

# **INTRODUCTION TO PYTHON: DAY FOUR**

**STEPHANIE SPIELMAN**

**BIG DATA IN BIOLOGY SUMMER SCHOOL, 2015**

**CENTER FOR COMPUTATIONAL BIOLOGY AND BIOINFORMATICS**

**UNIVERSITY OF TEXAS AT AUSTIN**

# PYTHON MODULES

- **Separate libraries of code that provide specific functionality for a certain set of tasks**
- **Some are part of *base Python* and some are not**

# A FEW BASE-PYTHON MODULES

- `os` and `shutil`
  - Useful for interacting with the operating system
- `sys`
  - Useful for interacting with the Python interpreter
- `subprocess`
  - Useful for calling external software from your Python script
- `re`
  - Regular expressions

# LOADING MODULES IN A SCRIPT

- Use the **import** command at the **\*top\*** of your script:

```
import os
```

```
import os as opsys
```

```
from os import *
```

```
from os import <function/submodule>
```

# LOADING MODULES IN A SCRIPT

- Use the import command at the **\*top\*** of your script:

```
import os  
import os as opsys
```

} use as `os.function_name()`  
`opsys.function_name()`

```
from os import *  
from os import <function/submodule>
```

} use as `function_name()`

# THE OS/SHUTIL MODULES

- Functions provide UNIX commands

# THE OS/SHTIL MODULES

- Functions provide UNIX commands

os/shutil function	UNIX equivalent
os.remove("filename")	rm filename
os.rmdir("directory")	rm -r directory
os.chdir("directory")	cd directory
os.listdir("directory")	ls directory
os.mkdir("directory")	mkdir directory
shutil.copy("oldfile", "newfile")	cp oldfile newfile
shutil.move("oldfile", "newfile")	mv oldfile newfile

# THE SYS MODULE

- A few attributes I find useful:
  - `sys.path`
  - `sys.argv`



# THE SYS MODULE

- **A few attributes I find useful:**
  - `sys.path`
  - `sys.argv`
- **`sys.path` is a list of directories in your PYTHONPATH**

# THE SYS MODULE

- **A few attributes I find useful:**
  - `sys.path`
  - `sys.argv`
- **`sys.path` is a list of directories in your PYTHONPATH**

```
import sys
```

```
# Add directories as usual, with append!  
sys.path.append("directory/I/want/to/access")
```

# THE SYS MODULE

- **A few attributes I find useful:**
  - `sys.path`
  - `sys.argv`
- **`sys.argv` is a list of command-line input arguments**

# THE SYS MODULE

- **A few attributes I find useful:**
  - `sys.path`
  - `sys.argv`
- **`sys.argv` is a list of command-line input arguments**
  - `sys.argv[0]` is the name of the script

# THE SUBPROCESS MODULE

- Use `subprocess.call()` to run external processes and/or softwares

```
import subprocess, sys
```

```
# Call an external software, FastTree
```

```
result = subprocess.call("FastTree infile > outfile", shell=True)
```

# THE SUBPROCESS MODULE

- Use `subprocess.call()` to run external processes and/or softwares

```
import subprocess, sys
```

```
# Call an external software, FastTree
```

```
result = subprocess.call("FastTree infile > outfile", shell=True)
```

```
# Variable "result" stores the UNIX exit code (1 = error, 0 = ok)
```

# THE SUBPROCESS MODULE

- Use `subprocess.call()` to run external processes and/or softwares

```
import subprocess, sys

# Call an external software, FastTree
result = subprocess.call("FastTree infile > outfile", shell=True)

# Variable "result" stores the UNIX exit code (1 = error, 0 = ok)
if result != 0:
    print "There was an error in the external command!"
    sys.exit() # Immediately exits, entire script stops running
```

# THE RE MODULE

- **Beyond the scope of this course, but oh so very useful!!**
- **Regular expression = epically flexible pattern-matching**



# USEFUL EXTERNAL MODULES

- **NumPy and SciPy**
  - Excellent for numerical analysis, working with matrices, etc.
  - TELL YOUR MATLAB FRIENDS!
- **matplotlib**
  - Plotting!
- **pandas**
  - Data manipulation and high-performance data structures
- **scikit-learn**
  - Data mining/analysis and machine learning
- **IPython**
  - cracked-out python interpreter
- **DendroPy**
  - Phylogenetic tree analysis and manipulation (not builder)

# **BIOPYTHON IS A STANDARD BIOINFORMATICS TOOL**

- **Convenient for parsing and manipulating  
sequence data**

# BIOPYTHON IS A STANDARD BIOINFORMATICS TOOL

- Convenient for parsing and manipulating sequence data
- Epic tutorial linked on the course website:  
<http://biopython.org/DIST/docs/tutorial/Tutorial.html>
- And this wiki: <http://biopython.org/wiki/Biopython>

# THE LANGUAGE OF BIOPYTHON

- **Sequences are stored as Seq objects**
  - Simply the sequence string and the alphabet

# THE LANGUAGE OF BIOPYTHON

- **Sequences are stored as Seq objects**
  - Simply the sequence string and the alphabet

```
from Bio.Seq import Seq
from Bio.Alphabet import * # not always needed
```

```
# Define a Seq object
my_seq = Seq("ACTAGACAA")
my_seq = Seq("ACTAGACAA", IUPACAmbiguousDNA)
```

# THE LANGUAGE OF BIOPYTHON

- **Sequences are stored as Seq objects**
  - Simply the sequence string and the alphabet

```
from Bio.Seq import Seq
from Bio.Alphabet import * # not always needed
```

```
# Define a Seq object
my_seq = Seq("ACTAGACAA")
my_seq = Seq("ACTAGACAA", IUPACAmbiguousDNA)
```

```
# Manipulate with various methods. Use dir() for more!
my_seq.translate()
my_seq.transcribe()
my_seq.complement()
my_seq.tomutable()
```

# THE LANGUAGE OF BIOPYTHON

- Sequence *records* are stored as SeqRecord objects

# THE LANGUAGE OF BIOPYTHON

- **Sequence *records* are stored as SeqRecord objects**

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

```
# Define a SeqRecord object, with a Seq object  
my_seq = Seq("ACTAGACAA")  
my_seqrecord = SeqRecord(my_seq, id = "unique_id")
```



# THE LANGUAGE OF BIOPYTHON

- **Sequence *records* are stored as SeqRecord objects**

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

```
# Define a SeqRecord object, with a Seq object
my_seq = Seq("ACTAGACAA")
my_seqrecord = SeqRecord(my_seq, id = "unique_id")
```

```
# Attributes and methods of SeqRecord objects (again,
dir() )
my_seqrecord.id
my_seqrecord.seq
my_seqrecord.description
```

# SEQ OBJECTS ARE **\*NOT\*** STRINGS!

```
my_seq = Seq("ACTAGACAA")
```

```
# Can re-cast to a string, as needed  
my_string_seq = str(my_seq)
```

# READING SEQUENCE DATA

- **Two input/output BioPython modules:**
  - SeqIO for sequence files
  - AlignIO for multiple sequence alignment files

# READING SEQUENCE DATA

- **Two input/output BioPython modules:**
  - SeqIO for sequence files
  - AlignIO for multiple sequence alignment files
- **Two main functions for reading:**
  - .read() if file has one sequence/alignment
  - .parse() if file has multiple sequences/alignments

# BIOPYTHON I/O SYNTAX

```
from Bio import AlignIO  
from Bio import SeqIO
```

```
# .read() and .parse() take 2 arguments:
```

```
# <AlignIO/SeqIO>.<read/parse>("filename", "format")
```

# **BIOPYTHON READS FILES AS SEQRECORD OBJECTS**



# BIOPYTHON READS FILES AS SEQRECORD OBJECTS

```
from Bio import AlignIO

# Read in an alignment file
alignment_records = AlignIO.read("aln.phy", "phylip")

# We can loop over records!
for record in alignment_records:
    print record.id
    print record.seq # Consider re-casting to str()
```

# BIOPYTHON READS FILES AS SEQRECORD OBJECTS

```
from Bio import AlignIO

# Read in an alignment file
alignment_records = AlignIO.read("aln.phy", "phylip")

# We can loop over records!
for record in alignment_records:
    print record.id
    print record.seq # Consider re-casting to str()
```

Remember: import the Seq, SeqRecord modules to manipulate these!



# BIOPYTHON READS FILES AS SEQRECORD OBJECTS

```
from Bio import SeqIO
```

```
# Read in an file with many unaligned sequences  
seq_records = list(SeqIO.read("seqs.fasta", "fasta"))
```

# BIOPYTHON READS FILES AS SEQRECORD OBJECTS

```
from Bio import SeqIO

# Read in an file with many unaligned sequences
seq_records = list(SeqIO.read("seqs.fasta", "fasta"))

# Again, loop
for rec in seq_records:
    # commands
```

# CONVERTING SEQUENCE DATA FORMAT

- Easily convert between sequence file formats with `.convert()`

# CONVERTING SEQUENCE DATA FORMAT

- Easily convert between sequence file formats with `.convert()`

```
from Bio import AlignIO
```

```
# Input file is FASTA, but we want PHYLIP!
```

```
#AlignIO.convert(<infile>, <informat>, <outfile>,  
outformat>)
```

```
AlignIO.convert("in.fasta", "fasta", "out.phy", "phylip")
```

# WRITING SEQUENCE DATA

- Use the `.write()` method to write SeqRecord object(s) to a file

# WRITING SEQUENCE DATA

- Use the `.write()` method to write `SeqRecord` object(s) to a file

```
from Bio import SeqIO
```

```
SeqIO.write(<record(s)>, <outfile>, <outformat>)
```

# WRITING SEQUENCE DATA

- Use the `.write()` method to write `SeqRecord` object(s) to a file

```
from Bio import SeqIO
```

```
SeqIO.write(<record(s)>, <outfile>, <outformat>)
```

- For various reasons, just use `SeqIO` for writing

# WRITING A SINGLE SEQUENCE TO A FILE

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

# Define a SeqRecord object
seq_object = Seq("ACGTC")
seq_record = SeqRecord(seq_object, id = "my_id")

# Write it to a file
SeqIO.write(seq_record, "outfile.fasta", "fasta")
```



# WRITING MULTIPLE SEQUENCES TO A FILE

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

# Save lots of SeqRecord objects in a list!
recs = []
for i in range(10):
    seq_object = Seq(<some sequence>)
    seq_record = SeqRecord(seq_object, id = <some_id>)
    recs.append(seq_record)

SeqIO.write(recs, "outfile.fasta", "fasta")
```

**EXERCISE BREAK**

# **SCRIPTING**

**For the remainder of the class, we will focus on writing scripts!**

**Goal:**

- **Parse sequence data files and extract meaningful information**
- **Perform some calculations**

# SCRIPT #1

Determine the average *GC-content* for all sequences in a given file

# SCRIPT #1

**Determine the average *GC-content* for all sequences in a given file**

- Your script should include two functions:
  - A function for reading in a sequence file
  - A function for computing the GC-content for a *single sequence* (why?)

## SCRIPT #2

Determine the *pairwise distance* between two sequences

## SCRIPT #2

Determine the *pairwise distance* between two sequences

ACGTAAA  
AGGTAAT

## SCRIPT #2

Determine the *pairwise distance* between two sequences

ACGTAA  
AGGTAAT

$$\text{Distance} = \# \text{diff} / \text{length} = 2 / 7 = 0.286$$



# SCRIPT #2

Determine the *pairwise distance* between two sequences

- Your script should include two functions:
  - A function for reading in a sequence file
  - A function for computing pair-wise distances

# SCRIPT #2

Determine the *pairwise distance* between two sequences

- Your script should include two functions:
  - A function for reading in a sequence file
  - A function for computing pair-wise distances
- Important considerations:
  - Allow for different sequence file formats
  - The sequences must be the same length

# SCRIPT #2

Determine the *pairwise distance* between two sequences

- Your script should include two functions:
  - A function for reading in a sequence file
  - A function for computing pair-wise distances
- Important considerations:
  - Allow for different sequence file formats
  - The sequences must be the same length (how can we enforce this?)