Biopython

MBV-INFx410

Fall 2015

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

Fake class and object example

- Class: MyCup
- MyCup has:
  - attribute contents
  - method heat
- An object of the class MyCup is created like this:
  - mycup = MyCup(“Water”)
    - Here: the attribute contents is assigned the value “Water”
- Find out what the content is (access attribute):
  - mycup.contents– will report the contents
- Heat contents (use method):
  - mycup.heat() – will heat contents, in this case “Water”

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 of that class 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

Working with sequences

- Biopython has many ways of working with sequence data
- Focus on:
  - Alphabet package
  - Seq class
  - SeqRecord class
  - SeqIO package
- 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

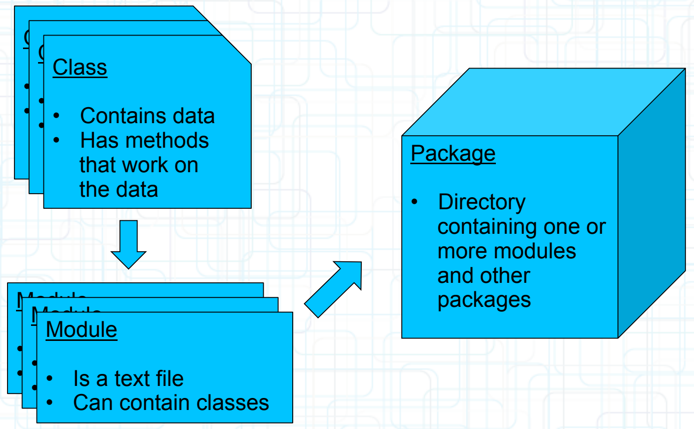
```
>>> import Bio.Alphabet
>>> Bio.Alphabet.ThreeLetterProtein.letters
['Ala', 'Asx', 'Cys', 'Asp', 'Glu', 'Phe', 'Gly', 'His', 'Ile',
'Lys', 'Leu', 'Met', 'Asn', 'Pro', 'Gln', 'Arg', 'Ser', 'Thr',
'Sec', 'Val', 'Trp', 'Xaa', 'Tyr', 'Glx']
>>> from Bio.Alphabet import IUPAC
>>> IUPAC.IUPACProtein.letters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

NOTE: have to import Alphabets to use them

Can now print all of the common three letter abbreviations

Can work with both ambigous and unabigous sequences

Packages, modules and classes


Class
Contains data
Has methods that work on the data
Module
Is a text file
Can contain classes
Package
Directory containing one or more modules and other packages

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
  - a module can contain 0 to many classes
- IUPAC module contains class IUPACProtein and other classes specifying alphabets
- IUPACProtein class specifies objects that have the attribute letters

Class Seq

- Represents one sequence with its alphabet

  newseq = Seq(string, alphabet)
- Has attributes that keeps the string and the alphabet
- Methods:
  - newseq.translate()
  - newseq.transcribe()
  - newseq.complement()
  - newseq.reverse_complement()
  - ...

Using Seq

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

Import classes
Create object

Use methods

New object, different alphabet

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'>
>>>
```

Slicing

Length of Seq

Lower case

Printing

Convert as needed

Can 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

SeqRecord example

```
>>> 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=[])
>>>
```

Existing seq object

SeqRecord is a class found inside the Bio.SeqRecord module

Using existing seq object to create a SeqRecord with an identifier

Several other attributes that for now don't have any value

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

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

>>> 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'])
>>>
```

Import necessary classes

Create object,assign values to attributes

Print object

The SeqIO package

- 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: both input and output files can contain several sequences

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

- 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

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

Parsing fasta files

- 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
- Create script file:
  - Import SeqIO
  - Open the file
  - For each entry (record) in file
    - print record
  - Close file

convert.py – read sequences

```
import sys
from Bio import SeqIO

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

for record in SeqIO.parse(fh, "genbank"):
    # Print the entire record
    print record
fh.close()
```

Need to import SeqIO, otherwise methods not available!

Opening file, as before

Per genbank record in file

Each genbank entry is printed

Closing file, as before

Try it out with mb.gbk!

convert.py results 1

```
ID: NM_005368.2
Name: NM_005368
Description: Homo sapiens myoglobin (MB), transcript variant 1, mRNA.
Number of features: 11
/comment=REVIEWED REFSEQ: This record has been curated by NCBI staff. The
reference sequence was derived from BU585249.1, BQ956082.1 and
BC014547.1.
On Mar 4, 2004 this sequence version replaced gi:4885476.
Summary: This gene encodes a member of the globin superfamily and
is expressed in skeletal and cardiac muscles. The encoded protein
is a haemoprotein contributing to intracellular oxygen storage and
transcellular facilitated diffusion of oxygen. At least three
alternatively spliced transcript variants encoding the same protein
have been reported. [provided by RefSeq, Jul 2008].
…..
```

convert.py take 2

- Modification: get only sequence and id information
- Fasta description line consists of id and description
- Print
  - The id
  - The description
  - The sequence

convert.py take 2

```
import sys
from Bio import SeqIO

# 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()
```

Select only the attributes that you actually want! Available since the parser has read it in the file

convert.py results 2

```
NM_005368.2
Homo sapiens myoglobin (MB), transcript variant 1, mRNA.
GCAGCCTCAAACCCCAGCTGTTGGGGCCAGGACACCC
AGTGAGCCCATACTTGCTCTTTTTGTCTTCTTCAGACTG
CGCCATGGGGCTCAGCGACGGGGAATGGCAGTTGGTG
CTGAACGTCTGGGGGAAGGTGGAGGCTGACATCCCAG
GCCATGGGCAGGAAGTCCTCATCAGGCTCTTTAAGGGT
CACCCAGAGACTCTGGAGAAGTTTGACAAGTTCAAGCA
.....
XM_001081975.2
PREDICTED: Macaca mulatta myoglobin, transcript variant 1 (MB), mRNA.
CCTCTCCCCACCCCCAGCCCTGGCCGCTTGGCTGGAAG
CTCTGCGAGGACAGCTGGGGAGAAGGGGAGCTGTGAC
TGCGCCATGGGGCTCAGCGACGGGGAATGGCAGTTGG
.....
```

Other available info

- Description:
  - seqRecord.description
- Annotations:
  - seqRecord.annotations.keys()
- Taxonomy:
  - seqRecord.annotations['taxonomy']

Writing files

- Note: sequences is here a list containing several SeqRecords
- Can write any iterable containing SeqRecords to a file
- By specifying format, we specify what information to print out – no need to specify what we want to write out

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

convert.py take 3

- Modification: write output to file
  - Open outfile
  - Per record,
    - Write it to file in fasta format
  - Close input file
  - Close output file

convert.py take 3

```
import sys
from Bio import SeqIO

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

for record in SeqIO.parse(fh, "genbank"):
    # Use SeqIO to write properly
    # formatted record
    SeqIO.write(record, fo, "fasta")

# ...and closing files
fh.close()
fo.close()
```

Open both input and output file

Per entry in the gbk file

Write out record in fasta format

Close both input and output file

name_w_organism.py

- Starting point: genbank file
- Goal: print fasta file with description lines that begin with organism name
- Process:
  - Read in fasta sequences as SeqRecords
  - Open output handle
  - Per fasta sequence:
    - figure out the organism name
    - change description line
    - print to output file
  - Close files

name_w_organism.py

```
import sys
from Bio import SeqIO

# 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"):
    # using SeqRecord annotation dictionary
    # to get the correct name
    organism = record.annotations['organism']
    species_name = organism.split()[1]
    # adding it onto the record id
    record.id = species_name + "_" + record.id
    SeqIO.write(record, fo, "fasta")
fh.close()
fo.close()
```

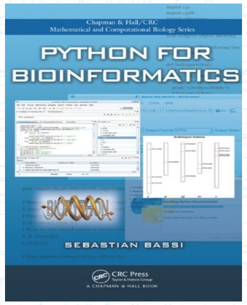
Tips and hints

- 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


Chapman & Hall/CRC
Mathematical and Computational Biology Series
PYTHON FOR BIOINFORMATICS
SEBASTIAN BASSI
CRC Press
A CHAPMAN & HALL BOOK