Introduction to Biopython

Python libraries for computational molecular biology

http://www.biopython.org

Biopython functionality and tools

- Tools to parse bioinformatics files into Python data structures
- Supports the following formats:
  - Blast output
  - Clustalw
  - FASTA
  - PubMed and Medline
  - ExPASy files
  - SCOP
  - SwissProt
  - PDB
- Files in the supported formats can be iterated over record by record or indexed and accessed via a dictionary interface

Biopython functionality and tools

- Tools to deal with on-line bioinformatics destinations (NCBI, ExPASy)
- Interface to common bioinformatics programs (Blast, ClustalW)
- A sequence obj dealing with seqs, seq IDs, seq features
- A Seq Record obj
- Tools for operations on sequences
- Tools for dealing with alignments
- Tools to manage protein structures
- Tools to run applications

Sequence Objects

Sequence Record objects

Sequence Object

- Seq objects vs Python strings:
  - They have different methods
  - The Seq object has the attribute **`alphabet`** (biological meaning of Seq)

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("AGTACACTGGT")
>>> my_seq
Seq('AGTACACTGGT', Alphabet())
>>> my_seq.alphabet
Alphabet()
>>> print my_seq
AGTACACTGGT
```

Write the program to a file and run it

The **`alphabet`** attribute

- Alphabets are defined in the **`Bio.Alphabet`** module
- We will use the IUPAC alphabets (http://www.chem.qmw.ac.uk/iupac)
- **`Bio.Alphabet.IUPAC`** provides definitions for DNA, RNA and proteins + provides extension and customisation of basic definitions:
  - **`IUPACProtein (IUPAC standard AA)`**
  - **`ExtendedIUPACProtein (+ selenocysteine, X, etc)`**
  - **`IUPACUnambiguousDNA (basic GATC letters)`**
  - **`IUPACAmbiguousDNA (+ ambiguity letters)`**
  - **`ExtendedIUPACDNA (+ modified bases)`**
  - **`IUPACUnambiguousRNA`**
  - **`IUPACAmbiguousRNA`**

Transcription

```
from Bio.Seq import Seq
coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
messenger_rna = coding_dna.transcribe()
cDNA = messenger_rna.back_transcribe()
print coding_dna
print messenger_rna
print cDNA
```

All transcribe() does is a switch T --> U

The Seq object also includes a back-transcription method.

Translation

```
from Bio.Seq import Seq

coding_dna =
Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG")
messenger_rna = coding_dna.transcribe()
protein1 = coding_dna.translate()
protein2 = messenger_rna.translate()

print protein1
Print protein2
```

Read the sequence from `ap006852.fasta` and translate it using Biopython

```
from Bio.Seq import Seq

InFile = open("ap006852.fasta")

seq = ''
for line in InFile:
   if line[0] == ">":
      header = line.strip()
   else:
      seq = seq + line.strip().upper()

seq = Seq(seq)
protein = seq.translate()

print header
print protein
```

Sequence Record objects

```
>>> from Bio.SeqRecord import SeqRecord
>>> help(SeqRecord)
```

Attributes:

`id`	- Identifier such as a locus tag (string)
`seq`	- The sequence itself (Seq object)
`name`	- Sequence name, e.g. gene name (string)
`description`	- Additional text (string)
`dbxrefs`	- List of db cross references (list of strings)
`features`	- Any (sub)features defined (list of SeqFeature objects)
`annotations`	- Further information about the whole sequence (dictionary)

Sequence Record objects

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

tmp_seq =
Seq('MKQHKAMIVALIVICITAVVAALVTRKDLCEVHIRTGQTEVAVF')

#define a Seq Record
seq_rec = SeqRecord(tmp_seq)
seq_rec.id = 'YP_025292.1'
seq_rec.description = 'toxic membrane protein'

print seq_rec
```

Sequence Record objects

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

tmp_seq =
Seq('MKQHKAMIVALIVICITAVVAALVTRKDLCEVHIRTGQTEVAVF')

#another way to define a Seq Record
seq_rec =
SeqRecord(Seq('MKQHKAMIVALIVICITAVVAALVTRKDLCEVHIRTGQTE
VAVF'), id = 'YP_025292.1', name='HokC',
description='toxic membrane protein',dbxrefs=[])

print seq_rec
```

The `format()` method

It returns a string containing your record formatted using one of the output file formats supported by `Bio.SeqIO`

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

rec =
SeqRecord(Seq("MGSNKSKPKDASQRRRSLEPSENVHGAGGAFPA
SQTPSKPASADGHRGPSAAFVPPAAEPKLFGGFNSSDTVTSPQRAGAL
AGGVTTFVALYDYESRTETDLSFKKGERLQIVNNTRKVDVREGDWWLA
HSLSTGQTGYIPS", generic_protein), id = "P05480",
description = "SRC_MOUSE Neuronal proto-oncogene
tyrosine-protein kinase Src: MY TEST")

print rec.format("fasta")
```

Parsing Swiss-Prot files and ENSEMBLE records

Bio.SeqIO

- **Sequence I/O**
  - Parsing or Reading Sequences
  - Writing Sequence Files

A simple interface for working with assorted file formats in a uniform way

```
>>>from Bio import SeqIO
>>>help(SeqIO)
```

`SeqIO.parse()`

Reads in sequence data as `SeqRecord` objects

It expects two arguments:

- An object (called **handle**) to read the data. It can be:
  - a file opened for reading
  - the output from a command line program
  - data downloaded from the internet

- A lower case string specifying the sequence format (see http://biopython.org/wiki/SeqIO for a full listing of supported formats).

The object returned by `SeqIO.parse()` is an iterator which returns `SeqRecord` objects

```
>>> from Bio import SeqIO
>>> handle = open("P05480.fasta")
>>> for seq_rec in SeqIO.parse(handle,
"fasta"):
...     print seq_rec.id
...     print repr(seq_rec.seq)
...     print len(seq_rec)
...
sp|P05480|SRC_MOUSE
Seq('MGSNKSKPKDASQRRRSLERGPSA...ENL',
SingleLetterAlphabet())
541
>>> handle.close()
```

```
from Bio import SeqIO

handle = open("1293613.gbk")

record =  SeqIO.parse(handle, "genbank")

for seq_rec in record:
    print seq_rec.id
    print str(seq_rec.seq)
    print len(seq_rec)

handle.close()
```

```
from Bio import SeqIO

handle = open("1293613.gbk")

for seq_rec in SeqIO.parse(handle, "genbank"):
    print seq_rec.id
    print repr(seq_rec.seq)
    print len(seq_rec)

handle.close()
```

Iterating over the records in a multipe sequence file

```
>>> from Bio import SeqIO
>>> handle = open("SwissProt-Human.fasta")
>>> all_rec = SeqIO.parse(handle,"fasta")
>>> for rec in all_rec:
...     print rec.id
...
sp|P31946|1433B_HUMAN
sp|P62258|1433E_HUMAN
sp|Q04917|1433F_HUMAN
sp|P61981|1433G_HUMAN
sp|P31947|1433S_HUMAN
sp|P27348|1433T_HUMAN
sp|P63104|1433Z_HUMAN
>>> handle.close()
```

Parsing sequences from the net

Handles are not always from files

```
import urllib2

handle = urllib2.urlopen("http://
www.uniprot.org/uniprot/B2TYV6.fasta")

F = handle.read()

print F
```

Parsing sequences from the net

Handles are not always from files

Parsing SwissProt sequence from the Internet

```
from Bio import ExPASy
from Bio import SeqIO

handle = ExPASy.get_sprot_raw("P04637")

seq_record = SeqIO.read(handle,"swiss")

handle.close()

print seq_record.id
print seq_record.name
print seq_record.description:w
```

Parsing sequences from the net

Parsing SwissProt sequences from the Internet

```
from Bio import ExPASy
from Bio import SeqIO

AC_list = ['P04637', 'P0CQ42', 'Q13671']
records = []
for ac in AC_list:
    handle = ExPASy.get_sprot_raw(ac)
    record = SeqIO.read(handle, "swiss")
    records.append(record)
out = open('myfile.fasta','w')
for rec in records:
    print rec.id
    fasta = Bio.SeqIO.write(rec, out, "fasta")
out.close()
```

Parsing sequences from the net

Parsing GenBank records from the Internet

```
from Bio import Entrez
Entrez.email = "allegra.via@uniroma1.it"
from Bio import SeqIO

handle = Entrez.efetch(db =
"nucleotide",rettype = "fasta",id = "6273291")
seq_record = SeqIO.read(handle,"fasta")

handle.close()

print seq_record.description
print seq_record.id
```

Indexing really large files

`Bio.SeqIO.index()` returns a dictionary without keeping everything in memory.

It works fine even for million of sequences

The main drawback is less flexibility: it is read-only

```
>>> from Bio import SeqIO
>>> recs_dict = SeqIO.index("ncbi_gene.fasta", "fasta")
>>> len(recs_dict)
34
>>> recs_dict.keys()
['M69013', 'M69012', 'AJ580952', 'J03005', 'J03004', 'L13858',
'L04510', 'M94539', 'M19650', 'A10421', 'AJ002990', 'A06663',
'A06662', 'S62035', 'M57424', 'M90035', 'A06280', 'X95521',
'X95520', 'M28269', 'S50017', 'L13857', 'AJ345013', 'M31328',
'AB038040', 'AB020593', 'M17219', 'DQ854814', 'M27543', 'X62025',
'M90043', 'L22075', 'X56614', 'M90027']
>>> print recs_dict['M57424']
ID: M57424
Name: M57424
Description: M57424 Human adenine nucleotide translocator-2 (ANT-2)
gene, complete cds. : Location:1..1000
Number of features: 0
Seq('gagctctggaatagaatacagtagaggcatcatgctcaaagagagtagcagatg...agc',
SingleLetterAlphabet())
```

Writing sequence files

`Bio.SeqIO.write()`

This function takes three arguments:

1. SeqRecord objects
2. a handle to write to
3. a sequence format

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

Rec1 = SeqRecord(Seq("ACCA"), id="1", description="")
Rec2 = SeqRecord(Seq("CDRFAA"), id="2", description="")
Rec3 = SeqRecord(Seq("GRKLM"), id="3", description="")

My_records = [Rec1, Rec2, Rec3]

handle = open("MySeqs.fas","w")

SeqIO.write(My_records, handle, "fasta")

handle.close()
```

Converting between sequence file formats

You can do file conversion by combining `Bio.SeqIO.parse()` and `Bio.SeqIO.write()`

```
from Bio import SeqIO

In_handle = open ("1293613.gbk", "r")
Out_handle = open("1293613.fasta", "w")

records = SeqIO.parse(In_handle, "genbank")
SeqIO.write(records, Out_handle, "fasta")

In_handle.close()
Out_handle.close()
```

Search the Entrez nucleotide database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez
from Bio import SeqIO

Entrez.email = "allegra.via@uniroma1.it”
# search entries by keywords
handle = Entrez.esearch(db="nucleotide", term="Homo
sapiens AND mRNA AND MapK")
records = Entrez.read(handle)
print records['Count']

FewRecords = records['IdList'][0:3]
# retrieve and parse entries from the database
handle = Entrez.efetch(db = "nucleotide",retmode="xml",id
= FewRecords)

records = Entrez.parse(handle)
for record in records:
    print record.keys()
    print record['GBSeq_primary-accession']
    print record['GBSeq_sequence']
    print len(records)
```

Search the Entrez Pubmed database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez
from Bio import Medline

keyword = "PyCogent”
# search publications in PubMed
Entrez.email = ”allegra.via@uniroma1.it"
handle = Entrez.esearch(db="pubmed", term=keyword)
record = Entrez.read(handle)
pmids = record['IdList']
print pmids

# retrieve Medline entries from PubMed
handle = Entrez.efetch(db="pubmed", id=pmids,
rettype="medline", retmode="text")
medline_records = Medline.parse(handle)
records = list(medline_records)

n = 1
for record in records:
    if keyword in record["TI"]: print n, ')', record["TI"]
```

Search the Entrez protein database by keyword(s)

You can do it using a combination of `Entrez.esearch()` and `Entrez.efetch()`

```
from Bio import Entrez

Entrez.email = "allegra.via@gmail.com"

# search entries by keywords
handle = Entrez.esearch(db = "protein", term = "Human AND
cancer AND p21 AND Map kinase", rettype = "fasta")
records = Entrez.read(handle)

FewRecords = records['IdList'][0:3]
print FewRecords
ID_list = ",".join(FewRecords)

# retrieve and parse entries from the database
handle = Entrez.efetch(db = "protein", id = ID_list,
rettype="fasta", retmode="xml")

records= Entrez.read(handle)
rec = list(records)
print rec[0].keys()
print rec[0]['TSeq_defline']
```

Running BLAST

Running Blast

Locally

From the UNIX shell

Blast command line

From a script

Using `os.system()`

Using `Biopython`

Over the Internet

Using `Bio.Blast.NCBIWWW`

Using Web Programming

Using a Web Browser

Running BLAST locally

```
blastProgram –query Query –out OutFile -db Database
```

Using the UNIX shell command line

```
$blastp -query P05480.fasta -out blast_output -db nr.00
```

Running BLAST locally

From a script

Using os.system()

```
import os
S = "blastp -query P05480.fasta -out \
blast_output -db nr.00"
os.system(S)
```

Running BLAST locally

From a script **Using Biopython**

```
import os

from Bio.Blast.Applications import NcbiblastpCommandline
cline = NcbiblastpCommandline(query = "P05480.fasta",
                              db = "nr.00",
                              out = "Blast.out",
                              evalue = 0.001)
print cline
os.system(str(cline))
```

Running BLAST locally

From a script **Using Biopython**

```
import os

# outfmt = 5 generates the output in XML format:
cline = NcbiblastpCommandline(query = "P05480.fasta",
                              db = "nr.00",
                              out = "Blast.xml",
                              evalue = 0.001,
                              outfmt = 5)
print cline
os.system(str(cline))
```

Running BLAST locally

From a script **Using Biopython**

```
# Run BLAST+ with nucleotide sequences.In order to run the
# following command, you need to download or format
# a nucleotide database.
# my_gene.fasta must be in the directory where you
# run the script.

from Bio.Blast.Applications import NcbiblastnCommandline
cline = NcbiblastnCommandline(query = "my_gene.fasta",
                              db = "nt", strand = "plus",
                              evalue = 0.001,
                              out = ”Blast.xml",
                              outfmt = 5)
print cline

os.system(str(cline))
```

Running BLAST over the Internet

and saving the output to a file

Bio.Blast.NCBIWWW

```
from Bio.Blast import NCBIWWW

result_handle = NCBIWWW.qblast("blastn","nr","8332116")

save_file = open("qblast_blastn.out", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

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)

Save the output locally in xml format

```
from Bio.Blast import NCBIWWW

result_handle =
NCBIWWW.qblast("blastn","nr","8332116",
format_type="XML")

save_file = open("qblast_blastn.xml", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

Running BLAST and saving the output to a XML file

```
from Bio.Blast import NCBIWWW

result_handle = NCBIWWW.qblast("blastn","nr","8332116",
format_type = "XML")

save_file = open("qblast_blastn.out", "w")
save_file.write(result_handle.read())
save_file.close()

result_handle.close()
```

Parsing the BLAST output

You can get BLAST output in XML in various ways: for the parser it does not matter how the output was generated as long as it is in XML format

- Use Biopython to run BLAST over the internet
- Use Biopython to run BLAST locally
- Do the BLAST search yourself on the NCBI site through your web browser, and then save the results (choose XML format for the output)
- Run BLAST locally without using Biopython, and save the output in a file (choose XML format for the output file)

Parsing the BLAST output

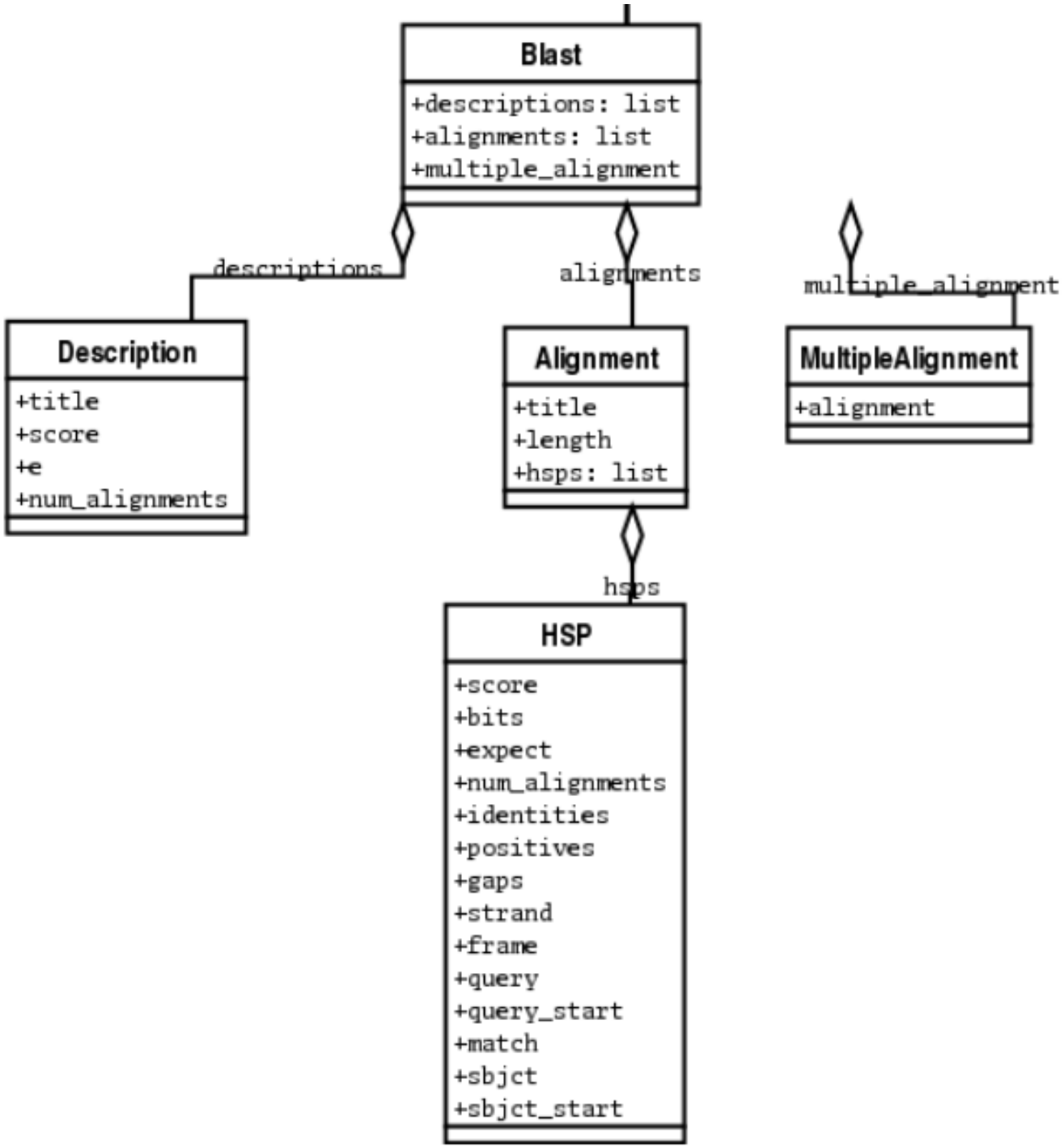
Bio.Blast.NCBIXML

```
from Bio.Blast import NCBIXML

result_handle = open("qblast_blastn.out")

#If you expect a single BLAST result
blast_record = NCBIXML.read(result_handle)

#If you have lots of results
blast_records = NCBIXML.parse(result_handle)
```


Blast
+descriptions: list
+alignments: list
+multiple_alignment
descriptions
alignments
multiple_alignment
Description
+title
+score
+e
+num_alignments
Alignment
+title
+length
+hsps: list
MultipleAlignment
+alignment
hsps
HSP
+score
+bits
+expect
+num_alignments
+identities
+positives
+gaps
+strand
+frame
+query
+query_start
+match
+sbjct
+sbjct_start

```
>>> from Bio.Blast import NCBIXML
>>> result_handle = open("qblast_blastn.out")
>>> blast_record = NCBIXML.read(result_handle)
>>> for alignment in blast_record.alignments:
...     for hsp in alignment.hsps:
...         print hsp.score
...
370.0
354.0
354.0
336.0
292.0
292.0
278.0
276.0
```

```
>>> from Bio.Blast import NCBIXML
>>> blast_results = open("qblast_blastn.out")
>>> blast_records = NCBIXML.parse(blast_results)
>>> for blast_rec in blast_records:
...     for alignment in blast_rec.alignments:
...         for hsp in alignment.hsps:
...             print hps.score, hps.expect
...
370.0 6.67609e-88
354.0 1.47051e-83
354.0 1.47051e-83
336.0 1.13052e-78
292.0 9.91696e-67
292.0 9.91696e-67
278.0 6.25828e-63
276.0 2.18435e-62
264.0 3.94941e-59
```

The Biopython module name is **`Bio`**

It must be downloaded and installed
(**`http://biopython.org/wiki/Download`**)

You need to install **`numpy`** first

```
>>>import Bio
```