Section 2: Python Basics

Before you get started on anything else, you'll need to understand Python, the programming language used throughout this tutorial. Let's go through some of the basics of Python together.

Strings & Print()

A **String** is a simple data type that can include letters, numbers, and symbols. Simply speaking, it's a sequence of characters. Strings must be surrounded by a pair of quotation marks. Here's a couple examples:

- "This is a string!"
- "1234 & 5678"
- "P Sherman 42 Wallaby Way Sydney"

We use the **print()** function to display outputs in the console. In this case, the output will show up right below the coding cell.

Now, it's your turn. Using the print() function provided below, enter a String inside the parenthesis that says *Hello World!*

Click "Run" to display the output.

```
In [1]: print("jhhuh")
```

```
jhhuh
```

Comments

Before we continue further, I want to show you how to comment the code you're working with. Commenting your code makes it easier to understand, both for yourself in the future and for other people. Comments are parts of the code that Python will not try to run. There's two different types of comments: Single-line, and multi-line.

Single-Line comments are comments that only take up one line in your code. These kinds of comments are good if you need to include very brief information, such as quick notes for what a part of your code does. Single line comments are denoted with a a hashtag (#).

Multi-line comments, on the other hand, can be more thorough and are used for comments that take up multiple lines of code. Muti-line comments can also be used to "comment-out" parts of your code, if you do not want it to run. These kinds of comments are denoted by three pairs of quotation marks (""") on either side of the comment.

Here's a couple examples of each:

- #This is a single line comment
- """This is a multi line comment.

```
Anything I say and/or any code I write will not be run by Python
```

End comment."""

Now you try. First, run the code below to see the output. Then, add a single-line above the first print() function in the cell below. Then, comment-out the second and third print() funtions using a multi-line comment. Now run it again.

In [32]:
```
print("Add your single line comment above this function")

print("Comment this function out.")
print("And this one too!")


#You see how the last two Strings do not show up when you run the code the sec
ond time?
```

```
Add your single line comment above this function
Comment this function out.
And this one too!
```

Variables and Assignment

One way to store and work with different types of data is by using **variables**. Variables stores data under the name you decide to give it. You can **assign** the variable different pieces of data by using the equals (=) symbol. For example, if I wrote

myFirstVariable = 24 mySecondVariable = "Hello!"

(***Please do note that the "=" sign is for assignment, not for comparison. The symbol for comparison is a double-equals sign (==).***)

In the above examples, I assigned myFirstVariable to hold the number 24. I assigned mySecondVariable to hold the String "Hello!" In each case, the name (myFirstVariable/mySecondVariable) is the name of the variable, but it holds different types of data. One holds a number, and one holds a String.

Create two different variables in the cell below and assign them any number or String value you want. Then, print both variables out using two print() functions. Be sure to put the name of your variables inside the print function in order to see it printed out on the screen.

In [33]:
```
# This is example code below.

var1 = "last"
var2 = 64

print(var1)
print(var2)

#Notice that only the data each variable holds is printed out! Not the variabl
e name itself!
```

```
last
64
```

Operations

Python allows you to perform different mathematical operations in your code, including but not limited to:

- Addition (using the + sign)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Exponents (double asterisks with no space in between: **... Ex: 2** 3 is 2 raised to the power of 3.)
- Modulus (% ...this finds the remainder after you divide two numbers)

Here's a quick example:

- ten = 5 * 2

In the above example, "ten" is the *name* of the variable. The variable is then assigned the product of 5 and 2 (10).

Practice using the operations named above. Do the following: Declare a variable named "first" and assign it two numbers that are added together to equal 15.
Declare a second variable that is assigned the value 14. Print out both variables using print().

Create a third variable called "mod" and assign it to be your first variable modulo your second variable. Print out your third variable. It should print out a 1.

```
In [34]: # Insert your code here
```

Conditional Statements and Booleans

A **boolean** can only have the values "True" or "False", either implied or explicitly stated. For example, study the following lines:

myBoolean = True mySecondBoolean = False

myBoolean == mySecondBoolean

In the first line of code above, I assigned myBoolean the value "True" (Capitalization matters!). I then assigned mySecondBoolean the value "False". If I printed these variables out, they'd print out "True" and "False" respectively. This explicitly named either variable as either True or False.

Now, in the third line, I used the double-equals (==) operator to check if myBoolean equals mySecondBoolean. But True does NOT equal False, therefore if I printed the third line out, it would read False. In the third line of code, the True/False statement is implicit.

In the code cell provided below, declare your own variables and copy what I've done above to see the code run for yourself. Then, try changing the True/False values and see what difference it makes.

In [35]:
```
# Insert code here.
```

Now let's look at conditional statements. In regular plain English, conditional statements are written in an "if...then...else" format. *If* the condition is met, *then* the second half of the sentence is valid. *Else* the other statement is valid. ("If it snows heavily, then they may cancel school! Or else we'll have to attend school this Monday.)

Conditional statements in programming work the same way. If the condition is met, then the code below it will be executed. If it is not met, then nothing happens and that piece of code is ignored. In Python, the format for writing conditional statements looks like this:

if [statement here]: CODETOEXECUTE_A

elif [statement]: CODETOEXECUTE_B

else: CODETOEXECUTE_C

**Be mindful of the indentations! They DO matter. Any code that is indented will only be executed if the if statement above it is met.

If you only want one conditional statement, you can use just the first block of code (if ([statement here]): CODETOEXECUTE_A). If there are two conditional statements, use the first and last block of code (if ([statement here]): CODETOEXECUTE_A and else: CODETOEXECUTE_C).

Look at the code provided below. Run it and look at the result. Which conditional statement was met? Then, anipulate the code to make each if-statement true in turn. (This means you'll hit "RUN" three times. Once after every edit.)

In [36]:
```
x = 4


if x == 2:
    print("x equals 2!")
elif x == 45:
    print("x equals 45!")
else:
    print("x equals 46")
```

```
x equals 46
```

Python Lists

Inside Python, there's a data structure called a List. Lists hold any number of any kind of variable. Here's how to declare a list in Python.

myFirstList = []

Then, you can add variables to that list by using the append() function. For example:

myFirstList.append(1) myFirstList.append(2) myFirstList.append(3)

This adds the numbers 1, 2, and 3 to your List. In order to go through your lists, you can use the following format to print out individual variables.

print(myFirstList[0]) print(myFirstList[1]) print(myFirstList[2])

NOTE: Lists begin with 0 as the first index. So our "1" is stored at index 0, "2" is stored at index 1, and so on.

If you're trying to print the whole list, you can just print it like this:

print(myFirstList)

This will print out all your variables that you've stored, in order.

Now you try: The code below is incorrect and incomplete. Add the necessary pieces of code to make this code run correctly and print out the list.

In [37]:
```
#This code is incorrect. Fix the mistakes.

myList = ()

myList.append("String 1")
myList.append("String 2")

print(myList[2])



#The code should print out "String 2" followed by ['String 1', 'String 2'] on
 the next line.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-37-dce20218bc0a> in <module>()
      3 myList = ()
      4
----> 5 myList.append("String 1")
      6 myList.append("String 2")
      7

AttributeError: 'tuple' object has no attribute 'append'
```

Dictionaries

A Dictionary is similar to a List, but works with two values (a key and a value) instead of using an index. A Dictionary can be declared like this:

favoriteColors = { "Jane" : "purple", "John" : "blue", "Mary" : "pink"

}

Copy/paste the code above and print out the result using print(favoriteColors).

In [ ]:
```
# This is the code they'd have.


favoriteColors = {
       "Jane" : "purple",
       "John" : "blue",
       "Mary" : "pink"

       }

print(favoriteColors)
```

You can remove values from your Dictionary by using the .pop() method. Example:

favoriteColors.pop("Jane")

This would remove Jane from your Dictionary. Try this in the cell above, where you've pasted the Dictionary I gave you. Use the .pop() method under the print() method. Then print the newly edited dictionary using a second print() function. See what happens when you

NOw it's your turn! Create your own Dictionary and practice adding and deleting key/value pairs. Print the result each time, to see how this has changed your Dictionary.

In [ ]:
```
# Insert your code here
```

That's all for this Section. Now you know all the basic concepts in Python and are ready to head on over to Section 3!