Notes on using BioPython

What is BioPython

The `BioPython` libraries extend the standard `Python` language with objects and methods pertaining to computational molecular biology. These objects allow the allow the manipulation and analysis of sequence data while hiding a large proportion of the inner workings from the user. An object such as a Sequence object holds not only data, but functions or methods which operate on the data. What this means is that a BioPython object may not only hold a sequence of DNA but also the methods to reverse complement the sequence, output its accession number, or maybe list the coding exons.

The Seq object

The following program shows the creation of a sequence object *my_seq* with a string of nucleotides using the `Seq` constructor function. Notice that we have imported `Seq` from `Bio.Seq` in the first few lines. Importing `Seq` gives us a template (a class, in object-oriented terminology) for creating objects with the `Seq` properties (variables and methods). Once we have the `Seq` object *my_seq* we can find the sequence length (which is just the string length), the reverse complement of the sequence, and other properties, using the functions available in the `Seq` module in the `Biopython` libraru. .

```
#!/usr/bin/python

from Bio.Seq import Seq

#create a sequence object
my_seq = Seq('CATGTAGACTAG')

seq_length =  len(my_seq)

# print the sequence length
print seq_length

# to print length and text, must explicitly cast the length (type int) to string
```

```
print 'Sequence length is ' + str(seq_length)

seq_revcomp = my_seq.reverse_complement()

print 'Reverse complement sequence is ' + seq_revcomp
# end of program
```

If a `Seq` object is created without nominating a sequence alphabet, it is assumed to be `IUPAC.unambiguous_dna`. There are a number of different alphabets available, catering for DNA, RNA and Proteins. These alphabets can be unambiguous (characters are A, C, G, or T for DNA), or can be extended alphabets. To define an alphabet when creating a `Seq` object we also have to import another library, *e.g.* `from Bio.Alphabet import IUPAC`.

```
#!/usr/local/bin/python

from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

#create a Seq object and specify the alphabet
my_seq = Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
...
```

With the above example, we create a `Seq` object with a sequence string and a specific alphabet. If we were to add elements to a sequence which are not in the alphabet an error would be raised, for example, trying to add Z to a sequence specified as `IUPAC.unambiguous_dna` would result in an error.

1 I'm not sure this is true with newer versions... Note that to print out the sequence alphabet object we force it to a string representation using the `str()` function. This can be handy when debugging if you don't know what type of object you're dealing with. You can also use the `repr()` function.

More information on the `Seq` object can be found at `http://biopython.org/wiki/Seq`

The SeqIO object

The SeqIO object provides a standard interface to read and write a number of sequence file formats such as `FASTA`, `GenBank`, `embl`, `clustal` etc. The object provides parsers which create data structures holding all of the information in the sequence file. In a FASTA file this might only be the header information (an id number and maybe a description), and the sequence itself. A GenBank file holds a wealth of additional information with the annotation of the sequence along with the organism name, journal references and comments.

In the example below, we open a FASTA file named Pvivax_contigs.fna using the mode "rU". The "rU" means universal readline. Windows, Macintosh and Unix text files all have different newline characters at the end of a line of text. The universal read mode allows for this, meaning the file can be sourced from different operating systems and still read correctly. The open function creates a handle to the open file, which is used in subsequent manipulations.

The SeqIO object has a method parse which takes two parameters; a handle to the open file and the file type. The parse method creates a list of sequence objects. If there is only one sequence in the input file, the list will only have one Seq object entry. We then go through the list using the for loop printing each Seq object.

```
#!/usr/local/bin/python

from Bio import SeqIO

# open the file
input_handle = open("Pvivax_contigs.fna", "rU")

# parse the file into a list of Seq objects
seqlist = SeqIO.parse(input_handle, "fasta")

# go through the list of Seq objects
for record in seqlist:
    # print each Seq objects id
    print record.id

#end for loop

# close the opened file
input_handle.close()

# end of program
```

To write a Seq object to a file, we use the SeqIO.write() method. Th write method takes parameters of a SeqRecord list, output handle and a format string (”w” for writable). In the example below we open two files; one for reading from and one for writing to. As in the above example, all of the sequences from the FASTA file Pvivax_contigs.fna are parsed into a Seq object list. This time we get the length of each sequence (record.seq) and if less than 900 bases long, add the Seq object to the short_list which is initialised to an empty list. The records in short_list is then written to to the output file (Pvivax_contigs_short.fna).

```
#!/usr/local/bin/python

from Bio import SeqIO

# open the input file
input_handle = open("Pvivax_contigs.fna", "rU")

# create and open the output file
output_handle = open("Pvivax_contigs_short.fna", "w")

# parse the file into a list of Seq objects
seqlist = SeqIO.parse(input_handle, "fasta")

# create an empty list for all the short Seq object records
short_list = []


# Print a line at the top of the output file
# this shows how to use the print statement to print to a file
print >>output_handle, 'This is list of sequences with less than 900 bases'

# print a blank line
print >>output_handle, '\n'


# go through the list of Seq objects
for record in seqlist:
    # get the length of the sequence in the Seq record
    length = len(record.seq);
    # if the length of the sequence is less than 900 bases
    if length < 900:

    # add it to the list
        short_list.append(record)

    # end if
#end for loop

# write the newly created list to the output file
SeqIO.write(short_list,output_handle, "fasta")


# close the opened files
input_handle.close()
output_handle.close()

# end of program
```

Using the `Entrez` package we can access `GenBank` files directly from `NCBI`. For example if we know the sequence id, we can download the `GenBank` record in one step and convert it to a `Seq` object. When using `Entrez` to query the `NCBI` databases, `NCBI` requires that you include your email address with the query. An example is given below.

```
#!/usr/local/bin/python
#

from Bio import SeqIO
from Bio import Entrez
import os
import sys

# This environemtal variable needs to be set for mundula. All web traffic
# normally passes via the proxy address - we need to bypass it.
os.environ['http_proxy']=''

# Need your email address when querying NCBI using Entrez
Entrez.email = 'your_email_address'
```

```
# this example queries the genome database
database = 'nuccore'

search_id = 'NC_003416'

# Query the Entrez database
# We search the genome database using the search term 'NC_003416'
try:
search_handle = Entrez.esearch(db=database,term=search_id,
usehistory="y", retmax=100)

# search_results holds the results of the search
search_results = Entrez.read(search_handle)
search_handle.close()
except:
    print "Problem with network connection."
    exit(1)

# In this case there is only one sequence found, but a general search
# term may return many sequences

gi_list = search_results["IdList"]

# retrieve the actual sequence, "gbwithparts" is the type of record
# to retrieve - this means genbank with all parts. Use "fasta" to retrieve
# a FASTA sequence
fetch_handle = Entrez.efetch(db="nucleotide", id=gi_list[0], rettype="gb",
retmode="text")

# convert the sequence data to a record list, in this case
# there is only one record
record_list = SeqIO.parse(fetch_handle,'genbank');

# if a record list exists
if record_list:
# do for each record
    for seq_record in record_list:
        # print out the sequence id and the length of the sequence
        print 'Sequence id:     ' + seq_record.id
        print 'Sequence length: ' + str(len(seq_record.seq))
```

```
else:
    print 'No records found from parsed file'

fetch_handle.close()

sys.exit(0);


# end of program
```

More examples using the `SeqIO` object can be found at `http://biopython.org/wiki/SeqIO`

The SeqRecord object

As mentioned in the section above, a `GenBank` file can hold a lot more information than just a DNA or protein sequence. To access this extra information we need to understand the type of data structures that make up a `GenBank` record object.

In a `GenBank` file there are *features*, which are listed on the left side. Many *features* have *qualifiers*, which are indented and start with a `'/'`.

This is an example of the beginning of a `GenBank` file.

```
LOCUS       AJ417719               13605 bp    DNA     circular INV 15-APR-2005
DEFINITION  Necator americanus complete mitochondrial genome.
ACCESSION   AJ417719
VERSION     AJ417719.2  GI:31873488
KEYWORDS    ATPase subunit 6; ATPase6 gene; COI gene; COII gene; COIII gene;
            cytb gene; cytochrome b; cytochrome oxidase subunit I; cytochrome
            oxidase subunit II; cytochrome oxidase subunit III; dh1 gene; dh2
            gene; dh3 gene; dh4 gene; dh4L gene; dh5 gene; dh6 gene; l-rRNA
            gene; NADH dehydrogenase subunit 1; NADH dehydrogenase subunit 2;
            NADH dehydrogenase subunit 3; NADH dehydrogenase subunit 4; NADH
            dehydrogenase subunit 4L; NADH dehydrogenase subunit 5; NADH
            dehydrogenase subunit 6; s-rRNA gene; transfer RNA-Ala; transfer
            RNA-Arg; transfer RNA-Asn; transfer RNA-Asp; transfer RNA-Cys;
            transfer RNA-Gln; transfer RNA-Glu; transfer RNA-Gly; transfer
            RNA-His; transfer RNA-Ile; transfer RNA-Leu(CUN); transfer
            RNA-Leu(UUR); transfer RNA-Lys; transfer RNA-Met; transfer RNA-Phe;
            transfer RNA-Pro; transfer RNA-Ser(AGN); transfer RNA-Ser(UCN);
            transfer RNA-Thr; transfer RNA-Trp; transfer RNA-Tyr; transfer
            RNA-Val; tRNA-Ala gene; tRNA-Arg gene; tRNA-Asn gene; tRNA-Asp
            gene; tRNA-Cys gene; tRNA-Gln gene; tRNA-Glu gene; tRNA-Gly gene;
            tRNA-His gene; tRNA-Ile gene; tRNA-Leu(CUN) gene; tRNA-Leu(UUR)
            gene; tRNA-Lys gene; tRNA-Met gene; tRNA-Phe gene; tRNA-Pro gene;
            tRNA-Ser(AGN) gene; tRNA-Ser(UCN) gene; tRNA-Thr gene; tRNA-Trp
            gene; tRNA-Tyr gene; tRNA-Val gene.
SOURCE      mitochondrion Necator americanus
  ORGANISM  Necator americanus
            Eukaryota; Metazoa; Nematoda; Chromadorea; Rhabditida; Strongylida;
            Ancylostomatoidea; Ancylostomatidae; Bunostominae; Necator.
REFERENCE   1
  AUTHORS   Hu,M., Chilton,N.B. and Gasser,R.B.
  TITLE     The mitochondrial genomes of the human hookworms, Ancylostoma
            duodenale and Necator americanus (Nematoda: Secernentea)
  JOURNAL   Int. J. Parasitol. 32 (2), 145-158 (2002)
   PUBMED   11812491
REFERENCE   2
  AUTHORS   Hu,M.
  TITLE     Direct Submission
  JOURNAL   Submitted (26-OCT-2001) Hu M., Department of Veterinary Science,
            The University of Melbourne, 250 Princes Highway, Werribee, 3030,
            Victoria, 3030, AUSTRALIA
  REMARK    revised by [4]
REFERENCE   3  (bases 1 to 13605)
  AUTHORS   Hu,M.
  TITLE     Direct Submission
  JOURNAL   Submitted (13-JUN-2003) Hu M., Department of Veterinary Science,
            The University of Melbourne, 250 Princes Highway, Werribee, 3030,
            Victoria, 3030, AUSTRALIA
COMMENT     On Jun 17, 2003 this sequence version replaced gi:18873695.
FEATURES             Location/Qualifiers
     source          1..13605
                     /organism="Necator americanus"
                     /organelle="mitochondrion"
                     /mol_type="genomic DNA"
                     /isolate="Zhejiang"
                     /host="Human"
                     /db_xref="taxon:51031"
                     /country="China:Zhejiang"
     gene            1..54
                     /gene="tRNA-Pro"
     tRNA            1..54
                     /gene="tRNA-Pro"
                     /product="tRNA-Pro"
     gene            58..112
                     /gene="tRNA-Val"
     tRNA            58..112
                     /gene="tRNA-Val"
                     /product="tRNA-Val"
     gene            113..547
                     /gene="dh6"
     CDS             113..547
                     /gene="dh6"
                     /codon_start=1
                     /transl_table=5
                     /product="NADH dehydrogenase subunit 6"
                     /protein_id="CAD10441.2"
                     /db_xref="GI:31873489"
                     /db_xref="GOA:Q8SK23"
                     /db_xref="UniProtKB/TrEMBL:Q8SK23"
                     /translation="MYKFFLLISLFGALMSYMNMDPMKSSFFLILSMMMCMPMLSFSG
                     YVWFSYFICLLFLSGIFVILVYFSSLSKISISKGYVVLLVFFLTLLVFGMNFGVVVAS
                     VSLNVFYYSIFWWLFFYLLLILLFFMNFTSYFLNFSGALRKL"

--- file continues on with more annotated features ------
```

All of the features in the record above are listed down the left hand side, such as ACCESSION, SOURCE, tRNA, gene etc are all accessible via the SeqRecord object. Note that some features appear only once, such as ACCESSION and ORGANISM, others have multiple entries such as REFERENCE, while the bulk of the file is made up of multiple sequence features such as CDS (coding sequence) and tRNAs.

Qualifiers in the record above include `/gene`, `/codon_start`, and `/translation_start`, among others.

To get an idea what is in the SeqRecord object, you can print out its contents using the repr() function. Here is a fragment of code.

```
# We have opened a file or downloaded a genbank file from NCBI
# and assigned the file to input_handle

# parse the file into a list of Seq objects
seqlist = SeqIO.parse(input_handle, "genbank")

# go through the list of Seq objects
for record in seqlist:
print repr(record)
#end for loop
```

A `GenBank` file can contain multiple `GenBank` sequences, but this is rare for `GenBank` files, although common in `FASTA` files. So that the same `SeqIO` object can handle all sorts of files, a `GenBank` file is assumed to hold multiple sequences. In the example below we create a `SeqRecord` object, which could be printed out in the same way as a downloaded `GenBank` record.

```
SeqRecord(seq=Seq('CAACATGTAGTTTAATAAAAATTTAATAT
TTGGGTTATTAAGATAAGTTGTTGA...AAC' , IUPACAmbiguousDNA()),
 id='AJ417719.2',
 name='AJ417719',
 description='Necator a mericanus complete mitochondrial genome.',
 dbxrefs=[])
```

We can see that the `SeqRecord` is made up of a `Seq` object, an `id`, a `name`, `description` and a `dbxrefs[]` (database cross-reference) list. These fields can be accesseed via their methods. For example:

```
record.id
record.name
record.description
```

What the string representation of the `SeqRecord` doesn't show is the list of *features* it holds. These can be accessed by changing the code fragment to:

```
# go through the list of Seq objects
for record in seqlist:
# the features component is a list with the GenBank annotations
for feature in record.features:
```

```
# print a string representration of each feature
print repr(feature)
#end for loop
```

Output from the above code will read something like

```
Bio.SeqFeature.SeqFeature(Bio.SeqFeature.FeatureLocation(Bio.SeqFeature.ExactPosit
Bio.SeqFeature.ExactPosition(112)), type='gene', strand=1) ...
...
```

Note how each feature in the list has a type, a location, and a strand direction etc. A feature also holds a dictionary object with keys such as 'product'. A code fragment could be:

```
for feature in record.features:
# if the type of feature is a coding segment
if feature.type == 'CDS':
# print out the list from the dictionary with the key 'product'
# note there is only one entry in the list, the gene name
# eg NADH dehydrogenase subunit 6
for x in feature.qualifiers['product']:
print x
```

The journal annotations for the `SeqRecord` are held in a dictionary at the top level. and can be accessed using record.annotations['references'] assuming the `SeqRecord` variable is called 'record'.

More examples using the `SeqRecord` object can be found at `http://biopython.org/wiki/SeqRecord`

R. Hall, March 2013
revised by L. Stern March 2015