INTRODUCTION TO PYTHON: DAY THREE

STEPHANIE SPIELMAN

BIG DATA IN BIOLOGY SUMMER SCHOOL, 2015
CENTER FOR COMPUTATIONAL BIOLOGY AND BIOINFORMATICS
UNIVERSITY OF TEXAS AT AUSTIN

SOME USEFUL FUNCTIONS

- `help()` **will return information about a particular function**

SOME USEFUL FUNCTIONS

- `help()` **will return information about a particular function**

- `dir()` **will return a list of which methods/attributes/functions can be used with a given object**
  - Ignore the ones with __underscores__

FUNCTIONS IN PYTHON

- **We've used several built-in functions:**
  - `len()`, `sum()`, `round()`

FUNCTIONS IN PYTHON

- **We've used several built-in functions:**
  - `len()`, `sum()`, `round()`

- **We can *write our own* functions too**
  - Reusability
  - Modular design and organization
  - Readability
  - Debugging!!

WRITING CUSTOM FUNCTIONS

- **Reference example:**

```
my_list = [1, 2, 3, 4, 5, 6]

a = len(my_list)   # here, a = 6
```

WRITING CUSTOM FUNCTIONS

- **Reference example:**

```
my_list = [1, 2, 3, 4, 5, 6]

a = len(my_list)   # here, a = 6
```

a: the returned value
len: the function name
my_list: the argument to the function

WRITING CUSTOM FUNCTIONS

```
# Anatomy of a function definition

def function_name(...arguments...):
    ...
    ...  |  Python code
    ...  |

    return returned_value
```

WRITING CUSTOM FUNCTIONS

```
# Anatomy of a function definition

def function_name(...arguments...):
    ...
    ...  |  Python code
    ...  |

    return returned_value
```

SO HOW CAN WE RE-WRITE THE LEN() FUNCTION?

```
# Example function construction
def my_len(item):

    # Loop over item to count its size
    j = 0
    for entry in item:
        j += 1

    # Return the size
    return j
```

SO HOW CAN WE RE-WRITE THE LEN() FUNCTION?

```
# Example function construction
def my_len(item):

    # Loop over item to count its size
    j = 0
    for entry in item:
        j += 1

    # Return the size
    return j

# Now we can use the function!
my_list = [1,2,3,4,5,6]
b = my_len(my_list)
print b
    6
```

SO HOW CAN WE RE-WRITE THE LEN() FUNCTION?

```
# Example function construction
def my_len(item):

    # Loop over item to count its size
    j = 0
    for entry in item:
        j += 1

    # Return the size
    return j

# Now we can use the function!
my_list = [1,2,3,4,5,6]
b = my_len(my_list)
print b
    6
```

Function names should be meaningful

Arguments are *arbitrary variable names*

Variables defined/used in the function *exist only in the function*

SO HOW CAN WE RE-WRITE THE LEN() FUNCTION?

```
# Example function construction
def my_len(item):

    # Loop over item to count its size
    j = 0
    for entry in item:
        j += 1

    # Return the size
    return j

# Now we can use the function!
my_list = [1,2,3,4,5,6]
b = my_len(my_list)
print b
    6
print j
    NameError: name 'j' is not defined
```

Function names should be meaningful

Arguments are *arbitrary variable names*

Variables defined/used in the function *exist only in the function*

FUNCTIONS ARE GENERIC FORMULAS

```
def triangle_area(l, w):
    area = l*w / 2.0
    return area
```

FUNCTIONS ARE GENERIC FORMULAS

```
def triangle_area(l, w):
    area = l*w / 2.0
    return area
```

Why 2.0 and not 2?

FUNCTIONS ARE GENERIC FORMULAS

```
def triangle_area(l, w):
    area = l*w / 2.0
    return area


# Usage 1
area = triangle_area(7, 6)
```

FUNCTIONS ARE GENERIC FORMULAS

```
def triangle_area(l, w):
    area = l*w / 2.0
    return area


# Usage 1
area = triangle_area(7, 6)

# Usage 2
length = 7
width = 6
area = triangle_area(length, width)
```

FUNCTIONS ARE GENERIC FORMULAS

```
def triangle_area(l, w):
    area = l*w / 2.0
    return area


# Usage 1
area = triangle_area(7, 6)

# Usage 2
length = 7
width = 6
area = triangle_area(length, width)

# Usage 3
l = 7
w = 6
area = triangle_area(l, w)
```

USE TEST CASES TO ENSURE YOUR FUNCTION WORKS

- **After writing a function, *always* test it with input that you know should work**

USE TEST CASES TO ENSURE YOUR FUNCTION WORKS

- **After writing a function, *always* test it with input that you know should work**

```
def triangle_area(l, w):
    area = l*w/ 2.0
    return area

# Before using the function all over the place, make sure
    that l=7, w=6 prints 21

print triangle_area(7,6)
    21
```

A NOTE ON SCOPE

- **Scope: the portion of your code where a certain variable/function exists**
- **In Python, scope is basically top-to-bottom**
- **Punch-line: define functions at the *top* of your script!**

RETURNING MULTIPLE VALUES

```
def square_cube(x):
    square = x**2
    cube   = x**3
    return square, cube    # separate values with a comma
```

RETURNING MULTIPLE VALUES

```
def square_cube(x):
    square = x**2
    cube   = x**3
    return square, cube    # separate values with a comma

s, c = square_cube(5)
print s
    25
print c
    125
```

RETURNING MULTIPLE VALUES

```
def square_cube(x):
    square = x**2
    cube   = x**3
    return square, cube    # separate values with a comma

s, c = square_cube(5)
print s
    25
print c
    125

# Equivalent usage
answer = square_cube(5)
print answer[0]
    25
print answer[1]
    125
```

FUNCTIONS DON'T NEED TO RETURN ANYTHING!

```
def square_cube(x):
    square = x**2
    cube   = x**3
    print str(x) + " squared is " + str(square) + ", and " +
    str(x) + " cubed is " + str(cube)
```

FUNCTIONS DON'T NEED TO RETURN ANYTHING!

```
def square_cube(x):
    square = x**2
    cube   = x**3
    print str(x) + " squared is " + str(square) + ", and " +
    str(x) + " cubed is " + str(cube)



# Simply call the function
square_cube(3)
    3 squared is 9, and 3 cubed is 27
```

FUNCTIONS DON'T NEED TO RETURN ANYTHING!

```
def square_cube(x):
    square = x**2
    cube   = x**3
    print str(x) + " squared is " + str(square) + ", and " +
    str(x) + " cubed is " + str(cube)



# Simply call the function
square_cube(3)
    3 squared is 9, and 3 cubed is 27


# What if you try to save a returned value?
a = square_cube(3)
```

FUNCTIONS DON'T NEED TO RETURN ANYTHING!

```
def square_cube(x):
    square = x**2
    cube   = x**3
    print str(x) + " squared is " + str(square) + ", and " +
    str(x) + " cubed is " + str(cube)



# Simply call the function
square_cube(3)
    3 squared is 9, and 3 cubed is 27


# What if you try to save a returned value?
a = square_cube(3)
print a
    None
```

EXERCISE BREAK

HANDLING ERRORS IN PYTHON

- **Error messages in Python are informative!**
  - See attached error cheatsheet

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