Intro to Python

Access this notebook on GITHUB or COLAB

TECHNOLOGY SCHOOL 1.0 (2022)
DATA ANALYSIS AND DATA VISUALIZATION

Table of Contents

Click on the links to go directly to specific sections on the notebook.

Estimated time needed: **25 min**

Python - Let's get you writing some Python Code now!</h1>

Welcome! This notebook will teach you the basics of the Python programming language. Although the information presented here is quite basic, it is an important foundation that will help you read and write Python code. By the end of this notebook, you'll know the basics of Python, including how to write basic commands, understand some basic types, and how to perform simple operations on them.

Does Python know about your error before it runs your code?

Python is what is called an *interpreted language*. Compiled languages examine your entire program at compile time, and are able to warn you about a whole class of errors prior to execution. In contrast, Python interprets your script line by line as it executes it. Python will stop executing the entire program when it encounters an error (unless the error is expected and handled by the programmer, a more advanced subject that we'll cover later on in this course).

```
### Import Dependencies
```

When learning Python, you will probably browse blogs and other web resources that claim certain things are `Pythonic`. Python has an opinionated way of doing things, mostly captured in the Zen of Python.

In [ ]:
```
import this
```

`Pythonic` practices are those which the general Python community has agreed are preferable, sometimes this is purely a stylistic consideration and other times it may be related to the way the Python runs.

Making your code `Pythonic` can also be useful when other Python programmers need to interact with it as they will be familiar with the idioms and paradigms you use.

Imports

In the cells above you might have noticed we used the `import <package>` syntax. This construct allows us to include code from other python files or more generally modules (collections of files) and packages (collection of modules) into the current code we are working with. For the purposes of this course, we have installed all the packages you will need on your machine, but for working with packages, some recommended tools are:

- conda
- pip

With installed packages (usually installed with one of those two "package managers"), we can import the package with the `import` command. We can also import only parts of the package. For example, one package we will use in the course is called `pandas`. We can import `pandas`

In [ ]:
```
#import pandas, import numpy
```

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

#pd.DataFrame
```

In [ ]:
```
#Datframe is a module of the pandas package i.e. a module is a collection of files

# from pandas import DataFrame as dframe

# dframe
```

In [ ]:
```
pip install pandas 1.20xxxx
```

In [ ]:
```
conda install pandas 1.20xxxx
```

### Python Syntax

When learning a new programming language, it is customary to start with an "hello world" example. As simple as it is, this one line of code will ensure that we know how to print a string in output and how to execute code within cells in a notebook.

In [ ]:
```
print ("Hello Python !")
```

After executing the cell above, you should see that Python prints `Hello, Python!`. Congratulations on running your first Python code!

[Tip:] `print()` is a function. You passed the string `'Hello, Python!'` as an argument to instruct Python on what to print.

In [ ]:
```
# Try your first Python output
print(0)
```

Comments in Python

In addition to writing code, note that it's always a good idea to add comments to your code. It will help others understand what you were trying to accomplish (the reason why you wrote a given snippet of code). Not only does this help **other people** understand your code, it can also serve as a reminder **to you** when you come back to it weeks or months later.

To write comments in Python, use the number symbol `#` before writing your comment. When you run your code, Python will ignore everything past the `#` on a given line.

In [ ]:
```
# Practice on writing comments

print('This is a TechOrigin Class')
print('Hello, Python!') # This line prints a string
print('Hi')
```

After executing the cell above, you should notice that `This line prints a string` did not appear in the output, because it was a comment (and thus ignored by Python).

The second line was also not executed because `print('Hi')` was preceded by the number sign (`#`) as well! Since this isn't an explanatory comment from the programmer, but an actual line of code, we might say that the programmer *commented out* that second line of code.

[Tip]: To execute the Python code in the code cell below, click on the cell to select it and press `Shift` + `Enter`.

### Variables

We can store results for later use. We can store a result in the computer's memory by assigning it to a **variable**.

In [ ]:
```
a = 5
```

In [ ]:
```
print(a)
```

In [ ]:
```
first_result = 1 + 1
final_result = first_result * 3.5

print(final_result)
```

Here we were able to use the result of the first calculation (stored in the variable `first_result`) in our second calculation. We store the second result in `final_result`, which we can print at the end of the cell.

- Variables help us keep track of the information we need to successfully execute a program.
- Variables are objectsthat are stored in memory
- Variables can be used to store a variety of types of information.

Since variables can be used to store so many types of information, it's a good idea to give those variables descriptive names like I did. This helps us write code that is easy to read, helps when we're trying to find and fix mistakes or share code with others.

Some Strict Rules for Naming Variables

- It start with a-z, A-Z, a1-z0, Aa-Zz, etc
- It should not have spaces between them. eg
- It should not start with numbers. e.x. 10teachers
- It is prefer to combine multiple words in camelCase. ex. myName or my_Name
- It should not bear the python reserved words. e.g True, if, Except, etc

### Strings

A **string:** is a sequence of characters.

In [ ]:
```
# Use quotation marks for defining string
"Michael Jackson"
```

It can also be a combination of spaces and digits:

In [ ]:
```
# Digitals and spaces in string
type('1 2 3 4 5 6 ')
```

A string can also be a combination of special characters :

In [ ]:
```
# Special characters in string

'@#2_#]&*^%$'
```

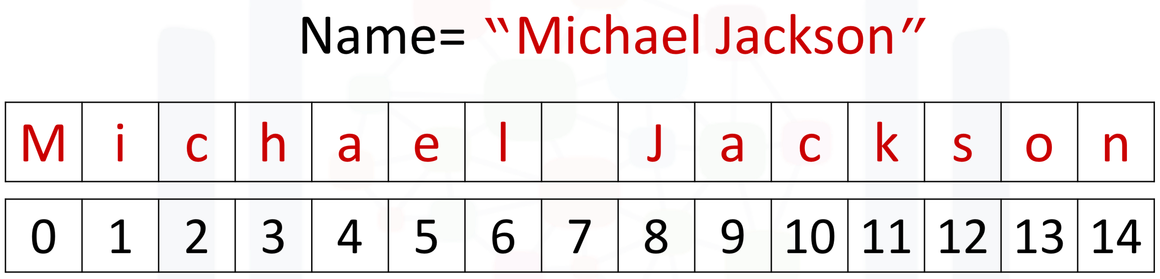
We can bind or assign a string to another variable:

```
In [ ]:
# Assign string to variable
Name= "Michael Jackson"
```

```
In [ ]:
Name
```

String Indexing

It is helpful to think of a string as an ordered sequence. Each element in the sequence can be accessed using an index represented by the array of numbers:



The first index can be accessed as follows:

```
In [ ]:
# Print the first element in the string
print(Name[0])
```

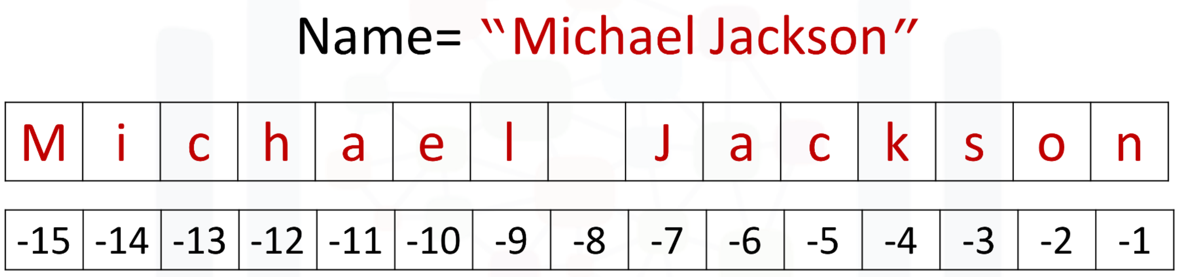
```
In [ ]:
# Print the element on index 6 in the string
print(Name[6])
```

Moreover, we can access the 13th index:

```
In [ ]:
# Print the element on the 13th index in the string
print(Name[13])
```

Negative Indexing

We can also use negative indexing with strings:



Negative index can help us to count the element from the end of the string.

The last element is given by the index -1:

```
In [ ]:
# Print the last element in the string
print(Name[-1])
```

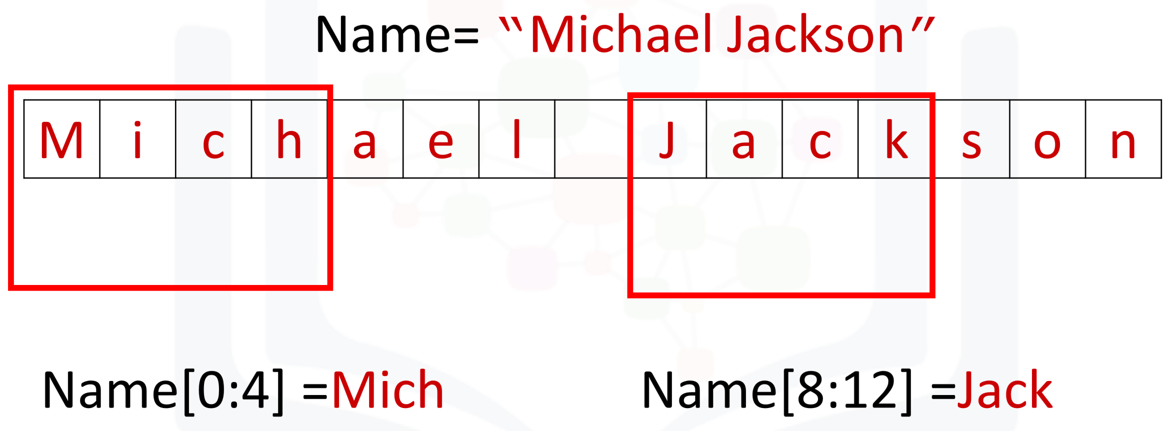
In [ ]:
```python
# Print the first element in the string
print(Name[-15])
```

Len Function

We can find the number of characters in a string by using `len`, short for length:

In [ ]:
```python
# Find the length of string
len("Michael Jackson")
```

Slicing



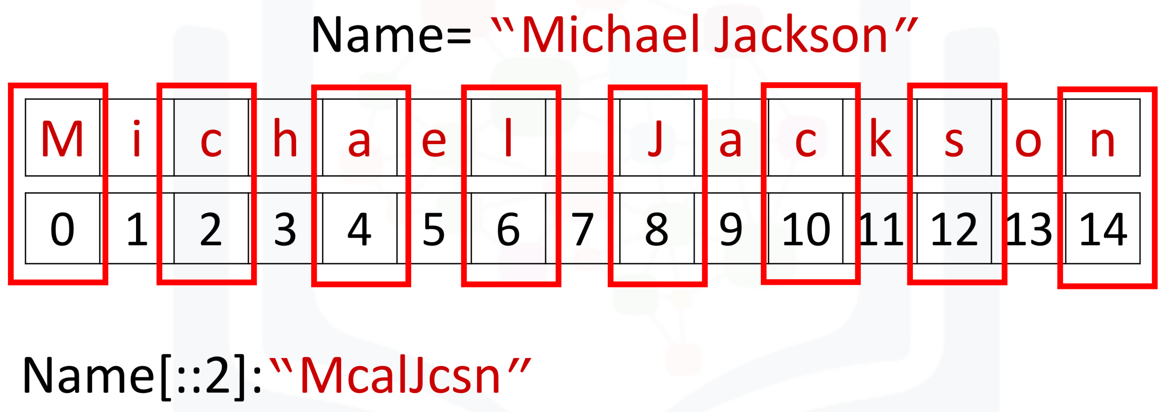
[Tip]: When taking the slice, the first number means the index (start at 0), and the second number means the length from the index to the last element you want (start at 1)

In [ ]:
```python
# Take the slice on variable Name with only index 0 to index 3
Name = "Michael Jackson"
Name[0:4]
```

In [ ]:
```python
# Take the slice on variable Name with only index 8 to index 11
Name[8:12]
```

Stride

We can also input a stride value as follows, with the '2' indicating that we are selecting every second variable:



In [ ]:
```
# Get every other element. The elements on the index 0, 2, 4 ...
Name[::2]
```

In [ ]:
```
# Get every third element. The elments on index 0, 3, 6 ...

Name[2::4]
```

We can also incorporate slicing with the stride. In this case, we select the first five elements and then use the stride:

In [ ]:
```
# Get every second element in the range from index 0 to index 4
Name[0:5:2]
```

Concatenate Strings

We can concatenate or combine strings by using the addition symbols, and the result is a new string that is a combination of both:

In [ ]:
```
Name9 = 'Chimaobi'
```

In [ ]:
```
# Concatenate two strings

Statement = Name + ' is the best ' + Name9
Statement
```

To replicate values of a string we simply multiply the string by the number of times we would like to replicate it. In this case, the number is three. The result is a new string, and this new string consists of three copies of the original string:

In [ ]:
```
# Print the string for 3 times

print(3 * "Michael Jackson")
```

You can create a new string by setting it to the original variable. Concatenated with a new string, the result is a new string that changes from Michael Jackson to “Michael Jackson is the best".

In [ ]:
```
# Concatenate strings

Name = "Michael Jackson"
Name = Name + " is the best"
Name
```

Escape Sequences

Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, back slash "n" represents a new line. The output is given by a new line after the back slash "n" is encountered:

In [ ]:
```
# New line escape sequence
```

```
print(" Michael Jackson \n is the best creature" )
```

Similarly, back slash "t" represents a tab:

In [ ]:
```
# Tab escape sequence

print(" Michael Jackson \t is the best" )
```

If you want to place a back slash in your string, use a double back slash:

In [ ]:
```
# Include back slash in string

print(" Michael Jackson \\ is the best" )
```

We can also place an "r" before the string to display the backslash:

In [ ]:
```
# r will tell python that string will be display as raw string
print(r"bobby weds \n amara")
```

String Operations

There are many string operation methods in Python that can be used to manipulate the data. We are going to use some basic string operations on the data.

Let's try with the method `upper`; this method converts lower case characters to upper case characters:

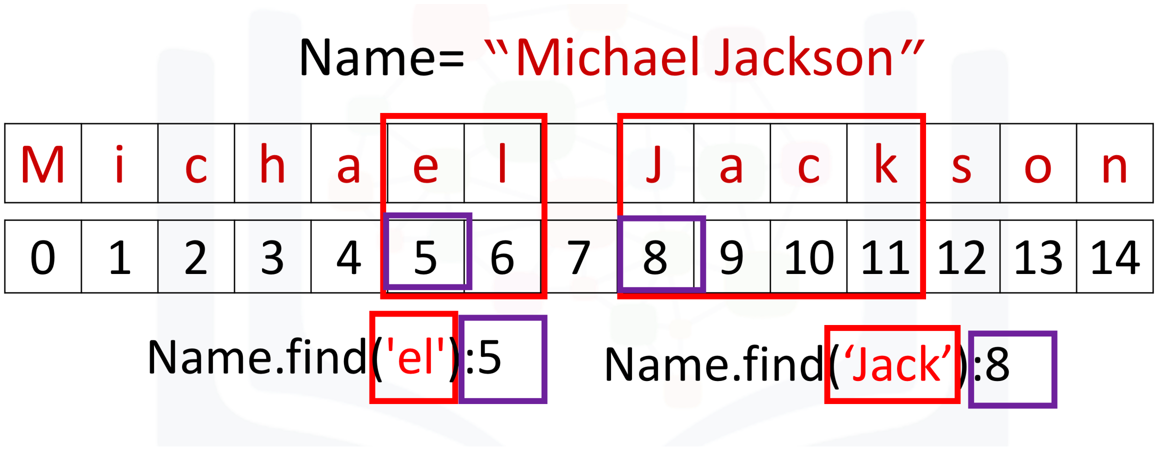
In [ ]:
```
# Convert all the characters in string to upper case

A = "Thriller is the sixth studio album"
print("before upper:", A)
B = A.upper()
print("After upper:", B)
C = A.title()
print("After upper:", C)
```

In [ ]:
```
# Convert all the characters in string to lower case

A = "Thriller Is The Sixth Studio Album"
print("before lower:", A)
B = A.lower()
print("After lower:", B)
```

The method `replace` replaces a segment of the string, i.e. a substring with a new string. We input the part of the string we would like to change. The second argument is what we would like to exchange the segment with, and the result is a new string with the segment changed:

In [ ]:
```
# Replace the old substring with the new target substring is the segment has been found i

A = "Michael Jackson is the best"
B = A.replace("Michael","Janet")
B
```

The method `find` finds a sub-string. The argument is the substring you would like to find, and the output is the first index of the sequence. We can find the sub-string `jack` or `el .`



In [ ]:
```
# Find the substring in the string. Only the index of the first elment of substring in st

Name = "Michael Jackson"
Name.find('e')
```

In [ ]:
```
# Find the substring in the string.
Name.find('Jack')
```

If the sub-string is not in the string then the output is a negative one. For example, the string 'Jasdfasdasdf' is not a substring:

In [ ]:
```
# If cannot find the substring in the string

Name.find('Jasdfasdasdf')
```

Use a stride value of 2 to print out every second character of the string `E`:

In [ ]:
```
# Write your code below and press Shift+Enter to execute
E = 'clocrkr1e1c1t'
E[::2]
```

Print out a backslash:

In [ ]:
```
# Write your code below and press Shift+Enter to execute
yourName = 'hello \\ chisom'
yourName
```

Consider the variable `G`, and find the first index of the sub-string `snow`:

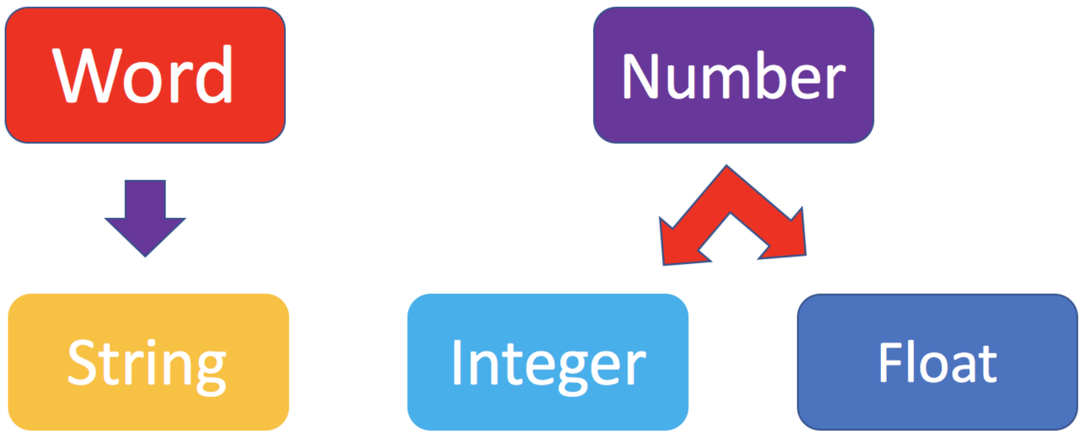
In [ ]:
```
# Write your code below and press Shift+Enter to execute

G = "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \
Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \
Everywhere that Mary went The lamb was sure to go"
G.find("snow")
```

In the variable `G`, replace the sub-string `Mary` with `Bob`:

```
In [ ]:
# Write your code below and press Shift+Enter to execute
G = "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \
Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \
Everywhere that Mary went The lamb was sure to go"
B = G.replace ("Mary", "Bob")
B
```

### Object Types

Python is an object-oriented language. There are many different types of objects in Python. Let's start with the most common object types: *strings*, *integers* and *floats*. Anytime you write words (text) in Python, you're using *character strings* (strings for short). The most common numbers, on the other hand, are *integers* (e.g. -1, 0, 100) and *floats*, which represent real numbers (e.g. 3.14, -42.0).



```
In [ ]:
# ??
11
```

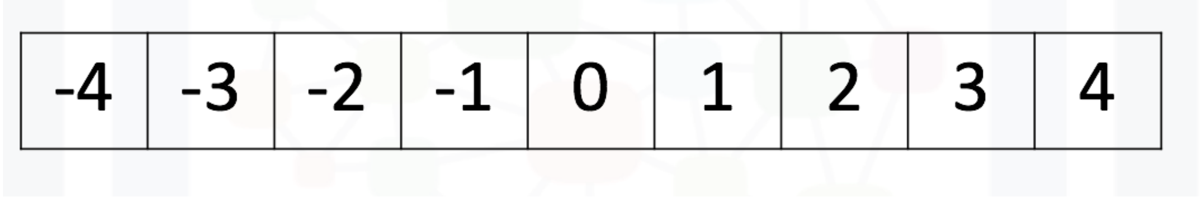
```
In [ ]:
# ??
2.14
```

```
In [ ]:
# ??
"Hello, Python 101!"
```

```
In [ ]:
# ??
True
```

Integers

Here are some examples of integers. Integers can be negative or positive numbers:



We can verify this is the case by using, you guessed it, the `type()` function:

In [ ]:
```python
# Print the type of -1
type(-1)
```

In [ ]:
```python
# Print the type of 4
type(4)
```

In [ ]:
```python
# Print the type of 10
type(10)
```

Floats

Floats represent real numbers; they are a superset of integer numbers but also include "numbers with decimals". There are some limitations when it comes to machines representing real numbers, but floating point numbers are a good representation in most cases. You can learn more about the specifics of floats for your runtime environment, by checking the value of `sys.float_info`. This will also tell you what's the largest and smallest number that can be represented with them.

Once again, can test some examples with the `type()` function:

In [ ]:
```python
# Print the type of 1.0
type(1.0) # Notice that 1 is an int, and 1.0 is a float
#import sys
#sys.float_info
```

In [ ]:
```python
# Print the type of 0.5
type(0.5)
```

In [ ]:
```python
# Print the type of 0.50
type(0.50)
```

In [ ]:
```python
# System settings about float type
import sys
sys.float_info
```

Booleans

Boolean is another important type in Python. An object of type *Boolean* can take on one of two values: `True` or `False`:

In [ ]:
```python
# Value true
True
```

Notice that the value `True` has an uppercase "T". The same is true for `False` (i.e. you must use the uppercase "F").

In [ ]:
```
# Value false
False
```

```
### Type Casting
```

In [ ]:
```
type(1.1)
```

In [ ]:
```
# Casting 1.1 to integer will result in loss of information
int(1.1)
```

Knowing the Object Types

You can get Python to tell you the type of an expression by using the built-in `type()` function. You'll notice that Python refers to integers as `int`, floats as `float`, and character strings as `str`.

In [ ]:
```
# Type of 12
type(12)
type(11)
```

In [ ]:
```
# Type of 2.14
type(2.14)
```

In [ ]:
```
# Type of "Hello, Python 101!"
type("Hello, Python 101!")
```

Converting from one object type to a different object type

You can change the type of the object in Python; this is called typecasting. For example, you can convert an *integer* into a *float* (e.g. 2 to 2.0).

Let's try it:

Converting integers to floats

In [ ]:
```
# Convert 2 to a float
float(2)
```

In [ ]:
```
# Convert integer 2 to a float and check its type
type(float(2))
```

When we convert an integer into a float, we don't really change the value (i.e., the significand) of the number. However, if we cast a float into an integer, we could potentially lose some information. For example, if we cast the float 1.1 to integer we will get 1 and lose the decimal information (i.e., 0.1):

Converting from strings to integers or floats

Sometimes, we can have a string that contains a number within it. If this is the case, we can cast that string that represents a number into an integer using `int()`:

In [ ]:
```
# Convert a string into an integer
int('1')
```

But if you try to do so with a string that is not a perfect match for a number, you'll get an error. Try the following:

In [ ]:
```
# Convert a string into an integer with error
int('1 or 2 people')
```

You can also convert strings containing floating point numbers into *float* objects:

In [ ]:
```
'1.2'
```

In [ ]:
```
type("1.2")
```

In [ ]:
```
# Convert the string "1.2" into a float
float('1.2')
```

In [ ]:
```
type(1.2)
```

[Tip:] Note that strings can be represented with single quotes (`'1.2'`) or double quotes (`"1.2"`), but you can't mix both (e.g., `"1.2'`).

Converting numbers to strings

If we can convert strings to numbers, it is only natural to assume that we can convert numbers to strings, right?

In [ ]:
```
type('1')
```

And there is no reason why we shouldn't be able to make floats into strings as well:

In [ ]:
```
# Type of True
type(True)
```

In [ ]:
```
# Type of False
type(False)
```

Converting booleans to other types

We can cast boolean objects to other data types. If we cast a boolean with a value of `True` to an integer or float we will get a one. If we cast a boolean with a value of `False` to an integer or float we will get a zero. Similarly, if we cast a 1 to a Boolean, you get a `True`. And if we cast a 0 to a Boolean we will get a `False`. Let's give it a try:

In [ ]:
```
# Convert True to int
int(True)
```

In [ ]:
```
# Covert False to int
int(False)
```

In [ ]:
```
# Convert 1 to boolean
bool(1)
```

In [ ]:
```
# Convert 0 to boolean
bool(0)
```

In [ ]:
```
# Convert True to float
float(True)
```

### Assignment Link

Now we would try out some practical examples with what we have learnt so far ! Let us try out this notebook

### After Thoughts ??

Is it easy so far and what are you already dreaming of??

About this Instructor:

</div>

ChisomLoius is very passionate about Data Analysis and Machine Learning and does lot of free lance teaching and learning. Holding a B.Eng. in Petroleum Engineering, my focused is leveraging the knowledge of Data Science and Machine Learning to help build solutions in Education and High Tech Security. I currently work as a Petrochemist.

#
M
Ir

Visit our website, or further enquire more informatio via our email.
