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

READING AND WRITING FILES IN PYTHON

- This is where Python really shines!

READING AND WRITING FILES IN PYTHON

- **Python does not deal with files directly**
  - We interact with files via special variables, called **handles**

READING AND WRITING FILES IN PYTHON

- **Python does not deal with files directly**
  - We interact with files via special variables, called **handles**

- **Interact with files in 3 main *modes*:**
  - Read-only (`"r"`)
  - Write-only (`"w"`)
  - Append (`"a"`)

OPENING FILES FOR READING

```
# Name of file to open
filename = "my_file_with_important_stuff.txt"

# Define handle with the .open() function
file_handle = open(filename, "r")  # two arguments

# Read the file contents with the .read() method
file_contents = file_handle.read()

# Close the file when done with the .close() method (!!!)
file_handle.close()
```

OPENING FILES FOR READING

```
# Name of file to open
filename = "my_file_with_important_stuff.txt"

# Define handle with the .open() function
file_handle = open(filename, "r")  # two arguments

# Read the file contents with the .read() method
file_contents = file_handle.read()

# Close the file when done with the .close() method (!!!)
file_handle.close()
```

OPENING FILES FOR READING

```
# Name of file to open
filename = "my_file_with_important_stuff.txt"

# Define handle with the .open() function
file_handle = open(filename, "r")  # two arguments

# Read the file contents with the .read() method
file_contents = file_handle.read()

# Close the file when done with the .close() method (!!!)
file_handle.close()

print file_contents
   Line 1 of file.
   Line 2 of file.
   Line 3 of file.
   ...
```

The entire body of the file, as a **single string**!

LOOPING OVER LINES IN A FILE

```
filename = "my_file_with_important_stuff.txt"

file_handle = open(filename, "r")
file_contents = file_handle.read()
file_handle.close()
```

LOOPING OVER LINES IN A FILE

```
filename = "my_file_with_important_stuff.txt"

file_handle = open(filename, "r")
file_contents = file_handle.read()
file_handle.close()

# We can convert file_contents to a list using .split()
file_contents_list = file_contents.split("\n")  # or \r
```

LOOPING OVER LINES IN A FILE

```
# Better option: use the .readlines() method

file_handle = open(filename, "r")
file_lines = file_handle.readlines()
file_handle.close()

# file_lines is a list
print file_lines
    ["Line 1 of file.\n", "Line 2 of file.\n", "Line 3 of
    file.\n", ...]
```

LOOPING OVER LINES IN A FILE

```
# Better option: use the .readlines() method

file_handle = open(filename, "r")
file_lines = file_handle.readlines()
file_handle.close()

# file_lines is a list
print file_lines
    ["Line 1 of file.\n", "Line 2 of file.\n", "Line 3 of
    file.\n", ...]

for line in file_lines:
    print line

    Line 1 of file.
    Line 2 of file.
    Line 3 of file.
    ...
```

OPENING FILES FOR WRITING

```
# Name of file to open
filename = "my_file_to_write_to.txt"

# Define handle with the .open() function
file_handle = open(filename, "w")  # note the mode!

# Write to the file with the .write() method
file_handle.write("Line 1 of the file.\n")
file_handle.write("Line 2 of the file.\n")

# Close the file when done with the .close() method (!!!)
file_handle.close()
```

OPENING FILES FOR WRITING

```
# Name of file to open
filename = "my_file_to_write_to.txt"

# Define handle with the .open() function
file_handle = open(filename, "w")  # note the mode!

# Write to the file with the .write() method
file_handle.write("Line 1 of the file.\n")
file_handle.write("Line 2 of the file.\n")

# Close the file when done with the .close() method (!!!)
file_handle.close()
```

CAUTION: writing to file **overwrites** the file, if it exists already.

ADD TO AN EXISTING FILE WITH APPEND-MODE

```
filename = "my_file_to_append_to.txt"

# Define handle with the .open() function
file_handle = open(filename, "a")  # note the mode!

# Write to the file with the .write() method
file_handle.write("Adding this line to the file.\n")

# Close the file when done with the .close() method (!!!)
file_handle.close()
```

BUT STEPHANIE, I'M REALLY LAZY!

```
# Use open and close
file_handle = open(filename, "r")
file_handle.close()
```

BUT STEPHANIE, I'M REALLY LAZY!

```
# Use open and close
file_handle = open(filename, "r")
file_handle.close()
```

```
# Use with control-flow (no need for close!)
with open(filename, "r") as file_handle:
   # do stuff to file_handle
```

BUT STEPHANIE, I'M REALLY LAZY!

```
# Use open and close
file_handle = open(filename, "r")
file_handle.close()
```

```
# Use with control-flow (no need for close!)
with open(filename, "r") as file_handle:
    # do stuff to file_handle
```

REMEMBER FILE PATHS!!

```
filename = "my_file.txt"

file_handle = open(filename, "r")
    IOError: [Errno 2] No such file or directory:
'my_file.txt'
```

REMEMBER FILE PATHS!!

```
filename = "my_file.txt"

file_handle = open(filename, "r")
    IOError: [Errno 2] No such file or directory:
'my_file.txt'


# Solution: include the full path!

filename = "my_file.txt"
path = "/path/to/files/"

file_handle = open(path + filename, "r")
```

EXERCISE BREAK

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 **o**perating **s**ystem
- `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()`

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/SHUTIL 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`

LOOPING OVER FILES WITH OS.LISTDIR

```
import os

directory = "my/directory/with/tons/of/files/"

# Obtain list of files in directory
files = os.listdir(directory)

# Loop over files that end with .txt
for file in files:
   if file.endswith(".txt"):

      f = open(directory + file, "r")
      # do something with file
      f.close()
```

LOOPING OVER FILES WITH OS.LISTDIR

```
import os

directory = "my/directory/with/tons/of/files/"

# Obtain list of files in directory
files = os.listdir(directory)

# Loop over files that end with .txt
for file in files:
   if file.endswith(".txt"):

      f = open(directory + file, "r")
      # do something with file
      f.close()
```

THE SYS MODULE

- **A few variables/functions I find useful:**
  - `sys.path`
  - `sys.argv`
  - `sys.exit()`

USING SYS.PATH

- `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")
```

USING SYS.ARGV

- `sys.argv` is a list of command-line input arguments

USING SYS.EXIT()

- `sys.exit()` **will immediately stop the interpreter and exit out of the script**

USING SYS.EXIT()

- `sys.exit()` **will immediately stop the interpreter and exit out of the script**

```
import sys

if something_important == False:
   print "Oh no, something is wrong!!!"
   sys.exit()
```

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
```

CREATING YOUR OWN MODULES

- Any python script can be imported into another!

CREATING YOUR OWN MODULES

- **Any python script can be imported into another!**

```
# Import a script named useful_functions.py

import sys
sys.path.append("/path/to/useful_functions.py/")

import useful_functions.py

from useful_functions.py import *
```

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)

INSTALL EXTERNAL MODULES

- Use the program `pip` from a bash terminal

INSTALL EXTERNAL MODULES

- **Use the program `pip` from a bash terminal**
  - Linux users can obtain pip with:

    ```
    sudo apt-get install pip
    ```

  - Mac users w/ homebrew have it already (comes with Python)

INSTALL EXTERNAL MODULES

- **Use the program `pip` from a bash terminal**
  - Linux users can obtain pip with:

    ```
    sudo apt-get install pip
    ```

  - Mac users w/ homebrew have it already (comes with Python)

- **Install package named XXX with:**

  ```
  pip install XXX
  ```

EXERCISE BREAK