Python Programming Exercises 8

Notes: in this set of exercises we will learn how to install external libraries and start to use the BioPython library.

1. While external libraries can be installed manually, the preferred method to install Python modules is using Pip. Depending on your distribution/operating system, pip may be called pip3 for Python 3.

   If you do not know the exact name of what you want to install you can use the `search` subcommand to search by keyword:

   ```
   dredd:~ ajm$ pip3 search biopython
   ```

   Once you know the name of the library that you want, you can use the `install` command to install both it and all dependencies (Windows users should not use "sudo", but instead be logged in as an administrator):

   ```
   dredd:~ ajm$ sudo pip3 install biopython
   ```

   If you do not have administrator rights on your computer, then libraries can be installed just for your own use using the `--user` flag with the `install` subcommand:

   ```
   dredd:~ ajm$ pip3 install --user biopython
   ```

2. If you have not already installed BioPython, do so now. On Linux/MacOS this can be accomplished with pip (see install command above). Under Windows we recommend that you use the BioPython Windows installation program (see `http://wasabiapp.org/pythonsetup/pythonsetup_windows/`).

   Test your installation by typing the following into the Python interpreter:

```
>>> import Bio
>>> Bio.__version__
1.65
```

   If you have any problems, ask the demonstrators.

3. BioPython **Seq** data types take two arguments to create: a sequence and an alphabet:

```
from Bio.Seq import Seq
from Bio.Alphabet import generic_dna

dna = Seq("AAATTTGGGCCC", generic_dna)

print(dna)
```

   The **Seq** type shares many of the same methods as the string type (e.g. `lower, count, find, split, strip`). **Seq** variables are immutable, so you cannot do dna[0] = "A", but you can use them as keys to dictionaries.

4. The **Seq** type has numerous methods to manipulate sequence data (these examples continue from the previous question and assume you have a **Seq** variable called `dna`). Each of these methods return a new **Seq** object (due to immutability).

   Complement (dna → dna, rna → rna):

   ```
   print(dna.complement())
   ```

   Reverse complement (dna → dna, rna → rna):

   ```
   print(dna.reverse_complement())
   ```

   Transcribe (dna → rna):

   ```
   rna = dna.transcribe()
   print(rna)
   ```

   Back transcribe (rna → dna):

   ```
   dna2 = rna.back_transcribe()

   if dna == dna2 :
       print("got back the original sequence:", dna)
   ```

   Translate (dna or rna → protein):

   ```
   prot_from_dna = dna.translate()
   prot_from_rna = rna.translate()

   if prot_from_dna == prot_from_rna :
       print("protein translated from dna and rna is the same:", prot_from_dna)
   ```

   While you can try to perform any of these operation on a variable of type **Seq**, it will result in a **ValueError** if the operation does not make sense, e.g. complementing a protein.

5. The last set of exercises contained details about how to use the `urllib.requests` module from the Python stdlib to access the Ensembl REST API. Here it is again, (this is actually a lie, previously the content-type was set to text/x-fasta, but here we just want the sequence data):

```
from urllib.request import urlopen

url = "http://rest.ensembl.org/sequence/id/ENST00000380152?content-type=text/plain;type=protein"

f = urlopen(url)

for line in f :
    print(line.decode('utf-8'), end="")

f.close()
```

Use this code to write a function that will accept an Ensembl transcript identifier (in the example ENST00000380152, a splicing isoform of the BRCA2 gene) and return a **Seq** object containing the protein data (you will need to use `generic_protein` from the **Bio.Alphabet** module instead of `generic_dna`).

6. We rarely deal with just bare sequences, but need to associate metadata, such as identifiers, with sequence data. In BioPython we use the **SeqRecord** data type from the **Bio.SeqRecord** module for this purpose:

```
from Bio.Seq import Seq
from Bio.Alphabet import generic_protein
from Bio.SeqRecord import SeqRecord

record = SeqRecord(
            Seq("MLLSPSLLLLLLLGAPRGCAEGVAAALTPERL", generic_protein),
            id='ENSP00000283243', description='PLA2R1_fragment')
```

Rewrite your function from the previous question to return a **SeqRecord** object instead and set the `id` value to the Ensembl transcript identifier.

7. The two main fields you will care about in the **SeqRecord** type are `id` (a string identifier) and `seq` a **Seq** object. We can print a **SeqRecord** out in FASTA format as follows:

```
print(">", record.id, sep="")
print(record.seq)
```

Fortunately we don't need to be this ourselves, because **SeqRecords** can be converted to string representations for many different sequence data file formats using the `format(···)` method:

```
print(record.format('fasta'))
```

Use this and your answer from the previous question to write a script that accepts an arbitrary number of Ensembl transcript identifiers as command line arguments (i.e. using `sys.argv`) and prints them out in FASTA format.

8. The **Bio.SeqIO** module contains a function called `write(`$records$, $file$, $format$`)` that is used to write **SeqRecords** to a file. It accepts three arguments: $records$ (a single **SeqRecord** or a **list** of **SeqRecords**), $file$ (a string containing the file name or a file object, i.e. created by `open("filename", "w")`) and $format$ (a string stating the output file format, e.g. "fasta"):

```
from Bio.Seq import Seq
from Bio.Alphabet import generic_protein
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

record = SeqRecord(
            Seq("MLLSPSLLLLLLLGAPRGCAEGVAAALTPERL", generic_protein),
            id='ENSP00000283243',
            description='PLA2R1_fragment')

SeqIO.write(record, "output.fasta", "fasta")
```

Using this code, take your answer from the previous question and instead of printing to the console, write to a file.

9. The **Bio.SeqIO** module contains a function called `parse(`$file, format$`)` to read files containing sequences that takes two arguments: $file$ (a string containing the file name or a file object created by `open("filename", "r")`) and $format$ (a string stating the input file format, e.g. "fasta"):

```
from Bio import SeqIO

for record in SeqIO.parse("input.fasta", "fasta") :
    print(record.id, len(record.seq))
```

Write a script that can read a FASTA file of coding sequences (CDS) and convert it to protein sequences. These protein sequences should be written to a new FASTA file. The names of the input and output FASTA files should be provided as command line arguments. There is an example input file on the course webpage (`http://wasabiapp.org/python_course/zebra_fish_short_cds.fasta`).

10. What happens if you give your script from the previous question a file that does not contain coding sequences? Fix your script so it fails gracefully with an appropriate error message. (For example, cdna sequences `http://wasabiapp.org/python_course/zebra_fish_short_cdna.fasta`)