Study Unit 1

Introduction to Programming in Python

Introduction to Programming in Python Outline

- Definition of programming
- Various programming languages
- Anaconda distribution
- Jupyter Notebook and/or Jupyter Lab
- JupyterLab notebook Interface
- Expression, data type, and variable assignment

Study Unit Duration

This Study Unit requires a minimum of 4 hours' formal study time.

You may spend an additional 2-3 hours on revision.

Preamble

Programming is a way of instructing the computer to perform various tasks. These instructions are written in a language that the computer understands. Just like Ethiopians can understand Amharic, Somalia understand Somali, Kenyans and Ugandans understand Swahili, so is the case with computers. Computers understand instructions that are written in a specific syntax called a programming language.

In this study unit, you will be taught how to install Anaconda and how to navigate through Notebook. You will also get to know expressions, data types and how to perform basic arithmetical and logical operations.

Learning Outcomes of Study Unit 1

Upon completion of this study unit, you should be able to:

1.1 Describe various computer programming languages

1.2 Navigate through Anaconda distribution and work with Jupyter Lab Notebook

1.3 Perform basic arithmetic operations in Python

1.4 Assign values to variables and work with different Python data types

1.5 Work with comparison and logical operators

Terminologies, Acronyms and their Meaning

.ipynb	Jupyter notebook or Jupyter lab extension
.py	Python extension
IDE	Integrated Development environment (IDE)
R	R programming language
EDA	Exploratory Data Analysis

NaN	Not a Number
Np	Numpy
Pd	Pandas
Os	Operating system
AI	Artificial Intelligence
Int	Integer data type
Str	String data type
bool	Boolean data type

1.1 Introduction to Programming

Programming is a way of instructing the computer to perform various tasks. These instructions are written in a language that the computer understands. The instruction could be as simple as:

- Adding the population of Ethiopia and Sudan.
- What is the square root of 16?

Just like Ethiopians can understand Amharic, Somalia understand Somali, Kenyans and Ugandans understand Swahili, so is the case with computers. Computers understand instructions that are written in a specific syntax called a programming language.

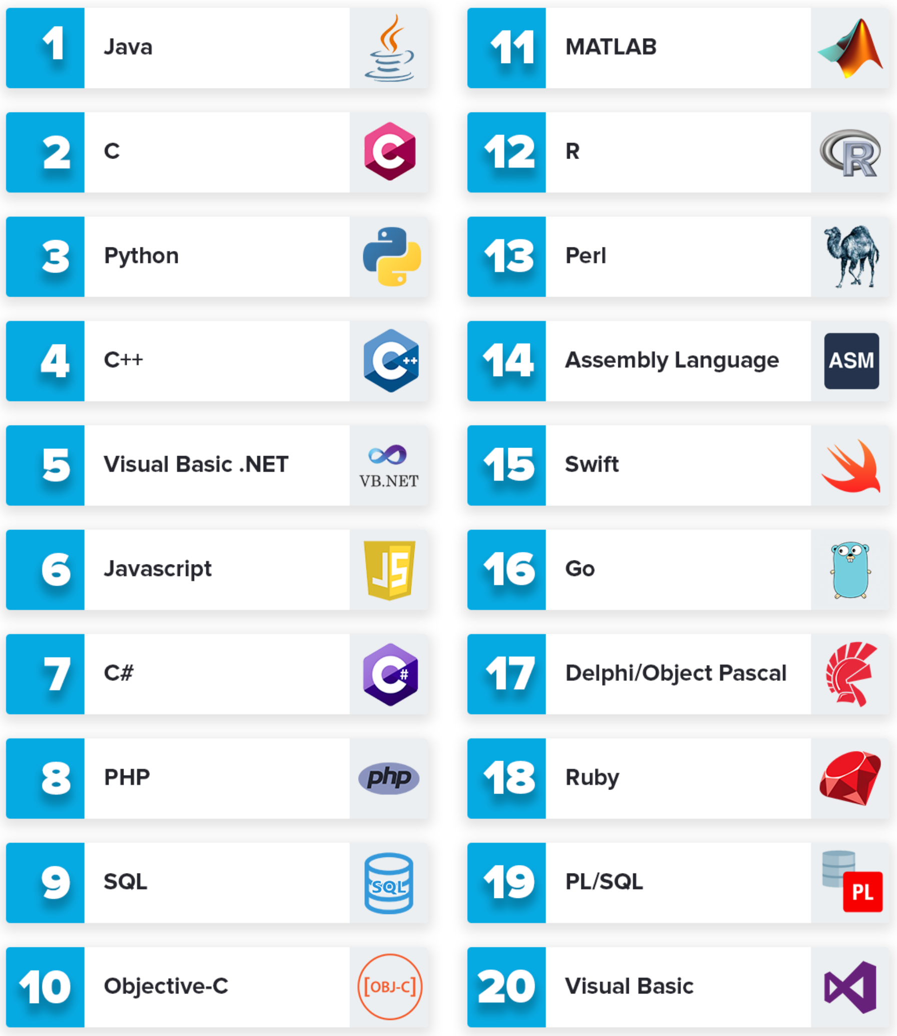
1.1.1 What is a Programming Language?

A programming language is the set of instructions through which we can interact with computers. This instruction(s) is given in form of syntax or codes by a computer programmer to execute certain tasks. Programmers use specialized languages to communicate with computers for a certain task.

Source: Study Point

1.1.2 Various programming languages

Some of the popular Programming languages are Python, C, C++, Java, JavaScript, and R. The figure below shows some of the programming languages that are available in the world today:



1	Java	11	MATLAB
2	C	12	R
3	Python	13	Perl
4	C++	14	Assembly Language
5	Visual Basic .NET	15	Swift
6	Javascript	16	Go
7	C#	17	Delphi/Object Pascal
8	PHP	18	Ruby
9	SQL	19	PL/SQL
10	Objective-C	20	Visual Basic

Source: Coding Dojo.

1.1.3 Why you should learn Computer Programming?

- Programming is fun
- Programming increases your critical thinking
- You can automate a task easily
- You can earn more money by coding
- The backbone of a Technology Company

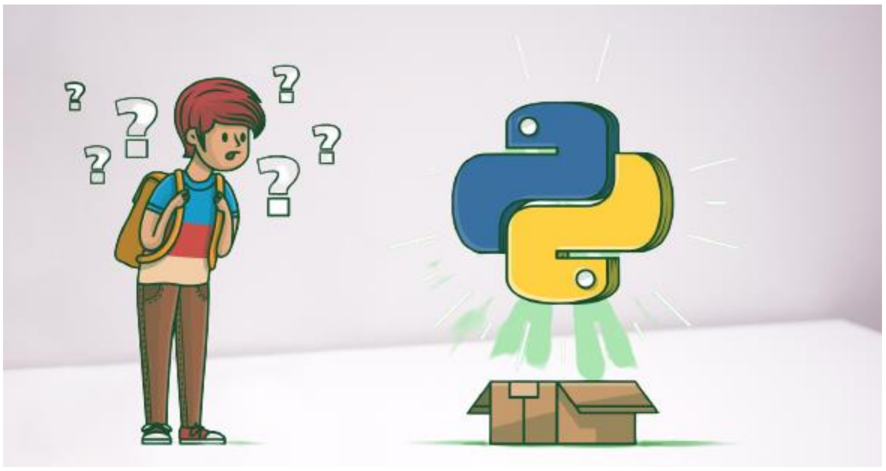
1.1.4 Programing Languages for Data Science

There are several programming languages such as Python, R, Scala, and Julia for data science and you that would soon become a data scientist (a data steward) should learn and master at least one language for data science project. This course will teach you Python programming language.

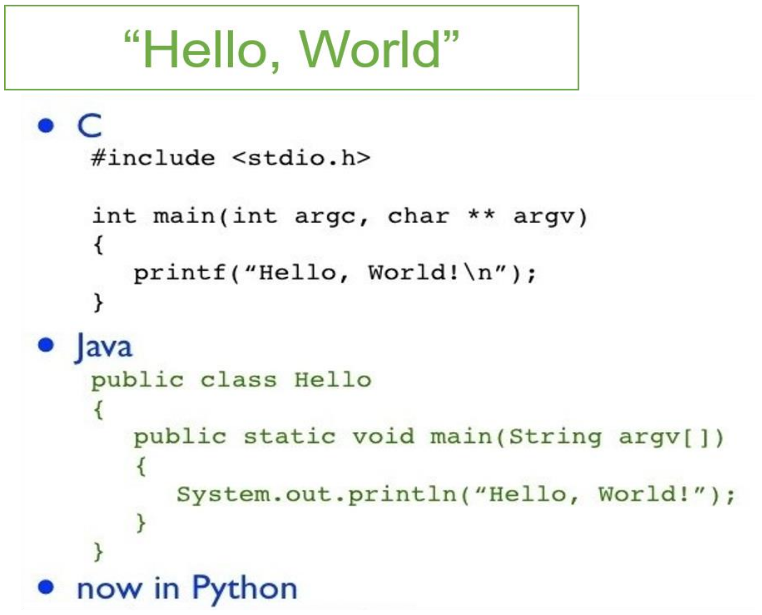
1.1.5 Python Programming language

Python is a powerful general-purpose programming language. It is an open-source and easy-to-use language that was created by Guido van Rossum in 1991. Python is used in data science, web development, and so on. Python is one of the most widely used data science programming language in the world today. Its simple easy-to-use syntax makes Python an excellent language to learn to program for beginners. Python was designed for readability, and has some similarities to the English language with influence from mathematics.

1.1.6 Why Python?



- It is easy to learn, and the code is readable.
- It is one of the most popular programming languages
- There is a lot of Open Source contribution on Python. i.e. people has made a lot of free contribution on python to make programming easier for others.
- Instructions are given in a shorter code compared to that of other programming languages. For example, the figure below shows how to print Hello World. Mere looking at the code, we would see that python’s syntax is shorter and easy to read.



“Hello, World”

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
   printf(“Hello, World!\n”);
}
```

- Java

```
public class Hello
{
   public static void main(String argv[])
   {
      System.out.println(“Hello, World!”);
   }
}
```

- now in Python

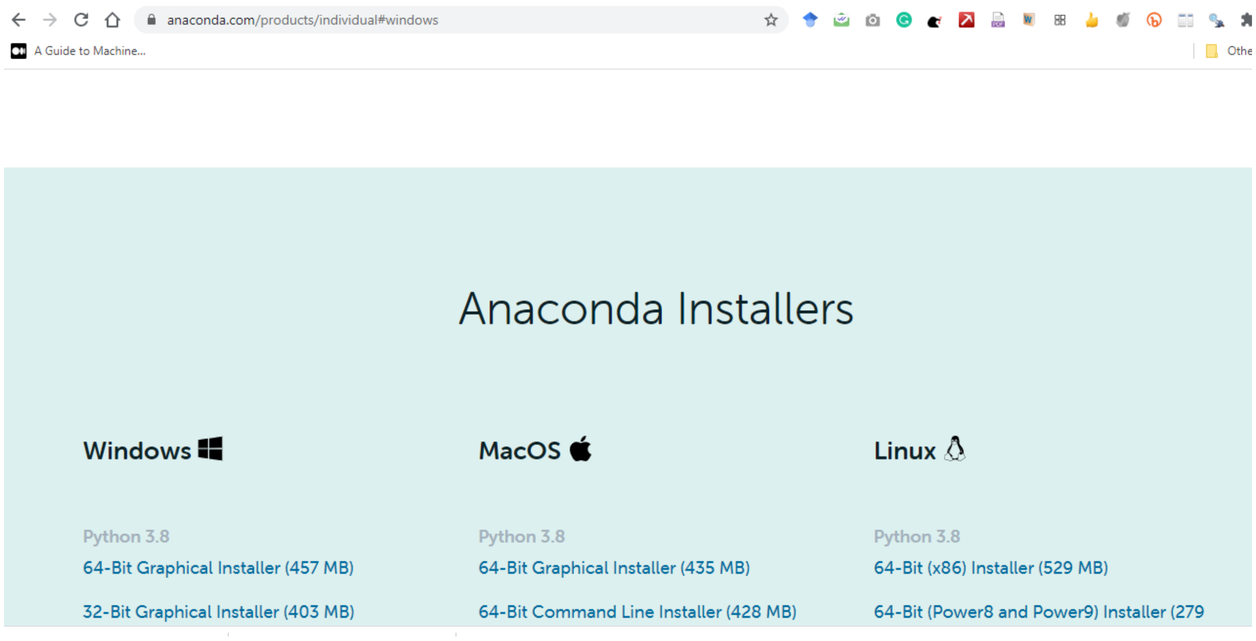
```
print(“Hello, World!”)
```

1.2 Introduction to Anaconda Distribution

Anaconda is a distribution of the Python for scientific computing, that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS.

To install Anaconda, please follow the following steps:

- Download Anaconda Navigator App using via this link https://www.anaconda.com/products/individual#windows, then click on Download



Anaconda Installers

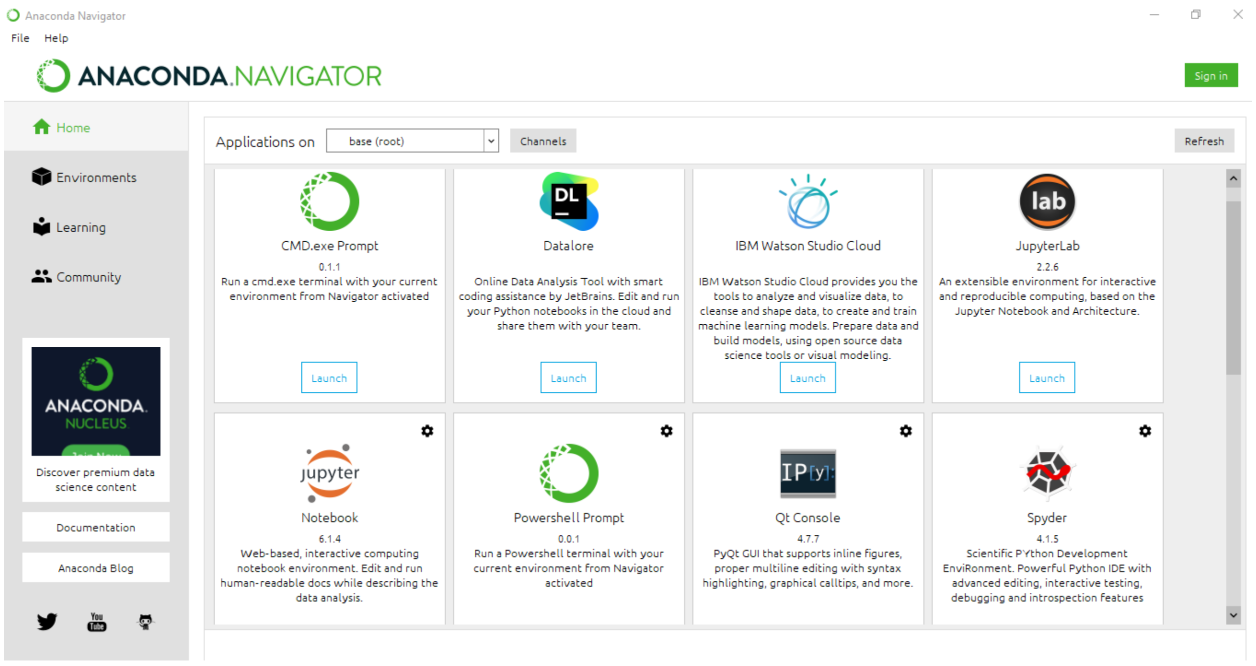
Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (457 MB)	64-Bit Graphical Installer (435 MB)	64-Bit (x86) Installer (529 MB)
32-Bit Graphical Installer (403 MB)	64-Bit Command Line Installer (428 MB)	64-Bit (Power8 and Power9) Installer (279

2. The output above is what you will see after clicking the link, you can select the operating system you are using. In this case, I will click on Windows (64-Bit Graphical Installer (457 MB)) science I am using a windows operating system with 64-bit.

3. Then your Anaconda will start downloading. After download, then you can install it.

1.2.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications without using command-line commands.



Discover premium data science content

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

JupyterLab
2.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

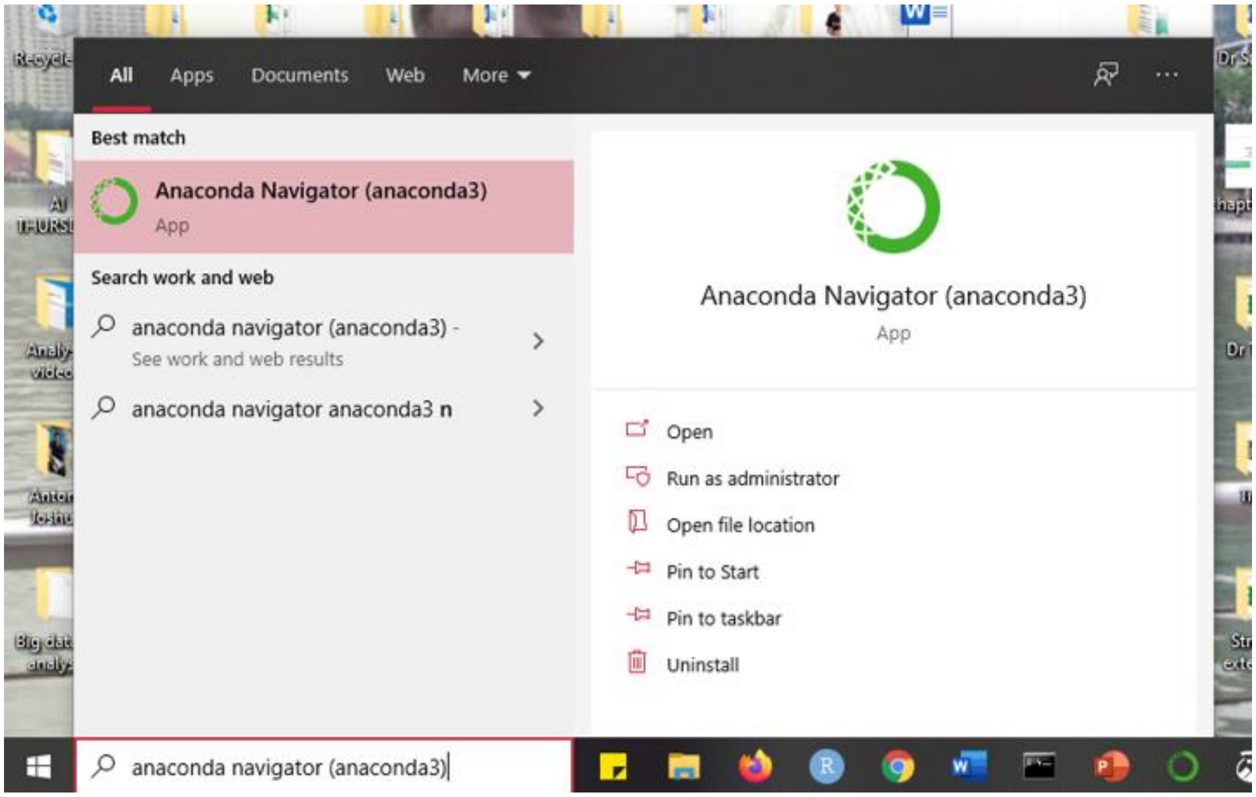
Notebook
6.1.4
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console
4.7.7
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

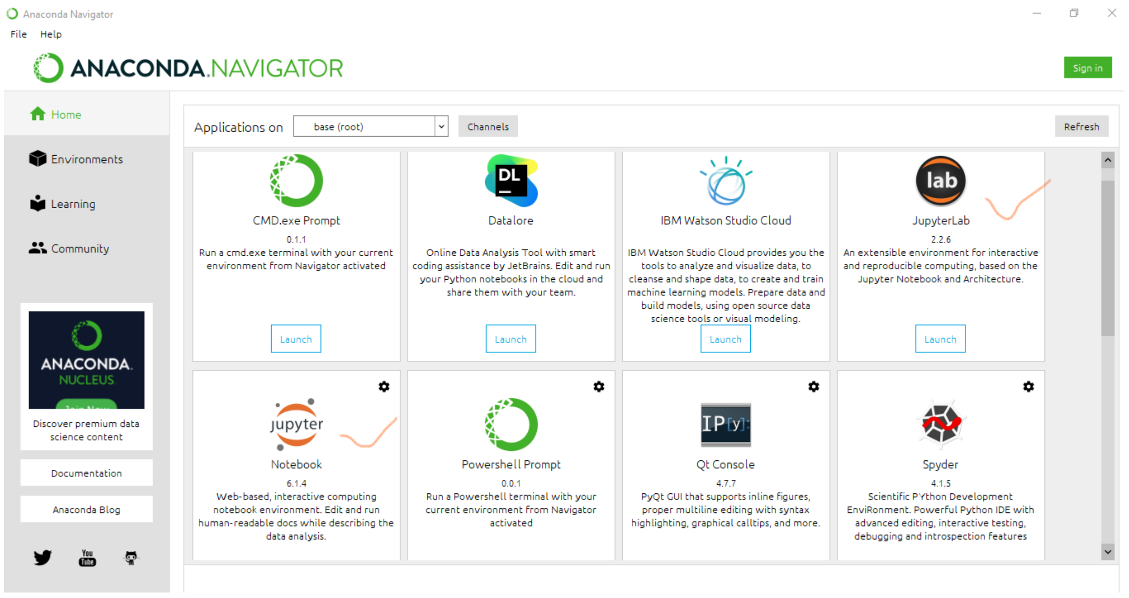
Spyder
4.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

You can launch Anaconda Navigator by typing Anaconda Navigator on the search bar, then double click on the Anaconda Navigator (Anconda3)


All
Apps
Documents
Web
More
Best match
Anaconda Navigator (anaconda3)
App
Search work and web
anaconda navigator (anaconda3) -
See work and web results
anaconda navigator anaconda3 n
Anaconda Navigator (anaconda3)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall
anaconda navigator (anaconda3)

1.2.2 Introduction to Jupyter Notebook and/or Jupyter Lab

Among the applications present in the Anaconda are JupyterLab and Jupyter Notebook



Discover premium data science content

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

JupyterLab
2.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.1.4
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

Qt Console
4.7.7
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
4.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

JupyterLab and/or Jupyter notebook is a web application that allows you to create and share documents that contain:

- live code (e.g. Python code)
- explanatory text (written in markdown syntax)
- visualizations

This notebook supports Python and other programming languages. It provides an environment for data science enthusiasts who just started out their career in data science field. This IDE supports markdown and enables you to add HTML components, images and videos.

1.2.3 Difference between JupyterLab and Jupyter Notebook

Both JupyterLab and Jupyter Notebook are browser compatible interactive python with the extension .ipynb where you can divide the various portions of the code into various individually executable cells for the sake of better readability.

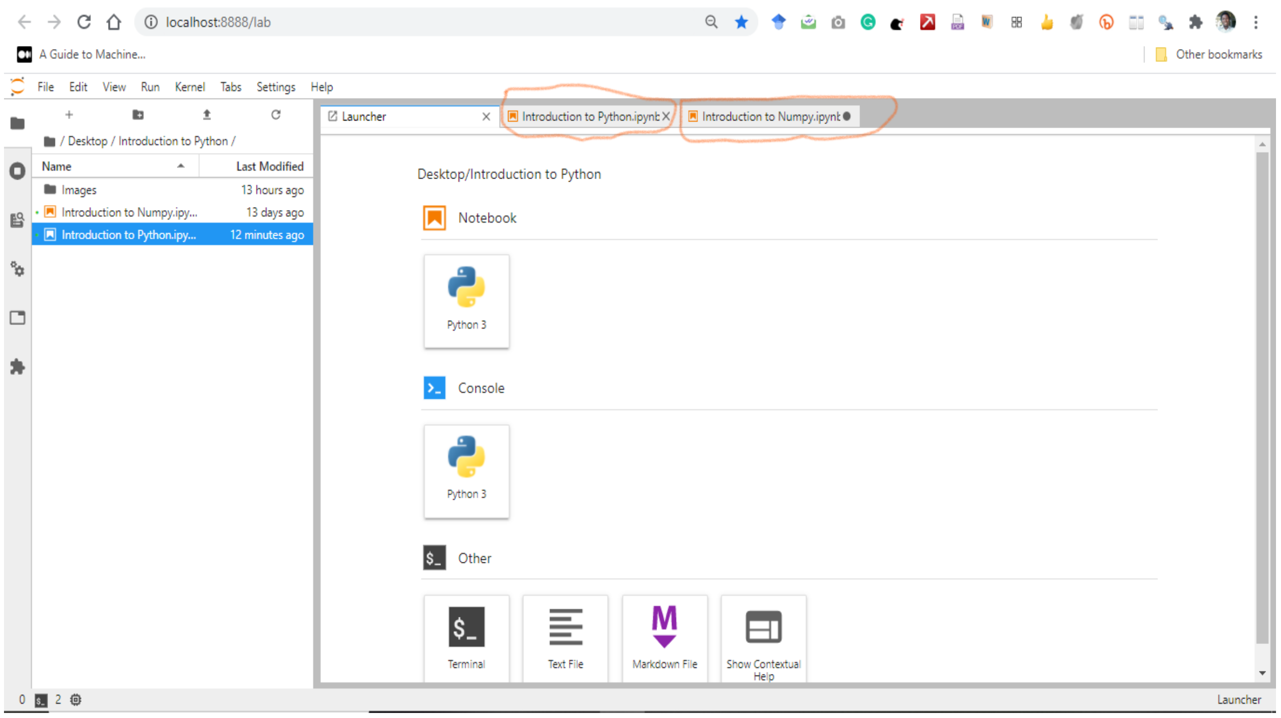
In Jupyterlab, you can create a Python script directly (.py), create notebook (.ipynb), open terminal etc. Jupyter Notebook allows for only notebook (.ipynb) files while providing you the choice to choose between the two versions of Python i.e. python 2 or python 3.

JupyterLab runs in a single tab, with sub-tabs displayed within that one tab while Jupyter Notebook opens new notebooks in new tabs.

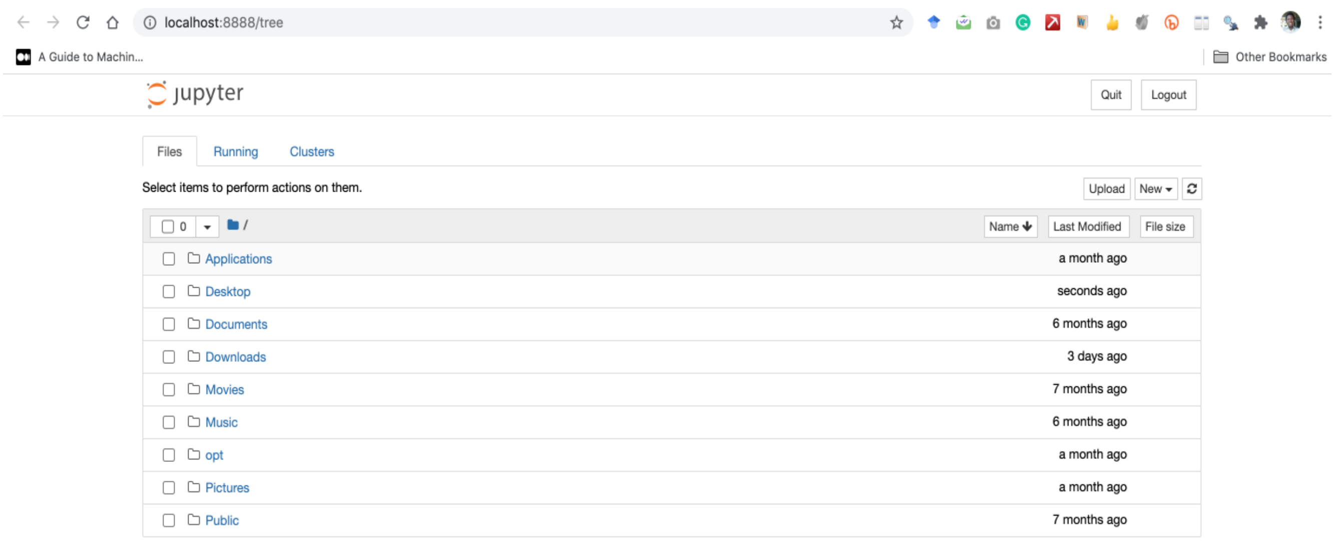
JupyterLab can open multiple notebooks (.ipynb) files inside a single browser tab. Whereas, Jupyter Notebook will create new tab to open new notebook (.ipynb) files every time. Hovering between various tabs of browser is tedious.

1.2.4 JupyterLab interface

Many tabs in JupyterLab notebook

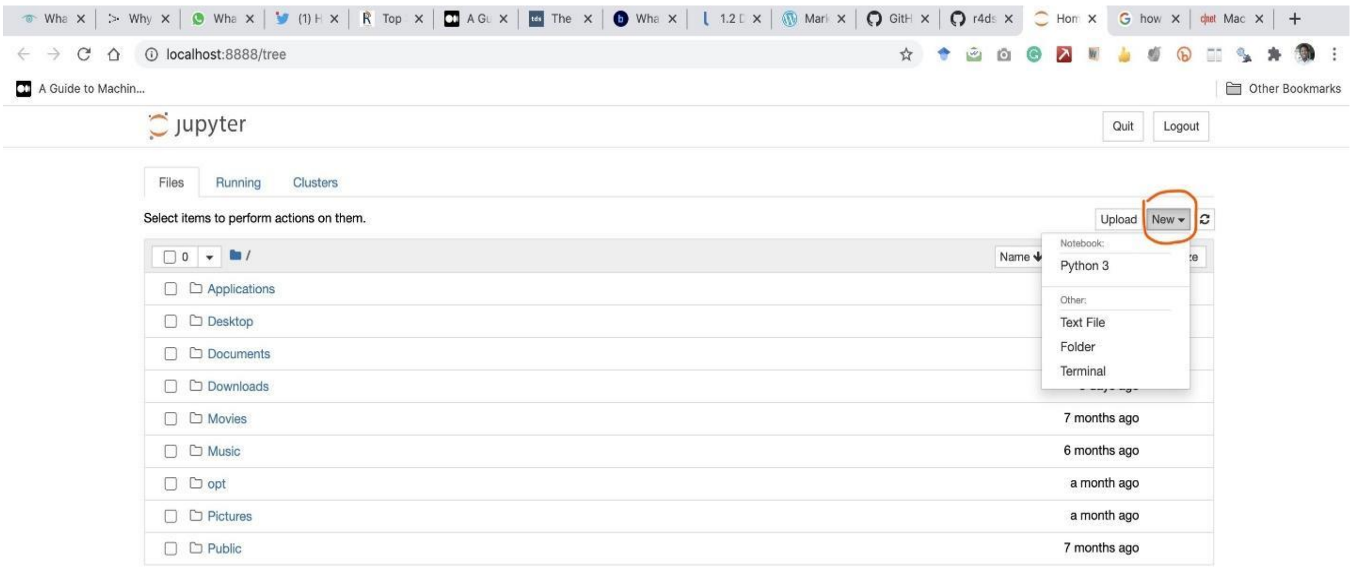


1.2.5 Jupyter Notebook interface



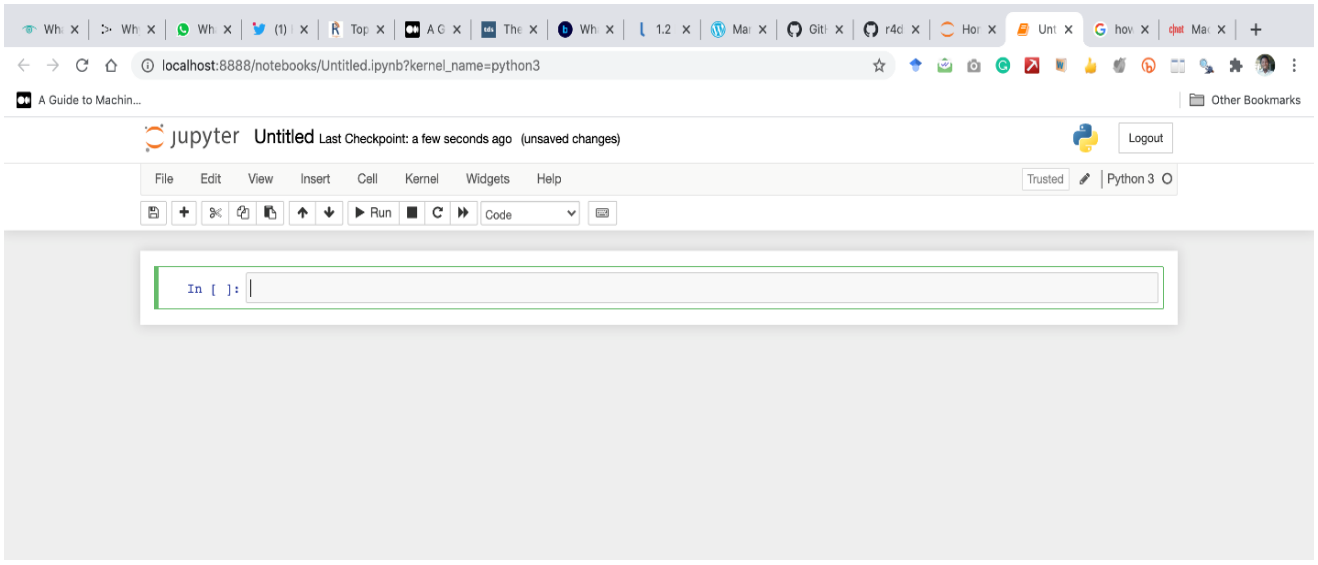
Select items to perform actions on them.

Applications	a month ago
Desktop	seconds ago
Documents	6 months ago
Downloads	3 days ago
Movies	7 months ago
Music	6 months ago
opt	a month ago
Pictures	a month ago
Public	7 months ago



Select items to perform actions on them.

Applications	
Desktop	
Documents	
Downloads	
Movies	7 months ago
Music	6 months ago
opt	a month ago
Pictures	a month ago
Public	7 months ago


localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [ ]:

1.2.6 How to launch JupyterLab and/or Jupyter Notebook

Now that you have understood JupyterLab and/or Jupyter Notebook, let's see how to launch it.

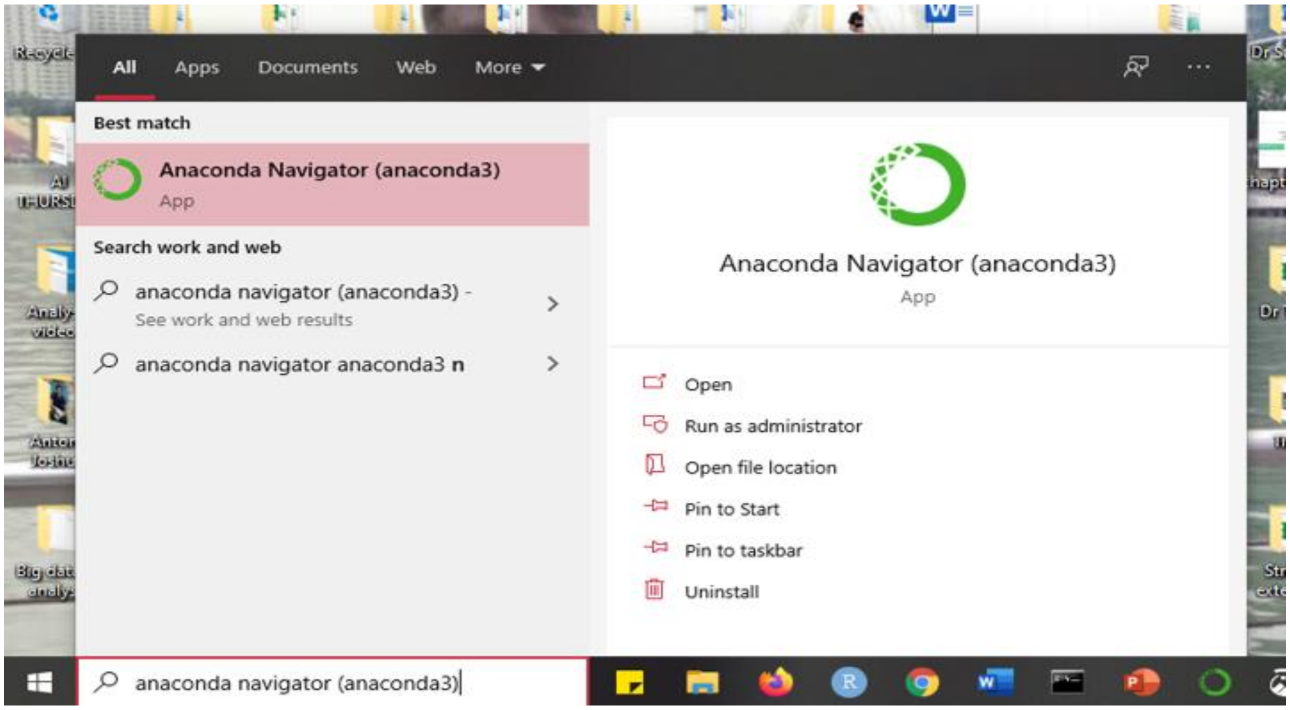
There are two ways of launching JupyterLab and/or Jupyter Notebook.

1. Anaconda Navigator

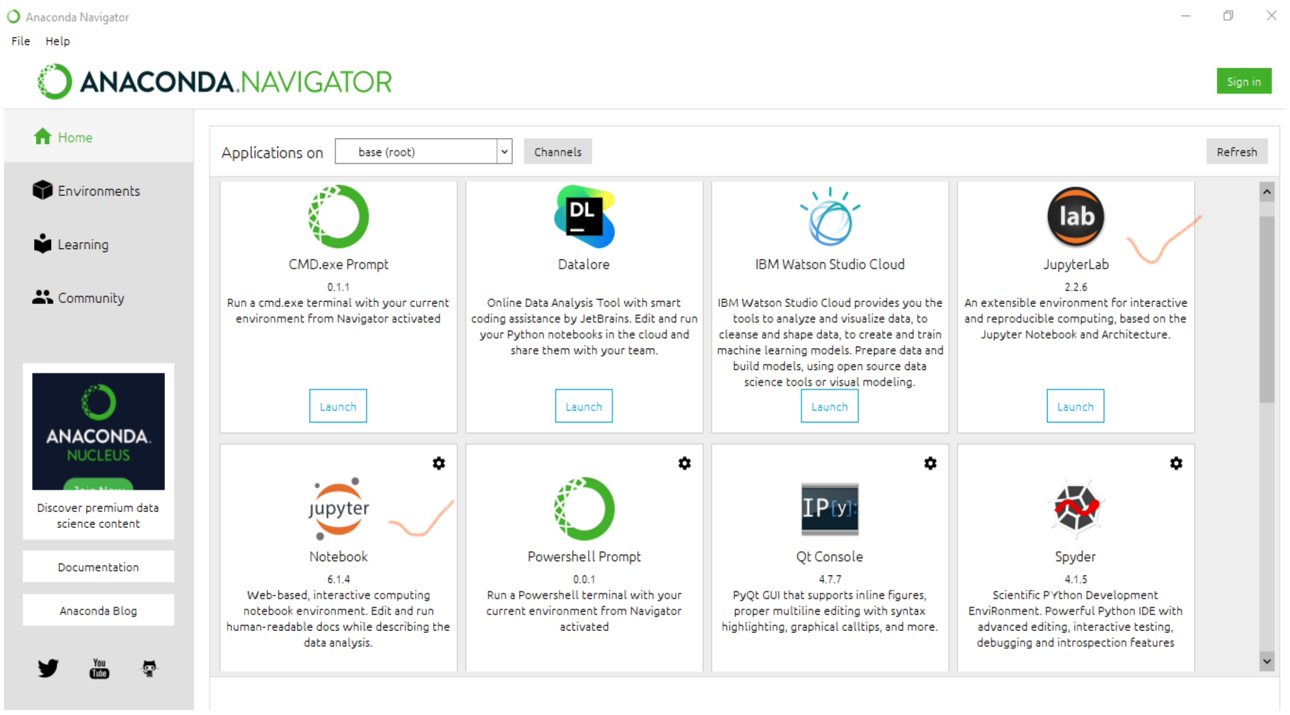
2. Through terminal or Anaconda Prompt

Method 1: Anaconda Navigator

You can launch Anaconda Navigator by typing Anaconda Navigator on the search bar

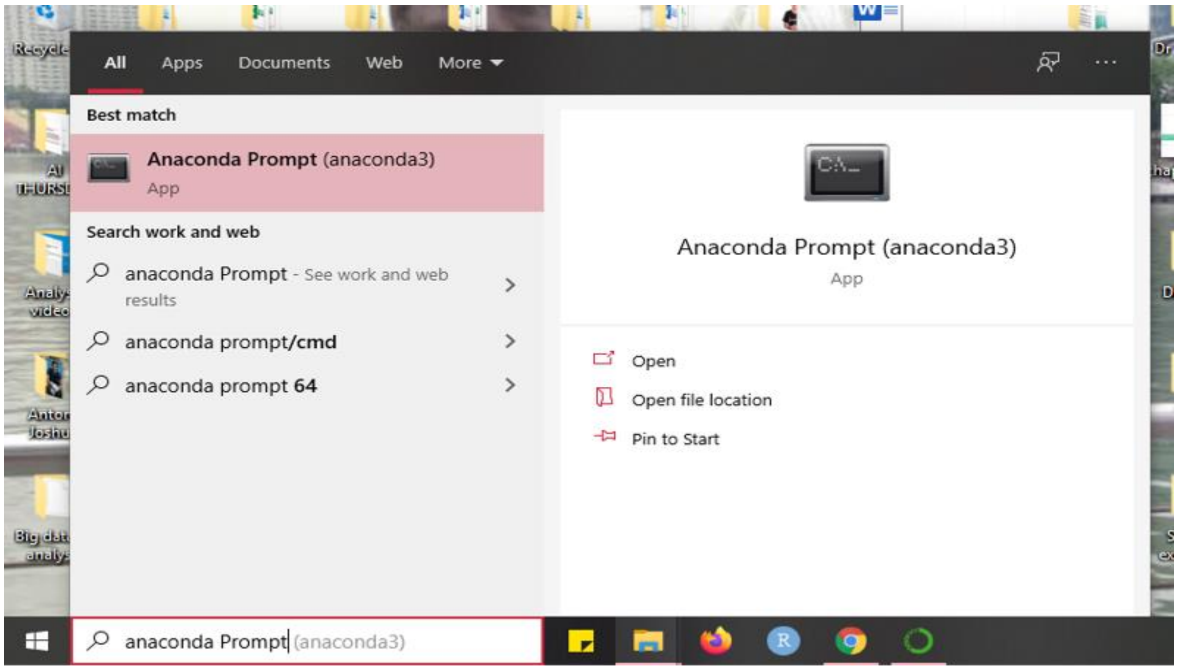

All
Apps
Documents
Web
More
Best match
Anaconda Navigator (anaconda3)
App
Search work and web
anaconda navigator (anaconda3) - See work and web results
anaconda navigator anaconda3 n
Anaconda Navigator (anaconda3)
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall
anaconda navigator (anaconda3)

Locate JupyterLab or Notebook and then click on Launch

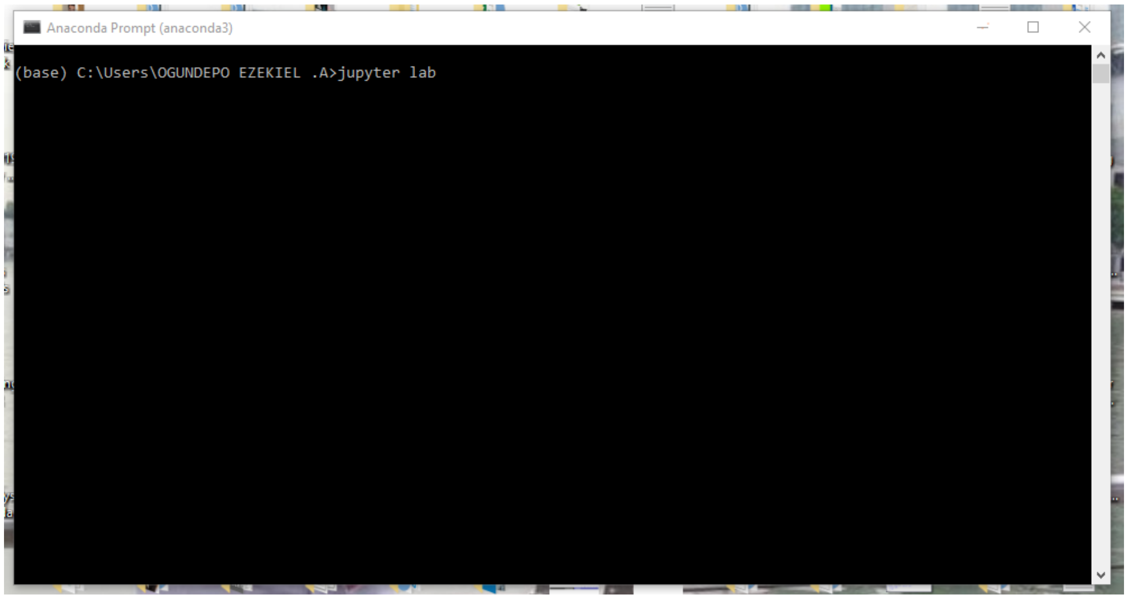

Anaconda Navigator
File
Help
ANACONDA.NAVIGATOR
Sign in
Home
Environments
Learning
Community
ANACONDA NUCLEUS
Discover premium data science content
Documentation
Anaconda Blog
Applications on
base (root)
Channels
Refresh
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
Datalore
Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.
Launch
IBM Watson Studio Cloud
IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.
Launch
JupyterLab
2.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.1.4
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Qt Console
4.7.7
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Spyder
4.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Method 2: Through terminal or Anaconda Prompt

We can also launch JupyterLab and/or Jupyter Notebook through the terminal or Anaconda Prompt. To do that, type Anaconda Prompt on the search bar and double click on Anaconda Prompt (Anaconda3)



A command prompt window will come up, then type jupyter lab and press enter (or return) from your keyboard.



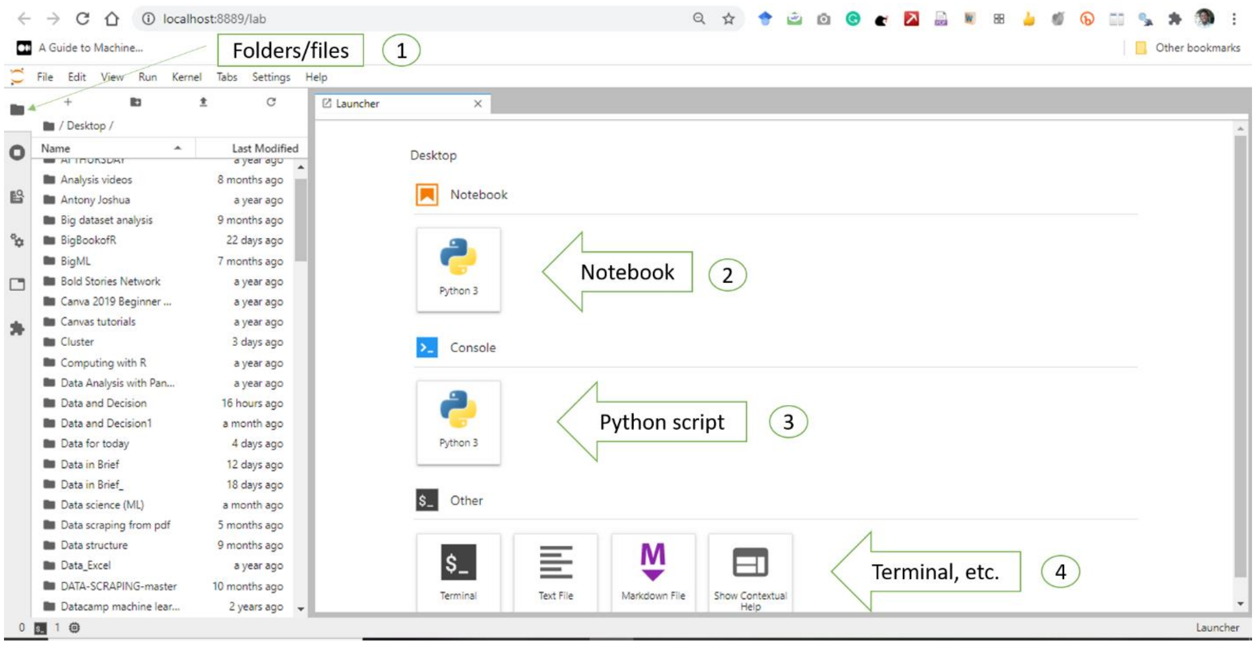
```
(base) C:\Users\OGUNDEPO EZEKIEL .A>jupyter lab
```

Important note

Note: For Jupyter Notebook you will type jupyter notebook

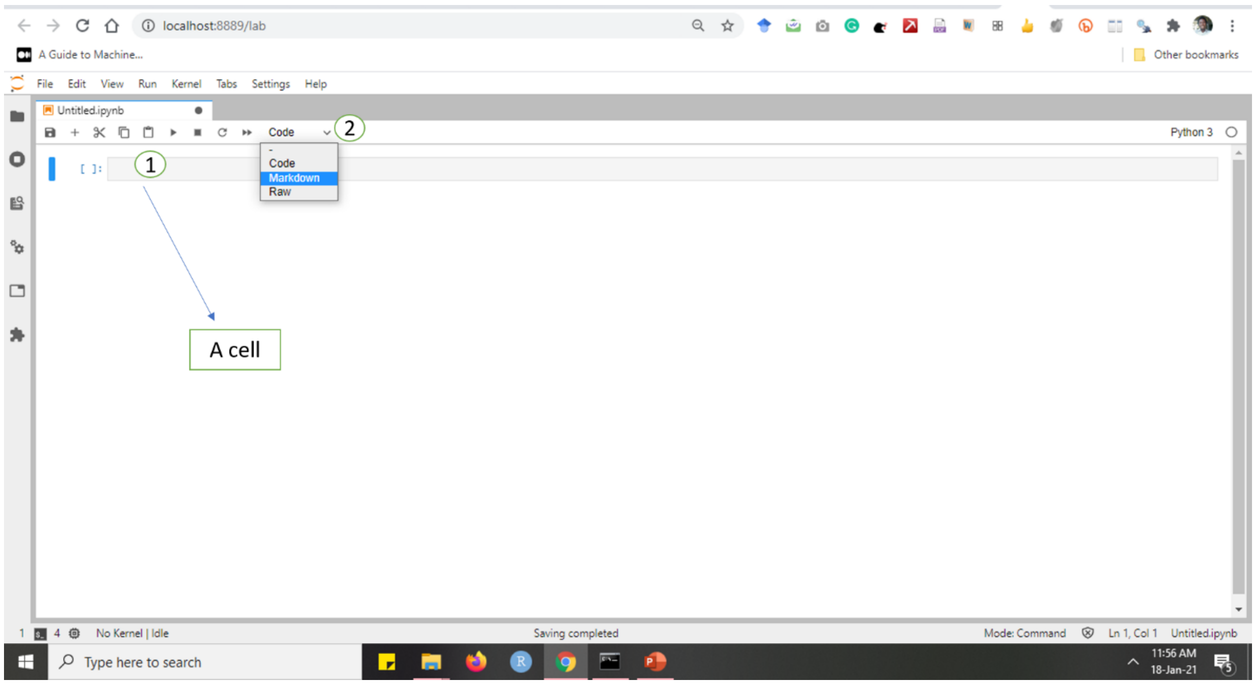
1.2.7 How to navigate through JupyterLab notebook Interface

After launching JupyterLab, you will see interface that looks like this:

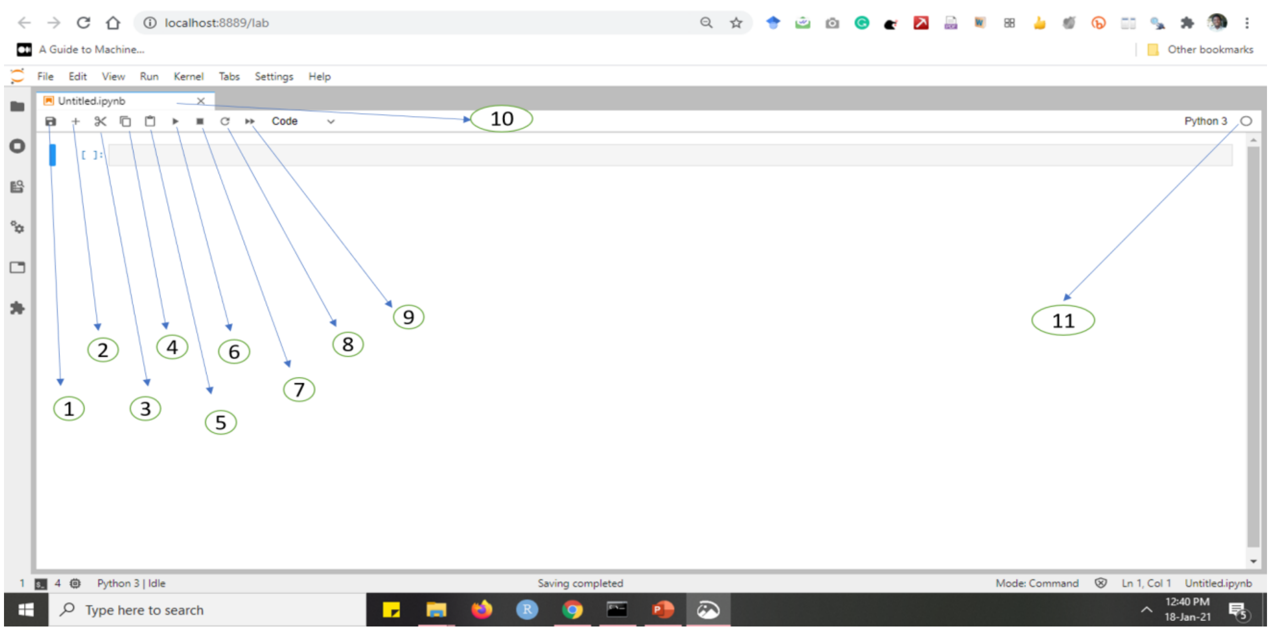


- 1 shows where you can select a folder or file that may contain your already worked on Jupyter Notebook (.ipynb).
- 2 shows where you can create a new notebook
- 3 shows where to create a new script
- 4 we can install some new package(s) with the terminal

Let's start by clicking on Python 3 to create a new notebook



- 1 a new cell
- 2 shows where to change the type of cell. We have Code (a pure python code), Markdown (markdown texts) or Raw (run it as it is)



- 1 to save the notebook you are currently working on

- 2 to create a new cell
- 3 to cut or delete a cell
- 4 to copy a cell (which you may want to paste in another cell)
- 5 to paste a cell that was initially copied
- 6 to run the selected cell
- 7 to stop the kernel (to stop Python from working)
- 8 to restart the kernel
- 9 to restart the kernel and rerun the whole notebook
- 10 name of the notebook you are working on
- 11 status of the notebook you are working on

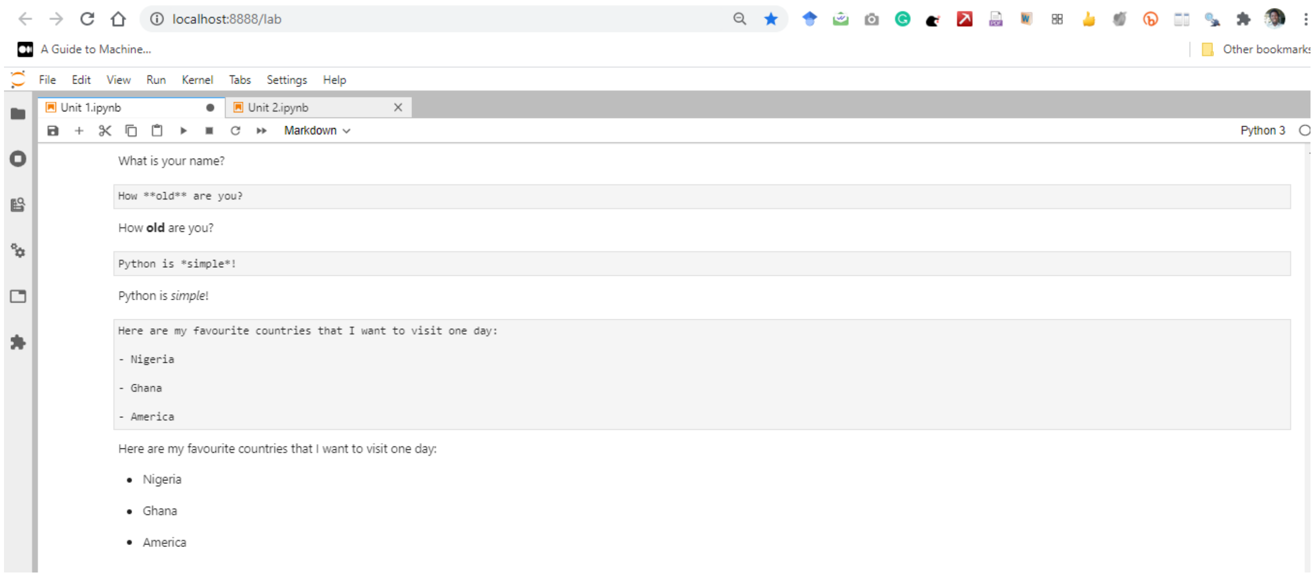
Working with JupyterLab Notebook

In this course, I will prefer you use JupyterLab notebook because of the aforementioned advantages that were discussed above. If you have already closed your JupyterLab notebook, you can launch it again! In this case, I will use Anaconda prompt instead of Anaconda Navigator. Do you remember how to launch Jupyterlab notebook? If no, visit click here.

Notebook cell

A cell in Jupyter notebook can accept python code or markdown. For example, the cell that contains what is your name? is a markdown. In a markdown, you can write the full explanatory of what the code does with texts mixed with simple text formatting such as bold face, italic, bulleted list, etc.

Example of cells in Markdown



```
What is your name?

How **old** are you?
```

How **old** are you?

```
Python is *simple*!
```

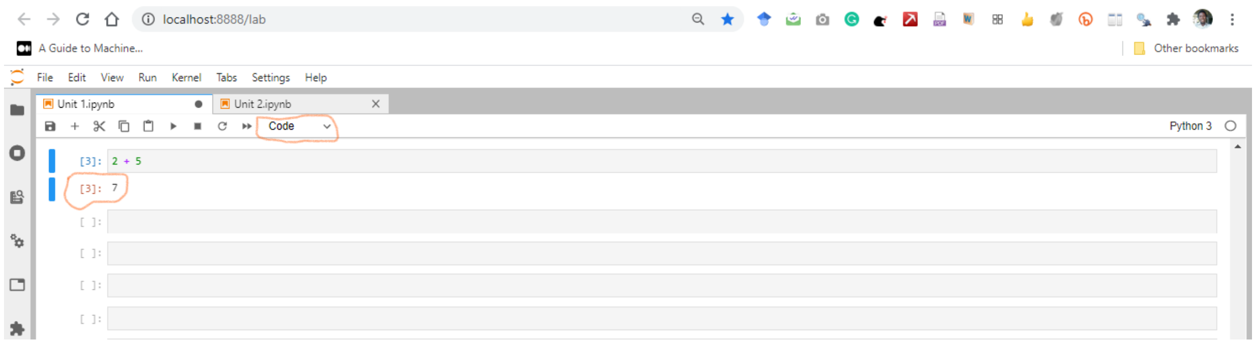
Python is *simple*!

```
Here are my favourite countries that I want to visit one day:
- Nigeria
- Ghana
- America
```

Here are my favourite countries that I want to visit one day:

- Nigeria
- Ghana
- America

The cell that contains 2 + 5 is a Python code and you can easily see the result (7) immediately below the cell.



```
[3]: 2 + 5
[3]: 7
[ ]:
[ ]:
[ ]:
[ ]:
```

Additional resources

For more resources in this section please consider the following:

- http://bit.ly/why-jupyter-lab
- https://www.youtube.com/watch?v=7wfPqAyYADY
- https://www.datacamp.com/community/tutorials/markdown-in-jupyter-notebook
- https://realpython.com/jupyter-notebook-introduction/

1.3 Expressions and Basic Arithmetic Operations in Python

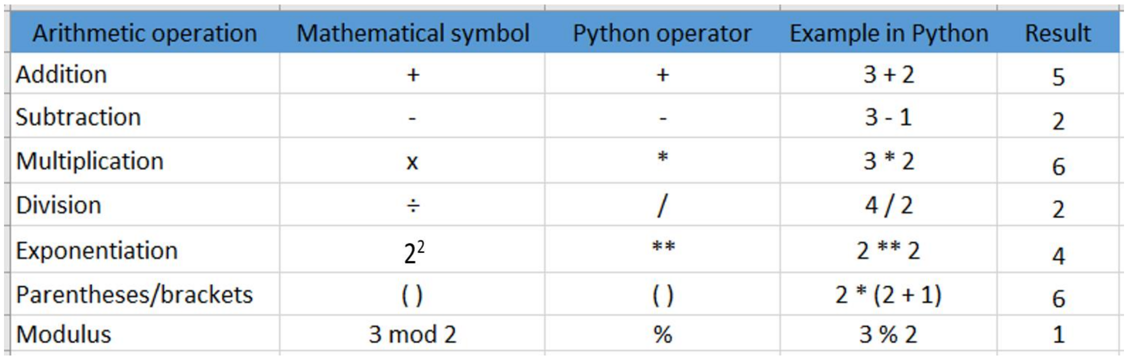
1.3.1 Python as a Calculator

In its most basic form, Python can be used as a calculator to perform basic arithmetic operations.

Addition(+)
Floor Division(//)
Subtraction(-)
Division(/)
Exponentiation (**)
Multiplication(*)
Modulus(%)

Consider the following arithmetic operations:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation (or Powers)
- Modulo



Arithmetic operation	Mathematical symbol	Python operator	Example in Python	Result
Addition	+	+	3 + 2	5
Subtraction	-	-	3 - 1	2
Multiplication	x	*	3 * 2	6
Division	÷	/	4 / 2	2
Exponentiation	2^2	**	2 ** 2	4
Parentheses/brackets	( )	( )	2 * (2 + 1)	6
Modulus	3 mod 2	%	3 % 2	1

+, -, *, /, **, and % are called arithmetic operators.

1.3.2 Expressions

Python is well able to perform the well know mathematical operations. An expression is an operation or a set of operations in python that can evaluated to determine its value i.e. **1 + 2** is an expression and the result **3** is a value.

Addition

Example 1

Add 8 and 2 together i.e. 8 + 2

```
8 + 2
```

```
10
```

Example 2

Calculate 1 + 3 + 2

```
1 + 3 + 2
```

```
6
```

Example 3

Programming in Python (CS2 1) class consists of students from seven (7) different countries. 7 are from Sudan, 15 are from Kenya, 10 are from Ethiopia, 18 are from South Sudan, 6 are from Sudan, 11 are Somalia while the remaining 14 are from Uganda. How many students are in the class altogether?

```python
7 + 15 + 10 + 18 + 6 + 11 + 14
```

```
81
```

Subtraction

Example 1

Calculate *8 - 5*

```python
8 - 5
```

```
3
```

Example 2

Calculate *20 - 15*

```python
20-15
```

```
5
```

Example 3

There are 30 sweets in a box, Zula picked 6 sweets from the box, Jamal picked 8 sweets, Adam picked 5, while Sifa picked 4. How many sweet is remaining in the box.

```python
30 - 6 - 8 - 5 -4
```

```
7
```

There are 7 sweets that are still remaining in the box.

Multiplication

Example 1

What is **2** × **3**?

```
2 * 3
```

6

Example 2

What is **5** × **4**?

```
5 * 4
```

20

Example 3

What is **−2** × **4** ?

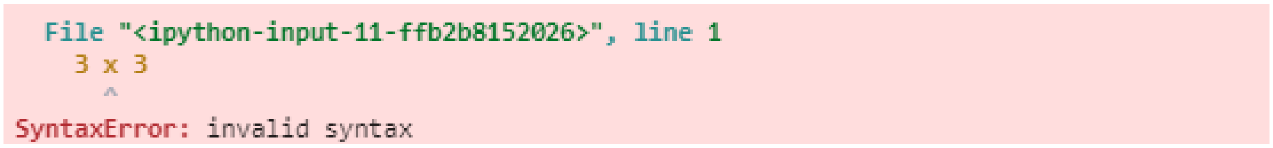
```
-2 * 4
```

-8

Example 4

What is **3** × **3** ?

```
3 x 3
```



```
  File "<ipython-input-11-ffb2b8152026>", line 1
    3 x 3
      ^
SyntaxError: invalid syntax
```

It is quite common in programming to make mistakes, even for programmers with years of experience. For most of the part, python will tell you where you have made a mistake by providing an error message. It is important to read the message and understand where you have made a mistake and resolve it accordingly.

In this case, our error is where we have use × instead of *. Therefore, we corrected the error as shown in the cell below:

```
3 * 3
```

```
9
```

Division

Example 1

Calculate 10 / 5

```
10 / 5
```

```
2.0
```

Example 2

Calculate 200 / 20

```
200 / 20
```

```
10.0
```

Example 3

Calculate 11 / 2

```
11 / 2
```

```
5.5
```

Exponentiation (or Powers)

Example 1

Calculate $\mathbf{4^2}$

```python
4 ** 2
```

```
16
```

Example 2

Calculate $\mathbf{3^2}$

```python
3 ** 2
```

```
9
```

Example 3

Calculate $\mathbf{9^2}$

```python
9 ** 2
```

```
81
```

Modulus

The modulo (or "modulus" or "mod") is the remainder after dividing one number by another. For example, $9 \bmod 2 = 1$. Because $9/2 = 4$ with a remainder of 1. In mathematics, we write that as $9 \bmod 2 = 1$ and in Python we write it as $9 \% 2 = 1$

Example 1

Calculate 9 mod 2

```python
9 % 2
```

```
1
```

Example 2

Calculate **11** % **3**

```
11 % 3
```

```
2
```

That is, 2 is the remainder when you divide 11 by 3

<u>Example 3</u>

Calculate **21** % **4**

```
21 % 4
```

```
1
```

<u>Example 4</u>

Calculate 15 % 6

```
15 % 6
```

```
3
```

The function divmod() in Python can be used to get the quotient and the remainder when dividing a number by another number. A quotient in mathematics is the actual result of the division of two numbers.

Example 1

When you divide 5 by 2, the actual result is 2 and the remainder is 1

```
divmod(5, 2)
```

```
(2, 1)
```

The quotient is 2 and the remainder is 1

Class activity 1 (Pilot Question 1)

What is the result of $\boldsymbol{divmod(18,4)}$

Pilot Answer 1

(4, 2)

The floor division (//)

The floor division (//) rounds the result down to the nearest whole number

Example 1

This is a normal division

```
15 / 2
```

```
7.5
```

Example 2

This is a floor division

```
15 // 2
```

```
7
```

Class activity 2 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Calculate the following in Python:

- 2 + 6 -12
- 4 * 3 - 8
- 81/6
- 9 mod 2
- 16 % 3
- $\mathbf{2^3}$
- divmod(17, 5)
- 11 // 5

Parentheses or brackets

Parentheses are used to denote grouping of operation in mathematics. It denotes modifications to normal order of operations. Remember BODMAS in mathematics? We shall use BEDMAS in programming:

B - Bracket

E - Exponentiation

D - Division

M - Multiplication

A - Addition

S - Subtraction

In an expression like $\mathbf{3 \times (2 + 3)}$, the part of the expression within the parentheses, $\mathbf{(2 + 3) = 5}$, is evaluated first, and then this result is used in the rest of the expression i.e. $\mathbf{3 \times 5 = 15}$.

Example 1

```
3 * (2 + 3)
```

15

Example 2

$\mathbf{2^2 + 1}$

```
2**2 + 1
```

5

Example 3

$\mathbf{(3 + 2) \times (6 - 4)}$

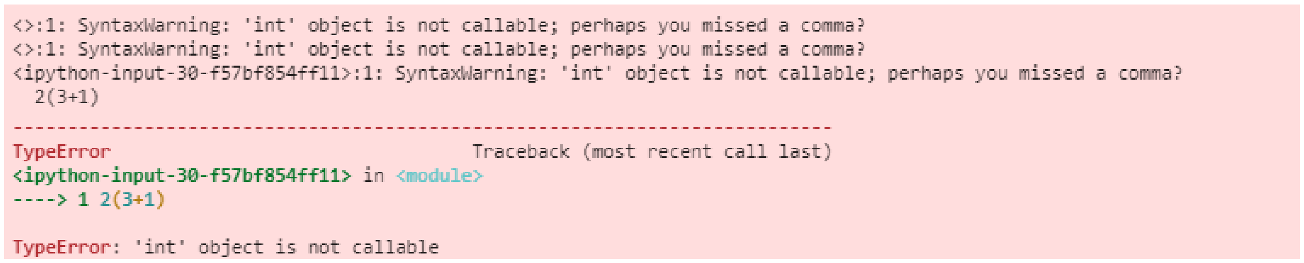
```
(3 + 2) * (6 - 4)
```

10

Example 4

Use python to evaluate $\mathbf{2(3 + 1)}$

```
2(3+1)
```



```
<>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<ipython-input-30-f57bf854ff11>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
  2(3+1)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-f57bf854ff11> in <module>
----> 1 2(3+1)

TypeError: 'int' object is not callable
```

This throws an error because we need to add an operator (*) before the parentheses or brackets.

```
2 * (3 + 1)
```

8

There must be an operator $(+, -, *, /)$ before and/or after the parentheses.

<u>Example 5</u>

What is the value of $\mathbf{3} \times (\mathbf{3} + \mathbf{2})$?

```
3 * (3 + 2)
```

15

<u>Example 6</u>

Calculate $(\mathbf{8} - \mathbf{7}) \times \mathbf{2}$

```
(8-7) * 2
```

2

<u>Example 7</u>

What is the value of (6 - 4) + 2?

```
(6 - 4) + 2
```

```
4
```

Example 8

What is the value of $(\mathbf{3} + \mathbf{2}) \times (\mathbf{6} - \mathbf{4}) + \mathbf{2}$?

```
(3 + 2) * (6 - 4) + 2
```

```
12
```

Comment in Python

Adding comments to your code is a common practice in all programming languages.

This is important because:

1. It makes you remember a line of code you have written after leaving it for a while
2. It makes other who want to check your code to understand it line by line
3. It is good for documentation purpose

In order to add comments to your codes, you will use the # sign before writing your comments.

Comments are not run as Python code, so they will not influence your result. Python ignores any anything that has # sign.

Example 1

```
# This adds 2 and 8 together. The result will be 10

2 + 8
```

```
10
```

As you can see, *This adds 2 and 8 together. The result will be 10* is our comment and we do that by using # sign.

Example 2

```
# You will know how to import packages in Python later

# to get square root of a number, we will import numpy package and then use numpy.sqrt() function

import numpy

numpy.sqrt(4)
```

```
2.0
```

As you can see, we use # to tell you what you are about to learn and how to get the square root of a number by using a comment.

1.3.3 Comparison Operators

We can compare two or more values by using comparison operators. These types of operators compare one value to another based on a condition. The result can either be *True* or *False* known as Boolean (bool). The following are the most common comparison operators in python:

- double equal to = =
- not equal to !=
- greater than >
- less than <
- greater than or equal to >=
- less than or equal to <=

Double equal operator = =

This operator returns True if the two values are equal, otherwise it returns False

```
# Is 5 equal to 3?


5 = = 3
```

```
False
```

As you know the answer is False.

```
# Is 5 equal to 5?


5 = = 5
```

```
True
```

Yes! 5 is equal to 5.

```
# Below we compare whether two numbers i.e. 5 and 7 are equal


5 = = 7
```

```
False
```

Since the two values are not equal, then the result is False

Class Activity 3 (Pilot Question 2)

Write Python Code To Compare 5 And 6

Pilot Answer 2

5 = = 6

Not equal to operator !=

This operator returns *True* if the two values are not equal to each other, otherwise it returns False.

```
# Below we compare whether 5 is not equal to 10

5 != 10
```

True

```
# Below we compare whether 200 is not equal 100

200 != 100
```

True

Greater than operator >

This operator returns True if the value on the left side of operator (>) is bigger than the value on the right side of the operator and False if otherwise.

```
# is 100 greater than 30?

100 > 30
```

True

Since 100 is greater than 5, the result is True

```
# is -60 greater than 50?

-60 > 50
```

False

The answer is False. 50 is greater than -60

```
# is -6 greater than - 5?

-6 > -5
```

False

As you can see, -6 is less than -5.

Less than operator <

This operator returns True if the value on the left side of the operator (<) is smaller than value on the right side of the operator and False if otherwise.

```
# is 100 less than 1000

100 < 1000
```

True

The result of the above expression is True since 100 is less than 10000

```
# is -6 less than - 5?

-6 < -5
```

True

As you can see, − 6 is less than − 5 .

Greater than or equal to operator >=

This operator returns *True* if the value on the left side of the operator (>=) is bigger than or equal to the value on the right side of the operator and *False* if otherwise.

```
# Below we compare whether 100 is greater than or equal to 1000

100 >= 1000
```

```
False
```

Since 100 is not greater than or equal to 1000, the result is False

```
# Below we compare whether 60 is greater than or equal to 45

60 >= 45
```

```
True
```

```
# We compare whether 100 is greater than or equal to 100

100 >= 100
```

```
True
```

Less than or equal to operator <=

This operator returns *True* if the value on the left side of the operator (<=) is smaller than or equal to the value on the right side of the operator and *False* if otherwise.

```
# is  100 less than or equal to 1000 ?

100 <= 1000
```

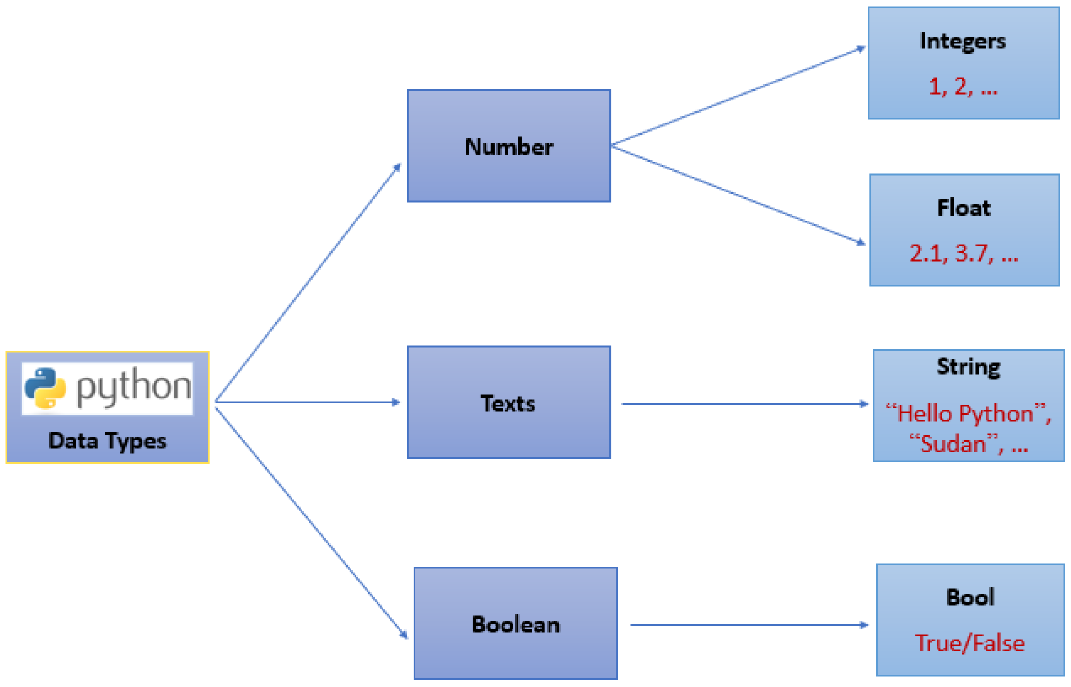
```
True
```

Yes, 100 is less than or equal to 1000

1.4 Data Types in Python

Since Python is an object-oriented programming language. That is, we have many different objects in Python. The most common ones are strings, integers, floats, and Boolean.

- Words or texts in Python are known as strings (str for short) e.g. “Kenya”, “Male”, “Corona virus”
- Number without decimal are known as integers (int for short) e.g. $-1, 0, 9$
- Number with decimal are known as floats e.g. $3.142, -1.6, 4.2$
- Value with either True or False are known as Boolean (bool for short)



Python can tell you the data type by using type() function. Python refers integer number as *int*, floats as *float*, character or text as str and Boolean as *bool*.

1.4.1 Integer data type

Integers are numerical (number) values that are positive or negative whole numbers without decimals

Examples

```
type(20)
```

```
<class 'int'>
```

```
type(2)
```

```
<class 'int'>
```

```
type(2021)
```

```
<class 'int'>
```

1.4.2 Float data type

Float (floating point) numbers are numbers with a decimal point.

Examples

```
type(3.142)
```

```
<class 'float'>
```

```
type(11.124)
```

```
<class 'float'>
```

```
type(-2021.2)
```

```
<class 'float'>
```

1.4.3 String data type

Characters or texts are known as string in Python. A string object must be inside a single or double quotation. Otherwise, Python will throw an error.

Example 1

```
"I am from Sudan"
```

‘I am from Sudan’

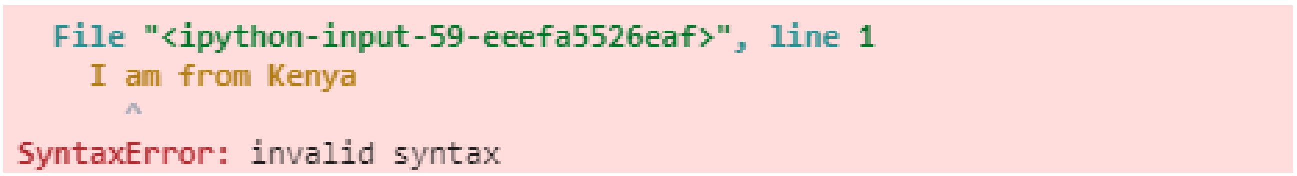
```
type("I am from Sudan")
```

<class ‘str’>

As you just learnt, this is a string(*str*) data type.

Example 2

```
I am from Kenya
```



```
  File "<ipython-input-59-eeefa5526eaf>", line 1
    I am from Kenya
      ^
SyntaxError: invalid syntax
```

There is a *SyntaxError* because we don’t put I am from Kenya in a quotation. Now, Let’s correct the error by putting the texts in a double quotation.

```
"I am from Kenya"
```

‘I am from Kenya’

```
type("I am from Kenya")
```

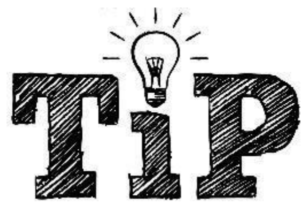
<class ‘str’>

Strings can be represented by both single quotes ' ' or double quotes " ". However, you can’t mix them *i.e.* " ' or ' ".

Example 3

```
"Python is so simple"
```

```
'Python is so simple'
```



It is common to see most Python users using the function print() in their line of codes. For now, all you need to know is that it instructs python on what to print.

```
print("Hello, I am from Ethiopia and I can speak Amharic")
```

```
Hello, I am from Ethiopia and I can speak Amharic
```

```
print(type("Hello, I am from Ethiopia and I can speak Amharic"))
```

```
<class 'str'>
```

```
print(4+6)
```

```
10
```

```
print(type(14.6))
```

```
<class 'float'>
```

1.4.4 Boolean data type

A Boolean or bool is an object in python that can take two values i.e. True or False.

Values of a Boolean must always start with a capital letter and then followed by small letters. This allows the programming language to identify the value as a Boolean. When we check the type of *True* or *False*, the result is a *bool* which represents the Boolean data type.

Example 1

Check for the data type of a Boolean True

```
# To check the data type

type(True)
```

```
<class ‘bool’>
```

Example 2

Check for the data type of a Boolean False

```
# To check the data type

type(False)
```

```
<class ‘bool’>
```

Some operations can also result to a Boolean data type. For example:

```
8 > 6
```

```
True
```

```
6 < - 10
```

```
False
```

```
"Kenya" != "South Sudan"
```

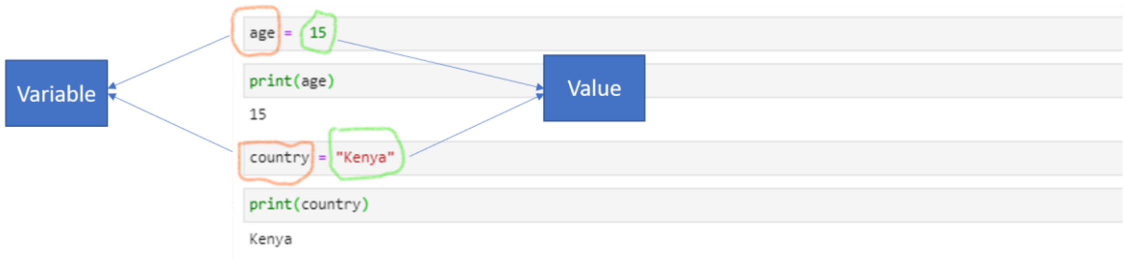
```
True
```

```
"South Sudan" != "Kenya"
```

```
True
```

1.5 Variable Assignment

In the simplest terms, a variable is a way at which a programmer stores a value so that it can be used over and over without having to repeat oneself. A variable is just like a container that stores a value. For example, In the image below, *age* is a variable that stores the value of **15** or the value of **15** is stored into the variable age. Also, *country* is a variable that stores or hold the value of "Kenya". The equal operator (sign) is called an assignment.



```
age = 15
print(age)
15
country = "Kenya"
print(country)
Kenya
```

Example 1

```
age = 15
```

To access what is in the variable *age*, you can do that by putting the name of the variable in the print() function or simply type the name of the variable without a *print()* function.

```
print(age)
```

```
15
```

```
age
```

```
15
```

We can also access the kind of data type in the variable age.

```
type(age)
```

```
<class 'int'>
```

The value store in the variable *age* is an integer (*int*) data type

<u>Example 2</u>

```
country = "Kenya"

print(country)
```

```
Kenya
```

```
type(country)
```

```
<class ‘str’>
```

The value stored in a variable *country* is a string (*str*) data type

Class Activity 4 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

- What is your name? Assign the value to the variable *name*
- How old are you? Assign the value to the variable *age*
- Are you a male or female? Assign the value to the variable *gender*
- What is the name of your country? Assign the value to the variable *country*
- What language can you speak? Assign the value to the variable *language*
- True or False, do you smile today? Assign the value to the variable *smile*

The list of the variables to create and assign values to are:

Name, Age, Gender, Country, Language, Smile

Programming is about automating mundane tasks that otherwise would have taken ages! It also makes sense for the programmer to make their work easier by having a single representation of knowledge at only one point of the programme. This in turn makes it easier to understand code and reduce the number of lines one has to write and hence saves time!

When a value is stored in a variable, you can then use that variable to do further calculation

Example 1

```
# Age now

age = 15

age
```

```
15
```

```
# Next year age

age + 1
```

```
16
```

```
# Age last two years

age - 2
```

```
13
```

Example 2

```
# Assigning values to Variables

x = 2

y = 3

z = 4
```

```
x + y
```

```
5
```

```
z - x - y
```

```
-1
```

```
x * z
```

```
8
```

```
x ** 2
```

```
4
```

```
z % y
```

```
1
```

```
divmod(z, x)
```

```
(2, 0)
```

Example 3

We can assign multiple values to multiple variables at the same time in Python

```
name, age, gender, country = "Jamal", 16, "Male", "Sudan"

print(name)
```

```
Jamal
```

```
print(age)
```

```
16
```

```
print(gender)
```

```
Male
```

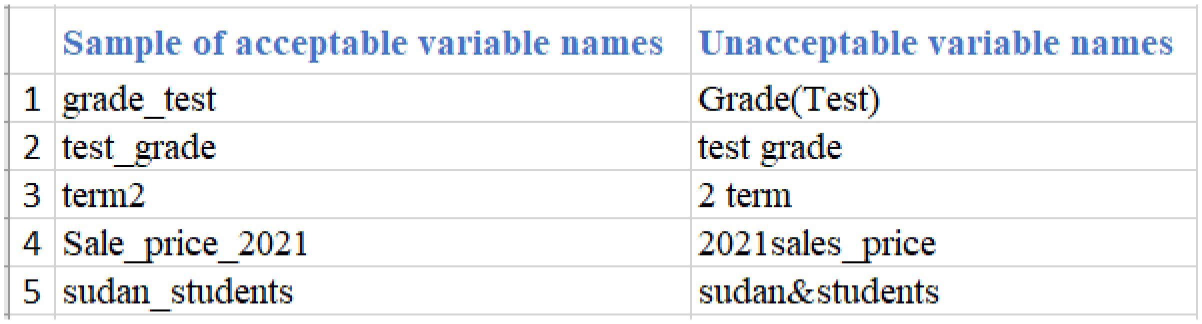
```
print(country)
```

```
Sudan
```

1.5.1 Rules for naming variables

- All variables must begin with a letter of the alphabet
- After the initial letter, variable names can also contain underscore (_) and/or a number. No spaces or special characters, however are allowed
- Uppercase strings are different from lowercase string in Python

Example



	Sample of acceptable variable names	Unacceptable variable names
1	grade_test	Grade(Test)
2	test_grade	test grade
3	term2	2 term
4	Sale_price_2021	2021sales_price
5	sudan_students	sudan&students

It is a good practice to give meaningful variable name so that when someone else or you in the future can understand what you were trying to accomplish. The best naming convention is to choose a variable name that will tell the reader of the program what the variable represents.

Class Activity 5 (Peer to Peer Review Activity)

Peer to Peer Interaction

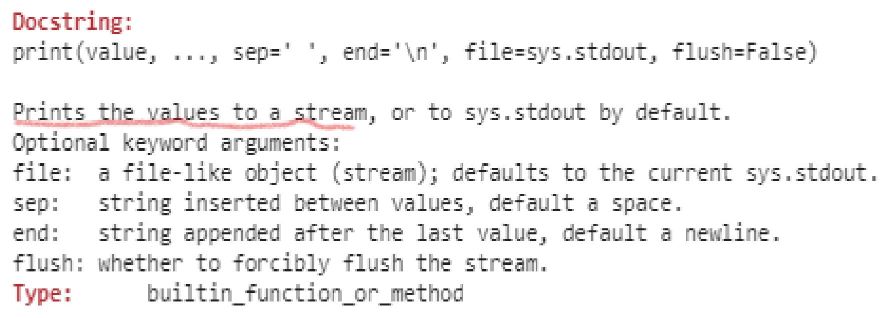
Visit the LMS, locate forum activity and participate in the discussion

Discuss among your colleagues why some variable names in the examples above are acceptable and why some are not acceptable in Python

1.5.2 How to get Information about a Function?

If you want to become a good programmer, it is essential to read functions documentations. This is like a user guide or manual about a specific function. There are many ways to get help in Python, but the simplest way is to question the function by using a question mark. For example, to read the help file or documentation of print() function, we shall use ?print

```
?print
```

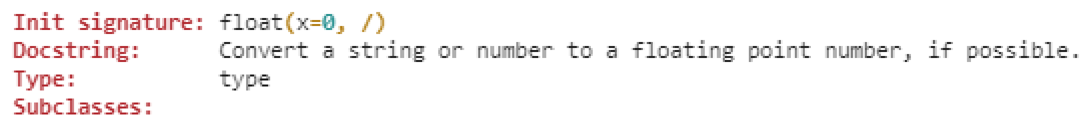


```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

It prints the value of an object.

Let’s see what the float() function does by using ?float

```
?float
```

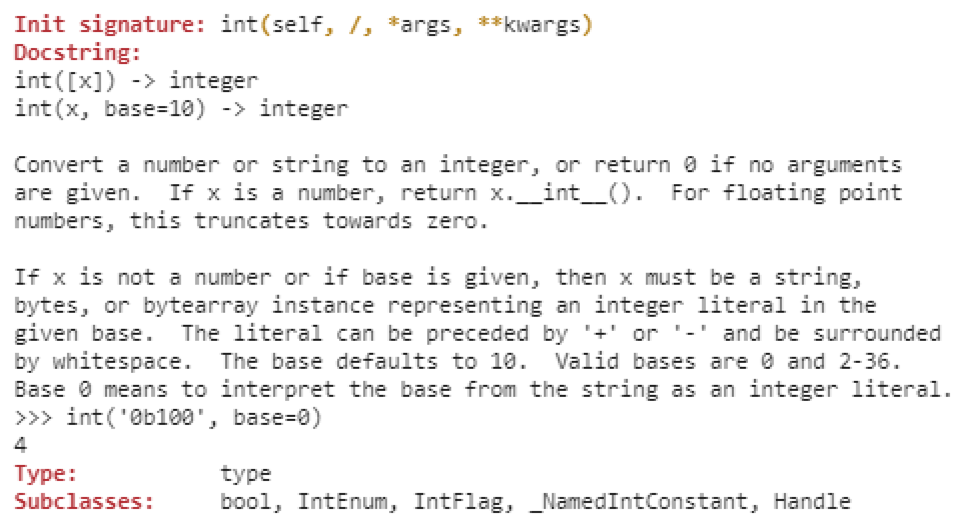


```
Init signature: float(x=0, /)
Docstring:      Convert a string or number to a floating point number, if possible.
Type:           type
Subclasses:
```

It converts a string or number to a floating point number, if possible.

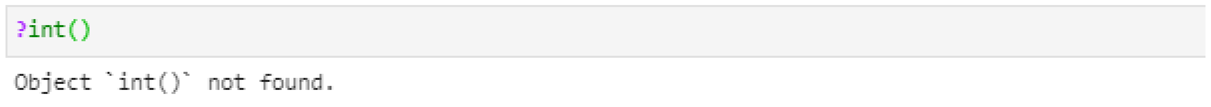
Don't use parentheses in the function when seeking for help in Python. For example, if you need a help on what int() function does, just type ?int without parentheses.

```
?int
```



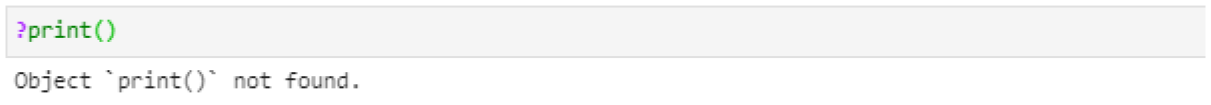
```
Init signature: int(self, /, *args, **kwargs)
Docstring:
int([x]) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given.  If x is a number, return x.__int__().  For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base.  The literal can be preceded by '+' or '-' and be surrounded
by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
Type:           type
Subclasses:     bool, IntEnum, IntFlag, _NamedIntConstant, Handle
```

If you include parentheses, you will have the following:



```
?int()
```

```
Object `int()` not found.
```



```
?print()
```

```
Object `print()` not found.
```

As you can see, that is not what you are looking for.

1.5.3 Data Type Conversions

Python allows us to convert objects from one data type to another. For example, you may need to convert a *float* to an integer (*int*). In programming this is called typecasting and has a lot of very many useful applications as will be shown later.

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 an integer, we will get 1 and therefore, lose the decimal information (i.e. 0.1)

Example 1

Let us convert an integer *9* into a float i.e. *int* to *float*. First assign the value of 9 to a variable *kenya_shilling*.

```
kenya_shilling = 9
```

Let us check the data type in the variable *kenya_shilling* by using type() function

```
type(kenya_shilling)
```

<class 'int'>

Now that we are sure that the value in the variable *kenya_shilling* is an integer (*int*) data type. So, we can convert it to a float data type by using a *float()* function. Save the result as *kenya_shilling_float*

```
kenya_shilling_float = float(kenya_shilling)

type(kenya_shilling_float)
```

```
<class 'float'>
```

As you can see, we have cast the integer (int) to a float data type

Example 2

Let us try and convert a float to an integer. As we said above, we will end up losing information so you should be careful about this type of conversion.

Convert **9.345** to an integer

```
int(9.345)
```

```
9
```

The result of the above code is 9, i.e. we lose 0.345 which is a lot of information. For example, if it was an amount of money transfer, it would be potentially wrong to report that the amount was 9 and not 9.345.

Example 3

We can also convert a pure number that is recognised as string (*str*) due to one reason or the other to float or an integer. This is only possible when the string looks like an *int* or a *float* otherwise python will throw an error

```
# Convert a string into an integer

age = "20"

type(age)
```

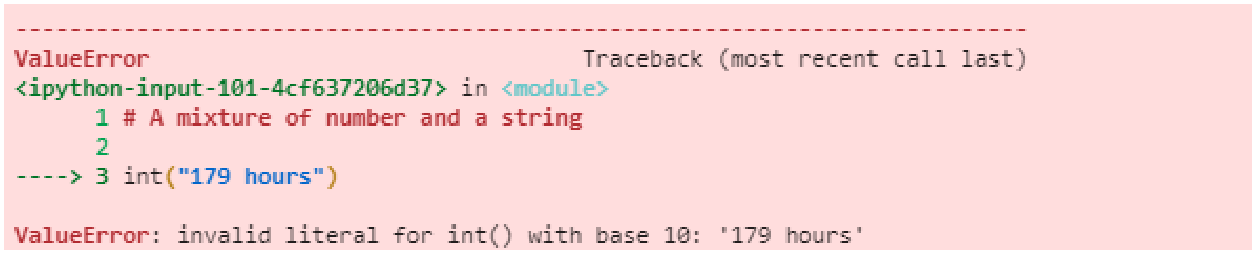
<class ‘str’>

Because the value of age is in a quotation, therefore, Python sees this as a string. We can convert the value in age to an integer by using int()function.

```
# Converting an inperfect string to an integer

new_age = int(age)

type(new_age)
```

<class ‘int’>

```
# A mixture of number and a string

int("179 hours")
```



```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-101-4cf637206d37> in <module>
      1 # A mixture of number and a string
      2
----> 3 int("179 hours")

ValueError: invalid literal for int() with base 10: '179 hours'
```

This throws an error because this is not a pure number. Other data types can also be converted to strings by using str() function e.g.

```
# Convert a float to string
str(9.7)
```

‘9.7’

```
# Convert an int to a string
str(6)
```

‘6’

Example 4

In our introduction to Boolean data type, we said that Boolean can take two values True or False. It is often common in practice to represent 1 as True and *0* as False. This can be seen when we convert Booleans to integer (*int*) or *float*.

```
# Convert False to integer

int(False)
```

```
0
```

```
# Convert True to integer

int(True)
```

```
1
```

```
# Convert integer 1 to bool

bool(1)
```

```
True
```

```
# Convert integer 0 to bool

bool(0)
```

```
False
```

1.5.3 Logical Operators

In the last section we talked about some mathematical operators such as +, -, *, /, **, and % and comparison operators such as ==, !=, >, >=, <, and <=. In this section, you will learn about logical operators.

Logical operators test whether an expression is *True* or *False*. There are three types of logical operators namely *and, or, not*.

- *and* - This operator returns a False when it encounters a *False* condition
- *or* - This operator returns a True when it encounters at least a *True* condition

- *not*- This operator returns *True* if the condition is *False* and *True* if the condition is *False*

Example 1: Using *and* operator

We define a variable *mins* to represent the number of minutes a money transfer transaction in South Sudan takes to be completed.

```
mins = 3

# Run this cell to see the outcome


mins < 7 and mins < 1
```

False

Since the two expressions are both False i.e.

mins < 7 is False

mins < 1 is False

Therefore, mins < 7 and mins < 1 will be *False*

```
# To cross check


False and False
```

False

Example 2: Using *or* operator

We define a variable *year* to represent the number of years Aden a Somalia has been living in Ethiopia

```
year = 12

# Run this cell to see the outcome

year = = 12 or  year < 10
```

True

Since one expression is True i.e.

year = = 12 is True

year < 10 is False

Therefore, year = = 12 or year < 10 will be *True*

Class activity 5 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

What will be the result of the following expressions?

minute = 18

minute < 7 and minute < 1

Example 3: Using *not* operator

We define a variable *mins* to represent the number of minutes a money transfer transaction in Sudan takes to be completed.

```
mins = 10

mins > 5
```

True

```
not mins > 5
```

False

As you know, anything that is not *True* is *False*! For example:

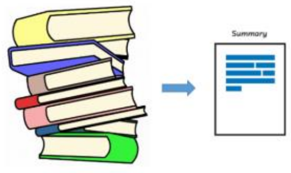
```
not True
```

False

```
not False
```

True

Summary of Study Unit 1



In this study unit, you have learnt that:

1. Python can be used as a calculator to do some arithmetic operations
2. Data type includes integer, float, string, and Boolean
3. Boolean data type can either be True or False
4. String data type can be created by using either single quote ‘ ’ or double quote “ ”.
5. Python makes use of # to make a comment
6. Comparison operators include = =, !=, >, <, >=, and <=.

Additional resources

For more resources in this section please consider the following:

- https://bit.ly/Python-for-beginners-YouTube