

Useful modules, Biopython

Week 8, day 1

os.path

- `from os import path (etc.)`
- Platform-specific methods to manipulate file paths and names
- Because of differences across OSs:
 - `/Users/agapow`
 - `C:\Documents & Settings\Users\agapow`
 - `:Documents`

Querying paths

- If path points to README.txt at /Users/agapow/Sites (i.e. /Users/agapow/Sites/README.txt):
 - `os.path.exists (path): True`
 - `os.path.basename (path): 'README.txt'`
 - `os.path.dirname (path): 'Users/agapow/Sites'`
 - `os.path.isfile (path): True`
 - `os.path.isdir (path): False`

Example of os.path

```
def extractDataFromFile (filepath):  
    ## Preconditions:  
    assert (os.path.exists (filepath)), \  
        "'%s' doesn't exist" % filepath  
    assert (os.path.isfile (filepath)), \  
        "'%s' is a directory not a file" % filepath  
    ## Main:  
    theFileHandle = open (filepath, 'r')
```

Extracting paths

- `path = Users/agapow/Sites/README.txt`
- `os.path.splitext (path): ('Users/agapow/Sites/README', '.txt')`
- `os.path.splitdrive (path): ('', 'Users/agapow/Sites/README.txt')`
- `os.path.join ('a', 'b', 'c'): 'a/b/c'`

Getting other files

```
# CAIC requires '.phyl' & '.blen' files in same dir
theBlenPath = 'Users/agapow/data/carn.blen'
theStem, theExt = os.path.splitext (theBlenPath)
thePhylPath = theStem + '.phyl'

# to find 'data' dir in same folder
theDir = os.path.dirname (theBlenPath)
theSubDir = os.path.join (theDir, 'data')
assert (os.path.exists (theSubDir))
assert (os.path.isdir (theSubDir))
```

tempfile

- `import tempfile`
- What if you need to create a temporary physical file?
- `mktemp()`: returns path
- `mkdtemp(): (dir)` returns path
- `gettempdir()`: return directory of temp files

Using tempfile

```
# webservice needs to do a bunch of processing
import tempfile
# make a workspace for processing
theTDir = tempfile.mkdtemp()
# make your working files in new space
theSeqFilePath = os.path.join (theTDir, 's.fasta')
[...]

# clean up?
```

optparse

- `import optparse`
- For parsing commandline arguments to program
 - `myscript -f data.txt --nosound --limit 3`
- Unix/GNU-style short (`-f`) and long (`--file`) arguments with different types of parameters
- Get sensible `-h / --help` option for free

optparse calls

- `OptionParser()`: parser constructor
- `option_parser.add_option()`: add option and action and constraints
- `option_parser.set_defaults()`: give value for options if they aren't passed on commandline
- `option_parser.parse_args()`: get commandline args

optparse calls 2

- `OptionParser()`: no arguments
- `option_parser.set_defaults()`: key=value arguments
- `option_parser.parse_args()`: returns options (data structure with options as members with values) and arguments (everything else)

optparse calls 3

- `option_parser.add_option (short_arg, long_arg, action=, type=, nargs=, dest=, help=)`
- *short_arg*: e.g. '-f', '-d'
- *long_arg*: e.g. '--file', '--dontcount'
- *action*: what to do with value
 - *store*: store value after arg
 - *store_true*: store True

optparse calls 4

- *type*: 'int', 'long', 'choice', 'float'
- *dest*: name of member to store in
- *nargs*: the number of arguments to expect
- *help*: a description message to print at prompt

Optparse example

```
import optparse
p = optparse.OptionParser()

p.add_option('-f', '--file', dest='infile',
             action='store', type='string')
p.add_option('--debuglevel', dest='dlevel',
             action='store', type='int')
p.add_option('-p', dest='precise',
             action='store_true')
```

Optparse example

```
# myscript --file d.txt -p alpha.data beta.data

if __name__ == '__main__':
    import optparse
    p = optparse.OptionParser()
    [...]
    p.set_defaults (infile='', dlevel=0, precise=False)
    opt, args = p.parse_args()

# opt.infile = 'd.txt'
# opt.precise = True
# opt.dlevel = 0
# args = ['alpha.data', 'beta.data']
```

Also ...

- Get `-h / --help` for free
- Get `-v / --version` for free
- Exit with error if unrecognised argument passed

Biopython

- www.biopython.org: A large number of bioinformatics related modules
 - Common file format reading & writing
 - Classes for usual data (sequences, alignments etc.)
 - Usual operations (alignments, Blasting)
 - Data base queries etc.

But ...

- (cough) Under-documented
- Inconsistent
- Evolving
- Some bleeding edge

Start with the obvious

- `import Bio`
- `Seq`: a class for simple biosequences
- `from Bio import Seq`
- `mySeq = Seq.Seq('ACGTTTGCGC')`
- Acts like a string (index, length, slicing)

Bio.Seq example

```
>>> import Bio
>>> from Bio import Seq
>>> mySeq = Seq.Seq ('acggtcggtggggccc')
>>> mySeq
Seq ('acggtcggtggggccc', Alphabet())
>>> mySeq[0]
Seq ('a', Alphabet())
>>> mySeq[:5]
Seq ('acggt', Alphabet())
>>> mySeq[:5] + mySeq[-3:]
Seq ('acggtccc', Alphabet())
```

What's an alphabet?

- Allowable symbols in sequence
- Governs conversion & compatibility of sequences
- Can't combine two Seqs with different alphabets
- Defined in `Bio.Alphabet.IUPAC`
- Useful functions in `Bio.Alphabet`

Make your own alphabet

```
import Bio
from Bio import Alphabet
from Alphabet import IUPAC
```

```
GAPPEDAMBIG_DNA_ALPHABET = Alphabet.Gapped (
    IUPAC.ambiguous_dna, '-')
```

```
LegalDnaLetters = IUPAC.ambiguous_dna.letters
LegalGappedDnaLetters = GAPPEDAMBIG_DNA_ALPHABET.letters
```

And SeqRecords?

- A wrapper class for Seqs
- Contains a Seq (as `.seq`) and SeqAnnotations
- Some Biopython functions take Seqs, others take SeqRecords