.1094/PHYTO-09-14-0245-R.

title: Raising the sugar content--orchid bees overcome the constraints of suction feeding through man
authors: ['Pokorny T', 'Lunau K', 'Eltz T']
source: PLoS One. 2014 Nov 25;9(11):e113823. doi: 10.1371/journal.pone.0113823. eCollection 2014.

title: Using ecological niche models and niche analyses to understand speciation patterns: the case o
authors: ['Silva DP', 'Vilela B', 'De Marco P Jr', 'Nemesio A']
source: PLoS One. 2014 Nov 25;9(11):e113246. doi: 10.1371/journal.pone.0113246. eCollection 2014.

title: Rapid cytolysis of Mycobacterium tuberculosis by faropenem, an orally bioavailable beta-lactam
authors: ['Dhar N', 'Dubee V', 'Ballell L', 'Cuinet G', 'Hugonnet JE', 'Signorino-Gelo F', 'Barros D
source: Antimicrob Agents Chemother. 2015 Feb;59(2):1308-19. doi: 10.1128/AAC.03461-14. Epub 2014 Nov

title: The genome sequence of the orchid Phalaenopsis equestris.
authors: ['Cai J', 'Liu X', 'Vanneste K', 'Proost S', 'Tsai WC', 'Liu KW', 'Chen LJ', 'He Y', 'Xu Q',
source: Nat Genet. 2015 Jan;47(1):65-72. doi: 10.1038/ng.3149. Epub 2014 Nov 24.

title: Establishment of an efficient in vitro regeneration protocol for rapid and mass propagation of
authors: ['Nongdam P', 'Tikendra L']
source: ScientificWorldJournal. 2014;2014:740150. doi: 10.1155/2014/740150. Epub 2014 Oct 20.

title: Characterization of arbuscular mycorrhizal fungus communities of Aquilaria crassna and Tectona
authors: ['Chaiyasen A', 'Young JP', 'Teaumroong N', 'Gavinlertvatana P', 'Lumyong S']
source: PLoS One. 2014 Nov 14;9(11):e112591. doi: 10.1371/journal.pone.0112591. eCollection 2014.

title: Floral isolation is the major reproductive barrier between a pair of rewarding orchid sister s
authors: ['Sun M', 'Schluter PM', 'Gross K', 'Schiestl FP']
source: J Evol Biol. 2015 Jan;28(1):117-29. doi: 10.1111/jeb.12544. Epub 2015 Jan 5.

title: Temporal variation in mycorrhizal diversity and carbon and nitrogen stable isotope abundance i
authors: ['Ercole E', 'Adamo M', 'Rodda M', 'Gebauer G', 'Girlanda M', 'Perotto S']
source: New Phytol. 2015 Feb;205(3):1308-19. doi: 10.1111/nph.13109. Epub 2014 Nov 10.

title: A deep transcriptomic analysis of pod development in the vanilla orchid (Vanilla planifolia).
authors: ['Rao X', 'Krom N', 'Tang Y', 'Widiez T', 'Havkin-Frenkel D', 'Belanger FC', 'Dixon RA', 'Ch
source: BMC Genomics. 2014 Nov 7;15:964. doi: 10.1186/1471-2164-15-964.

title: The life of phi: the development of phi thickenings in roots of the orchids of the genus Milto
authors: ['Idris NA', 'Collings DA']
source: Planta. 2015 Feb;241(2):489-506. doi: 10.1007/s00425-014-2194-z. Epub 2014 Nov 7.

title: Genic rather than genome-wide differences between sexually deceptive Ophrys orchids with diffe
authors: ['Sedeek KE', 'Scopece G', 'Staedler YM', 'Schonenberger J', 'Cozzolino S', 'Schiestl FP',
source: Mol Ecol. 2014 Dec;23(24):6192-205. doi: 10.1111/mec.12992. Epub 2014 Nov 27.

title: Comparative proteomic analysis of labellum and inner lateral petals in Cymbidium ensifolium fl
authors: ['Li X', 'Xu W', 'Chowdhury MR', 'Jin F']
source: Int J Mol Sci. 2014 Oct 31;15(11):19877-97. doi: 10.3390/ijms151119877.

title: In vitro propagation and reintroduction of the endangered Renanthera imschootiana Rolfe.
authors: ['Wu K', 'Zeng S', 'Lin D', 'Teixeira da Silva JA', 'Bu Z', 'Zhang J', 'Duan J']
source: PLoS One. 2014 Oct 28;9(10):e110033. doi: 10.1371/journal.pone.0110033. eCollection 2014.

title: The velamen protects photosynthetic orchid roots against UV-B damage, and a large dated phylog
authors: ['Chomicki G', 'Bidel LP', 'Ming F', 'Coiro M', 'Zhang X', 'Wang Y', 'Baissac Y', 'Jay-Allem
source: New Phytol. 2015 Feb;205(3):1330-41. doi: 10.1111/nph.13106. Epub 2014 Oct 23.
```

```
title: Prolonged exposure to elevated temperature induces floral transition via up-regulation of cyto
authors: ['Chin DC', 'Shen CH', 'SenthilKumar R', 'Yeh KW']
source: Plant Cell Physiol. 2014 Dec;55(12):2164-76. doi: 10.1093/pcp/pcu146. Epub 2014 Oct 14.

title: Mycorrhizal fungal diversity and community composition in a lithophytic and epiphytic orchid.
authors: ['Xing X', 'Gai X', 'Liu Q', 'Hart MM', 'Guo S']
source: Mycorrhiza. 2015 May;25(4):289-96. doi: 10.1007/s00572-014-0612-5. Epub 2014 Oct 17.

title: Topical atorvastatin ameliorates 12-O-tetradecanoylphorbol-13-acetate induced skin inflammatio
authors: ['Kulkarni NM', 'Muley MM', 'Jaji MS', 'Vijaykanth G', 'Raghul J', 'Reddy NK', 'Vishwakarma
source: Arch Pharm Res. 2015 Jun;38(6):1238-47. doi: 10.1007/s12272-014-0496-0. Epub 2014 Oct 14.

title: Crystal structure of 3-methyl-2,6-bis-(4-methyl-1,3-thia-zol-5-yl)piperidin-4-one.
authors: ['Manimaran A', 'Sethusankar K', 'Ganesan S', 'Ananthan S']
source: Acta Crystallogr Sect E Struct Rep Online. 2014 Aug 30;70(Pt 9):o1055. doi: 10.1107/S1600536

title: Molecular phylogeny and evolutionary history of the Eurasiatic orchid genus Himantoglossum s.l
authors: ['Sramko G', 'Attila MV', 'Hawkins JA', 'Bateman RM']
source: Ann Bot. 2014 Dec;114(8):1609-26. doi: 10.1093/aob/mcu179. Epub 2014 Oct 7.

title: A flavonoid isolated from Streptomyces sp. (ERINLG-4) induces apoptosis in human lung cancer A
authors: ['Balachandran C', 'Sangeetha B', 'Duraipandiyan V', 'Raj MK', 'Ignacimuthu S', 'Al-Dhabi NA
source: Chem Biol Interact. 2014 Dec 5;224:24-35. doi: 10.1016/j.cbi.2014.09.019. Epub 2014 Oct 5.

title: The folklore medicinal orchids of Sikkim.
authors: ['Panda AK', 'Mandal D']
source: Anc Sci Life. 2013 Oct;33(2):92-6. doi: 10.4103/0257-7941.139043.

title: Effect of cryopreservation on in vitro seed germination and protocorm growth of Mediterranean
authors: ['Pirondini A', 'Sgarbi E']
source: Cryo Letters. 2014 Jul-Aug;35(4):327-35.

title: Do chlorophyllous orchids heterotrophically use mycorrhizal fungal carbon?
authors: ['Selosse MA', 'Martos F']
source: Trends Plant Sci. 2014 Nov;19(11):683-5.

title: Vanillin - Bioconversion and Bioengineering of the most popular plant flavour and its de novo
authors: ['Gallage NJ', 'Moeller BL']
source: Mol Plant. 2014 Sep 30. pii: ssu105.

title: Authenticity and traceability of vanilla flavors by analysis of stable isotopes of carbon and
authors: ['Hansen AM', 'Fromberg A', 'Frandsen HL']
source: J Agric Food Chem. 2014 Oct 22;62(42):10326-31. doi: 10.1021/jf503055k. Epub 2014 Oct 13.

title: Are winter-active species vulnerable to climate warming? A case study with the wintergreen ter
authors: ['Marchin RM', 'Dunn RR', 'Hoffmann WA']
source: Oecologia. 2014 Dec;176(4):1161-72. doi: 10.1007/s00442-014-3074-8. Epub 2014 Sep 26.

title: Antimicrobial compounds from leaf extracts of Jatropha curcas, Psidium guajava, and Andrograph
authors: ['Rahman MM', 'Ahmad SH', 'Mohamed MT', 'Ab Rahman MZ']
source: ScientificWorldJournal. 2014;2014:635240. doi: 10.1155/2014/635240. Epub 2014 Aug 26.

title: Eugenol synthase genes in floral scent variation in Gymnadenia species.
authors: ['Gupta AK', 'Schauvinhold I', 'Pichersky E', 'Schiestl FP']
source: Funct Integr Genomics. 2014 Dec;14(4):779-88. doi: 10.1007/s10142-014-0397-9. Epub 2014 Sep 2

title: Bavituximab plus paclitaxel and carboplatin for the treatment of advanced non-small-cell lung
authors: ['Digumarti R', 'Bapsy PP', 'Suresh AV', 'Bhattacharyya GS', 'Dasappa L', 'Shan JS', 'Gerbe
```

```
source: Lung Cancer. 2014 Nov;86(2):231-6. doi: 10.1016/j.lungcan.2014.08.010. Epub 2014 Aug 24.

title: Isolation and differential expression of a novel MAP kinase gene DoMPK4 in Dendrobium officina
authors: ['Zhang G', 'Li YM', 'Hu BX', 'Zhang DW', 'Guo SX']
source: Yao Xue Xue Bao. 2014 Jul;49(7):1076-83.

title: RNA interference-based gene silencing of phytoene synthase impairs growth, carotenoids, and pl
authors: ['Liu JX', 'Chiou CY', 'Shen CH', 'Chen PJ', 'Liu YC', 'Jian CD', 'Shen XL', 'Shen FQ', 'Yeh
source: Springerplus. 2014 Aug 28;3:478. doi: 10.1186/2193-1801-3-478. eCollection 2014.

title: Development of phylogenetic markers for Sebacina (Sebacinaceae) mycorrhizal fungi associated w
authors: ['Ruibal MP', 'Peakall R', 'Foret S', 'Linde CC']
source: Appl Plant Sci. 2014 Jun 4;2(6). pii: apps.1400015. doi: 10.3732/apps.1400015. eCollection 20

title: Characterization of 13 microsatellite markers for Diuris basaltica (Orchidaceae) and related s
authors: ['Ahrens CW', 'James EA']
source: Appl Plant Sci. 2014 Jan 7;2(1). pii: apps.1300069. doi: 10.3732/apps.1300069. eCollection 20

title: Pregnancy influences the plasma pharmacokinetics but not the cerebrospinal fluid pharmacokinet
authors: ['Mahat MY', 'Thippeswamy BS', 'Khan FR', 'Edunuri R', 'Nidhyanandan S', 'Chaudhary S']
source: Eur J Pharm Sci. 2014 Dec 18;65:38-44. doi: 10.1016/j.ejps.2014.08.012. Epub 2014 Sep 6.

title: Temporal and spatial regulation of anthocyanin biosynthesis provide diverse flower colour inte
authors: ['Wang L', 'Albert NW', 'Zhang H', 'Arathoon S', 'Boase MR', 'Ngo H', 'Schwinn KE', 'Davies
source: Planta. 2014 Nov;240(5):983-1002. doi: 10.1007/s00425-014-2152-9. Epub 2014 Sep 3.

title: Deep sequencing-based comparative transcriptional profiles of Cymbidium hybridum roots in resp
authors: ['Zhao X', 'Zhang J', 'Chen C', 'Yang J', 'Zhu H', 'Liu M', 'Lv F']
source: BMC Genomics. 2014 Aug 31;15:747. doi: 10.1186/1471-2164-15-747.

title: A microfluidic system integrated with buried optical fibers for detection of Phalaenopsis orch
authors: ['Lin CL', 'Chang WH', 'Wang CH', 'Lee CH', 'Chen TY', 'Jan FJ', 'Lee GB']
source: Biosens Bioelectron. 2015 Jan 15;63:572-9. doi: 10.1016/j.bios.2014.08.013. Epub 2014 Aug 17

title: Role of Auxin in orchid development.
authors: ['Darling-Novak S', 'Luna LJ', 'Gamage RN']
source: Plant Signal Behav. 2014 Aug 25;9. pii: e32169.

title: Predicting progression of Alzheimer's disease using ordinal regression.
authors: ['Doyle OM', 'Westman E', 'Marquand AF', 'Mecocci P', 'Vellas B', 'Tsolaki M', 'Kloszewska I
source: PLoS One. 2014 Aug 20;9(8):e105542. doi: 10.1371/journal.pone.0105542. eCollection 2014.

title: Identity and specificity of Rhizoctonia-like fungi from different populations of Liparis japon
authors: ['Ding R', 'Chen XH', 'Zhang LJ', 'Yu XD', 'Qu B', 'Duan R', 'Xu YF']
source: PLoS One. 2014 Aug 20;9(8):e105573. doi: 10.1371/journal.pone.0105573. eCollection 2014.

title: Labellar anatomy and secretion in Bulbophyllum Thouars (Orchidaceae: Bulbophyllinae) sect. Rad
authors: ['Davies KL', 'Stpiczynska M']
source: Ann Bot. 2014 Oct;114(5):889-911. doi: 10.1093/aob/mcu153. Epub 2014 Aug 13.

title: Molecular characterization and functional analysis of a Flowering locus T homolog gene from a
authors: ['Li DM', 'L FB', 'Zhu GF', 'Sun YB', 'Liu HL', 'Liu JW', 'Wang Z']
source: Genet Mol Res. 2014 Aug 7;13(3):5982-94. doi: 10.4238/2014.August.7.14.

title: Composition and conservation of Orchidaceae on an inselberg in the Brazilian Atlantic Forest a
authors: ['Pessanha AS', 'Menini Neto L', 'Forzza RC', 'Nascimento MT']
source: Rev Biol Trop. 2014 Jun;62(2):829-41.

title: The complete chloroplast genome of Phalaenopsis "Tiny Star".
```

```
authors: ['Kim GB', 'Kwon Y', 'Yu HJ', 'Lim KB', 'Seo JH', 'Mun JH']
source: Mitochondrial DNA. 2016 Mar;27(2):1300-2. doi: 10.3109/19401736.2014.945566. Epub 2014 Aug 5.

title: The cultural and ecological impacts of aboriginal tourism: a case study on Taiwan's Tao tribe.
authors: ['Liu TM', 'Lu DJ']
source: Springerplus. 2014 Jul 8;3:347. doi: 10.1186/2193-1801-3-347. eCollection 2014.

title: Verifying likelihoods for low template DNA profiles using multiple replicates.
authors: ['Steele CD', 'Greenhalgh M', 'Balding DJ']
source: Forensic Sci Int Genet. 2014 Nov;13:82-9. doi: 10.1016/j.fsigen.2014.06.018. Epub 2014 Jul 10

title: Development of microsatellite markers of vandaceous orchids for species and variety identifica
authors: ['Peyachoknagul S', 'Nettuwakul C', 'Phuekvilai P', 'Wannapinpong S', 'Srikulnath K']
source: Genet Mol Res. 2014 Jul 24;13(3):5441-5. doi: 10.4238/2014.July.24.23.

title: Bayesian estimates of transition probabilities in seven small lithophytic orchid populations:
authors: ['Tremblay RL', 'McCarthy MA']
source: PLoS One. 2014 Jul 28;9(7):e102859. doi: 10.1371/journal.pone.0102859. eCollection 2014.

title: Occurrence of Bacillus amyloliquefaciens as a systemic endophyte of vanilla orchids.
authors: ['White JF Jr', 'Torres MS', 'Sullivan RF', 'Jabbour RE', 'Chen Q', 'Tadych M', 'Irizarry I
source: Microsc Res Tech. 2014 Nov;77(11):874-85. doi: 10.1002/jemt.22410. Epub 2014 Jul 25.

title: The orchid-bee faunas (Hymenoptera: Apidae) of "Reserva Ecologica Michelin", "RPPN Serra Bonit
authors: ['Nemesio A']
source: Braz J Biol. 2014 Feb;74(1):16-22.

title: The corbiculate bees arose from New World oil-collecting bees: implications for the origin of
authors: ['Martins AC', 'Melo GA', 'Renner SS']
source: Mol Phylogenet Evol. 2014 Nov;80:88-94. doi: 10.1016/j.ympev.2014.07.003. Epub 2014 Jul 15.

title: Evaluation of the predacious mite Hemicheyletia wellsina (Acari: Cheyletidae) as a predator of
authors: ['Ray HA', 'Hoy MA']
source: Exp Appl Acarol. 2014 Nov;64(3):287-98. doi: 10.1007/s10493-014-9833-8. Epub 2014 Jul 18.

title: De novo transcriptome assembly from inflorescence of Orchis italica: analysis of coding and no
authors: ['De Paolo S', 'Salvemini M', 'Gaudio L', 'Aceto S']
source: PLoS One. 2014 Jul 15;9(7):e102155. doi: 10.1371/journal.pone.0102155. eCollection 2014.

title: Desiccation tolerance, longevity and seed-siring ability of entomophilous pollen from UK nativ
authors: ['Marks TR', 'Seaton PT', 'Pritchard HW']
source: Ann Bot. 2014 Sep;114(3):561-9. doi: 10.1093/aob/mcu139. Epub 2014 Jul 8.

title: High levels of effective long-distance dispersal may blur ecotypic divergence in a rare terres
authors: ['Vanden Broeck A', 'Van Landuyt W', 'Cox K', 'De Bruyn L', 'Gyselings R', 'Oostermeijer G',
source: BMC Ecol. 2014 Jul 7;14:20. doi: 10.1186/1472-6785-14-20.

title: Conservation genetics of an endangered lady's slipper orchid: Cypripedium japonicum in China.
authors: ['Qian X', 'Li QJ', 'Liu F', 'Gong MJ', 'Wang CX', 'Tian M']
source: Int J Mol Sci. 2014 Jun 30;15(7):11578-96. doi: 10.3390/ijms150711578.

title: Multiplex RT-PCR detection of three common viruses infecting orchids.
authors: ['Ali RN', 'Dann AL', 'Cross PA', 'Wilson CR']
source: Arch Virol. 2014 Nov;159(11):3095-9. doi: 10.1007/s00705-014-2161-9. Epub 2014 Jul 1.

title: Agrobacterium-mediated transformation of the recalcitrant Vanda Kasem's Delight orchid with hi
authors: ['Gnasekaran P', 'Antony JJ', 'Uddain J', 'Subramaniam S']
source: ScientificWorldJournal. 2014;2014:583934. doi: 10.1155/2014/583934. Epub 2014 Apr 8.
```

```
title: Cold response in Phalaenopsis aphrodite and characterization of PaCBF1 and PaICE1.
authors: ['Peng PH', 'Lin CH', 'Tsai HW', 'Lin TY']
source: Plant Cell Physiol. 2014 Sep;55(9):1623-35. doi: 10.1093/pcp/pcu093. Epub 2014 Jun 27.

title: Volatile fingerprint of italian populations of orchids using solid phase microextraction and g
authors: ['Manzo A', 'Panseri S', 'Vagge I', 'Giorgi A']
source: Molecules. 2014 Jun 11;19(6):7913-36. doi: 10.3390/molecules19067913.

title: [Molecular characterization of a HMG-CoA reductase gene from a rare and endangered medicinal p
authors: ['Zhang L', 'Wang JT', 'Zhang DW', 'Zhang G', 'Guo SX']
source: Yao Xue Xue Bao. 2014 Mar;49(3):411-8.

title: Antitumor activity of ethanolic extract of Dendrobium formosum in T-cell lymphoma: an in vitro
authors: ['Prasad R', 'Koch B']
source: Biomed Res Int. 2014;2014:753451. doi: 10.1155/2014/753451. Epub 2014 May 18.

title: New insight into the regulation of floral morphogenesis.
authors: ['Tsai WC', 'Pan ZJ', 'Su YY', 'Liu ZJ']
source: Int Rev Cell Mol Biol. 2014;311:157-82. doi: 10.1016/B978-0-12-800179-0.00003-9.

title: Effects of pollination limitation and seed predation on female reproductive success of a decep
authors: ['Walsh RP', 'Arnold PM', 'Michaels HJ']
source: AoB Plants. 2014 Jun 9;6. pii: plu031. doi: 10.1093/aobpla/plu031.

title: Multiple isoforms of phosphoenolpyruvate carboxylase in the Orchidaceae (subtribe Oncidiinae):
authors: ['Silvera K', 'Winter K', 'Rodriguez BL', 'Albion RL', 'Cushman JC']
source: J Exp Bot. 2014 Jul;65(13):3623-36. doi: 10.1093/jxb/eru234. Epub 2014 Jun 9.

title: Comparative chloroplast genomes of photosynthetic orchids: insights into evolution of the Orch
authors: ['Luo J', 'Hou BW', 'Niu ZT', 'Liu W', 'Xue QY', 'Ding XY']
source: PLoS One. 2014 Jun 9;9(6):e99016. doi: 10.1371/journal.pone.0099016. eCollection 2014.

title: Speciation via floral heterochrony and presumed mycorrhizal host switching of endemic butterfl
authors: ['Bateman RM', 'Rudall PJ', 'Bidartondo MI', 'Cozzolino S', 'Tranchida-Lombardo V', 'Carine
source: Am J Bot. 2014 Jun 6;101(6):979-1001.

title: Pollen competition between two sympatric Orchis species (Orchidaceae): the overtaking of consp
authors: ['Luca A', 'Palermo AM', 'Bellusci F', 'Pellegrino G']
source: Plant Biol (Stuttg). 2015 Jan;17(1):219-25. doi: 10.1111/plb.12199. Epub 2014 May 30.

title: In vitro conservation of Dendrobium germplasm.
authors: ['Teixeira da Silva JA', 'Zeng S', 'Galdiano RF Jr', 'Dobranszki J', 'Cardoso JC', 'Vendrame
source: Plant Cell Rep. 2014 Sep;33(9):1413-23. doi: 10.1007/s00299-014-1631-6. Epub 2014 May 21.

title: Gigantol, a bibenzyl from Dendrobium draconis, inhibits the migratory behavior of non-small ce
authors: ['Charoenrungruang S', 'Chanvorachote P', 'Sritularak B', 'Pongrakhananon V']
source: J Nat Prod. 2014 Jun 27;77(6):1359-66. doi: 10.1021/np500015v. Epub 2014 May 20.

title: The analysis of the inflorescence miRNome of the orchid Orchis italica reveals a DEF-like MADS
authors: ['Aceto S', 'Sica M', 'De Paolo S', "D'Argenio V", 'Cantiello P', 'Salvatore F', 'Gaudio L']
source: PLoS One. 2014 May 15;9(5):e97839. doi: 10.1371/journal.pone.0097839. eCollection 2014.

title: Factors affecting the distribution pattern of wild plants with extremely small populations in
authors: ['Chen Y', 'Yang X', 'Yang Q', 'Li D', 'Long W', 'Luo W']
source: PLoS One. 2014 May 15;9(5):e97751. doi: 10.1371/journal.pone.0097751. eCollection 2014.

title: [Temporal and spatial variations of soil NO(3-)-N in Orychophragmus violaceus/spring maize rot
authors: ['Xiong J', 'Wang GL', 'Cao WD', 'Bai JS', 'Zeng NH', 'Yang L', 'Gao SJ', 'Shimizu K']
source: Ying Yong Sheng Tai Xue Bao. 2014 Feb;25(2):467-73.
```

```
title: Chemical composition, potential toxicity, and quality control procedures of the crude drug of
authors: ['Morales-Sanchez V', 'Rivero-Cruz I', 'Laguna-Hernandez G', 'Salazar-Chavez G', 'Mata R']
source: J Ethnopharmacol. 2014 Jul 3;154(3):790-7. doi: 10.1016/j.jep.2014.05.006. Epub 2014 May 10.

title: Nutritional regulation in mixotrophic plants: new insights from Limodorum abortivum.
authors: ['Bellino A', 'Alfani A', 'Selosse MA', 'Guerrieri R', 'Borghetti M', 'Baldantoni D']
source: Oecologia. 2014 Jul;175(3):875-85. doi: 10.1007/s00442-014-2940-8. Epub 2014 May 11.

title: Evaluation of internal control for gene expression in Phalaenopsis by quantitative real-time P
authors: ['Yuan XY', 'Jiang SH', 'Wang MF', 'Ma J', 'Zhang XY', 'Cui B']
source: Appl Biochem Biotechnol. 2014 Jul;173(6):1431-45. doi: 10.1007/s12010-014-0951-x. Epub 2014 M

title: Male interference with pollination efficiency in a hermaphroditic orchid.
authors: ['Duffy KJ', 'Johnson SD']
source: J Evol Biol. 2014 Aug;27(8):1751-6. doi: 10.1111/jeb.12395. Epub 2014 May 6.

title: Biodegradation of polystyrene-graft-starch copolymers in three different types of soil.
authors: ['Nikolic V', 'Velickovic S', 'Popovic A']
source: Environ Sci Pollut Res Int. 2014;21(16):9877-86. doi: 10.1007/s11356-014-2946-0. Epub 2014 Ma

title: Mycorrhizal compatibility and symbiotic reproduction of Gavilea australis, an endangered terre
authors: ['Fracchia S', 'Aranda-Rickert A', 'Flachsland E', 'Terada G', 'Sede S']
source: Mycorrhiza. 2014 Nov;24(8):627-34. doi: 10.1007/s00572-014-0579-2. Epub 2014 Apr 30.

title: Bioguided identification of antifungal and antiproliferative compounds from the Brazilian orch
authors: ['Porte LF', 'Santin SM', 'Chiavelli LU', 'Silva CC', 'Faria TJ', 'Faria RT', 'Ruiz AL', 'Ca
source: Z Naturforsch C. 2014 Jan-Feb;69(1-2):46-52.

title: Helvolic acid, an antibacterial nortriterpenoid from a fungal endophyte, sp. of orchid endemic
authors: ['Ratnaweera PB', 'Williams DE', 'de Silva ED', 'Wijesundera RL', 'Dalisay DS', 'Andersen RJ
source: Mycology. 2014 Mar;5(1):23-28. Epub 2014 Mar 25.

title: Gene expression in mycorrhizal orchid protocorms suggests a friendly plant-fungus relationship
authors: ['Perotto S', 'Rodda M', 'Benetti A', 'Sillo F', 'Ercole E', 'Rodda M', 'Girlanda M', 'Murat
source: Planta. 2014 Jun;239(6):1337-49. doi: 10.1007/s00425-014-2062-x. Epub 2014 Apr 24.

title: Synthesis and mechanistic studies of a novel homoisoflavanone inhibitor of endothelial cell gr
authors: ['Basavarajappa HD', 'Lee B', 'Fei X', 'Lim D', 'Callaghan B', 'Mund JA', 'Case J', 'Rajashe
source: PLoS One. 2014 Apr 21;9(4):e95694. doi: 10.1371/journal.pone.0095694. eCollection 2014.

title: A new phylogenetic analysis sheds new light on the relationships in the Calanthe alliance (Orc
authors: ['Zhai JW', 'Zhang GQ', 'Li L', 'Wang M', 'Chen LJ', 'Chung SW', 'Rodriguez FJ', 'Francisco-
source: Mol Phylogenet Evol. 2014 Aug;77:216-22. doi: 10.1016/j.ympev.2014.04.005. Epub 2014 Apr 18.

title: Molecular systematics of subtribe Orchidinae and Asian taxa of Habenariinae (Orchideae, Orchic
authors: ['Jin WT', 'Jin XH', 'Schuiteman A', 'Li DZ', 'Xiang XG', 'Huang WC', 'Li JW', 'Huang LQ']
source: Mol Phylogenet Evol. 2014 Aug;77:41-53. doi: 10.1016/j.ympev.2014.04.004. Epub 2014 Apr 16.

title: Memory for expectation-violating concepts: the effects of agents and cultural familiarity.
authors: ['Porubanova M', 'Shaw DJ', 'McKay R', 'Xygalatas D']
source: PLoS One. 2014 Apr 8;9(4):e90684. doi: 10.1371/journal.pone.0090684. eCollection 2014.

title: Discovery of pyrazines as pollinator sex pheromones and orchid semiochemicals: implications fo
authors: ['Bohman B', 'Phillips RD', 'Menz MH', 'Berntsson BW', 'Flematti GR', 'Barrow RA', 'Dixon KW
source: New Phytol. 2014 Aug;203(3):939-52. doi: 10.1111/nph.12800. Epub 2014 Apr 3.

title: Floral colleters in Pleurothallidinae (Epidendroideae: Orchidaceae).
authors: ['Cardoso-Gustavson P', 'Campbell LM', 'Mazzoni-Viveiros SC', 'de Barros F']
```

```
source: Am J Bot. 2014 Apr;101(4):587-97. doi: 10.3732/ajb.1400012. Epub 2014 Mar 31.

title: Expression of paralogous SEP-, FUL-, AG- and STK-like MADS-box genes in wild-type and peloric
authors: ['Acri-Nunes-Miranda R', 'Mondragon-Palomino M']
source: Front Plant Sci. 2014 Mar 12;5:76. doi: 10.3389/fpls.2014.00076. eCollection 2014.

title: Spatio-temporal Genetic Structure of a Tropical Bee Species Suggests High Dispersal Over a Fra
authors: ['Suni SS', 'Bronstein JL', 'Brosi BJ']
source: Biotropica. 2014 Mar 1;46(2):202-209.

title: 2,2'-[Benzene-1,2-diylbis(iminomethanediyl)]diphenol derivative bearing two amine and hydroxyl
authors: ['Tayade K', 'Sahoo SK', 'Patil R', 'Singh N', 'Attarde S', 'Kuwar A']
source: Spectrochim Acta A Mol Biomol Spectrosc. 2014 May 21;126:312-6. doi: 10.1016/j.saa.2014.02.00

title: Climate, physiological tolerance and sex-biased dispersal shape genetic structure of Neotropic
authors: ['Lopez-Uribe MM', 'Zamudio KR', 'Cardoso CF', 'Danforth BN']
source: Mol Ecol. 2014 Apr;23(7):1874-90. doi: 10.1111/mec.12689. Epub 2014 Mar 18.

title: There is more to pollinator-mediated selection than pollen limitation.
authors: ['Sletvold N', 'Agren J']
source: Evolution. 2014 Jul;68(7):1907-18. doi: 10.1111/evo.12405. Epub 2014 Apr 16.

title: Three new bioactive phenolic glycosides from Liparis odorata.
authors: ['Li B', 'Liu H', 'Zhang D', 'Lai X', 'Liu B', 'Xu X', 'Xu P']
source: Nat Prod Res. 2014;28(8):522-9. doi: 10.1080/14786419.2014.880916. Epub 2014 Mar 17.

title: The evolution of floral deception in Epipactis veratrifolia (Orchidaceae): from indirect defe
authors: ['Jin XH', 'Ren ZX', 'Xu SZ', 'Wang H', 'Li DZ', 'Li ZY']
source: BMC Plant Biol. 2014 Mar 12;14:63. doi: 10.1186/1471-2229-14-63.

title: Sexual safety practices of massage parlor-based sex workers and their clients.
authors: ['Kolar K', 'Atchison C', 'Bungay V']
source: AIDS Care. 2014;26(9):1100-4. doi: 10.1080/09540121.2014.894611. Epub 2014 Mar 12.

title: Identification of warm day and cool night conditions induced flowering-related genes in a Pha
authors: ['Li DM', 'Lu FB', 'Zhu GF', 'Sun YB', 'Xu YC', 'Jiang MD', 'Liu JW', 'Wang Z']
source: Genet Mol Res. 2014 Feb 14;13(3):7037-51. doi: 10.4238/2014.February.14.7.

title: Transcriptional mapping of the messenger and leader RNAs of orchid fleck virus, a bisegmented
authors: ['Kondo H', 'Maruyama K', 'Chiba S', 'Andika IB', 'Suzuki N']
source: Virology. 2014 Mar;452-453:166-74. doi: 10.1016/j.virol.2014.01.007. Epub 2014 Feb 4.

title: Flower development of Phalaenopsis orchid involves functionally divergent SEPALLATA-like gene
authors: ['Pan ZJ', 'Chen YY', 'Du JS', 'Chen YY', 'Chung MC', 'Tsai WC', 'Wang CN', 'Chen HH']
source: New Phytol. 2014 May;202(3):1024-42. doi: 10.1111/nph.12723. Epub 2014 Feb 14.

title: In situ seed baiting to isolate germination-enhancing fungi for an epiphytic orchid, Dendrobi
authors: ['Zi XM', 'Sheng CL', 'Goodale UM', 'Shao SC', 'Gao JY']
source: Mycorrhiza. 2014 Oct;24(7):487-99. doi: 10.1007/s00572-014-0565-8. Epub 2014 Feb 23.

title: Combination of vildagliptin and rosiglitazone ameliorates nonalcoholic fatty liver disease in
authors: ['Mookkan J', 'De S', 'Shetty P', 'Kulkarni NM', 'Devisingh V', 'Jaji MS', 'Lakshmi VP', 'C
source: Indian J Pharmacol. 2014 Jan-Feb;46(1):46-50. doi: 10.4103/0253-7613.125166.

title: The colonization patterns of different fungi on roots of Cymbidium hybridum plantlets and the
authors: ['Zhao XL', 'Yang JZ', 'Liu S', 'Chen CL', 'Zhu HY', 'Cao JX']
source: World J Microbiol Biotechnol. 2014 Jul;30(7):1993-2003. doi: 10.1007/s11274-014-1623-2. Epub

title: Pollinator specificity drives strong prepollination reproductive isolation in sympatric sexua
```

```
authors: ['Whitehead MR', 'Peakall R']
source: Evolution. 2014 Jun;68(6):1561-75. doi: 10.1111/evo.12382. Epub 2014 Mar 26.

title: Floral scent emitted by white and coloured morphs in orchids.
authors: ['Dormont L', 'Delle-Vedove R', 'Bessiere JM', 'Schatz B']
source: Phytochemistry. 2014 Apr;100:51-9. doi: 10.1016/j.phytochem.2014.01.009. Epub 2014 Feb 10.

title: HIV risk behaviors of male injecting drug users and associated non-condom use with regular fem
authors: ['Mishra RK', 'Ganju D', 'Ramesh S', 'Lalmuanpuii M', 'Biangtung L', 'Humtsoe C', 'Saggurti
source: Harm Reduct J. 2014 Feb 13;11:5. doi: 10.1186/1477-7517-11-5.

title: Antimicrobial activity of cold and hot successive pseudobulb extracts of Flickingeria nodosa
authors: ['Nagananda GS', 'Satishchandra N']
source: Pak J Biol Sci. 2013 Oct 15;16(20):1189-93.

title: Evidence for isolation-by-habitat among populations of an epiphytic orchid species on a small
authors: ['Mallet B', 'Martos F', 'Blambert L', 'Pailler T', 'Humeau L']
source: PLoS One. 2014 Feb 3;9(2):e87469. doi: 10.1371/journal.pone.0087469. eCollection 2014.

title: Stable isotope cellular imaging reveals that both live and degenerating fungal pelotons transf
authors: ['Kuga Y', 'Sakamoto N', 'Yurimoto H']
source: New Phytol. 2014 Apr;202(2):594-605. doi: 10.1111/nph.12700. Epub 2014 Feb 3.

title: Isolation and characterisation of degradation impurities in the cefazolin sodium drug substanc
authors: ['Sivakumar B', 'Parthasarathy K', 'Murugan R', 'Jeyasudha R', 'Murugan S', 'Saranghdar RJ']
source: Sci Pharm. 2013 Jun 4;81(4):933-50. doi: 10.3797/scipharm.1304-14. eCollection 2013 Dec.

title: Antimycobacterial evaluation of novel hybrid arylidene thiazolidine-2,4-diones.
authors: ['Ponnuchamy S', 'Kanchithalaivan S', 'Ranjith Kumar R', 'Ali MA', 'Choon TS']
source: Bioorg Med Chem Lett. 2014 Feb 15;24(4):1089-93. doi: 10.1016/j.bmcl.2014.01.007. Epub 2014 J

title: A framework for assessing supply-side wildlife conservation.
authors: ['Phelps J', 'Carrasco LR', 'Webb EL']
source: Conserv Biol. 2014 Feb;28(1):244-57. doi: 10.1111/cobi.12160. Epub 2013 Nov 1.

title: Impact of primer choice on characterization of orchid mycorrhizal communities using 454 pyrose
authors: ['Waud M', 'Busschaert P', 'Ruyters S', 'Jacquemyn H', 'Lievens B']
source: Mol Ecol Resour. 2014 Jul;14(4):679-99. doi: 10.1111/1755-0998.12229. Epub 2014 Mar 14.

title: 3-Isopropyl-1-{2-[(1-methyl-1H-tetra-zol-5-yl)sulfan-yl]acet-yl}-2,6-di-phenyl-pi peridin-4-on
authors: ['Ganesan S', 'Sugumar P', 'Ananthan S', 'Ponnuswamy MN']
source: Acta Crystallogr Sect E Struct Rep Online. 2013 Oct 2;69(Pt 11):o1598. doi: 10.1107/S16005368

title: Carbon and nitrogen gain during the growth of orchid seedlings in nature.
authors: ['Stockel M', 'Tesitelova T', 'Jersakova J', 'Bidartondo MI', 'Gebauer G']
source: New Phytol. 2014 Apr;202(2):606-15. doi: 10.1111/nph.12688. Epub 2014 Jan 21.

title: Growth promotion-related miRNAs in Oncidium orchid roots colonized by the endophytic fungus Pi
authors: ['Ye W', 'Shen CH', 'Lin Y', 'Chen PJ', 'Xu X', 'Oelmuller R', 'Yeh KW', 'Lai Z']
source: PLoS One. 2014 Jan 7;9(1):e84920. doi: 10.1371/journal.pone.0084920. eCollection 2014.

title: Seedling development and evaluation of genetic stability of cryopreserved Dendrobium hybrid ma
authors: ['Galdiano RF Jr', 'de Macedo Lemos EG', 'de Faria RT', 'Vendrame WA']
source: Appl Biochem Biotechnol. 2014 Mar;172(5):2521-9. doi: 10.1007/s12010-013-0699-8. Epub 2014 Ja

title: Systematic revision of Platanthera in the Azorean archipelago: not one but three species, incl
authors: ['Bateman RM', 'Rudall PJ', 'Moura M']
source: PeerJ. 2013 Dec 10;1:e218. doi: 10.7717/peerj.218. eCollection 2013.
```

```
title: Deep sequencing-based analysis of the Cymbidium ensifolium floral transcriptome.
authors: ['Li X', 'Luo J', 'Yan T', 'Xiang L', 'Jin F', 'Qin D', 'Sun C', 'Xie M']
source: PLoS One. 2013 Dec 31;8(12):e85480. doi: 10.1371/journal.pone.0085480. eCollection 2013.

title: Pyrazines Attract Catocheilus Thynnine Wasps.
authors: ['Bohman B', 'Peakall R']
source: Insects. 2014 Jun 19;5(2):474-87. doi: 10.3390/insects5020474.

title: Comparison of hypoglycemic and antioxidative effects of polysaccharides from four different De
authors: ['Pan LH', 'Li XF', 'Wang MN', 'Zha XQ', 'Yang XF', 'Liu ZJ', 'Luo YB', 'Luo JP']
source: Int J Biol Macromol. 2014 Mar;64:420-7. doi: 10.1016/j.ijbiomac.2013.12.024. Epub 2013 Dec 2

title: Caught in the act: pollination of sexually deceptive trap-flowers by fungus gnats in Pterosty
authors: ['Phillips RD', 'Scaccabarozzi D', 'Retter BA', 'Hayes C', 'Brown GR', 'Dixon KW', 'Peakall
source: Ann Bot. 2014 Mar;113(4):629-41. doi: 10.1093/aob/mct295. Epub 2013 Dec 22.

title: Pollinator deception in the orchid mantis.
authors: ["O'Hanlon JC", 'Holwell GI', 'Herberstein ME']
source: Am Nat. 2014 Jan;183(1):126-32. doi: 10.1086/673858. Epub 2013 Sep 23.

title: Coexisting orchid species have distinct mycorrhizal communities and display strong spatial seg
authors: ['Jacquemyn H', 'Brys R', 'Merckx VS', 'Waud M', 'Lievens B', 'Wiegand T']
source: New Phytol. 2014 Apr;202(2):616-27. doi: 10.1111/nph.12640. Epub 2013 Dec 11.

title: Structurally characterized arabinogalactan from Anoectochilus formosanus as an immuno-modulat
authors: ['Yang LC', 'Hsieh CC', 'Lu TJ', 'Lin WC']
source: Phytomedicine. 2014 Apr 15;21(5):647-55. doi: 10.1016/j.phymed.2013.10.032. Epub 2013 Dec 4.

title: Proteome changes in Oncidium sphacelatum (Orchidaceae) at different trophic stages of symbioti
authors: ['Valadares RB', 'Perotto S', 'Santos EC', 'Lambais MR']
source: Mycorrhiza. 2014 Jul;24(5):349-60. doi: 10.1007/s00572-013-0547-2. Epub 2013 Dec 6.

title: First flowering hybrid between autotrophic and mycoheterotrophic plant species: breakthrough i
authors: ['Ogura-Tsujita Y', 'Miyoshi K', 'Tsutsumi C', 'Yukawa T']
source: J Plant Res. 2014 Mar;127(2):299-305. doi: 10.1007/s10265-013-0612-0. Epub 2013 Dec 6.

title: Relative importance of pollen and seed dispersal across a Neotropical mountain landscape for a
authors: ['Kartzinel TR', 'Shefferson RP', 'Trapnell DW']
source: Mol Ecol. 2013 Dec;22(24):6048-59. doi: 10.1111/mec.12551. Epub 2013 Nov 8.

title: DNA barcoding of Orchidaceae in Korea.
authors: ['Kim HM', 'Oh SH', 'Bhandari GS', 'Kim CS', 'Park CW']
source: Mol Ecol Resour. 2014 May;14(3):499-507. doi: 10.1111/1755-0998.12207. Epub 2013 Dec 16.

title: A modified ABCDE model of flowering in orchids based on gene expression profiling studies of t
authors: ['Su CL', 'Chen WC', 'Lee AY', 'Chen CY', 'Chang YC', 'Chao YT', 'Shih MC']
source: PLoS One. 2013 Nov 12;8(11):e80462. doi: 10.1371/journal.pone.0080462. eCollection 2013.

title: Mycorrhizal preferences and fine spatial structure of the epiphytic orchid Epidendrum rhopalos
authors: ['Riofrio ML', 'Cruz D', 'Torres E', 'de la Cruz M', 'Iriondo JM', 'Suarez JP']
source: Am J Bot. 2013 Dec;100(12):2339-48. doi: 10.3732/ajb.1300069. Epub 2013 Nov 19.

title: Combining microtomy and confocal laser scanning microscopy for structural analyses of plant-fu
authors: ['Rath M', 'Grolig F', 'Haueisen J', 'Imhof S']
source: Mycorrhiza. 2014 May;24(4):293-300. doi: 10.1007/s00572-013-0530-y. Epub 2013 Nov 19.

title: High-resolution secondary ion mass spectrometry analysis of carbon dynamics in mycorrhizas fo
authors: ['Bougoure J', 'Ludwig M', 'Brundrett M', 'Cliff J', 'Clode P', 'Kilburn M', 'Grierson P']
source: Plant Cell Environ. 2014 May;37(5):1223-30. doi: 10.1111/pce.12230. Epub 2013 Dec 12.
```

```
title: Donkey orchid symptomless virus: a viral 'platypus' from Australian terrestrial orchids.
authors: ['Wylie SJ', 'Li H', 'Jones MG']
source: PLoS One. 2013 Nov 5;8(11):e79587. doi: 10.1371/journal.pone.0079587. eCollection 2013.

title: Genome-wide annotation, expression profiling, and protein interaction studies of the core cell
authors: ['Lin HY', 'Chen JC', 'Wei MJ', 'Lien YC', 'Li HH', 'Ko SS', 'Liu ZH', 'Fang SC']
source: Plant Mol Biol. 2014 Jan;84(1-2):203-26. doi: 10.1007/s11103-013-0128-y. Epub 2013 Sep 25.

title: Effect of plasmolysis on protocorm-like bodies of Dendrobium Bobby Messina orchid following cr
authors: ['Antony JJ', 'Mubbarakh SA', 'Mahmood M', 'Subramaniam S']
source: Appl Biochem Biotechnol. 2014 Feb;172(3):1433-44. doi: 10.1007/s12010-013-0636-x. Epub 2013 N

title: The orchid-bee fauna (Hymenoptera: Apidae) of 'RPPN Feliciano Miguel Abdala' revisited: releva
authors: ['Nemesio A', 'Paula IR']
source: Braz J Biol. 2013 Aug;73(3):515-20. doi: 10.1590/S1519-69842013000300008.

title: Community of orchid bees (Hymenoptera: Apidae) in transitional vegetation between Cerrado and
authors: ['Pires EP', 'Morgado LN', 'Souza B', 'Carvalho CF', 'Nemesio A']
source: Braz J Biol. 2013 Aug;73(3):507-13. doi: 10.1590/S1519-69842013000300007.

title: The AP2-like gene OitaAP2 is alternatively spliced and differentially expressed in inflorescen
authors: ['Salemme M', 'Sica M', 'Iazzetti G', 'Gaudio L', 'Aceto S']
source: PLoS One. 2013 Oct 21;8(10):e77454. doi: 10.1371/journal.pone.0077454. eCollection 2013.

title: Identification and characterization of the microRNA transcriptome of a moth orchid Phalaenopsi
authors: ['Chao YT', 'Su CL', 'Jean WH', 'Chen WC', 'Chang YC', 'Shih MC']
source: Plant Mol Biol. 2014 Mar;84(4-5):529-48. doi: 10.1007/s11103-013-0150-0. Epub 2013 Oct 31.

title: [Some aspects of underground organs of spotleaf orchis growth and phenolic compound accumulati
authors: ['Marakaev OA', 'Tselebrovskii MV', 'Nikolaeva TN', 'Zagoskina NV']
source: Izv Akad Nauk Ser Biol. 2013 May-Jun;(3):315-23.

title: Floral elaiophores in Lockhartia Hook. (Orchidaceae: Oncidiinae): their distribution, diversit
authors: ['Blanco MA', 'Davies KL', 'Stpiczynska M', 'Carlsward BS', 'Ionta GM', 'Gerlach G']
source: Ann Bot. 2013 Dec;112(9):1775-91. doi: 10.1093/aob/mct232. Epub 2013 Oct 29.

title: Pollinator shifts and the evolution of spur length in the moth-pollinated orchid Platanthera b
authors: ['Boberg E', 'Alexandersson R', 'Jonsson M', 'Maad J', 'Agren J', 'Nilsson LA']
source: Ann Bot. 2014 Jan;113(2):267-75. doi: 10.1093/aob/mct217. Epub 2013 Oct 29.

title: Conservation value and permeability of neotropical oil palm landscapes for orchid bees.
authors: ['Livingston G', 'Jha S', 'Vega A', 'Gilbert L']
source: PLoS One. 2013 Oct 17;8(10):e78523. doi: 10.1371/journal.pone.0078523. eCollection 2013.

title: First records and description of metallic red females of Euglossa (Alloglossura) gorgonensis C
authors: ['Hinojosa-Diaz IA', 'Brosi BJ']
source: Zookeys. 2013 Sep 25;(335):113-9. doi: 10.3897/zookeys.335.6134. eCollection 2013.

title: Cryopreservation of Brassidium Shooting Star orchid using the PVS3 method supported with preli
authors: ['Mubbarakh SA', 'Rahmah S', 'Rahman ZA', 'Sah NN', 'Subramaniam S']
source: Appl Biochem Biotechnol. 2014 Jan;172(2):1131-45. doi: 10.1007/s12010-013-0597-0. Epub 2013 O

title: Niche conservatism and the future potential range of Epipactis helleborine (Orchidaceae).
authors: ['Kolanowska M']
source: PLoS One. 2013 Oct 15;8(10):e77352. doi: 10.1371/journal.pone.0077352. eCollection 2013.

title: Orchid protocorm-like bodies are somatic embryos.
authors: ['Lee YI', 'Hsu ST', 'Yeung EC']
```

```
source: Am J Bot. 2013 Nov;100(11):2121-31. doi: 10.3732/ajb.1300193. Epub 2013 Oct 17.

title: Genetic diversity and population differentiation of Calanthe tsoongiana, a rare and endemic or
authors: ['Qian X', 'Wang CX', 'Tian M']
source: Int J Mol Sci. 2013 Oct 14;14(10):20399-413. doi: 10.3390/ijms141020399.

title: Highly diverse and spatially heterogeneous mycorrhizal symbiosis in a rare epiphyte is unrelat
authors: ['Kartzinel TR', 'Trapnell DW', 'Shefferson RP']
source: Mol Ecol. 2013 Dec;22(23):5949-61. doi: 10.1111/mec.12536. Epub 2013 Nov 6.

title: The IGSF1 deficiency syndrome: characteristics of male and female patients.
authors: ['Joustra SD', 'Schoenmakers N', 'Persani L', 'Campi I', 'Bonomi M', 'Radetti G', 'Beck-Pecc
source: J Clin Endocrinol Metab. 2013 Dec;98(12):4942-52. doi: 10.1210/jc.2013-2743. Epub 2013 Oct 9.

title: A pollinator shift explains floral divergence in an orchid species complex in South Africa.
authors: ['Peter CI', 'Johnson SD']
source: Ann Bot. 2014 Jan;113(2):277-88. doi: 10.1093/aob/mct216. Epub 2013 Oct 9.

title: Floral adaptation to local pollinator guilds in a terrestrial orchid.
authors: ['Sun M', 'Gross K', 'Schiestl FP']
source: Ann Bot. 2014 Jan;113(2):289-300. doi: 10.1093/aob/mct219. Epub 2013 Oct 9.

title: Challenges and prospects in the telemetry of insects.
authors: ['Daniel Kissling W', 'Pattemore DE', 'Hagen M']
source: Biol Rev Camb Philos Soc. 2014 Aug;89(3):511-30. doi: 10.1111/brv.12065. Epub 2013 Oct 8.

title: Identification and characterization of process-related impurities of trans-resveratrol.
authors: ['Sivakumar B', 'Murugan R', 'Baskaran A', 'Khadangale BP', 'Murugan S', 'Senthilkumar UP']
source: Sci Pharm. 2013 Mar 17;81(3):683-95. doi: 10.3797/scipharm.1301-17. eCollection 2013.

title: Vanilla--its science of cultivation, curing, chemistry, and nutraceutical properties.
authors: ['Anuradha K', 'Shyamala BN', 'Naidu MM']
source: Crit Rev Food Sci Nutr. 2013;53(12):1250-76. doi: 10.1080/10408398.2011.563879.

title: Dichorhavirus: a proposed new genus for Brevipalpus mite-transmitted, nuclear, bacilliform, bi
authors: ['Dietzgen RG', 'Kuhn JH', 'Clawson AN', 'Freitas-Astua J', 'Goodin MM', 'Kitajima EW', 'Kon
source: Arch Virol. 2014 Mar;159(3):607-19. doi: 10.1007/s00705-013-1834-0. Epub 2013 Oct 1.

title: Moscatilin induces apoptosis and mitotic catastrophe in human esophageal cancer cells.
authors: ['Chen CA', 'Chen CC', 'Shen CC', 'Chang HH', 'Chen YJ']
source: J Med Food. 2013 Oct;16(10):869-77. doi: 10.1089/jmf.2012.2617. Epub 2013 Sep 27.

title: Composition of Cypripedium calceolus (Orchidaceae) seeds analyzed by attenuated total reflecta
authors: ['Barsberg S', 'Rasmussen HN', 'Kodahl N']
source: Am J Bot. 2013 Oct;100(10):2066-73. doi: 10.3732/ajb.1200646. Epub 2013 Sep 26.

title: Perspectives on MADS-box expression during orchid flower evolution and development.
authors: ['Mondragon-Palomino M']
source: Front Plant Sci. 2013 Sep 23;4:377. doi: 10.3389/fpls.2013.00377. eCollection 2013.

title: Simple-sequence repeat markers of Cattleya coccinea (Orchidaceae), an endangered species of th
authors: ['Novello M', 'Rodrigues JF', 'Pinheiro F', 'Oliveira GC', 'Veasey EA', 'Koehler S']
source: Genet Mol Res. 2013 Sep 3;12(3):3274-8. doi: 10.4238/2013.September.3.3.

title: Floral odour chemistry defines species boundaries and underpins strong reproductive isolation
authors: ['Peakall R', 'Whitehead MR']
source: Ann Bot. 2014 Jan;113(2):341-55. doi: 10.1093/aob/mct199. Epub 2013 Sep 19.

title: Identification and symbiotic ability of Psathyrellaceae fungi isolated from a photosynthetic o
```

```
authors: ['Yagame T', 'Funabiki E', 'Nagasawa E', 'Fukiharu T', 'Iwase K']
source: Am J Bot. 2013 Sep;100(9):1823-30. doi: 10.3732/ajb.1300099. Epub 2013 Sep 11.

title: Promoting role of an endophyte on the growth and contents of kinsenosides and flavonoids of An
authors: ['Zhang FS', 'Lv YL', 'Zhao Y', 'Guo SX']
source: J Zhejiang Univ Sci B. 2013 Sep;14(9):785-92. doi: 10.1631/jzus.B1300056.

title: Collection and trade of wild-harvested orchids in Nepal.
authors: ['Subedi A', 'Kunwar B', 'Choi Y', 'Dai Y', 'van Andel T', 'Chaudhary RP', 'de Boer HJ', 'Gr
source: J Ethnobiol Ethnomed. 2013 Aug 31;9(1):64. doi: 10.1186/1746-4269-9-64.

title: Direct detection of orchid viruses using nanorod-based fiber optic particle plasmon resonance
authors: ['Lin HY', 'Huang CH', 'Lu SH', 'Kuo IT', 'Chau LK']
source: Biosens Bioelectron. 2014 Jan 15;51:371-8. doi: 10.1016/j.bios.2013.08.009. Epub 2013 Aug 17.

title: The evolution of floral nectaries in Disa (Orchidaceae: Disinae): recapitulation or diversifyi
authors: ['Hobbhahn N', 'Johnson SD', 'Bytebier B', 'Yeung EC', 'Harder LD']
source: Ann Bot. 2013 Nov;112(7):1303-19. doi: 10.1093/aob/mct197. Epub 2013 Aug 29.

title: [Molecular cloning and characterization of S-adenosyl-L-methionine decarboxylase gene (DoSAMDC
authors: ['Zhao MM', 'Zhang G', 'Zhang DW', 'Guo SX']
source: Yao Xue Xue Bao. 2013 Jun;48(6):946-52.

title: Preliminary genetic linkage maps of Chinese herb Dendrobium nobile and D. moniliforme.
authors: ['Feng S', 'Zhao H', 'Lu J', 'Liu J', 'Shen B', 'Wang H']
source: J Genet. 2013;92(2):205-12.

title: Evidence of separate karyotype evolutionary pathway in Euglossa orchid bees by cytogenetic ana
authors: ['Fernandes A', 'Werneck HA', 'Pompolo SG', 'Lopes DM']
source: An Acad Bras Cienc. 2013 Sep;85(3):937-44. doi: 10.1590/S0001-37652013005000050.

title: Histological and micro-CT evidence of stigmatic rostellum receptivity promoting auto-pollinati
authors: ['Gamisch A', 'Staedler YM', 'Schonenberger J', 'Fischer GA', 'Comes HP']
source: PLoS One. 2013 Aug 13;8(8):e72688. doi: 10.1371/journal.pone.0072688. eCollection 2013.

title: Virus-induced gene silencing unravels multiple transcription factors involved in floral growth
authors: ['Hsieh MH', 'Pan ZJ', 'Lai PH', 'Lu HC', 'Yeh HH', 'Hsu CC', 'Wu WL', 'Chung MC', 'Wang SS
source: J Exp Bot. 2013 Sep;64(12):3869-84. doi: 10.1093/jxb/ert218.

title: Habitat fragmentation effects on the orchid bee communities in remnant forests of southeastern
authors: ['Knoll Fdo R', 'Penatti NC']
source: Neotrop Entomol. 2012 Oct;41(5):355-65. doi: 10.1007/s13744-012-0057-5. Epub 2012 Jun 29.

title: Spatial-temporal variation in orchid bee communities (Hymenoptera: Apidae) in remnants of arbo
authors: ['Andrade-Silva AC', 'Nemesio A', 'de Oliveira FF', 'Nascimento FS']
source: Neotrop Entomol. 2012 Aug;41(4):296-305. doi: 10.1007/s13744-012-0053-9. Epub 2012 Jun 26.

title: Old Fragments of Forest Inside an Urban Area Are Able to Keep Orchid Bee (Hymenoptera: Apidae
authors: ['Ferreira RP', 'Martins C', 'Dutra MC', 'Mentone CB', 'Antonini Y']
source: Neotrop Entomol. 2013 Jul 16.

title: The ecological basis for biogeographic classification: an example in orchid bees (Apidae: Eugl
authors: ['Parra-H A', 'Nates-Parra G']
source: Neotrop Entomol. 2012 Dec;41(6):442-9. doi: 10.1007/s13744-012-0069-1. Epub 2012 Aug 24.

title: Benefits to rare plants and highway safety from annual population reductions of a "native inva
authors: ['Engeman RM', 'Guerrant T', 'Dunn G', 'Beckerman SF', 'Anchor C']
source: Environ Sci Pollut Res Int. 2014 Jan;21(2):1592-7. doi: 10.1007/s11356-013-2056-4. Epub 2013
```

```
title: Start Codon Targeted (SCoT) marker reveals genetic diversity of Dendrobium nobile Lindl., an e
authors: ['Bhattacharyya P', 'Kumaria S', 'Kumar S', 'Tandon P']
source: Gene. 2013 Oct 15;529(1):21-6. doi: 10.1016/j.gene.2013.07.096. Epub 2013 Aug 11.

title: Vegetation context influences the strength and targets of pollinator-mediated selection in a d
authors: ['Sletvold N', 'Grindeland JM', 'Agren J']
source: Ecology. 2013 Jun;94(6):1236-42.

title: Pollination and floral ecology of Arundina graminifolia (Orchidaceae) at the northern border o
authors: ['Sugiura N']
source: J Plant Res. 2014;127(1):131-9. doi: 10.1007/s10265-013-0587-x. Epub 2013 Aug 7.

title: The orchid-bee faunas (Hymenoptera: Apidae) of 'Parque Nacional do Monte Pascoal', 'Parque Nac
authors: ['Nemesio A']
source: Braz J Biol. 2013 May;73(2):437-46. doi: 10.1590/S1519-69842013000200028.

title: The orchid-bee faunas (Hymenoptera: Apidae) of two Atlantic Forest remnants in southern Bahia,
authors: ['Nemesio A']
source: Braz J Biol. 2013 May;73(2):375-81. doi: 10.1590/S1519-69842013000200018.

title: Are orchid bees at risk? First comparative survey suggests declining populations of forest-dep
authors: ['Nemesio A']
source: Braz J Biol. 2013 May;73(2):367-74. doi: 10.1590/S1519-69842013000200017.

title: The orchid-bee fauna (Hymenoptera: Apidae) of 'Reserva Biologica de Una', a hotspot in the Atl
authors: ['Nemesio A']
source: Braz J Biol. 2013 May;73(2):347-52. doi: 10.1590/S1519-69842013000200014.

title: Ancestral deceit and labile evolution of nectar production in the African orchid genus Disa.
authors: ['Johnson SD', 'Hobbhahn N', 'Bytebier B']
source: Biol Lett. 2013 Jul 31;9(5):20130500. doi: 10.1098/rsbl.2013.0500. Print 2013 Oct 23.

title: Orchid bee (Hymenoptera: Apidae) community from a gallery forest in the Brazilian Cerrado.
authors: ['Silva FS']
source: Rev Biol Trop. 2012 Jun;60(2):625-33.

title: Genome assembly of citrus leprosis virus nuclear type reveals a close association with orchid
authors: ['Roy A', 'Stone A', 'Otero-Colina G', 'Wei G', 'Choudhary N', 'Achor D', 'Shao J', 'Levy L
source: Genome Announc. 2013 Jul 25;1(4). pii: e00519-13. doi: 10.1128/genomeA.00519-13.

title: Complete genome sequence of Habenaria mosaic virus, a new potyvirus infecting a terrestrial or
authors: ['Kondo H', 'Maeda T', 'Gara IW', 'Chiba S', 'Maruyama K', 'Tamada T', 'Suzuki N']
source: Arch Virol. 2014 Jan;159(1):163-6. doi: 10.1007/s00705-013-1784-6. Epub 2013 Jul 16.

title: Modulation of physical environment makes placental mesenchymal stromal cells suitable for ther
authors: ['Mathew SA', 'Rajendran S', 'Gupta PK', 'Bhonde R']
source: Cell Biol Int. 2013 Nov;37(11):1197-204. doi: 10.1002/cbin.10154. Epub 2013 Aug 13.

title: Genetic inference of epiphytic orchid colonization; it may only take one.
authors: ['Trapnell DW', 'Hamrick JL', 'Ishibashi CD', 'Kartzinel TR']
source: Mol Ecol. 2013 Jul;22(14):3680-92. doi: 10.1111/mec.12338.

title: Microbial diversity in the floral nectar of seven Epipactis (Orchidaceae) species.
authors: ['Jacquemyn H', 'Lenaerts M', 'Tyteca D', 'Lievens B']
source: Microbiologyopen. 2013 Aug;2(4):644-58. doi: 10.1002/mbo3.103. Epub 2013 Jul 8.

title: [Sequence analysis of LEAFY homologous gene from Dendrobium moniliforme and application for id
authors: ['Xing WR', 'Hou BW', 'Guan JJ', 'Luo J', 'Ding XY']
source: Yao Xue Xue Bao. 2013 Apr;48(4):597-603.
```

```
title: Components of reproductive isolation between Orchis mascula and Orchis pauciflora.
authors: ['Scopece G', 'Croce A', 'Lexer C', 'Cozzolino S']
source: Evolution. 2013 Jul;67(7):2083-93. doi: 10.1111/evo.12091. Epub 2013 Mar 29.

title: Phylogeographic structure and outbreeding depression reveal early stages of reproductive isola
authors: ['Pinheiro F', 'Cozzolino S', 'de Barros F', 'Gouveia TM', 'Suzuki RM', 'Fay MF', 'Palma-Sil
source: Evolution. 2013 Jul;67(7):2024-39. doi: 10.1111/evo.12085. Epub 2013 Mar 21.

title: Asymbiotic seed germination and in vitro conservation of Coelogyne nervosa A. Rich. an endemic
authors: ['Abraham S', 'Augustine J', 'Thomas TD']
source: Physiol Mol Biol Plants. 2012 Jul;18(3):245-51. doi: 10.1007/s12298-012-0118-6.

title: Endophytic and mycorrhizal fungi associated with roots of endangered native orchids from the A
authors: ['Oliveira SF', 'Bocayuva MF', 'Veloso TG', 'Bazzolli DM', 'da Silva CC', 'Pereira OL', 'Kas
source: Mycorrhiza. 2014 Jan;24(1):55-64. doi: 10.1007/s00572-013-0512-0. Epub 2013 Jun 30.

title: Convergent evolution of floral signals underlies the success of Neotropical orchids.
authors: ['Papadopulos AS', 'Powell MP', 'Pupulin F', 'Warner J', 'Hawkins JA', 'Salamin N', 'Chittka
source: Proc Biol Sci. 2013 Jun 26;280(1765):20130960. doi: 10.1098/rspb.2013.0960. Print 2013 Aug 22

title: 3,3'-{[(Biphenyl-2,2'-di-yl)bis-(methyl-ene)]bis-(-oxy)}bis-[N-(4-chloro-phen-yl) benzamide].
authors: ['Rajadurai R', 'Padmanabhan R', 'Meenakshi Sundaram SS', 'Ananthan S']
source: Acta Crystallogr Sect E Struct Rep Online. 2013 May 18;69(Pt 6):o914-5. doi: 10.1107/S160053

title: 3,5-Dimethyl-1-{2-[(5-methyl-1,3,4-thia-diazol-2-yl)sulfan-yl]acet-yl}-2,6-diphen ylpiperidin-
authors: ['Ganesan S', 'Sugumar P', 'Ananthan S', 'Ponnuswamy MN']
source: Acta Crystallogr Sect E Struct Rep Online. 2013 May 11;69(Pt 6):o845. doi: 10.1107/S16005368

title: Symbiotic seed germination and protocorm development of Aa achalensis Schltr., a terrestrial o
authors: ['Sebastian F', 'Vanesa S', 'Eduardo F', 'Graciela T', 'Silvana S']
source: Mycorrhiza. 2014 Jan;24(1):35-43. doi: 10.1007/s00572-013-0510-2. Epub 2013 Jun 20.

title: [Sibling species of rein orchid (Gymnadenia:Orchidaceae, Magnoliophyta) in Russia].
authors: ['Efimov PG']
source: Genetika. 2013 Mar;49(3):343-54.

title: Detection of viruses directly from the fresh leaves of a Phalaenopsis orchid using a microflui
authors: ['Chang WH', 'Yang SY', 'Lin CL', 'Wang CH', 'Li PC', 'Chen TY', 'Jan FJ', 'Lee GB']
source: Nanomedicine. 2013 Nov;9(8):1274-82. doi: 10.1016/j.nano.2013.05.016. Epub 2013 Jun 8.

title: Floral visual signal increases reproductive success in a sexually deceptive orchid.
authors: ['Rakosy D', 'Streinzer M', 'Paulus HF', 'Spaethe J']
source: Arthropod Plant Interact. 2012 Dec 1;6(4):671-681.

title: Moscatilin inhibits lung cancer cell motility and invasion via suppression of endogenous react
authors: ['Kowitdamrong A', 'Chanvorachote P', 'Sritularak B', 'Pongrakhananon V']
source: Biomed Res Int. 2013;2013:765894. doi: 10.1155/2013/765894. Epub 2013 May 8.

title: Transcriptome and proteome data reveal candidate genes for pollinator attraction in sexually d
authors: ['Sedeek KE', 'Qi W', 'Schauer MA', 'Gupta AK', 'Poveda L', 'Xu S', 'Liu ZJ', 'Grossniklaus
source: PLoS One. 2013 May 29;8(5):e64621. doi: 10.1371/journal.pone.0064621. Print 2013.

title: Beyond orchids and dandelions: testing the 5-HTT "risky" allele for evidence of phenotypic cap
authors: ['Conley D', 'Rauscher E', 'Siegal ML']
source: Biodemography Soc Biol. 2013;59(1):37-56. doi: 10.1080/19485565.2013.774620.

title: Molecular cloning and spatiotemporal expression of an APETALA1/FRUITFULL-like MADS-box gene fr
authors: ['Tian Y', 'Yuan X', 'Jiang S', 'Cui B', 'Su J']
```

```
source: Sheng Wu Gong Cheng Xue Bao. 2013 Feb;29(2):203-13.

title: Antigenotoxic effect, composition and antioxidant activity of Dendrobium speciosum.
authors: ['Moretti M', 'Cossignani L', 'Messina F', 'Dominici L', 'Villarini M', 'Curini M', 'Marcotu
source: Food Chem. 2013 Oct 15;140(4):660-5. doi: 10.1016/j.foodchem.2012.10.022. Epub 2012 Nov 2.

title: Exposure to HIV prevention programmes associated with improved condom use and uptake of HIV te
authors: ['Armstrong G', 'Medhi GK', 'Kermode M', 'Mahanta J', 'Goswami P', 'Paranjape R']
source: BMC Public Health. 2013 May 15;13:476. doi: 10.1186/1471-2458-13-476.

title: Significant spatial aggregation and fine-scale genetic structure in the homosporous fern Cyrto
authors: ['Chung MY', 'Chung MG']
source: New Phytol. 2013 Aug;199(3):663-72. doi: 10.1111/nph.12293. Epub 2013 May 7.

title: Catalase and superoxide dismutase activities and the total protein content of protocorm-like b
authors: ['Poobathy R', 'Sinniah UR', 'Xavier R', 'Subramaniam S']
source: Appl Biochem Biotechnol. 2013 Jul;170(5):1066-79. doi: 10.1007/s12010-013-0241-z. Epub 2013 M

title: Organ homologies in orchid flowers re-interpreted using the Musk Orchid as a model.
authors: ['Rudall PJ', 'Perl CD', 'Bateman RM']
source: PeerJ. 2013 Feb 12;1:e26. doi: 10.7717/peerj.26. Print 2013.

title: Accessing local knowledge to identify where species of conservation concern occur in a tropica
authors: ['Padmanaba M', 'Sheil D', 'Basuki I', 'Liswanti N']
source: Environ Manage. 2013 Aug;52(2):348-59. doi: 10.1007/s00267-013-0051-7. Epub 2013 May 1.

title: Spatial patterns of photosynthesis in thin- and thick-leaved epiphytic orchids: unravelling C3
authors: ['Rodrigues MA', 'Matiz A', 'Cruz AB', 'Matsumura AT', 'Takahashi CA', 'Hamachi L', 'Felix L
source: Ann Bot. 2013 Jul;112(1):17-29. doi: 10.1093/aob/mct090. Epub 2013 Apr 25.

title: Transcriptome analysis of Cymbidium sinense and its application to the identification of genes
authors: ['Zhang J', 'Wu K', 'Zeng S', 'Teixeira da Silva JA', 'Zhao X', 'Tian CE', 'Xia H', 'Duan J
source: BMC Genomics. 2013 Apr 24;14:279. doi: 10.1186/1471-2164-14-279.

title: Orchid fleck virus structural proteins N and P form intranuclear viroplasm-like structures in
authors: ['Kondo H', 'Chiba S', 'Andika IB', 'Maruyama K', 'Tamada T', 'Suzuki N']
source: J Virol. 2013 Jul;87(13):7423-34. doi: 10.1128/JVI.00270-13. Epub 2013 Apr 24.

title: Scale-up of a comprehensive harm reduction programme for people injecting opioids: lessons fro
authors: ['Lalmuanpuii M', 'Biangtung L', 'Mishra RK', 'Reeve MJ', 'Tzudier S', 'Singh AL', 'Sinate R
source: Bull World Health Organ. 2013 Apr 1;91(4):306-12. doi: 10.2471/BLT.12.108274. Epub 2013 Feb 2

title: Complete chloroplast genome of the genus Cymbidium: lights into the species identification, ph
authors: ['Yang JB', 'Tang M', 'Li HT', 'Zhang ZR', 'Li DZ']
source: BMC Evol Biol. 2013 Apr 18;13:84. doi: 10.1186/1471-2148-13-84.

title: A novel aphrodisiac compound from an orchid that activates nitric oxide synthases.
authors: ['Subramoniam A', 'Gangaprasad A', 'Sureshkumar PK', 'Radhika J', 'Arun KB']
source: Int J Impot Res. 2013 Nov-Dec;25(6):212-6. doi: 10.1038/ijir.2013.18. Epub 2013 Apr 18.

title: A new orchid genus, Danxiaorchis, and phylogenetic analysis of the tribe Calypsoeae.
authors: ['Zhai JW', 'Zhang GQ', 'Chen LJ', 'Xiao XJ', 'Liu KW', 'Tsai WC', 'Hsiao YY', 'Tian HZ', 'Z
source: PLoS One. 2013 Apr 4;8(4):e60371. doi: 10.1371/journal.pone.0060371. Print 2013.

title: Detection of ancestry informative HLA alleles confirms the admixed origins of Japanese populat
authors: ['Nakaoka H', 'Mitsunaga S', 'Hosomichi K', 'Shyh-Yuh L', 'Sawamoto T', 'Fujiwara T', 'Tsuts
source: PLoS One. 2013;8(4):e60793. doi: 10.1371/journal.pone.0060793. Epub 2013 Apr 5.

title: A new molecular phylogeny and a new genus, Pendulorchis, of the Aerides-Vanda alliance (Orchid
authors: ['Zhang GQ', 'Liu KW', 'Chen LJ', 'Xiao XJ', 'Zhai JW', 'Li LQ', 'Cai J', 'Hsiao YY', 'Rao W
source: PLoS One. 2013;8(4):e60097. doi: 10.1371/journal.pone.0060097. Epub 2013 Apr 5.

title: Catalog of Erycina pusilla miRNA and categorization of reproductive phase-related miRNAs and t
authors: ['Lin CS', 'Chen JJ', 'Huang YT', 'Hsu CT', 'Lu HC', 'Chou ML', 'Chen LC', 'Ou CI', 'Liao DC
source: Plant Mol Biol. 2013 May;82(1-2):193-204. doi: 10.1007/s11103-013-0055-y. Epub 2013 Apr 11.

title: The rare terrestrial orchid Nervilia nipponica consistently associates with a single group of
authors: ['Nomura N', 'Ogura-Tsujita Y', 'Gale SW', 'Maeda A', 'Umata H', 'Hosaka K', 'Yukawa T']
source: J Plant Res. 2013 Sep;126(5):613-23. doi: 10.1007/s10265-013-0552-8. Epub 2013 Apr 6.

title: Compatible fungi, suitable medium, and appropriate developmental stage essential for stable as
authors: ['Hajong S', 'Kumaria S', 'Tandon P']
source: J Basic Microbiol. 2013 Dec;53(12):1025-33. doi: 10.1002/jobm.201200411. Epub 2013 Apr 2.

title: The host bias of three epiphytic Aeridinae orchid species is reflected, but not explained, by
authors: ['Gowland KM', 'van der Merwe MM', 'Linde CC', 'Clements MA', 'Nicotra AB']
source: Am J Bot. 2013 Apr;100(4):764-77. doi: 10.3732/ajb.1200411. Epub 2013 Apr 1.

title: Tubastatin, a selective histone deacetylase 6 inhibitor shows anti-inflammatory and anti-rheum
authors: ['Vishwakarma S', 'Iyer LR', 'Muley M', 'Singh PK', 'Shastry A', 'Saxena A', 'Kulathingal J
source: Int Immunopharmacol. 2013 May;16(1):72-8. doi: 10.1016/j.intimp.2013.03.016. Epub 2013 Mar 2

title: Microsatellite markers in the western prairie fringed orchid, Platanthera praeclara (Orchidace
authors: ['Ross AA', 'Aldrich-Wolfe L', 'Lance S', 'Glenn T', 'Travers SE']
source: Appl Plant Sci. 2013 Mar 22;1(4). pii: apps.1200413. doi: 10.3732/apps.1200413. eCollection 2

title: Discovery of adamantane based highly potent HDAC inhibitors.
authors: ['Gopalan B', 'Ponpandian T', 'Kachhadia V', 'Bharathimohan K', 'Vignesh R', 'Sivasudar V',
source: Bioorg Med Chem Lett. 2013 May 1;23(9):2532-7. doi: 10.1016/j.bmcl.2013.03.002. Epub 2013 Mar

title: Mycorrhizas alter nitrogen acquisition by the terrestrial orchid Cymbidium goeringii.
authors: ['Wu J', 'Ma H', 'Xu X', 'Qiao N', 'Guo S', 'Liu F', 'Zhang D', 'Zhou L']
source: Ann Bot. 2013 Jun;111(6):1181-7. doi: 10.1093/aob/mct062. Epub 2013 Mar 26.

title: Variation in nutrient-acquisition patterns by mycorrhizal fungi of rare and common orchids exp
authors: ['Nurfadilah S', 'Swarts ND', 'Dixon KW', 'Lambers H', 'Merritt DJ']
source: Ann Bot. 2013 Jun;111(6):1233-41. doi: 10.1093/aob/mct064. Epub 2013 Mar 26.

title: Bioanalytical method development, validation and quantification of dorsomorphin in rat plasma
authors: ['Karthikeyan K', 'Mahat MY', 'Chandrasekaran S', 'Gopal K', 'Franklin PX', 'Sivakumar BJ',
source: Biomed Chromatogr. 2013 Aug;27(8):1018-26. doi: 10.1002/bmc.2899. Epub 2013 Mar 21.

title: Cryopreservation of orchid mycorrhizal fungi: a tool for the conservation of endangered specie
authors: ['Ercole E', 'Rodda M', 'Molinatti M', 'Voyron S', 'Perotto S', 'Girlanda M']
source: J Microbiol Methods. 2013 May;93(2):134-7. doi: 10.1016/j.mimet.2013.03.003. Epub 2013 Mar 1

title: Climate warming alters effects of management on population viability of threatened species: r
authors: ['Sletvold N', 'Dahlgren JP', 'Oien DI', 'Moen A', 'Ehrlen J']
source: Glob Chang Biol. 2013 Sep;19(9):2729-38. doi: 10.1111/gcb.12167. Epub 2013 Jul 14.

title: [Molecular characterization of a mitogen-activated protein kinase gene DoMPK1 in Dendrobium o
authors: ['Zhang G', 'Zhao MM', 'Song C', 'Zhang DW', 'Li B', 'Guo SX']
source: Yao Xue Xue Bao. 2012 Dec;47(12):1703-9.

title: Contributions of covariance: decomposing the components of stochastic population growth in Cy
authors: ['Davison R', 'Nicole F', 'Jacquemyn H', 'Tuljapurkar S']
source: Am Nat. 2013 Mar;181(3):410-20. doi: 10.1086/669155. Epub 2013 Jan 18.
```

```
title: Phylogenetic and microsatellite markers for Tulasnella (Tulasnellaceae) mycorrhizal fungi asso
authors: ['Ruibal MP', 'Peakall R', 'Smith LM', 'Linde CC']
source: Appl Plant Sci. 2013 Mar 5;1(3). pii: apps.1200394. doi: 10.3732/apps.1200394. eCollection 20

title: Ascertaining the role of Taiwan as a source for the Austronesian expansion.
authors: ['Mirabal S', 'Cadenas AM', 'Garcia-Bertrand R', 'Herrera RJ']
source: Am J Phys Anthropol. 2013 Apr;150(4):551-64. doi: 10.1002/ajpa.22226. Epub 2013 Feb 26.

title: Acquisition of species-specific perfume blends: influence of habitat-dependent compound availa
authors: ['Pokorny T', 'Hannibal M', 'Quezada-Euan JJ', 'Hedenstrom E', 'Sjoberg N', 'Bang J', 'Eltz
source: Oecologia. 2013 Jun;172(2):417-25. doi: 10.1007/s00442-013-2620-0. Epub 2013 Feb 27.

title: Fertilizing ability of cryopreserved pollinia of Luisia macrantha, an endemic orchid of Wester
authors: ['Ajeeshkumar S', 'Decruse SW']
source: Cryo Letters. 2013 Jan-Feb;34(1):20-9.

title: A narrowly endemic photosynthetic orchid is non-specific in its mycorrhizal associations.
authors: ['Pandey M', 'Sharma J', 'Taylor DL', 'Yadon VL']
source: Mol Ecol. 2013 Apr;22(8):2341-54. doi: 10.1111/mec.12249. Epub 2013 Feb 21.

title: Global transcriptome analysis and identification of a CONSTANS-like gene family in the orchid
authors: ['Chou ML', 'Shih MC', 'Chan MT', 'Liao SY', 'Hsu CT', 'Haung YT', 'Chen JJ', 'Liao DC', 'Wu
source: Planta. 2013 Jun;237(6):1425-41. doi: 10.1007/s00425-013-1850-z. Epub 2013 Feb 16.

title: Overexpression of DOSOC1, an ortholog of Arabidopsis SOC1, promotes flowering in the orchid De
authors: ['Ding L', 'Wang Y', 'Yu H']
source: Plant Cell Physiol. 2013 Apr;54(4):595-608. doi: 10.1093/pcp/pct026. Epub 2013 Feb 8.

title: SPAR methods revealed high genetic diversity within populations and high gene flow of Vanda co
authors: ['Manners V', 'Kumaria S', 'Tandon P']
source: Gene. 2013 Apr 25;519(1):91-7. doi: 10.1016/j.gene.2013.01.037. Epub 2013 Feb 8.

title: Oriental orchid (Cymbidium floribundum) attracts the Japanese honeybee (Apis cerana japonica)
authors: ['Sugahara M', 'Izutsu K', 'Nishimura Y', 'Sakamoto F']
source: Zoolog Sci. 2013 Feb;30(2):99-104. doi: 10.2108/zsj.30.99.

title: [Cloning and expression analysis of a calcium-dependent protein kinase gene in Dendrobium offi
authors: ['Zhang G', 'Zhao MM', 'Li B', 'Song C', 'Zhang DW', 'Guo SX']
source: Yao Xue Xue Bao. 2012 Nov;47(11):1548-54.

title: Sperm elution: an improved two phase recovery method for sexual assault samples.
authors: ['Hulme P', 'Lewis J', 'Davidson G']
source: Sci Justice. 2013 Mar;53(1):28-33. doi: 10.1016/j.scijus.2012.05.003. Epub 2012 May 28.

title: Marital satisfaction and physical health: evidence for an orchid effect.
authors: ['South SC', 'Krueger RF']
source: Psychol Sci. 2013 Mar 1;24(3):373-8. doi: 10.1177/0956797612453116. Epub 2013 Jan 28.

title: Optimizing virus-induced gene silencing efficiency with Cymbidium mosaic virus in Phalaenopsis
authors: ['Hsieh MH', 'Lu HC', 'Pan ZJ', 'Yeh HH', 'Wang SS', 'Chen WH', 'Chen HH']
source: Plant Sci. 2013 Mar;201-202:25-41. doi: 10.1016/j.plantsci.2012.11.003. Epub 2012 Nov 26.

title: Diversity and evolutionary patterns of bacterial gut associates of corbiculate bees.
authors: ['Koch H', 'Abrol DP', 'Li J', 'Schmid-Hempel P']
source: Mol Ecol. 2013 Apr;22(7):2028-44. doi: 10.1111/mec.12209. Epub 2013 Jan 24.

title: Fungal host specificity is not a bottleneck for the germination of Pyroleae species (Ericaceae
authors: ['Hynson NA', 'Weiss M', 'Preiss K', 'Gebauer G', 'Treseder KK']
source: Mol Ecol. 2013 Mar;22(5):1473-81. doi: 10.1111/mec.12180. Epub 2013 Jan 24.
```

```
title: Morphological, ecological and genetic aspects associated with endemism in the Fly Orchid group
authors: ['Triponez Y', 'Arrigo N', 'Pellissier L', 'Schatz B', 'Alvarez N']
source: Mol Ecol. 2013 Mar;22(5):1431-46. doi: 10.1111/mec.12169. Epub 2013 Jan 21.

title: Australian orchids and the doctors they commemorate.
authors: ['Pearn JH']
source: Med J Aust. 2013 Jan 21;198(1):52-4.

title: Orchidstra: an integrated orchid functional genomics database.
authors: ['Su CL', 'Chao YT', 'Yen SH', 'Chen CY', 'Chen WC', 'Chang YC', 'Shih MC']
source: Plant Cell Physiol. 2013 Feb;54(2):e11. doi: 10.1093/pcp/pct004. Epub 2013 Jan 16.

title: First Report of Sclerotium Rot on Cymbidium Orchids Caused by Sclerotium rolfsii in Korea.
authors: ['Han KS', 'Lee SC', 'Lee JS', 'Soh JW', 'Kim S']
source: Mycobiology. 2012 Dec;40(4):263-4. doi: 10.5941/MYCO.2012.40.4.263. Epub 2012 Dec 26.

title: Genetic linkage map of EST-SSR and SRAP markers in the endangered Chinese endemic herb Dendrob
authors: ['Lu JJ', 'Wang S', 'Zhao HY', 'Liu JJ', 'Wang HZ']
source: Genet Mol Res. 2012 Dec 21;11(4):4654-67. doi: 10.4238/2012.December.21.1.

title: OrchidBase 2.0: comprehensive collection of Orchidaceae floral transcriptomes.
authors: ['Tsai WC', 'Fu CH', 'Hsiao YY', 'Huang YM', 'Chen LJ', 'Wang M', 'Liu ZJ', 'Chen HH']
source: Plant Cell Physiol. 2013 Feb;54(2):e7. doi: 10.1093/pcp/pcs187. Epub 2013 Jan 10.

title: Adding perches for cross-pollination ensures the reproduction of a self-incompatible orchid.
authors: ['Liu ZJ', 'Chen LJ', 'Liu KW', 'Li LQ', 'Rao WH', 'Zhang YT', 'Tang GD', 'Huang LQ']
source: PLoS One. 2013;8(1):e53695. doi: 10.1371/journal.pone.0053695. Epub 2013 Jan 7.

title: Aerial roots of epiphytic orchids: the velamen radicum and its role in water and nutrient upta
authors: ['Zotz G', 'Winkler U']
source: Oecologia. 2013 Mar;171(3):733-41. doi: 10.1007/s00442-012-2575-6. Epub 2013 Jan 6.

title: The OitaAG and OitaSTK genes of the orchid Orchis italica: a comparative analysis with other C
authors: ['Salemme M', 'Sica M', 'Gaudio L', 'Aceto S']
source: Mol Biol Rep. 2013 May;40(5):3523-35. doi: 10.1007/s11033-012-2426-x. Epub 2013 Jan 1.

title: Mycorrhizal preference promotes habitat invasion by a native Australian orchid: Microtis media
authors: ['De Long JR', 'Swarts ND', 'Dixon KW', 'Egerton-Warburton LM']
source: Ann Bot. 2013 Mar;111(3):409-18. doi: 10.1093/aob/mcs294. Epub 2012 Dec 28.

title: [Plant rhabdoviruses with bipartite genomes].
authors: ['Kondo H']
source: Uirusu. 2013;63(2):143-54.

title: Functional characterization of Candida albicans Hos2 histone deacetylase.
authors: ['Karthikeyan G', 'Paul-Satyaseela M', 'Dhatchana Moorthy N', 'Gopalaswamy R', 'Narayanan S
source: F1000Res. 2013 Nov 11;2:238. doi: 10.12688/f1000research.2-238.v3. eCollection 2013.

title: Eufriesea zhangi sp. n. (Hymenoptera: Apidae: Euglossina), a new orchid bee from Brazil reveal
authors: ['Nemesio A', 'Junior JE', 'Santos FR']
source: Zootaxa. 2013 Feb 4;3609:568-82. doi: 10.11646/zootaxa.3609.6.2.

title: Specificity and preference of mycorrhizal associations in two species of the genus Dendrobium
authors: ['Xing X', 'Ma X', 'Deng Z', 'Chen J', 'Wu F', 'Guo S']
source: Mycorrhiza. 2013 May;23(4):317-24. doi: 10.1007/s00572-012-0473-8. Epub 2012 Dec 28.

title: Transcriptomic analysis of floral organs from Phalaenopsis orchid by using oligonucleotide mic
authors: ['Hsiao YY', 'Huang TH', 'Fu CH', 'Huang SC', 'Chen YJ', 'Huang YM', 'Chen WH', 'Tsai WC',
source: Gene. 2013 Apr 10;518(1):91-100. doi: 10.1016/j.gene.2012.11.069. Epub 2012 Dec 20.

title: mRNA profiling using a minimum of five mRNA markers per body fluid and a novel scoring method
authors: ['Roeder AD', 'Haas C']
source: Int J Legal Med. 2013 Jul;127(4):707-21. doi: 10.1007/s00414-012-0794-3. Epub 2012 Dec 20.

title: Monophyly or paraphyly--the taxonomy of Holcoglossum (Aeridinae: Orchidaceae).
authors: ['Xiang X', 'Li D', 'Jin X', 'Hu H', 'Zhou H', 'Jin W', 'Lai Y']
source: PLoS One. 2012;7(12):e52050. doi: 10.1371/journal.pone.0052050. Epub 2012 Dec 14.

title: A comparative study of vitrification and encapsulation-vitrification for cryopreservation of p
authors: ['Gogoi K', 'Kumaria S', 'Tandon P']
source: Cryo Letters. 2012 Nov-Dec;33(6):443-52.

title: Understanding the association between injecting and sexual risk behaviors of injecting drug us
authors: ['Suohu K', 'Humtsoe C', 'Saggurti N', 'Sabarwal S', 'Mahapatra B', 'Kermode M']
source: Harm Reduct J. 2012 Dec 18;9:40. doi: 10.1186/1477-7517-9-40.

title: A new antibacterial phenanthrenequinone from Dendrobium sinense.
authors: ['Chen XJ', 'Mei WL', 'Zuo WJ', 'Zeng YB', 'Guo ZK', 'Song XQ', 'Dai HF']
source: J Asian Nat Prod Res. 2013;15(1):67-70. doi: 10.1080/10286020.2012.740473. Epub 2012 Dec 11.

title: The Doctrine of Signatures, Materia Medica of Orchids, and the Contributions of Doctor - Orchi
authors: ['Pearn J']
source: Vesalius. 2012 Dec;18(2):99-106.

title: Reference-free comparative genomics of 174 chloroplasts.
authors: ['Kua CS', 'Ruan J', 'Harting J', 'Ye CX', 'Helmus MR', 'Yu J', 'Cannon CH']
source: PLoS One. 2012;7(11):e48995. doi: 10.1371/journal.pone.0048995. Epub 2012 Nov 20.

title: Multiple shoot induction from axillary bud cultures of the medicinal orchid, Dendrobium longic
authors: ['Dohling S', 'Kumaria S', 'Tandon P']
source: AoB Plants. 2012;2012:pls032. doi: 10.1093/aobpla/pls032. Epub 2012 Nov 5.

title: Germination failure is not a critical stage of reproductive isolation between three congeneric
authors: ['De Hert K', 'Honnay O', 'Jacquemyn H']
source: Am J Bot. 2012 Nov;99(11):1884-90. doi: 10.3732/ajb.1200381. Epub 2012 Nov 6.

title: Three new cryptic species of Euglossa from Brazil (Hymenoptera, Apidae).
authors: ['Nemesio A', 'Engel MS']
source: Zookeys. 2012;(222):47-68. doi: 10.3897/zookeys.222.3382. Epub 2012 Sep 21.

title: Two new species of Euglossa from South America, with notes on their taxonomic affinities (Hyme
authors: ['Hinojosa-Diaz IA', 'Nemesio A', 'Engel MS']
source: Zookeys. 2012;(221):63-79. doi: 10.3897/zookeys.221.3659. Epub 2012 Sep 13.

title: Genetic variation and structure within 3 endangered Calanthe species (Orchidaceae) from Korea
authors: ['Chung MY', 'Lopez-Pujol J', 'Maki M', 'Moon MO', 'Hyun JO', 'Chung MG']
source: J Hered. 2013 Mar;104(2):248-62. doi: 10.1093/jhered/ess088. Epub 2012 Nov 1.

title: Briacavatolides D-F, new briaranes from the Taiwanese octocoral Briareum excavatum.
authors: ['Wang SK', 'Yeh TT', 'Duh CY']
source: Mar Drugs. 2012 Sep;10(9):2103-10. doi: 10.3390/md10092103. Epub 2012 Sep 24.

title: Observational Research in Childhood Infectious Diseases (ORChID): a dynamic birth cohort study
authors: ['Lambert SB', 'Ware RS', 'Cook AL', 'Maguire FA', 'Whiley DM', 'Bialasiewicz S', 'Mackay IN
source: BMJ Open. 2012 Oct 31;2(6). pii: e002134. doi: 10.1136/bmjopen-2012-002134. Print 2012.

title: Microsatellite primers for the neotropical epiphyte Epidendrum firmum (Orchidaceae).
```

```
authors: ['Kartzinel TR', 'Trapnell DW', 'Glenn TC']
source: Am J Bot. 2012 Nov;99(11):e450-2. doi: 10.3732/ajb.1200232. Epub 2012 Oct 31.

title: The production of a key floral volatile is dependent on UV light in a sexually deceptive orchi
authors: ['Falara V', 'Amarasinghe R', 'Poldy J', 'Pichersky E', 'Barrow RA', 'Peakall R']
source: Ann Bot. 2013 Jan;111(1):21-30. doi: 10.1093/aob/mcs228. Epub 2012 Oct 22.

title: Exotic and indigenous viruses infect wild populations and captive collections of temperate ter
authors: ['Wylie SJ', 'Li H', 'Dixon KW', 'Richards H', 'Jones MG']
source: Virus Res. 2013 Jan;171(1):22-32. doi: 10.1016/j.virusres.2012.10.003. Epub 2012 Oct 23.

title: Orchid fleck virus: an unclassified bipartite, negative-sense RNA plant virus.
authors: ['Peng de W', 'Zheng GH', 'Zheng ZZ', 'Tong QX', 'Ming YL']
source: Arch Virol. 2013 Feb;158(2):313-23. doi: 10.1007/s00705-012-1506-5. Epub 2012 Oct 16.

title: Untangling above- and belowground mycorrhizal fungal networks in tropical orchids.
authors: ['Leake JR', 'Cameron DD']
source: Mol Ecol. 2012 Oct;21(20):4921-4. doi: 10.1111/j.1365-294X.2012.05718.x.

title: Pre-adaptations and the evolution of pollination by sexual deception: Cope's rule of speciali
authors: ['Vereecken NJ', 'Wilson CA', 'Hotling S', 'Schulz S', 'Banketov SA', 'Mardulyn P']
source: Proc Biol Sci. 2012 Dec 7;279(1748):4786-94. doi: 10.1098/rspb.2012.1804. Epub 2012 Oct 10.

title: The mirror crack'd: both pigment and structure contribute to the glossy blue appearance of the
authors: ['Vignolini S', 'Davey MP', 'Bateman RM', 'Rudall PJ', 'Moyroud E', 'Tratt J', 'Malmgren S',
source: New Phytol. 2012 Dec;196(4):1038-47. doi: 10.1111/j.1469-8137.2012.04356.x. Epub 2012 Oct 9.
```

The output for this looks like:

```
title: Sex pheromone mimicry in the early spider orchid (ophrys sphegodes):
patterns of hydrocarbons as the key mechanism for pollination by sexual
deception [In Process Citation]
authors: ['Schiestl FP', 'Ayasse M', 'Paulus HF', 'Lofstedt C', 'Hansson BS',
'Ibarra F', 'Francke W']
source: J Comp Physiol [A] 2000 Jun;186(6):567-74
```

Especially interesting to note is the list of authors, which is returned as a standard Python list. This makes it easy to manipulate and search using standard Python tools. For instance, we could loop through a whole bunch of entries searching for a particular author with code like the following:

```
In [60]: search_author = "Waits T"

In [61]: for record in records:
            if not "AU" in record:
                continue
            if search_author in record["AU"]:
                print("Author %s found: %s" % (search_author, record["SO"]))
```

Hopefully this section gave you an idea of the power and flexibility of the Entrez and Medline interfaces and how they can be used together.

10.14.2 Searching, downloading, and parsing Entrez Nucleotide records

Here we'll show a simple example of performing a remote Entrez query. In section [sec:orchids] of the parsing examples, we talked about using NCBI's Entrez website to search the NCBI nucleotide databases for info on Cypripedioideae, our friends the lady slipper orchids. Now, we'll look at how to automate that process using a Python script. In this example, we'll just show how to connect, get the results, and parse them, with the Entrez module doing all of the work.

First, we use EGQuery to find out the number of results we will get before actually downloading them. EGQuery will tell us how many search results were found in each of the databases, but for this example we are only interested in nucleotides:

```
In [62]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.egquery(term="Cypripedioideae")
         record = Entrez.read(handle)
         for row in record["eGQueryResult"]:
             if row["DbName"]=="nuccore":
                 print(row["Count"])
```

```
4247
```

So, we expect to find 814 Entrez Nucleotide records (this is the number I obtained in 2008; it is likely to increase in the future). If you find some ridiculously high number of hits, you may want to reconsider if you really want to download all of them, which is our next step:

```
In [63]: from Bio import Entrez
         handle = Entrez.esearch(db="nucleotide", term="Cypripedioideae", retmax=814)
         record = Entrez.read(handle)
```

Here, `record` is a Python dictionary containing the search results and some auxiliary information. Just for information, let's look at what is stored in this dictionary:

```
In [64]: print(record.keys())
```

```
dict_keys(['TranslationStack', 'IdList', 'TranslationSet', 'QueryTranslation', 'Count', 'RetStart',
```

First, let's check how many results were found:

```
In [65]: print(record["Count"])
```

```
4247
```

which is the number we expected. The 814 results are stored in `record['IdList']`:

```
In [66]: len(record["IdList"])
```

```
Out[66]: 814
```

Let's look at the first five results:

```
In [67]: record["IdList"][:5]
```

```
Out[67]: ['874509867', '874509089', '844174433', '937957673', '694174838']
```

[sec:entrez-batched-efetch] We can download these records using `efetch`. While you could download these records one by one, to reduce the load on NCBI's servers, it is better to fetch a bunch of records at the same time, shown below. However, in this situation you should ideally be using the history feature described later in Section *History and WebEnv*.

```
In [68]: idlist = ",".join(record["IdList"][:5])
         print(idlist)
```

```
874509867,874509089,844174433,937957673,694174838
```

```
In [69]: handle = Entrez.efetch(db="nucleotide", id=idlist, retmode="xml")
         records = Entrez.read(handle)
         len(records)
```

```
Out[69]: 5
```

Each of these records corresponds to one GenBank record.

```
In [70]: print(records[0].keys())
```

```
dict_keys(['GBSeq_division', 'GBSeq_moltype', 'GBSeq_definition', 'GBSeq_topology', 'GBSeq_locus', '
In [71]: print(records[0]["GBSeq_primary-accession"])
```

```
KP644081
```

```
In [72]: print(records[0]["GBSeq_other-seqids"])
```

```
['gnl|uoguelph|SCBI449-14.rbcLa', 'gb|KP644081.1|', 'gi|874509867']
```

```
In [73]: print(records[0]["GBSeq_definition"])
```

```
Cypripedium calceolus voucher SNP_13_0359 ribulose-1,5-bisphosphate carboxylase/oxygenase large subun
```

```
In [74]: print(records[0]["GBSeq_organism"])
```

```
Cypripedium calceolus
```

You could use this to quickly set up searches – but for heavy usage, see Section *History and WebEnv*.

10.14.3 Searching, downloading, and parsing GenBank records

The GenBank record format is a very popular method of holding information about sequences, sequence features, and other associated sequence information. The format is a good way to get information from the NCBI databases at http://www.ncbi.nlm.nih.gov/.

In this example we'll show how to query the NCBI databases,to retrieve the records from the query, and then parse them using `Bio.SeqIO` - something touched on in Section [sec:SeqIO_GenBank_Online]. For simplicity, this example *does not* take advantage of the WebEnv history feature – see Section *History and WebEnv* for this.

First, we want to make a query and find out the ids of the records to retrieve. Here we'll do a quick search for one of our favorite organisms, *Opuntia* (prickly-pear cacti). We can do quick search and get back the GIs (GenBank identifiers) for all of the corresponding records. First we check how many records there are:

```
In [75]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.egquery(term="Opuntia AND rpl16")
         record = Entrez.read(handle)
         for row in record["eGQueryResult"]:
             if row["DbName"]=="nuccore":
                 print(row["Count"])
```

```
26
```

Now we download the list of GenBank identifiers:

```
In [76]: handle = Entrez.esearch(db="nuccore", term="Opuntia AND rpl16")
         record = Entrez.read(handle)
         gi_list = record["IdList"]
         gi_list
```

```
Out[76]: ['377581039', '330887241', '330887240', '330887239', '330887238', '330887237', '330887236',
```

Now we use these GIs to download the GenBank records - note that with older versions of Biopython you had to supply a comma separated list of GI numbers to Entrez, as of Biopython 1.59 you can pass a list and this is converted for you:

```
In [77]: gi_str = ",".join(gi_list)
         handle = Entrez.efetch(db="nuccore", id=gi_str, rettype="gb", retmode="text")
```

If you want to look at the raw GenBank files, you can read from this handle and print out the result:

```
In [78]: text = handle.read()
         print(text)
```

```
LOCUS       HQ621368                 399 bp    DNA     linear   PLN 26-FEB-2012
DEFINITION  Opuntia decumbens voucher Martinez & Eggli 146a (ZSS) ribosomal
            protein L16 (rpl16) gene, partial cds; chloroplast.
ACCESSION   HQ621368
VERSION     HQ621368.1  GI:377581039
KEYWORDS    .
SOURCE      chloroplast Opuntia decumbens
  ORGANISM  Opuntia decumbens
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 399)
  AUTHORS   Arakaki,M., Christin,P.A., Nyffeler,R., Lendel,A., Eggli,U.,
            Ogburn,R.M., Spriggs,E., Moore,M.J. and Edwards,E.J.
  TITLE     Contemporaneous and recent radiations of the world's major
            succulent plant lineages
  JOURNAL   Proc. Natl. Acad. Sci. U.S.A. 108 (20), 8379-8384 (2011)
   PUBMED   21536881
REFERENCE   2  (bases 1 to 399)
  AUTHORS   Arakaki,M., Christin,P.-A., Nyffeler,R., Eggli,U., Ogburn,R.M.,
            Spriggs,E., Moore,M.J. and Edwards,E.J.
  TITLE     Direct Submission
  JOURNAL   Submitted (15-NOV-2010) Department of Ecology and Evolutionary
            Biology, Brown University, 80 Waterman St., Providence, RI 02912,
            USA
COMMENT     ##Assembly-Data-START##
            Assembly Method       :: MIRA V3rc4; Geneious v. 4.8
            Sequencing Technology :: 454
            ##Assembly-Data-END##
FEATURES             Location/Qualifiers
     source          1..399
                     /organism="Opuntia decumbens"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /specimen_voucher="Martinez & Eggli 146a (ZSS)"
                     /db_xref="taxon:867482"
                     /tissue_type="stem"
                     /note="authority: Opuntia decumbens Salm-Dyck"
     gene            <1..>399
                     /gene="rpl16"
     CDS             <1..>399
                     /gene="rpl16"
                     /codon_start=1
                     /transl_table=11
                     /product="ribosomal protein L16"
                     /protein_id="AFB70658.1"
                     /db_xref="GI:377581040"
                     /translation="NPKRTRFCKQHRGRMKGISYRGNRICFGRYALQALEPAWITSRQ
                     IEAGRRAMTRNARRGGKIWVRIFPDKPVTVKSAESRMGSGKGSHLYWVVVVKPGRILY
                     EISGVSENIARRAISIAASKMPVRTQFIISG"
ORIGIN
        1 aaccccaaaa gaaccagatt ctgtaaacaa catagaggaa gaatgaaggg aatatcttat
       61 cgggggaatc gtatttgttt cggaagatat gctcttcagg cacttgagcc tgcttggatc
      121 acgtctagac aaatagaagc aggtcggcga gcaatgacgc gaaatgcacg ccgcggtgga
      181 aaaatatggg tacgtatatt tccagacaaa ccagttacag taaaatctgc ggaaagccgt
      241 atgggttcgg ggaaaggatc ccacctatat tgggtagttg ttgtcaaacc cggtcgaata
      301 ctttatgaaa taagcggagt atcagaaaat atagcccgaa gggctatctc gatagcggca
      361 tctaaaatgc ctgtacgaac tcaattcatt atttcagga
```

```
//

LOCUS       HM041482                1197 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Cylindropuntia tunicata ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041482
VERSION     HM041482.1  GI:330887241
KEYWORDS    .
SOURCE      chloroplast Cylindropuntia tunicata
  ORGANISM  Cylindropuntia tunicata
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Cylindropuntia.
REFERENCE   1  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1197
                     /organism="Cylindropuntia tunicata"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:766221"
     gene            <1..>1197
                     /gene="rpl16"
     misc_feature    <1094..>1197
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gtgatatacg aaacagtaag agcccatagt atgaagtatg aactaataac tatagaacta
       61 ataaccaact catcgcatca cattatctgg atccaaagaa gcagtcaaga taggatattt
      121 tggtcctatc attgcagcaa ctgaattttt tttttcataa acaagaaatc gaatgagttg
      181 tcaagcaaaa gaaaaaaaaa aaaagaaaaa tatacnttaa aggaggggga tgcggataaa
      241 tggaaaggcg aaagaaagaa aaaaatgaat ctaaatgata tacgattcca ctatgtaagg
      301 tctttgaatc atatcataaa agacaatgta ataaagcatg aatacagatt cacacataat
      361 tatctgatat gaatctattc atagaaaaaa gaaaaaagta agagcctccg gccaataaag
      421 actaagaggg gttggctcaa aaacaaagtt cattaagagc tcccattgta gaattcagac
      481 ctaatcatta atcaagaagc gatgggaacg atgtaatcca tgaatacaga agattcaatt
      541 gaaaaaagaa tcctaatgat tcattgggga ggatggcgga acgaaccaga gaccaattca
      601 tctattctga aaagtgataa actaatccta taaaactaaa atagatattg aaagagtaaa
      661 tattcgcccg cgaaaattcc ttttttatta aattgctcat attttatttt agcaatgcaa
      721 tctaataaaa tatatctata caaaaaaaca tagacaaact atatatataa tatttcaaat
      781 tcccttatat atccaaatat aaaaatatct aataaattag atgaatatca aagaatctat
      841 tgatttagtg tattattaaa tgtatatctt aattcaatat tattattcta ttcattttta
      901 ttcattttca aatttataat atattaatct atatattaat ttagaattct attctaattc
      961 gaattcaatt tttaaatatt cattcatatt caattaaaat tgaaattttt tcattcgcga
     1021 ggagccggat gagaagaaac tctcatgtcc ggttctgtag tagagatgga attaagaaaa
     1081 aaccatcaac tataacccca aaagaaccag attctgtaaa caacatagag gaagaatgaa
     1141 gggaatatct tatcgggga atcgtatttg tttcggaaaa tatgctctca ggcacga
//

LOCUS       HM041481                1200 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia palmadora ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041481
VERSION     HM041481.1  GI:330887240
KEYWORDS    .
SOURCE      chloroplast Opuntia palmadora
  ORGANISM  Opuntia palmadora
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1200)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1200)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1200
                     /organism="Opuntia palmadora"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001118"
     gene            <1..>1200
                     /gene="rpl16"
     misc_feature    <1098..>1200
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 tgatatacga aaagtaagag cccatagtat gaagtatgaa ctaataacta tagaactaat
       61 aaccaactca tcgcatcaca ttatctggat ccaaagaagc agtcaagata ggatattttg
      121 gtcctatcat tgcagcaact gaattttttt ttcataaaca agaaatcaaa tgagttgtca
      181 agcaaaagaa aaaaaaaaga aaaatatacn ttaaaggagg gggatgcgga taaatggaaa
      241 ggcgaaagaa agaaaaaaat gaatctaaat gatatacgat tccactatgt aaggtctttg
      301 aatcatatca taaaagacaa tgtaataaag catgaataca gattcacaca taattatctg
      361 atatgaatct attcatagaa aaaagaaaaa agtaagagcc tccgggccaa taaagactaa
      421 gagggttggg ctcaagaaca aagttcatta agagctccat tgtagaattc agacctaatc
      481 attaatcaag aagcgatggg aacgatgtaa tccatgaata cagaagattc aattgaaaaa
      541 gaatcctaat gattcattgg gaaggatggc ggaacgaacc agagaccaat tcatctattc
      601 tgaaaagtga taaactaatc ctataaaact aaaatagata ttgaaagagt aaatattcgc
      661 ccgcgaaaat tcctttttta ttaaattgct cacattttat tttagcaatg caatctaata
      721 aaatatatct atacaaaaaa atatagacaa actatatata taatatattt caaatttcct
      781 tatatatcct aatataaaaa tatctaataa attagatgaa tatcaaagaa tctattgatt
      841 tagtgtatta ttaaatgtat atcttaattc aatattatta ttctattcat ttttattatt
      901 catttttatt cattttcaaa tttagaatat attaatctat atattaattt agaattctat
      961 tctaattcga attcaatttt taaatattca tattcaatta aaattgaaat tttttcattc
     1021 gcgaggagcc ggatgagaag aaactctcac gtccggttct gtagtagagg tggaattaag
     1081 aaaaaaccat caactataac cccaaaagaa ccagattctg taaacaacat agaggaagaa
     1141 tgaagggaat atcttatcgg gggaatcgta tttgtttcgg aagatatgct ctcagcacga
//

LOCUS       HM041480                1153 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia microdasys ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041480
VERSION     HM041480.1  GI:330887239
KEYWORDS    .
SOURCE      chloroplast Opuntia microdasys
  ORGANISM  Opuntia microdasys
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1153)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1153)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1153
                     /organism="Opuntia microdasys"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:169217"
     gene            <1..>1153
                     /gene="rpl16"
     misc_feature    <1079..>1153
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gcccatagta tgaagtatga actaataact atagaactaa taaccaactc atcgcatcac
       61 attatctgga tccaaagaag cagtcaagat aggatatttt ggtcctatca ttgcagcaac
      121 tgaatttttt ttttcataaa caagaaatca aatgagttgt caagcaaaag aaaaaaaaaa
      181 aaaaaaatat actttaaggg ggggggatgg ggataaaggg aaaggggaaa aaaaaaaaaa
      241 aatgaatcta aatgatatac aattccacta tgaaaggtct ttgaatcata tcaaaaaaaa
      301 caatgtaata aagcaggaat acagattccc acataattat ctgatatgaa tcttttcata
      361 aaaaaaaaaa aaaagtaaga gcctccggcc aataaagact aagagggttg gctcaagaac
      421 aaagttcatt aagggctcca ttgtagaatt cagacctaat cattaatcaa gaggcgatgg
      481 gaacgatgta atccatgaat acagaagatt caattgaaaa agaatcctaa tgattcattg
      541 ggaaggatgg cggaacgaac cagagaccaa ttcatctatt ctgaaaagtg aaaaactaat
      601 cctataaaac taaaatagat attgaaagag taaatattcg cccgcgaaaa ttcctttttt
      661 attaaattgc tcacatttta ttttagcaat gcaatctaat aaaatatatc tatacaaaaa
      721 aatatagaca aactatatat ataatatatt tcaaatttcc ttatatatcc taatataaaa
      781 atatctaata aattagatga atatcaaaga atctattgat ttagtgtatt attaaatgta
      841 tatcttaatt caatattatt attctattca tttttattat tcatttttat tcattttcaa
      901 atttagaata tattaatcta tatattaatt tataattcta ttctaattcg aattcaattt
      961 ttaaatattc atattcaatt aaaattgaaa ttttttcatt cgcgaggagc cggatgagaa
     1021 gaaactctca cgtccggttc tgtagtagag gtggaattaa gaaaaaacca tcaactataa
     1081 ccccaaaaga accagattct gtaaacaaca tagaggaaga atgaagggaa tatcttatcg
     1141 ggggatatcg tat
//

LOCUS       HM041479                1197 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia megasperma ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041479
VERSION     HM041479.1  GI:330887238
KEYWORDS    .
SOURCE      chloroplast Opuntia megasperma
  ORGANISM  Opuntia megasperma
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1197
                     /organism="Opuntia megasperma"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001117"
     gene            <1..>1197
                     /gene="rpl16"
     misc_feature    <1098..>1197
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gatatacgaa aagtaagagc ccatagtatg aagtatgaac taataactat agaactaata
       61 accaactcat cgcatcacat tatccggatc caaagaagca gtcaagatag gatattttgg
      121 tcctatcatt gcagcaactg aatttttttt tcataaacaa gaaatcaaat gagttgtcaa
      181 gcaaaagaaa aaaaaaaaag aaaaatatac tttaaaggag ggggatgcgg ataaatggaa
      241 aggcgaaaga aagaaaaaaa tgaatctaaa tgatatacga ttccnctatg taaggtcttt
      301 gaatcatatc ataaaagaca atgtaataaa gcatgaatac agattcacac ataattatct
      361 gatatgaatc tattcataga aaaaagaaaa aagtaagagc ctccgggcca ataaagacta
      421 agagggttgg ctcaagaaca aagttcatta agagctccat tgtagaattc agacctaatc
      481 attaatcaag aagcgatggg aacgatgtaa tccatgaata cagaagattc aattgaaaaa
      541 gaatcctaat gattcattgg gaaggatggc ggaacgaacc agagaccaat tcatctattc
      601 tgaaaagtga taaactaatc ctataaaact aaaatagata ttgaaagagt aaatattcgc
      661 ccgcgaaaat tcctttttta ttaaattgct cacattttat tttagcaatg caatctaata
      721 aaatatatct atacaaaaaa atatagacaa actatatata taatatattt caaatttcct
      781 tatatatcct aatataaaaa tatctaataa attagatgaa tatcaaagaa tctattgatt
      841 tagtgtatta ttaaatgtat atcttaattc aatattttta ttctattcat ttttattatt
      901 catttttatt cattttcaaa tttagaatat attaatctat atattaattt agaattctat
      961 tctaattcga attcaatttt taaatattca tattcaatta aaattgaaat tttttcattc
     1021 gcgaggagcc ggatgagaag aaactctcac gtccggttct gtagtagagg tggaattaag
     1081 aaaaaaccat caactataac cccaaaagaa ccagattctg taaacaacat agaggaagaa
     1141 tgaagggaat atcttatcgg gggaatcgta tttgtttcgg aagatatgct ctcagca
//

LOCUS       HM041478                1187 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia macbridei ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041478
VERSION     HM041478.1  GI:330887237
KEYWORDS    .
SOURCE      chloroplast Opuntia macbridei
  ORGANISM  Opuntia macbridei
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1187)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1187)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1187
                     /organism="Opuntia macbridei"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001116"
     gene            <1..>1187
                     /gene="rpl16"
     misc_feature    <1090..>1187
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 aaaagtaaga gcccatagta tgaagtatga actaataact atagaactaa taaccaactc
       61 atcgcatcac attatctgga tccaaagaag cagtcaagat aggatatttt ggtcctatca
      121 ttgcagcaac tgaatttttt tttcataaac aagaaatcaa atgagttgtc aagcaaaaga
      181 aaaaaaaaaa agaaaaatat acattaaagg agggggatgc ggataaatgg aaaggcgaaa
      241 gaaagaaaaa aatgaatcta aatgatatac gattccacta tgtaaggtct ttgaatcata
      301 tcataaaaga caatgtaata aagcatgaat acagattcac acataattat ctgatatgaa
      361 tctattcata gaaaaaagaa aaaagtaaga gcctccggcc aataaagact aagagggttg
      421 gctcaagaac aaagttcatt aagggctcca tttgtagaat tcagacctaa tcattaatca
      481 agaagcgatg ggaacgatgt aattccatga atacagaaga ttcaattgaa aaagatccta
      541 atgattcatt gggaaggatg gcggacgaac cagagaccaa ttcatctatt ctgaaaagtg
      601 ataaactaat cctataaaac taaaatagat attgaaagag taaatattcg cccgcgaaaa
      661 ttccttttt attaaattgc tcacatttta ttttagcaat gcaatctaat aaaatatatc
      721 tatacaaaaa aaatatagac aaactatata tataatatat ttcaaatttc cttatatatc
      781 ctaatataaa aatatctaat aatttagatg aatatcaaag aatctattga tttagtgtat
      841 tattaaatgt atatcttaat tcaatattat tattctattc atttttatta ttcattttta
      901 ttcattttca aatttagaat atattaatct atatattaat ttagaattct attctaattc
      961 gaattcaatt tttaaatatt catattcaat taaaattgaa attttttcat tcgcgaggag
     1021 ccggatgaga agaaactctc acgtccggtt ctgtagtaga ggtggaatta agaaaaaacc
     1081 atcaactata accccaaaag aaccagattc tgtaaacaac atagaggaag aatgaaggga
     1141 atatcttatc gggggaatcg tatttgtttc ggaagatatg ctctcag
//

LOCUS       HM041477                1197 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Cylindropuntia leptocaulis ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041477
VERSION     HM041477.1  GI:330887236
KEYWORDS    .
SOURCE      chloroplast Cylindropuntia leptocaulis
  ORGANISM  Cylindropuntia leptocaulis
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Cylindropuntia.
REFERENCE   1  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1197)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1197
                     /organism="Cylindropuntia leptocaulis"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:866983"
     gene            <1..>1197
                     /gene="rpl16"
     misc_feature    <1096..>1197
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 ttgtgngnct cctgaagagt aggagcccct agtatgaagt atgaactaat aactatagaa
       61 ctaataacca actcatcgca tcacattatc cggatccaaa aaagcagtca agataggata
      121 ttttggtcct atcattgcag caactgaatt tttttttca taaacaagaa atcgaatgag
      181 ttgtcaagca aaagaaaaaa aaagaaaaat atactttaaa ggagggggat gcggataaat
      241 ggaaaggcga aagaaagaaa aaaatgaatc taaatgatat aggattcccc tatgtaaggt
      301 ctttgaatca tatcataaaa gacaatgtaa taaagcatga atacagattc ccacataatt
      361 atctgatatg aatctattcc tagaaaaaag aaaaaagtaa gagcctccgg ccaataaaga
      421 ctaagagggt tggctcaaga acaaagttca ttaaaagctc ccttgtagaa ttcagaccta
      481 atcnttaatc aagaagcgat gggaacgatg taatccctga atacagaaga ttcaattgaa
      541 aaagaatcct aatgattcat tgggaaggat ggcggaacga accagagacc aattcatcta
      601 ttctgaaaag tgataaacta atcctataaa actaaaatag atattgaaag agtaaatatt
      661 cgcccgcgaa atttcctttt ttattaaatt gctcatattt ttttttagca atgcaatcta
      721 ataaaatata tctctacaaa aaaacataga caaactatat atatatatat atataatatt
      781 tcaaattccc ttatatatcc aaatataaaa atatctaata aattagatga atatcaaaga
      841 atctattgat ttagtgtatt attaaatgta tatcttaatt caatattatt attctattca
      901 tttttattca ttttcaaatt tataatatat taatctatat attaatatag aattctattc
      961 taattcgaat tcaattttta aatattcata ttcaattaaa attgaaattt tttcattcgc
     1021 gaggagccgg atgagaagaa actctcatgt ccggttctgt agtagagatg gaattaagaa
     1081 aaaaccatca actataaccc caaaagaacc ggattctgta aacaacatag aggaagaatg
     1141 aagggaatat cttgtcgggg gaatcgatnn gtncggaant natgntcgcn gcgcgcc
//

LOCUS       HM041476                1205 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia lasiacantha ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041476
VERSION     HM041476.1  GI:330887235
KEYWORDS    .
SOURCE      chloroplast Opuntia lasiacantha
  ORGANISM  Opuntia lasiacantha
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1205)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1205)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1205
                     /organism="Opuntia lasiacantha"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:547104"
     gene            <1..>1205
                     /gene="rpl16"
     misc_feature    <1103..>1205
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gggcccnnna ngangaaaag tagagcccat agtatgaagt atgaactaat aactatagaa
       61 ctaataacca actcatcgca tcacattatc tggatccaaa gaagcagtca agataggata
      121 ttttggtcct atcattgcag caactgaatt ttttttttca taaacaagaa atcaaatgag
      181 ttgtcaagca aaagaaaaaa aaaaagaaaa atatccttta aaggaggggg atgcggataa
      241 atggaaaggc gaaagaaaga aaaaaatgaa tctaaatgat atacgattcc cctatgtaag
      301 gtctttgaat catatcataa aagacaatgt aataaagcat gaatacagat tcccccataa
      361 ttatctgata tgaatctatt cctagaaaaa agaaaaaagt aagagcctcc ggccaataaa
      421 gactaagagg gttggctcaa gaacaaagtt cattaagggc tccattgtag aattcagacc
      481 taatcattaa tcaagaggcg atgggaacga tgtaatccat gaatacagaa gattcaattg
      541 aaaaagaatc ctaatgattc attgggaagg atggcggaac gaaccagaga ccaattcatc
      601 tattctgaaa agtgataaac taatcctata aaactaaaat agatattgaa agagtaaata
      661 ttcgcccgcg aaaattcctt ttttattaaa ttgctcacat tttattttag caatgcaatc
      721 taataaaatc tatctataca aaaaaatata gacaaactat atatataata tatttcaaat
      781 ttccttatat atcctaatat aaaaatatct aataaattag atgaatatca aagaatctat
      841 tgatttagtg tattattaaa tgtatatctt aattcaatat tattattcta ttcattttta
      901 ttattcattt ttattcattt tcaaatttag aatatattaa tctatatatt aatttataat
      961 tctattctaa ttcgaattca atttttaaat attcatattc aattaaaatt gaaatttttt
     1021 cattcgcgag gagccggatg agaagaaact ctcacgtccg gttctgtagt agaggtggaa
     1081 ttaagaaaaa accatcaact ataaccccaa aagaaccaga ttctgtaaac aacatagagg
     1141 aagaatgaag ggaatatctt atcgagggaa tcgtatttgt ttcggaagat agtnctngcn
     1201 nggtg
//

LOCUS       HM041474                1163 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia helleri ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041474
VERSION     HM041474.1  GI:330887233
KEYWORDS    .
SOURCE      chloroplast Opuntia helleri
  ORGANISM  Opuntia helleri
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1163)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1163)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1163
                     /organism="Opuntia helleri"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001115"
     gene            <1..>1163
                     /gene="rpl16"
     misc_feature    <1081..>1163
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gagcccatag tatgaagtat gaactaataa ctatagaact aataaccaac tcatcgcatc
       61 acattatccg gatccaaaga agcagtcaag ataggatatt ttggtcctat cattgcagca
      121 actgaatttt tttttcataa acaagaaatc aaatgagttg tcaagcaaaa gaaaaaaaaa
      181 aaagaaaaat atacattaaa ggagggggat gcggataaat ggaaaggcga aagaaagaaa
      241 aaaatgaatc taaatgatat acgattccnc tatgtaaggt ctttgaatca tatcataaaa
      301 gacaatgtaa taaagcatga atacagattc acacataatt atctgatatg aatctattca
      361 tagaaaaaag aaaaaagtaa gagcctccgg ccaataaaga ctaagagggt tggctcaaga
      421 acaaagttca ttaagggctc cattgtagaa ttcagaccta atcattaatc aagaagcgat
      481 gggaacgatg taatccatga atacagaaga ttcaattgaa aaagaatcct aatgattcat
      541 tgggaaggat ggcggaacga accagagacc aattcatcta ttctgaaaag tgataaacta
      601 atcctataaa actaaaatag atattgaaag agtaaatatt cgcccgcgaa aattcctttt
      661 ttattaaatt gctcacattt tattttagca atgcaatcta ataaaatata tctatacaaa
      721 aaaatataga caaactatat atataatata tttaaaattt ccttatatat cctaatataa
      781 aaatatctaa taaattagat gaatatcaaa gaatctattg atttagtgta ttattaaatg
      841 tatatcttaa ttcaatattt ttattctatt catttttatt attcattttt attcattttc
      901 aaatttagaa tatattaatc tatatattaa tttagaattc tattctaatt cgaattcaat
      961 ttttaaatat tcatattcaa ttaaaattga aatttttttca ttcgcgagga gccggatgag
     1021 aagaaactct cacgtccggt tctgtagtag aggtggaatt aagaaaaaac catcaactat
     1081 aaccccaaaa gaaccagatt ctgtaaacaa catagaggaa gaatgaaggg aatatcttat
     1141 cgggggaatc gtatttgttt cgg
//

LOCUS       HM041473                1203 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia excelsa ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041473
VERSION     HM041473.1  GI:330887232
KEYWORDS    .
SOURCE      chloroplast Opuntia excelsa
  ORGANISM  Opuntia excelsa
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1203)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1203)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1203
                     /organism="Opuntia excelsa"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:867487"
     gene            <1..>1203
                     /gene="rpl16"
     misc_feature    <1103..>1203
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 ccgnncnttg nnanacagaa nagtagagcc cnttntntga agtatgaact aatcactatt
```

```
       61 gaactaatcc ccnactcatc gcatcacatt atctggatcc aaagaagcag tcaagatagg
      121 atattttggt cctatcattg cagcaactga attttttttt tcctaaacaa gaaatcaaat
      181 gagttgtcaa gcaaaagaaa aaaaagaaaa atatacatta aaggaggggg atgcggataa
      241 atggaaaggc gaaagaaaga aaaaaatgaa tctaaatgat atacgattcc cctatgtaag
      301 gtctttgaat catatcataa aagacaatgt aataaagcat gaatacagat tcccacataa
      361 ttatctgata tgaatctatt catagaaaaa agaaaaaagt aagagcctcc ggccaataaa
      421 gactaaaagg gttggctcaa gaacaaagtt cattaagggc tccattgtaa aattcagacc
      481 taatcattaa tcaagaggcg atgggaacga tgtaatccat gaatacagaa gattcaattg
      541 aaaaagaatc ctaatgattc attgggaagg atggcggaac gaaccagaga ccaattcatc
      601 tattctgaaa agtgataaac taatcctata aaactaaaat agatattgaa agagtaaata
      661 ttcgcccgcg aaaattcctt ttttattaaa ttgctcacat tttattttag caatgcaatc
      721 taataaaatc tatctataca aaaaaatata gacaaactat atatataata tatttcaaat
      781 ttccttatat atcctaatat aaaaatatct aataaattag atgaatatca aagaatctat
      841 tgatttagtg tattattaaa tgtatatctt aattcaatat tattattcta ttcattttta
      901 ttattcattt ttattcattt tcaaatttag aatatattaa tctatatatt aatttagaat
      961 tctattctaa ttcgaattca atttttaaat attcatattc aattaaaatt gaaatttttt
     1021 cattcgcgag gagccggatg agaagaaact ctcacgtccg gttctgtagt agaggtggaa
     1081 ttaagaaaaa accatcaact ataaccccaa aagaaccaga ttctgtaaac aacatagagg
     1141 aagaatgaag ggaatatctt atcgggggaa tcgtatngtg cnggctngtg cancgcgggc
     1201 nng
//

LOCUS       HM041472                1182 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Opuntia echios ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041472
VERSION     HM041472.1  GI:330887231
KEYWORDS    .
SOURCE      chloroplast Opuntia echios
  ORGANISM  Opuntia echios
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1182)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1182)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1182
                     /organism="Opuntia echios"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:412453"
     gene            <1..>1182
                     /gene="rpl16"
     misc_feature    <1085..>1182
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gtaagagccc atagtatgaa gtatgaacta ataactatag aactaataac caactcatcg
       61 catcacatta tccggatcca aagaagcagt caagatagga tattttggtc ctatcattgc
      121 agcaactgaa tttttttttc ataaacaaga aatcaaatga gttgtcaagc aaaagaaaaa
      181 aaaaaaaaaa aaatatacat taaaggaggg ggatgcggat aaatggaaag gcgaaagaaa
      241 gaaaaaaatg aatctaaatg atatacgatt ccactatgta aggtctttga atcatatcat
      301 aaaagacaat gtaataaagc atgaatacag attcacacat aattatctga tatgaatcta
      361 ttcatagaaa aaagaaaaaa gtaagagcct ccggccaata aagactaaga ggttgggctc
      421 aagaacaaag ttcattaagg gctccattgt agaattcaga cctaatcatt aatcaagaag
      481 cgatgggaac gatgtaatcc atgaatacag aagattcaat tgaaaaagaa tcctaatgat
      541 tcattgggaa ggatggcgga acgaaccaga gaccaattca tctattctga aaagtgataa
      601 actaatccta taaaactaaa atagatattg aaagagtaaa tattcgcccg cgaaaattcc
      661 tttttttatta aattgctcac attttatttt agcaatgcaa tctaataaaa tatatctata
      721 caaaaaaata tagacaaact atatatataa tatatttcaa atttccttat atatcctaat
      781 ataaaaatat ctaataaatt agatgaatat caaagaatct attgatttag tgtattatta
      841 aatgtatatc ttaattcaat attattattc tattcatttt tattattcat ttttattcat
      901 tttcaaattt agaatatatt aatctatata ttaatttaga attctattct aattcgaatt
      961 caatttttaa atattcatat tcaattaaaa ttgaaatttt ttcattcgcg aggagccgga
     1021 tgagaagaaa ctctcacgtc cggttctgta gtagaggtgg aattaagaaa aaaccatcaa
     1081 ctataacccc aaaagaacca gattctgtaa acaacataga ggaagaatga agggaatatc
     1141 ttatcggggg aatcgtattt gtttcggaag atggctacta ta
//

LOCUS       HM041469                1189 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea sp. THH-2011 ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041469
VERSION     HM041469.1  GI:330887228
KEYWORDS    .
SOURCE      chloroplast Opuntia sp. THH-2011
  ORGANISM  Opuntia sp. THH-2011
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1189)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1189)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1189
                     /organism="Opuntia sp. THH-2011"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001114"
     gene            <1..>1189
                     /gene="rpl16"
     misc_feature    <1096..>1189
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 atatacgaaa aagtagagcc catagtatga agtatgaact aataactata gaactaataa
       61 ccaactcatc gcatcacatt atctggatcc aaagaagcag tcaagatagg atattttggt
      121 cctatcattg cagcaactga atttttttt cataaacaag aaatcaaatg agttgtcaag
      181 caaaagaaaa aaaaaaaaga aaaatatact ttaagggagg gggatgcgga taaatggaaa
      241 ggcgaaagaa agaaaaaaat gaatctaaat gatatacgat tccactatgt aaggtctttg
      301 aatcatatca taaaagacaa tgtaataaag catgaataca gattcacaca taattatctg
      361 gtatgaatct attcatagaa aaaagaaaaa agtaagaccc tccggccaat aaagactaag
      421 agggttggct caagaacaaa gttcattaag ggctccattg tagaattcag acctaatcat
      481 taatcaagaa gcgatgggaa cgatgtaatc catgaataca gaagattcaa ttgaaaaaga
      541 atcctaatga ttcattggga aggatggcgg aacgaaccag agaccaattc atctattctg
      601 aaaagtgata aactaatcct ataaaactaa aatagatatt gaaagagtaa atattcgccc
      661 gcgaaaattc ctttttttatt aaattgctca cattttattt tagcaatgca atctaataaa
      721 atatatctat acaaaaaaat atagacaaac tatatatata atatatttca aatttcctta
      781 tatatcctaa tataaaaata tctaataaat tagatgaata tcaaagaatc tattgattta
      841 gtgtattatt aaatgtatat cttaattcaa tattattatt ctattcattt ttattattca
      901 tttttattca ttttcaaatt tagaatatat taatctatat attaatttag aattctattc
      961 taattcgaat tcaattttta aatattcata ttcaattaaa attgaaattt tttcattcgc
     1021 gaggagccgg atgagaagaa actctcacgt ccggttctgt agtagaggtg gaattaagaa
     1081 aaaaccatca actataaccc caaaagaacc agattctgta aacaacatag aggaagaatg
     1141 aagggaatat cttatcgggg gaatcgtatt tgtttcggaa gatatgctc
//

LOCUS       HM041468                1202 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea lutea ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041468
VERSION     HM041468.1  GI:330887227
KEYWORDS    .
SOURCE      chloroplast Opuntia lutea
  ORGANISM  Opuntia lutea
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1202)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1202)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1202
                     /organism="Opuntia lutea"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001113"
     gene            <1..>1202
                     /gene="rpl16"
     misc_feature    <1099..>1202
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gatatacgaa aaagtaagag cccatagtat gaagtatgaa ctaataacta tagaactaat
       61 aaccaactca tcgcatcaca ttatctggat ccaaagaagc agtcaagata ggatattttg
      121 gtcctatcat tgcagcaact gaatttttttt ttcataaaca agaaatcaaa tgagttgtca
      181 agcaaaagaa aaaaaaaaaa gaaaaatata ctttaaggga gggggatgcg gataaatgga
      241 aaggcgaaag aaagaaaaaa atgaatctaa atgatatacg attcccccta tgtaaggtct
      301 ttgaatcata tcataaaaga caatgtaata aagcatgaat acagattcac acataattat
      361 ctgatatgaa tctattcata gaaaaaagaa aaaagtaaga ccctccggcc aataaagact
      421 aagagggttg gctcaagaac aaagttcatt aagggctcca ttgtagaatt cagacctaat
      481 cattaatcaa gaagcgatgg gaacgatgta atccatgaat acagaagatt caattgaaaa
      541 agaatcctaa tgattcattg ggaaggatgg cggaacgaac cagagaccaa ttcatctatt
      601 ctgaaaagtg ataaactaat cctataaaac taaaatagat attgaaagag taaatattcg
      661 cccgcgaaaa ttcctttttt attaaattgc tcacatttta ttttagcaat gcaatctaat
      721 aaaatatatc tatacaaaaa aatatagaca aactatatat ataatatatt tcaaatttcc
      781 ttatatatcc taatataaaa atatctaata aattagatga atatcaaaga atctattgat
      841 ttagtgtatt attaaatgta tatcttaatt caatattatt attctattca tttttattat
      901 tcatttttat tcattttcaa atttagaata tattaatcta tatattaatt tagaattcta
      961 ttctaattcg aattcaattt ttaaatattc atattcaatt aaaattgaaa ttttttcatt
     1021 cgcgaggagc cggatgagaa gaaactctca cgtccggttc tgtagtagag gtggaattaa
     1081 gaaaaaacca tcaactataa ccccaaaaga accagattct gtaaacaaca tagaggaaga
     1141 atgaagggaa tatcttatcg gggggaatcgt atttgtttcg gaagatatgc tctcaggcac
     1201 ga
//

LOCUS       HM041467                1199 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea karwinskiana ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041467
VERSION     HM041467.1  GI:330887226
KEYWORDS    .
SOURCE      chloroplast Opuntia karwinskiana
  ORGANISM  Opuntia karwinskiana
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1199)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1199)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1199
                     /organism="Opuntia karwinskiana"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001112"
     gene            <1..>1199
                     /gene="rpl16"
     misc_feature    <1098..>1199
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gtgatatcga aaaagtagag cccatagtat gaagtatgaa ctaataacta tagaactaat
       61 aaccaactca tcgcatcaca ttatctggat ccaaagaagc agtcaagata ggatattttg
      121 gtcctatcat tgcagcaact gaattttttt ttcataaaca agaaatcaaa tgagttgtca
      181 agcaaaagaa aaaaaaaaaa gaaaaattta ctttaaggga gggggatgcg gataaatgga
      241 aaggcgaaag aaagaaaaaa atgaatctaa atgatatacg attcccctat gtagggtctt
      301 tgaatcatat cataaaaaac aatgtaataa agcatgaata cagattcccc cataattatc
      361 tggtatgaat cttttcatag aaaaaaaaaa aaagtaagag cctccggcca ataaaaacta
      421 aaagggttgg ctcaagaaca aagttcatta agggctccat tgtagaattc agacctaatc
      481 nttaatcaag aagcgatggg aacgatgtaa tccatgaata cagaagattc aattgaaaaa
      541 gaatcctaat gattcattgg gaaggatggc ggaacgaacc agagaccaat tcatctattc
      601 tgaaaagtga taaactaatc ctataaaact aaaatagata ttgaaagagt aaatattcgc
      661 ccgcgaaaat tccttttta ttaaattgct cacattttat tttagcaatg caatctaata
      721 aaatatatct atacaaaaaa atatagacaa actatatata taatatattt caaatttcct
      781 tatatatcct aatataaaaa tatctaataa attagatgaa tatcaaagaa tctattgatt
      841 tagtgtatta ttaaatgtat atcttaattc aatattatta ttctattcat ttttattatt
      901 catttttatt cattttcaaa tttagaatat attaatctat atattaattt agaattctat
      961 tctaattcga attcaatttt taaatattca tattcaatta aaattgaaat tttttcattc
     1021 gcgaggagcc ggatgagaag aaactctcac gtccggttct gtagtagagg tggaattaag
     1081 aaaaaaccat caactataac cccaaaagaa ccagattctg taaacaacat agaggaagaa
     1141 tgaagggaat atcttatgcg ggggaatcgt attgtttcgg aagatatgct ctgcggccc
//

LOCUS       HM041466                1205 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea gaumeri ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041466
VERSION     HM041466.1  GI:330887225
KEYWORDS    .
SOURCE      chloroplast Nopalea gaumeri
  ORGANISM  Nopalea gaumeri
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1205)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1205)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
```

```
     source          1..1205
                     /organism="Nopalea gaumeri"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001111"
     gene            <1..>1205
                     /gene="rpl16"
     misc_feature    <1103..>1205
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gctgtgatat acgaaanagt aagagcccat agtatgaagt atgaactaat aactatagaa
       61 ctaataacca actcatcgca tcacattatc tggatccaaa gaagcagtca agataggata
      121 ttttggtcct atcattgcag caactgaatt ttttttcat aaacaagaaa tcaaatgagt
      181 tgtcaagcaa aagaaaaaaa aaaaaaaaaa tatacattaa aggaggggga tgcggataaa
      241 tggaaaggcg aaagaaagaa aaaaatgaat ctaaatgata tacgattcca ctatgtaagg
      301 tctttgaatc atatcataaa agacaatgta ataaagcatg aatacagatt cacacataat
      361 tatctgaata tgaatctatt catagaaaaa agaaaaaagt aagaccctcc ggccaataaa
      421 gactaaaggg gttggctcaa gaacaaagtt cattaagggc tccattgtag aattcagacc
      481 taatcattaa tcaagaagcg atgggaacga tgtaatccat gaatacagaa gattcaattg
      541 aaaaagaatc ctaatgattc attgggaagg atggcggaac gaaccagaga ccaattcatc
      601 tattctgaaa agtgataaac taatcctata aaactaaaat agatattgaa agagtaaata
      661 ttcgcccgcg aaaattcctt ttttattaaa ttgctcacat tttattttag caatgcaatc
      721 taataaaata tatctataca aaaaaatata gacaaactat atatataata tatttcaaat
      781 ttccttatat atcctaatat aaaaatatct aataaattag atgaatatca aagaatctat
      841 tgatttagtg tattattaaa tgtatatctt aattcaatat tattattcta ttcattttta
      901 ttattcattt ttattcattt tcaaatttag aatatattaa tctatatatt aatttagaat
      961 tctattctaa ttcgaattca atttttaaat attcatattc aattaaaatt gaaatttttt
     1021 cattcgcgag gagccggatg agaagaaact ctcacgtccg gttctgtagt agaggtggaa
     1081 ttaagaaaaa accatcaact ataaccccaa aagaaccaga ttctgtaaac aacatagagg
     1141 aagaatgaag ggaatatctt atcgggggaa tcgtatttgt ttcggaagat atgctctcag
     1201 cacga
//

LOCUS       HM041465                1190 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea dejecta ribosomal protein L16-like (rpl16) gene, partial
            sequence; chloroplast.
ACCESSION   HM041465
VERSION     HM041465.1  GI:330887224
KEYWORDS    .
SOURCE      chloroplast Opuntia dejecta
  ORGANISM  Opuntia dejecta
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1190)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1190)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1190
                     /organism="Opuntia dejecta"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:1001110"
     gene            <1..>1190
                     /gene="rpl16"
     misc_feature    <1096..>1190
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 tgatatacga aanagtaaga gcccatagta tgaagtatga actaataact atagaactaa
       61 taaccaactc atcgcatcac attatctgga tccaaagaag cagtcaagat aggatatttt
      121 ggtcctatca ttgcagcaac tgaatttttt tttcataaac aagaaatcaa atgagttgtc
      181 aagcaaaaga aaaaaaaaaa aaaaaatata ctttaangga gggggatgcg gataaatgga
      241 aaggcgaaag aaagaaaaaa atgaatctaa atgatatacg attccactat gtaaggtctt
      301 tgaatcatat cataaaagac aatgtaataa agcatgaata cagattcaca cataattatc
      361 tgtatgatct attcatagaa aaaagaaaaa agtaagagcc tccggccaat aaagactaag
      421 agggttggct caagaacaaa gttcattaag ggctccattg tagaattcag acctaatcat
      481 taatcaagaa gcgatgggaa cgatgtaatc catgaataca gaagattcaa ttgaaaaaga
      541 atcctaatga ttcattggga aggatggcgg aacgaaccag agaccaattc atctattctg
      601 aaaagtgata aactaatcct ataaaactaa aatagatatt gaaagagtaa atattcgccc
      661 gcgaaaattc ctttttttatt aaattgctca cattttattt tagcaatgca atctaataaa
      721 atatatctat acaaaaaaat atagacaaac tatatatata atatatttca aatttcctta
      781 tatatcctaa tataaaaata tctaataaat tagatgaata tcaaagaatc tattgattta
      841 gtgtattatt aaatgtatat cttaattcaa tattattatt ctattcattt ttattattca
      901 tttttattca ttttcaaatt tagaatatat taatctatat attaatttag aattctattc
      961 taattcgaat tcaattttta aatattcata ttcaattaaa attgaaattt tttcattcgc
     1021 gaggagccgg atgagaagaa actctcacgt ccggttctgt agtagaggtg gaattaagaa
     1081 aaaaccatca actataaccc caaaagaacc agattctgta aacaacatag aggaagaatg
     1141 aagggaatat cttatcgggg gaatcgtatt tgtttcggaa gatatgctct
//

LOCUS       HM041464                1184 bp    DNA     linear   PLN 03-MAY-2011
DEFINITION  Nopalea cochenillifera ribosomal protein L16-like (rpl16) gene,
            partial sequence; chloroplast.
ACCESSION   HM041464
VERSION     HM041464.1  GI:330887223
KEYWORDS    .
SOURCE      chloroplast Opuntia cochenillifera
  ORGANISM  Opuntia cochenillifera
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 1184)
  AUTHORS   Hernandez-Hernandez,T., Hernandez,H.M., De-Nova,J.A., Puente,R.,
            Eguiarte,L.E. and Magallon,S.
  TITLE     Phylogenetic relationships and evolution of growth form in
            Cactaceae (Caryophyllales, Eudicotyledoneae)
  JOURNAL   Am. J. Bot. 98 (1), 44-61 (2011)
   PUBMED   21613084
REFERENCE   2  (bases 1 to 1184)
  AUTHORS   Hernandez-Hernandez,T., Magallon,S.A., Hernandez,H.M., De-Nova,A.,
            Puente,R. and Eguiarte,L.E.
  TITLE     Direct Submission
  JOURNAL   Submitted (17-MAR-2010) Departamento de Botanica, Instituto de
            Biologia, Universidad Nacional Autonoma de Mexico, 3er Circuito de
            Ciudad Universitaria, Ciudad Universitaria, Coyoacan, Distrito
            Federal C.P. 04510, Mexico
FEATURES             Location/Qualifiers
     source          1..1184
                     /organism="Opuntia cochenillifera"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:338184"
     gene            <1..>1184
                     /gene="rpl16"
     misc_feature    <1102..>1184
                     /gene="rpl16"
                     /note="similar to ribosomal protein L16"
ORIGIN
        1 gctgtgatat acgaaaaagt aagagcccat agtatgaagt atgaactaac aactatagaa
       61 ctaataacca actcatcgca tcacattatc tggatccaaa gaagcagtca agataggata
      121 ttttggtcct atcattgcag caactgaatt tttttttcat aaacaagaaa tcaaatgagt
      181 tgtcaagcaa aagaaaaaaa aaaaaaaaaa tatactttaa aggaggggga tgcggataaa
      241 tggaaaggcg aaagaaagaa aaaaatgaat ctaaatgata tacgattcca ctatgtaagg
      301 tctttgaatc atatcataaa agacaatgta ataaagcatg aatacagatt cccacataat
      361 tatctgatat gaatctattc atagaaaaaa gaaaaaagta agagcctccg gccaataaag
      421 actaagaggg ttggctcaag aacaaagttc attaagggct ccattgtaga attcagacct
      481 aatcattaat caagaagcga tgggaacgat gtaatccatg aatacagaag attcaattga
      541 aaaagaatcc taatgattca ttgggaagga tggcggaacg aaccagagac caattcatct
      601 attctgaaaa gtgataaact aatcctataa aactaaaata gatattgaaa gagtaaatat
      661 tcgcccgcga aaattccttt tttattaaat tgctcacatt ttattttagc aatgcaatct
      721 aataaaatat atctatacaa aaaaatatag acaaactata tatataatat atttcaaatt
      781 tccttatata tcctaatata aaaatatcta ataaattaga tgaatatcaa agaatctatt
      841 gatttagtgt attattaaat gtatatctta attcaatatt attattctat tcatttttat
      901 tattcatttt tattcatttt caaatttaga atatattaat ctatatatta atttagaatt
      961 ctattctaat tcgaattcaa tttttaaata ttcatattca attaaaattg aaattttttc
     1021 attcgcgagg agccggatga gaagaaactc tcacgtccgg ttctgtagta gaggtggaat
     1081 taagaaaaaa ccatcaacta taaccccaaa agaaccagat tctgtaaaca acatagagga
     1141 agaatgaagg gaatatctta tcgggggaat cgtatttgtt tcgg
//

LOCUS       AY851612                 892 bp    DNA     linear   PLN 10-APR-2007
DEFINITION  Opuntia subulata rpl16 gene, intron; chloroplast.
ACCESSION   AY851612
VERSION     AY851612.1  GI:57240072
KEYWORDS    .
SOURCE      chloroplast Austrocylindropuntia subulata
  ORGANISM  Austrocylindropuntia subulata
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Austrocylindropuntia.
REFERENCE   1  (bases 1 to 892)
  AUTHORS   Butterworth,C.A. and Wallace,R.S.
  TITLE     Molecular Phylogenetics of the Leafy Cactus Genus Pereskia
            (Cactaceae)
  JOURNAL   Syst. Bot. 30 (4), 800-808 (2005)
REFERENCE   2  (bases 1 to 892)
  AUTHORS   Butterworth,C.A. and Wallace,R.S.
  TITLE     Direct Submission
  JOURNAL   Submitted (10-DEC-2004) Desert Botanical Garden, 1201 North Galvin
            Parkway, Phoenix, AZ 85008, USA
```

```
FEATURES             Location/Qualifiers
     source          1..892
                     /organism="Austrocylindropuntia subulata"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:106982"
     gene            <1..>892
                     /gene="rpl16"
     intron          <1..>892
                     /gene="rpl16"
ORIGIN
        1 cattaaagaa gggggatgcg gataaatgga aaggcgaaag aaagaaaaaa atgaatctaa
       61 atgatatacg attccactat gtaaggtctt tgaatcatat cataaaagac aatgtaataa
      121 agcatgaata cagattcaca cataattatc tgatatgaat ctattcatag aaaaaagaaa
      181 aaagtaagag cctccggcca ataaagacta agagggttgg ctcaagaaca aagttcatta
      241 agagctccat tgtagaattc agacctaatc attaatcaag aagcgatggg aacgatgtaa
      301 tccatgaata cagaagattc aattgaaaaa gatcctaatg atcattggga aggatggcgg
      361 aacgaaccag agaccaattc atctattctg aaaagtgata aactaatcct ataaaactaa
      421 aatagatatt gaaagagtaa atattcgccc gcgaaaattc cttttttatt aaattgctca
      481 tattttattt tagcaatgca atctaataaa atatatctat acaaaaaaat atagacaaac
      541 tatatatata taatatattt caaatttcct tatataccca aatataaaaa tatctaataa
      601 attagatgaa tatcaaagaa tctattgatt tagtgtatta ttaaatgtat atcttaattc
      661 aatattatta ttctattcat ttttattcat tttcaaattt ataatatatt aatctatata
      721 ttaatttata attctattct aattcgaatt caatttttaa atattcatat tcaattaaaa
      781 ttgaaatttt ttcattcgcg aggagccgga tgagaagaaa ctctcatgtc cggttctgta
      841 gtagagatgg aattaagaaa aaaccatcaa ctataacccc aagagaacca ga
//

LOCUS       AY851611                 881 bp    DNA     linear   PLN 10-APR-2007
DEFINITION  Opuntia polyacantha rpl16 gene, intron; chloroplast.
ACCESSION   AY851611
VERSION     AY851611.1  GI:57240071
KEYWORDS    .
SOURCE      chloroplast Opuntia polyacantha
  ORGANISM  Opuntia polyacantha
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Opuntia.
REFERENCE   1  (bases 1 to 881)
  AUTHORS   Butterworth,C.A. and Wallace,R.S.
  TITLE     Molecular Phylogenetics of the Leafy Cactus Genus Pereskia
            (Cactaceae)
  JOURNAL   Syst. Bot. 30 (4), 800-808 (2005)
REFERENCE   2  (bases 1 to 881)
  AUTHORS   Butterworth,C.A. and Wallace,R.S.
  TITLE     Direct Submission
  JOURNAL   Submitted (10-DEC-2004) Desert Botanical Garden, 1201 North Galvin
            Parkway, Phoenix, AZ 85008, USA
FEATURES             Location/Qualifiers
     source          1..881
                     /organism="Opuntia polyacantha"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:307728"
     gene            <1..>881
                     /gene="rpl16"
     intron          <1..>881
                     /gene="rpl16"
```

```
ORIGIN
        1 cattaaagga gggggatgcg gataaatgga aaggcgaaag aaagaaaaaa atgaatctaa
       61 atgatatacg attccactat gtaaggtctt tgaatcatat cataaaagac aatgtaataa
      121 agcatgaata cagattcaca cataattatc tgatatgaat ctattcatag aaaaaagaaa
      181 aaagtaagag cctccggcca ataaagacta agagggttgg ctcaagaaca aagttcatta
      241 agggctccat tgtagaattc agacctaatc attaatcaag aagcgatggg aacgatgtaa
      301 tccatgaata cagaagattc aattgaaaaa gatcctaatg atcattggga aggatggcgg
      361 aacgaaccag agaccaattc atctattctg aaaagtgata aactaatcct ataaaactaa
      421 aatagatatt gaaagagtaa atattcgccc gcgaaaattc cttttttatt aaattgctca
      481 cattttattt tagcaatgca atctaataaa atatatctat acaaaaaaat atagacaaac
      541 tctatatata atatatttca aatttcctta tatatcctaa tataaaaata tctaataaat
      601 tagatgaata tcaaagaatc tattgattta gtgtattatt aaatgtatat cttaattcaa
      661 tattattatt ctattcattt tcaaatttag aatatattaa tctatatatt aatttagaat
      721 tctattctaa ttcgaattca atttttaaat attcatattc aattaaaatt gaaatttttt
      781 cattcgcgag gagccggatg agaagaaact ctcacgtccg gttactgtag tagaggtgga
      841 attaagaaaa aaccatcaac tataacccca aaagaaccag a
//

LOCUS       AF191661                 895 bp    DNA     linear   PLN 07-NOV-1999
DEFINITION  Opuntia kuehnrichiana rpl16 gene; chloroplast gene for chloroplast
            product, partial intron sequence.
ACCESSION   AF191661
VERSION     AF191661.1  GI:6273287
KEYWORDS    .
SOURCE      chloroplast Cumulopuntia sphaerica
  ORGANISM  Cumulopuntia sphaerica
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; Gunneridae;
            Pentapetalae; Caryophyllales; Cactineae; Cactaceae; Opuntioideae;
            Cumulopuntia.
REFERENCE   1  (bases 1 to 895)
  AUTHORS   Dickie,S.L. and Wallace,R.S.
  TITLE     Phylogeny of the subfamily Opuntioideae (Cactaceae)
  JOURNAL   Unpublished
REFERENCE   2  (bases 1 to 895)
  AUTHORS   Dickie,S.L. and Wallace,R.S.
  TITLE     Direct Submission
  JOURNAL   Submitted (28-SEP-1999) Botany, Iowa State University, 353 Bessey
            Hall, Ames, IA 50011-1020, USA
FEATURES             Location/Qualifiers
     source          1..895
                     /organism="Cumulopuntia sphaerica"
                     /organelle="plastid:chloroplast"
                     /mol_type="genomic DNA"
                     /db_xref="taxon:106979"
                     /note="subfamily Opuntioideae; synonym: Cumulopuntia
                     kuenrichiana"
     gene            <1..>895
                     /gene="rpl16"
     intron          <1..>895
                     /gene="rpl16"
ORIGIN
        1 tatacattaa agaagggggа tgcggataaa tggaaaggcg aaagaaagaa aaaaatgaat
       61 ctaaatgata tacgattcca ctatgtaagg tctttgaatc atatcataaa agacaatgta
      121 ataaagcatg aatacagatt cacacataat tatctgatat gaatctattc atagaaaaaa
      181 gaaaaaagta agagcctccg gccaataaag actaagaggg ttggctcaag aacaaagttc
      241 attaagagct ccattgtaga attcagacct aatcattaat caagaagcga tgggaacgat
      301 gtaatccatg aatacagaag attcaattga aaaagatcct atgatccatt gggaaggatg
      361 gcggaacgaa ccagagacca attcatctat tctgaaaagt gataaactaa tcctataaaa
      421 ctaaaataga tattgaaaga gtaaatattc gcccgcgaaa attccttttt tttttaaatt
      481 gctcatattt tattttagca atgcaatcta ataaaatata tctatacaaa aaaataaaga
      541 caaactatat atataatata tttcaaattt ccttatatat ccaaatataa aaatatctaa
      601 taaattagat gaatatcaaa gaatctattg atttagtgta ttattaaatg tatatcttaa
      661 ttcaatatta ttattctatt catttttatt cattttcaat tttataatat attaatctat
      721 atattaattt ataattctat tctaattcga attcaatttt taaatattca tattcaatta
      781 aaattgaaat tttttcattc gcgaggagcc ggatgagaag aaactctcat gtccggttct
      841 gtagtagaga tggaattaag aaaaaaccat caactataac cccaagagaa ccaga
//
```

In this case, we are just getting the raw records. To get the records in a more Python-friendly form, we can use `Bio.SeqIO` to parse the GenBank data into `SeqRecord` objects, including `SeqFeature` objects (see Chapter [chapter:Bio.SeqIO]):

```
In [79]: from Bio import SeqIO
         handle = Entrez.efetch(db="nuccore", id=gi_str, rettype="gb", retmode="text")
         records = SeqIO.parse(handle, "gb")
```

We can now step through the records and look at the information we are interested in:

```
In [80]: for record in records:
             print("%s, length %i, with %i features" \
             % (record.name, len(record), len(record.features)))
```

```
HQ621368, length 399, with 3 features
HM041482, length 1197, with 3 features
HM041481, length 1200, with 3 features
HM041480, length 1153, with 3 features
HM041479, length 1197, with 3 features
HM041478, length 1187, with 3 features
HM041477, length 1197, with 3 features
HM041476, length 1205, with 3 features
HM041474, length 1163, with 3 features
HM041473, length 1203, with 3 features
HM041472, length 1182, with 3 features
HM041469, length 1189, with 3 features
HM041468, length 1202, with 3 features
HM041467, length 1199, with 3 features
HM041466, length 1205, with 3 features
HM041465, length 1190, with 3 features
HM041464, length 1184, with 3 features
AY851612, length 892, with 3 features
AY851611, length 881, with 3 features
AF191661, length 895, with 3 features
```

Using these automated query retrieval functionality is a big plus over doing things by hand. Although the module should obey the NCBI's max three queries per second rule, the NCBI have other recommendations like avoiding peak hours. See Section [sec:entrez-guidelines]. In particular, please note that for simplicity, this example does not use the WebEnv history feature. You should use this for any non-trivial search and download work, see Section *History and WebEnv*.

Finally, if plan to repeat your analysis, rather than downloading the files from the NCBI and parsing them immediately (as shown in this example), you should just download the records *once* and save them to your hard disk, and then parse the local file.

10.14.4 Finding the lineage of an organism

Staying with a plant example, let's now find the lineage of the Cypripedioideae orchid family. First, we search the Taxonomy database for Cypripedioideae, which yields exactly one NCBI taxonomy identifier:

```
In [81]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"     # Always tell NCBI who you are
         handle = Entrez.esearch(db="Taxonomy", term="Cypripedioideae")
         record = Entrez.read(handle)
         record["IdList"]
```

```
Out[81]: ['158330']
```

```
In [82]: record["IdList"][0]
```

```
Out[82]: '158330'
```

Now, we use `efetch` to download this entry in the Taxonomy database, and then parse it:

```
In [83]: handle = Entrez.efetch(db="Taxonomy", id="158330", retmode="xml")
         records = Entrez.read(handle)
```

Again, this record stores lots of information:

```
In [84]: records[0].keys()
```

```
Out[84]: dict_keys(['PubDate', 'ScientificName', 'Division', 'MitoGeneticCode', 'GeneticCode', 'Creat
```

We can get the lineage directly from this record:

```
In [85]: records[0]["Lineage"]
```

```
Out[85]: 'cellular organisms; Eukaryota; Viridiplantae; Streptophyta; Streptophytina; Embryophyta; Tr
```

The record data contains much more than just the information shown here - for example look under `LineageEx` instead of `Lineage` and you'll get the NCBI taxon identifiers of the lineage entries too.

10.15 Using the history and WebEnv

Often you will want to make a series of linked queries. Most typically, running a search, perhaps refining the search, and then retrieving detailed search results. You *can* do this by making a series of separate calls to Entrez. However, the NCBI prefer you to take advantage of their history support - for example combining ESearch and EFetch.

Another typical use of the history support would be to combine EPost and EFetch. You use EPost to upload a list of identifiers, which starts a new history session. You then download the records with EFetch by referring to the session (instead of the identifiers).

10.15.1 Searching for and downloading sequences using the history

Suppose we want to search and download all the *Opuntia* rpl16 nucleotide sequences, and store them in a FASTA file. As shown in Section [sec:entrez-search-fetch-genbank], we can naively combine `Bio.Entrez.esearch()` to get a list of GI numbers, and then call `Bio.Entrez.efetch()` to download them all.

However, the approved approach is to run the search with the history feature. Then, we can fetch the results by reference to the search results - which the NCBI can anticipate and cache.

To do this, call `Bio.Entrez.esearch()` as normal, but with the additional argument of `usehistory="y"`,

```
In [86]: from Bio import Entrez
         Entrez.email = "history.user@example.com"
         search_handle = Entrez.esearch(db="nucleotide",term="Opuntia[orgn] and rpl16", usehistory="y
         search_results = Entrez.read(search_handle)
         search_handle.close()
```

When you get the XML output back, it will still include the usual search results. However, you also get given two additional pieces of information, the `WebEnv` session cookie, and the `QueryKey`:

```
In [87]: gi_list = search_results["IdList"]
         count = int(search_results["Count"])
         assert count == len(gi_list)
         print("The WebEnv is {}".format(search_results["WebEnv"]))
         print("The QueryKey is {}".format(search_results["QueryKey"]))
```

```
The WebEnv is NCID_1_946410500_130.14.18.34_9001_1452651901_1799213676_0MetA0_S_MegaStore_F_1
The QueryKey is 1
```

Having stored these values in variables session_cookie and query_key we can use them as parameters to `Bio.Entrez.efetch()` instead of giving the GI numbers as identifiers.

While for small searches you might be OK downloading everything at once, it is better to download in batches. You use the retstart and retmax parameters to specify which range of search results you want returned (starting entry using zero-based counting, and maximum number of results to return). Sometimes you will get intermittent errors from Entrez, HTTPError 5XX, we use a try except pause retry block to address this. For example,

```
from Bio import Entrez
import time
try:
    from urllib.error import HTTPError  # for Python 3
except ImportError:
    from urllib2 import HTTPError  # for Python 2
batch_size = 3
out_handle = open("orchid_rpl16.fasta", "w")
for start in range(0, count, batch_size):
    end = min(count, start+batch_size)
    print("Going to download record %i to %i" % (start+1, end))
    attempt = 1
    while attempt <= 3:
        try:
            fetch_handle = Entrez.efetch(db="nucleotide", rettype="fasta", retmode=
↪"text",
                                         retstart=start, retmax=batch_size,
                                         webenv=webenv, query_key=query_key)
        except HTTPError as err:
            if 500 <= err.code <= 599:
                print("Received error from server %s" % err)
                print("Attempt %i of 3" % attempt)
                attempt += 1
                time.sleep(15)
            else:
                raise
    data = fetch_handle.read()
    fetch_handle.close()
    out_handle.write(data)
out_handle.close()
```

For illustrative purposes, this example downloaded the FASTA records in batches of three. Unless you are downloading genomes or chromosomes, you would normally pick a larger batch size.

10.15.2 Searching for and downloading abstracts using the history

Here is another history example, searching for papers published in the last year about the *Opuntia*, and then downloading them into a file in MedLine format:

```
from Bio import Entrez
import time
try:
    from urllib.error import HTTPError  # for Python 3
except ImportError:
    from urllib2 import HTTPError  # for Python 2
Entrez.email = "history.user@example.com"
search_results = Entrez.read(Entrez.esearch(db="pubmed",
                                            term="Opuntia[ORGN]",
                                            reldate=365, datetype="pdat",
                                            usehistory="y"))
count = int(search_results["Count"])
print("Found %i results" % count)

batch_size = 10
out_handle = open("recent_orchid_papers.txt", "w")
for start in range(0,count,batch_size):
    end = min(count, start+batch_size)
    print("Going to download record %i to %i" % (start+1, end))
    attempt = 1
    while attempt <= 3:
        try:
            fetch_handle = Entrez.efetch(db="pubmed",rettype="medline",
                                         retmode="text",retstart=start,
                                         retmax=batch_size,
                                         webenv=search_results["WebEnv"],
                                         query_key=search_results["QueryKey"])
        except HTTPError as err:
            if 500 <= err.code <= 599:
                print("Received error from server %s" % err)
                print("Attempt %i of 3" % attempt)
                attempt += 1
                time.sleep(15)
            else:
                raise
    data = fetch_handle.read()
    fetch_handle.close()
    out_handle.write(data)
out_handle.close()
```

At the time of writing, this gave 28 matches - but because this is a date dependent search, this will of course vary. As described in Section [subsec:entrez-and-medline] above, you can then use `Bio.Medline` to parse the saved records.

10.15.3 Searching for citations

Back in Section [sec:elink] we mentioned ELink can be used to search for citations of a given paper. Unfortunately this only covers journals indexed for PubMed Central (doing it for all the journals in PubMed would mean a lot more work for the NIH). Let's try this for the Biopython PDB parser paper, PubMed ID 14630660:

```
In [88]: from Bio import Entrez
         Entrez.email = "A.N.Other@example.com"
         pmid = "14630660"
         results = Entrez.read(Entrez.elink(dbfrom="pubmed", db="pmc",
         LinkName="pubmed_pmc_refs", from_uid=pmid))
         pmc_ids = [link["Id"] for link in results[0]["LinkSetDb"][0]["Link"]]
         pmc_ids
```

```
Out[88]: ['4595337',
          '4584387',
          '4520291',
          '4301084',
          '4173008',
          '4155247',
          '3879085',
          '3661355',
          '3531324',
          '3461403',
          '3394275',
          '3394243',
          '3312550',
          '3253748',
          '3161047',
          '3144279',
          '3122320',
          '3102222',
          '3096039',
          '3057020',
          '3036045',
          '2944279',
          '2902450',
          '2744707',
          '2705363',
          '2682512',
          '2483496',
          '2447749',
          '2098836',
          '2072961',
          '1933154',
          '1848001',
          '1192790',
          '1190160']
```

Great - eleven articles. But why hasn't the Biopython application note been found (PubMed ID 19304878)? Well, as you might have guessed from the variable names, there are not actually PubMed IDs, but PubMed Central IDs. Our application note is the third citing paper in that list, PMCID 2682512.

So, what if (like me) you'd rather get back a list of PubMed IDs? Well we can call ELink again to translate them. This becomes a two step process, so by now you should expect to use the history feature to accomplish it (Section *History and WebEnv*).

But first, taking the more straightforward approach of making a second (separate) call to ELink:

```
In [89]: results2 = Entrez.read(Entrez.elink(dbfrom="pmc", db="pubmed", LinkName="pmc_pubmed",
         from_uid=",".join(pmc_ids)))
         pubmed_ids = [link["Id"] for link in results2[0]["LinkSetDb"][0]["Link"]]
         pubmed_ids
```

```
Out[89]: ['26439842',
          '26306092',
          '26288158',
          '25420233',
          '24930144',
          '24849577',
          '24267035',
          '23717802',
          '23300419',
          '22784991',
          '22564897',
          '22553365',
          '22479120',
          '22361291',
          '21801404',
          '21637529',
          '21521828',
          '21500218',
          '21471012',
          '21467571',
          '20813068',
          '20607693',
          '20525384',
          '19698094',
          '19450287',
          '19304878',
          '18645234',
          '18502776',
          '18052533',
          '17888163',
          '17567620',
          '17397254',
          '16095538',
          '15985178']
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 11

Swiss-Prot and ExPASy

11.1 Parsing Swiss-Prot files

Swiss-Prot (http://www.expasy.org/sprot) is a hand-curated database of protein sequences. Biopython can parse the “plain text” Swiss-Prot file format, which is still used for the UniProt Knowledgebase which combined Swiss-Prot, TrEMBL and PIR-PSD. We do not (yet) support the UniProtKB XML file format.

11.1.1 Parsing Swiss-Prot records

In Section [sec:SeqIO_ExPASy_and_SwissProt], we described how to extract the sequence of a Swiss-Prot record as a `SeqRecord` object. Alternatively, you can store the Swiss-Prot record in a `Bio.SwissProt.Record` object, which in fact stores the complete information contained in the Swiss-Prot record. In this section, we describe how to extract `Bio.SwissProt.Record` objects from a Swiss-Prot file.

To parse a Swiss-Prot record, we first get a handle to a Swiss-Prot record. There are several ways to do so, depending on where and how the Swiss-Prot record is stored:

- Open a Swiss-Prot file locally: `>>> handle = open("myswissprotfile.dat")`
- Open a gzipped Swiss-Prot file:

```
In [2]: import gzip
        handle = gzip.open("myswissprotfile.dat.gz")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-5b562ec6e633> in <module>()
      1 import gzip
----> 2 handle = gzip.open("myswissprotfile.dat.gz")

/home/tiago_antao/miniconda/lib/python3.5/gzip.py in open(filename, mode, compresslevel, encoding, e
     51     gz_mode = mode.replace("t", "")
     52     if isinstance(filename, (str, bytes)):
---> 53         binary_file = GzipFile(filename, gz_mode, compresslevel)
```

```
     54     elif hasattr(filename, "read") or hasattr(filename, "write"):
     55         binary_file = GzipFile(None, gz_mode, compresslevel, filename)

/home/tiago_antao/miniconda/lib/python3.5/gzip.py in __init__(self, filename, mode, compresslevel, fi
    161             mode += 'b'
    162         if fileobj is None:
--> 163             fileobj = self.myfileobj = builtins.open(filename, mode or 'rb')
    164         if filename is None:
    165             filename = getattr(fileobj, 'name', '')

FileNotFoundError: [Errno 2] No such file or directory: 'myswissprotfile.dat.gz'
```

- Open a Swiss-Prot file over the internet:

```
In [5]: import urllib.request
        handle = urllib.request.urlopen("http://www.somelocation.org/data/someswissprotfile.dat")


---------------------------------------------------------------------------
gaierror                                  Traceback (most recent call last)
/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in do_open(self, http_class, req, **http_
   1239             try:
-> 1240                 h.request(req.get_method(), req.selector, req.data, headers)
   1241             except OSError as err: # timeout error

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in request(self, method, url, body, headers)
   1082         """Send a complete request to the server."""
-> 1083         self._send_request(method, url, body, headers)
   1084

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _send_request(self, method, url, body, he
   1127             body = body.encode('iso-8859-1')
-> 1128         self.endheaders(body)
   1129

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in endheaders(self, message_body)
   1078             raise CannotSendHeader()
-> 1079         self._send_output(message_body)
   1080

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _send_output(self, message_body)
    910
--> 911         self.send(msg)
    912         if message_body is not None:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in send(self, data)
    853             if self.auto_open:
--> 854                 self.connect()
    855             else:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in connect(self)
    825         self.sock = self._create_connection(
--> 826             (self.host,self.port), self.timeout, self.source_address)    827         self.soc

/home/tiago_antao/miniconda/lib/python3.5/socket.py in create_connection(address, timeout, source_adc
    692     err = None
--> 693     for res in getaddrinfo(host, port, 0, SOCK_STREAM):
    694         af, socktype, proto, canonname, sa = res

/home/tiago_antao/miniconda/lib/python3.5/socket.py in getaddrinfo(host, port, family, type, proto, f
    731     addrlist = []
--> 732     for res in _socket.getaddrinfo(host, port, family, type, proto, flags):
    733         af, socktype, proto, canonname, sa = res

gaierror: [Errno -2] Name or service not known

During handling of the above exception, another exception occurred:

URLError                                  Traceback (most recent call last)
<ipython-input-5-19b02eddd780> in <module>()
      1 import urllib.request
----> 2 handle = urllib.request.urlopen("http://www.somelocation.org/data/someswissprotfile.dat")

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in urlopen(url, data, timeout, cafile, ca
    160     else:
    161         opener = _opener
--> 162     return opener.open(url, data, timeout)
    163 
    164 def install_opener(opener):

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    463             req = meth(req)
    464 
--> 465         response = self._open(req, data)
    466 
    467         # post-process response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _open(self, req, data)
    481         protocol = req.type
    482         result = self._call_chain(self.handle_open, protocol, protocol +
--> 483                                   '_open', req)    484         if result:
    485             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_open(self, req)
   1266 
   1267     def http_open(self, req):
-> 1268         return self.do_open(http.client.HTTPConnection, req)
   1269 
   1270     http_request = AbstractHTTPHandler.do_request_

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in do_open(self, http_class, req, **http_
   1240                 h.request(req.get_method(), req.selector, req.data, headers)
   1241             except OSError as err: # timeout error
-> 1242                 raise URLError(err)
   1243             r = h.getresponse()
   1244         except:

URLError: <urlopen error [Errno -2] Name or service not known>
```

- Open a Swiss-Prot file over the internet from the ExPASy database (see section [subsec:expasy_swissprot]):

```
In [6]: from Bio import ExPASy
        handle = ExPASy.get_sprot_raw(myaccessionnumber)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-c988fb329f11> in <module>()
      1 from Bio import ExPASy
----> 2 handle = ExPASy.get_sprot_raw(myaccessionnumber)

NameError: name 'myaccessionnumber' is not defined
```

The key point is that for the parser, it doesn't matter how the handle was created, as long as it points to data in the Swiss-Prot format.

We can use `Bio.SeqIO` as described in Section [sec:SeqIO_ExPASy_and_SwissProt] to get file format agnostic `SeqRecord` objects. Alternatively, we can use `Bio.SwissProt` get `Bio.SwissProt.Record` objects, which are a much closer match to the underlying file format.

To read one Swiss-Prot record from the handle, we use the function `read()`:

```
In [7]: from Bio import SwissProt
        record = SwissProt.read(handle)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-4150bec69677> in <module>()
      1 from Bio import SwissProt
----> 2 record = SwissProt.read(handle)

NameError: name 'handle' is not defined
```

This function should be used if the handle points to exactly one Swiss-Prot record. It raises a `ValueError` if no Swiss-Prot record was found, and also if more than one record was found.

We can now print out some information about this record:

```
In [8]: print(record.description)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-040616e07cd0> in <module>()
----> 1 print(record.description)

NameError: name 'record' is not defined
```

```
In [9]: for ref in record.references:
            print("authors:", ref.authors)
            print("title:", ref.title)
```

```
  File "<ipython-input-9-a58c3c1279e1>", line 2
    print("authors:", ref.authors)
        ^
IndentationError: expected an indented block
```

```
In [10]: print(record.organism_classification)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-e3473e90e26a> in <module>()
----> 1 print(record.organism_classification)

NameError: name 'record' is not defined
```

To parse a file that contains more than one Swiss-Prot record, we use the `parse` function instead. This function allows us to iterate over the records in the file.

For example, let's parse the full Swiss-Prot database and collect all the descriptions. You can download this from the ExPAYs FTP site as a single gzipped-file `uniprot_sprot.dat.gz` (about 300MB). This is a compressed file containing a single file, `uniprot_sprot.dat` (over 1.5GB).

As described at the start of this section, you can use the Python library `gzip` to open and uncompress a `.gz` file, like this:

```
In [12]: import gzip
         handle = gzip.open("data/uniprot_sprot.dat.gz")
```

However, uncompressing a large file takes time, and each time you open the file for reading in this way, it has to be decompressed on the fly. So, if you can spare the disk space you'll save time in the long run if you first decompress the file to disk, to get the `uniprot_sprot.dat` file inside. Then you can open the file for reading as usual:

```
In [13]: handle = open("data/uniprot_sprot.dat")
```

As of June 2009, the full Swiss-Prot database downloaded from ExPASy contained 468851 Swiss-Prot records. One concise way to build up a list of the record descriptions is with a list comprehension:

```
In [15]: from Bio import SwissProt
         handle = open("data/uniprot_sprot.dat")
         descriptions = [record.description for record in SwissProt.parse(handle)]
         len(descriptions)


Out[15]: 549832

In [16]: descriptions[:5]


Out[16]: ['RecName: Full=Putative transcription factor 001R;',
          'RecName: Full=Uncharacterized protein 002L;',
          'RecName: Full=Uncharacterized protein 002R;',
          'RecName: Full=Uncharacterized protein 003L;',
          'RecName: Full=Uncharacterized protein 3R; Flags: Precursor;']
```

Or, using a for loop over the record iterator:

```
In [17]: from Bio import SwissProt
         descriptions = []
         handle = open("data/uniprot_sprot.dat")
         for record in SwissProt.parse(handle):
             descriptions.append(record.description)

In [18]: len(descriptions)

Out[18]: 549832
```

Because this is such a large input file, either way takes about eleven minutes on my new desktop computer (using the uncompressed `uniprot_sprot.dat` file as input).

It is equally easy to extract any kind of information you'd like from Swiss-Prot records. To see the members of a Swiss-Prot record, use

```
In [19]: dir(record)

Out[19]: ['__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__gt__',
          '__hash__',
          '__init__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'accessions',
          'annotation_update',
          'comments',
          'created',
          'cross_references',
          'data_class',
          'description',
          'entry_name',
          'features',
          'gene_name',
          'host_organism',
          'host_taxonomy_id',
          'keywords',
          'molecule_type',
          'organelle',
          'organism',
          'organism_classification',
          'references',
          'seqinfo',
          'sequence',
          'sequence_length',
          'sequence_update',
          'taxonomy_id']
```

11.1.2 Parsing the Swiss-Prot keyword and category list

Swiss-Prot also distributes a file `keywlist.txt`, which lists the keywords and categories used in Swiss-Prot. The file contains entries in the following form:

```
ID   2Fe-2S.
AC   KW-0001
DE   Protein which contains at least one 2Fe-2S iron-sulfur cluster: 2 iron
DE   atoms complexed to 2 inorganic sulfides and 4 sulfur atoms of
DE   cysteines from the protein.
SY   Fe2S2; [2Fe-2S] cluster; [Fe2S2] cluster; Fe2/S2 (inorganic) cluster;
SY   Di-mu-sulfido-diiron; 2 iron, 2 sulfur cluster binding.
GO   GO:0051537; 2 iron, 2 sulfur cluster binding
HI   Ligand: Iron; Iron-sulfur; 2Fe-2S.
HI   Ligand: Metal-binding; 2Fe-2S.
CA   Ligand.
```

```
//
ID   3D-structure.
AC   KW-0002
DE   Protein, or part of a protein, whose three-dimensional structure has
DE   been resolved experimentally (for example by X-ray crystallography or
DE   NMR spectroscopy) and whose coordinates are available in the PDB
DE   database. Can also be used for theoretical models.
HI   Technical term: 3D-structure.
CA   Technical term.
//
ID   3Fe-4S.
...
```

The entries in this file can be parsed by the `parse` function in the `Bio.SwissProt.KeyWList` module. Each entry is then stored as a `Bio.SwissProt.KeyWList.Record`, which is a Python dictionary.

```
In [20]: from Bio.SwissProt import KeyWList
         handle = open("data/keywlist.txt")
         records = KeyWList.parse(handle)
         for record in records:
             print(record['ID'])
             print(record['DE'])
```

```
2Fe-2S.
Protein which contains at least one 2Fe-2S iron-sulfur cluster: 2 iron atoms complexed to 2 inorganic
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-20-8c149ecad36c> in <module>()
      3 records = KeyWList.parse(handle)
      4 for record in records:
----> 5     print(record['ID'])
      6     print(record['DE'])

KeyError: 'ID'
```

This prints

```
2Fe-2S.
Protein which contains at least one 2Fe-2S iron-sulfur cluster: 2 iron atoms
complexed to 2 inorganic sulfides and 4 sulfur atoms of cysteines from the
protein.
...
```

11.2 Parsing Prosite records

Prosite is a database containing protein domains, protein families, functional sites, as well as the patterns and profiles to recognize them. Prosite was developed in parallel with Swiss-Prot. In Biopython, a Prosite record is represented by the `Bio.ExPASy.Prosite.Record` class, whose members correspond to the different fields in a Prosite record.

In general, a Prosite file can contain more than one Prosite records. For example, the full set of Prosite records, which can be downloaded as a single file (`prosite.dat`) from the ExPASy FTP site, contains 2073 records (version 20.24 released on 4 December 2007). To parse such a file, we again make use of an iterator:

```
In [ ]: from Bio.ExPASy import Prosite
        handle = open("myprositefile.dat")
        records = Prosite.parse(handle)
```

We can now take the records one at a time and print out some information. For example, using the file containing the complete Prosite database, we'd find

```
In [ ]: from Bio.ExPASy import Prosite
        handle = open("prosite.dat")
        records = Prosite.parse(handle)
        record = next(records)
        record.accession
```

```
In [ ]: record.name
```

```
In [ ]: record.pdoc
```

```
In [ ]: record = next(records)
        record.accession
```

```
In [ ]: record.name
```

```
In [ ]: record.pdoc
```

```
In [ ]: record = next(records)
        record.accession
```

```
In [ ]: record.name
```

```
In [ ]: record.pdoc
```

and so on. If you're interested in how many Prosite records there are, you could use

```
In [ ]: from Bio.ExPASy import Prosite
        handle = open("prosite.dat")
        records = Prosite.parse(handle)
        n = 0
        for record in records:
            n += 1
```

```
In [ ]: n
```

To read exactly one Prosite from the handle, you can use the `read` function:

```
In [ ]: from Bio.ExPASy import Prosite
        handle = open("mysingleprositerecord.dat")
        record = Prosite.read(handle)
```

This function raises a ValueError if no Prosite record is found, and also if more than one Prosite record is found.

11.3 Parsing Prosite documentation records

In the Prosite example above, the `record.pdoc` accession numbers `'PDOC00001'`, `'PDOC00004'`, `'PDOC00005'` and so on refer to Prosite documentation. The Prosite documentation records are available from ExPASy as individual files, and as one file (`prosite.doc`) containing all Prosite documentation records.

We use the parser in `Bio.ExPASy.Prodoc` to parse Prosite documentation records. For example, to create a list of all accession numbers of Prosite documentation record, you can use

```
In [ ]: from Bio.ExPASy import Prodoc
        handle = open("prosite.doc")
        records = Prodoc.parse(handle)
        accessions = [record.accession for record in records]
```

Again a `read()` function is provided to read exactly one Prosite documentation record from the handle.

11.4 Parsing Enzyme records

ExPASy's Enzyme database is a repository of information on enzyme nomenclature. A typical Enzyme record looks as follows:

```
ID   3.1.1.34
DE   Lipoprotein lipase.
AN   Clearing factor lipase.
AN   Diacylglycerol lipase.
AN   Diglyceride lipase.
CA   Triacylglycerol + H(2)O = diacylglycerol + a carboxylate.
CC   -!- Hydrolyzes triacylglycerols in chylomicrons and very low-density
CC       lipoproteins (VLDL).
CC   -!- Also hydrolyzes diacylglycerol.
PR   PROSITE; PDOC00110;
DR   P11151, LIPL_BOVIN ;  P11153, LIPL_CAVPO ;  P11602, LIPL_CHICK ;
DR   P55031, LIPL_FELCA ;  P06858, LIPL_HUMAN ;  P11152, LIPL_MOUSE ;
DR   O46647, LIPL_MUSVI ;  P49060, LIPL_PAPAN ;  P49923, LIPL_PIG   ;
DR   Q06000, LIPL_RAT   ;  Q29524, LIPL_SHEEP ;
//
```

In this example, the first line shows the EC (Enzyme Commission) number of lipoprotein lipase (second line). Alternative names of lipoprotein lipase are "clearing factor lipase", "diacylglycerol lipase", and "diglyceride lipase" (lines 3 through 5). The line starting with "CA" shows the catalytic activity of this enzyme. Comment lines start with "CC". The "PR" line shows references to the Prosite Documentation records, and the "DR" lines show references to Swiss-Prot records. Not of these entries are necessarily present in an Enzyme record.

In Biopython, an Enzyme record is represented by the `Bio.ExPASy.Enzyme.Record` class. This record derives from a Python dictionary and has keys corresponding to the two-letter codes used in Enzyme files. To read an Enzyme file containing one Enzyme record, use the `read` function in `Bio.ExPASy.Enzyme`:

```
In [ ]: from Bio.ExPASy import Enzyme
        with open("data/lipoprotein.txt") as handle:
            record = Enzyme.read(handle)
```

```
In [ ]: record["ID"]
```

```
In [ ]: record["DE"]
```

```
In [ ]: record["AN"]
```

```
In [ ]: record["CA"]
```

```
In [ ]: record["PR"]
```

```
In [ ]: record["CC"]
```

```
In [ ]: record["DR"]
```

The `read` function raises a ValueError if no Enzyme record is found, and also if more than one Enzyme record is found.

The full set of Enzyme records can be downloaded as a single file (`enzyme.dat`) from the ExPASy FTP site, containing 4877 records (release of 3 March 2009). To parse such a file containing multiple Enzyme records, use the `parse` function in `Bio.ExPASy.Enzyme` to obtain an iterator:

```
In [ ]: from Bio.ExPASy import Enzyme
        handle = open("enzyme.dat")
        records = Enzyme.parse(handle)
```

We can now iterate over the records one at a time. For example, we can make a list of all EC numbers for which an Enzyme record is available:

```
In [ ]: ecnumbers = [record["ID"] for record in records]
```

11.5 Accessing the ExPASy server

Swiss-Prot, Prosite, and Prosite documentation records can be downloaded from the ExPASy web server at http://www.expasy.org. Six kinds of queries are available from ExPASy:

get_prodoc_entry To download a Prosite documentation record in HTML format

get_prosite_entry To download a Prosite record in HTML format

get_prosite_raw To download a Prosite or Prosite documentation record in raw format

get_sprot_raw To download a Swiss-Prot record in raw format

sprot_search_ful To search for a Swiss-Prot record

sprot_search_de To search for a Swiss-Prot record

To access this web server from a Python script, we use the `Bio.ExPASy` module.

11.5.1 Retrieving a Swiss-Prot record

Let's say we are looking at chalcone synthases for Orchids (see section [sec:orchids] for some justification for looking for interesting things about orchids). Chalcone synthase is involved in flavanoid biosynthesis in plants, and flavanoids make lots of cool things like pigment colors and UV protectants.

If you do a search on Swiss-Prot, you can find three orchid proteins for Chalcone Synthase, id numbers O23729, O23730, O23731. Now, let's write a script which grabs these, and parses out some interesting information.

First, we grab the records, using the `get_sprot_raw()` function of `Bio.ExPASy`. This function is very nice since you can feed it an id and get back a handle to a raw text record (no HTML to mess with!). We can the use `Bio.SwissProt.read` to pull out the Swiss-Prot record, or `Bio.SeqIO.read` to get a SeqRecord. The following code accomplishes what I just wrote:

```
In [21]: from Bio import ExPASy
         from Bio import SwissProt

In [22]: accessions = ["O23729", "O23730", "O23731"]
         records = []

In [23]: for accession in accessions:
             handle = ExPASy.get_sprot_raw(accession)
             record = SwissProt.read(handle)
             records.append(record)
```

```
---------------------------------------------------------------------------
IncompleteRead                            Traceback (most recent call last)
<ipython-input-23-37f8f432c5c4> in <module>()
      1 for accession in accessions:
      2     handle = ExPASy.get_sprot_raw(accession)
----> 3     record = SwissProt.read(handle)
      4     records.append(record)

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SwissProt/__init__.py in read(handle)
    132         raise ValueError("No SwissProt record found")
    133     # We should have reached the end of the record by now
--> 134     remainder = handle.read()
    135     if remainder:
    136         raise ValueError("More than one SwissProt record found")
```

```
/home/tiago_antao/miniconda/lib/python3.5/http/client.py in read(self, amt)
    444             else:
    445                 try:
--> 446                     s = self._safe_read(self.length)
    447                 except IncompleteRead:
    448                     self._close_conn()

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _safe_read(self, amt)
    592             chunk = self.fp.read(min(amt, MAXAMOUNT))
    593             if not chunk:
--> 594                 raise IncompleteRead(b''.join(s), amt)
    595             s.append(chunk)
    596             amt -= len(chunk)

IncompleteRead: IncompleteRead(0 bytes read, 3386 more expected)
```

If the accession number you provided to `ExPASy.get_sprot_raw` does not exist, then `SwissProt.read(handle)` will raise a `ValueError`. You can catch `ValueException` exceptions to detect invalid accession numbers:

```
In [24]: for accession in accessions:
             handle = ExPASy.get_sprot_raw(accession)
             try:
                 record = SwissProt.read(handle)
             except ValueException:
                 print("WARNING: Accession %s not found" % accession)
             records.append(record)

---------------------------------------------------------------------------
IncompleteRead                            Traceback (most recent call last)
<ipython-input-24-e050fd24d4b9> in <module>()
      3     try:
----> 4         record = SwissProt.read(handle)
      5     except ValueException:

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SwissProt/__init__.py in read(handle)
    133     # We should have reached the end of the record by now
--> 134     remainder = handle.read()
    135     if remainder:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in read(self, amt)
    445                 try:
--> 446                     s = self._safe_read(self.length)
    447                 except IncompleteRead:

/home/tiago_antao/miniconda/lib/python3.5/http/client.py in _safe_read(self, amt)
    593             if not chunk:
--> 594                 raise IncompleteRead(b''.join(s), amt)
    595             s.append(chunk)

IncompleteRead: IncompleteRead(0 bytes read, 3386 more expected)

During handling of the above exception, another exception occurred:

NameError                                 Traceback (most recent call last)
<ipython-input-24-e050fd24d4b9> in <module>()
      3     try:
      4         record = SwissProt.read(handle)
----> 5     except ValueException:
      6         print("WARNING: Accession %s not found" % accession)
      7     records.append(record)

NameError: name 'ValueException' is not defined
```

11.5.2 Searching Swiss-Prot

Now, you may remark that I knew the records' accession numbers beforehand. Indeed, `get_sprot_raw()` needs either the entry name or an accession number. When you don't have them handy, you can use one of the `sprot_search_de()` or `sprot_search_ful()` functions.

`sprot_search_de()` searches in the ID, DE, GN, OS and OG lines; `sprot_search_ful()` searches in (nearly) all the fields. They are detailed on http://www.expasy.org/cgi-bin/sprot-search-de and http://www.expasy.org/cgi-bin/sprot-search-ful respectively. Note that they don't search in TrEMBL by default (argument `trembl`). Note also that they return HTML pages; however, accession numbers are quite easily extractable:

```
In [25]: from Bio import ExPASy
         import re
```

```
In [26]: handle = ExPASy.sprot_search_de("Orchid Chalcone Synthase")
         # or:
         # handle = ExPASy.sprot_search_ful("Orchid and {Chalcone Synthase}")
         html_results = handle.read()
         if "Number of sequences found" in html_results:
             ids = re.findall(r'HREF="/uniprot/(\w+)"', html_results)
         else:
             ids = re.findall(r'href="/cgi-bin/niceprot\.pl\?(\w+)"', html_results)
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
<ipython-input-26-ee45c5db7457> in <module>()
----> 1 handle = ExPASy.sprot_search_de("Orchid Chalcone Synthase")
      2 # or:
      3 # handle = ExPASy.sprot_search_ful("Orchid and {Chalcone Synthase}")
      4 html_results = handle.read()
      5 if "Number of sequences found" in html_results:

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/ExPASy/__init__.py in sprot_search_de(tex
    111     options = _urlencode(variables)
    112     fullcgi = "%s?%s" % (cgi, options)
--> 113     handle = _urlopen(fullcgi)
    114     return handle

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in urlopen(url, data, timeout, cafile, ca
    160     else:
    161         opener = _opener
--> 162     return opener.open(url, data, timeout)
    163 
    164 def install_opener(opener):

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
```

```
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302
```

```
/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582 
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    507         if http_err:
    508             args = (dict, 'default', 'http_error_default') + orig_args
--> 509             return self._call_chain(*args)
    510 
    511 # XXX probably also want an abstract factory that knows when it makes

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_default(self, req, fp, code
    587 class HTTPDefaultErrorHandler(BaseHandler):
    588     def http_error_default(self, req, fp, code, msg, hdrs):
--> 589         raise HTTPError(req.full_url, code, msg, hdrs, fp)
    590 
    591 class HTTPRedirectHandler(BaseHandler):

HTTPError: HTTP Error 301: Moved Permanently
```

11.5.3 Retrieving Prosite and Prosite documentation records

Prosite and Prosite documentation records can be retrieved either in HTML format, or in raw format. To parse Prosite and Prosite documentation records with Biopython, you should retrieve the records in raw format. For other purposes, however, you may be interested in these records in HTML format.

To retrieve a Prosite or Prosite documentation record in raw format, use `get_prosite_raw()`. For example, to download a Prosite record and print it out in raw text format, use

```
In [27]: from Bio import ExPASy
         handle = ExPASy.get_prosite_raw('PS00001')
         text = handle.read()
         print(text)
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
<ipython-input-27-8c96cea49d6d> in <module>()
      1 from Bio import ExPASy
----> 2 handle = ExPASy.get_prosite_raw('PS00001')
      3 text = handle.read()
      4 print(text)
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/ExPASy/__init__.py in get_prosite_raw(id,
     62     For a non-existing key, ExPASy returns nothing.
     63     """
---> 64     return _urlopen("%s?%s" % (cgi, id))
     65 
     66 

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in urlopen(url, data, timeout, cafile, ca
    160     else:
    161         opener = _opener
--> 162     return opener.open(url, data, timeout)
    163 
    164 def install_opener(opener):

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
```

```
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    507         if http_err:
    508             args = (dict, 'default', 'http_error_default') + orig_args
--> 509             return self._call_chain(*args)
    510 
    511 # XXX probably also want an abstract factory that knows when it makes

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_default(self, req, fp, code
    587 class HTTPDefaultErrorHandler(BaseHandler):
    588     def http_error_default(self, req, fp, code, msg, hdrs):
--> 589         raise HTTPError(req.full_url, code, msg, hdrs, fp)
    590 
    591 class HTTPRedirectHandler(BaseHandler):

HTTPError: HTTP Error 301: Moved Permanently
```

To retrieve a Prosite record and parse it into a `Bio.Prosite.Record` object, use

```
In [28]: from Bio import ExPASy
         from Bio import Prosite
         handle = ExPASy.get_prosite_raw('PS00001')
         record = Prosite.read(handle)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-28-f4e345e9eb60> in <module>()
      1 from Bio import ExPASy
----> 2 from Bio import Prosite
      3 handle = ExPASy.get_prosite_raw('PS00001')
      4 record = Prosite.read(handle)

ImportError: cannot import name 'Prosite'
```

The same function can be used to retrieve a Prosite documentation record and parse it into a `Bio.ExPASy.Prodoc.Record` object:

```
In [29]: from Bio import ExPASy
         from Bio.ExPASy import Prodoc
         handle = ExPASy.get_prosite_raw('PDOC00001')
         record = Prodoc.read(handle)
```

```
---------------------------------------------------------------------------
HTTPError                                 Traceback (most recent call last)
<ipython-input-29-7482db7f5b87> in <module>()
      1 from Bio import ExPASy
      2 from Bio.ExPASy import Prodoc
----> 3 handle = ExPASy.get_prosite_raw('PDOC00001')
      4 record = Prodoc.read(handle)

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/ExPASy/__init__.py in get_prosite_raw(id,
     62     For a non-existing key, ExPASy returns nothing.
     63     """
---> 64     return _urlopen("%s?%s" % (cgi, id))
     65 
     66 

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in urlopen(url, data, timeout, cafile, ca
    160     else:
    161         opener = _opener
--> 162     return opener.open(url, data, timeout)
    163 
    164 def install_opener(opener):

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
```

```
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    501             http_err = 0
    502         args = (dict, proto, meth_name) + args
--> 503         result = self._call_chain(*args)
    504         if result:
    505             return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_302(self, req, fp, code, ms
    684         fp.close()
    685 
--> 686         return self.parent.open(new, timeout=req.timeout)
    687 
    688     http_error_301 = http_error_303 = http_error_307 = http_error_302

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in open(self, fullurl, data, timeout)
    469         for processor in self.process_response.get(protocol, []):
    470             meth = getattr(processor, meth_name)
--> 471             response = meth(req, response)
    472 
    473         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_response(self, request, response)
    579         if not (200 <= code < 300):
    580             response = self.parent.error(
--> 581                 'http', request, response, code, msg, hdrs)    582
    583         return response

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in error(self, proto, *args)
    507         if http_err:
    508             args = (dict, 'default', 'http_error_default') + orig_args
--> 509             return self._call_chain(*args)
    510
    511 # XXX probably also want an abstract factory that knows when it makes

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in _call_chain(self, chain, kind, meth_na
    441         for handler in handlers:
    442             func = getattr(handler, meth_name)
--> 443             result = func(*args)
    444             if result is not None:
    445                 return result

/home/tiago_antao/miniconda/lib/python3.5/urllib/request.py in http_error_default(self, req, fp, code
    587 class HTTPDefaultErrorHandler(BaseHandler):
    588     def http_error_default(self, req, fp, code, msg, hdrs):
--> 589         raise HTTPError(req.full_url, code, msg, hdrs, fp)
    590
    591 class HTTPRedirectHandler(BaseHandler):

HTTPError: HTTP Error 301: Moved Permanently
```

For non-existing accession numbers, `ExPASy.get_prosite_raw` returns a handle to an emptry string. When faced with an empty string, `Prosite.read` and `Prodoc.read` will raise a ValueError. You can catch these exceptions to detect invalid accession numbers.

The functions `get_prosite_entry()` and `get_prodoc_entry()` are used to download Prosite and Prosite documentation records in HTML format. To create a web page showing one Prosite record, you can use

```
In [ ]: from Bio import ExPASy
        handle = ExPASy.get_prosite_entry('PS00001')
        html = handle.read()
        output = open("mypositerecord.html", "w")
        output.write(html)
        output.close()
```

and similarly for a Prosite documentation record:

```
In [30]: from Bio import ExPASy
         handle = ExPASy.get_prodoc_entry('PDOC00001')
         html = handle.read()
         output = open("myprodocrecord.html", "w")
         output.write(html)
         output.close()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-1fead8290c3a> in <module>()
      3 html = handle.read()
      4 output = open("myprodocrecord.html", "w")
----> 5 output.write(html)
      6 output.close()
```

```
TypeError: write() argument must be str, not bytes
```

For these functions, an invalid accession number returns an error message in HTML format.

11.6 Scanning the Prosite database

ScanProsite allows you to scan protein sequences online against the Prosite database by providing a UniProt or PDB sequence identifier or the sequence itself. For more information about ScanProsite, please see the ScanProsite documentation as well as the documentation for programmatic access of ScanProsite.

You can use Biopython's `Bio.ExPASy.ScanProsite` module to scan the Prosite database from Python. This module both helps you to access ScanProsite programmatically, and to parse the results returned by ScanProsite. To scan for Prosite patterns in the following protein sequence:

```
MEHKEVVLLLLLFLKSGQGEPLDDYVNTQGASLFSVTKKQLGAGSIEECAAKCEEDEEFT
CRAFQYHSKEQQCVIMAENRKSSIIIRMRDVVLFEKKVYLSECKTGNGKNYRGTMSKTKN
```

you can use the following code:

```
In [31]: sequence = "MEHKEVVLLLLLFLKSGQGEPLDDYVNTQGASLFSVTKKQLGAGSIEECAAKCEEDEEFT

  File "<ipython-input-31-a3137c4e2fe3>", line 1
    sequence = "MEHKEVVLLLLLFLKSGQGEPLDDYVNTQGASLFSVTKKQLGAGSIEECAAKCEEDEEFT
                                                                             ^

SyntaxError: EOL while scanning string literal


In [32]: from Bio.ExPASy import ScanProsite
         handle = ScanProsite.scan(seq=sequence)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-fbf16249a192> in <module>()
      1 from Bio.ExPASy import ScanProsite
----> 2 handle = ScanProsite.scan(seq=sequence)

NameError: name 'sequence' is not defined
```

By executing `handle.read()`, you can obtain the search results in raw XML format. Instead, let's use `Bio.ExPASy.ScanProsite.read` to parse the raw XML into a Python object:

```
In [33]: result = ScanProsite.read(handle)
         type(result)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-33-8794b65cb5b5> in <module>()
----> 1 result = ScanProsite.read(handle)
      2 type(result)

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/ExPASy/ScanProsite.py in read(handle)
     65     saxparser = Parser()
     66     saxparser.setContentHandler(content_handler)
---> 67     saxparser.parse(handle)
     68     record = content_handler.record
     69     return record

/home/tiago_antao/miniconda/lib/python3.5/xml/sax/expatreader.py in parse(self, source)
    108         self.reset()
    109         self._cont_handler.setDocumentLocator(ExpatLocator(self))
--> 110         xmlreader.IncrementalParser.parse(self, source)
    111 
    112     def prepareParser(self, source):

/home/tiago_antao/miniconda/lib/python3.5/xml/sax/xmlreader.py in parse(self, source)
    125             self.feed(buffer)
    126             buffer = file.read(self._bufsize)
--> 127         self.close()
    128 
    129     def feed(self, data):

/home/tiago_antao/miniconda/lib/python3.5/xml/sax/expatreader.py in close(self)
    220             return
    221         try:
--> 222             self.feed("", isFinal = 1)
    223             self._cont_handler.endDocument()
    224             self._parsing = 0

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/ExPASy/ScanProsite.py in feed(self, data
     85         # fed to the parser.
     86         if self.firsttime:
---> 87             if data[:5].decode('utf-8') != "<?xml":
     88                 raise ValueError(data)
     89         self.firsttime = False

AttributeError: 'str' object has no attribute 'decode'
```

A `Bio.ExPASy.ScanProsite.Record` object is derived from a list, with each element in the list storing one ScanProsite hit. This object also stores the number of hits, as well as the number of search sequences, as returned by ScanProsite. This ScanProsite search resulted in six hits:

```
In [ ]: result.n_seq
```

```
In [ ]: result.n_match
```

```
In [ ]: len(result)
```

```
In [ ]: result[0]
```

```
In [ ]: result[1]
```

```
In [ ]: result[2]
```

```
In [ ]: result[3]
```

```
In [ ]: result[4]
```

```
In [ ]: result[5]
```

Other ScanProsite parameters can be passed as keyword arguments; see the documentation for programmatic access of ScanProsite for more information. As an example, passing `lowscore=1` to include matches with low level scores lets use find one additional hit:

```
In [ ]: handle = ScanProsite.scan(seq=sequence, lowscore=1)
        result = ScanProsite.read(handle)
        result.n_match
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 12

Going 3D: The PDB module

Bio.PDB is a Biopython module that focuses on working with crystal structures of biological macromolecules. Among other things, Bio.PDB includes a PDBParser class that produces a Structure object, which can be used to access the atomic data in the file in a convenient manner. There is limited support for parsing the information contained in the PDB header.

12.1 Reading and writing crystal structure files

12.1.1 Reading a PDB file

First we create a `PDBParser` object:

```
In [1]: from Bio.PDB.PDBParser import PDBParser
        p = PDBParser(PERMISSIVE=1)
```

The PERMISSIVE flag indicates that a number of common problems (see [problem structures]) associated with PDB files will be ignored (but note that some atoms and/or residues will be missing). If the flag is not present a PDBConstructionException will be generated if any problems are detected during the parse operation.

The Structure object is then produced by letting the `PDBParser` object parse a PDB file (the PDB file in this case is called 'pdb1fat.ent', '1fat' is a user defined name for the structure):

```
In [2]: structure_id = "1fat"
        filename = "data/pdb1fat.ent"
        structure = p.get_structure(structure_id, filename)
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
```

```
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
```

You can extract the header and trailer (simple lists of strings) of the PDB file from the PDBParser object with the get_header and get_trailer methods. Note however that many PDB files contain headers with incomplete or erroneous information. Many of the errors have been fixed in the equivalent mmCIF files. *Hence, if you are interested in the header information, it is a good idea to extract information from mmCIF files using the* `MMCIF2Dict` *tool described below, instead of parsing the PDB header.*

Now that is clarified, let's return to parsing the PDB header. The structure object has an attribute called `header` which is a Python dictionary that maps header records to their values.

Example:

```
In [3]: resolution = structure.header['resolution']
        keywords = structure.header['keywords']
```

The available keys are `name`, `head`, `deposition_date`, `release_date`, `structure_method`, `resolution`, `structure_reference` (which maps to a list of references), `journal_reference`, `author`, and `compound` (which maps to a dictionary with various information about the crystallized compound).

The dictionary can also be created without creating a `Structure` object, ie. directly from the PDB file:

```
In [4]: file = open(filename, 'r')
        header_dict = parse_pdb_header(file)
        file.close()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-48b9e36dd9f4> in <module>()
      1 file = open(filename, 'r')
----> 2 header_dict = parse_pdb_header(file)
      3 file.close()

NameError: name 'parse_pdb_header' is not defined
```

12.1.2 Reading an mmCIF file

Similarly to the case the case of PDB files, first create an `MMCIFParser` object:

```
In [5]: from Bio.PDB.MMCIFParser import MMCIFParser
        parser = MMCIFParser()
```

Then use this parser to create a structure object from the mmCIF file:

```
In [6]: structure = parser.get_structure('1fat', 'data/1fat.cif')
```

```
---------------------------------------------------------------------------
PDBConstructionException                  Traceback (most recent call last)
<ipython-input-6-218e52367060> in <module>()
----> 1 structure = parser.get_structure('1fat', 'data/1fat.cif')

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/MMCIFParser.py in get_structure(self,
     59                 warnings.filterwarnings("ignore", category=PDBConstructionWarning)
     60         self._mmcif_dict = MMCIF2Dict(filename)
---> 61         self._build_structure(structure_id)
     62         return self._structure_builder.get_structure()
     63

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/MMCIFParser.py in _build_structure(se
    177             element = element_list[i] if element_list else None
    178             structure_builder.init_atom(name, coord, tempfactor, occupancy, altloc,
--> 179                 name, element=element)    180             if aniso_flag == 1:
    181                 u = (aniso_u11[i], aniso_u12[i], aniso_u13[i],

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py in init_atom(self
    239         else:
    240             # The atom is not disordered
--> 241             residue.add(self.atom)
    242 
    243     def set_anisou(self, anisou_array):

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/Residue.py in add(self, atom)
     78         if self.has_id(atom_id):
     79             raise PDBConstructionException(
---> 80                 "Atom %s defined twice in residue %s" % (atom_id, self))     81         Entit
     82 

PDBConstructionException: Atom C1 defined twice in residue <Residue NAG het=H_NAG resseq=253 icode= >
```

To have some more low level access to an mmCIF file, you can use the `MMCIF2Dict` class to create a Python dictionary that maps all mmCIF tags in an mmCIF file to their values. If there are multiple values (like in the case of tag `_atom_site.Cartn_y`, which holds the y coordinates of all atoms), the tag is mapped to a list of values. The dictionary is created from the mmCIF file as follows:

```
In [7]: from Bio.PDB.MMCIF2Dict import MMCIF2Dict
        mmcif_dict = MMCIF2Dict('data/1fat.cif')
```

Example: get the solvent content from an mmCIF file:

```
In [8]: sc = mmcif_dict['_exptl_crystal.density_percent_sol']
```

Example: get the list of the y coordinates of all atoms

```
In [9]: y_list = mmcif_dict['_atom_site.Cartn_y']
```

12.1.3 Reading files in the PDB XML format

That's not yet supported, but we are definitely planning to support that in the future (it's not a lot of work). Contact the Biopython developers () if you need this).

12.1.4 Writing PDB files

Use the PDBIO class for this. It's easy to write out specific parts of a structure too, of course.

Example: saving a structure

```
In [11]: from Bio.PDB import PDBIO
         io = PDBIO()
         io.set_structure(s)
         io.save('out.pdb')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-bcca00d727d1> in <module>()
      1 from Bio.PDB import PDBIO
      2 io = PDBIO()
```

```
----> 3 io.set_structure(s)
      4 io.save('out.pdb')

NameError: name 's' is not defined
```

If you want to write out a part of the structure, make use of the `Select` class (also in `PDBIO`). Select has four methods:

- `accept_model(model)`
- `accept_chain(chain)`
- `accept_residue(residue)`
- `accept_atom(atom)`

By default, every method returns 1 (which means the model/chain/residue/atom is included in the output). By subclassing `Select` and returning 0 when appropriate you can exclude models, chains, etc. from the output. Cumbersome maybe, but very powerful. The following code only writes out glycine residues:

```
In [14]: from Bio.PDB.PDBIO import Select
         class GlySelect(Select):
             def accept_residue(self, residue):
                 if residue.get_name() == 'GLY':
                     return True
                 else:
                     return False
```

```
In [15]: io = PDBIO()
         io.set_structure(s)
         io.save('gly_only.pdb', GlySelect())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-10edd36b5ec5> in <module>()
      1 io = PDBIO()
----> 2 io.set_structure(s)
      3 io.save('gly_only.pdb', GlySelect())

NameError: name 's' is not defined
```

If this is all too complicated for you, the `Dice` module contains a handy `extract` function that writes out all residues in a chain between a start and end residue.

12.2 Structure representation

The overall layout of a `Structure` object follows the so-called SMCRA (Structure/Model/Chain/Residue/Atom) architecture:

- A structure consists of models
- A model consists of chains
- A chain consists of residues
- A residue consists of atoms

This is the way many structural biologists/bioinformaticians think about structure, and provides a simple but efficient way to deal with structure. Additional stuff is essentially added when needed. A UML diagram of the `Structure` object (forget about the `Disordered` classes for now) is shown in Fig. [fig:smcra]. Such a data structure is not necessarily best suited for the representation of the macromolecular content of a structure, but it is absolutely necessary for a good interpretation of the data present in a file that describes the structure (typically a PDB or MMCIF file). If this hierarchy cannot represent the contents of a structure file, it is fairly certain that the file contains an error or at least does not describe the structure unambiguously. If a SMCRA data structure cannot be generated, there is reason to suspect a problem. Parsing a PDB file can thus be used to detect likely problems. We will give several examples of this in section [problem structures].

Structure, Model, Chain and Residue are all subclasses of the Entity base class. The Atom class only (partly) implements the Entity interface (because an Atom does not have children).

For each Entity subclass, you can extract a child by using a unique id for that child as a key (e.g. you can extract an Atom object from a Residue object by using an atom name string as a key, you can extract a Chain object from a Model object by using its chain identifier as a key).

Disordered atoms and residues are represented by DisorderedAtom and DisorderedResidue classes, which are both subclasses of the DisorderedEntityWrapper base class. They hide the complexity associated with disorder and behave exactly as Atom and Residue objects.

In general, a child Entity object (i.e. Atom, Residue, Chain, Model) can be extracted from its parent (i.e. Residue, Chain, Model, Structure, respectively) by using an id as a key.

```
In [16]: child_entity = parent_entity[child_id]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-4f5136545867> in <module>()
----> 1 child_entity = parent_entity[child_id]

NameError: name 'parent_entity' is not defined
```

You can also get a list of all child Entities of a parent Entity object. Note that this list is sorted in a specific way (e.g. according to chain identifier for Chain objects in a Model object).

```
In [ ]: child_list = parent_entity.get_list()
```

You can also get the parent from a child:

```
In [17]: parent_entity = child_entity.get_parent()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-9871f6bbb145> in <module>()
----> 1 parent_entity = child_entity.get_parent()

NameError: name 'child_entity' is not defined
```

At all levels of the SMCRA hierarchy, you can also extract a *full id*. The full id is a tuple containing all id's starting from the top object (Structure) down to the current object. A full id for a Residue object e.g. is something like:

```
In [18]: full_id = residue.get_full_id()
         print(full_id)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-985fe60adf7d> in <module>()
----> 1 full_id = residue.get_full_id()
      2 print(full_id)

NameError: name 'residue' is not defined
```

This corresponds to:

- The Structure with id '"1abc'"
- The Model with id 0

- The Chain with id '"A'"
- The Residue with id ('" '", 10, '"A'").

The Residue id indicates that the residue is not a hetero-residue (nor a water) because it has a blank hetero field, that its sequence identifier is 10 and that its insertion code is '"A'".

To get the entity's id, use the `get_id` method:

```
In [19]: entity.get_id()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-f97a391c9568> in <module>()
----> 1 entity.get_id()

NameError: name 'entity' is not defined
```

You can check if the entity has a child with a given id by using the `has_id` method:

```
In [ ]: entity.has_id(entity_id)
```

The length of an entity is equal to its number of children:

```
In [ ]: nr_children = len(entity)
```

It is possible to delete, rename, add, etc. child entities from a parent entity, but this does not include any sanity checks (e.g. it is possible to add two residues with the same id to one chain). This really should be done via a nice Decorator class that includes integrity checking, but you can take a look at the code (Entity.py) if you want to use the raw interface.

12.2.1 Structure

The Structure object is at the top of the hierarchy. Its id is a user given string. The Structure contains a number of Model children. Most crystal structures (but not all) contain a single model, while NMR structures typically consist of several models. Disorder in crystal structures of large parts of molecules can also result in several models.

12.2.2 Model

The id of the Model object is an integer, which is derived from the position of the model in the parsed file (they are automatically numbered starting from 0). Crystal structures generally have only one model (with id 0), while NMR files usually have several models. Whereas many PDB parsers assume that there is only one model, the `Structure` class in `Bio.PDB` is designed such that it can easily handle PDB files with more than one model.

As an example, to get the first model from a Structure object, use

```
In [20]: first_model = structure[0]
```

The Model object stores a list of Chain children.

12.2.3 Chain

The id of a Chain object is derived from the chain identifier in the PDB/mmCIF file, and is a single character (typically a letter). Each Chain in a Model object has a unique id. As an example, to get the Chain object with identifier "A" from a Model object, use

```
In [21]: chain_A = model["A"]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-45b288421d99> in <module>()
----> 1 chain_A = model["A"]

NameError: name 'model' is not defined
```

The Chain object stores a list of Residue children.

12.2.4 Residue

A residue id is a tuple with three elements:

- The **hetero-field** (hetfield): this is
  - `'W'` in the case of a water molecule;
  - `'H_'` followed by the residue name for other hetero residues (e.g. `'H_GLC'` in the case of a glucose molecule);
  - blank for standard amino and nucleic acids.

  This scheme is adopted for reasons described in section [hetero problems].
- The **sequence identifier** (resseq), an integer describing the position of the residue in the chain (e.g., 100);
- The **insertion code** (icode); a string, e.g. 'A'. The insertion code is sometimes used to preserve a certain desirable residue numbering scheme. A Ser 80 insertion mutant (inserted e.g. between a Thr 80 and an Asn 81 residue) could e.g. have sequence identifiers and insertion codes as follows: Thr 80 A, Ser 80 B, Asn 81. In this way the residue numbering scheme stays in tune with that of the wild type structure.

The id of the above glucose residue would thus be `('H_GLC', 100, 'A')`. If the hetero-flag and insertion code are blank, the sequence identifier alone can be used:

```
In [22]: # Full id
         residue = chain[(' ', 100, ' ')]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-c937bc6cdff3> in <module>()
      1 # Full id
----> 2 residue = chain[(' ', 100, ' ')]

NameError: name 'chain' is not defined
```

```
In [23]: residue = chain[100]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-6ca58ca97644> in <module>()
----> 1 residue = chain[100]

NameError: name 'chain' is not defined
```

The reason for the hetero-flag is that many, many PDB files use the same sequence identifier for an amino acid and a hetero-residue or a water, which would create obvious problems if the hetero-flag was not used.

Unsurprisingly, a Residue object stores a set of Atom children. It also contains a string that specifies the residue name (e.g. "ASN") and the segment identifier of the residue (well known to X-PLOR users, but not used in the construction of the SMCRA data structure).

Let's look at some examples. Asn 10 with a blank insertion code would have residue id (' ', 10, ' '). Water 10 would have residue id ('W', 10, ' '). A glucose molecule (a hetero residue with residue name GLC) with sequence identifier

10 would have residue id ('H_GLC', 10, ' '). In this way, the three residues (with the same insertion code and sequence identifier) can be part of the same chain because their residue id's are distinct.

In most cases, the hetflag and insertion code fields will be blank, e.g. (' ', 10, ' '). In these cases, the sequence identifier can be used as a shortcut for the full id:

```
In [ ]: # use full id
        res10 = chain[(' ', 10, ' ')]
```

```
In [ ]: res10 = chain[10]
```

Each Residue object in a Chain object should have a unique id. However, disordered residues are dealt with in a special way, as described in section [point mutations].

A Residue object has a number of additional methods:

```
In [24]: residue.get_resname()       # returns the residue name, e.g. "ASN"
         residue.is_disordered()     # returns 1 if the residue has disordered atoms
         residue.get_segid()         # returns the SEGID, e.g. "CHN1"
         residue.has_id(name)        # test if a residue has a certain atom
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-a34e77e319d2> in <module>()
----> 1 residue.get_resname()       # returns the residue name, e.g. "ASN"
      2 residue.is_disordered()     # returns 1 if the residue has disordered atoms
      3 residue.get_segid()         # returns the SEGID, e.g. "CHN1"
      4 residue.has_id(name)        # test if a residue has a certain atom

NameError: name 'residue' is not defined
```

You can use `is_aa(residue)` to test if a Residue object is an amino acid.

12.2.5 Atom

The Atom object stores the data associated with an atom, and has no children. The id of an atom is its atom name (e.g. "OG" for the side chain oxygen of a Ser residue). An Atom id needs to be unique in a Residue. Again, an exception is made for disordered atoms, as described in section [disordered atoms].

The atom id is simply the atom name (eg. `'CA'`). In practice, the atom name is created by stripping all spaces from the atom name in the PDB file.

However, in PDB files, a space can be part of an atom name. Often, calcium atoms are called `'CA..'` in order to distinguish them from Cα atoms (which are called `'.CA.'`). In cases were stripping the spaces would create problems (ie. two atoms called `'CA'` in the same residue) the spaces are kept.

In a PDB file, an atom name consists of 4 chars, typically with leading and trailing spaces. Often these spaces can be removed for ease of use (e.g. an amino acid C$:raw-latex:`alpha `$ atom is labeled ".CA." in a PDB file, where the dots represent spaces). To generate an atom name (and thus an atom id) the spaces are removed, unless this would result in a name collision in a Residue (i.e. two Atom objects with the same atom name and id). In the latter case, the atom name including spaces is tried. This situation can e.g. happen when one residue contains atoms with names ".CA." and "CA..", although this is not very likely.

The atomic data stored includes the atom name, the atomic coordinates (including standard deviation if present), the B factor (including anisotropic B factors and standard deviation if present), the altloc specifier and the full atom name including spaces. Less used items like the atom element number or the atomic charge sometimes specified in a PDB file are not stored.

To manipulate the atomic coordinates, use the `transform` method of the `Atom` object. Use the `set_coord` method to specify the atomic coordinates directly.

An Atom object has the following additional methods:

```
In [25]: a.get_name()       # atom name (spaces stripped, e.g. "CA")
         a.get_id()         # id (equals atom name)
         a.get_coord()      # atomic coordinates
         a.get_vector()     # atomic coordinates as Vector object
         a.get_bfactor()    # isotropic B factor
         a.get_occupancy()  # occupancy
         a.get_altloc()     # alternative location specifier
         a.get_sigatm()     # standard deviation of atomic parameters
         a.get_siguij()     # standard deviation of anisotropic B factor
         a.get_anisou()     # anisotropic B factor
         a.get_fullname()   # atom name (with spaces, e.g. ".CA.")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-2b9bddc52a80> in <module>()
----> 1 a.get_name()       # atom name (spaces stripped, e.g. "CA")
      2 a.get_id()         # id (equals atom name)
      3 a.get_coord()      # atomic coordinates
      4 a.get_vector()     # atomic coordinates as Vector object
      5 a.get_bfactor()    # isotropic B factor

NameError: name 'a' is not defined
```

To represent the atom coordinates, siguij, anisotropic B factor and sigatm Numpy arrays are used.

The `get_vector` method returns a `Vector` object representation of the coordinates of the `Atom` object, allowing you to do vector operations on atomic coordinates. `Vector` implements the full set of 3D vector operations, matrix multiplication (left and right) and some advanced rotation-related operations as well.

As an example of the capabilities of Bio.PDB's `Vector` module, suppose that you would like to find the position of a Gly residue's Cβ atom, if it had one. Rotating the N atom of the Gly residue along the Cα-C bond over -120 degrees roughly puts it in the position of a virtual Cβ atom. Here's how to do it, making use of the `rotaxis` method (which can be used to construct a rotation around a certain axis) of the `Vector` module:

```
In [26]: # get atom coordinates as vectors
         n = residue['N'].get_vector()
         c = residue['C'].get_vector()
         ca = residue['CA'].get_vector()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-e0172a5c1106> in <module>()
      1 # get atom coordinates as vectors
----> 2 n = residue['N'].get_vector()
      3 c = residue['C'].get_vector()
      4 ca = residue['CA'].get_vector()

NameError: name 'residue' is not defined
```

```
In [27]: n = n - ca
         c = c - ca
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-c8c7b51cbaf3> in <module>()
----> 1 n = n - ca
      2 c = c - ca

NameError: name 'n' is not defined
```

```
In [28]: rot = rotaxis(-pi * 120.0/180.0, c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-6b8da02d67ce> in <module>()
----> 1 rot = rotaxis(-pi * 120.0/180.0, c)

NameError: name 'rotaxis' is not defined
```

```
In [29]: cb_at_origin = n.left_multiply(rot)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-b23ae22bff68> in <module>()
----> 1 cb_at_origin = n.left_multiply(rot)

NameError: name 'n' is not defined
```

```
In [30]: cb = cb_at_origin + ca
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-30-790f34c5734f> in <module>()
----> 1 cb = cb_at_origin + ca

NameError: name 'cb_at_origin' is not defined
```

This example shows that it's possible to do some quite nontrivial vector operations on atomic data, which can be quite useful. In addition to all the usual vector operations (cross (use `**`), and dot (use `*`) product, angle, norm, etc.) and the above mentioned `rotaxis` function, the `Vector` module also has methods to rotate (`rotmat`) or reflect (`refmat`) one vector on top of another.

12.2.6 Extracting a specific `Atom/Residue/Chain/Model` from a Structure

These are some examples:

```
In [31]: model = structure[0]
         chain = model['A']
         residue = chain[100]
         atom = residue['CA']
```

Note that you can use a shortcut:

```
In [32]: atom = structure[0]['A'][100]['CA']
```

12.3 Disorder

Bio.PDB can handle both disordered atoms and point mutations (i.e. a Gly and an Ala residue in the same position).

12.3.1 General approach[disorder problems]

Disorder should be dealt with from two points of view: the atom and the residue points of view. In general, we have tried to encapsulate all the complexity that arises from disorder. If you just want to loop over all Cα atoms, you do not care that some residues have a disordered side chain. On the other hand it should also be possible to represent disorder completely in the data structure. Therefore, disordered atoms or residues are stored in special objects that behave as if there is no disorder. This is done by only representing a subset of the disordered atoms or residues. Which subset is picked (e.g. which of the two disordered OG side chain atom positions of a Ser residue is used) can be specified by the user.

12.3.2 Disordered atoms[disordered atoms]

Disordered atoms are represented by ordinary `Atom` objects, but all `Atom` objects that represent the same physical atom are stored in a `DisorderedAtom` object (see Fig. [fig:smcra]). Each `Atom` object in a `DisorderedAtom` object can be uniquely indexed using its altloc specifier. The `DisorderedAtom` object forwards all uncaught method calls to the selected Atom object, by default the one that represents the atom with the highest occupancy. The user can of course change the selected `Atom` object, making use of its altloc specifier. In this way atom disorder is represented correctly without much additional complexity. In other words, if you are not interested in atom disorder, you will not be bothered by it.

Each disordered atom has a characteristic altloc identifier. You can specify that a `DisorderedAtom` object should behave like the `Atom` object associated with a specific altloc identifier:

```
In [33]: atom.disordered_select('A') # select altloc A atom
         print(atom.get_altloc())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-33-b874ea06f929> in <module>()
----> 1 atom.disordered_select('A') # select altloc A atom
      2 print(atom.get_altloc())

AttributeError: 'Atom' object has no attribute 'disordered_select'
```

```
In [34]: atom.disordered_select('B') # select altloc B atom
         print(atom.get_altloc())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-f32dfd0feabf> in <module>()
----> 1 atom.disordered_select('B') # select altloc B atom
      2 print(atom.get_altloc())

AttributeError: 'Atom' object has no attribute 'disordered_select'
```

12.3.3 Disordered residues

Common case

The most common case is a residue that contains one or more disordered atoms. This is evidently solved by using DisorderedAtom objects to represent the disordered atoms, and storing the DisorderedAtom object in a Residue object just like ordinary Atom objects. The DisorderedAtom will behave exactly like an ordinary atom (in fact the atom with the highest occupancy) by forwarding all uncaught method calls to one of the Atom objects (the selected Atom object) it contains.

Point mutations[point mutations]

A special case arises when disorder is due to a point mutation, i.e. when two or more point mutants of a polypeptide are present in the crystal. An example of this can be found in PDB structure 1EN2.

Since these residues belong to a different residue type (e.g. let's say Ser 60 and Cys 60) they should not be stored in a single `Residue` object as in the common case. In this case, each residue is represented by one `Residue` object, and both `Residue` objects are stored in a single `DisorderedResidue` object (see Fig. [fig:smcra]).

The `DisorderedResidue` object forwards all uncaught methods to the selected `Residue` object (by default the last `Residue` object added), and thus behaves like an ordinary residue. Each `Residue` object in a `DisorderedResidue` object can be uniquely identified by its residue name. In the above example, residue Ser

60 would have id "SER" in the `DisorderedResidue` object, while residue Cys 60 would have id "CYS". The user can select the active `Residue` object in a `DisorderedResidue` object via this id.

Example: suppose that a chain has a point mutation at position 10, consisting of a Ser and a Cys residue. Make sure that residue 10 of this chain behaves as the Cys residue.

```
In [35]: residue = chain[10]
         residue.disordered_select('CYS')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-35-c375d43705d3> in <module>()
      1 residue = chain[10]
----> 2 residue.disordered_select('CYS')

AttributeError: 'Residue' object has no attribute 'disordered_select'
```

In addition, you can get a list of all `Atom` objects (ie. all `DisorderedAtom` objects are 'unpacked' to their individual `Atom` objects) using the `get_unpacked_list` method of a `(Disordered)Residue` object.

12.4 Hetero residues

12.4.1 Associated problems[hetero problems]

A common problem with hetero residues is that several hetero and non-hetero residues present in the same chain share the same sequence identifier (and insertion code). Therefore, to generate a unique id for each hetero residue, waters and other hetero residues are treated in a different way.

Remember that Residue object have the tuple (hetfield, resseq, icode) as id. The hetfield is blank (" ") for amino and nucleic acids, and a string for waters and other hetero residues. The content of the hetfield is explained below.

12.4.2 Water residues

The hetfield string of a water residue consists of the letter "W". So a typical residue id for a water is ("W", 1, " ").

12.4.3 Other hetero residues

The hetfield string for other hetero residues starts with "H_" followed by the residue name. A glucose molecule e.g. with residue name "GLC" would have hetfield "H_GLC". Its residue id could e.g. be ("H_GLC", 1, " ").

12.5 Navigating through a Structure object

```
In [39]: from Bio.PDB.PDBParser import PDBParser
         parser = PDBParser()
         structure = parser.get_structure("test", "data/pdb1fat.ent")
         model = structure[0]
         chain = model["A"]
         residue = chain[1]
         atom = residue["CA"]
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstruct
```

```
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
```

```
In [40]: p = PDBParser()
         structure = p.get_structure('X', 'data/pdb1fat.ent')
         for model in structure:
             for chain in model:
                 for residue in chain:
                     for atom in residue:
                         print(atom)
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
```

```
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
```

```
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
```

```
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
```

```
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
```

```
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
```

```
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
```

```
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
```

```
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
```

```
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
```

```
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
```

```
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
```

```
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
```

```
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
```

```
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
```

```
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
```

```
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
```

```
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
```

```
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/StructureBuilder.py:87: PDBConstructi
  PDBConstructionWarning)
```

There is a shortcut if you want to iterate over all atoms in a structure:

```
In [41]: atoms = structure.get_atoms()
         for atom in atoms:
             print(atom)
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
```

```
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
```

```
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
```

```
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
```

```
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
```

```
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
```

```
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
```

```
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
```

```
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
```

```
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
```

```
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
```

```
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
```

```
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
```

```
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
```

```
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
```

```
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
```

```
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
```

```
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
```

```
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
```

```
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
```

Similarly, to iterate over all atoms in a chain, use

```
In [42]: atoms = chain.get_atoms()
         for atom in atoms:
             print(atom)
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
```

```
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
```

```
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
```

```
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
```

```
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
```

```
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom ND1>
<Atom CD2>
<Atom CE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
```

```
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom NE>
<Atom CZ>
<Atom NH1>
<Atom NH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
```

```
<Atom OH>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom NE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
```

```
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
```

```
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom CD1>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom OE1>
<Atom OE2>
<Atom N>
<Atom CA>
<Atom C>
```

```
<Atom O>
<Atom CB>
<Atom OG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom ND2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom OD1>
<Atom OD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG1>
<Atom CG2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom NE1>
<Atom CE2>
<Atom CE3>
<Atom CZ2>
<Atom CZ3>
<Atom CH2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
```

```
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom CE1>
<Atom CE2>
<Atom CZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD>
<Atom CE>
<Atom NZ>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom CG>
<Atom CD1>
<Atom CD2>
<Atom N>
<Atom CA>
<Atom C>
<Atom O>
<Atom CB>
<Atom OG>
<Atom C1>
<Atom C2>
<Atom C3>
<Atom C4>
<Atom C5>
<Atom C6>
<Atom C7>
<Atom C8>
<Atom N2>
<Atom O3>
<Atom O4>
<Atom O5>
```

```
<Atom O6>
<Atom O7>
<Atom MN>
<Atom CA>
<Atom O>
<Atom O>
<Atom O>
<Atom O>
```

or if you want to iterate over all residues in a model:

```
In [43]: residues = model.get_residues()
         for residue in residues:
             print(residue)
```

```
<Residue SER het=  resseq=1 icode= >
<Residue ASN het=  resseq=2 icode= >
<Residue ASP het=  resseq=3 icode= >
<Residue ILE het=  resseq=4 icode= >
<Residue TYR het=  resseq=5 icode= >
<Residue PHE het=  resseq=6 icode= >
<Residue ASN het=  resseq=7 icode= >
<Residue PHE het=  resseq=8 icode= >
<Residue GLN het=  resseq=9 icode= >
<Residue ARG het=  resseq=10 icode= >
<Residue PHE het=  resseq=11 icode= >
<Residue ASN het=  resseq=12 icode= >
<Residue GLU het=  resseq=13 icode= >
<Residue THR het=  resseq=14 icode= >
<Residue ASN het=  resseq=15 icode= >
<Residue LEU het=  resseq=16 icode= >
<Residue ILE het=  resseq=17 icode= >
<Residue LEU het=  resseq=18 icode= >
<Residue GLN het=  resseq=19 icode= >
<Residue ARG het=  resseq=20 icode= >
<Residue ASP het=  resseq=21 icode= >
<Residue ALA het=  resseq=22 icode= >
<Residue SER het=  resseq=23 icode= >
<Residue VAL het=  resseq=24 icode= >
<Residue SER het=  resseq=25 icode= >
<Residue SER het=  resseq=26 icode= >
<Residue SER het=  resseq=27 icode= >
<Residue GLY het=  resseq=28 icode= >
<Residue GLN het=  resseq=29 icode= >
<Residue LEU het=  resseq=30 icode= >
<Residue ARG het=  resseq=31 icode= >
<Residue LEU het=  resseq=32 icode= >
<Residue THR het=  resseq=33 icode= >
<Residue ASN het=  resseq=34 icode= >
<Residue LEU het=  resseq=35 icode= >
<Residue ASN het=  resseq=36 icode= >
<Residue ASN het=  resseq=38 icode= >
<Residue GLY het=  resseq=39 icode= >
<Residue GLU het=  resseq=40 icode= >
<Residue PRO het=  resseq=41 icode= >
<Residue ARG het=  resseq=42 icode= >
<Residue VAL het=  resseq=43 icode= >
<Residue GLY het=  resseq=44 icode= >
<Residue SER het=  resseq=45 icode= >
<Residue LEU het=  resseq=46 icode= >
```

```
<Residue GLY het=  resseq=47 icode= >
<Residue ARG het=  resseq=48 icode= >
<Residue ALA het=  resseq=49 icode= >
<Residue PHE het=  resseq=50 icode= >
<Residue TYR het=  resseq=51 icode= >
<Residue SER het=  resseq=52 icode= >
<Residue ALA het=  resseq=53 icode= >
<Residue PRO het=  resseq=54 icode= >
<Residue ILE het=  resseq=55 icode= >
<Residue GLN het=  resseq=56 icode= >
<Residue ILE het=  resseq=57 icode= >
<Residue TRP het=  resseq=58 icode= >
<Residue ASP het=  resseq=59 icode= >
<Residue ASN het=  resseq=60 icode= >
<Residue THR het=  resseq=61 icode= >
<Residue THR het=  resseq=62 icode= >
<Residue GLY het=  resseq=63 icode= >
<Residue THR het=  resseq=64 icode= >
<Residue VAL het=  resseq=65 icode= >
<Residue ALA het=  resseq=66 icode= >
<Residue SER het=  resseq=67 icode= >
<Residue PHE het=  resseq=68 icode= >
<Residue ALA het=  resseq=69 icode= >
<Residue THR het=  resseq=70 icode= >
<Residue SER het=  resseq=71 icode= >
<Residue PHE het=  resseq=72 icode= >
<Residue THR het=  resseq=73 icode= >
<Residue PHE het=  resseq=74 icode= >
<Residue ASN het=  resseq=75 icode= >
<Residue ILE het=  resseq=76 icode= >
<Residue GLN het=  resseq=77 icode= >
<Residue VAL het=  resseq=78 icode= >
<Residue PRO het=  resseq=79 icode= >
<Residue ASN het=  resseq=80 icode= >
<Residue ASN het=  resseq=81 icode= >
<Residue ALA het=  resseq=82 icode= >
<Residue GLY het=  resseq=83 icode= >
<Residue PRO het=  resseq=84 icode= >
<Residue ALA het=  resseq=85 icode= >
<Residue ASP het=  resseq=86 icode= >
<Residue GLY het=  resseq=87 icode= >
<Residue LEU het=  resseq=88 icode= >
<Residue ALA het=  resseq=89 icode= >
<Residue PHE het=  resseq=90 icode= >
<Residue ALA het=  resseq=91 icode= >
<Residue LEU het=  resseq=92 icode= >
<Residue VAL het=  resseq=93 icode= >
<Residue PRO het=  resseq=94 icode= >
<Residue VAL het=  resseq=95 icode= >
<Residue GLY het=  resseq=96 icode= >
<Residue SER het=  resseq=97 icode= >
<Residue GLN het=  resseq=98 icode= >
<Residue PRO het=  resseq=99 icode= >
<Residue LYS het=  resseq=100 icode= >
<Residue ASP het=  resseq=101 icode= >
<Residue LYS het=  resseq=102 icode= >
<Residue GLY het=  resseq=103 icode= >
<Residue GLY het=  resseq=104 icode= >
<Residue PHE het=  resseq=105 icode= >
```

```
<Residue LEU het=  resseq=106 icode= >
<Residue GLY het=  resseq=107 icode= >
<Residue LEU het=  resseq=108 icode= >
<Residue PHE het=  resseq=109 icode= >
<Residue ASP het=  resseq=110 icode= >
<Residue GLY het=  resseq=111 icode= >
<Residue SER het=  resseq=112 icode= >
<Residue ASN het=  resseq=113 icode= >
<Residue SER het=  resseq=114 icode= >
<Residue ASN het=  resseq=115 icode= >
<Residue PHE het=  resseq=116 icode= >
<Residue HIS het=  resseq=117 icode= >
<Residue THR het=  resseq=118 icode= >
<Residue VAL het=  resseq=119 icode= >
<Residue ALA het=  resseq=120 icode= >
<Residue VAL het=  resseq=121 icode= >
<Residue GLU het=  resseq=122 icode= >
<Residue PHE het=  resseq=123 icode= >
<Residue ASP het=  resseq=124 icode= >
<Residue THR het=  resseq=125 icode= >
<Residue LEU het=  resseq=126 icode= >
<Residue TYR het=  resseq=127 icode= >
<Residue ASN het=  resseq=128 icode= >
<Residue LYS het=  resseq=129 icode= >
<Residue ASP het=  resseq=130 icode= >
<Residue TRP het=  resseq=131 icode= >
<Residue ASP het=  resseq=132 icode= >
<Residue PRO het=  resseq=133 icode= >
<Residue THR het=  resseq=134 icode= >
<Residue GLU het=  resseq=135 icode= >
<Residue ARG het=  resseq=136 icode= >
<Residue HIS het=  resseq=137 icode= >
<Residue ILE het=  resseq=138 icode= >
<Residue GLY het=  resseq=139 icode= >
<Residue ILE het=  resseq=140 icode= >
<Residue ASP het=  resseq=141 icode= >
<Residue VAL het=  resseq=142 icode= >
<Residue ASN het=  resseq=143 icode= >
<Residue SER het=  resseq=144 icode= >
<Residue ILE het=  resseq=145 icode= >
<Residue ARG het=  resseq=146 icode= >
<Residue SER het=  resseq=147 icode= >
<Residue ILE het=  resseq=148 icode= >
<Residue LYS het=  resseq=149 icode= >
<Residue THR het=  resseq=150 icode= >
<Residue THR het=  resseq=151 icode= >
<Residue ARG het=  resseq=152 icode= >
<Residue TRP het=  resseq=153 icode= >
<Residue ASP het=  resseq=154 icode= >
<Residue PHE het=  resseq=155 icode= >
<Residue VAL het=  resseq=156 icode= >
<Residue ASN het=  resseq=157 icode= >
<Residue GLY het=  resseq=158 icode= >
<Residue GLU het=  resseq=159 icode= >
<Residue ASN het=  resseq=160 icode= >
<Residue ALA het=  resseq=161 icode= >
<Residue GLU het=  resseq=162 icode= >
<Residue VAL het=  resseq=163 icode= >
<Residue LEU het=  resseq=164 icode= >
```

```
<Residue ILE het=  resseq=165 icode= >
<Residue THR het=  resseq=166 icode= >
<Residue TYR het=  resseq=167 icode= >
<Residue ASP het=  resseq=168 icode= >
<Residue SER het=  resseq=169 icode= >
<Residue SER het=  resseq=170 icode= >
<Residue THR het=  resseq=171 icode= >
<Residue ASN het=  resseq=172 icode= >
<Residue LEU het=  resseq=173 icode= >
<Residue LEU het=  resseq=174 icode= >
<Residue VAL het=  resseq=175 icode= >
<Residue ALA het=  resseq=176 icode= >
<Residue SER het=  resseq=177 icode= >
<Residue LEU het=  resseq=178 icode= >
<Residue VAL het=  resseq=179 icode= >
<Residue TYR het=  resseq=180 icode= >
<Residue PRO het=  resseq=181 icode= >
<Residue SER het=  resseq=182 icode= >
<Residue GLN het=  resseq=183 icode= >
<Residue LYS het=  resseq=184 icode= >
<Residue THR het=  resseq=185 icode= >
<Residue SER het=  resseq=186 icode= >
<Residue PHE het=  resseq=187 icode= >
<Residue ILE het=  resseq=188 icode= >
<Residue VAL het=  resseq=189 icode= >
<Residue SER het=  resseq=190 icode= >
<Residue ASP het=  resseq=191 icode= >
<Residue THR het=  resseq=192 icode= >
<Residue VAL het=  resseq=193 icode= >
<Residue ASP het=  resseq=194 icode= >
<Residue LEU het=  resseq=195 icode= >
<Residue LYS het=  resseq=196 icode= >
<Residue SER het=  resseq=197 icode= >
<Residue VAL het=  resseq=198 icode= >
<Residue LEU het=  resseq=199 icode= >
<Residue PRO het=  resseq=200 icode= >
<Residue GLU het=  resseq=201 icode= >
<Residue TRP het=  resseq=202 icode= >
<Residue VAL het=  resseq=203 icode= >
<Residue SER het=  resseq=204 icode= >
<Residue VAL het=  resseq=205 icode= >
<Residue GLY het=  resseq=206 icode= >
<Residue PHE het=  resseq=207 icode= >
<Residue SER het=  resseq=208 icode= >
<Residue ALA het=  resseq=209 icode= >
<Residue THR het=  resseq=210 icode= >
<Residue THR het=  resseq=211 icode= >
<Residue GLY het=  resseq=212 icode= >
<Residue ILE het=  resseq=213 icode= >
<Residue ASN het=  resseq=214 icode= >
<Residue LYS het=  resseq=215 icode= >
<Residue GLY het=  resseq=216 icode= >
<Residue ASN het=  resseq=217 icode= >
<Residue VAL het=  resseq=218 icode= >
<Residue GLU het=  resseq=219 icode= >
<Residue THR het=  resseq=220 icode= >
<Residue ASN het=  resseq=221 icode= >
<Residue ASP het=  resseq=222 icode= >
<Residue VAL het=  resseq=223 icode= >
```

```
<Residue LEU het=  resseq=224 icode= >
<Residue SER het=  resseq=225 icode= >
<Residue TRP het=  resseq=226 icode= >
<Residue SER het=  resseq=227 icode= >
<Residue PHE het=  resseq=228 icode= >
<Residue ALA het=  resseq=229 icode= >
<Residue SER het=  resseq=230 icode= >
<Residue LYS het=  resseq=231 icode= >
<Residue LEU het=  resseq=232 icode= >
<Residue SER het=  resseq=233 icode= >
<Residue NAG het=H_NAG resseq=253 icode= >
<Residue  MN het=H_ MN resseq=254 icode= >
<Residue  CA het=H_ CA resseq=255 icode= >
<Residue HOH het=W resseq=305 icode= >
<Residue HOH het=W resseq=306 icode= >
<Residue HOH het=W resseq=307 icode= >
<Residue HOH het=W resseq=308 icode= >
<Residue SER het=  resseq=1 icode= >
<Residue ASN het=  resseq=2 icode= >
<Residue ASP het=  resseq=3 icode= >
<Residue ILE het=  resseq=4 icode= >
<Residue TYR het=  resseq=5 icode= >
<Residue PHE het=  resseq=6 icode= >
<Residue ASN het=  resseq=7 icode= >
<Residue PHE het=  resseq=8 icode= >
<Residue GLN het=  resseq=9 icode= >
<Residue ARG het=  resseq=10 icode= >
<Residue PHE het=  resseq=11 icode= >
<Residue ASN het=  resseq=12 icode= >
<Residue GLU het=  resseq=13 icode= >
<Residue THR het=  resseq=14 icode= >
<Residue ASN het=  resseq=15 icode= >
<Residue LEU het=  resseq=16 icode= >
<Residue ILE het=  resseq=17 icode= >
<Residue LEU het=  resseq=18 icode= >
<Residue GLN het=  resseq=19 icode= >
<Residue ARG het=  resseq=20 icode= >
<Residue ASP het=  resseq=21 icode= >
<Residue ALA het=  resseq=22 icode= >
<Residue SER het=  resseq=23 icode= >
<Residue VAL het=  resseq=24 icode= >
<Residue SER het=  resseq=25 icode= >
<Residue SER het=  resseq=26 icode= >
<Residue SER het=  resseq=27 icode= >
<Residue GLY het=  resseq=28 icode= >
<Residue GLN het=  resseq=29 icode= >
<Residue LEU het=  resseq=30 icode= >
<Residue ARG het=  resseq=31 icode= >
<Residue LEU het=  resseq=32 icode= >
<Residue THR het=  resseq=33 icode= >
<Residue ASN het=  resseq=34 icode= >
<Residue LEU het=  resseq=35 icode= >
<Residue ASN het=  resseq=36 icode= >
<Residue GLY het=  resseq=37 icode= >
<Residue ASN het=  resseq=38 icode= >
<Residue GLY het=  resseq=39 icode= >
<Residue GLU het=  resseq=40 icode= >
<Residue PRO het=  resseq=41 icode= >
<Residue ARG het=  resseq=42 icode= >
```

```
<Residue VAL het=  resseq=43 icode= >
<Residue GLY het=  resseq=44 icode= >
<Residue SER het=  resseq=45 icode= >
<Residue LEU het=  resseq=46 icode= >
<Residue GLY het=  resseq=47 icode= >
<Residue ARG het=  resseq=48 icode= >
<Residue ALA het=  resseq=49 icode= >
<Residue PHE het=  resseq=50 icode= >
<Residue TYR het=  resseq=51 icode= >
<Residue SER het=  resseq=52 icode= >
<Residue ALA het=  resseq=53 icode= >
<Residue PRO het=  resseq=54 icode= >
<Residue ILE het=  resseq=55 icode= >
<Residue GLN het=  resseq=56 icode= >
<Residue ILE het=  resseq=57 icode= >
<Residue TRP het=  resseq=58 icode= >
<Residue ASP het=  resseq=59 icode= >
<Residue ASN het=  resseq=60 icode= >
<Residue THR het=  resseq=61 icode= >
<Residue THR het=  resseq=62 icode= >
<Residue GLY het=  resseq=63 icode= >
<Residue THR het=  resseq=64 icode= >
<Residue VAL het=  resseq=65 icode= >
<Residue ALA het=  resseq=66 icode= >
<Residue SER het=  resseq=67 icode= >
<Residue PHE het=  resseq=68 icode= >
<Residue ALA het=  resseq=69 icode= >
<Residue THR het=  resseq=70 icode= >
<Residue SER het=  resseq=71 icode= >
<Residue PHE het=  resseq=72 icode= >
<Residue THR het=  resseq=73 icode= >
<Residue PHE het=  resseq=74 icode= >
<Residue ASN het=  resseq=75 icode= >
<Residue ILE het=  resseq=76 icode= >
<Residue GLN het=  resseq=77 icode= >
<Residue VAL het=  resseq=78 icode= >
<Residue PRO het=  resseq=79 icode= >
<Residue ASN het=  resseq=80 icode= >
<Residue ASN het=  resseq=81 icode= >
<Residue ALA het=  resseq=82 icode= >
<Residue GLY het=  resseq=83 icode= >
<Residue PRO het=  resseq=84 icode= >
<Residue ALA het=  resseq=85 icode= >
<Residue ASP het=  resseq=86 icode= >
<Residue GLY het=  resseq=87 icode= >
<Residue LEU het=  resseq=88 icode= >
<Residue ALA het=  resseq=89 icode= >
<Residue PHE het=  resseq=90 icode= >
<Residue ALA het=  resseq=91 icode= >
<Residue LEU het=  resseq=92 icode= >
<Residue VAL het=  resseq=93 icode= >
<Residue PRO het=  resseq=94 icode= >
<Residue VAL het=  resseq=95 icode= >
<Residue GLY het=  resseq=96 icode= >
<Residue SER het=  resseq=97 icode= >
<Residue GLN het=  resseq=98 icode= >
<Residue PRO het=  resseq=99 icode= >
<Residue LYS het=  resseq=100 icode= >
<Residue ASP het=  resseq=101 icode= >
```

```
<Residue LYS het=  resseq=102 icode= >
<Residue GLY het=  resseq=103 icode= >
<Residue GLY het=  resseq=104 icode= >
<Residue PHE het=  resseq=105 icode= >
<Residue LEU het=  resseq=106 icode= >
<Residue GLY het=  resseq=107 icode= >
<Residue LEU het=  resseq=108 icode= >
<Residue PHE het=  resseq=109 icode= >
<Residue ASP het=  resseq=110 icode= >
<Residue GLY het=  resseq=111 icode= >
<Residue SER het=  resseq=112 icode= >
<Residue ASN het=  resseq=113 icode= >
<Residue SER het=  resseq=114 icode= >
<Residue ASN het=  resseq=115 icode= >
<Residue PHE het=  resseq=116 icode= >
<Residue HIS het=  resseq=117 icode= >
<Residue THR het=  resseq=118 icode= >
<Residue VAL het=  resseq=119 icode= >
<Residue ALA het=  resseq=120 icode= >
<Residue VAL het=  resseq=121 icode= >
<Residue GLU het=  resseq=122 icode= >
<Residue PHE het=  resseq=123 icode= >
<Residue ASP het=  resseq=124 icode= >
<Residue THR het=  resseq=125 icode= >
<Residue LEU het=  resseq=126 icode= >
<Residue TYR het=  resseq=127 icode= >
<Residue ASN het=  resseq=128 icode= >
<Residue LYS het=  resseq=129 icode= >
<Residue ASP het=  resseq=130 icode= >
<Residue TRP het=  resseq=131 icode= >
<Residue ASP het=  resseq=132 icode= >
<Residue PRO het=  resseq=133 icode= >
<Residue THR het=  resseq=134 icode= >
<Residue GLU het=  resseq=135 icode= >
<Residue ARG het=  resseq=136 icode= >
<Residue HIS het=  resseq=137 icode= >
<Residue ILE het=  resseq=138 icode= >
<Residue GLY het=  resseq=139 icode= >
<Residue ILE het=  resseq=140 icode= >
<Residue ASP het=  resseq=141 icode= >
<Residue VAL het=  resseq=142 icode= >
<Residue ASN het=  resseq=143 icode= >
<Residue SER het=  resseq=144 icode= >
<Residue ILE het=  resseq=145 icode= >
<Residue ARG het=  resseq=146 icode= >
<Residue SER het=  resseq=147 icode= >
<Residue ILE het=  resseq=148 icode= >
<Residue LYS het=  resseq=149 icode= >
<Residue THR het=  resseq=150 icode= >
<Residue THR het=  resseq=151 icode= >
<Residue ARG het=  resseq=152 icode= >
<Residue TRP het=  resseq=153 icode= >
<Residue ASP het=  resseq=154 icode= >
<Residue PHE het=  resseq=155 icode= >
<Residue VAL het=  resseq=156 icode= >
<Residue ASN het=  resseq=157 icode= >
<Residue GLY het=  resseq=158 icode= >
<Residue GLU het=  resseq=159 icode= >
<Residue ASN het=  resseq=160 icode= >
```

```
<Residue ALA het=  resseq=161 icode= >
<Residue GLU het=  resseq=162 icode= >
<Residue VAL het=  resseq=163 icode= >
<Residue LEU het=  resseq=164 icode= >
<Residue ILE het=  resseq=165 icode= >
<Residue THR het=  resseq=166 icode= >
<Residue TYR het=  resseq=167 icode= >
<Residue ASP het=  resseq=168 icode= >
<Residue SER het=  resseq=169 icode= >
<Residue SER het=  resseq=170 icode= >
<Residue THR het=  resseq=171 icode= >
<Residue ASN het=  resseq=172 icode= >
<Residue LEU het=  resseq=173 icode= >
<Residue LEU het=  resseq=174 icode= >
<Residue VAL het=  resseq=175 icode= >
<Residue ALA het=  resseq=176 icode= >
<Residue SER het=  resseq=177 icode= >
<Residue LEU het=  resseq=178 icode= >
<Residue VAL het=  resseq=179 icode= >
<Residue TYR het=  resseq=180 icode= >
<Residue PRO het=  resseq=181 icode= >
<Residue SER het=  resseq=182 icode= >
<Residue GLN het=  resseq=183 icode= >
<Residue LYS het=  resseq=184 icode= >
<Residue THR het=  resseq=185 icode= >
<Residue SER het=  resseq=186 icode= >
<Residue PHE het=  resseq=187 icode= >
<Residue ILE het=  resseq=188 icode= >
<Residue VAL het=  resseq=189 icode= >
<Residue SER het=  resseq=190 icode= >
<Residue ASP het=  resseq=191 icode= >
<Residue THR het=  resseq=192 icode= >
<Residue VAL het=  resseq=193 icode= >
<Residue ASP het=  resseq=194 icode= >
<Residue LEU het=  resseq=195 icode= >
<Residue LYS het=  resseq=196 icode= >
<Residue SER het=  resseq=197 icode= >
<Residue VAL het=  resseq=198 icode= >
<Residue LEU het=  resseq=199 icode= >
<Residue PRO het=  resseq=200 icode= >
<Residue GLU het=  resseq=201 icode= >
<Residue TRP het=  resseq=202 icode= >
<Residue VAL het=  resseq=203 icode= >
<Residue SER het=  resseq=204 icode= >
<Residue VAL het=  resseq=205 icode= >
<Residue GLY het=  resseq=206 icode= >
<Residue PHE het=  resseq=207 icode= >
<Residue SER het=  resseq=208 icode= >
<Residue ALA het=  resseq=209 icode= >
<Residue THR het=  resseq=210 icode= >
<Residue THR het=  resseq=211 icode= >
<Residue GLY het=  resseq=212 icode= >
<Residue ILE het=  resseq=213 icode= >
<Residue ASN het=  resseq=214 icode= >
<Residue LYS het=  resseq=215 icode= >
<Residue GLY het=  resseq=216 icode= >
<Residue ASN het=  resseq=217 icode= >
<Residue VAL het=  resseq=218 icode= >
<Residue GLU het=  resseq=219 icode= >
```

```
<Residue THR het=  resseq=220 icode= >
<Residue ASN het=  resseq=221 icode= >
<Residue ASP het=  resseq=222 icode= >
<Residue VAL het=  resseq=223 icode= >
<Residue LEU het=  resseq=224 icode= >
<Residue SER het=  resseq=225 icode= >
<Residue TRP het=  resseq=226 icode= >
<Residue SER het=  resseq=227 icode= >
<Residue PHE het=  resseq=228 icode= >
<Residue ALA het=  resseq=229 icode= >
<Residue SER het=  resseq=230 icode= >
<Residue LYS het=  resseq=231 icode= >
<Residue LEU het=  resseq=232 icode= >
<Residue SER het=  resseq=233 icode= >
<Residue NAG het=H_NAG resseq=253 icode= >
<Residue  MN het=H_ MN resseq=254 icode= >
<Residue  CA het=H_ CA resseq=255 icode= >
<Residue HOH het=W resseq=301 icode= >
<Residue HOH het=W resseq=302 icode= >
<Residue HOH het=W resseq=303 icode= >
<Residue HOH het=W resseq=304 icode= >
<Residue SER het=  resseq=1 icode= >
<Residue ASN het=  resseq=2 icode= >
<Residue ASP het=  resseq=3 icode= >
<Residue ILE het=  resseq=4 icode= >
<Residue TYR het=  resseq=5 icode= >
<Residue PHE het=  resseq=6 icode= >
<Residue ASN het=  resseq=7 icode= >
<Residue PHE het=  resseq=8 icode= >
<Residue GLN het=  resseq=9 icode= >
<Residue ARG het=  resseq=10 icode= >
<Residue PHE het=  resseq=11 icode= >
<Residue ASN het=  resseq=12 icode= >
<Residue GLU het=  resseq=13 icode= >
<Residue THR het=  resseq=14 icode= >
<Residue ASN het=  resseq=15 icode= >
<Residue LEU het=  resseq=16 icode= >
<Residue ILE het=  resseq=17 icode= >
<Residue LEU het=  resseq=18 icode= >
<Residue GLN het=  resseq=19 icode= >
<Residue ARG het=  resseq=20 icode= >
<Residue ASP het=  resseq=21 icode= >
<Residue ALA het=  resseq=22 icode= >
<Residue SER het=  resseq=23 icode= >
<Residue VAL het=  resseq=24 icode= >
<Residue SER het=  resseq=25 icode= >
<Residue SER het=  resseq=26 icode= >
<Residue SER het=  resseq=27 icode= >
<Residue GLY het=  resseq=28 icode= >
<Residue GLN het=  resseq=29 icode= >
<Residue LEU het=  resseq=30 icode= >
<Residue ARG het=  resseq=31 icode= >
<Residue LEU het=  resseq=32 icode= >
<Residue THR het=  resseq=33 icode= >
<Residue ASN het=  resseq=34 icode= >
<Residue LEU het=  resseq=35 icode= >
<Residue ASN het=  resseq=36 icode= >
<Residue ASN het=  resseq=38 icode= >
<Residue GLY het=  resseq=39 icode= >
```

```
<Residue GLU het=  resseq=40 icode= >
<Residue PRO het=  resseq=41 icode= >
<Residue ARG het=  resseq=42 icode= >
<Residue VAL het=  resseq=43 icode= >
<Residue GLY het=  resseq=44 icode= >
<Residue SER het=  resseq=45 icode= >
<Residue LEU het=  resseq=46 icode= >
<Residue GLY het=  resseq=47 icode= >
<Residue ARG het=  resseq=48 icode= >
<Residue ALA het=  resseq=49 icode= >
<Residue PHE het=  resseq=50 icode= >
<Residue TYR het=  resseq=51 icode= >
<Residue SER het=  resseq=52 icode= >
<Residue ALA het=  resseq=53 icode= >
<Residue PRO het=  resseq=54 icode= >
<Residue ILE het=  resseq=55 icode= >
<Residue GLN het=  resseq=56 icode= >
<Residue ILE het=  resseq=57 icode= >
<Residue TRP het=  resseq=58 icode= >
<Residue ASP het=  resseq=59 icode= >
<Residue ASN het=  resseq=60 icode= >
<Residue THR het=  resseq=61 icode= >
<Residue THR het=  resseq=62 icode= >
<Residue GLY het=  resseq=63 icode= >
<Residue THR het=  resseq=64 icode= >
<Residue VAL het=  resseq=65 icode= >
<Residue ALA het=  resseq=66 icode= >
<Residue SER het=  resseq=67 icode= >
<Residue PHE het=  resseq=68 icode= >
<Residue ALA het=  resseq=69 icode= >
<Residue THR het=  resseq=70 icode= >
<Residue SER het=  resseq=71 icode= >
<Residue PHE het=  resseq=72 icode= >
<Residue THR het=  resseq=73 icode= >
<Residue PHE het=  resseq=74 icode= >
<Residue ASN het=  resseq=75 icode= >
<Residue ILE het=  resseq=76 icode= >
<Residue GLN het=  resseq=77 icode= >
<Residue VAL het=  resseq=78 icode= >
<Residue PRO het=  resseq=79 icode= >
<Residue ASN het=  resseq=80 icode= >
<Residue ASN het=  resseq=81 icode= >
<Residue ALA het=  resseq=82 icode= >
<Residue GLY het=  resseq=83 icode= >
<Residue PRO het=  resseq=84 icode= >
<Residue ALA het=  resseq=85 icode= >
<Residue ASP het=  resseq=86 icode= >
<Residue GLY het=  resseq=87 icode= >
<Residue LEU het=  resseq=88 icode= >
<Residue ALA het=  resseq=89 icode= >
<Residue PHE het=  resseq=90 icode= >
<Residue ALA het=  resseq=91 icode= >
<Residue LEU het=  resseq=92 icode= >
<Residue VAL het=  resseq=93 icode= >
<Residue PRO het=  resseq=94 icode= >
<Residue VAL het=  resseq=95 icode= >
<Residue GLY het=  resseq=96 icode= >
<Residue SER het=  resseq=97 icode= >
<Residue GLN het=  resseq=98 icode= >
```

```
<Residue PRO het=  resseq=99 icode= >
<Residue LYS het=  resseq=100 icode= >
<Residue ASP het=  resseq=101 icode= >
<Residue LYS het=  resseq=102 icode= >
<Residue GLY het=  resseq=103 icode= >
<Residue GLY het=  resseq=104 icode= >
<Residue PHE het=  resseq=105 icode= >
<Residue LEU het=  resseq=106 icode= >
<Residue GLY het=  resseq=107 icode= >
<Residue LEU het=  resseq=108 icode= >
<Residue PHE het=  resseq=109 icode= >
<Residue ASP het=  resseq=110 icode= >
<Residue GLY het=  resseq=111 icode= >
<Residue SER het=  resseq=112 icode= >
<Residue ASN het=  resseq=113 icode= >
<Residue SER het=  resseq=114 icode= >
<Residue ASN het=  resseq=115 icode= >
<Residue PHE het=  resseq=116 icode= >
<Residue HIS het=  resseq=117 icode= >
<Residue THR het=  resseq=118 icode= >
<Residue VAL het=  resseq=119 icode= >
<Residue ALA het=  resseq=120 icode= >
<Residue VAL het=  resseq=121 icode= >
<Residue GLU het=  resseq=122 icode= >
<Residue PHE het=  resseq=123 icode= >
<Residue ASP het=  resseq=124 icode= >
<Residue THR het=  resseq=125 icode= >
<Residue LEU het=  resseq=126 icode= >
<Residue TYR het=  resseq=127 icode= >
<Residue ASN het=  resseq=128 icode= >
<Residue LYS het=  resseq=129 icode= >
<Residue ASP het=  resseq=130 icode= >
<Residue TRP het=  resseq=131 icode= >
<Residue ASP het=  resseq=132 icode= >
<Residue PRO het=  resseq=133 icode= >
<Residue THR het=  resseq=134 icode= >
<Residue GLU het=  resseq=135 icode= >
<Residue ARG het=  resseq=136 icode= >
<Residue HIS het=  resseq=137 icode= >
<Residue ILE het=  resseq=138 icode= >
<Residue GLY het=  resseq=139 icode= >
<Residue ILE het=  resseq=140 icode= >
<Residue ASP het=  resseq=141 icode= >
<Residue VAL het=  resseq=142 icode= >
<Residue ASN het=  resseq=143 icode= >
<Residue SER het=  resseq=144 icode= >
<Residue ILE het=  resseq=145 icode= >
<Residue ARG het=  resseq=146 icode= >
<Residue SER het=  resseq=147 icode= >
<Residue ILE het=  resseq=148 icode= >
<Residue LYS het=  resseq=149 icode= >
<Residue THR het=  resseq=150 icode= >
<Residue THR het=  resseq=151 icode= >
<Residue ARG het=  resseq=152 icode= >
<Residue TRP het=  resseq=153 icode= >
<Residue ASP het=  resseq=154 icode= >
<Residue PHE het=  resseq=155 icode= >
<Residue VAL het=  resseq=156 icode= >
<Residue ASN het=  resseq=157 icode= >
```

```
<Residue GLY het=  resseq=158 icode= >
<Residue GLU het=  resseq=159 icode= >
<Residue ASN het=  resseq=160 icode= >
<Residue ALA het=  resseq=161 icode= >
<Residue GLU het=  resseq=162 icode= >
<Residue VAL het=  resseq=163 icode= >
<Residue LEU het=  resseq=164 icode= >
<Residue ILE het=  resseq=165 icode= >
<Residue THR het=  resseq=166 icode= >
<Residue TYR het=  resseq=167 icode= >
<Residue ASP het=  resseq=168 icode= >
<Residue SER het=  resseq=169 icode= >
<Residue SER het=  resseq=170 icode= >
<Residue THR het=  resseq=171 icode= >
<Residue ASN het=  resseq=172 icode= >
<Residue LEU het=  resseq=173 icode= >
<Residue LEU het=  resseq=174 icode= >
<Residue VAL het=  resseq=175 icode= >
<Residue ALA het=  resseq=176 icode= >
<Residue SER het=  resseq=177 icode= >
<Residue LEU het=  resseq=178 icode= >
<Residue VAL het=  resseq=179 icode= >
<Residue TYR het=  resseq=180 icode= >
<Residue PRO het=  resseq=181 icode= >
<Residue SER het=  resseq=182 icode= >
<Residue GLN het=  resseq=183 icode= >
<Residue LYS het=  resseq=184 icode= >
<Residue THR het=  resseq=185 icode= >
<Residue SER het=  resseq=186 icode= >
<Residue PHE het=  resseq=187 icode= >
<Residue ILE het=  resseq=188 icode= >
<Residue VAL het=  resseq=189 icode= >
<Residue SER het=  resseq=190 icode= >
<Residue ASP het=  resseq=191 icode= >
<Residue THR het=  resseq=192 icode= >
<Residue VAL het=  resseq=193 icode= >
<Residue ASP het=  resseq=194 icode= >
<Residue LEU het=  resseq=195 icode= >
<Residue LYS het=  resseq=196 icode= >
<Residue SER het=  resseq=197 icode= >
<Residue VAL het=  resseq=198 icode= >
<Residue LEU het=  resseq=199 icode= >
<Residue PRO het=  resseq=200 icode= >
<Residue GLU het=  resseq=201 icode= >
<Residue TRP het=  resseq=202 icode= >
<Residue VAL het=  resseq=203 icode= >
<Residue SER het=  resseq=204 icode= >
<Residue VAL het=  resseq=205 icode= >
<Residue GLY het=  resseq=206 icode= >
<Residue PHE het=  resseq=207 icode= >
<Residue SER het=  resseq=208 icode= >
<Residue ALA het=  resseq=209 icode= >
<Residue THR het=  resseq=210 icode= >
<Residue THR het=  resseq=211 icode= >
<Residue GLY het=  resseq=212 icode= >
<Residue ILE het=  resseq=213 icode= >
<Residue ASN het=  resseq=214 icode= >
<Residue LYS het=  resseq=215 icode= >
<Residue GLY het=  resseq=216 icode= >
```

```
<Residue ASN het=  resseq=217 icode= >
<Residue VAL het=  resseq=218 icode= >
<Residue GLU het=  resseq=219 icode= >
<Residue THR het=  resseq=220 icode= >
<Residue ASN het=  resseq=221 icode= >
<Residue ASP het=  resseq=222 icode= >
<Residue VAL het=  resseq=223 icode= >
<Residue LEU het=  resseq=224 icode= >
<Residue SER het=  resseq=225 icode= >
<Residue TRP het=  resseq=226 icode= >
<Residue SER het=  resseq=227 icode= >
<Residue PHE het=  resseq=228 icode= >
<Residue ALA het=  resseq=229 icode= >
<Residue SER het=  resseq=230 icode= >
<Residue LYS het=  resseq=231 icode= >
<Residue LEU het=  resseq=232 icode= >
<Residue SER het=  resseq=233 icode= >
<Residue NAG het=H_NAG resseq=253 icode= >
<Residue  MN het=H_ MN resseq=254 icode= >
<Residue  CA het=H_ CA resseq=255 icode= >
<Residue HOH het=W resseq=313 icode= >
<Residue HOH het=W resseq=314 icode= >
<Residue HOH het=W resseq=315 icode= >
<Residue HOH het=W resseq=316 icode= >
<Residue SER het=  resseq=1 icode= >
<Residue ASN het=  resseq=2 icode= >
<Residue ASP het=  resseq=3 icode= >
<Residue ILE het=  resseq=4 icode= >
<Residue TYR het=  resseq=5 icode= >
<Residue PHE het=  resseq=6 icode= >
<Residue ASN het=  resseq=7 icode= >
<Residue PHE het=  resseq=8 icode= >
<Residue GLN het=  resseq=9 icode= >
<Residue ARG het=  resseq=10 icode= >
<Residue PHE het=  resseq=11 icode= >
<Residue ASN het=  resseq=12 icode= >
<Residue GLU het=  resseq=13 icode= >
<Residue THR het=  resseq=14 icode= >
<Residue ASN het=  resseq=15 icode= >
<Residue LEU het=  resseq=16 icode= >
<Residue ILE het=  resseq=17 icode= >
<Residue LEU het=  resseq=18 icode= >
<Residue GLN het=  resseq=19 icode= >
<Residue ARG het=  resseq=20 icode= >
<Residue ASP het=  resseq=21 icode= >
<Residue ALA het=  resseq=22 icode= >
<Residue SER het=  resseq=23 icode= >
<Residue VAL het=  resseq=24 icode= >
<Residue SER het=  resseq=25 icode= >
<Residue SER het=  resseq=26 icode= >
<Residue SER het=  resseq=27 icode= >
<Residue GLY het=  resseq=28 icode= >
<Residue GLN het=  resseq=29 icode= >
<Residue LEU het=  resseq=30 icode= >
<Residue ARG het=  resseq=31 icode= >
<Residue LEU het=  resseq=32 icode= >
<Residue THR het=  resseq=33 icode= >
<Residue ASN het=  resseq=34 icode= >
<Residue LEU het=  resseq=35 icode= >
```

```
<Residue ASN het=  resseq=38 icode= >
<Residue GLY het=  resseq=39 icode= >
<Residue GLU het=  resseq=40 icode= >
<Residue PRO het=  resseq=41 icode= >
<Residue ARG het=  resseq=42 icode= >
<Residue VAL het=  resseq=43 icode= >
<Residue GLY het=  resseq=44 icode= >
<Residue SER het=  resseq=45 icode= >
<Residue LEU het=  resseq=46 icode= >
<Residue GLY het=  resseq=47 icode= >
<Residue ARG het=  resseq=48 icode= >
<Residue ALA het=  resseq=49 icode= >
<Residue PHE het=  resseq=50 icode= >
<Residue TYR het=  resseq=51 icode= >
<Residue SER het=  resseq=52 icode= >
<Residue ALA het=  resseq=53 icode= >
<Residue PRO het=  resseq=54 icode= >
<Residue ILE het=  resseq=55 icode= >
<Residue GLN het=  resseq=56 icode= >
<Residue ILE het=  resseq=57 icode= >
<Residue TRP het=  resseq=58 icode= >
<Residue ASP het=  resseq=59 icode= >
<Residue ASN het=  resseq=60 icode= >
<Residue THR het=  resseq=61 icode= >
<Residue THR het=  resseq=62 icode= >
<Residue GLY het=  resseq=63 icode= >
<Residue THR het=  resseq=64 icode= >
<Residue VAL het=  resseq=65 icode= >
<Residue ALA het=  resseq=66 icode= >
<Residue SER het=  resseq=67 icode= >
<Residue PHE het=  resseq=68 icode= >
<Residue ALA het=  resseq=69 icode= >
<Residue THR het=  resseq=70 icode= >
<Residue SER het=  resseq=71 icode= >
<Residue PHE het=  resseq=72 icode= >
<Residue THR het=  resseq=73 icode= >
<Residue PHE het=  resseq=74 icode= >
<Residue ASN het=  resseq=75 icode= >
<Residue ILE het=  resseq=76 icode= >
<Residue GLN het=  resseq=77 icode= >
<Residue VAL het=  resseq=78 icode= >
<Residue PRO het=  resseq=79 icode= >
<Residue ASN het=  resseq=80 icode= >
<Residue ASN het=  resseq=81 icode= >
<Residue ALA het=  resseq=82 icode= >
<Residue GLY het=  resseq=83 icode= >
<Residue PRO het=  resseq=84 icode= >
<Residue ALA het=  resseq=85 icode= >
<Residue ASP het=  resseq=86 icode= >
<Residue GLY het=  resseq=87 icode= >
<Residue LEU het=  resseq=88 icode= >
<Residue ALA het=  resseq=89 icode= >
<Residue PHE het=  resseq=90 icode= >
<Residue ALA het=  resseq=91 icode= >
<Residue LEU het=  resseq=92 icode= >
<Residue VAL het=  resseq=93 icode= >
<Residue PRO het=  resseq=94 icode= >
<Residue VAL het=  resseq=95 icode= >
<Residue GLY het=  resseq=96 icode= >
```

```
<Residue SER het=  resseq=97 icode= >
<Residue GLN het=  resseq=98 icode= >
<Residue PRO het=  resseq=99 icode= >
<Residue LYS het=  resseq=100 icode= >
<Residue ASP het=  resseq=101 icode= >
<Residue LYS het=  resseq=102 icode= >
<Residue GLY het=  resseq=103 icode= >
<Residue GLY het=  resseq=104 icode= >
<Residue PHE het=  resseq=105 icode= >
<Residue LEU het=  resseq=106 icode= >
<Residue GLY het=  resseq=107 icode= >
<Residue LEU het=  resseq=108 icode= >
<Residue PHE het=  resseq=109 icode= >
<Residue ASP het=  resseq=110 icode= >
<Residue GLY het=  resseq=111 icode= >
<Residue SER het=  resseq=112 icode= >
<Residue ASN het=  resseq=113 icode= >
<Residue SER het=  resseq=114 icode= >
<Residue ASN het=  resseq=115 icode= >
<Residue PHE het=  resseq=116 icode= >
<Residue HIS het=  resseq=117 icode= >
<Residue THR het=  resseq=118 icode= >
<Residue VAL het=  resseq=119 icode= >
<Residue ALA het=  resseq=120 icode= >
<Residue VAL het=  resseq=121 icode= >
<Residue GLU het=  resseq=122 icode= >
<Residue PHE het=  resseq=123 icode= >
<Residue ASP het=  resseq=124 icode= >
<Residue THR het=  resseq=125 icode= >
<Residue LEU het=  resseq=126 icode= >
<Residue TYR het=  resseq=127 icode= >
<Residue ASN het=  resseq=128 icode= >
<Residue LYS het=  resseq=129 icode= >
<Residue ASP het=  resseq=130 icode= >
<Residue TRP het=  resseq=131 icode= >
<Residue ASP het=  resseq=132 icode= >
<Residue PRO het=  resseq=133 icode= >
<Residue THR het=  resseq=134 icode= >
<Residue GLU het=  resseq=135 icode= >
<Residue ARG het=  resseq=136 icode= >
<Residue HIS het=  resseq=137 icode= >
<Residue ILE het=  resseq=138 icode= >
<Residue GLY het=  resseq=139 icode= >
<Residue ILE het=  resseq=140 icode= >
<Residue ASP het=  resseq=141 icode= >
<Residue VAL het=  resseq=142 icode= >
<Residue ASN het=  resseq=143 icode= >
<Residue SER het=  resseq=144 icode= >
<Residue ILE het=  resseq=145 icode= >
<Residue ARG het=  resseq=146 icode= >
<Residue SER het=  resseq=147 icode= >
<Residue ILE het=  resseq=148 icode= >
<Residue LYS het=  resseq=149 icode= >
<Residue THR het=  resseq=150 icode= >
<Residue THR het=  resseq=151 icode= >
<Residue ARG het=  resseq=152 icode= >
<Residue TRP het=  resseq=153 icode= >
<Residue ASP het=  resseq=154 icode= >
<Residue PHE het=  resseq=155 icode= >
```

```
<Residue VAL het=  resseq=156 icode= >
<Residue ASN het=  resseq=157 icode= >
<Residue GLY het=  resseq=158 icode= >
<Residue GLU het=  resseq=159 icode= >
<Residue ASN het=  resseq=160 icode= >
<Residue ALA het=  resseq=161 icode= >
<Residue GLU het=  resseq=162 icode= >
<Residue VAL het=  resseq=163 icode= >
<Residue LEU het=  resseq=164 icode= >
<Residue ILE het=  resseq=165 icode= >
<Residue THR het=  resseq=166 icode= >
<Residue TYR het=  resseq=167 icode= >
<Residue ASP het=  resseq=168 icode= >
<Residue SER het=  resseq=169 icode= >
<Residue SER het=  resseq=170 icode= >
<Residue THR het=  resseq=171 icode= >
<Residue ASN het=  resseq=172 icode= >
<Residue LEU het=  resseq=173 icode= >
<Residue LEU het=  resseq=174 icode= >
<Residue VAL het=  resseq=175 icode= >
<Residue ALA het=  resseq=176 icode= >
<Residue SER het=  resseq=177 icode= >
<Residue LEU het=  resseq=178 icode= >
<Residue VAL het=  resseq=179 icode= >
<Residue TYR het=  resseq=180 icode= >
<Residue PRO het=  resseq=181 icode= >
<Residue SER het=  resseq=182 icode= >
<Residue GLN het=  resseq=183 icode= >
<Residue LYS het=  resseq=184 icode= >
<Residue THR het=  resseq=185 icode= >
<Residue SER het=  resseq=186 icode= >
<Residue PHE het=  resseq=187 icode= >
<Residue ILE het=  resseq=188 icode= >
<Residue VAL het=  resseq=189 icode= >
<Residue SER het=  resseq=190 icode= >
<Residue ASP het=  resseq=191 icode= >
<Residue THR het=  resseq=192 icode= >
<Residue VAL het=  resseq=193 icode= >
<Residue ASP het=  resseq=194 icode= >
<Residue LEU het=  resseq=195 icode= >
<Residue LYS het=  resseq=196 icode= >
<Residue SER het=  resseq=197 icode= >
<Residue VAL het=  resseq=198 icode= >
<Residue LEU het=  resseq=199 icode= >
<Residue PRO het=  resseq=200 icode= >
<Residue GLU het=  resseq=201 icode= >
<Residue TRP het=  resseq=202 icode= >
<Residue VAL het=  resseq=203 icode= >
<Residue SER het=  resseq=204 icode= >
<Residue VAL het=  resseq=205 icode= >
<Residue GLY het=  resseq=206 icode= >
<Residue PHE het=  resseq=207 icode= >
<Residue SER het=  resseq=208 icode= >
<Residue ALA het=  resseq=209 icode= >
<Residue THR het=  resseq=210 icode= >
<Residue THR het=  resseq=211 icode= >
<Residue GLY het=  resseq=212 icode= >
<Residue ILE het=  resseq=213 icode= >
<Residue ASN het=  resseq=214 icode= >
```

```
<Residue LYS het=  resseq=215 icode= >
<Residue GLY het=  resseq=216 icode= >
<Residue ASN het=  resseq=217 icode= >
<Residue VAL het=  resseq=218 icode= >
<Residue GLU het=  resseq=219 icode= >
<Residue THR het=  resseq=220 icode= >
<Residue ASN het=  resseq=221 icode= >
<Residue ASP het=  resseq=222 icode= >
<Residue VAL het=  resseq=223 icode= >
<Residue LEU het=  resseq=224 icode= >
<Residue SER het=  resseq=225 icode= >
<Residue TRP het=  resseq=226 icode= >
<Residue SER het=  resseq=227 icode= >
<Residue PHE het=  resseq=228 icode= >
<Residue ALA het=  resseq=229 icode= >
<Residue SER het=  resseq=230 icode= >
<Residue LYS het=  resseq=231 icode= >
<Residue LEU het=  resseq=232 icode= >
<Residue SER het=  resseq=233 icode= >
<Residue NAG het=H_NAG resseq=253 icode= >
<Residue  MN het=H_ MN resseq=254 icode= >
<Residue  CA het=H_ CA resseq=255 icode= >
<Residue HOH het=W resseq=309 icode= >
<Residue HOH het=W resseq=310 icode= >
<Residue HOH het=W resseq=311 icode= >
<Residue HOH het=W resseq=312 icode= >
```

You can also use the `Selection.unfold_entities` function to get all residues from a structure:

```
In [45]: from Bio.PDB import Selection
         res_list = Selection.unfold_entities(structure, 'R')
```

or to get all atoms from a chain:

```
In [46]: atom_list = Selection.unfold_entities(chain, 'A')
```

Obviously, `A=atom, R=residue, C=chain, M=model, S=structure`. You can use this to go up in the hierarchy, e.g. to get a list of (unique) `Residue` or `Chain` parents from a list of `Atoms`:

```
In [47]: residue_list = Selection.unfold_entities(atom_list, 'R')
         chain_list = Selection.unfold_entities(atom_list, 'C')
```

For more info, see the API documentation.

```
In [50]: residue_id = ("H_GLC", 10, " ")
         residue = chain[residue_id]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-50-47dfa9146b6b> in <module>()
      1 residue_id = ("H_GLC", 10, " ")
----> 2 residue = chain[residue_id]

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/Chain.py in __getitem__(self, id)
     68         """
     69         id = self._translate_id(id)
---> 70         return Entity.__getitem__(self, id)
     71 
     72     def __contains__(self, id):

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/Entity.py in __getitem__(self, id)
     36     def __getitem__(self, id):
```

```
     37         "Return the child with given id."
---> 38         return self.child_dict[id]
     39 
     40     def __delitem__(self, id):

KeyError: ('H_GLC', 10, ' ')
```

```
In [51]: for residue in chain.get_list():
             residue_id = residue.get_id()
             hetfield = residue_id[0]
             if hetfield[0]=="H":
                 print(residue_id)
```

```
('H_NAG', 253, ' ')
('H_ MN', 254, ' ')
('H_ CA', 255, ' ')
```

```
In [52]: for model in structure.get_list():
             for chain in model.get_list():
                 for residue in chain.get_list():
                     if residue.has_id("CA"):
                         ca = residue["CA"]
                     if ca.get_bfactor() > 50.0:
                         print(ca.get_coord())
```

```
[ 37.08000183 -30.40399933  47.35200119]
[ 36.82699966 -34.76300049  44.60400009]
[ 35.19300079 -31.86599922  42.84199905]
[ 32.28699875 -31.66300011  45.20399857]
[ 22.45100021  16.85000038  45.15999985]
[ 20.17900085  16.3920002   42.16999817]
[ 17.9090004   13.9630003   43.97000122]
[ 24.11400032 -36.94100189  43.60400009]
[ 22.43400002 -37.85499954  46.84600067]
[ 23.10899925 -10.96800041  59.67699814]
[ 19.83499908 -11.48900032  61.50299835]
[ 21.12599945  -8.67500019  63.69900131]
[ 19.63299942  -5.29899979  63.30799866]
[ 22.27599907  -3.39400005  61.48799896]
[ 21.125       -1.45500004  58.4090004 ]
[  9.02000046 -23.44199944  48.05500031]
[ 26.77499962 -32.12200165  55.05599976]
[ 19.93099976  11.31599998  35.06800079]
[ 19.93099976  11.31599998  35.06800079]
[ 19.93099976  11.31599998  35.06800079]
[ 45.60100174  30.21899986  45.20999908]
[ 45.82899857  33.47800064  43.29399872]
[ 43.58800125  31.43799973  40.97499847]
[ 45.64300156 -16.40500069  24.57699966]
[ 43.34000015 -15.44699955  21.70999908]
[ 51.13499832  37.78300095  30.4640007 ]
[ 64.04499817  10.45400047  28.92099953]
[ 65.66000366   7.33699989  30.40200043]
[ 65.47299957   4.10500002  28.50699997]
[ 62.89400101   2.18499994  30.39999962]
[ 60.10699844   0.67199999  28.34600067]
[ 36.89599991  -9.72000027  20.24900055]
[ 36.89599991  -9.72000027  20.24900055]
[ 36.89599991  -9.72000027  20.24900055]
[-17.72599983   6.13700008  16.51300049]
[-19.03300095   8.92700005  19.9109993 ]
```

```
[-15.58300018   8.59599972  21.48399925]
[-16.37599945   5.02199984  22.30900002]
[ 24.58099937 -17.54800034   5.62300014]
[ 21.87999916 -19.56699944   7.3499999 ]
[-18.51499939   1.29100001  31.17300034]
[-20.9810009   -1.54499996  30.89100075]
[ -7.83500004 -16.55999947   8.75599957]
[ -8.91399956 -19.92300034  10.04100037]
[ -8.25300026 -21.03499985   6.48799992]
[ -5.17000008 -23.0510006    5.83099985]
[ -2.93899989 -20.56900024   4.13700008]
[  0.57999998 -20.36300087   5.58199978]
[-22.0170002   -4.579       20.48500061]
[ 24.98600006 -12.17199993  12.50699997]
[ 24.98600006 -12.17199993  12.50699997]
[ 24.98600006 -12.17199993  12.50699997]
[ 33.56000137  -2.94300008 -16.57900047]
[ 34.2859993    0.74599999 -17.20999908]
[  9.2329998   21.38899994  20.3239994 ]
[ 11.71500015  23.47599983  18.3560009 ]
[ 43.62400055   3.8440001  -16.69400024]
[ 44.19100189   6.90600014 -18.87000084]
[ 20.48500061  22.73900032 -10.22299957]
[ 22.24099922  26.02400017 -10.59899998]
[ 18.68099976  27.40600014 -10.7510004 ]
[ 17.35700035  29.2310009   -7.77699995]
[ 15.05200005  26.73600006  -6.26200008]
[ 15.51399994  26.13299942  -2.54299998]
[ 41.99399948  21.29599953  -5.5619998 ]
[ 34.60699844  10.92800045 -22.17900085]
[ 15.42099953  15.41899967  21.87299919]
[ 15.42099953  15.41899967  21.87299919]
[ 15.42099953  15.41899967  21.87299919]
```

```
In [53]: for model in structure.get_list():
             for chain in model.get_list():
                 for residue in chain.get_list():
                     if residue.is_disordered():
                         resseq = residue.get_id()[1]
                         resname = residue.get_resname()
                         model_id = model.get_id()
                         chain_id = chain.get_id()
                         print(model_id, chain_id, resname, resseq)
```

This will make sure that the SMCRA data structure will behave as if only the atoms with altloc A are present.

```
In [54]: for model in structure.get_list():
             for chain in model.get_list():
                 for residue in chain.get_list():
                     if residue.is_disordered():
                         for atom in residue.get_list():
                             if atom.is_disordered() and atom.disordered_has_id("A"):
                                 atom.disordered_select("A")
```

To extract polypeptides from a structure, construct a list of `Polypeptide` objects from a `Structure` object using `PolypeptideBuilder` as follows:

```
In [55]: model_nr = 1
         polypeptide_list = build_peptides(structure, model_nr)
         for polypeptide in polypeptide_list:
             print(polypeptide)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-55-bab75e37e9d8> in <module>()
      1 model_nr = 1
----> 2 polypeptide_list = build_peptides(structure, model_nr)
      3 for polypeptide in polypeptide_list:
      4     print(polypeptide)

NameError: name 'build_peptides' is not defined
```

A Polypeptide object is simply a UserList of Residue objects, and is always created from a single Model (in this case model 1). You can use the resulting `Polypeptide` object to get the sequence as a `Seq` object or to get a list of Cα atoms as well. Polypeptides can be built using a C-N or a Cα-C:math:*alpha* distance criterion.

Example:

```
In [ ]: # Using C-N
        ppb = PPBuilder()
        for pp in ppb.build_peptides(structure):
            print(pp.get_sequence())
```

```
In [ ]: ppb = CaPPBuilder()
        for pp in ppb.build_peptides(structure):
            print(pp.get_sequence())
```

Note that in the above case only model 0 of the structure is considered by `PolypeptideBuilder`. However, it is possible to use `PolypeptideBuilder` to build `Polypeptide` objects from `Model` and `Chain` objects as well.

The first thing to do is to extract all polypeptides from the structure (as above). The sequence of each polypeptide can then easily be obtained from the `Polypeptide` objects. The sequence is represented as a Biopython `Seq` object, and its alphabet is defined by a `ProteinAlphabet` object.

Example:

```
In [56]: seq = polypeptide.get_sequence()
         print(seq)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-56-7b9db59a51fe> in <module>()
----> 1 seq = polypeptide.get_sequence()
      2 print(seq)

NameError: name 'polypeptide' is not defined
```

12.6 Analyzing structures

12.6.1 Measuring distances

The minus operator for atoms has been overloaded to return the distance between two atoms.

```
In [57]: # Get some atoms
         ca1 = residue1['CA']
         ca2 = residue2['CA']
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-2f7f3dc9b562> in <module>()
      1 # Get some atoms
----> 2 ca1 = residue1['CA']
```

```
      3 ca2 = residue2['CA']

NameError: name 'residue1' is not defined
```

```
In [ ]: distance = ca1-ca2
```

12.6.2 Measuring angles

Use the vector representation of the atomic coordinates, and the `calc_angle` function from the `Vector` module:

```
In [58]: vector1 = atom1.get_vector()
         vector2 = atom2.get_vector()
         vector3 = atom3.get_vector()
         angle = calc_angle(vector1, vector2, vector3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-c61cceff7731> in <module>()
----> 1 vector1 = atom1.get_vector()
      2 vector2 = atom2.get_vector()
      3 vector3 = atom3.get_vector()
      4 angle = calc_angle(vector1, vector2, vector3)

NameError: name 'atom1' is not defined
```

12.6.3 Measuring torsion angles

Use the vector representation of the atomic coordinates, and the `calc_dihedral` function from the `Vector` module:

```
In [59]: vector1 = atom1.get_vector()
         vector2 = atom2.get_vector()
         vector3 = atom3.get_vector()
         vector4 = atom4.get_vector()
         angle = calc_dihedral(vector1, vector2, vector3, vector4)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-59-03917accffd8> in <module>()
----> 1 vector1 = atom1.get_vector()
      2 vector2 = atom2.get_vector()
      3 vector3 = atom3.get_vector()
      4 vector4 = atom4.get_vector()
      5 angle = calc_dihedral(vector1, vector2, vector3, vector4)

NameError: name 'atom1' is not defined
```

12.6.4 Determining atom-atom contacts

Use `NeighborSearch` to perform neighbor lookup. The neighbor lookup is done using a KD tree module written in C (see `Bio.KDTree`), making it very fast. It also includes a fast method to find all point pairs within a certain distance of each other.

12.6.5 Superimposing two structures

Use a `Superimposer` object to superimpose two coordinate sets. This object calculates the rotation and translation matrix that rotates two lists of atoms on top of each other in such a way that their RMSD is minimized. Of course, the two lists need to contain the same number of atoms. The `Superimposer` object can also apply the rotation/translation to a list of atoms. The rotation and translation are stored as a tuple in the `rotran` attribute of the `Superimposer` object (note that the rotation is right multiplying!). The RMSD is stored in the `rmsd` attribute.

The algorithm used by `Superimposer` comes from @golub1989 [Golub & Van Loan] and makes use of singular value decomposition (this is implemented in the general `Bio.SVDSuperimposer` module).

Example:

```
In [61]: from Bio.PDB import Superimposer
         sup = Superimposer()
```

```
In [62]: sup.set_atoms(fixed, moving)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-62-90d683778703> in <module>()
----> 1 sup.set_atoms(fixed, moving)

NameError: name 'fixed' is not defined
```

```
In [63]: print(sup.rotran)
         print(sup.rms)
```

```
None
None
```

```
In [64]: sup.apply(moving)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-64-c5ac66567782> in <module>()
----> 1 sup.apply(moving)

NameError: name 'moving' is not defined
```

To superimpose two structures based on their active sites, use the active site atoms to calculate the rotation/translation matrices (as above), and apply these to the whole molecule.

12.6.6 Mapping the residues of two related structures onto each other

First, create an alignment file in FASTA format, then use the `StructureAlignment` class. This class can also be used for alignments with more than two structures.

12.6.7 Calculating the Half Sphere Exposure

Half Sphere Exposure (HSE) is a new, 2D measure of solvent exposure @hamelryck2005. Basically, it counts the number of Cα atoms around a residue in the direction of its side chain, and in the opposite direction (within a radius of 13). Despite its simplicity, it outperforms many other measures of solvent exposure.

HSE comes in two flavors: HSEα and HSEβ. The former only uses the Cα atom positions, while the latter uses the Cα and Cβ atom positions. The HSE measure is calculated by the `HSExposure` class, which can also calculate the contact number. The latter class has methods which return dictionaries that map a `Residue` object to its corresponding HSEα, HSEβ and contact number values.

Example:

```
In [66]: from Bio.PDB import HSExposure
         model = structure[0]
         hse = HSExposure()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-b00b7a96aff9> in <module>()
      1 from Bio.PDB import HSExposure
      2 model = structure[0]
----> 3 hse = HSExposure()

TypeError: 'module' object is not callable
```

```
In [67]: exp_ca = hse.calc_hs_exposure(model, option='CA3')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-67-72378e1b79de> in <module>()
----> 1 exp_ca = hse.calc_hs_exposure(model, option='CA3')

NameError: name 'hse' is not defined
```

```
In [ ]: exp_cb=hse.calc_hs_exposure(model, option='CB')
```

```
In [ ]: exp_fs = hse.calc_fs_exposure(model)
```

```
In [ ]: print(exp_ca[some_residue])
```

12.6.8 Determining the secondary structure

For this functionality, you need to install DSSP (and obtain a license for it — free for academic use, see http://www.cmbi.kun.nl/gv/dssp/). Then use the `DSSP` class, which maps `Residue` objects to their secondary structure (and accessible surface area). The DSSP codes are listed in Table [cap:DSSP-codes]. Note that DSSP (the program, and thus by consequence the class) cannot handle multiple models!

Code	Secondary structure
H	α-helix
B	Isolated β-bridge residue
E	Strand
G	3-10 helix
I	Π-helix
T	Turn
S	Bend
•	Other

Table: [cap:DSSP-codes]DSSP codes in Bio.PDB.

The `DSSP` class can also be used to calculate the accessible surface area of a residue. But see also section [subsec:residue_depth].

12.6.9 Calculating the residue depth[subsec:residue_depth]

Residue depth is the average distance of a residue's atoms from the solvent accessible surface. It's a fairly new and very powerful parameterization of solvent accessibility. For this functionality, you need to install Michel Sanner's MSMS program (http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html). Then use the `ResidueDepth` class. This class behaves as a dictionary which maps `Residue` objects to corresponding (residue depth, Cα depth) tuples. The Cα depth is the distance of a residue's Cα atom to the solvent accessible surface.

Example:

```
In [69]: from Bio.PDB import ResidueDepth
         model = structure[0]
         rd = ResidueDepth(model, pdb_file)
         residue_depth, ca_depth=rd[some_residue]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-69-1139411bd4aa> in <module>()
      1 from Bio.PDB import ResidueDepth
      2 model = structure[0]
----> 3 rd = ResidueDepth(model, pdb_file)
      4 residue_depth, ca_depth=rd[some_residue]

NameError: name 'pdb_file' is not defined
```

You can also get access to the molecular surface itself (via the `get_surface` function), in the form of a Numeric Python array with the surface points.

12.7 Common problems in PDB files

It is well known that many PDB files contain semantic errors (not the structures themselves, but their representation in PDB files). Bio.PDB tries to handle this in two ways. The PDBParser object can behave in two ways: a restrictive way and a permissive way, which is the default.

Example:

```
In [70]: # Permissive parser
         parser = PDBParser(PERMISSIVE=1)
         parser = PDBParser() # The same (default)
```

```
In [71]: strict_parser = PDBParser(PERMISSIVE=0)
```

In the permissive state (DEFAULT), PDB files that obviously contain errors are "corrected" (i.e. some residues or atoms are left out). These errors include:

- Multiple residues with the same identifier
- Multiple atoms with the same identifier (taking into account the altloc identifier)

These errors indicate real problems in the PDB file (for details see @hamelryck2003a [Hamelryck and Manderick, 2003]). In the restrictive state, PDB files with errors cause an exception to occur. This is useful to find errors in PDB files.

Some errors however are automatically corrected. Normally each disordered atom should have a non-blank altloc identifier. However, there are many structures that do not follow this convention, and have a blank and a non-blank identifier for two disordered positions of the same atom. This is automatically interpreted in the right way.

Sometimes a structure contains a list of residues belonging to chain A, followed by residues belonging to chain B, and again followed by residues belonging to chain A, i.e. the chains are 'broken'. This is also correctly interpreted.

12.7.1 Examples[problem structures]

The PDBParser/Structure class was tested on about 800 structures (each belonging to a unique SCOP superfamily). This takes about 20 minutes, or on average 1.5 seconds per structure. Parsing the structure of the large ribosomal subunit (1FKK), which contains about 64000 atoms, takes 10 seconds on a 1000 MHz PC.

Three exceptions were generated in cases where an unambiguous data structure could not be built. In all three cases, the likely cause is an error in the PDB file that should be corrected. Generating an exception in these cases is much better than running the chance of incorrectly describing the structure in a data structure.

Duplicate residues

One structure contains two amino acid residues in one chain with the same sequence identifier (resseq 3) and icode. Upon inspection it was found that this chain contains the residues Thr A3, . . . , Gly A202, Leu A3, Glu A204. Clearly, Leu A3 should be Leu A203. A couple of similar situations exist for structure 1FFK (which e.g. contains Gly B64, Met B65, Glu B65, Thr B67, i.e. residue Glu B65 should be Glu B66).

Duplicate atoms

Structure 1EJG contains a Ser/Pro point mutation in chain A at position 22. In turn, Ser 22 contains some disordered atoms. As expected, all atoms belonging to Ser 22 have a non-blank altloc specifier (B or C). All atoms of Pro 22 have altloc A, except the N atom which has a blank altloc. This generates an exception, because all atoms belonging to two residues at a point mutation should have non-blank altloc. It turns out that this atom is probably shared by Ser and Pro 22, as Ser 22 misses the N atom. Again, this points to a problem in the file: the N atom should be present in both the Ser and the Pro residue, in both cases associated with a suitable altloc identifier.

12.7.2 Automatic correction

Some errors are quite common and can be easily corrected without much risk of making a wrong interpretation. These cases are listed below.

A blank altloc for a disordered atom

Normally each disordered atom should have a non-blank altloc identifier. However, there are many structures that do not follow this convention, and have a blank and a non-blank identifier for two disordered positions of the same atom. This is automatically interpreted in the right way.

Broken chains

Sometimes a structure contains a list of residues belonging to chain A, followed by residues belonging to chain B, and again followed by residues belonging to chain A, i.e. the chains are “broken”. This is correctly interpreted.

12.7.3 Fatal errors

Sometimes a PDB file cannot be unambiguously interpreted. Rather than guessing and risking a mistake, an exception is generated, and the user is expected to correct the PDB file. These cases are listed below.

Duplicate residues

All residues in a chain should have a unique id. This id is generated based on:

- The sequence identifier (resseq).
- The insertion code (icode).
- The hetfield string (“W” for waters and “H_” followed by the residue name for other hetero residues)

- The residue names of the residues in the case of point mutations (to store the Residue objects in a DisorderedResidue object).

If this does not lead to a unique id something is quite likely wrong, and an exception is generated.

Duplicate atoms

All atoms in a residue should have a unique id. This id is generated based on:

- The atom name (without spaces, or with spaces if a problem arises).
- The altloc specifier.

If this does not lead to a unique id something is quite likely wrong, and an exception is generated.

12.8 Accessing the Protein Data Bank

12.8.1 Downloading structures from the Protein Data Bank

Structures can be downloaded from the PDB (Protein Data Bank) by using the `retrieve_pdb_file` method on a `PDBList` object. The argument for this method is the PDB identifier of the structure.

```
In [73]: from Bio.PDB import PDBList
         pdbl = PDBList()
         pdbl.retrieve_pdb_file('1FAT')
```

```
Downloading PDB structure '1FAT'...
```

```
Out[73]: '/home/tiago_antao/biopython-notebook/notebooks/fa/pdb1fat.ent'
```

The `PDBList` class can also be used as a command-line tool:

```
python PDBList.py 1fat
```

The downloaded file will be called `pdb1fat.ent` and stored in the current working directory. Note that the `retrieve_pdb_file` method also has an optional argument `pdir` that specifies a specific directory in which to store the downloaded PDB files.

The `retrieve_pdb_file` method also has some options to specify the compression format used for the download, and the program used for local decompression (default `.Z` format and `gunzip`). In addition, the PDB ftp site can be specified upon creation of the `PDBList` object. By default, the server of the Worldwide Protein Data Bank (ftp://ftp.wwpdb.org/pub/pdb/data/structures/divided/pdb/) is used. See the API documentation for more details. Thanks again to Kristian Rother for donating this module.

12.8.2 Downloading the entire PDB

The following commands will store all PDB files in the `/data/pdb` directory:

```
python PDBList.py all /data/pdb

python PDBList.py all /data/pdb -d
```

The API method for this is called `download_entire_pdb`. Adding the `-d` option will store all files in the same directory. Otherwise, they are sorted into PDB-style subdirectories according to their PDB ID's. Depending on the traffic, a complete download will take 2-4 days.

12.8.3 Keeping a local copy of the PDB up to date

This can also be done using the `PDBList` object. One simply creates a `PDBList` object (specifying the directory where the local copy of the PDB is present) and calls the `update_pdb` method:

```
In [75]: pl = PDBList(pdb='/tmp/data/pdb')
         pl.update_pdb()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-726659b0f2c5> in <module>()
      1 pl = PDBList(pdb='/tmp/data/pdb')
----> 2 pl.update_pdb()

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/PDBList.py in update_pdb(self)
    236         assert os.path.isdir(self.obsolete_pdb)
    237 
--> 238         new, modified, obsolete = self.get_recent_changes()
    239 
    240         for pdb_code in new + modified:

/home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/PDB/PDBList.py in get_recent_changes(sel
    114             recent = filter(str.isdigit,
    115                             (x.split()[-1] for x in handle.readlines())
--> 116                             )[-1]    117 
    118         path = self.pdb_server + '/pub/pdb/data/status/%s/' % (recent)

TypeError: 'filter' object is not subscriptable
```

CHAPTER 13

Bio.PopGen: Population genetics

Bio.PopGen is a Biopython module supporting population genetics, available in Biopython 1.44 onwards.

The medium term objective for the module is to support widely used data formats, applications and databases. This module is currently under intense development and support for new features should appear at a rather fast pace. Unfortunately this might also entail some instability on the API, especially if you are using a development version. APIs that are made available on our official public releases should be much more stable.

13.1 GenePop

GenePop (http://genepop.curtin.edu.au/) is a popular population genetics software package supporting Hardy-Weinberg tests, linkage desiquilibrium, population diferentiation, basic statistics, Fst and migration estimates, among others. GenePop does not supply sequence based statistics as it doesn't handle sequence data. The GenePop file format is supported by a wide range of other population genetic software applications, thus making it a relevant format in the population genetics field.

Bio.PopGen provides a parser and generator of GenePop file format. Utilities to manipulate the content of a record are also provided. Here is an example on how to read a GenePop file (you can find example GenePop data files in the Test/PopGen directory of Biopython):

```
In [1]: from copy import deepcopy
        from Bio.PopGen import GenePop

        handle = open("data/example.gen")
        master_rec = GenePop.read(handle)
        handle.close()
```

This will read a file called example.gen and parse it. If you do print rec, the record will be output again, in GenePop format.

The most important information in rec will be the loci names and population information (but there is more – use help(GenePop.Record) to check the API documentation). Loci names can be found on rec.loci_list. Population information can be found on rec.populations. Populations is a list with one element per population. Each element is itself a list of individuals, each individual is a pair composed by individual name and a list of alleles (2 per marker), here is an example for rec.populations:

```
[
    [
        ('Ind1', [(1, 2),    (3, 3), (200, 201)],
        ('Ind2', [(2, None), (3, 3), (None, None)],
    ],
    [
        ('Other1', [(1, 1),  (4, 3), (200, 200)],
    ]
]
```

So we have two populations, the first with two individuals, the second with only one. The first individual of the first population is called Ind1, allelic information for each of the 3 loci follows. Please note that for any locus, information might be missing (see as an example, Ind2 above).

A few utility functions to manipulate GenePop records are made available, here is an example:

13.2 Operations on GenePop records

```
In [2]: print('Total populations %d' % len(master_rec.populations))
        rec = deepcopy(master_rec)
        pos = 0
        rec.remove_population(pos)
        print('After remove %d' % len(rec.populations))
        #Removes a population from a record, pos is the population position in
        #  rec.populations, remember that it starts on position 0.
        #  rec is altered.
```

```
Total populations 3
After remove 2
```

```
In [3]: print('Loci names: %s' %  ', '.join(master_rec.loci_list))
        rec = deepcopy(master_rec)
        pos = 0
        rec.remove_locus_by_position(pos)
        print('After removal: %s' %  ', '.join(rec.loci_list))
        #Removes a locus by its position, pos is the locus position in
        #  rec.loci_list, remember that it starts on position 0.
        #  rec is altered.
```

```
Loci names: loci1, another, and finally
After removal: another, and finally
```

```
In [5]: name = 'loci1'
        rec.remove_locus_by_name(name)
        #Removes a locus by its name, name is the locus name as in
        #  rec.loci_list. If the name doesn't exist the function fails
        #  silently.
        #  rec is altered.
```

```
In [7]: rec_loci = rec.split_in_loci(master_rec)
        #Splits a record in loci, that is, for each loci, it creates a new
        #  record, with a single loci and all populations.
        #  The result is returned in a dictionary, being each key the locus name.
        #  The value is the GenePop record.
        #  rec is not altered.
```

```
In [12]: pop_names = ['pop1', 'pop2', 'pop3']
         rec_pops = rec.split_in_pops(pop_names)
         #Splits a record in populations, that is, for each population, it creates
         #  a new record, with a single population and all loci.
         #  The result is returned in a dictionary, being each key
         #  the population name. As population names are not available in GenePop,
         #  they are passed in array (pop_names).
         #  The value of each dictionary entry is the GenePop record.
         #  rec is not altered.
```

GenePop does not support population names, a limitation which can be cumbersome at times. Functionality to enable population names is currently being planned for Biopython. These extensions won't break compatibility in any way with the standard format. In the medium term, we would also like to support the GenePop web service.

13.3 Coalescent simulation

A coalescent simulation is a backward model of population genetics with relation to time. A simulation of ancestry is done until the Most Recent Common Ancestor (MRCA) is found. This ancestry relationship starting on the MRCA and ending on the current generation sample is sometimes called a genealogy. Simple cases assume a population of constant size in time, haploidy, no population structure, and simulate the alleles of a single locus under no selection pressure.

Coalescent theory is used in many fields like selection detection, estimation of demographic parameters of real populations or disease gene mapping.

The strategy followed in the Biopython implementation of the coalescent was not to create a new, built-in, simulator from scratch but to use an existing one, SIMCOAL2 (http://cmpg.unibe.ch/software/simcoal2/). SIMCOAL2 allows for, among others, population structure, multiple demographic events, simulation of multiple types of loci (SNPs, sequences, STRs/microsatellites and RFLPs) with recombination, diploidy multiple chromosomes or ascertainment bias. Notably SIMCOAL2 doesn't support any selection model. We recommend reading SIMCOAL2's documentation, available in the link above.

The input for SIMCOAL2 is a file specifying the desired demography and genome, the output is a set of files (typically around 1000) with the simulated genomes of a sample of individuals per subpopulation. This set of files can be used in many ways, like to compute confidence intervals where which certain statistics (e.g., Fst or Tajima D) are expected to lie. Real population genetics datasets statistics can then be compared to those confidence intervals.

Biopython coalescent code allows to create demographic scenarios and genomes and to run SIMCOAL2.

13.3.1 Creating scenarios

Creating a scenario involves both creating a demography and a chromosome structure. In many cases (e.g. when doing Approximate Bayesian Computations – ABC) it is important to test many parameter variations (e.g. vary the effective population size, Ne, between 10, 50, 500 and 1000 individuals). The code provided allows for the simulation of scenarios with different demographic parameters very easily.

Below we see how we can create scenarios and then how simulate them.

Demography

A few predefined demographies are built-in, all have two shared parameters: sample size (called sample_size on the template, see below for its use) per deme and deme size, i.e. subpopulation size (pop_size). All demographies are available as templates where all parameters can be varied, each template has a system name. The prefedined demographies/templates are:

Single population, constant size The standard parameters are enough to specify it. Template name: simple.

Single population, bottleneck As seen on the figure below. The parameters are current population size (pop_size on template ne3 on figure), time of expansion, given as the generation in the past when it occurred (expand_gen), effective population size during bottleneck (ne2), time of contraction (contract_gen) and original size in the remote past (ne3). Template name: bottle.

Island model The typical island model. The total number of demes is specified by total_demes and the migration rate by mig. Template name island.

Stepping stone model - 1 dimension The stepping stone model in 1 dimension, extremes disconnected. The total number of demes is total_demes, migration rate is mig. Template name is ssm_1d.

Stepping stone model - 2 dimensions The stepping stone model in 2 dimensions, extremes disconnected. The parameters are x for the horizontal dimension and y for the vertical (being the total number of demes x times y), migration rate is mig. Template name is ssm_2d.

```
In [14]: #FIGURE HERE
```

In our first example, we will generate a template for a single population, constant size model with a sample size of 30 and a deme size of 500. The code for this is:

```
In [15]: from Bio.PopGen.SimCoal.Template import generate_simcoal_from_template

         generate_simcoal_from_template('simple',
             [(1, [('SNP', [24, 0.0005, 0.0])])],
             [('sample_size', [30]),
             ('pop_size', [100])])
```

Executing this code snippet will generate a file on the current directory called simple_100_300.par this file can be given as input to SIMCOAL2 to simulate the demography (below we will see how Biopython can take care of calling SIMCOAL2).

This code consists of a single function call, let's discuss it parameter by parameter.

The first parameter is the template id (from the list above). We are using the id 'simple' which is the template for a single population of constant size along time.

The second parameter is the chromosome structure. Please ignore it for now, it will be explained in the next section.

The third parameter is a list of all required parameters (recall that the simple model only needs sample_size and pop_size) and possible values (in this case each parameter only has a possible value).

Now, let's consider an example where we want to generate several island models, and we are interested in varying the number of demes: 10, 50 and 100 with a migration rate of 1%. Sample size and deme size will be the same as before. Here is the code:

```
In [16]: from Bio.PopGen.SimCoal.Template import generate_simcoal_from_template

         generate_simcoal_from_template('island',
             [(1, [('SNP', [24, 0.0005, 0.0])])],
             [('sample_size', [30]),
             ('pop_size', [100]),
             ('mig', [0.01]),
             ('total_demes', [10, 50, 100])])
```

In this case, 3 files will be generated: island_100_0.01_100_30.par, island_10_0.01_100_30.par and island_50_0.01_100_30.par. Notice the rule to make file names: template name, followed by parameter values in reverse order.

A few, arguably more esoteric template demographies exist (please check the Bio/PopGen/SimCoal/data directory on Biopython source tree). Furthermore it is possible for the user to create new templates. That functionality will be discussed in a future version of this document.

Chromosome structure

We strongly recommend reading SIMCOAL2 documentation to understand the full potential available in modeling chromosome structures. In this subsection we only discuss how to implement chromosome structures using the Biopython interface, not the underlying SIMCOAL2 capabilities.

We will start by implementing a single chromosome, with 24 SNPs with a recombination rate immediately on the right of each locus of 0.0005 and a minimum frequency of the minor allele of 0. This will be specified by the following list (to be passed as second parameter to the function generate_simcoal_from_template):

`python [(1, [('SNP', [24, 0.0005, 0.0])])]`

This is actually the chromosome structure used in the above examples.

The chromosome structure is represented by a list of chromosomes, each chromosome (i.e., each element in the list) is composed by a tuple (a pair): the first element is the number of times the chromosome is to be repeated (as there might be interest in repeating the same chromosome many times). The second element is a list of the actual components of the chromosome. Each element is again a pair, the first member is the locus type and the second element the parameters for that locus type. Confused? Before showing more examples let's review the example above: We have a list with one element (thus one chromosome), the chromosome is a single instance (therefore not to be repeated), it is composed of 24 SNPs, with a recombination rate of 0.0005 between each consecutive SNP, the minimum frequency of the minor allele is 0.0 (i.e, it can be absent from a certain population).

Let's see a more complicated example:

```
[
  (5, [
        ('SNP', [24, 0.0005, 0.0])
      ]
  ),
  (2, [
        ('DNA', [10, 0.0, 0.00005, 0.33]),
        ('RFLP', [1, 0.0, 0.0001]),
        ('MICROSAT', [1, 0.0, 0.001, 0.0, 0.0])
      ]
  )
]
```

We start by having 5 chromosomes with the same structure as above (i.e., 24 SNPs). We then have 2 chromosomes which have a DNA sequence with 10 nucleotides, 0.0 recombination rate, 0.0005 mutation rate, and a transition rate of 0.33. Then we have an RFLP with 0.0 recombination rate to the next locus and a 0.0001 mutation rate. Finally we have a microsatellite (or STR), with 0.0 recombination rate to the next locus (note, that as this is a single microsatellite which has no loci following, this recombination rate here is irrelevant), with a mutation rate of 0.001, geometric parameter of 0.0 and a range constraint of 0.0 (for information about this parameters please consult the SIMCOAL2 documentation, you can use them to simulate various mutation models, including the typical – for microsatellites – stepwise mutation model among others).

13.3.2 Running SIMCOAL2

We now discuss how to run SIMCOAL2 from inside Biopython. It is required that the binary for SIMCOAL2 is called simcoal2 (or simcoal2.exe on Windows based platforms), please note that the typical name when downloading the program is in the format simcoal2_x_y. As such, when installing SIMCOAL2 you will need to rename of the downloaded executable so that Biopython can find it.

It is possible to run SIMCOAL2 on files that were not generated using the method above (e.g., writing a parameter file by hand), but we will show an example by creating a model using the framework presented above.

```
In [18]: from Bio.PopGen.SimCoal.Template import generate_simcoal_from_template
         from Bio.PopGen.SimCoal.Controller import SimCoalController


         generate_simcoal_from_template('simple',
             [
               (5, [
                    ('SNP', [24, 0.0005, 0.0])
                  ]
               ),
               (2, [
                    ('DNA', [10, 0.0, 0.00005, 0.33]),
                    ('RFLP', [1, 0.0, 0.0001]),
                    ('MICROSAT', [1, 0.0, 0.001, 0.0, 0.0])
                  ]
               )
             ],
             [('sample_size', [30]),
             ('pop_size', [100])])

         #Simcoal not installed in the tutorial
         #ctrl = SimCoalController('.')
         #ctrl.run_simcoal('simple_100_30.par', 50)
```

The lines of interest are the last two (plus the new import). Firstly a controller for the application is created. The directory where the binary is located has to be specified.

The simulator is then run on the last line: we know, from the rules explained above, that the input file name is simple_100_30.par for the simulation parameter file created. We then specify that we want to run 50 independent simulations, by default Biopython requests a simulation of diploid data, but a third parameter can be added to simulate haploid data (adding as a parameter the string '0'). SIMCOAL2 will now run (please note that this can take quite a lot of time) and will create a directory with the simulation results. The results can now be analysed (typically studying the data with Arlequin3). In the future Biopython might support reading the Arlequin3 format and thus allowing for the analysis of SIMCOAL2 data inside Biopython.

Source of the materials: Biopython Tutorial and Cookbook (adapted)

CHAPTER 14

Phylogenetics with Bio.Phylo

14.1 Demo: what is in a tree?

Lets open an example newick file

```
In [1]: import copy
        from io import StringIO

        from Bio import Phylo
        from Bio.Phylo.Applications import PhymlCommandline
        from Bio.Phylo.PAML import codeml
        from Bio.Phylo.PhyloXML import Phylogeny

        %matplotlib inline

        tree = Phylo.read("data/simple.dnd", "newick")
```

Printing the tree object as a string gives us a look at the entire object hierarchy.

```
In [2]: print(tree)
```

```
Tree(rooted=False, weight=1.0)
    Clade()
        Clade()
            Clade()
                Clade(name='A')
                Clade(name='B')
            Clade()
                Clade(name='C')
                Clade(name='D')
        Clade()
            Clade(name='E')
            Clade(name='F')
            Clade(name='G')
```

The Tree object contains global information about the tree, such as whether it's rooted or unrooted. It has one root clade, and under that, it's nested lists of clades all the way down to the tips.

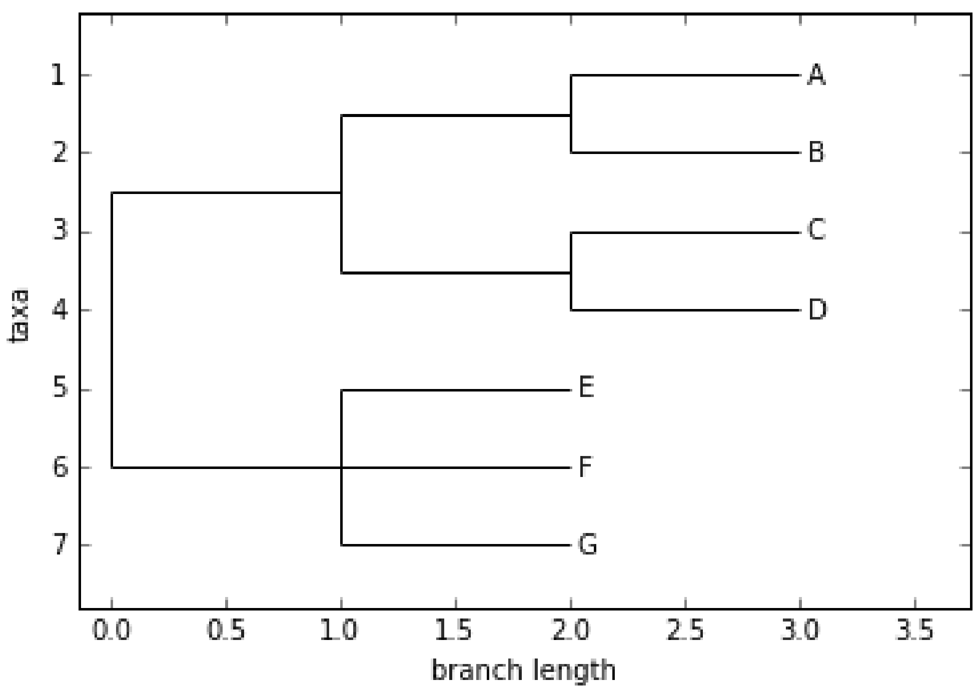
The function draw_ascii creates a simple ASCII-art (plain text) dendrogram. This is a convenient visualization for interactive exploration, in case better graphical tools aren't available.

```
In [3]: Phylo.draw_ascii(tree)
```

```
                                                    ________________________ A
                           ________________________|
                          |                        |________________________ B
  ________________________|
 |                        |                         ________________________ C
 |                        |________________________|
_|                                                 |________________________ D
 |
 |                         ________________________ E
 |                        |
 |________________________|________________________ F
                          |
                          |________________________ G
```

you can create a graphic using the draw function

```
In [4]: tree.rooted = True
        Phylo.draw(tree)
```



14.1.1 Coloring branches within a tree

The functions draw and draw_graphviz support the display of different colors and branch widths in a tree. As of Biopython 1.59, the color and width attributes are available on the basic Clade object and there's nothing extra required to use them. Both attributes refer to the branch leading the given clade, and apply recursively, so all descendent branches will also inherit the assigned width and color values during display.

In earlier versions of Biopython, these were special features of PhyloXML trees, and using the attributes required first converting the tree to a subclass of the basic tree object called Phylogeny, from the Bio.Phylo.PhyloXML module.

In Biopython 1.55 and later, this is a convenient tree method:

```
In [5]: tree = tree.as_phyloxml()
        tree = Phylogeny.from_tree(tree)
```

Note that the file formats Newick and Nexus don't support branch colors or widths, so if you use these attributes in Bio.Phylo, you will only be able to save the values in PhyloXML format. (You can still save a tree as Newick or Nexus, but the color and width values will be skipped in the output file.)

Now we can begin assigning colors. First, we'll color the root clade gray. We can do that by assigning the 24-bit color value as an RGB triple, an HTML-style hex string, or the name of one of the predefined colors.

```
In [6]: tree.root.color = (128, 128, 128)
        tree.root.color = "#808080"        # This is one alternative
        tree.root.color = "gray"           # This is another
```

Colors for a clade are treated as cascading down through the entire clade, so when we colorize the root here, it turns the whole tree gray. We can override that by assigning a different color lower down on the tree.

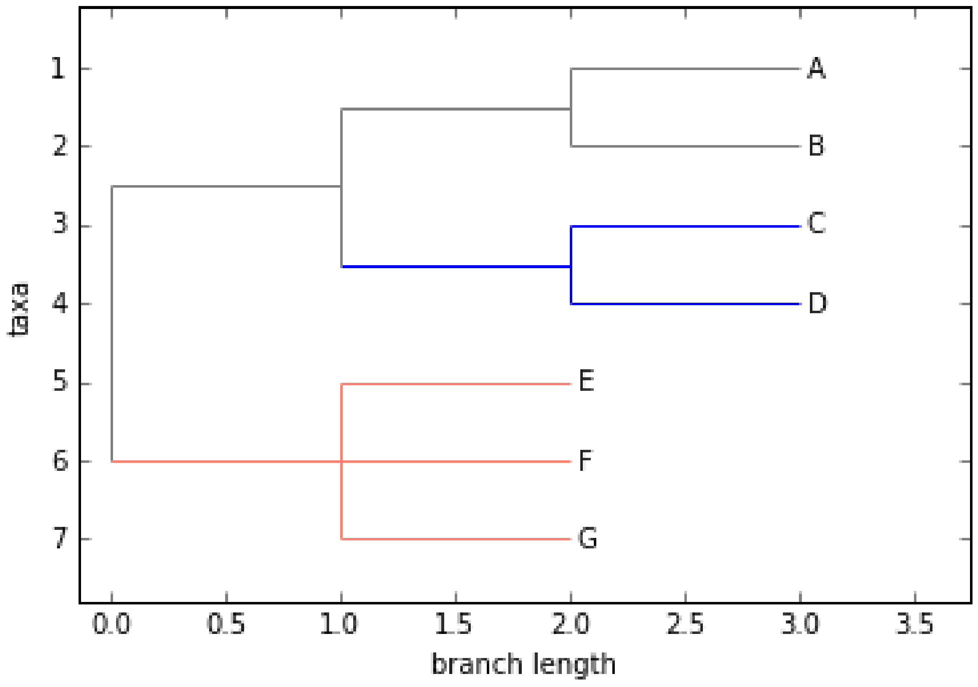
Let's target the most recent common ancestor (MRCA) of the nodes named "E" and "F". The common_ancestor method returns a reference to that clade in the original tree, so when we color that clade "salmon", the color will show up in the original tree.

```
In [7]: mrca = tree.common_ancestor({"name": "E"}, {"name": "F"})
        mrca.color = "salmon"
```

If we happened to know exactly where a certain clade is in the tree, in terms of nested list entries, we can jump directly to that position in the tree by indexing it. Here, the index [0,1] refers to the second child of the first child of the root.

```
In [8]: tree.clade[0, 1].color = "blue"
```

```
In [9]: Phylo.draw(tree)
```



Note that a clade's color includes the branch leading to that clade, as well as its descendents. The common ancestor of E and F turns out to be just under the root, and with this coloring we can see exactly where the root of the tree is.

My, we've accomplished a lot! Let's take a break here and save our work. Call the write function with a file name or handle — here we use standard output, to see what would be written — and the format phyloxml. PhyloXML saves the colors we assigned, so you can open this phyloXML file in another tree viewer like Archaeopteryx, and the colors will show up there, too.

14.2 I/O functions

Like SeqIO and AlignIO, Phylo handles file input and output through four functions: parse, read, write and convert, all of which support the tree file formats Newick, NEXUS, phyloXML and NeXML, as well as the Comparative Data Analysis Ontology (CDAO).

The read function parses a single tree in the given file and returns it. Careful; it will raise an error if the file contains more than one tree, or no trees.

```
In [10]: #from Bio import Phylo
         tree = Phylo.read("data/int_node_labels.nwk", "newick")
         print(tree)
```

```
Tree(rooted=False, weight=1.0)
    Clade(branch_length=75.0, name='gymnosperm')
        Clade(branch_length=25.0, name='Coniferales')
            Clade(branch_length=25.0)
                Clade(branch_length=10.0, name='Tax+nonSci')
                    Clade(branch_length=90.0, name='Taxaceae')
                        Clade(branch_length=125.0, name='Cephalotaxus')
                        Clade(branch_length=25.0, name='TT1')
                            Clade(branch_length=100.0, name='Taxus')
                            Clade(branch_length=100.0, name='Torreya')
                    Clade(branch_length=15.0, name='nonSci')
                        Clade(branch_length=15.11, name='Taw+others')
                            Clade(branch_length=49.060001, name='STCC')
                                Clade(branch_length=5.83, name='CupCallTax')
                                    Clade(branch_length=30.0, name='CJCPTT')
                                        Clade(branch_length=5.0, name='CCJCP')
                                            Clade(branch_length=5.0, name='CJCP')
                                                Clade(branch_length=5.0, name='CP')
                                                    Clade(branch_length=85.0, name='Calocedrus')
                                                    Clade(branch_length=85.0, name='Platycladus')
                                                Clade(branch_length=5.0, name='CJ')
                                                    Clade(branch_length=85.0, name='Cupressus')
                                                    Clade(branch_length=85.0, name='Juniperus')
                                            Clade(branch_length=95.0, name='Chamaecyparis')
                                        Clade(branch_length=92.13, name='TT2')
                                            Clade(branch_length=7.87, name='Thuja')
                                            Clade(branch_length=7.87, name='Thujopsis')
                                    Clade(branch_length=5.0, name='CTG')
                                        Clade(branch_length=5.0, name='CT')
                                            Clade(branch_length=120.0, name='Cryptomeria')
                                            Clade(branch_length=120.0, name='Taxodium')
                                        Clade(branch_length=125.0, name='Glyptostrobus')
                                Clade(branch_length=5.83, name='Sequoioid')
                                    Clade(branch_length=5.0, name='MS')
                                        Clade(branch_length=125.0, name='Metasequoia')
                                        Clade(branch_length=125.0, name='Sequoia')
                                    Clade(branch_length=130.0, name='Sequoiadendron')
                            Clade(branch_length=184.889999, name='Taiwania')
                        Clade(branch_length=200.0, name='Cunninghamia')
                Clade(branch_length=225.0, name='Sciadopitys')
            Clade(branch_length=66.0, name='Pinaceae')
```

```
                Clade(branch_length=24.0, name='NTPAK')
                    Clade(branch_length=54.0, name='AK')
                        Clade(branch_length=106.0, name='Abies')
                        Clade(branch_length=106.0, name='Keteleeria')
                    Clade(branch_length=4.0, name='NTP')
                        Clade(branch_length=156.0, name='Pseudolarix')
                        Clade(branch_length=156.0, name='Tsuga')
                Clade(branch_length=16.0, name='Pinoideae')
                    Clade(branch_length=81.0, name='LP')
                        Clade(branch_length=87.0, name='Larix')
                        Clade(branch_length=87.0, name='Pseudotsuga')
                    Clade(branch_length=13.0, name='PPC')
                        Clade(branch_length=155.0, name='Picea')
                        Clade(branch_length=155.0, name='Pinus')
        Clade(branch_length=275.0, name='Ginkgo')
```

To handle multiple (or an unknown number of) trees, use the parse function iterates through each of the trees in the given file:

```
In [11]: trees = Phylo.parse("data/phyloxml_examples.xml", "phyloxml")
         trees = list(trees)
         for tree in trees:
             print(tree)
```

```
Phylogeny(description='phyloXML allows to use either a "branch_length" attribute...', name='example 
    Clade()
        Clade(branch_length=0.06)
            Clade(branch_length=0.102, name='A')
            Clade(branch_length=0.23, name='B')
        Clade(branch_length=0.4, name='C')
Phylogeny(description='phyloXML allows to use either a "branch_length" attribute...', name='example 
    Clade()
        Clade(branch_length=0.06)
            Clade(branch_length=0.102, name='A')
            Clade(branch_length=0.23, name='B')
        Clade(branch_length=0.4, name='C')
Phylogeny(name='same example, with support of type "bootstrap"', rooted=True)
    Clade()
        Clade(branch_length=0.06, name='AB')
            Confidence(type='bootstrap', value=89.0)
            Clade(branch_length=0.102, name='A')
            Clade(branch_length=0.23, name='B')
        Clade(branch_length=0.4, name='C')
Phylogeny(name='same example, with species and sequence', rooted=True)
    Clade()
        Clade(name='AB')
            Clade(name='A')
                Sequence()
                    Annotation(desc='alcohol dehydrogenase')
                        Confidence(type='probability', value=0.99)
                E. coli
            Clade(name='B')
                Sequence()
                    Annotation(desc='alcohol dehydrogenase')
                        Confidence(type='probability', value=0.91)
                B. subtilis
        Clade(name='C')
            Sequence()
                Annotation(desc='alcohol dehydrogenase')
                    Confidence(type='probability', value=0.67)
            C. elegans
Phylogeny(name='same example, with gene duplication information and seque...', rooted=True)
    Clade()
        Events(speciations=1)
        Clade()
            Events(duplications=1)
            Clade()
                Sequence(id_source='x', name='alcohol dehydrogenase', symbol='adhB')
                    ncbi:AAB80874
                Bacillus subtilis
            Clade()
                Sequence(id_source='y', name='alcohol dehydrogenase', symbol='gbsB')
                    ncbi:CAB15083
                Bacillus subtilis
        Clade()
            Sequence(id_source='z', name='alcohol dehydrogenase', symbol='ADHX')
                ncbi:Q17335
                Annotation(ref='InterPro:IPR002085')
            Caenorhabditis elegans
    SequenceRelation(id_ref_0='x', id_ref_1='y', type='paralogy')
    SequenceRelation(id_ref_0='x', id_ref_1='z', type='orthology')
    SequenceRelation(id_ref_0='y', id_ref_1='z', type='orthology')
Phylogeny(name='similar example, with more detailed sequence data', rooted=True)
    Clade()
        Clade()
            Clade()
                Sequence(name='Alcohol dehydrogenase class-3', symbol='ADHX')
                    TDATGKPIKCMAAIAWEAKKPLSIEEVEVAPPKSGEVRIKILHSGVCHTD
                    UniProtKB:P81431
                    Annotation(ref='EC:1.1.1.1')
                    Annotation(ref='GO:0004022')
                OCTVU
                    NCBI:6645
            Clade()
                Sequence(name='Reticulon-4-interacting protein 1 homolog, mitochondrial ...', symbol=
                    MKGILLNGYGESLDLLEYKTDLPVPKPIKSQVLIKIHSTSINPLDNVMRK
                    UniProtKB:Q54II4
                    Annotation(ref='GO:0008270')
                    Annotation(ref='GO:0016491')
                DICDI
                    NCBI:44689
        Clade()
            Sequence(name='NADH-dependent butanol dehydrogenase B', symbol='ADHB')
                MVDFEYSIPTRIFFGKDKINVLGRELKKYGSKVLIVYGGGSIKRNGIYDK
                UniProtKB:Q04945
                Annotation(ref='GO:0046872')
                Annotation(ref='KEGG:Tetrachloroethene degradation')
            CLOAB
                NCBI:1488
Phylogeny(name='network, node B is connected to TWO nodes: AB and C', rooted=False)
    Clade()
        Clade(branch_length=0.06, id_source='ab', name='AB')
            Clade(branch_length=0.102, id_source='a', name='A')
            Clade(branch_length=0.23, id_source='b', name='B')
        Clade(branch_length=0.4, id_source='c', name='C')
    CladeRelation(id_ref_0='b', id_ref_1='c', type='network_connection')
Phylogeny(name='same example, using property elements to indicate a "dept...', rooted=True)
    Clade()
        Clade(name='AB')
```

```
            Clade(name='A')
                Property(applies_to='clade', datatype='xsd:integer', ref='NOAA:depth', unit='METRIC:m
            Clade(name='B')
                Property(applies_to='clade', datatype='xsd:integer', ref='NOAA:depth', unit='METRIC:m
        Clade(name='C')
            Property(applies_to='clade', datatype='xsd:integer', ref='NOAA:depth', unit='METRIC:m', v
Phylogeny(name='same example, using property elements to indicate a "dept...', rooted=True)
    Property(applies_to='node', datatype='xsd:integer', id_ref='id_a', ref='NOAA:depth', unit='METRIC
    Property(applies_to='node', datatype='xsd:integer', id_ref='id_b', ref='NOAA:depth', unit='METRIC
    Property(applies_to='node', datatype='xsd:integer', id_ref='id_c', ref='NOAA:depth', unit='METRIC
    Clade()
        Clade(name='AB')
            Clade(id_source='id_a', name='A')
            Clade(id_source='id_b', name='B')
        Clade(id_source='id_c', name='C')
Phylogeny(description='a pylogeny of some monitor lizards', name='monitor lizards', rooted=True)
    Clade()
        Varanus
            NCBI:8556
            http://www.embl-heidelberg.de/~uetz/families/Varanidae.html
        Clade()
            Distribution(desc='Africa')
            Varanus niloticus
                NCBI:62046
        Clade()
            Odatria
            Clade()
                Distribution(desc='Australia')
                Varanus storri
                    NCBI:169855
            Clade()
                Distribution(desc='Asia')
                Varanus timorensis
                    NCBI:62053
Phylogeny(name='A tree with phylogeographic information', rooted=True)
    Clade()
        Clade()
            Clade(name='A')
                Distribution(desc='Hirschweg, Winterthur, Switzerland')
                    Point(alt=472.0, geodetic_datum='WGS84', lat=47.481277, long=8.769303)
            Clade(name='B')
                Distribution(desc='Nagoya, Aichi, Japan')
                    Point(alt=10.0, geodetic_datum='WGS84', lat=35.155904, long=136.915863)
            Clade(name='C')
                Distribution(desc='ETH Zürich')
                    Point(alt=452.0, geodetic_datum='WGS84', lat=47.376334, long=8.548108)
        Clade(name='D')
            Distribution(desc='San Diego')
                Point(alt=104.0, geodetic_datum='WGS84', lat=32.880933, long=-117.217543)
Phylogeny(name='A tree with date information', rooted=True)
    Clade()
        Clade()
            Clade(name='A')
                425.0 mya
            Clade(name='B')
                320.0 mya
        Clade(name='C')
            600.0 mya
Phylogeny(name='Using another XML language to store an alignment', rooted=True)
```

```
    Clade()
        Clade()
            Clade(name='A')
            Clade(name='B')
        Clade(name='C')
```

Write a tree or iterable of trees back to file with the write function:

```
In [12]: tree1 = trees[0]
         Phylo.write(tree1, "data/tree1.nwk", "newick")
```

```
Out[12]: 1
```

```
In [13]: others = trees[1:]
         Phylo.write(others, "data/other_trees.nwk", "newick")
```

```
Out[13]: 12
```

Convert files between any of the supported formats with the convert function:

```
In [14]: Phylo.convert("data/tree1.nwk", "newick", "data/tree1.xml", "nexml")
```

```
Out[14]: 1
```

```
In [15]: Phylo.convert("data/other_trees.xml", "phyloxml", "data/other_trees.nex", "nexus")
```

```
Out[15]: 12
```

To use strings as input or output instead of actual files, use StringIO as you would with SeqIO and AlignIO:

```
In [16]: handle = StringIO("(((A,B),(C,D)),(E,F,G));")
         tree = Phylo.read(handle, "newick")
```

14.3 View and export trees

The simplest way to get an overview of a Tree object is to print it:

```
In [17]: tree = Phylo.read("data/example.xml", "phyloxml")
         print(tree)
```

```
Phylogeny(description='phyloXML allows to use either a "branch_length" attribute...', name='example
    Clade()
        Clade(branch_length=0.06)
            Clade(branch_length=0.102, name='A')
            Clade(branch_length=0.23, name='B')
        Clade(branch_length=0.4, name='C')
```

This is essentially an outline of the object hierarchy Biopython uses to represent a tree. But more likely, you'd want to see a drawing of the tree. There are three functions to do this.

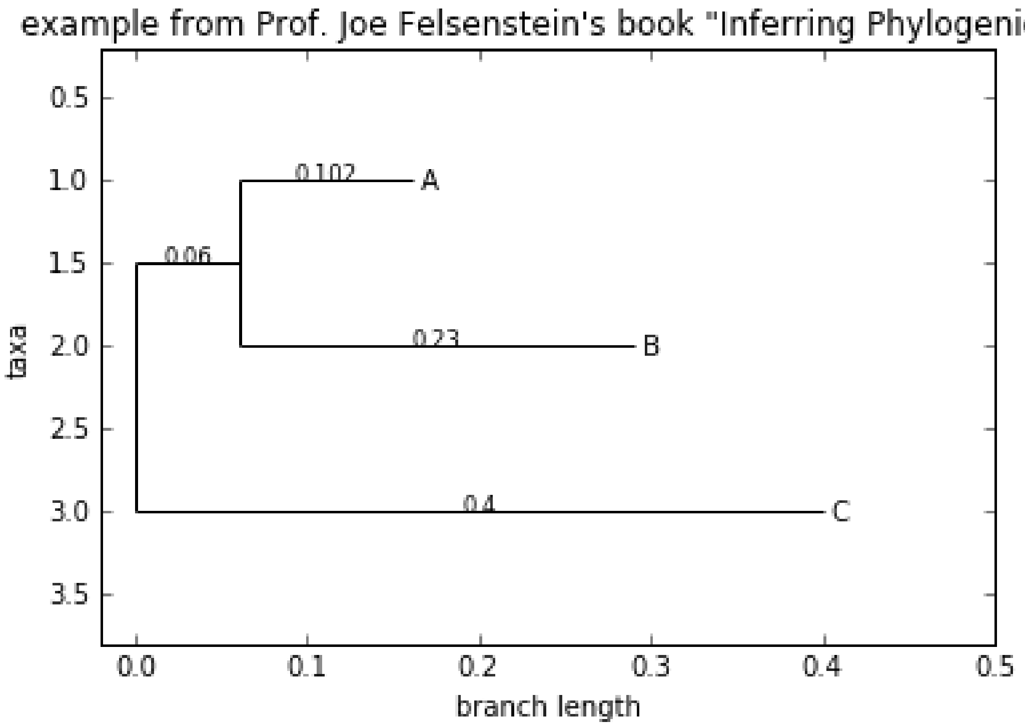
As we saw in the demo, draw_ascii prints an ascii-art drawing of the tree (a rooted phylogram) to standard output, or an open file handle if given. Not all of the available information about the tree is shown, but it provides a way to quickly view the tree without relying on any external dependencies.

```
In [18]: Phylo.draw_ascii(tree)
```

```
             __________________ A
  __________|
_|          |___________________________________________ B
 |
 |___________________________________________________________________________ C
```

The draw function draws a more attractive image using the matplotlib library. See the API documentation for details on the arguments it accepts to customize the output.

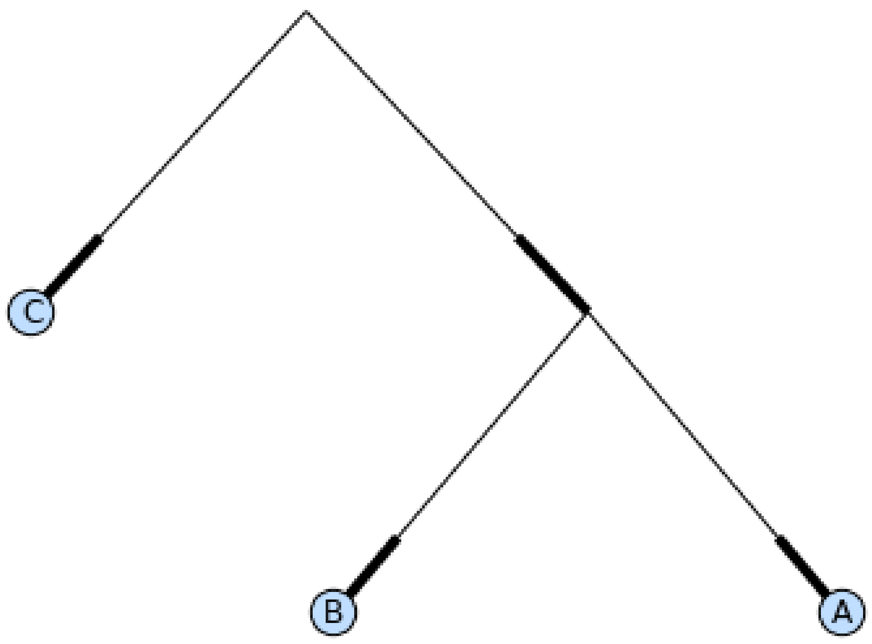
```
In [19]: Phylo.draw(tree, branch_labels=lambda c: c.branch_length)
```



draw_graphviz draws an unrooted cladogram, but requires that you have Graphviz, PyDot or PyGraphviz, NetworkX, and matplotlib (or pylab) installed. Using the same example as above, and the dot program included with Graphviz, let's draw a rooted tree (see Fig. 13.3):

```
In [22]: #Sadly we need to monkey patch...
         import networkx
         from networkx.drawing import nx_agraph
         networkx.graphviz_layout = nx_agraph.graphviz_layout
```

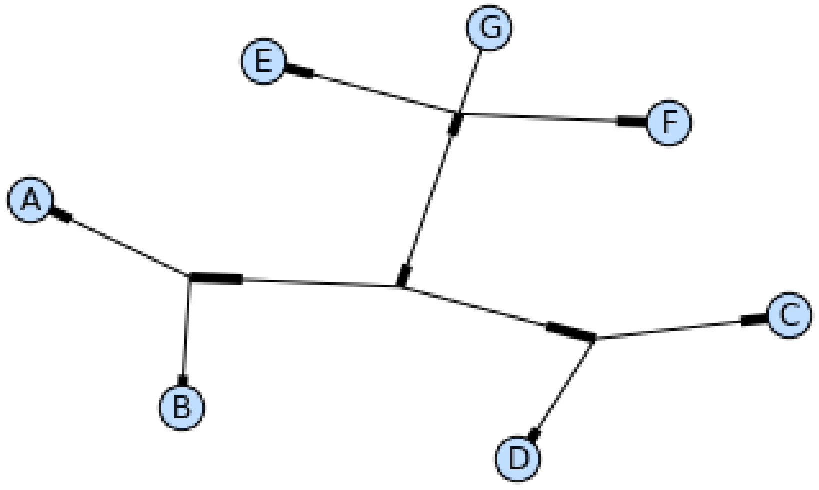
```
In [23]: Phylo.draw_graphviz(tree, prog='dot')
```



This exports the tree object to a NetworkX graph, uses Graphviz to lay out the nodes, and displays it using matplotlib. There are a number of keyword arguments that can modify the resulting diagram, including most of those accepted by the NetworkX functions networkx.draw and networkx.draw_graphviz.

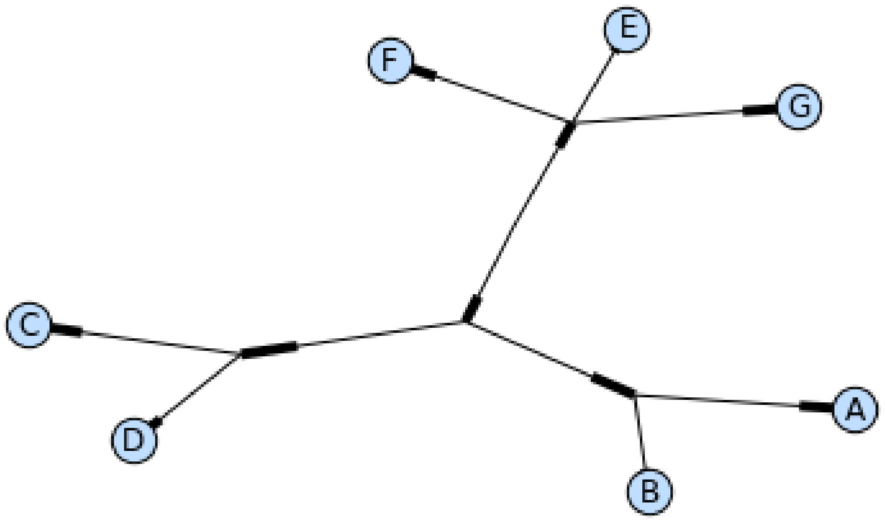
The display is also affected by the rooted attribute of the given tree object. Rooted trees are shown with a "head" on each branch indicating direction (see Fig. 13.3):

```
In [24]: tree = Phylo.read("data/simple.dnd", "newick")
         tree.rooted = True
         Phylo.draw_graphviz(tree)
```



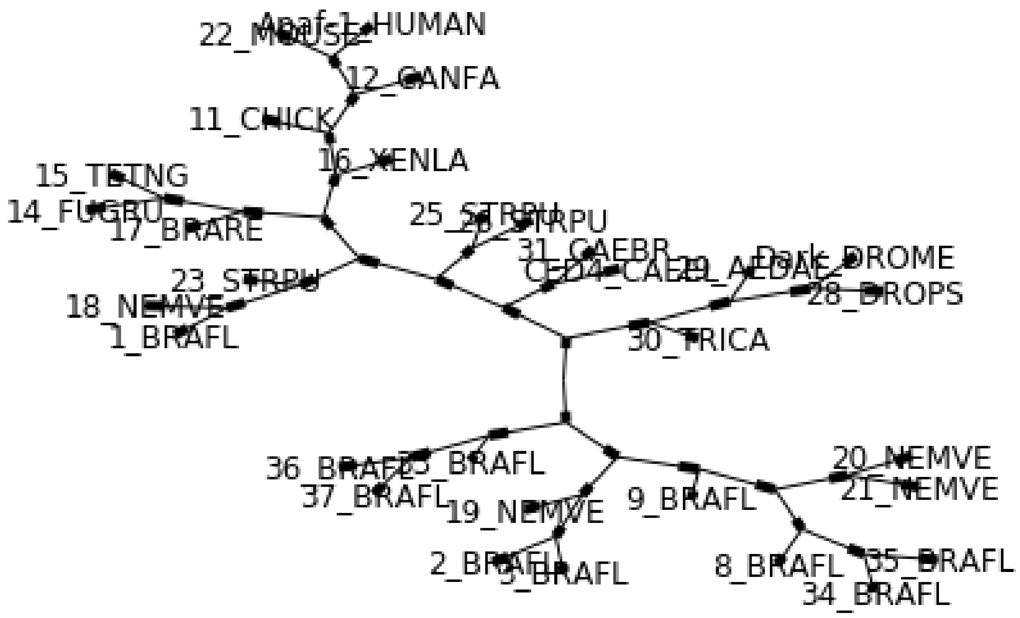
The “prog” argument specifies the Graphviz engine used for layout. The default, twopi, behaves well for any size tree, reliably avoiding crossed branches. The neato program may draw more attractive moderately-sized trees, but sometimes will cross branches (see Fig. 13.3). The dot program may be useful with small trees, but tends to do surprising things with the layout of larger trees.

```
In [25]: Phylo.draw_graphviz(tree, prog="neato")
```



This viewing mode is particularly handy for exploring larger trees, because the matplotlib viewer can zoom in on a selected region, thinning out a cluttered graphic.

```
In [26]: tree = Phylo.read("data/apaf.xml", "phyloxml")
         Phylo.draw_graphviz(tree, prog="neato", node_size=0)
```



Note that branch lengths are not displayed accurately, because Graphviz ignores them when creating the node layouts. The branch lengths are retained when exporting a tree as a NetworkX graph object (to_networkx), however.

See the Phylo page on the Biopython wiki (http://biopython.org/wiki/Phylo) for descriptions and examples of the more advanced functionality in draw_ascii, draw_graphviz and to_networkx.

14.4 Using Tree and Clade objects

The Tree objects produced by parse and read are containers for recursive sub-trees, attached to the Tree object at the root attribute (whether or not the phylogenic tree is actually considered rooted). A Tree has globally applied information for the phylogeny, such as rootedness, and a reference to a single Clade; a Clade has node- and clade-specific information, such as branch length, and a list of its own descendent Clade instances, attached at the clades attribute.

So there is a distinction between tree and tree.root. In practice, though, you rarely need to worry about it. To smooth over the difference, both Tree and Clade inherit from TreeMixin, which contains the implementations for methods that would be commonly used to search, inspect or modify a tree or any of its clades. This means that almost all of the methods supported by tree are also available on tree.root and any clade below it. (Clade also has a root property, which returns the clade object itself.)

14.4.1 Search and traversal methods

For convenience, we provide a couple of simplified methods that return all external or internal nodes directly as a list:

get_terminals makes a list of all of this tree's terminal (leaf) nodes. get_nonterminals makes a list of all of this tree's nonterminal (internal) nodes. These both wrap a method with full control over tree traversal, find_clades. Two more traversal methods, find_elements and find_any, rely on the same core functionality and accept the same arguments, which we'll call a "target specification" for lack of a better description. These specify which objects in the tree will be matched and returned during iteration. The first argument can be any of the following types:

A TreeElement instance, which tree elements will match by identity — so searching with a Clade instance as the target will find that clade in the tree; A string, which matches tree elements' string representation — in particular, a clade's name (added in Biopython 1.56); A class or type, where every tree element of the same type (or sub-type) will be matched; A dictionary where keys are tree element attributes and values are matched to the corresponding attribute of each tree element. This one gets even more elaborate: If an int is given, it matches numerically equal attributes, e.g. 1 will match 1 or 1.0 If a boolean is given (True or False), the corresponding attribute value is evaluated as a boolean and checked for the same None matches None If a string is given, the value is treated as a regular expression (which must match the whole string in the corresponding element attribute, not just a prefix). A given string without special regex characters will match string attributes exactly, so if you don't use regexes, don't worry about it. For example, in a tree with clade names Foo1, Foo2 and Foo3, tree.find_clades({"name": "Foo1"}) matches Foo1, {"name": "Foo.*"} matches all three clades, and {"name": "Foo"} doesn't match anything. Since floating-point arithmetic can produce some strange behavior, we don't support matching floats directly. Instead, use the boolean True to match every element with a nonzero value in the specified attribute, then filter on that attribute manually with an inequality (or exact number, if you like living dangerously).

If the dictionary contains multiple entries, a matching element must match each of the given attribute values — think "and", not "or".

A function taking a single argument (it will be applied to each element in the tree), returning True or False. For convenience, LookupError, AttributeError and ValueError are silenced, so this provides another safe way to search for floating-point values in the tree, or some more complex characteristic. After the target, there are two optional keyword arguments:

terminal — A boolean value to select for or against terminal clades (a.k.a. leaf nodes): True searches for only terminal clades, False for non-terminal (internal) clades, and the default, None, searches both terminal and non-terminal clades, as well as any tree elements lacking the is_terminal method. order — Tree traversal order: "preorder" (default) is depth-first search, "postorder" is DFS with child nodes preceding parents, and "level" is breadth-first search. Finally, the methods accept arbitrary keyword arguments which are treated the same way as a dictionary target specification: keys indicate the name of the element attribute to search for, and the argument value (string, integer, None or boolean) is compared to the value of each attribute found. If no keyword arguments are given, then any TreeElement types are matched. The code for this is generally shorter than passing a dictionary as the target specification: tree.find_clades({"name": "Foo1"}) can be shortened to tree.find_clades(name="Foo1").

(In Biopython 1.56 or later, this can be even shorter: tree.find_clades("Foo1"))

Now that we've mastered target specifications, here are the methods used to traverse a tree:

find_clades Find each clade containing a matching element. That is, find each element as with find_elements, but return the corresponding clade object. (This is usually what you want.) The result is an iterable through all matching objects, searching depth-first by default. This is not necessarily the same order as the elements appear in the Newick, Nexus or XML source file!

find_elements Find all tree elements matching the given attributes, and return the matching elements themselves. Simple Newick trees don't have complex sub-elements, so this behaves the same as find_clades on them. PhyloXML trees often do have complex objects attached to clades, so this method is useful for extracting those. find_any Return the first element found by find_elements(), or None. This is also useful for checking whether any matching element exists in the tree, and can be used in a conditional. Two more methods help navigating between nodes in the tree:

get_path List the clades directly between the tree root (or current clade) and the given target. Returns a list of all clade objects along this path, ending with the given target, but excluding the root clade. trace List of all clade object between two targets in this tree. Excluding start, including finish.

14.4.2 Information methods

These methods provide information about the whole tree (or any clade).

common_ancestor Find the most recent common ancestor of all the given targets. (This will be a Clade object). If no target is given, returns the root of the current clade (the one this method is called from); if 1 target is given, this returns the target itself. However, if any of the specified targets are not found in the current tree (or clade), an exception is raised. count_terminals Counts the number of terminal (leaf) nodes within the tree. depths Create a mapping of tree clades to depths. The result is a dictionary where the keys are all of the Clade instances in the tree, and the values are the distance from the root to each clade (including terminals). By default the distance is the cumulative branch length leading to the clade, but with the unit_branch_lengths=True option, only the number of branches (levels in the tree) is counted. distance Calculate the sum of the branch lengths between two targets. If only one target is specified, the other is the root of this tree. total_branch_length Calculate the sum of all the branch lengths in this tree. This is usually just called the "length" of the tree in phylogenetics, but we use a more explicit name to avoid confusion with Python terminology. The rest of these methods are boolean checks:

is_bifurcating True if the tree is strictly bifurcating; i.e. all nodes have either 2 or 0 children (internal or external, respectively). The root may have 3 descendents and still be considered part of a bifurcating tree. is_monophyletic Test if all of the given targets comprise a complete subclade — i.e., there exists a clade such that its terminals are the same set as the given targets. The targets should be terminals of the tree. For convenience, this method returns the common ancestor (MCRA) of the targets if they are monophyletic (instead of the value True), and False otherwise. is_parent_of True if target is a descendent of this tree — not required to be a direct descendent. To check direct descendents of a clade, simply use list membership testing: if subclade in clade: ... is_preterminal True if all direct descendents are terminal; False if any direct descendent is not terminal.

14.4.3 Modification methods

These methods modify the tree in-place. If you want to keep the original tree intact, make a complete copy of the tree first, using Python's copy module:

```
In [27]: tree = Phylo.read('data/example.xml', 'phyloxml')
         newtree = copy.deepcopy(tree)
```

collapse Deletes the target from the tree, relinking its children to its parent. collapse_all Collapse all the descendents of this tree, leaving only terminals. Branch lengths are preserved, i.e. the distance to each terminal stays the same. With a target specification (see above), collapses only the internal nodes matching the specification. ladderize Sort clades in-place according to the number of terminal nodes. Deepest clades are placed last by default. Use reverse=True to sort clades deepest-to-shallowest. prune Prunes a terminal clade from the tree. If taxon is from a bifurcation, the connecting node will be collapsed and its branch length added to remaining terminal node. This might no longer be a meaningful value. root_with_outgroup Reroot this tree with the outgroup clade containing the given targets, i.e. the common ancestor of the outgroup. This method is only available on Tree objects, not Clades. If the outgroup is identical to self.root, no change occurs. If the outgroup clade is terminal (e.g. a single terminal node is given as the outgroup), a new bifurcating root clade is created with a 0-length branch to the given outgroup. Otherwise, the internal node at the base of the outgroup becomes a trifurcating root for the whole tree. If the original root was bifurcating, it is dropped from the tree.

In all cases, the total branch length of the tree stays the same.

root_at_midpoint Reroot this tree at the calculated midpoint between the two most distant tips of the tree. (This uses root_with_outgroup under the hood.) split Generate n (default 2) new descendants. In a species tree, this is a speciation event. New clades have the given branch_length and the same name as this clade's root plus an integer suffix (counting from 0) — for example, splitting a clade named "A" produces the sub-clades "A0" and "A1". See the Phylo page on the Biopython wiki (http://biopython.org/wiki/Phylo) for more examples of using the available methods.

14.4.4 Features of PhyloXML trees

The phyloXML file format includes fields for annotating trees with additional data types and visual cues.

See the PhyloXML page on the Biopython wiki (http://biopython.org/wiki/PhyloXML) for descriptions and examples of using the additional annotation features provided by PhyloXML.

14.5 Running external applications

While Bio.Phylo doesn't infer trees from alignments itself, there are third-party programs available that do. These are supported through the module Bio.Phylo.Applications, using the same general framework as Bio.Emboss.Applications, Bio.Align.Applications and others.

Biopython 1.58 introduced a wrapper for PhyML (http://www.atgc-montpellier.fr/phyml/). The program accepts an input alignment in phylip-relaxed format (that's Phylip format, but without the 10-character limit on taxon names) and a variety of options. A quick example:

```
In [29]: #cmd = PhymlCommandline(input='Tests/Phylip/random.phy')
         cmd = PhymlCommandline(input='data/random.phy')
         out_log, err_log = cmd()
```

This generates a tree file and a stats file with the names [input filename]_phyml_tree.txt and [input filename]_phyml_stats.txt. The tree file is in Newick format:

```
In [30]: tree = Phylo.read('data/random.phy_phyml_tree.txt', 'newick')
         Phylo.draw_ascii(tree)
```

```
  _________ A
 |
_|__________________ F
 |
 |                  , B
 |__________________|
                    |                 , E
                    |_________________|
                                      |         , G
                                      |_________|
                                                |_______ I
                                                |
                                                |      ___ C
                                                |_____|
                                                      |     _ J
                                                      |____|
                                                           |      _________ D
                                                           |_____|
                                                                 | H
```

A similar wrapper for RAxML (http://sco.h-its.org/exelixis/software.html) was added in Biopython 1.60, and FastTree (http://www.microbesonline.org/fasttree/) in Biopython 1.62.

Note that some popular Phylip programs, including dnaml and protml, are already available through the EMBOSS wrappers in Bio.Emboss.Applications if you have the Phylip extensions to EMBOSS installed on your system. See Section 6.4 for some examples and clues on how to use programs like these.

14.6 PAML integration

iopython 1.58 brought support for PAML (http://abacus.gene.ucl.ac.uk/software/paml.html), a suite of programs for phylogenetic analysis by maximum likelihood. Currently the programs codeml, baseml and yn00 are implemented. Due to PAML's usage of control files rather than command line arguments to control runtime options, usage of this wrapper strays from the format of other application wrappers in Biopython.

A typical workflow would be to initialize a PAML object, specifying an alignment file, a tree file, an output file and a working directory. Next, runtime options are set via the set_options() method or by reading an existing control file. Finally, the program is run via the run() method and the output file is automatically parsed to a results dictionary.

Here is an example of typical usage of codeml:

```
In [32]: cml = codeml.Codeml()
         cml.alignment = "data/alignment.phylip"
         cml.tree = "data/species.tree"
         cml.out_file = "data/results.out"
         cml.working_dir = "./"
         cml.set_options(seqtype=1,
                 verbose=0,
                 noisy=0,
                 RateAncestor=0,
                 model=0,
                 NSsites=[0, 1, 2],
                 CodonFreq=2,
                 cleandata=1,
                 fix_alpha=1,
                 kappa=4.54006)
         results = cml.run()
         ns_sites = results.get("NSsites")
         m0 = ns_sites.get(0)
         m0_params = m0.get("parameters")
         print(m0_params.get("omega"))
```

```
0.0001
```

Existing output files may be parsed as well using a module's read() function:

```
In [33]: results = codeml.read("data/results.out")
         print(results.get("codon model"))
```

```
F3x4
```

Detailed documentation for this new module currently lives on the Biopython wiki: http://biopython.org/wiki/PAML

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 15

Sequence motif analysis using Bio.motifs

This chapter gives an overview of the functionality of the `Bio.motifs` package included in Biopython. It is intended for people who are involved in the analysis of sequence motifs, so I'll assume that you are familiar with basic notions of motif analysis. In case something is unclear, please look at Section [sec:links] for some relevant links.

Most of this chapter describes the new `Bio.motifs` package included in Biopython 1.61 onwards, which is replacing the older `Bio.Motif` package introduced with Biopython 1.50, which was in turn based on two older former Biopython modules, `Bio.AlignAce` and `Bio.MEME`. It provides most of their functionality with a unified motif object implementation.

Speaking of other libraries, if you are reading this you might be interested in TAMO, another python library designed to deal with sequence motifs. It supports more *de-novo* motif finders, but it is not a part of Biopython and has some restrictions on commercial use.

15.1 Motif objects

Since we are interested in motif analysis, we need to take a look at `Motif` objects in the first place. For that we need to import the Bio.motifs library:

```
In [1]: from Bio import motifs
```

and we can start creating our first motif objects. We can either create a `Motif` object from a list of instances of the motif, or we can obtain a `Motif` object by parsing a file from a motif database or motif finding software.

15.1.1 Creating a motif from instances

Suppose we have these instances of a DNA motif:

```
In [2]: from Bio.Seq import Seq
        instances = [Seq("TACAA"),
            Seq("TACGC"),
            Seq("TACAC"),
            Seq("TACCC"),
            Seq("AACCC"),
            Seq("AATGC"),
            Seq("AATGC")]
```

then we can create a Motif object as follows:

```
In [3]: m = motifs.create(instances)
```

The instances are saved in an attribute `m.instances`, which is essentially a Python list with some added functionality, as described below. Printing out the Motif object shows the instances from which it was constructed:

```
In [4]: print(m)
```

```
TACAA
TACGC
TACAC
TACCC
AACCC
AATGC
AATGC
```

The length of the motif is defined as the sequence length, which should be the same for all instances:

```
In [5]: len(m)
```

```
Out[5]: 5
```

The Motif object has an attribute `.counts` containing the counts of each nucleotide at each position. Printing this counts matrix shows it in an easily readable format:

```
In [6]: print(m.counts)
```

```
        0      1      2      3      4
A:   3.00   7.00   0.00   2.00   1.00
C:   0.00   0.00   5.00   2.00   6.00
G:   0.00   0.00   0.00   3.00   0.00
T:   4.00   0.00   2.00   0.00   0.00
```

You can access these counts as a dictionary:

```
In [ ]: m.counts['A']
```

but you can also think of it as a 2D array with the nucleotide as the first dimension and the position as the second dimension:

```
In [7]: m.counts['T', 0]
```

```
Out[7]: 4
```

```
In [8]: m.counts['T', 2]
```

```
Out[8]: 2
```

```
In [9]: m.counts['T', 3]
```

```
Out[9]: 0
```

You can also directly access columns of the counts matrix

```
In [ ]: m.counts[:, 3]
```

Instead of the nucleotide itself, you can also use the index of the nucleotide in the sorted letters in the alphabet of the motif:

```
In [ ]: m.alphabet
```

```
In [ ]: m.alphabet.letters
In [ ]: sorted(m.alphabet.letters)
```

```
In [ ]: m.counts['A',:]
```

```
In [ ]: m.counts[0,:]
```

The motif has an associated consensus sequence, defined as the sequence of letters along the positions of the motif for which the largest value in the corresponding columns of the `.counts` matrix is obtained:

```
In [ ]: m.consensus
```

as well as an anticonsensus sequence, corresponding to the smallest values in the columns of the `.counts` matrix:

```
In [ ]: m.anticonsensus
```

You can also ask for a degenerate consensus sequence, in which ambiguous nucleotides are used for positions where there are multiple nucleotides with high counts:

```
In [ ]: m.degenerate_consensus
```

Here, W and R follow the IUPAC nucleotide ambiguity codes: W is either A or T, and V is A, C, or G @cornish1985. The degenerate consensus sequence is constructed following the rules specified by Cavener @cavener1987.

We can also get the reverse complement of a motif:

```
In [ ]: r = m.reverse_complement()
        r.consensus
```

```
In [ ]: r.degenerate_consensus
```

```
In [ ]: print(r)
```

The reverse complement and the degenerate consensus sequence are only defined for DNA motifs.

15.1.2 Creating a sequence logo

If we have internet access, we can create a weblogo:

```
In [ ]: m.weblogo("mymotif.png")
```

We should get our logo saved as a PNG in the specified file.

15.2 Reading motifs

Creating motifs from instances by hand is a bit boring, so it's useful to have some I/O functions for reading and writing motifs. There are not any really well established standards for storing motifs, but there are a couple of formats that are more used than others.

15.2.1 JASPAR

One of the most popular motif databases is JASPAR. In addition to the motif sequence information, the JASPAR database stores a lot of meta-information for each motif. The module `Bio.motifs` contains a specialized class `jaspar.Motif` in which this meta-information is represented as attributes:

- `matrix_id` - the unique JASPAR motif ID, e.g. 'MA0004.1'
- `name` - the name of the TF, e.g. 'Arnt'
- `collection` - the JASPAR collection to which the motif belongs, e.g. 'CORE'
- `tf_class` - the structual class of this TF, e.g. 'Zipper-Type'

- `tf_family` - the family to which this TF belongs, e.g. 'Helix-Loop-Helix'
- `species` - the species to which this TF belongs, may have multiple values, these are specified as taxonomy IDs, e.g. 10090
- `tax_group` - the taxonomic supergroup to which this motif belongs, e.g. 'vertebrates'
- `acc` - the accession number of the TF protein, e.g. 'P53762'
- `data_type` - the type of data used to construct this motif, e.g. 'SELEX'
- `medline` - the Pubmed ID of literature supporting this motif, may be multiple values, e.g. 7592839
- `pazar_id` - external reference to the TF in the PAZAR database, e.g. 'TF0000003'
- `comment` - free form text containing notes about the construction of the motif

The `jaspar.Motif` class inherits from the generic `Motif` class and therefore provides all the facilities of any of the motif formats — reading motifs, writing motifs, scanning sequences for motif instances etc.

JASPAR stores motifs in several different ways including three different flat file formats and as an SQL database. All of these formats facilitate the construction of a counts matrix. However, the amount of meta information described above that is available varies with the format.

The JASPAR `sites` format

The first of the three flat file formats contains a list of instances. As an example, these are the beginning and ending lines of the JASPAR `Arnt.sites` file showing known binding sites of the mouse helix-loop-helix transcription factor Arnt.

```
>MA0004 ARNT 1
CACGTGatgtcctc
>MA0004 ARNT 2
CACGTGggaggtac
>MA0004 ARNT 3
CACGTGccgcgcgc
...
>MA0004 ARNT 18
AACGTGacagccctcc
>MA0004 ARNT 19
AACGTGcacatcgtcc
>MA0004 ARNT 20
aggaatCGCGTGc
```

The parts of the sequence in capital letters are the motif instances that were found to align to each other.

We can create a `Motif` object from these instances as follows:

```
In [ ]: from Bio import motifs
        with open("Arnt.sites") as handle:
            arnt = motifs.read(handle, "sites")
```

The instances from which this motif was created is stored in the `.instances` property:

```
In [ ]: print(arnt.instances[:3])
```

```
In [ ]: for instance in arnt.instances:
            print(instance)
```

The counts matrix of this motif is automatically calculated from the instances:

```
In [ ]: print(arnt.counts)
```

This format does not store any meta information.

The JASPAR pfm format

JASPAR also makes motifs available directly as a count matrix, without the instances from which it was created. This pfm format only stores the counts matrix for a single motif. For example, this is the JASPAR file SRF.pfm containing the counts matrix for the human SRF transcription factor:

```
2 9 0 1 32 3 46 1 43 15 2 2
 1 33 45 45 1 1 0 0 0 1 0 1
39 2 1 0 0 0 0 0 0 0 44 43
 4 2 0 0 13 42 0 45 3 30 0 0
```

We can create a motif for this count matrix as follows:

```
In [ ]: with open("SRF.pfm") as handle:
            srf = motifs.read(handle, "pfm")
```

```
In [ ]: print(srf.counts)
```

As this motif was created from the counts matrix directly, it has no instances associated with it:

```
In [ ]: print(srf.instances)
```

We can now ask for the consensus sequence of these two motifs:

```
In [ ]: print(arnt.counts.consensus)
```

```
In [ ]: print(srf.counts.consensus)
```

As with the instances file, no meta information is stored in this format.

The JASPAR format jaspar

The jaspar file format allows multiple motifs to be specified in a single file. In this format each of the motif records consist of a header line followed by four lines defining the counts matrix. The header line begins with a > character (similar to the Fasta file format) and is followed by the unique JASPAR matrix ID and the TF name. The following example shows a jaspar formatted file containing the three motifs Arnt, RUNX1 and MEF2A:

```
>MA0004.1 Arnt
A   [ 4 19  0  0  0  0 ]
C   [16  0 20  0  0  0 ]
G   [ 0  1  0 20  0 20 ]
T   [ 0  0  0  0 20  0 ]
>MA0002.1 RUNX1
A   [10 12  4  1  2  2  0  0  0  8 13 ]
C   [ 2  2  7  1  0  8  0  0  1  2  2 ]
G   [ 3  1  1  0 23  0 26 26  0  0  4 ]
T   [11 11 14 24  1 16  0  0 25 16  7 ]
>MA0052.1 MEF2A
A   [ 1  0 57  2  9  6 37  2 56  6 ]
C   [50  0  1  1  0  0  0  0  0  0 ]
G   [ 0  0  0  0  0  0  0  0  2 50 ]
T   [ 7 58  0 55 49 52 21 56  0  2 ]
```

The motifs are read as follows:

```
In [ ]: fh = open("jaspar_motifs.txt")
        for m in motifs.parse(fh, "jaspar"))
            print(m)
```

Note that printing a JASPAR motif yields both the counts data and the available meta-information.

Accessing the JASPAR database

In addition to parsing these flat file formats, we can also retrieve motifs from a JASPAR SQL database. Unlike the flat file formats, a JASPAR database allows storing of all possible meta information defined in the JASPAR `Motif` class. It is beyond the scope of this document to describe how to set up a JASPAR database (please see the main JASPAR website). Motifs are read from a JASPAR database using the `Bio.motifs.jaspar.db` module. First connect to the JASPAR database using the JASPAR5 class which models the the latest JASPAR schema:

```
In [ ]: from Bio.motifs.jaspar.db import JASPAR5
```

```
In [ ]: JASPAR_DB_HOST = <hostname>
        JASPAR_DB_NAME = <db_name>
        JASPAR_DB_USER = <user>
        JASPAR_DB_PASS = <passord>
```

```
In [ ]: jdb = JASPAR5(
        host=JASPAR_DB_HOST,
        name=JASPAR_DB_NAME,
        user=JASPAR_DB_USER,
        password=JASPAR_DB_PASS
        )
```

Now we can fetch a single motif by its unique JASPAR ID with the `fetch_motif_by_id` method. Note that a JASPAR ID conists of a base ID and a version number seperated by a decimal point, e.g. 'MA0004.1'. The `fetch_motif_by_id` method allows you to use either the fully specified ID or just the base ID. If only the base ID is provided, the latest version of the motif is returned.

```
In [ ]: arnt = jdb.fetch_motif_by_id("MA0004")
```

Printing the motif reveals that the JASPAR SQL database stores much more meta-information than the flat files:

```
In [ ]: print(arnt)
```

We can also fetch motifs by name. The name must be an exact match (partial matches or database wildcards are not currently supported). Note that as the name is not guaranteed to be unique, the `fetch_motifs_by_name` method actually returns a list.

```
In [ ]: motifs = jdb.fetch_motifs_by_name("Arnt")
        print(motifs[0])
```

The `fetch_motifs` method allows you to fetch motifs which match a specified set of criteria. These criteria include any of the above described meta information as well as certain matrix properties such as the minimum information content (`min_ic` in the example below), the minimum length of the matrix or the minimum number of sites used to construct the matrix. Only motifs which pass ALL the specified criteria are returned. Note that selection criteria which correspond to meta information which allow for multiple values may be specified as either a single value or a list of values, e.g. `tax_group` and `tf_family` in the example below.

```
In [ ]: motifs = jdb.fetch_motifs(
        collection = 'CORE',
        tax_group = ['vertebrates', 'insects'],
        tf_class = 'Winged Helix-Turn-Helix',
        tf_family = ['Forkhead', 'Ets'],
        min_ic = 12
        )
        for motif in motifs:
            pass # do something with the motif
```

Compatibility with Perl TFBS modules

An important thing to note is that the JASPAR `Motif` class was designed to be compatible with the popular Perl TFBS modules. Therefore some specifics about the choice of defaults for background and pseudocounts as well as how information content is computed and sequences searched for instances is based on this compatibility criteria. These choices are noted in the specific subsections below.

- **Choice of background:**
  The Perl `TFBS` modules appear to allow a choice of custom background probabilities (although the documentation states that uniform background is assumed). However the default is to use a uniform background. Therefore it is recommended that you use a uniform background for computing the position-specific scoring matrix (PSSM). This is the default when using the Biopython `motifs` module.
- **Choice of pseudocounts:**
  By default, the Perl `TFBS` modules use a pseudocount equal to $\sqrt{N} * \text{bg[nucleotide]}$, where N represents the total number of sequences used to construct the matrix. To apply this same pseudocount formula, set the motif `pseudocounts` attribute using the `jaspar.calculate\_pseudcounts()` function:

```
In [ ]: motif.pseudocounts = motifs.jaspar.calculate_pseudocounts(motif)
```

```
Note that it is possible for the counts matrix to have an unequal
number of sequences making up the columns. The pseudocount
computation uses the average number of sequences making up
the matrix. However, when `normalize` is called on the counts
matrix, each count value in a column is divided by the total number
of sequences making up that specific column, not by the average
number of sequences. This differs from the Perl `TFBS` modules
because the normalization is not done as a separate step and so the
average number of sequences is used throughout the computation of
the pssm. Therefore, for matrices with unequal column counts, the
PSSM computed by the `motifs` module will differ somewhat from the
pssm computed by the Perl `TFBS` modules.
```

- **Computation of matrix information content:**
  The information content (IC) or specificity of a matrix is computed using the `mean` method of the `PositionSpecificScoringMatrix` class. However of note, in the Perl `TFBS` modules the default behaviour is to compute the IC without first applying pseudocounts, even though by default the PSSMs are computed using pseudocounts as described above.
- **Searching for instances:**
  Searching for instances with the Perl `TFBS` motifs was usually performed using a relative score threshold, i.e. a score in the range 0 to 1. In order to compute the absolute PSSM score corresponding to a relative score one can use the equation:

```
In [ ]: abs_score =  (pssm.max - pssm.min) * rel_score + pssm.min
```

```
To convert the absolute score of an instance back to a relative
score, one can use the equation:
```

```
In [ ]: rel_score = (abs_score - pssm.min) / (pssm.max - pssm.min)
```

```
For example, using the Arnt motif before, let's search a sequence
with a relative score threshold of 0.8.
```

```
In [ ]: test_seq=Seq("TAAGCGTGCACGCGCAACACGTGCATTA", unambiguous_dna)
        arnt.pseudocounts = motifs.jaspar.calculate_pseudocounts(arnt)
        pssm = arnt.pssm
        max_score = pssm.max
        min_score = pssm.min
        abs_score_threshold = (max_score - min_score) * 0.8 + min_score
        for position, score in pssm.search(test_seq,
```

```
In [ ]: rel_score = (score - min_score) / (max_score - min_score)
        print("Position %d: score = %5.3f, rel. score = %5.3f" % (
```

15.2.2 MEME

MEME @bailey1994 is a tool for discovering motifs in a group of related DNA or protein sequences. It takes as input a group of DNA or protein sequences and outputs as many motifs as requested. Therefore, in contrast to JASPAR files, MEME output files typically contain multiple motifs. This is an example.

At the top of an output file generated by MEME shows some background information about the MEME and the version of MEME used:

```
********************************************************************************
MEME - Motif discovery tool
********************************************************************************
MEME version 3.0 (Release date: 2004/08/18 09:07:01)
...
```

Further down, the input set of training sequences is recapitulated:

```
********************************************************************************
TRAINING SET
********************************************************************************
DATAFILE= INO_up800.s
ALPHABET= ACGT
Sequence name            Weight Length  Sequence name            Weight Length
-------------            ------ ------  -------------            ------ ------
CHO1                     1.0000    800  CHO2                     1.0000    800
FAS1                     1.0000    800  FAS2                     1.0000    800
ACC1                     1.0000    800  INO1                     1.0000    800
OPI3                     1.0000    800
********************************************************************************
```

and the exact command line that was used:

```
********************************************************************************
COMMAND LINE SUMMARY
********************************************************************************
This information can also be useful in the event you wish to report a
problem with the MEME software.

command: meme -mod oops -dna -revcomp -nmotifs 2 -bfile yeast.nc.6.freq INO_up800.s
...
```

Next is detailed information on each motif that was found:

```
********************************************************************************
MOTIF  1        width =   12   sites =   7   llr = 95   E-value = 2.0e-001
********************************************************************************
--------------------------------------------------------------------------------
```

```
        Motif 1 Description
--------------------------------------------------------------------------------
Simplified        A  :::9:a::::3:
pos.-specific     C  ::a:9:11691a
probability       G  ::::1::94:4:
matrix            T  aa:1::9::11:
```

To parse this file (stored as `meme.dna.oops.txt`), use

```
In [ ]: handle = open("meme.dna.oops.txt")
        record = motifs.parse(handle, "meme")
        handle.close()
```

The `motifs.parse` command reads the complete file directly, so you can close the file after calling `motifs.parse`. The header information is stored in attributes:

```
In [ ]: record.version
```

```
In [ ]: record.datafile
```

```
In [ ]: record.command
```

```
In [ ]: record.alphabet
```

```
In [ ]: record.sequences
```

The record is an object of the `Bio.motifs.meme.Record` class. The class inherits from list, and you can think of `record` as a list of Motif objects:

```
In [ ]: len(record)
```

```
In [ ]: motif = record[0]
        print(motif.consensus)
```

```
In [ ]: print(motif.degenerate_consensus)
```

In addition to these generic motif attributes, each motif also stores its specific information as calculated by MEME. For example,

```
In [ ]: motif.num_occurrences
```

```
In [ ]: motif.length
```

```
In [ ]: evalue = motif.evalue
        print("%3.1g" % evalue)
```

```
In [ ]: motif.name
```

In addition to using an index into the record, as we did above, you can also find it by its name:

```
In [ ]: motif = record['Motif 1']
```

Each motif has an attribute `.instances` with the sequence instances in which the motif was found, providing some information on each instance:

```
In [ ]: len(motif.instances)
```

```
In [ ]: motif.instances[0]
```

```
In [ ]: motif.instances[0].motif_name
```

```
In [ ]: motif.instances[0].sequence_name
```

```
In [ ]: motif.instances[0].start
```

```
In [ ]: motif.instances[0].strand
```

```
In [ ]: motif.instances[0].length
```

```
In [ ]: pvalue = motif.instances[0].pvalue
        print("%5.3g" % pvalue)
```

MAST

15.2.3 TRANSFAC

TRANSFAC is a manually curated database of transcription factors, together with their genomic binding sites and DNA binding profiles @matys2003. While the file format used in the TRANSFAC database is nowadays also used by others, we will refer to it as the TRANSFAC file format.

A minimal file in the TRANSFAC format looks as follows:

```
ID  motif1
P0      A      C      G      T
01      1      2      2      0      S
02      2      1      2      0      R
03      3      0      1      1      A
04      0      5      0      0      C
05      5      0      0      0      A
06      0      0      4      1      G
07      0      1      4      0      G
08      0      0      0      5      T
09      0      0      5      0      G
10      0      1      2      2      K
11      0      2      0      3      Y
12      1      0      3      1      G
//
```

This file shows the frequency matrix of motif `motif1` of 12 nucleotides. In general, one file in the TRANSFAC format can contain multiple motifs. For example, this is the contents of the example TRANSFAC file `transfac.dat`:

```
VV  EXAMPLE January 15, 2013
XX
//
ID  motif1
```

```
P0      A       C       G       T
01      1       2       2       0       S
02      2       1       2       0       R
03      3       0       1       1       A
...
11      0       2       0       3       Y
12      1       0       3       1       G
//
ID  motif2
P0      A       C       G       T
01      2       1       2       0       R
02      1       2       2       0       S
...
09      0       0       0       5       T
10      0       2       0       3       Y
//
```

To parse a TRANSFAC file, use

```
In [ ]: handle = open("transfac.dat")
        record = motifs.parse(handle, "TRANSFAC")
        handle.close()
```

The overall version number, if available, is stored as `record.version`:

```
In [ ]: record.version
```

Each motif in `record` is in instance of the `Bio.motifs.transfac.Motif` class, which inherits both from the `Bio.motifs.Motif` class and from a Python dictionary. The dictionary uses the two-letter keys to store any additional information about the motif:

```
In [ ]: motif = record[0]
        motif.degenerate_consensus # Using the Bio.motifs.Motif method
```

```
In [ ]: motif['ID'] # Using motif as a dictionary
```

TRANSFAC files are typically much more elaborate than this example, containing lots of additional information about the motif. Table [table:transfaccodes] lists the two-letter field codes that are commonly found in TRANSFAC files:

[table:transfaccodes]

AC	Accession number
AS	Accession numbers, secondary
BA	Statistical basis
BF	Binding factors
BS	Factor binding sites underlying the matrix
CC	Comments
CO	Copyright notice
DE	Short factor description
DR	External databases
DT	Date created/updated
HC	Subfamilies
HP	Superfamilies
ID	Identifier
NA	Name of the binding factor
OC	Taxonomic classification
OS	Species/Taxon
OV	Older version
PV	Preferred version
TY	Type
XX	Empty line; these are not stored in the Record.

Table: Fields commonly found in TRANSFAC files

Each motif also has an attribute `.references` containing the references associated with the motif, using these two-letter keys:

RN	Reference number
RA	Reference authors
RL	Reference data
RT	Reference title
RX	PubMed ID

Table: Fields used to store references in TRANSFAC files

Printing the motifs writes them out in their native TRANSFAC format:

```
In [ ]: print(record)
```

You can export the motifs in the TRANSFAC format by capturing this output in a string and saving it in a file:

```
In [ ]: text = str(record)
        handle = open("mytransfacfile.dat", 'w')
        handle.write(text)
        handle.close()
```

15.3 Writing motifs

Speaking of exporting, let's look at export functions in general. We can use the `format` method to write the motif in the simple JASPAR `pfm` format:

```
In [ ]: print(arnt.format("pfm"))
```

Similarly, we can use `format` to write the motif in the JASPAR `jaspar` format:

```
In [ ]: print(arnt.format("jaspar"))
```

To write the motif in a TRANSFAC-like matrix format, use

```
In [ ]: print(m.format("transfac"))
```

To write out multiple motifs, you can use `motifs.write`. This function can be used regardless of whether the motifs originated from a TRANSFAC file. For example,

```
In [ ]: two_motifs = [arnt, srf]
        print(motifs.write(two_motifs, 'transfac'))
```

Or, to write multiple motifs in the `jaspar` format:

```
In [ ]: two_motifs = [arnt, mef2a]
        print(motifs.write(two_motifs, "jaspar"))
```

15.4 Position-Weight Matrices

The `.counts` attribute of a Motif object shows how often each nucleotide appeared at each position along the alignment. We can normalize this matrix by dividing by the number of instances in the alignment, resulting in the probability of each nucleotide at each position along the alignment. We refer to these probabilities as the position-weight matrix. However, beware that in the literature this term may also be used to refer to the position-specific scoring matrix, which we discuss below.

Usually, pseudocounts are added to each position before normalizing. This avoids overfitting of the position-weight matrix to the limited number of motif instances in the alignment, and can also prevent probabilities from becoming zero. To add a fixed pseudocount to all nucleotides at all positions, specify a number for the `pseudocounts` argument:

```
In [ ]: pwm = m.counts.normalize(pseudocounts=0.5)
        print(pwm)
```

Alternatively, `pseudocounts` can be a dictionary specifying the pseudocounts for each nucleotide. For example, as the GC content of the human genome is about 40%, you may want to choose the pseudocounts accordingly:

```
In [ ]: pwm = m.counts.normalize(pseudocounts={'A':0.6, 'C': 0.4, 'G': 0.4, 'T': 0.6})
        print(pwm)
```

The position-weight matrix has its own methods to calculate the consensus, anticonsensus, and degenerate consensus sequences:

```
In [ ]: pwm.consensus
```

```
In [ ]: pwm.anticonsensus
```

```
In [ ]: pwm.degenerate_consensus
```

Note that due to the pseudocounts, the degenerate consensus sequence calculated from the position-weight matrix is slightly different from the degenerate consensus sequence calculated from the instances in the motif:

```
In [ ]: m.degenerate_consensus
```

The reverse complement of the position-weight matrix can be calculated directly from the `pwm`:

```
In [ ]: rpwm = pwm.reverse_complement()
        print(rpwm)
```

15.5 Position-Specific Scoring Matrices

Using the background distribution and PWM with pseudo-counts added, it's easy to compute the log-odds ratios, telling us what are the log odds of a particular symbol to be coming from a motif against the background. We can use the `.log_odds()` method on the position-weight matrix:

```
In [ ]: pssm = pwm.log_odds()
        print(pssm)
```

Here we can see positive values for symbols more frequent in the motif than in the background and negative for symbols more frequent in the background. 0.0 means that it's equally likely to see a symbol in the background and in the motif.

This assumes that A, C, G, and T are equally likely in the background. To calculate the position-specific scoring matrix against a background with unequal probabilities for A, C, G, T, use the `background` argument. For example, against a background with a 40% GC content, use

```
In [ ]: background = {'A':0.3,'C':0.2,'G':0.2,'T':0.3}
        pssm = pwm.log_odds(background)
        print(pssm)
```

The maximum and minimum score obtainable from the PSSM are stored in the `.max` and `.min` properties:

```
In [ ]: print("%4.2f" % pssm.max)
```

```
In [ ]: print("%4.2f" % pssm.min)
```

The mean and standard deviation of the PSSM scores with respect to a specific background are calculated by the `.mean` and `.std` methods.

```
In [ ]: mean = pssm.mean(background)
        std = pssm.std(background)
        print("mean = %0.2f, standard deviation = %0.2f" % (mean, std))
```

A uniform background is used if `background` is not specified. The mean is particularly important, as its value is equal to the Kullback-Leibler divergence or relative entropy, and is a measure for the information content of the motif compared to the background. As in Biopython the base-2 logarithm is used in the calculation of the log-odds scores, the information content has units of bits.

The `.reverse_complement`, `.consensus`, `.anticonsensus`, and `.degenerate_consensus` methods can be applied directly to PSSM objects.

15.6 Searching for instances

The most frequent use for a motif is to find its instances in some sequence. For the sake of this section, we will use an artificial sequence like this:

```
In [ ]: test_seq=Seq("TACACTGCATTACAACCCAAGCATTA", m.alphabet)
        len(test_seq)
```

15.6.1 Searching for exact matches

The simplest way to find instances, is to look for exact matches of the true instances of the motif:

```
In [ ]: for pos, seq in m.instances.search(test_seq):
        print("%i %s" % (pos, seq))
```

We can do the same with the reverse complement (to find instances on the complementary strand):

```
In [ ]: for pos, seq in r.instances.search(test_seq):
        print("%i %s" % (pos, seq))
```

15.6.2 Searching for matches using the PSSM score

It's just as easy to look for positions, giving rise to high log-odds scores against our motif:

```
In [ ]: for position, score in pssm.search(test_seq, threshold=3.0):
        print("Position %d: score = %5.3f" % (position, score))
```

The negative positions refer to instances of the motif found on the reverse strand of the test sequence, and follow the Python convention on negative indices. Therefore, the instance of the motif at `pos` is located at `test_seq[pos:pos+len(m)]` both for positive and for negative values of `pos`.

You may notice the threshold parameter, here set arbitrarily to 3.0. This is in log_2, so we are now looking only for words, which are eight times more likely to occur under the motif model than in the background. The default threshold is 0.0, which selects everything that looks more like the motif than the background.

You can also calculate the scores at all positions along the sequence:

```
In [ ]: pssm.calculate(test_seq)
```

In general, this is the fastest way to calculate PSSM scores. The scores returned by `pssm.calculate` are for the forward strand only. To obtain the scores on the reverse strand, you can take the reverse complement of the PSSM:

```
In [ ]: rpssm = pssm.reverse_complement()
        rpssm.calculate(test_seq)
```

15.6.3 Selecting a score threshold

If you want to use a less arbitrary way of selecting thresholds, you can explore the distribution of PSSM scores. Since the space for a score distribution grows exponentially with motif length, we are using an approximation with a given precision to keep computation cost manageable:

```
In [ ]: distribution = pssm.distribution(background=background, precision=10**4)
```

The `distribution` object can be used to determine a number of different thresholds. We can specify the requested false-positive rate (probability of "finding" a motif instance in background generated sequence):

```
In [ ]: threshold = distribution.threshold_fpr(0.01)
        print("%5.3f" % threshold)
```

or the false-negative rate (probability of “not finding” an instance generated from the motif):

```
In [ ]: threshold = distribution.threshold_fnr(0.1)
        print("%5.3f" % threshold)
```

or a threshold (approximately) satisfying some relation between the false-positive rate and the false-negative rate ($\frac{\mathrm{fnr}}{\mathrm{fpr}} \simeq t$):

```
In [ ]: threshold = distribution.threshold_balanced(1000)
        print("%5.3f" % threshold)
```

or a threshold satisfying (roughly) the equality between the false-positive rate and the $-log$ of the information content (as used in patser software by Hertz and Stormo):

```
In [ ]: threshold = distribution.threshold_patser()
        print("%5.3f" % threshold)
```

For example, in case of our motif, you can get the threshold giving you exactly the same results (for this sequence) as searching for instances with balanced threshold with rate of 1000.

```
In [ ]: threshold = distribution.threshold_fpr(0.01)
        print("%5.3f" % threshold)
```

```
In [ ]: for position, score in pssm.search(test_seq, threshold=threshold):
        print("Position %d: score = %5.3f" % (position, score))
```

15.7 Each motif object has an associated Position-Specific Scoring Matrix

To facilitate searching for potential TFBSs using PSSMs, both the position-weight matrix and the position-specific scoring matrix are associated with each motif. Using the Arnt motif as an example:

```
In [ ]: from Bio import motifs
        with open("Arnt.sites") as handle:
        motif = motifs.read(handle, 'sites')
```

```
In [ ]: print(motif.counts)
```

```
In [ ]: print(motif.pwm)
```

```
In [ ]: print(motif.pssm)
```

The negative infinities appear here because the corresponding entry in the frequency matrix is 0, and we are using zero pseudocounts by default:

```
In [ ]: for letter in "ACGT":
        print("%s: %4.2f" % (letter, motif.pseudocounts[letter]))
```

If you change the `.pseudocounts` attribute, the position-frequency matrix and the position-specific scoring matrix are recalculated automatically:

```
In [ ]: motif.pseudocounts = 3.0
        for letter in "ACGT":
        print("%s: %4.2f" % (letter, motif.pseudocounts[letter]))
```

```
In [ ]: print(motif.pwm)
```

```
In [ ]: print(motif.pssm)
```

You can also set the `.pseudocounts` to a dictionary over the four nucleotides if you want to use different pseudocounts for them. Setting `motif.pseudocounts` to `None` resets it to its default value of zero.

The position-specific scoring matrix depends on the background distribution, which is uniform by default:

```
In [ ]: for letter in "ACGT":
        print("%s: %4.2f" % (letter, motif.background[letter]))
```

Again, if you modify the background distribution, the position-specific scoring matrix is recalculated:

```
In [ ]: motif.background = {'A': 0.2, 'C': 0.3, 'G': 0.3, 'T': 0.2}
        print(motif.pssm)
```

Setting `motif.background` to `None` resets it to a uniform distribution:

```
In [ ]: motif.background = None
        for letter in "ACGT":
        print("%s: %4.2f" % (letter, motif.background[letter]))
```

If you set `motif.background` equal to a single value, it will be interpreted as the GC content:

```
In [ ]: motif.background = 0.8
        for letter in "ACGT":
        print("%s: %4.2f" % (letter, motif.background[letter]))
```

Note that you can now calculate the mean of the PSSM scores over the background against which it was computed:

```
In [ ]: print("%f" % motif.pssm.mean(motif.background))
```

as well as its standard deviation:

```
In [ ]: print("%f" % motif.pssm.std(motif.background))
```

and its distribution:

```
In [ ]: distribution = motif.pssm.distribution(background=motif.background)
        threshold = distribution.threshold_fpr(0.01)
        print("%f" % threshold)
```

Note that the position-weight matrix and the position-specific scoring matrix are recalculated each time you call `motif.pwm` or `motif.pssm`, respectively. If speed is an issue and you want to use the PWM or PSSM repeatedly, you can save them as a variable, as in

```
In [ ]: pssm = motif.pssm
```

15.8 Comparing motifs

Once we have more than one motif, we might want to compare them.

Before we start comparing motifs, I should point out that motif boundaries are usually quite arbitrary. This means we often need to compare motifs of different lengths, so comparison needs to involve some kind of alignment. This means we have to take into account two things:

- alignment of motifs
- some function to compare aligned motifs

To align the motifs, we use ungapped alignment of PSSMs and substitute zeros for any missing columns at the beginning and end of the matrices. This means that effectively we are using the background distribution for columns missing from the PSSM. The distance function then returns the minimal distance between motifs, as well as the corresponding offset in their alignment.

To give an example, let us first load another motif, which is similar to our test motif `m`:

```
In [ ]: with open("REB1.pfm") as handle:
            m_reb1 = motifs.read(handle, "pfm")
```

```
In [ ]: m_reb1.consensus
```

```
In [ ]: print(m_reb1.counts)
```

To make the motifs comparable, we choose the same values for the pseudocounts and the background distribution as our motif `m`:

```
In [ ]: m_reb1.pseudocounts = {'A':0.6, 'C': 0.4, 'G': 0.4, 'T': 0.6}
        m_reb1.background = {'A':0.3,'C':0.2,'G':0.2,'T':0.3}
        pssm_reb1 = m_reb1.pssm
        print(pssm_reb1)
```

We'll compare these motifs using the Pearson correlation. Since we want it to resemble a distance measure, we actually take $1 - r$, where r is the Pearson correlation coefficient (PCC):

```
In [ ]: distance, offset = pssm.dist_pearson(pssm_reb1)
        print("distance = %5.3g" % distance)
```

```
In [ ]: print(offset)
```

This means that the best PCC between motif `m` and `m_reb1` is obtained with the following alignment:

```
m:      bbTACGCbb
m_reb1: GTTACCCGG
```

where `b` stands for background distribution. The PCC itself is roughly $1 - 0.239 = 0.761$.

15.9 *De novo* motif finding

Currently, Biopython has only limited support for *de novo* motif finding. Namely, we support running and parsing of AlignAce and MEME. Since the number of motif finding tools is growing rapidly, contributions of new parsers are welcome.

15.9.1 MEME

Let's assume, you have run MEME on sequences of your choice with your favorite parameters and saved the output in the file `meme.out`. You can retrieve the motifs reported by MEME by running the following piece of code:

```
In [ ]: from Bio import motifs
        with open("meme.out") as handle:
        motifsM = motifs.parse(handle, "meme")
```

```
In [ ]: motifsM
```

Besides the most wanted list of motifs, the result object contains more useful information, accessible through properties with self-explanatory names:

- `.alphabet`
- `.datafile`
- `.sequence_names`
- `.version`
- `.command`

The motifs returned by the MEME Parser can be treated exactly like regular Motif objects (with instances), they also provide some extra functionality, by adding additional information about the instances.

```
In [ ]: motifsM[0].consensus
```

```
In [ ]: motifsM[0].instances[0].sequence_name
```

```
In [ ]: motifsM[0].instances[0].start
```

```
In [ ]: motifsM[0].instances[0].strand
```

```
In [ ]: motifsM[0].instances[0].pvalue
```

15.9.2 AlignAce

We can do very similar things with the AlignACE program. Assume, you have your output in the file `alignace.out`. You can parse your output with the following code:

```
In [ ]: from Bio import motifs
        with open("alignace.out") as handle:
        motifsA = motifs.parse(handle, "alignace")
```

Again, your motifs behave as they should:

```
In [ ]: motifsA[0].consensus
```

In fact you can even see, that AlignAce found a very similar motif as MEME. It is just a longer version of a reverse complement of the MEME motif:

```
In [ ]: motifsM[0].reverse_complement().consensus
```

If you have AlignAce installed on the same machine, you can also run it directly from Biopython. A short example of how this can be done is shown below (other parameters can be specified as keyword parameters):

```
In [ ]: command="/opt/bin/AlignACE"
        input_file="test.fa"
        from Bio.motifs.applications import AlignAceCommandline
        cmd = AlignAceCommandline(cmd=command, input=input_file, gcback=0.6, numcols=10)
        stdout, stderr= cmd()
```

Since AlignAce prints all of its output to standard output, you can get to your motifs by parsing the first part of the result:

```
In [ ]: motifs = motifs.parse(stdout, "alignace")
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 16

Cluster analysis

Cluster analysis is the grouping of items into clusters based on the similarity of the items to each other. In bioinformatics, clustering is widely used in gene expression data analysis to find groups of genes with similar gene expression profiles. This may identify functionally related genes, as well as suggest the function of presently unknown genes.

The Biopython module `Bio.Cluster` provides commonly used clustering algorithms and was designed with the application to gene expression data in mind. However, this module can also be used for cluster analysis of other types of data. `Bio.Cluster` and the underlying C Clustering Library is described by De Hoon *et al.* @dehoon2004.

The following four clustering approaches are implemented in `Bio.Cluster`:

- Hierarchical clustering (pairwise centroid-, single-, complete-, and average-linkage);
- k-means, k-medians, and k-medoids clustering;
- Self-Organizing Maps;
- Principal Component Analysis.

16.1 Data representation

The data to be clustered are represented by a $n \times m$ Numerical Python array `data`. Within the context of gene expression data clustering, typically the rows correspond to different genes whereas the columns correspond to different experimental conditions. The clustering algorithms in `Bio.Cluster` can be applied both to rows (genes) and to columns (experiments).

16.2 Missing values

Often in microarray experiments, some of the data values are missing, which is indicated by an additional $n \times m$ Numerical Python integer array `mask`. If `mask[i,j]==0`, then `data[i,j]` is missing and is ignored in the analysis.

16.3 Random number generator

The k-means/medians/medoids clustering algorithms and Self-Organizing Maps (SOMs) include the use of a random number generator. The uniform random number generator in `Bio.Cluster` is based on the algorithm by L'Ecuyer @lecuyer1988, while random numbers following the binomial distribution are generated using the BTPE algorithm by Kachitvichyanukul and Schmeiser @kachitvichyanukul1988. The random number generator is initialized automatically during its first call. As this random number generator uses a combination of two multiplicative linear congruential generators, two (integer) seeds are needed for initialization, for which we use the system-supplied random number generator `rand` (in the C standard library). We initialize this generator by calling `srand` with the epoch time in seconds, and use the first two random numbers generated by `rand` as seeds for the uniform random number generator in `Bio.Cluster`.

16.3.1 Distance functions

In order to cluster items into groups based on their similarity, we should first define what exactly we mean by *similar*. `Bio.Cluster` provides eight distance functions, indicated by a single character, to measure similarity, or conversely, distance:

- `'e'`: Euclidean distance;
- `'b'`: City-block distance.
- `'c'`: Pearson correlation coefficient;
- `'a'`: Absolute value of the Pearson correlation coefficient;
- `'u'`: Uncentered Pearson correlation (equivalent to the cosine of the angle between two data vectors);
- `'x'`: Absolute uncentered Pearson correlation;
- `'s'`: Spearman's rank correlation;
- `'k'`: Kendall's τ.

The first two are true distance functions that satisfy the triangle inequality:

$$d\left(\underline{u},\underline{v}\right) \leq d\left(\underline{u},\underline{w}\right) + d\left(\underline{w},\underline{v}\right) \text{ for all } \underline{u}, \underline{v}, \underline{w},$$
$$and are therefore refered to as * metrics * . In everyday language, this$$

means that the shortest distance between two points is a straight line.

The remaining six distance measures are related to the correlation coefficient, where the distance d is defined in terms of the correlation r by $d = 1 - r$. Note that these distance functions are *semi-metrics* that do not satisfy the triangle inequality. For example, for

$$\underline{u} = \left(1, 0, -1\right);$$

$$\underline{v} = \left(1, 1, 0\right);$$

$$\underline{w} = \left(0, 1, 1\right);$$
$$we find a Pearson distance$$

$d\left(\underline{u},\underline{w}\right) = 1.8660$, while $d\left(\underline{u},\underline{v}\right) + d\left(\underline{v},\underline{w}\right) = 1.6340$.

16.4 Euclidean distance

In `Bio.Cluster`, we define the Euclidean distance as

$$d = \frac{1}{n} \sum_{i=1}^{n} \left(x_i - y_i\right)^{2}.$$

Only those terms are included in the summation for which both x_i and y_i are present, and the denominator n is chosen accordingly. As the expression data x_i and y_i are subtracted directly from each other, we should make sure that the expression data are properly normalized when using the Euclidean distance.

16.5 City-block distance

The city-block distance, alternatively known as the Manhattan distance, is related to the Euclidean distance. Whereas the Euclidean distance corresponds to the length of the shortest path between two points, the city-block distance is the sum of distances along each dimension. As gene expression data tend to have missing values, in `Bio.Cluster` we define the city-block distance as the sum of distances divided by the number of dimensions:

$$d = \frac{1}{n} \sum_{i=1}^{n} \left|x_i - y_i\right|.$$

This is equal to the distance you would have to walk between two points in a city, where you have to walk along city blocks. As for the Euclidean distance, the expression data are subtracted directly from each other, and we should therefore make sure that they are properly normalized.

16.6 The Pearson correlation coefficient

The Pearson correlation coefficient is defined as

$$r = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{\sigma_x} \right) \left(\frac{y_i - \bar{y}}{\sigma_y} \right),$$

$$in which : math : `\bar{x}, \bar{y}` are the sample mean of : math : `x` and$$

y respectively, and σ_x, σ_y are the sample standard deviation of x and y. The Pearson correlation coefficient is a measure for how well a straight line can be fitted to a scatterplot of x and y. If all the points in the scatterplot lie on a straight line, the Pearson correlation coefficient is either +1 or -1, depending on whether the slope of line is positive or negative. If the Pearson correlation coefficient is equal to zero, there is no correlation between x and y.

The *Pearson distance* is then defined as

$$d_{\mathrm{P}} \equiv 1 - r.$$

As the Pearson correlation coefficient lies between -1 and 1, the Pearson distance lies between 0 and 2.

16.7 Absolute Pearson correlation

By taking the absolute value of the Pearson correlation, we find a number between 0 and 1. If the absolute value is 1, all the points in the scatter plot lie on a straight line with either a positive or a negative slope. If the absolute value is equal to zero, there is no correlation between x and y.

The corresponding distance is defined as

$$d_{\mathrm{A}} \equiv 1 - \left|r\right|,$$

where r is the Pearson correlation coefficient. As the absolute value of the Pearson correlation coefficient lies between 0 and 1, the corresponding distance lies between 0 and 1 as well.

In the context of gene expression experiments, the absolute correlation is equal to 1 if the gene expression profiles of two genes are either exactly the same or exactly opposite. The absolute correlation coefficient should therefore be used with care.

16.8 Uncentered correlation (cosine of the angle)

In some cases, it may be preferable to use the *uncentered correlation* instead of the regular Pearson correlation coefficient. The uncentered correlation is defined as

$$r_{\mathrm{U}} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{x_i}{\sigma_x^{(0)}}\right) \left(\frac{y_i}{\sigma_y^{(0)}}\right),$$

$$where$$

$$\sigma_x^{(0)} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2};$$

$$\sigma_y^{(0)} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} y_i^2}.$$

$$This is the same expression as for the regular Pearson correlation$$

coefficient, except that the sample means $\bar{x}, \bar{y}$ are set equal to zero. The uncentered correlation may be appropriate if there is a zero reference state. For instance, in the case of gene expression data given in terms of log-ratios, a log-ratio equal to zero corresponds to the green and red signal being equal, which means that the experimental manipulation did not affect the gene expression.

The distance corresponding to the uncentered correlation coefficient is defined as

$$d_{\mathrm{U}} \equiv 1 - r_{\mathrm{U}},$$

where r_{U} is the uncentered correlation. As the uncentered correlation coefficient lies between -1 and 1, the corresponding distance lies between 0 and 2.

The uncentered correlation is equal to the cosine of the angle of the two data vectors in n-dimensional space, and is often referred to as such.

16.9 Absolute uncentered correlation

As for the regular Pearson correlation, we can define a distance measure using the absolute value of the uncentered correlation:

$$d_{\mathrm{AU}} \equiv 1 - \left|r_{\mathrm{U}}\right|,$$

where r_{U} is the uncentered correlation coefficient. As the absolute value of the uncentered correlation coefficient lies between 0 and 1, the corresponding distance lies between 0 and 1 as well.

Geometrically, the absolute value of the uncentered correlation is equal to the cosine between the supporting lines of the two data vectors (i.e., the angle without taking the direction of the vectors into consideration).

16.10 Spearman rank correlation

The Spearman rank correlation is an example of a non-parametric similarity measure, and tends to be more robust against outliers than the Pearson correlation.

To calculate the Spearman rank correlation, we replace each data value by their rank if we would order the data in each vector by their value. We then calculate the Pearson correlation between the two rank vectors instead of the data vectors.

As in the case of the Pearson correlation, we can define a distance measure corresponding to the Spearman rank correlation as

$$d_{\mathrm{S}} \equiv 1 - r_{\mathrm{S}},$$

where r_{S} is the Spearman rank correlation.

16.11 Kendall's τ

Kendall's τ is another example of a non-parametric similarity measure. It is similar to the Spearman rank correlation, but instead of the ranks themselves only the relative ranks are used to calculate τ (see Snedecor & Cochran @snedecor1989).

We can define a distance measure corresponding to Kendall's τ as

$$d_{\mathrm{K}} \equiv 1 - \tau.$$

As Kendall's τ is always between -1 and 1, the corresponding distance will be between 0 and 2.

16.12 Weighting

For most of the distance functions available in `Bio.Cluster`, a weight vector can be applied. The weight vector contains weights for the items in the data vector. If the weight for item i is w_i, then that item is treated as if it occurred w_i times in the data. The weight do not have to be integers. For the Spearman rank correlation and Kendall's τ, weights do not have a well-defined meaning and are therefore not implemented.

16.13 Calculating the distance matrix

The distance matrix is a square matrix with all pairwise distances between the items in `data`, and can be calculated by the function `distancematrix` in the `Bio.Cluster` module:

```
In [1]: from Bio.Cluster import distancematrix
        matrix = distancematrix(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-42e78a39e140> in <module>()
      1 from Bio.Cluster import distancematrix
----> 2 matrix = distancematrix(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data` (required)
  Array containing the data for the items.
- `mask` (default: `None`)
  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.
- `weight` (default: `None`)
  The weights to be used when calculating distances. If `weight==None`, then equal weights are assumed.
- `transpose` (default: `0`)
  Determines if the distances between the rows of `data` are to be calculated (`transpose==0`), or between the columns of `data` (`transpose==1`).
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).

To save memory, the distance matrix is returned as a list of 1D arrays. The number of columns in each row is equal to the row number. Hence, the first row has zero elements. An example of the return value is

```
[array([]),
 array([1.]),
 array([7., 3.]),
 array([4., 2., 6.])]
```

This corresponds to the distance matrix

$$\begin{pmatrix} 0 & 1 & 7 & 4 \\ 1 & 0 & 3 & 2 \\ 7 & 3 & 0 & 6 \\ 4 & 2 & 6 & 0 \end{pmatrix}.$$

16.13.1 Calculating cluster properties

16.14 Calculating the cluster centroids

The centroid of a cluster can be defined either as the mean or as the median of each dimension over all cluster items. The function `clustercentroids` in `Bio.Cluster` can be used to calculate either:

```
In [2]: from Bio.Cluster import clustercentroids
        cdata, cmask = clustercentroids(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-866b8da2b76e> in <module>()
      1 from Bio.Cluster import clustercentroids
----> 2 cdata, cmask = clustercentroids(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data` (required)
  Array containing the data for the items.
- `mask` (default: `None`)
  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.

- `clusterid` (default: `None`)

  Vector of integers showing to which cluster each item belongs. If `clusterid` is `None`, then all items are assumed to belong to the same cluster.

- `method` (default: `'a'`)

  Specifies whether the arithmetic mean (`method=='a'`) or the median (`method=='m'`) is used to calculate the cluster center.

- `transpose` (default: `0`)

  Determines if the centroids of the rows of `data` are to be calculated (`transpose==0`), or the centroids of the columns of `data` (`transpose==1`).

This function returns the tuple `(cdata, cmask)`. The centroid data are stored in the 2D Numerical Python array `cdata`, with missing data indicated by the 2D Numerical Python integer array `cmask`. The dimensions of these arrays are (number of clusters, number of columns) if `transpose` is `0`, or (number of rows, number of clusters) if `transpose` is `1`. Each row (if `transpose` is `0`) or column (if `transpose` is `1`) contains the averaged data corresponding to the centroid of each cluster.

16.15 Calculating the distance between clusters

Given a distance function between *items*, we can define the distance between two *clusters* in several ways. The distance between the arithmetic means of the two clusters is used in pairwise centroid-linkage clustering and in k-means clustering. In k-medoids clustering, the distance between the medians of the two clusters is used instead. The shortest pairwise distance between items of the two clusters is used in pairwise single-linkage clustering, while the longest pairwise distance is used in pairwise maximum-linkage clustering. In pairwise average-linkage clustering, the distance between two clusters is defined as the average over the pairwise distances.

To calculate the distance between two clusters, use

```
In [3]: from Bio.Cluster import clusterdistance
        distance = clusterdistance(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-7af12629a96b> in <module>()
      1 from Bio.Cluster import clusterdistance
----> 2 distance = clusterdistance(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data` (required)

  Array containing the data for the items.

- `mask` (default: `None`)

  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.

- `weight` (default: `None`)

  The weights to be used when calculating distances. If `weight==None`, then equal weights are assumed.

- `index1` (default: `0`)

  A list containing the indices of the items belonging to the first cluster. A cluster containing only one item i can be represented either as a list `[i]`, or as an integer `i`.

- `index2` (default: `0`)

A list containing the indices of the items belonging to the second cluster. A cluster containing only one items i can be represented either as a list `[i]`, or as an integer `i`.

- `method` (default: `'a'`)
  Specifies how the distance between clusters is defined:
  - `'a'`: Distance between the two cluster centroids (arithmetic mean);
  - `'m'`: Distance between the two cluster centroids (median);
  - `'s'`: Shortest pairwise distance between items in the two clusters;
  - `'x'`: Longest pairwise distance between items in the two clusters;
  - `'v'`: Average over the pairwise distances between items in the two clusters.
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).
- `transpose` (default: `0`)
  If `transpose==0`, calculate the distance between the rows of `data`. If `transpose==1`, calculate the distance between the columns of `data`.

16.15.1 Partitioning algorithms

Partitioning algorithms divide items into k clusters such that the sum of distances over the items to their cluster centers is minimal. The number of clusters k is specified by the user. Three partitioning algorithms are available in `Bio.Cluster`:

- k-means clustering
- k-medians clustering
- k-medoids clustering

These algorithms differ in how the cluster center is defined. In k-means clustering, the cluster center is defined as the mean data vector averaged over all items in the cluster. Instead of the mean, in k-medians clustering the median is calculated for each dimension in the data vector. Finally, in k-medoids clustering the cluster center is defined as the item which has the smallest sum of distances to the other items in the cluster. This clustering algorithm is suitable for cases in which the distance matrix is known but the original data matrix is not available, for example when clustering proteins based on their structural similarity.

The expectation-maximization (EM) algorithm is used to find this partitioning into k groups. In the initialization of the EM algorithm, we randomly assign items to clusters. To ensure that no empty clusters are produced, we use the binomial distribution to randomly choose the number of items in each cluster to be one or more. We then randomly permute the cluster assignments to items such that each item has an equal probability to be in any cluster. Each cluster is thus guaranteed to contain at least one item.

We then iterate:

- Calculate the centroid of each cluster, defined as either the mean, the median, or the medoid of the cluster;
- Calculate the distances of each item to the cluster centers;
- For each item, determine which cluster centroid is closest;
- Reassign each item to its closest cluster, or stop the iteration if no further item reassignments take place.

To avoid clusters becoming empty during the iteration, in k-means and k-medians clustering the algorithm keeps track of the number of items in each cluster, and prohibits the last remaining item in a cluster from being reassigned to a different cluster. For k-medoids clustering, such a check is not needed, as the item that functions as the cluster centroid has a zero distance to itself, and will therefore never be closer to a different cluster.

As the initial assignment of items to clusters is done randomly, usually a different clustering solution is found each time the EM algorithm is executed. To find the optimal clustering solution, the k-means algorithm is repeated many times, each time starting from a different initial random clustering. The sum of distances of the items to their cluster center is saved for each run, and the solution with the smallest value of this sum will be returned as the overall clustering solution.

How often the EM algorithm should be run depends on the number of items being clustered. As a rule of thumb, we can consider how often the optimal solution was found; this number is returned by the partitioning algorithms as implemented in this library. If the optimal solution was found many times, it is unlikely that better solutions exist than the one that was found. However, if the optimal solution was found only once, there may well be other solutions with a smaller within-cluster sum of distances. If the number of items is large (more than several hundreds), it may be difficult to find the globally optimal solution.

The EM algorithm terminates when no further reassignments take place. We noticed that for some sets of initial cluster assignments, the EM algorithm fails to converge due to the same clustering solution reappearing periodically after a small number of iteration steps. We therefore check for the occurrence of such periodic solutions during the iteration. After a given number of iteration steps, the current clustering result is saved as a reference. By comparing the clustering result after each subsequent iteration step to the reference state, we can determine if a previously encountered clustering result is found. In such a case, the iteration is halted. If after a given number of iterations the reference state has not yet been encountered, the current clustering solution is saved to be used as the new reference state. Initially, ten iteration steps are executed before resaving the reference state. This number of iteration steps is doubled each time, to ensure that periodic behavior with longer periods can also be detected.

16.16 k-means and k-medians

The k-means and k-medians algorithms are implemented as the function `kcluster` in `Bio.Cluster`:

```
In [4]: from Bio.Cluster import kcluster
        clusterid, error, nfound = kcluster(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-c0b4d185df6f> in <module>()
      1 from Bio.Cluster import kcluster
----> 2 clusterid, error, nfound = kcluster(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data` (required)
  Array containing the data for the items.
- `nclusters` (default: `2`)
  The number of clusters k.
- `mask` (default: `None`)
  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.
- `weight` (default: `None`)
  The weights to be used when calculating distances. If `weight==None`, then equal weights are assumed.
- `transpose` (default: `0`)
  Determines if rows (`transpose` is `0`) or columns (`transpose` is `1`) are to be clustered.
- `npass` (default: `1`)

The number of times the k-means/-medians clustering algorithm is performed, each time with a different (random) initial condition. If `initialid` is given, the value of `npass` is ignored and the clustering algorithm is run only once, as it behaves deterministically in that case.

- `method` (default: `a`)
  describes how the center of a cluster is found:
  - `method=='a'`: arithmetic mean (k-means clustering);
  - `method=='m'`: median (k-medians clustering).

  For other values of `method`, the arithmetic mean is used.
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]). Whereas all eight distance measures are accepted by `kcluster`, from a theoretical viewpoint it is best to use the Euclidean distance for the k-means algorithm, and the city-block distance for k-medians.
- `initialid` (default: `None`)
  Specifies the initial clustering to be used for the EM algorithm. If `initialid==None`, then a different random initial clustering is used for each of the `npass` runs of the EM algorithm. If `initialid` is not `None`, then it should be equal to a 1D array containing the cluster number (between `0` and `nclusters-1`) for each item. Each cluster should contain at least one item. With the initial clustering specified, the EM algorithm is deterministic.

This function returns a tuple `(clusterid, error, nfound)`, where `clusterid` is an integer array containing the number of the cluster to which each row or cluster was assigned, `error` is the within-cluster sum of distances for the optimal clustering solution, and `nfound` is the number of times this optimal solution was found.

16.17 k-medoids clustering

The `kmedoids` routine performs k-medoids clustering on a given set of items, using the distance matrix and the number of clusters passed by the user:

```
In [5]: from Bio.Cluster import kmedoids
        clusterid, error, nfound = kmedoids(distance)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-6cf17eb1aebc> in <module>()
      1 from Bio.Cluster import kmedoids
----> 2 clusterid, error, nfound = kmedoids(distance)

NameError: name 'distance' is not defined
```

where the following arguments are defined: , nclusters=2, npass=1, initialid=None)|

- `distance` (required)
  The matrix containing the distances between the items; this matrix can be specified in three ways:
  - as a 2D Numerical Python array (in which only the left-lower part of the array will be accessed):

```
distance = array([[0.0, 1.1, 2.3],
                  [1.1, 0.0, 4.5],
                  [2.3, 4.5, 0.0]])
```

```
-   as a 1D Numerical Python array containing consecutively the
    distances in the left-lower part of the distance matrix:
```

```
distance = array([1.1, 2.3, 4.5])
```

```
-   as a list containing the rows of the left-lower part of the
    distance matrix:
```

```
distance = [array([]),
                    array([1.1]),
                    array([2.3, 4.5])
                   ]
```

```
These three expressions correspond to the same distance matrix.
```

- `nclusters` (default: `2`)
  The number of clusters k.
- `npass` (default: `1`)
  The number of times the k-medoids clustering algorithm is performed, each time with a different (random) initial condition. If `initialid` is given, the value of `npass` is ignored, as the clustering algorithm behaves deterministically in that case.
- `initialid` (default: `None`)
  Specifies the initial clustering to be used for the EM algorithm. If `initialid==None`, then a different random initial clustering is used for each of the `npass` runs of the EM algorithm. If `initialid` is not `None`, then it should be equal to a 1D array containing the cluster number (between `0` and `nclusters-1`) for each item. Each cluster should contain at least one item. With the initial clustering specified, the EM algorithm is deterministic.

This function returns a tuple `(clusterid, error, nfound)`, where `clusterid` is an array containing the number of the cluster to which each item was assigned, `error` is the within-cluster sum of distances for the optimal k-medoids clustering solution, and `nfound` is the number of times the optimal solution was found. Note that the cluster number in `clusterid` is defined as the item number of the item representing the cluster centroid.

16.17.1 Hierarchical clustering

Hierarchical clustering methods are inherently different from the k-means clustering method. In hierarchical clustering, the similarity in the expression profile between genes or experimental conditions are represented in the form of a tree structure. This tree structure can be shown graphically by programs such as Treeview and Java Treeview, which has contributed to the popularity of hierarchical clustering in the analysis of gene expression data.

The first step in hierarchical clustering is to calculate the distance matrix, specifying all the distances between the items to be clustered. Next, we create a node by joining the two closest items. Subsequent nodes are created by pairwise joining of items or nodes based on the distance between them, until all items belong to the same node. A tree structure can then be created by retracing which items and nodes were merged. Unlike the EM algorithm, which is used in k-means clustering, the complete process of hierarchical clustering is deterministic.

Several flavors of hierarchical clustering exist, which differ in how the distance between subnodes is defined in terms of their members. In `Bio.Cluster`, pairwise single, maximum, average, and centroid linkage are available.

- In pairwise single-linkage clustering, the distance between two nodes is defined as the shortest distance among the pairwise distances between the members of the two nodes.
- In pairwise maximum-linkage clustering, alternatively known as pairwise complete-linkage clustering, the distance between two nodes is defined as the longest distance among the pairwise distances between the members of the two nodes.

- In pairwise average-linkage clustering, the distance between two nodes is defined as the average over all pairwise distances between the items of the two nodes.
- In pairwise centroid-linkage clustering, the distance between two nodes is defined as the distance between their centroids. The centroids are calculated by taking the mean over all the items in a cluster. As the distance from each newly formed node to existing nodes and items need to be calculated at each step, the computing time of pairwise centroid-linkage clustering may be significantly longer than for the other hierarchical clustering methods. Another peculiarity is that (for a distance measure based on the Pearson correlation), the distances do not necessarily increase when going up in the clustering tree, and may even decrease. This is caused by an inconsistency between the centroid calculation and the distance calculation when using the Pearson correlation: Whereas the Pearson correlation effectively normalizes the data for the distance calculation, no such normalization occurs for the centroid calculation.

For pairwise single-, complete-, and average-linkage clustering, the distance between two nodes can be found directly from the distances between the individual items. Therefore, the clustering algorithm does not need access to the original gene expression data, once the distance matrix is known. For pairwise centroid-linkage clustering, however, the centroids of newly formed subnodes can only be calculated from the original data and not from the distance matrix.

The implementation of pairwise single-linkage hierarchical clustering is based on the SLINK algorithm (R. Sibson, 1973), which is much faster and more memory-efficient than a straightforward implementation of pairwise single-linkage clustering. The clustering result produced by this algorithm is identical to the clustering solution found by the conventional single-linkage algorithm. The single-linkage hierarchical clustering algorithm implemented in this library can be used to cluster large gene expression data sets, for which conventional hierarchical clustering algorithms fail due to excessive memory requirements and running time.

16.18 Representing a hierarchical clustering solution

The result of hierarchical clustering consists of a tree of nodes, in which each node joins two items or subnodes. Usually, we are not only interested in which items or subnodes are joined at each node, but also in their similarity (or distance) as they are joined. To store one node in the hierarchical clustering tree, we make use of the class `Node`, which defined in `Bio.Cluster`. An instance of `Node` has three attributes:

- `left`
- `right`
- `distance`

Here, `left` and `right` are integers referring to the two items or subnodes that are joined at this node, and `distance` is the distance between them. The items being clustered are numbered from 0 to $(\text{number of items} - 1)$, while clusters are numbered from -1 to $-(\text{number of items} - 1)$. Note that the number of nodes is one less than the number of items.

To create a new `Node` object, we need to specify `left` and `right`; `distance` is optional.

```
In [6]: from Bio.Cluster import Node
        Node(2, 3)
```

```
Out[6]: (2, 3): 0
```

```
In [7]: Node(2, 3, 0.91)
```

```
Out[7]: (2, 3): 0.91
```

The attributes `left`, `right`, and `distance` of an existing `Node` object can be modified directly:

```
In [8]: node = Node(4, 5)
        node.left = 6
        node.right = 2
        node.distance = 0.73
        node
```

```
Out[8]: (6, 2): 0.73
```

An error is raised if `left` and `right` are not integers, or if `distance` cannot be converted to a floating-point value.

The Python class `Tree` represents a full hierarchical clustering solution. A `Tree` object can be created from a list of `Node` objects:

```
In [9]: from Bio.Cluster import Node, Tree
        nodes = [Node(1, 2, 0.2), Node(0, 3, 0.5), Node(-2, 4, 0.6), Node(-1, -3, 0.9)]
        tree = Tree(nodes)
        print(tree)
```

```
(1, 2): 0.2
(0, 3): 0.5
(-2, 4): 0.6
(-1, -3): 0.9
```

The `Tree` initializer checks if the list of nodes is a valid hierarchical clustering result:

```
In [11]: nodes = [Node(1, 2, 0.2), Node(0, 2, 0.5)]
         try:
             Tree(nodes)
             raise Exception("Should not arrive here")
         except ValueError:
             print("This tree is problematic")
```

```
This tree is problematic
```

Individual nodes in a `Tree` object can be accessed using square brackets:

```
In [12]: nodes = [Node(1, 2, 0.2), Node(0, -1, 0.5)]
         tree = Tree(nodes)
         tree[0]
```

```
Out[12]: (1, 2): 0.2
```

```
In [13]: tree[1]
```

```
Out[13]: (0, -1): 0.5
```

```
In [14]: tree[-1]
```

```
Out[14]: (0, -1): 0.5
```

As a `Tree` object is read-only, we cannot change individual nodes in a `Tree` object. However, we can convert the tree to a list of nodes, modify this list, and create a new tree from this list:

```
In [15]: tree = Tree([Node(1, 2, 0.1), Node(0, -1, 0.5), Node(-2, 3, 0.9)])
         print(tree)
```

```
(1, 2): 0.1
(0, -1): 0.5
(-2, 3): 0.9
```

```
In [16]: nodes = tree[:]
         nodes[0] = Node(0, 1, 0.2)
         nodes[1].left = 2
         tree = Tree(nodes)
         print(tree)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-b8ceb34eb030> in <module>()
----> 1 nodes = tree[:]
      2 nodes[0] = Node(0, 1, 0.2)
      3 nodes[1].left = 2
      4 tree = Tree(nodes)
      5 print(tree)

TypeError: sequence index must be integer, not 'slice'
```

This guarantees that any `Tree` object is always well-formed.

To display a hierarchical clustering solution with visualization programs such as Java Treeview, it is better to scale all node distances such that they are between zero and one. This can be accomplished by calling the `scale` method on an existing `Tree` object:

```
In [12]: tree.scale()
```

This method takes no arguments, and returns `None`.

After hierarchical clustering, the items can be grouped into k clusters based on the tree structure stored in the `Tree` object by cutting the tree:

```
In [13]: clusterid = tree.cut(nclusters=1)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-633abf1755ad> in <module>()
----> 1 clusterid = tree.cut(nclusters=1)

TypeError: cut() takes no keyword arguments
```

where `nclusters` (defaulting to `1`) is the desired number of clusters k. This method ignores the top $k-1$ linking events in the tree structure, resulting in k separated clusters of items. The number of clusters k should be positive, and less than or equal to the number of items. This method returns an array `clusterid` containing the number of the cluster to which each item is assigned.

16.19 Performing hierarchical clustering

To perform hierarchical clustering, use the `treecluster` function in `Bio.Cluster`.

```
In [14]: from Bio.Cluster import treecluster
         tree = treecluster(data)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-1233e37a5203> in <module>()
      1 from Bio.Cluster import treecluster
----> 2 tree = treecluster(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data`
  Array containing the data for the items.
- `mask` (default: `None`)
  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.
- `weight` (default: `None`)
  The weights to be used when calculating distances. If `weight==None`, then equal weights are assumed.
- `transpose` (default: `0`)
  Determines if rows (`transpose==0`) or columns (`transpose==1`) are to be clustered.

- `method` (default: `'m'`)
  defines the linkage method to be used:
  - `method=='s'`: pairwise single-linkage clustering
  - `method=='m'`: pairwise maximum- (or complete-) linkage clustering
  - `method=='c'`: pairwise centroid-linkage clustering
  - `method=='a'`: pairwise average-linkage clustering
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).

To apply hierarchical clustering on a precalculated distance matrix, specify the `distancematrix` argument when calling `treecluster` function instead of the `data` argument:

```
In [17]: from Bio.Cluster import treecluster
         tree = treecluster(distancematrix=distance)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-f938aacc6e43> in <module>()
      1 from Bio.Cluster import treecluster
----> 2 tree = treecluster(distancematrix=distance)

NameError: name 'distance' is not defined
```

In this case, the following arguments are defined:

- `distancematrix`
  The distance matrix, which can be specified in three ways:
  - as a 2D Numerical Python array (in which only the left-lower part of the array will be accessed):

```
distance = array([[0.0, 1.1, 2.3],
                  [1.1, 0.0, 4.5],
                  [2.3, 4.5, 0.0]])
```

```
-   as a 1D Numerical Python array containing consecutively the
    distances in the left-lower part of the distance matrix:
```

```
distance = array([1.1, 2.3, 4.5])
```

```
-   as a list containing the rows of the left-lower part of the
    distance matrix:
```

```
distance = [array([]),
            array([1.1]),
            array([2.3, 4.5])
```

```
These three expressions correspond to the same distance matrix. As
`treecluster` may shuffle the values in the distance matrix as part
of the clustering algorithm, be sure to save this array in a
different variable before calling `treecluster` if you need
it later.
```

- `method`
  The linkage method to be used:

- `method=='s'`: pairwise single-linkage clustering
- `method=='m'`: pairwise maximum- (or complete-) linkage clustering
- `method=='a'`: pairwise average-linkage clustering

While pairwise single-, maximum-, and average-linkage clustering can be calculated from the distance matrix alone, pairwise centroid-linkage cannot.

When calling `treecluster`, either `data` or `distancematrix` should be `None`.

This function returns a `Tree` object. This object contains $($number of items $-1)$ nodes, where the number of items is the number of rows if rows were clustered, or the number of columns if columns were clustered. Each node describes a pairwise linking event, where the node attributes `left` and `right` each contain the number of one item or subnode, and `distance` the distance between them. Items are numbered from 0 to $($number of items $-1)$, while clusters are numbered -1 to $-($number of items $-1)$.

16.19.1 Self-Organizing Maps

Self-Organizing Maps (SOMs) were invented by Kohonen to describe neural networks (see for instance Kohonen, 1997 @kohonen1997). Tamayo (1999) first applied Self-Organizing Maps to gene expression data @tamayo1999.

SOMs organize items into clusters that are situated in some topology. Usually a rectangular topology is chosen. The clusters generated by SOMs are such that neighboring clusters in the topology are more similar to each other than clusters far from each other in the topology.

The first step to calculate a SOM is to randomly assign a data vector to each cluster in the topology. If rows are being clustered, then the number of elements in each data vector is equal to the number of columns.

An SOM is then generated by taking rows one at a time, and finding which cluster in the topology has the closest data vector. The data vector of that cluster, as well as those of the neighboring clusters, are adjusted using the data vector of the row under consideration. The adjustment is given by

$$\Delta \underline{x}_{\textrm{cell}} = \tau \cdot \left(\underline{x}_{\textrm{row}} - \underline{x}_{\textrm{cell}} \right).$$
$$The parameter : math : `\tau` is a parameter that decreases at each$$

iteration step. We have used a simple linear function of the iteration step:

$$\tau = \tau_{\textrm{init}} \cdot \left(1 - {i \over n}\right),$$
$$: math : `\tau_{\textrm{init}}` is the initial value of : math : `\tau` as$$

specified by the user, i is the number of the current iteration step, and n is the total number of iteration steps to be performed. While changes are made rapidly in the beginning of the iteration, at the end of iteration only small changes are made.

All clusters within a radius R are adjusted to the gene under consideration. This radius decreases as the calculation progresses as

$$R = R_{\textrm{max}} \cdot \left(1 - {i \over n}\right),$$

in which the maximum radius is defined as

$$R_{\textrm{max}} = \sqrt{N_x^2 + N_y^2},$$
$$where : math : `\left(N_x, N_y\right)` are the dimensions of the rectangle$$

defining the topology.

The function `somcluster` implements the complete algorithm to calculate a Self-Organizing Map on a rectangular grid. First it initializes the random number generator. The node data are then initialized using the random number generator. The order in which genes or microarrays are used to modify the SOM is also randomized. The total number of iterations in the SOM algorithm is specified by the user.

To run `somcluster`, use

```
In [19]: from Bio.Cluster import somcluster
         clusterid, celldata = somcluster(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-24fcbff7f835> in <module>()
      1 from Bio.Cluster import somcluster
----> 2 clusterid, celldata = somcluster(data)

NameError: name 'data' is not defined
```

where the following arguments are defined:

- `data` (required)
  Array containing the data for the items.
- `mask` (default: `None`)
  Array of integers showing which data are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If `mask==None`, then all data are present.
- `weight` (default: `None`)
  contains the weights to be used when calculating distances. If `weight==None`, then equal weights are assumed.
- `transpose` (default: `0`)
  Determines if rows (`transpose` is `0`) or columns (`transpose` is `1`) are to be clustered.
- `nxgrid, nygrid` (default: `2, 1`)
  The number of cells horizontally and vertically in the rectangular grid on which the Self-Organizing Map is calculated.
- `inittau` (default: `0.02`)
  The initial value for the parameter τ that is used in the SOM algorithm. The default value for `inittau` is 0.02, which was used in Michael Eisen's Cluster/TreeView program.
- `niter` (default: `1`)
  The number of iterations to be performed.
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).

This function returns the tuple `(clusterid, celldata)`:

- `clusterid`:
  An array with two columns, where the number of rows is equal to the number of items that were clustered. Each row contains the x and y coordinates of the cell in the rectangular SOM grid to which the item was assigned.
- `celldata`:
  An array with dimensions (`nxgrid`, `nygrid`, number of columns) if rows are being clustered, or (`nxgrid`, `nygrid`, number of rows) if columns are being clustered. Each element `[ix][iy]` of this array is a 1D vector containing the gene expression data for the centroid of the cluster in the grid cell with coordinates `[ix][iy]`.

16.19.2 Principal Component Analysis

Principal Component Analysis (PCA) is a widely used technique for analyzing multivariate data. A practical example of applying Principal Component Analysis to gene expression data is presented by Yeung and Ruzzo (2001) @yeung2001.

In essence, PCA is a coordinate transformation in which each row in the data matrix is written as a linear sum over basis vectors called principal components, which are ordered and chosen such that each maximally explains the remaining variance in the data vectors. For example, an $n \times 3$ data matrix can be represented as an ellipsoidal cloud of n points in three dimensional space. The first principal component is the longest axis of the ellipsoid, the second principal component the second longest axis of the ellipsoid, and the third principal component is the shortest axis. Each row in the data matrix can be reconstructed as a suitable linear combination of the principal components. However, in order to reduce the dimensionality of the data, usually only the most important principal components are retained. The remaining variance present in the data is then regarded as unexplained variance.

The principal components can be found by calculating the eigenvectors of the covariance matrix of the data. The corresponding eigenvalues determine how much of the variance present in the data is explained by each principal component.

Before applying principal component analysis, typically the mean is subtracted from each column in the data matrix. In the example above, this effectively centers the ellipsoidal cloud around its centroid in 3D space, with the principal components describing the variation of points in the ellipsoidal cloud with respect to their centroid.

The function `pca` below first uses the singular value decomposition to calculate the eigenvalues and eigenvectors of the data matrix. The singular value decomposition is implemented as a translation in C of the Algol procedure `svd` @golub1971, which uses Householder bidiagonalization and a variant of the QR algorithm. The principal components, the coordinates of each data vector along the principal components, and the eigenvalues corresponding to the principal components are then evaluated and returned in decreasing order of the magnitude of the eigenvalue. If data centering is desired, the mean should be subtracted from each column in the data matrix before calling the `pca` routine.

To apply Principal Component Analysis to a rectangular matrix `data`, use

```
In [17]: from Bio.Cluster import pca
         columnmean, coordinates, components, eigenvalues = pca(data)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-5edc82169621> in <module>()
      1 from Bio.Cluster import pca
----> 2 columnmean, coordinates, components, eigenvalues = pca(data)

NameError: name 'data' is not defined
```

This function returns a tuple `columnmean, coordinates, components, eigenvalues`:

- `columnmean`
  Array containing the mean over each column in `data`.
- `coordinates`
  The coordinates of each row in `data` with respect to the principal components.
- `components`
  The principal components.
- `eigenvalues`
  The eigenvalues corresponding to each of the principal components.

The original matrix `data` can be recreated by calculating `columnmean + dot(coordinates, components)`.

16.19.3 Handling Cluster/TreeView-type files

Cluster/TreeView are GUI-based codes for clustering gene expression data. They were originally written by Michael Eisen while at Stanford University. `Bio.Cluster` contains functions for reading and writing data files that correspond to the format specified for Cluster/TreeView. In particular, by saving a clustering result in that format, TreeView can be used to visualize the clustering results. We recommend using Alok Saldanha's http://jtreeview.sourceforge.net/Java TreeView program, which can display hierarchical as well as k-means clustering results.

An object of the class `Record` contains all information stored in a Cluster/TreeView-type data file. To store the information contained in the data file in a `Record` object, we first open the file and then read it:

```
In [20]: from Bio import Cluster
         handle = open("mydatafile.txt")
         record = Cluster.read(handle)
         handle.close()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-20-99d49cd96435> in <module>()
      1 from Bio import Cluster
----> 2 handle = open("mydatafile.txt")
      3 record = Cluster.read(handle)
      4 handle.close()

FileNotFoundError: [Errno 2] No such file or directory: 'mydatafile.txt'
```

This two-step process gives you some flexibility in the source of the data. For example, you can use

```
In [19]: import gzip # Python standard library
         handle = gzip.open("mydatafile.txt.gz")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-19-42371407c8bc> in <module>()
      1 import gzip # Python standard library
----> 2 handle = gzip.open("mydatafile.txt.gz")

/home/tiago_antao/miniconda/lib/python3.5/gzip.py in open(filename, mode, compresslevel, encoding, e
     51     gz_mode = mode.replace("t", "")
     52     if isinstance(filename, (str, bytes)):
---> 53         binary_file = GzipFile(filename, gz_mode, compresslevel)
     54     elif hasattr(filename, "read") or hasattr(filename, "write"):
     55         binary_file = GzipFile(None, gz_mode, compresslevel, filename)

/home/tiago_antao/miniconda/lib/python3.5/gzip.py in __init__(self, filename, mode, compresslevel, f
    161             mode += 'b'
    162         if fileobj is None:
--> 163             fileobj = self.myfileobj = builtins.open(filename, mode or 'rb')
    164         if filename is None:
    165             filename = getattr(fileobj, 'name', '')

FileNotFoundError: [Errno 2] No such file or directory: 'mydatafile.txt.gz'
```

to open a gzipped file, or

```
In [20]: import urllib # Python standard library
         handle = urllib.urlopen("http://somewhere.org/mydatafile.txt")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-20-ea06f324a41c> in <module>()
```

```
      1 import urllib # Python standard library
----> 2 handle = urllib.urlopen("http://somewhere.org/mydatafile.txt")

AttributeError: module 'urllib' has no attribute 'urlopen'
```

to open a file stored on the Internet before calling `read`.

The `read` command reads the tab-delimited text file `mydatafile.txt` containing gene expression data in the format specified for Michael Eisen's Cluster/TreeView program. For a description of this file format, see the manual to Cluster/TreeView. It is available at Michael Eisen's lab website and at our website.

A `Record` object has the following attributes:

- `data`

  The data array containing the gene expression data. Genes are stored row-wise, while microarrays are stored column-wise.

- `mask`

  This array shows which elements in the `data` array, if any, are missing. If `mask[i,j]==0`, then `data[i,j]` is missing. If no data were found to be missing, `mask` is set to `None`.

- `geneid`

  This is a list containing a unique description for each gene (i.e., ORF numbers).

- `genename`

  This is a list containing a description for each gene (i.e., gene name). If not present in the data file, `genename` is set to `None`.

- `gweight`

  The weights that are to be used to calculate the distance in expression profile between genes. If not present in the data file, `gweight` is set to `None`.

- `gorder`

  The preferred order in which genes should be stored in an output file. If not present in the data file, `gorder` is set to `None`.

- `expid`

  This is a list containing a description of each microarray, e.g. experimental condition.

- `eweight`

  The weights that are to be used to calculate the distance in expression profile between microarrays. If not present in the data file, `eweight` is set to `None`.

- `eorder`

  The preferred order in which microarrays should be stored in an output file. If not present in the data file, `eorder` is set to `None`.

- `uniqid`

  The string that was used instead of UNIQID in the data file.

After loading a `Record` object, each of these attributes can be accessed and modified directly. For example, the data can be log-transformed by taking the logarithm of `record.data`.

16.20 Calculating the distance matrix

To calculate the distance matrix between the items stored in the record, use

```
In [21]: matrix = record.distancematrix()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-dca24589e99d> in <module>()
----> 1 matrix = record.distancematrix()

NameError: name 'record' is not defined
```

where the following arguments are defined:

- `transpose` (default: `0`)
  Determines if the distances between the rows of `data` are to be calculated (`transpose==0`), or between the columns of `data` (`transpose==1`).
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).

This function returns the distance matrix as a list of rows, where the number of columns of each row is equal to the row number (see section [subsec:distancematrix]).

16.21 Calculating the cluster centroids

To calculate the centroids of clusters of items stored in the record, use

```
In [22]: cdata, cmask = record.clustercentroids()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-b7634e141823> in <module>()
----> 1 cdata, cmask = record.clustercentroids()

NameError: name 'record' is not defined
```

- `clusterid` (default: `None`)
  Vector of integers showing to which cluster each item belongs. If `clusterid` is not given, then all items are assumed to belong to the same cluster.
- `method` (default: `'a'`)
  Specifies whether the arithmetic mean (`method=='a'`) or the median (`method=='m'`) is used to calculate the cluster center.
- `transpose` (default: `0`)
  Determines if the centroids of the rows of `data` are to be calculated (`transpose==0`), or the centroids of the columns of `data` (`transpose==1`).

This function returns the tuple `cdata, cmask`; see section [subsec:clustercentroids] for a description.

16.22 Calculating the distance between clusters

To calculate the distance between clusters of items stored in the record, use

```
In [23]: distance = record.clusterdistance()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-d87103d6884f> in <module>()
----> 1 distance = record.clusterdistance()
```

```
NameError: name 'record' is not defined
```

where the following arguments are defined:

- `index1` (default: `0`)
  A list containing the indices of the items belonging to the first cluster. A cluster containing only one item i can be represented either as a list `[i]`, or as an integer `i`.
- `index2` (default: `0`)
  A list containing the indices of the items belonging to the second cluster. A cluster containing only one item i can be represented either as a list `[i]`, or as an integer `i`.
- `method` (default: `'a'`)
  Specifies how the distance between clusters is defined:
  - `'a'`: Distance between the two cluster centroids (arithmetic mean);
  - `'m'`: Distance between the two cluster centroids (median);
  - `'s'`: Shortest pairwise distance between items in the two clusters;
  - `'x'`: Longest pairwise distance between items in the two clusters;
  - `'v'`: Average over the pairwise distances between items in the two clusters.
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).
- `transpose` (default: `0`)
  If `transpose==0`, calculate the distance between the rows of `data`. If `transpose==1`, calculate the distance between the columns of `data`.

16.23 Performing hierarchical clustering

To perform hierarchical clustering on the items stored in the record, use

```
In [24]: tree = record.treecluster()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-455e50ad0270> in <module>()
----> 1 tree = record.treecluster()

NameError: name 'record' is not defined
```

where the following arguments are defined:

- `transpose` (default: `0`)
  Determines if rows (`transpose==0`) or columns (`transpose==1`) are to be clustered.
- `method` (default: `'m'`)
  defines the linkage method to be used:
  - `method=='s'`: pairwise single-linkage clustering
  - `method=='m'`: pairwise maximum- (or complete-) linkage clustering
  - `method=='c'`: pairwise centroid-linkage clustering
  - `method=='a'`: pairwise average-linkage clustering

- `dist` (default: `'e'`, Euclidean distance)

  Defines the distance function to be used (see [sec:distancefunctions]).

- `transpose`

  Determines if genes or microarrays are being clustered. If `transpose==0`, genes (rows) are being clustered. If `transpose==1`, microarrays (columns) are clustered.

This function returns a `Tree` object. This object contains $(\text{number of items} - 1)$ nodes, where the number of items is the number of rows if rows were clustered, or the number of columns if columns were clustered. Each node describes a pairwise linking event, where the node attributes `left` and `right` each contain the number of one item or subnode, and `distance` the distance between them. Items are numbered from 0 to $(\text{number of items} - 1)$, while clusters are numbered -1 to $-(\text{number of items} - 1)$.

16.24 Performing k-means or k-medians clustering

To perform k-means or k-medians clustering on the items stored in the record, use

```
In [25]: clusterid, error, nfound = record.kcluster()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-bab4db892231> in <module>()
----> 1 clusterid, error, nfound = record.kcluster()

NameError: name 'record' is not defined
```

where the following arguments are defined:

- `nclusters` (default: `2`)

  The number of clusters k.

- `transpose` (default: `0`)

  Determines if rows (`transpose` is `0`) or columns (`transpose` is `1`) are to be clustered.

- `npass` (default: `1`)

  The number of times the k-means/-medians clustering algorithm is performed, each time with a different (random) initial condition. If `initialid` is given, the value of `npass` is ignored and the clustering algorithm is run only once, as it behaves deterministically in that case.

- `method` (default: `a`)

  describes how the center of a cluster is found:

  - `method=='a'`: arithmetic mean (k-means clustering);
  - `method=='m'`: median (k-medians clustering).

  For other values of `method`, the arithmetic mean is used.

- `dist` (default: `'e'`, Euclidean distance)

  Defines the distance function to be used (see [sec:distancefunctions]).

This function returns a tuple `(clusterid, error, nfound)`, where `clusterid` is an integer array containing the number of the cluster to which each row or cluster was assigned, `error` is the within-cluster sum of distances for the optimal clustering solution, and `nfound` is the number of times this optimal solution was found.

16.25 Calculating a Self-Organizing Map

To calculate a Self-Organizing Map of the items stored in the record, use

```
In [26]: clusterid, celldata = record.somcluster()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-337f60498164> in <module>()
----> 1 clusterid, celldata = record.somcluster()

NameError: name 'record' is not defined
```

where the following arguments are defined:

- `transpose` (default: `0`)
  Determines if rows (`transpose` is `0`) or columns (`transpose` is `1`) are to be clustered.
- `nxgrid, nygrid` (default: `2, 1`)
  The number of cells horizontally and vertically in the rectangular grid on which the Self-Organizing Map is calculated.
- `inittau` (default: `0.02`)
  The initial value for the parameter τ that is used in the SOM algorithm. The default value for `inittau` is 0.02, which was used in Michael Eisen's Cluster/TreeView program.
- `niter` (default: `1`)
  The number of iterations to be performed.
- `dist` (default: `'e'`, Euclidean distance)
  Defines the distance function to be used (see [sec:distancefunctions]).

This function returns the tuple `(clusterid, celldata)`:

- `clusterid`:
  An array with two columns, where the number of rows is equal to the number of items that were clustered. Each row contains the x and y coordinates of the cell in the rectangular SOM grid to which the item was assigned.
- `celldata`:
  An array with dimensions $(\texttt{nxgrid}, \texttt{nygrid}, \text{number of columns})$ if rows are being clustered, or $(\texttt{nxgrid}, \texttt{nygrid}, \text{number of rows})$ if columns are being clustered. Each element `[ix][iy]` of this array is a 1D vector containing the gene expression data for the centroid of the cluster in the grid cell with coordinates `[ix][iy]`.

16.26 Saving the clustering result

To save the clustering result, use

```
In [27]: record.save(jobname, geneclusters, expclusters)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-7f22d4d413d4> in <module>()
----> 1 record.save(jobname, geneclusters, expclusters)
```

```
NameError: name 'record' is not defined
```

where the following arguments are defined:

- `jobname`
  The string `jobname` is used as the base name for names of the files that are to be saved.
- `geneclusters`
  This argument describes the gene (row-wise) clustering result. In case of k-means clustering, this is a 1D array containing the number of the cluster each gene belongs to. It can be calculated using `kcluster`. In case of hierarchical clustering, `geneclusters` is a `Tree` object.
- `expclusters`
  This argument describes the (column-wise) clustering result for the experimental conditions. In case of k-means clustering, this is a 1D array containing the number of the cluster each experimental condition belongs to. It can be calculated using `kcluster`. In case of hierarchical clustering, `expclusters` is a `Tree` object.

This method writes the text file `jobname.cdt`, `jobname.gtr`, `jobname.atr`, `jobname*.kgg`, and/or `jobname*.kag` for subsequent reading by the Java TreeView program. If `geneclusters` and `expclusters` are both `None`, this method only writes the text file `jobname.cdt`; this file can subsequently be read into a new `Record` object.

16.26.1 Example calculation

This is an example of a hierarchical clustering calculation, using single linkage clustering for genes and maximum linkage clustering for experimental conditions. As the Euclidean distance is being used for gene clustering, it is necessary to scale the node distances `genetree` such that they are all between zero and one. This is needed for the Java TreeView code to display the tree diagram correctly. To cluster the experimental conditions, the uncentered correlation is being used. No scaling is needed in this case, as the distances in `exptree` are already between zero and two. The example data `cyano.txt` can be found in the `data` subdirectory.

```
In [28]: from Bio import Cluster
         handle = open("cyano.txt")
         record = Cluster.read(handle)
         handle.close()
         genetree = record.treecluster(method='s')
         genetree.scale()
         exptree = record.treecluster(dist='u', transpose=1)
         record.save("cyano_result", genetree, exptree)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-28-80f17db22da5> in <module>()
      1 from Bio import Cluster
----> 2 handle = open("cyano.txt")
      3 record = Cluster.read(handle)
      4 handle.close()
      5 genetree = record.treecluster(method='s')

FileNotFoundError: [Errno 2] No such file or directory: 'cyano.txt'
```

This will create the files `cyano_result.cdt`, `cyano_result.gtr`, and `cyano_result.atr`.

Similarly, we can save a k-means clustering solution:

```
In [29]: from Bio import Cluster
         handle = open("cyano.txt")
```

```
record = Cluster.read(handle)
handle.close()
(geneclusters, error, ifound) = record.kcluster(nclusters=5, npass=1000)
(expclusters, error, ifound) = record.kcluster(nclusters=2, npass=100, transpose=1)
record.save("cyano_result", geneclusters, expclusters)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-29-99295949c49b> in <module>()
      1 from Bio import Cluster
----> 2 handle = open("cyano.txt")
      3 record = Cluster.read(handle)
      4 handle.close()
      5 (geneclusters, error, ifound) = record.kcluster(nclusters=5, npass=1000)

FileNotFoundError: [Errno 2] No such file or directory: 'cyano.txt'
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 17

Supervised learning methods

Note the supervised learning methods described in this chapter all require Numerical Python (numpy) to be installed.

17.1 The Logistic Regression Model

17.1.1 Background and Purpose

Logistic regression is a supervised learning approach that attempts to distinguish K classes from each other using a weighted sum of some predictor variables x_i. The logistic regression model is used to calculate the weights β_i of the predictor variables. In Biopython, the logistic regression model is currently implemented for two classes only ($K = 2$); the number of predictor variables has no predefined limit.

As an example, let's try to predict the operon structure in bacteria. An operon is a set of adjacent genes on the same strand of DNA that are transcribed into a single mRNA molecule. Translation of the single mRNA molecule then yields the individual proteins. For *Bacillus subtilis*, whose data we will be using, the average number of genes in an operon is about 2.4.

As a first step in understanding gene regulation in bacteria, we need to know the operon structure. For about 10% of the genes in *Bacillus subtilis*, the operon structure is known from experiments. A supervised learning method can be used to predict the operon structure for the remaining 90% of the genes.

For such a supervised learning approach, we need to choose some predictor variables x_i that can be measured easily and are somehow related to the operon structure. One predictor variable might be the distance in base pairs between genes. Adjacent genes belonging to the same operon tend to be separated by a relatively short distance, whereas adjacent genes in different operons tend to have a larger space between them to allow for promoter and terminator sequences. Another predictor variable is based on gene expression measurements. By definition, genes belonging to the same operon have equal gene expression profiles, while genes in different operons are expected to have different expression profiles. In practice, the measured expression profiles of genes in the same operon are not quite identical due to the presence of measurement errors. To assess the similarity in the gene expression profiles, we assume that the measurement errors follow a normal distribution and calculate the corresponding log-likelihood score.

We now have two predictor variables that we can use to predict if two adjacent genes on the same strand of DNA belong to the same operon:

- x_1: the number of base pairs between them;
- x_2: their similarity in expression profile.

In a logistic regression model, we use a weighted sum of these two predictors to calculate a joint score S:

$$S = \beta_0 + \beta_1 x_1 + \beta_2 x_2.$$

The logistic regression model gives us appropriate values for the parameters β_0, β_1, β_2 using two sets of example genes:

- OP: Adjacent genes, on the same strand of DNA, known to belong to the same operon;
- NOP: Adjacent genes, on the same strand of DNA, known to belong to different operons.

In the logistic regression model, the probability of belonging to a class depends on the score via the logistic function. For the two classes OP and NOP, we can write this as

$$\begin{aligned} \Pr(\mathrm{OP}|x_1, x_2) &= \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2)} \\ \Pr(\mathrm{NOP}|x_1, x_2) &= \frac{1}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2)} \\ Using a set of gene pairs for which it is known whether they belong to \end{aligned}$$

the same operon (class OP) or to different operons (class NOP), we can calculate the weights β_0, β_1, β_2 by maximizing the log-likelihood corresponding to the probability functions ([eq:OP]) and ([eq:NOP]).

17.1.2 Training the logistic regression model

Gene pair	Intergene distance (x_1)	Gene expression score (x_2)	Class
cotJA — *cotJB*	-53	-200.78	OP
yesK — *yesL*	117	-267.14	OP
lplA — *lplB*	57	-163.47	OP
lplB — *lplC*	16	-190.30	OP
lplC — *lplD*	11	-220.94	OP
lplD — *yetF*	85	-193.94	OP
yfmT — *yfmS*	16	-182.71	OP
yfmF — *yfmE*	15	-180.41	OP
citS — *citT*	-26	-181.73	OP
citM — *yflN*	58	-259.87	OP
yfiI — *yfiJ*	126	-414.53	NOP
lipB — *yfiQ*	191	-249.57	NOP
yfiU — *yfiV*	113	-265.28	NOP
yfhH — *yfhI*	145	-312.99	NOP
cotY — *cotX*	154	-213.83	NOP
yjoB — *rapA*	147	-380.85	NOP
ptsI — *splA*	93	-291.13	NOP

Table: Adjacent gene pairs known to belong to the same operon (class OP) or to different operons (class NOP). Intergene distances are negative if the two genes overlap.

[table:training]

Table [table:training] lists some of the *Bacillus subtilis* gene pairs for which the operon structure is known. Let's calculate the logistic regression model from these data:

```
In [1]: from Bio import LogisticRegression
        xs = [[-53, -200.78], [117, -267.14], [57, -163.47], [16, -190.30],
              [11, -220.94], [85, -193.94], [16, -182.71], [15, -180.41],
              [-26, -181.73], [58, -259.87], [126, -414.53], [191, -249.57],
              [113, -265.28], [145, -312.99], [154, -213.83], [147, -380.85],[93, -291.13]]
        ys = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
        model = LogisticRegression.train(xs, ys)
```

Here, `xs` and `ys` are the training data: `xs` contains the predictor variables for each gene pair, and `ys` specifies if the gene pair belongs to the same operon (`1`, class OP) or different operons (`0`, class NOP). The resulting logistic regression model is stored in `model`, which contains the weights β_0, β_1, and β_2:

```
In [2]: model.beta
```

```
Out[2]: [8.983029015714472, -0.035968960444850887, 0.021813956629835204]
```

Note that β_1 is negative, as gene pairs with a shorter intergene distance have a higher probability of belonging to the same operon (class OP). On the other hand, β_2 is positive, as gene pairs belonging to the same operon typically have a higher similarity score of their gene expression profiles. The parameter β_0 is positive due to the higher prevalence of operon gene pairs than non-operon gene pairs in the training data.

The function `train` has two optional arguments: `update_fn` and `typecode`. The `update_fn` can be used to specify a callback function, taking as arguments the iteration number and the log-likelihood. With the callback function, we can for example track the progress of the model calculation (which uses a Newton-Raphson iteration to maximize the log-likelihood function of the logistic regression model):

```
In [8]: def show_progress(iteration, loglikelihood):
            print("Iteration:", iteration, "Log-likelihood function:", loglikelihood)

        model = LogisticRegression.train(xs, ys, update_fn=show_progress)
```

```
Iteration: <built-in function iter> Log-likelihood function: -11.7835020695
Iteration: <built-in function iter> Log-likelihood function: -7.15886767672
Iteration: <built-in function iter> Log-likelihood function: -5.76877209868
Iteration: <built-in function iter> Log-likelihood function: -5.11362294338
Iteration: <built-in function iter> Log-likelihood function: -4.74870642433
Iteration: <built-in function iter> Log-likelihood function: -4.50026077146
Iteration: <built-in function iter> Log-likelihood function: -4.31127773737
Iteration: <built-in function iter> Log-likelihood function: -4.16015043396
Iteration: <built-in function iter> Log-likelihood function: -4.03561719785
Iteration: <built-in function iter> Log-likelihood function: -3.93073282192
Iteration: <built-in function iter> Log-likelihood function: -3.84087660929
Iteration: <built-in function iter> Log-likelihood function: -3.76282560605
Iteration: <built-in function iter> Log-likelihood function: -3.69425027154
Iteration: <built-in function iter> Log-likelihood function: -3.6334178602
Iteration: <built-in function iter> Log-likelihood function: -3.57900855837
Iteration: <built-in function iter> Log-likelihood function: -3.52999671386
Iteration: <built-in function iter> Log-likelihood function: -3.48557145163
Iteration: <built-in function iter> Log-likelihood function: -3.44508206139
Iteration: <built-in function iter> Log-likelihood function: -3.40799948447
Iteration: <built-in function iter> Log-likelihood function: -3.3738885624
Iteration: <built-in function iter> Log-likelihood function: -3.3423876581
Iteration: <built-in function iter> Log-likelihood function: -3.31319343769
Iteration: <built-in function iter> Log-likelihood function: -3.2860493346
Iteration: <built-in function iter> Log-likelihood function: -3.2607366863
Iteration: <built-in function iter> Log-likelihood function: -3.23706784091
Iteration: <built-in function iter> Log-likelihood function: -3.21488073614
Iteration: <built-in function iter> Log-likelihood function: -3.19403459259
Iteration: <built-in function iter> Log-likelihood function: -3.17440646052
Iteration: <built-in function iter> Log-likelihood function: -3.15588842703
Iteration: <built-in function iter> Log-likelihood function: -3.13838533947
```

```
Iteration: <built-in function iter> Log-likelihood function: -3.12181293595
Iteration: <built-in function iter> Log-likelihood function: -3.10609629966
Iteration: <built-in function iter> Log-likelihood function: -3.09116857282
Iteration: <built-in function iter> Log-likelihood function: -3.07696988017
Iteration: <built-in function iter> Log-likelihood function: -3.06344642288
Iteration: <built-in function iter> Log-likelihood function: -3.05054971191
Iteration: <built-in function iter> Log-likelihood function: -3.03823591619
Iteration: <built-in function iter> Log-likelihood function: -3.02646530573
Iteration: <built-in function iter> Log-likelihood function: -3.01520177394
Iteration: <built-in function iter> Log-likelihood function: -3.00441242601
Iteration: <built-in function iter> Log-likelihood function: -2.99406722296
Iteration: <built-in function iter> Log-likelihood function: -2.98413867259
```

The iteration stops once the increase in the log-likelihood function is less than 0.01. If no convergence is reached after 500 iterations, the `train` function returns with an `AssertionError`.

The optional keyword `typecode` can almost always be ignored. This keyword allows the user to choose the type of Numeric matrix to use. In particular, to avoid memory problems for very large problems, it may be necessary to use single-precision floats (Float8, Float16, etc.) rather than double, which is used by default.

17.1.3 Using the logistic regression model for classification

Classification is performed by calling the `classify` function. Given a logistic regression model and the values for x_1 and x_2 (e.g. for a gene pair of unknown operon structure), the `classify` function returns `1` or `0`, corresponding to class OP and class NOP, respectively. For example, let's consider the gene pairs *yxcE*, *yxcD* and *yxiB*, *yxiA*:

Gene pair	Intergene distance x_1	Gene expression score x_2
yxcE — *yxcD*	6	-173.143442352
yxiB — *yxiA*	309	-271.005880394

Table: Adjacent gene pairs of unknown operon status.

The logistic regression model classifies *yxcE*, *yxcD* as belonging to the same operon (class OP), while *yxiB*, *yxiA* are predicted to belong to different operons:

```
In [5]: print("yxcE, yxcD:", LogisticRegression.classify(model, [6, -173.143442352]))
        print("yxiB, yxiA:", LogisticRegression.classify(model, [309, -271.005880394]))
```

```
yxcE, yxcD: 1
yxiB, yxiA: 0
```

(which, by the way, agrees with the biological literature).

To find out how confident we can be in these predictions, we can call the `calculate` function to obtain the probabilities (equations ([eq:OP]) and [eq:NOP]) for class OP and NOP. For *yxcE*, *yxcD* we find

```
In [6]: q, p = LogisticRegression.calculate(model, [6, -173.143442352])
        print("class OP: probability =", p, "class NOP: probability =", q)
```

```
class OP: probability = 0.993242163503 class NOP: probability = 0.00675783649744
```

and for *yxiB*, *yxiA*

```
In [7]: q, p = LogisticRegression.calculate(model, [309, -271.005880394])
        print("class OP: probability =", p, "class NOP: probability =", q)
```

```
class OP: probability = 0.000321211251817 class NOP: probability = 0.999678788748
```

To get some idea of the prediction accuracy of the logistic regression model, we can apply it to the training data:

```
In [9]: for i in range(len(ys)):
            print("True:", ys[i], "Predicted:", LogisticRegression.classify(model, xs[i]))
```

```
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
```

showing that the prediction is correct for all but one of the gene pairs. A more reliable estimate of the prediction accuracy can be found from a leave-one-out analysis, in which the model is recalculated from the training data after removing the gene to be predicted:

```
In [10]: for i in range(len(ys)):
             print("True:", ys[i], "Predicted:", LogisticRegression.classify(model, xs[i]))
```

```
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
```

The leave-one-out analysis shows that the prediction of the logistic regression model is incorrect for only two of the gene pairs, which corresponds to a prediction accuracy of 88%.

17.1.4 Logistic Regression, Linear Discriminant Analysis, and Support Vector Machines

The logistic regression model is similar to linear discriminant analysis. In linear discriminant analysis, the class probabilities also follow equations ([eq:OP]) and ([eq:NOP]). However, instead of estimating the coefficients β directly, we first fit a normal distribution to the predictor variables x. The coefficients β are then calculated from the means and covariances of the normal distribution. If the distribution of x is indeed normal, then we expect linear discriminant analysis to perform better than the logistic regression model. The logistic regression model, on the other hand, is more robust to deviations from normality.

Another similar approach is a support vector machine with a linear kernel. Such an SVM also uses a linear combination of the predictors, but estimates the coefficients β from the predictor variables x near the boundary region between the classes. If the logistic regression model (equations ([eq:OP]) and ([eq:NOP])) is a good description for x away from the boundary region, we expect the logistic regression model to perform better than an SVM with a linear kernel, as it relies on more data. If not, an SVM with a linear kernel may perform better.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman: *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2001. Chapter 4.4.

17.2 k-Nearest Neighbors

17.2.1 Background and purpose

The k-nearest neighbors method is a supervised learning approach that does not need to fit a model to the data. Instead, data points are classified based on the categories of the k nearest neighbors in the training data set.

In Biopython, the k-nearest neighbors method is available in `Bio.kNN`. To illustrate the use of the k-nearest neighbor method in Biopython, we will use the same operon data set as in section [sec:LogisticRegression].

17.2.2 Initializing a k-nearest neighbors model

Using the data in Table [table:training], we create and initialize a k-nearest neighbors model as follows:

```
In [11]: from Bio import kNN
         k = 3
         model = kNN.train(xs, ys, k)
```

where `xs` and `ys` are the same as in Section [subsec:LogisticRegressionTraining]. Here, `k` is the number of neighbors k that will be considered for the classification. For classification into two classes, choosing an odd number for k lets you avoid tied votes. The function name `train` is a bit of a misnomer, since no model training is done: this function simply stores `xs`, `ys`, and `k` in `model`.

17.2.3 Using a k-nearest neighbors model for classification

To classify new data using the k-nearest neighbors model, we use the `classify` function. This function takes a data point (x_1, x_2) and finds the k-nearest neighbors in the training data set `xs`. The data point (x_1, x_2) is then classified based on which category (`ys`) occurs most among the k neighbors.

For the example of the gene pairs *yxcE*, *yxcD* and *yxiB*, *yxiA*, we find:

```
In [12]: x = [6, -173.143442352]
         print("yxcE, yxcD:", kNN.classify(model, x))
```

```
yxcE, yxcD: 1
```

```
In [13]: x = [309, -271.005880394]
         print("yxiB, yxiA:", kNN.classify(model, x))
```

```
yxiB, yxiA: 0
```

In agreement with the logistic regression model, *yxcE*, *yxcD* are classified as belonging to the same operon (class OP), while *yxiB*, *yxiA* are predicted to belong to different operons.

The `classify` function lets us specify both a distance function and a weight function as optional arguments. The distance function affects which k neighbors are chosen as the nearest neighbors, as these are defined as the neighbors with the smallest distance to the query point (x, y). By default, the Euclidean distance is used. Instead, we could for example use the city-block (Manhattan) distance:

```
In [15]: def cityblock(x1, x2):
             assert len(x1)==2
             assert len(x2)==2
             distance = abs(x1[0]-x2[0]) + abs(x1[1]-x2[1])
             return distance

         x = [6, -173.143442352]
         print("yxcE, yxcD:", kNN.classify(model, x, distance_fn = cityblock))
```

```
yxcE, yxcD: 1
```

The weight function can be used for weighted voting. For example, we may want to give closer neighbors a higher weight than neighbors that are further away:

```
In [17]: from math import exp
         def weight(x1, x2):
             assert len(x1)==2
             assert len(x2)==2
             return exp(-abs(x1[0]-x2[0]) - abs(x1[1]-x2[1]))

         x = [6, -173.143442352]
         print("yxcE, yxcD:", kNN.classify(model, x, weight_fn = weight))
```

```
yxcE, yxcD: 1
```

By default, all neighbors are given an equal weight.

To find out how confident we can be in these predictions, we can call the `calculate` function, which will calculate the total weight assigned to the classes OP and NOP. For the default weighting scheme, this reduces to the number of neighbors in each category. For *yxcE*, *yxcD*, we find

```
In [18]: x = [6, -173.143442352]
         weight = kNN.calculate(model, x)
         print("class OP: weight =", weight[0], "class NOP: weight =", weight[1])
```

```
class OP: weight = 0.0 class NOP: weight = 3.0
```

which means that all three neighbors of `x1`, `x2` are in the NOP class. As another example, for *yesK*, *yesL* we find

```
In [19]: x = [117, -267.14]
         weight = kNN.calculate(model, x)
         print("class OP: weight =", weight[0], "class NOP: weight =", weight[1])
```

```
class OP: weight = 2.0 class NOP: weight = 1.0
```

which means that two neighbors are operon pairs and one neighbor is a non-operon pair.

To get some idea of the prediction accuracy of the k-nearest neighbors approach, we can apply it to the training data:

```
In [20]: for i in range(len(ys)):
             print("True:", ys[i], "Predicted:", kNN.classify(model, xs[i]))
```

```
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 0 Predicted: 0
```

```
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
```

showing that the prediction is correct for all but two of the gene pairs. A more reliable estimate of the prediction accuracy can be found from a leave-one-out analysis, in which the model is recalculated from the training data after removing the gene to be predicted:

```
In [25]: for i in range(len(ys)):
             model = kNN.train(xs[:i]+xs[i+1:], ys[:i]+ys[i+1:], 3)
             print("True:", ys[i], "Predicted:", kNN.classify(model, xs[i]))
```

```
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 1
True: 1 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 1
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 0
True: 0 Predicted: 1
```

Source of the materials: Biopython cookbook (adapted)

```
In [1]: #Lets load notebook's Image
        from IPython.core.display import Image
```

```
In [2]: from reportlab.lib import colors
        from reportlab.lib.units import cm
        from Bio.Graphics import GenomeDiagram
        from Bio import SeqIO
```

CHAPTER 18

Graphics including GenomeDiagram

The Bio.Graphics module depends on the third party Python library ReportLab. Although focused on producing PDF files, ReportLab can also create encapsulated postscript (EPS) and (SVG) files. In addition to these vector based images, provided certain further dependencies such as the Python Imaging Library (PIL) are installed, ReportLab can also output bitmap images (including JPEG, PNG, GIF, BMP and PICT formats).

18.1 GenomeDiagram

18.1.1 Introduction

The **Bio.Graphics.GenomeDiagram** module was added to Biopython 1.50, having previously been available as a separate Python module dependent on Biopython.

As the name might suggest, GenomeDiagram was designed for drawing whole genomes, in particular prokaryotic genomes, either as linear diagrams (optionally broken up into fragments to fit better) or as circular wheel diagrams. It proved also well suited to drawing quite detailed figures for smaller genomes such as phage, plasmids or mitochrondia.

This module is easiest to use if you have your genome loaded as a SeqRecord object containing lots of SeqFeature objects - for example as loaded from a GenBank file.

18.1.2 Diagrams, tracks, feature-sets and features

GenomeDiagram uses a nested set of objects. At the top level, you have a diagram object representing a sequence (or sequence region) along the horizontal axis (or circle). A diagram can contain one or more tracks, shown stacked vertically (or radially on circular diagrams). These will typically all have the same length and represent the same sequence region. You might use one track to show the gene locations, another to show regulatory regions, and a third track to show the GC percentage.

The most commonly used type of track will contain features, bundled together in feature-sets. You might choose to use one feature-set for all your CDS features, and another for tRNA features. This isn't required - they can all go in the same feature-set, but it makes it easier to update the properties of just selected features (e.g. make all the tRNA features red).

There are two main ways to build up a complete diagram. Firstly, the top down approach where you create a diagram object, and then using its methods add track(s), and use the track methods to add feature-set(s), and use their methods to add the features. Secondly, you can create the individual objects separately (in whatever order suits your code), and then combine them.

18.1.3 A top down example

We're going to draw a whole genome from a SeqRecord object read in from a GenBank file. This example uses the pPCP1 plasmid from Yersinia pestis biovar Microtus (NC_005816.gb)

```
In [3]: record = SeqIO.read("data/NC_005816.gb", "genbank")
```

We're using a top down approach, so after loading in our sequence we next create an empty diagram, then add an (empty) track, and to that add an (empty) feature set:

```
In [4]: gd_diagram = GenomeDiagram.Diagram("Yersinia pestis biovar Microtus plasmid pPCP1")
        gd_track_for_features = gd_diagram.new_track(1, name="Annotated Features")
        gd_feature_set = gd_track_for_features.new_set()
```

Now the fun part - we take each gene SeqFeature object in our SeqRecord, and use it to generate a feature on the diagram. We're going to color them blue, alternating between a dark blue and a light blue.

```
In [5]: for feature in record.features:
            if feature.type != "gene":
                #Exclude this feature
                continue
            if len(gd_feature_set) % 2 == 0:
                color = colors.blue
            else:
                color = colors.lightblue
            gd_feature_set.add_feature(feature, color=color, label=True)
```

Now we come to actually making the output file. This happens in two steps, first we call the draw method, which creates all the shapes using ReportLab objects. Then we call the write method which renders these to the requested file format. Note you can output in multiple file formats:

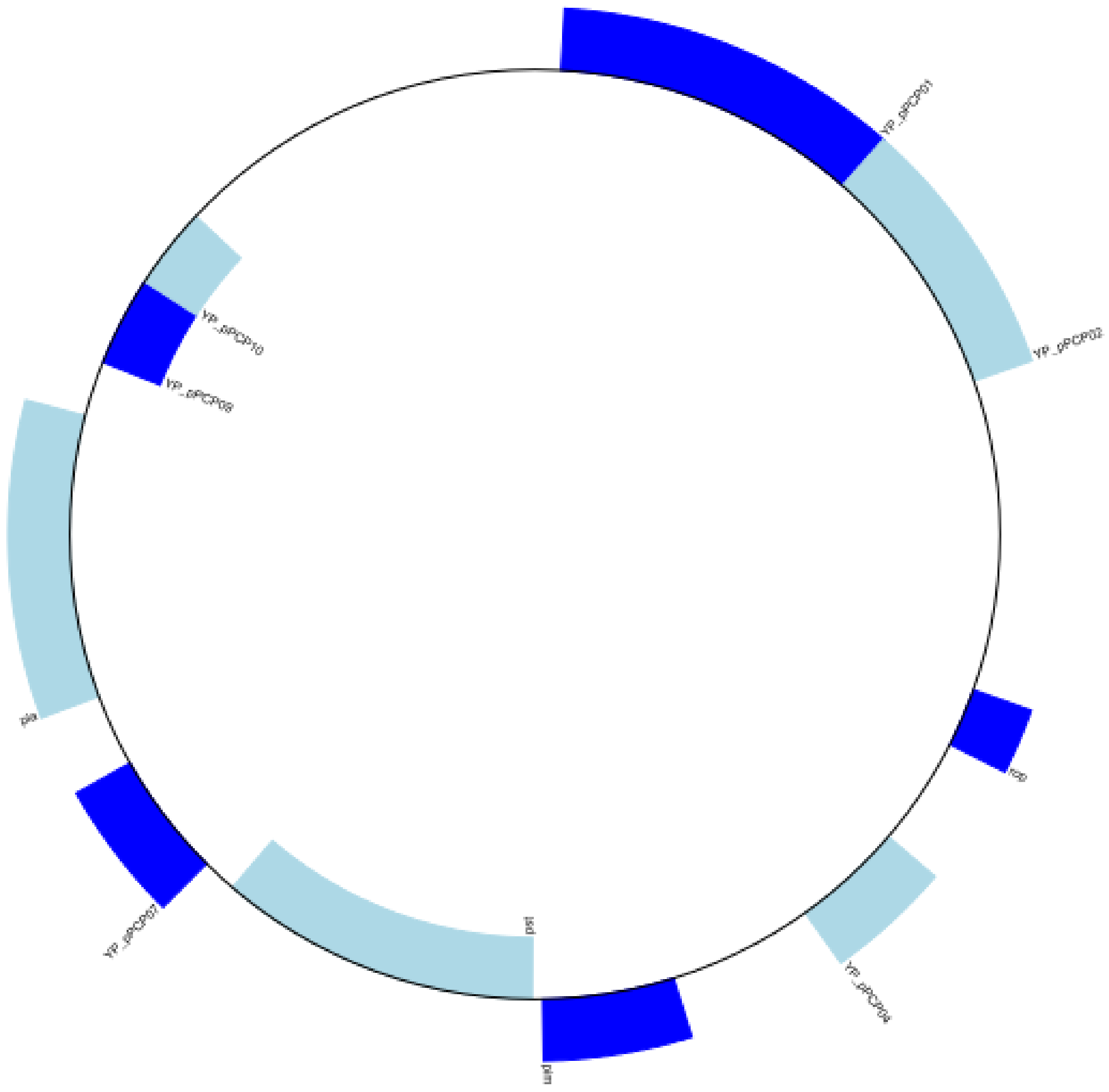
```
In [6]: gd_diagram.draw(format="linear", orientation="landscape", pagesize='A4',
                        fragments=4, start=0, end=len(record))
        gd_diagram.write("data/plasmid_linear.png", "png")
```

Lets have a look at the previous one:

Notice that the fragments argument which we set to four controls how many pieces the genome gets broken up into.

If you want to do a circular figure, then try this:

```
In [7]: gd_diagram.draw(format="circular", circular=True, pagesize=(20*cm,20*cm),
                        start=0, end=len(record), circle_core=0.7)
        gd_diagram.write("data/plasmid_circular.png", "PNG")
        Image("data/plasmid_circular.png")
```



These figures are not very exciting, but we've only just got started.

18.1.4 A bottom up example

Now let's produce exactly the same figures, but using the bottom up approach. This means we create the different objects directly (and this can be done in almost any order) and then combine them.

```
In [8]: record = SeqIO.read("data/NC_005816.gb", "genbank")

        #Create the feature set and its feature objects,
        gd_feature_set = GenomeDiagram.FeatureSet()
        for feature in record.features:
            if feature.type != "gene":
                #Exclude this feature
                continue
            if len(gd_feature_set) % 2 == 0:
                color = colors.blue
            else:
                color = colors.lightblue
            gd_feature_set.add_feature(feature, color=color, label=True)
        #(this for loop is the same as in the previous example)

        #Create a track, and a diagram
        gd_track_for_features = GenomeDiagram.Track(name="Annotated Features")
        gd_diagram = GenomeDiagram.Diagram("Yersinia pestis biovar Microtus plasmid pPCP1")

        #Now have to glue the bits together...
        gd_track_for_features.add_set(gd_feature_set)
        gd_diagram.add_track(gd_track_for_features, 1)
```

You can now call the draw and write methods as before to produce a linear or circular diagram, using the code at the end of the top-down example above. The figures should be identical.

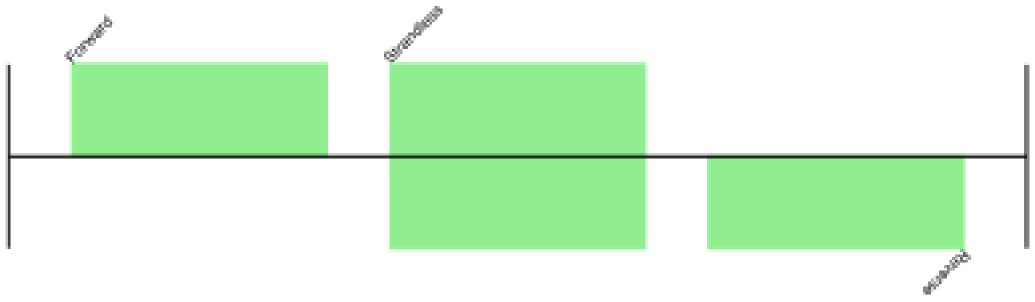
18.1.5 Features without a SeqFeature

In the above example we used a SeqRecord's SeqFeature objects to build our diagram. Sometimes you won't have SeqFeature objects, but just the coordinates for a feature you want to draw. You have to create minimal SeqFeature object, but this is easy:

```
In [9]: from Bio.SeqFeature import SeqFeature, FeatureLocation
        my_seq_feature = SeqFeature(FeatureLocation(50,100),strand=+1)
```

For strand, use +1 for the forward strand, -1 for the reverse strand, and None for both. Here is a short self contained example:

```
In [10]: gdd = GenomeDiagram.Diagram('Test Diagram')
         gdt_features = gdd.new_track(1, greytrack=False)
         gds_features = gdt_features.new_set()

         #Add three features to show the strand options,
         feature = SeqFeature(FeatureLocation(25, 125), strand=+1)
         gds_features.add_feature(feature, name="Forward", label=True)
         feature = SeqFeature(FeatureLocation(150, 250), strand=None)
         gds_features.add_feature(feature, name="Strandless", label=True)
         feature = SeqFeature(FeatureLocation(275, 375), strand=-1)
         gds_features.add_feature(feature, name="Reverse", label=True)

         gdd.draw(format='linear', pagesize=(15*cm,4*cm), fragments=1,
                  start=0, end=400)
         gdd.write("data/GD_labels_default.png", "png")
         Image("data/GD_labels_default.png")
```



The top part of the image in the next subsection shows the output (in the default feature color, pale green).

Notice that we have used the name argument here to specify the caption text for these features. This is discussed in more detail next.

18.1.6 Feature captions

Recall we used the following (where feature was a SeqFeature object) to add a feature to the diagram:

```
In [11]: gd_feature_set.add_feature(feature, color=color, label=True)
```

```
Out[11]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27bfa2bfd0>
```

In the example above the SeqFeature annotation was used to pick a sensible caption for the features. By default the following possible entries under the SeqFeature object's qualifiers dictionary are used: gene, label, name, locus_tag, and product. More simply, you can specify a name directly:

```
In [12]: gd_feature_set.add_feature(feature, color=color, label=True, name="My Gene")
```

```
Out[12]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27bfa3b048>
```

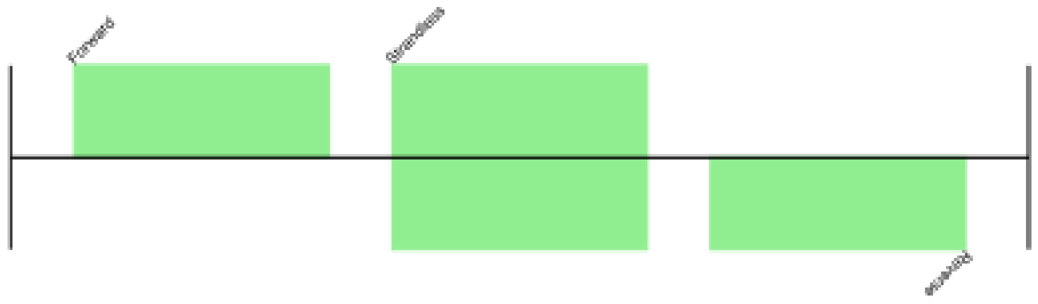
In addition to the caption text for each feature's label, you can also choose the font, position (this defaults to the start of the sigil, you can also choose the middle or at the end) and orientation (for linear diagrams only, where this defaults to rotated by 45 degrees):

```
In [13]: #Large font, parallel with the track
         gd_feature_set.add_feature(feature, label=True, color="green",
                                    label_size=25, label_angle=0)

         #Very small font, perpendicular to the track (towards it)
         gd_feature_set.add_feature(feature, label=True, color="purple",
                                    label_position="end",
                                    label_size=4, label_angle=90)

         #Small font, perpendicular to the track (away from it)
         gd_feature_set.add_feature(feature, label=True, color="blue",
                                    label_position="middle",
                                    label_size=6, label_angle=-90)
```

```
Out[13]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27cc0b2d30>
```

Combining each of these three fragments with the complete example in the previous section should give something like this:

```
In [14]: gdd.draw(format='linear', pagesize=(15*cm,4*cm), fragments=1,
                  start=0, end=400)
         gdd.write("data/GD_labels.png", "png")
         Image("data/GD_labels.png")
```



We've not shown it here, but you can also set label_color to control the label's color.

You'll notice the default font is quite small - this makes sense because you will usually be drawing many (small) features on a page, not just a few large ones as shown here.

18.1.7 Feature sigils

The examples above have all just used the default sigil for the feature, a plain box, which was all that was available in the last publicly released standalone version of GenomeDiagram. Arrow sigils were included when GenomeDiagram was added to Biopython 1.50:

```
In [15]: #Default uses a BOX sigil
         gd_feature_set.add_feature(feature)

         #You can make this explicit:
         gd_feature_set.add_feature(feature, sigil="BOX")

         #Or opt for an arrow:
         gd_feature_set.add_feature(feature, sigil="ARROW")

         #Box with corners cut off (making it an octagon)
         gd_feature_set.add_feature(feature, sigil="OCTO")

         #Box with jagged edges (useful for showing breaks in contains)
         gd_feature_set.add_feature(feature, sigil="JAGGY")

         #Arrow which spans the axis with strand used only for direction
         gd_feature_set.add_feature(feature, sigil="BIGARROW")

Out[15]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27bfa3a7f0>
```

These are shown below. Most sigils fit into a bounding box (as given by the default BOX sigil), either above or below the axis for the forward or reverse strand, or straddling it (double the height) for strand-less features. The BIGARROW sigil is different, always straddling the axis with the direction taken from the feature's stand.

18.1.8 Arrow sigils

We introduced the arrow sigils in the previous section. There are two additional options to adjust the shapes of the arrows, firstly the thickness of the arrow shaft, given as a proportion of the height of the bounding box:

```
In [16]: #Full height shafts, giving pointed boxes:
         gd_feature_set.add_feature(feature, sigil="ARROW", color="brown",
                                    arrowshaft_height=1.0)
         #Or, thin shafts:
         gd_feature_set.add_feature(feature, sigil="ARROW", color="teal",
                                    arrowshaft_height=0.2)
         #Or, very thin shafts:
         gd_feature_set.add_feature(feature, sigil="ARROW", color="darkgreen",
                                    arrowshaft_height=0.1)

Out[16]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27cc0c0400>
```

The results are shown below:

Secondly, the length of the arrow head - given as a proportion of the height of the bounding box (defaulting to 0.5, or 50%):

```
In [17]: #Short arrow heads:
         gd_feature_set.add_feature(feature, sigil="ARROW", color="blue",
                                    arrowhead_length=0.25)
         #Or, longer arrow heads:
```

```
        gd_feature_set.add_feature(feature, sigil="ARROW", color="orange",
                                   arrowhead_length=1)
        #Or, very very long arrow heads (i.e. all head, no shaft, so triangles):
        gd_feature_set.add_feature(feature, sigil="ARROW", color="red",
                                   arrowhead_length=10000)
```

```
Out[17]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27cc0bc208>
```

The results are shown below:

Biopython 1.61 adds a new BIGARROW sigil which always stradles the axis, pointing left for the reverse strand or right otherwise:

```
In [18]: #A large arrow straddling the axis:
         gd_feature_set.add_feature(feature, sigil="BIGARROW")
```

```
Out[18]: <Bio.Graphics.GenomeDiagram._Feature.Feature at 0x7f27bfa2be80>
```

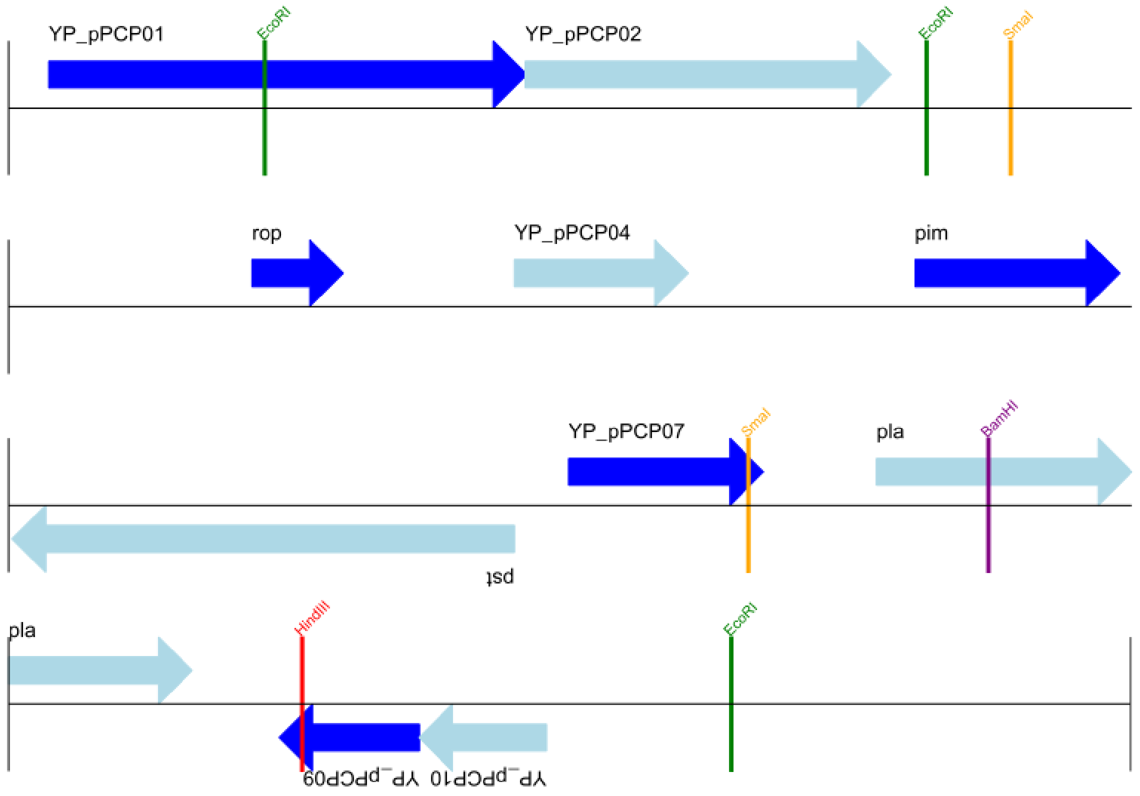
All the shaft and arrow head options shown above for the ARROW sigil can be used for the BIGARROW sigil too.

18.1.9 A nice example

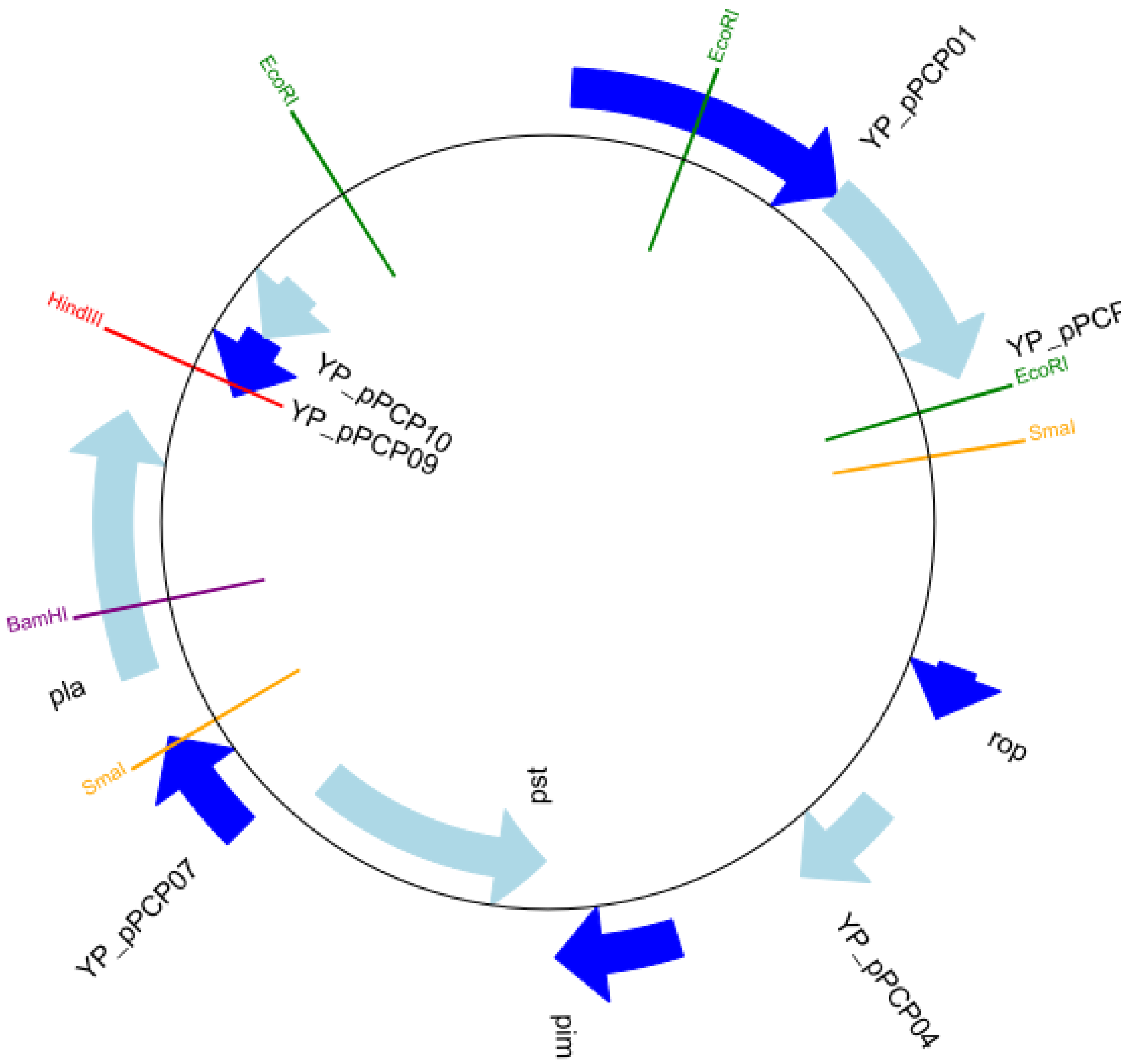
Now let's return to the pPCP1 plasmid from Yersinia pestis biovar Microtus, and the top down approach used above, but take advantage of the sigil options we've now discussed. This time we'll use arrows for the genes, and overlay them with strand-less features (as plain boxes) showing the position of some restriction digest sites.

```
In [19]: record = SeqIO.read("data/NC_005816.gb", "genbank")

         gd_diagram = GenomeDiagram.Diagram(record.id)
         gd_track_for_features = gd_diagram.new_track(1, name="Annotated Features")
         gd_feature_set = gd_track_for_features.new_set()

         for feature in record.features:
             if feature.type != "gene":
                 #Exclude this feature
                 continue
             if len(gd_feature_set) % 2 == 0:
                 color = colors.blue
             else:
                 color = colors.lightblue
             gd_feature_set.add_feature(feature, sigil="ARROW",
                                        color=color, label=True,
                                        label_size = 14, label_angle=0)

         #I want to include some strandless features, so for an example
         #will use EcoRI recognition sites etc.
         for site, name, color in [("GAATTC","EcoRI",colors.green),
                                   ("CCCGGG","SmaI",colors.orange),
                                   ("AAGCTT","HindIII",colors.red),
                                   ("GGATCC","BamHI",colors.purple)]:
             index = 0
             while True:
                 index  = record.seq.find(site, start=index)
                 if index == -1 : break
                 feature = SeqFeature(FeatureLocation(index, index+len(site)))
                 gd_feature_set.add_feature(feature, color=color, name=name,
                                            label=True, label_size = 10,
                                            label_color=color)
                 index += len(site)
```

```
        gd_diagram.draw(format="linear", pagesize='A4', fragments=4,
                        start=0, end=len(record))
        gd_diagram.write("data/plasmid_linear_nice.png", "png")

        Image("data/plasmid_linear_nice.png")
```



```
In [20]: gd_diagram.draw(format="circular", circular=True, pagesize=(20*cm,20*cm),
                        start=0, end=len(record), circle_core = 0.5)
        gd_diagram.write("data/plasmid_circular_nice.png", "png")

        Image("data/plasmid_circular_nice.png")
```



18.1.10 Multiple tracks

All the examples so far have used a single track, but you can have more than one track – for example show the genes on one, and repeat regions on another. In this example we're going to show three phage genomes side by side to scale, inspired by Figure 6 in Proux et al. (2002). We'll need the GenBank files for the following three phage:

- NC_002703 – Lactococcus phage Tuc2009, complete genome (38347 bp)
- AF323668 – Bacteriophage bIL285, complete genome (35538 bp)
- NC_003212 – Listeria innocua Clip11262, complete genome, of which we are focussing only on integrated prophage 5 (similar length).

You can download these using Entrez if you like. For the third record we've worked out where the phage is integrated into the genome, and slice the record to extract it, and must also reverse complement to match the orientation of the first two phage:

```
In [21]: A_rec = SeqIO.read("data/NC_002703.gbk", "gb")
         B_rec = SeqIO.read("data/AF323668.gbk", "gb")
```

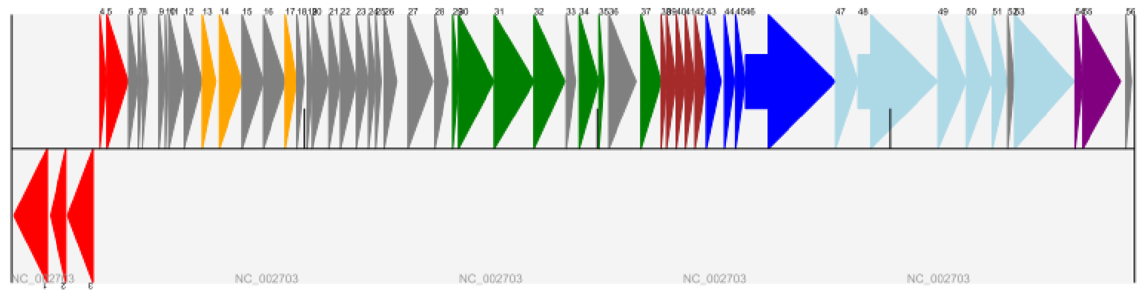
The figure we are imitating used different colors for different gene functions. One way to do this is to edit the GenBank file to record color preferences for each feature - something Sanger's Artemis editor does, and which GenomeDiagram should understand. Here however, we'll just hard code three lists of colors.

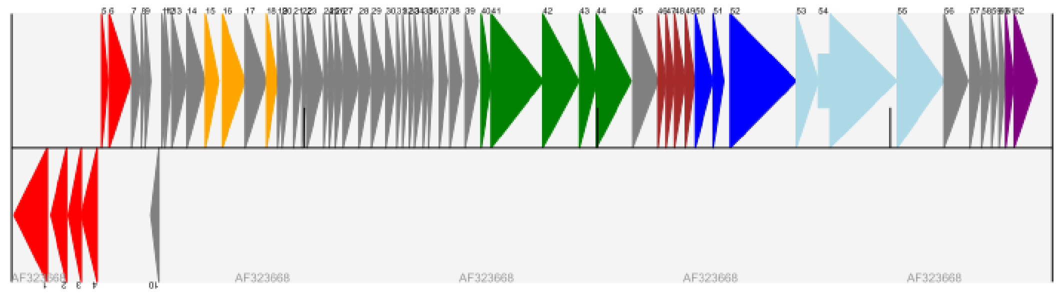
Note that the annotation in the GenBank files doesn't exactly match that shown in Proux et al., they have drawn some unannotated genes.

```
In [22]: from reportlab.lib.colors import red, grey, orange, green, brown, blue, lightblue, purple

         A_colors = [red]*5 + [grey]*7 + [orange]*2 + [grey]*2 + [orange] + [grey]*11 + [green]*4 \
                  + [grey] + [green]*2 + [grey, green] + [brown]*5 + [blue]*4 + [lightblue]*5 \
                  + [grey, lightblue] + [purple]*2 + [grey]
         B_colors = [red]*6 + [grey]*8 + [orange]*2 + [grey] + [orange] + [grey]*21 + [green]*5 \
                  + [grey] + [brown]*4 + [blue]*3 + [lightblue]*3 + [grey]*5 + [purple]*2
```

Now to draw them – this time we add three tracks to the diagram, and also notice they are given different start/end values to reflect their different lengths.

```
In [23]: name = "data/Proux Fig 6"
         gd_diagram = GenomeDiagram.Diagram(name)
         max_len = 0
         for record, gene_colors in zip([A_rec, B_rec], [A_colors, B_colors]):
             max_len = max(max_len, len(record))
             gd_track_for_features = gd_diagram.new_track(1,
                                     name=record.name,
                                     greytrack=True,
                                     start=0, end=len(record))
             gd_feature_set = gd_track_for_features.new_set()

             i = 0
             for feature in record.features:
                 if feature.type != "gene":
                     #Exclude this feature
                     continue
                 gd_feature_set.add_feature(feature, sigil="ARROW",
                                            color=gene_colors[i], label=True,
                                            name = str(i+1),
                                            label_position="start",
                                            label_size = 6, label_angle=0)
                 i+=1

         gd_diagram.draw(format="linear", pagesize='A4', fragments=1,
                         start=0, end=max_len)
         gd_diagram.write(name + ".png", "png")
         Image(name + ".png")
```





I did wonder why in the original manuscript there were no red or orange genes marked in the bottom phage. Another important point is here the phage are shown with different lengths - this is because they are all drawn to the same scale (they are different lengths).

The key difference from the published figure is they have color-coded links between similar proteins – which is what we will do in the next section.

18.1.11 Cross-Links between tracks

Biopython 1.59 added the ability to draw cross links between tracks - both simple linear diagrams as we will show here, but also linear diagrams split into fragments and circular diagrams.

Continuing the example from the previous section inspired by Figure 6 from Proux et al. 2002, we would need a list of cross links between pairs of genes, along with a score or color to use. Realistically you might extract this from a BLAST file computationally, but here I have manually typed them in.

My naming convention continues to refer to the three phage as A, B and C. Here are the links we want to show between A and B, given as a list of tuples (percentage similarity score, gene in A, gene in B).

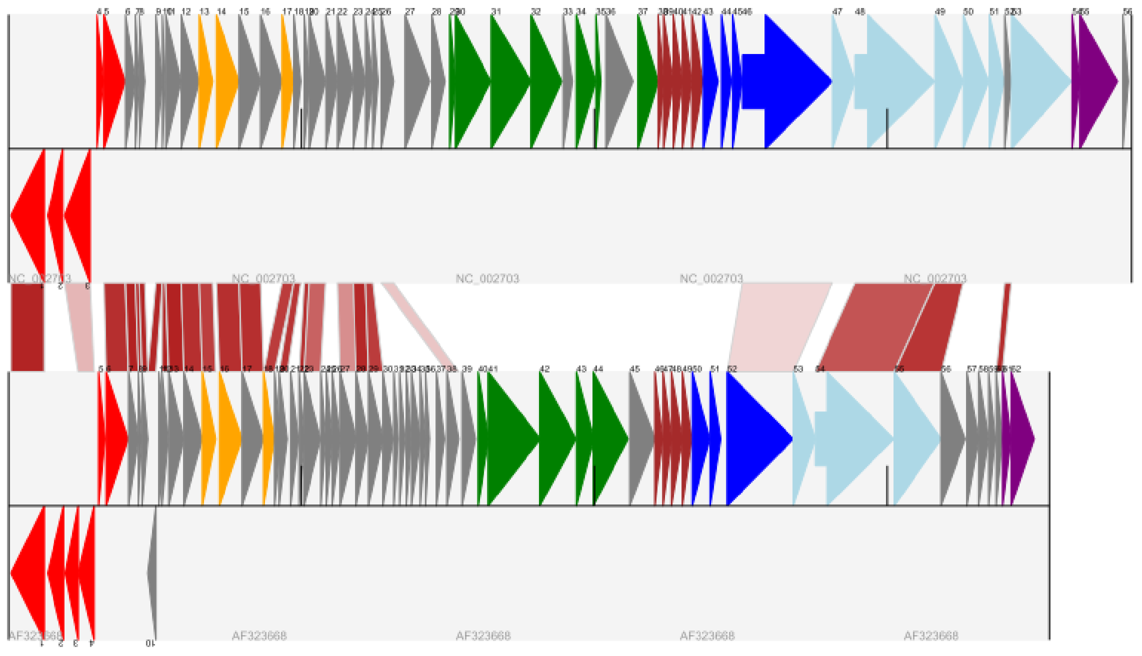
```
In [24]: #Tuc2009 (NC_002703) vs bIL285 (AF323668)
         A_vs_B = [
             (99, "Tuc2009_01", "int"),
             (33, "Tuc2009_03", "orf4"),
             (94, "Tuc2009_05", "orf6"),
             (100,"Tuc2009_06", "orf7"),
             (97, "Tuc2009_07", "orf8"),
             (98, "Tuc2009_08", "orf9"),
             (98, "Tuc2009_09", "orf10"),
             (100,"Tuc2009_10", "orf12"),
             (100,"Tuc2009_11", "orf13"),
             (94, "Tuc2009_12", "orf14"),
             (87, "Tuc2009_13", "orf15"),
             (94, "Tuc2009_14", "orf16"),
             (94, "Tuc2009_15", "orf17"),
             (88, "Tuc2009_17", "rusA"),
             (91, "Tuc2009_18", "orf20"),
             (93, "Tuc2009_19", "orf22"),
             (71, "Tuc2009_20", "orf23"),
             (51, "Tuc2009_22", "orf27"),
             (97, "Tuc2009_23", "orf28"),
             (88, "Tuc2009_24", "orf29"),
             (26, "Tuc2009_26", "orf38"),
             (19, "Tuc2009_46", "orf52"),
             (77, "Tuc2009_48", "orf54"),
             (91, "Tuc2009_49", "orf55"),
             (95, "Tuc2009_52", "orf60"),
         ]
```

For the first and last phage these identifiers are locus tags, for the middle phage there are no locus tags so I've used gene names instead. The following little helper function lets us lookup a feature using either a locus tag or gene name:

```
In [25]: def get_feature(features, id, tags=["locus_tag", "gene"]):
             """Search list of SeqFeature objects for an identifier under the given tags."""
             for f in features:
                 for key in tags:
                     #tag may not be present in this feature
                     for x in f.qualifiers.get(key, []):
                         if x == id:
                              return f
             raise KeyError(id)
```

We can now turn those list of identifier pairs into SeqFeature pairs, and thus find their location co-ordinates. We can now add all that code and the following snippet to the previous example (just before the gd_diagram.draw(...) line – see the finished example script Proux_et_al_2002_Figure_6.py included in the Doc/examples folder of the Biopython source code) to add cross links to the figure:

```
In [ ]: from Bio.Graphics.GenomeDiagram import CrossLink
        from reportlab.lib import colors
        #Note it might have been clearer to assign the track numbers explicitly...
        for rec_X, tn_X, rec_Y, tn_Y, X_vs_Y in [(A_rec, 2, B_rec, 1, A_vs_B)]:
            track_X = gd_diagram.tracks[tn_X]
            track_Y = gd_diagram.tracks[tn_Y]
            for score, id_X, id_Y in X_vs_Y:
                feature_X = get_feature(rec_X.features, id_X)
                feature_Y = get_feature(rec_Y.features, id_Y)
                color = colors.linearlyInterpolatedColor(colors.white, colors.firebrick, 0, 100, scor
                link_xy = CrossLink((track_X, feature_X.location.start, feature_X.location.end),
                                    (track_Y, feature_Y.location.start, feature_Y.location.end),
                                    color, colors.lightgrey)
                gd_diagram.cross_track_links.append(link_xy)
        gd_diagram.draw(format="linear", pagesize='A4', fragments=1,
                        start=0, end=max_len)
        gd_diagram.write("data/cross.png", "png")
        Image("data/cross.png")
```



There are several important pieces to this code. First the GenomeDiagram object has a cross_track_links attribute which is just a list of CrossLink objects. Each CrossLink object takes two sets of track-specific co-ordinates (here given as tuples, you can alternatively use a GenomeDiagram.Feature object instead). You can optionally supply a colour, border color, and say if this link should be drawn flipped (useful for showing inversions).

You can also see how we turn the BLAST percentage identity score into a colour, interpolating between white (0%) and a dark red (100%). In this example we don't have any problems with overlapping cross-links. One way to tackle that is to use transparency in ReportLab, by using colors with their alpha channel set. However, this kind of shaded color scheme combined with overlap transparency would be difficult to interpret. The result:

There is still a lot more that can be done within Biopython to help improve this figure. First of all, the cross links in this case are between proteins which are drawn in a strand specific manor. It can help to add a background region (a feature using the 'BOX' sigil) on the feature track to extend the cross link. Also, we could reduce the vertical height of the feature tracks to allocate more to the links instead – one way to do that is to allocate space for empty tracks. Furthermore, in cases like this where there are no large gene overlaps, we can use the axis-straddling BIGARROW sigil, which allows us to further reduce the vertical space needed for the track. These improvements are demonstrated in the example script Proux_et_al_2002_Figure_6.py.

Beyond that, finishing touches you might want to do manually in a vector image editor include fine tuning the placement of gene labels, and adding other custom annotation such as highlighting particular regions.

Although not really necessary in this example since none of the cross-links overlap, using a transparent color in ReportLab is a very useful technique for superimposing multiple links. However, in this case a shaded color scheme should be avoided.

18.2 Chromosomes

The Bio.Graphics.BasicChromosome module allows drawing of chromosomes. There is an example in Jupe et al. (2012) (open access) using colors to highlight different gene families.

18.2.1 Simple Chromosomes

Important: To continue this example you have first to download a few chromosomes from Arabidopsis thaliana, the code to help you is here:

Very important: This is slow and clogs the network, you only need to do this once (even if you close the notebook as the download will be persistent)

```
In [ ]: from ftplib import FTP
        ftp = FTP('ftp.ncbi.nlm.nih.gov')
        print("Logging in")
        ftp.login()
        ftp.cwd('genomes/archive/old_genbank/A_thaliana/OLD/')
        print("Starting download - This can be slow!")
        for chro, name in [
                ("CHR_I", "NC_003070.fna"), ("CHR_I", "NC_003070.gbk"),
                ("CHR_II", "NC_003071.fna"), ("CHR_II", "NC_003071.gbk"),
                ("CHR_III", "NC_003072.fna"), ("CHR_III", "NC_003072.gbk"),
                ("CHR_IV", "NC_003073.fna"), ("CHR_IV", "NC_003073.gbk"),
                ("CHR_V", "NC_003074.fna"), ("CHR_V", "NC_003074.gbk")]:
            print("Downloading", chro, name)
            ftp.retrbinary('RETR %s/%s' % (chro, name), open('data/%s' % name, 'wb').write)
        ftp.quit()
        print('Done')
```

Here is a very simple example - for which we'll use Arabidopsis thaliana.

You can skip this bit, but first I downloaded the five sequenced chromosomes from the NCBI's FTP site (per the code above) and then parsed them with Bio.SeqIO to find out their lengths. You could use the GenBank files for this, but it is faster to use the FASTA files for the whole chromosomes:

```
In [ ]: from Bio import SeqIO
        entries = [("Chr I", "NC_003070.fna"),
                   ("Chr II", "NC_003071.fna"),
                   ("Chr III", "NC_003072.fna"),
                   ("Chr IV", "NC_003073.fna"),
                   ("Chr V", "NC_003074.fna")]
        for (name, filename) in entries:
           record = SeqIO.read("data/" + filename, "fasta")
           print(name, len(record))
```

This gave the lengths of the five chromosomes, which we'll now use in the following short demonstration of the BasicChromosome module:

```
In [ ]: from reportlab.lib.units import cm
        from Bio.Graphics import BasicChromosome

        entries = [("Chr I", 30432563),
                   ("Chr II", 19705359),
                   ("Chr III", 23470805),
                   ("Chr IV", 18585042),
                   ("Chr V", 26992728)]

        max_len = 30432563 #Could compute this
```

```
telomere_length = 1000000 #For illustration

chr_diagram = BasicChromosome.Organism(output_format="png")
chr_diagram.page_size = (29.7*cm, 21*cm) #A4 landscape

for name, length in entries:
    cur_chromosome = BasicChromosome.Chromosome(name)
    #Set the scale to the MAXIMUM length plus the two telomeres in bp,
    #want the same scale used on all five chromosomes so they can be
    #compared to each other
    cur_chromosome.scale_num = max_len + 2 * telomere_length

    #Add an opening telomere
    start = BasicChromosome.TelomereSegment()
    start.scale = telomere_length
    cur_chromosome.add(start)

    #Add a body - using bp as the scale length here.
    body = BasicChromosome.ChromosomeSegment()
    body.scale = length
    cur_chromosome.add(body)

    #Add a closing telomere
    end = BasicChromosome.TelomereSegment(inverted=True)
    end.scale = telomere_length
    cur_chromosome.add(end)

    #This chromosome is done
    chr_diagram.add(cur_chromosome)

chr_diagram.draw("data/simple_chrom.png", "Arabidopsis thaliana")
Image("data/simple_chrom.png")
```

This example is deliberately short and sweet. The next example shows the location of features of interest.

Continuing from the previous example, let's also show the tRNA genes. We'll get their locations by parsing the GenBank files for the five Arabidopsis thaliana chromosomes. You'll need to download these files from the NCBI FTP site.

```
In [ ]: entries = [("Chr I", "NC_003070.gbk"),
                   ("Chr II", "NC_003071.gbk"),
                   ("Chr III", "NC_003072.gbk"),
                   ("Chr IV", "NC_003073.gbk"),
                   ("Chr V", "NC_003074.gbk")]

        max_len = 30432563 #Could compute this
        telomere_length = 1000000 #For illustration

        chr_diagram = BasicChromosome.Organism(output_format="png")
        chr_diagram.page_size = (29.7*cm, 21*cm) #A4 landscape

        for index, (name, filename) in enumerate(entries):
            record = SeqIO.read("data/" + filename,"genbank")
            length = len(record)
            features = [f for f in record.features if f.type=="tRNA"]
            #Record an Artemis style integer color in the feature's qualifiers,
            #1 = Black, 2 = Red, 3 = Green, 4 = blue, 5 =cyan, 6 = purple
            for f in features: f.qualifiers["color"] = [index+2]

            cur_chromosome = BasicChromosome.Chromosome(name)
            #Set the scale to the MAXIMUM length plus the two telomeres in bp,
            #want the same scale used on all five chromosomes so they can be
            #compared to each other
            cur_chromosome.scale_num = max_len + 2 * telomere_length

            #Add an opening telomere
            start = BasicChromosome.TelomereSegment()
            start.scale = telomere_length
            cur_chromosome.add(start)

            #Add a body - again using bp as the scale length here.
            body = BasicChromosome.AnnotatedChromosomeSegment(length, features)
            body.scale = length
            cur_chromosome.add(body)

            #Add a closing telomere
            end = BasicChromosome.TelomereSegment(inverted=True)
            end.scale = telomere_length
            cur_chromosome.add(end)

            #This chromosome is done
            chr_diagram.add(cur_chromosome)

        chr_diagram.draw("data/tRNA_chrom.png", "Arabidopsis thaliana")
        Image("data/tRNA_chrom.png")
```

CHAPTER 19

KEGG

KEGG (http://www.kegg.jp/) is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information, especially large-scale molecular datasets generated by genome sequencing and other high-throughput experimental technologies.

Please note that the KEGG parser implementation in Biopython is incomplete. While the KEGG website indicates many flat file formats, only parsers and writers for compound, enzyme, and map are currently implemented. However, a generic parser is implemented to handle the other formats.

19.1 Parsing KEGG records

Parsing a KEGG record is as simple as using any other file format parser in Biopython. (Before running the following codes, please open http://rest.kegg.jp/get/ec:5.4.2.2 with your web browser and save it as ec_5.4.2.2.txt.)

```
In [2]: !wget http://rest.kegg.jp/get/ec:5.4.2.2 -O ec_5.4.2.2.txt
```

```
--2016-01-12 20:44:45--  http://rest.kegg.jp/get/ec:5.4.2.2
Resolving rest.kegg.jp (rest.kegg.jp)... 133.103.200.77
Connecting to rest.kegg.jp (rest.kegg.jp)|133.103.200.77|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: ‘ec_5.4.2.2.txt’

ec_5.4.2.2.txt          [   <=>                ] 105.99K   141KB/s   in 0.8s

2016-01-12 20:44:46 (141 KB/s) - ‘ec_5.4.2.2.txt’ saved [108534]
```

```
In [3]: from Bio.KEGG import Enzyme

        records = Enzyme.parse(open("ec_5.4.2.2.txt"))

        record = list(records)[0]

        record.classname
```

```
Out[3]: ['Isomerases;',
         'Intramolecular transferases;',
         'Phosphotransferases (phosphomutases)']
```

```
In [4]: record.entry
```

```
Out[4]: '5.4.2.2'
```

The following section will shows how to download the above enzyme using the KEGG api as well as how to use the generic parser with data that does not have a custom parser implemented.

19.2 Querying the KEGG API

Biopython has full support for the querying of the KEGG api. Querying all KEGG endpoints are supported; all methods documented by KEGG (http://www.kegg.jp/kegg/rest/keggapi.html) are supported. The interface has some validation of queries which follow rules defined on the KEGG site. However, invalid queries which return a 400 or 404 must be handled by the user.

First, here is how to extend the above example by downloading the relevant enzyme and passing it through the Enzyme parser.

```
In [5]: from Bio.KEGG import REST

        from Bio.KEGG import Enzyme

        request = REST.kegg_get("ec:5.4.2.2")

        open("ec_5.4.2.2.txt", 'w').write(request.read().decode("utf-8"))
```

```
Out[5]: 108534
```

```
In [6]: records = Enzyme.parse(open("ec_5.4.2.2.txt"))

        record = list(records)[0]

        record.classname
```

```
Out[6]: ['Isomerases;',
         'Intramolecular transferases;',
         'Phosphotransferases (phosphomutases)']
```

```
In [7]: record.entry
```

```
Out[7]: '5.4.2.2'
```

Now, here's a more realistic example which shows a combination of querying the KEGG API. This will demonstrate how to extract a unique set of all human pathway gene symbols which relate to DNA repair. The steps that need to be taken to do so are as follows. First, we need to get a list of all human pathways. Secondly, we need to filter those for ones which relate to "repair". Lastly, we need to get a list of all the gene symbols in all repair pathways.

```
In [8]: from Bio.KEGG import REST

        human_pathways = REST.kegg_list("pathway", "hsa").read()

        human_pathways.decode("utf-8").split("\n")[0:5]
```

```
Out[8]: ['path:hsa00010\tGlycolysis / Gluconeogenesis - Homo sapiens (human)',
         'path:hsa00020\tCitrate cycle (TCA cycle) - Homo sapiens (human)',
         'path:hsa00030\tPentose phosphate pathway - Homo sapiens (human)',
         'path:hsa00040\tPentose and glucuronate interconversions - Homo sapiens (human)',
         'path:hsa00051\tFructose and mannose metabolism - Homo sapiens (human)']
```

```
In [9]: # Filter all human pathways for repair pathways
        repair_pathways = []
        for line in human_pathways.decode("utf-8").rstrip().split("\n"):
            entry, description = line.split("\t")
            if "repair" in description:
                repair_pathways.append(entry)

        repair_pathways
```

```
Out[9]: ['path:hsa03410', 'path:hsa03420', 'path:hsa03430']
```

```
In [10]: # Get the genes for pathways and add them to a list
         repair_genes = []
         for pathway in repair_pathways:
             pathway_file = REST.kegg_get(pathway).read()  # query and read each pathway

             # iterate through each KEGG pathway file, keeping track of which section
             # of the file we're in, only read the gene in each pathway
             current_section = None
             for line in pathway_file.decode("utf-8").rstrip().split("\n"):
                 section = line[:12].strip()  # section names are within 12 columns
                 if not section == "":
                     current_section = section

                 if current_section == "GENE":
                     gene_identifiers, gene_description = line[12:].split("; ")
                     gene_id, gene_symbol = gene_identifiers.split()

                     if not gene_symbol in repair_genes:
                         repair_genes.append(gene_symbol)

         print("There are %d repair pathways and %d repair genes. The genes are:" % \
                 (len(repair_pathways), len(repair_genes)))
         print(", ".join(repair_genes))
```

```
There are 3 repair pathways and 78 repair genes. The genes are:
OGG1, NTHL1, NEIL1, NEIL2, NEIL3, UNG, TDG, SMUG1, MUTYH, MPG, MBD4, APEX1, APEX2, POLB, POLL, HMGB1,
```

The KEGG API wrapper is compatible with all endpoints. Usage is essentially replacing all slashes in the url with commas and using that list as arguments to the corresponding method in the KEGG module. Here are a few examples from the api documentation (http://www.kegg.jp/kegg/docs/keggapi.html).

```
/list/hsa:10458+ece:Z5100          -> REST.kegg_list(["hsa:10458", "ece:Z5100"])
/find/compound/300-310/mol_weight   -> REST.kegg_find("compound", "300-310", "mol_
↪weight")
/get/hsa:10458+ece:Z5100/aaseq     -> REST.kegg_get(["hsa:10458", "ece:Z5100"],
↪"aaseq")
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 20

Cookbook – Cool things to do with it

Biopython now has two collections of "cookbook" examples – this chapter (which has been included in this tutorial for many years and has gradually grown), and http://biopython.org/wiki/Category:Cookbook which is a user contributed collection on our wiki.

We're trying to encourage Biopython users to contribute their own examples to the wiki. In addition to helping the community, one direct benefit of sharing an example like this is that you could also get some feedback on the code from other Biopython users and developers - which could help you improve all your Python code.

In the long term, we may end up moving all of the examples in this chapter to the wiki, or elsewhere within the tutorial.

20.1 Working with sequence files

This section shows some more examples of sequence input/output, using the `Bio.SeqIO` module described in Chapter [chapter:Bio.SeqIO].

20.1.1 Filtering a sequence file

Often you'll have a large file with many sequences in it (e.g. FASTA file or genes, or a FASTQ or SFF file of reads), a separate shorter list of the IDs for a subset of sequences of interest, and want to make a new sequence file for this subset.

Let's say the list of IDs is in a simple text file, as the first word on each line. This could be a tabular file where the first column is the ID. Try something like this:

```
from Bio import SeqIO
input_file = "big_file.sff"
id_file = "short_list.txt"
output_file = "short_list.sff"
wanted = set(line.rstrip("\n").split(None,1)[0] for line in open(id_file))
print("Found %i unique identifiers in %s" % (len(wanted), id_file))
records = (r for r in SeqIO.parse(input_file, "sff") if r.id in wanted)
count = SeqIO.write(records, output_file, "sff")
```

```
print("Saved %i records from %s to %s" % (count, input_file, output_file))
if count < len(wanted):
    print("Warning %i IDs not found in %s" % (len(wanted)-count, input_file))
```

Note that we use a Python `set` rather than a `list`, this makes testing membership faster.

20.1.2 Producing randomised genomes

Let's suppose you are looking at genome sequence, hunting for some sequence feature – maybe extreme local GC% bias, or possible restriction digest sites. Once you've got your Python code working on the real genome it may be sensible to try running the same search on randomised versions of the same genome for statistical analysis (after all, any "features" you've found could just be there just by chance).

For this discussion, we'll use the GenBank file for the pPCP1 plasmid from *Yersinia pestis biovar Microtus*. The file is included with the Biopython unit tests under the GenBank folder, or you can get it from our website, `` `NC_005816.gb `` <http://biopython.org/SRC/biopython/Tests/GenBank/NC_005816.gb>'__. This file contains one and only one record, so we can read it in as a `SeqRecord` using the `Bio.SeqIO.read()` function:

```
In [2]: from Bio import SeqIO
        original_rec = SeqIO.read("data/NC_005816.gb", "genbank")
```

So, how can we generate a shuffled versions of the original sequence? I would use the built in Python `random` module for this, in particular the function `random.shuffle` – but this works on a Python list. Our sequence is a `Seq` object, so in order to shuffle it we need to turn it into a list:

```
In [3]: import random
        nuc_list = list(original_rec.seq)
        random.shuffle(nuc_list) #acts in situ!
```

Now, in order to use `Bio.SeqIO` to output the shuffled sequence, we need to construct a new `SeqRecord` with a new `Seq` object using this shuffled list. In order to do this, we need to turn the list of nucleotides (single letter strings) into a long string – the standard Python way to do this is with the string object's join method.

```
In [4]: from Bio.Seq import Seq
        from Bio.SeqRecord import SeqRecord
        shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet),
            id="Shuffled", description="Based on %s" % original_rec.id)
```

Let's put all these pieces together to make a complete Python script which generates a single FASTA file containing 30 randomly shuffled versions of the original sequence.

This first version just uses a big for loop and writes out the records one by one (using the `SeqRecord`'s format method described in Section [sec:Bio.SeqIO-and-StringIO]):

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

original_rec = SeqIO.read("NC_005816.gb","genbank")

handle = open("shuffled.fasta", "w")
for i in range(30):
    nuc_list = list(original_rec.seq)
    random.shuffle(nuc_list)
    shuffled_rec = SeqRecord(Seq("".join(nuc_list), original_rec.seq.alphabet), \
                             id="Shuffled%i" % (i+1), \
                             description="Based on %s" % original_rec.id)
    handle.write(shuffled_rec.format("fasta"))
handle.close()
```

Personally I prefer the following version using a function to shuffle the record and a generator expression instead of the for loop:

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

def make_shuffle_record(record, new_id):
    nuc_list = list(record.seq)
    random.shuffle(nuc_list)
    return SeqRecord(Seq("".join(nuc_list), record.seq.alphabet), \
           id=new_id, description="Based on %s" % original_rec.id)

original_rec = SeqIO.read("NC_005816.gb","genbank")
shuffled_recs = (make_shuffle_record(original_rec, "Shuffled%i" % (i+1)) \
                 for i in range(30))
handle = open("shuffled.fasta", "w")
SeqIO.write(shuffled_recs, handle, "fasta")
handle.close()
```

20.1.3 Translating a FASTA file of CDS entries

Suppose you've got an input file of CDS entries for some organism, and you want to generate a new FASTA file containing their protein sequences. i.e. Take each nucleotide sequence from the original file, and translate it. Back in Section [sec:translation] we saw how to use the `Seq` object's `translate method`, and the optional `cds` argument which enables correct translation of alternative start codons.

We can combine this with `Bio.SeqIO` as shown in the reverse complement example in Section [sec:SeqIO-reverse-complement]. The key point is that for each nucleotide `SeqRecord`, we need to create a protein `SeqRecord` - and take care of naming it.

You can write you own function to do this, choosing suitable protein identifiers for your sequences, and the appropriate genetic code. In this example we just use the default table and add a prefix to the identifier:

```
from Bio.SeqRecord import SeqRecord
def make_protein_record(nuc_record):
    """Returns a new SeqRecord with the translated sequence (default table)."""
    return SeqRecord(seq = nuc_record.seq.translate(cds=True), \
                     id = "trans_" + nuc_record.id, \
                     description = "translation of CDS, using default table")
```

We can then use this function to turn the input nucleotide records into protein records ready for output. An elegant way and memory efficient way to do this is with a generator expression:

```
from Bio import SeqIO
proteins = (make_protein_record(nuc_rec) for nuc_rec in \
            SeqIO.parse("coding_sequences.fasta", "fasta"))
SeqIO.write(proteins, "translations.fasta", "fasta")
```

This should work on any FASTA file of complete coding sequences. If you are working on partial coding sequences, you may prefer to use `nuc_record.seq.translate(to_stop=True)` in the example above, as this wouldn't check for a valid start codon etc.

20.1.4 Making the sequences in a FASTA file upper case

Often you'll get data from collaborators as FASTA files, and sometimes the sequences can be in a mixture of upper and lower case. In some cases this is deliberate (e.g. lower case for poor quality regions), but usually it is not important. You may want to edit the file to make everything consistent (e.g. all upper case), and you can do this easily using the `upper()` method of the `SeqRecord` object (added in Biopython 1.55):

```
from Bio import SeqIO
records = (rec.upper() for rec in SeqIO.parse("mixed.fas", "fasta"))
count = SeqIO.write(records, "upper.fas", "fasta")
print("Converted %i records to upper case" % count)
```

How does this work? The first line is just importing the `Bio.SeqIO` module. The second line is the interesting bit – this is a Python generator expression which gives an upper case version of each record parsed from the input file (`mixed.fas`). In the third line we give this generator expression to the `Bio.SeqIO.write()` function and it saves the new upper cases records to our output file (`upper.fas`).

The reason we use a generator expression (rather than a list or list comprehension) is this means only one record is kept in memory at a time. This can be really important if you are dealing with large files with millions of entries.

20.1.5 Sorting a sequence file

Suppose you wanted to sort a sequence file by length (e.g. a set of contigs from an assembly), and you are working with a file format like FASTA or FASTQ which `Bio.SeqIO` can read, write (and index).

If the file is small enough, you can load it all into memory at once as a list of `SeqRecord` objects, sort the list, and save it:

```
from Bio import SeqIO
records = list(SeqIO.parse("ls_orchid.fasta","fasta"))
records.sort(cmp=lambda x,y: cmp(len(x),len(y)))
SeqIO.write(records, "sorted_orchids.fasta", "fasta")
```

The only clever bit is specifying a comparison function for how to sort the records (here we sort them by length). If you wanted the longest records first, you could flip the comparison or use the reverse argument:

```
from Bio import SeqIO
records = list(SeqIO.parse("ls_orchid.fasta","fasta"))
records.sort(cmp=lambda x,y: cmp(len(y),len(x)))
SeqIO.write(records, "sorted_orchids.fasta", "fasta")
```

Now that's pretty straight forward - but what happens if you have a very large file and you can't load it all into memory like this? For example, you might have some next-generation sequencing reads to sort by length. This can be solved using the `Bio.SeqIO.index()` function.

```
from Bio import SeqIO
#Get the lengths and ids, and sort on length
len_and_ids = sorted((len(rec), rec.id) for rec in \
                     SeqIO.parse("ls_orchid.fasta","fasta"))
ids = reversed([id for (length, id) in len_and_ids])
del len_and_ids #free this memory
record_index = SeqIO.index("ls_orchid.fasta", "fasta")
records = (record_index[id] for id in ids)
SeqIO.write(records, "sorted.fasta", "fasta")
```

First we scan through the file once using `Bio.SeqIO.parse()`, recording the record identifiers and their lengths in a list of tuples. We then sort this list to get them in length order, and discard the lengths. Using this sorted list of identifiers `Bio.SeqIO.index()` allows us to retrieve the records one by one, and we pass them to `Bio.SeqIO.write()` for output.

These examples all use `Bio.SeqIO` to parse the records into `SeqRecord` objects which are output using `Bio.SeqIO.write()`. What if you want to sort a file format which `Bio.SeqIO.write()` doesn't support, like the plain text SwissProt format? Here is an alternative solution using the `get_raw()` method added to `Bio.SeqIO.index()` in Biopython 1.54 (see Section [sec:seqio-index-getraw]).

```
from Bio import SeqIO
#Get the lengths and ids, and sort on length
len_and_ids = sorted((len(rec), rec.id) for rec in \
                     SeqIO.parse("ls_orchid.fasta","fasta"))
ids = reversed([id for (length, id) in len_and_ids])
del len_and_ids #free this memory
record_index = SeqIO.index("ls_orchid.fasta", "fasta")
handle = open("sorted.fasta", "w")
for id in ids:
    handle.write(record_index.get_raw(id))
handle.close()
```

As a bonus, because it doesn't parse the data into `SeqRecord` objects a second time it should be faster.

20.1.6 Simple quality filtering for FASTQ files

The FASTQ file format was introduced at Sanger and is now widely used for holding nucleotide sequencing reads together with their quality scores. FASTQ files (and the related QUAL files) are an excellent example of per-letter-annotation, because for each nucleotide in the sequence there is an associated quality score. Any per-letter-annotation is held in a `SeqRecord` in the `letter_annotations` dictionary as a list, tuple or string (with the same number of elements as the sequence length).

One common task is taking a large set of sequencing reads and filtering them (or cropping them) based on their quality scores. The following example is very simplistic, but should illustrate the basics of working with quality data in a `SeqRecord` object. All we are going to do here is read in a file of FASTQ data, and filter it to pick out only those records whose PHRED quality scores are all above some threshold (here 20).

For this example we'll use some real data downloaded from the ENA sequence read archive, ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz (2MB) which unzips to a 19MB file `SRR020192.fastq`. This is some Roche 454 GS FLX single end data from virus infected California sea lions (see http://www.ebi.ac.uk/ena/data/view/SRS004476 for details).

First, let's count the reads:

```
from Bio import SeqIO
count = 0
for rec in SeqIO.parse("SRR020192.fastq", "fastq"):
    count += 1
print("%i reads" % count)
```

Now let's do a simple filtering for a minimum PHRED quality of 20:

```
from Bio import SeqIO
good_reads = (rec for rec in \
              SeqIO.parse("SRR020192.fastq", "fastq") \
              if min(rec.letter_annotations["phred_quality"]) >= 20)
count = SeqIO.write(good_reads, "good_quality.fastq", "fastq")
print("Saved %i reads" % count)
```

This pulled out only 14580 reads out of the 41892 present. A more sensible thing to do would be to quality trim the reads, but this is intended as an example only.

FASTQ files can contain millions of entries, so it is best to avoid loading them all into memory at once. This example uses a generator expression, which means only one `SeqRecord` is created at a time - avoiding any memory limitations.

20.1.7 Trimming off primer sequences

For this example we're going to pretend that `GATGACGGTGT` is a 5' primer sequence we want to look for in some FASTQ formatted read data. As in the example above, we'll use the `SRR020192.fastq` file downloaded from the ENA (ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz). The same approach would work with any other supported file format (e.g. FASTA files).

This code uses `Bio.SeqIO` with a generator expression (to avoid loading all the sequences into memory at once), and the `Seq` object's `startswith` method to see if the read starts with the primer sequence:

```
from Bio import SeqIO
primer_reads = (rec for rec in \
                SeqIO.parse("SRR020192.fastq", "fastq") \
                if rec.seq.startswith("GATGACGGTGT"))
count = SeqIO.write(primer_reads, "with_primer.fastq", "fastq")
print("Saved %i reads" % count)
```

That should find 13819 reads from `SRR014849.fastq` and save them to a new FASTQ file, `with_primer.fastq`.

Now suppose that instead you wanted to make a FASTQ file containing these reads but with the primer sequence removed? That's just a small change as we can slice the `SeqRecord` (see Section [sec:SeqRecord-slicing]) to remove the first eleven letters (the length of our primer):

```
from Bio import SeqIO
trimmed_primer_reads = (rec[11:] for rec in \
                        SeqIO.parse("SRR020192.fastq", "fastq") \
                        if rec.seq.startswith("GATGACGGTGT"))
count = SeqIO.write(trimmed_primer_reads, "with_primer_trimmed.fastq", "fastq")
print("Saved %i reads" % count)
```

Again, that should pull out the 13819 reads from `SRR020192.fastq`, but this time strip off the first ten characters, and save them to another new FASTQ file, `with_primer_trimmed.fastq`.

Finally, suppose you want to create a new FASTQ file where these reads have their primer removed, but all the other reads are kept as they were? If we want to still use a generator expression, it is probably clearest to define our own trim function:

```
from Bio import SeqIO
def trim_primer(record, primer):
    if record.seq.startswith(primer):
        return record[len(primer):]
    else:
        return record

trimmed_reads = (trim_primer(record, "GATGACGGTGT") for record in \
                 SeqIO.parse("SRR020192.fastq", "fastq"))
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
print("Saved %i reads" % count)
```

This takes longer, as this time the output file contains all 41892 reads. Again, we're used a generator expression to avoid any memory problems. You could alternatively use a generator function rather than a generator expression.

```
from Bio import SeqIO
def trim_primers(records, primer):
    """Removes perfect primer sequences at start of reads.

    This is a generator function, the records argument should
    be a list or iterator returning SeqRecord objects.
    """
    len_primer = len(primer) #cache this for later
    for record in records:
        if record.seq.startswith(primer):
            yield record[len_primer:]
        else:
            yield record

original_reads = SeqIO.parse("SRR020192.fastq", "fastq")
trimmed_reads = trim_primers(original_reads, "GATGACGGTGT")
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
print("Saved %i reads" % count)
```

This form is more flexible if you want to do something more complicated where only some of the records are retained – as shown in the next example.

20.1.8 Trimming off adaptor sequences

This is essentially a simple extension to the previous example. We are going to going to pretend `GATGACGGTGT` is an adaptor sequence in some FASTQ formatted read data, again the `SRR020192.fastq` file from the NCBI (ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz).

This time however, we will look for the sequence *anywhere* in the reads, not just at the very beginning:

```
from Bio import SeqIO

def trim_adaptors(records, adaptor):
    """Trims perfect adaptor sequences.

    This is a generator function, the records argument should
    be a list or iterator returning SeqRecord objects.
    """
    len_adaptor = len(adaptor) #cache this for later
    for record in records:
        index = record.seq.find(adaptor)
        if index == -1:
            #adaptor not found, so won't trim
            yield record
        else:
            #trim off the adaptor
            yield record[index+len_adaptor:]

original_reads = SeqIO.parse("SRR020192.fastq", "fastq")
trimmed_reads = trim_adaptors(original_reads, "GATGACGGTGT")
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
print("Saved %i reads" % count)
```

Because we are using a FASTQ input file in this example, the `SeqRecord` objects have per-letter-annotation for the quality scores. By slicing the `SeqRecord` object the appropriate scores are used on the trimmed records, so we can output them as a FASTQ file too.

Compared to the output of the previous example where we only looked for a primer/adaptor at the start of each read, you may find some of the trimmed reads are quite short after trimming (e.g. if the adaptor was found in the middle rather than near the start). So, let's add a minimum length requirement as well:

```
from Bio import SeqIO

def trim_adaptors(records, adaptor, min_len):
    """Trims perfect adaptor sequences, checks read length.

    This is a generator function, the records argument should
    be a list or iterator returning SeqRecord objects.
    """
    len_adaptor = len(adaptor) #cache this for later
    for record in records:
        len_record = len(record) #cache this for later
        if len(record) < min_len:
           #Too short to keep
           continue
        index = record.seq.find(adaptor)
        if index == -1:
            #adaptor not found, so won't trim
            yield record
        elif len_record - index - len_adaptor >= min_len:
            #after trimming this will still be long enough
            yield record[index+len_adaptor:]

original_reads = SeqIO.parse("SRR020192.fastq", "fastq")
trimmed_reads = trim_adaptors(original_reads, "GATGACGGTGT", 100)
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
print("Saved %i reads" % count)
```

By changing the format names, you could apply this to FASTA files instead. This code also could be extended to do a fuzzy match instead of an exact match (maybe using a pairwise alignment, or taking into account the read quality scores), but that will be much slower.

20.1.9 Converting FASTQ files

Back in Section [sec:SeqIO-conversion] we showed how to use `Bio.SeqIO` to convert between two file formats. Here we'll go into a little more detail regarding FASTQ files which are used in second generation DNA sequencing. Please refer to Cock *et al.* (2009) @cock2010 for a longer description. FASTQ files store both the DNA sequence (as a string) and the associated read qualities.

PHRED scores (used in most FASTQ files, and also in QUAL files, ACE files and SFF files) have become a *de facto* standard for representing the probability of a sequencing error (here denoted by P_e) at a given base using a simple base ten log transformation:

$$Q_{\text{PHRED}} = -10 \times \log_{10}(P_e)$$

This means a wrong read ($P_e = 1$) gets a PHRED quality of 0, while a very good read like $P_e = 0.00001$ gets a PHRED quality of 50. While for raw sequencing data qualities higher than this are rare, with post processing such as read mapping or assembly, qualities of up to about 90 are possible (indeed, the MAQ tool allows for PHRED scores in the range 0 to 93 inclusive).

The FASTQ format has the potential to become a *de facto* standard for storing the letters and quality scores for a sequencing read in a single plain text file. The only fly in the ointment is that there are at least three versions of the FASTQ format which are incompatible and difficult to distinguish...

1. The original Sanger FASTQ format uses PHRED qualities encoded with an ASCII offset of 33. The NCBI are using this format in their Short Read Archive. We call this the `fastq` (or `fastq-sanger`) format in `Bio.SeqIO`.
2. Solexa (later bought by Illumina) introduced their own version using Solexa qualities encoded with an ASCII offset of 64. We call this the `fastq-solexa` format.
3. Illumina pipeline 1.3 onwards produces FASTQ files with PHRED qualities (which is more consistent), but encoded with an ASCII offset of 64. We call this the `fastq-illumina` format.

The Solexa quality scores are defined using a different log transformation:

$$Q_{\text{Solexa}} = -10 \times \log_{10}\left(\frac{P_e}{1-P_e}\right)$$

Given Solexa/Illumina have now moved to using PHRED scores in version 1.3 of their pipeline, the Solexa quality scores will gradually fall out of use. If you equate the error estimates (P_e) these two equations allow conversion between the two scoring systems - and Biopython includes functions to do this in the `Bio.SeqIO.QualityIO` module, which are called if you use `Bio.SeqIO` to convert an old Solexa/Illumina file into a standard Sanger FASTQ file:

```
from Bio import SeqIO
SeqIO.convert("solexa.fastq", "fastq-solexa", "standard.fastq", "fastq")
```

If you want to convert a new Illumina 1.3+ FASTQ file, all that gets changed is the ASCII offset because although encoded differently the scores are all PHRED qualities:

```
from Bio import SeqIO
SeqIO.convert("illumina.fastq", "fastq-illumina", "standard.fastq", "fastq")
```

Note that using `Bio.SeqIO.convert()` like this is *much* faster than combining `Bio.SeqIO.parse()` and `Bio.SeqIO.write()` because optimised code is used for converting between FASTQ variants (and also for FASTQ to FASTA conversion).

For good quality reads, PHRED and Solexa scores are approximately equal, which means since both the `fasta-solexa` and `fastq-illumina` formats use an ASCII offset of 64 the files are almost the same. This was a deliberate design choice by Illumina, meaning applications expecting the old `fasta-solexa` style files will probably be OK using the newer `fastq-illumina` files (on good data). Of course, both variants are very different from the original FASTQ standard as used by Sanger, the NCBI, and elsewhere (format name `fastq` or `fastq-sanger`).

For more details, see the built in help (also online):

```
In [5]: from Bio.SeqIO import QualityIO
        help(QualityIO)
```

```
Help on module Bio.SeqIO.QualityIO in Bio.SeqIO:

NAME
    Bio.SeqIO.QualityIO - Bio.SeqIO support for the FASTQ and QUAL file formats.

DESCRIPTION
    Note that you are expected to use this code via the Bio.SeqIO interface, as
    shown below.

    The FASTQ file format is used frequently at the Wellcome Trust Sanger Institute
    to bundle a FASTA sequence and its PHRED quality data (integers between 0 and
    90).  Rather than using a single FASTQ file, often paired FASTA and QUAL files
```

```
are used containing the sequence and the quality information separately.

The PHRED software reads DNA sequencing trace files, calls bases, and
assigns a non-negative quality value to each called base using a logged
transformation of the error probability, Q = -10 log10( Pe ), for example::

    Pe = 1.0,         Q =  0
    Pe = 0.1,         Q = 10
    Pe = 0.01,        Q = 20
    ...
    Pe = 0.00000001,  Q = 80
    Pe = 0.000000001, Q = 90

In typical raw sequence reads, the PHRED quality valuea will be from 0 to 40.
In the QUAL format these quality values are held as space separated text in
a FASTA like file format.  In the FASTQ format, each quality values is encoded
with a single ASCI character using chr(Q+33), meaning zero maps to the
character "!" and for example 80 maps to "q".  For the Sanger FASTQ standard
the allowed range of PHRED scores is 0 to 93 inclusive. The sequences and
quality are then stored in pairs in a FASTA like format.

Unfortunately there is no official document describing the FASTQ file format,
and worse, several related but different variants exist. For more details,
please read this open access publication::

    The Sanger FASTQ file format for sequences with quality scores, and the
    Solexa/Illumina FASTQ variants.
    P.J.A.Cock (Biopython), C.J.Fields (BioPerl), N.Goto (BioRuby),
    M.L.Heuer (BioJava) and P.M. Rice (EMBOSS).
    Nucleic Acids Research 2010 38(6):1767-1771
    http://dx.doi.org/10.1093/nar/gkp1137

The good news is that Roche 454 sequencers can output files in the QUAL format,
and sensibly they use PHREP style scores like Sanger.  Converting a pair of
FASTA and QUAL files into a Sanger style FASTQ file is easy. To extract QUAL
files from a Roche 454 SFF binary file, use the Roche off instrument command
line tool "sffinfo" with the -q or -qual argument.  You can extract a matching
FASTA file using the -s or -seq argument instead.

The bad news is that Solexa/Illumina did things differently - they have their
own scoring system AND their own incompatible versions of the FASTQ format.
Solexa/Illumina quality scores use Q = - 10 log10 ( Pe / (1-Pe) ), which can
be negative.  PHRED scores and Solexa scores are NOT interchangeable (but a
reasonable mapping can be achieved between them, and they are approximately
equal for higher quality reads).

Confusingly early Solexa pipelines produced a FASTQ like file but using their
own score mapping and an ASCII offset of 64. To make things worse, for the
Solexa/Illumina pipeline 1.3 onwards, they introduced a third variant of the
FASTQ file format, this time using PHRED scores (which is more consistent) but
with an ASCII offset of 64.

i.e. There are at least THREE different and INCOMPATIBLE variants of the FASTQ
file format: The original Sanger PHRED standard, and two from Solexa/Illumina.

The good news is that as of CASAVA version 1.8, Illumina sequencers will
produce FASTQ files using the standard Sanger encoding.

You are expected to use this module via the Bio.SeqIO functions, with the
```

```
following format names:

    - "qual" means simple quality files using PHRED scores (e.g. from Roche 454)
    - "fastq" means Sanger style FASTQ files using PHRED scores and an ASCII
      offset of 33 (e.g. from the NCBI Short Read Archive and Illumina 1.8+).
      These can potentially hold PHRED scores from 0 to 93.
    - "fastq-sanger" is an alias for "fastq".
    - "fastq-solexa" means old Solexa (and also very early Illumina) style FASTQ
      files, using Solexa scores with an ASCII offset 64. These can hold Solexa
      scores from -5 to 62.
    - "fastq-illumina" means newer Illumina 1.3 to 1.7 style FASTQ files, using
      PHRED scores but with an ASCII offset 64, allowing PHRED scores from 0
      to 62.

We could potentially add support for "qual-solexa" meaning QUAL files which
contain Solexa scores, but thus far there isn't any reason to use such files.

For example, consider the following short FASTQ file::

    @EAS54_6_R1_2_1_413_324
    CCCTTCTTGTCTTCAGCGTTTCTCC
    +
    ;;3;;;;;;;;;;;;7;;;;;;;88
    @EAS54_6_R1_2_1_540_792
    TTGGCAGGCCAAGGCCGATGGATCA
    +
    ;;;;;;;;;;;7;;;;;-;;;3;83
    @EAS54_6_R1_2_1_443_348
    GTTGCTTCTGGCGTGGGTGGGGGGG
    +
    ;;;;;;;;;;;9;7;;.7;393333

This contains three reads of length 25.  From the read length these were
probably originally from an early Solexa/Illumina sequencer but this file
follows the Sanger FASTQ convention (PHRED style qualities with an ASCII
offet of 33).  This means we can parse this file using Bio.SeqIO using
"fastq" as the format name:

>>> from Bio import SeqIO
>>> for record in SeqIO.parse("Quality/example.fastq", "fastq"):
...     print("%s %s" % (record.id, record.seq))
EAS54_6_R1_2_1_413_324 CCCTTCTTGTCTTCAGCGTTTCTCC
EAS54_6_R1_2_1_540_792 TTGGCAGGCCAAGGCCGATGGATCA
EAS54_6_R1_2_1_443_348 GTTGCTTCTGGCGTGGGTGGGGGGG

The qualities are held as a list of integers in each record's annotation:

>>> print(record)
ID: EAS54_6_R1_2_1_443_348
Name: EAS54_6_R1_2_1_443_348
Description: EAS54_6_R1_2_1_443_348
Number of features: 0
Per letter annotation for: phred_quality
Seq('GTTGCTTCTGGCGTGGGTGGGGGGG', SingleLetterAlphabet())
>>> print(record.letter_annotations["phred_quality"])
[26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 24, 26, 22, 26, 26, 13, 22, 26, 18, 24, 18, 18, 18,

You can use the SeqRecord format method to show this in the QUAL format:
```

```
>>> print(record.format("qual"))
>EAS54_6_R1_2_1_443_348
26 26 26 26 26 26 26 26 26 26 26 24 26 22 26 26 13 22 26 18
24 18 18 18 18
<BLANKLINE>

Or go back to the FASTQ format, use "fastq" (or "fastq-sanger"):

>>> print(record.format("fastq"))
@EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
+
;;;;;;;;;;;9;7;;.7;393333
<BLANKLINE>

Or, using the Illumina 1.3+ FASTQ encoding (PHRED values with an ASCII offset
of 64):

>>> print(record.format("fastq-illumina"))
@EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
+
ZZZZZZZZZZZXZVZZMVZRXRRRR
<BLANKLINE>

You can also get Biopython to convert the scores and show a Solexa style
FASTQ file:

>>> print(record.format("fastq-solexa"))
@EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
+
ZZZZZZZZZZZXZVZZMVZRXRRRR
<BLANKLINE>

Notice that this is actually the same output as above using "fastq-illumina"
as the format! The reason for this is all these scores are high enough that
the PHRED and Solexa scores are almost equal. The differences become apparent
for poor quality reads. See the functions solexa_quality_from_phred and
phred_quality_from_solexa for more details.

If you wanted to trim your sequences (perhaps to remove low quality regions,
or to remove a primer sequence), try slicing the SeqRecord objects.  e.g.

>>> sub_rec = record[5:15]
>>> print(sub_rec)
ID: EAS54_6_R1_2_1_443_348
Name: EAS54_6_R1_2_1_443_348
Description: EAS54_6_R1_2_1_443_348
Number of features: 0
Per letter annotation for: phred_quality
Seq('TTCTGGCGTG', SingleLetterAlphabet())
>>> print(sub_rec.letter_annotations["phred_quality"])
[26, 26, 26, 26, 26, 26, 24, 26, 22, 26]
>>> print(sub_rec.format("fastq"))
@EAS54_6_R1_2_1_443_348
TTCTGGCGTG
+
;;;;;;9;7;
```

```
<BLANKLINE>

If you wanted to, you could read in this FASTQ file, and save it as a QUAL file:

>>> from Bio import SeqIO
>>> record_iterator = SeqIO.parse("Quality/example.fastq", "fastq")
>>> with open("Quality/temp.qual", "w") as out_handle:
...     SeqIO.write(record_iterator, out_handle, "qual")
3

You can of course read in a QUAL file, such as the one we just created:

>>> from Bio import SeqIO
>>> for record in SeqIO.parse("Quality/temp.qual", "qual"):
...     print("%s %s" % (record.id, record.seq))
EAS54_6_R1_2_1_413_324 ?????????????????????????
EAS54_6_R1_2_1_540_792 ?????????????????????????
EAS54_6_R1_2_1_443_348 ?????????????????????????

Notice that QUAL files don't have a proper sequence present!  But the quality
information is there:

>>> print(record)
ID: EAS54_6_R1_2_1_443_348
Name: EAS54_6_R1_2_1_443_348
Description: EAS54_6_R1_2_1_443_348
Number of features: 0
Per letter annotation for: phred_quality
UnknownSeq(25, alphabet = SingleLetterAlphabet(), character = '?')
>>> print(record.letter_annotations["phred_quality"])
[26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 24, 26, 22, 26, 26, 13, 22, 26, 18, 24, 18, 18, 18,

Just to keep things tidy, if you are following this example yourself, you can
delete this temporary file now:

>>> import os
>>> os.remove("Quality/temp.qual")

Sometimes you won't have a FASTQ file, but rather just a pair of FASTA and QUAL
files.  Because the Bio.SeqIO system is designed for reading single files, you
would have to read the two in separately and then combine the data.  However,
since this is such a common thing to want to do, there is a helper iterator
defined in this module that does this for you - PairedFastaQualIterator.

Alternatively, if you have enough RAM to hold all the records in memory at once,
then a simple dictionary approach would work:

>>> from Bio import SeqIO
>>> reads = SeqIO.to_dict(SeqIO.parse("Quality/example.fasta", "fasta"))
>>> for rec in SeqIO.parse("Quality/example.qual", "qual"):
...     reads[rec.id].letter_annotations["phred_quality"]=rec.letter_annotations["phred_quality"]

You can then access any record by its key, and get both the sequence and the
quality scores.

>>> print(reads["EAS54_6_R1_2_1_540_792"].format("fastq"))
@EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
+
;;;;;;;;;;;7;;;;;-;;;3;83
<BLANKLINE>

It is important that you explicitly tell Bio.SeqIO which FASTQ variant you are
using ("fastq" or "fastq-sanger" for the Sanger standard using PHRED values,
"fastq-solexa" for the original Solexa/Illumina variant, or "fastq-illumina"
for the more recent variant), as this cannot be detected reliably
automatically.

To illustrate this problem, let's consider an artifical example:

>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import generic_dna
>>> from Bio.SeqRecord import SeqRecord
>>> test = SeqRecord(Seq("NACGTACGTA", generic_dna), id="Test",
... description="Made up!")
>>> print(test.format("fasta"))
>Test Made up!
NACGTACGTA
<BLANKLINE>
>>> print(test.format("fastq"))
Traceback (most recent call last):
 ...
ValueError: No suitable quality scores found in letter_annotations of SeqRecord (id=Test).

We created a sample SeqRecord, and can show it in FASTA format - but for QUAL
or FASTQ format we need to provide some quality scores. These are held as a
list of integers (one for each base) in the letter_annotations dictionary:

>>> test.letter_annotations["phred_quality"] = [0, 1, 2, 3, 4, 5, 10, 20, 30, 40]
>>> print(test.format("qual"))
>Test Made up!
0 1 2 3 4 5 10 20 30 40
<BLANKLINE>
>>> print(test.format("fastq"))
@Test Made up!
NACGTACGTA
+
!"#$%&+5?I
<BLANKLINE>

We can check this FASTQ encoding - the first PHRED quality was zero, and this
mapped to a exclamation mark, while the final score was 40 and this mapped to
the letter "I":

>>> ord('!') - 33
0
>>> ord('I') - 33
40
>>> [ord(letter)-33 for letter in '!"#$%&+5?I']
[0, 1, 2, 3, 4, 5, 10, 20, 30, 40]

Similarly, we could produce an Illumina 1.3 to 1.7 style FASTQ file using PHRED
scores with an offset of 64:

>>> print(test.format("fastq-illumina"))
@Test Made up!
NACGTACGTA
+
    @ABCDEJT^h
    <BLANKLINE>

    And we can check this too - the first PHRED score was zero, and this mapped to
    "@", while the final score was 40 and this mapped to "h":

    >>> ord("@") - 64
    0
    >>> ord("h") - 64
    40
    >>> [ord(letter)-64 for letter in "@ABCDEJT^h"]
    [0, 1, 2, 3, 4, 5, 10, 20, 30, 40]

    Notice how different the standard Sanger FASTQ and the Illumina 1.3 to 1.7 style
    FASTQ files look for the same data! Then we have the older Solexa/Illumina
    format to consider which encodes Solexa scores instead of PHRED scores.

    First let's see what Biopython says if we convert the PHRED scores into Solexa
    scores (rounding to one decimal place):

    >>> for q in [0, 1, 2, 3, 4, 5, 10, 20, 30, 40]:
    ...     print("PHRED %i maps to Solexa %0.1f" % (q, solexa_quality_from_phred(q)))
    PHRED 0 maps to Solexa -5.0
    PHRED 1 maps to Solexa -5.0
    PHRED 2 maps to Solexa -2.3
    PHRED 3 maps to Solexa -0.0
    PHRED 4 maps to Solexa 1.8
    PHRED 5 maps to Solexa 3.3
    PHRED 10 maps to Solexa 9.5
    PHRED 20 maps to Solexa 20.0
    PHRED 30 maps to Solexa 30.0
    PHRED 40 maps to Solexa 40.0

    Now here is the record using the old Solexa style FASTQ file:

    >>> print(test.format("fastq-solexa"))
    @Test Made up!
    NACGTACGTA
    +
    ;;>@BCJT^h
    <BLANKLINE>

    Again, this is using an ASCII offset of 64, so we can check the Solexa scores:

    >>> [ord(letter)-64 for letter in ";;>@BCJT^h"]
    [-5, -5, -2, 0, 2, 3, 10, 20, 30, 40]

    This explains why the last few letters of this FASTQ output matched that using
    the Illumina 1.3 to 1.7 format - high quality PHRED scores and Solexa scores
    are approximately equal.

CLASSES
    Bio.SeqIO.Interfaces.SequentialSequenceWriter(Bio.SeqIO.Interfaces.SequenceWriter)
        FastqIlluminaWriter
        FastqPhredWriter
        FastqSolexaWriter
        QualPhredWriter

    class FastqIlluminaWriter(Bio.SeqIO.Interfaces.SequentialSequenceWriter)
```

```
 |  Write Illumina 1.3+ FASTQ format files (with PHRED quality scores).
 |
 |  This outputs FASTQ files like those from the Solexa/Illumina 1.3+ pipeline,
 |  using PHRED scores and an ASCII offset of 64. Note these files are NOT
 |  compatible with the standard Sanger style PHRED FASTQ files which use an
 |  ASCII offset of 32.
 |
 |  Although you can use this class directly, you are strongly encouraged to
 |  use the Bio.SeqIO.write() function with format name "fastq-illumina"
 |  instead. This code is also called if you use the .format("fastq-illumina")
 |  method of a SeqRecord. For example,
 |
 |  >>> from Bio import SeqIO
 |  >>> record = SeqIO.read("Quality/sanger_faked.fastq", "fastq-sanger")
 |  >>> print(record.format("fastq-illumina"))
 |  @Test PHRED qualities from 40 to 0 inclusive
 |  ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTN
 |  +
 |  hgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@
 |  <BLANKLINE>
 |
 |  Note that Illumina FASTQ files have an upper limit of PHRED quality 62, which is
 |  encoded as ASCII 126, the tilde. If your quality scores are truncated to fit, a
 |  warning is issued.
 |
 |  Method resolution order:
 |      FastqIlluminaWriter
 |      Bio.SeqIO.Interfaces.SequentialSequenceWriter
 |      Bio.SeqIO.Interfaces.SequenceWriter
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  write_record(self, record)
 |      Write a single FASTQ record to the file.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequentialSequenceWriter:
 |
 |  __init__(self, handle)
 |      Creates the writer object.
 |
 |      Use the method write_file() to actually record your sequence records.
 |
 |  write_file(self, records)
 |      Use this to write an entire file containing the given records.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      This method can only be called once.  Returns the number of records
 |      written.
 |
 |  write_footer(self)
 |
 |  write_header(self)
 |
 |  write_records(self, records)
 |      Write multiple record to the output file.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      Once you have called write_header() you can call write_record()
 |      and/or write_records() as many times as needed.  Then call
 |      write_footer() and close().
 |
 |      Returns the number of records written.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
 |  clean(self, text)
 |      Use this to avoid getting newlines in the output.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class FastqPhredWriter(Bio.SeqIO.Interfaces.SequentialSequenceWriter)
 |  Class to write standard FASTQ format files (using PHRED quality scores).
 |
 |  Although you can use this class directly, you are strongly encouraged
 |  to use the Bio.SeqIO.write() function instead via the format name "fastq"
 |  or the alias "fastq-sanger".  For example, this code reads in a standard
 |  Sanger style FASTQ file (using PHRED scores) and re-saves it as another
 |  Sanger style FASTQ file:
 |
 |  >>> from Bio import SeqIO
 |  >>> record_iterator = SeqIO.parse("Quality/example.fastq", "fastq")
 |  >>> with open("Quality/temp.fastq", "w") as out_handle:
 |  ...     SeqIO.write(record_iterator, out_handle, "fastq")
 |  3
 |
 |  You might want to do this if the original file included extra line breaks,
 |  which while valid may not be supported by all tools.  The output file from
 |  Biopython will have each sequence on a single line, and each quality
 |  string on a single line (which is considered desirable for maximum
 |  compatibility).
 |
 |  In this next example, an old style Solexa/Illumina FASTQ file (using Solexa
 |  quality scores) is converted into a standard Sanger style FASTQ file using
 |  PHRED qualities:
 |
 |  >>> from Bio import SeqIO
 |  >>> record_iterator = SeqIO.parse("Quality/solexa_example.fastq", "fastq-solexa")
 |  >>> with open("Quality/temp.fastq", "w") as out_handle:
 |  ...     SeqIO.write(record_iterator, out_handle, "fastq")
 |  5
 |
 |  This code is also called if you use the .format("fastq") method of a
 |  SeqRecord, or .format("fastq-sanger") if you prefer that alias.
 |
 |  Note that Sanger FASTQ files have an upper limit of PHRED quality 93, which is
 |  encoded as ASCII 126, the tilde. If your quality scores are truncated to fit, a
 |  warning is issued.
 |
 |  P.S. To avoid cluttering up your working directory, you can delete this
 |  temporary file now:
 |
 |  >>> import os
 |  >>> os.remove("Quality/temp.fastq")
 |
 |  Method resolution order:
 |      FastqPhredWriter
 |      Bio.SeqIO.Interfaces.SequentialSequenceWriter
 |      Bio.SeqIO.Interfaces.SequenceWriter
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  write_record(self, record)
 |      Write a single FASTQ record to the file.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequentialSequenceWriter:
 |
 |  __init__(self, handle)
 |      Creates the writer object.
 |
 |      Use the method write_file() to actually record your sequence records.
 |
 |  write_file(self, records)
 |      Use this to write an entire file containing the given records.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      This method can only be called once.  Returns the number of records
 |      written.
 |
 |  write_footer(self)
 |
 |  write_header(self)
 |
 |  write_records(self, records)
 |      Write multiple record to the output file.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      Once you have called write_header() you can call write_record()
 |      and/or write_records() as many times as needed.  Then call
 |      write_footer() and close().
 |
 |      Returns the number of records written.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
 |  clean(self, text)
 |      Use this to avoid getting newlines in the output.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
  |  __dict__
  |      dictionary for instance variables (if defined)
  |
  |  __weakref__
  |      list of weak references to the object (if defined)

class FastqSolexaWriter(Bio.SeqIO.Interfaces.SequentialSequenceWriter)
 |  Write old style Solexa/Illumina FASTQ format files (with Solexa qualities).
 |
 |  This outputs FASTQ files like those from the early Solexa/Illumina
 |  pipeline, using Solexa scores and an ASCII offset of 64. These are
 |  NOT compatible with the standard Sanger style PHRED FASTQ files.
 |
 |  If your records contain a "solexa_quality" entry under letter_annotations,
 |  this is used, otherwise any "phred_quality" entry will be used after
 |  conversion using the solexa_quality_from_phred function. If neither style
 |  of quality scores are present, an exception is raised.
 |
 |  Although you can use this class directly, you are strongly encouraged
 |  to use the Bio.SeqIO.write() function instead.  For example, this code
 |  reads in a FASTQ file and re-saves it as another FASTQ file:
 |
 |  >>> from Bio import SeqIO
 |  >>> record_iterator = SeqIO.parse("Quality/solexa_example.fastq", "fastq-solexa")
 |  >>> with open("Quality/temp.fastq", "w") as out_handle:
 |  ...     SeqIO.write(record_iterator, out_handle, "fastq-solexa")
 |  5
 |
 |  You might want to do this if the original file included extra line breaks,
 |  which (while valid) may not be supported by all tools.  The output file
 |  from Biopython will have each sequence on a single line, and each quality
 |  string on a single line (which is considered desirable for maximum
 |  compatibility).
 |
 |  This code is also called if you use the .format("fastq-solexa") method of
 |  a SeqRecord. For example,
 |
 |  >>> record = SeqIO.read("Quality/sanger_faked.fastq", "fastq-sanger")
 |  >>> print(record.format("fastq-solexa"))
 |  @Test PHRED qualities from 40 to 0 inclusive
 |  ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTN
 |  +
 |  hgfedcba`_^]\[ZYXWVUTSRQPONMLKJHGFECB@>;;
 |  <BLANKLINE>
 |
 |  Note that Solexa FASTQ files have an upper limit of Solexa quality 62, which is
 |  encoded as ASCII 126, the tilde.  If your quality scores must be truncated to fit,
 |  a warning is issued.
 |
 |  P.S. Don't forget to delete the temp file if you don't need it anymore:
 |
 |  >>> import os
 |  >>> os.remove("Quality/temp.fastq")
 |
 |  Method resolution order:
 |      FastqSolexaWriter
 |      Bio.SeqIO.Interfaces.SequentialSequenceWriter
 |      Bio.SeqIO.Interfaces.SequenceWriter
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  write_record(self, record)
 |      Write a single FASTQ record to the file.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequentialSequenceWriter:
 |
 |  __init__(self, handle)
 |      Creates the writer object.
 |
 |      Use the method write_file() to actually record your sequence records.
 |
 |  write_file(self, records)
 |      Use this to write an entire file containing the given records.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      This method can only be called once.  Returns the number of records
 |      written.
 |
 |  write_footer(self)
 |
 |  write_header(self)
 |
 |  write_records(self, records)
 |      Write multiple record to the output file.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      Once you have called write_header() you can call write_record()
 |      and/or write_records() as many times as needed.  Then call
 |      write_footer() and close().
 |
 |      Returns the number of records written.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
 |  clean(self, text)
 |      Use this to avoid getting newlines in the output.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Bio.SeqIO.Interfaces.SequenceWriter:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class QualPhredWriter(Bio.SeqIO.Interfaces.SequentialSequenceWriter)
 |  Class to write QUAL format files (using PHRED quality scores).
 |
 |  Although you can use this class directly, you are strongly encouraged
 |  to use the Bio.SeqIO.write() function instead.  For example, this code
 |  reads in a FASTQ file and saves the quality scores into a QUAL file:
 |
 |  >>> from Bio import SeqIO
 |  >>> record_iterator = SeqIO.parse("Quality/example.fastq", "fastq")
 |  >>> with open("Quality/temp.qual", "w") as out_handle:
 |  ...     SeqIO.write(record_iterator, out_handle, "qual")
 |  3
 |
 |  This code is also called if you use the .format("qual") method of a
 |  SeqRecord.
 |
 |  P.S. Don't forget to clean up the temp file if you don't need it anymore:
 |
 |  >>> import os
 |  >>> os.remove("Quality/temp.qual")
 |
 |  Method resolution order:
 |      QualPhredWriter
 |      Bio.SeqIO.Interfaces.SequentialSequenceWriter
 |      Bio.SeqIO.Interfaces.SequenceWriter
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, handle, wrap=60, record2title=None)
 |      Create a QUAL writer.
 |
 |      Arguments:
 |       - handle - Handle to an output file, e.g. as returned
 |                  by open(filename, "w")
 |       - wrap   - Optional line length used to wrap sequence lines.
 |                  Defaults to wrapping the sequence at 60 characters
 |                  Use zero (or None) for no wrapping, giving a single
 |                  long line for the sequence.
 |       - record2title - Optional function to return the text to be
 |                  used for the title line of each record.  By default
 |                  a combination of the record.id and record.description
 |                  is used.  If the record.description starts with the
 |                  record.id, then just the record.description is used.
 |
 |      The record2title argument is present for consistency with the
 |      Bio.SeqIO.FastaIO writer class.
 |
 |  write_record(self, record)
 |      Write a single QUAL record to the file.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from Bio.SeqIO.Interfaces.SequentialSequenceWriter:
 |
 |  write_file(self, records)
 |      Use this to write an entire file containing the given records.
 |
 |      records - A list or iterator returning SeqRecord objects
 |
 |      This method can only be called once.  Returns the number of records
 |      written.
 |
 |  write_footer(self)
 |
 |  write_header(self)
 |
     |  write_records(self, records)
     |      Write multiple record to the output file.
     |      
     |      records - A list or iterator returning SeqRecord objects
     |      
     |      Once you have called write_header() you can call write_record()
     |      and/or write_records() as many times as needed.  Then call
     |      write_footer() and close().
     |      
     |      Returns the number of records written.
     |  
     |  ----------------------------------------------------------------------
     |  Methods inherited from Bio.SeqIO.Interfaces.SequenceWriter:
     |  
     |  clean(self, text)
     |      Use this to avoid getting newlines in the output.
     |  
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from Bio.SeqIO.Interfaces.SequenceWriter:
     |  
     |  __dict__
     |      dictionary for instance variables (if defined)
     |  
     |  __weakref__
     |      list of weak references to the object (if defined)

FUNCTIONS
    FastqGeneralIterator(handle)
        Iterate over Fastq records as string tuples (not as SeqRecord objects).
        
        This code does not try to interpret the quality string numerically.  It
        just returns tuples of the title, sequence and quality as strings.  For
        the sequence and quality, any whitespace (such as new lines) is removed.
        
        Our SeqRecord based FASTQ iterators call this function internally, and then
        turn the strings into a SeqRecord objects, mapping the quality string into
        a list of numerical scores.  If you want to do a custom quality mapping,
        then you might consider calling this function directly.
        
        For parsing FASTQ files, the title string from the "@" line at the start
        of each record can optionally be omitted on the "+" lines.  If it is
        repeated, it must be identical.
        
        The sequence string and the quality string can optionally be split over
        multiple lines, although several sources discourage this.  In comparison,
        for the FASTA file format line breaks between 60 and 80 characters are
        the norm.
        
        **WARNING** - Because the "@" character can appear in the quality string,
        this can cause problems as this is also the marker for the start of
        a new sequence.  In fact, the "+" sign can also appear as well.  Some
        sources recommended having no line breaks in the  quality to avoid this,
        but even that is not enough, consider this example::
        
            @071113_EAS56_0053:1:1:998:236
            TTTCTTGCCCCCATAGACTGAGACCTTCCCTAAATA
            +071113_EAS56_0053:1:1:998:236
            IIIIIIIIIIIIIIIIIIIIIIIIIIIIICII+III
            @071113_EAS56_0053:1:1:182:712
            ACCCAGCTAATTTTTGTATTTTTGTTAGAGACAGTG
            +
            @IIIIIIIIIIIIIICDIIIII<%<6&-*).(*%+
            @071113_EAS56_0053:1:1:153:10
            TGTTCTGAAGGAAGGTGTGCGTGCGTGTGTGTGTGT
            +
            IIIIIIIIIIIICIIGIIIII>IAIIIE65I=II:6
            @071113_EAS56_0053:1:3:990:501
            TGGGAGGTTTTATGTGGA
            AAGCAGCAATGTACAAGA
            +
            IIIIIII.IIIIII1@44
            @-7.%<&+/$/%4(++(%
```

```
This is four PHRED encoded FASTQ entries originally from an NCBI source
(given the read length of 36, these are probably Solexa Illumina reads where
the quality has been mapped onto the PHRED values).

This example has been edited to illustrate some of the nasty things allowed
in the FASTQ format.  Firstly, on the "+" lines most but not all of the
(redundant) identifiers are omitted.  In real files it is likely that all or
none of these extra identifiers will be present.

Secondly, while the first three sequences have been shown without line
breaks, the last has been split over multiple lines.  In real files any line
breaks are likely to be consistent.

Thirdly, some of the quality string lines start with an "@" character.  For
the second record this is unavoidable.  However for the fourth sequence this
only happens because its quality string is split over two lines.  A naive
parser could wrongly treat any line starting with an "@" as the beginning of
a new sequence!  This code copes with this possible ambiguity by keeping
track of the length of the sequence which gives the expected length of the
quality string.

Using this tricky example file as input, this short bit of code demonstrates
what this parsing function would return:

>>> with open("Quality/tricky.fastq", "rU") as handle:
...     for (title, sequence, quality) in FastqGeneralIterator(handle):
...         print(title)
...         print("%s %s" % (sequence, quality))
...
071113_EAS56_0053:1:1:998:236
TTTCTTGCCCCCATAGACTGAGACCTTCCCTAAATA IIIIIIIIIIIIIIIIIIIIIIIIIIIIICII+III
071113_EAS56_0053:1:1:182:712
ACCCAGCTAATTTTTGTATTTTTGTTAGAGACAGTG @IIIIIIIIIIIIIICDIIIII<%<6&-*).(*%+
071113_EAS56_0053:1:1:153:10
TGTTCTGAAGGAAGGTGTGCGTGCGTGTGTGTGTGT IIIIIIIIIIIICIIGIIIII>IAIIIE65I=II:6
071113_EAS56_0053:1:3:990:501
TGGGAGGTTTTATGTGGAAAGCAGCAATGTACAAGA IIIIIII.IIIIII1@44@-7.%<&+/$/%4(++(%

Finally we note that some sources state that the quality string should
start with "!" (which using the PHRED mapping means the first letter always
has a quality score of zero).  This rather restrictive rule is not widely
observed, so is therefore ignored here.  One plus point about this "!" rule
is that (provided there are no line breaks in the quality sequence) it
would prevent the above problem with the "@" character.
```

```
FastqIlluminaIterator(handle, alphabet=SingleLetterAlphabet(), title2ids=None)
    Parse Illumina 1.3 to 1.7 FASTQ like files (which differ in the quality mapping).

    The optional arguments are the same as those for the FastqPhredIterator.

    For each sequence in Illumina 1.3+ FASTQ files there is a matching string
    encoding PHRED integer qualities using ASCII values with an offset of 64.

    >>> from Bio import SeqIO
    >>> record = SeqIO.read("Quality/illumina_faked.fastq", "fastq-illumina")
    >>> print("%s %s" % (record.id, record.seq))
    Test ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTN
    >>> max(record.letter_annotations["phred_quality"])
    40
    >>> min(record.letter_annotations["phred_quality"])
    0

    NOTE - Older versions of the Solexa/Illumina pipeline encoded Solexa scores
    with an ASCII offset of 64. They are approximately equal but only for high
    quality reads. If you have an old Solexa/Illumina file with negative
    Solexa scores, and try and read this as an Illumina 1.3+ file it will fail:

    >>> record2 = SeqIO.read("Quality/solexa_faked.fastq", "fastq-illumina")
    Traceback (most recent call last):
       ...
    ValueError: Invalid character in quality string

    NOTE - True Sanger style FASTQ files use PHRED scores with an offset of 33.

FastqPhredIterator(handle, alphabet=SingleLetterAlphabet(), title2ids=None)
    Generator function to iterate over FASTQ records (as SeqRecord objects).

        - handle - input file
        - alphabet - optional alphabet
        - title2ids - A function that, when given the title line from the FASTQ
          file (without the beginning >), will return the id, name and
          description (in that order) for the record as a tuple of
          strings.  If this is not given, then the entire title line
          will be used as the description, and the first word as the
          id and name.

    Note that use of title2ids matches that of Bio.SeqIO.FastaIO.

    For each sequence in a (Sanger style) FASTQ file there is a matching string
    encoding the PHRED qualities (integers between 0 and about 90) using ASCII
    values with an offset of 33.

    For example, consider a file containing three short reads::

        @EAS54_6_R1_2_1_413_324
        CCCTTCTTGTCTTCAGCGTTTCTCC
        +
        ;;3;;;;;;;;;;;;7;;;;;;;88
        @EAS54_6_R1_2_1_540_792
        TTGGCAGGCCAAGGCCGATGGATCA
        +
        ;;;;;;;;;;;7;;;;;-;;;3;83
        @EAS54_6_R1_2_1_443_348
        GTTGCTTCTGGCGTGGGTGGGGGGG
        +
        ;;;;;;;;;;;9;7;;.7;393333

    For each sequence (e.g. "CCCTTCTTGTCTTCAGCGTTTCTCC") there is a matching
    string encoding the PHRED qualities using a ASCII values with an offset of
    33 (e.g. ";;3;;;;;;;;;;;;7;;;;;;;88").

    Using this module directly you might run:

    >>> with open("Quality/example.fastq", "rU") as handle:
    ...     for record in FastqPhredIterator(handle):
    ...         print("%s %s" % (record.id, record.seq))
    EAS54_6_R1_2_1_413_324 CCCTTCTTGTCTTCAGCGTTTCTCC
    EAS54_6_R1_2_1_540_792 TTGGCAGGCCAAGGCCGATGGATCA
    EAS54_6_R1_2_1_443_348 GTTGCTTCTGGCGTGGGTGGGGGGG

    Typically however, you would call this via Bio.SeqIO instead with "fastq"
    (or "fastq-sanger") as the format:

    >>> from Bio import SeqIO
    >>> with open("Quality/example.fastq", "rU") as handle:
    ...     for record in SeqIO.parse(handle, "fastq"):
    ...         print("%s %s" % (record.id, record.seq))
    EAS54_6_R1_2_1_413_324 CCCTTCTTGTCTTCAGCGTTTCTCC
    EAS54_6_R1_2_1_540_792 TTGGCAGGCCAAGGCCGATGGATCA
    EAS54_6_R1_2_1_443_348 GTTGCTTCTGGCGTGGGTGGGGGGG

    If you want to look at the qualities, they are record in each record's
    per-letter-annotation dictionary as a simple list of integers:

    >>> print(record.letter_annotations["phred_quality"])
    [26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 24, 26, 22, 26, 26, 13, 22, 26, 18, 24, 18, 18,

FastqSolexaIterator(handle, alphabet=SingleLetterAlphabet(), title2ids=None)
    Parsing old Solexa/Illumina FASTQ like files (which differ in the quality mapping).

    The optional arguments are the same as those for the FastqPhredIterator.

    For each sequence in Solexa/Illumina FASTQ files there is a matching string
    encoding the Solexa integer qualities using ASCII values with an offset
    of 64.  Solexa scores are scaled differently to PHRED scores, and Biopython
    will NOT perform any automatic conversion when loading.

    NOTE - This file format is used by the OLD versions of the Solexa/Illumina
    pipeline. See also the FastqIlluminaIterator function for the NEW version.

    For example, consider a file containing these five records::

        @SLXA-B3_649_FC8437_R1_1_1_610_79
        GATGTGCAATACCTTTGTAGAGGAA
        +SLXA-B3_649_FC8437_R1_1_1_610_79
        YYYYYYYYYYYYYYYYYYWYWYYSU
        @SLXA-B3_649_FC8437_R1_1_1_397_389
        GGTTTGAGAAAGAGAAATGAGATAA
        +SLXA-B3_649_FC8437_R1_1_1_397_389
        YYYYYYYYYWYYYYWWYYYWYWYWW
        @SLXA-B3_649_FC8437_R1_1_1_850_123
        GAGGGTGTTGATCATGATGATGGCG
        +SLXA-B3_649_FC8437_R1_1_1_850_123
        YYYYYYYYYYYYYWYYWYYSYYYSY
        @SLXA-B3_649_FC8437_R1_1_1_362_549
        GGAAACAAAGTTTTTCTCAACATAG
        +SLXA-B3_649_FC8437_R1_1_1_362_549
        YYYYYYYYYYYYYYYYYYWWWWYWY
        @SLXA-B3_649_FC8437_R1_1_1_183_714
        GTATTATTTAATGGCATACACTCAA
        +SLXA-B3_649_FC8437_R1_1_1_183_714
        YYYYYYYYYYWYYYYWYWWUWWWQQ

    Using this module directly you might run:

    >>> with open("Quality/solexa_example.fastq", "rU") as handle:
    ...     for record in FastqSolexaIterator(handle):
    ...         print("%s %s" % (record.id, record.seq))
    SLXA-B3_649_FC8437_R1_1_1_610_79 GATGTGCAATACCTTTGTAGAGGAA
    SLXA-B3_649_FC8437_R1_1_1_397_389 GGTTTGAGAAAGAGAAATGAGATAA
    SLXA-B3_649_FC8437_R1_1_1_850_123 GAGGGTGTTGATCATGATGATGGCG
    SLXA-B3_649_FC8437_R1_1_1_362_549 GGAAACAAAGTTTTTCTCAACATAG
    SLXA-B3_649_FC8437_R1_1_1_183_714 GTATTATTTAATGGCATACACTCAA

    Typically however, you would call this via Bio.SeqIO instead with
    "fastq-solexa" as the format:

    >>> from Bio import SeqIO
    >>> with open("Quality/solexa_example.fastq", "rU") as handle:
    ...     for record in SeqIO.parse(handle, "fastq-solexa"):
    ...         print("%s %s" % (record.id, record.seq))
    SLXA-B3_649_FC8437_R1_1_1_610_79 GATGTGCAATACCTTTGTAGAGGAA
    SLXA-B3_649_FC8437_R1_1_1_397_389 GGTTTGAGAAAGAGAAATGAGATAA
    SLXA-B3_649_FC8437_R1_1_1_850_123 GAGGGTGTTGATCATGATGATGGCG
    SLXA-B3_649_FC8437_R1_1_1_362_549 GGAAACAAAGTTTTTCTCAACATAG
    SLXA-B3_649_FC8437_R1_1_1_183_714 GTATTATTTAATGGCATACACTCAA

    If you want to look at the qualities, they are recorded in each record's
    per-letter-annotation dictionary as a simple list of integers:

    >>> print(record.letter_annotations["solexa_quality"])
    [25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 23, 25, 25, 25, 25, 23, 25, 23, 23, 21, 23, 23, 23,

    These scores aren't very good, but they are high enough that they map
    almost exactly onto PHRED scores:

    >>> print("%0.2f" % phred_quality_from_solexa(25))
    25.01

    Let's look at faked example read which is even worse, where there are
    more noticeable differences between the Solexa and PHRED scores::

         @slxa_0001_1_0001_01
         ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTNNNNNN
         +slxa_0001_1_0001_01
         hgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;

    Again, you would typically use Bio.SeqIO to read this file in (rather than
    calling the Bio.SeqIO.QualtityIO module directly).  Most FASTQ files will
    contain thousands of reads, so you would normally use Bio.SeqIO.parse()
    as shown above.  This example has only as one entry, so instead we can
    use the Bio.SeqIO.read() function:
```

```
>>> from Bio import SeqIO
>>> with open("Quality/solexa_faked.fastq", "rU") as handle:
...     record = SeqIO.read(handle, "fastq-solexa")
>>> print("%s %s" % (record.id, record.seq))
slxa_0001_1_0001_01 ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTNNNNNN
>>> print(record.letter_annotations["solexa_quality"])
[40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18,

These quality scores are so low that when converted from the Solexa scheme
into PHRED scores they look quite different:

>>> print("%0.2f" % phred_quality_from_solexa(-1))
2.54
>>> print("%0.2f" % phred_quality_from_solexa(-5))
1.19

Note you can use the Bio.SeqIO.write() function or the SeqRecord's format
method to output the record(s):

>>> print(record.format("fastq-solexa"))
@slxa_0001_1_0001_01
ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTNNNNNN
+
hgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;
<BLANKLINE>

Note this output is slightly different from the input file as Biopython
has left out the optional repetition of the sequence identifier on the "+"
line.  If you want the to use PHRED scores, use "fastq" or "qual" as the
output format instead, and Biopython will do the conversion for you:

>>> print(record.format("fastq"))
@slxa_0001_1_0001_01
ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTNNNNNN
+
IHGFEDCBA@?>=<;:9876543210/.-,++*)('&&%%$$##""
<BLANKLINE>

>>> print(record.format("qual"))
>slxa_0001_1_0001_01
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
20 19 18 17 16 15 14 13 12 11 10 10 9 8 7 6 5 5 4 4 3 3 2 2
1 1
<BLANKLINE>

As shown above, the poor quality Solexa reads have been mapped to the
equivalent PHRED score (e.g. -5 to 1 as shown earlier).
```

```
PairedFastaQualIterator(fasta_handle, qual_handle, alphabet=SingleLetterAlphabet(), title2ids=Nor
    Iterate over matched FASTA and QUAL files as SeqRecord objects.

    For example, consider this short QUAL file with PHRED quality scores::

        >EAS54_6_R1_2_1_413_324
        26 26 18 26 26 26 26 26 26 26 26 26 26 26 26 22 26 26 26 26
        26 26 26 23 23
        >EAS54_6_R1_2_1_540_792
        26 26 26 26 26 26 26 26 26 26 26 22 26 26 26 26 26 12 26 26
```

```
            26 18 26 23 18
            >EAS54_6_R1_2_1_443_348
            26 26 26 26 26 26 26 26 26 26 26 24 26 22 26 26 13 22 26 18
            24 18 18 18 18

        And a matching FASTA file::

            >EAS54_6_R1_2_1_413_324
            CCCTTCTTGTCTTCAGCGTTTCTCC
            >EAS54_6_R1_2_1_540_792
            TTGGCAGGCCAAGGCCGATGGATCA
            >EAS54_6_R1_2_1_443_348
            GTTGCTTCTGGCGTGGGTGGGGGGG

        You can parse these separately using Bio.SeqIO with the "qual" and
        "fasta" formats, but then you'll get a group of SeqRecord objects with
        no sequence, and a matching group with the sequence but not the
        qualities.  Because it only deals with one input file handle, Bio.SeqIO
        can't be used to read the two files together - but this function can!
        For example,

        >>> with open("Quality/example.fasta", "rU") as f:
        ...     with open("Quality/example.qual", "rU") as q:
        ...         for record in PairedFastaQualIterator(f, q):
        ...             print("%s %s" % (record.id, record.seq))
        ...
        EAS54_6_R1_2_1_413_324 CCCTTCTTGTCTTCAGCGTTTCTCC
        EAS54_6_R1_2_1_540_792 TTGGCAGGCCAAGGCCGATGGATCA
        EAS54_6_R1_2_1_443_348 GTTGCTTCTGGCGTGGGTGGGGGGG

        As with the FASTQ or QUAL parsers, if you want to look at the qualities,
        they are in each record's per-letter-annotation dictionary as a simple
        list of integers:

        >>> print(record.letter_annotations["phred_quality"])
        [26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 24, 26, 22, 26, 26, 13, 22, 26, 18, 24, 18, 18,

        If you have access to data as a FASTQ format file, using that directly
        would be simpler and more straight forward.  Note that you can easily use
        this function to convert paired FASTA and QUAL files into FASTQ files:

        >>> from Bio import SeqIO
        >>> with open("Quality/example.fasta", "rU") as f:
        ...     with open("Quality/example.qual", "rU") as q:
        ...         SeqIO.write(PairedFastaQualIterator(f, q), "Quality/temp.fastq", "fastq")
        ...
        3

        And don't forget to clean up the temp file if you don't need it anymore:

        >>> import os
        >>> os.remove("Quality/temp.fastq")

    QualPhredIterator(handle, alphabet=SingleLetterAlphabet(), title2ids=None)
        For QUAL files which include PHRED quality scores, but no sequence.

        For example, consider this short QUAL file::

            >EAS54_6_R1_2_1_413_324
            26 26 18 26 26 26 26 26 26 26 26 26 26 26 26 22 26 26 26 26
            26 26 26 23 23
            >EAS54_6_R1_2_1_540_792
            26 26 26 26 26 26 26 26 26 26 26 22 26 26 26 26 26 12 26 26
            26 18 26 23 18
            >EAS54_6_R1_2_1_443_348
            26 26 26 26 26 26 26 26 26 26 26 24 26 22 26 26 13 22 26 18
            24 18 18 18 18

        Using this module directly you might run:

        >>> with open("Quality/example.qual", "rU") as handle:
        ...     for record in QualPhredIterator(handle):
        ...         print("%s %s" % (record.id, record.seq))
        EAS54_6_R1_2_1_413_324 ?????????????????????????
        EAS54_6_R1_2_1_540_792 ?????????????????????????
        EAS54_6_R1_2_1_443_348 ?????????????????????????

        Typically however, you would call this via Bio.SeqIO instead with "qual"
        as the format:

        >>> from Bio import SeqIO
        >>> with open("Quality/example.qual", "rU") as handle:
        ...     for record in SeqIO.parse(handle, "qual"):
        ...         print("%s %s" % (record.id, record.seq))
        EAS54_6_R1_2_1_413_324 ?????????????????????????
        EAS54_6_R1_2_1_540_792 ?????????????????????????
        EAS54_6_R1_2_1_443_348 ?????????????????????????

        Becase QUAL files don't contain the sequence string itself, the seq
        property is set to an UnknownSeq object.  As no alphabet was given, this
        has defaulted to a generic single letter alphabet and the character "?"
        used.

        By specifying a nucleotide alphabet, "N" is used instead:

        >>> from Bio import SeqIO
        >>> from Bio.Alphabet import generic_dna
        >>> with open("Quality/example.qual", "rU") as handle:
        ...     for record in SeqIO.parse(handle, "qual", alphabet=generic_dna):
        ...         print("%s %s" % (record.id, record.seq))
        EAS54_6_R1_2_1_413_324 NNNNNNNNNNNNNNNNNNNNNNNNN
        EAS54_6_R1_2_1_540_792 NNNNNNNNNNNNNNNNNNNNNNNNN
        EAS54_6_R1_2_1_443_348 NNNNNNNNNNNNNNNNNNNNNNNNN

        However, the quality scores themselves are available as a list of integers
        in each record's per-letter-annotation:

        >>> print(record.letter_annotations["phred_quality"])
        [26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 24, 26, 22, 26, 26, 13, 22, 26, 18, 24, 18, 18,

        You can still slice one of these SeqRecord objects with an UnknownSeq:

        >>> sub_record = record[5:10]
        >>> print("%s %s" % (sub_record.id, sub_record.letter_annotations["phred_quality"]))
        EAS54_6_R1_2_1_443_348 [26, 26, 26, 26, 26]

        As of Biopython 1.59, this parser will accept files with negatives quality
        scores but will replace them with the lowest possible PHRED score of zero.
```

```
        This will trigger a warning, previously it raised a ValueError exception.

    log(...)
        log(x[, base])

        Return the logarithm of x to the given base.
        If the base not specified, returns the natural logarithm (base e) of x.

    phred_quality_from_solexa(solexa_quality)
        Convert a Solexa quality (which can be negative) to a PHRED quality.

        PHRED and Solexa quality scores are both log transformations of a
        probality of error (high score = low probability of error). This function
        takes a Solexa score, transforms it back to a probability of error, and
        then re-expresses it as a PHRED score. This assumes the error estimates
        are equivalent.

        The underlying formulas are given in the documentation for the sister
        function solexa_quality_from_phred, in this case the operation is::

            phred_quality = 10*log(10**(solexa_quality/10.0) + 1, 10)

        This will return a floating point number, it is up to you to round this to
        the nearest integer if appropriate.  e.g.

        >>> print("%0.2f" % round(phred_quality_from_solexa(80), 2))
        80.00
        >>> print("%0.2f" % round(phred_quality_from_solexa(20), 2))
        20.04
        >>> print("%0.2f" % round(phred_quality_from_solexa(10), 2))
        10.41
        >>> print("%0.2f" % round(phred_quality_from_solexa(0), 2))
        3.01
        >>> print("%0.2f" % round(phred_quality_from_solexa(-5), 2))
        1.19

        Note that a solexa_quality less then -5 is not expected, will trigger a
        warning, but will still be converted as per the logarithmic mapping
        (giving a number between 0 and 1.19 back).

        As a special case where None is used for a "missing value", None is
        returned:

        >>> print(phred_quality_from_solexa(None))
        None

    solexa_quality_from_phred(phred_quality)
        Covert a PHRED quality (range 0 to about 90) to a Solexa quality.

        PHRED and Solexa quality scores are both log transformations of a
        probality of error (high score = low probability of error). This function
        takes a PHRED score, transforms it back to a probability of error, and
        then re-expresses it as a Solexa score. This assumes the error estimates
        are equivalent.

        How does this work exactly? Well the PHRED quality is minus ten times the
        base ten logarithm of the probability of error::

            phred_quality = -10*log(error,10)
```

```
Therefore, turning this round::

    error = 10 ** (- phred_quality / 10)

Now, Solexa qualities use a different log transformation::

    solexa_quality = -10*log(error/(1-error),10)

After substitution and a little manipulation we get::

     solexa_quality = 10*log(10**(phred_quality/10.0) - 1, 10)

However, real Solexa files use a minimum quality of -5. This does have a
good reason - a random base call would be correct 25% of the time,
and thus have a probability of error of 0.75, which gives 1.25 as the PHRED
quality, or -4.77 as the Solexa quality. Thus (after rounding), a random
nucleotide read would have a PHRED quality of 1, or a Solexa quality of -5.

Taken literally, this logarithic formula would map a PHRED quality of zero
to a Solexa quality of minus infinity. Of course, taken literally, a PHRED
score of zero means a probability of error of one (i.e. the base call is
definitely wrong), which is worse than random! In practice, a PHRED quality
of zero usually means a default value, or perhaps random - and therefore
mapping it to the minimum Solexa score of -5 is reasonable.

In conclusion, we follow EMBOSS, and take this logarithmic formula but also
apply a minimum value of -5.0 for the Solexa quality, and also map a PHRED
quality of zero to -5.0 as well.

Note this function will return a floating point number, it is up to you to
round this to the nearest integer if appropriate.  e.g.

>>> print("%0.2f" % round(solexa_quality_from_phred(80), 2))
80.00
>>> print("%0.2f" % round(solexa_quality_from_phred(50), 2))
50.00
>>> print("%0.2f" % round(solexa_quality_from_phred(20), 2))
19.96
>>> print("%0.2f" % round(solexa_quality_from_phred(10), 2))
9.54
>>> print("%0.2f" % round(solexa_quality_from_phred(5), 2))
3.35
>>> print("%0.2f" % round(solexa_quality_from_phred(4), 2))
1.80
>>> print("%0.2f" % round(solexa_quality_from_phred(3), 2))
-0.02
>>> print("%0.2f" % round(solexa_quality_from_phred(2), 2))
-2.33
>>> print("%0.2f" % round(solexa_quality_from_phred(1), 2))
-5.00
>>> print("%0.2f" % round(solexa_quality_from_phred(0), 2))
-5.00

Notice that for high quality reads PHRED and Solexa scores are numerically
equal. The differences are important for poor quality reads, where PHRED
has a minimum of zero but Solexa scores can be negative.

Finally, as a special case where None is used for a "missing value", None
```

```
        is returned:

        >>> print(solexa_quality_from_phred(None))
        None

DATA
    SANGER_SCORE_OFFSET = 33
    SOLEXA_SCORE_OFFSET = 64
    __docformat__ = 'restructuredtext en'
    print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0)...
    single_letter_alphabet = SingleLetterAlphabet()

FILE
    /home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/QualityIO.py
```

20.1.10 Converting FASTA and QUAL files into FASTQ files

FASTQ files hold *both* sequences and their quality strings. FASTA files hold *just* sequences, while QUAL files hold *just* the qualities. Therefore a single FASTQ file can be converted to or from *paired* FASTA and QUAL files.

Going from FASTQ to FASTA is easy:

```python
from Bio import SeqIO
SeqIO.convert("example.fastq", "fastq", "example.fasta", "fasta")
```

Going from FASTQ to QUAL is also easy:

```python
from Bio import SeqIO
SeqIO.convert("example.fastq", "fastq", "example.qual", "qual")
```

However, the reverse is a little more tricky. You can use `Bio.SeqIO.parse()` to iterate over the records in a *single* file, but in this case we have two input files. There are several strategies possible, but assuming that the two files are really paired the most memory efficient way is to loop over both together. The code is a little fiddly, so we provide a function called `PairedFastaQualIterator` in the `Bio.SeqIO.QualityIO` module to do this. This takes two handles (the FASTA file and the QUAL file) and returns a `SeqRecord` iterator:

```python
from Bio.SeqIO.QualityIO import PairedFastaQualIterator
for record in PairedFastaQualIterator(open("example.fasta"), open("example.qual")):
   print(record)
```

This function will check that the FASTA and QUAL files are consistent (e.g. the records are in the same order, and have the same sequence length). You can combine this with the `Bio.SeqIO.write()` function to convert a pair of FASTA and QUAL files into a single FASTQ files:

```python
from Bio import SeqIO
from Bio.SeqIO.QualityIO import PairedFastaQualIterator
handle = open("temp.fastq", "w") #w=write
records = PairedFastaQualIterator(open("example.fasta"), open("example.qual"))
count = SeqIO.write(records, handle, "fastq")
handle.close()
print("Converted %i records" % count)
```

20.1.11 Indexing a FASTQ file

FASTQ files are often very large, with millions of reads in them. Due to the sheer amount of data, you can't load all the records into memory at once. This is why the examples above (filtering and trimming) iterate over the file looking at just one `SeqRecord` at a time.

However, sometimes you can't use a big loop or an iterator - you may need random access to the reads. Here the `Bio.SeqIO.index()` function may prove very helpful, as it allows you to access any read in the FASTQ file by its name (see Section [sec:SeqIO-index]).

Again we'll use the `SRR020192.fastq` file from the ENA (ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz), although this is actually quite a small FASTQ file with less than $50,000$ reads:

```
In [9]: !wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz
        !gzip -d SRR020192.fastq.gz
```

```
In [10]: from Bio import SeqIO
         fq_dict = SeqIO.index("SRR020192.fastq", "fastq")
         len(fq_dict)
```

```
Out[10]: 41892
```

```
In [12]: fq_dict["SRR020192.23186"].seq
```

```
Out[12]: Seq('GTCCCAGTATTCGGATTTGTCTGCCAAAACAATGAAATTGACACAGTTTACAAC...CCG', SingleLetterAlphabet())
```

When testing this on a FASTQ file with seven million reads, indexing took about a minute, but record access was almost instant.

The example in Section [sec:SeqIO-sort] show how you can use the `Bio.SeqIO.index()` function to sort a large FASTA file – this could also be used on FASTQ files.

20.1.12 Converting SFF files

If you work with 454 (Roche) sequence data, you will probably have access to the raw data as a Standard Flowgram Format (SFF) file. This contains the sequence reads (called bases) with quality scores and the original flow information.

A common task is to convert from SFF to a pair of FASTA and QUAL files, or to a single FASTQ file. These operations are trivial using the `Bio.SeqIO.convert()` function (see Section [sec:SeqIO-conversion]):

```
In [14]: from Bio import SeqIO
         SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff", "reads.fasta", "fasta")
```

```
Out[14]: 10
```

```
In [15]: SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff", "reads.qual", "qual")
```

```
Out[15]: 10
```

```
In [16]: SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff", "reads.fastq", "fastq")
```

```
Out[16]: 10
```

Remember the convert function returns the number of records, in this example just ten. This will give you the *untrimmed* reads, where the leading and trailing poor quality sequence or adaptor will be in lower case. If you want the *trimmed* reads (using the clipping information recorded within the SFF file) use this:

```
In [17]: from Bio import SeqIO
         SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff-trim", "trimmed.fasta", "fasta")
```

```
Out[17]: 10
```

```
In [18]: SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff-trim", "trimmed.qual", "qual")
```

```
Out[18]: 10
```

```
In [19]: SeqIO.convert("data/E3MFGYR02_random_10_reads.sff", "sff-trim", "trimmed.fastq", "fastq")
```

```
Out[19]: 10
```

If you run Linux, you could ask Roche for a copy of their "off instrument" tools (often referred to as the Newbler tools). This offers an alternative way to do SFF to FASTA or QUAL conversion at the command line (but currently FASTQ output is not supported), e.g.

```
$ sffinfo -seq -notrim E3MFGYR02_random_10_reads.sff > reads.fasta
$ sffinfo -qual -notrim E3MFGYR02_random_10_reads.sff > reads.qual
$ sffinfo -seq -trim E3MFGYR02_random_10_reads.sff > trimmed.fasta
$ sffinfo -qual -trim E3MFGYR02_random_10_reads.sff > trimmed.qual
```

The way Biopython uses mixed case sequence strings to represent the trimming points deliberately mimics what the Roche tools do.

For more information on the Biopython SFF support, consult the built in help:

```
In [20]: from Bio.SeqIO import SffIO
         help(SffIO)
```

```
Help on module Bio.SeqIO.SffIO in Bio.SeqIO:

NAME
    Bio.SeqIO.SffIO - Bio.SeqIO support for the binary Standard Flowgram Format (SFF) file format.

DESCRIPTION
    SFF was designed by 454 Life Sciences (Roche), the Whitehead Institute for
    Biomedical Research and the Wellcome Trust Sanger Institute. SFF was also used
    as the native output format from early versions of Ion Torrent's PGM platform
    as well. You are expected to use this module via the Bio.SeqIO functions under
    the format name "sff" (or "sff-trim" as described below).

    For example, to iterate over the records in an SFF file,

        >>> from Bio import SeqIO
        >>> for record in SeqIO.parse("Roche/E3MFGYR02_random_10_reads.sff", "sff"):
        ...     print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
        ...
        E3MFGYR02JWQ7T 265 tcagGGTCTACATGTTGGTT...
        E3MFGYR02JA6IL 271 tcagTTTTTTTTGGAAAGGA...
        E3MFGYR02JHD4H 310 tcagAAAGACAAGTGGTATC...
        E3MFGYR02GFKUC 299 tcagCGGCCGGGCCTCTCAT...
        E3MFGYR02FTGED 281 tcagTGGTAATGGGGGGAAA...
        E3MFGYR02FR9G7 261 tcagCTCCGTAAGAAGGTGC...
        E3MFGYR02GAZMS 278 tcagAAAGAAGTAAGGTAAA...
        E3MFGYR02HHZ8O 221 tcagACTTTCTTCTTTACCG...
        E3MFGYR02GPGB1 269 tcagAAGCAGTGGTATCAAC...
        E3MFGYR02F7Z7G 219 tcagAATCATCCACTTTTTA...

    Each SeqRecord object will contain all the annotation from the SFF file,
    including the PHRED quality scores.

        >>> print("%s %i" % (record.id, len(record)))
        E3MFGYR02F7Z7G 219
        >>> print("%s..." % record.seq[:10])
        tcagAATCAT...
        >>> print("%r..." % (record.letter_annotations["phred_quality"][:10]))
        [22, 21, 23, 28, 26, 15, 12, 21, 28, 21]...

    Notice that the sequence is given in mixed case, the central upper case region
```

```
corresponds to the trimmed sequence. This matches the output of the Roche
tools (and the 3rd party tool sff_extract) for SFF to FASTA.

    >>> print(record.annotations["clip_qual_left"])
    4
    >>> print(record.annotations["clip_qual_right"])
    134
    >>> print(record.seq[:4])
    tcag
    >>> print("%s...%s" % (record.seq[4:20], record.seq[120:134]))
    AATCATCCACTTTTTA...CAAAACACAAACAG
    >>> print(record.seq[134:])
    atcttatcaacaaaactcaaagttcctaactgagacacgcaacaggggataagacaaggcacacaggggataggnnnnnnnnnnn

The annotations dictionary also contains any adapter clip positions
(usually zero), and information about the flows. e.g.

    >>> len(record.annotations)
    11
    >>> print(record.annotations["flow_key"])
    TCAG
    >>> print(record.annotations["flow_values"][:10])
    (83, 1, 128, 7, 4, 84, 6, 106, 3, 172)
    >>> print(len(record.annotations["flow_values"]))
    400
    >>> print(record.annotations["flow_index"][:10])
    (1, 2, 3, 2, 2, 0, 3, 2, 3, 3)
    >>> print(len(record.annotations["flow_index"]))
    219

Note that to convert from a raw reading in flow_values to the corresponding
homopolymer stretch estimate, the value should be rounded to the nearest 100:

    >>> print("%r..." % [int(round(value, -2)) // 100
    ...                  for value in record.annotations["flow_values"][:10]])
    ...
    [1, 0, 1, 0, 0, 1, 0, 1, 0, 2]...

If a read name is exactly 14 alphanumeric characters, the annotations
dictionary will also contain meta-data about the read extracted by
interpretting the name as a 454 Sequencing System "Universal" Accession
Number. Note that if a read name happens to be exactly 14 alphanumeric
characters but was not generated automatically, these annotation records
will contain nonsense information.

    >>> print(record.annotations["region"])
    2
    >>> print(record.annotations["time"])
    [2008, 1, 9, 16, 16, 0]
    >>> print(record.annotations["coords"])
    (2434, 1658)

As a convenience method, you can read the file with SeqIO format name "sff-trim"
instead of "sff" to get just the trimmed sequences (without any annotation
except for the PHRED quality scores and anything encoded in the read names):

    >>> from Bio import SeqIO
    >>> for record in SeqIO.parse("Roche/E3MFGYR02_random_10_reads.sff", "sff-trim"):
    ...     print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
    ...
    E3MFGYR02JWQ7T 260 GGTCTACATGTTGGTTAACC...
    E3MFGYR02JA6IL 265 TTTTTTTTGGAAAGGAAAAC...
    E3MFGYR02JHD4H 292 AAAGACAAGTGGTATCAACG...
    E3MFGYR02GFKUC 295 CGGCCGGGCCTCTCATCGGT...
    E3MFGYR02FTGED 277 TGGTAATGGGGGGAAATTTA...
    E3MFGYR02FR9G7 256 CTCCGTAAGAAGGTGCTGCC...
    E3MFGYR02GAZMS 271 AAAGAAGTAAGGTAAATAAC...
    E3MFGYR02HHZ8O 150 ACTTTCTTCTTTACCGTAAC...
    E3MFGYR02GPGB1 221 AAGCAGTGGTATCAACGCAG...
    E3MFGYR02F7Z7G 130 AATCATCCACTTTTTAACGT...
```

Looking at the final record in more detail, note how this differs to the
example above:

```
    >>> print("%s %i" % (record.id, len(record)))
    E3MFGYR02F7Z7G 130
    >>> print("%s..." % record.seq[:10])
    AATCATCCAC...
    >>> print("%r..." % record.letter_annotations["phred_quality"][:10])
    [26, 15, 12, 21, 28, 21, 36, 28, 27, 27]...
    >>> len(record.annotations)
    3
    >>> print(record.annotations["region"])
    2
    >>> print(record.annotations["coords"])
    (2434, 1658)
    >>> print(record.annotations["time"])
    [2008, 1, 9, 16, 16, 0]
```

You might use the Bio.SeqIO.convert() function to convert the (trimmed) SFF
reads into a FASTQ file (or a FASTA file and a QUAL file), e.g.

```
    >>> from Bio import SeqIO
    >>> try:
    ...     from StringIO import StringIO # Python 2
    ... except ImportError:
    ...     from io import StringIO # Python 3
    ...
    >>> out_handle = StringIO()
    >>> count = SeqIO.convert("Roche/E3MFGYR02_random_10_reads.sff", "sff",
    ...                       out_handle, "fastq")
    ...
    >>> print("Converted %i records" % count)
    Converted 10 records
```

The output FASTQ file would start like this:

```
    >>> print("%s..." % out_handle.getvalue()[:50])
    @E3MFGYR02JWQ7T
    tcagGGTCTACATGTTGGTTAACCCGTACTGATT...
```

Bio.SeqIO.index() provides memory efficient random access to the reads in an
SFF file by name. SFF files can include an index within the file, which can
be read in making this very fast. If the index is missing (or in a format not
yet supported in Biopython) the file is indexed by scanning all the reads -
which is a little slower. For example,

```
    >>> from Bio import SeqIO
    >>> reads = SeqIO.index("Roche/E3MFGYR02_random_10_reads.sff", "sff")
    >>> record = reads["E3MFGYR02JHD4H"]
    >>> print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
    E3MFGYR02JHD4H 310 tcagAAAGACAAGTGGTATC...
    >>> reads.close()

Or, using the trimmed reads:

    >>> from Bio import SeqIO
    >>> reads = SeqIO.index("Roche/E3MFGYR02_random_10_reads.sff", "sff-trim")
    >>> record = reads["E3MFGYR02JHD4H"]
    >>> print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
    E3MFGYR02JHD4H 292 AAAGACAAGTGGTATCAACG...
    >>> reads.close()

You can also use the Bio.SeqIO.write() function with the "sff" format. Note
that this requires all the flow information etc, and thus is probably only
useful for SeqRecord objects originally from reading another SFF file (and
not the trimmed SeqRecord objects from parsing an SFF file as "sff-trim").

As an example, let's pretend this example SFF file represents some DNA which
was pre-amplified with a PCR primers AAAGANNNNN. The following script would
produce a sub-file containing all those reads whose post-quality clipping
region (i.e. the sequence after trimming) starts with AAAGA exactly (the non-
degenerate bit of this pretend primer):

    >>> from Bio import SeqIO
    >>> records = (record for record in
    ...            SeqIO.parse("Roche/E3MFGYR02_random_10_reads.sff", "sff")
    ...            if record.seq[record.annotations["clip_qual_left"]:].startswith("AAAGA"))
    ...
    >>> count = SeqIO.write(records, "temp_filtered.sff", "sff")
    >>> print("Selected %i records" % count)
    Selected 2 records

Of course, for an assembly you would probably want to remove these primers.
If you want FASTA or FASTQ output, you could just slice the SeqRecord. However,
if you want SFF output we have to preserve all the flow information - the trick
is just to adjust the left clip position!

    >>> from Bio import SeqIO
    >>> def filter_and_trim(records, primer):
    ...     for record in records:
    ...         if record.seq[record.annotations["clip_qual_left"]:].startswith(primer):
    ...             record.annotations["clip_qual_left"] += len(primer)
    ...             yield record
    ...
    >>> records = SeqIO.parse("Roche/E3MFGYR02_random_10_reads.sff", "sff")
    >>> count = SeqIO.write(filter_and_trim(records, "AAAGA"),
    ...                     "temp_filtered.sff", "sff")
    ...
    >>> print("Selected %i records" % count)
    Selected 2 records

We can check the results, note the lower case clipped region now includes the "AAAGA"
sequence:

    >>> for record in SeqIO.parse("temp_filtered.sff", "sff"):
    ...     print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
```

```
        ...
        E3MFGYR02JHD4H 310 tcagaaagaCAAGTGGTATC...
        E3MFGYR02GAZMS 278 tcagaaagaAGTAAGGTAAA...
        >>> for record in SeqIO.parse("temp_filtered.sff", "sff-trim"):
        ...     print("%s %i %s..." % (record.id, len(record), record.seq[:20]))
        ...
        E3MFGYR02JHD4H 287 CAAGTGGTATCAACGCAGAG...
        E3MFGYR02GAZMS 266 AGTAAGGTAAATAACAAACG...
        >>> import os
        >>> os.remove("temp_filtered.sff")

    For a description of the file format, please see the Roche manuals and:
    http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?cmd=show&f=formats&m=doc&s=formats

CLASSES
    Bio.SeqIO.Interfaces.SequenceWriter(builtins.object)
        SffWriter

    class SffWriter(Bio.SeqIO.Interfaces.SequenceWriter)
     |  SFF file writer.
     |
     |  Method resolution order:
     |      SffWriter
     |      Bio.SeqIO.Interfaces.SequenceWriter
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, handle, index=True, xml=None)
     |      Creates the writer object.
     |
     |      - handle - Output handle, ideally in binary write mode.
     |      - index - Boolean argument, should we try and write an index?
     |      - xml - Optional string argument, xml manifest to be recorded in the index
     |        block (see function ReadRocheXmlManifest for reading this data).
     |
     |  write_file(self, records)
     |      Use this to write an entire file containing the given records.
     |
     |  write_header(self)
     |
     |  write_record(self, record)
     |      Write a single additional record to the output file.
     |
     |      This assumes the header has been done.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from Bio.SeqIO.Interfaces.SequenceWriter:
     |
     |  clean(self, text)
     |      Use this to avoid getting newlines in the output.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from Bio.SeqIO.Interfaces.SequenceWriter:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

FUNCTIONS
    ReadRocheXmlManifest(handle)
        Reads any Roche style XML manifest data in the SFF "index".

        The SFF file format allows for multiple different index blocks, and Roche
        took advantage of this to define their own index block which also embeds
        an XML manifest string. This is not a publically documented extension to
        the SFF file format, this was reverse engineered.

        The handle should be to an SFF file opened in binary mode. This function
        will use the handle seek/tell functions and leave the handle in an
        arbitrary location.

        Any XML manifest found is returned as a Python string, which you can then
        parse as appropriate, or reuse when writing out SFF files with the
        SffWriter class.

        Returns a string, or raises a ValueError if an Roche manifest could not be
        found.

    SffIterator(handle, alphabet=DNAAlphabet(), trim=False)
        Iterate over Standard Flowgram Format (SFF) reads (as SeqRecord objects).

            - handle - input file, an SFF file, e.g. from Roche 454 sequencing.
              This must NOT be opened in universal read lines mode!
            - alphabet - optional alphabet, defaults to generic DNA.
            - trim - should the sequences be trimmed?

        The resulting SeqRecord objects should match those from a paired FASTA
        and QUAL file converted from the SFF file using the Roche 454 tool
        ssfinfo. i.e. The sequence will be mixed case, with the trim regions
        shown in lower case.

        This function is used internally via the Bio.SeqIO functions:

        >>> from Bio import SeqIO
        >>> for record in SeqIO.parse("Roche/E3MFGYR02_random_10_reads.sff", "sff"):
        ...     print("%s %i" % (record.id, len(record)))
        ...
        E3MFGYR02JWQ7T 265
        E3MFGYR02JA6IL 271
        E3MFGYR02JHD4H 310
        E3MFGYR02GFKUC 299
        E3MFGYR02FTGED 281
        E3MFGYR02FR9G7 261
        E3MFGYR02GAZMS 278
        E3MFGYR02HHZ8O 221
        E3MFGYR02GPGB1 269
        E3MFGYR02F7Z7G 219

        You can also call it directly:

        >>> with open("Roche/E3MFGYR02_random_10_reads.sff", "rb") as handle:
        ...     for record in SffIterator(handle):
        ...         print("%s %i" % (record.id, len(record)))
        ...
        E3MFGYR02JWQ7T 265
        E3MFGYR02JA6IL 271
        E3MFGYR02JHD4H 310
        E3MFGYR02GFKUC 299
        E3MFGYR02FTGED 281
        E3MFGYR02FR9G7 261
        E3MFGYR02GAZMS 278
        E3MFGYR02HHZ8O 221
        E3MFGYR02GPGB1 269
        E3MFGYR02F7Z7G 219

        Or, with the trim option:

        >>> with open("Roche/E3MFGYR02_random_10_reads.sff", "rb") as handle:
        ...     for record in SffIterator(handle, trim=True):
        ...         print("%s %i" % (record.id, len(record)))
        ...
        E3MFGYR02JWQ7T 260
        E3MFGYR02JA6IL 265
        E3MFGYR02JHD4H 292
        E3MFGYR02GFKUC 295
        E3MFGYR02FTGED 277
        E3MFGYR02FR9G7 256
        E3MFGYR02GAZMS 271
        E3MFGYR02HHZ8O 150
        E3MFGYR02GPGB1 221
        E3MFGYR02F7Z7G 130

DATA
    __docformat__ = 'restructuredtext en'
    print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0)...

FILE
    /home/tiago_antao/miniconda/lib/python3.5/site-packages/Bio/SeqIO/SffIO.py
```

20.1.13 Identifying open reading frames

A very simplistic first step at identifying possible genes is to look for open reading frames (ORFs). By this we mean look in all six frames for long regions without stop codons – an ORF is just a region of nucleotides with no in frame stop codons.

Of course, to find a gene you would also need to worry about locating a start codon, possible promoters – and in Eukaryotes there are introns to worry about too. However, this approach is still useful in viruses and Prokaryotes.

To show how you might approach this with Biopython, we'll need a sequence to search, and as an example we'll again use the bacterial plasmid – although this time we'll start with a plain FASTA file with no pre-marked genes: `` `NC_005816.fna `` <http://biopython.org/SRC/biopython/Tests/GenBank/NC_005816.fna>'__. This is a bacterial sequence, so we'll want to use NCBI codon table 11 (see Section [sec:translation] about translation).

```
In [22]: from Bio import SeqIO
         record = SeqIO.read("data/NC_005816.fna", "fasta")
         table = 11
         min_pro_len = 100
```

Here is a neat trick using the `Seq` object's `split` method to get a list of all the possible ORF translations in the six reading frames:

```
In [24]: for strand, nuc in [(+1, record.seq), (-1, record.seq.reverse_complement())]:
             for frame in range(3):
                 length = 3 * ((len(record)-frame) // 3) #Multiple of three
                 for pro in nuc[frame:frame+length].translate(table).split("*"):
                     if len(pro) >= min_pro_len:
                         print("%s...%s - length %i, strand %i, frame %i"
                               % (pro[:30], pro[-3:], len(pro), strand, frame))
```

```
GCLMKKSSIVATIITILSGSANAASSQLIP...YRF - length 315, strand 1, frame 0
KSGELRQTPPASSTLHLRLILQRSGVMMEL...NPE - length 285, strand 1, frame 1
GLNCSFFSICNWKFIDYINRLFQIIYLCKN...YYH - length 176, strand 1, frame 1
VKKILYIKALFLCTVIKLRRFIFSVNNMKF...DLP - length 165, strand 1, frame 1
NQIQGVICSPDSGEFMVTFETVMEIKILHK...GVA - length 355, strand 1, frame 2
RRKEHVSKKRRPQKRPRRRRFFHRLRPPDE...PTR - length 128, strand 1, frame 2
TGKQNSCQMSAIWQLRQNTATKTRQNRARI...AIK - length 100, strand 1, frame 2
QGSGYAFPHASILSGIAMSHFYFLVLHAVK...CSD - length 114, strand -1, frame 0
IYSTSEHTGEQVMRTLDEVIASRSPESQTR...FHV - length 111, strand -1, frame 0
WGKLQVIGLSMWMVLFSQRFDDWLNEQEDA...ESK - length 125, strand -1, frame 1
RGIFMSDTMVVNGSGGVPAFLFSGSTLSSY...LLK - length 361, strand -1, frame 1
WDVKTVTGVLHHPFHLTFSLCPEGATQSGR...VKR - length 111, strand -1, frame 1
LSHTVTDFTDQMAQVGLCQCVNVFLDEVTG...KAA - length 107, strand -1, frame 2
RALTGLSAPGIRSQTSCDRLRELRYVPVSL...PLQ - length 119, strand -1, frame 2
```

Note that here we are counting the frames from the 5' end (start) of *each* strand. It is sometimes easier to always count from the 5' end (start) of the *forward* strand.

You could easily edit the above loop based code to build up a list of the candidate proteins, or convert this to a list comprehension. Now, one thing this code doesn't do is keep track of where the proteins are.

You could tackle this in several ways. For example, the following code tracks the locations in terms of the protein counting, and converts back to the parent sequence by multiplying by three, then adjusting for the frame and strand:

```
from Bio import SeqIO
record = SeqIO.read("NC_005816.gb","genbank")
table = 11
min_pro_len = 100

def find_orfs_with_trans(seq, trans_table, min_protein_length):
    answer = []
    seq_len = len(seq)
    for strand, nuc in [(+1, seq), (-1, seq.reverse_complement())]:
        for frame in range(3):
            trans = str(nuc[frame:].translate(trans_table))
            trans_len = len(trans)
            aa_start = 0
            aa_end = 0
            while aa_start < trans_len:
                aa_end = trans.find("*", aa_start)
                if aa_end == -1:
                    aa_end = trans_len
                if aa_end-aa_start >= min_protein_length:
                    if strand == 1:
                        start = frame+aa_start*3
                        end = min(seq_len,frame+aa_end*3+3)
                    else:
                        start = seq_len-frame-aa_end*3-3
                        end = seq_len-frame-aa_start*3
                    answer.append((start, end, strand,
                                   trans[aa_start:aa_end]))
                aa_start = aa_end+1
    answer.sort()
    return answer

orf_list = find_orfs_with_trans(record.seq, table, min_pro_len)
for start, end, strand, pro in orf_list:
    print("%s...%s - length %i, strand %i, %i:%i" \
          % (pro[:30], pro[-3:], len(pro), strand, start, end))
```

And the output:

```
NQIQGVICSPDSGEFMVTFETVMEIKILHK...GVA - length 355, strand 1, 41:1109
WDVKTVTGVLHHPFHLTFSLCPEGATQSGR...VKR - length 111, strand -1, 491:827
KSGELRQTPPASSTLHLRLILQRSGVMMEL...NPE - length 285, strand 1, 1030:1888
RALTGLSAPGIRSQTSCDRLRELRYVPVSL...PLQ - length 119, strand -1, 2830:3190
RRKEHVSKKRRPQKRPRRRRFFHRLRPPDE...PTR - length 128, strand 1, 3470:3857
GLNCSFFSICNWKFIDYINRLFQIIYLCKN...YYH - length 176, strand 1, 4249:4780
RGIFMSDTMVVNGSGGVPAFLFSGSTLSSY...LLK - length 361, strand -1, 4814:5900
VKKILYIKALFLCTVIKLRRFIFSVNNMKF...DLP - length 165, strand 1, 5923:6421
LSHTVTDFTDQMAQVGLCQCVNVFLDEVTG...KAA - length 107, strand -1, 5974:6298
GCLMKKSSIVATIITILSGSANAASSQLIP...YRF - length 315, strand 1, 6654:7602
IYSTSEHTGEQVMRTLDEVIASRSPESQTR...FHV - length 111, strand -1, 7788:8124
WGKLQVIGLSMWMVLFSQRFDDWLNEQEDA...ESK - length 125, strand -1, 8087:8465
TGKQNSCQMSAIWQLRQNTATKTRQNRARI...AIK - length 100, strand 1, 8741:9044
QGSGYAFPHASILSGIAMSHFYFLVLHAVK...CSD - length 114, strand -1, 9264:9609
```

If you comment out the sort statement, then the protein sequences will be shown in the same order as before, so you can check this is doing the same thing. Here we have sorted them by location to make it easier to compare to the actual annotation in the GenBank file (as visualised in Section [sec:gd_nice_example]).

If however all you want to find are the locations of the open reading frames, then it is a waste of time to translate every possible codon, including doing the reverse complement to search the reverse strand too. All you need to do is search for the possible stop codons (and their reverse complements). Using regular expressions is an obvious approach here (see the Python module `re`). These are an extremely powerful (but rather complex) way of describing search strings, which are supported in lots of programming languages and also command line tools like `grep` as well). You can find whole books about this topic!

20.2 Sequence parsing plus simple plots

This section shows some more examples of sequence parsing, using the `Bio.SeqIO` module described in Chapter [chapter:Bio.SeqIO], plus the Python library matplotlib's `pylab` plotting interface (see the matplotlib website for a tutorial). Note that to follow these examples you will need matplotlib installed - but without it you can still try the data parsing bits.

20.2.1 Histogram of sequence lengths

There are lots of times when you might want to visualise the distribution of sequence lengths in a dataset – for example the range of contig sizes in a genome assembly project. In this example we'll reuse our orchid FASTA file ls_orchid.fasta which has only 94 sequences.

First of all, we will use `Bio.SeqIO` to parse the FASTA file and compile a list of all the sequence lengths. You could do this with a for loop, but I find a list comprehension more pleasing:

```
In [28]: from Bio import SeqIO
         sizes = [len(rec) for rec in SeqIO.parse("data/ls_orchid.fasta", "fasta")]
         len(sizes), min(sizes), max(sizes)
```

```
Out[28]: (94, 572, 789)
```

```
In [29]: sizes
```

```
Out[29]: [740,
          753,
          748,
          744,
          733,
          718,
          730,
          704,
          740,
          709,
          700,
          726,
          753,
          699,
          658,
          752,
          726,
          765,
          755,
          742,
          762,
          745,
          750,
          731,
          741,
          740,
          727,
          711,
          743,
          727,
          757,
          770,
          767,
          759,
          750,
          788,
          774,
          789,
          688,
          719,
          743,
          737,
          728,
          740,
          696,
          732,
          731,
          735,
          720,
          740,
          629,
          572,
          587,
          700,
          636,
          716,
          592,
          716,
          733,
          626,
          737,
          740,
          574,
          594,
          610,
          730,
          641,
          702,
          733,
          738,
          736,
          732,
          745,
          744,
          738,
          739,
          740,
          745,
          695,
          745,
          743,
          730,
          706,
          744,
          742,
          694,
          712,
          715,
          688,
          784,
          721,
          703,
          744,
          592]
```

Now that we have the lengths of all the genes (as a list of integers), we can use the matplotlib histogram function to display it.

```
from Bio import SeqIO
sizes = [len(rec) for rec in SeqIO.parse("ls_orchid.fasta", "fasta")]

import pylab
pylab.hist(sizes, bins=20)
pylab.title("%i orchid sequences\nLengths %i to %i" \
            % (len(sizes),min(sizes),max(sizes)))
pylab.xlabel("Sequence length (bp)")
pylab.ylabel("Count")
pylab.show()
```

That should pop up a new window containing the following graph:

That should pop up a new window containing the graph shown in Figure [fig:seq-len-hist].

Notice that most of these orchid sequences are about 740 bp long, and there could be two distinct classes of sequence here with a subset of shorter sequences.

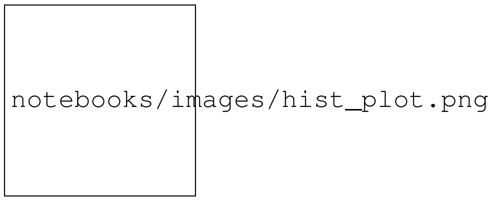


Fig. 20.1: Histogram of orchid sequence lengths.

Tip: Rather than using `pylab.show()` to show the plot in a window, you can also use `pylab.savefig(...)` to save the figure to a file (e.g. as a PNG or PDF).

20.2.2 Plot of sequence GC%

Another easily calculated quantity of a nucleotide sequence is the GC%. You might want to look at the GC% of all the genes in a bacterial genome for example, and investigate any outliers which could have been recently acquired by horizontal gene transfer. Again, for this example we'll reuse our orchid FASTA file ls_orchid.fasta.

First of all, we will use `Bio.SeqIO` to parse the FASTA file and compile a list of all the GC percentages. Again, you could do this with a for loop, but I prefer this:

```
from Bio import SeqIO
from Bio.SeqUtils import GC

gc_values = sorted(GC(rec.seq) for rec in SeqIO.parse("ls_orchid.fasta", "fasta"))
```

Having read in each sequence and calculated the GC%, we then sorted them into ascending order. Now we'll take this list of floating point values and plot them with matplotlib:

```
import pylab
pylab.plot(gc_values)
pylab.title("%i orchid sequences\nGC%% %0.1f to %0.1f" \
            % (len(gc_values),min(gc_values),max(gc_values)))
pylab.xlabel("Genes")
pylab.ylabel("GC%")
pylab.show()
```

As in the previous example, that should pop up a new window containing a graph:

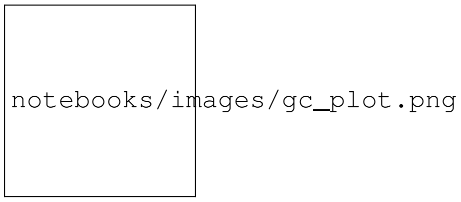


Fig. 20.2: Histogram of orchid sequence lengths.

As in the previous example, that should pop up a new window with the graph shown in Figure [fig:seq-gc-plot].

If you tried this on the full set of genes from one organism, you'd probably get a much smoother plot than this.

20.2.3 Nucleotide dot plots

A dot plot is a way of visually comparing two nucleotide sequences for similarity to each other. A sliding window is used to compare short sub-sequences to each other, often with a mis-match threshold. Here for simplicity we'll only look for perfect matches (shown in black

in Figure [fig:nuc-dot-plot]).

in the plot below).

To start off, we'll need two sequences. For the sake of argument, we'll just take the first two from our orchid FASTA file ls_orchid.fasta:

```
from Bio import SeqIO
handle = open("ls_orchid.fasta")
record_iterator = SeqIO.parse(handle, "fasta")
rec_one = next(record_iterator)
rec_two = next(record_iterator)
handle.close()
```

We're going to show two approaches. Firstly, a simple naive implementation which compares all the window sized sub-sequences to each other to compiles a similarity matrix. You could construct a matrix or array object, but here we just use a list of lists of booleans created with a nested list comprehension:

```
window = 7
seq_one = str(rec_one.seq).upper()
seq_two = str(rec_two.seq).upper()
data = [[(seq_one[i:i+window] <> seq_two[j:j+window]) \
        for j in range(len(seq_one)-window)] \
       for i in range(len(seq_two)-window)]
```

Note that we have *not* checked for reverse complement matches here. Now we'll use the matplotlib's `pylab.imshow()` function to display this data, first requesting the gray color scheme so this is done in black and white:

```
import pylab
pylab.gray()
pylab.imshow(data)
pylab.xlabel("%s (length %i bp)" % (rec_one.id, len(rec_one)))
pylab.ylabel("%s (length %i bp)" % (rec_two.id, len(rec_two)))
pylab.title("Dot plot using window size %i\n(allowing no mis-matches)" % window)
pylab.show()
```

That should pop up a new window containing a graph like this:

That should pop up a new window showing the graph in Figure [fig:nuc-dot-plot].

As you might have expected, these two sequences are very similar with a partial line of window sized matches along the diagonal. There are no off diagonal matches which would be indicative of inversions or other interesting events.

The above code works fine on small examples, but there are two problems applying this to larger sequences, which we will address below. First off all, this brute force approach to the all against all comparisons is very slow. Instead, we'll compile dictionaries mapping the window sized sub-sequences to their locations, and then take the set intersection to find those sub-sequences found in both sequences. This uses more memory, but is *much* faster. Secondly, the `pylab.imshow()` function is limited in the size of matrix it can display. As an alternative, we'll use the `pylab.scatter()` function.

We start by creating dictionaries mapping the window-sized sub-sequences to locations:

```
window = 7
dict_one = {}
```

```
dict_two = {}
for (seq, section_dict) in [(str(rec_one.seq).upper(), dict_one),
                            (str(rec_two.seq).upper(), dict_two)]:
    for i in range(len(seq)-window):
        section = seq[i:i+window]
        try:
            section_dict[section].append(i)
        except KeyError:
            section_dict[section] = [i]
#Now find any sub-sequences found in both sequences
#(Python 2.3 would require slightly different code here)
matches = set(dict_one).intersection(dict_two)
print("%i unique matches" % len(matches))
```

In order to use the `pylab.scatter()` we need separate lists for the x and y co-ordinates:

```
#Create lists of x and y co-ordinates for scatter plot
x = []
y = []
for section in matches:
    for i in dict_one[section]:
        for j in dict_two[section]:
            x.append(i)
            y.append(j)
```

We are now ready to draw the revised dot plot as a scatter plot:

```
import pylab
pylab.cla() #clear any prior graph
pylab.gray()
pylab.scatter(x,y)
pylab.xlim(0, len(rec_one)-window)
pylab.ylim(0, len(rec_two)-window)
pylab.xlabel("%s (length %i bp)" % (rec_one.id, len(rec_one)))
pylab.ylabel("%s (length %i bp)" % (rec_two.id, len(rec_two)))
pylab.title("Dot plot using window size %i\n(allowing no mis-matches)" % window)
pylab.show()
```

That should pop up a new window containing a graph like this:

That should pop up a new window showing the graph in Figure [fig:nuc-dot-plot-scatter].

Personally I find this second plot much easier to read! Again note that we have *not* checked for reverse complement matches here – you could extend this example to do this, and perhaps plot the forward matches in one color and the reverse matches in another.

20.2.4 Plotting the quality scores of sequencing read data

If you are working with second generation sequencing data, you may want to try plotting the quality data. Here is an example using two FASTQ files containing paired end reads, `SRR001666_1.fastq` for the forward reads, and `SRR001666_2.fastq` for the reverse reads. These were downloaded from the ENA sequence read archive FTP site (ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001666/SRR001666_1.fastq.gz and ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR001/SRR001666/SRR001666_2.fastq.gz), and are from *E. coli* – see http://www.ebi.ac.uk/ena/data/view/SRR001666 for details. In the following code the `pylab.subplot(...)` function is used in order to show the forward and reverse qualities on two subplots, side by side. There is also a little bit of code to only plot the first fifty reads.

```python
import pylab
from Bio import SeqIO
for subfigure in [1,2]:
    filename = "SRR001666_%i.fastq" % subfigure
    pylab.subplot(1, 2, subfigure)
    for i,record in enumerate(SeqIO.parse(filename, "fastq")):
        if i >= 50 : break #trick!
        pylab.plot(record.letter_annotations["phred_quality"])
    pylab.ylim(0,45)
    pylab.ylabel("PHRED quality score")
    pylab.xlabel("Position")
pylab.savefig("SRR001666.png")
print("Done")
```

You should note that we are using the `Bio.SeqIO` format name `fastq` here because the NCBI has saved these reads using the standard Sanger FASTQ format with PHRED scores. However, as you might guess from the read lengths, this data was from an Illumina Genome Analyzer and was probably originally in one of the two Solexa/Illumina FASTQ variant file formats instead.

This example uses the `pylab.savefig(...)` function instead of `pylab.show(...)`, but as mentioned before both are useful.

Fig. 20.3: Quality plot for some paired end reads.

The result is shown in Figure [fig:paired-end-qual-plot].

Here is the result:

20.3 Dealing with alignments

This section can been seen as a follow on to Chapter [chapter:Bio.AlignIO].

20.3.1 Calculating summary information

Once you have an alignment, you are very likely going to want to find out information about it. Instead of trying to have all of the functions that can generate information about an alignment in the alignment object itself, we've tried to separate out the functionality into separate classes, which act on the alignment.

Getting ready to calculate summary information about an object is quick to do. Let's say we've got an alignment object called `alignment`, for example read in using `Bio.AlignIO.read(...)` as described in Chapter [chapter:Bio.AlignIO]. All we need to do to get an object that will calculate summary information is:

```python
from Bio.Align import AlignInfo
summary_align = AlignInfo.SummaryInfo(alignment)
```

The `summary_align` object is very useful, and will do the following neat things for you:

1. Calculate a quick consensus sequence – see section [sec:consensus]
2. Get a position specific score matrix for the alignment – see section [sec:pssm]
3. Calculate the information content for the alignment – see section [sec:getting_info_content]
4. Generate information on substitutions in the alignment – section [sec:sub_matrix] details using this to generate a substitution matrix.

20.3.2 Calculating a quick consensus sequence

The `SummaryInfo` object, described in section [sec:summary_info], provides functionality to calculate a quick consensus of an alignment. Assuming we've got a `SummaryInfo` object called `summary_align` we can calculate a consensus by doing:

```
consensus = summary_align.dumb_consensus()
```

As the name suggests, this is a really simple consensus calculator, and will just add up all of the residues at each point in the consensus, and if the most common value is higher than some threshold value will add the common residue to the consensus. If it doesn't reach the threshold, it adds an ambiguity character to the consensus. The returned consensus object is Seq object whose alphabet is inferred from the alphabets of the sequences making up the consensus. So doing a `print consensus` would give:

```
consensus Seq('TATACATNAAAGNAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAAATGAAT
...', IUPACAmbiguousDNA())
```

You can adjust how `dumb_consensus` works by passing optional parameters:

the threshold This is the threshold specifying how common a particular residue has to be at a position before it is added. The default is 0.7 (meaning 70%).

the ambiguous character This is the ambiguity character to use. The default is 'N'.

the consensus alphabet This is the alphabet to use for the consensus sequence. If an alphabet is not specified than we will try to guess the alphabet based on the alphabets of the sequences in the alignment.

20.3.3 Position Specific Score Matrices

Position specific score matrices (PSSMs) summarize the alignment information in a different way than a consensus, and may be useful for different tasks. Basically, a PSSM is a count matrix. For each column in the alignment, the number of each alphabet letters is counted and totaled. The totals are displayed relative to some representative sequence along the left axis. This sequence may be the consesus sequence, but can also be any sequence in the alignment. For instance for the alignment,

```
GTATC
AT--C
CTGTC
```

the PSSM is:

```
G A T C
    G 1 1 0 1
    T 0 0 3 0
    A 1 1 0 0
    T 0 0 2 0
    C 0 0 0 3
```

Let's assume we've got an alignment object called `c_align`. To get a PSSM with the consensus sequence along the side we first get a summary object and calculate the consensus sequence:

```
summary_align = AlignInfo.SummaryInfo(c_align)
consensus = summary_align.dumb_consensus()
```

Now, we want to make the PSSM, but ignore any `N` ambiguity residues when calculating this:

```
my_pssm = summary_align.pos_specific_score_matrix(consensus,
                                                  chars_to_ignore = ['N'])
```

Two notes should be made about this:

1. To maintain strictness with the alphabets, you can only include characters along the top of the PSSM that are in the alphabet of the alignment object. Gaps are not included along the top axis of the PSSM.
2. The sequence passed to be displayed along the left side of the axis does not need to be the consensus. For instance, if you wanted to display the second sequence in the alignment along this axis, you would need to do:

```
second_seq = alignment.get_seq_by_num(1)
    my_pssm = summary_align.pos_specific_score_matrix(second_seq
                                                      chars_to_ignore = ['N'])
```

The command above returns a `PSSM` object. To print out the PSSM as shown above, we simply need to do a `print(my_pssm)`, which gives:

```
A   C   G   T
T  0.0 0.0 0.0 7.0
A  7.0 0.0 0.0 0.0
T  0.0 0.0 0.0 7.0
A  7.0 0.0 0.0 0.0
C  0.0 7.0 0.0 0.0
A  7.0 0.0 0.0 0.0
T  0.0 0.0 0.0 7.0
T  1.0 0.0 0.0 6.0
...
```

You can access any element of the PSSM by subscripting like `your_pssm[sequence_number][residue_count_name]`. For instance, to get the counts for the 'A' residue in the second element of the above PSSM you would do:

```
In [31]: print(my_pssm[1]["A"])

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-9d60637622f7> in <module>()
----> 1 print(my_pssm[1]["A"])

NameError: name 'my_pssm' is not defined
```

The structure of the PSSM class hopefully makes it easy both to access elements and to pretty print the matrix.

20.3.4 Information Content

A potentially useful measure of evolutionary conservation is the information content of a sequence.

A useful introduction to information theory targeted towards molecular biologists can be found at http://www.lecb.ncifcrf.gov/~toms/paper/primer/. For our purposes, we will be looking at the information content of a consesus sequence, or a portion of a consensus sequence. We calculate information content at a particular column in a multiple sequence alignment using the following formula:

$$IC_j = \sum_{i=1}^{N_a} P_{ij} \log\left(\frac{P_{ij}}{Q_i}\right)$$

where:

- IC_j – The information content for the j-th column in an alignment.
- N_a – The number of letters in the alphabet.
- P_{ij} – The frequency of a particular letter i in the j-th column (i. e. if G occurred 3 out of 6 times in an aligment column, this would be 0.5)
- Q_i – The expected frequency of a letter i. This is an optional argument, usage of which is left at the user's discretion. By default, it is automatically assigned to $0.05 = 1/20$ for a protein alphabet, and $0.25 = 1/4$ for a nucleic acid alphabet. This is for geting the information content without any assumption of prior distributions. When assuming priors, or when using a non-standard alphabet, you should supply the values for Q_i.

Well, now that we have an idea what information content is being calculated in Biopython, let's look at how to get it for a particular region of the alignment.

First, we need to use our alignment to get an alignment summary object, which we'll assume is called `summary_align` (see section [sec:summary_info]) for instructions on how to get this. Once we've got this object, calculating the information content for a region is as easy as:

```
info_content = summary_align.information_content(5, 30,
                                                 chars_to_ignore = ['N'])
```

Wow, that was much easier then the formula above made it look! The variable `info_content` now contains a float value specifying the information content over the specified region (from 5 to 30 of the alignment). We specifically ignore the ambiguity residue 'N' when calculating the information content, since this value is not included in our alphabet (so we shouldn't be interested in looking at it!).

As mentioned above, we can also calculate relative information content by supplying the expected frequencies:

```
expect_freq = {
    'A' : .3,
    'G' : .2,
    'T' : .3,
    'C' : .2}
```

The expected should not be passed as a raw dictionary, but instead by passed as a `SubsMat.FreqTable` object (see section [sec:freq_table] for more information about FreqTables). The FreqTable object provides a standard for associating the dictionary with an Alphabet, similar to how the Biopython Seq class works.

To create a FreqTable object, from the frequency dictionary you just need to do:

```
from Bio.Alphabet import IUPAC
from Bio.SubsMat import FreqTable

e_freq_table = FreqTable.FreqTable(expect_freq, FreqTable.FREQ,
                                   IUPAC.unambiguous_dna)
```

Now that we've got that, calculating the relative information content for our region of the alignment is as simple as:

```
info_content = summary_align.information_content(5, 30,
                                                 e_freq_table = e_freq_table,
                                                 chars_to_ignore = ['N'])
```

Now, `info_content` will contain the relative information content over the region in relation to the expected frequencies.

The value return is calculated using base 2 as the logarithm base in the formula above. You can modify this by passing the parameter `log_base` as the base you want:

```
info_content = summary_align.information_content(5, 30, log_base = 10,
                                                 chars_to_ignore = ['N'])
```

Well, now you are ready to calculate information content. If you want to try applying this to some real life problems, it would probably be best to dig into the literature on information content to get an idea of how it is used. Hopefully your digging won't reveal any mistakes made in coding this function!

20.3.5 Substitution Matrices

Substitution matrices are an extremely important part of everyday bioinformatics work. They provide the scoring terms for classifying how likely two different residues are to substitute for each other. This is essential in doing sequence comparisons. The book "Biological Sequence Analysis" by Durbin et al. provides a really nice introduction to Substitution Matrices and their uses. Some famous substitution matrices are the PAM and BLOSUM series of matrices.

Biopython provides a ton of common substitution matrices, and also provides functionality for creating your own substitution matrices.

20.3.6 Using common substitution matrices

20.3.7 Creating your own substitution matrix from an alignment

A very cool thing that you can do easily with the substitution matrix classes is to create your own substitution matrix from an alignment. In practice, this is normally done with protein alignments. In this example, we'll first get a Biopython alignment object and then get a summary object to calculate info about the alignment. The file containing protein.aln (also available online here) contains the Clustalw alignment output.

```
In [33]: from Bio import AlignIO
         from Bio import Alphabet
         from Bio.Alphabet import IUPAC
         from Bio.Align import AlignInfo
         filename = "data/protein.aln"
         alpha = Alphabet.Gapped(IUPAC.protein)
         c_align = AlignIO.read(filename, "clustal", alphabet=alpha)
         summary_align = AlignInfo.SummaryInfo(c_align)
```

Sections [sec:align_clustal] and [sec:summary_info] contain more information on doing this.

Now that we've got our `summary_align` object, we want to use it to find out the number of times different residues substitute for each other. To make the example more readable, we'll focus on only amino acids with polar charged side chains. Luckily, this can be done easily when generating a replacement dictionary, by passing in all of the characters that should be ignored. Thus we'll create a dictionary of replacements for only charged polar amino acids using:

```
In [34]: replace_info = summary_align.replacement_dictionary(["G", "A", "V", "L", "I",
         "M", "P", "F", "W", "S",
         "T", "N", "Q", "Y", "C"])
```

This information about amino acid replacements is represented as a python dictionary which will look something like (the order can vary):

```
{('R', 'R'): 2079.0, ('R', 'H'): 17.0, ('R', 'K'): 103.0, ('R', 'E'): 2.0,
('R', 'D'): 2.0, ('H', 'R'): 0, ('D', 'H'): 15.0, ('K', 'K'): 3218.0,
('K', 'H'): 24.0, ('H', 'K'): 8.0, ('E', 'H'): 15.0, ('H', 'H'): 1235.0,
('H', 'E'): 18.0, ('H', 'D'): 0, ('K', 'D'): 0, ('K', 'E'): 9.0,
('D', 'R'): 48.0, ('E', 'R'): 2.0, ('D', 'K'): 1.0, ('E', 'K'): 45.0,
('K', 'R'): 130.0, ('E', 'D'): 241.0, ('E', 'E'): 3305.0,
('D', 'E'): 270.0, ('D', 'D'): 2360.0}
```

This information gives us our accepted number of replacements, or how often we expect different things to substitute for each other. It turns out, amazingly enough, that this is all of the information we need to go ahead and create a substitution matrix. First, we use the replacement dictionary information to create an Accepted Replacement Matrix (ARM):

```
In [35]: from Bio import SubsMat
         my_arm = SubsMat.SeqMat(replace_info)
```

With this accepted replacement matrix, we can go right ahead and create our log odds matrix (i. e. a standard type Substitution Matrix):

```
In [36]: my_lom = SubsMat.make_log_odds_matrix(my_arm)
```

The log odds matrix you create is customizable with the following optional arguments:

- `exp_freq_table` – You can pass a table of expected frequencies for each alphabet. If supplied, this will be used instead of the passed accepted replacement matrix when calculate expected replacments.
- `logbase` - The base of the logarithm taken to create the log odd matrix. Defaults to base 10.
- `factor` - The factor to multiply each matrix entry by. This defaults to 10, which normally makes the matrix numbers easy to work with.
- `round_digit` - The digit to round to in the matrix. This defaults to 0 (i. e. no digits).

Once you've got your log odds matrix, you can display it prettily using the function `print_mat`. Doing this on our created matrix gives:

```
In [37]: my_lom.print_mat()
```

```
D   2
E  -1   1
H  -5  -4   3
K -10  -5  -4   1
R  -4  -8  -4  -2   2
   D   E   H   K   R
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 21

The Biopython testing framework

Biopython has a regression testing framework (the file `run_tests.py`) based on unittest, the standard unit testing framework for Python. Providing comprehensive tests for modules is one of the most important aspects of making sure that the Biopython code is as bug-free as possible before going out. It also tends to be one of the most undervalued aspects of contributing. This chapter is designed to make running the Biopython tests and writing good test code as easy as possible. Ideally, every module that goes into Biopython should have a test (and should also have documentation!). All our developers, and anyone installing Biopython from source, are strongly encouraged to run the unit tests.

21.1 Running the tests

When you download the Biopython source code, or check it out from our source code repository, you should find a subdirectory call `Tests`. This contains the key script `run_tests.py`, lots of individual scripts named `test_XXX.py`, a subdirectory called `output` and lots of other subdirectories which contain input files for the test suite.

As part of building and installing Biopython you will typically run the full test suite at the command line from the Biopython source top level directory using the following:

```
python setup.py test
```

This is actually equivalent to going to the `Tests` subdirectory and running:

```
python run_tests.py
```

You'll often want to run just some of the tests, and this is done like this:

```
python run_tests.py test_SeqIO.py test_AlignIO.py
```

When giving the list of tests, the `.py` extension is optional, so you can also just type:

```
python run_tests.py test_SeqIO test_AlignIO
```

To run the docstring tests (see section [section:doctest]), you can use

```
python run_tests.py doctest
```

By default, `run_tests.py` runs all tests, including the docstring tests.

If an individual test is failing, you can also try running it directly, which may give you more information.

Importantly, note that the individual unit tests come in two types:

- Simple print-and-compare scripts. These unit tests are essentially short example Python programs, which print out various output text. For a test file named `test_XXX.py` there will be a matching text file called `test_XXX` under the `output` subdirectory which contains the expected output. All that the test framework does to is run the script, and check the output agrees.
- Standard `unittest`- based tests. These will `import unittest` and then define `unittest.TestCase` classes, each with one or more sub-tests as methods starting with `test_` which check some specific aspect of the code. These tests should not print any output directly.

Currently, about half of the Biopython tests are `unittest`-style tests, and half are print-and-compare tests.

Running a simple print-and-compare test directly will usually give lots of output on screen, but does not check the output matches the expected output. If the test is failing with an exception error, it should be very easy to locate where exactly the script is failing. For an example of a print-and-compare test, try:

```
python test_SeqIO.py
```

The `unittest`-based tests instead show you exactly which sub-section(s) of the test are failing. For example,

```
python test_Cluster.py
```

21.1.1 Running the tests using Tox

Like most Python projects, you can also use Tox to run the tests on multiple Python versions, provided they are already installed in your system.

We do not provide the configuration `tox.ini` file in our code base because of difficulties pinning down user-specific settings (e.g. executable names of the Python versions). You may also only be interested in testing Biopython only against a subset of the Python versions that we support.

If you are interested in using Tox, you may start with the example `tox.ini` shown below:

```
[tox]
envlist = py26, py27, pypy, py33, py34, jython

[testenv]
changedir = Tests
commands = {envpython} run_tests.py --offline
deps =
    numpy
```

Using the template above, executing `tox` will test your Biopython code against Python2.6, Python2.7, PyPy, Python3.3, Python3.4, and Jython. It assumes that those Pythons' executables are named accordingly: python2.6 for Python2.6, and so on.

21.2 Writing tests

Let's say you want to write some tests for a module called `Biospam`. This can be a module you wrote, or an existing module that doesn't have any tests yet. In the examples below, we assume that `Biospam` is a module that does simple math.

Each Biopython test can have three important files and directories involved with it:

1. `test_Biospam.py` – The actual test code for your module.
2. `Biospam` [optional]– A directory where any necessary input files will be located. Any output files that will be generated should also be written here (and preferably cleaned up after the tests are done) to prevent clogging up the main Tests directory.
3. `output/Biospam` – [for print-and-compare tests only] This file contains the expected output from running `test_Biospam.py`. This file is not needed for `unittest`-style tests, since there the validation is done in the test script `test_Biospam.py` itself.

It's up to you to decide whether you want to write a print-and-compare test script or a `unittest`-style test script. The important thing is that you cannot mix these two styles in a single test script. Particularly, don't use `unittest` features in a print-and-compare test.

Any script with a `test_` prefix in the `Tests` directory will be found and run by `run_tests.py`. Below, we show an example test script `test_Biospam.py` both for a print-and-compare test and for a `unittest`-based test. If you put this script in the Biopython `Tests` directory, then `run_tests.py` will find it and execute the tests contained in it:

```
$ python run_tests.py
test_Ace ... ok
test_AlignIO ... ok
test_BioSQL ... ok
test_BioSQL_SeqIO ... ok
test_Biospam ... ok
test_CAPS ... ok
test_Clustalw ... ok
...
----------------------------------------------------------------------
Ran 107 tests in 86.127 seconds
```

21.2.1 Writing a print-and-compare test

A print-and-compare style test should be much simpler for beginners or novices to write - essentially it is just an example script using your new module.

Here is what you should do to make a print-and-compare test for the `Biospam` module.

1. Write a script called `test_Biospam.py`
   - This script should live in the Tests directory
   - The script should test all of the important functionality of the module (the more you test the better your test is, of course!).
   - Try to avoid anything which might be platform specific, such as printing floating point numbers without using an explicit formatting string to avoid having too many decimal places (different platforms can give very slightly different values).

2. If the script requires files to do the testing, these should go in the directory Tests/Biospam (if you just need something generic, like a FASTA sequence file, or a GenBank record, try and use an existing sample input file instead).

3. Write out the test output and verify the output to be correct.

   There are two ways to do this:

   (a) The long way:

      - Run the script and write its output to a file. On UNIX (including Linux and Mac OS X) machines, you would do something like: `python test_Biospam.py > test_Biospam` which would write the output to the file `test_Biospam`.
      - Manually look at the file `test_Biospam` to make sure the output is correct. When you are sure it is all right and there are no bugs, you need to quickly edit the `test_Biospam` file so that the first line is: '`test_Biospam`' (no quotes).
      - copy the `test_Biospam` file to the directory Tests/output

   (b) The quick way:

      - Run `python run_tests.py -g test_Biospam.py`. The regression testing framework is nifty enough that it'll put the output in the right place in just the way it likes it.
      - Go to the output (which should be in `Tests/output/test_Biospam`) and double check the output to make sure it is all correct.

4. Now change to the Tests directory and run the regression tests with `python run_tests.py`. This will run all of the tests, and you should see your test run (and pass!).

5. That's it! Now you've got a nice test for your module ready to check in, or submit to Biopython. Congratulations!

As an example, the `test_Biospam.py` test script to test the `addition` and `multiplication` functions in the `Biospam` module could look as follows:

```
from __future__ import print_function
from Bio import Biospam

print("2 + 3 =", Biospam.addition(2, 3))
print("9 - 1 =", Biospam.addition(9, -1))
print("2 * 3 =", Biospam.multiplication(2, 3))
print("9 * (- 1) =", Biospam.multiplication(9, -1))
```

We generate the corresponding output with `python run_tests.py -g test_Biospam.py`, and check the output file `output/test_Biospam`:

```
test_Biospam
2 + 3 = 5
9 - 1 = 8
2 * 3 = 6
9 * (- 1) = -9
```

Often, the difficulty with larger print-and-compare tests is to keep track which line in the output corresponds to which command in the test script. For this purpose, it is important to print out some markers to help you match lines in the input script with the generated output.

21.2.2 Writing a unittest-based test

We want all the modules in Biopython to have unit tests, and a simple print-and-compare test is better than no test at all. However, although there is a steeper learning curve, using the `unittest` framework gives a more structured result, and if there is a test failure this can clearly pinpoint which part of the test is going wrong. The sub-tests can also be run individually which is helpful for testing or debugging.

The `unittest`-framework has been included with Python since version 2.1, and is documented in the Python Library Reference (which I know you are keeping under your pillow, as recommended). There is also online documentaion for unittest. If you are familiar with the `unittest` system (or something similar like the nose test framework), you shouldn't have any trouble. You may find looking at the existing example within Biopython helpful too.

Here's a minimal `unittest`-style test script for `Biospam`, which you can copy and paste to get started:

```
import unittest
from Bio import Biospam

class BiospamTestAddition(unittest.TestCase):

    def test_addition1(self):
        result = Biospam.addition(2, 3)
        self.assertEqual(result, 5)

    def test_addition2(self):
        result = Biospam.addition(9, -1)
        self.assertEqual(result, 8)

class BiospamTestDivision(unittest.TestCase):

    def test_division1(self):
        result = Biospam.division(3.0, 2.0)
        self.assertAlmostEqual(result, 1.5)

    def test_division2(self):
        result = Biospam.division(10.0, -2.0)
        self.assertAlmostEqual(result, -5.0)


if __name__ == "__main__":
    runner = unittest.TextTestRunner(verbosity = 2)
    unittest.main(testRunner=runner)
```

In the division tests, we use `assertAlmostEqual` instead of `assertEqual` to avoid tests failing due to roundoff errors; see the `unittest` chapter in the Python documentation for details and for other functionality available in `unittest` (online reference).

These are the key points of `unittest`-based tests:

- Test cases are stored in classes that derive from `unittest.TestCase` and cover one basic aspect of your code
- You can use methods `setUp` and `tearDown` for any repeated code which should be run before and after each test method. For example, the `setUp` method might be used to create an instance of the object you are testing, or open a file handle. The `tearDown` should do any "tidying up", for example closing the file handle.
- The tests are prefixed with `test_` and each test should cover one specific part of what you are trying to test. You can have as many tests as you want in a class.
- At the end of the test script, you can use

```
if __name__ == "__main__":
        runner = unittest.TextTestRunner(verbosity = 2)
        unittest.main(testRunner=runner)
```

```
to execute the tests when the script is run by itself (rather than
imported from `run_tests.py`). If you run this script, then you'll
see something like the following:
```

$ python test_BiospamMyModule.py test_addition1 (**main**.TestAddition) ... ok test_addition2 (**main**.TestAddition) ... ok test_division1 (**main**.TestDivision) ... ok test_division2 (**main**.TestDivision) ... ok

- To indicate more clearly what each test is doing, you can add docstrings to each test. These are shown when running the tests, which can be useful information if a test is failing.

```
import unittest
    from Bio import Biospam

    class BiospamTestAddition(unittest.TestCase):

        def test_addition1(self):
            """An addition test"""
            result = Biospam.addition(2, 3)
            self.assertEqual(result, 5)

        def test_addition2(self):
            """A second addition test"""
            result = Biospam.addition(9, -1)
            self.assertEqual(result, 8)

    class BiospamTestDivision(unittest.TestCase):

        def test_division1(self):
            """Now let's check division"""
            result = Biospam.division(3.0, 2.0)
            self.assertAlmostEqual(result, 1.5)

        def test_division2(self):
            """A second division test"""
            result = Biospam.division(10.0, -2.0)
            self.assertAlmostEqual(result, -5.0)


    if __name__ == "__main__":
        runner = unittest.TextTestRunner(verbosity = 2)
        unittest.main(testRunner=runner)
```

```
Running the script will now show you:
```

```
$ python test_BiospamMyModule.py
An addition test ... ok
A second addition test ... ok
Now let's check division ... ok
A second division test ... ok
```

If your module contains docstring tests (see section [section:doctest]), you may want to include those in the tests to be run. You can do so as follows by modifying the code under `if __name__ == "__main__":` to look like this:

```
if __name__ == "__main__":
    unittest_suite = unittest.TestLoader().loadTestsFromName("test_Biospam")
    doctest_suite = doctest.DocTestSuite(Biospam)
    suite = unittest.TestSuite((unittest_suite, doctest_suite))
    runner = unittest.TextTestRunner(sys.stdout, verbosity = 2)
    runner.run(suite)
```

This is only relevant if you want to run the docstring tests when you execute `python test_Biospam.py`; with `python run_tests.py`, the docstring tests are run automatically (assuming they are included in the list of docstring tests in `run_tests.py`, see the section below).

21.3 Writing doctests

Python modules, classes and functions support built in documentation using docstrings. The doctest framework (included with Python) allows the developer to embed working examples in the docstrings, and have these examples automatically tested.

Currently only a small part of Biopython includes doctests. The `run_tests.py` script takes care of running the doctests. For this purpose, at the top of the `run_tests.py` script is a manually compiled list of modules to test, which allows us to skip modules with optional external dependencies which may not be installed (e.g. the Reportlab and NumPy libraries). So, if you've added some doctests to the docstrings in a Biopython module, in order to have them included in the Biopython test suite, you must update `run_tests.py` to include your module. Currently, the relevant part of `run_tests.py` looks as follows:

```
# This is the list of modules containing docstring tests.
# If you develop docstring tests for other modules, please add
# those modules here.
DOCTEST_MODULES = ["Bio.Seq",
                   "Bio.SeqRecord",
                   "Bio.SeqIO",
                   "...",
                  ]
#Silently ignore any doctests for modules requiring numpy!
try:
    import numpy
    DOCTEST_MODULES.extend(["Bio.Statistics.lowess"])
except ImportError:
    pass
```

Note that we regard doctests primarily as documentation, so you should stick to typical usage. Generally complicated examples dealing with error conditions and the like would be best left to a dedicated unit test.

Note that if you want to write doctests involving file parsing, defining the file location complicates matters. Ideally use relative paths assuming the code will be run from the `Tests` directory, see the `Bio.SeqIO` doctests for an example of this.

To run the docstring tests only, use

```
$ python run_tests.py doctest
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

Consider looking at the Substitution Matrices example in Chapter 19 - Cookbook

CHAPTER 22

Advanced

22.1 Parser Design

Many of the older Biopython parsers were built around an event-oriented design that includes Scanner and Consumer objects.

Scanners take input from a data source and analyze it line by line, sending off an event whenever it recognizes some information in the data. For example, if the data includes information about an organism name, the scanner may generate an `organism_name` event whenever it encounters a line containing the name.

Consumers are objects that receive the events generated by Scanners. Following the previous example, the consumer receives the `organism_name` event, and the processes it in whatever manner necessary in the current application.

This is a very flexible framework, which is advantageous if you want to be able to parse a file format into more than one representation. For example, the `Bio.GenBank` module uses this to construct either `SeqRecord` objects or file-format-specific record objects.

More recently, many of the parsers added for `Bio.SeqIO` and `Bio.AlignIO` take a much simpler approach, but only generate a single object representation (`SeqRecord` and `MultipleSeqAlignment` objects respectively). In some cases the `Bio.SeqIO` parsers actually wrap another Biopython parser - for example, the `Bio.SwissProt` parser produces SwissProt format specific record objects, which get converted into `SeqRecord` objects.

22.2 Substitution Matrices

22.2.1 SubsMat

This module provides a class and a few routines for generating substitution matrices, similar to BLOSUM or PAM matrices, but based on user-provided data. Additionally, you may select a matrix from MatrixInfo.py, a collection of established substitution matrices. The `SeqMat` class derives from a dictionary:

```
class SeqMat(dict)
```

The dictionary is of the form `{(i1,j1):n1, (i1,j2):n2,...,(ik,jk):nk}` where i, j are alphabet letters, and n is a value.

1. Attributes

   (a) `self.alphabet`: a class as defined in Bio.Alphabet

   (b) `self.ab_list`: a list of the alphabet's letters, sorted. Needed mainly for internal purposes

2. Methods

```
__init__(self,data=None,alphabet=None, mat_name='', build_later=0):
```

```
1.   `data`: can be either a dictionary, or another
     SeqMat instance.

2.   `alphabet`: a Bio.Alphabet instance. If not provided,
     construct an alphabet from data.

3.   `mat_name`: matrix name, such as "BLOSUM62" or "PAM250"

4.   `build_later`: default false. If true, user may supply only
     alphabet and empty dictionary, if intending to build the
     matrix later. this skips the sanity check of alphabet
     size vs. matrix size.
```

```
entropy(self,obs_freq_mat)
```

```
1.   `obs_freq_mat`: an observed frequency matrix. Returns the
     matrix's entropy, based on the frequency in `obs_freq_mat`.
     The matrix instance should be LO or SUBS.
```

```
sum(self)
```

```
Calculates the sum of values for each letter in the matrix's
alphabet, and returns it as a dictionary of the form
`{i1: s1, i2: s2,...,in:sn}`, where:

-    i: an alphabet letter;

-    s: sum of all values in a half-matrix for that letter;

-    n: number of letters in alphabet.
```

```
print_mat(self,f,format="%4d",bottomformat="%4s",alphabet=None)
```

```
prints the matrix to file handle f. `format` is the format field
for the matrix values; `bottomformat` is the format field for
the bottom row, containing matrix letters. Example output for a
3-letter alphabet matrix:
```

```
A 23
        B 12 34
        C 7  22  27
          A   B   C
```

```
The `alphabet` optional argument is a string of all characters
in the alphabet. If supplied, the order of letters along the
axes is taken from the string, rather than by
alphabetical order.
```

3. Usage

   The following section is laid out in the order by which most people wish to generate a log-odds matrix. Of course, interim matrices can be generated and investigated. Most people just want a log-odds matrix, that's all.

   (a) Generating an Accepted Replacement Matrix

   Initially, you should generate an accepted replacement matrix (ARM) from your data. The values in ARM are the counted number of replacements according to your data. The data could be a set of pairs or multiple alignments. So for instance if Alanine was replaced by Cysteine 10 times, and Cysteine by Alanine 12 times, the corresponding ARM entries would be:

```
('A','C'): 10, ('C','A'): 12
```

```
as order doesn't matter, user can already provide only one
entry:
```

```
('A','C'): 22
```

```
    A SeqMat instance may be initialized with either a full (first
    method of counting: 10, 12) or half (the latter method, 22)
    matrices. A full protein alphabet matrix would be of the size
    20x20 = 400. A half matrix of that alphabet would be 20x20/2 +
    20/2 = 210. That is because same-letter entries don't change.
    (The matrix diagonal). Given an alphabet size of N:

    1.  Full matrix size: N\*N

    2.  Half matrix size: N(N+1)/2

    The SeqMat constructor automatically generates a half-matrix, if
    a full matrix is passed. If a half matrix is passed, letters in
    the key should be provided in alphabetical order: ('A','C') and
    not ('C',A').

    At this point, if all you wish to do is generate a log-odds
    matrix, please go to the section titled Example of Use. The
    following text describes the nitty-gritty of internal functions,
    to be used by people who wish to investigate their
    nucleotide/amino-acid frequency data more thoroughly.

2.  Generating the observed frequency matrix (OFM)
```

```
    Use:
```

```
OFM = SubsMat._build_obs_freq_mat(ARM)
```

```
    The OFM is generated from the ARM, only instead of replacement
    counts, it contains replacement frequencies.

3.  Generating an expected frequency matrix (EFM)

    Use:
```

```
EFM = SubsMat._build_exp_freq_mat(OFM,exp_freq_table)
```

```
1.  `exp_freq_table`: should be a FreqTable instance. See
    section \[sec:freq\_table\] for detailed information
    on FreqTable. Briefly, the expected frequency table has the
    frequencies of appearance for each member of the alphabet.
    It is implemented as a dictionary with the alphabet letters
    as keys, and each letter's frequency as a value. Values sum
    to 1.

The expected frequency table can (and generally should) be
generated from the observed frequency matrix. So in most cases
you will generate `exp_freq_table` using:
```

```
from Bio import SubsMat
from Bio.SubsMat import _build_obs_freq_mat
OFM = _build_obs_freq_mat(ARM)
exp_freq_table = SubsMat._exp_freq_table_from_obs_freq(OFM)
EFM = SubsMat._build_exp_freq_mat(OFM, exp_freq_table)
```

```
    But you can supply your own `exp_freq_table`, if you wish

4.  Generating a substitution frequency matrix (SFM)

    Use:
```

```
SFM = SubsMat._build_subs_mat(OFM,EFM)
```

```
    Accepts an OFM, EFM. Provides the division product of the
    corresponding values.

5.  Generating a log-odds matrix (LOM)

    Use:
```

```
LOM=SubsMat._build_log_odds_mat(SFM[,logbase=10,factor=10.0,round_digit=1])
```

```
1.  Accepts an SFM.

2.  `logbase`: base of the logarithm used to generate the
    log-odds values.
```

```
3.  `factor`: factor used to multiply the log-odds values. Each
    entry is generated by log(LOM\[key\])\*factor And rounded to
    the `round_digit` place after the decimal point,
    if required.
```

4. Example of use

   As most people would want to generate a log-odds matrix, with minimum hassle, SubsMat provides one function which does it all:

```
make_log_odds_matrix(acc_rep_mat,exp_freq_table=None,logbase=10,
                          factor=10.0,round_digit=0):
```

```
1.  `acc_rep_mat`: user provided accepted replacements matrix

2.  `exp_freq_table`: expected frequencies table. Used if provided,
    if not, generated from the `acc_rep_mat`.

3.  `logbase`: base of logarithm for the log-odds matrix. Default
    base 10.

4.  `round_digit`: number after decimal digit to which result should
    be rounded. Default zero.
```

22.3 FreqTable

```
FreqTable.FreqTable(UserDict.UserDict)
```

1. Attributes:

   (a) `alphabet`: A Bio.Alphabet instance.

   (b) `data`: frequency dictionary

   (c) `count`: count dictionary (in case counts are provided).

2. Functions:

   (a) `read_count(f)`: read a count file from stream f. Then convert to frequencies.

   (b) `read_freq(f)`: read a frequency data file from stream f. Of course, we then don't have the counts, but it is usually the letter frequencies which are interesting.

3. Example of use: The expected count of the residues in the database is sitting in a file, whitespace delimited, in the following format (example given for a 3-letter alphabet):

```
A   35
    B   65
    C   100
```

```
And will be read using the
`FreqTable.read_count(file_handle)` function.

An equivalent frequency file:
```

```
A  0.175
    B  0.325
    C  0.5
```

```
Conversely, the residue frequencies or counts can be passed as
a dictionary. Example of a count dictionary (3-letter alphabet):
```

```
{'A': 35, 'B': 65, 'C': 100}
```

```
Which means that an expected data count would give a 0.5 frequency
for 'C', a 0.325 probability of 'B' and a 0.175 probability of 'A'
out of 200 total, sum of A, B and C)

A frequency dictionary for the same data would be:
```

```
{'A': 0.175, 'B': 0.325, 'C': 0.5}
```

```
Summing up to 1.

When passing a dictionary as an argument, you should indicate
whether it is a count or a frequency dictionary. Therefore the
FreqTable class constructor requires two arguments: the dictionary
itself, and FreqTable.COUNT or FreqTable.FREQ indicating counts or
frequencies, respectively.

Read expected counts. readCount will already generate the
frequencies Any one of the following may be done to geerate the
frequency table (ftab):
```

```
from Bio.SubsMat import *
ftab = FreqTable.FreqTable(my_frequency_dictionary, FreqTable.FREQ)
ftab = FreqTable.FreqTable(my_count_dictionary, FreqTable.COUNT)
ftab = FreqTable.read_count(open('myCountFile'))
ftab = FreqTable.read_frequency(open('myFrequencyFile'))
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 23

Where to go from here – contributing to Biopython

23.1 Bug Reports + Feature Requests

Getting feedback on the Biopython modules is very important to us. Open-source projects like this benefit greatly from feedback, bug-reports (and patches!) from a wide variety of contributors.

The main forums for discussing feature requests and potential bugs are the Biopython mailing lists:

- – An unmoderated list for discussion of anything to do with Biopython.
- – A more development oriented list that is mainly used by developers (but anyone is free to contribute!).

Additionally, if you think you've found a new bug, you can submit it to our issue tracker at https://github.com/biopython/biopython/issues (this has replaced the older tracker hosted at http://redmine.open-bio.org/projects/biopython). This way, it won't get buried in anyone's Inbox and forgotten about.

23.2 Mailing lists and helping newcomers

We encourage all our uses to sign up to the main Biopython mailing list. Once you've got the hang of an area of Biopython, we'd encourage you to help answer questions from beginners. After all, you were a beginner once.

23.3 Contributing Documentation

We're happy to take feedback or contributions - either via a bug-report or on the Mailing List. While reading this tutorial, perhaps you noticed some topics you were interested in which were missing, or not clearly explained. There is also Biopython's built in documentation (the docstrings, these are also online), where again, you may be able to help fill in any blanks.

23.4 Contributing cookbook examples

As explained in Chapter [chapter:cookbook], Biopython now has a wiki collection of user contributed "cookbook" examples, http://biopython.org/wiki/Category:Cookbook – maybe you can add to this?

23.5 Maintaining a distribution for a platform

We currently provide source code archives (suitable for any OS, if you have the right build tools installed), and Windows Installers which are just click and run. This covers all the major operating systems.

Most major Linux distributions have volunteers who take these source code releases, and compile them into packages for Linux users to easily install (taking care of dependencies etc). This is really great and we are of course very grateful. If you would like to contribute to this work, please find out more about how your Linux distribution handles this.

Below are some tips for certain platforms to maybe get people started with helping out:

Windows – Windows products typically have a nice graphical installer that installs all of the essential components in the right place. We use Distutils to create a installer of this type fairly easily.

You must first make sure you have a C compiler on your Windows computer, and that you can compile and install things (this is the hard bit - see the Biopython installation instructions for info on how to do this).

Once you are setup with a C compiler, making the installer just requires doing:

```
python setup.py bdist_wininst
```

```
Now you've got a Windows installer. Congrats! At the moment we have
no trouble shipping installers built on 32 bit windows. If anyone
would like to look into supporting 64 bit Windows that would
be great.
```

RPMs – RPMs are pretty popular package systems on some Linux platforms. There is lots of documentation on RPMs available at http://www.rpm.org to help you get started with them. To create an RPM for your platform is really easy. You just need to be able to build the package from source (having a C compiler that works is thus essential) – see the Biopython installation instructions for more info on this.

To make the RPM, you just need to do:

```
python setup.py bdist_rpm
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 24

Appendix: Useful stuff about Python

If you haven't spent a lot of time programming in Python, many questions and problems that come up in using Biopython are often related to Python itself. This section tries to present some ideas and code that come up often (at least for us!) while using the Biopython libraries. If you have any suggestions for useful pointers that could go here, please contribute!

24.1 What the heck is a handle?

Handles are mentioned quite frequently throughout this documentation, and are also fairly confusing (at least to me!). Basically, you can think of a handle as being a "wrapper" around text information.

Handles provide (at least) two benefits over plain text information:

1. They provide a standard way to deal with information stored in different ways. The text information can be in a file, or in a string stored in memory, or the output from a command line program, or at some remote website, but the handle provides a common way of dealing with information in all of these formats.
2. They allow text information to be read incrementally, instead of all at once. This is really important when you are dealing with huge text files which would use up all of your memory if you had to load them all.

Handles can deal with text information that is being read (e. g. reading from a file) or written (e. g. writing information to a file). In the case of a "read" handle, commonly used functions are `read()`, which reads the entire text information from the handle, and `readline()`, which reads information one line at a time. For "write" handles, the function `write()` is regularly used.

The most common usage for handles is reading information from a file, which is done using the built-in Python function `open`. Here, we open a handle to the file m_cold.fasta (also available online here):

```
In [5]: handle = open("data/m_cold.fasta", "r")
        handle.readline()
```

```
Out[5]: ">gi|8332116|gb|BE037100.1|BE037100 MP14H09 MP Mesembryanthemum crystallinum cDNA 5' similar
```

Handles are regularly used in Biopython for passing information to parsers. For example, since Biopython 1.54 the main functions in `Bio.SeqIO` and `Bio.AlignIO` have allowed you to use a filename instead of a handle:

```
In [7]: from Bio import SeqIO
        for record in SeqIO.parse("data/m_cold.fasta", "fasta"):
            print(record.id, len(record))
```

```
gi|8332116|gb|BE037100.1|BE037100 1111
```

On older versions of Biopython you had to use a handle, e.g.

```
In [9]: from Bio import SeqIO
        handle = open("data/m_cold.fasta", "r")
        for record in SeqIO.parse(handle, "fasta"):
            print(record.id, len(record))
        handle.close()
```

```
gi|8332116|gb|BE037100.1|BE037100 1111
```

This pattern is still useful - for example suppose you have a gzip compressed FASTA file you want to parse:

```
import gzip
from Bio import SeqIO
handle = gzip.open("m_cold.fasta.gz")
for record in SeqIO.parse(handle, "fasta"):
    print(record.id, len(record))
handle.close()
```

See Section [sec:SeqIO_compressed] for more examples like this, including reading bzip2 compressed files.

24.1.1 Creating a handle from a string

One useful thing is to be able to turn information contained in a string into a handle. The following example shows how to do this using `cStringIO` from the Python standard library:

```
In [1]: my_info = 'A string\n with multiple lines.'
        print(my_info)
```

```
A string
 with multiple lines.
```

```
In [4]: from io import StringIO
        my_info_handle = StringIO(my_info)
        first_line = my_info_handle.readline()
        print(first_line)
```

```
A string

```

```
In [10]: second_line = my_info_handle.readline()
         print(second_line)
```

```
 with multiple lines.
```

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 25

About the contents

The contents of these notebooks have several sources (mostly the Biopython Tutorial and Peter Cock's workshop). Credit to the source of will be given at the start of the notebook.

The current version of these notebooks losely follows the structure of the Tutorial.

As IPython Notebook is generaly available with matplotlib, NumPy and SciPy I will make use of these if I found it relevant.

25.1 Authorship

The authors of the Biopython tutorial and cookbook are: Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczynski

The people doing the conversion to IPython Notebook format are: Vincent Davis and Tiago Antao

Source of the materials: Biopython cookbook (adapted) Status: Draft

CHAPTER 26

References

1. Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, Michiel J. L. de Hoon: "Biopython: freely available Python tools for computational molecular biology and bioinformatics". Bioinformatics 25 (11), 1422–1423 (2009). doi:10.1093/bioinformatics/btp163,

2. Leighton Pritchard, Jennifer A. White, Paul R.J. Birch, Ian K. Toth: "GenomeDiagram: a python package for the visualization of large-scale genomic data". Bioinformatics 22 (5): 616–617 (2006). doi:10.1093/bioinformatics/btk021,

3. Ian K. Toth, Leighton Pritchard, Paul R. J. Birch: "Comparative genomics reveals what makes an enterobacterial plant pathogen". Annual Review of Phytopathology 44: 305–336 (2006). doi:10.1146/annurev.phyto.44.070505.143444,

4. Géraldine A. van der Auwera, Jaroslaw E. Król, Haruo Suzuki, Brian Foster, Rob van Houdt, Celeste J. Brown, Max Mergeay, Eva M. Top: "Plasmids captured in C. metallidurans CH34: defining the PromA family of broad-host-range plasmids". Antonie van Leeuwenhoek 96 (2): 193–204 (2009). doi:10.1007/s10482-009-9316-9

5. Caroline Proux, Douwe van Sinderen, Juan Suarez, Pilar Garcia, Victor Ladero, Gerald F. Fitzgerald, Frank Desiere, Harald Brüssow: "The dilemma of phage taxonomy illustrated by comparative genomics of Sfi21-Like Siphoviridae in lactic acid bacteria". Journal of Bacteriology 184 (21): 6026–6036 (2002). http://dx.doi.org/10.1128/JB.184.21.6026-6036.2002

6. Florian Jupe, Leighton Pritchard, Graham J. Etherington, Katrin MacKenzie, Peter JA Cock, Frank Wright, Sanjeev Kumar Sharma1, Dan Bolser, Glenn J Bryan, Jonathan DG Jones, Ingo Hein: "Identification and localisation of the NB-LRR gene family within the potato genome". BMC Genomics 13: 75 (2012). http://dx.doi.org/10.1186/1471-2164-13-75

7. Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, Peter M. Rice: "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants". Nucleic Acids Research 38 (6): 1767–1771 (2010). doi:10.1093/nar/gkp1137

8. Patrick O. Brown, David Botstein: "Exploring the new world of the genome with DNA microarrays". Nature Genetics 21 (Supplement 1), 33–37 (1999). doi:10.1038/4462

9. Eric Talevich, Brandon M. Invergo, Peter J.A. Cock, Brad A. Chapman: "Bio.Phylo: A unified toolkit for processing, analyzing and visualizing phylogenetic trees in Biopython". BMC Bioinformatics 13: 209 (2012).

doi:10.1186/1471-2105-13-209

10. Athel Cornish-Bowden: "Nomenclature for incompletely specified bases in nucleic acid sequences: Recommendations 1984." Nucleic Acids Research 13 (9): 3021–3030 (1985). doi:10.1093/nar/13.9.3021

11. Douglas R. Cavener: "Comparison of the consensus sequence flanking translational start sites in Drosophila and vertebrates." Nucleic Acids Research 15 (4): 1353–1361 (1987). doi:10.1093/nar/15.4.1353

12. Timothy L. Bailey and Charles Elkan: "Fitting a mixture model by expectation maximization to discover motifs in biopolymers", Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology 28–36. AAAI Press, Menlo Park, California (1994).

13. Brad Chapman and Jeff Chang: "Biopython: Python tools for computational biology". ACM SIGBIO Newsletter 20 (2): 15–19 (August 2000).

14. Michiel J. L. de Hoon, Seiya Imoto, John Nolan, Satoru Miyano: "Open source clustering software". Bioinformatics 20 (9): 1453–1454 (2004). doi:10.1093/bioinformatics/bth078

15. Michiel B. Eisen, Paul T. Spellman, Patrick O. Brown, David Botstein: "Cluster analysis and display of genome-wide expression patterns". Proceedings of the National Academy of Science USA 95 (25): 14863–14868 (1998). doi:10.1073/pnas.96.19.10943-c

16. Gene H. Golub, Christian Reinsch: "Singular value decomposition and least squares solutions". In Handbook for Automatic Computation, 2, (Linear Algebra) (J. H. Wilkinson and C. Reinsch, eds), 134–151. New York: Springer-Verlag (1971).

17. Gene H. Golub, Charles F. Van Loan: Matrix computations, 2nd edition (1989).

18. Thomas Hamelryck and Bernard Manderick: 11PDB parser and structure class implemented in Python". Bioinformatics, 19 (17): 2308–2310 (2003) doi: 10.1093/bioinformatics/btg299.

19. Thomas Hamelryck: "Efficient identification of side-chain patterns using a multidimensional index tree". Proteins 51 (1): 96–108 (2003). doi:10.1002/prot.10338

20. Thomas Hamelryck: "An amino acid has two sides; A new 2D measure provides a different view of solvent exposure". Proteins 59 (1): 29–48 (2005). doi:10.1002/prot.20379.

21. John A. Hartiga. Clustering algorithms. New York: Wiley (1975).

22. Anil L. Jain, Richard C. Dubes: Algorithms for clustering data. Englewood Cliffs, N.J.: Prentice Hall (1988).

23. Voratas Kachitvichyanukul, Bruce W. Schmeiser: Binomial Random Variate Generation. Communications of the ACM 31 (2): 216–222 (1988). doi:10.1145/42372.42381

24. Teuvo Kohonen: "Self-organizing maps", 2nd Edition. Berlin; New York: Springer-Verlag (1997).

25. Pierre L'Ecuyer: "Efficient and Portable Combined Random Number Generators." Communications of the ACM 31 (6): 742–749,774 (1988). doi:10.1145/62959.62969

26. Indraneel Majumdar, S. Sri Krishna, Nick V. Grishin: "PALSSE: A program to delineate linear secondary structural elements from protein structures." BMC Bioinformatics, 6: 202 (2005). doi:10.1186/1471-2105-6-202.

27. V. Matys, E. Fricke, R. Geffers, E. Gössling, M. Haubrock, R. Hehl, K. Hornischer, D. Karas, A.E. Kel, O.V. Kel-Margoulis, D.U. Kloos, S. Land, B. Lewicki-Potapov, H. Michael, R. Münch, I. Reuter, S. Rotert, H. Saxel, M. Scheer, S. Thiele, E. Wingender E: "TRANSFAC: transcriptional regulation, from patterns to profiles." Nucleic Acids Research 31 (1): 374–378 (2003). doi:10.1093/nar/gkg108

28. Robin Sibson: "SLINK: An optimally efficient algorithm for the single-link cluster method". The Computer Journal 16 (1): 30–34 (1973). doi:10.1093/comjnl/16.1.30

29. George W. Snedecor, William G. Cochran: Statistical methods. Ames, Iowa: Iowa State University Press (1989).

30. Pablo Tamayo, Donna Slonim, Jill Mesirov, Qing Zhu, Sutisak Kitareewan, Ethan Dmitrovsky, Eric S. Lander, Todd R. Golub: "Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation". Proceedings of the National Academy of Science USA 96 (6): 2907–2912 (1999). doi:10.1073/pnas.96.6.2907

31. Robert C. Tryon, Daniel E. Bailey: Cluster analysis. New York: McGraw-Hill (1970).

32. John W. Tukey: "Exploratory data analysis". Reading, Mass.: Addison-Wesley Pub. Co. (1977).

33. Ka Yee Yeung, Walter L. Ruzzo: "Principal Component Analysis for clustering gene expression data". Bioinformatics 17 (9): 763–774 (2001). doi:10.1093/bioinformatics/17.9.763

34. Alok Saldanha: "Java Treeview—extensible visualization of microarray data". Bioinformatics 20 (17): 3246–3248 (2004). http://dx.doi.org/10.1093/bioinformatics/bth349