Module 1: Introduction to Python, Pandas and Jupyter Notebook

What is Python?

- Python is a high-level programming language.
- It was created by Guido van Rossum and first released in 1991.

Pandas

- Pandas is a library that is built on top of two core Python libraries—matplotlib for data visualization and NumPy for mathematical operations
- The focus of Pandas is data analysis in which we can analyze, filter, manipulate, and merge data
- It is almost the same as Spreadsheet or Microsoft Excel
- Pandas is powerful if we need to deal with datasets that have millions of rows

Jupyter Notebook

- Jupyter Notebook is the text editor that we are going to use in this workshop
- It is much easier as we can instantly see the output below each cell.
- We will get more in-depth look on Jupyter Notebook after this.

1. Installing Anaconda

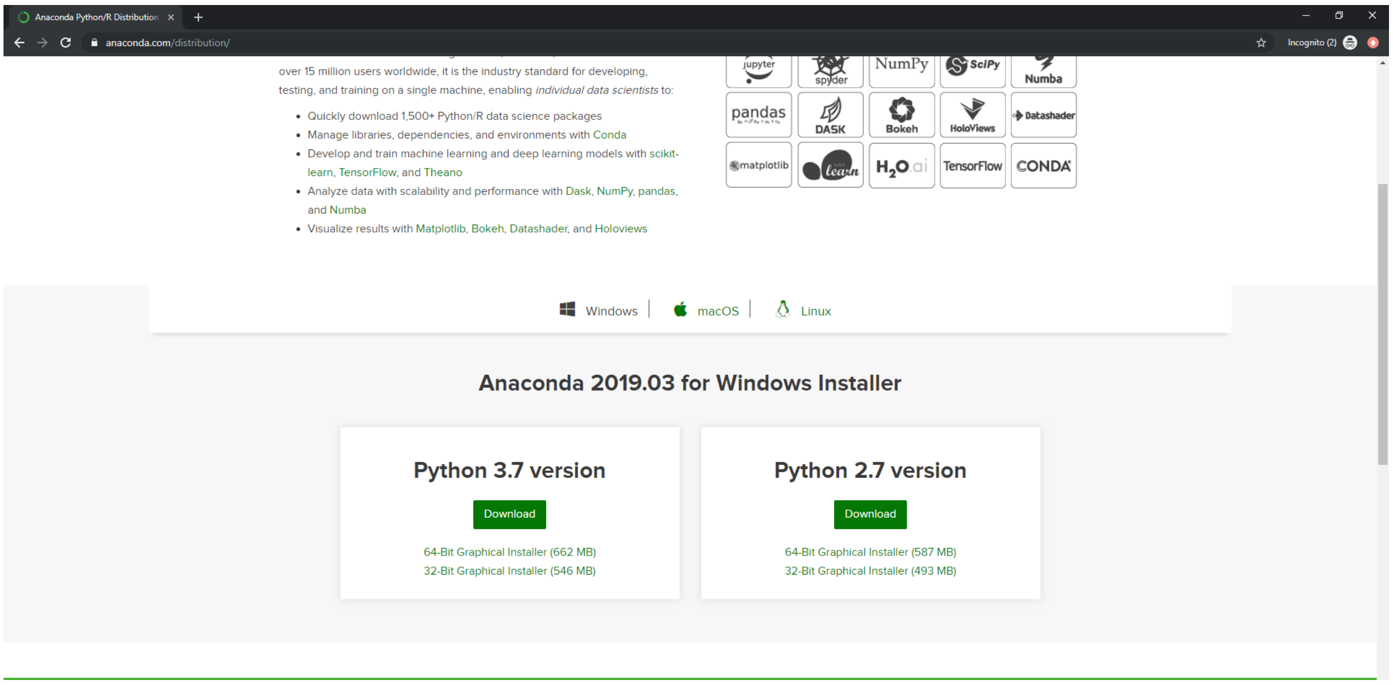
The easiest option to start coding with Python is using Anaconda distribution. It comes with many Python bundles, pandas, and 100+ data analysis libraries.

- The best thing is about Anaconda is that, it is completely **free**.

To start using it, first, go to Anaconda's official site:

https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)

Click download and you will arrive at the page displayed below.



Anaconda Python/R Distribution

anaconda.com/distribution/

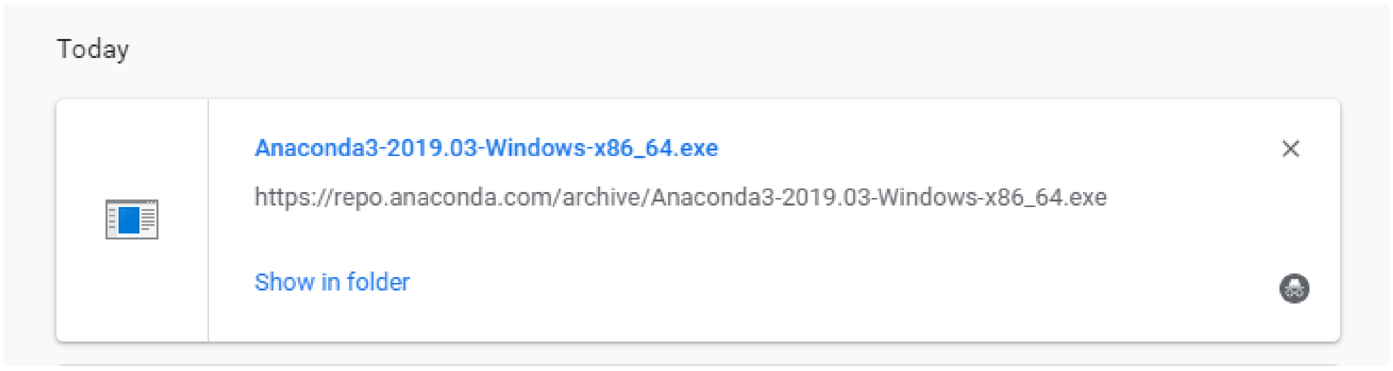
over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

Anaconda 2019.03 for Windows Installer

Python 3.7 version	Python 2.7 version
Download	Download
64-Bit Graphical Installer (662 MB)	64-Bit Graphical Installer (587 MB)
32-Bit Graphical Installer (546 MB)	32-Bit Graphical Installer (493 MB)

Download the installer based on the Operating System that you use.

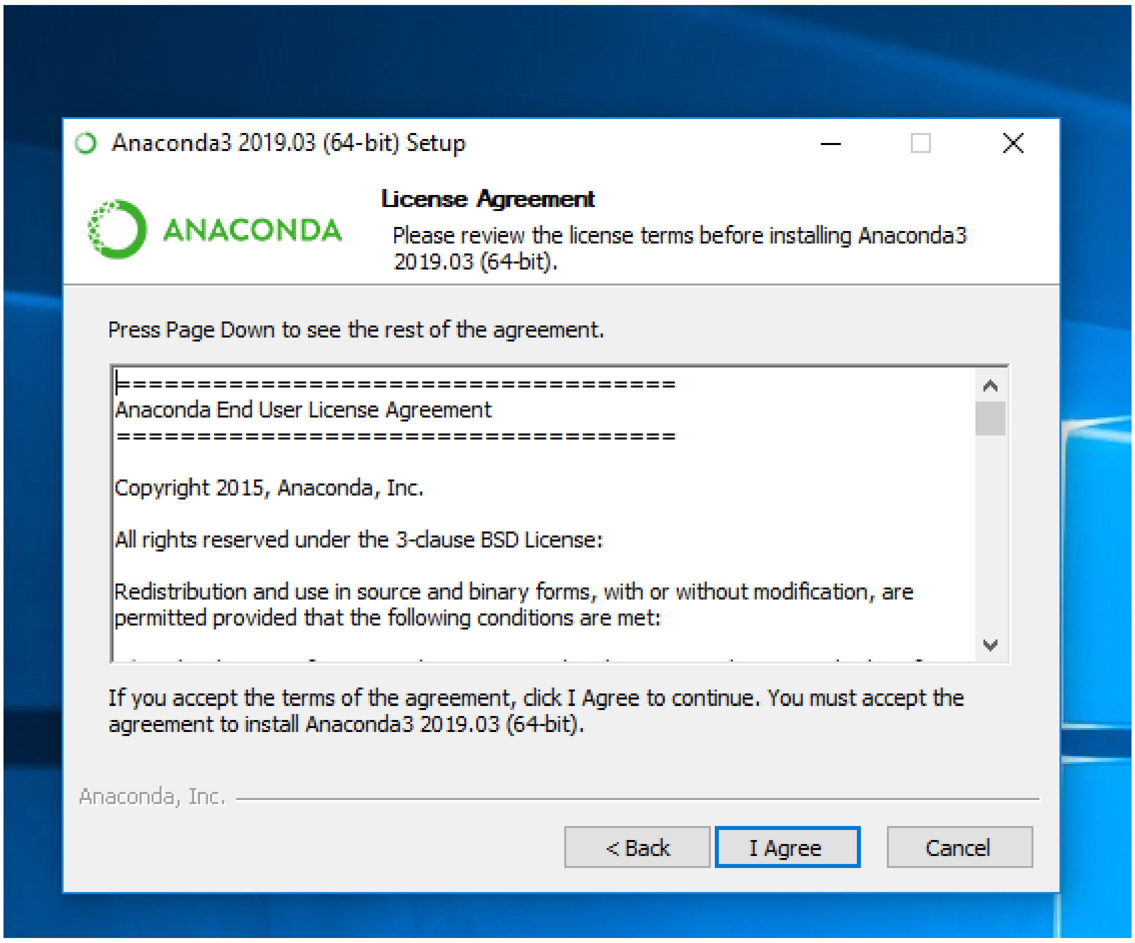


Today

Anaconda3-2019.03-Windows-x86_64.exe

https://repo.anaconda.com/archive/Anaconda3-2019.03-Windows-x86_64.exe

Once the download is finished, open the file to install it.



License Agreement

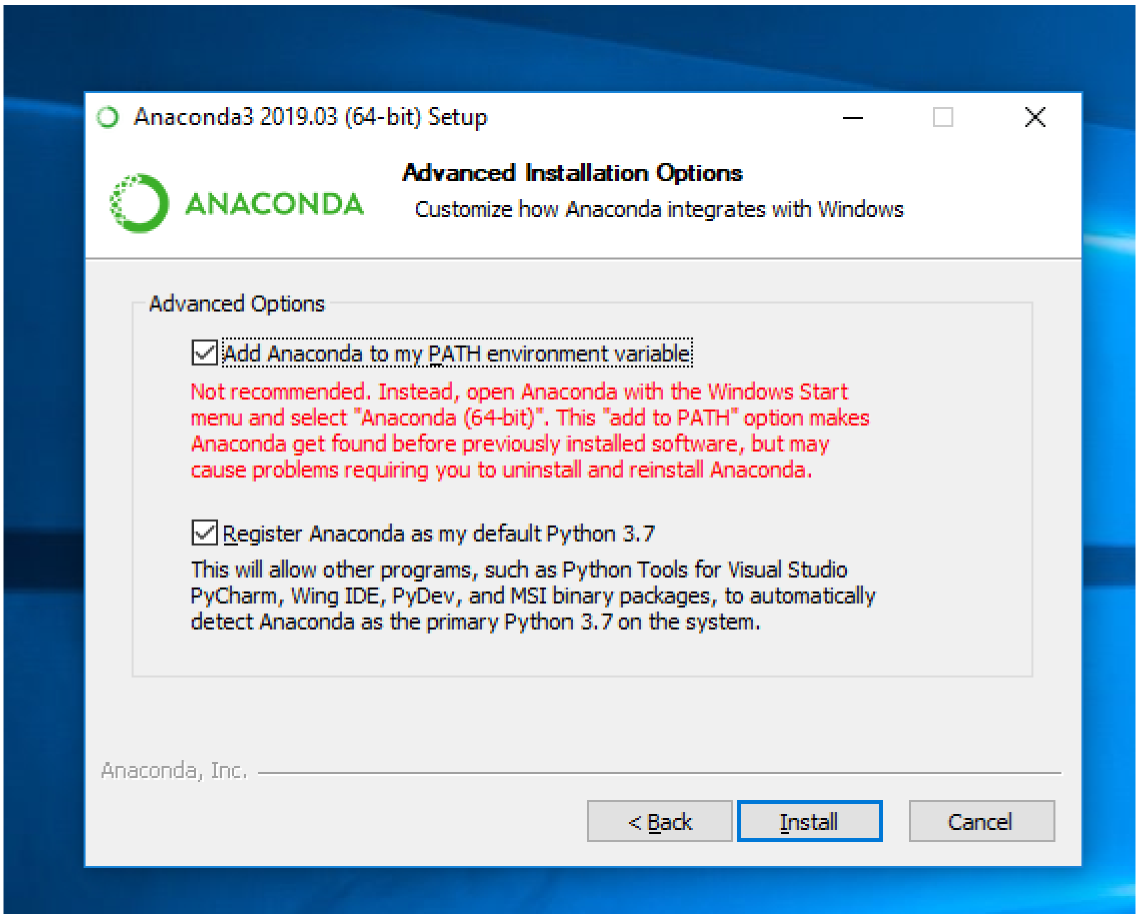
Please review the license terms before installing Anaconda3 2019.03 (64-bit).

Press Page Down to see the rest of the agreement.

```
===================================
Anaconda End User License Agreement
===================================

Copyright 2015, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2019.03 (64-bit).

Click **Next**, and then I **Agree** to start the installation process.



Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

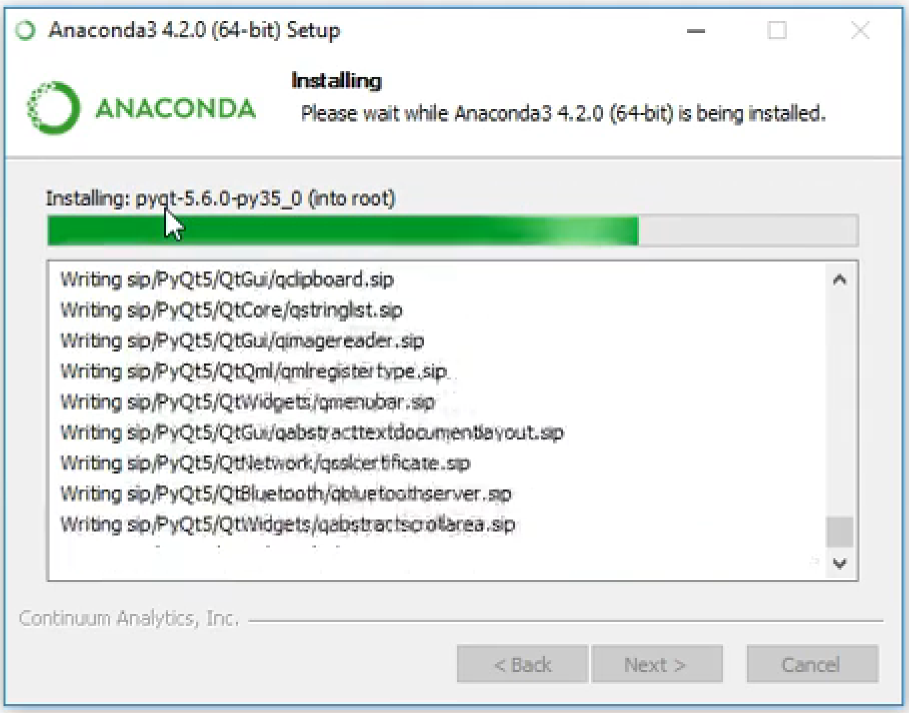
☑ Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as my default Python 3.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.

For this section, it is optional to tick on the first option.

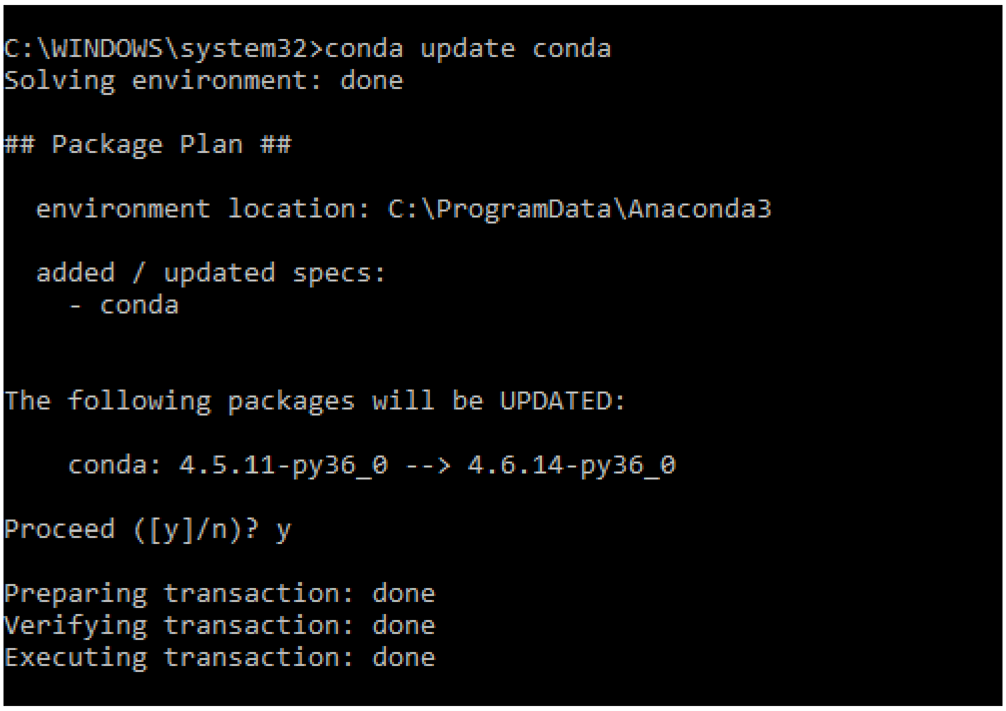

Anaconda3 4.2.0 (64-bit) Setup
ANACONDA
Installing
Please wait while Anaconda3 4.2.0 (64-bit) is being installed.
Continuum Analytics, Inc.
< Back
Next >
Cancel

```
Installing: pyqt-5.6.0-py35_0 (into root)
Writing sip/PyQt5/QtGui/qclipboard.sip
Writing sip/PyQt5/QtCore/qstringlist.sip
Writing sip/PyQt5/QtGui/qimagereader.sip
Writing sip/PyQt5/QtQml/qmlregistertype.sip
Writing sip/PyQt5/QtWidgets/qmenubar.sip
Writing sip/PyQt5/QtGui/qabstracttextdocumentlayout.sip
Writing sip/PyQt5/QtNetwork/qsslcertificate.sip
Writing sip/PyQt5/QtBluetooth/qbluetoothserver.sip
Writing sip/PyQt5/QtWidgets/qabstractscrollarea.sip
```

Now, wait until the installation is completed.

2. Update libraries using Command Prompt (CMD)

- Click Windows button and search for CMD
- Type **conda update conda** to update Conda libraries.



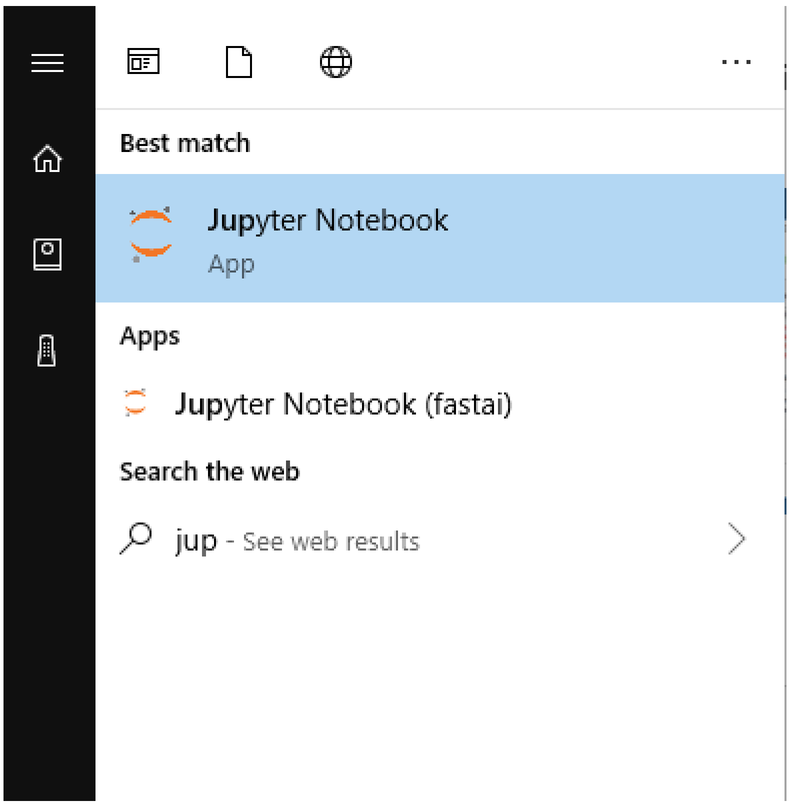
```
C:\WINDOWS\system32>conda update conda
Solving environment: done

## Package Plan ##

  environment location: C:\ProgramData\Anaconda3

  added / updated specs:
    - conda


The following packages will be UPDATED:

    conda: 4.5.11-py36_0 --> 4.6.14-py36_0

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

- As you can see here, I'm trying to update the latest library of Conda. The version I have now is 4.5 while the latest one is 4.6. It is always better to use the most recent library.

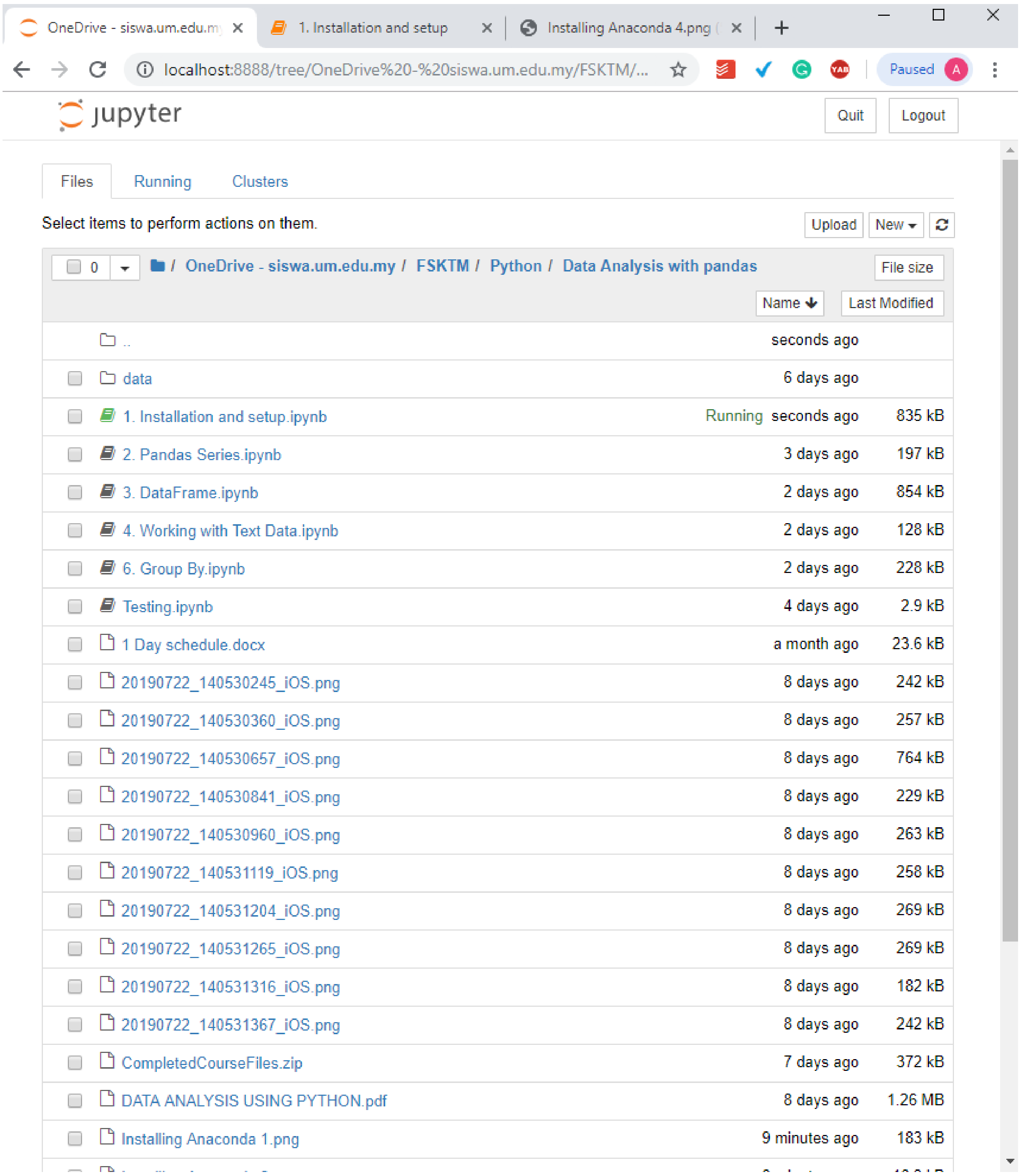
- Next, type **conda update pandas** to update the Pandas library.

Starting Jupyter Programme

Search for Jupyter Notebook and run the program.



You should see the notebook open in your browser.

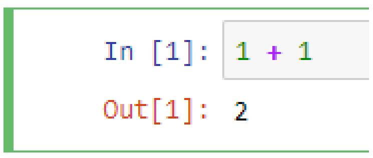

OneDrive - siswa.um.edu.my
1. Installation and setup
Installing Anaconda 4.png
localhost:8888/tree/OneDrive%20-%20siswa.um.edu.my/FSKTM/...
Paused
jupyter
Quit
Logout
Files
Running
Clusters
Upload
New
File size
Name
Last Modified

Select items to perform actions on them.

0 / OneDrive - siswa.um.edu.my / FSKTM / Python / Data Analysis with pandas

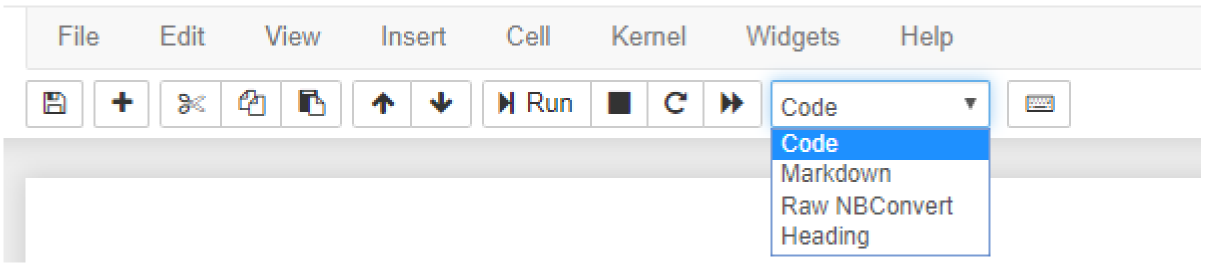
..		seconds ago	
data		6 days ago	
1. Installation and setup.ipynb	Running	seconds ago	835 kB
2. Pandas Series.ipynb		3 days ago	197 kB
3. DataFrame.ipynb		2 days ago	854 kB
4. Working with Text Data.ipynb		2 days ago	128 kB
6. Group By.ipynb		2 days ago	228 kB
Testing.ipynb		4 days ago	2.9 kB
1 Day schedule.docx		a month ago	23.6 kB
20190722_140530245_iOS.png		8 days ago	242 kB
20190722_140530360_iOS.png		8 days ago	257 kB
20190722_140530657_iOS.png		8 days ago	764 kB
20190722_140530841_iOS.png		8 days ago	229 kB
20190722_140530960_iOS.png		8 days ago	263 kB
20190722_140531119_iOS.png		8 days ago	258 kB
20190722_140531204_iOS.png		8 days ago	269 kB
20190722_140531265_iOS.png		8 days ago	269 kB
20190722_140531316_iOS.png		8 days ago	182 kB
20190722_140531367_iOS.png		8 days ago	242 kB
CompletedCourseFiles.zip		7 days ago	372 kB
DATA ANALYSIS USING PYTHON.pdf		8 days ago	1.26 MB
Installing Anaconda 1.png		9 minutes ago	183 kB

3. Cell Type and cell Modes

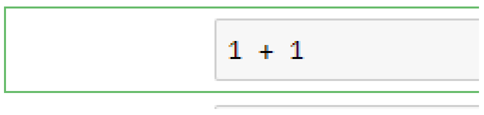


```
In [1]:  1 + 1

Out[1]:  2
```

A cell is a multiline text input field, and its contents can be executed by using Shift-Enter, or by clicking the “Play” button at the toolbar. The above shows an example of a code cell. [1] is the execution number, 1 + 1 is the code and 2 is the output.



In Jupyter Notebook, there are actually 4 different types of cell you can choose to work with. Let's try changing the above cell type to Markdown.



```
1 + 1
```

As you can see, changing the cell type to Markdown will no longer give an output of 2 when you run it. Instead, the code, 1 + 1 is now considered as a text.

This is basically what markdown does and it is useful for writting comments, making header (using #) or bulleted lists (using >) and embedding video.

The benefit of Markdown is, it enables the reader to understand the code easier because the coder can write the description of their code.

Example of a markdown:

Header

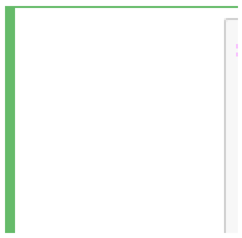
bullet lists

- Name : Ali
- Age: 30
- Faculty : Computer Science

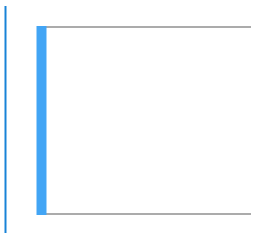
> a
>
> read more: https://medium.com/ibm-data-science-experience/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed (https://medium.com/ibm-data-science-experience/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed)
>
> *Indent text using ">"*

Edit mode and Command mode

Edit mode is when you can type into the cell, like a normal text editor and run it by hitting shift + enter. It is indicated by a green cell border and left margin.



Command mode is when we can edit the notebook as a whole. This mode enables us to use shortcut keys to perform notebook and cell actions efficiently. Clicking the Escape key will bring you to this mode and it is indicated by the grey cell border and blue left margin.



In this mode, pressing "A" will create a new layer of cell. We can also try pressing the Escape + "M" key to change the current cell type to Markdown without using our mouse.

4. Code execution

There are two ways to run the cell/code

- *Ctrl + Enter* .

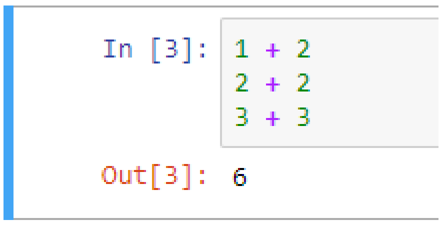
    - Runs the cell.

- *Shift + Enter*.

    - Runs the cell then move to the next cell.
    - If it it the last cell, a new cell will be generated automatically.

If the cell has a long line of code, it will only display the output of the last line.

Eg :



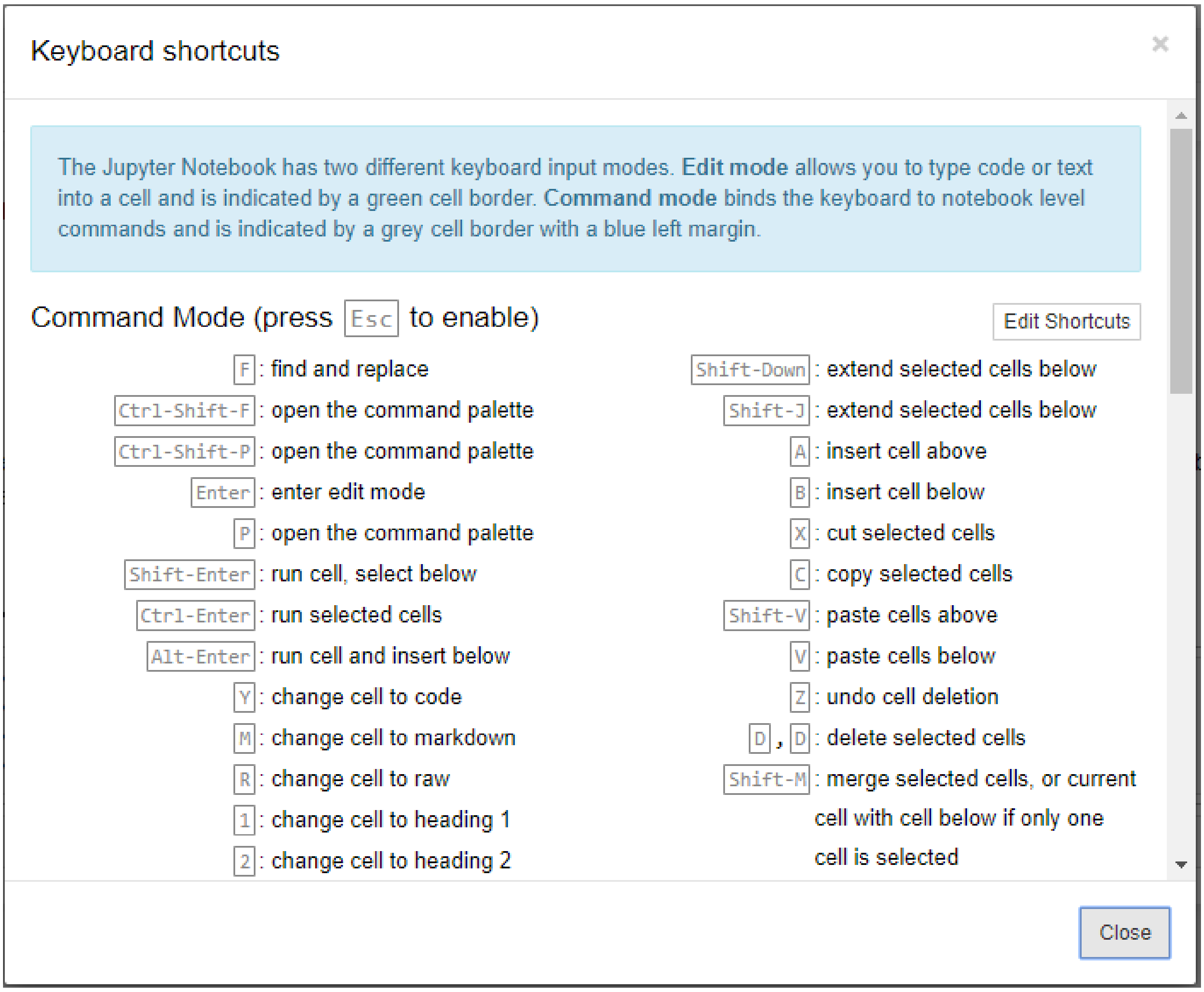
```
In [3]: 1 + 2
        2 + 2
        3 + 3

Out[3]: 6
```

Every cell is independent of each other. It does not depend on which cells is in order first, but rather which is executed first. The number in bracket beside the cell indicates the cell execution number.

5. Popular keyboard shortcuts

- ***Esc + H*** : H stands for help. It will open a list of shortcuts.

- ***Esc + X*** : Cut the cells. It can be used as delete too.
- ***Esc + B*** : Creates a new cell below.
- ***Esc + A*** : Creates a new cell Above.



The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code or text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

Command Mode (press `Esc` to enable)

`F` : find and replace	`Shift-Down` : extend selected cells below
`Ctrl-Shift-F` : open the command palette	`Shift-J` : extend selected cells below
`Ctrl-Shift-P` : open the command palette	`A` : insert cell above
`Enter` : enter edit mode	`B` : insert cell below
`P` : open the command palette	`X` : cut selected cells
`Shift-Enter` : run cell, select below	`C` : copy selected cells
`Ctrl-Enter` : run selected cells	`Shift-V` : paste cells above
`Alt-Enter` : run cell and insert below	`V` : paste cells below
`Y` : change cell to code	`Z` : undo cell deletion
`M` : change cell to markdown	`D`, `D` : delete selected cells
`R` : change cell to raw	`Shift-M` : merge selected cells, or current cell with cell below if only one cell is selected
`1` : change cell to heading 1	
`2` : change cell to heading 2	

6. Import libraries into Jupyter Notebook

- A library is a prewritten code that will extend the functionality of python programming language.
- Library is managed by Anaconda distribution which comes with more than 100 libraries.
- However, library is not imported by default to save memory

To import a library you have to type:

import LibraryName **as** aliasName

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

import pandas as pd
Pandas is a software library written for the Python programming language to do data manipulation and analysis. Community standards use pd as a short form for pandas. This is to ensure other readers can read it easier.

import numpy as np
Numpy stands for Numerical python. Pandas library is built on top of Numpy library. We can generate 1 million rows of random data using it.

import matplotlib.pyplot as plt
Rather than importing the whole library, here I choose a specific module, . This way allows me to save a lot of memory.
%matplotlib inline
This line is for rendering the graph inside Jupyter Notebook, not in a new windows terminal.

Make sure to test whether the import is working or not.

In [2]:

```
pd.__version__
```

Out[2]:

```
'0.20.3'
```

7. Basic Python - Data types & Variables

Comment is a line of code that is ignored by python interpreter. It is written after the hashtag (#) symbol and is not executed. Comments are usually used to write the explanation of the code

In [3]:

```
#Code of 1+1
1+1
```

Out[3]:

```
2
```

Data type

Integer

Whole numbers without fractional components.

- Eg: 1, 2, 3 or a = 1, b = 2

In [4]:

```
bornDate = 1998
thisYear = 2019
Age = thisYear - bornDate
Age
```

Out[4]:

```
21
```

Floating points

Numbers with fractional components.

- Eg : 3.14, 9.9, - 10.117 `

In [5]:

```
height = 1.64
weight =  58.5
bmi = weight/height**2
bmi
```

Out[5]:

```
21.75044616299822
```

Strings

A string is a collection of character(s) inside a double quotation mark (" "). For example:

In [6]:

```
"My name is Amin Hakim."
```

Out[6]:

```
'My name is Amin Hakim.'
```

The are many things we can do with a string. For instance, we can combine it with float number using .forrmat function.

In [7]:

```
"My height is {}m and my weight is {}kg".format(height, weight)
```

Out[7]:

```
'My height is 1.64m and my weight is 58.5kg'
```

We can also choose to retrieve a part of the string only.

In [8]:

```
name = "Muhamad Amin Hakim"
name[8:]
```

Out[8]:

```
'Amin Hakim'
```

In python, there is a built-in function to know what data type we are working with. It is called the type() function. Inside the bracket, we can specify the data that we want to identify its datatype.

int stands for Integer.
str stands for Strings.

In [9]:

```
type("I'm eating bread for breakfast")
```

Out[9]:

```
str
```

In [10]:

```
type(12345)
```

Out[10]:

```
int
```

Boolean

Datatype for True or False. It is useful when we want to set a condition. Boolean can also be used to represent On/Off or Yes/No. Bool stands for boolean.

In [11]:

```
False
```

Out[11]:

```
False
```

In [12]:

```
True
```

Out[12]:

```
True
```

In [13]:

```
type(True)
```

Out[13]:

```
bool
```

Variables

Variables are used to store data so that they can be used multiple times. Variables can also store complex mathematical expressions. You can see in the previous example on how we calculate the BMI using height and weights.

Everything on the right side is calculated first before being assigned to the variable.

In [14]:

```
money = 35
money
```

Out[14]:

```
35
```

In [15]:

```
money = money + 5
money
```

Out[15]:

```
40
```

We also can combine two strings from different variables. For example, we want to combine first name and last name:

In [16]:

```
firstName = "Amin Hakim"
lastname = "Sazali"

Full_Name = firstName + " " + lastname
print(Full_Name)
```

```
Amin Hakim Sazali
```

8. Basic Python - Lists

A list is a data container that can store multiple values. In other programming languages, lists are known as arrays.

In [17]:

```
[1,2,3,4]
```

Out[17]:

```
[1, 2, 3, 4]
```

```
In [18]:
```

```
["John", "Ali", "Abu"]
```

```
Out[18]:

['John', 'Ali', 'Abu']
```

```
In [19]:
```

```
type(["3.14,25.42, 56.7"])
```

```
Out[19]:

list
```

In python, lists can contain mixed data types. However, it is more recommended to set a fixed data type to be used in a list.

```
In [20]:
```

```
listA = ["John", "Ali", 45, 50, 24.5, 26.7]
print(listA)
```

```
['John', 'Ali', 45, 50, 24.5, 26.7]
```

There is a built-in function in python to calculate the length of the list called len().

```
In [21]:
```

```
len(listA)
```

```
Out[21]:

6
```

Every list has its index and it starts with 0.

```
In [22]:
```

```
name = ["John", "Ali", "Abu", "Aliff"]
print(name[0])
```

```
John
```

Since index starts with 0, the last index will be the total number of list minus 1.

```
In [23]:
```

```
name[len(name) - 1]
```

```
Out[23]:

'Aliff'
```

We can also access the elements of the list using negative number. Negative numbers indicates that we count backwards starting from the end of the list.

In [24]:

```
print(name[-1])
```

```
Aliff
```

Elements in the list can also be accessed by specfiying the range of index we want.

In [25]:

```
print(name[2:4])
```

```
['Abu', 'Aliff']
```

ACTIVITY: Add your name into the name list

hint : Use append() method.

9. Basic Python - Dictionary

Python dictionary is one-to-one mapping of keys and values. It is just like a traditional dictionary whereby the key is the word and the value is the explanation or description of it . In other programming languages, it is known as map. The key is unique and cannot be repeated but multiple keys can have the same value. Here is an example of a dictionary for a menu including the prices.

In [26]:

```
menu = {
    "Filet-O-Fish" : 10,
    "Starbucks" : 12.50,
    "Fried Rice" : 5.99,
    "Fried Noodle" : 5.99,
    "Big Mac" : 8.75

}
```

We can get the price of an item in the menu through same way we access a list's elements but instead of specifying the index, we use its key.

In [27]:

```
menu["Fried Rice"]
```

Out[27]:

```
5.99
```

If we try to access using a key which is not in the dictionary, we will get an error.

In [28]:

```
menu["Tuna"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-28-fa5145211809> in <module>()
----> 1 menu["Tuna"]

KeyError: 'Tuna'
```

10. Basic Python - Operators

Mathematical operations include addition (+), subtraction (-), multiplication(*) and division (/).

Try substituting the + in the code below with another operator.

In [29]:

```
num = 1 + 2
print(num)
```

```
3
```

Booleans operator

- Used for checking certain conditions.
- Returns either True or False

In [30]:

```
num == 3
```

Out[30]:

```
True
```

In [31]:

```
num == 2
```

Out[31]:

```
False
```

Exclamation mark, **!** indicates **not**. " != " means not equal to.

In [32]:

```
5 != 10
```

Out[32]:

```
True
```

In [33]:

```
5 is not 10
```

Out[33]:

```
True
```

In [34]:

```
"Python" == "Java"
```

Out[34]:

```
False
```

Range operator

- Notice the differences in the code below.
- Include " = " sign after the range operator to indicate bigger/smaller **and** equals to.

In [35]:

```
8 > 10
```

Out[35]:

```
False
```

In [36]:

```
8 > 8
```

Out[36]:

```
False
```

In [37]:

```
8 >= 8
```

Out[37]:

```
True
```

11. Basic Python - Functions

- Functions are reusable chunks of code which can be used over and over again.
- Functions are used to simplify long code and make it more readable.
- A function can have parameter(s) or argument(s) for passing data to itself.

Here is a simple function to print "Hello World".

In [38]:

```
def print_helloWorld():
    print("Hello World")
```

The function above have already been defined so we can call it as many times as we want.

In [39]:

```
print_helloWorld()
print_helloWorld()
print_helloWorld()
```

```
Hello World
Hello World
Hello World
```

Since we called it three times, it will print three "Hello World"s.

Next, let's create a function to convert Malaysian Ringgit (MYR) to US Dollar (USD). Assume that the current exchange rate is **1 USD = 4.11 MYR**

In [40]:

```
def convert_to_dollar(ringgit):
    return ringgit / 4.11
```

We can then do the conversion by simply calling the function as shown below.

In [41]:

```
convert_to_dollar(10)
```

Out[41]:

```
2.4330900243309
```

The function above might return long decimal places. To display in only 2 decimal places, we can use the **format** function.

In [42]:

```
print("{:.2f} USD ".format(convert_to_dollar(10)))
```

```
2.43 USD
```

Apart from **.f**, there are **.d**, .**%** and **.e** formats too. For more information on how to do formatting, you can visit the link here: https://mkaz.blog/code/python-string-format-cookbook/ (https://mkaz.blog/code/python-string-format-cookbook/)

In [43]:

```
num = 23
print("{:0>5d} USD ".format(num))
```

```
00023 USD
```

12. Basic Python - Conditional Statement

If we want to have conditions to execute our code, then we would need to use the **IF statement**.

For instance, we want to set a threshold for a high salary and a low salary. We define the high salary as more than RM3000 and less than that will be a low salary. Changing the salary value to 2000, 5000 and 3000 in the example will give us different outputs.

In [44]:

```
salary = 4000

if salary > 3000:
    print("High Salary")
else:
    print("Low Salary")
```

```
High Salary
```

In [45]:

```
salary = 3000

if salary > 3000:
    print("High Salary")
else:
    print("Low Salary")
```

```
Low Salary
```

Notice that, when the salary is equal to 3000, it is considered as Low Salary. This is because **>** notation means more than while **>=** means more than or equal to. For less than, we use **<** notation and **<=** for less than or equal to.

Symbols	Meaning
>	More than
>=	More than or equal to
<	Less than
<=	Less than or equal to

Conditional Statement with List

If we want to check elements in the list, we can also use the if statement combines with **in keyword**.

In [46]:

```
items = ["orange", "apple", "mango", "papaya","carrot", "butterhead"]

if "orange" in items:
    print("Yes! Orange in the list.")
```

```
Yes! Orange in the list.
```

In [47]:

```
ic_or_email = "aminhakimsazali@gmail.com"
# ic_or_email = "570121-87-2911"

if "@" in ic_or_email:
    print("Email:", ic_or_email )
else:
    print("IC No.:" , ic_or_email )
```

```
Email: aminhakimsazali@gmail.com
```

If we have two or more condition, we can use elif

In [48]:

```
items = ["papaya","carrot", "butterhead"]

if ("orange" or "apple") in items:
    print("Fruits in the list")
elif "carrot" in  items:
    print("Vegetable in the list.")
elif "butterhead" in items:
    print("Butterhead in the list.")
else:
    print("You dont buy any fruit or vegetables! Eat healthy!")
```

```
Vegetable in the list.
```

Notice that, even though both carrot and butterhead are in the list, only one statement will be printed. This is because, once an **if else condition** is met, the other conditions after it is completely ignored.

13. Basic Python - Loops

There are two loops in Python.

- For loops
- While loops

For loops iterate in sequences. The example below will print out the numbers from 0 to the number before 5 (which is 4).

In [49]:

```python
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Another circumstance is when we want to print six Hello World's. In the first method, we copy and paste the same statement 6 times while in the second method, we simply enclose that statement inside a for loop.

In [50]:

```python
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

In [51]:

```python
for i in range(6):
    print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Both methods will give similar result despite the second one having a much shorter code.

For loops with list

We also use the **in** keyword to iterate over the elements of a list.

```
In [52]:
```

```
items = ["orange", "apple", "mango", "papaya","carrot", "butterhead"]

for barang in items:
    print(barang)
```

```
orange
apple
mango
papaya
carrot
butterhead
```

```
In [53]:
```

```
print(items[0])
print(items[1])
print(items[2])
print(items[3])
print(items[4])
print(items[5])
```

```
orange
apple
mango
papaya
carrot
butterhead
```

For loops with dictionary

Previously we learnt about the structure of a python dictionary. By now, you should understand the concept of keys and values.

Let's print the keys of a dictionary.

```
In [54]:
```

```
student_ID = {
    'AMIN' : '170028',
    'ALIFF' : '170030',
    'LUQMAN' :  '170026',
    'AQIFF' : '170001',
}

for ID in student_ID.keys():
    print(ID)
```

```
AMIN
ALIFF
LUQMAN
AQIFF
```

For loops with string

We can also print or retrieve every single character in a string using for loop.

In [55]:

```python
name = 'AMIRUL ASHRAFF'

for character in name:
    print(character)
```

```
A
M
I
R
U
L
 
A
S
H
R
A
F
F
```

While loops

While loop has the same functionality as **for loops**. However, as the name suggests, while loop is executed only if the **condition is True**.

Let's see the example below. **i < 5** is the condition set for this while loop. We first **initialized variable i = 0**. With every iteration, **i will be printed** and its **value will increase by 1** (increment by 1). The iteration stops until value of i is bigger than 5.

In [56]:

```python
i = 0
while (i < 5):
    print(i)
    i = i + 1
```

```
0
1
2
3
4
```

"break" and "continue" statements

Break : terminate and stop loops execution It will run the next line of code after the loops.
Continue : skip one iteration of the loop.

In [57]:

```
for i in range(10):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

Without the break statement, the for loop will print i value starts from 0 until 9. However in this loop, the break statement is executed once the i value is equal to 3. Hence, the loops terminated before 3 is printed.

In [58]:

```
for i in range(10):
    if i == 3:
        continue
    print(i)
```

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

Do you notice the missing value?

Once i is equal to 3, the loops skip the iteration and will continue with the next iteration (i = 4).

Nested Loops

Nested loops here means we have loops in loops. For example, we can have 2 for loops.

Tips : Indentation is important! Always make your codes looks neat.

In [59]:

```
print("i\tj")
for i in range(5):
    for j in range(3):
        print("{} \t {}".format(i,j))
```

```
i	j
0 	 0
0 	 1
0 	 2
1 	 0
1 	 1
1 	 2
2 	 0
2 	 1
2 	 2
3 	 0
3 	 1
3 	 2
4 	 0
4 	 1
4 	 2
```

From the example above, for every value of i, j will be printed 3 times. Besides, for every iteration of i loop, j will start will 0.