Wed, Nov 18

Lecture 10B:

Biopython

Practical Bioinformatics (Biol 4220)
Instructor: Michael Landis
Email: michael.landis@wustl.edu

Lecture 10B outline

1. Overview of Biopython
2. Sequence objects
3. Alignment objects
4. Other features
5. Lab 10B overview

Biopython is an open source Python library that provides a wide range of bioinformatics utilities.

Features include:

- reading and writing sequence data
- aligning sequences
- accessing and manipulating alignments
- parsing GenBank records
- handling BLAST calls
- accessing databases
- navigating and manipulating phylogenetic trees
- displaying genome architecture
- and more

Biopython Tutorial and Cookbook

Many code snippets in this lecture are tutorial excerpts:
http://biopython.org/DIST/docs/tutorial/Tutorial.html

2.2 Working with sequences

Disputably (of course!), the central object in bioinformatics is the sequence. Thus, we'll start with a quick introduction to the Biopython mechanisms for dealing with sequences, the `Seq` object, which we'll discuss in more detail in Chapter 3.

Most of the time when we think about sequences we have in my mind a string of letters like '`AGTACACTGGT`'. You can create such `Seq` object with this sequence as follows - the ">>>" represents the Python prompt followed by what you would type in:

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

screenshot of cookbook entry

Parsing a fasta file with native Python

```
> Species_A
AGTCCTAGCATGTTC
> Species_B
AGTCATAGCATGTTC
> Species_C
AGTCCTAGGATGTTC
> Species_D
AGTTCTGGCATGTTC
```

example.fasta

```
f = open('example.fasta', 'r')            # open file
d = {}                                    # create empty dict
lines = f.readlines()                     # get list of lines
entries = ''.join(lines).split('>')       # split into entries
entries = entries[1:]                     # drop 1st entry, ''
for s in entries:                         # loop over entries
    z = s.split('\n')                     # separate entry by \n
    species = z[0].strip()                # get species name
    seq = ''.join(z[1:])                  # get sequence info
    d[species] = [ nt for nt in seq ]     # store as name:seq

f.close()                                 # close file
print(d)                                  # print dict
```

read_fasta_native.py

```
>>> print(d)
{'Species_A': ['A', 'G', 'T', 'C', 'C', 'T', 'A', 'G', 'C', 'A', 'T', 'G', 'T', 'T', 'C'],
 'Species_B': ['A', 'G', 'T', 'C', 'A', 'T', 'A', 'G', 'C', 'A', 'T', 'G', 'T', 'T', 'C'],
 'Species_C': ['A', 'G', 'T', 'C', 'C', 'T', 'A', 'G', 'G', 'A', 'T', 'G', 'T', 'T', 'C'],
 'Species_D': ['A', 'G', 'T', 'T', 'C', 'T', 'G', 'G', 'C', 'A', 'T', 'G', 'T', 'T', 'C']}
```

dictionary-of-lists

Parsing a fasta file with Biopython

```
> Species_A
AGTCCTAGCATGTTC
> Species_B
AGTCATAGCATGTTC
> Species_C
AGTCCTAGGATGTTC
> Species_D
AGTTCTGGCATGTTC
```

example.fasta

```
from Bio import AlignIO
d = AlignIO.read('example.fasta', 'fasta')
print(d)
```

read_fasta_biopython.py

```
>>> print(d)
SingleLetterAlphabet() alignment with 4 rows and 15 columns
AGTCCTAGCATGTTC Species_A
AGTCATAGCATGTTC Species_B
AGTCCTAGGATGTTC Species_C
AGTTCTGGCATGTTC Species_D
```

Biopython alignment data structure

Biopython ***sequence objects*** behave like strings, while having expanded features for bioinformatics tasks

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq('GATTACA')          # create a sequence object
>>> my_seq                           # return value
Seq('GATTACA')
>>> my_seq[0:2]                      # extact subsequence
Seq('GA')
>>> len(my_seq)                      # get length
7
>>> Seq('r u rly a string?')         # no alphabet imposed
Seq('r u rly a string?')
```

Sequence objects are easily cast as strings; sequences support *upper, find, count, etc.*

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq('GATTACA')    # make sequence
>>> str(my_seq)                # typecast as string
'GATTACA'
>>> my_seq.lower()             # change case (.upper supported)
Seq('gattaca')
>>> my_seq.find('TAC')         # find start index for subseq
3
>>> my_seq.count('TAC')        # count occurrences
1
>>> Seq('AAAA').count('AA')    # non-overlapping count
2
>>> Seq('AAAA').count_overlap('AA') # overlapping count
3
>>> (my_seq.count('C')+my_seq.count('G')) / len(my_seq) * 100
28.57142857142857              # compute GC content
```

Sequence objects are indexed and concatenated using the same syntax as is used for strings

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq('GATCGATGGGCCTATATAGGA')
>>> my_seq[4:12]                # extract subsequence
Seq('GATGGGCC')
>>> my_seq[0::3]                # first codon position
Seq('GCTGTAG')
>>> my_seq[1::3]                # second codon position
Seq('AGGCATG')
>>> my_seq[2::3]                # third codon position
Seq('TAGCTAA')
>>> my_seq[:7]+Seq('NNNNNNN')+ my_seq[14:]
Seq('GATCGATNNNNNNNTATAGGA')
```

Sequence objects provide special methods
for computing properties of DNA sequences,
e.g. complement, reverse-complement, and GC %

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC')
>>> my_seq
Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC')
>>> my_seq.complement()          # return complement
Seq('CTAGCTACCCGGATATATCCTAGCTTTTAGCG')
>>> my_seq.reverse_complement()  # return reverse-complement
Seq('GCGATTTTCGATCCTATATAGGCCCATCGATC')
>>> from Bio.SeqUtils import GC
>>> GC(my_seq)                   # what is the GC content?
46.875
```

Biopython sequence objects support methods to *transcribe* DNA into mRNA and to *translate* mRNA (or DNA) into AA

```
>>> from Bio.Seq import Seq
>>> cDNA = Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG')
>>> cDNA                          # coding DNA sequence
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG')
>>> mRNA = cDNA.transcribe()      # transcribe DNA into mRNA
>>> mRNA
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', RNAAlphabet())
>>> mRNA.translate()              # translate mRNA into AA
Seq('MAIVMGR*KGAR*', HasStopCodon(ExtendedIUPACProtein(), '*'))
>>> cDNA.translate()              # translate DNA into AA
Seq('MAIVMGR*KGAR*', HasStopCodon(ExtendedIUPACProtein(), '*'))
>>> cDNA.translate(to_stop=True)  # terminate AA at stop codon
Seq('MAIVMGR', ExtendedIUPACProtein())
>>> # consider an alternate table (default: 'Standard')
>>> cDNA.translate(table='Vertebrate Mitochondrial')
Seq('MAIVMGRWKGAR*', HasStopCodon(ExtendedIUPACProtein(), '*')
>>> cDNA.translate(table='''Vertebrate Mitochondrial''', to_stop=True)
Seq('MAIVMGRWKGAR', ExtendedIUPACProtein())
```

Sequence objects are translated according to a CodonTable object; default is the Standard translation table

```
>>> from Bio.Data import CodonTable
>>> standard_table = CodonTable.unambiguous_dna_by_name["Standard"]
>>> standard_table.stop_codons            # returns all stop codons
['TAA', 'TAG', 'TGA']
>>> standard_table.start_codons           # returns all start codons
['TTG', 'CTG', 'ATG']
>>> standard_table.forward_table['GTG']   # returns AA for codon
'V'
>>> standard_table.back_table['V']        # returns ONE codon for AA
'GTT'
>>> print(standard_table)
```

```
Table 1 Standard, SGC0

  |  T      |  C      |  A      |  G      |
--+---------+---------+---------+---------+--
T | TTT F   | TCT S   | TAT Y   | TGT C   | T
T | TTC F   | TCC S   | TAC Y   | TGC C   | C
T | TTA L   | TCA S   | TAA Stop| TGA Stop| A
T | TTG L(s)| TCG S   | TAG Stop| TGG W   | G
--+---------+---------+---------+---------+--
C | CTT L   | CCT P   | CAT H   | CGT R   | T
C | CTC L   | CCC P   | CAC H   | CGC R   | C
C | CTA L   | CCA P   | CAA Q   | CGA R   | A
C | CTG L(s)| CCG P   | CAG Q   | CGG R   | G
--+---------+---------+---------+---------+--
A | ATT I   | ACT T   | AAT N   | AGT S   | T
A | ATC I   | ACC T   | AAC N   | AGC S   | C
A | ATA I   | ACA T   | AAA K   | AGA R   | A
A | ATG M(s)| ACG T   | AAG K   | AGG R   | G
--+---------+---------+---------+---------+--
G | GTT V   | GCT A   | GAT D   | GGT G   | T
G | GTC V   | GCC A   | GAC D   | GGC G   | C
G | GTA V   | GCA A   | GAA E   | GGA G   | A
G | GTG V   | GCG A   | GAG E   | GGG G   | G
--+---------+---------+---------+---------+--
```

Biopython supports over translation using over 40 genetic code tables; choose carefully!

```
>>> from Bio.Data import CodonTable
>>> CodonTable.unambiguous_dna_by_name.keys()      # list names of codon tables
dict_keys(['Standard', 'SGC0', 'Vertebrate Mitochondrial', 'SGC1',
           'Yeast Mitochondrial', 'SGC2', 'Mold Mitochondrial',
           'Protozoan Mitochondrial', 'Coelenterate Mitochondrial',
           'Mycoplasma', 'Spiroplasma', 'SGC3', 'Invertebrate Mitochondrial',
           'SGC4', 'Ciliate Nuclear', 'Dasycladacean Nuclear',
           'Hexamita Nuclear', 'SGC5', 'Echinoderm Mitochondrial',
           'Flatworm Mitochondrial', 'SGC8', 'Euplotid Nuclear', 'SGC9',
           'Bacterial', 'Archaeal', 'Plant Plastid', 'Alternative Yeast Nuclear',
           'Ascidian Mitochondrial', 'Alternative Flatworm Mitochondrial',
           'Blepharisma Macronuclear', 'Chlorophycean Mitochondrial',
           'Trematode Mitochondrial', 'Scenedesmus obliquus Mitochondrial',
           'Thraustochytrium Mitochondrial', 'Pterobranchia Mitochondrial',
           'Candidate Division SR1', 'Gracilibacteria',
           'Pachysolen tannophilus Nuclear', 'Karyorelict Nuclear',
           'Condylostoma Nuclear', 'Mesodinium Nuclear', 'Peritrich Nuclear',
           'Blastocrithidia Nuclear'])
>>> len(CodonTable.unambiguous_dna_by_name)  # how many?
43
```

Sequence objects are immutable, but can easily be converted to and from ***mutable sequences***

```
>>> from Bio.Seq import Seq
>>> my_seq                          # create sequence
Seq('GATTACA')
>>> my_seq[0] = 'C'                 # attempt to modify seq
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'Seq' object does not support item assignment
>>> mut_seq = my_seq.tomutable()    # convert to mutable seq
>>> mut_seq[0] = 'C'                # modify seq successfuly
>>> mut_seq
MutableSeq('CATTACA')
>>> new_seq = mut_seq.toseq()       # convert to immutable seq
>>> new_seq
Seq('CATTACA')
```

Read fasta files using *SeqIO.parse()*; this function returns a container of iterable SeqRecords

```
>>> from Bio import SeqIO
>>> # create iterable container of SeqRecords
>>> f = SeqIO.parse('example.fasta', 'fasta')
>>> for row in f:
...   print( row.id + ' : ' + row.seq )
...

Species_A : ACGCTG
Species_B : ACTCTG
Species_C : AGTATC
Species_D : AGTCTC

>>> # convert to dict of SeqRecords
>>> d = SeqIO.to_dict(SeqIO.parse('example.fasta','fasta'))
>>> d
{'Species_A': SeqRecord(seq=Seq('ACGCTG', SingleLetterAlphabet()), id='Speci
 'Species_B': SeqRecord(seq=Seq('ACTCTG', SingleLetterAlphabet()), id='Speci
 'Species_C': SeqRecord(seq=Seq('AGTATC', SingleLetterAlphabet()), id='Speci
 'Species_D': SeqRecord(seq=Seq('AGTCTC', SingleLetterAlphabet()), id='Speci
>>> d['Species_A']
SeqRecord(seq=Seq('ACGCTG', SingleLetterAlphabet()), id='Species_A', name='S
```

Write fasta files using *SeqIO.write()*; this function expects a list of *SeqRecord* objects

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> # make sequence records to write
>>> rec1 = SeqRecord(seq=Seq('ACGTTA'),id='Species_A',description='')
>>> rec2 = SeqRecord(seq=Seq('TCGTTA'),id='Species_B',description='')
>>> rec3 = SeqRecord(seq=Seq('ACGTGT'),id='Species_C',description='')
>>> my_records = [rec1, rec2, rec3]  # list of records
>>> SeqIO.write(my_records, 'new_file.fasta', 'fasta')
```

```
$ cat new_file.fasta
>Species_A <unknown description>
ACGTTA
>Species_B <unknown description>
TCGTTA
>Species_C <unknown description>
ACGTGT
```

Create *MultSeqAlignment* objects using *Bio.AlignIO*; access subsets of alignment with slice-indexing

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("new_file.fasta", "fasta")
>>> print(alignment)              # full alignment
SingleLetterAlphabet() alignment with 3 rows and 6 columns
ACGTTA Species_A
TCGTTA Species_B
ACGTGT Species_C
GCATGT Species_D
>>> print(alignment[1:3,:])
SingleLetterAlphabet() alignment with 2 rows and 6 columns
TCGTTA Species_B
ACGTGT Species_C
>>> print(alignment[:,3:5])
SingleLetterAlphabet() alignment with 4 rows and 2 columns
TT Species_A
TT Species_B
TG Species_C
TG Species_D
>>> print(alignment[1:3,3:5])
SingleLetterAlphabet() alignment with 2 rows and 2 columns
TT Species_B
TG Species_C
```

MultSeqAlignment objects can be constructed within Python (*i.e.* not parsed from file)

```
>>> from Bio.Seq import Seq
>>> from Bio.SeqRecord import SeqRecord
>>> from Bio.Align import MultipleSeqAlignment
>>> from Bio import AlignIO
>>> alignment = MultipleSeqAlignment(
...     [
...         SeqRecord(Seq("ACTCCTA"), id='seq1'),
...         SeqRecord(Seq("AAT-CTA"), id='seq2'),
...         SeqRecord(Seq("CCTACT-"), id='seq3'),
...         SeqRecord(Seq("TCTCCTC"), id='seq4'),
...     ]
... ) # create a list of SeqRecord objects
...
>>> print(alignment)
Alignment with 4 rows and 7 columns
ACTCCTA seq1
AAT-CTA seq2
CCTACT- seq3
TCTCCTC seq4

>>> # write new alignment to file
>>> AlignIO.write(alignment, 'new_alignment.fasta', 'fasta')
```

Combining ideas to write a Biopython script

```
#
from Bio import AlignIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment

ifn = 'example_nt.fasta'                    #
ofn = 'example_aa.fasta'                    #
fmt = 'fasta'                               #
align = AlignIO.read(ifn, fmt)              #
x = []                                      #
for row in align:                           #
    row.id = 'tmp_' + row.id                #
    s = row.seq.tomutable()                 #
    s[0:3] = 'GGG'                          #
    row.seq = s.toseq().translate()         #
    x.append(row)                           #

new_align = MultipleSeqAlignment(x)         #
AlignIO.write(new_align, ofn, fmt)          #
```

Combining ideas to write a Biopython script

```
# import libraries
from Bio import AlignIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment

ifn = 'example_nt.fasta'                # input filename
ofn = 'example_aa.fasta'                # output filename
fmt = 'fasta'                           # file format
align = AlignIO.read(ifn, fmt)          # read in nt fasta alignment
x = []                                  # create empty list
for row in align:                       # loop over alignment rows
    row.id = 'tmp_' + row.id            # change row id
    s = row.seq.tomutable()             # let row sequence be edited
    s[0:3] = 'GGG'                      # overwrite first codon w/ GGG
    row.seq = s.toseq().translate()     # translate into AAs
    x.append(row)                       # enter row into list, x

new_align = MultipleSeqAlignment(x)     # create new MSA from x
AlignIO.write(new_align, ofn, fmt)      # write new MSA as fasta
```

Lab 10B

github.com/WUSTL-Biol4220/home/labs/lab_10B.md