BioPython

Biopython and Custom Objects: Biopython is a very useful set of modules made for biologists. It's most useful attribute (in my book) is that it makes handling sequence data pretty easy. In particular it can *parse* several different popular file formats, convert between them (most of the time), and write them back out. It's a large library, and we won't explain much of it in detail in this class, so we'll direct you to its online tutorial: http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc4

It will be easier to understand their documentation if you understand a bit more Python lingo. Biopython (and every other library of its type) makes use of a new type of object, called a *class*. These are the most objecty objects out there, and for some people, the word 'object' is reserved for classes. Basically they are custom data structures that have their own set of attributes and functions. The idea is that you want to take data and pack it into a structure that the user can perform several known actions upon, *and no more*. We've mentioned this idea, called data hiding, before: it enhances repeatability.

```
Data --> Parser --> Custom Object --> Actions 1,2,3 etc. --> Output
```

Installing Biopython: http://biopython.org/wiki/Download

Reading and Writing: Biopython has really awesome functions for parsing sequence data files into Biopython objects. The object I use most are SeqRecord and Alignment objects. Use either `.read()`, if there's just one sequence/alignment in the file, or `.parse()` if there's several.

```
>>> from Bio import SeqIO
>>> records = SeqIO.parse('my_seqs.fas','fasta')
```

Now you have a variable (an iterator type) full of SeqRecord objects. And this works the same for GenBank files, which is really useful. File conversion is pretty easy too.

```
>>> SeqIO.convert('my_genbank.gbk','genbank','my_fasta.fas','fasta')
```

This will create a new fasta file from your GenBank file without even needing to read or parse them. I've had trouble converting to Nexus format, though, so be careful.

SeqRecord Objects (Tutorial 4.1): This class includes a sequence and several attributes. The ones you'll use most are 'id', 'name', 'description.' Following on our example above:

```
>>> for rec in records:
…       print rec
…
ID: gi|38489212|gb|AAR21291.1|
Name: gi|38489212|gb|AAR21291.1|
Description: gi|38489212|gb|AAR21291.1| sodium channel [Bacillus
   pseudofirmus OF4]
Number of features: 0
Seq('MENNPAEQQVPPLVALAQRIVFHKAFTPTIITLIIINAIIVGLETYPTVYQGYN...KKG',
   SingleLetterAlphabet())
…
```

We've just printed the objects themselves. The SeqRecord object has a Seq object nested within it, and several associated attributes, discussed above. There are no 'features' because we parsed it from a fasta file which only has one measly description line. If we'd parsed in a GenBank file, there'd be much more associated information. But you usually want these attributes as strings. since records was an iterator, and you are now at the bottom of it, you need to re-assign.

```
>>> records = SeqIO.parse('my_seqs.fas','fasta')
>>> for rec in records:
…       print rec.id
…
gi|38489212|gb|AAR21291.1|
```

I hope it's easy to see how useful this is. Tacking a simple `.split()[1]` on the end of `rec.id` above can get you the accession numbers for every sequence in the file. This is especially useful for extracting information from GenBank files, which are very complex. Biopython also has analogous functions for alignments.