Biopython

Karin Lagesen

karin.lagesen@bio.uio.no

Object oriented programming

- Biopython is object-oriented
- Some knowledge helps understand how biopython works
- OOP is a way of organizing data and methods that work on them in a coherent package
- OOP helps structure and organize the code

Classes and objects

- A class:
  - is a user defined type
  - is a mold for creating objects
  - specifies how an object can contain and process data
  - represents an abstraction or a template for how an object of that class will behave
- An object is an *instance* of a class
- All objects have a type – shows which class they were made from

Attributes and methods

- Classes specify two things:
  - attributes – data holders
  - methods – functions for this class
- Attributes are variables that will contain the data that each object will have
- Methods are functions that an object of that class will be able to perform

Class and object example

- Class: MySeq
- MySeq has:
  - attribute length
  - method translate
- An object of the class MySeq is created like this:
  - `myseq = MySeq("ATGGCCG")`
- Get sequence length:
  - `myseq.length`
- Get translation:
  - `myseq.translate()`

Summary

- An object has to be *instantiated*, i.e. created, to exist
- Every object has a certain type, i.e. is of a certain class
- The class decides which attributes and methods an object has
- Attributes and methods are accessed using . after the object variable name

Biopython

- Package that assists with processing biological data
- Consists of several modules – some with common operations, some more specialized
- Website: biopython.org

Biopython must be installed

- Not part of python per se, has to be installed
- Several versions of python available, not all compatible with biopython
- On freebee, several python versions available:

```
[karinlag@freebee]~/teaching% which python
/usr/bin/python
[karinlag@freebee]~/teaching%
```

- `module load python2` makes different version of python available:

```
[karinlag@freebee]~/teaching% module load python2
[karinlag@freebee]~/teaching% which python
/cluster/software/VERSIONS/python2-2.7.3/bin/python
```

Working with sequences

- Biopython has many ways of working with sequence data
- Covered today:
  - Alphabet
  - Seq
  - SeqRecord
  - SeqIO
- Other useful classes for working with alignments, blast searches and results etc are also available, not covered today

Class Alphabet

- Every sequence needs an alphabet
- CCTTGGCC – DNA or protein?
- Biopython contains several alphabets
  - DNA
  - RNA
  - Protein
  - the three above with IUPAC codes
  - ...and others
- Can all be found in Bio.Alphabet package

Alphabet example

- Go to freebee
- Do module load python2 (necessary to find biopython modules) – start python

NOTE: have to import Alphabets to use them

```
>>> import Bio.Alphabet
>>> Bio.Alphabet.ThreeLetterProtein.letters
['Ala', 'Asx', 'Cys', 'Asp', 'Glu', 'Phe', 'Gly', 'His', 'Ile',
'Lys', 'Leu', 'Met', 'Asn', 'Pro', 'Gln', 'Arg', 'Ser', 'Thr',
'Sec', 'Val', 'Trp', 'Xaa', 'Tyr', 'Glx']
>>> from Bio.Alphabet import IUPAC
>>> IUPAC.IUPACProtein.letters
'ACDEFGHIKLMNPQRSTVWY'
>>> IUPAC.unambiguous_dna.letters
'GATC'
>>>
```

Packages, modules and classes

- What happens here?

```
>>> from Bio.Alphabet import IUPAC
>>> IUPAC.IUPACProtein.letters
```

- Bio and Alphabet are packages
  - packages contain other packages and modules
- IUPAC is a module
  - a module is a file with python code
- IUPAC module contains class IUPACProtein and other classes specifying alphabets
- IUPACProtein has attribute letters

Seq

- Represents one sequence with its alphabet
- Methods:
  - `translate()`
  - `transcribe()`
  - `complement()`
  - `reverse_complement()`
  - `...`

Using Seq

Import classes
Create object

```
>>> from Bio.Seq import Seq
>>> import Bio.Alphabet
>>> seq = Seq("CCGGGTT", Bio.Alphabet.IUPAC.unambiguous_dna)
>>> seq
Seq('CCGGGTT', IUPACUnambiguousDNA())

>>> seq.transcribe()
Seq('CCGGGUU', IUPACUnambiguousRNA())
>>> seq.translate()
Seq('PG', IUPACProtein())
```

Use methods

```
>>> seq = Seq("CCGGGUU", Bio.Alphabet.IUPAC.unambiguous_rna)
```

New object, different alphabet

```
>>> seq.transcribe()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/site/VERSIONS/python-2.6.2/lib/python2.6/site-packages/Bio/Seq.py",
 line 830, in transcribe
    raise ValueError("RNA cannot be transcribed!")
ValueError: RNA cannot be transcribed!
>>> seq.translate()
Seq('PG', IUPACProtein())
>>>
```

Alphabet dictates which methods make sense

Seq as a string

- Most string methods work on Seqs
- If string is needed, do str(seq)

```
>>> seq = Seq('CCGGGTTAACGTA',Bio.Alphabet.IUPAC.unambiguous_dna)
>>> seq[:5]
Seq('CCGGG', IUPACUnambiguousDNA())
>>> len(seq)
13
>>> seq.lower()
Seq('ccgggttaacgta', DNAAlphabet())
>>> print seq
CCGGGTTAACGTA
>>> mystring = str(seq)
>>> print mystring
CCGGGTTAACGTA
>>> type(seq)
<class 'Bio.Seq.Seq'>
>>> type(mystring)
<type 'str'>
>>>
```

How to check what class or type an object is from

SeqRecord

- Seq contains the sequence and alphabet
- But sequences often come with a lot more
- SeqRecord = Seq + metadata
- Main attributes:
  - id – name or identifier
  - seq – seq object containing the sequence

Existing sequence

```
>>> seq
Seq('CCGGGTTAACGTA', IUPACUnambiguousDNA())
>>> from Bio.SeqRecord import SeqRecord
>>> seqRecord = SeqRecord(seq, id='001')
>>> seqRecord
SeqRecord(seq=Seq('CCGGGTTAACGTA', IUPACUnambiguousDNA()),
id='001', name='<unknown name>', description='<unknown description>',
dbxrefs=[])
>>>
```

SeqRecord is a class found inside the Bio.SeqRecord module

SeqRecord attributes

- From the biopython webpages:

Main attributes:

id - Identifier such as a locus tag (string)
seq - The sequence itself (Seq object or similar)

Additional attributes:

name - Sequence name, e.g. gene name (string)
description - Additional text (string)
dbxrefs - List of database cross references (list of strings)
features - Any (sub)features defined (list of SeqFeature objects)
annotations - Further information about the whole sequence (dictionary)
Most entries are strings, or lists of strings.
letter_annotations - Per letter/symbol annotation (restricted dictionary). This holds Python sequences (lists, strings or tuples) whose length matches that of the sequence. A typical use would be to hold a list of integers representing sequencing quality scores, or a string representing the secondary structure.

SeqRecords in practice...

```
>>> from Bio.SeqRecord import SeqRecord
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import DNAAlphabet
```

Import necessary classes

```
>>> seqRecord = SeqRecord(Seq('GCAGCCTCAAACCCCAGCTG',
… DNAAlphabet), id = 'NM_005368.2', name = 'NM_005368',
… description = 'Myoglobin var 1',
… dbxrefs = ['GeneID:4151', 'HGNC:6915'])
>>>
```

Create object

```
>>> seqRecord
SeqRecord(seq=Seq('GCAGCCTCAAACCCCAGCTG',
 <class 'Bio.Alphabet.DNAAlphabet'>), id='NM_005368.2',
name='NM_005368', description='Myoglobin var 1',
dbxrefs=['GeneID:4151', 'HGNC:6915'])
>>>
```

Print object

SeqIO

- How to get sequences in and out of files
- Retrieves sequences as SeqRecords, can write SeqRecords to files
- Reading:
  - `SeqIO.parse(filehandle, format)`
  - returns a generator that gives SeqRecords
- Writing:
  - `SeqIO.write(SeqRecord(s), filehandle, format)`

NOTE: examples in this section from http://biopython.org/wiki/SeqIO

SeqIO formats

- List: http://biopython.org/wiki/SeqIO
- Some examples:
  - fasta
  - genbank
  - several fastq-formats
  - ace
- Note: a format might be readable but not writable depending on biopython version

Reading a file

```
from Bio import SeqIO
fh = open("example.fasta", "r")
for record in SeqIO.parse(fh,"fasta") :
    print record.id
fh.close()
```

- SeqIO.parse returns a SeqRecord iterator
- An iterator will give you the next element the next time it is called
- Useful because if a file contains many records, we avoid putting all into memory all at once

Parsing fasta files

- Copy fasta file containing 3 sequences

```
cp ~karinlag/teaching/mb.fsa .
```

- In python interactive shell:

```
>>> from Bio import SeqIO
>>> fh = open("mb.fsa", "r")
>>> for record in SeqIO.parse(fh, "fasta"):
...     print record.id
...     print record.seq[:10]
...
NM_005368.2
GCAGCCTCAA
XM_001081975.2
CCTCTCCCCA
NM_001164047.1
TAGCTGCCCA
>>>
```

Import modules, open the file

Per element in file:

Print the identifier id

Print the first ten sequence letters

convert.py

- Goal: convert from genbank to fasta
- `cp ~karinlag/teaching/mb.gbk .`
- Create script file:
  - Import both sys and SeqIO
  - Take in file name as sys.argv[1]
  - For each record in file (remember:genbank!)
    - Print record
  - Close file
  - Save as convert.py
- Run script with mb.`gbk`

convert.py

```
[karinlag@freebee]~/teaching% cat convert.py
import sys
from Bio import SeqIO

# Open input file
fh = open(sys.argv[1], "r")

for record in SeqIO.parse(fh, "genbank"):
    # print the entire record
    print record
fh.close()
[karinlag@freebee]~/teaching%
```

Need to import SeqIO, otherwise methods not available!

One genbank entry is one record

convert.py

- Modification 1:
  - Print
    - The id
    - The description
    - The sequence

convert.py

```
[karinlag@freebee]~/teaching% cat convert1.py
from Bio import SeqIO
import sys

# Open the input file
fh = open(sys.argv[1], "r")

for record in SeqIO.parse(fh, "genbank"):
    # Print only id, description and sequence
    print record.id
    print record.description
    print record.seq
fh.close()
[karinlag@freebee]~/teaching%
```

Select only the attributes that you actually want!

Modifications

- Figure out how to:
  - print the description of each genbank entry
  - which annotations each entry has
  - print the taxonomy for each entry
- Description:
  - seqRecord.description
- Annotations:
  - seqRecord.annotations.keys()
- Taxonomy:
  - seqRecord.annotations['taxonomy']

Writing files

```
from Bio import SeqIO
sequences = ... # add code here
output_handle = open("example.fasta", "w")
SeqIO.write(sequences, output_handle, "fasta")
output_handle.close()
```

- Note: sequences is here a list containing several SeqRecords
- Can write any iterable containing SeqRecords to a file
- Can also write a single sequence

convert.py

- Modification 2:
  - Get output file name as sys.argv[2]
  - Open outfile
  - Per record,
    - Write it to file in fasta format
  - Close input file
  - Close output file
- Bonus question: can you think of how you would add the organism name to the id?

convert.py

```
[karinlag@freebee]~/teaching% cat convert2.py
from Bio import SeqIO
import sys

# Open the input file
fh = open(sys.argv[1], "r")
# Open the output file
fo = open(sys.argv[2], "w")

for record in SeqIO.parse(fh, "genbank"):
    # Use SeqIO to write properly
    # formatted record
    SeqIO.write(record, fo, "fasta")
fh.close()
fo.close()
# ...and closing files
[karinlag@freebee]~/teaching%
```

Open both input and output file

Write out record

Bonus question

- How to add organism name:
  - get the taxonomy list from the annotation dictionary
  - get the last element of the list with slicing off -1, the last element
  - concat the record.id with the orgname

```
orgname = record.annotations['taxonomy'][-1]
print record.id + "_" + orgname
```

Tips and hints

- Always comment your code – easier to understand later
- Never write lots of code without testing while writing – makes for less code to debug
- Always test on input where you know what the results should be
- If it went to easy, too well or too fast: it is probably wrong!

Learning more

- Recommended book:
  - Sebastian Bassi:
    Python for Bioinformatics
- www.python.org
  - has lots of documentations and beginner tutorials
- Google