Introduction to Programming with Python

- Your instructors:
  - Shannon Larson - BMES President
  - Nick Giroux - AEMB VP for Academic Affairs
- This Jupyter notebook is available here: BMES-UMD Python Crash Course (https://github.com/girouxns/Nov2017_BMES-UMD_Python)
- A significantly more comprehensive "bootcamp" style Jupyter notebook is available here: National Institute on Aging Python Bootcamp by Chris Coletta (https://github.com/colettace/July2017_NIA_Python_Course)

What is Python?

- A general-use programming language - it can perform calculations, work with text, and use a variety of data types
- An object-oriented programming language - more on this later

What is Anaconda?

- A distribution which includes several IDEs, python core language, and libraries of add-on functions
  - An IDE (integrated development environment) is where you write and evaluate lines of code

Jupyter vs. Spyder

- Jupyter: you're reading this using Jupyter right now
  - Live code and text that is evaluated in cells, easily shared as a PDF
- Spyder: Scientific PYthon Development EnviRonment
  - Traditional scripts (like a Matlab M-file), easily evaluated multiple times

Using Jupyter Cells

Command Mode - blue border

- Activate command mode by pressing Esc
- b - insert cell below; a - insert cell above
- dd - delete cell #### Edit Mode - green border
- Activate edit mode by double-clicking or pressing Enter
- Shift+Enter - run and advance to next cell
- Ctrl+Enter - run cell #### Code vs. Text Cells

- Code cells contain executable lines of python
- Text cells contain text (borders, lists, etc.) in Markdown

What can you do with python?

- Example: Nick's work on Alzheimer's Disease and mitochondrial DNA (https://github.com/girouxns/Nov2017_BMES-UMD_Python/blob/master/BMES_CrashCourse_2017_NickResearch.ipynb)
- Example: Bacterial growth curves (https://github.com/girouxns/Nov2017_BMES-UMD_Python/blob/master/bacterial_growth_curves.py)
- Example: Play hangman (https://github.com/girouxns/Nov2017_BMES-UMD_Python/blob/master/interactive_hangman.py)

How to get started

- One statement per line (for the most part)
- No semicolon needed at the end (like in Matlab)
- Review assignment, scalar and iterable data types, and loops

Assignment

- Assignment is performed using the equals sign (=)
- Variable names go to the left of the equals sign and their values on the right
- The value is saved to your computer's memory and can be recalled, manipulated, or passed to another variable

In [1]:

```
my_variable = 5
```

In [2]:

```
my_variable
```

Out[2]:

```
5
```

In [3]:

```
my_new_variable = my_variable
```

In [4]:

```
my_new_variable
```

Out[4]:

```
5
```

In [5]:

```
my_variable = 'hello'
```

In [6]:

```
my_variable
```

Out[6]:

```
'hello'
```

print() and basic operations

- Typing the name of your variables will prompt Jupyter to print their string representation
- Only your last variable will be printed without using print()

In [7]:

```
x = 5
y = 6
```

In [8]:

```
x
y
```

Out[8]:

```
6
```

In [9]:

```
print(x)
print(y)
```

```
5
6
```

In [10]:

```
print(x*y)
print(x+y)
x = 1
print(x*y)
```

```
30
11
6
```

Data Types in Python

- Different data means different types (numbers vs. words, for example)
- Classified as scalars or iterables

Integer (scalar)

- A counting number

In [11]:

```
-23
```

Out[11]:

```
-23
```

In [12]:

```
type(5)
```

Out[12]:

```
int
```

Float (scalar)

- A decimal

In [13]:

```
3.14
```

Out[13]:

```
3.14
```

In [14]:

```
1/3
```

Out[14]:

```
0.3333333333333333
```

In [15]:

```
type(3.14)
```

Out[15]:

```
float
```

Boolean (scalar)

- True or False; use logical operators (and, or, not)

In [16]:

```
True
```

Out[16]:

```
True
```

In [17]:

```
True or False
```

Out[17]:

```
True
```

In [18]:

```
not False
```

Out[18]:

```
True
```

String (iterable)

- Multiple characters within quotation marks (single or double is fine)

In [19]:

```
"Hello!"
```

Out[19]:

```
'Hello!'
```

In [20]:

```
"Hello!" == 'Hello!'
```

Out[20]:

```
True
```

In [21]:

```
"I can contain numbers like 1 and letters"
```

Out[21]:

```
'I can contain numbers like 1 and letters'
```

List (iterable)

- A collection of values (may be of mixed types)

In [22]:

```
list1 = ['a', 'b', 'c']
print(list1)
```

```
['a', 'b', 'c']
```

In [23]:

```
list2 = [1, 'hello', True]
print(list2)
```

```
[1, 'hello', True]
```

In [24]:

```
list('abc')
```

Out[24]:

```
['a', 'b', 'c']
```

Access i-th list element using [i]

- Note: indexing is 0-based

In [25]:

```
print(list1)
print(list1[0])
```

```
['a', 'b', 'c']
a
```

Given `list3` print the largest value.

In [26]:

```
list3 = [15, 3, 100, 1.5]
```

In [27]:

```
print(list3[2])
```

```
100
```

Iteration, Conditionals, and Loops

- Iterable data types can be "looped" through or over
- Conditional "if" statements evaluate as booleans
- Two important loops: for and while

For loop using a list

In [28]:

```
months = ['January', 'February', 'March', 'April', 'May', 'June']
print(months)
print(type(months))
```

```
['January', 'February', 'March', 'April', 'May', 'June']
<class 'list'>
```

In [29]:

```
for month in months:
    print(month)
```

```
January
February
March
April
May
June
```

In [30]:

```
counter = 0
while counter < 10:
    print(counter)
    counter += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

In [31]:

```
for month in months:
    if month[0] == 'M':
        print(month)
```

```
March
May
```

If you had to print only the even numbers between 0 and 10, how would you do it with a `for` loop? Hint: look at the `counter` example.

In [32]:

```
counter = 0
while counter < 10:
    print(counter)
    counter += 2
```

```
0
2
4
6
8
```

Code Skeleton 1: Finding the volume of a sphere

- Hint: $volume = \frac{4}{3} * pi * r^2$

In [33]:

```
radius = 2
pi = 3.14
volume = (4/3) * pi * radius ** 2
print(volume)
```

```
16.746666666666666
```

Code Skeleton 2: Student grades

- Print the failing (<70) grades and names of students with names that begin with 'A' or 'M'
- Given two lists: one with student first names, the other with corresponding grades

In [34]:

```
names = ['Abigail', 'Adele', 'AK', 'Alex', 'Anne', 'Bob', 'Caroll', 'Devin', 'Ma
ry', 'Melvin', 'Michael', 'Mike', 'Tyler']
grades = [65, 80, 91, 45, 95, 32, 100, 67, 70, 85, 0, 67, 99]
```

In [35]:

```
counter = 0
for name in names:
    if name[0] == 'A' or name[0] == 'M':
        if grades[counter] < 70:
            print(name + '\t' + str(grades[counter]))
    counter += 1
```

```
Abigail 65
Alex    45
Michael 0
Mike    67
```