**Kernel:** Python 3 (Ubuntu Linux)

# Python Basics

The goal of this notebook is to provide students with knowledge of the fundamentals of python. We will take a more interactive approach than most class room settings and ask the students to do some research on their own then bring their learnings back here to share with the class.

## Variables

Variables work by assigning a value to a specific word or string of characters. It can be done using the "=" sign. This assigns whatever is on the right the string of characters on the left. We'll go ahead and try assigning a few variables in the next block.

In [2]:
```
a = 1
b = 2
c = 3
a # when you put a variable by itself, its value will be printed (This is a comment - anything after a # on the same line will be ignored)
```

Out[2]: 1

We can also perform basic algebra to these variables and save the output to a new variable

In [3]:
```
d = a + b
d
```

Out[3]: 3

In [4]:
```
e = c - a
e
```

Out[4]: 2

In [5]:
```
f = b * c
f
```

Out[5]: 6

In [6]:
```
g = a / b
g
```

Out[6]: 0.5

Sometimes we want to update the value of one variable, so we can do this

In [7]:
```
g = g*2 # the same thing works for /, +, and -
g
```

Out[7]: 1.0

This can get quite annoying with longer variable names, so we can do the following

In [8]:
```
f/=2 # there is also +=, -=, and *=
f
```

Out[8]:
```
3.0
```

There are tons of different types of variables we can make. For this next section, we will ask the students to go out and figure out how to create the following types of variable: strings, floats and booleans. Once you have created these variables, try the same operations we used above and see what they do.

Create a few string variables here, then try to add them together, subtract them, multiply them and divide them to see what happens. You may get an error, and thats okay. Its important to know what can and can't be done with a type of data.

In [3]:
```
 a = "camp"
b = "AI"
c = a + b
print(b+" "+ a)
```

Out[3]:
```
AI camp
```

Create a few float variables here, then try to add them together, subtract them, multiply them and divide them to see what happens. You may get an error, and thats okay. Its important to know what can and can't be done with a type of data.

In [4]:
```
a = 4.567
b = 5.875
c = a + b
c
```

Out[4]:
```
10.442
```

Create a few boolean variables here, then try to add them together, subtract them, multiply them and divide them to see what happens. You may get an error, and thats okay. Its important to know what can and can't be done with a type of data.

In [5]:
```
a = True
b = False
c = a + b
d = b + c
d
```

Out[5]:
```
1
```

## Lists

Lists work similarly to the variables we created above but store multiple values. We'll cover the three types metioned above then ask the students to try and create their own versions of them.

First up, lists. you can create a list with the following code.

In [2]:
```
my_list = [1, 2, 3, 4, 5]
```

To access a specific item from the list, you would do the following.

In [3]:
```
my_list[2] # here, 2 is the index you are accessing
```

Out[3]:
```
3
```

The first element in the list is at index 0, then the second element is at index 1 and so on.

In [4]:
```
print(my_list[0], my_list[1], my_list[2], my_list[3], my_list[4])
```

Out[4]:
```
1 2 3 4 5
```

**TODO:** Now its your turn. We will have students create their own version of this data type.

Below, create a list of your favorite colors.

In [0]:
```
#create a list of your favorite colors
colors_fav = ["Blue", "Green", "Black"]
```

Now figure out how to `append()` an item to your list. Maybe google: "how to append an item to a list python"

In [7]:
```
# append another color to the list of your favorite colors

```

## For loops, while loops, and if statements

We will learn more about for loops in the next lesson. For now, we will use them to go through a list and print out each element.

In [16]:
```
for number in my_list:
  print(number)
```

Out[16]:
```
1
2
3
4
5
```

Next, we will look at how to make a list run a specific number of times.

In [17]:
```
for i in range(5): # range functions the same here as the list [0,1,2,3,4,5]
    print('i = ', i)
```

Out[17]:
```
i =  0
i =  1
i =  2
i =  3
i =  4
```

Next, we'll take a look at while loops. In general, you can make a while loop do anything that a for loop can do but with slightly different syntax, so if you find youself understanding while loops better than for loops, you can just focus on using while loops.

In the block below, we will make a while loop that runs 5 times. For while loops, we can't specify how long they will run for ahead of time so its important for us to create an achievable exit condition. An exit condition is just the condition that will cause the code to stop running. This is checked every time, immediatly before the code runs or reruns. We want to make sure that this condition will eventually happen, otherwise the code will run until something crashes or we manually close it

In [18]:
```
i = 0
while i < 5:
    print("Hello World")
    i = i + 1
```

Out[18]:
```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Now for if statements. The if-elif-else statement is used to conditionally execute a statement or a block of statements. Conditions can be true or false, execute one thing when the condition is true, something else when the condition is false.

In [19]:
```
a = 1
b = 2
if a > b:
    print("a is greater than b")
```

We can add else statements to the above. An else statement is what will be run if the original statement is false.

In [20]:
```
a = 1
b = 2
if a > b:
    print("a is greater than b")
else:
    print("b is less than a")
```

Out[20]:
```
b is less than a
```

Additionally, we can use elif statements to add additional conditions to the structure.

In [21]:
```
a = 1
b = 1
if a > b:
    print("a is greater than b")
elif a == b:
    print("a equals b")
else:
    print("b is less than a")
```

Out[21]:
```
a equals b
```

Now its your turn. Either break up into 3 teams and have each team solve one of the blocks or solve them all together.

In the next block, create a for loop that counts from 0 to 10 and prints out each number. Additionally, if the number is less than 3, print out "This is a small number".

In [16]:
```
for i in range (11):
    print('i = ', i)
    if i < 3:
        print("This is a small number")
```

Out[16]:
```
i =  0
This is a small number
i =  1
This is a small number
i =  2
This is a small number
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
```

```
i =  9
i =  10
```

In the next block, create a while loop that prints out all of the even numbers between 1 and 10. Additionally, if the number is between 5 and 9, print the number out a second time.

In [29]:
```
i=0
while i<11:
    i = i+1
    if (i%2)==0:
        print('i=',i)
    if i>5 and i<9:
        print('i=',i)
```

Out[29]:
```
i= 2
i= 4
i= 6
i= 6
i= 7
i= 8
i= 8
i= 10
```

In the next block, ask the user to input 3 items and add them all to a list. Then. create a loop that prints out each item that the user entered.

In [30]:
```
input("Enter 3 items:")
for
```

Out[30]:
```
Enter 3 items: 1,2,3
```

```
'1,2,3'
```

## Functions

Functions act like variables but instead of storing a value, they store lines of code. Below we can see how a function is created and used.

In [22]:
```
def my_first_function():
    print('Hello world')
```

In [23]:
```
my_first_function()
```

Out[23]:
```
Hello world
```

Whenever the computer sees the function being used, it automatically runs the code stored inside of that function.

Additionally, functions can take in one or multiple values to use in their code, as we can see below.

In [24]:
```
def add_numbers(a, b, c):
    total = a + b + c
    print(total)
```

In [25]:
```
add_numbers(1, 2, 3)
```

Out[25]:
```
6
```

Finnally, functions can have a "return" statement. This is what the function sends back to the place it was called. You can think of functions like black boxes where the parameters in the parentheses are the input and the return statement

is the output. This output can be saved to a variable to be used later. We will redefine the add_numbers function from above using a return statement to show this off.

In [26]:
```
def add_numbers(a, b, c):
    total = a + b + c
    return total
```

In [27]:
```
out = add_numbers(1, 2, 3)
out
```

Out[27]: 6

Now its your turn! You can either work on each of the following challenges in teams of 2 or walk through each one as a group. In any case, it is recommended to look at them at a high level together so that everyone understands the concepts in each section.

In the block below, create a function that takes in 2 numbers and returns True if the product of the numbers is positive and False in all other cases.

In [32]:
```
def two_num(x,y):
    if x*y > 0:
        return True
    else:
        return False
two_num(1,2)
```

Out[32]: True

In the block below, create a function that takes in a string and returns the string in reverse order. For example, if the input was "Hello" the output should be "olleH"

In [33]:
```
def rev(x):
    return x[::-1]
rev('for')
```

Out[33]: 'rof'

In the block below, create a function that takes in a list and prints out each element 1 at a time. For example, if the input was ['Dog', 'Cat', 'Hamster'], the output should be

Dog

Cat

Hamster

In [34]:
```
def list_print(mylist):
    for i in mylist:
        print(i)
mylist = ['horse', 'gorilla', 'hen']
list_print(mylist)
```

Out[34]:
```
horse
gorilla
hen
```

# Modules (Pandas and numpy)

## Why are there Pandas in Python? What is numpy?

In the following section we will be covering modules. Above, we leaned about functions and the power that they have. Modules let us use functions that other people have created to solve problems that we commonly encounter -- you can think of this as standing on the shoulders of giants by being just lazy enough to want to write code over and over again.

For example, if we find that we frequently have to get a random number, instead of coding that entire process every time, you could just import a function that either you or someone else created in the past. Basically, modules and importing lets us have access to the work of many other coders, making it a very powerful tool.

## Numpy - For Scientific Computing in Python

Below, we will import the numpy module and use a few of its functions to show off this power. Numpy is a library which is coded outside of python, making it faster. It gives us access to better and more efficient lists

For any module you encounter, you should be prepared to read the docs. The "docs" are how we can learn to use each module, and you can find Numpy's here: https://numpy.org/.

**For using modules, you'll want to import them like so below.**

In [7]:
```
import numpy as np #we rename numpy to np so we don't have to type 5 letters every time.
```

With Numpy, we can make a list of zeros like this:

In [8]:
```
my_array = np.zeros((2,3)) # (2,3) is the shape of the array
my_array
```

Out[8]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Or a list of random decimal numbers like this:

In [9]:
```
np.random.rand(3,2) # we don't need to put the shape in braces this time
```

Out[9]:
```
array([[0.51419005, 0.54441936],
       [0.85278788, 0.61982656],
       [0.59807822, 0.61977816]])
```

Just like regular variables, we can do operations on this list

In [10]:
```
my_array2 = my_array + 1
my_array3 = my_array2 * 2
print(my_array2)
print(my_array3)
```

Out[10]:
```
[[1. 1. 1.]
 [1. 1. 1.]]
[[2. 2. 2.]
 [2. 2. 2.]]
```

**TODO:** Make an array from my_array of all -3's:

In [32]:
```
#make an array of all -3's
```

In [0]:
```

```

## Pandas - A Data Scientist's Best Friend

With numpy briefly explored above, we will now check out a little bit about pandas. Pandas is an awesome module that helps us deal with large amounts of data in an easy to understand way. Below we will look at some of its many functions

First, we'll have to import it. It is common practice to `import pandas as pd` but you are free to import it as you see fit.

In [33]:
```
import pandas as pd
```

Pandas works with objects called **DataFrames**. Not sure what that is? Read the docs: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

Or you can simply look below.

In [34]:
```
data = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})
data
```

Out[34]:

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 4 | 7 |
| **1** | 2 | 5 | 8 |
| **2** | 3 | 6 | 9 |

Our dataframe is pretty small so it's easy to show the whole thing in the output, but for larger dataframes it is important to only show a few of the rows so we don't have to scroll all over. To do this, we can use the df.head(x) function to show the first x rows of the dataframe or tail(x) to show the last x number of rows

In [35]:
```
data.head(2)
```

Out[35]:

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 4 | 7 |
| **1** | 2 | 5 | 8 |

In [36]:
```
data.tail(1)
```

Out[36]:

| | A | B | C |
|---|---|---|---|
| **2** | 3 | 6 | 9 |

Alternatively you can look at just 1 row at a time by calling it similarly to a dictionary

In [37]:
```
data['A']
```

Out[37]:
```
0    1
1    2
2    3
Name: A, dtype: int64
```

You can also get various details about the dataframe with the following functions.

In [38]:
```
# Gives you the names of all of the columns
data.columns
```

Out[38]:
```
Index(['A', 'B', 'C'], dtype='object')
```

In [39]:
```
# Gives you the names of all the columns, the number of rows in that column and a
```

```
brief description of the values stored in those columns
data.describe()
```

Out[39]:

| | A | B | C |
|---|---|---|---|
| **count** | 3.0 | 3.0 | 3.0 |
| **mean** | 2.0 | 5.0 | 8.0 |
| **std** | 1.0 | 1.0 | 1.0 |
| **min** | 1.0 | 4.0 | 7.0 |
| **25%** | 1.5 | 4.5 | 7.5 |
| **50%** | 2.0 | 5.0 | 8.0 |
| **75%** | 2.5 | 5.5 | 8.5 |
| **max** | 3.0 | 6.0 | 9.0 |

In [40]:
```
# Tells you more about what is stored in those columns
data.info()
```

Out[40]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       3 non-null      int64
 1   B       3 non-null      int64
 2   C       3 non-null      int64
dtypes: int64(3)
memory usage: 200.0 bytes
```

**Next we will check out iterating through a dataframe.**

In [41]:
```
# Going through a row 1 value at a time
for i in data['A']:
    print(i)
```

Out[41]:
```
1
2
3
```

In [42]:
```
# going through the entire dataframe 1 value at a time
print(50*"-")
for col_name in data.columns:
    for value in data[col_name]:
        print(value)
print(50*"-")
```

Out[42]:
```
--------------------------------------------------
1
2
3
4
5
6
7
8
9
--------------------------------------------------
```

In [0]: