**Course: Advanced Bioinformatics**

**Module title: Introduction to BioPython**

**Module no. : 202**

**What can you do with Biopython?**

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.

**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.

**Working with sequences**

Disputably (of course!), the central object in bioinformatics is the sequence. Thus, we’ll start with a quick introduction to the Biopython mechanisms for dealing with sequences, the Seq object. 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 - the “>>>” represents the Python prompt followed by what you would type in:

>>> from Bio.Seq import Seq

>>> my\_seq = Seq("AGTACACTGGT")

>>> my\_seq

Seq('AGTACACTGGT', Alphabet())

>>> print(my\_seq)

AGTACACTGGT

>>> my\_seq.alphabet

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:

>>> my\_seq

Seq('AGTACACTGGT', Alphabet())

>>> my\_seq.complement()

Seq('TCATGTGACCA', Alphabet())

>>> my\_seq.reverse\_complement()

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.

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!

**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 (if you wonder why, have a look at some [Lady Slipper Orchids photos on Flickr](http://www.flickr.com/search/?q=lady+slipper+orchid&s=int&z=t), or try a [Google Image Search](http://images.google.com/images?q=lady%20slipper%20orchid)).

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*.

**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 more in Chapter [5](http://biopython.org/DIST/docs/tutorial/Tutorial.html#chapter:Bio.SeqIO). 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](http://www.ncbi.nlm.nih.gov/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](http://biopython.org/DIST/docs/tutorial/examples/ls_orchid.fasta) and [ls\_orchid.gbk](http://biopython.org/DIST/docs/tutorial/examples/ls_orchid.gbk), also included with the Biopython source code under docs/tutorial/examples/).

**Simple FASTA parsing example**

If you open the lady slipper orchids FASTA file [ls\_orchid.fasta](http://biopython.org/DIST/docs/tutorial/examples/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

CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTG

AATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGG

...

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:

from Bio import SeqIO

for seq\_record in SeqIO.parse("ls\_orchid.fasta", "fasta"):

print(seq\_record.id)

print(repr(seq\_record.seq))

print(len(seq\_record))

You should get something like this on your screen:

gi|2765658|emb|Z78533.1|CIZ78533

Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', SingleLetterAlphabet())

740

...

gi|2765564|emb|Z78439.1|PBZ78439

Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', SingleLetterAlphabet())

592

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.

**Simple GenBank parsing example**

Now let’s load the GenBank file [ls\_orchid.gbk](http://biopython.org/DIST/docs/tutorial/examples/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:

from Bio import SeqIO

for seq\_record in SeqIO.parse("ls\_orchid.gbk", "genbank"):

print(seq\_record.id)

print(repr(seq\_record.seq))

print(len(seq\_record))

This should give:

Z78533.1

Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC', IUPACAmbiguousDNA())

740

...

Z78439.1

Seq('CATTGTTGAGATCACATAATAATTGATCGAGTTAATCTGGAGGATCTGTTTACT...GCC', IUPACAmbiguousDNA())

592

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.

**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.

**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](http://www.ncbi.nlm.nih.gov/Entrez/) (and [PubMed](http://www.ncbi.nlm.nih.gov/PubMed/)) from the NCBI – See Chapter [9](http://biopython.org/DIST/docs/tutorial/Tutorial.html#chapter:entrez).
* [ExPASy](http://www.expasy.org/) – See Chapter [10](http://biopython.org/DIST/docs/tutorial/Tutorial.html#chapter:swiss_prot).
* [SCOP](http://scop.mrc-lmb.cam.ac.uk/scop/) – See the Bio.SCOP.search() function.

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.

**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.