

PYTHON QUICK REFRESHER

Introduction to Python

What is Python?

- Python is one of the most popular programming languages, created by **Guido van Rossum** and released in 1991.
- Python is an **interpreted** programming language as opposed to a compiled programming language.



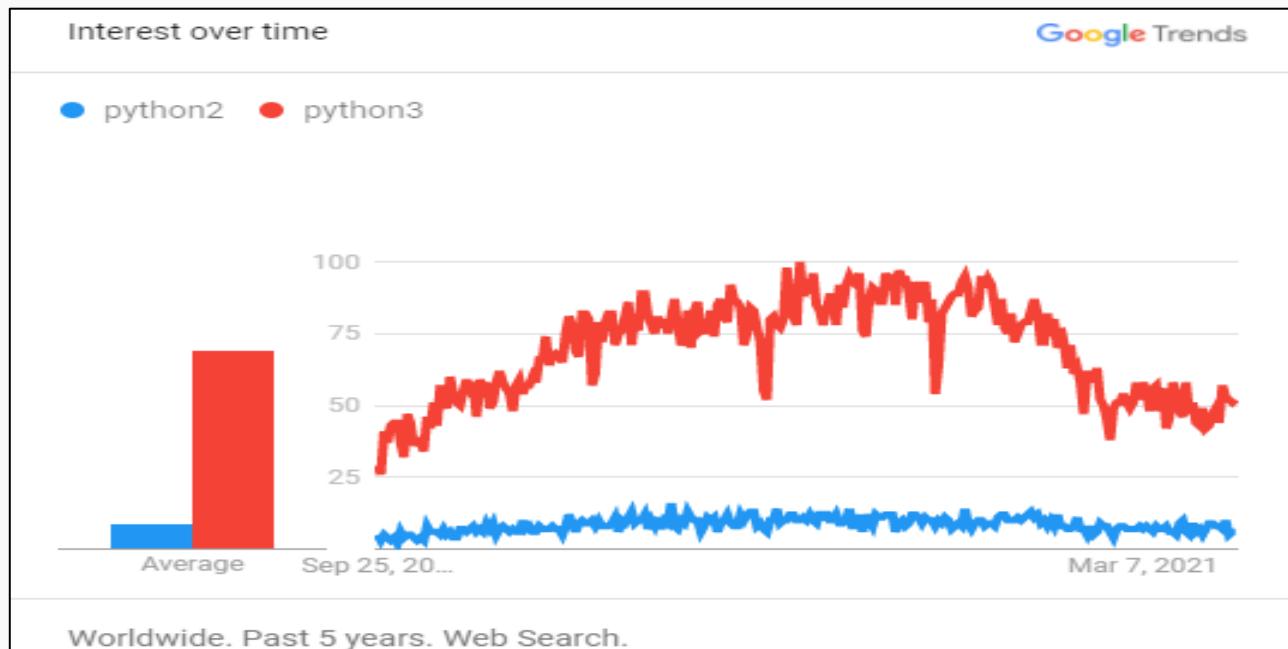
Why python?

- Python is a beginner-friendly language due to its simplified syntax which is close to natural language.
- It has been designed to write clear, logical code for small and large-scale projects.
- It is used extensively in the industry for software development, data science, artificial intelligence, and machine learning.

Setting Up Python

Python3 vs Python2

- Python2 is an outdated version of python. Python3 is the current version and is in-demand.
- In this course, we will be learning Python3 .



Setting Up Python – Windows (1/8)

- Visit: <https://www.python.org/>

The screenshot shows the Python.org homepage. The URL 'python.org' is highlighted in a red box in the browser's address bar. The page features a dark blue header with the Python logo and navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is a search bar with 'Search' and 'GO' buttons. A sidebar on the left contains a code snippet for generating a Fibonacci series:

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
    >>>     a, b = 0, 1
    >>>     while a < n:
    >>>         print(a, end=' ')
    >>>         a, b = b, a+b
    >>>     print()
    >>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

The main content area includes a section titled "Functions Defined" with text about Python's function definition capabilities and a link to "More about defining functions in Python 3". At the bottom, there is a call-to-action: "Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)". The browser's taskbar at the bottom shows various pinned icons and the system tray indicates it's 12:00 am on 26/09/2021.

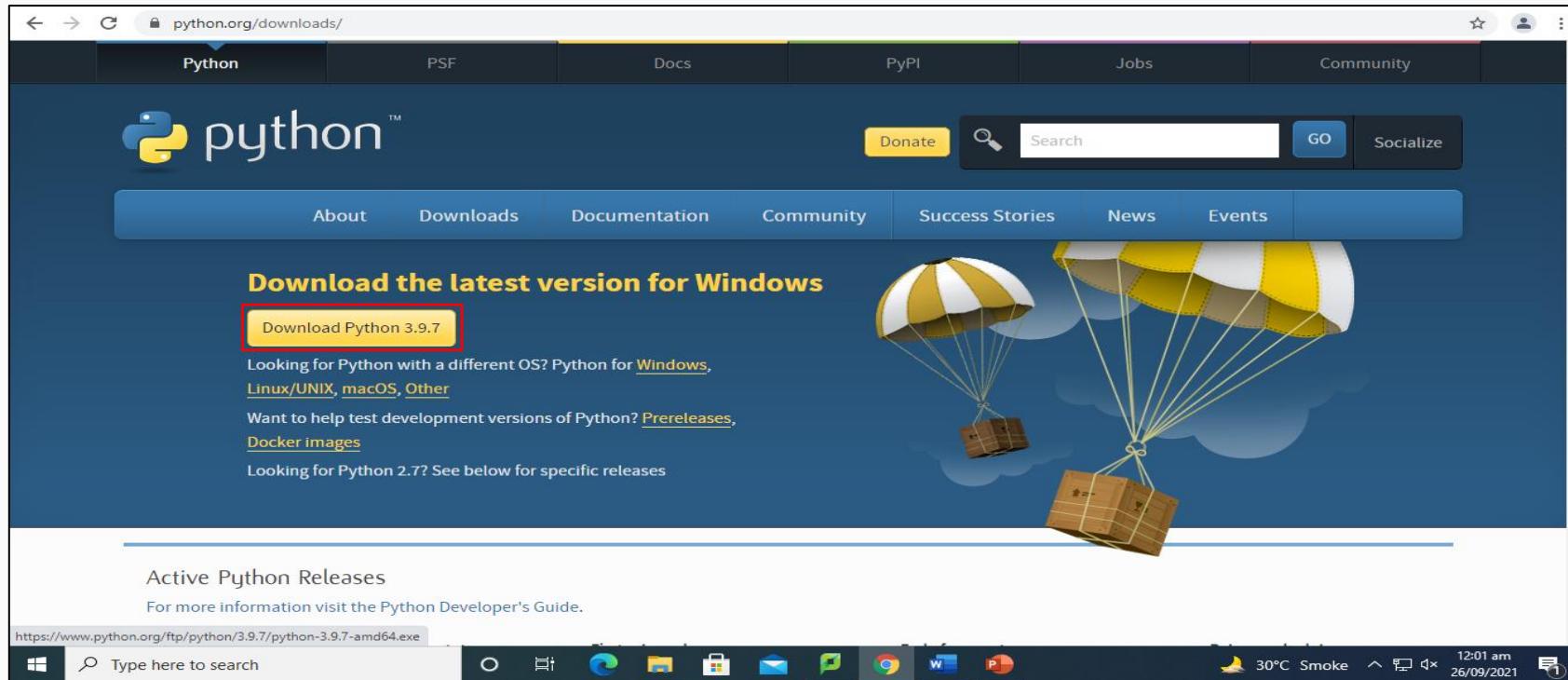
Setting Up Python – Windows (2/8)

- Click on Downloads.

The screenshot shows a web browser displaying the Python.org homepage. The URL in the address bar is <https://www.python.org/>. The page has a dark blue header with the Python logo and navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header, there's a search bar with a magnifying glass icon and a 'GO' button. A red box highlights the 'Downloads' tab, which is currently active. To the left of the 'Downloads' tab, there's a snippet of Python code demonstrating a Fibonacci sequence. The 'Downloads' menu includes options like All releases, Source code, Windows, macOS, Other Platforms, License, and Alternative Implementations. The main content area features a section titled 'Download for Windows' with a link to 'Python 3.9.7'. A note states that Python 3.9+ cannot be used on Windows 7 or earlier. Below this, it says Python can be used on many operating systems and environments, with a link to 'View the full list of downloads'. At the bottom of the page, a banner reads: 'Python is a programming language that lets you work quickly and integrate systems more effectively.' followed by a 'Learn More' link. The browser taskbar at the bottom shows the address bar, a search bar, and various pinned icons for Microsoft Edge, File Explorer, and other applications. The system tray on the right shows the date (26/09/2021), time (12:01 am), and battery status (30°C Smoke).

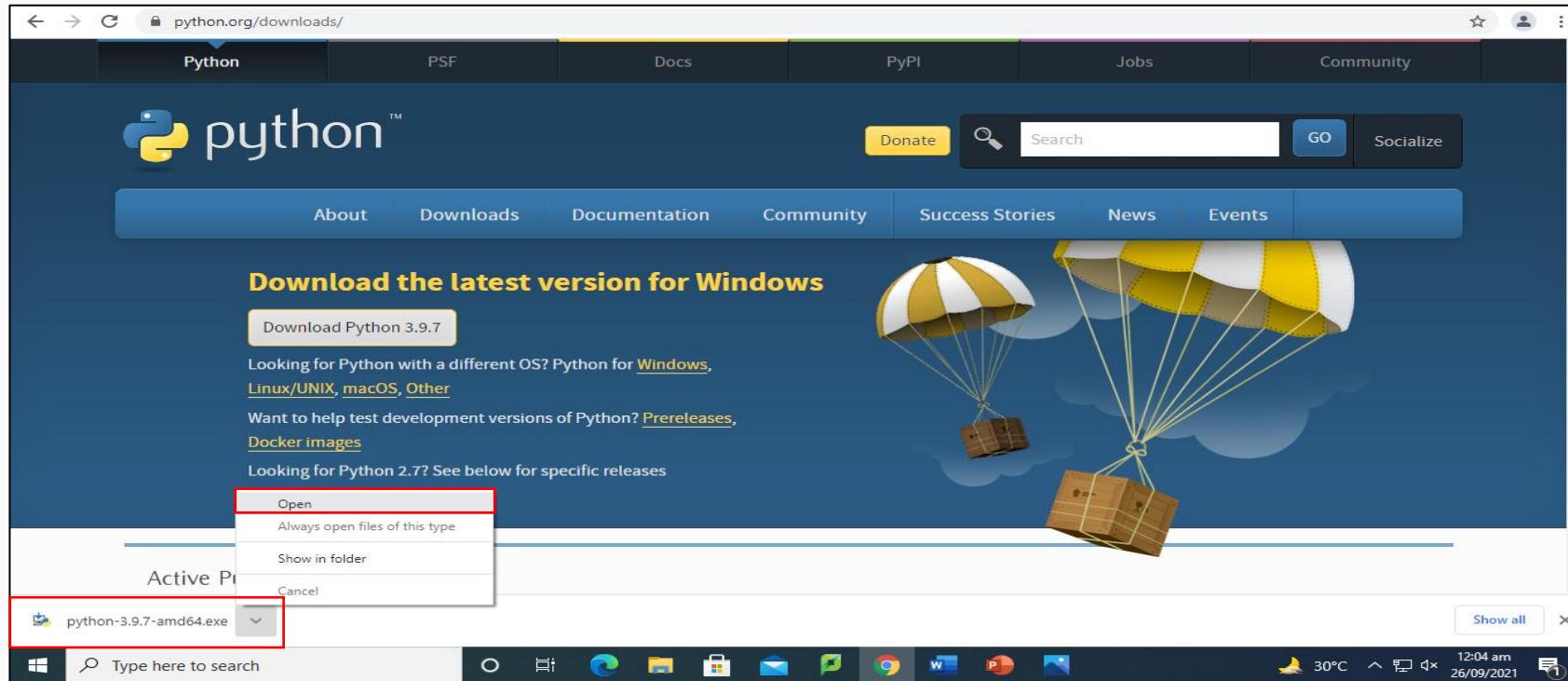
Setting Up Python – Windows (3/8)

- This will show you the latest version of Python3 for your operating system. Download it.



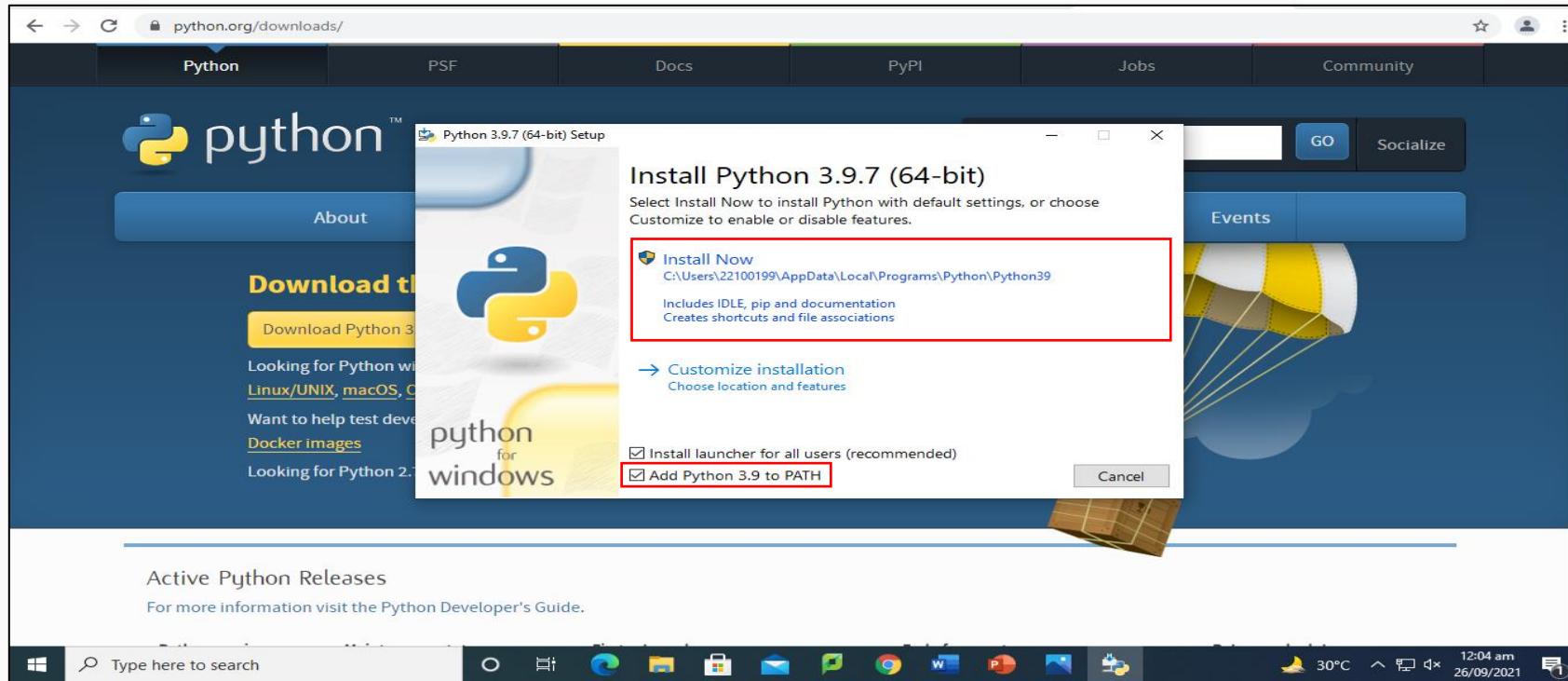
Setting Up Python – Windows (4/8)

- Once downloaded, click Open.



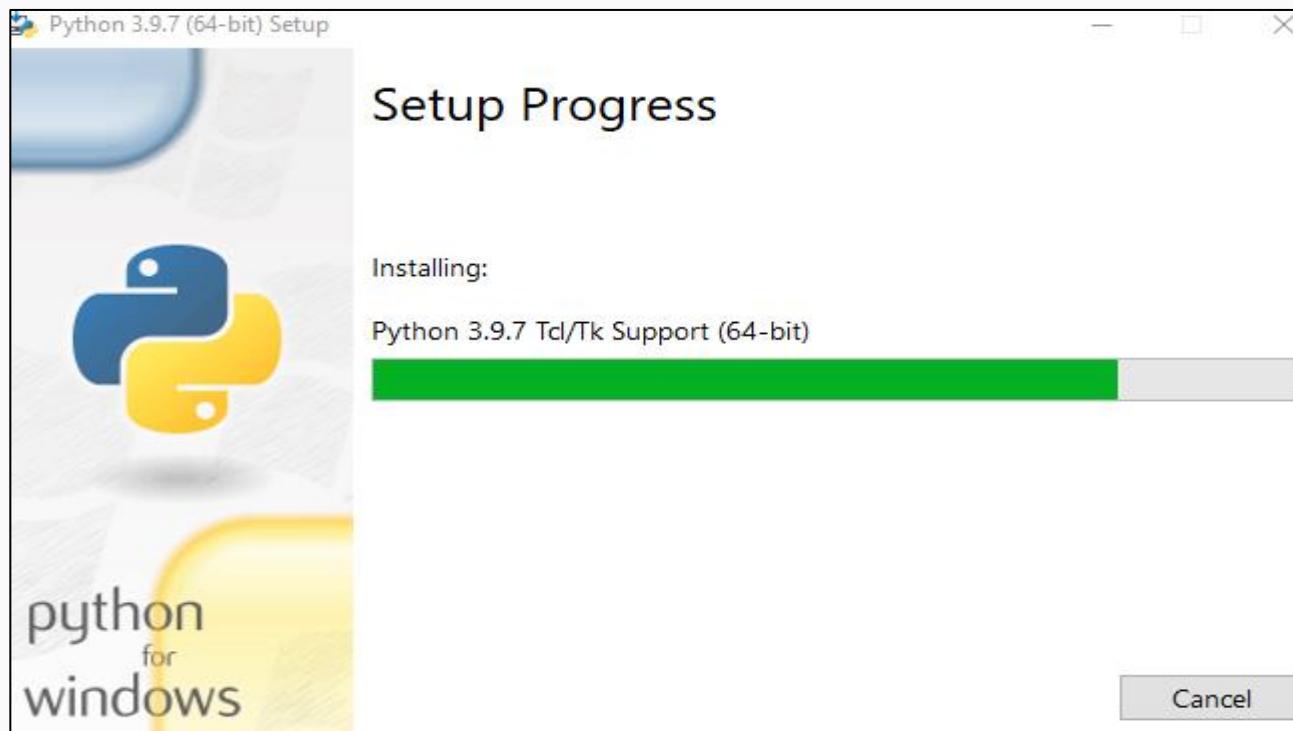
Setting Up Python – Windows (5/8)

- Installation wizard will open up. Make sure that 'Add Python 3.9 to Path' is checked. Click on Install Now.



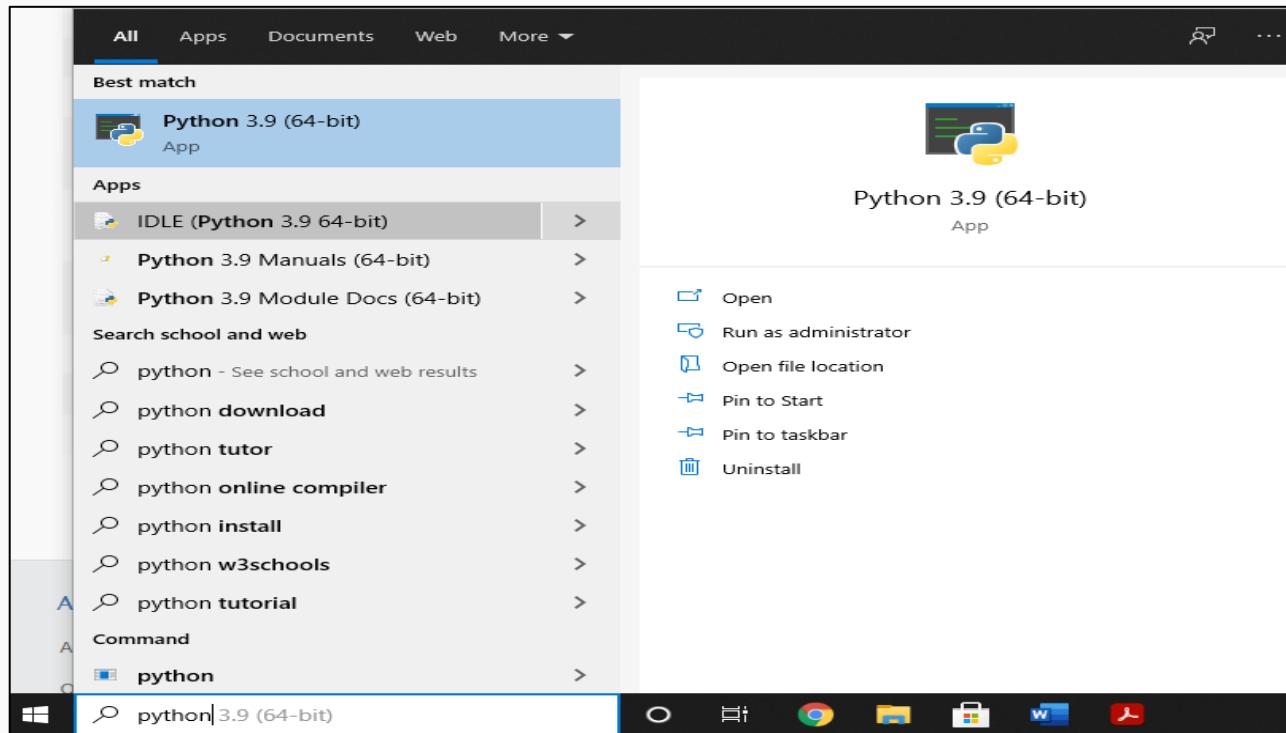
Setting Up Python – Windows (6/8)

- It will take sometime to install.



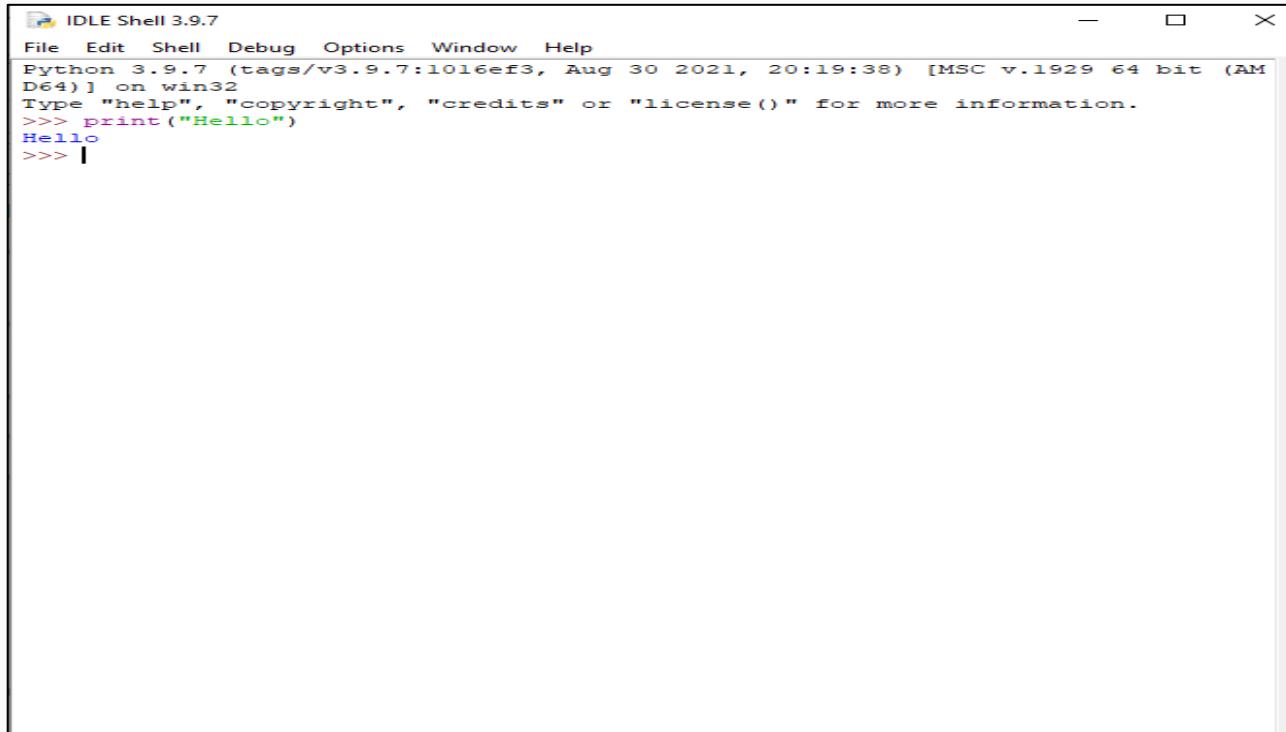
Setting Up Python – Windows (7/8)

- Once installed, type python in your search bar and click on ‘IDLE (Python 3.9 64-bit)’.



Setting Up Python – Windows (8/8)

- This will open up Python IDLE. Type print("Hello") and press Enter to execute.

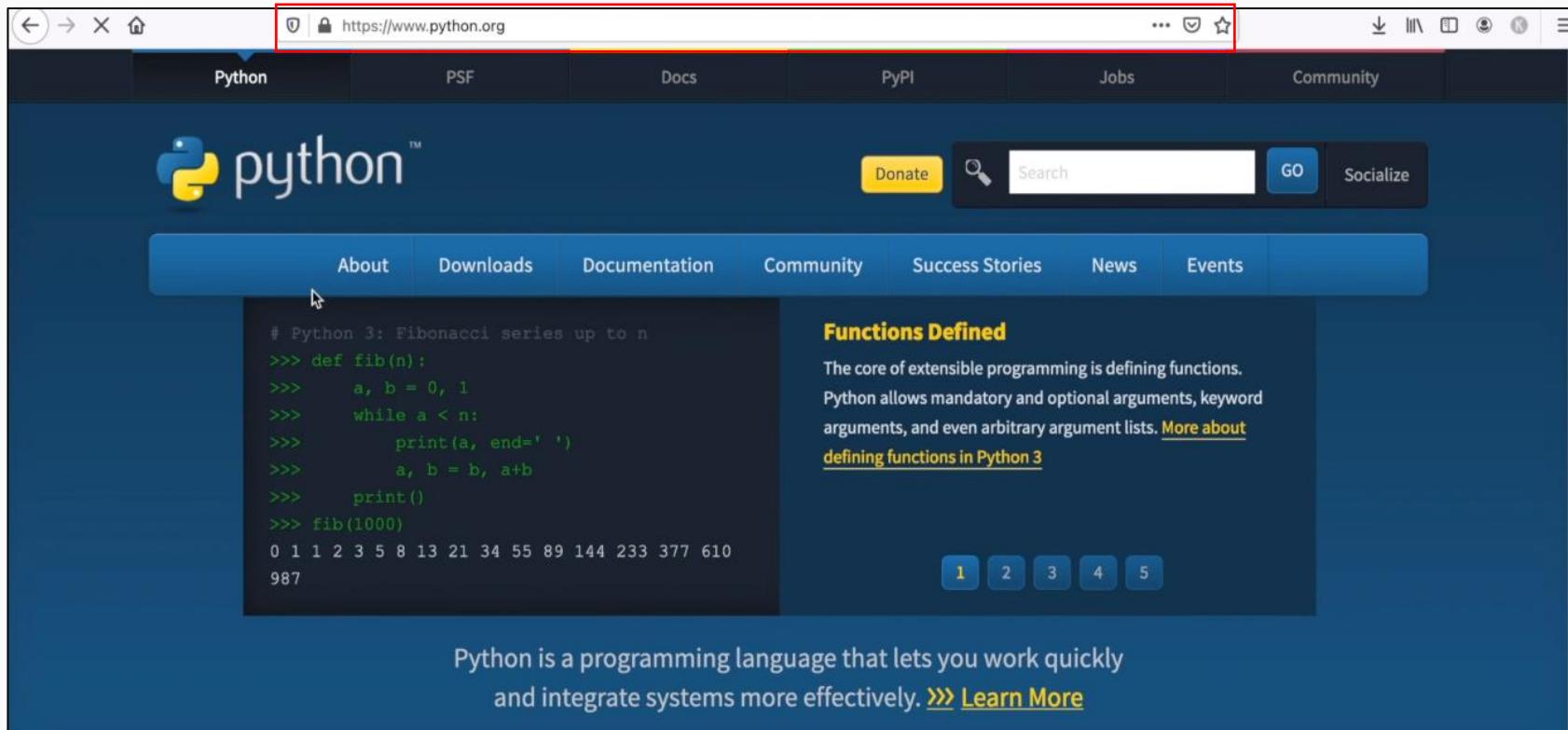


The screenshot shows the Python IDLE Shell window. The title bar reads "IDLE Shell 3.9.7". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python version information: "Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32". It also shows the copyright message: "Type "help", "copyright", "credits" or "license()" for more information.". A command line input shows the user typing "print('Hello')", followed by the output "Hello" and a new line indicator ">>> |".

```
File Edit Shell Debug Options Window Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> |
```

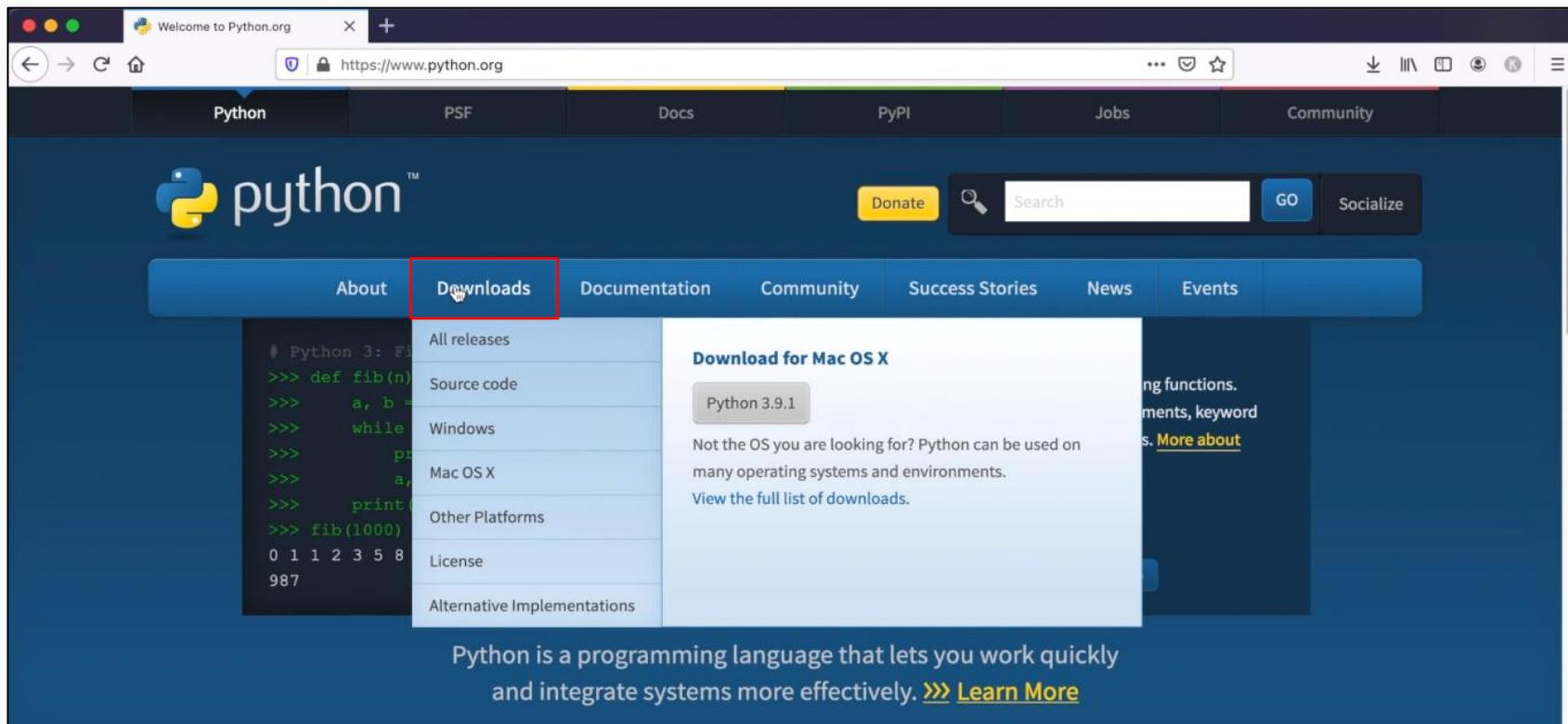
Setting Up Python – Mac (1/13)

- Visit: <https://www.python.org/>



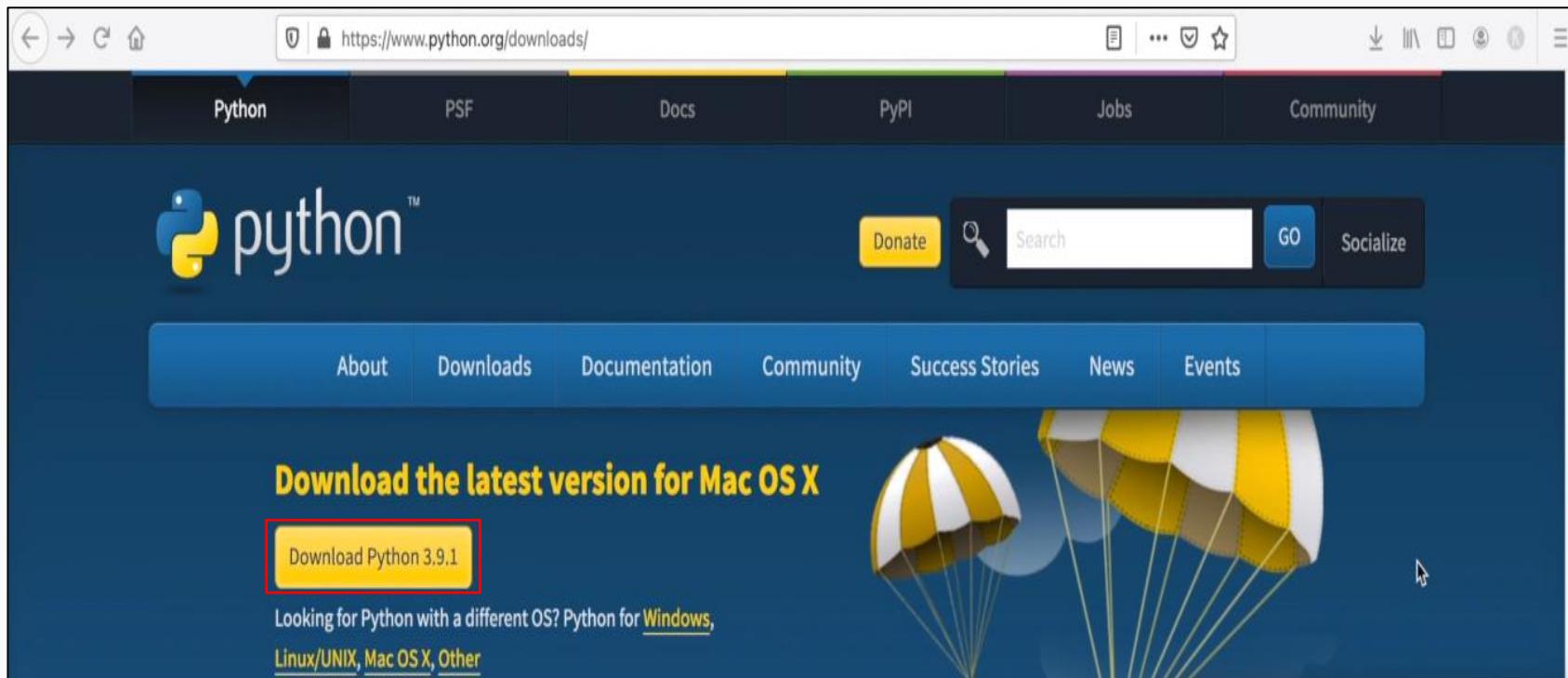
Setting Up Python – Mac (2/13)

- Click on Downloads.



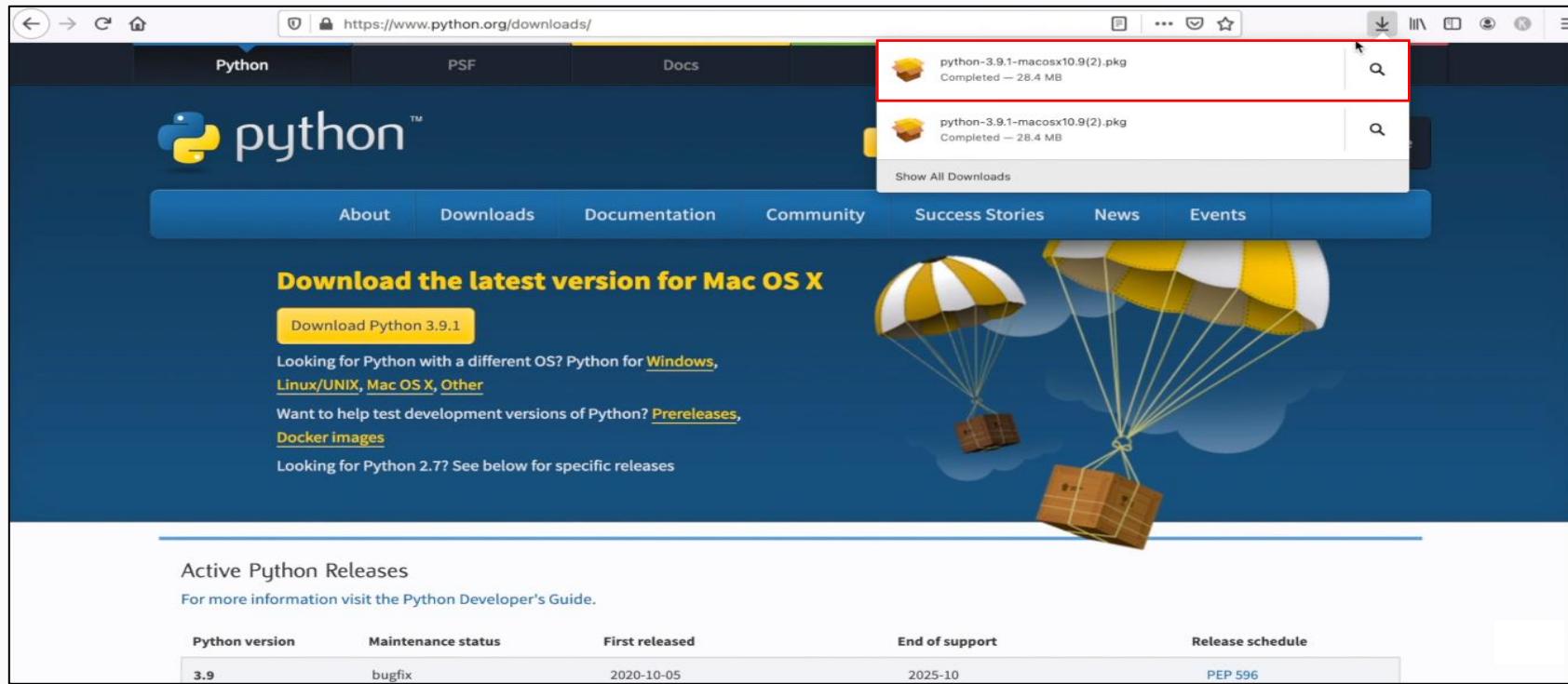
Setting Up Python – Mac (3/13)

- This will show you the latest version of Python3 for your operating system. Download it.



Setting Up Python – Mac (4/13)

- Once downloaded, click on the file.



Setting Up Python – Mac (5/13)

- Click continue.



Setting Up Python – Mac (6/13)

- Click continue.



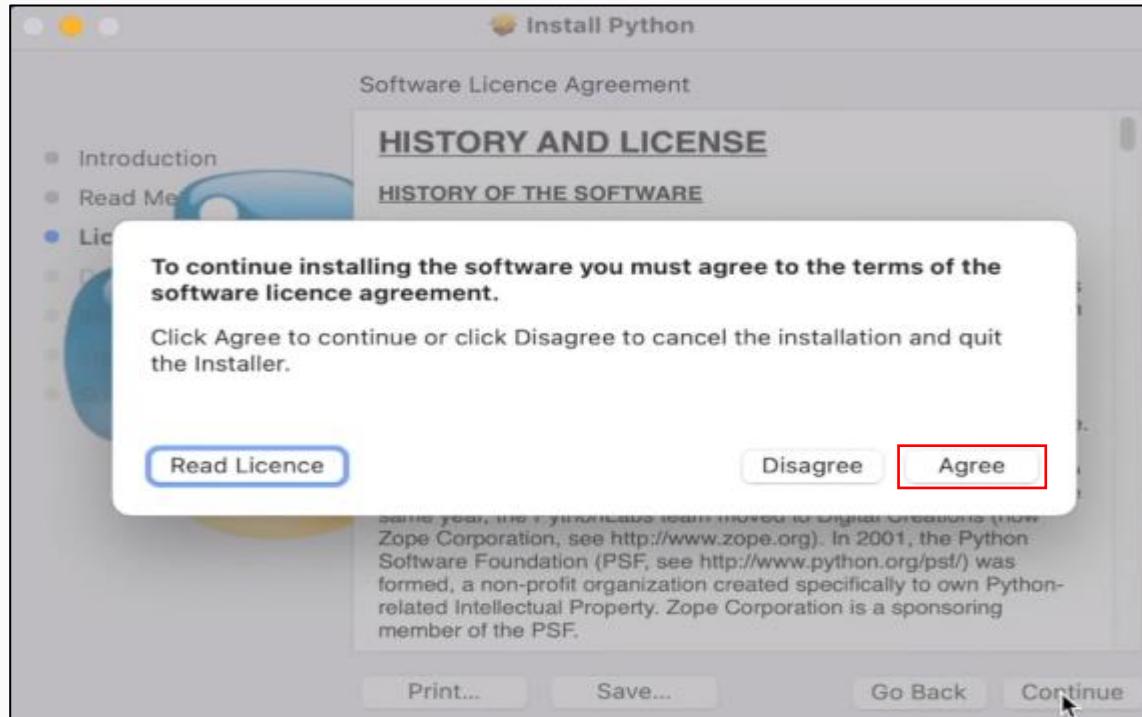
Setting Up Python – Mac (7/13)

- Click continue.



Setting Up Python – Mac (8/13)

- Read and accept the license agreement.



Setting Up Python – Mac (9/13)

- Click continue.



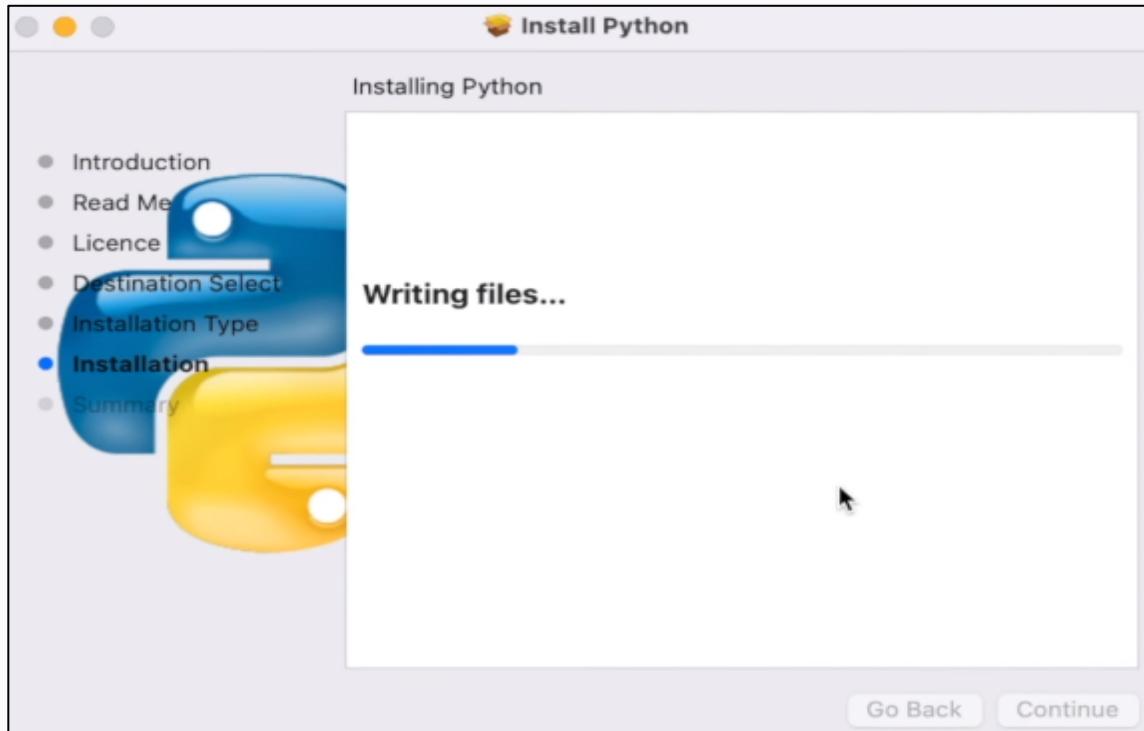
Setting Up Python – Mac (10/13)

- Click install. If prompted, enter your password.



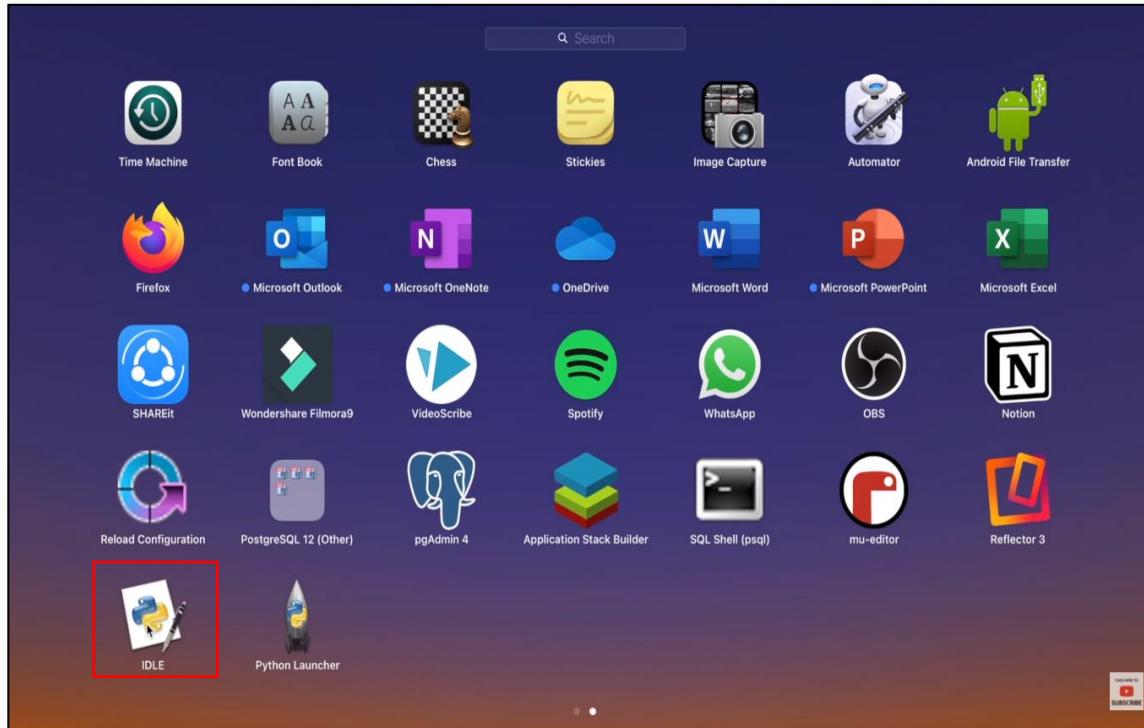
Setting Up Python – Mac (11/13)

- Wait for the installation to finish.



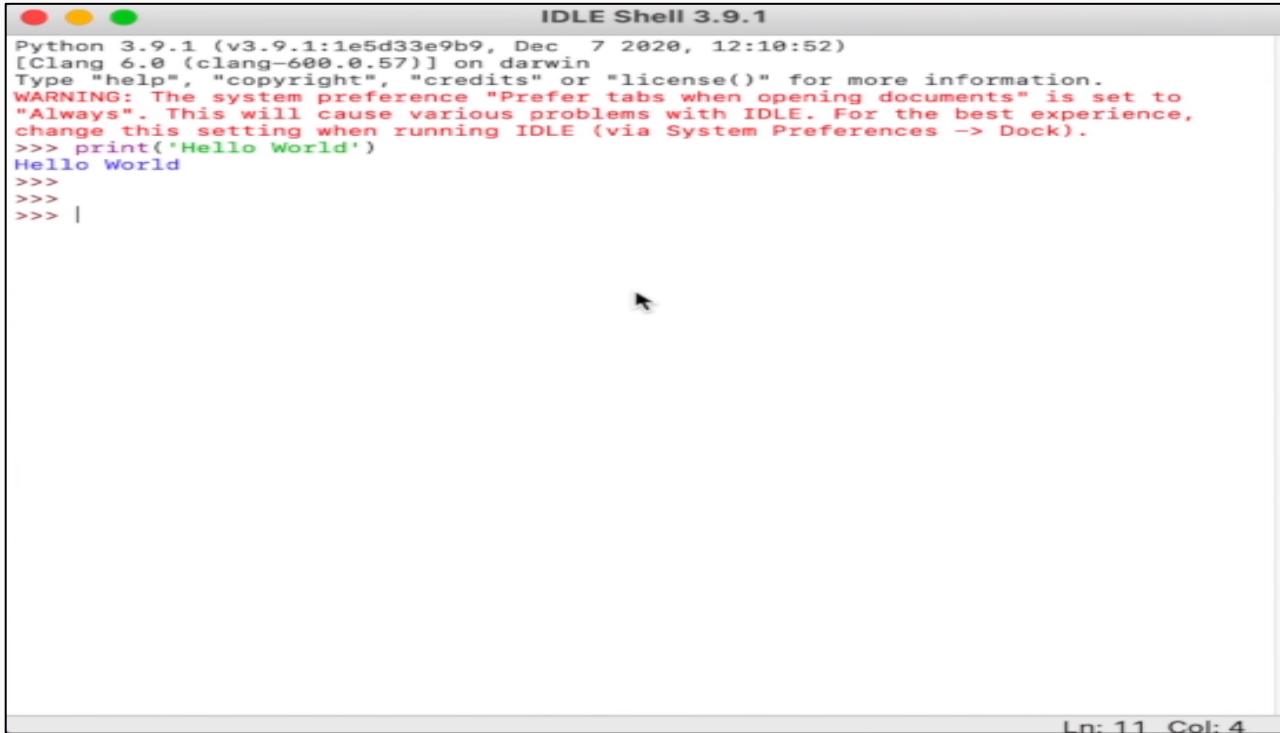
Setting Up Python – Mac (12/13)

- Once installed, search for IDLE in your spotlight and click on it.



Setting Up Python – Mac (13/13)

- This will open up Python IDLE. Type print("Hello World") and press Enter to execute.



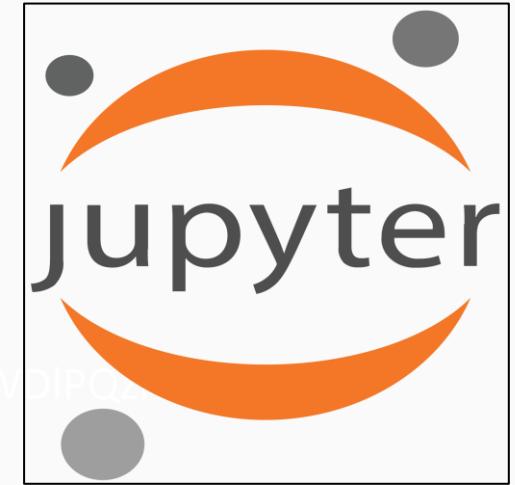
The screenshot shows a window titled "IDLE Shell 3.9.1". The window contains the following text:

```
Python 3.9.1 (v3.9.1:1e5d33e9b9, Dec  7 2020, 12:10:52)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
WARNING: The system preference "Prefer tabs when opening documents" is set to
"Always". This will cause various problems with IDLE. For the best experience,
change this setting when running IDLE (via System Preferences -> Dock).
>>> print('Hello World')
Hello World
>>>
>>>
>>> |
```

A cursor arrow is visible in the bottom-left area of the window. In the bottom right corner, there is a status bar with the text "Ln: 11 Col: 4".

Jupyter Notebook

- Jupyter Notebook is a web application that allows you to create documents that may contain code, visualizations, and narrative text.
- It is very popular and is used extensively in data science, artificial intelligence, and machine learning.
- A notebook has the .ipynb extension

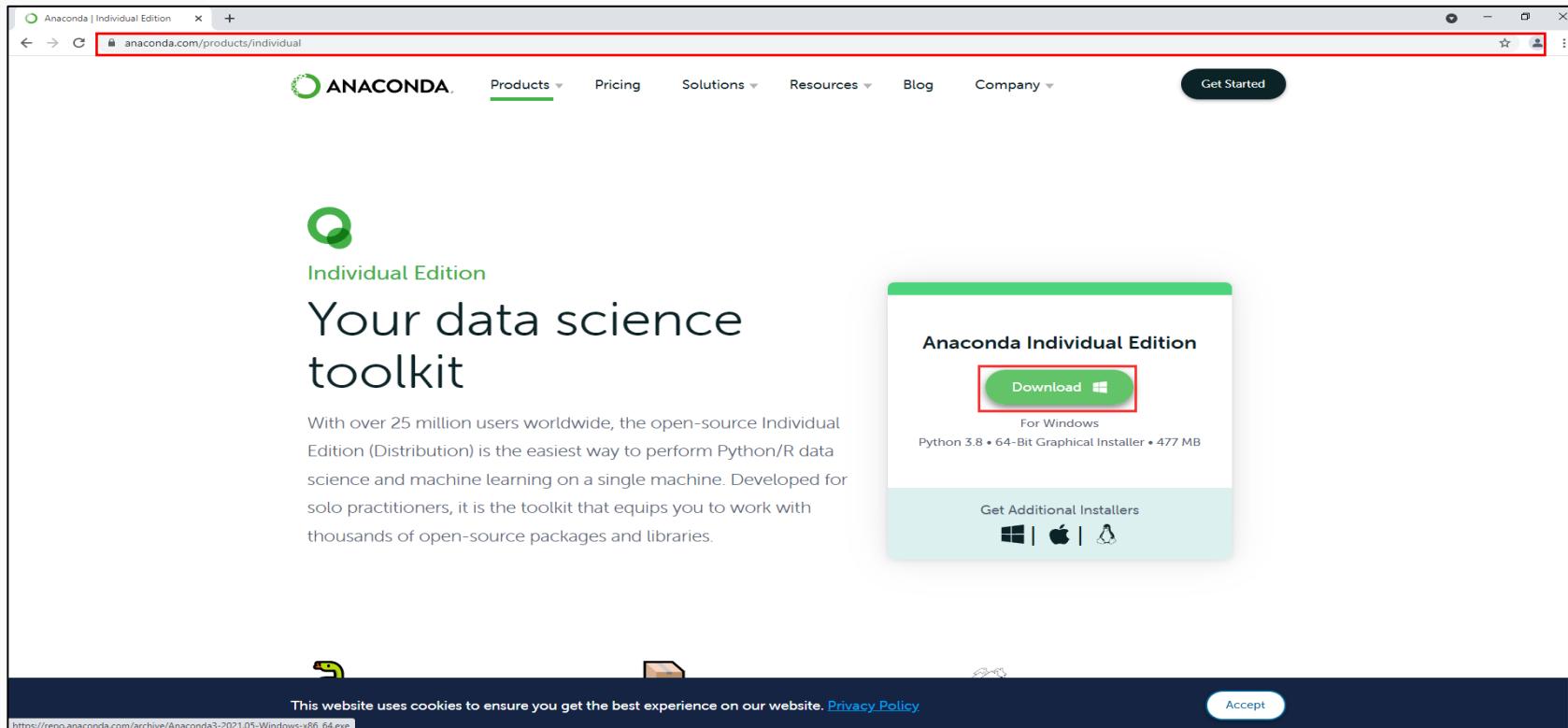


Alternatives to Jupyter Notebook

- Google Colaboratory is an alternative to Jupyter Notebook and does not require downloading anything.

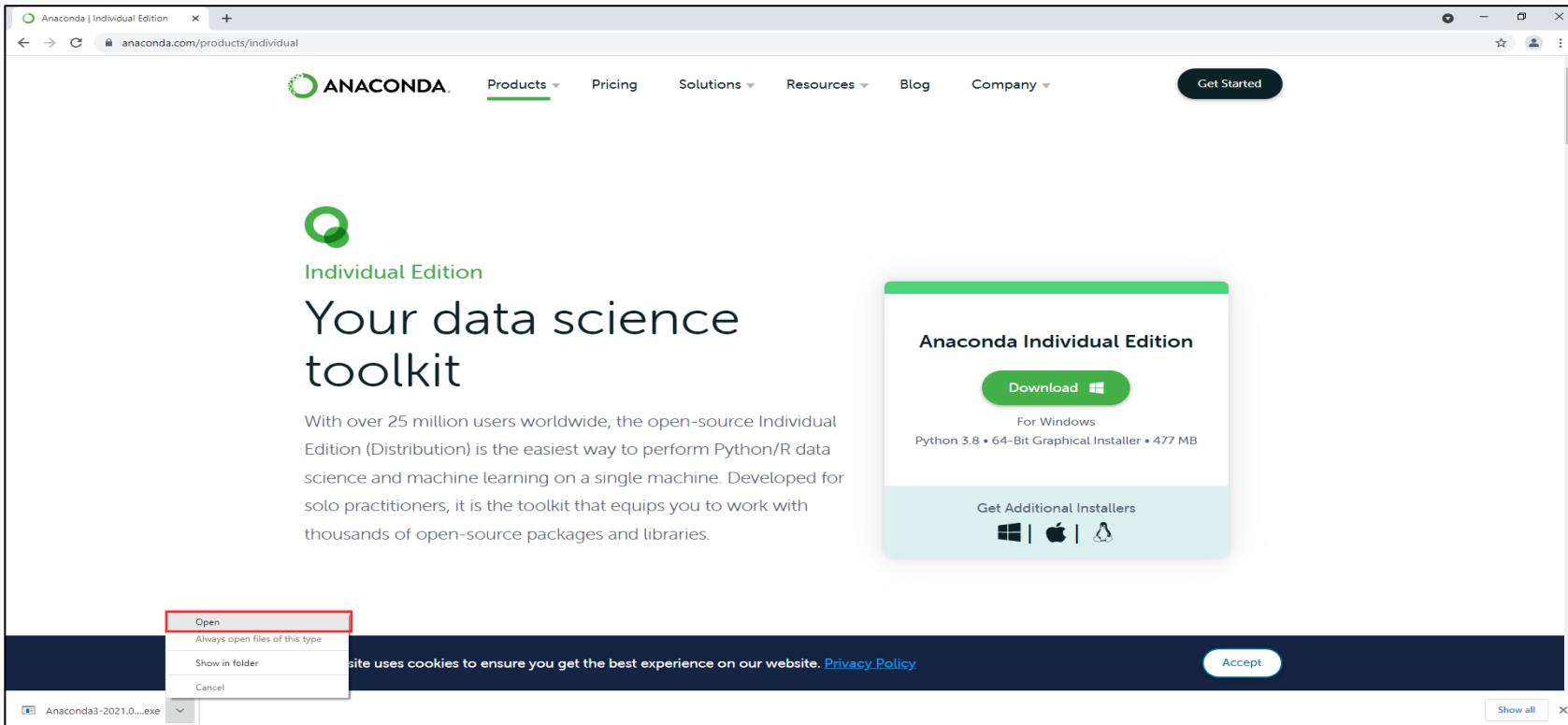
Anaconda for Windows (1/13)

- Visit: <https://www.anaconda.com/products/individual>
- Click on the Download button.



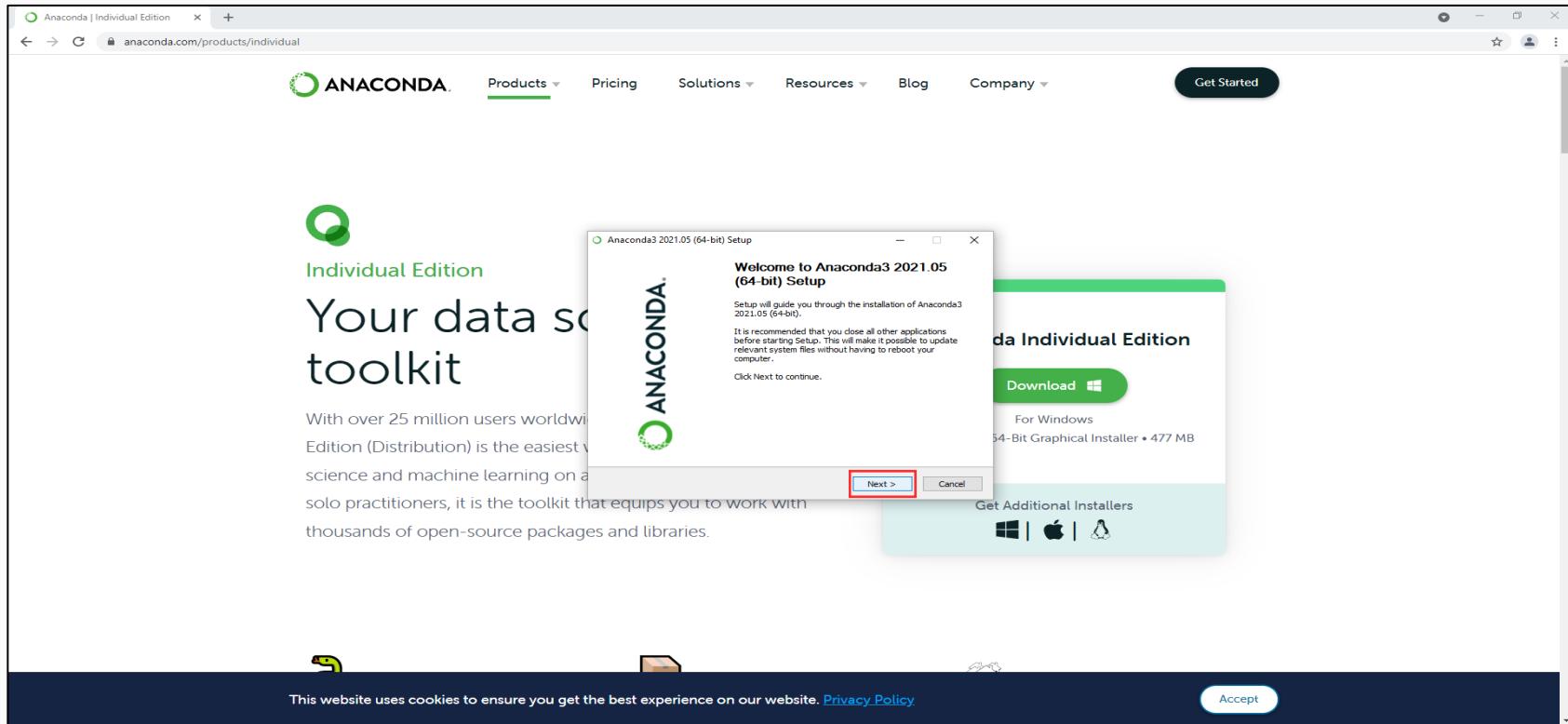
Anaconda for Windows (2/13)

- Once Downloaded, click on Open.



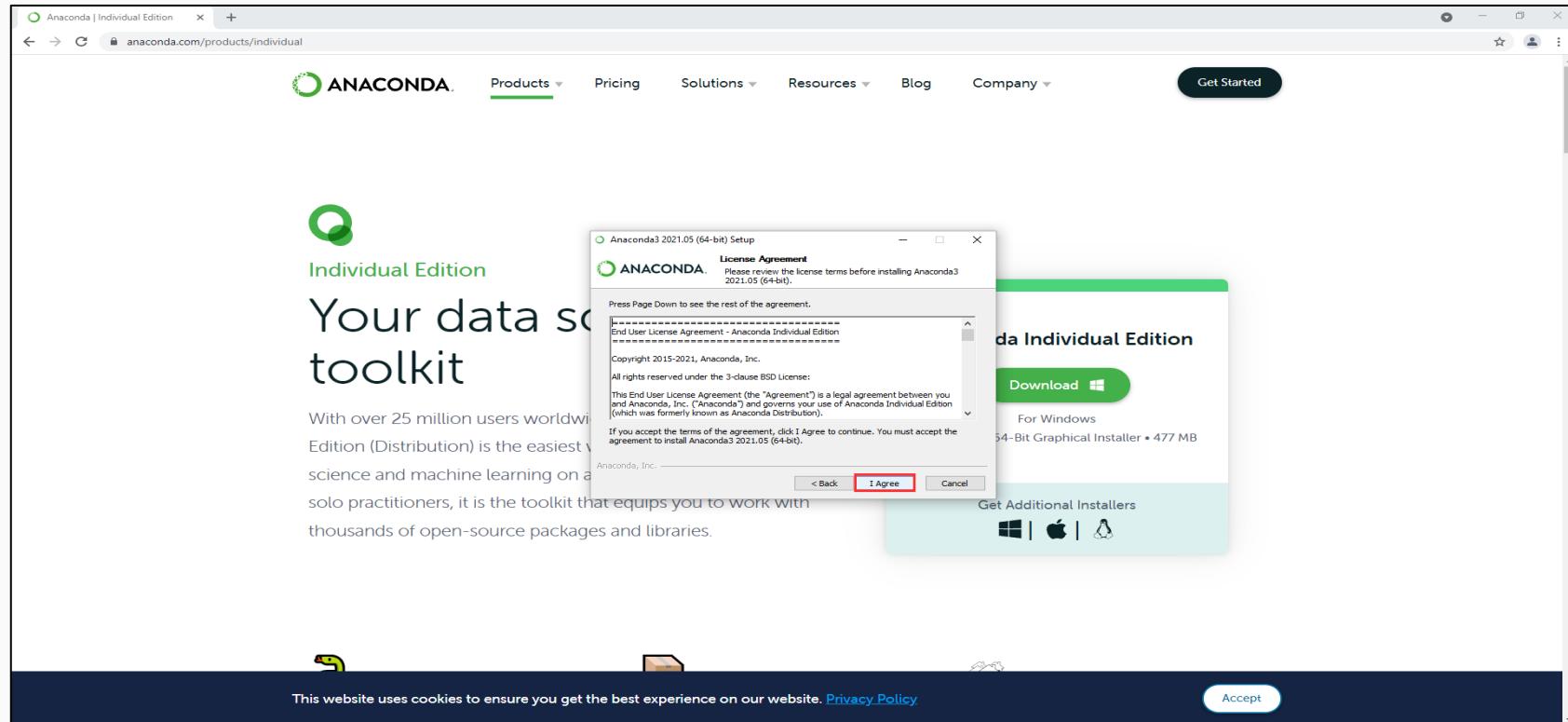
Anaconda for Windows (3/13)

- An installation Wizard will open up.
- Click on Next.



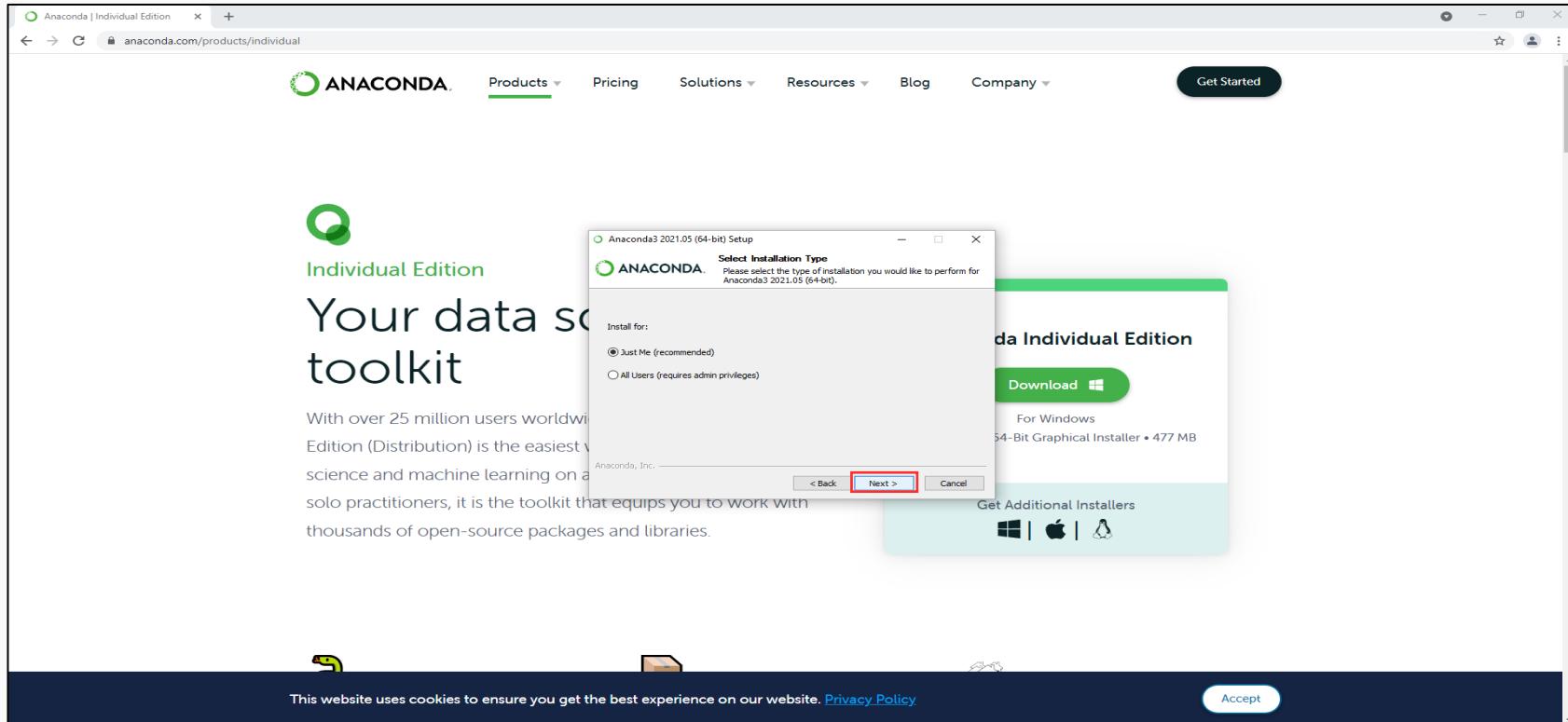
Anaconda for Windows (4/13)

- Read the License Agreement and click I Agree.



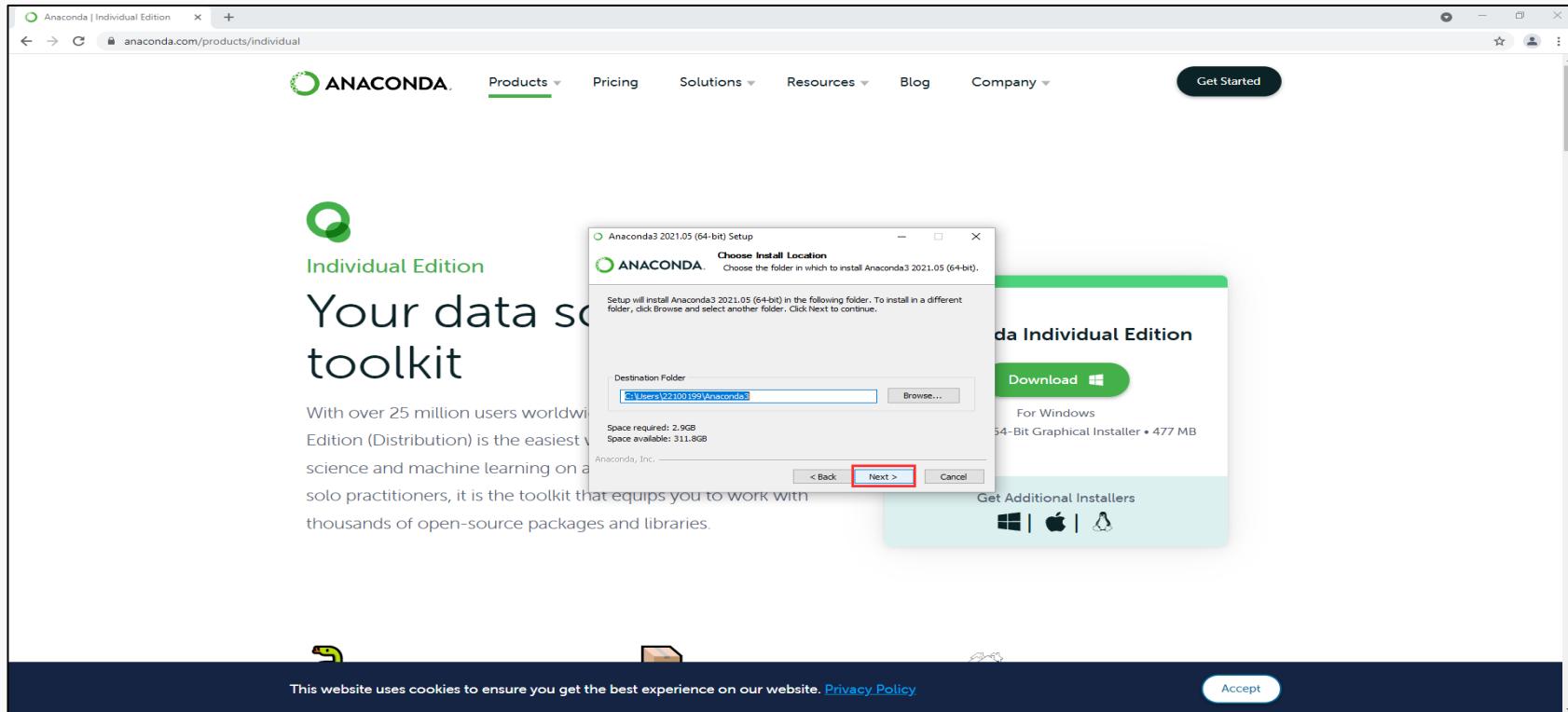
Anaconda for Windows (5/13)

- Click Next.



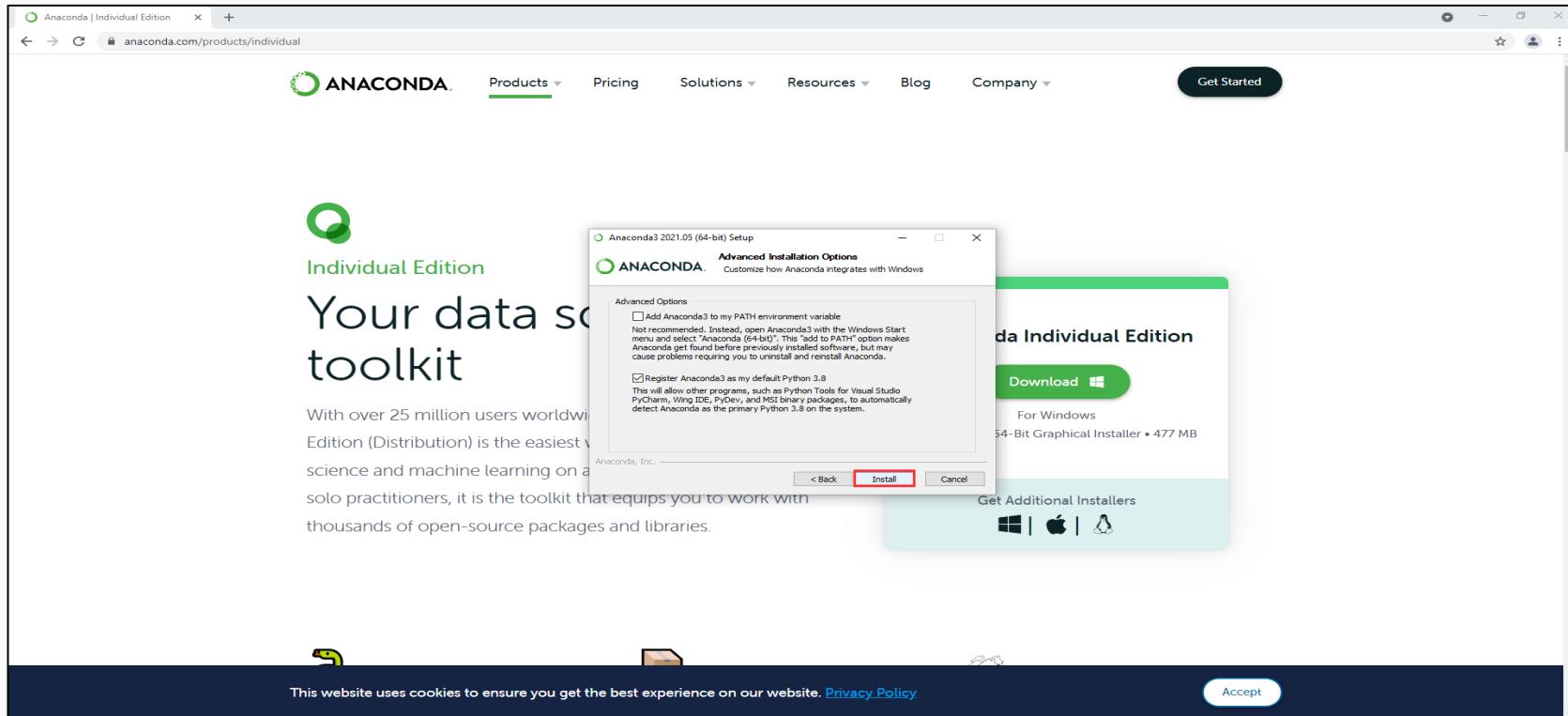
Anaconda for Windows (6/13)

- Choose the destination and click Next.



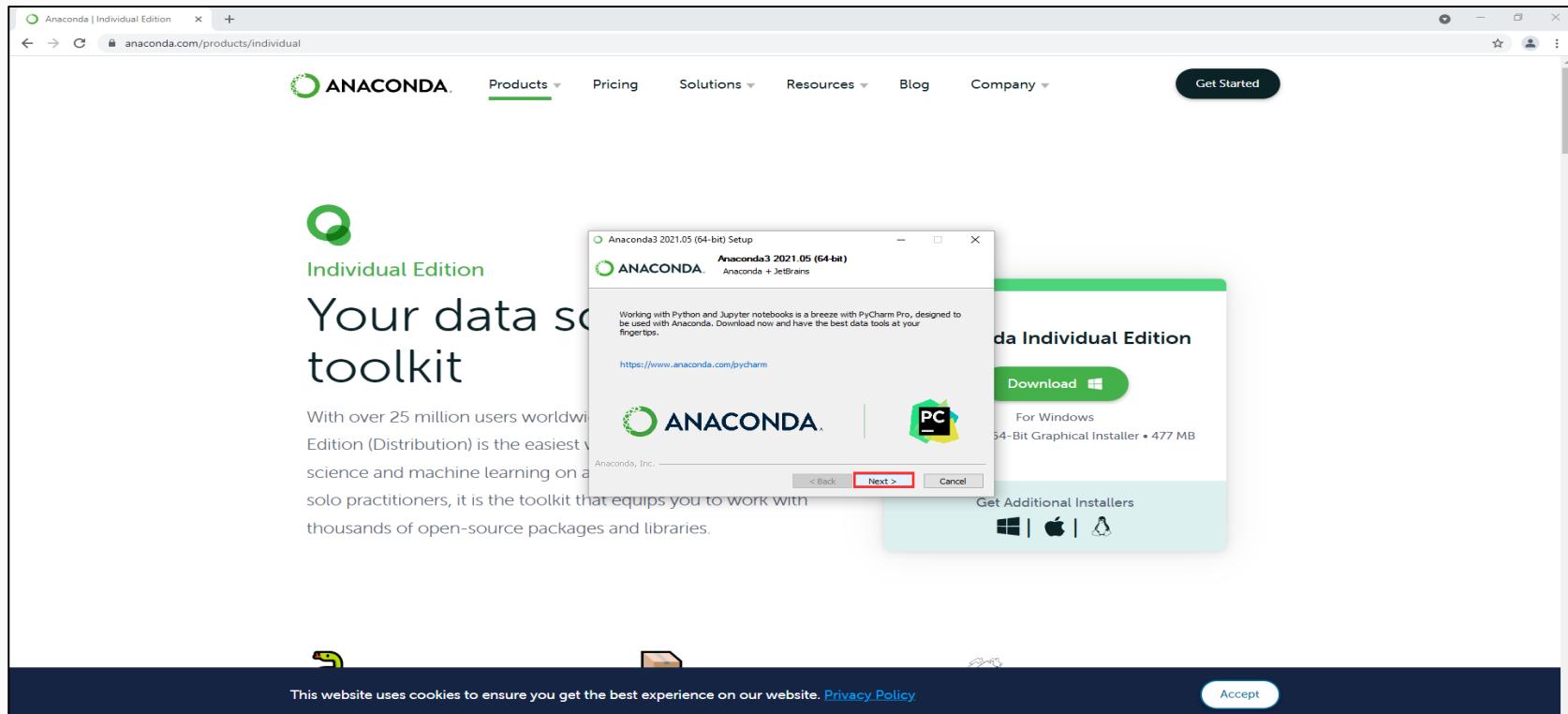
Anaconda for Windows (7/13)

- Click Install.



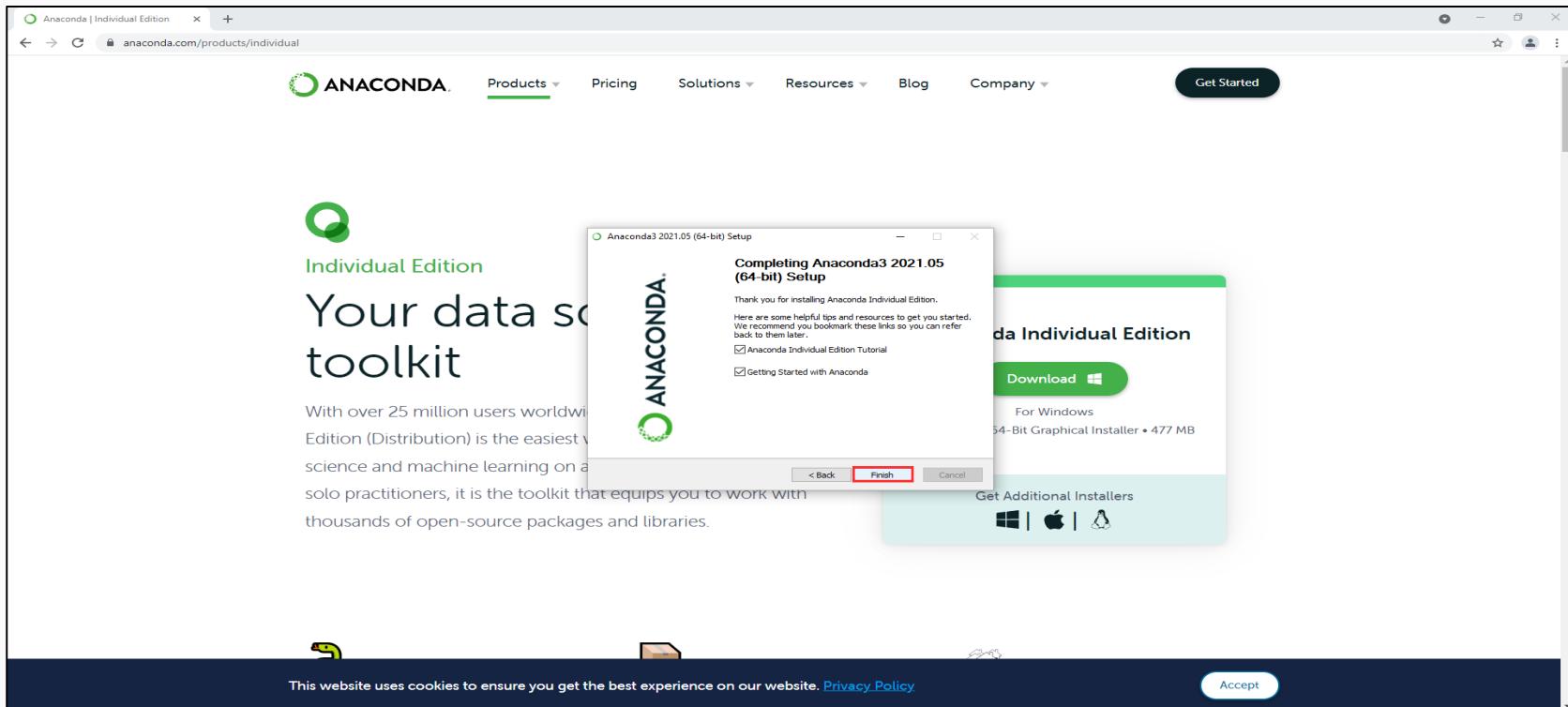
Anaconda for Windows (8/13)

- Wait for the installation to finish and click Next.



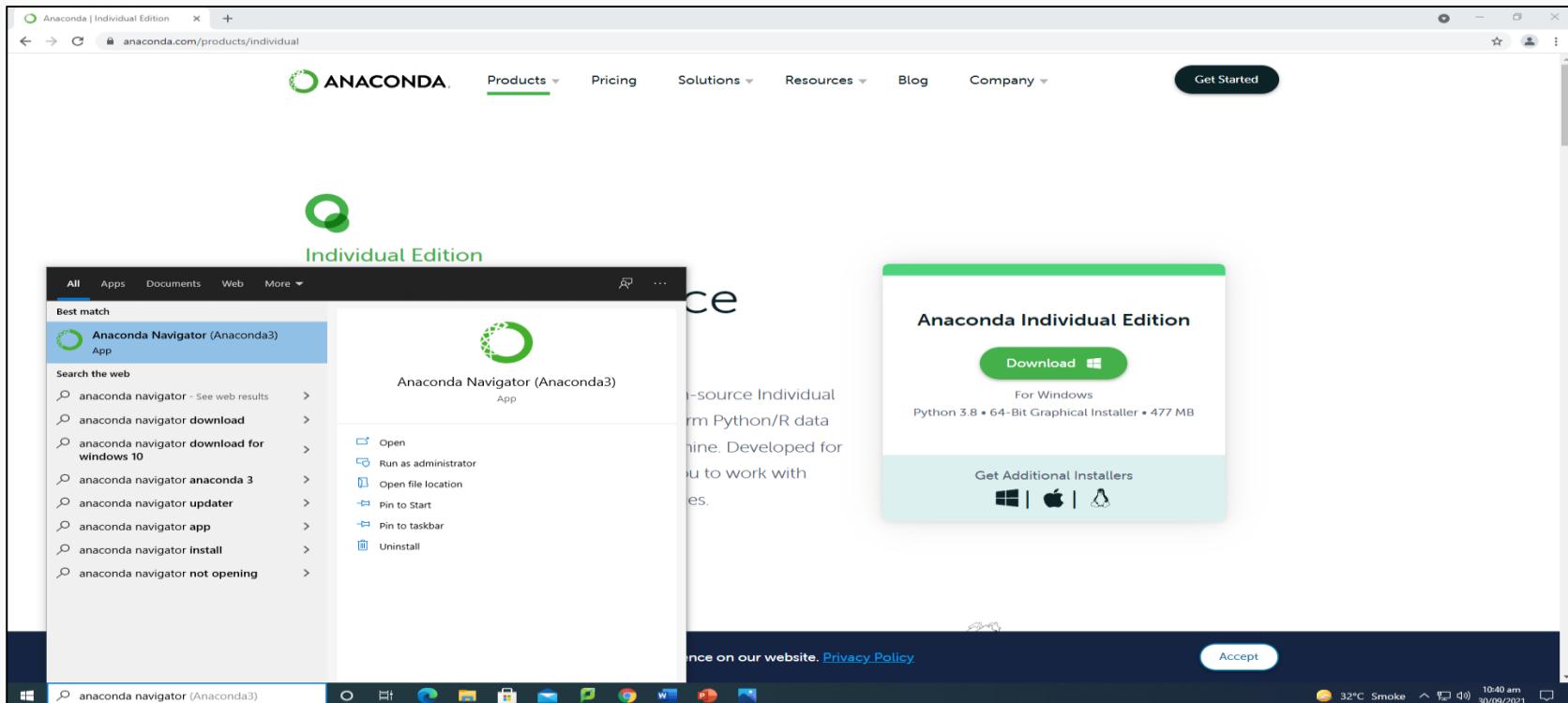
Anaconda for Windows (9/13)

- Click Finish.



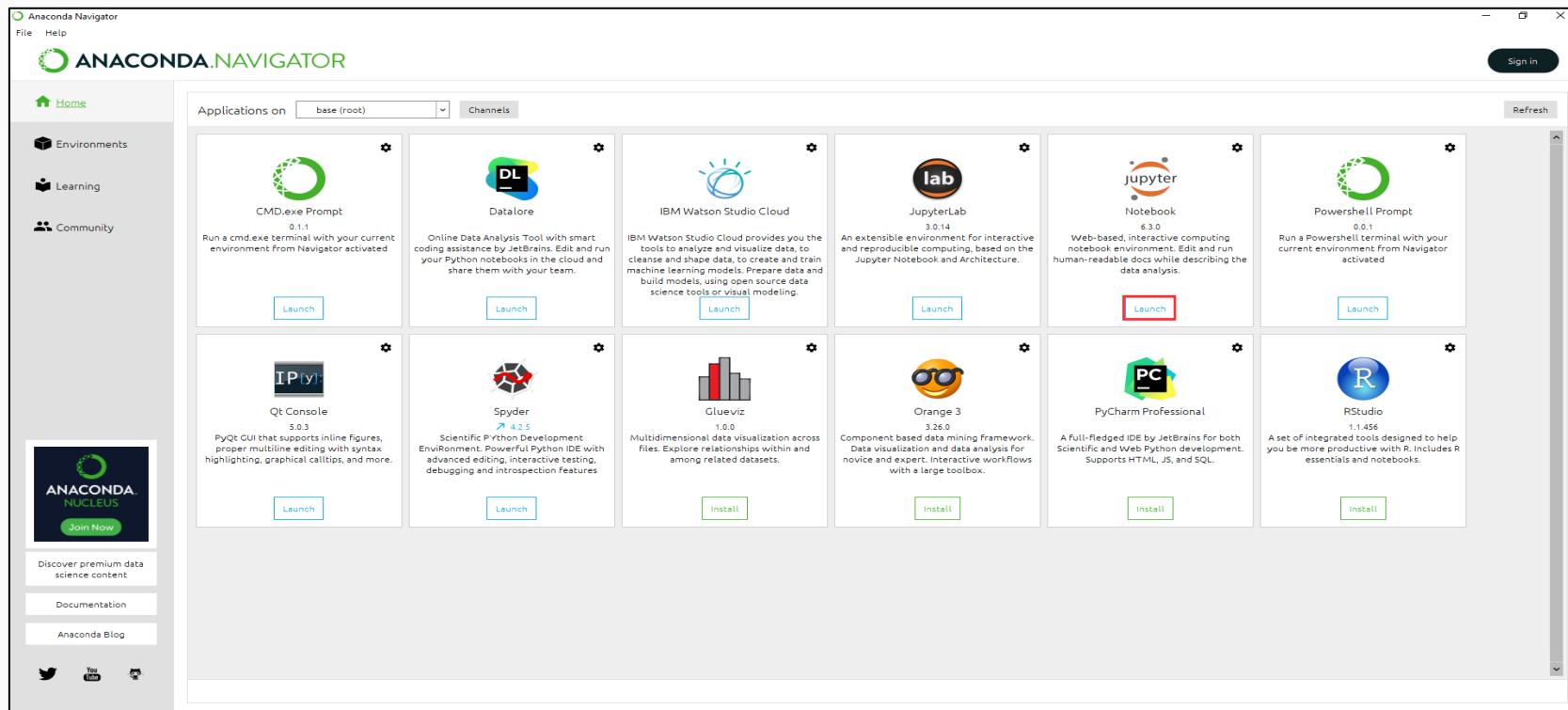
Anaconda for Windows (10/13)

- To launch Jupyter Notebook, search for anaconda navigator and open it.



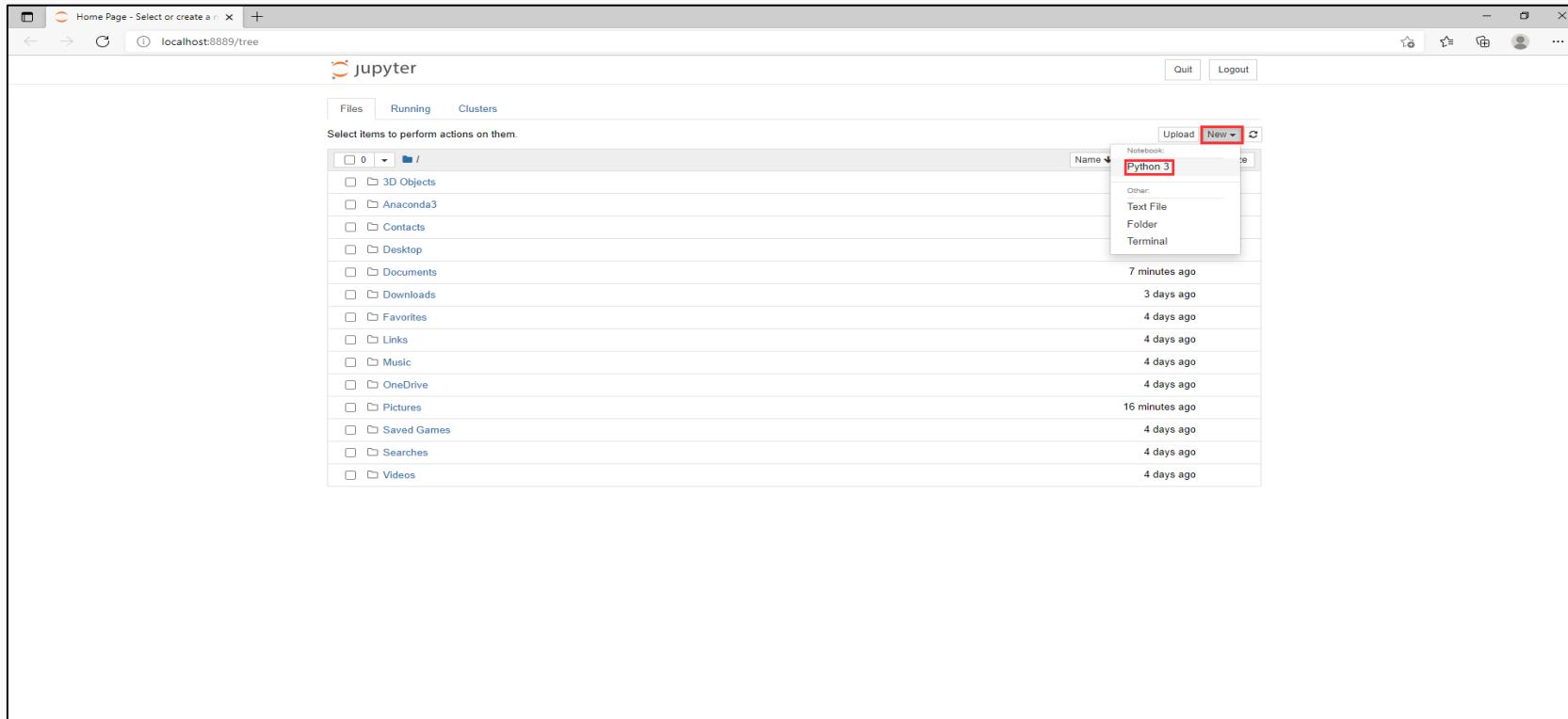
Anaconda for Windows (11/13)

- Following screen will open up.
- Launch Jupyter Notebook from here.



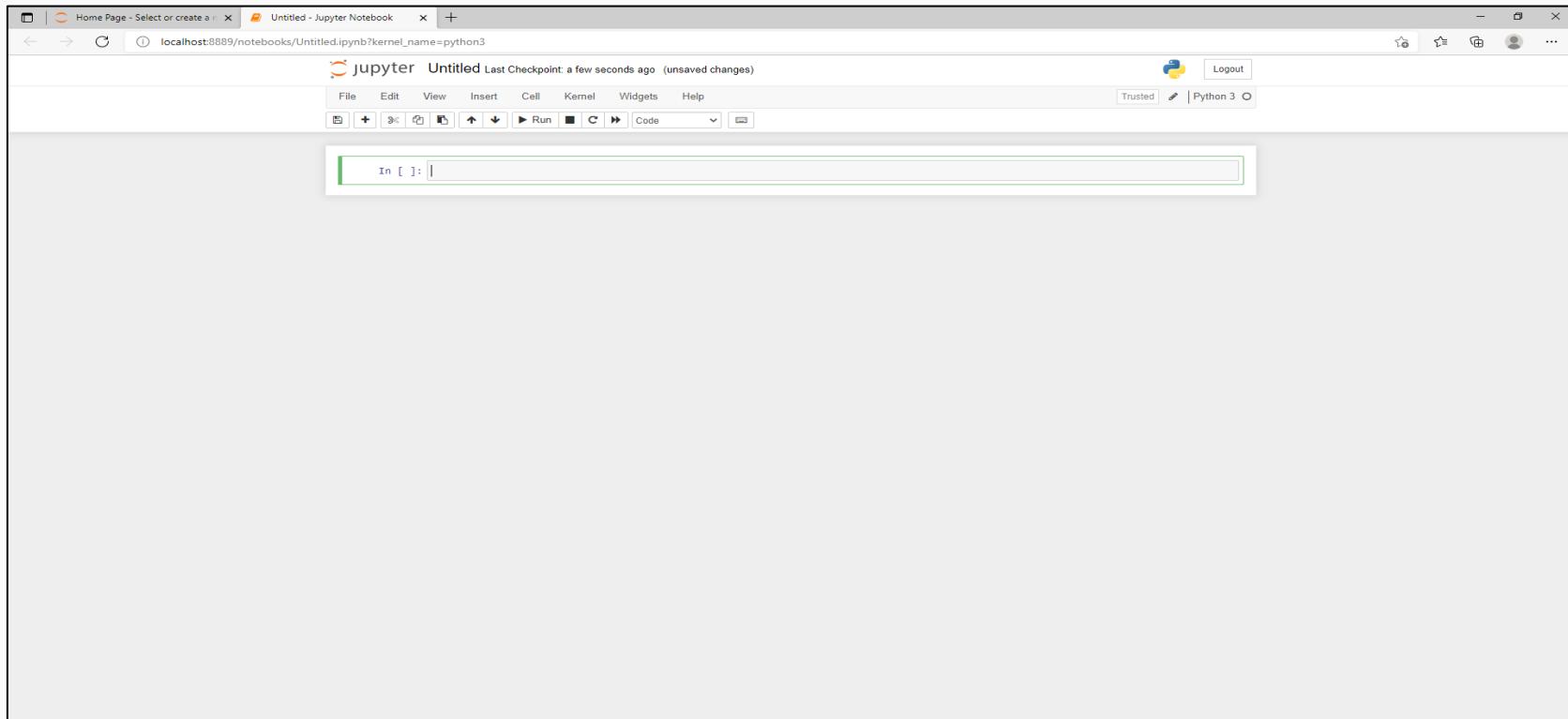
Anaconda for Windows (12/13)

- This will launch Jupyter Notebook in your default browser.
- Click on New -> Python3 to create a new Notebook.



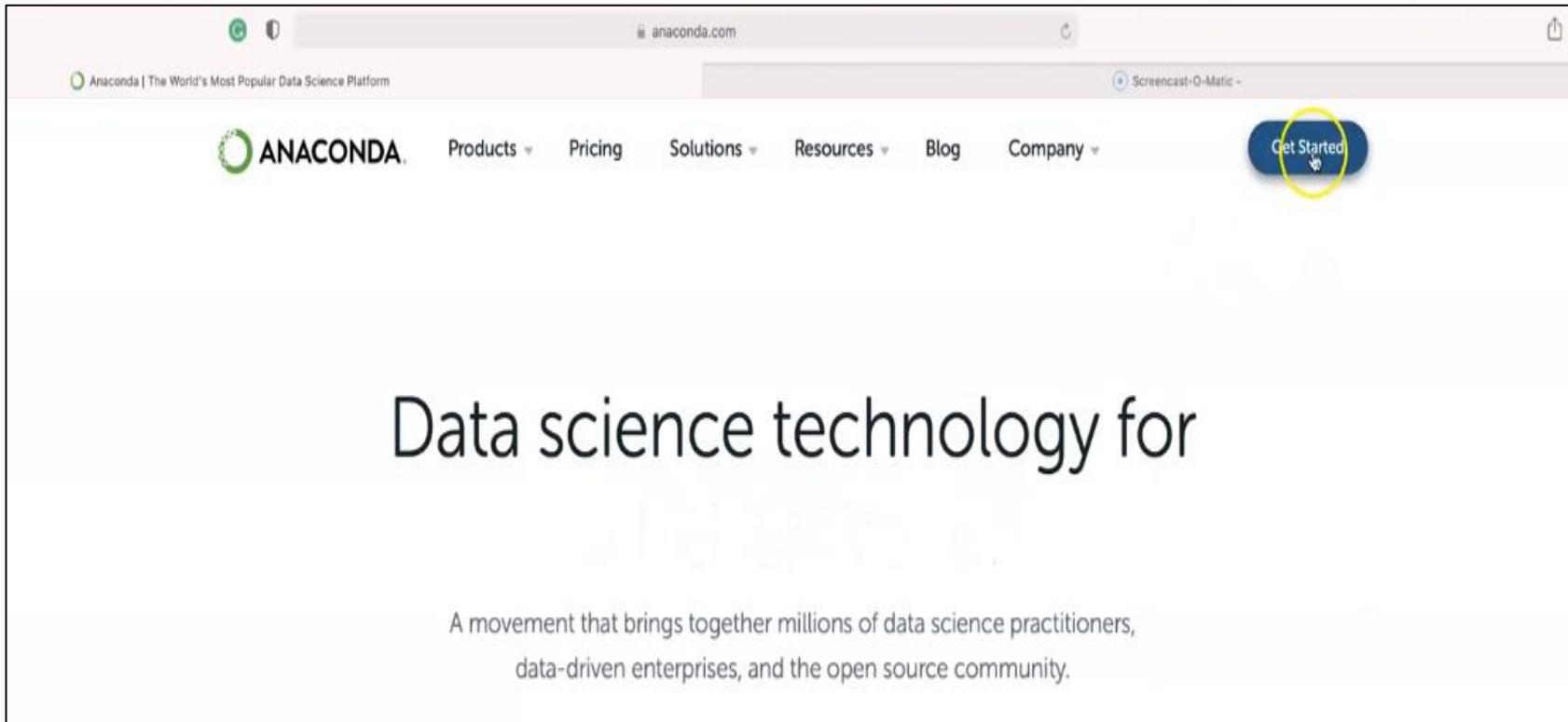
Anaconda for Windows (13/13)

- This is how a Notebook looks like.



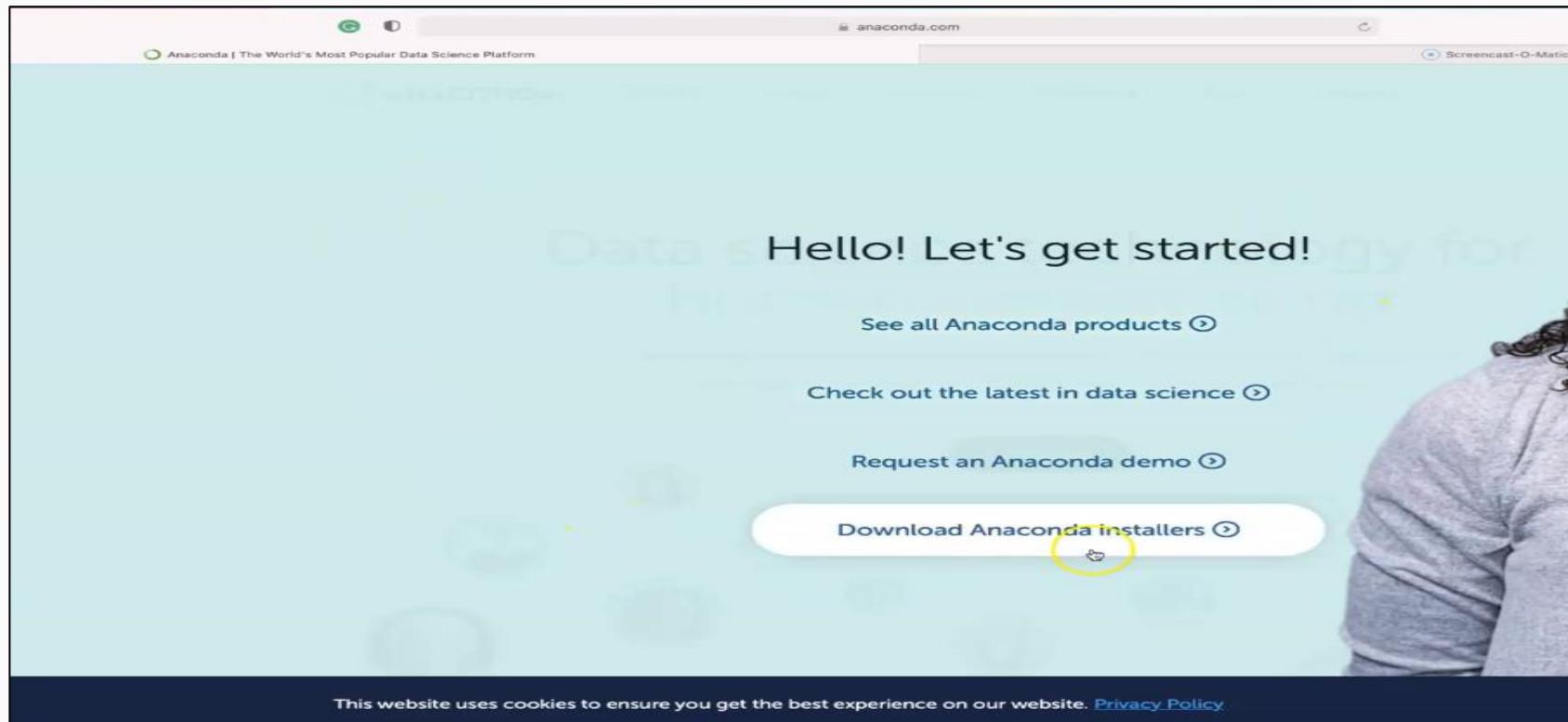
Anaconda for Mac (1/14)

- Visit: <https://www.anaconda.com/>
- Click on Get Started.



Anaconda for Mac (2/14)

- Click Download Anaconda Installers.



Anaconda for Mac (3/14)

- Download the graphical installer for Mac.

Anaconda Installers

Windows	MacOS	Linux
Python 3.8 64-Bit Graphical Installer (477 MB) 32-Bit Graphical Installer (409 MB)	Python 3.8 64-Bit Graphical Installer (440 MB)  64-Bit Command Line Installer (433 MB)	Python 3.8 64-Bit (x86) Installer (544 MB) 64-Bit (Power8 and Power9) Installer (285 MB) 64-Bit (AWS Graviton2 / ARM64) Installer (413 M) 64-bit (Linux on IBM Z & LinuxONE) Installer (292 M)

ADDITIONAL INSTALLERS

The archive has older versions of Anaconda Individual Edition installers. The Miniconda installer homepage can be found [here](#).

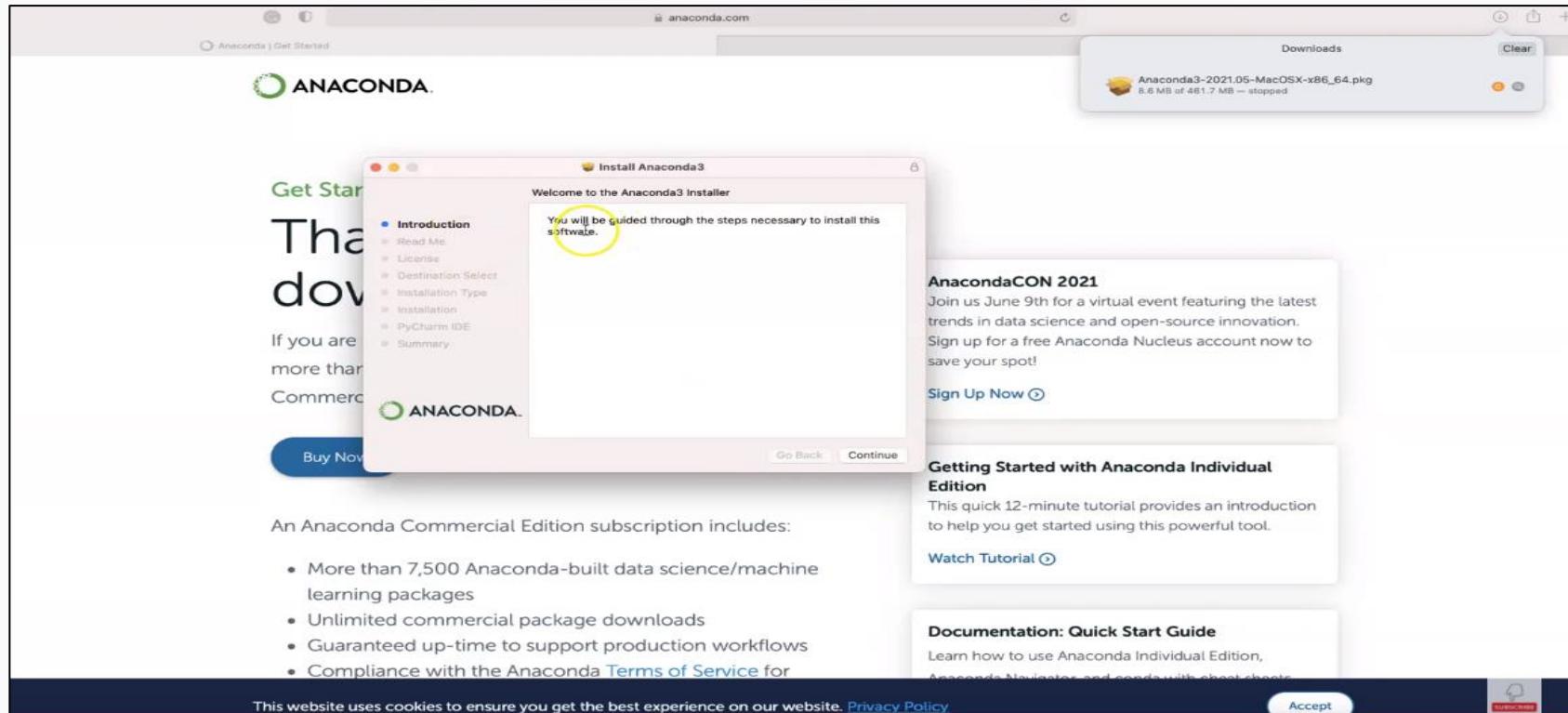
This website uses cookies to ensure you get the best experience on our website. [Privacy Policy](#)

Accept 

 SUBSCRIBE

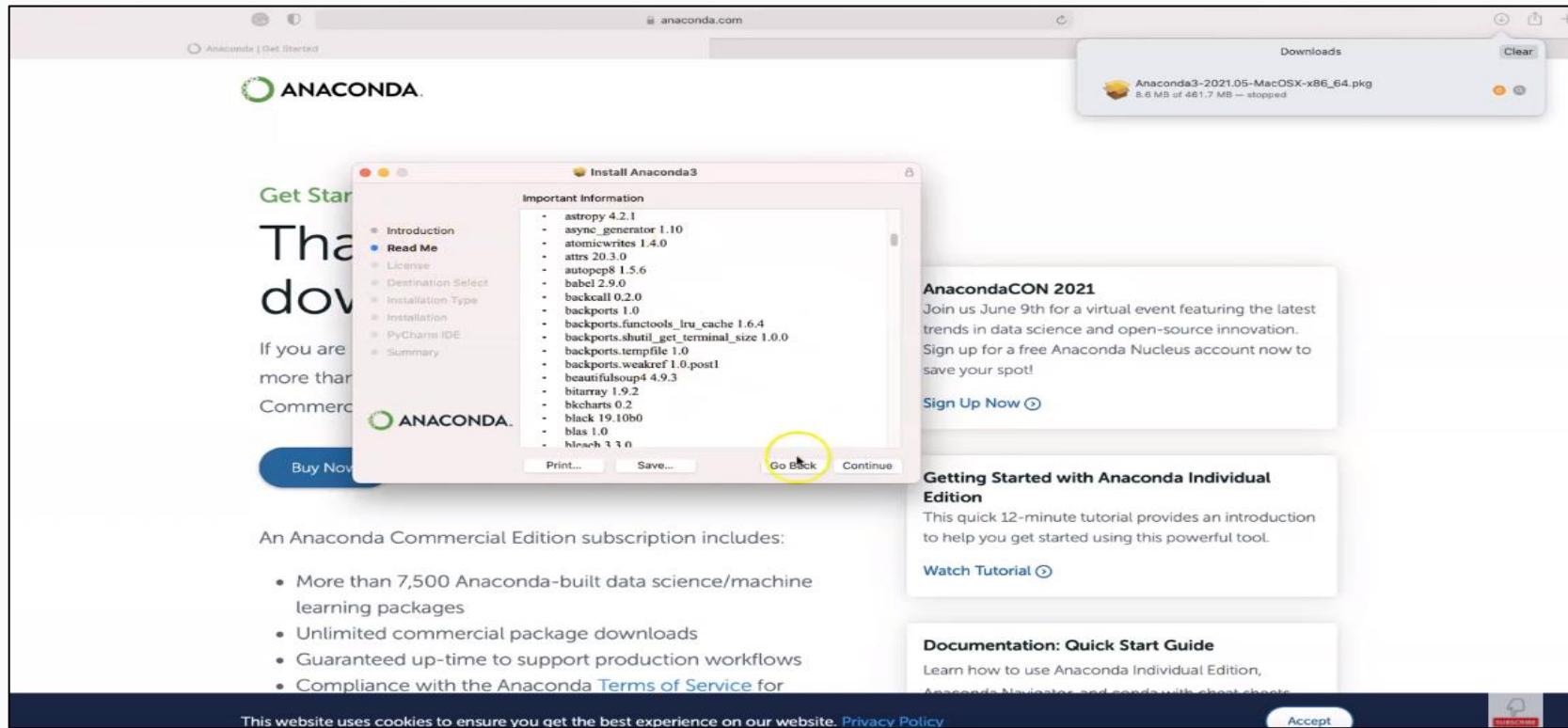
Anaconda for Mac (4/14)

- Once download, launch the installation wizard by double clicking on the file.
- Click Continue.



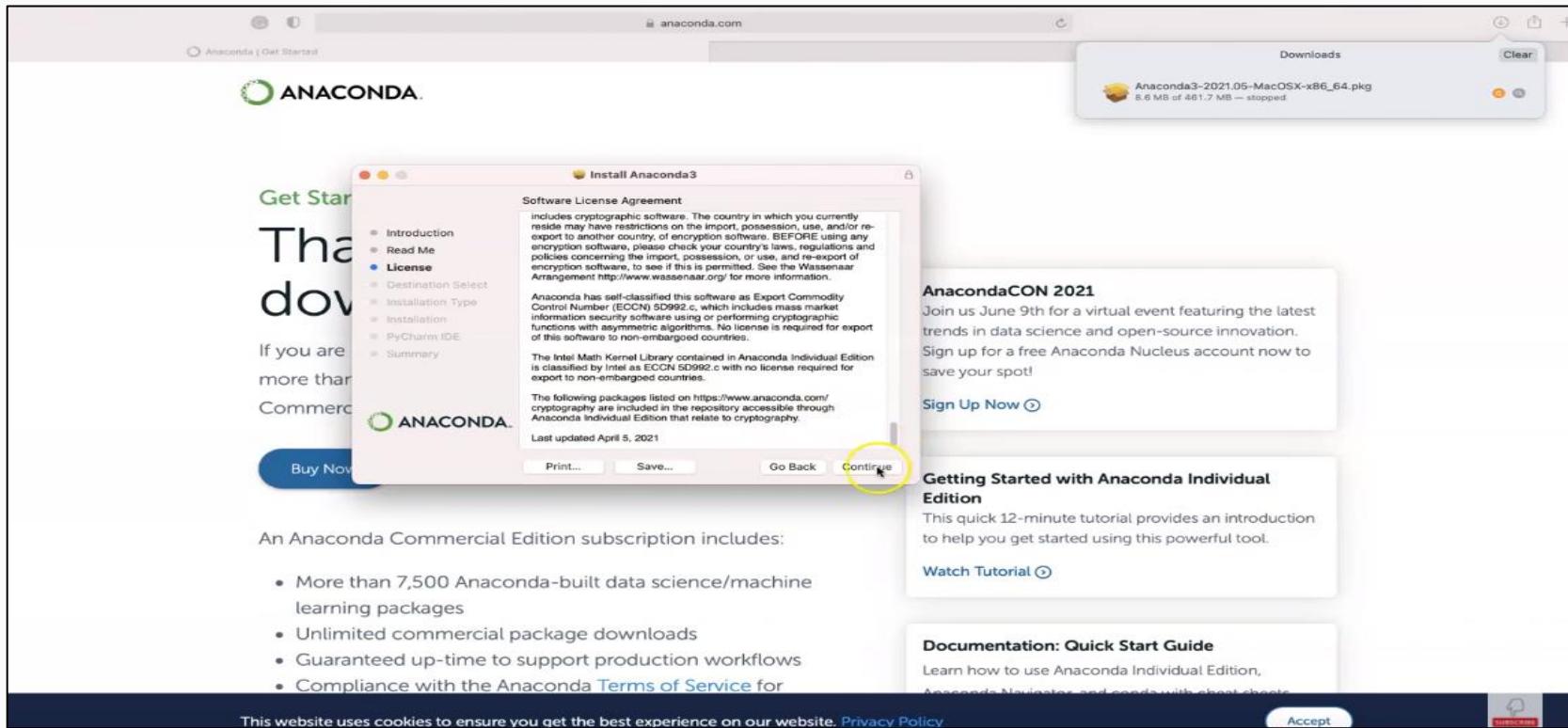
Anaconda for Mac (5/14)

- Read the Read Me file and click continue.



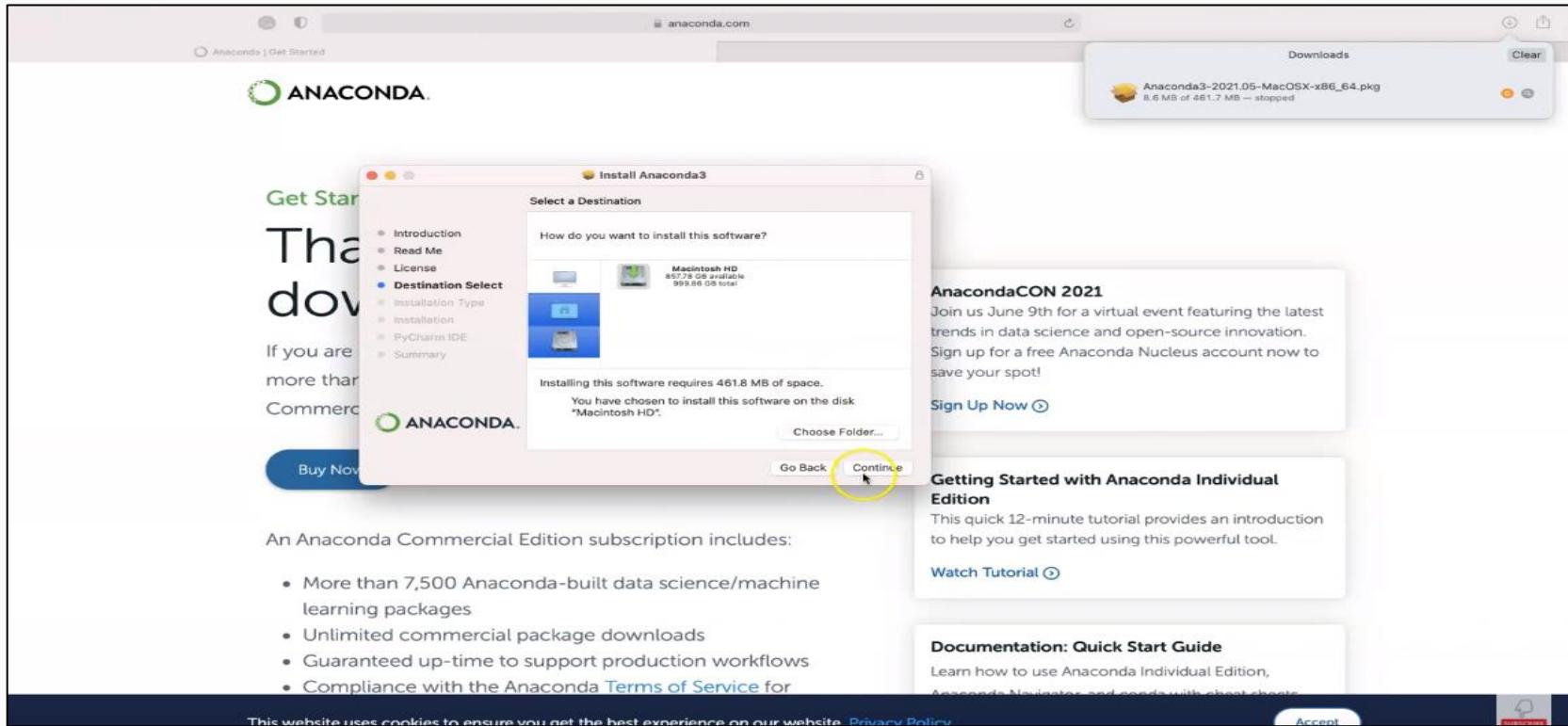
Anaconda for Mac (6/14)

- Read the License Agreement and click continue.



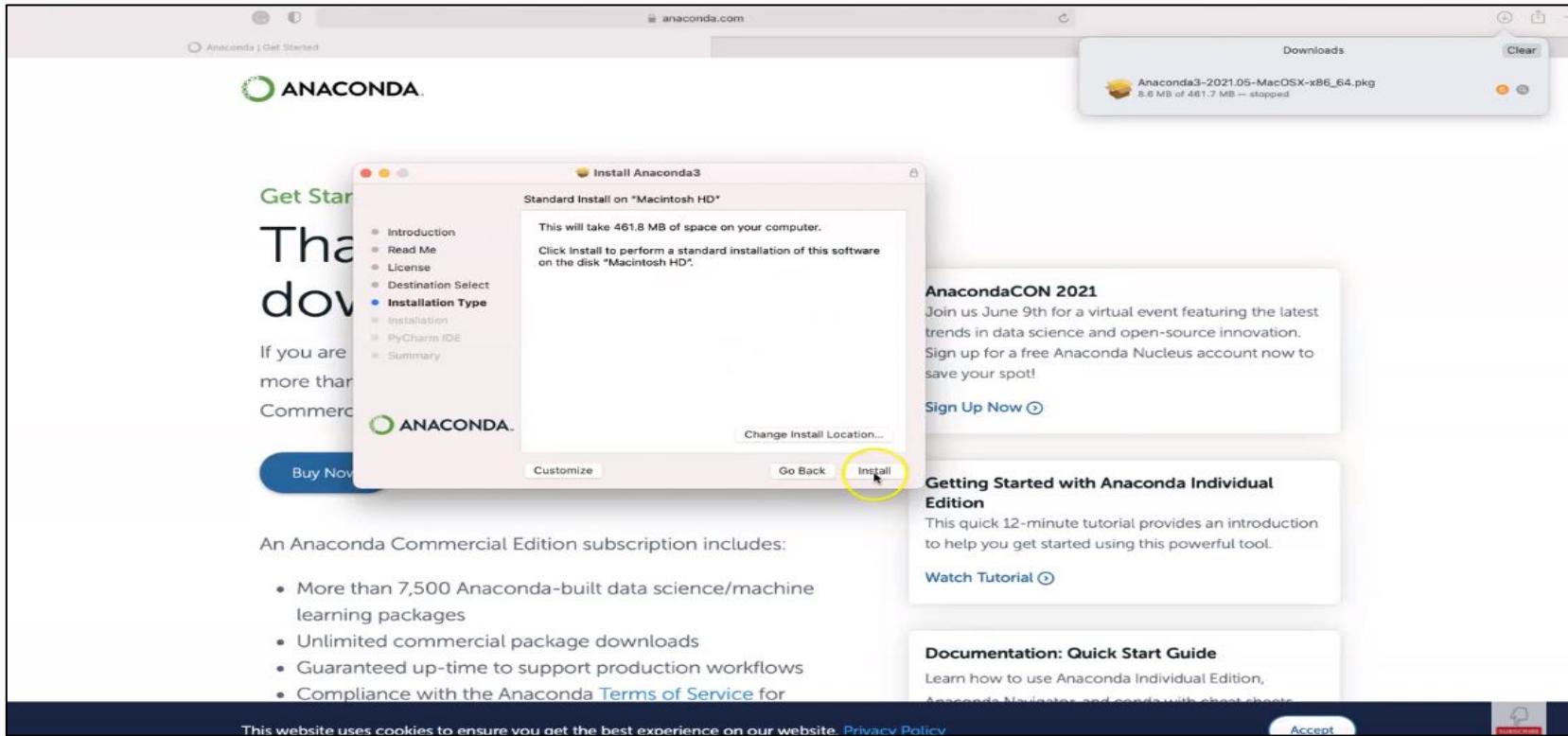
Anaconda for Mac (7/14)

- Select the destination and click Continue.



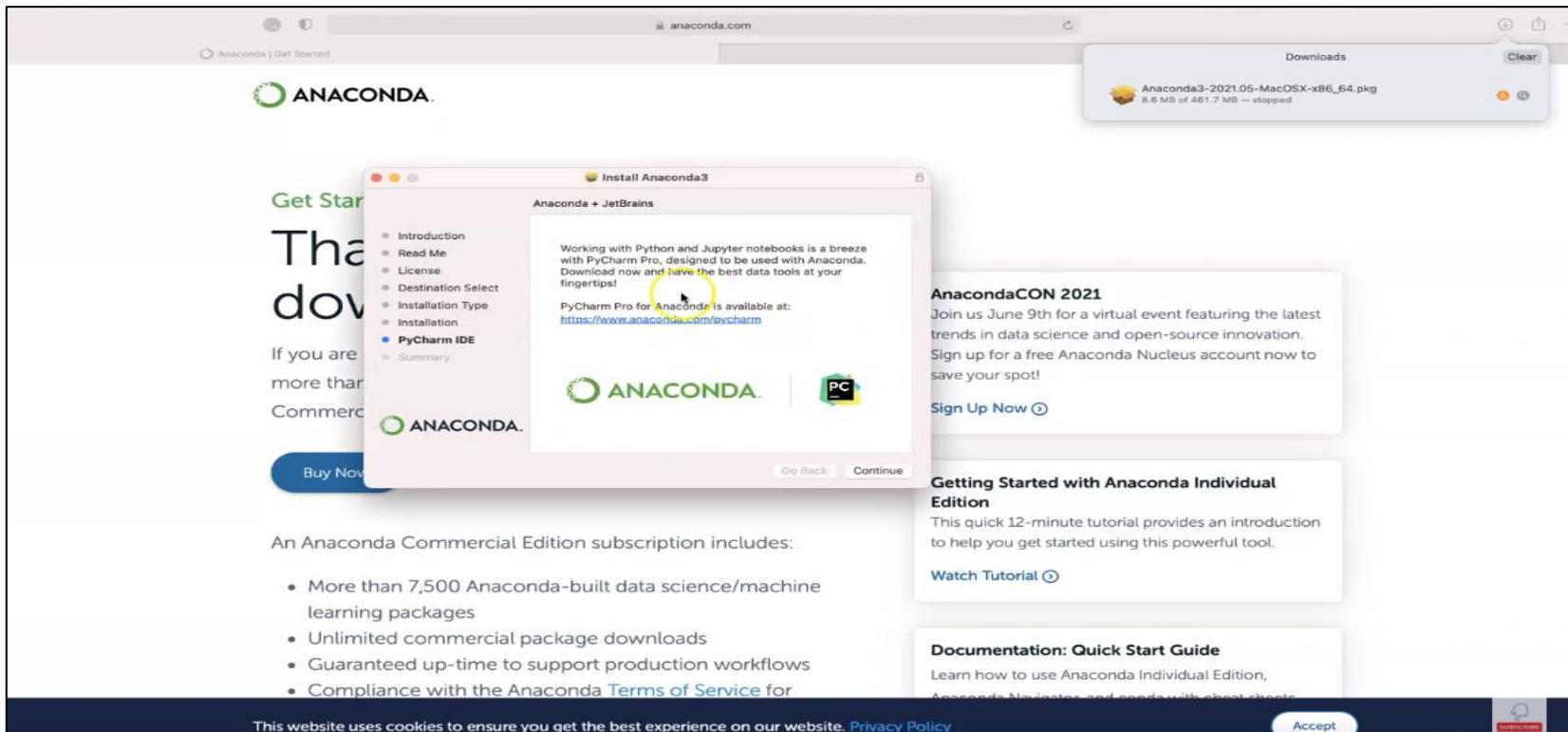
Anaconda for Mac (8/14)

- Click Install.



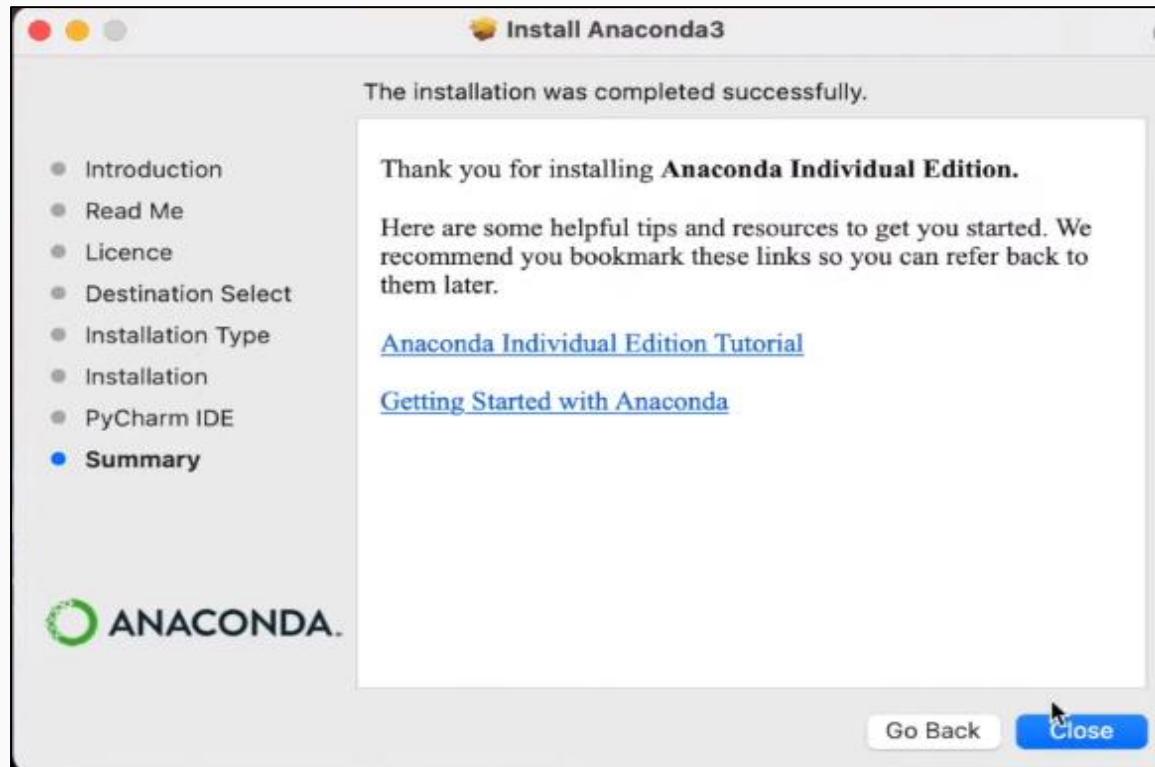
Anaconda for Mac (9/14)

- Click Continue.



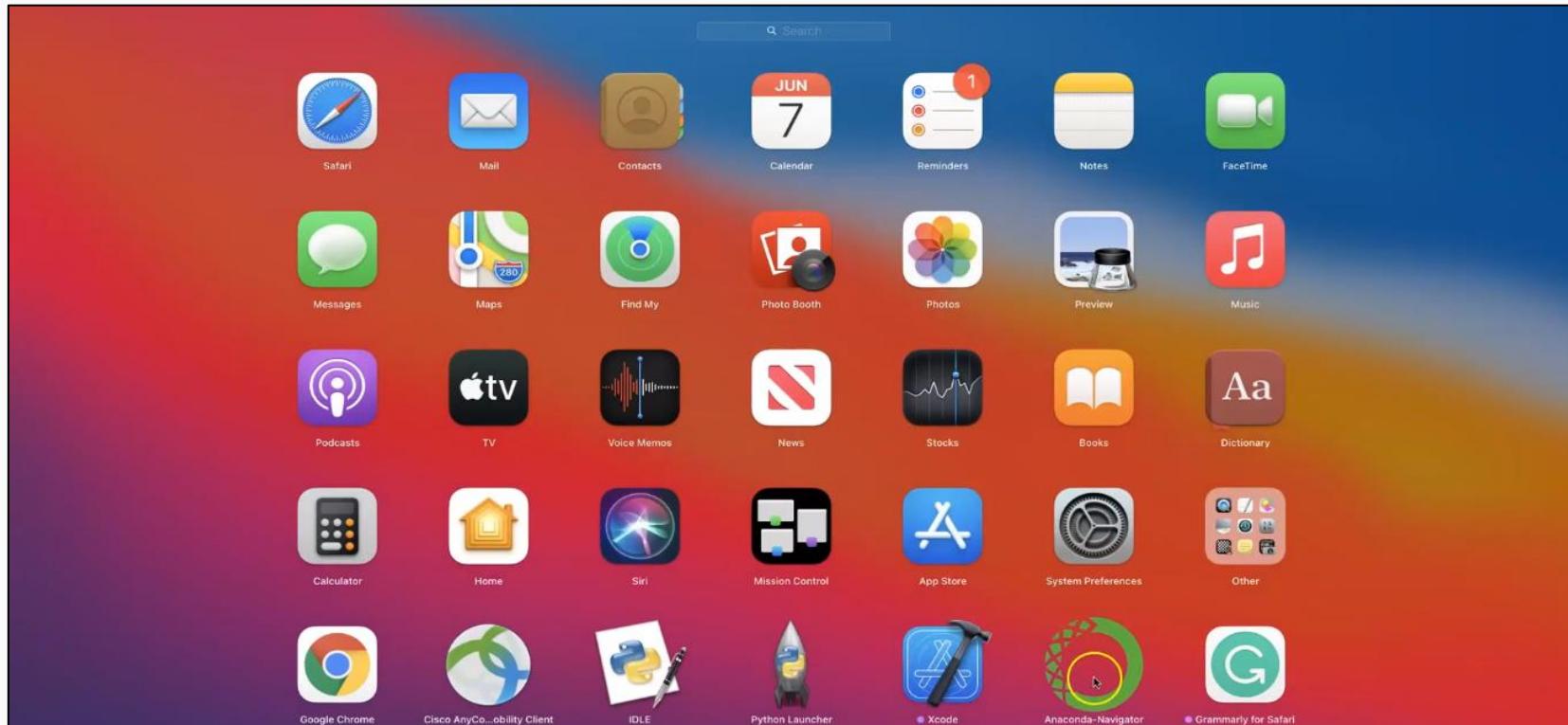
Anaconda for Mac (10/14)

- Once the installation finishes, click Close.



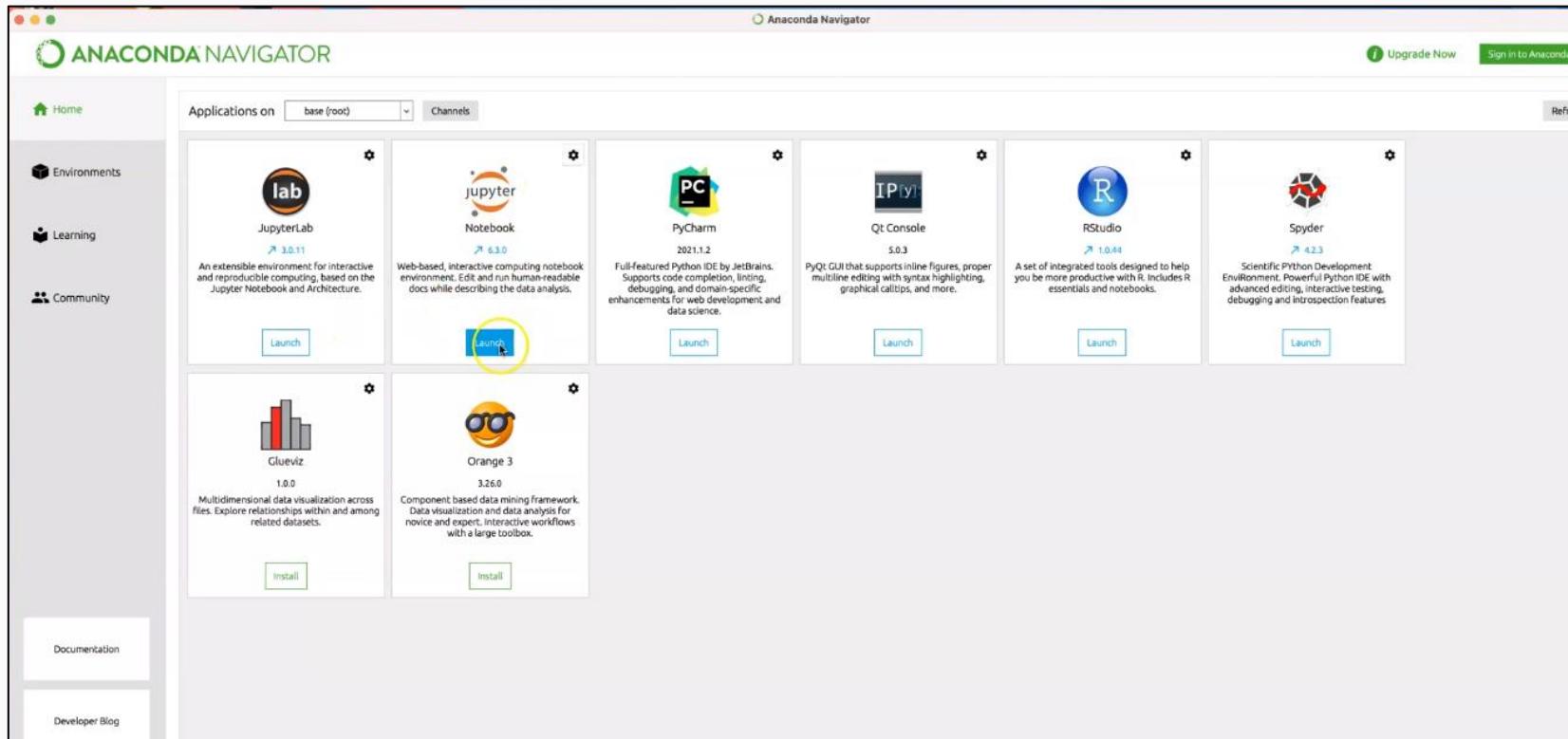
Anaconda for Mac (11/14)

- Open your spotlight and click on Anaconda Navigator to launch it.



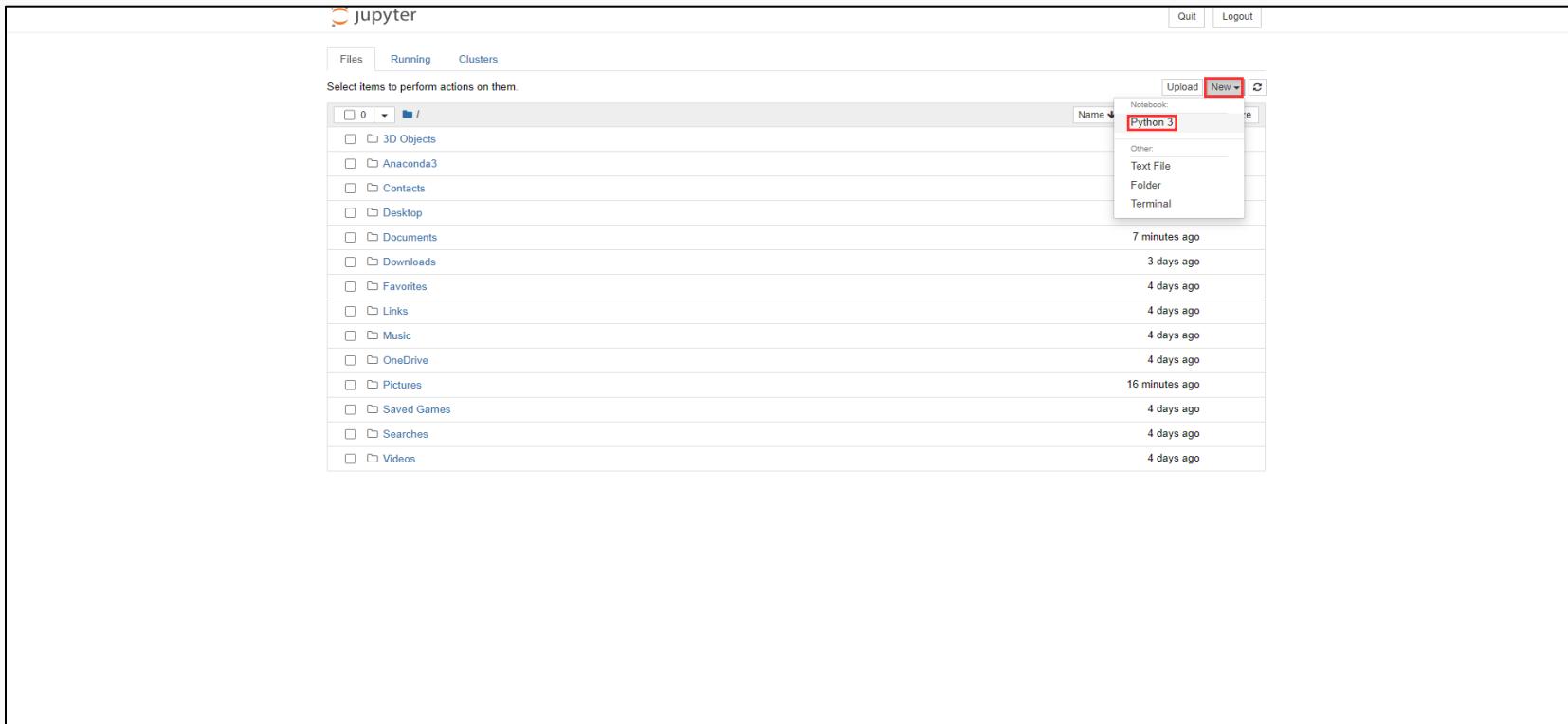
Anaconda for Mac (12/14)

- Launch Jupyter Notebook. It will take sometime to launch.



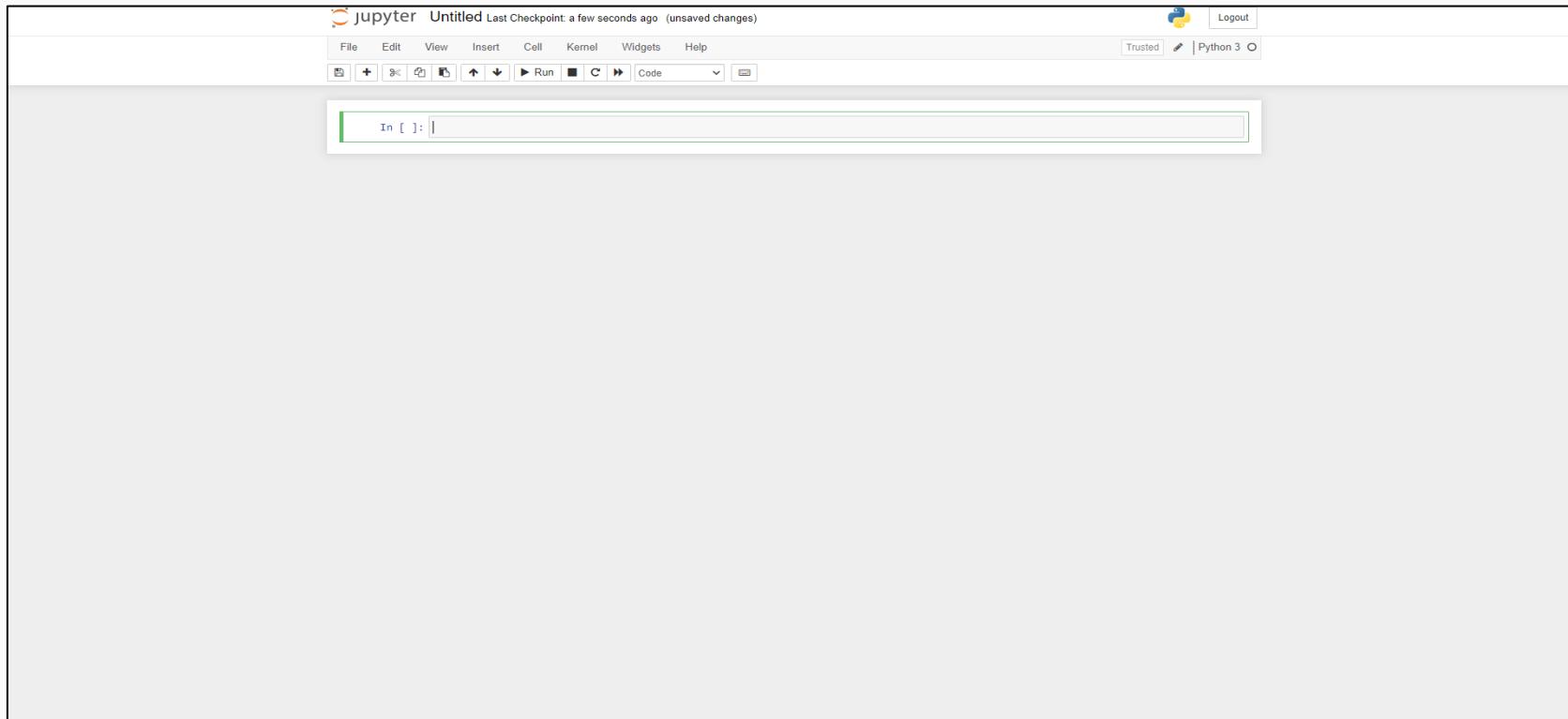
Anaconda for Mac (13/14)

- This will launch Jupyter Notebook in your default browser.
 - Click on New -> Python3 to create a new Notebook.



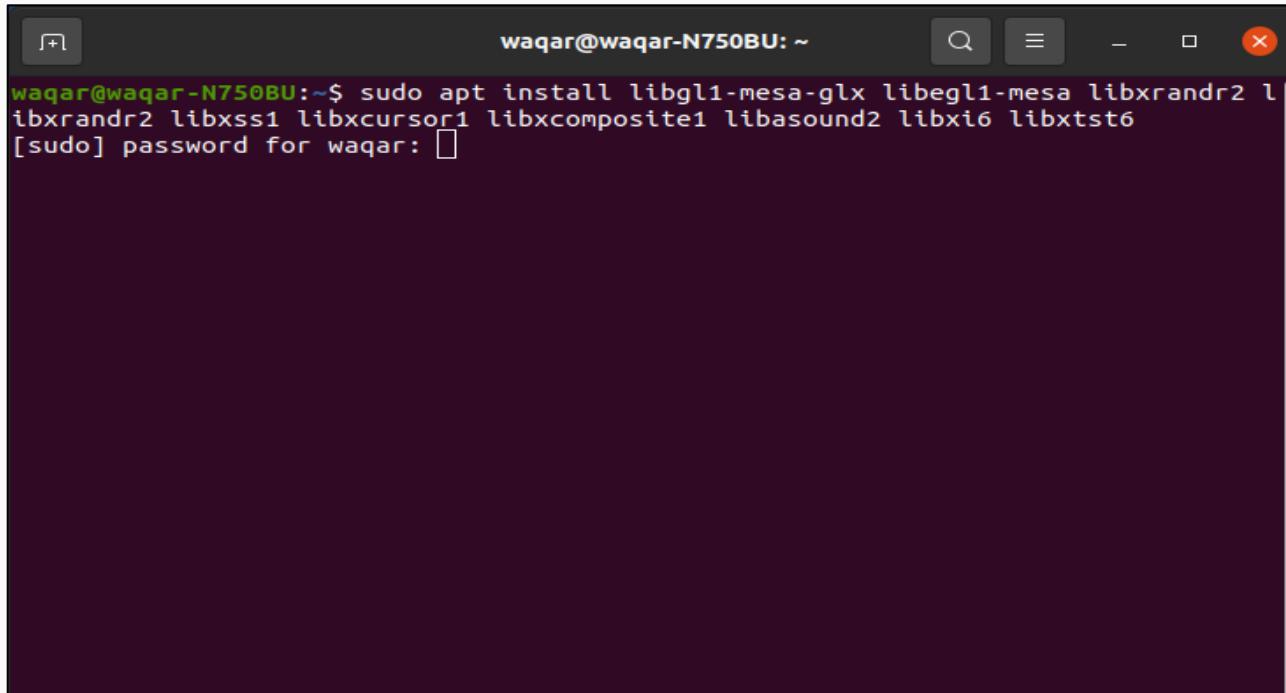
Anaconda for Mac (14/14)

- This is how a Notebook looks like.



Anaconda for Ubuntu (1/11)

- Open your terminal, paste the following command, and hit enter
sudo apt install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1 libcomposite1 libasound2 libxi6 libxtst6
- Type your password and hit Enter.



A screenshot of a dark-themed terminal window on an Ubuntu desktop. The title bar shows the session name "waqar@waqar-N750BU: ~". The main area of the terminal displays the following command and its execution:

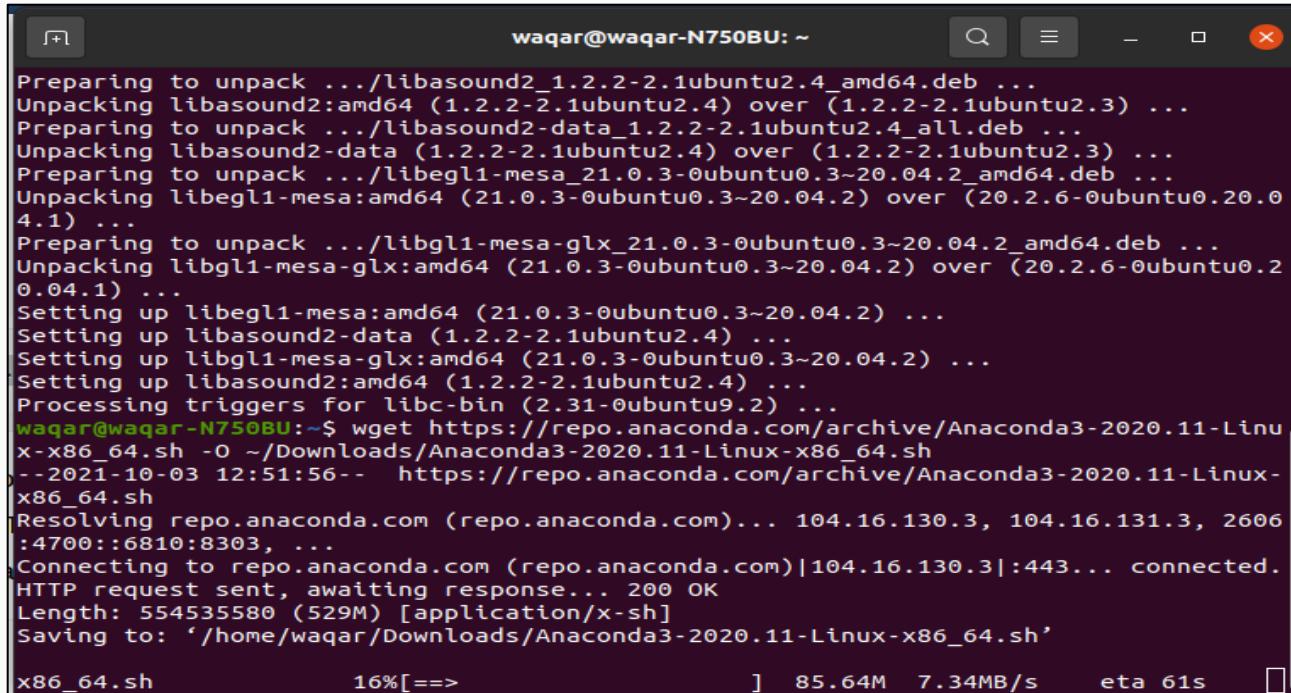
```
waqar@waqar-N750BU:~$ sudo apt install libgl1-mesa-glx libegl1-mesa libxrandr2 l
ibxrandr2 libxss1 libxcursor1 libcomposite1 libasound2 libxi6 libxtst6
[sudo] password for waqar: 
```

The terminal window has standard window controls (minimize, maximize, close) at the top right. The background of the window is dark, matching the desktop environment.

Anaconda for Ubuntu (2/11)

- Next give the following command and hit enter

```
wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-x86_64.sh -O  
~/Downloads/Anaconda3-2020.11-Linux-x86_64.sh
```



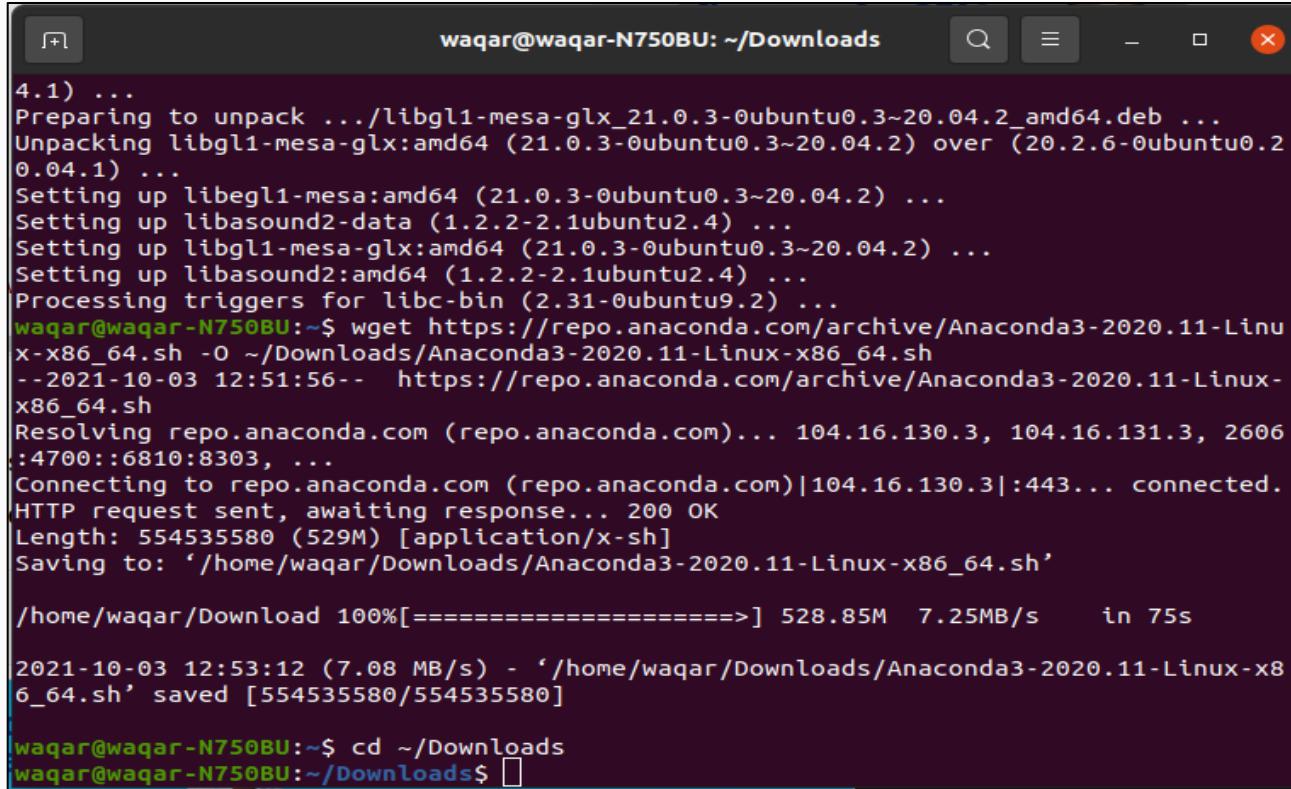
The screenshot shows a terminal window titled "waqar@waqar-N750BU: ~". The window displays the process of downloading and unpacking the Anaconda package. The output shows the extraction of multiple .deb files, followed by the execution of the Anaconda setup script. The terminal also shows the progress of the wget command at the bottom.

```
Preparing to unpack .../libasound2_1.2.2-2.1ubuntu2.4_amd64.deb ...
Unpacking libasound2:amd64 (1.2.2-2.1ubuntu2.4) over (1.2.2-2.1ubuntu2.3) ...
Preparing to unpack .../libasound2-data_1.2.2-2.1ubuntu2.4_all.deb ...
Unpacking libasound2-data (1.2.2-2.1ubuntu2.4) over (1.2.2-2.1ubuntu2.3) ...
Preparing to unpack .../libegl1-mesa_21.0.3-0ubuntu0.3~20.04.2_amd64.deb ...
Unpacking libegl1-mesa:amd64 (21.0.3-0ubuntu0.3~20.04.2) over (20.2.6-0ubuntu0.20.0
4.1) ...
Preparing to unpack .../libegl1-mesa-gl_21.0.3-0ubuntu0.3~20.04.2_amd64.deb ...
Unpacking libegl1-mesa-gl:amd64 (21.0.3-0ubuntu0.3~20.04.2) over (20.2.6-0ubuntu0.2
0.04.1) ...
Setting up libegl1-mesa:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2-data (1.2.2-2.1ubuntu2.4) ...
Setting up libegl1-mesa-gl:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2:amd64 (1.2.2-2.1ubuntu2.4) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
waqar@waqar-N750BU:~$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linu
x-x86_64.sh -O ~/Downloads/Anaconda3-2020.11-Linux-x86_64.sh
--2021-10-03 12:51:56-- https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-
x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.130.3, 104.16.131.3, 2606
:4700:::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.130.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 554535580 (529M) [application/x-sh]
Saving to: '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x86_64.sh'

x86_64.sh          16%[==>]           ]  85.64M   7.34MB/s    eta 61s
```

Anaconda for Ubuntu (3/11)

- Next give the following command and hit enter
cd ~/Downloads



The screenshot shows a terminal window titled "waqar@waqar-N750BU: ~/Downloads". The window contains the following text:

```
4.1) ...
Preparing to unpack .../libgl1-mesa-glx_21.0.3-0ubuntu0.3~20.04.2_amd64.deb ...
Unpacking libgl1-mesa-glx:amd64 (21.0.3-0ubuntu0.3~20.04.2) over (20.2.6-0ubuntu0.2
0.04.1) ...
Setting up libegl1-mesa:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2-data (1.2.2-2.1ubuntu2.4) ...
Setting up libgl1-mesa-glx:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2:amd64 (1.2.2-2.1ubuntu2.4) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
waqar@waqar-N750BU:~$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linu
x-x86_64.sh -O ~/Downloads/Anaconda3-2020.11-Linux-x86_64.sh
--2021-10-03 12:51:56-- https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-
x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.130.3, 104.16.131.3, 2606
:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.130.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 554535580 (529M) [application/x-sh]
Saving to: '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x86_64.sh'

/home/waqar/Download 100%[=====] 528.85M 7.25MB/s in 75s

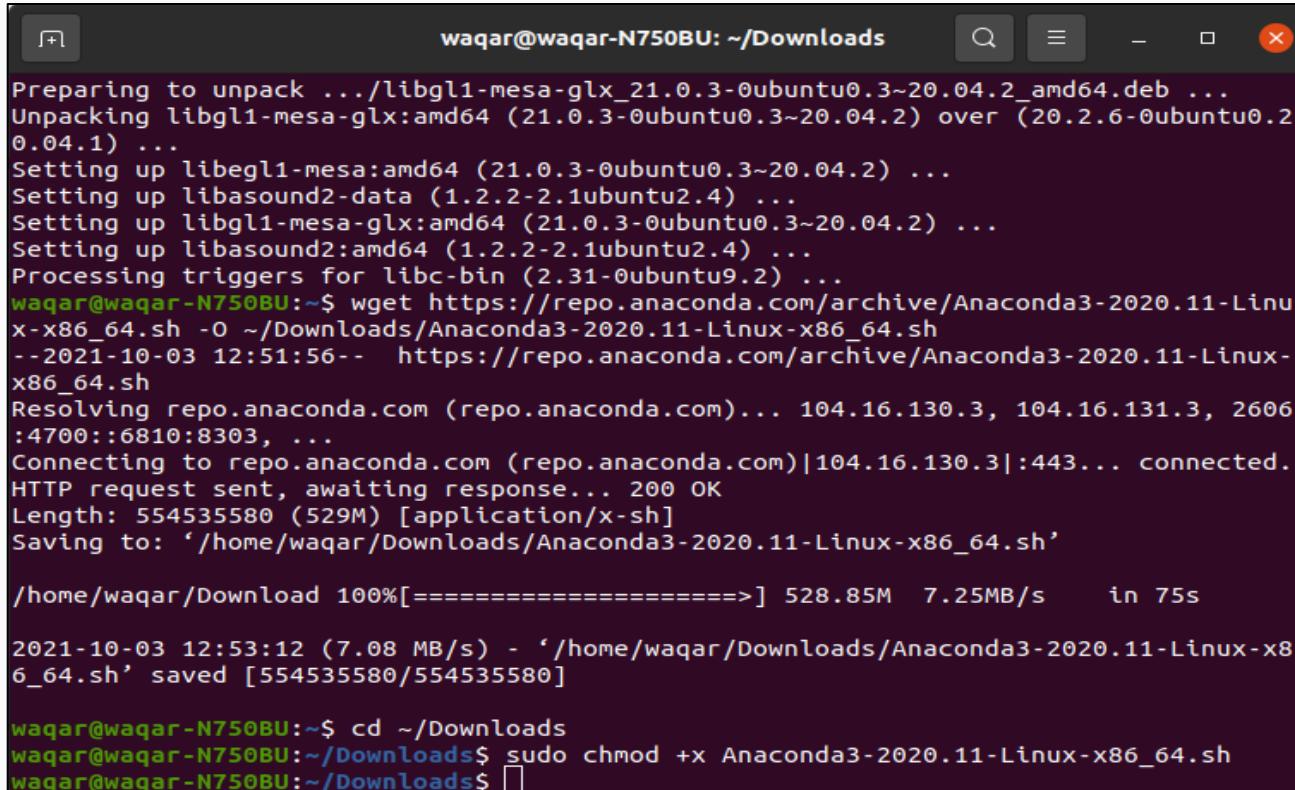
2021-10-03 12:53:12 (7.08 MB/s) - '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x8
6_64.sh' saved [554535580/554535580]

waqar@waqar-N750BU:~$ cd ~/Downloads
waqar@waqar-N750BU:~/Downloads$ 
```

Anaconda for Ubuntu (4/11)

- Next give the following command and hit enter

```
sudo chmod +x Anaconda3-2020.11-Linux-x86_64.sh
```



```
waqar@waqar-N750BU: ~/Downloads
Preparing to unpack .../libgl1-mesa-glx_21.0.3-0ubuntu0.3~20.04.2_amd64.deb ...
Unpacking libgl1-mesa-glx:amd64 (21.0.3-0ubuntu0.3~20.04.2) over (20.2.6-0ubuntu0.2
0.04.1) ...
Setting up libegl1-mesa:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2-data (1.2.2-2.1ubuntu2.4) ...
Setting up libgl1-mesa-glx:amd64 (21.0.3-0ubuntu0.3~20.04.2) ...
Setting up libasound2:amd64 (1.2.2-2.1ubuntu2.4) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
waqar@waqar-N750BU:~$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linu
x-x86_64.sh -O ~/Downloads/Anaconda3-2020.11-Linux-x86_64.sh
--2021-10-03 12:51:56-- https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-
x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.130.3, 104.16.131.3, 2606
:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.130.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 554535580 (529M) [application/x-sh]
Saving to: '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x86_64.sh'

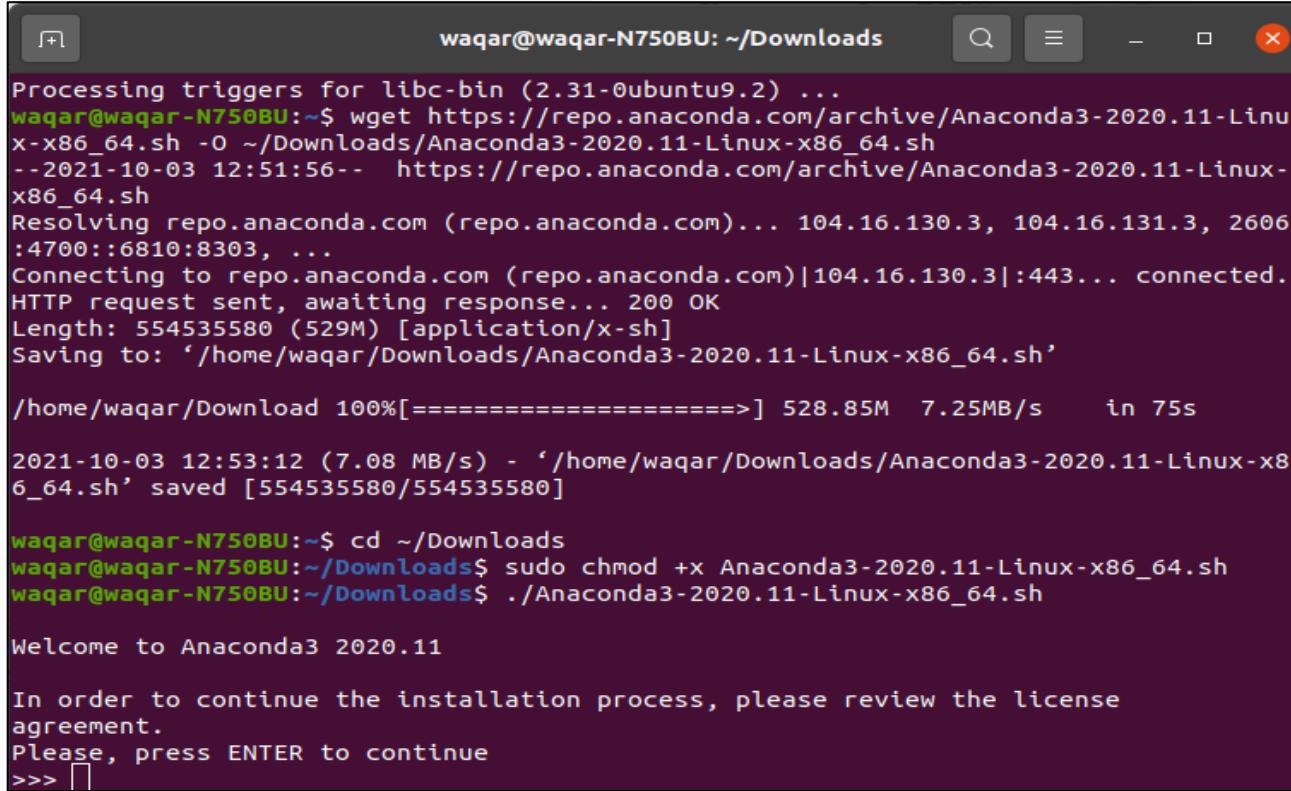
/home/waqar/Download 100%[=====] 528.85M 7.25MB/s in 75s

2021-10-03 12:53:12 (7.08 MB/s) - '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x8
6_64.sh' saved [554535580/554535580]

waqar@waqar-N750BU:~/Downloads$ cd ~/Downloads
waqar@waqar-N750BU:~/Downloads$ sudo chmod +x Anaconda3-2020.11-Linux-x86_64.sh
waqar@waqar-N750BU:~/Downloads$
```

Anaconda for Ubuntu (5/11)

- Next give the following command and hit enter
./Anaconda3-2020.11-Linux-x86_64.sh
- When prompted, press enter to continue.



The screenshot shows a terminal window titled "waqar@waqar-N750BU: ~/Downloads". The window contains the following text:

```
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
waqar@waqar-N750BU:~$ wget https://repo.anaconda.com/archive/Anaconda3-2020.11-Linu
x-x86_64.sh -O ~/Downloads/Anaconda3-2020.11-Linux-x86_64.sh
--2021-10-03 12:51:56-- https://repo.anaconda.com/archive/Anaconda3-2020.11-Linux-
x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.130.3, 104.16.131.3, 2606
:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.130.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 554535580 (529M) [application/x-sh]
Saving to: '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x86_64.sh'

/home/waqar/Download 100%[=====] 528.85M 7.25MB/s in 75s

2021-10-03 12:53:12 (7.08 MB/s) - '/home/waqar/Downloads/Anaconda3-2020.11-Linux-x8
6_64.sh' saved [554535580/554535580]

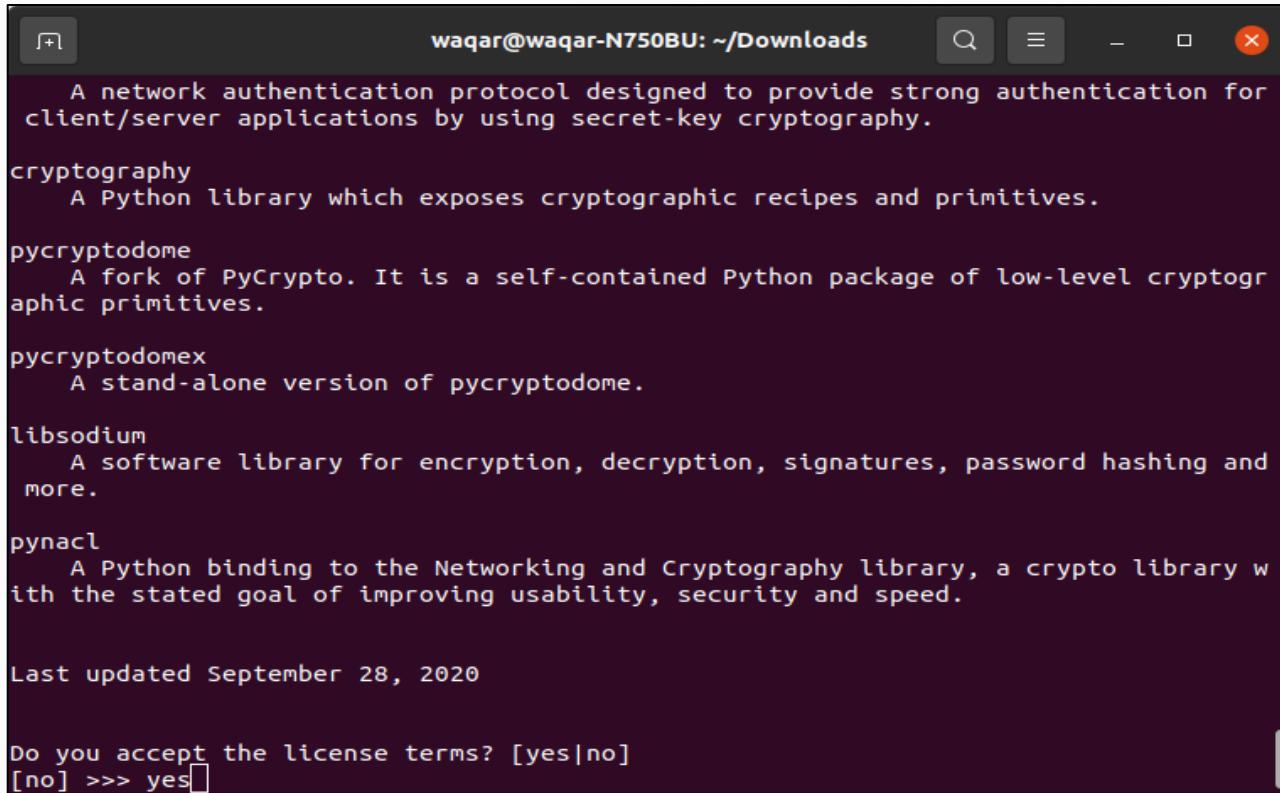
waqar@waqar-N750BU:~$ cd ~/Downloads
waqar@waqar-N750BU:~/Downloads$ sudo chmod +x Anaconda3-2020.11-Linux-x86_64.sh
waqar@waqar-N750BU:~/Downloads$ ./Anaconda3-2020.11-Linux-x86_64.sh

Welcome to Anaconda3 2020.11

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> |
```

Anaconda for Ubuntu (6/11)

- Press the Enter key to scroll through the License Agreement.
- After reading the Agreement, type yes in the prompt and hit enter.



A screenshot of a terminal window titled "waqar@waqar-N750BU: ~/Downloads". The window displays the Anaconda license agreement for Ubuntu, listing several cryptographic libraries: "cryptography", "pycryptodome", "pycryptodomex", "libsodium", and "pynacl". Each library has a brief description. At the bottom of the list, it says "Last updated September 28, 2020". A prompt at the bottom asks "Do you accept the license terms? [yes|no]" with "[no] >> yes" followed by a cursor. The terminal has a dark background with light-colored text and standard Linux-style window controls.

```
A network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography.

cryptography
    A Python library which exposes cryptographic recipes and primitives.

pycryptodome
    A fork of PyCrypto. It is a self-contained Python package of low-level cryptographic primitives.

pycryptodomex
    A stand-alone version of pycryptodome.

libsodium
    A software library for encryption, decryption, signatures, password hashing and more.

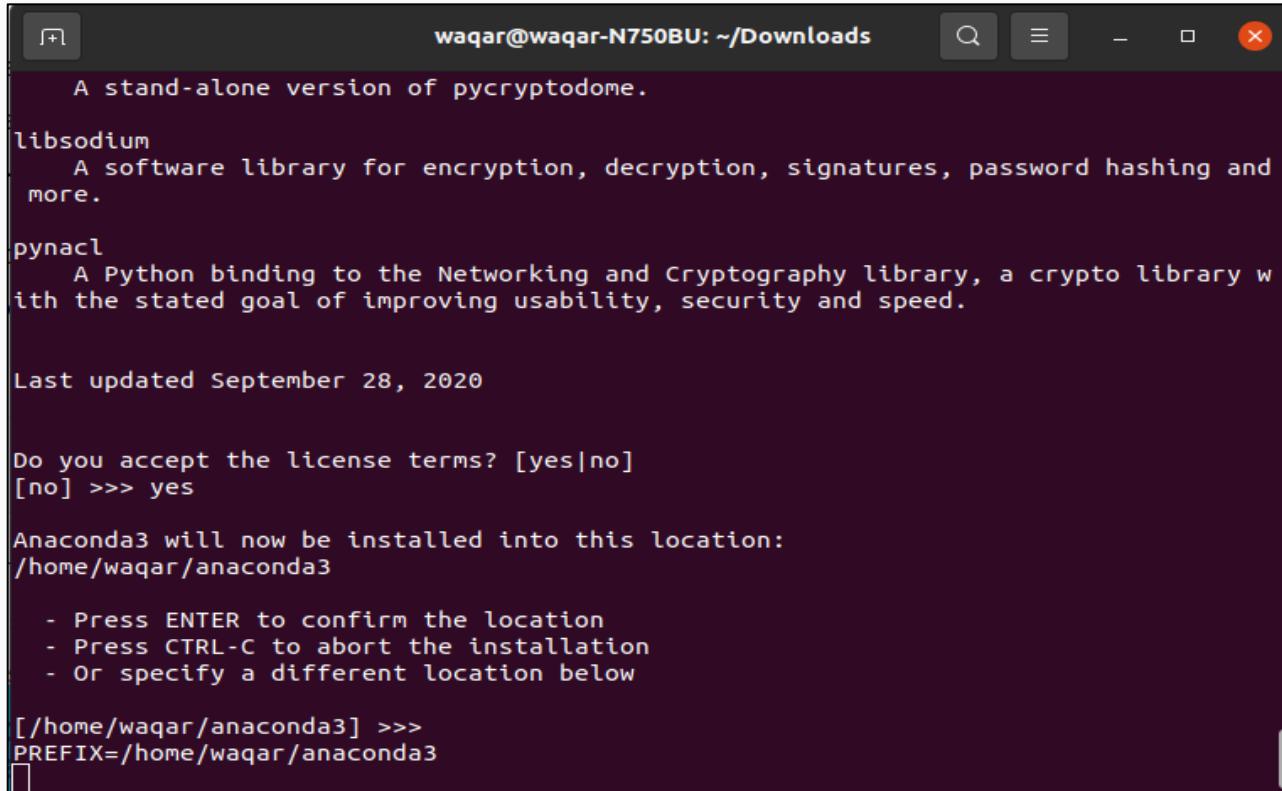
pynacl
    A Python binding to the Networking and Cryptography library, a crypto library with the stated goal of improving usability, security and speed.

Last updated September 28, 2020

Do you accept the license terms? [yes|no]
[no] >> yes
```

Anaconda for Ubuntu (7/11)

- Anaconda installer will ask you where do you want to install Anaconda. We suggest pressing Enter key to install it in the home directory.
- After sometime, Anaconda will be installed on your machine.



A stand-alone version of pycryptodome.
libsodium
A software library for encryption, decryption, signatures, password hashing and more.
pynacl
A Python binding to the Networking and Cryptography library, a crypto library with the stated goal of improving usability, security and speed.

Last updated September 28, 2020

Do you accept the license terms? [yes|no]
[no] >>> yes

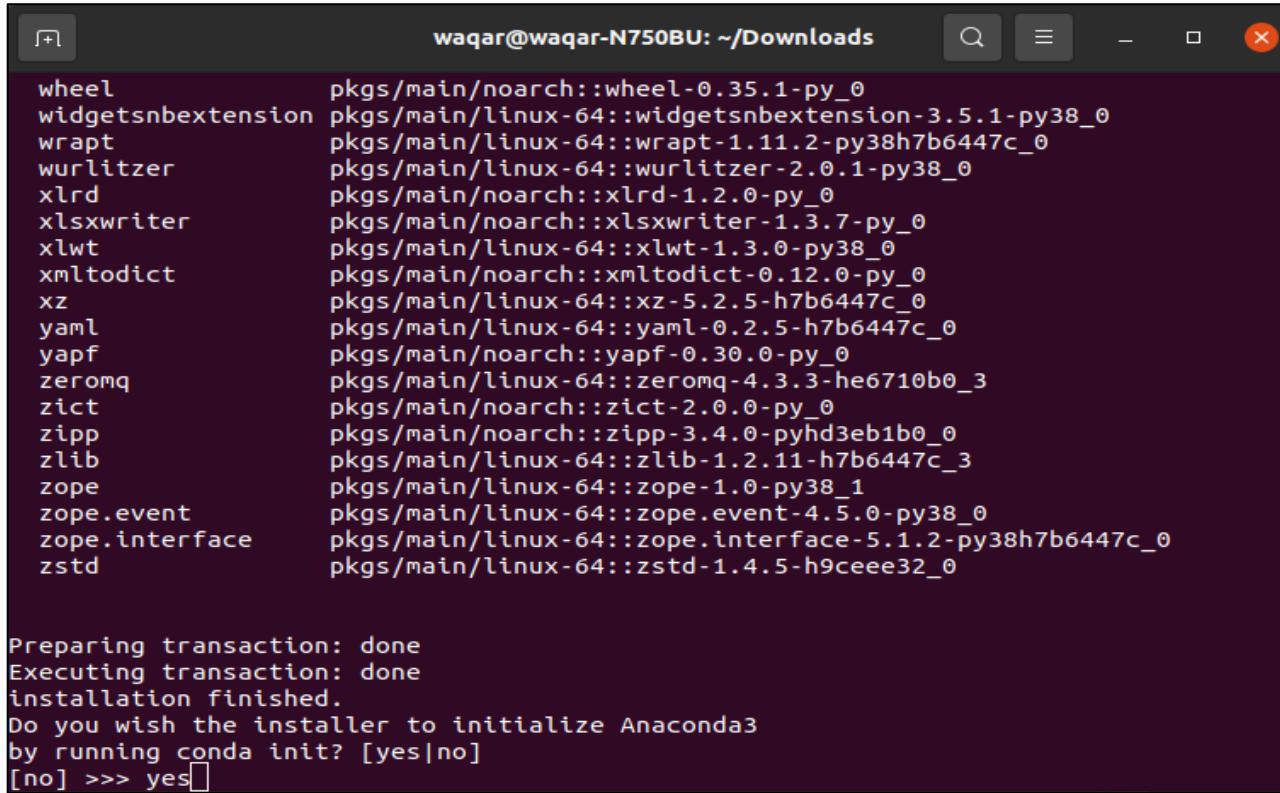
Anaconda3 will now be installed into this location:
/home/waqar/anaconda3

- Press ENTER to confirm the location
- Press CTRL-C to abort the installation
- Or specify a different location below

[/home/waqar/anaconda3] >>>
PREFIX=/home/waqar/anaconda3

Anaconda for Ubuntu (8/11)

- Once installation is finished, you will be asked “Do you wish the installer to initialize Anaconda3 by running conda init?” type “yes” and hit Enter. Setup will finish the installation process.



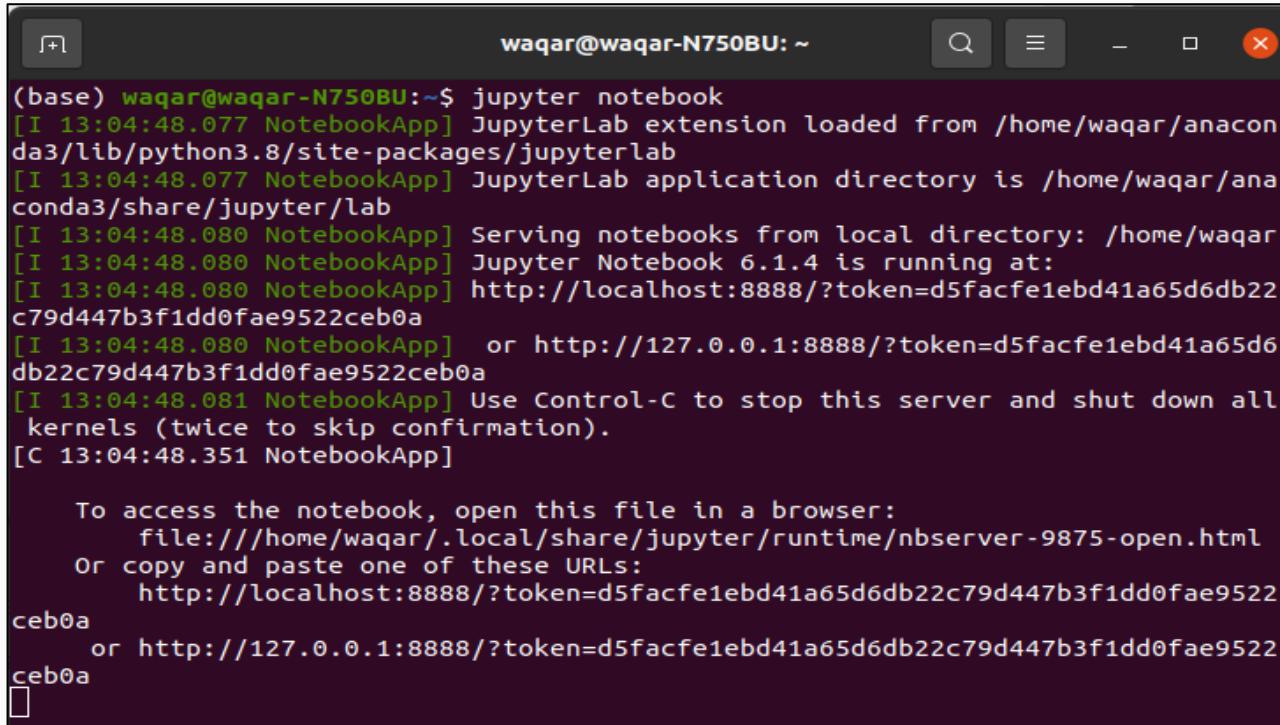
The screenshot shows a terminal window titled "waqar@waqar-N750BU: ~/Downloads". The window displays a list of packages installed, followed by transaction preparation and execution messages, and finally a prompt asking if the user wishes to initialize Anaconda3.

```
wheel                  pkgs/main/noarch::wheel-0.35.1-py_0
widgetsnbextension    pkgs/main/linux-64::widgetsnbextension-3.5.1-py38_0
wrapt                 pkgs/main/linux-64::wrapt-1.11.2-py38h7b6447c_0
wurlitzer              pkgs/main/linux-64::wurlitzer-2.0.1-py38_0
xlrd                  pkgs/main/noarch::xlrd-1.2.0-py_0
xlsxwriter             pkgs/main/noarch::xlsxwriter-1.3.7-py_0
xlwt                  pkgs/main/linux-64::xlwt-1.3.0-py38_0
xmltodict              pkgs/main/noarch::xmltodict-0.12.0-py_0
xz                     pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
yaml                  pkgs/main/linux-64::yaml-0.2.5-h7b6447c_0
yapf                  pkgs/main/noarch::yapf-0.30.0-py_0
zeromq                pkgs/main/linux-64::zeromq-4.3.3-he6710b0_3
zict                  pkgs/main/noarch::zict-2.0.0-py_0
zipp                  pkgs/main/noarch::zipp-3.4.0-pyhd3eb1b0_0
zlib                  pkgs/main/linux-64::zlib-1.2.11-h7b6447c_3
zope                  pkgs/main/linux-64::zope-1.0-py38_1
zope.event              pkgs/main/linux-64::zope.event-4.5.0-py38_0
zope.interface          pkgs/main/linux-64::zope.interface-5.1.2-py38h7b6447c_0
zstd                  pkgs/main/linux-64::zstd-1.4.5-h9ceee32_0

Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
```

Anaconda for Ubuntu (9/11)

- To launch Jupyter Notebook, restart your terminal.
- You will see (base) written before your home directory name.
- Type “jupyter notebook” without the quotation marks and hit enter.
- After a few minutes, Jupyter Notebook will run on your default browser.

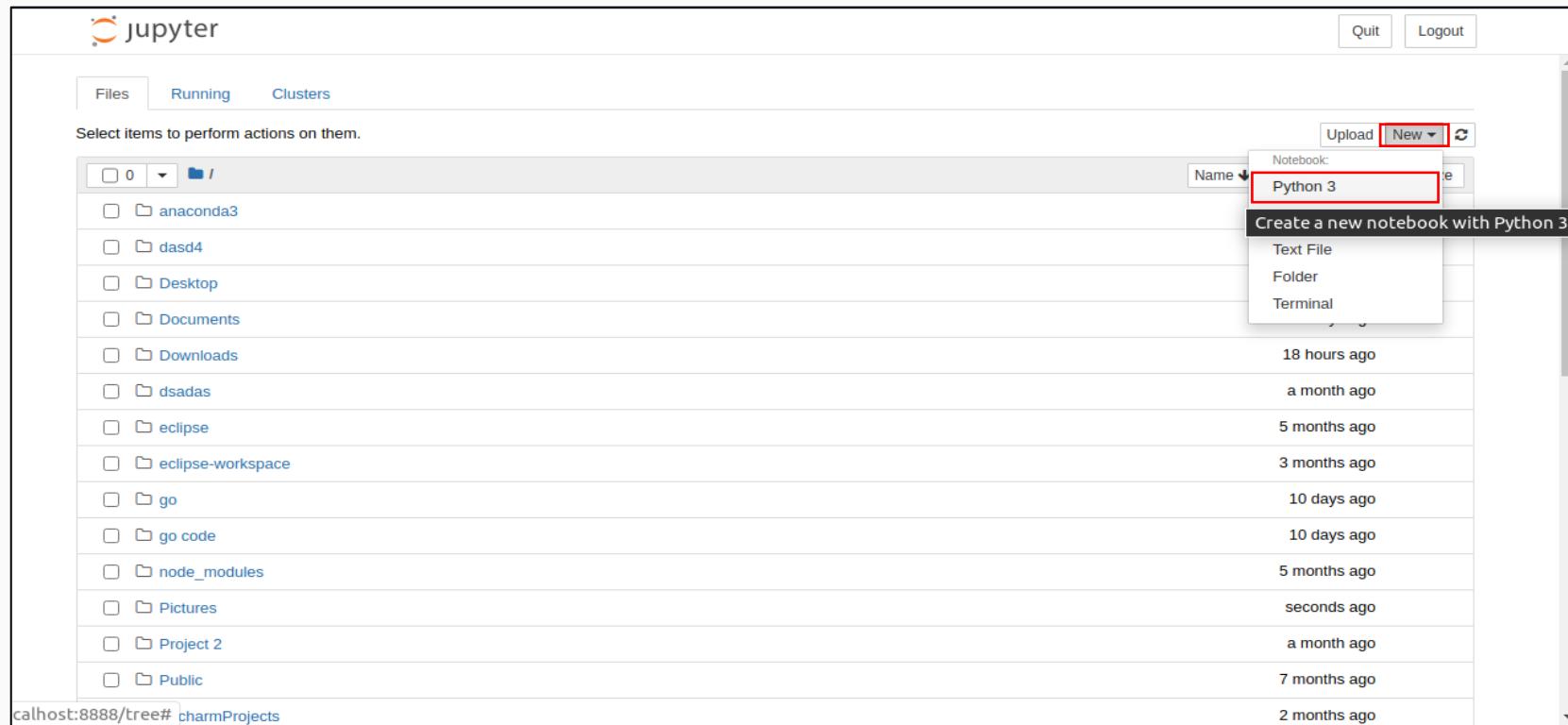


```
waqar@waqar-N750BU: ~
(base) waqar@waqar-N750BU:~$ jupyter notebook
[I 13:04:48.077 NotebookApp] JupyterLab extension loaded from /home/waqar/anaconda3/lib/python3.8/site-packages/jupyterlab
[I 13:04:48.077 NotebookApp] JupyterLab application directory is /home/waqar/anaconda3/share/jupyter/lab
[I 13:04:48.080 NotebookApp] Serving notebooks from local directory: /home/waqar
[I 13:04:48.080 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 13:04:48.080 NotebookApp] http://localhost:8888/?token=d5facfe1ebd41a65d6db22c79d447b3f1dd0fae9522ceb0a
[I 13:04:48.080 NotebookApp] or http://127.0.0.1:8888/?token=d5facfe1ebd41a65d6db22c79d447b3f1dd0fae9522ceb0a
[I 13:04:48.081 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:04:48.351 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/waqar/.local/share/jupyter/runtime/nbserver-9875-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=d5facfe1ebd41a65d6db22c79d447b3f1dd0fae9522ceb0a
or http://127.0.0.1:8888/?token=d5facfe1ebd41a65d6db22c79d447b3f1dd0fae9522ceb0a
```

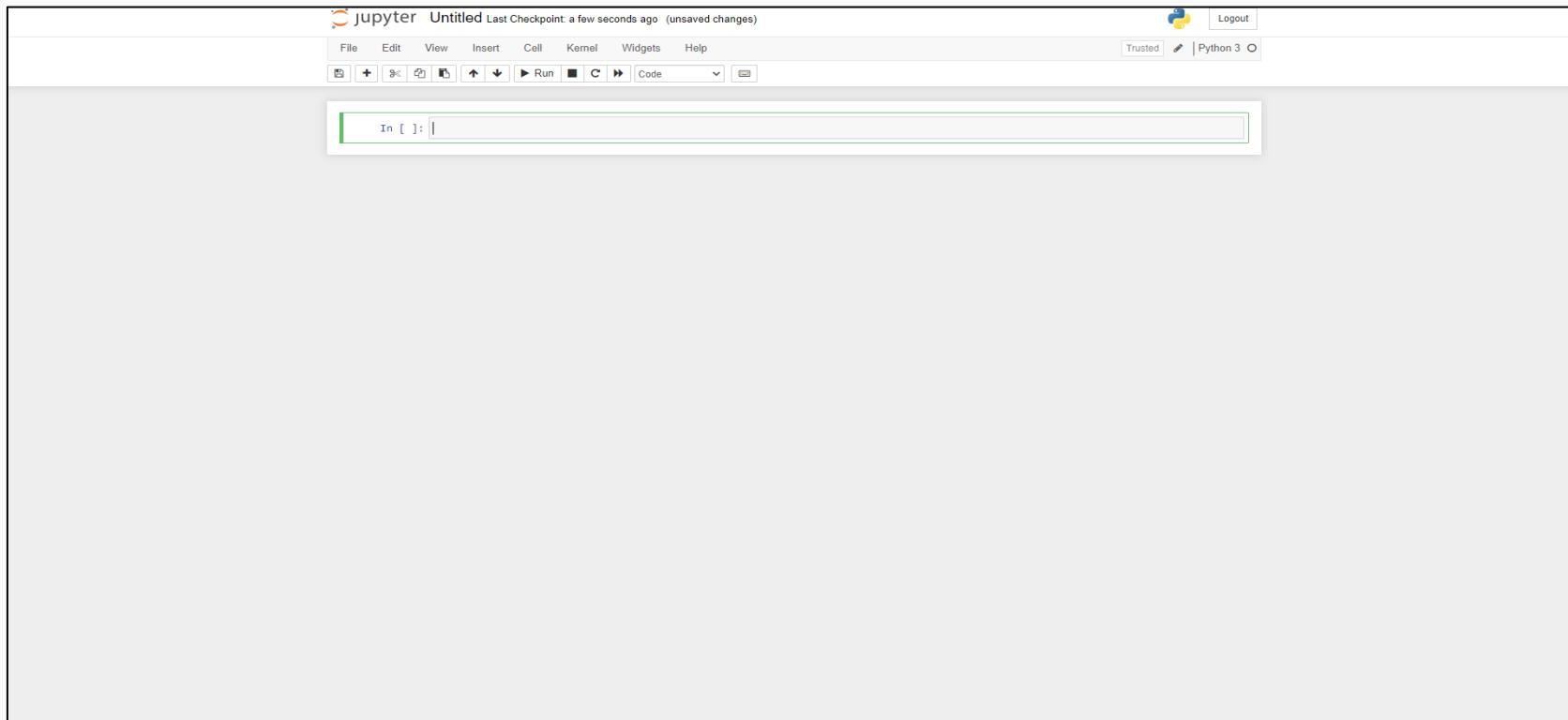
Anaconda for Mac (10/11)

- Click on New -> Python3 to create a new Notebook.



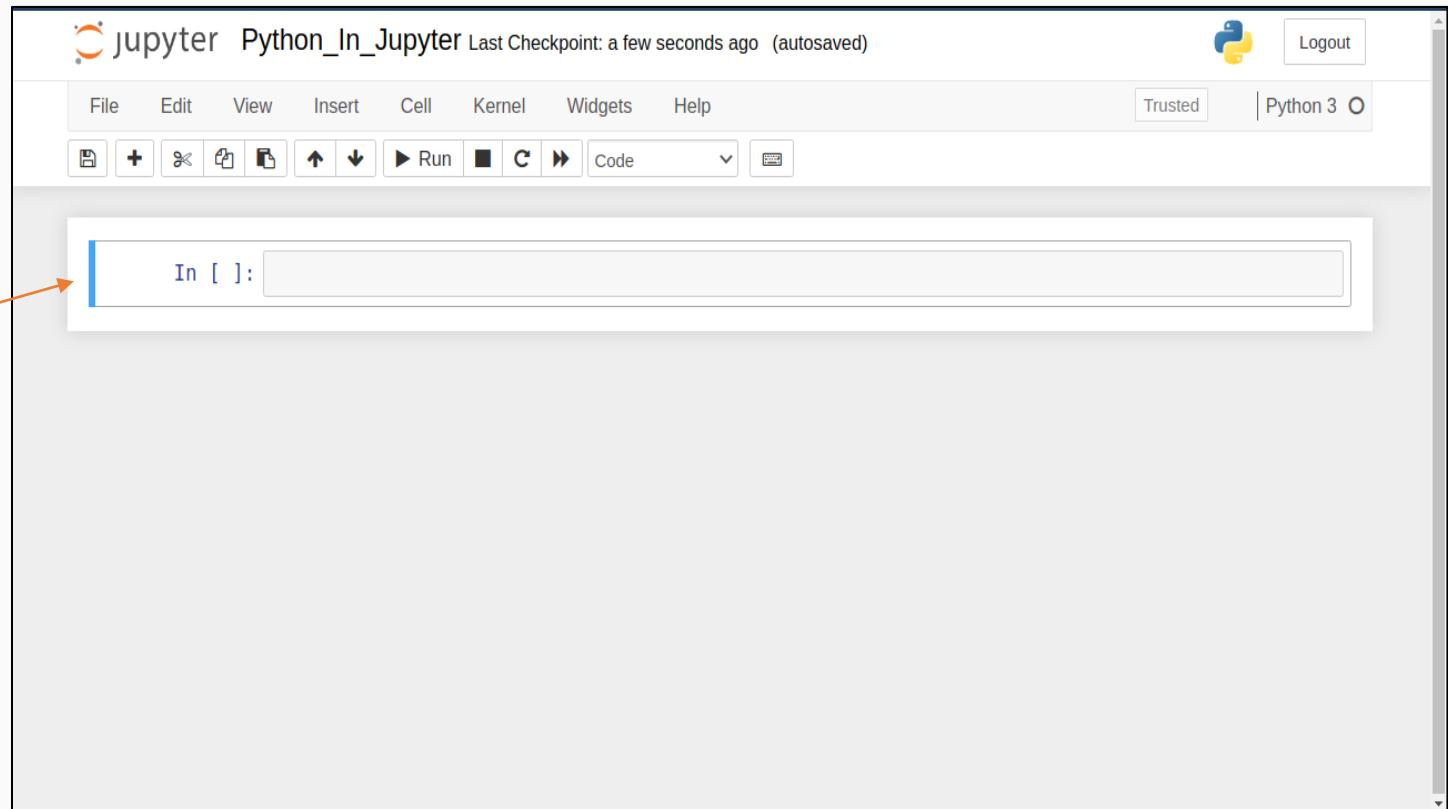
Anaconda for Ubuntu (11/11)

- This is how a Notebook looks like.



Implementing Python in Jupyter Notebook (1/2)

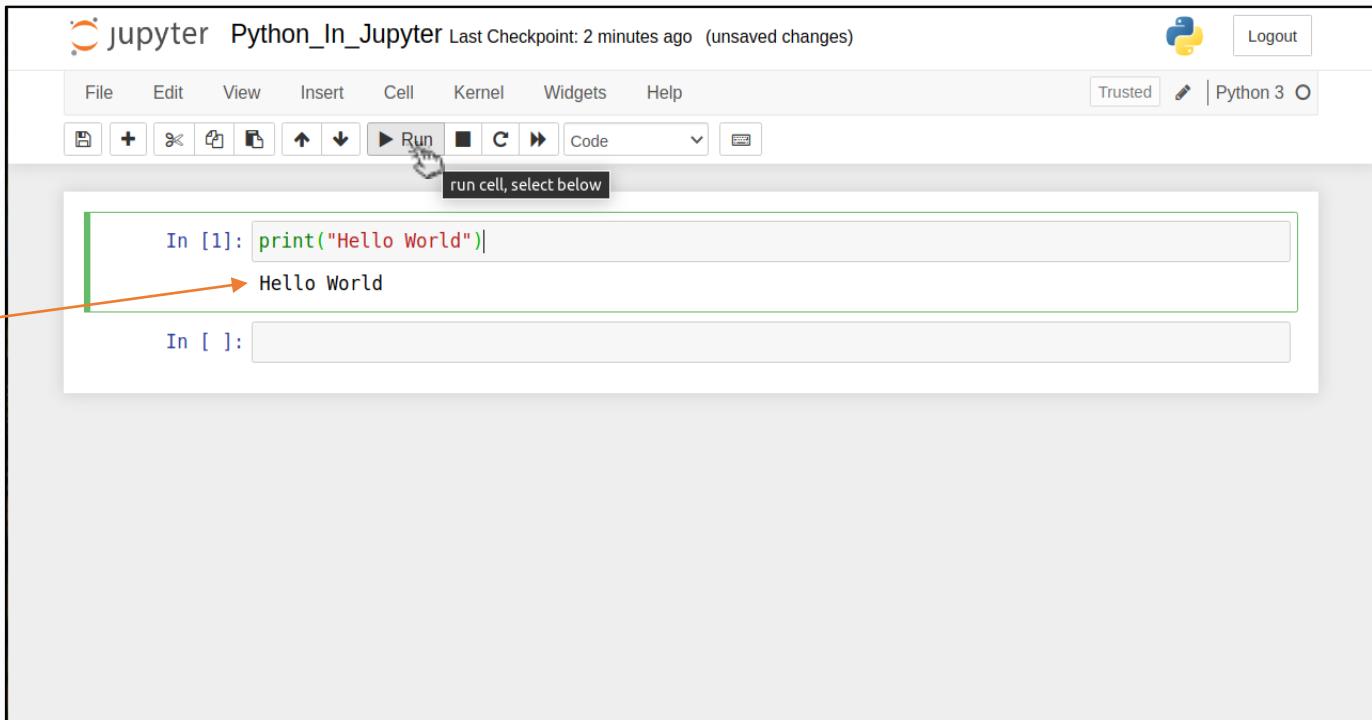
Upon launching Jupyter Notebook, this is what you will see!



This is where we write our code, called a cell. You can have as many of them as you want.

Implementing Python in Jupyter Notebook (2/2)

- Write your Python code inside a cell.
- Select the cell by clicking on it and then click on the Run button to execute the code.
- Output of a cell is produced right below it.



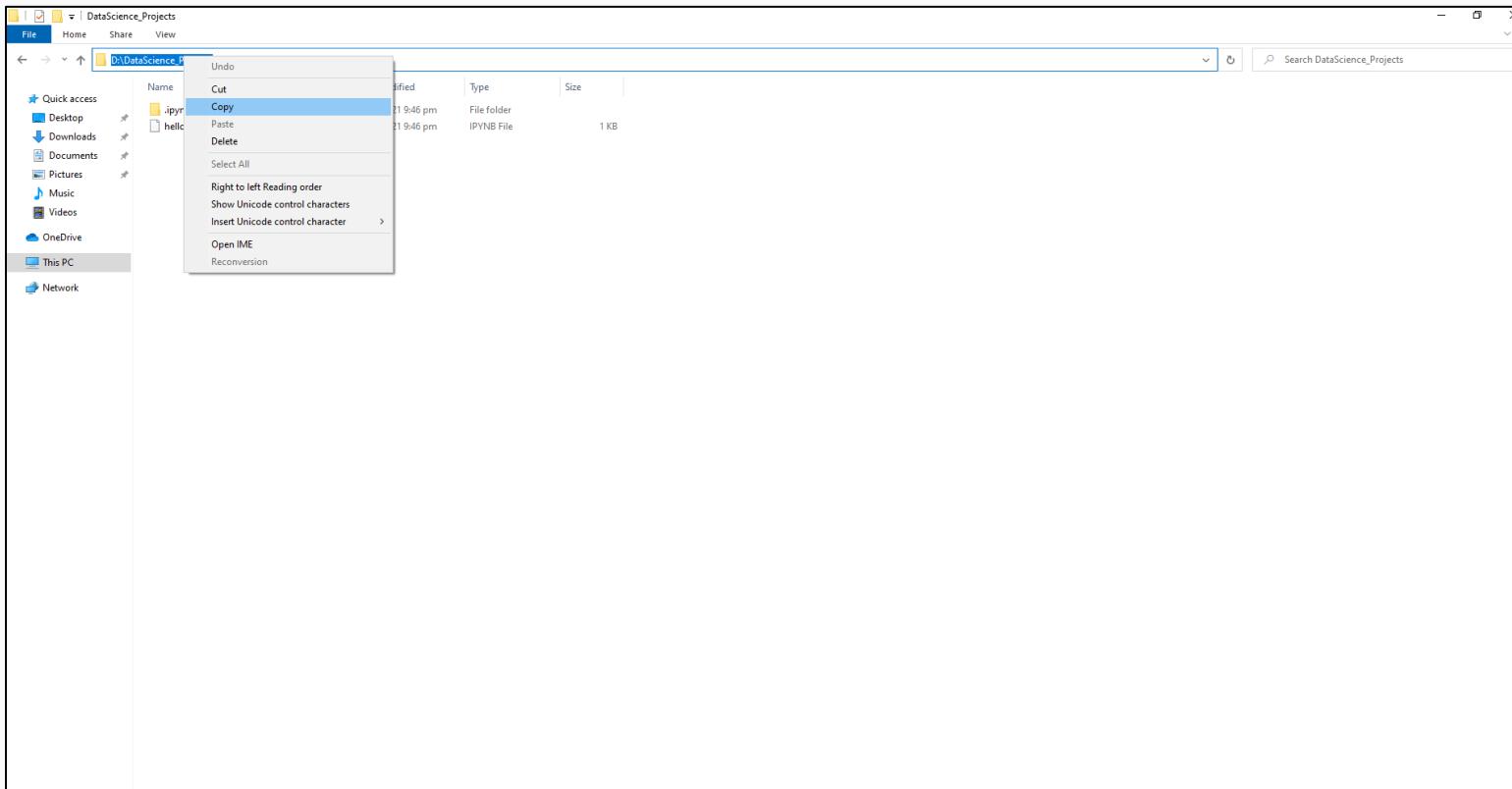
The screenshot shows a Jupyter Notebook interface with the title "jupyter Python_In_Jupyter" and a status bar indicating "Last Checkpoint: 2 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. A "Logout" button is also present. The main area displays a single code cell:

```
In [1]: print("Hello World")
```

The output of the cell is "Hello World", which is highlighted with a green border. An orange arrow points from the word "output" on the left to the "Hello World" text. The cell has a blue header "In []:".

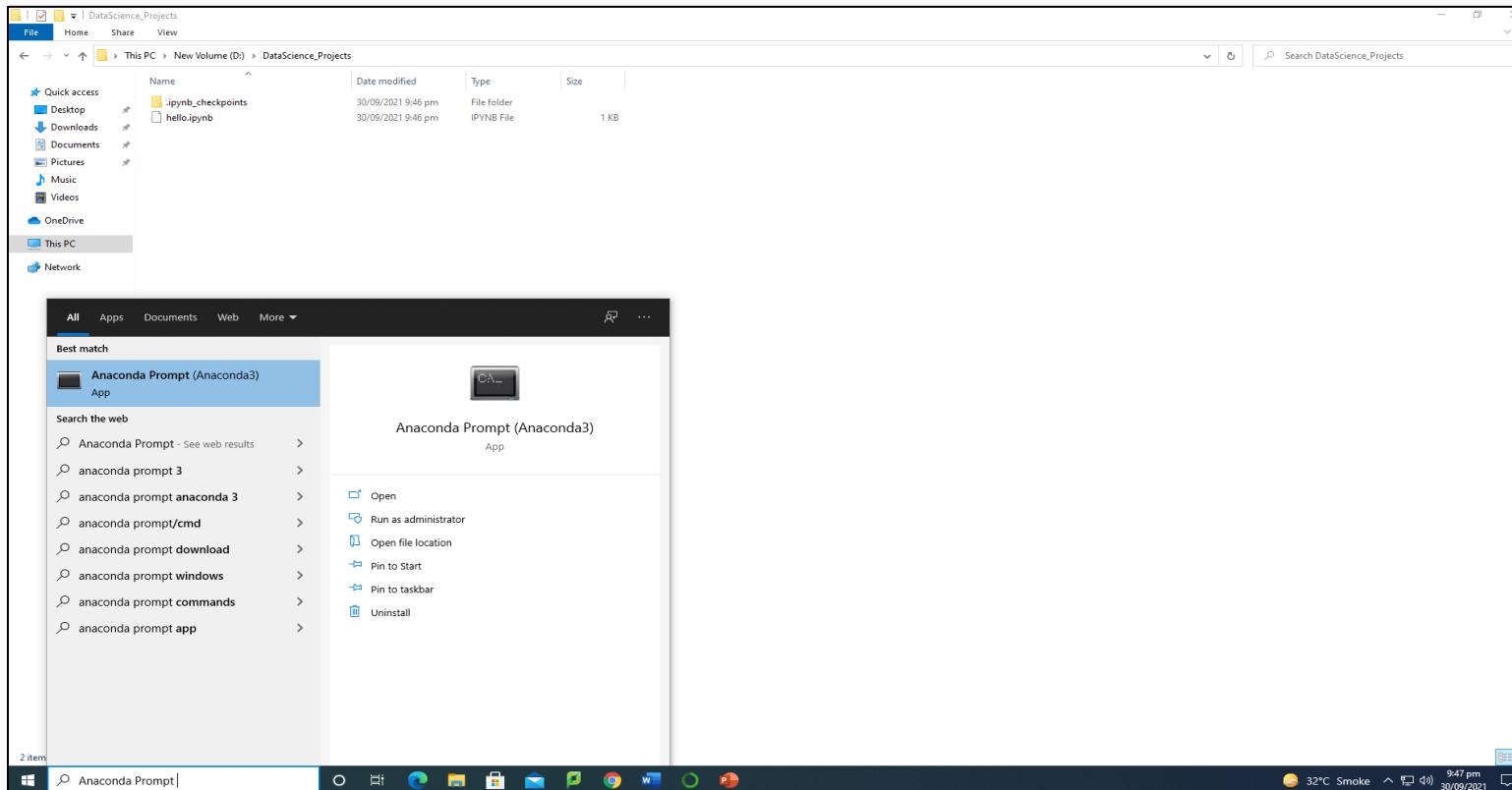
Managing Directories in Jupyter Notebook – Windows (1/7)

- To open Jupyter Notebook in a particular directory/folder in windows, copy the path of the directory.



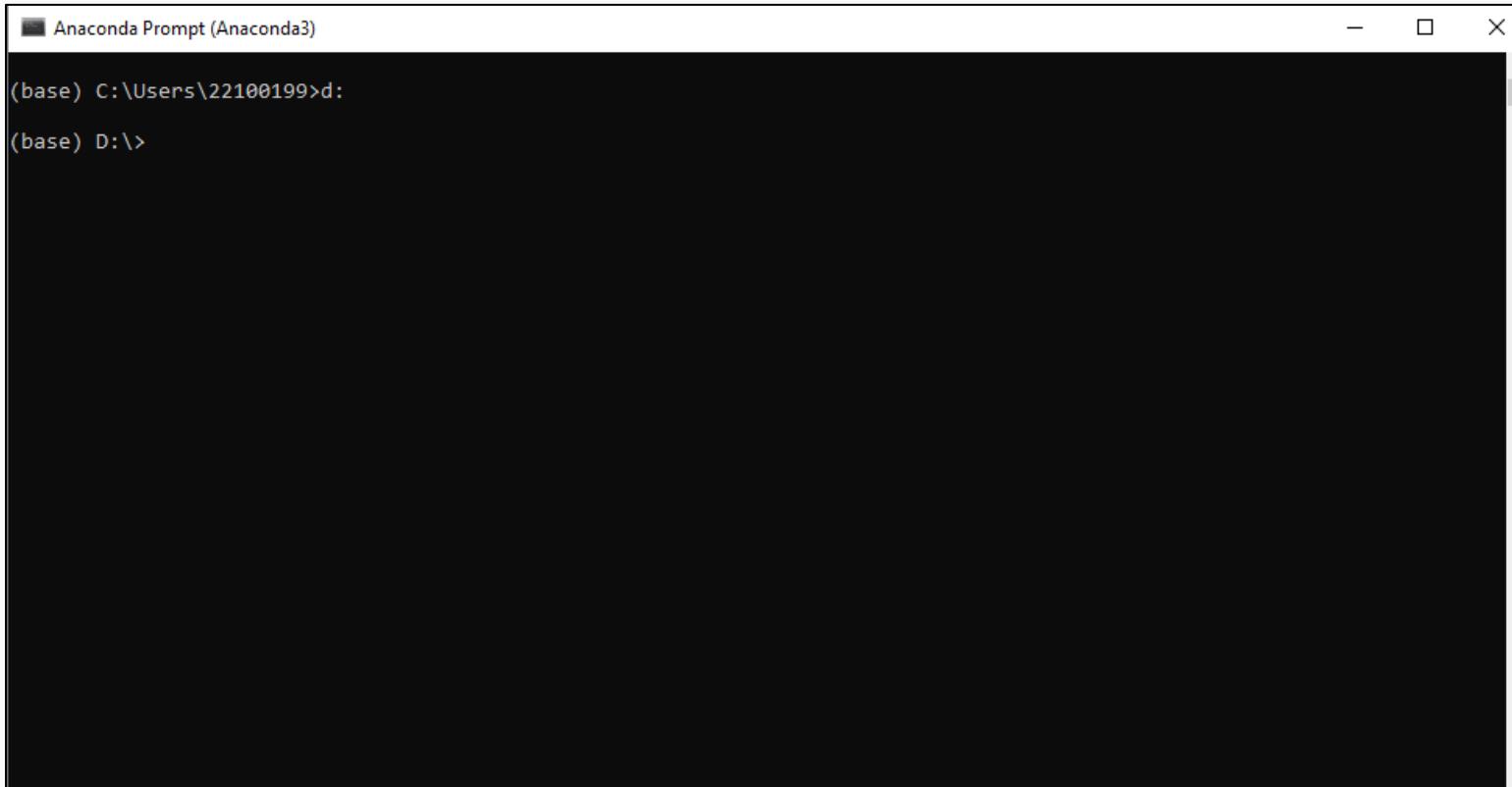
Managing Directories in Jupyter Notebook – Windows (2/7)

- In your search bar type Anaconda Prompt and launch it
 - Anaconda Prompt is the CLI of Anaconda



Managing Directories in Jupyter Notebook – Windows (3/7)

- Anaconda Prompt opens up in your C drive. Since we want to move to the D drive, we type d: and hit enter.

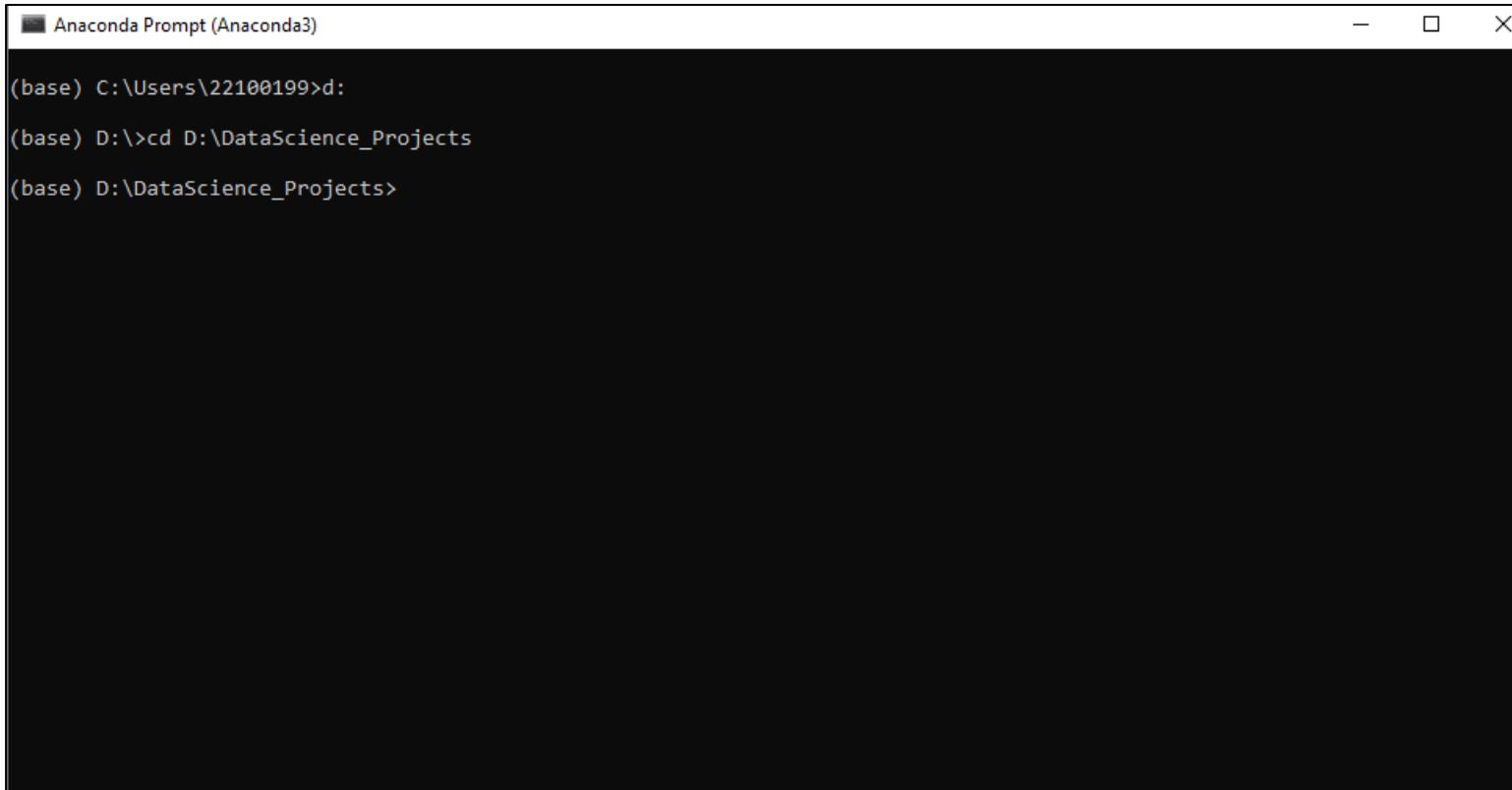


The screenshot shows a terminal window titled "Anaconda Prompt (Anaconda3)". The command prompt "(base)" is followed by the current directory "C:\Users\22100199>". The user then types "d:" and presses enter, changing the directory to "D:\>".

```
anaconda Prompt (Anaconda3)
(base) C:\Users\22100199>d:
(base) D:\>
```

Managing Directories in Jupyter Notebook – Windows (4/7)

- Next, to switch to the desired directory we type **cd** and then **paste the path of the directory**.



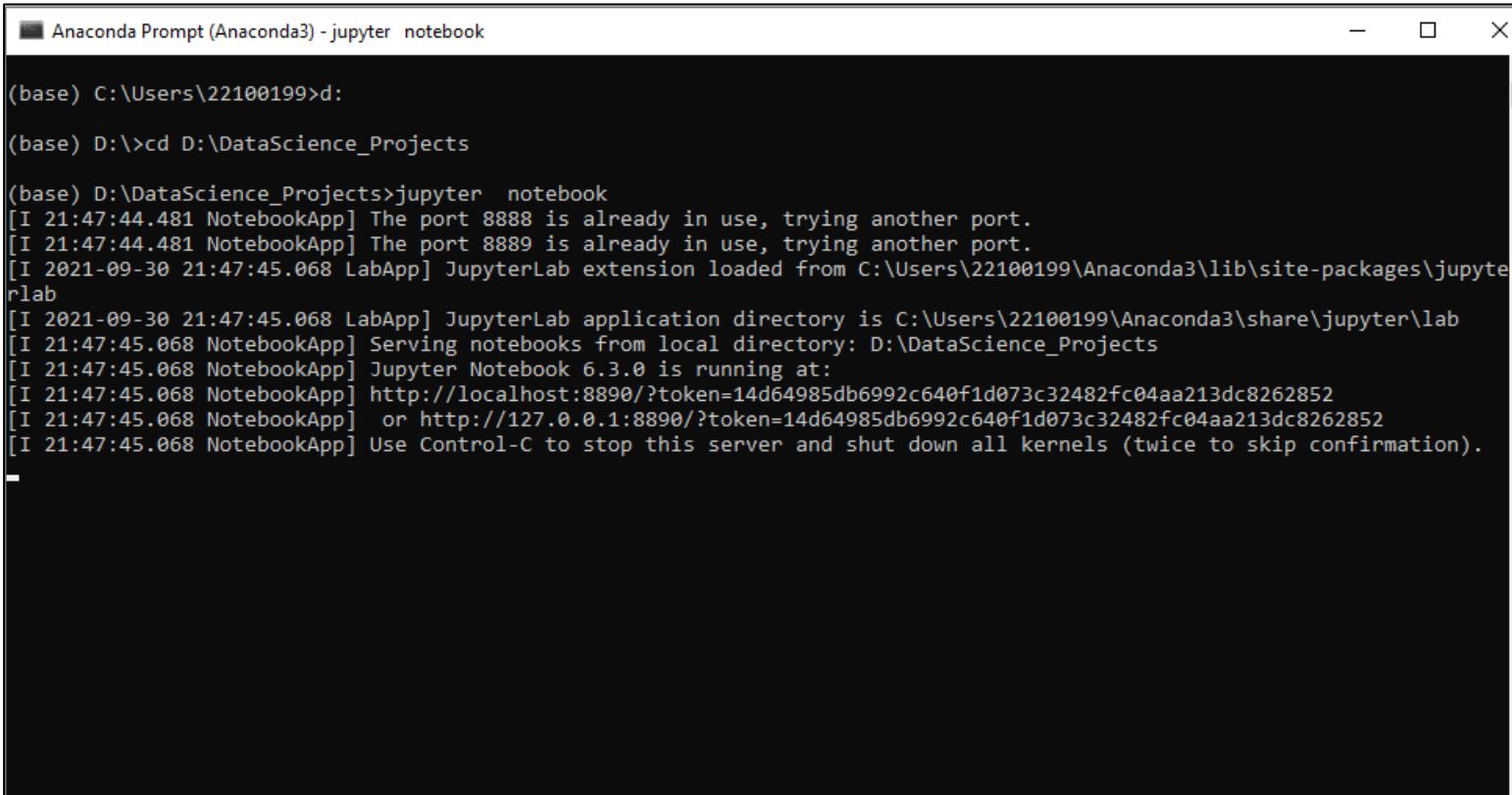
A screenshot of the Anaconda Prompt window titled "Anaconda Prompt (Anaconda3)". The window has a dark background and light-colored text. It shows the following command sequence:

```
(base) C:\Users\22100199>d:  
(base) D:\>cd D:\DataScience_Projects  
(base) D:\DataScience_Projects>
```

The window includes standard operating system controls (minimize, maximize, close) at the top right.

Managing Directories in Jupyter Notebook – Windows (5/7)

- Finally, we launch Jupyter Notebook by typing jupyter notebook.
- Jupyter Notebook takes a while to open.



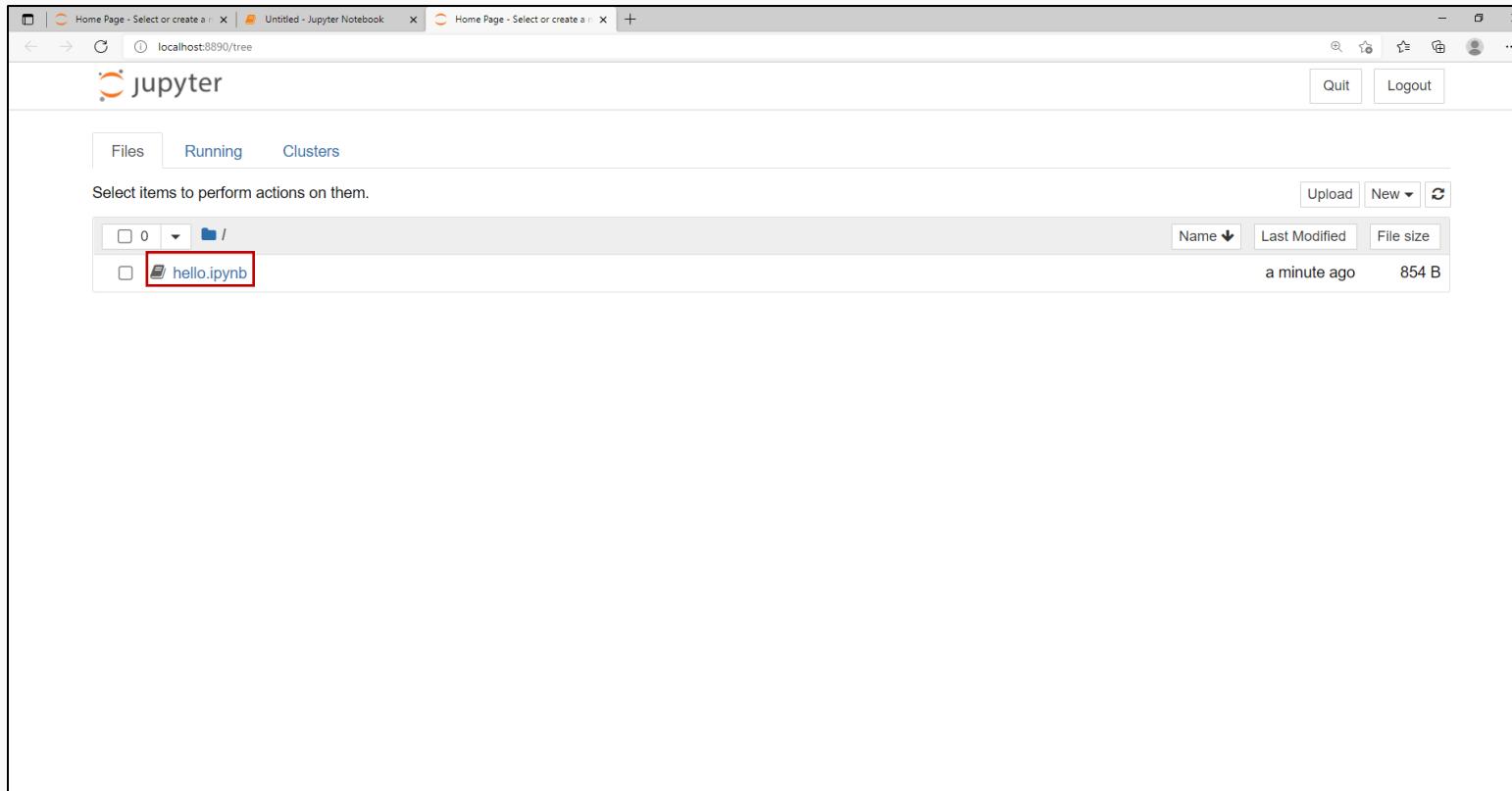
```
Anaconda Prompt (Anaconda3) - jupyter notebook

(base) C:\Users\22100199>d:
(base) D:\>cd D:\DataScience_Projects

(base) D:\DataScience_Projects>jupyter notebook
[I 21:47:44.481 NotebookApp] The port 8888 is already in use, trying another port.
[I 21:47:44.481 NotebookApp] The port 8889 is already in use, trying another port.
[I 2021-09-30 21:47:45.068 LabApp] JupyterLab extension loaded from C:\Users\22100199\Anaconda3\lib\site-packages\jupyterlab
[I 2021-09-30 21:47:45.068 LabApp] JupyterLab application directory is C:\Users\22100199\Anaconda3\share\jupyter\lab
[I 21:47:45.068 NotebookApp] Serving notebooks from local directory: D:\DataScience_Projects
[I 21:47:45.068 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 21:47:45.068 NotebookApp] http://localhost:8890/?token=14d64985db6992c640f1d073c32482fc04aa213dc8262852
[I 21:47:45.068 NotebookApp] or http://127.0.0.1:8890/?token=14d64985db6992c640f1d073c32482fc04aa213dc8262852
[I 21:47:45.068 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

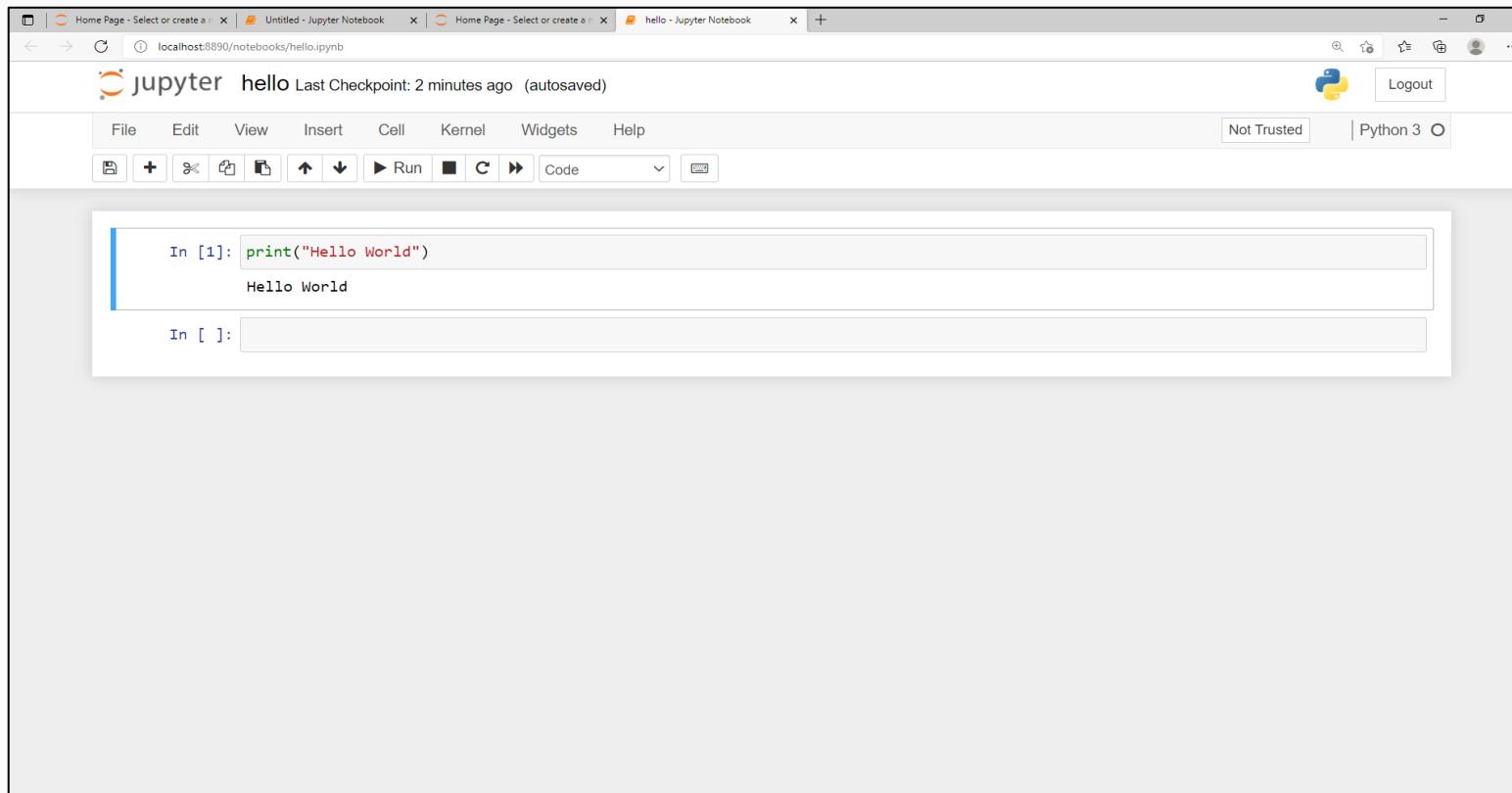
Managing Directories in Jupyter Notebook – Windows (6/7)

- Once it opens, we can either open an existing notebook or create a new one.
- In this case we open the existing notebook named hello.ipynb.



Managing Directories in Jupyter Notebook – Windows (7/7)

- This is how hello.ipynb looks like.



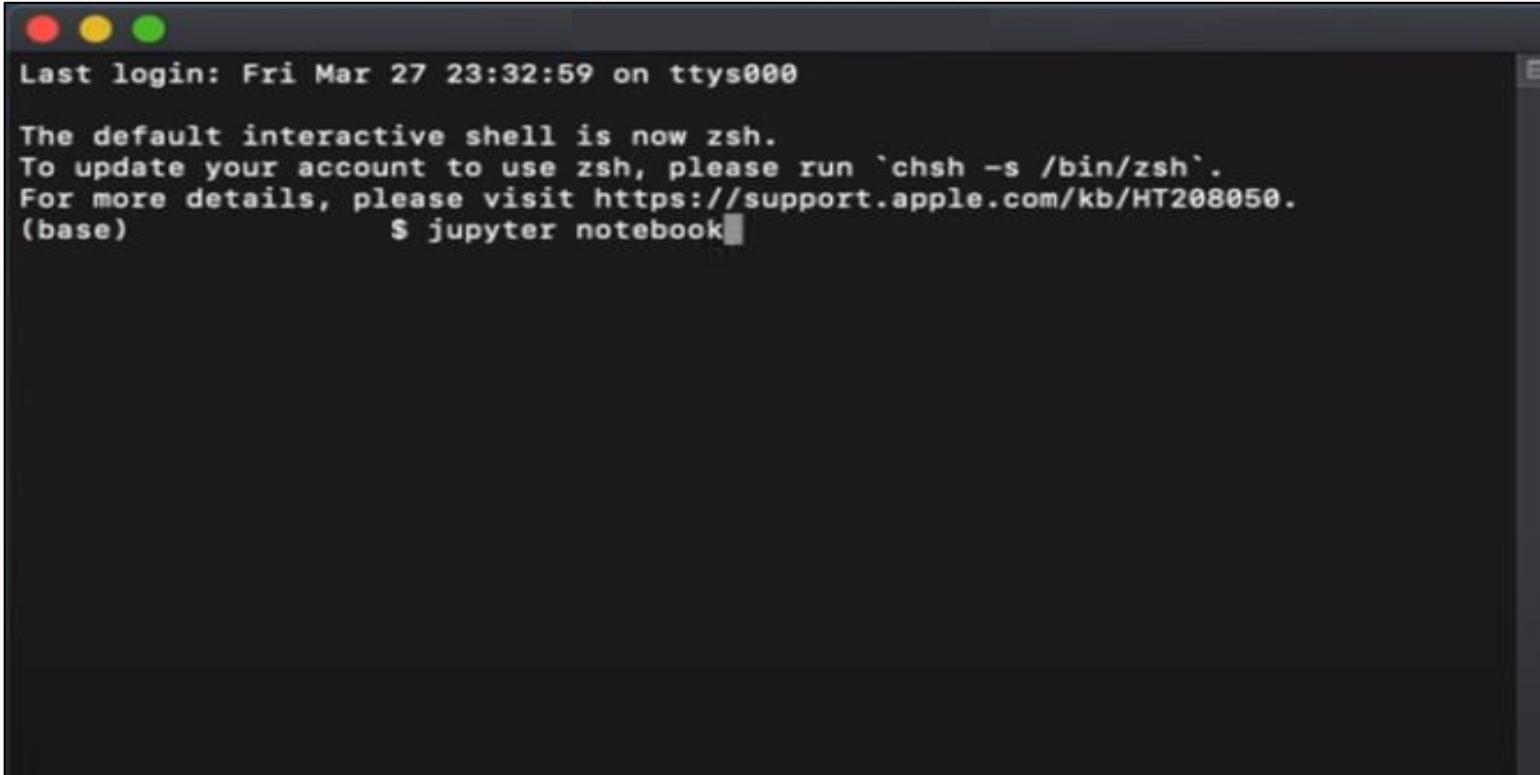
The screenshot shows a Jupyter Notebook interface running in a web browser. The title bar indicates the notebook is titled "hello". The toolbar includes standard options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help, along with a Python 3 kernel selector. The main workspace displays a single code cell:

```
In [1]: print("Hello World")
Hello World
```

The code cell contains the Python command `print("Hello World")`, which has been executed and produced the output "Hello World". A second, empty code cell is visible below it, labeled "In []:".

Managing Directories in Jupyter Notebook – Mac (1/3)

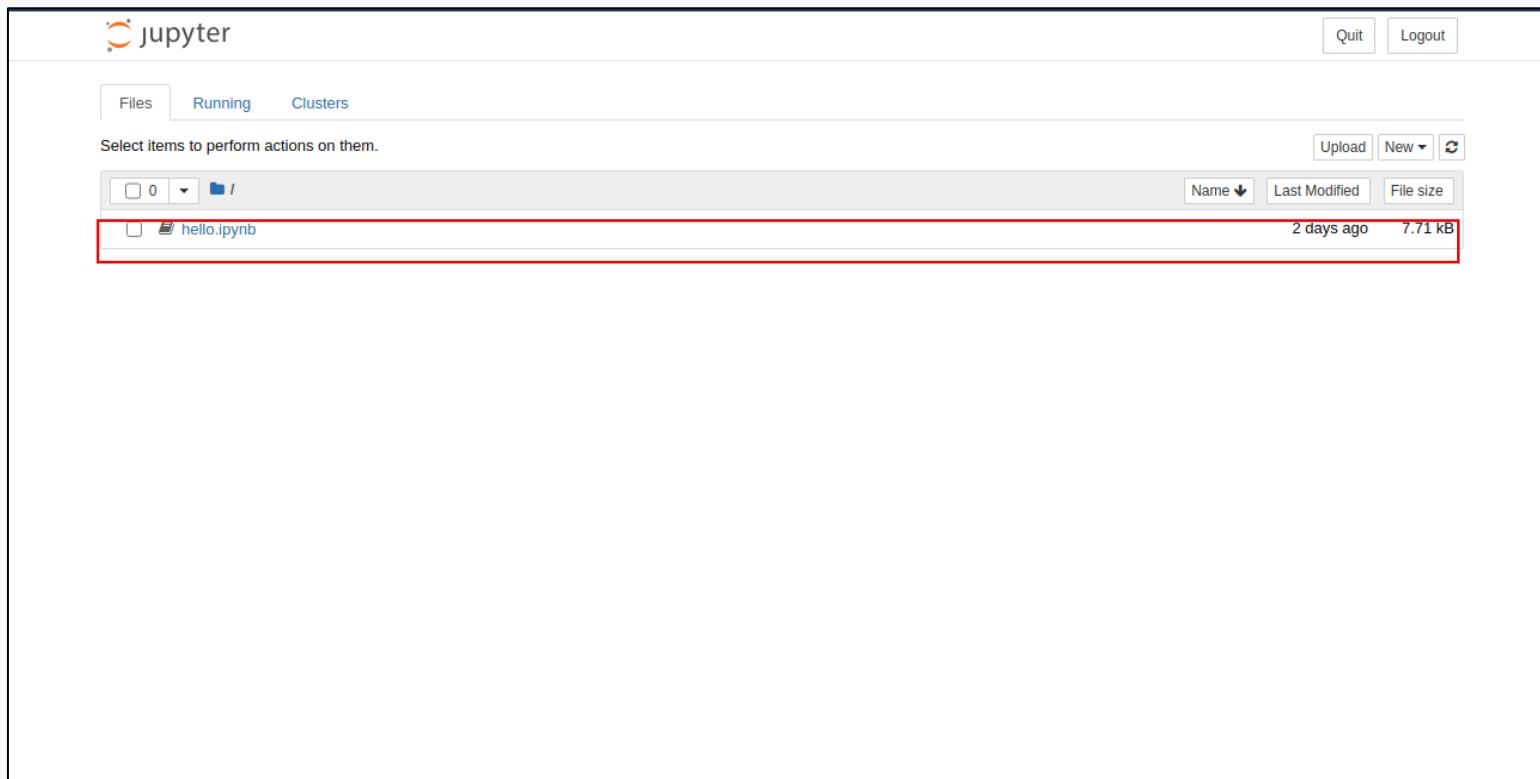
- To open Jupyter Notebook in a particular directory/folder in Mac, open that directory in terminal.
- Type ‘jupyter notebook’ without the quotation marks and hit Enter.



A screenshot of a Mac OS X terminal window. The window has a dark gray header bar with three colored window control buttons (red, yellow, green) on the left. The main area of the terminal is black and contains white text. At the top, it shows system information: "Last login: Fri Mar 27 23:32:59 on ttys000". Below that, there is a message about the default interactive shell being zsh, followed by instructions to update the account and a link to support.apple.com. At the bottom of the terminal window, the command "\$ jupyter notebook" is typed, with the cursor positioned at the end of the line.

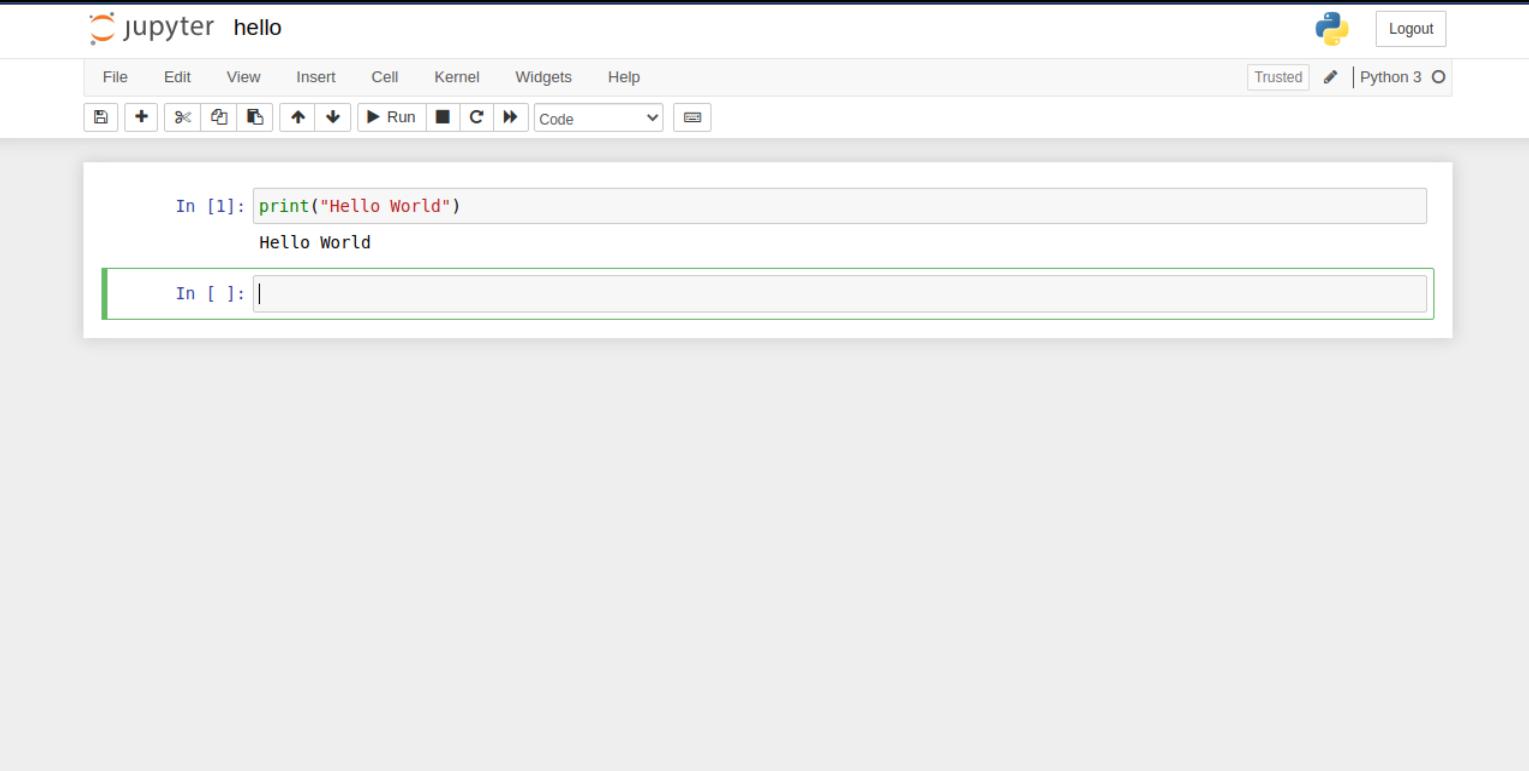
Managing Directories in Jupyter Notebook – Mac (2/3)

- Jupyter Notebook will launch in your browser.
- You can create a new Notebook or open an existing one.
- Let's open this 'hello.ipynb' file.



Managing Directories in Jupyter Notebook – Mac (3/3)

- ‘hello.ipynb’ contains a simple print statement.



The screenshot shows a Jupyter Notebook interface titled "jupyter hello". The top bar includes the title, a Python logo icon, and a "Logout" button. The menu bar has options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Below the menu is a toolbar with icons for file operations like Open, Save, and New, and cell execution controls like Run, Cell, and Kernel. A status bar indicates the cell is "Trusted" and uses "Python 3".

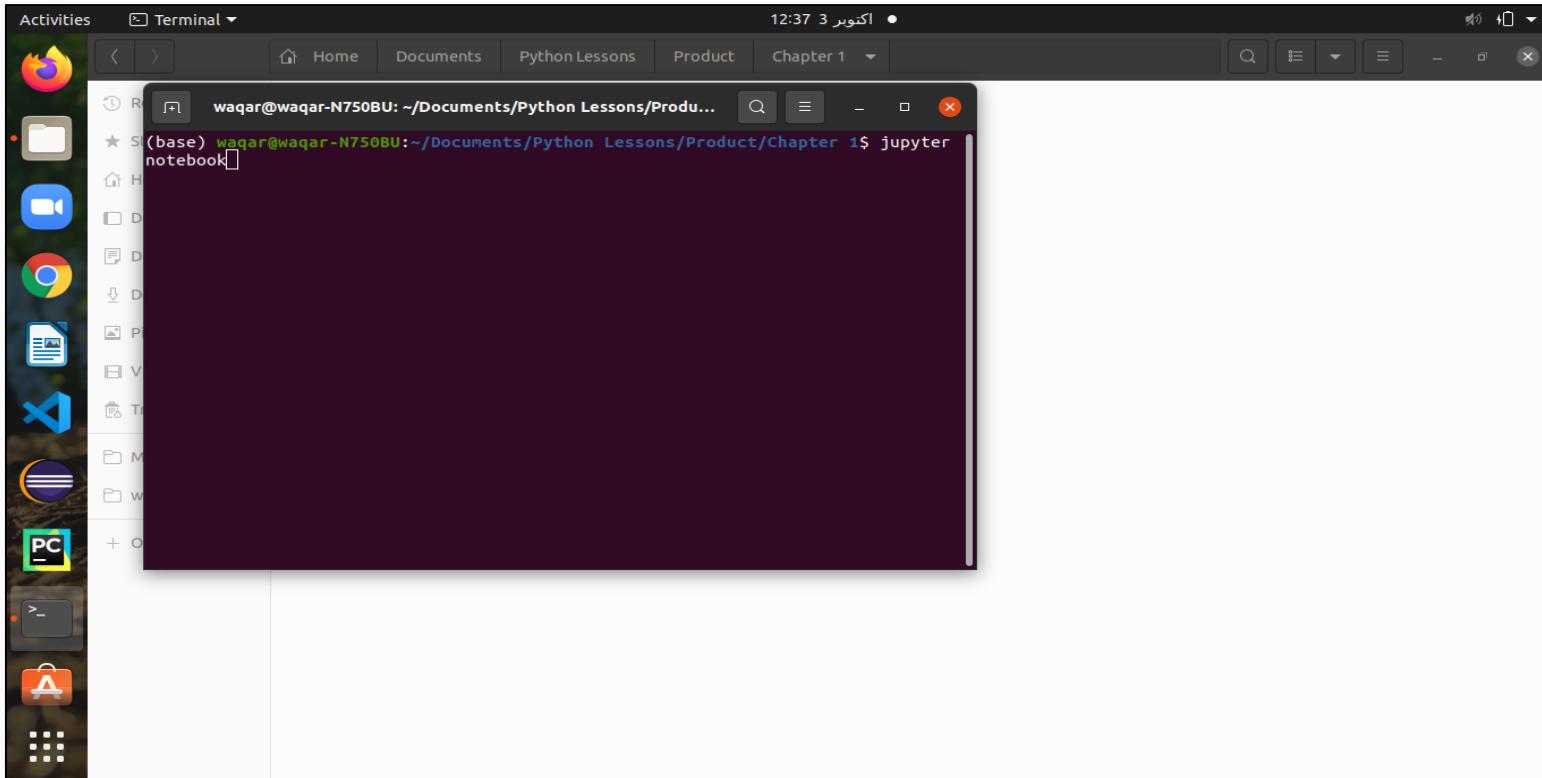
The main area displays a single code cell:

```
In [1]: print("Hello World")
Hello World
```

The input line "In [1]: print("Hello World")" is in blue. The output "Hello World" is in black. A new cell prompt "In []:" is visible at the bottom of the code cell, highlighted with a green border.

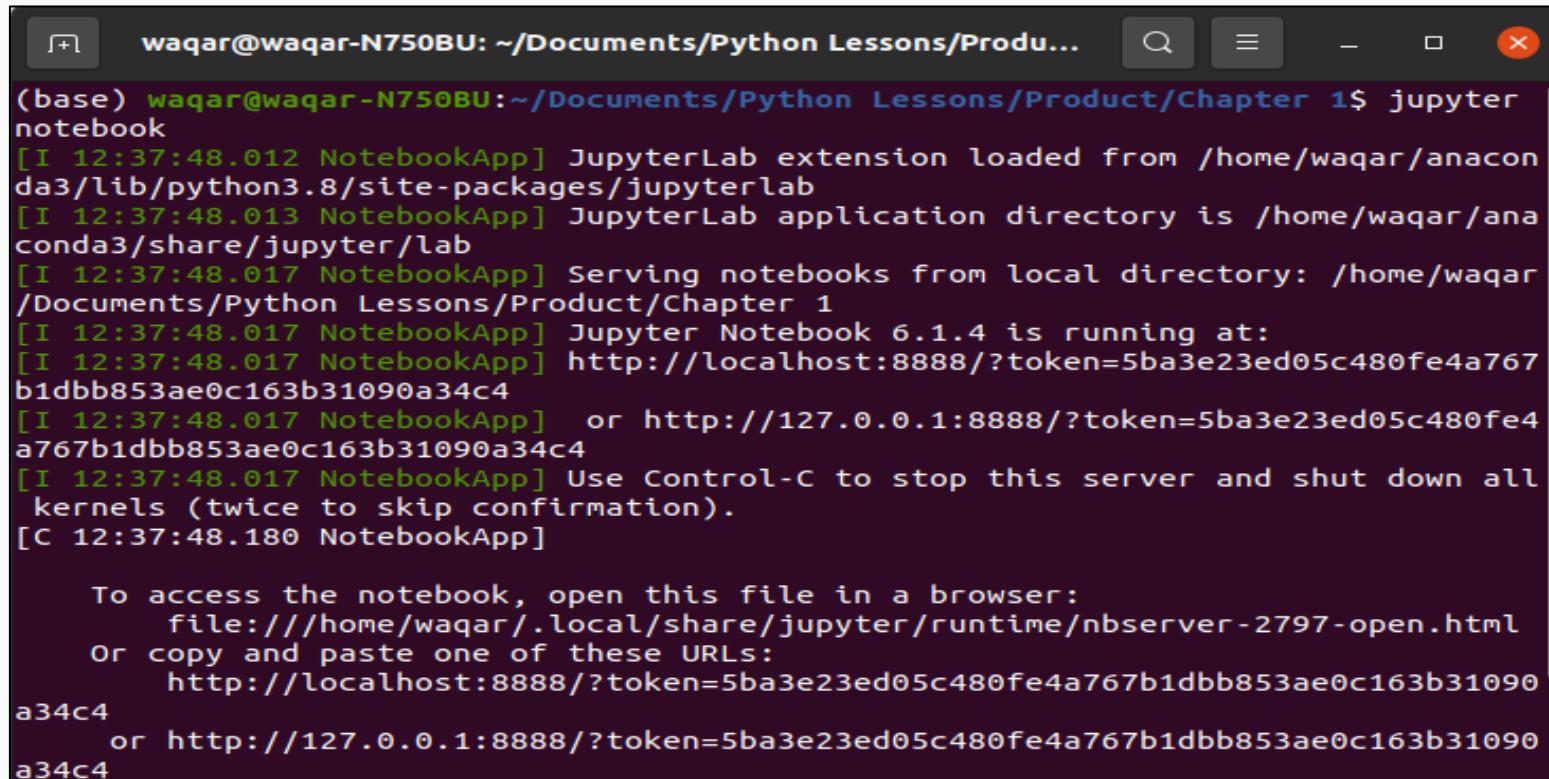
Managing Directories in Jupyter Notebook – Ubuntu (1/4)

- To open Jupyter Notebook in a particular directory/folder in Ubuntu, go to that directory, right click and select ‘open in terminal’.
- Type ‘jupyter notebook’ without quotation marks and hit Enter.



Managing Directories in Jupyter Notebook – Ubuntu (2/4)

- Wait for sometime.



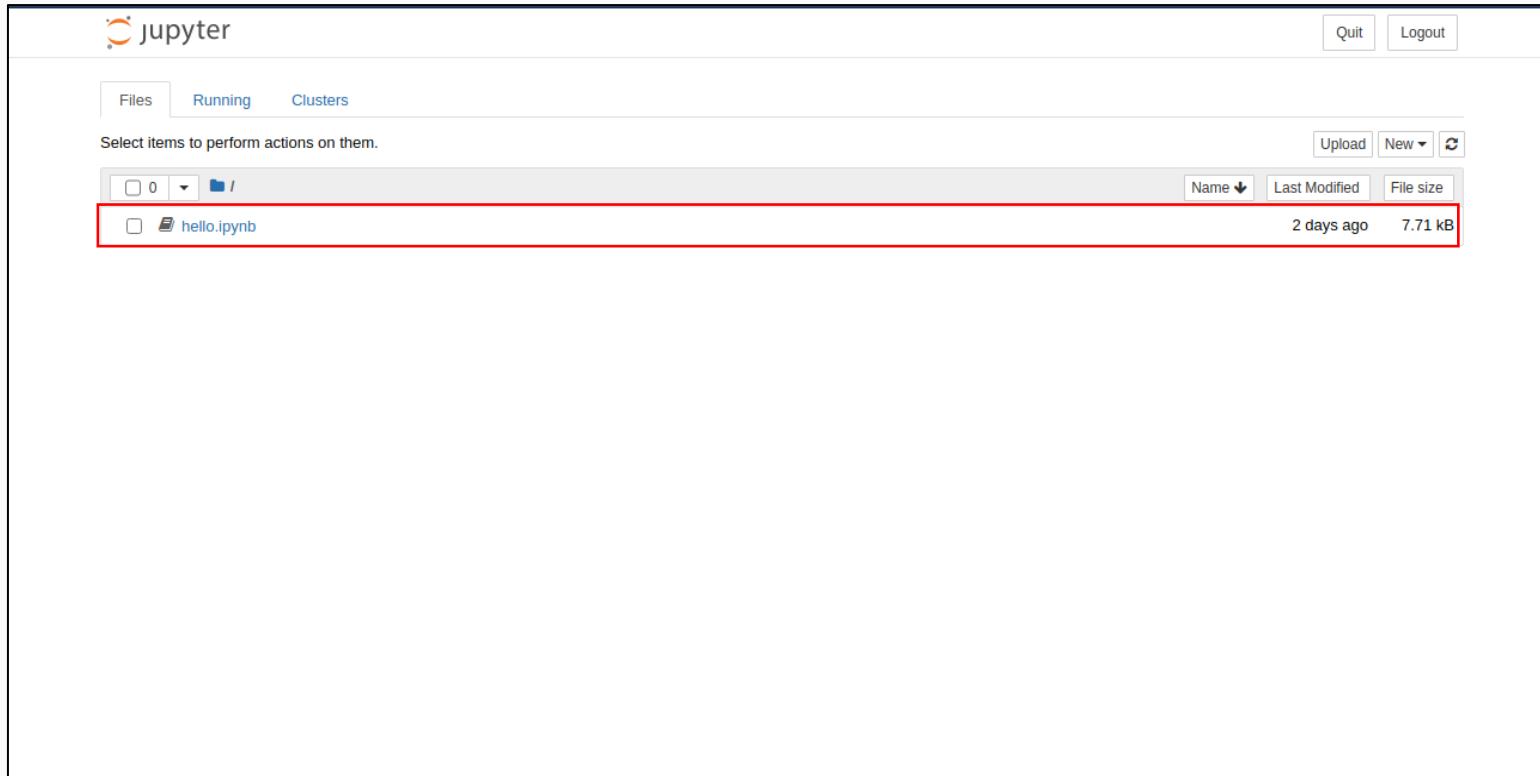
The screenshot shows a terminal window titled "waqar@waqar-N750BU: ~/Documents/Python Lessons/Product/Chapter 1\$". The window displays the output of a "jupyter notebook" command. The output indicates that the JupyterLab extension was loaded from /home/waqar/anaconda3/lib/python3.8/site-packages/jupyterlab, the application directory is /home/waqar/anaconda3/share/jupyter/lab, notebooks are being served from /home/waqar/Documents/Python Lessons/Product/Chapter 1, and the Jupyter Notebook 6.1.4 server is running at http://localhost:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4 or http://127.0.0.1:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4. It also suggests using Control-C to stop the server and shut down all kernels (twice to skip confirmation). The terminal then prompts the user to access the notebook via a browser, providing the URL file:///home/waqar/.local/share/jupyter/runtime/nbserver-2797-open.html or one of the two alternative URLs above it.

```
(base) waqar@waqar-N750BU:~/Documents/Python Lessons/Product/Chapter 1$ jupyter notebook
[I 12:37:48.012 NotebookApp] JupyterLab extension loaded from /home/waqar/anaconda3/lib/python3.8/site-packages/jupyterlab
[I 12:37:48.013 NotebookApp] JupyterLab application directory is /home/waqar/anaconda3/share/jupyter/lab
[I 12:37:48.017 NotebookApp] Serving notebooks from local directory: /home/waqar/Documents/Python Lessons/Product/Chapter 1
[I 12:37:48.017 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 12:37:48.017 NotebookApp] http://localhost:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4
[I 12:37:48.017 NotebookApp] or http://127.0.0.1:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4
[I 12:37:48.017 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:37:48.180 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/waqar/.local/share/jupyter/runtime/nbserver-2797-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4
or http://127.0.0.1:8888/?token=5ba3e23ed05c480fe4a767b1dbb853ae0c163b31090a34c4
```

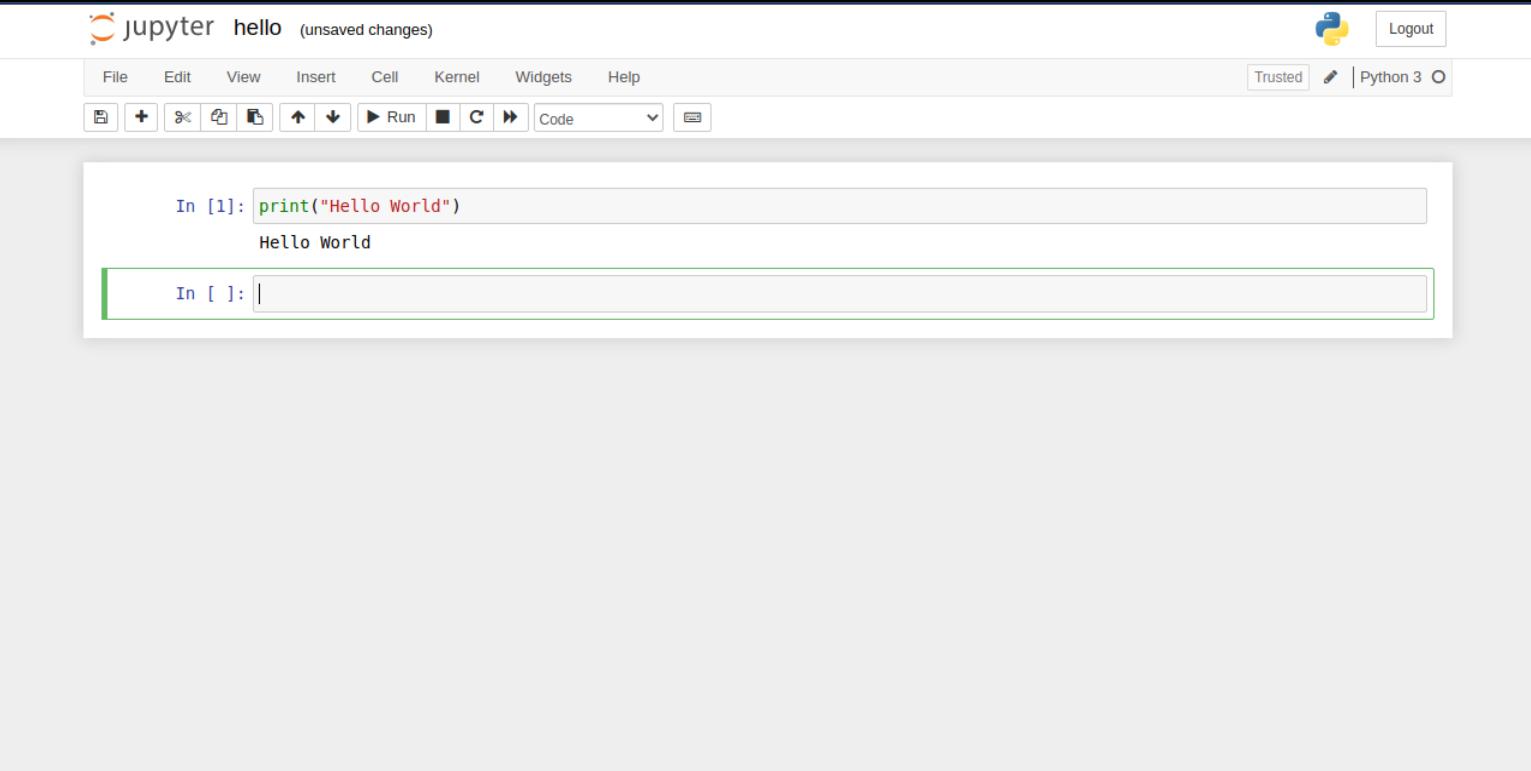
Managing Directories in Jupyter Notebook – Ubuntu (3/4)

- Jupyter Notebook will launch in your default browser.
- You can create a new Notebook or open an existing one.
- Let's open this 'hello.ipynb' file.



Managing Directories in Jupyter Notebook – Mac (4/4)

- ‘hello.ipynb’ contains a simple print statement.



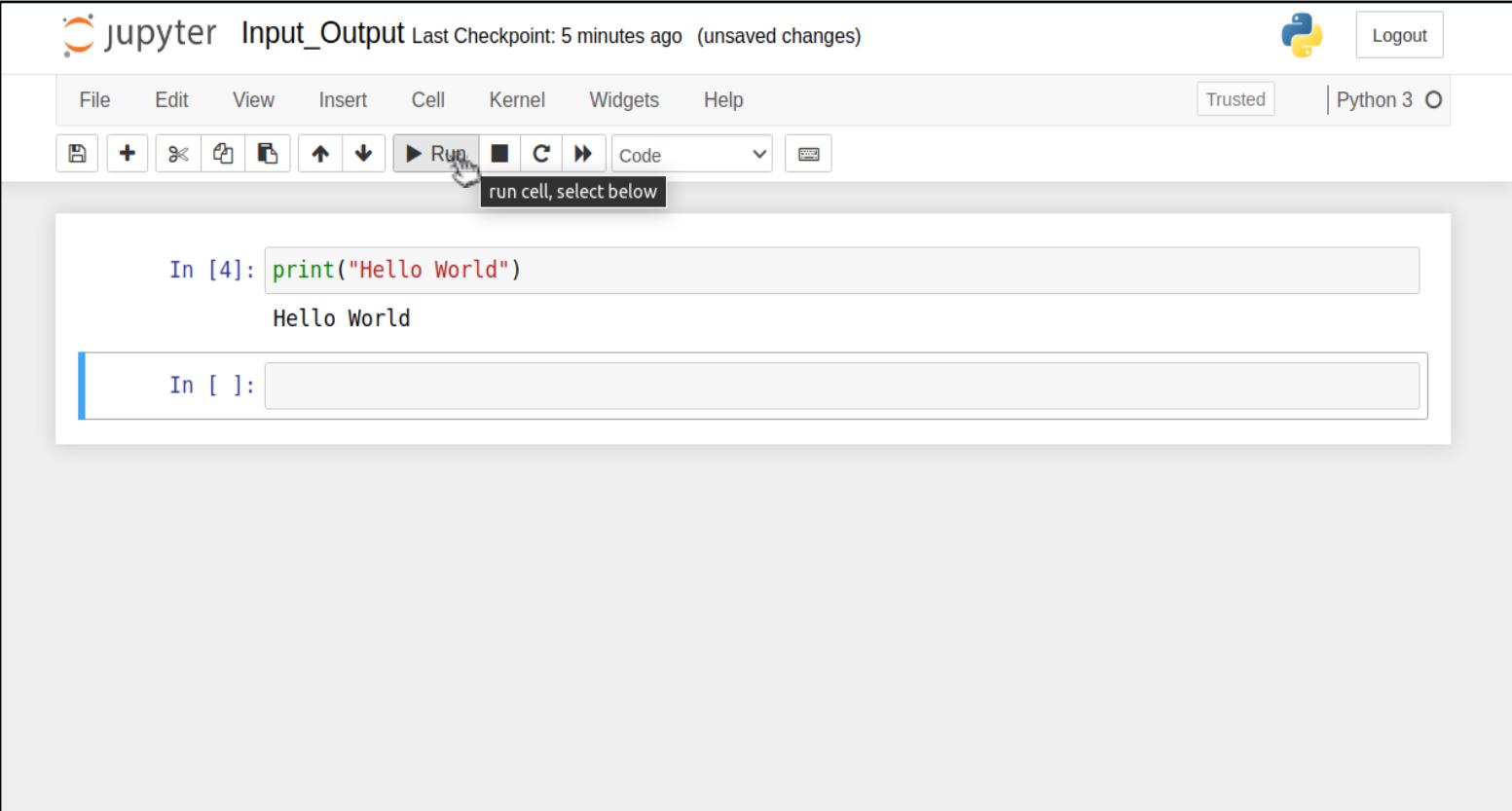
The screenshot shows a Jupyter Notebook interface on a Mac. The title bar reads "jupyter hello (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and various cell type icons. A Python logo icon and "Logout" button are also present. The main area displays a single code cell:

```
In [1]: print("Hello World")
Hello World
```

The input field for the next cell is highlighted with a green border.

Output

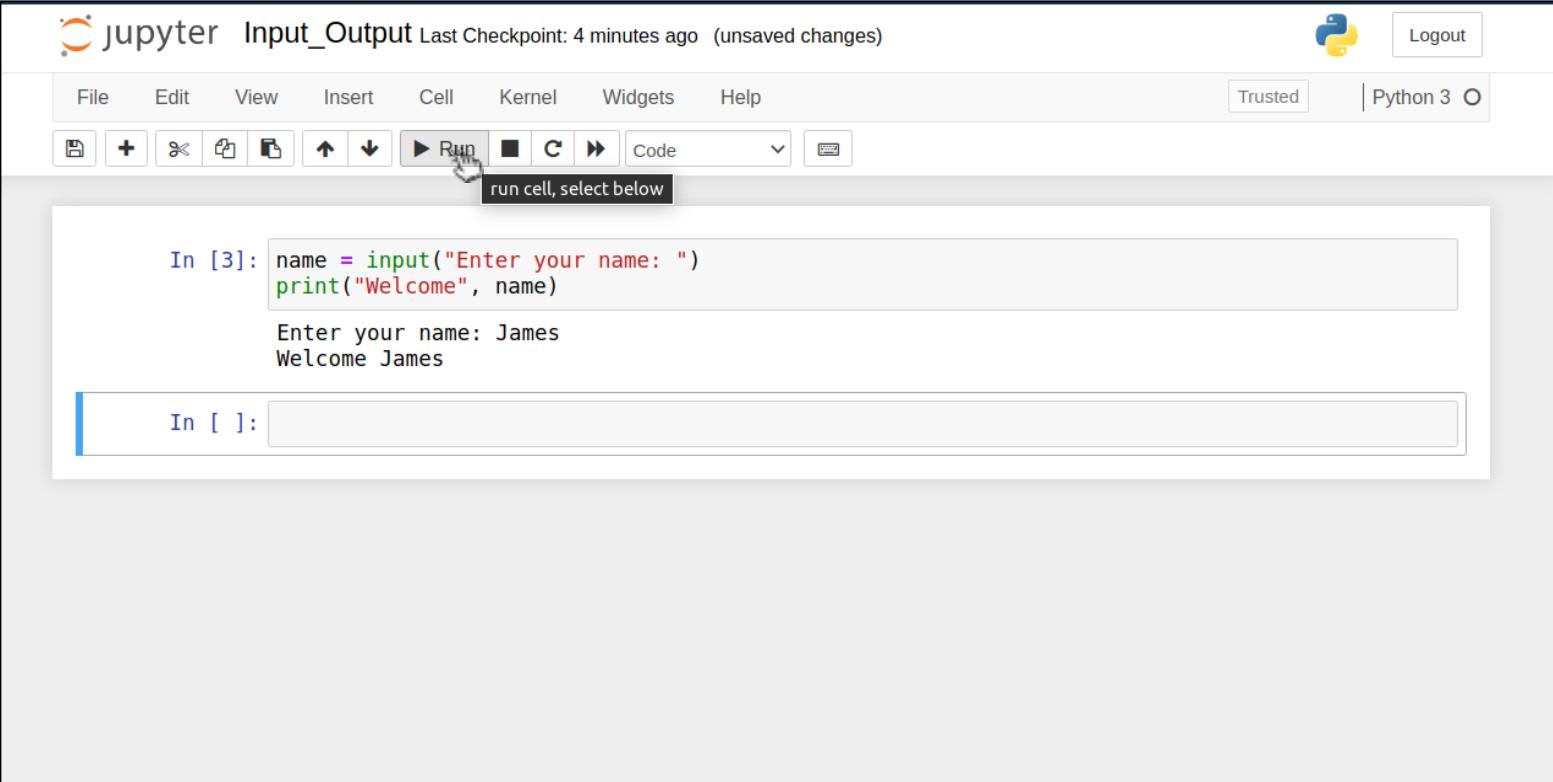
Use the print() function to produce output.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Input_Output" and a status bar indicating "Last Checkpoint: 5 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. A "Run" button is highlighted with a cursor. Below the toolbar, a code cell contains the Python command `In [4]: print("Hello World")`. The output cell shows the text "Hello World". A new input cell, "In []:", is currently selected.

Input

- Use the `input()` function to take input from the user.
- Here we are providing James as input.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Input_Output". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. A "Run" button is highlighted with a mouse cursor. Below the toolbar, a message says "run cell, select below". In the code cell, the following Python code is displayed:

```
In [3]: name = input("Enter your name: ")  
print("Welcome", name)
```

The output cell shows the result of running the code:

```
Enter your name: James  
Welcome James
```

A new input cell is ready for the next command, labeled "In []:".

Quiz Time

1. Which of the following is the correct input statement?

- a) Input("Enter your name: ")
- b) input("Enter your name: ")
- c) Input(Enter your name:)
- d) input(Enter your name:)

Quiz Time

1. Which of the following is the correct input statement?

- a) Input("Enter your name: ")
- b) input("Enter your name: ")**
- c) Input(Enter your name:)
- d) input(Enter your name:)

Working with different data types

- Python has following primitive data types
 - int
 - float
 - string
 - bool
- Python also has following built-in non-primitive data types
 - Lists
 - Dictionaries
 - Tuples

Working with different data types

int

An int (integer) is a whole number such as 1, 5, 65, 1000

float

A float is a decimal point number such as 1.5, 7.8, 100.0

string

A string is a sequence of characters which can include alphabets, numbers, whitespaces, and special characters. In python, we use single, double or triple quotes to denote a string.

- E.g., ‘Hello World’, “Hello World”, and “”Hello World”” are all valid strings

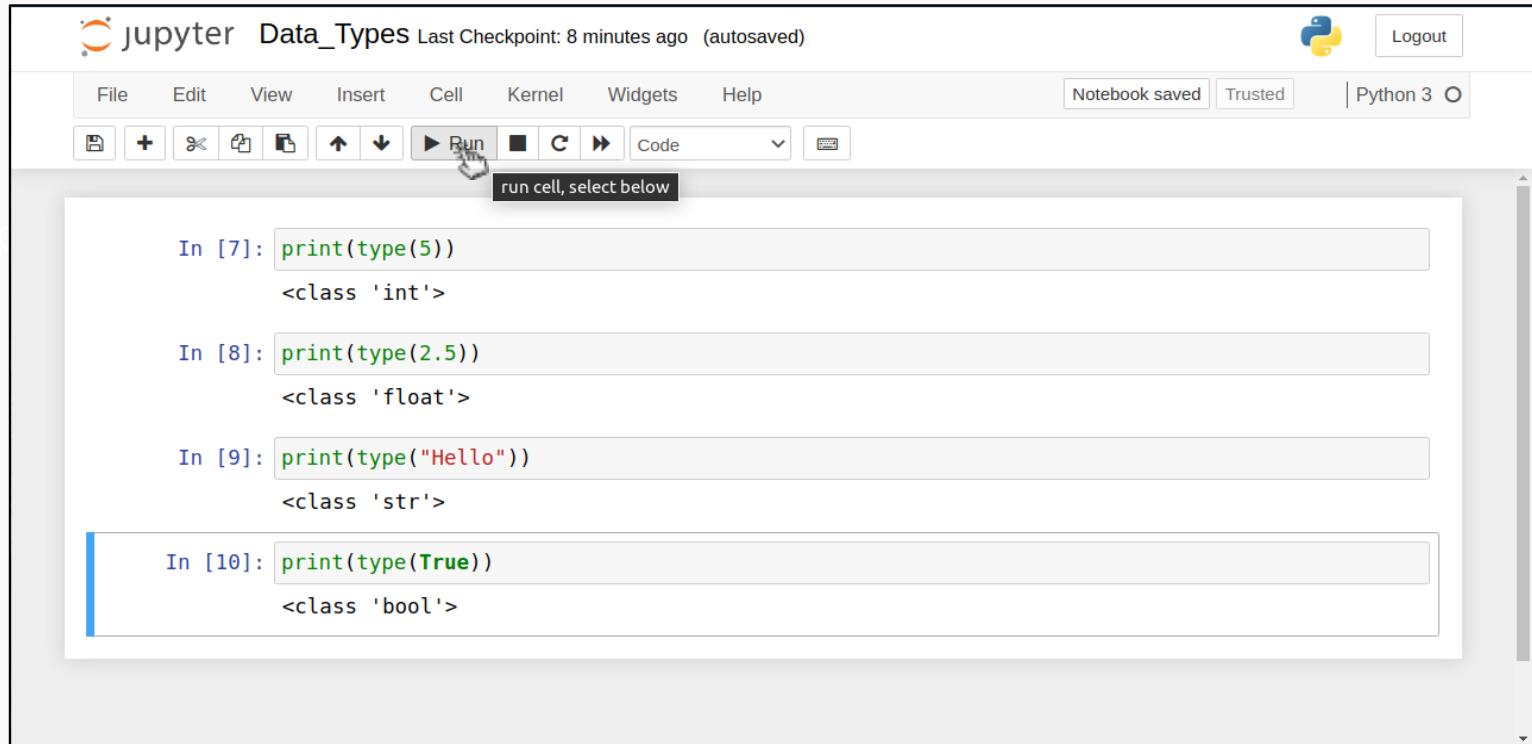
bool

A bool is either True or False

Working with different data types

type()

type() function is used to get the type of a value/variable.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Data_Types". The notebook has been autosaved 8 minutes ago. The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 kernel selector. A "Run" button is highlighted with a mouse cursor, and a tooltip says "run cell, select below". The notebook contains four code cells:

- In [7]: `print(type(5))`
Output: <class 'int'>
- In [8]: `print(type(2.5))`
Output: <class 'float'>
- In [9]: `print(type("Hello"))`
Output: <class 'str'>
- In [10]: `print(type(True))`
Output: <class 'bool'>

Variables

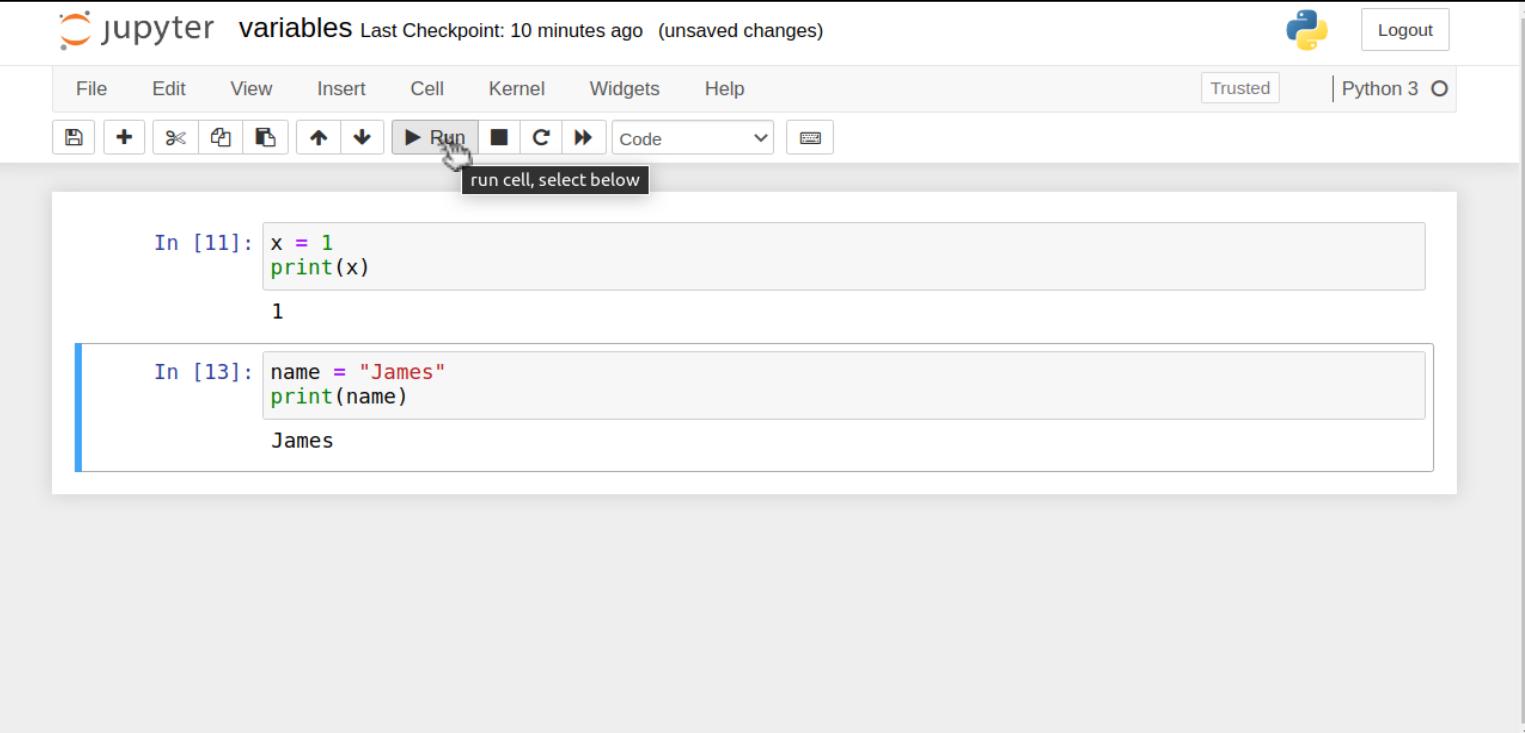
- Variables are placeholders for values.
- Python is a dynamically typed language, which means that the data types of variables are checked at runtime.
 - Hence, we do not need to declare the data type of a variable at the time of creating the variable.
- To print the value of a variable, pass the variable name to `print()` statement without quotation marks.

Rules for Variable Names

- Variable names can contain alphabets, integers, and/or underscore (_) character.
- Variable names can not begin with a number. They can only begin with a letter or an underscore (_).
- Variable names can not contain whitespaces and special characters.

Variables

Example in Jupyter



jupyter variables Last Checkpoint: 10 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [11]: `x = 1
print(x)`
1

In [13]: `name = "James"
print(name)`
James

Quiz Time

1. Which of the following is a valid variable name?

- a) _name
- b) @name
- c) 1name
- d) n a m e

2. What is the correct way of printing the value of a variable called num?

- a) print("num")
- b) print('num')
- c) print("")"num""")
- d) print(num)

Quiz Time

1. Which of the following is a valid variable name?

- a) **_name**
- b) @name
- c) 1name
- d) n a m e

2. What is the correct way of printing the value of a variable called num?

- a) print("num")
- b) print('num')
- c) print("")"num""")
- d) **print(num)**

Arithmetic Operators (1/3)

Addition

Use the plus (+) operator for addition

Subtraction

Use the minus (-) operator for subtraction

<https://unibio.zoom.us/j/83422537997?pwd=Z3VENmI0QTV2Q2xJWDIPQzhrVU5PZz09>

Multiplication

Use the asterisk (*) operator for multiplication

Arithmetic Operators (2/3)

Division

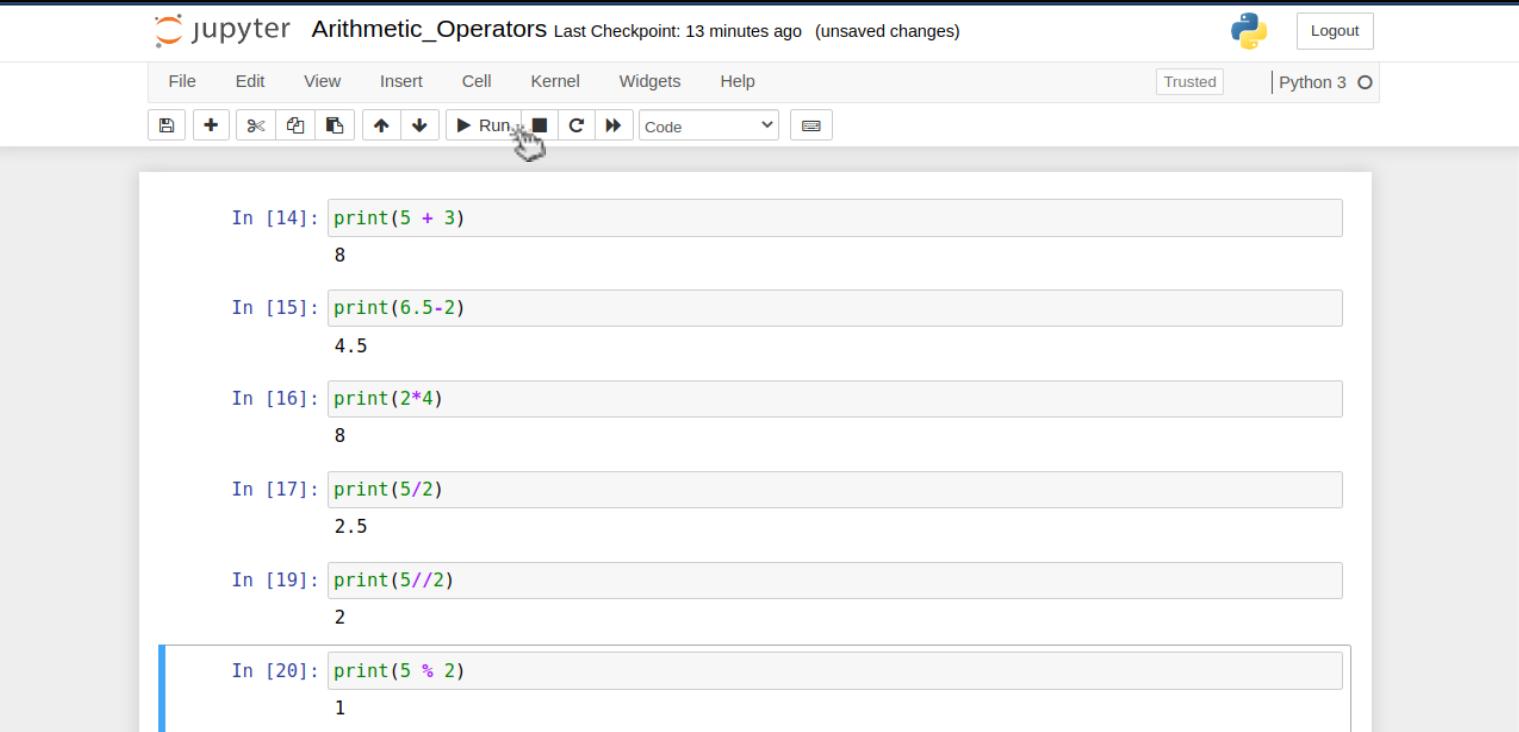
There are two types of division in Python

- Float Division: Use (/) for float division. If the result is an integer, it is converted to a float by adding .0
- Integer Division: Use (//) for integer division. If the result of is a float, it is converted to an integer by truncating the decimal part

Modulo

- Modulo operator returns the remainder of a division
- Use (%) operator modulo

Arithmetic Operators (3/3)



The screenshot shows a Jupyter Notebook interface with the title "Arithmetic Operators (3/3)". The notebook has a "Trusted" status and is using "Python 3". The toolbar includes buttons for file operations, cell creation, running cells, and help. Below the toolbar, several code cells are displayed, each showing a print statement and its output:

- In [14]: `print(5 + 3)`
8
- In [15]: `print(6.5-2)`
4.5
- In [16]: `print(2*4)`
8
- In [17]: `print(5/2)`
2.5
- In [19]: `print(5//2)`
2
- In [20]: `print(5 % 2)`
1

Quiz Time

1. What is the correct output of the float division $4/2$?

- a) 2
- b) 2.0
- c) 0
- d) 1

2. What is the correct output of the integer division $4//2$?

- a) 2
- b) 2.0
- c) 0
- d) 1

3. What is the correct output of the expression $9\%2$?

- a) 0
- b) 1
- c) 4.5
- d) 4

Quiz Time

1. What is the correct output of the float division $4/2$?

- a) 2
- b) 2.0**
- c) 0
- d) 1

2. What is the correct output of the integer division $4//2$?

- a) 2**
- b) 2.0
- c) 0
- d) 1

3. What is the correct output of the expression $9\%2$?

- a) 0
- b) 1**
- c) 4.5
- d) 4

Comparison Operators

- Comparison operators are used to compare two values with each other.
- The result of a comparison operator is either True or False.
- We have the following comparison operators
 - **Greater than (>)**
 - returns True if the value on the left is bigger than that on the right
 - **Less than (<)**
 - returns True if the value on the left is smaller than that on the right
 - **Greater than or equal to (>=)**
 - returns True if the value on the left is either bigger than or equal to the value on the right
 - **Less than or equal to (<=)**
 - returns True if the value on the left is either smaller than or equal to the value on the right
 - **Not equal to (!=)**
 - returns True if the value on the left is not equal to the value on the right
 - **Is equal to (==)**
 - returns True if the value on the left is equal to the value on the right

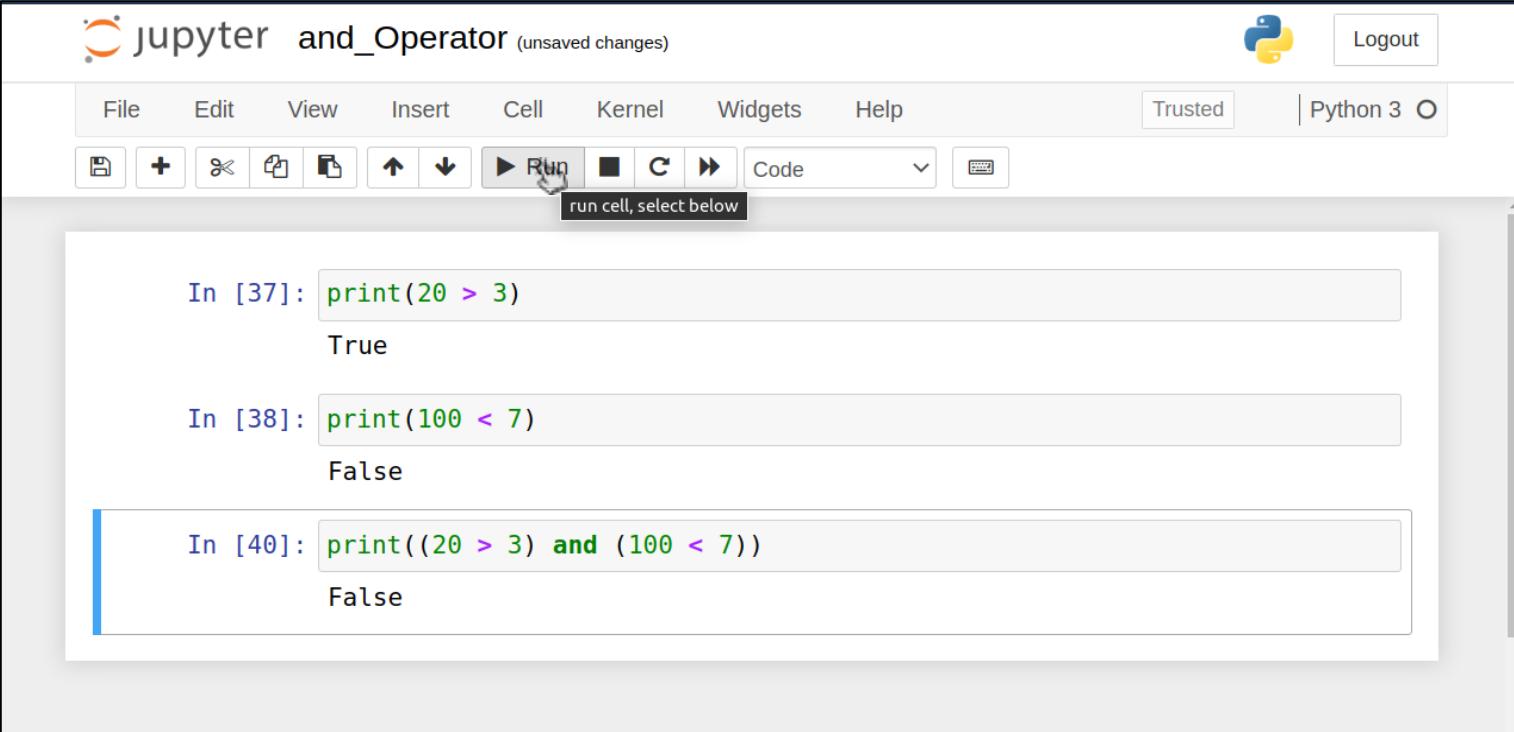
Logical Operators

- We can use logical operators to combine two or more conditions.
- Python has three logical operators
 - And operator (`and`)
 - Or operator (`or`)
 - Not operator (`not`)
- and/or operators are binary as they take two operands.
- Not is a unary operator as it takes only one operand.

Logical Operators

and operator

- and returns a True only if both the operands are True.
- In python, we use the keyword ‘and’ to perform and operation.



The screenshot shows a Jupyter Notebook interface with the title "Jupyter and_Operator (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The main area displays three code cells:

- In [37]: `print(20 > 3)`
True
- In [38]: `print(100 < 7)`
False
- In [40]: `print((20 > 3) and (100 < 7))`
False

Logical Operators

or operator

- or returns a True if at least one of the operands is True.
- In python, we use the keyword 'or' to perform or operation.

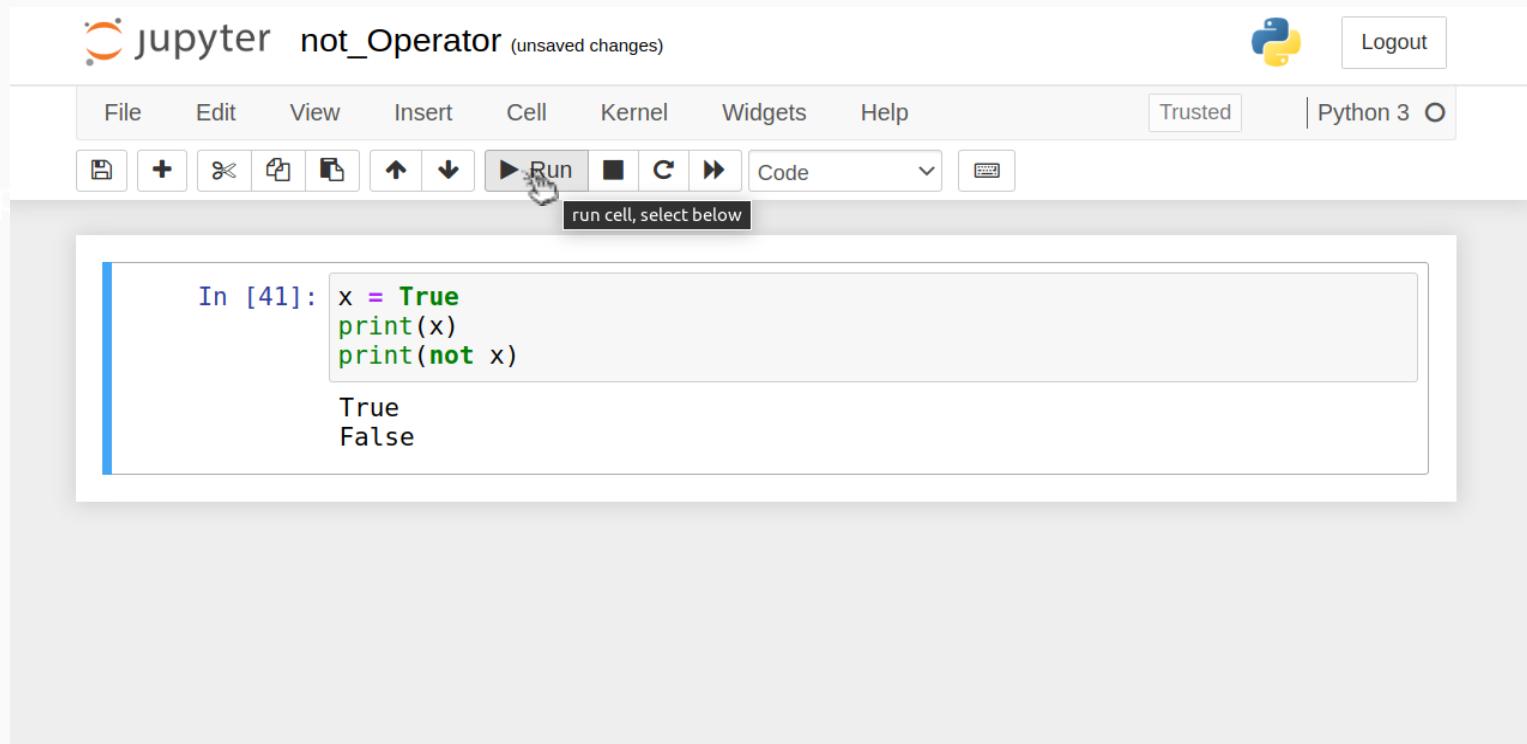
The screenshot shows a Jupyter Notebook interface with the title "Jupyter or_Operator (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. The toolbar also features standard notebook operations like Run, Cell, and Kernel. The code cells show the following executions:

- In [37]: `print(20 > 3)`
True
- In [38]: `print(100 < 7)`
False
- In [39]: `print((20 > 3) or (100 < 7))`
True

Logical Operators

not operator

- not is a unary operator.
- not changes True to False and vice versa.
- In python, we use the keyword 'not' to perform not operation.



The screenshot shows a Jupyter Notebook interface with the title "jupyter not_Operator (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. Below the toolbar is a toolbar with icons for file operations like Open, Save, and Run. A tooltip "run cell, select below" points to the Run button. The code cell contains the following Python code:

```
In [41]: x = True  
print(x)  
print(not x)
```

The output of the code is:

```
True  
False
```

Quiz Time

1. What is the correct output of the expression; $(4 > 3)$ and $(5 > 3)$?
 - a) True
 - b) False

2. What is the correct output of the expression; $(7 < 2)$ or $(8 < 3)$?
 - a) True
 - b) False

3. What is the correct output of the expression; not True?
 - a) True
 - b) False

Quiz Time

1. What is the correct output of the expression; $(4 > 3)$ and $(5 > 3)$?

- a) True
- b) False

2. What is the correct output of the expression; $(7 < 2)$ or $(8 < 3)$?

- a) True
- b) False

3. What is the correct output of the expression; not True?

- a) True
- b) False

Conditional Statements (1/4)

- Conditional statements are used to make decisions.
- In Python, we make decisions using if-else and elif statements.
- elif and else are both optional, however they must follow an if statement.
- We can have as many elif statements as we want. But there is only one if and one else statement.

Conditional Statements (2/4)

Control Flow

- Here is the general syntax of conditional statements

if condition:

 block of statements to execute

elif condition:

 block of statements to execute

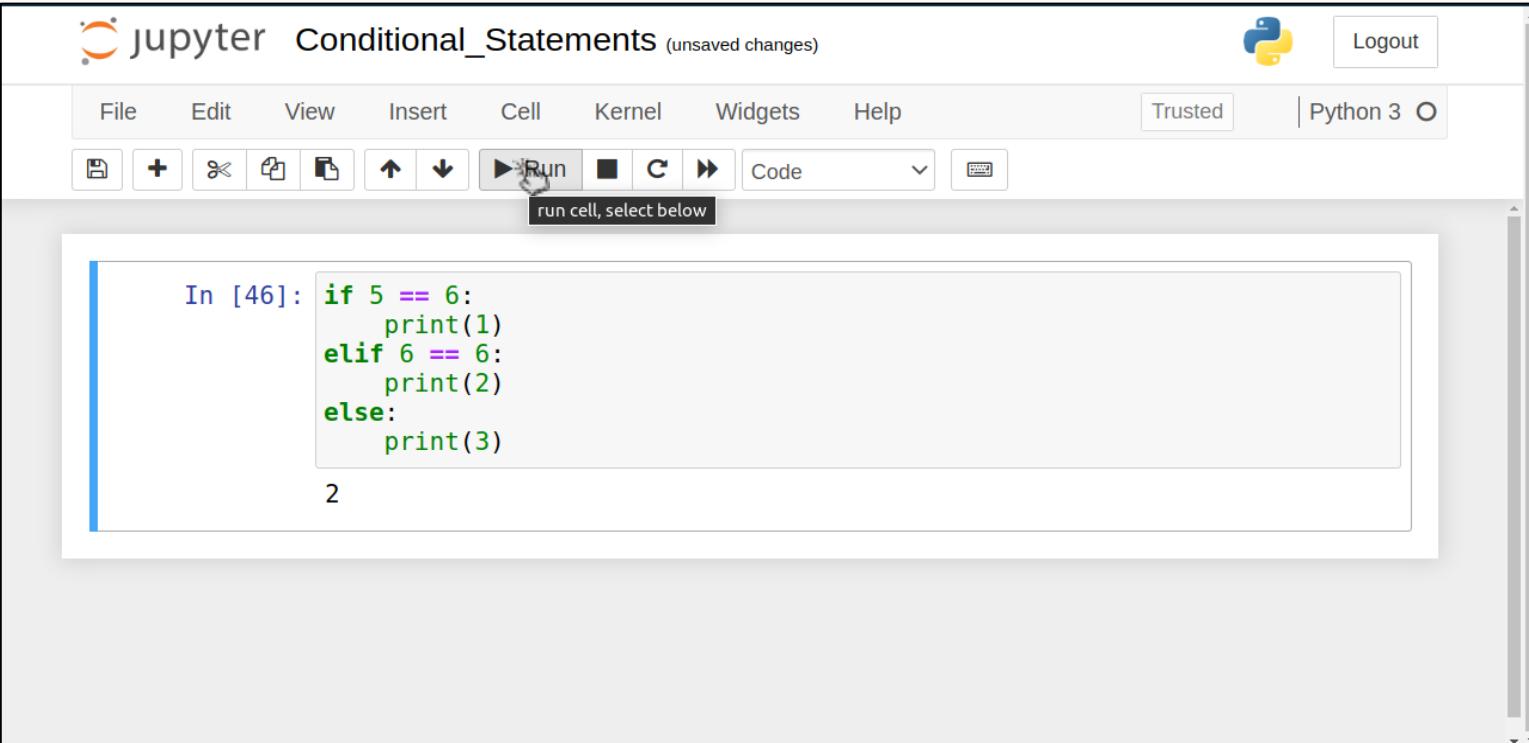
else:

 block of statements to execute

- Conditions are evaluated in a top-down manner.

Conditional Statements (3/4)

Example in Jupyter



The screenshot shows a Jupyter Notebook interface with the title "jupyter Conditional_Statements (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar is a toolbar with icons for file operations, cell selection, and execution. A tooltip "run cell, select below" is visible over the execution icon. The main area displays a code cell labeled "In [46]:" containing the following Python code:

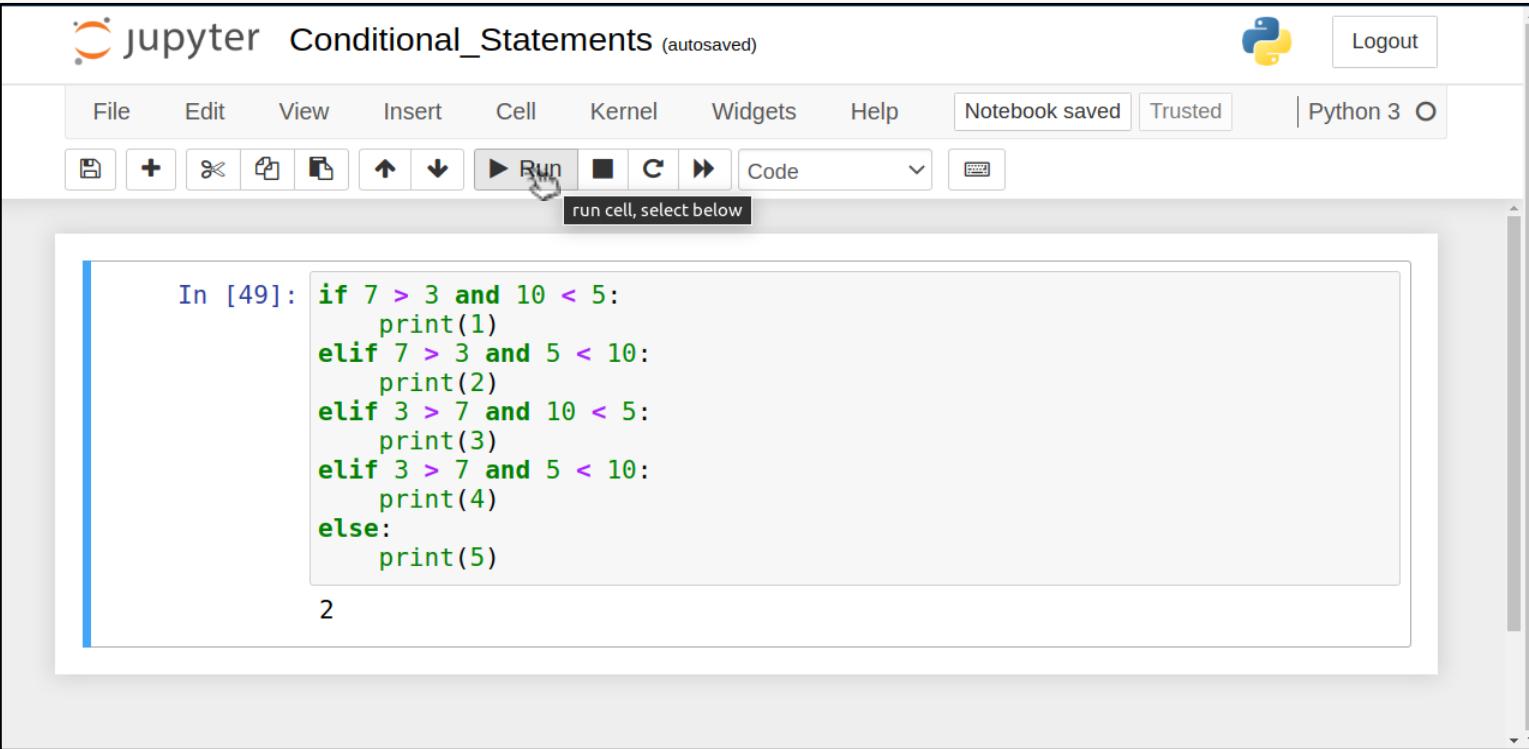
```
In [46]: if 5 == 6:  
    print(1)  
elif 6 == 6:  
    print(2)  
else:  
    print(3)
```

The output of the code is "2", displayed below the code cell.

Conditional Statements (4/4)

Combining Conditions

- We can combine two or more conditions together using logical operators.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Conditional_Statements (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and buttons for Run, Notebook saved, Trusted, and Python 3. A tooltip "run cell, select below" is visible over the Run button. The code cell contains the following Python code:

```
In [49]: if 7 > 3 and 10 < 5:  
    print(1)  
elif 7 > 3 and 5 < 10:  
    print(2)  
elif 3 > 7 and 10 < 5:  
    print(3)  
elif 3 > 7 and 5 < 10:  
    print(4)  
else:  
    print(5)
```

The output cell below the code cell displays the number 2, indicating that the second condition was met.

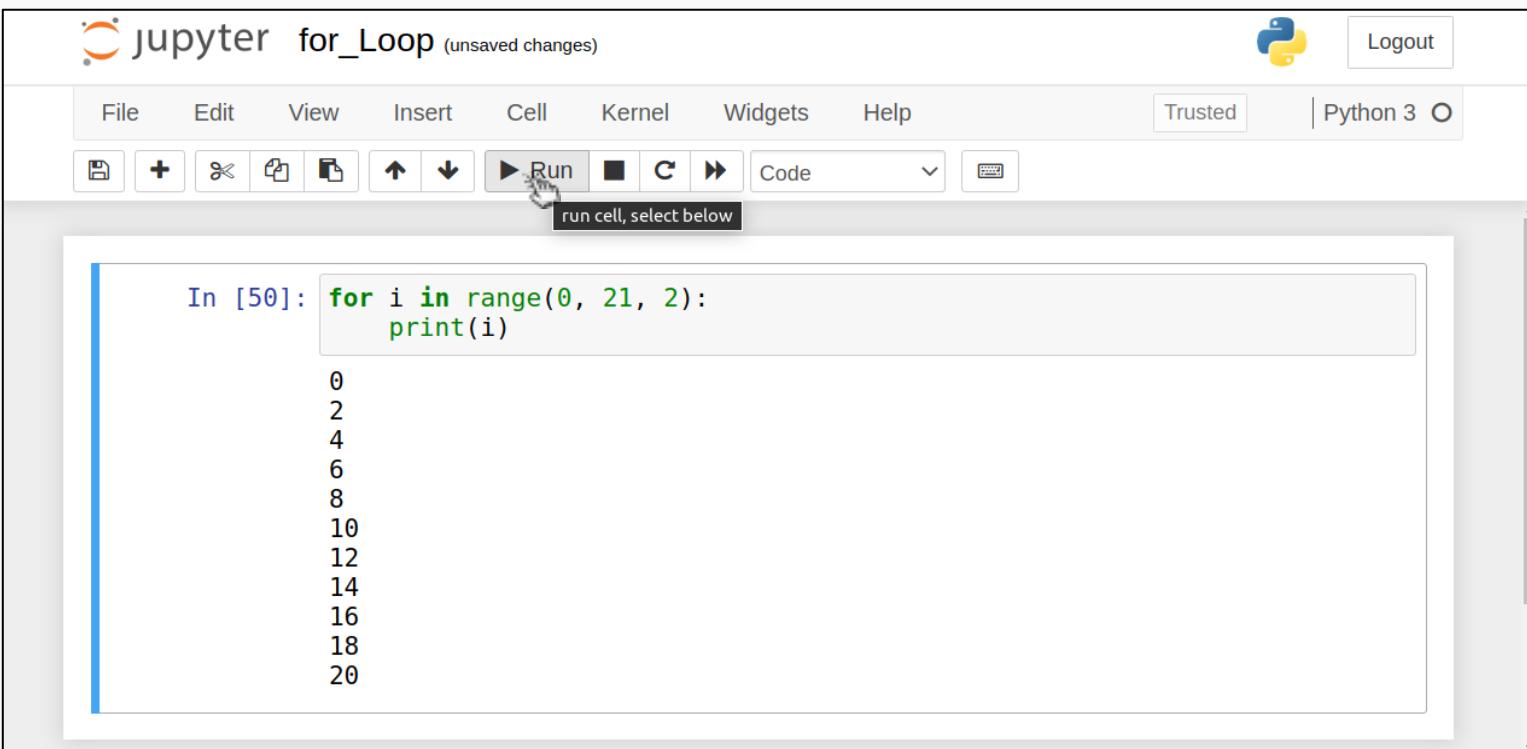
Loops

- Loops are used for repeating a set of statements.
- There are two types of loops in Python
 - For loops
 - While loops

<https://unibo.zoom.us/j/83422537997?pwd=Z3VENmI0QTV2Q2xJWDIPQzhrVU5PZz09>

for Loop (1/2)

- for loop takes a range() function.
- range() takes three arguments; start, stop, step.
- loop runs from start to stop - 1 with an increment equal to step after every iteration.



The screenshot shows a Jupyter Notebook interface with the title "Jupyter for_Loop (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The "Run" button in the toolbar is highlighted with a cursor. Below the toolbar, a code cell labeled "In [50]" contains the following Python code:

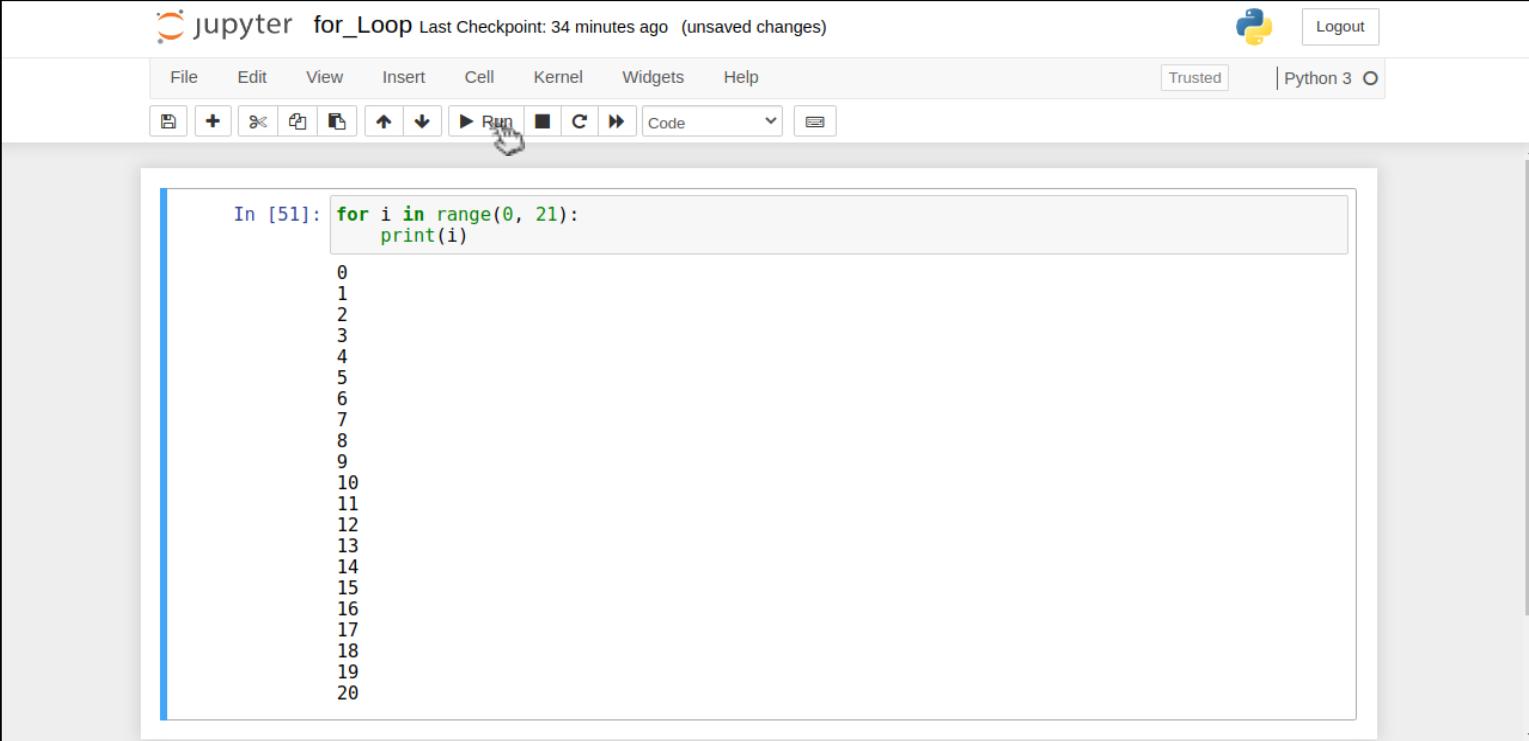
```
In [50]: for i in range(0, 21, 2):
    print(i)
```

The output of the code is displayed in the cell, showing the even numbers from 0 to 20:

```
0
2
4
6
8
10
12
14
16
18
20
```

for Loop (2/2)

- If we omit the step value, it is set to 1 by default.



The screenshot shows a Jupyter Notebook interface with the title "jupyter for_Loop". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. A "Run" button is highlighted with a cursor. The code cell "In [51]" contains the following Python code:

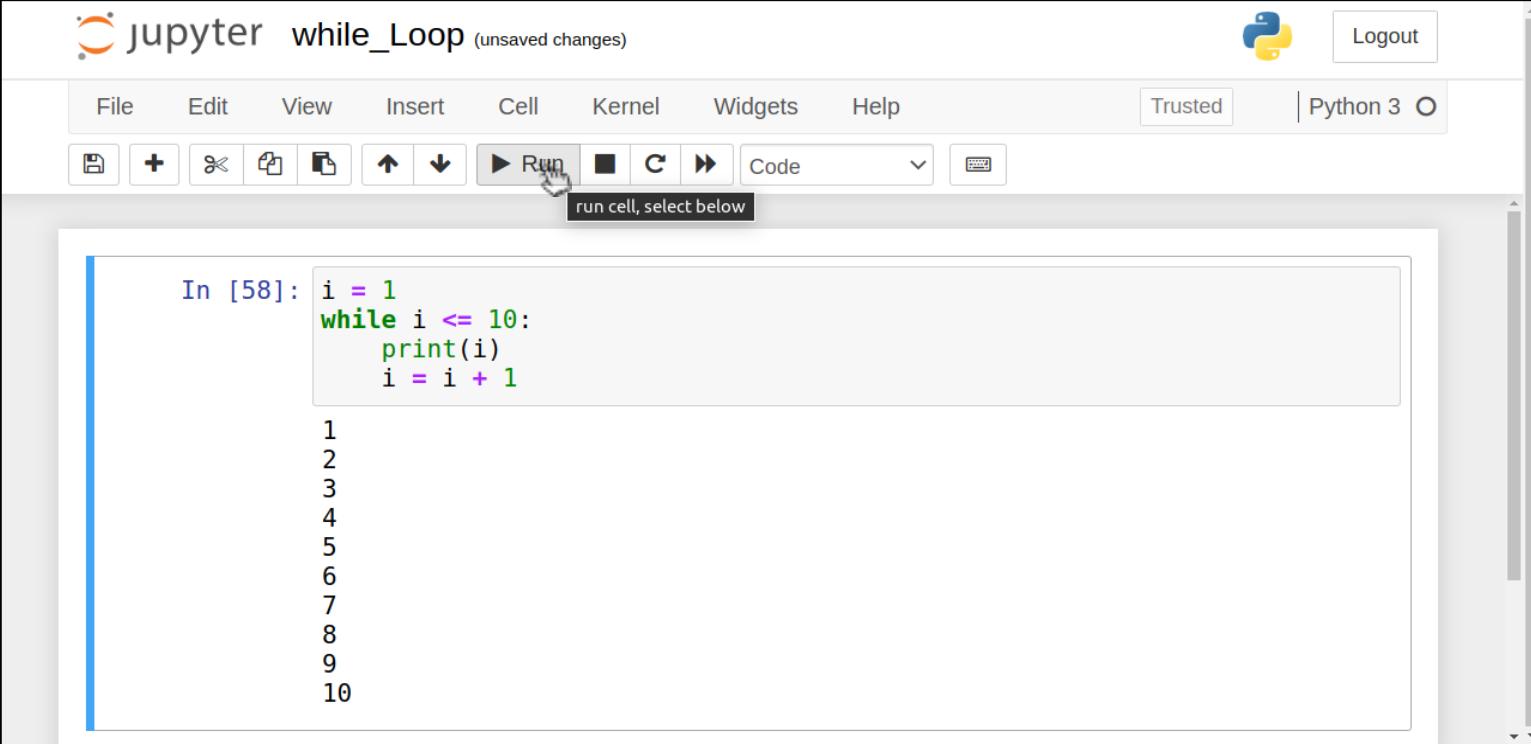
```
In [51]: for i in range(0, 21):
    print(i)
```

The output of the code is displayed below the cell, showing integers from 0 to 20, each on a new line:

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

while Loop (1/2)

- while loop takes a condition and runs as long as the condition is True.



The screenshot shows a Jupyter Notebook interface with the title "jupyter while_Loop (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and a Logout button. The toolbar buttons include icons for file operations, cell selection, and execution. A tooltip "run cell, select below" is visible over the Run button. The code cell contains the following Python code:

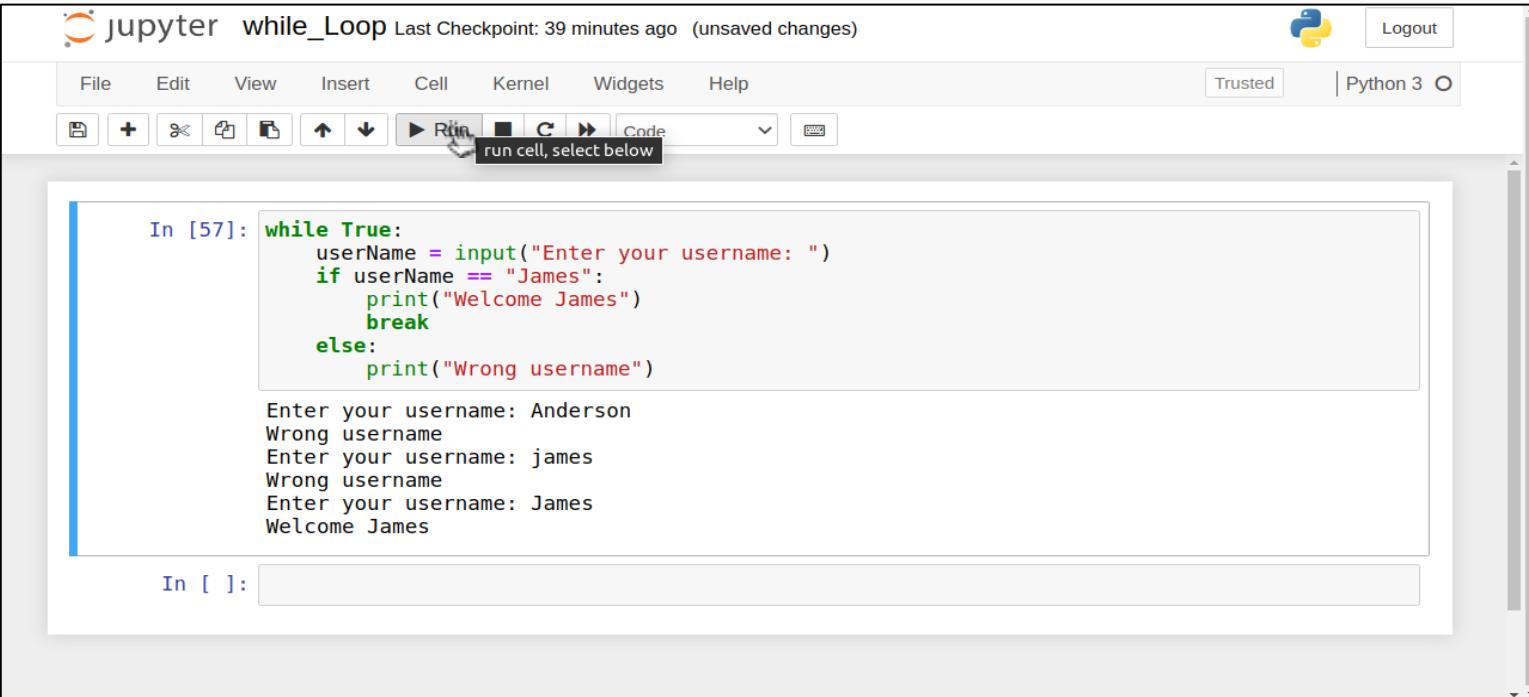
```
In [58]: i = 1
while i <= 10:
    print(i)
    i = i + 1
```

The output cell displays the numbers 1 through 10, each on a new line, indicating the execution of the loop.

while Loop (2/2)

Infinite Loop

- We can create an infinite loop using a while loop.
- We give a terminating condition inside the infinite loop and use the keyword ‘break’ to break out of the infinite loop.



The screenshot shows a Jupyter Notebook interface with the title "jupyter while_Loop". The notebook has a "Trusted" status and is using "Python 3". The toolbar includes buttons for File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Run, and Cell Type (Code). A tooltip "run cell, select below" points to the Run button. The code cell In [57] contains the following Python code:

```
In [57]: while True:  
    userName = input("Enter your username: ")  
    if userName == "James":  
        print("Welcome James")  
        break  
    else:  
        print("Wrong username")
```

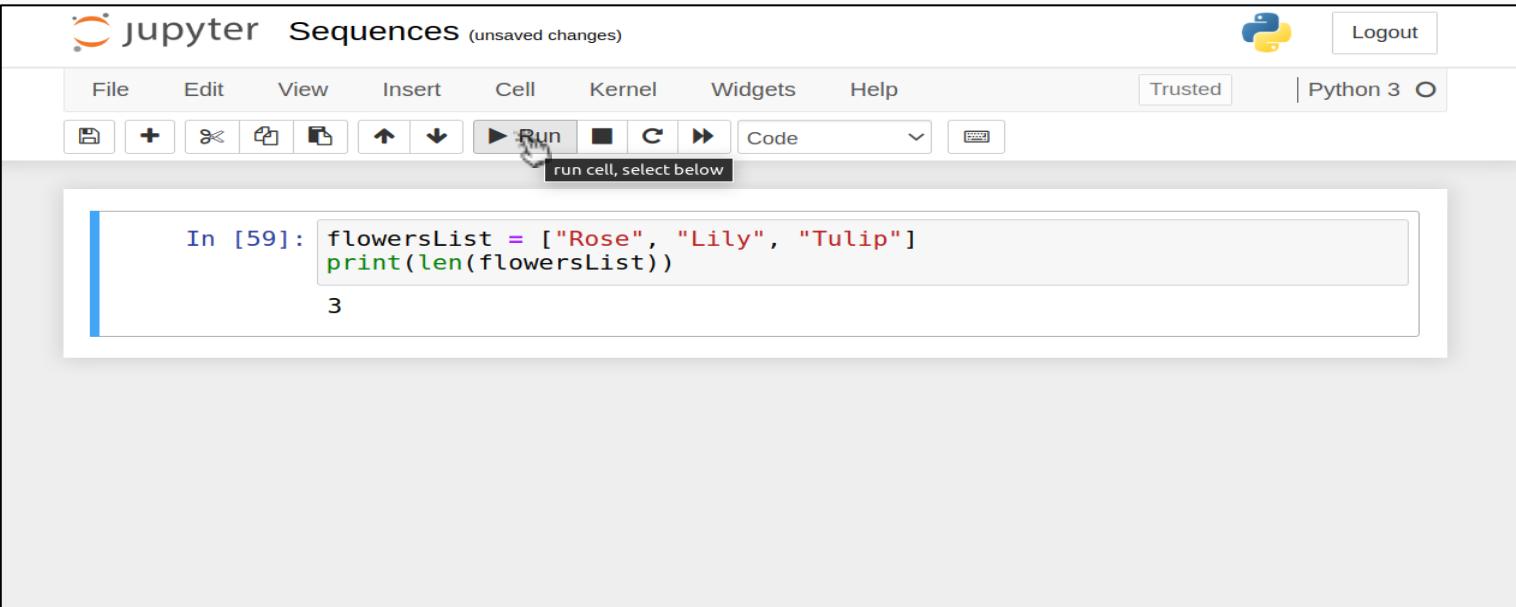
The output cell shows the user's input and the program's response:

```
Enter your username: Anderson  
Wrong username  
Enter your username: james  
Wrong username  
Enter your username: James  
Welcome James
```

The bottom cell is labeled "In []:".

Sequences

- Python has following built-in sequences (non-primitive data types)
 - Lists
 - Dictionaries
 - Tuples
- We can use the len() function to find the total number of elements in any sequence.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The toolbar buttons include file operations, cell selection, and run. A tooltip says "run cell, select below". The code cell contains:

```
In [59]: flowersList = ["Rose", "Lily", "Tulip"]
print(len(flowersList))
```

The output cell shows the result:

```
3
```

Lists (1/5)

- Lists are used to store multiple values.
- A list can have values of different data types.
- Lists are ***mutable*** i.e., they can be modified.

<https://unibo.zoom.us/j/83422537997?pwd=Z3VENmI0QTV2Q2xJWDIPQzhrVU5PZz09>

Lists (2/5)

List Indexing

- We can get the element at a particular position in the list using list indexing.
 - Remember, indices start at 0 and go until $\text{len}(\text{list}) - 1$.

Rose	Lily	Tulip
0	1	2

- Consider the list above that contains names of three flowers. Hence, it's length is 3.
- If we call this list flowersList, then;
 - `flowersList[0]` will give us Rose
 - `flowersList[1]` will give us Lily
 - `flowersList[2]` will give us Tulip

Lists (3/5)

List Slicing

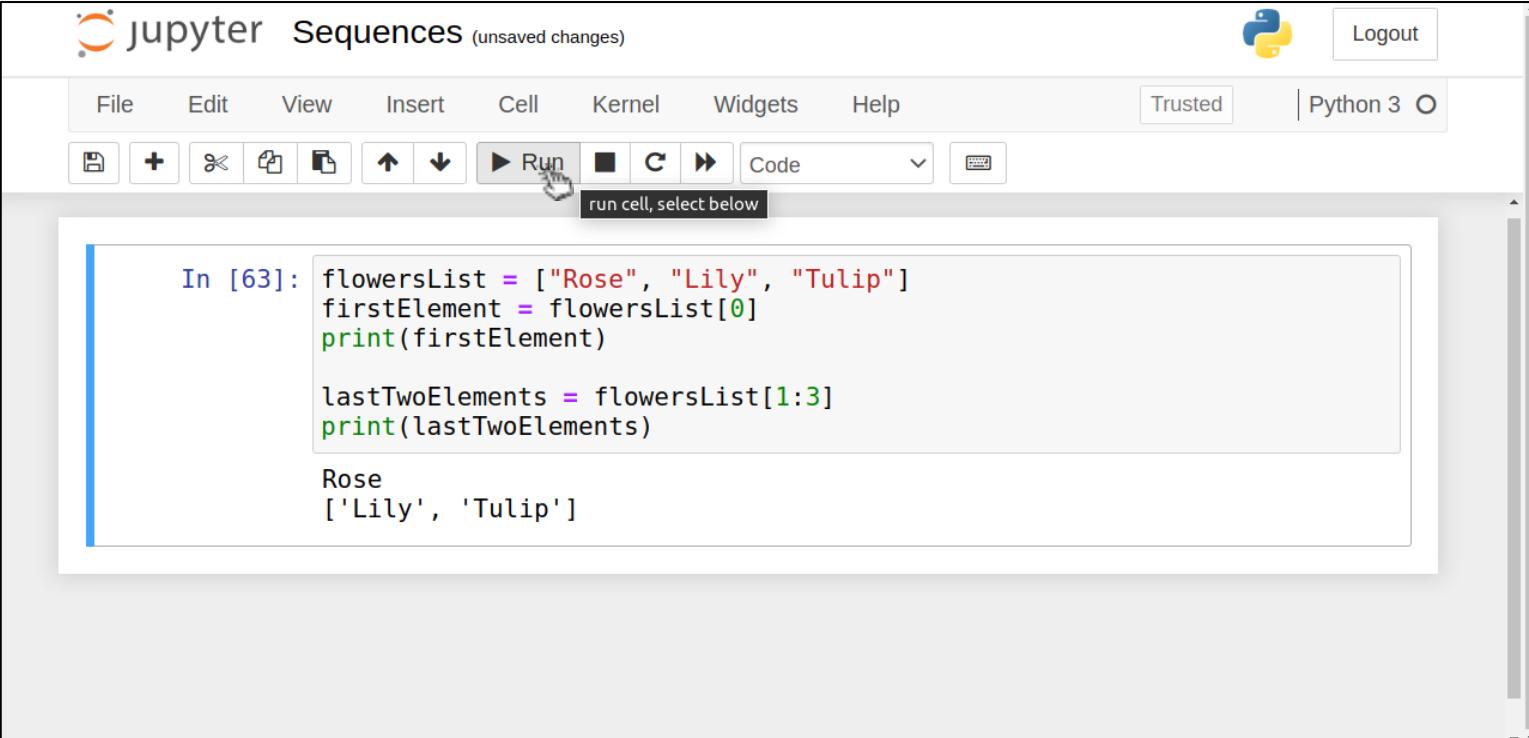
- We can also slice a list.
- For slicing a list, we give a starting-index (inclusive) and an end-index (exclusive).

Rose	Lily	Tulip
0	1	2

- If we call this list flowersList then;
 - flowersList[0:2] will give us [Rose, Lily]
 - flowersList[0:3] will give us [Rose, Lily, Tulip]
 - flowersList[1:2] will give us [Lily]

Lists (4/5)

Example in Jupyter



The screenshot shows a Jupyter Notebook interface titled "Sequences" with "unsaved changes". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and various cell management icons. A tooltip "run cell, select below" points to the "Run" button. The code cell contains the following Python code:

```
In [63]: flowersList = ["Rose", "Lily", "Tulip"]
firstElement = flowersList[0]
print(firstElement)

lastTwoElements = flowersList[1:3]
print(lastTwoElements)
```

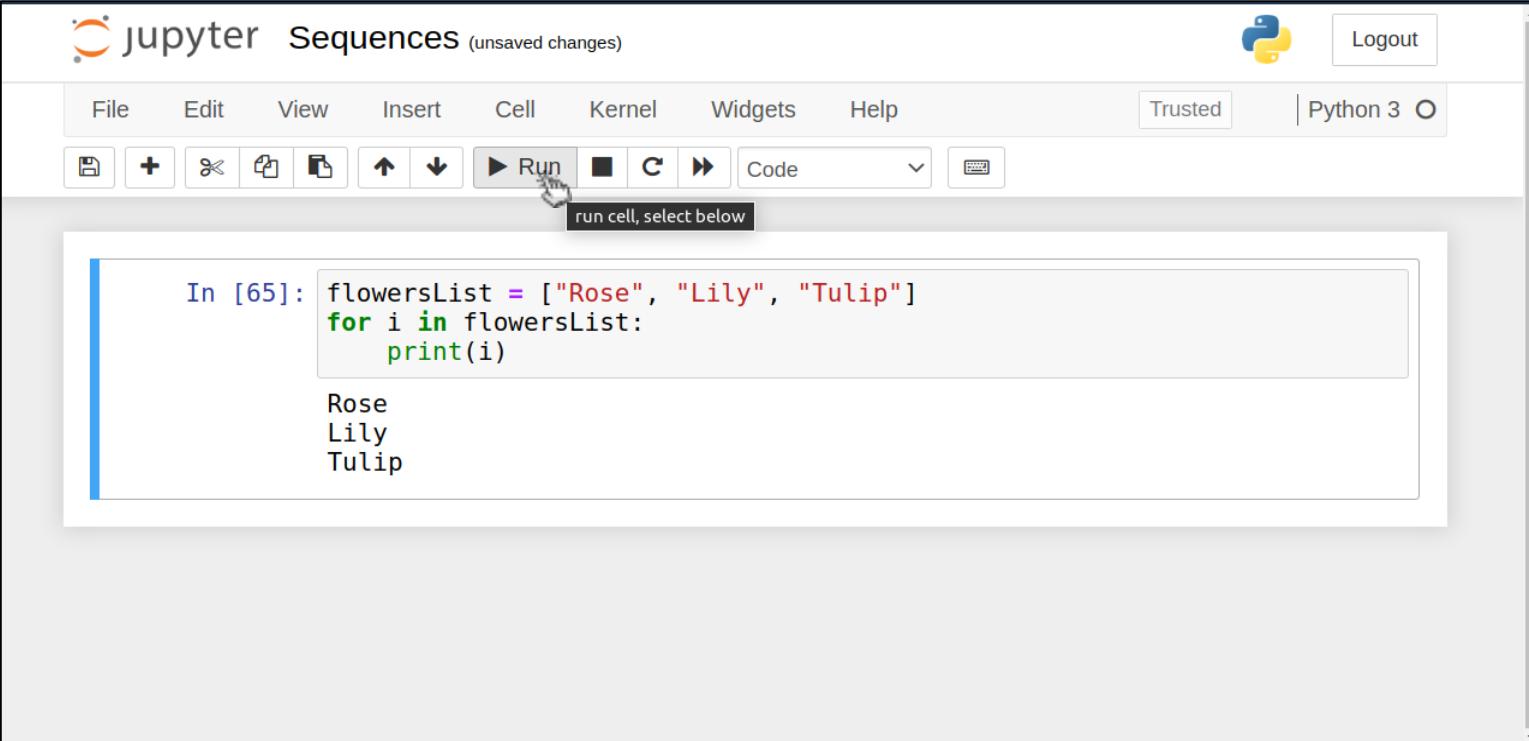
The output cell displays the results:

```
Rose
['Lily', 'Tulip']
```

Lists (5/5)

Iterating Over List

- We can loop over a list using a modified form of for loop.



The screenshot shows a Jupyter Notebook interface titled "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. Below the toolbar is a toolbar with icons for file operations and a "Run" button, which has a tooltip "run cell, select below". A code cell labeled "In [65]" contains the following Python code:

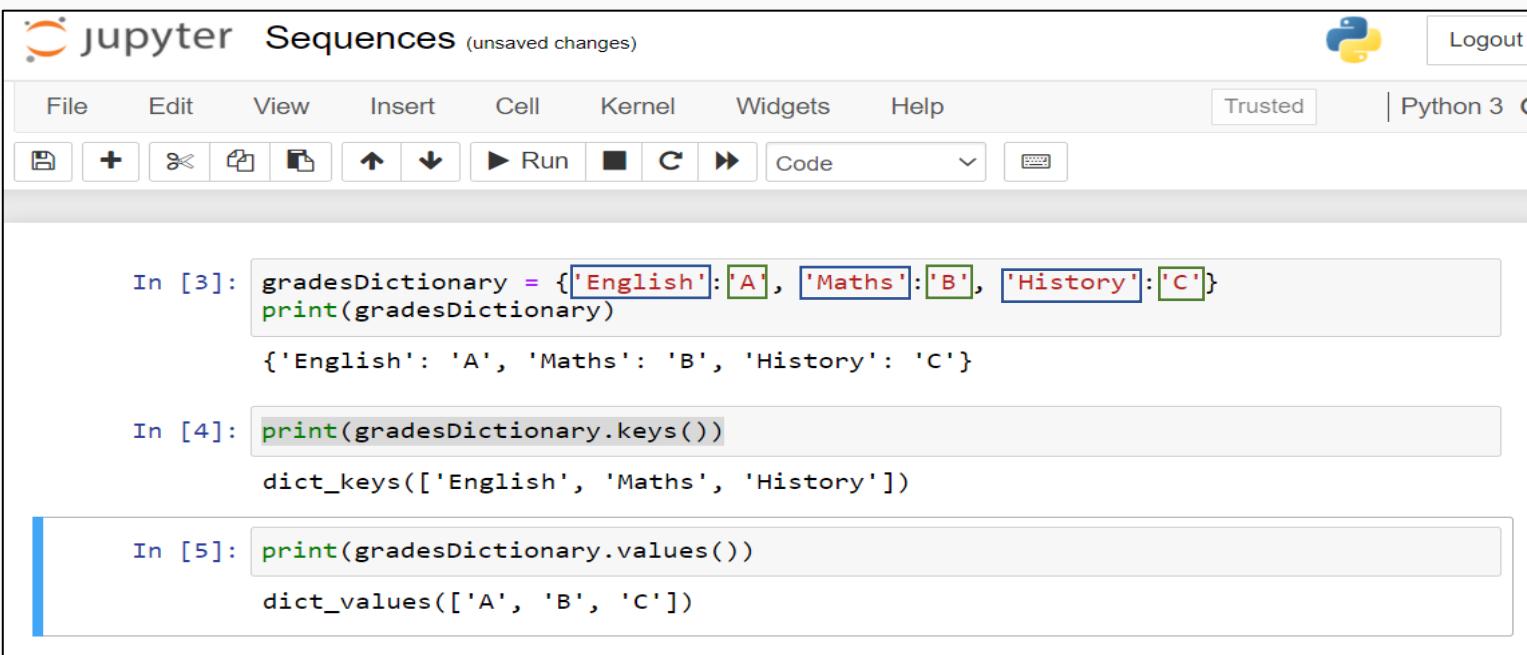
```
flowersList = ["Rose", "Lily", "Tulip"]
for i in flowersList:
    print(i)
```

The output of the code is displayed in the cell:

```
Rose
Lily
Tulip
```

Dictionaries (1/3)

- Dictionaries are used to store key, value pairs.
- A key and a value are separated by a colon (:).
- Each key, value pair is separated by a comma (,).
- Dictionaries are **mutable** i.e., they can be modified.
- We can use .keys() function to get all the keys and .values() function to get all the values in the dictionary.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Sequences (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. The toolbar below has icons for file operations, cell selection, and run. The code cells are as follows:

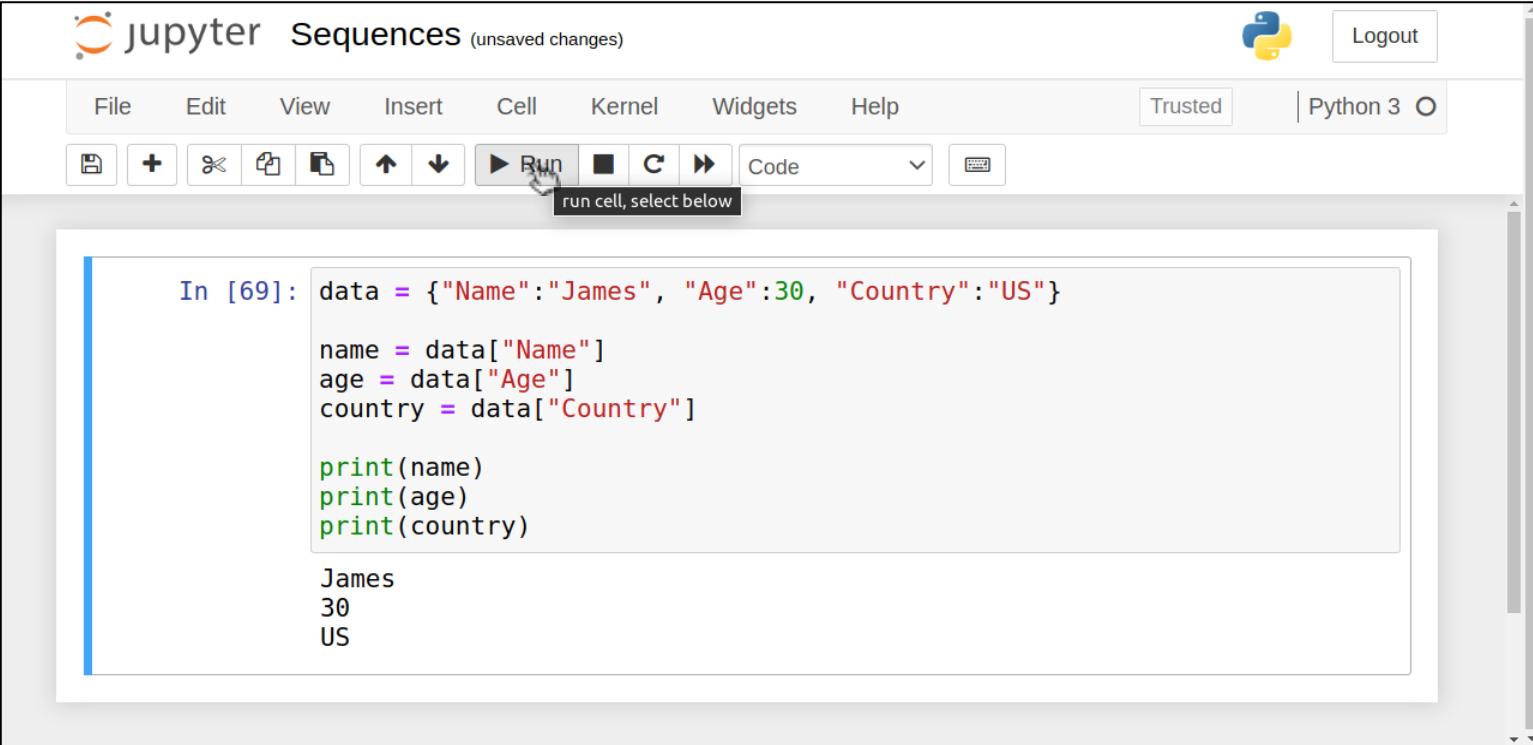
```
In [3]: gradesDictionary = {'English': 'A', 'Maths': 'B', 'History': 'C'}  
print(gradesDictionary)  
  
{'English': 'A', 'Maths': 'B', 'History': 'C'}  
  
In [4]: print(gradesDictionary.keys())  
  
dict_keys(['English', 'Maths', 'History'])  
  
In [5]: print(gradesDictionary.values())  
  
dict_values(['A', 'B', 'C'])
```

● Keys

● Values

Dictionaries (2/3)

- We can get a value associated with a key by passing the key.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The toolbar buttons include a file icon, a plus sign, a delete icon, a copy icon, a run button, a cell type icon, a code dropdown, and a keyboard icon. A tooltip "run cell, select below" is shown over the run button. The code cell contains the following Python code:

```
In [69]: data = {"Name": "James", "Age": 30, "Country": "US"}  
name = data["Name"]  
age = data["Age"]  
country = data["Country"]  
  
print(name)  
print(age)  
print(country)
```

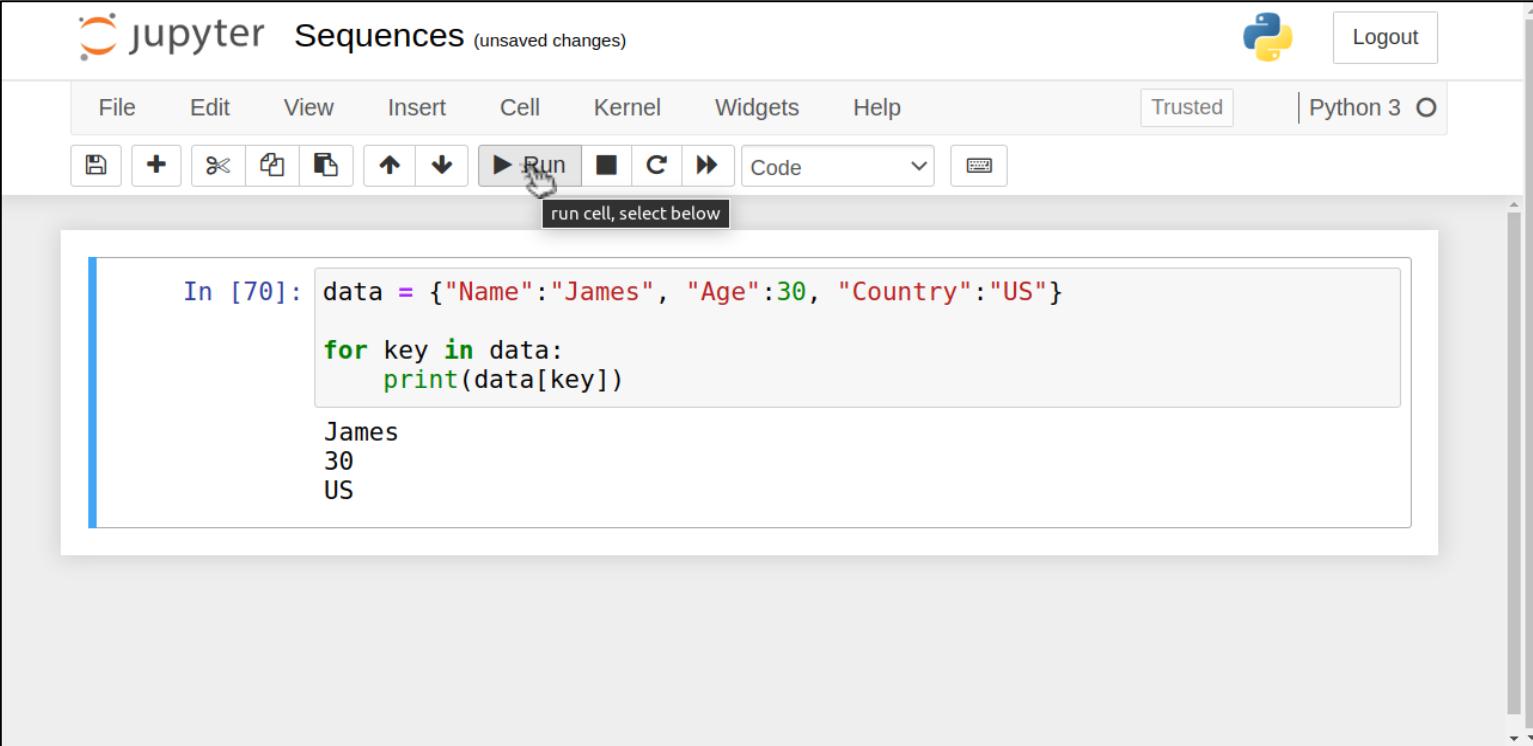
The output cell shows the results of the print statements:

```
James  
30  
US
```

Dictionaries (3/3)

Iterating Over Dictionary

- We can get all the key, value pairs in a Dictionary using a modified form of for loop.



The screenshot shows a Jupyter Notebook interface with the title "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. Below the toolbar is a toolbar with icons for file operations and cell execution. A tooltip "run cell, select below" is visible over the execution button. The code cell contains the following Python code:

```
In [70]: data = {"Name": "James", "Age": 30, "Country": "US"}  
for key in data:  
    print(data[key])
```

The output cell displays the results of the execution:

```
James  
30  
US
```

Tuples (1/5)

- Tuples are also used to store multiple values.
- Tuples can have values of different data types.
- Tuples are ***immutable*** i.e., they can not be modified.

<https://unibo.zoom.us/j/83422537997?pwd=Z3VENmI0QTV2Q2xJWDIPQzhrVU5PZz09>

Tuples (2/5)

Tuple Indexing

- We can index a tuple just as we can index a list.

Rose	Lily	Tulip
0	1	2

- Consider the tuple above that contains names of three flowers. Hence, it's length is 3.
- If we call this tuple flowersTuple, then;
 - flowersTuple[0] will give us Rose
 - flowersTuple[1] will give us Lily
 - flowersTuple[2] will give us Tulip

Tuples (3/5)

Tuple Slicing

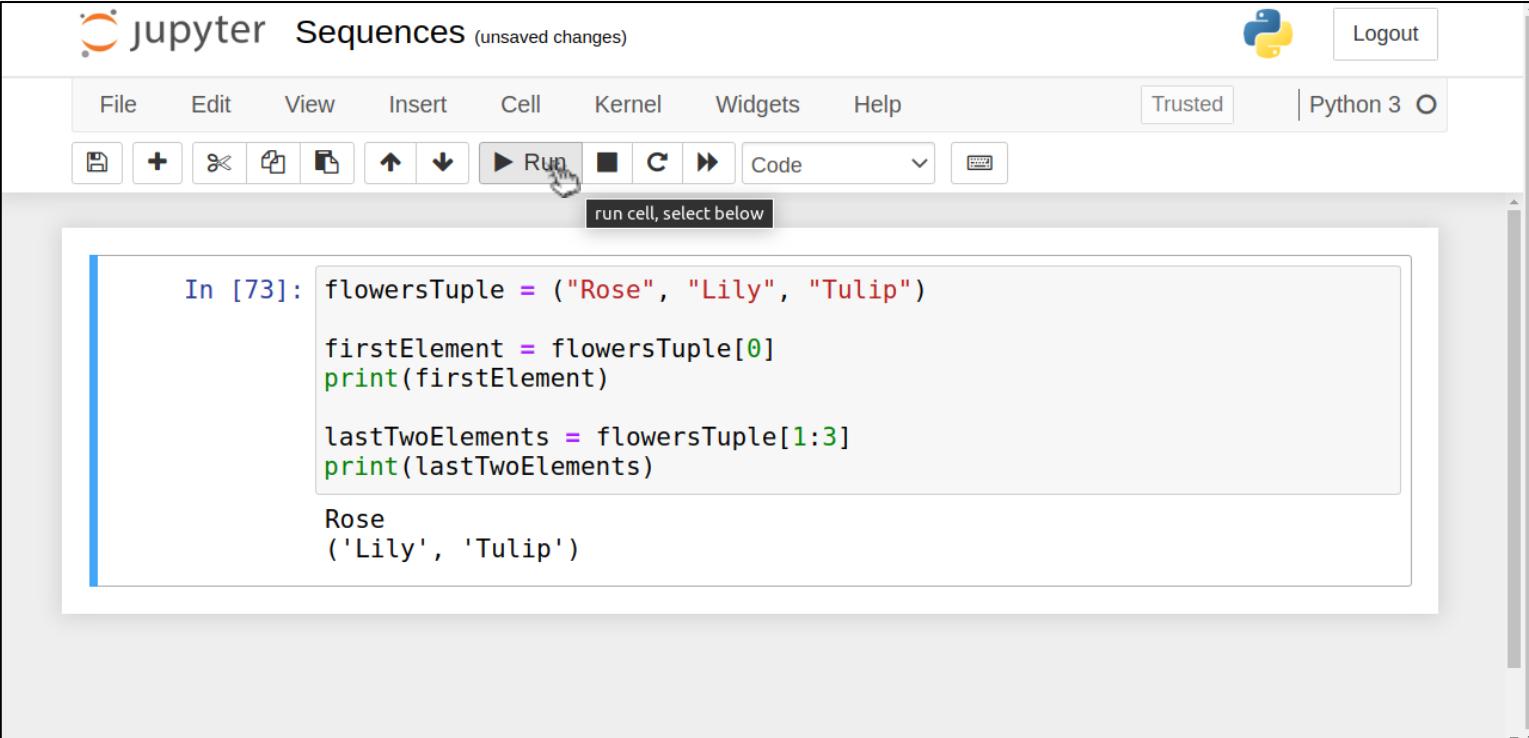
- Tuple slicing works the same way as list slicing.

Rose	Lily	Tulip
0	1	2

- If we call this tuple flowersTuple then;
 - flowersTuple[0:2] will give us (Rose, Lily)
 - flowersTuple[0:3] will give us (Rose, Lily, Tulip)
 - flowersTuple[1:2] will give us (Lily)

Tuples (4/5)

Example in Jupyter



The screenshot shows a Jupyter Notebook interface titled "Sequences" with "unsaved changes". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and various cell management icons. A tooltip "run cell, select below" points to the "Run" button. The code cell contains the following Python code:

```
In [73]: flowersTuple = ("Rose", "Lily", "Tulip")
firstElement = flowersTuple[0]
print(firstElement)

lastTwoElements = flowersTuple[1:3]
print(lastTwoElements)
```

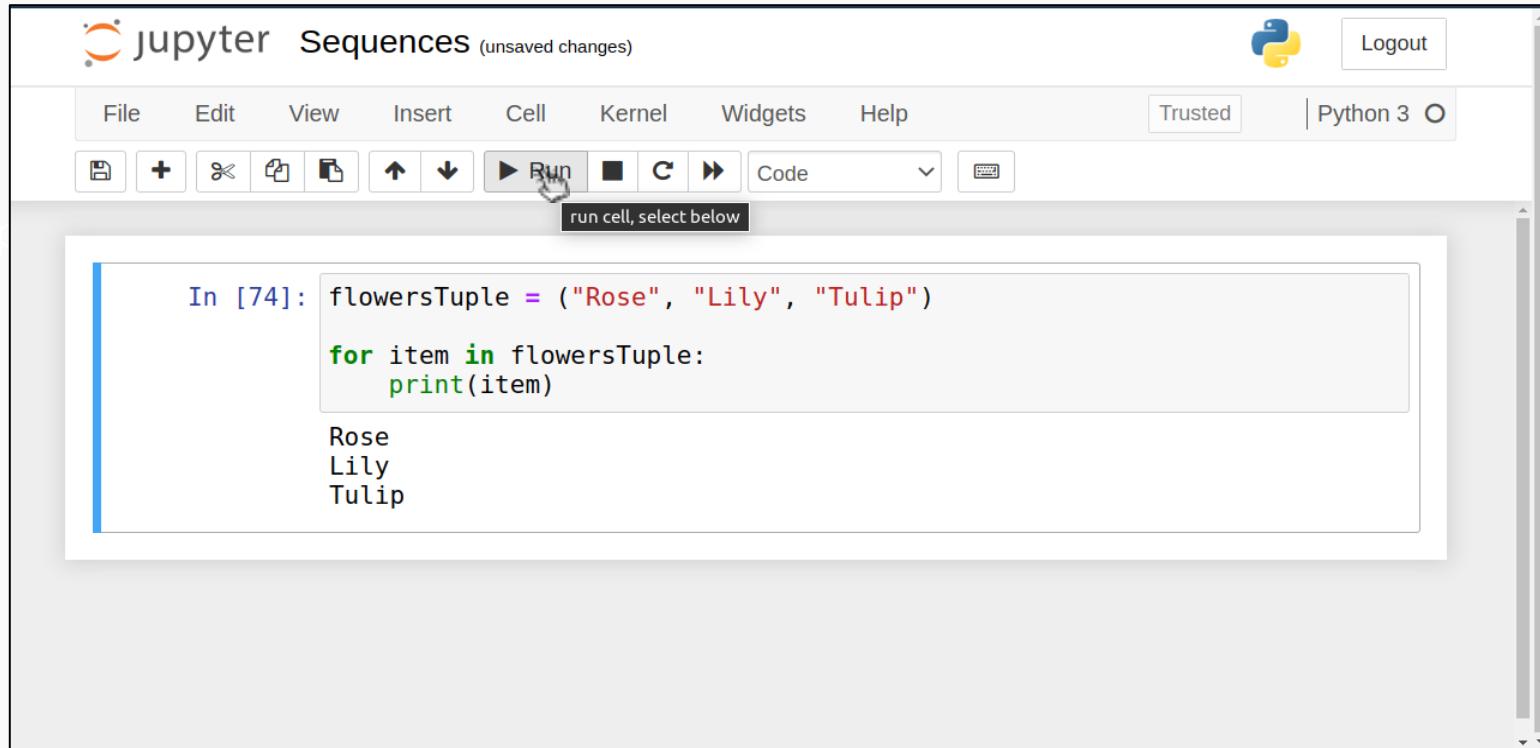
The output cell displays the results of the code execution:

```
Rose
('Lily', 'Tulip')
```

Tuples (5/5)

Iterating Over Tuple

- We can loop over a tuple using a modified form of for loop.



The screenshot shows a Jupyter Notebook interface titled "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The toolbar buttons include file operations, cell selection, and run. A tooltip says "run cell, select below". The code cell contains:

```
In [74]: flowersTuple = ("Rose", "Lily", "Tulip")
for item in flowersTuple:
    print(item)
```

The output cell shows the results of the execution:

```
Rose
Lily
Tulip
```

Quiz Time

1. Which of the following is the immutable sequence?

- a) List
- b) Dictionary
- c) Tuple

2. Consider a list `x = ["red", 1, True]`. What will `x[1]` give?

- a) "red"
- b) 1
- c) True

3. Consider a dictionary `y = {"name": "James", "age": 1}`. What is "age" in this dictionary?

- a) Key
- b) Value
- c) None

Quiz Time

1. Which of the following is the immutable sequence?

- a) List
- b) Dictionary
- c) Tuple

2. Consider a list `x = ["red", 1, True]`. What will `x[1]` give?

- a) "red"
- b) 1
- c) True

3. Consider a dictionary `y = {"name": "James", "age": 1}`. What is "age" in this dictionary?

- a) Key
- b) Value
- c) None

Built-in Functions

- Python has a lot of built-in functions e.g.,
 - `len()`, which returns the length of a sequence
 - `abs()`, which returns the absolute value
 - `max()`, which returns the maximum value in a sequence
- You can find a detailed list of built-in functions in Python at the following link:
<https://docs.python.org/3/library/functions.html>

<https://unibo.zoom.us/j/83422537997?pwd=Z3VENmI0QTV2O2xJWDIPQzhrVU5PZz09>

User-defined Functions (1/2)

- Apart from built-in functions, users can create their own functions as well.
- We use the 'def' keyword to define a function.
- A function should be defined before it is called.
- We do not need to declare the return type while defining a function.

The screenshot shows a Jupyter Notebook interface with the title "jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The "Cell" button is highlighted with a mouse cursor. Below the toolbar, there are buttons for file operations like Save, New, and Delete, along with Run, Cell, Kernel, and Help buttons. A tooltip "run cell, select below" is visible over the Run button. The notebook contains two code cells:

```
In [75]: def AddNumbers(x, y):
    sum = x + y
    return sum
```

```
In [76]: result = AddNumbers(5, 7)
print(result)
```

The output of the second cell is "12".

User-defined Functions (2/2)

- We can also return more than one value from a function.
- In this case we are returning 2 values from the functions, hence we need to store them in 2 variables, namely num1 and num2.

The screenshot shows a Jupyter Notebook interface with the title "Jupyter Sequences (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. The "Run" button is highlighted. Below the toolbar, there are buttons for file operations like Open, Save, and New, and cell navigation like Up, Down, and Run. A tooltip says "run cell, select below".

In [78]:

```
def IncrementByOne(x, y):
    x = x + 1
    y = y + 1
    return x, y
```

In [81]:

```
num1, num2 = IncrementByOne(5, 7)
print(num1)
print(num2)
```

6
8

Quiz Time

1. A function should be defined

- a) After it is called
- b) Before it is called
- c) Doesn't matter

2. A function in Python can only return 1 value. Is this statement True or False?

- a) True
- b) False

Quiz Time

1. A function should be defined

- a) After it is called
- b) Before it is called**
- c) Doesn't matter

2. A function in Python can only return 1 value. Is this statement True or False?

- a) True
- b) False**

Resources

Here are a few resources for you to revise Python syntax

- <https://www.w3schools.com/python/>
- <https://www.youtube.com/watch?v=rfscVS0vtbw>
- <https://www.tutorialspoint.com/python/index.htm>

References

- How to install python on Mac.

<https://www.youtube.com/watch?v=M323OL6K5vs>

- How to install Anaconda in Ubuntu

https://www.youtube.com/watch?v=dGm10q_y3xw

- How to install Anaconda in Mac

<https://www.youtube.com/watch?v=PJWnGNWQ1Dc>

- Managing Directories in Jupyter Noteook – Mac

<https://www.youtube.com/watch?v=G1Vm6wR0XIQ>

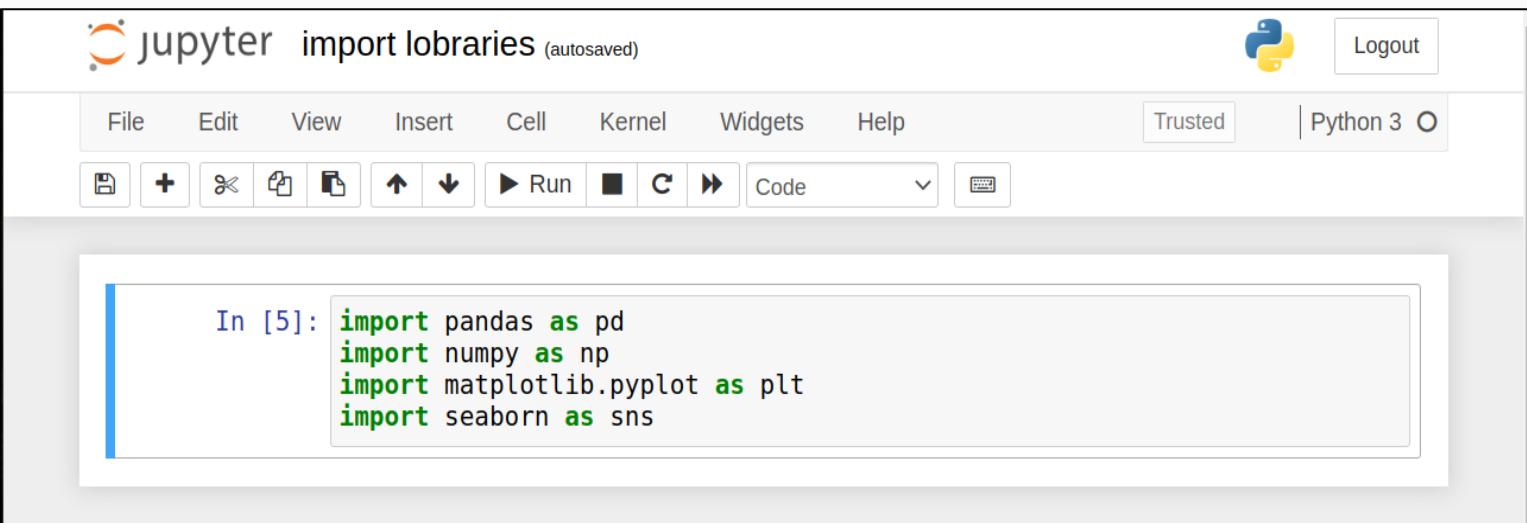
DATA SCIENCE WITH PYTHON

Installing Libraries

If you installed Anaconda, you do not need to download any libraries as it automatically installs all the popular data science libraries such as Pandas, Numpy, Matplotlib, Seaborn, etc.

Importing Libraries

- Open your Jupyter Notebook.
- To import a library we use the keyword **import** followed by library name.
- We can use the **as** keyword to use abbreviations for our library names.
- The common abbreviations used are
 - pd for pandas
 - np for numpy
 - plt for matplotlib.pyplot
 - sns for seaborn



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter import lobraries (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3
- Cell Content:** In [5]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Pandas Library for Data Science

- Pandas is a Python library for data manipulation and analysis.
- It allows exploring, cleaning, and processing tabular data.
- It provides two ways for storing data;
 - Series, which is one dimensional data structure
 - Data Frame, which is two dimensional data structure

DataFrame

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

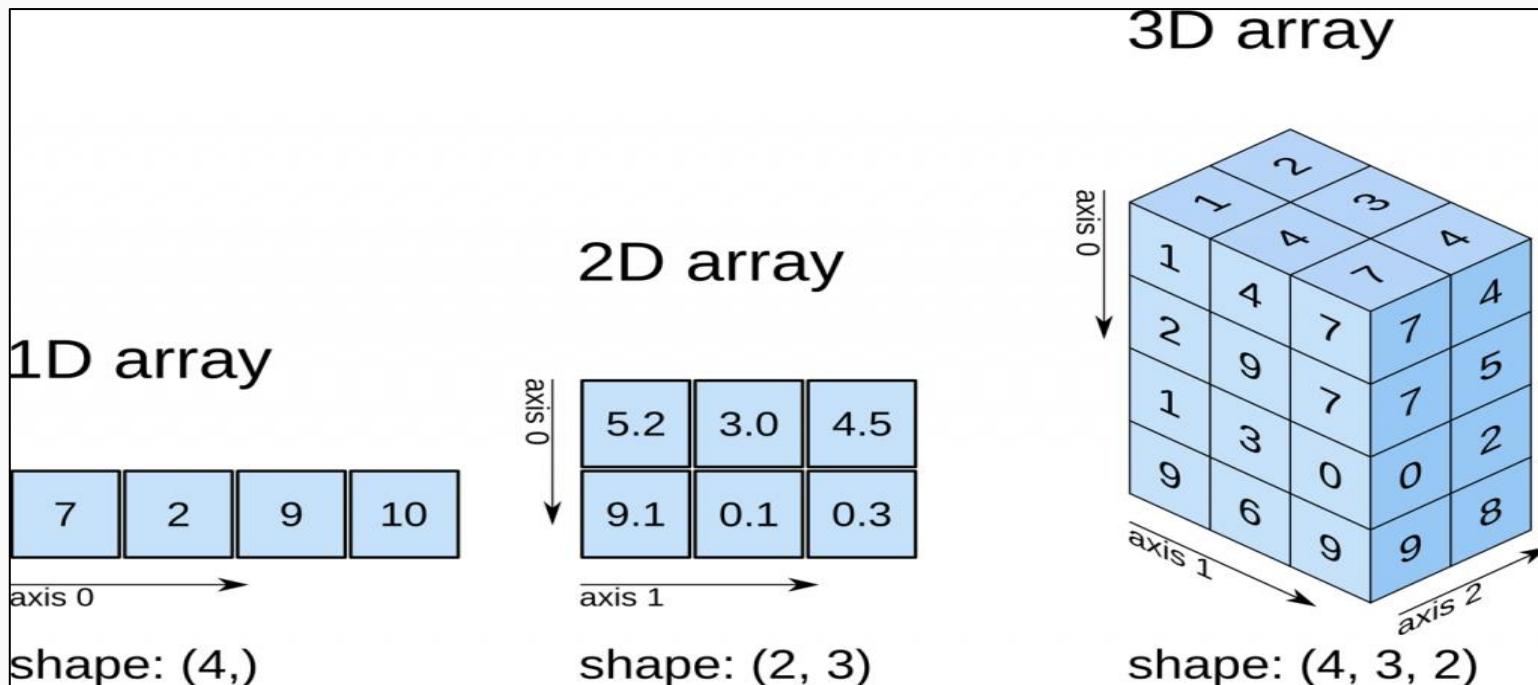
Series

0	70
1	120
2	70
3	50
4	110
5	110
6	110
7	130
8	90
9	90

Name: calories, dtype: int64

NumPy Library for Data Science

- NumPy stands for Numerical Python.
- It provides a data structure called NumPy array, which is a grid of values.
- It also provides a collection of high-level mathematical functions which can be performed on multi-dimensional NumPy arrays.

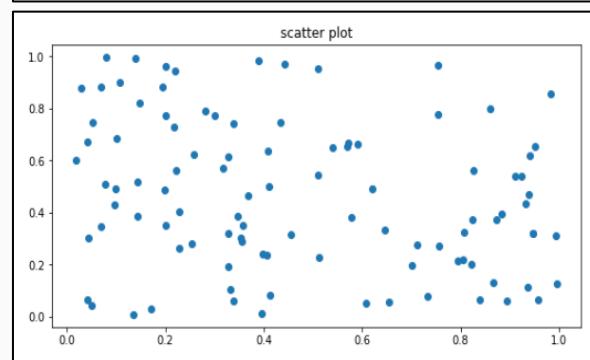
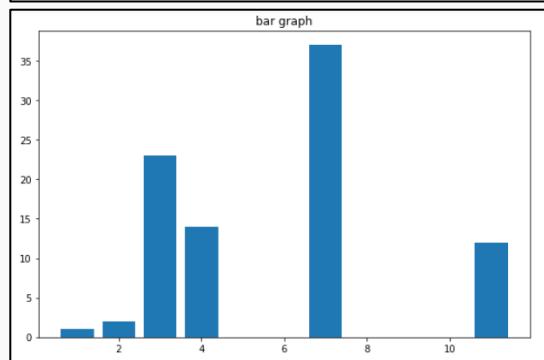
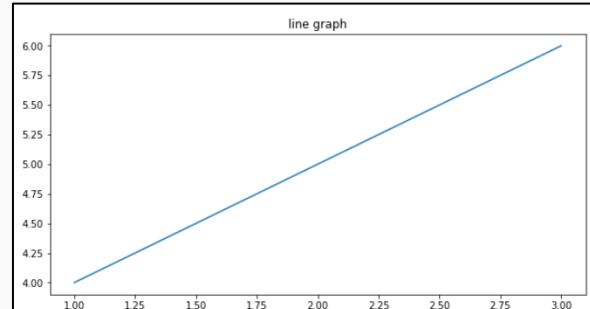
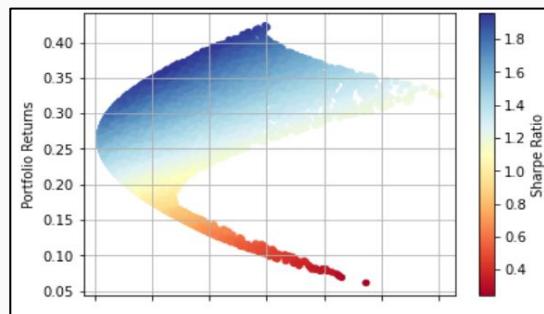


Pandas vs NumPy

NumPy	Pandas
NumPy and Pandas are both Python libraries for Data Science	
It is used for scientific computing	It is used for data manipulation such as storing, exploring, cleaning, and processing the data
It provides NumPy arrays which can be multidimensional	It provides two data structures; <ul style="list-style-type: none">• Series (one dimensional)• Data frames (two dimensional)
We use Pandas for data manipulation and NumPy for Mathematical Computations	
Since Pandas Series and Data Frames can be thought of as one and two dimensional NumPy arrays respectively, we can apply NumPy mathematical functions on them as well	

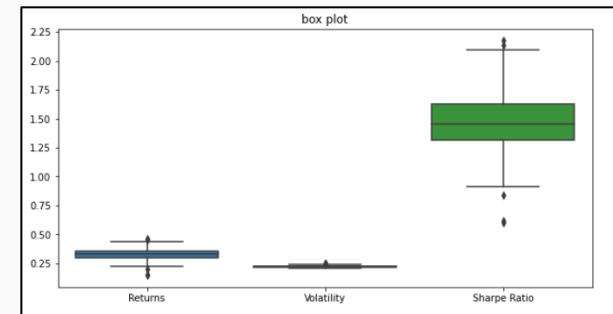
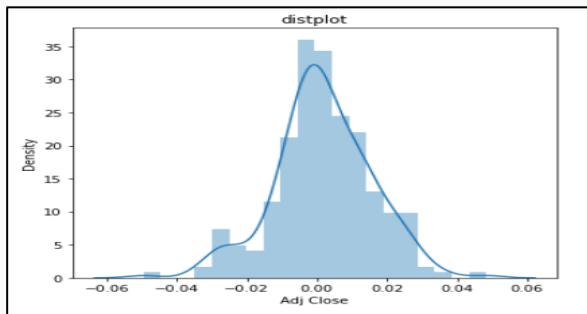
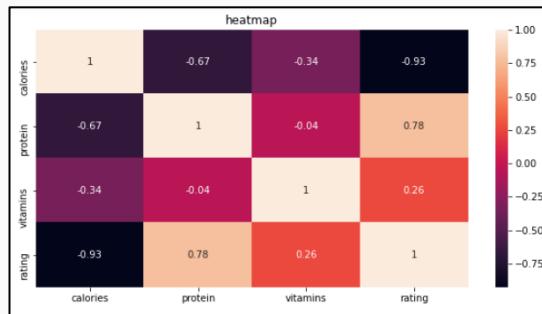
Matplotlib Library for Data Science

- Matplotlib is a visualization Python library, i.e., it is used for plotting graphs.
- The pyplot module inside of the Matplotlib provides the interface to underlying plotting functionality of the Matplotlib.
- We can create a number of different types of graphs using Matplotlib such as line graphs, bar graphs, histograms, scatter plots, area plots, pie plots, and so on.



Seaborn Library for Data Science

- Seaborn is another visualization Python library built on top of Matplotlib.
- It extends the functionality of Matplotlib and allows creating a variety of different graphs with fewer syntax.



NumPy Arrays

What are NumPy Arrays

- NumPy array is a multidimensional data structure designed to handle large data sets easily.
- A NumPy array is called **ndarray**.
- We can find the number of dimensions of a NumPy array using **.ndim**.

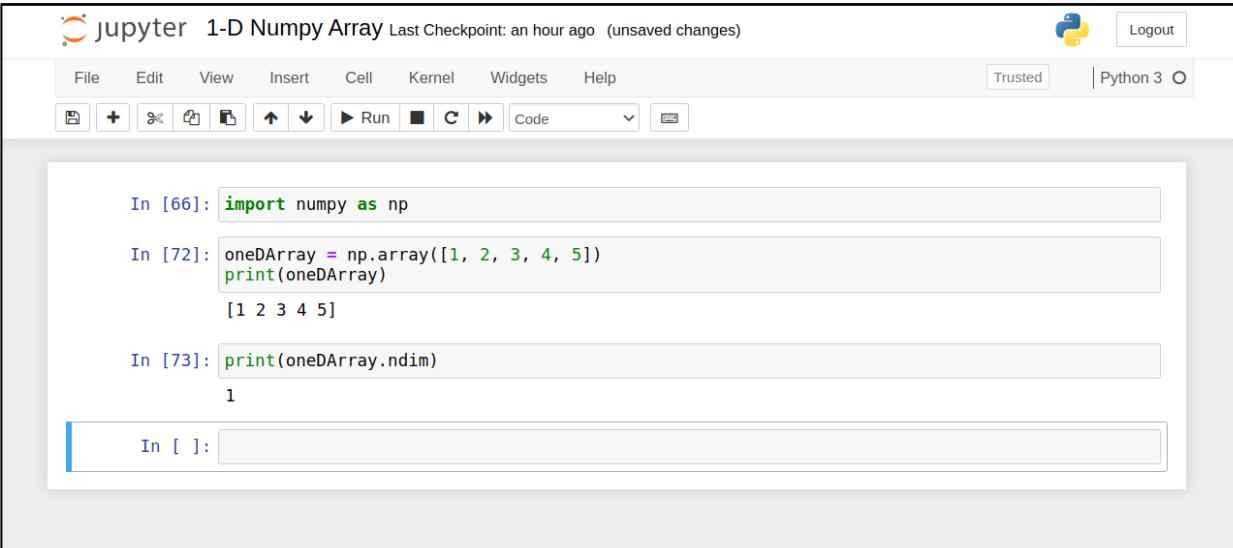
NumPy Arrays vs Python Lists

- NumPy arrays provide more built-in functionality as compared to Python lists.
- NumPy arrays make working with huge multi-dimensional data sets much easier with fewer syntax.
- NumPy arrays are also more efficient than Python lists in terms of memory consumption and speed.

Creating NumPy Arrays (1/3)

1-D NumPy Arrays

- A 1-D NumPy array is where each element of the outermost array is a 0-D array (scalar).
- We can create a NumPy array using the array() function in the NumPy library.
- We can create a NumPy array using either Python lists or tuples.
- To create a 1-D NumPy Array, we provide a 1-D Python list or tuple to the array() function.



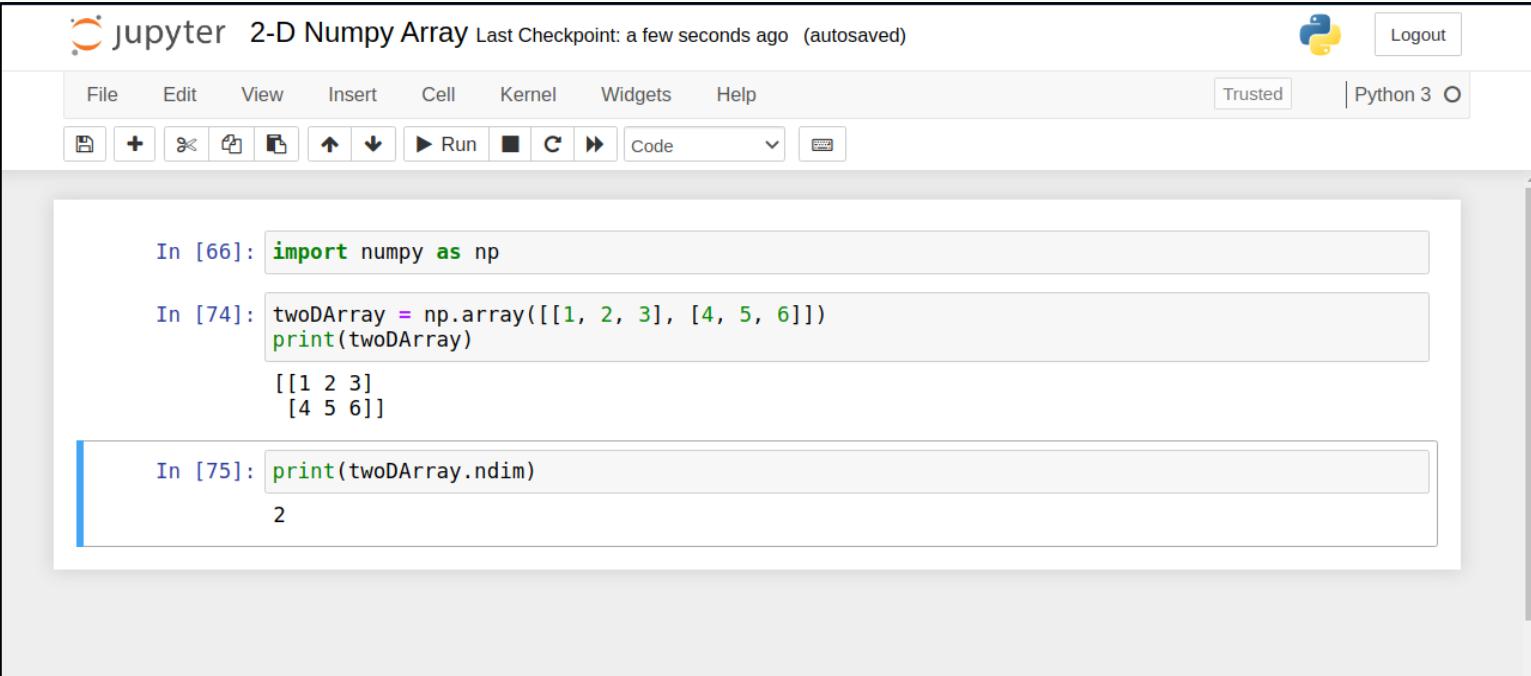
The screenshot shows a Jupyter Notebook interface titled "jupyter 1-D Numpy Array". The notebook has a "Python 3" kernel selected. The code cell In [66] contains the command `import numpy as np`. The code cell In [72] contains the command `oneDArray = np.array([1, 2, 3, 4, 5])` followed by `print(oneDArray)`, which outputs the list `[1 2 3 4 5]`. The code cell In [73] contains the command `print(oneDArray.ndim)`, which outputs the value `1`. A new code cell, In [], is currently active at the bottom.

```
In [66]: import numpy as np
In [72]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)
[1 2 3 4 5]
In [73]: print(oneDArray.ndim)
1
In [ ]:
```

Creating NumPy Arrays (2/3)

2-D NumPy Arrays

- A 2-D NumPy array is where each element in the outermost array is a 1-D array.
- To create a 2-D NumPy Array, we provide a 2-D Python list or tuple to the array() function.



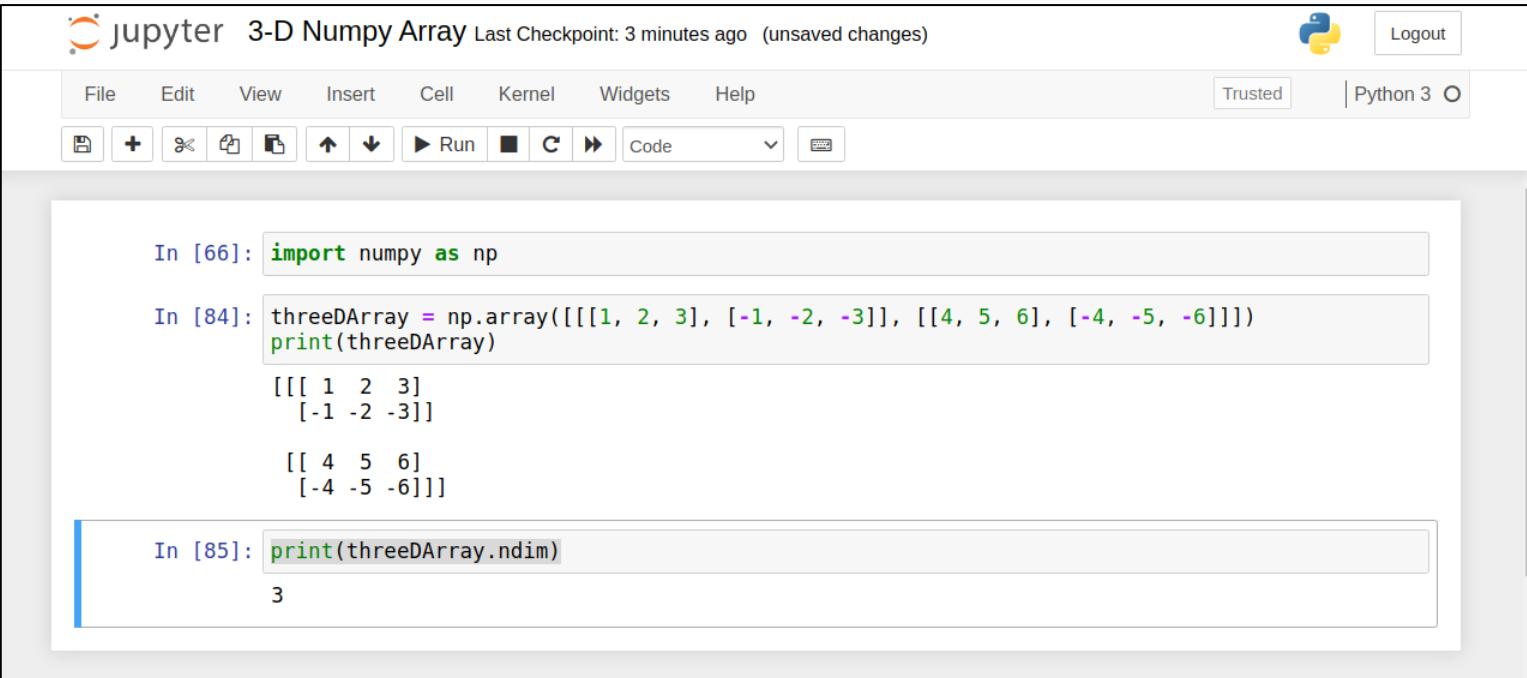
The screenshot shows a Jupyter Notebook interface with the title "jupyter 2-D Numpy Array" and a status bar indicating "Last Checkpoint: a few seconds ago (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code editor contains three cells:

- In [66]: `import numpy as np`
- In [74]: `twoDArray = np.array([[1, 2, 3], [4, 5, 6]])`
`print(twoDArray)`
[[1 2 3]
 [4 5 6]]
- In [75]: `print(twoDArray.ndim)`
2

Creating NumPy Arrays (3/3)

3-D NumPy Arrays

- A 3-D NumPy array is where each element of the outermost array is a 2-D array.
- To create a 3-D NumPy Array, we provide a 3-D Python list or tuple to the array() function.



The screenshot shows a Jupyter Notebook interface with the title "jupyter 3-D Numpy Array". The notebook has a "Trusted" status and is running on "Python 3". The code cell In [66] contains the command `import numpy as np`. The code cell In [84] contains the command `threeDArray = np.array([[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]])` followed by `print(threeDArray)`. The output of this cell is:

```
[[[ 1  2  3]
  [-1 -2 -3]]

 [[ 4  5  6]
  [-4 -5 -6]]]
```

The code cell In [85] contains the command `print(threeDArray.ndim)`. The output of this cell is the number 3.

Quiz Time

1. How many dimensions are in the array [[[1, 2, 3, 4]]]

- a) 1
- b) 2
- c) 3

Quiz Time

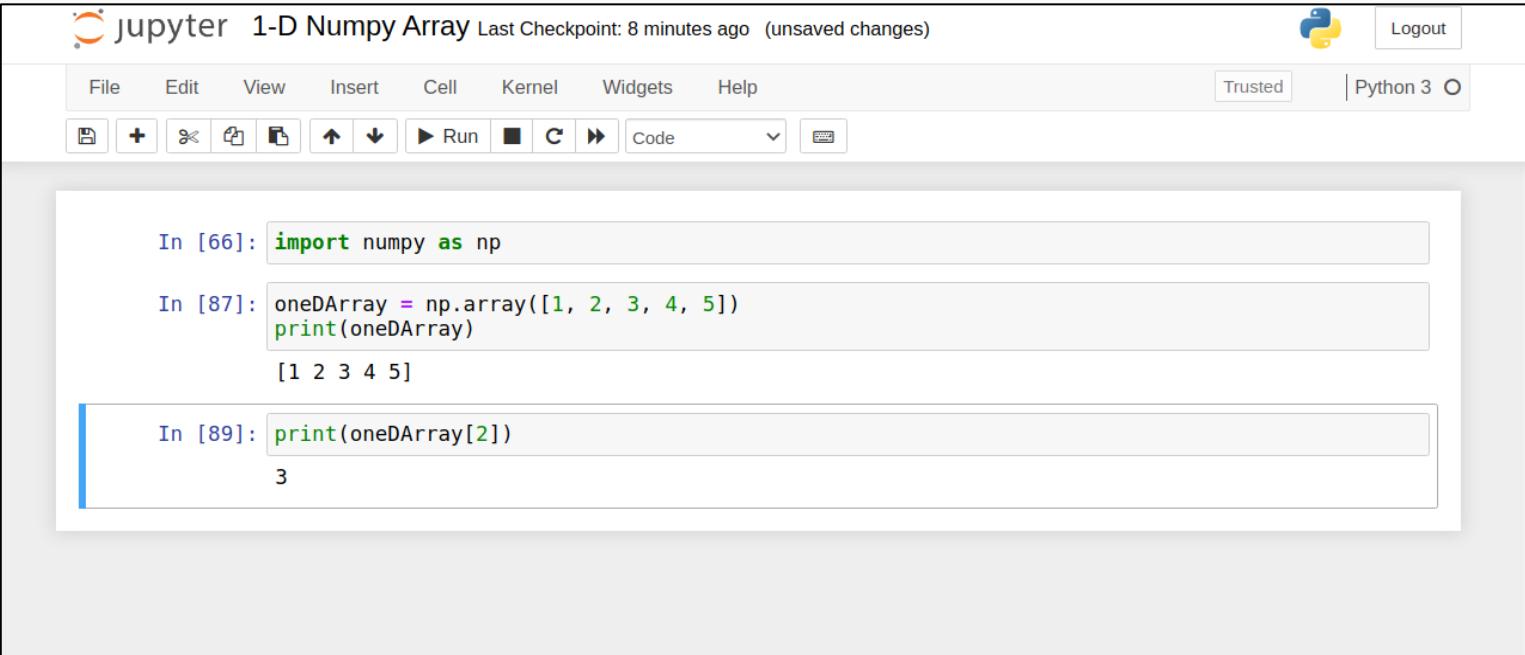
1. How many dimensions are in the array [[[1, 2, 3, 4]]]

- a) 1
- b) 2
- c) 3

Indexing NumPy Arrays (1/8)

1-D NumPy Arrays

- Indexing a 1-D NumPy array is the same as indexing a 1-D Python list.
- Provide the index of the element inside the square brackets to get that element.



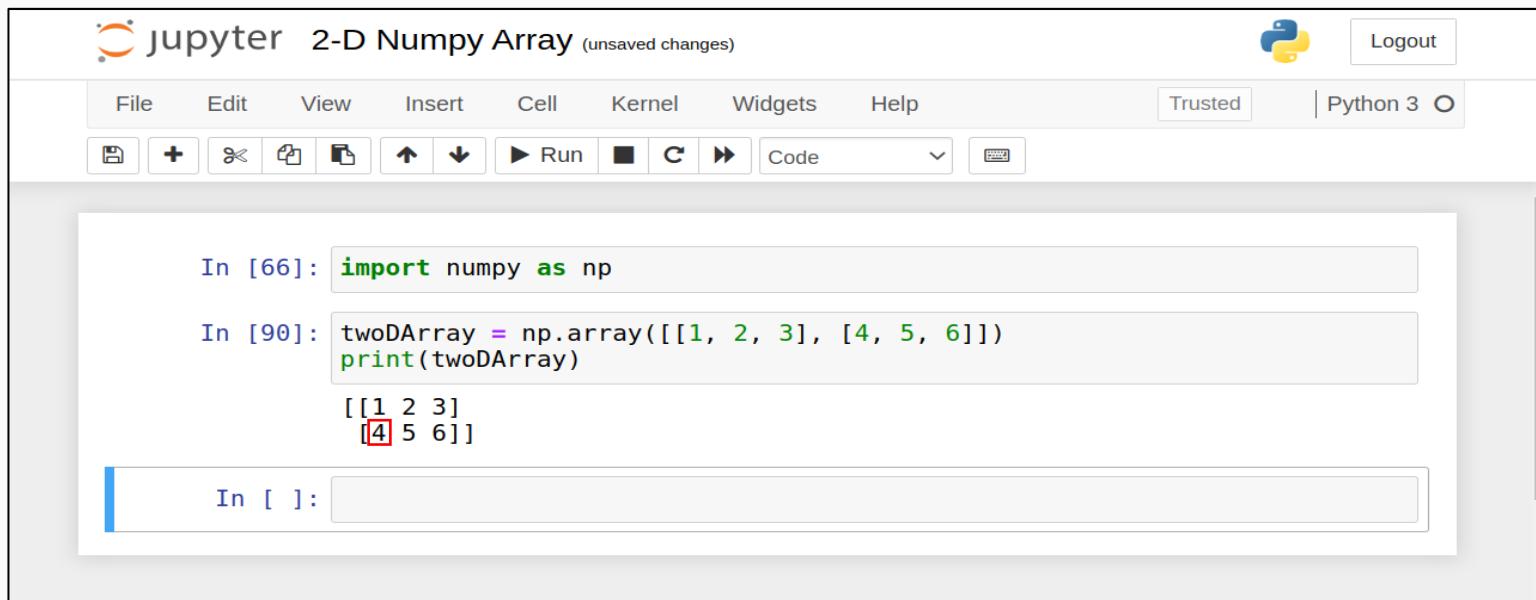
The screenshot shows a Jupyter Notebook interface with the title "jupyter 1-D Numpy Array". The notebook has a "Trusted" status and is running in "Python 3". The code cell In [66] contains the command `import numpy as np`. The code cell In [87] contains the command `oneDArray = np.array([1, 2, 3, 4, 5])` followed by `print(oneDArray)`, which outputs `[1 2 3 4 5]`. The code cell In [89] contains the command `print(oneDArray[2])`, which outputs the value `3`.

```
In [66]: import numpy as np
In [87]: oneDArray = np.array([1, 2, 3, 4, 5])
          print(oneDArray)
[1 2 3 4 5]
In [89]: print(oneDArray[2])
3
```

Indexing NumPy Arrays (2/8)

2-D NumPy Arrays

- To index a 2-D NumPy array, we provide 2 values inside the square brackets ([]).
 - First value is the index of the inner array
 - Second value is the index of the element inside the inner array
- In the following example, we get the first element of the second array.



The screenshot shows a Jupyter Notebook interface with the title "jupyter 2-D Numpy Array (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are standard notebook controls: file, new, cell, run, and code dropdown. The main area displays two code cells:

```
In [66]: import numpy as np
```

```
In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
print(twoDArray)
```

[[1 2 3]
[4 5 6]]

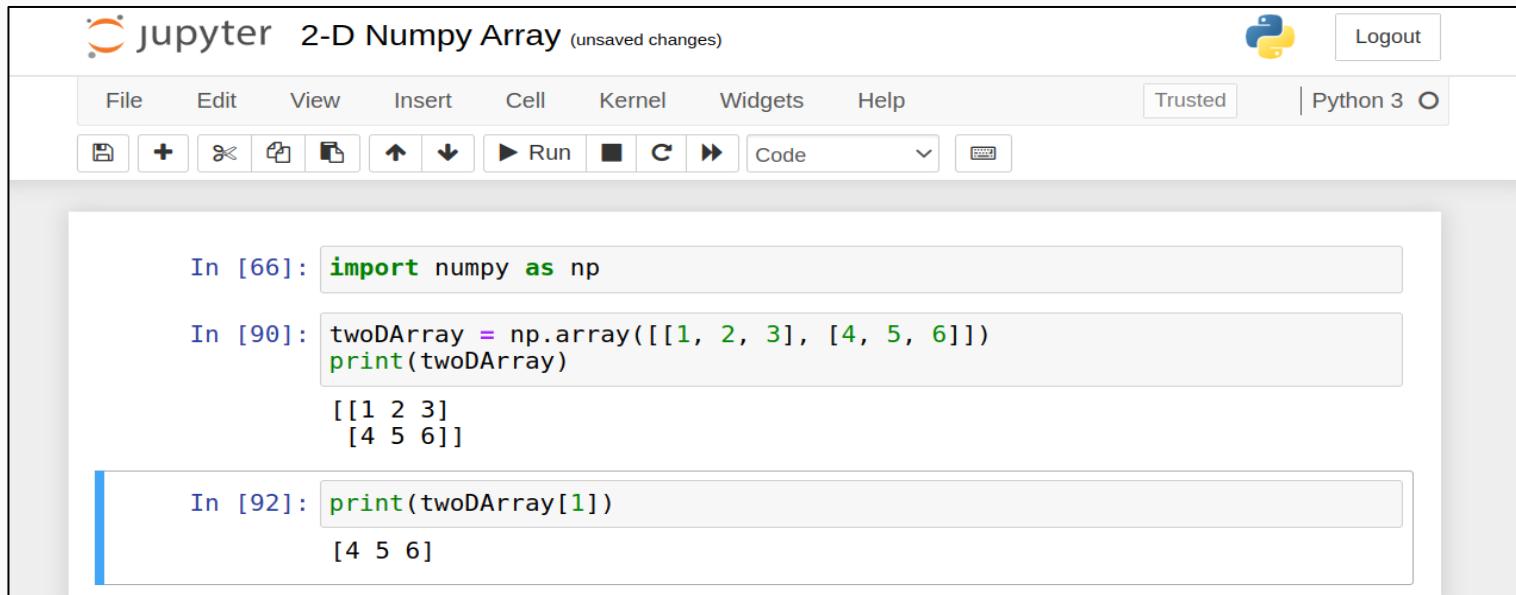
A third cell is partially visible at the bottom:

```
In [ ]:
```

Indexing NumPy Arrays (3/8)

2-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say twoDArray[1], we get the second array.



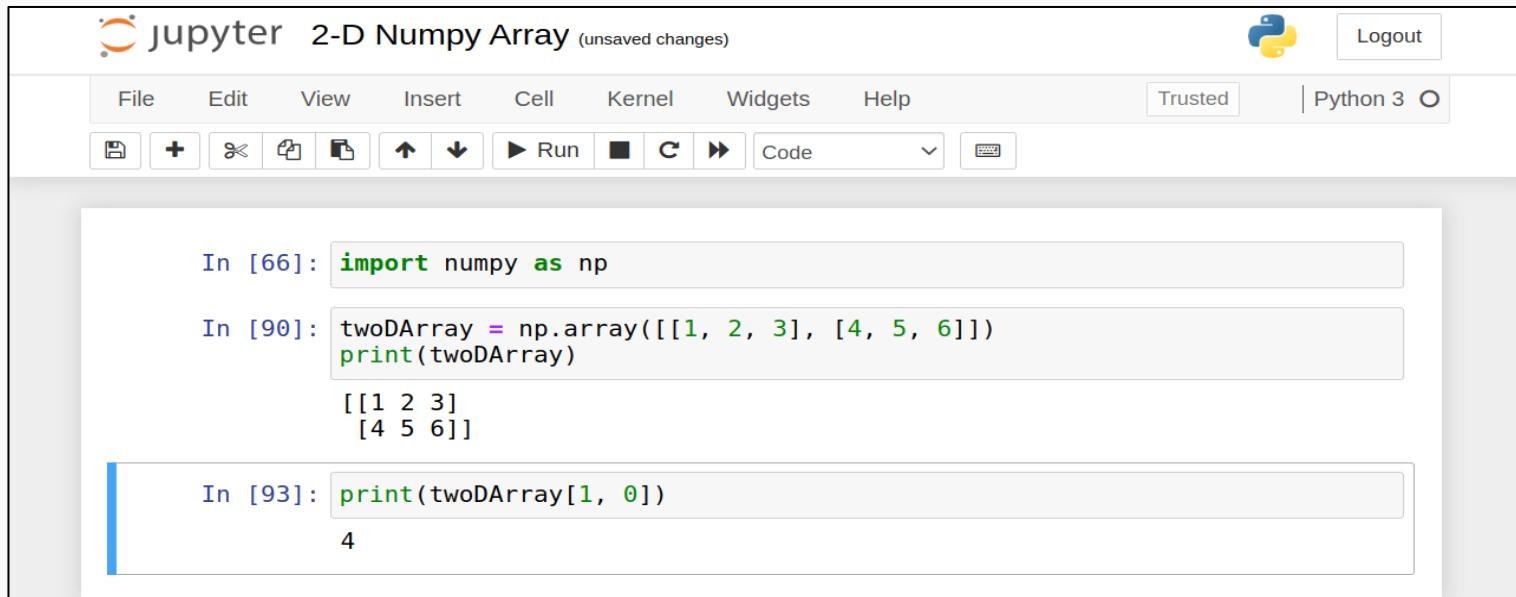
The screenshot shows a Jupyter Notebook interface titled "jupyter 2-D Numpy Array (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code editor displays three cells:

- In [66]: `import numpy as np`
- In [90]: `twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
print(twoDArray)`
Output: `[[1 2 3]
 [4 5 6]]`
- In [92]: `print(twoDArray[1])`
Output: `[4 5 6]`

Indexing NumPy Arrays (4/8)

2-D NumPy Arrays

- The second dimension contains 3 elements.
- If we say `twoDArray[1, 0]`, we get the first element of the second array.



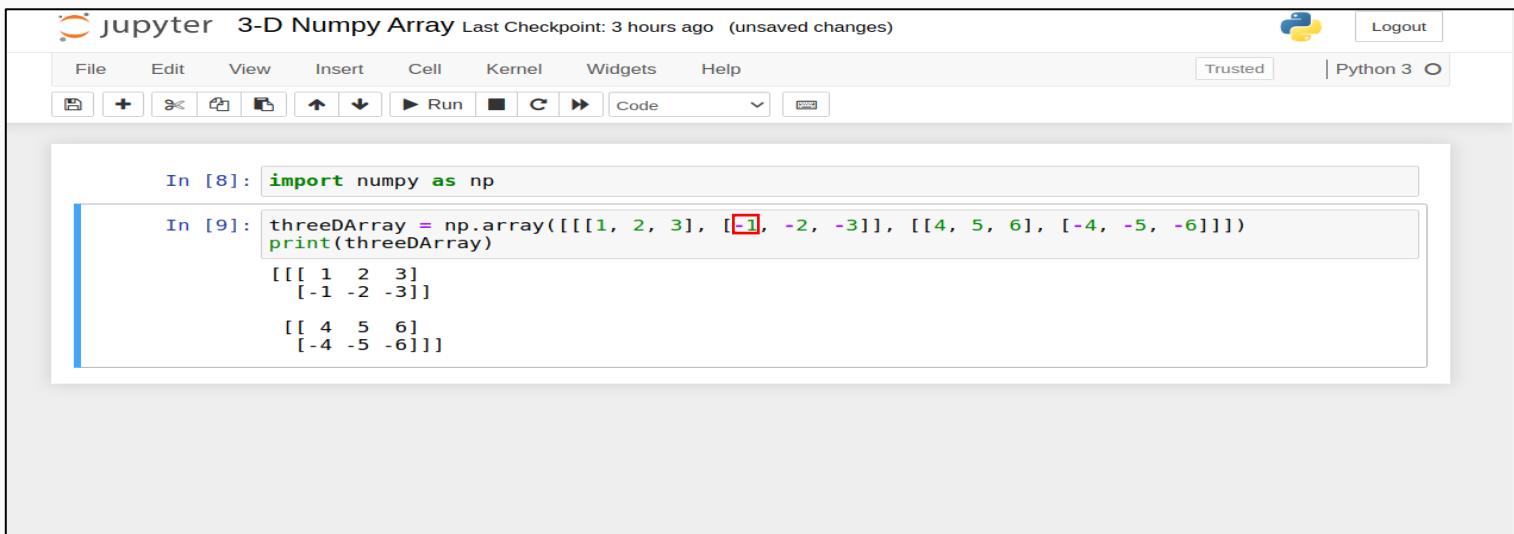
The screenshot shows a Jupyter Notebook interface with the title "jupyter 2-D Numpy Array (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. Below the toolbar is a toolbar with icons for file operations, cell selection, and run. The notebook has three cells:

- In [66]: `import numpy as np`
- In [90]: `twoDArray = np.array([[1, 2, 3], [4, 5, 6]])`
`print(twoDArray)`
[[1 2 3]
 [4 5 6]]
- In [93]: `print(twoDArray[1, 0])`
4

Indexing NumPy Arrays (5/8)

3-D NumPy Arrays

- To index a 3-D NumPy array, we provide 3 values inside the square brackets ([]).
 - First value is the index of the inner 2-D array in the first dimension.
 - Second value is the index of the inner 1-D array in the second dimension.
 - Third value is the index of the element in the third dimension.
- In the following example, we get the first element of the second array of the first array.



The screenshot shows a Jupyter Notebook interface with the title "jupyter 3-D Numpy Array". The notebook has two cells:

- In [8]: `import numpy as np`
- In [9]: `threeDArray = np.array([[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]])`
The output of this cell is:

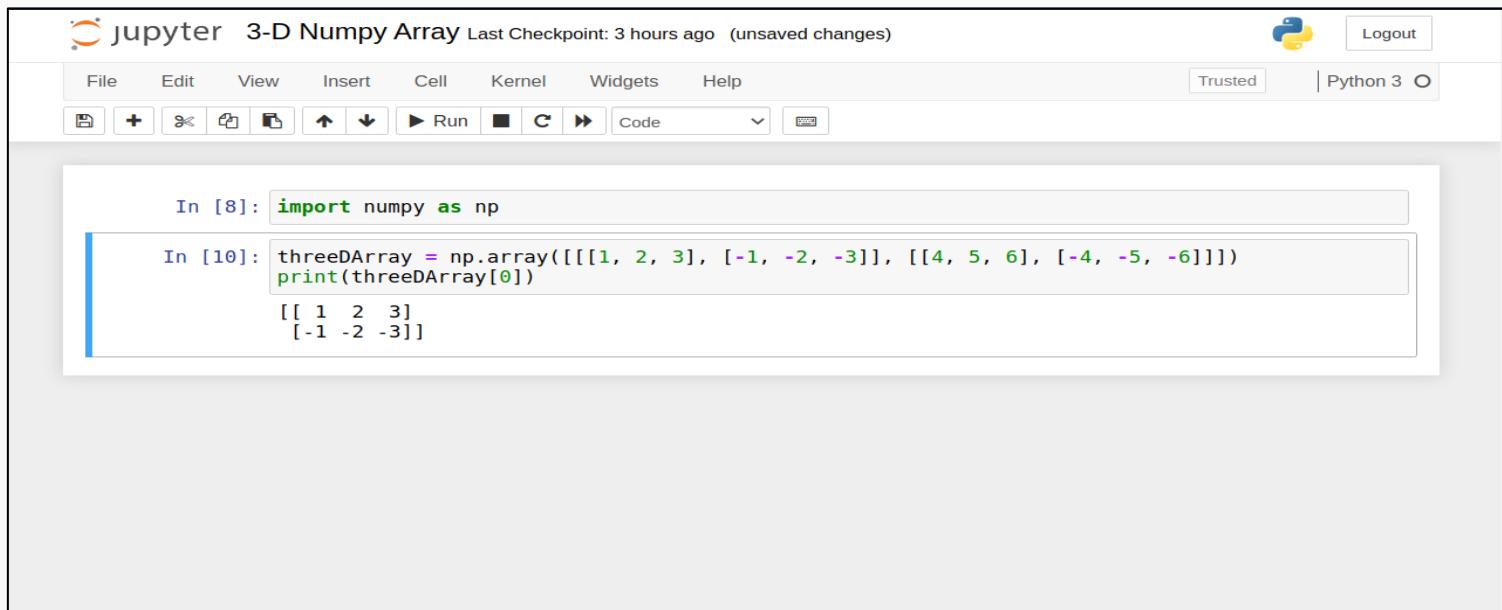
```
[[[ 1  2  3]
  [-1 -2 -3]]
 [[ 4  5  6]
  [-4 -5 -6]]]
```

A red box highlights the index `[-1, -2, -3]` in the second row of the first 2D array. A blue vertical bar highlights the entire second row of the first 2D array.

Indexing NumPy Arrays (6/8)

3-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say threeDArray[0], we get the first array.



The screenshot shows a Jupyter Notebook interface titled "Jupyter 3-D Numpy Array". The notebook has a "Trusted" status and is running on "Python 3". The code cell In [10] contains the following Python code:

```
In [8]: import numpy as np
In [10]: threeDArray = np.array([[1, 2, 3], [-1, -2, -3], [4, 5, 6], [-4, -5, -6]])
          print(threeDArray[0])
[[ 1  2  3]
 [-1 -2 -3]]
```

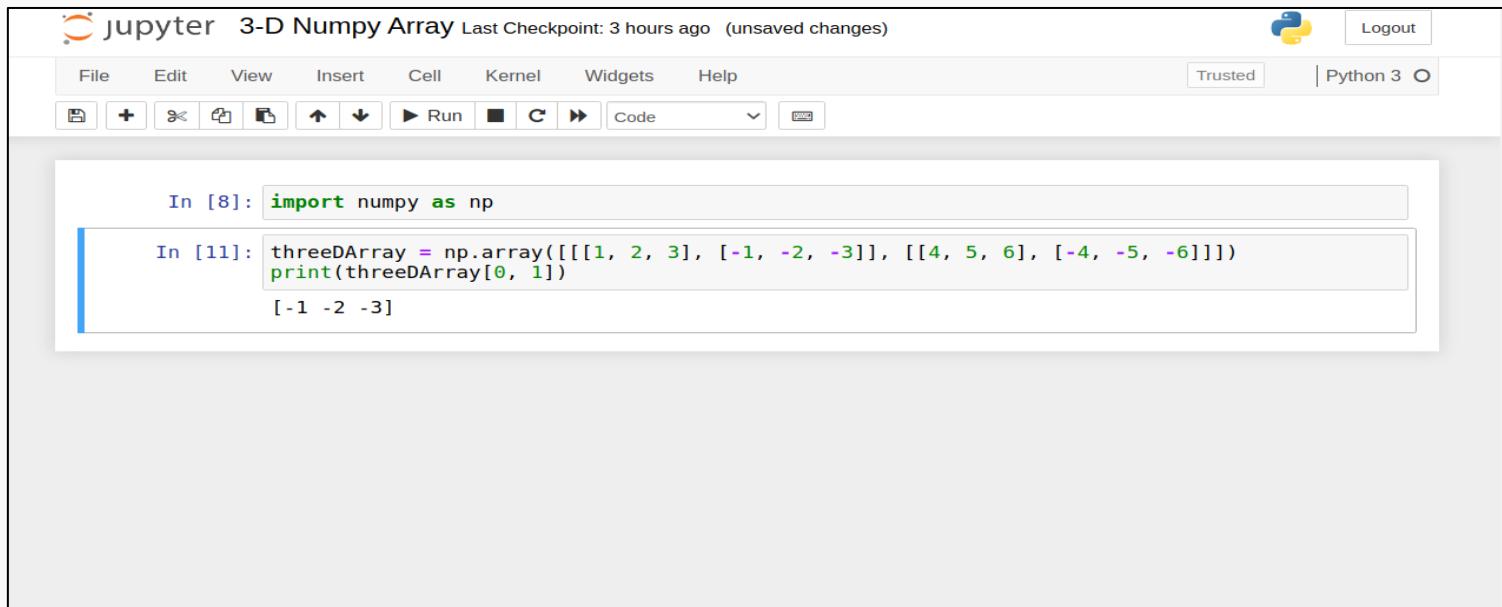
The output of the code is displayed in the cell below, showing the first array from the three-dimensional array:

```
[[ 1  2  3]
 [-1 -2 -3]]
```

Indexing NumPy Arrays (7/8)

3-D NumPy Arrays

- The second dimension again contains 2 arrays.
- If we say threeDArray[0, 1], we get the second array of the first array.



The screenshot shows a Jupyter Notebook interface with the title "jupyter 3-D Numpy Array". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar below the menu has icons for file operations, run, and cell selection. The code editor area displays two code cells:

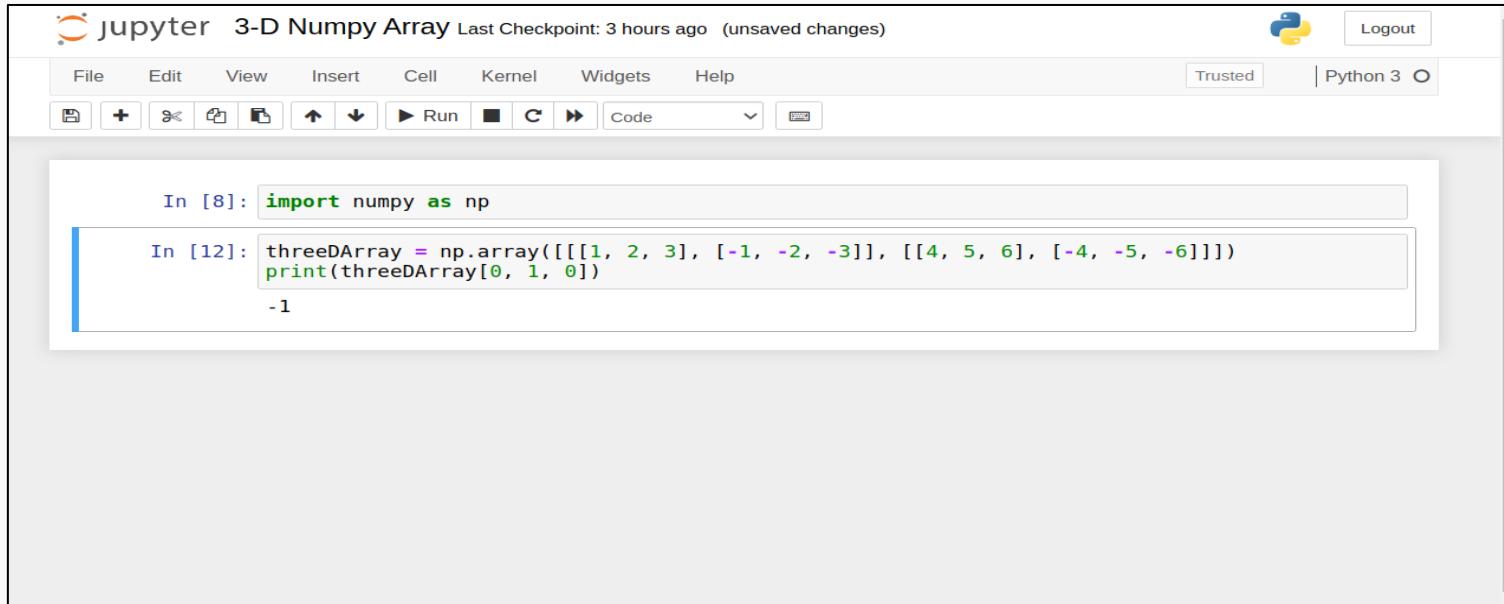
```
In [8]: import numpy as np
In [11]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
          print(threeDArray[0, 1])
          [-1 -2 -3]
```

The output cell (In [11]) shows the result of the print statement: `[-1 -2 -3]`.

Indexing NumPy Arrays (8/8)

3-D NumPy Arrays

- The third dimension contains 3 values.
- If we say threeDArray[0, 1, 0], we get the first element of the second array of the first array.



The screenshot shows a Jupyter Notebook interface with the title "jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cell In [8] contains `import numpy as np`. The code cell In [12] contains `threeDArray = np.array([[1, 2, 3], [-1, -2, -3], [[4, 5, 6], [-4, -5, -6]]])` and `print(threeDArray[0, 1, 0])`, which outputs `-1`.

```
In [8]: import numpy as np
In [12]: threeDArray = np.array([[1, 2, 3], [-1, -2, -3], [[4, 5, 6], [-4, -5, -6]]])
          print(threeDArray[0, 1, 0])
-1
```

Quiz Time

1. Consider a numpy array $x = [[[1, 2, 3, 4]]]$. What will $x[0, 0, 0]$ return?

- a) 1
- b) 3
- c) 4

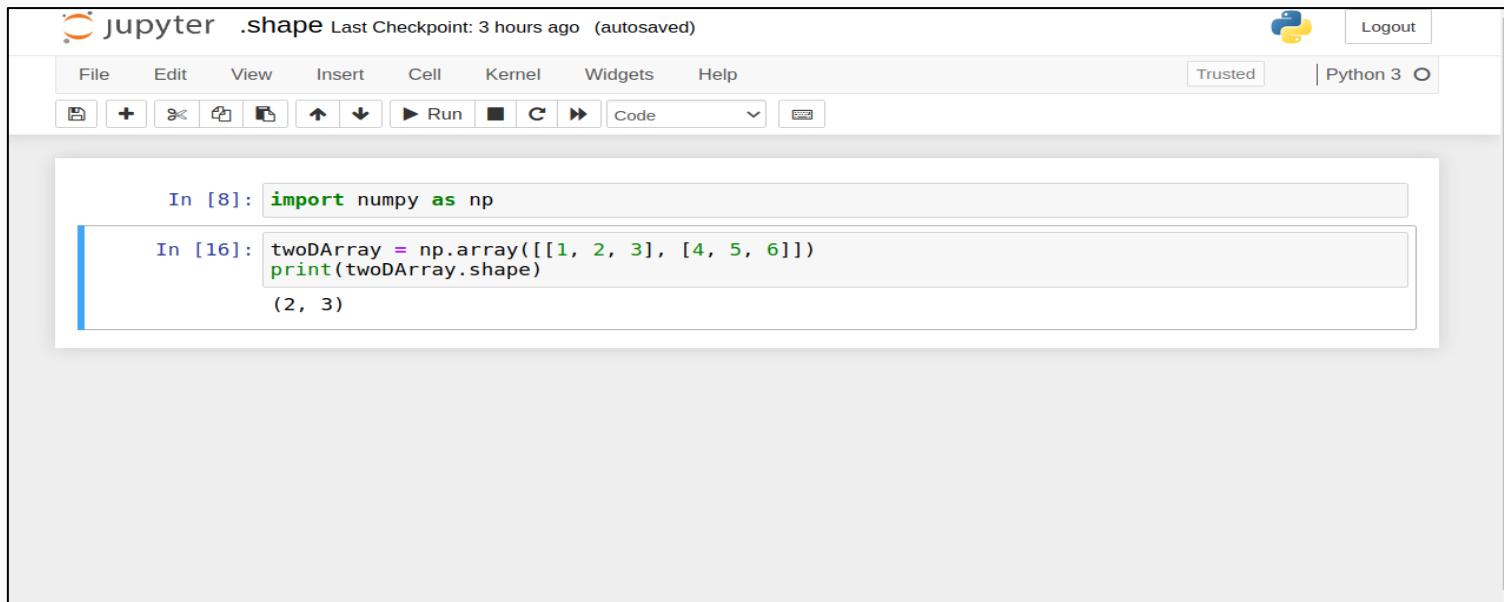
Quiz Time

1. Consider a numpy array $x = [[[1, 2, 3, 4]]]$. What will $x[0, 0, 0]$ return?

- a) 1
- b) 3
- c) 4

Array Shape

- NumPy arrays have a shape attribute which returns a tuple.
 - First value of the tuple gives the number of dimensions in the array
 - Second value of the tuple gives the number of elements in each dimension



The screenshot shows a Jupyter Notebook interface with the title bar "jupyter .shape Last Checkpoint: 3 hours ago (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and various cell type icons. A status bar indicates "Trusted" and "Python 3".

In [8]: `import numpy as np`

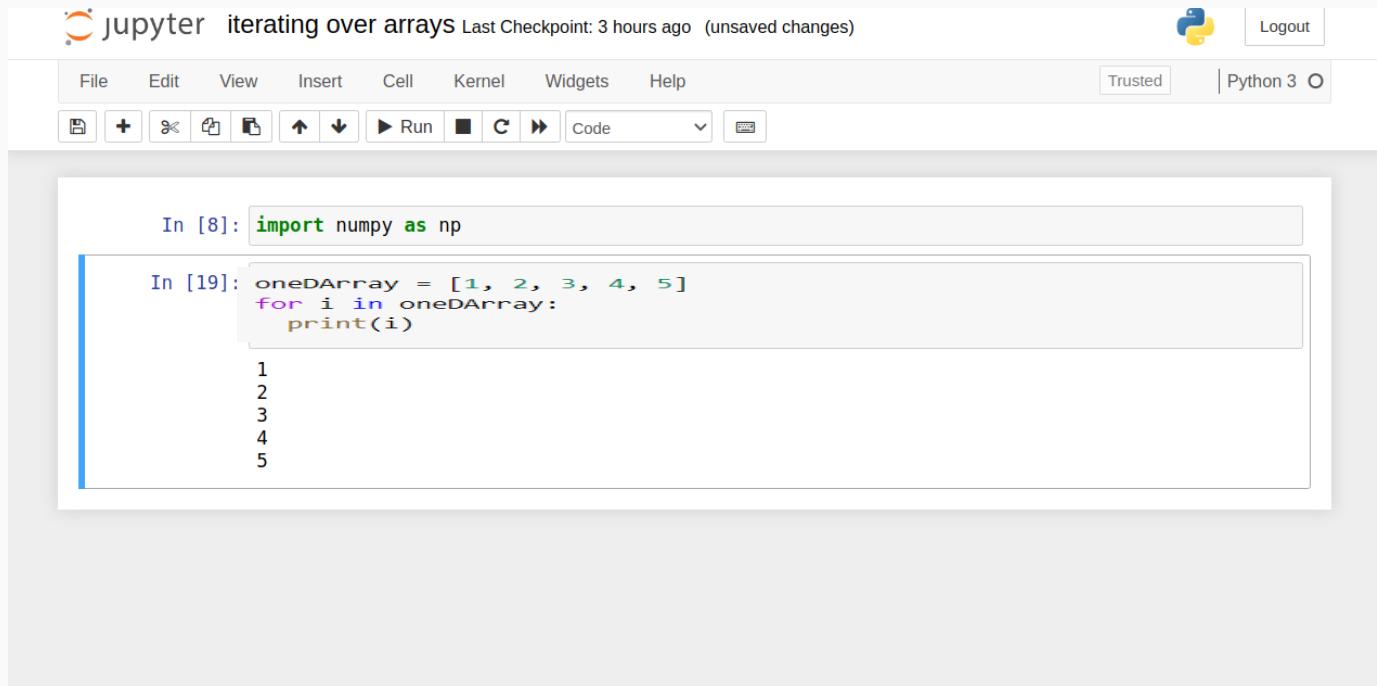
In [16]: `twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
print(twoDArray.shape)`

(2, 3)

Iterating Over NumPy Arrays (1/8)

1-D NumPy Arrays

- We can use a for loop to iterate over a 1-D array just as we do in a 1-D Python list.



The screenshot shows a Jupyter Notebook interface with the title "jupyter iterating over arrays". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are standard notebook controls for cell selection, running, and code input. Two code cells are visible:

```
In [8]: import numpy as np
```

```
In [19]: oneDArray = [1, 2, 3, 4, 5]
for i in oneDArray:
    print(i)
```

The output of the second cell is:

```
1
2
3
4
5
```

Iterating Over NumPy Arrays (2/8)

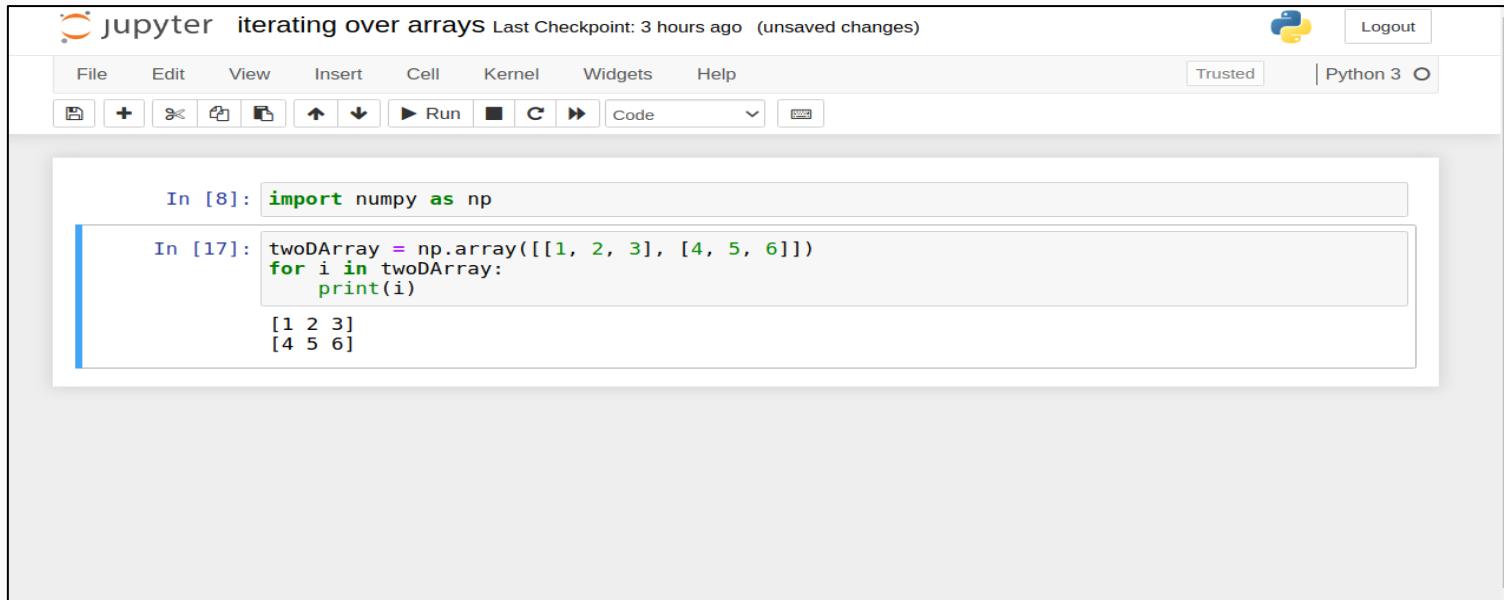
2-D NumPy Arrays

- We can use a nested for loop to iterate over a 2-D array.
 - The outer for loop iterates over the outer array.
 - The inner for loop iterates over the inner array.

Iterating Over NumPy Arrays (3/8)

2-D NumPy Arrays

- We use a for loop to iterate over the outer array.
- We print all the inner arrays.



The screenshot shows a Jupyter Notebook interface with the title "jupyter iterating over arrays". The notebook has a "Python 3" kernel and is set to "Trusted". The code cell In [8] contains the command `import numpy as np`. The code cell In [17] contains the following Python code:

```
twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
for i in twoDArray:
    print(i)
```

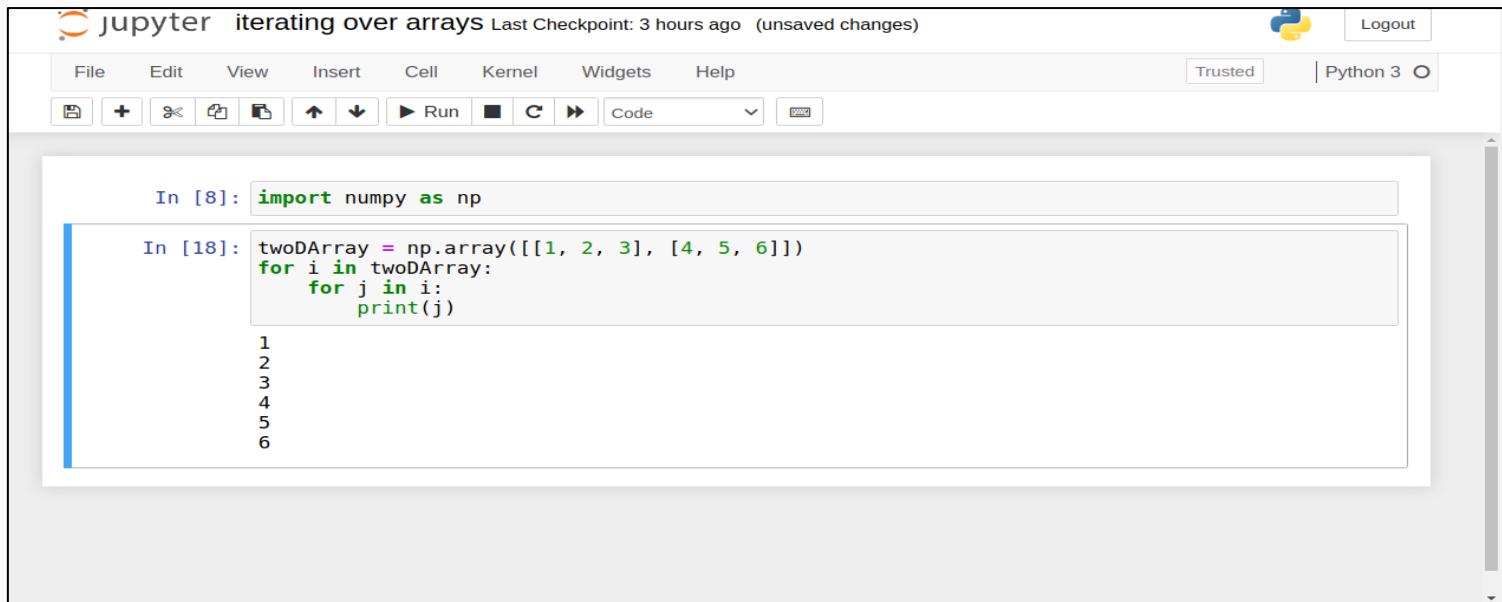
The output of this code is displayed in the cell below, showing the two 1x3 arrays:

```
[1 2 3]
[4 5 6]
```

Iterating Over NumPy Arrays (4/8)

2-D NumPy Arrays

- We use another for loop nested inside the outer for loop to iterate over the inner array.
- We print all the elements in each of the inner arrays.



The screenshot shows a Jupyter Notebook interface with the title "jupyter iterating over arrays". The notebook has a "Trusted" status and is using a "Python 3" kernel. The toolbar includes standard buttons for file operations, cell selection, and run. In the code editor, two cells are visible:

```
In [8]: import numpy as np
```

```
In [18]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
for i in twoDArray:
    for j in i:
        print(j)
```

The output cell (In [18]) displays the numbers 1 through 6, each on a new line, indicating the execution of the nested loops.

Iterating Over NumPy Arrays (5/8)

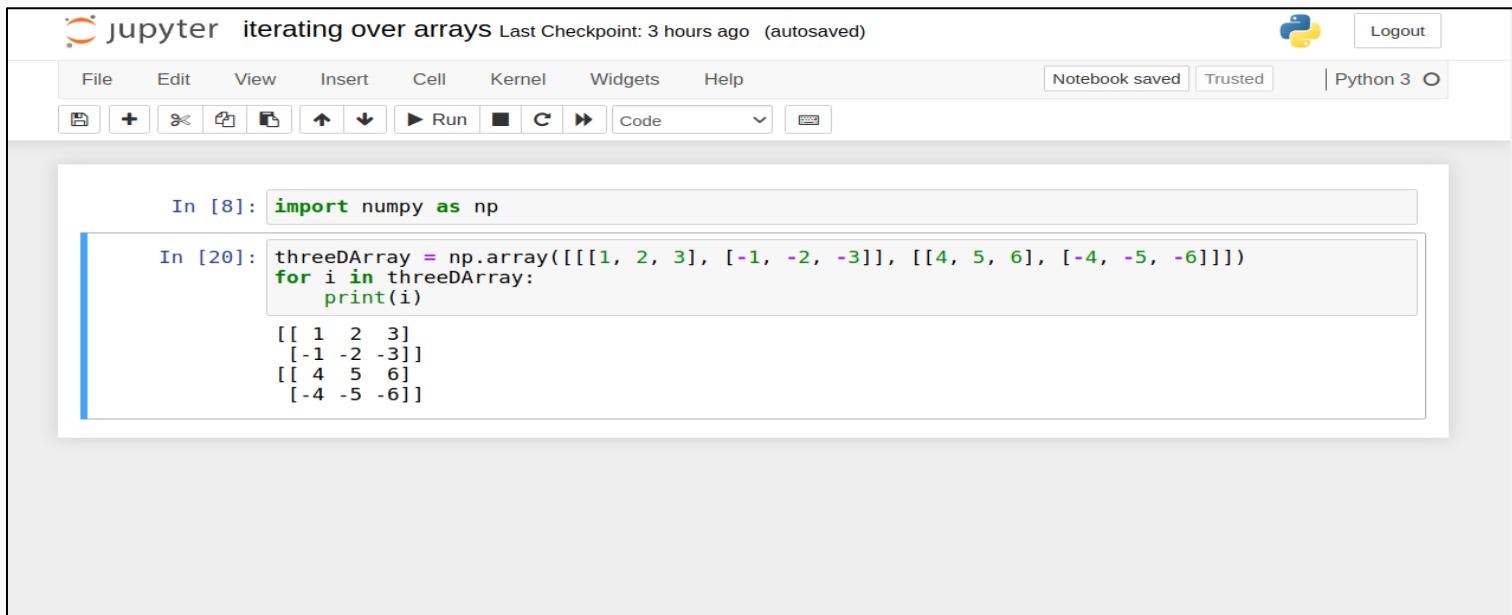
3-D NumPy Arrays

- We can use 3 nested for loops to iterate over a 3-D array.
 - The outermost for loop iterates over the arrays in the first dimension.
 - The middle for loop iterates over the arrays in the second dimension.
 - The innermost for loop iterates over all the elements in the third dimension.

Iterating Over NumPy Arrays (6/8)

3-D NumPy Arrays

- Outermost array contains 2 arrays both of which are 2-D.
- We use a for loop to print these 2-D arrays.



The screenshot shows a Jupyter Notebook interface with the title "jupyter iterating over arrays" and a status bar indicating "Last Checkpoint: 3 hours ago (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 icon. The notebook area has two cells:

- In [8]: `import numpy as np`
- In [20]:

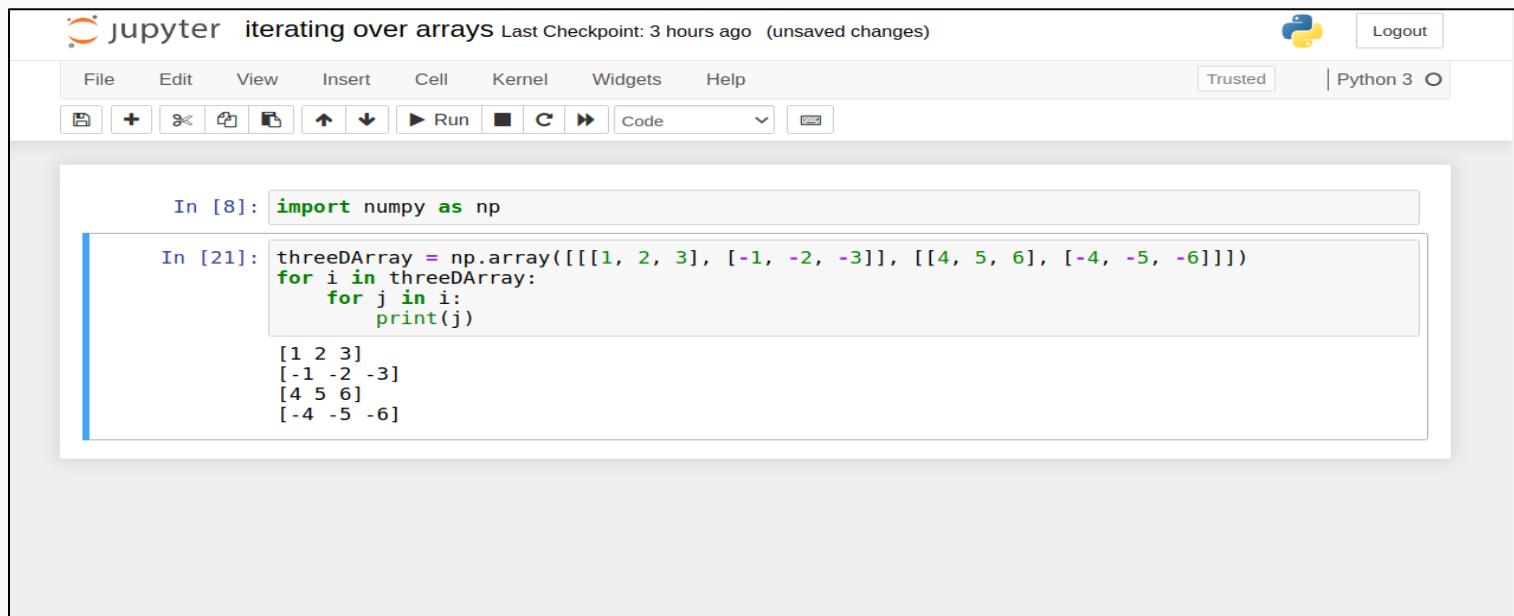
```
threeDArray = np.array([[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]])
for i in threeDArray:
    print(i)
```

[[1 2 3]
 [-1 -2 -3]
 [[4 5 6]
 [-4 -5 -6]]]

Iterating Over NumPy Arrays (7/8)

3-D NumPy Arrays

- Each of the 2-D array contains 2 arrays in the second dimension, each of which is 1-D.
- We use another for loop nested within the first for loop to print these 1-D arrays.



The screenshot shows a Jupyter Notebook interface with the title "Jupyter iterating over arrays". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3, and Logout. The code cell In [8] contains the command `import numpy as np`. The code cell In [21] contains the following Python code:

```
threeDArray = np.array([[1, 2, 3], [-1, -2, -3], [4, 5, 6], [-4, -5, -6]])
for i in threeDArray:
    for j in i:
        print(j)
```

The output of the code is displayed in the cell, showing the following 1-D arrays:

```
[1 2 3]
[-1 -2 -3]
[4 5 6]
[-4 -5 -6]
```

Iterating Over NumPy Arrays (8/8)

3-D NumPy Arrays

- Each of the 1-D array contains 3 elements in the third dimension, each of which is 1-D.
- We use another for loop nested within the first 2 for loops to print these elements.



The screenshot shows a Jupyter Notebook interface with the title "jupyter iterating over arrays". The top bar includes "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help" menus, along with "Trusted" and "Python 3" status indicators. The main area has two code cells:

```
In [8]: import numpy as np
```

```
In [22]: threeDArray = np.array([[1, 2, 3], [-1, -2, -3], [4, 5, 6], [-4, -5, -6]])
for i in threeDArray:
    for j in i:
        for k in j:
            print(k)
```

The output of the second cell is displayed below it:

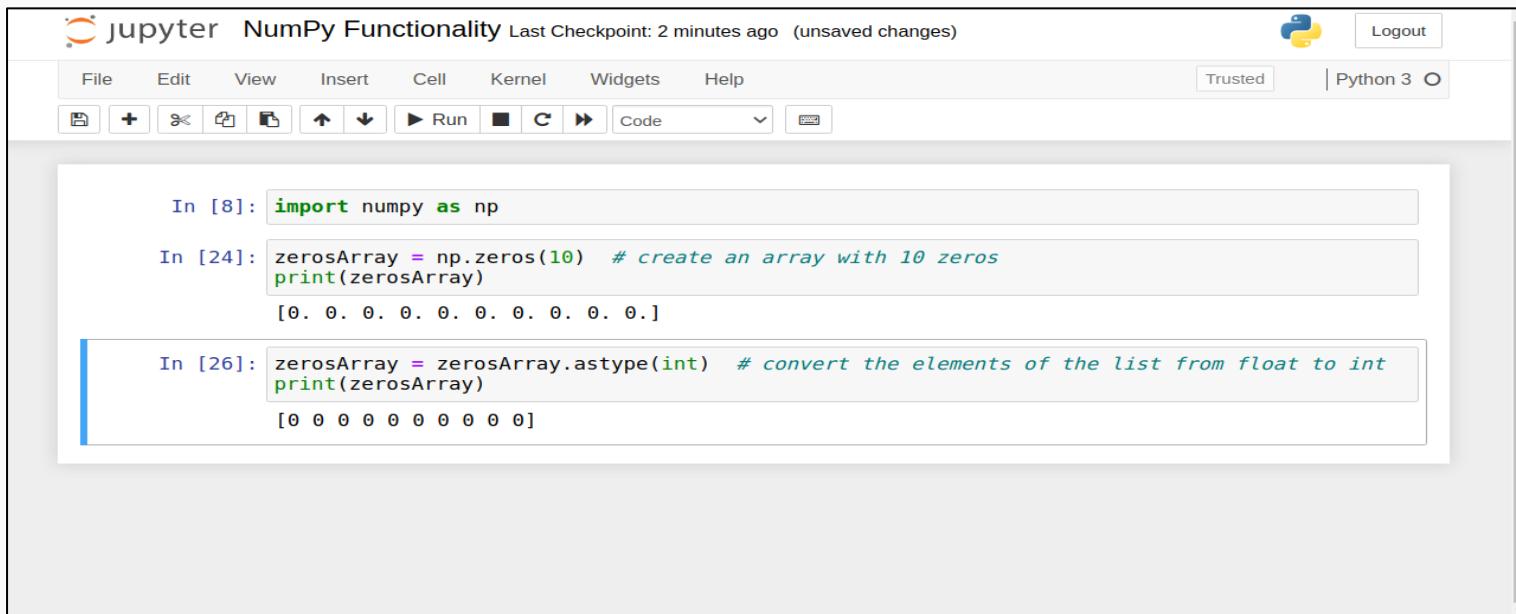
```
1
2
3
-1
-2
-3
4
5
6
-4
-5
-6
```

Mathematics for Data Science

- NumPy provides us with a huge collection of high-level functions for multi-dimensional arrays.
- Let's have a look at some of the functionality provided by NumPy.

.zeros()

- To create a NumPy array prefilled with zeros, we can use the `.zeros()` built-in NumPy function.
- `.zeros()` gives us a list prefilled with float zeros. To convert this list into integer list, we use the `.astype()` function.

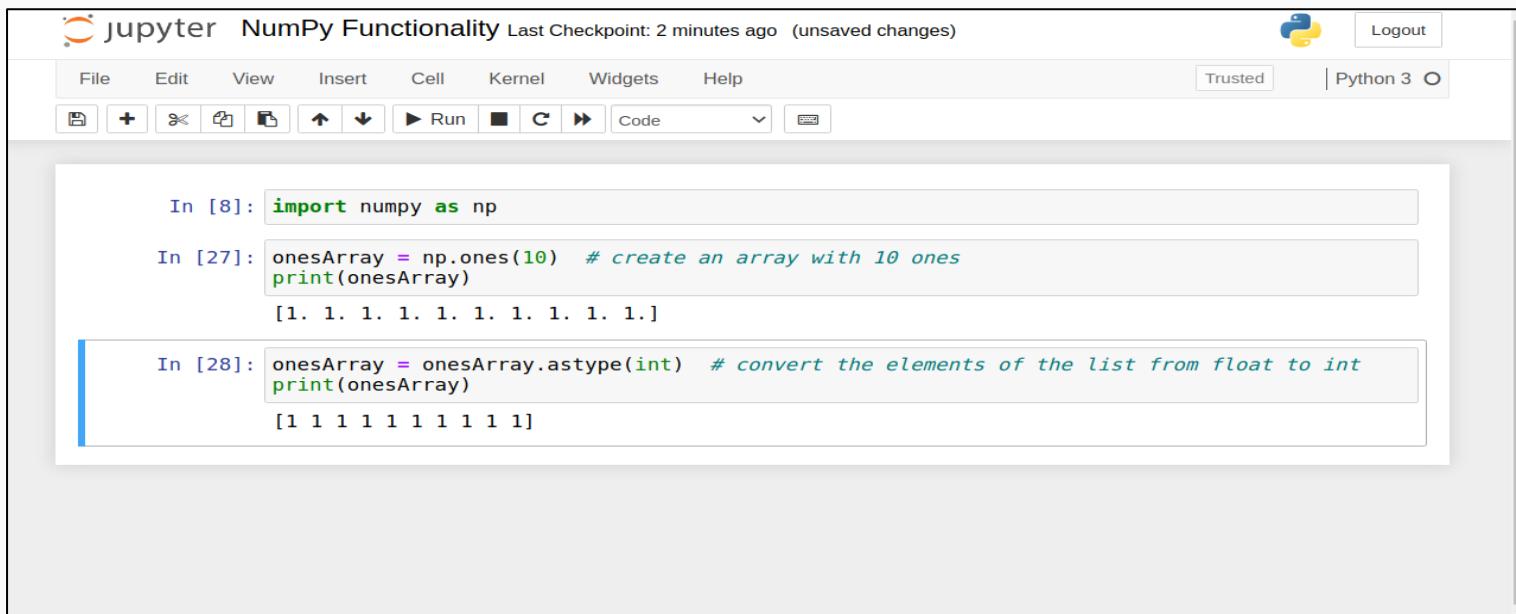


The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 2 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code editor contains three cells:

- In [8]: `import numpy as np`
- In [24]: `zerosArray = np.zeros(10) # create an array with 10 zeros`
`print(zerosArray)`
`[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]`
- In [26]: `zerosArray = zerosArray.astype(int) # convert the elements of the list from float to int`
`print(zerosArray)`
`[0 0 0 0 0 0 0 0 0 0]`

.ones()

- To create a NumPy array prefilled with ones, we can use the `.ones()` built-in NumPy function.
- `.ones()` gives us a list prefilled with float ones. To convert this list into integer list, we use the `.astype()` function.

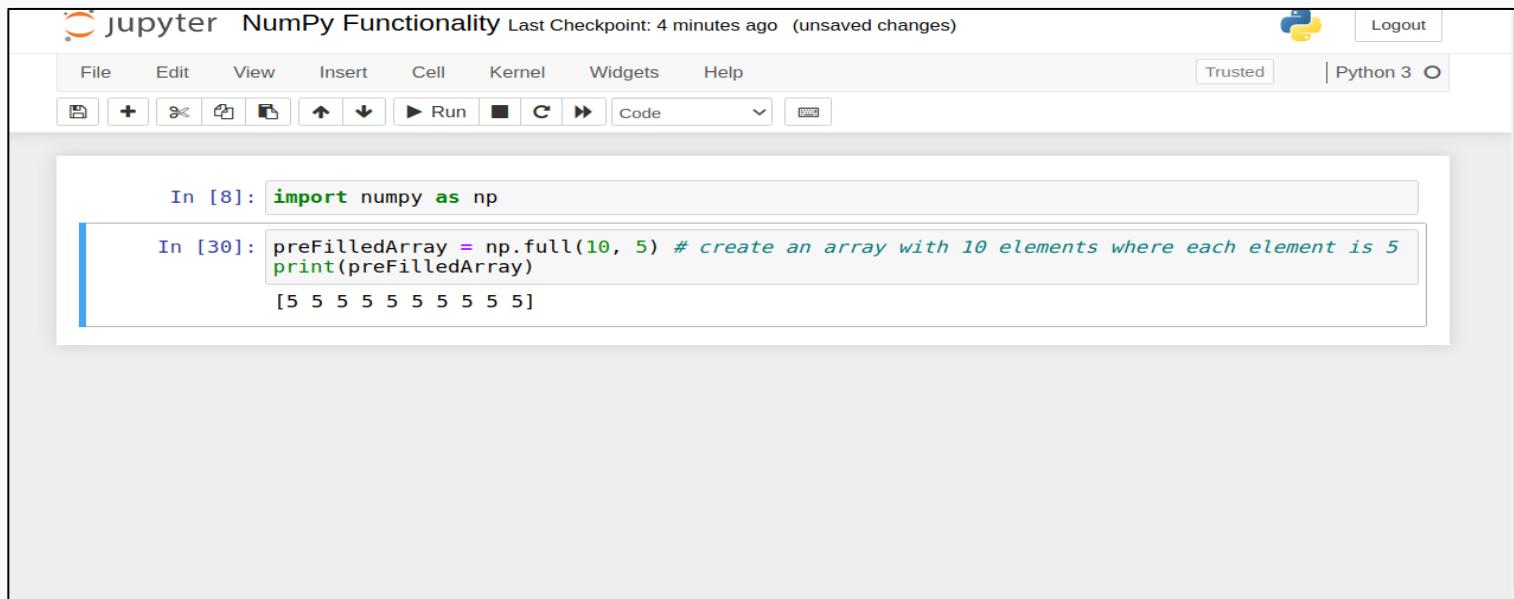


The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 2 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code editor contains three cells:

- In [8]: `import numpy as np`
- In [27]: `onesArray = np.ones(10) # create an array with 10 ones`
`print(onesArray)`
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
- In [28]: `onesArray = onesArray.astype(int) # convert the elements of the list from float to int`
`print(onesArray)`
[1 1 1 1 1 1 1 1 1 1]

.full()

- To create a NumPy array prefilled with some specific number, we can use the `.full()` built-in NumPy function.
 - First argument in the `.full()` function is the size of the array
 - Second argument in the `.full()` function is the value that we want our list to be pre-filled with



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cell In [8] contains `import numpy as np`. The code cell In [30] contains `preFilledArray = np.full(10, 5) # create an array with 10 elements where each element is 5` followed by `print(preFilledArray)`. The output of the code is `[5 5 5 5 5 5 5 5 5 5]`.

Quiz Time

1. What is the correct syntax to create a numpy array of 9 elements filled with all zeros (float)?
 - a) np.zeros()
 - b) np.zeros(0)
 - c) np.zeros(9)

Quiz Time

1. What is the correct syntax to create a numpy array of 9 elements filled with all zeros (float)?
 - a) np.zeros()
 - b) np.zeros(0)
 - c) np.zeros(9)

Scalar Operations (1/5)

Addition

- We can add a scalar to a NumPy array simply by using the (+) operator.
- The scalar quantity is added to each of the elements of the array.
- Note that adding a scalar to a Python list will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality". The notebook has three cells:

- In [38]:** `matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [39]:** `print(matrix + 2)`
Output:
`[[3 4 5]
 [6 7 8]]`
- In [40]:** `pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList + 2)`
Output:

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-40-788a2ae497a2> in <module>  
      1 pythonList = [[1, 2, 3], [4, 5, 6]]  
----> 2 print(pythonList + 2)  
  
TypeError: can only concatenate list (not "int") to list
```

Scalar Operations (2/5)

Subtraction

- We can subtract a scalar from a NumPy array simply by using the (-) operator.
- The scalar quantity is subtracted from each of the elements of the array.
- Note that subtracting a scalar from a Python list will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 8 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Run, and Code buttons. The Python version is set to "Python 3".

In [41]:

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
```

[[1 2 3]
 [4 5 6]]

In [42]:

```
print(matrix - 2)
```

[[[-1 0 1]
 [2 3 4]]]

In [43]:

```
pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList - 2)
```

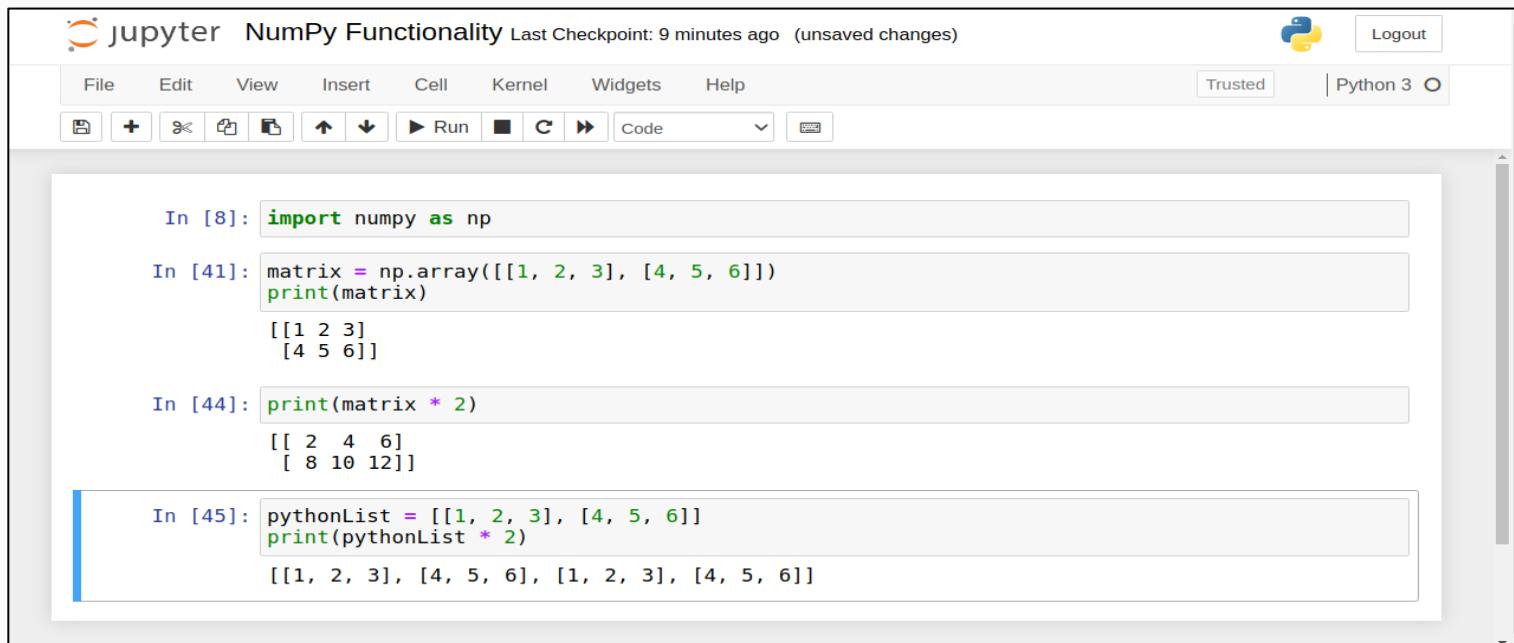
TypeError
<ipython-input-43-0cdc8cdac650> in <module>
 1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList - 2)

TypeError: unsupported operand type(s) for -: 'list' and 'int'

Scalar Operations (3/5)

Multiplication

- We can multiply a scalar with a NumPy array simply by using the (*) operator.
- The scalar quantity is multiplied with each of the elements of the array.
- Note that multiplying a scalar with a Python list will result in list concatenation.



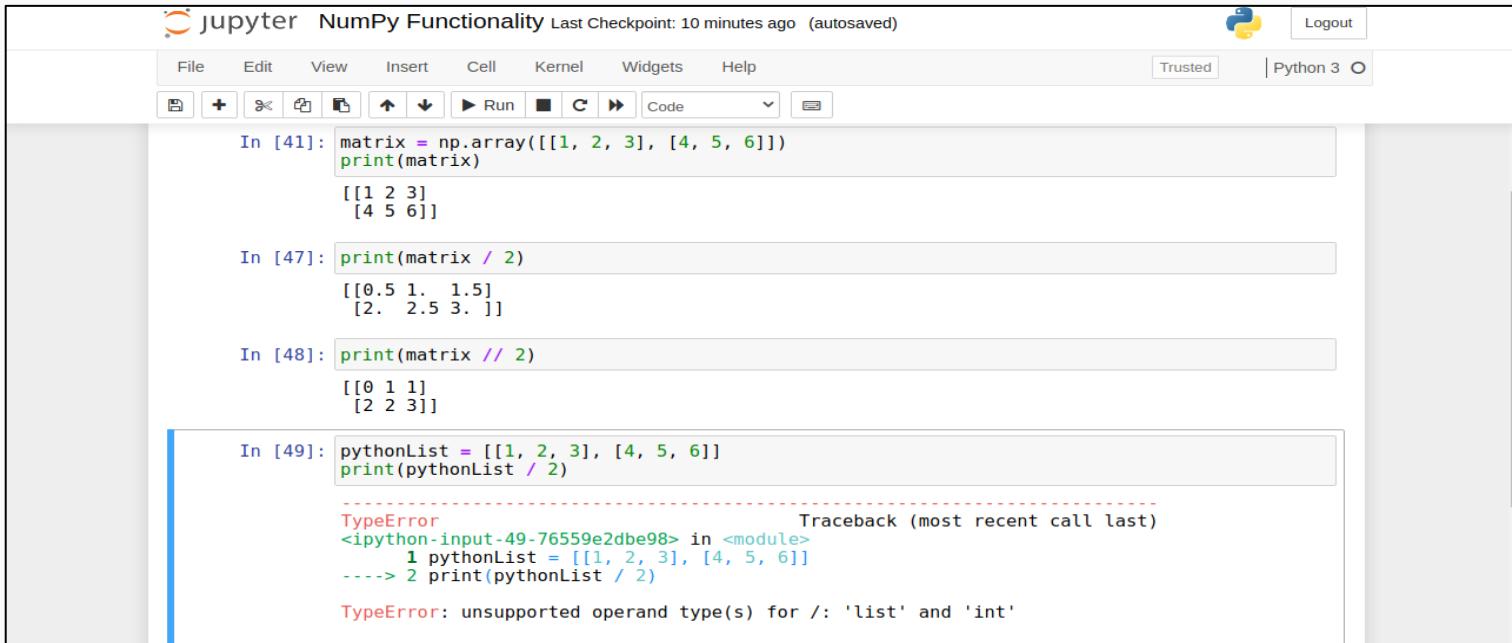
The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and a status bar indicating "Last Checkpoint: 9 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are several icons for file operations like new, open, save, and run. The notebook area contains the following code cells:

- In [8]: `import numpy as np`
- In [41]: `matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)`
[[1 2 3]
[4 5 6]]
- In [44]: `print(matrix * 2)`
[[2 4 6]
[8 10 12]]
- In [45]: `pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList * 2)`
[[1, 2, 3], [4, 5, 6], [1, 2, 3], [4, 5, 6]]

Scalar Operations (4/5)

Division

- We can divide a NumPy array by a scalar simply by using the (/) operator for float division or (//) operator for integer division.
- Each of the elements of the array is divided by the scalar.
- Note that dividing a Python list by a scalar will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 10 minutes ago (autosaved)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cells and their outputs are as follows:

- In [41]: `matrix = np.array([[1, 2, 3], [4, 5, 6]])`
[[1 2 3]
[4 5 6]]
- In [47]: `print(matrix / 2)`
[[0.5 1. 1.5]
[2. 2.5 3.]]
- In [48]: `print(matrix // 2)`
[[0 1 1]
[2 2 3]]
- In [49]: `pythonList = [[1, 2, 3], [4, 5, 6]]`
`print(pythonList / 2)`

`TypeError`
<ipython-input-49-76559e2dbe98> in <module>
 1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList / 2)

`TypeError: unsupported operand type(s) for /: 'list' and 'int'`

Scalar Operations (5/5)

Power

- We can raise each element of a NumPy array to a power simply by using the `(**)` operator.
- Note that raising the elements of a Python list using `(**)` operator will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality". The notebook has a "Python 3" kernel selected. The code cell In [41] contains:

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
```

The output is:

```
[[1 2 3]
 [4 5 6]]
```

The code cell In [50] contains:

```
print(matrix ** 2)
```

The output is:

```
[[ 1  4  9]
 [16 25 36]]
```

The code cell In [51] contains:

```
pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList ** 2)
```

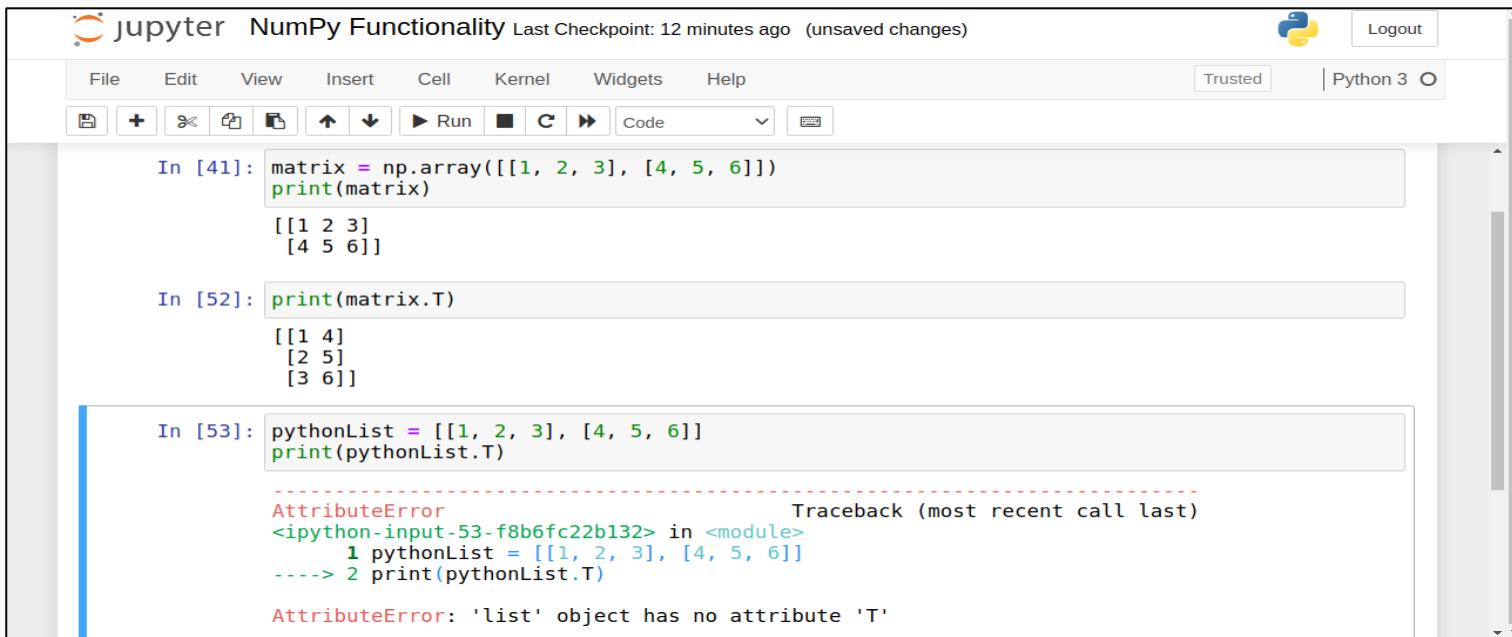
This cell results in a `TypeError`:

```
----- Traceback (most recent call last)
<ipython-input-51-3013918ac0> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList ** 2)

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Transpose

- We can take the transpose of a NumPy array by putting .T at the end of the array.
- Note that taking transpose of a Python list will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "Jupyter NumPy Functionality". The notebook has three cells:

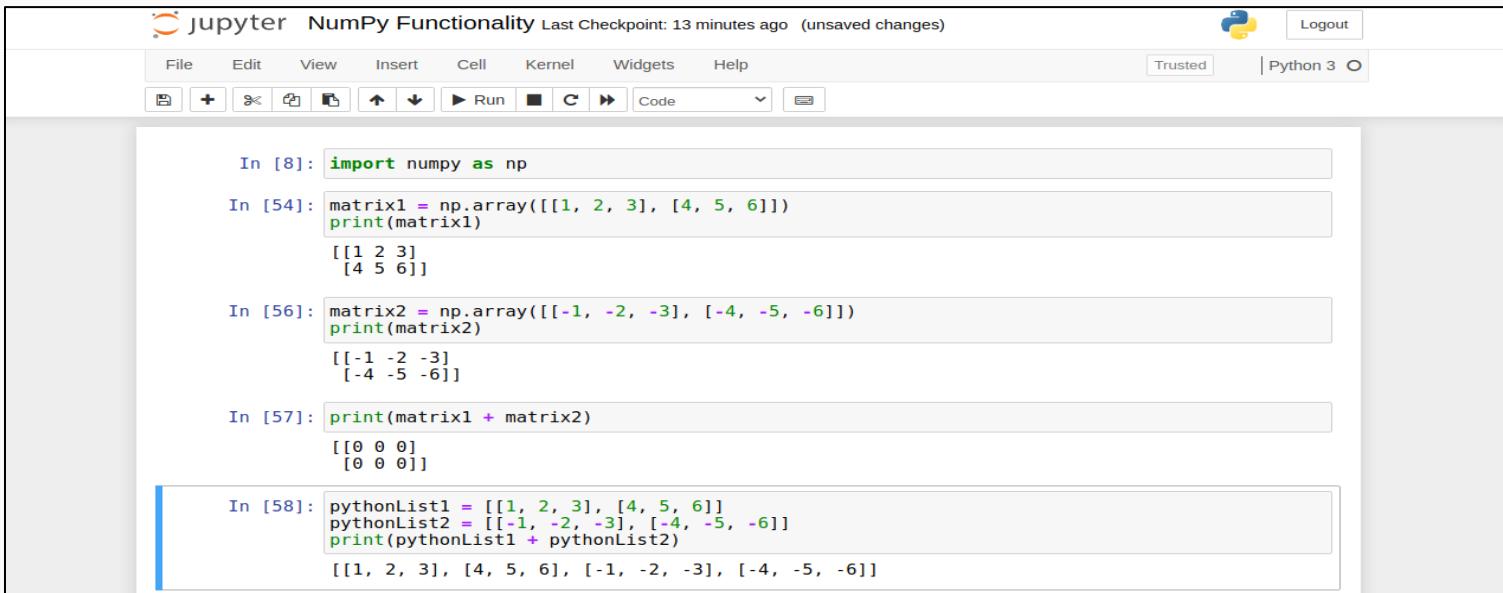
- In [41]:** `matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [52]:** `print(matrix.T)`
Output:
`[[1 4]
 [2 5]
 [3 6]]`
- In [53]:** `pythonList = [[1, 2, 3], [4, 5, 6]]
print(pythonList.T)`
Output:

```
-----  
AttributeError                                 Traceback (most recent call last)  
<ipython-input-53-f8b6fc22b132> in <module>  
      1 pythonList = [[1, 2, 3], [4, 5, 6]]  
----> 2 print(pythonList.T)  
  
AttributeError: 'list' object has no attribute 'T'
```

Element-wise Operations (1/4)

Addition

- We can add the elements of two NumPy arrays simply by using (+) operator.
- Each element of the first array is added to the corresponding element of the second array.
- Note that adding two Python lists using plus (+) operator is not possible. Instead the lists are concatenated if we use (+) operator.



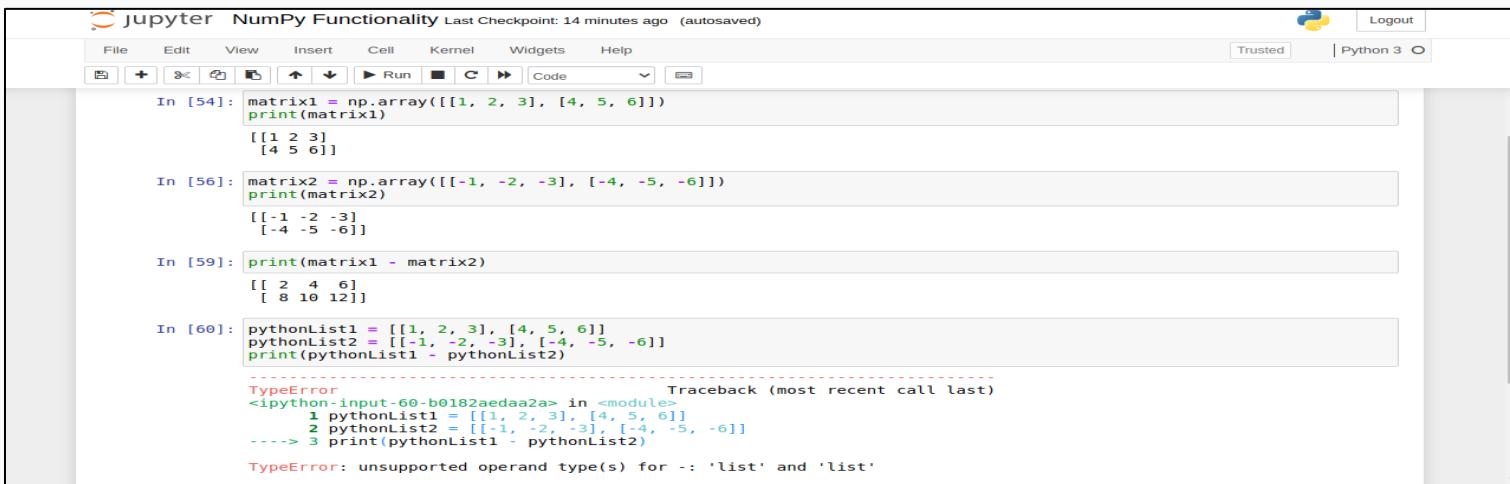
The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 13 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code cells are as follows:

```
In [8]: import numpy as np
In [54]: matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)
[[1 2 3]
 [4 5 6]]
In [56]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]
In [57]: print(matrix1 + matrix2)
[[0 0 0]
 [0 0 0]]
In [58]: pythonList1 = [[1, 2, 3], [4, 5, 6]]
pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
print(pythonList1 + pythonList2)
[[1, 2, 3], [4, 5, 6], [-1, -2, -3], [-4, -5, -6]]
```

Element-wise Operations (2/4)

Subtraction

- We can subtract the elements of two NumPy arrays simply by using (-) operator.
- Each element of the second array is subtracted from the corresponding element of the first array.
- Note that subtracting the elements of two Python lists using (-) operator will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality Last Checkpoint: 14 minutes ago (autosaved)". The notebook has a "Trusted" status and is using "Python 3".

Cell In [54]:

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)
[[1 2 3]
 [4 5 6]]
```

Cell In [56]:

```
matrix2 = np.array([-1, -2, -3], [-4, -5, -6])
print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]
```

Cell In [59]:

```
print(matrix1 - matrix2)
[[ 2  4  6]
 [ 8 10 12]]
```

Cell In [60]:

```
pythonList1 = [[1, 2, 3], [4, 5, 6]]
pythonList2 = [[1, 2, 3], [4, 5, 6]]
print(pythonList1 - pythonList2)
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-60-b0182aedaa2a> in <module>  
      1 pythonList1 = [[1, 2, 3], [4, 5, 6]]  
      2 pythonList2 = [[1, 2, 3], [4, 5, 6]]  
----> 3 print(pythonList1 - pythonList2)  
  
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Element-wise Operations (3/4)

Multiplication

- We can multiply the elements of two NumPy arrays simply by using (*) operator.
- Each element of the first array is multiplied by the corresponding element of the second array.
- Note that multiplying the elements of two Python lists using (*) operator will result in an error.

The screenshot shows a Jupyter Notebook interface with the title "Jupyter NumPy Functionality". The notebook has a "Python 3" kernel selected. The code cells show the following operations:

- In [54]:

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)
[[1 2 3]
 [4 5 6]]
```
- In [56]:

```
matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]
```
- In [62]:

```
print(matrix1 * matrix2)
[[-1 -4 -9]
 [-16 -25 -36]]
```
- In [61]:

```
pythonList1 = [[1, 2, 3], [4, 5, 6]]
pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
print(pythonList1 * pythonList2)
```

The last cell (In [61]) results in a `TypeError`:

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-61-e9ba5f4c2fff> in <module>  
      1 pythonList1 = [[1, 2, 3], [4, 5, 6]]  
      2 pythonList2 = [[-1, -2, -3], [-4, -5, -6]]  
----> 3 print(pythonList1 * pythonList2)  
  
TypeError: can't multiply sequence by non-int of type 'list'
```

Element-wise Operations (4/4)

Division

- We can divide the elements of two NumPy arrays simply by using (/) operator.
- Each element of the first array is divided by the corresponding element of the second array.
- Note that dividing the elements of two Python lists using (/) operator will result in an error.

The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality". The notebook has a toolbar with File, Edit, View, Insert, Cell, Kernel, Widgets, and Help buttons. A status bar at the top right indicates "Trusted" and "Python 3". The code cells are numbered 54 through 69:

- In [54]:

```
matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)
[[1 2 3]
 [4 5 6]]
```
- In [56]:

```
matrix2 = np.array([[-1, -2, -3], [-4, -5, -6]])
print(matrix2)
[[-1 -2 -3]
 [-4 -5 -6]]
```
- In [67]:

```
print(matrix1 / matrix2)
[[-1. -1. -1.]
 [-1. -1. -1.]]
```
- In [68]:

```
print(matrix1 // matrix2)
[[-1 -1 -1]
 [-1 -1 -1]]
```
- In [69]:

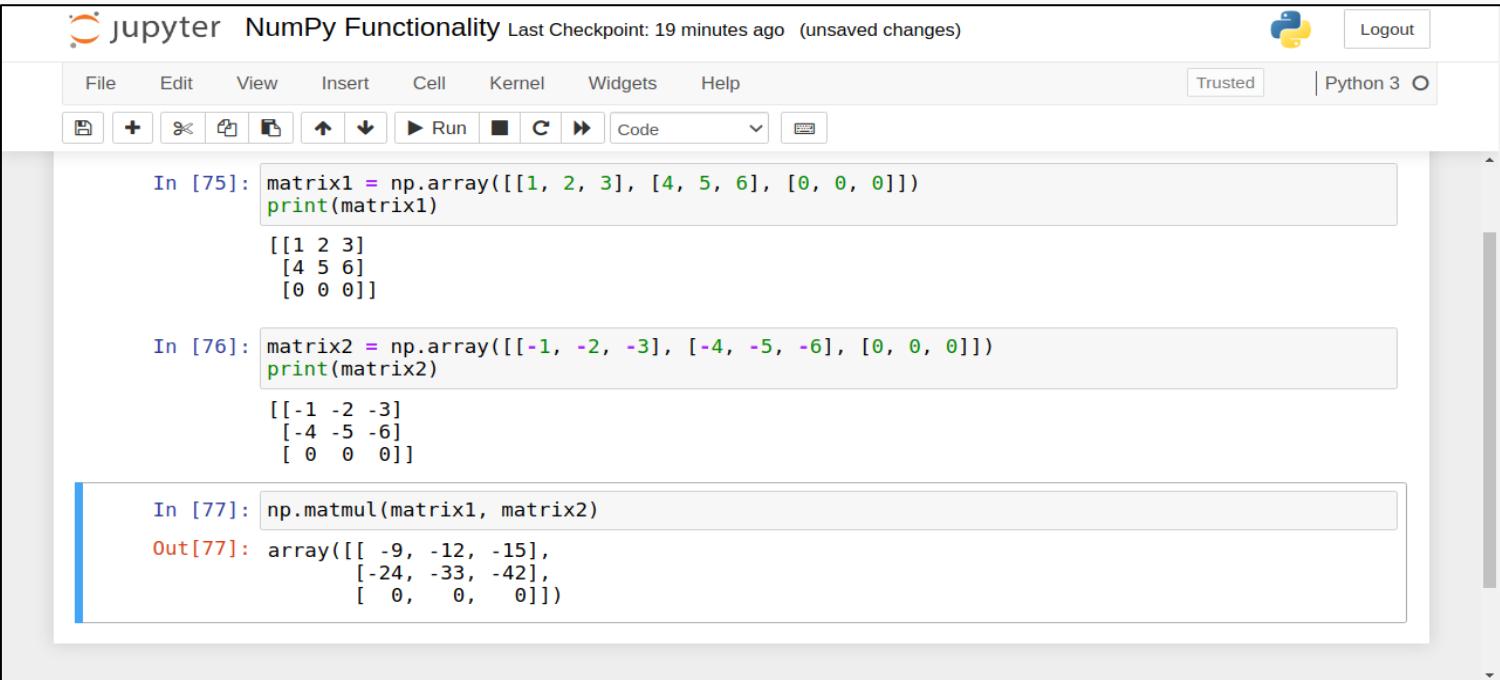
```
pythonList1 = [[1, 2, 3], [4, 5, 6]]
pythonList2 = [[-1, -2, -3], [-4, -5, -6]]
print(pythonList1 / pythonList2)

TypeError: unsupported operand type(s) for /: 'list' and 'list'
```

The last cell (In [69]) shows a `TypeError` because it attempts to divide two Python lists, which is not supported.

Matrix Multiplication

- Apart from elementwise multiplication, NumPy also provides us with a built-in function to compute the matrix multiplication of two arrays.
- We use the `.matmul()` function inside the NumPy library for matrix multiplication of two arrays.



The screenshot shows a Jupyter Notebook interface with the title "Jupyter NumPy Functionality". The notebook has a "Python 3" kernel selected. The code cell In [75] contains:

```
In [75]: matrix1 = np.array([[1, 2, 3], [4, 5, 6], [0, 0, 0]])
print(matrix1)
```

The output of this cell is:

```
[[1 2 3]
 [4 5 6]
 [0 0 0]]
```

The code cell In [76] contains:

```
In [76]: matrix2 = np.array([[-1, -2, -3], [-4, -5, -6], [0, 0, 0]])
print(matrix2)
```

The output of this cell is:

```
[[-1 -2 -3]
 [-4 -5 -6]
 [ 0  0  0]]
```

The code cell In [77] contains:

```
In [77]: np.matmul(matrix1, matrix2)
```

The output of this cell is:

```
Out[77]: array([[-9, -12, -15],
 [-24, -33, -42],
 [ 0,  0,  0]])
```

Quiz Time

1. Which of the following statements is correct?

- a) * is used for array multiplication
- b) * is used for element-wise multiplication
- c) * is used for power

Quiz Time

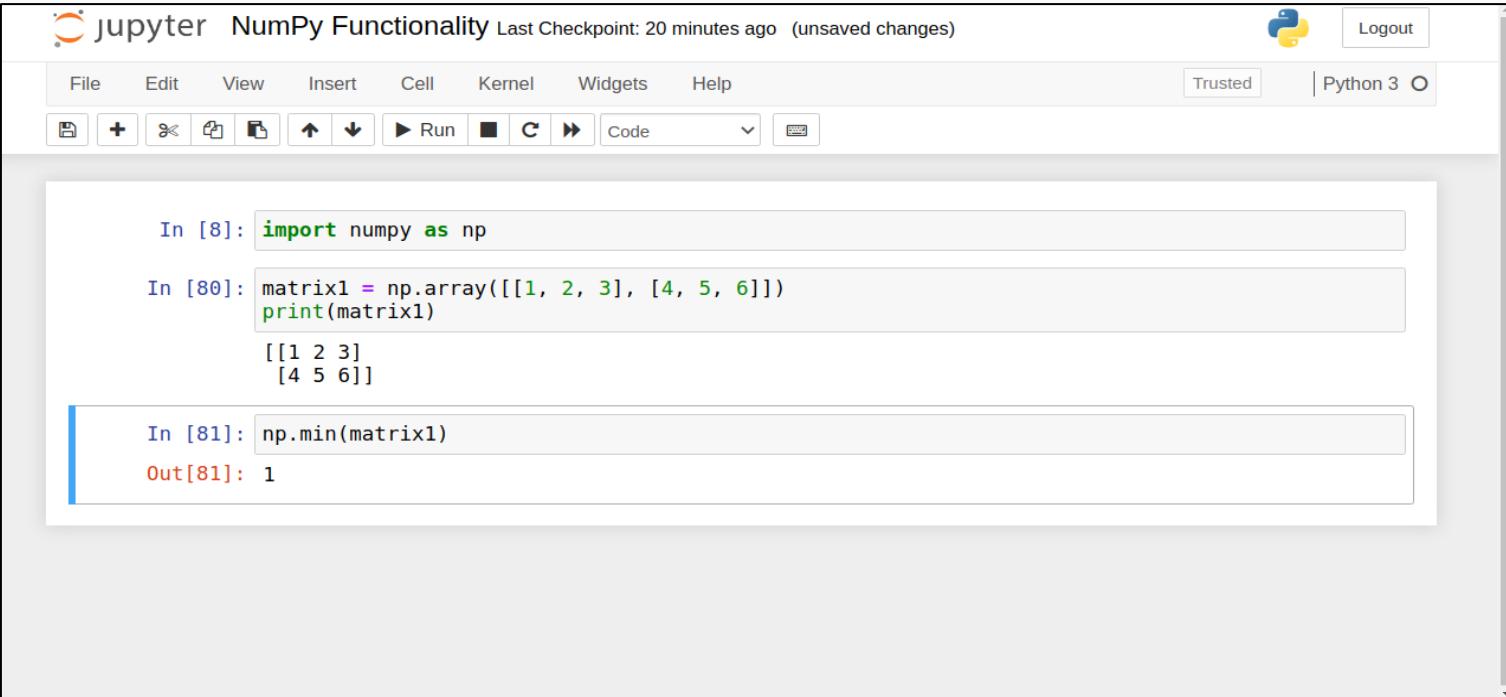
1. Which of the following statements is correct?

- a) * is used for array multiplication
- b) * is used for element-wise multiplication
- c) * is used for power

Statistics (1/7)

.min()

- .min() function gives us the minimum value in a NumPy array.
- This function can also be applied on Python lists.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 20 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are standard notebook controls for cell selection, running, and saving.

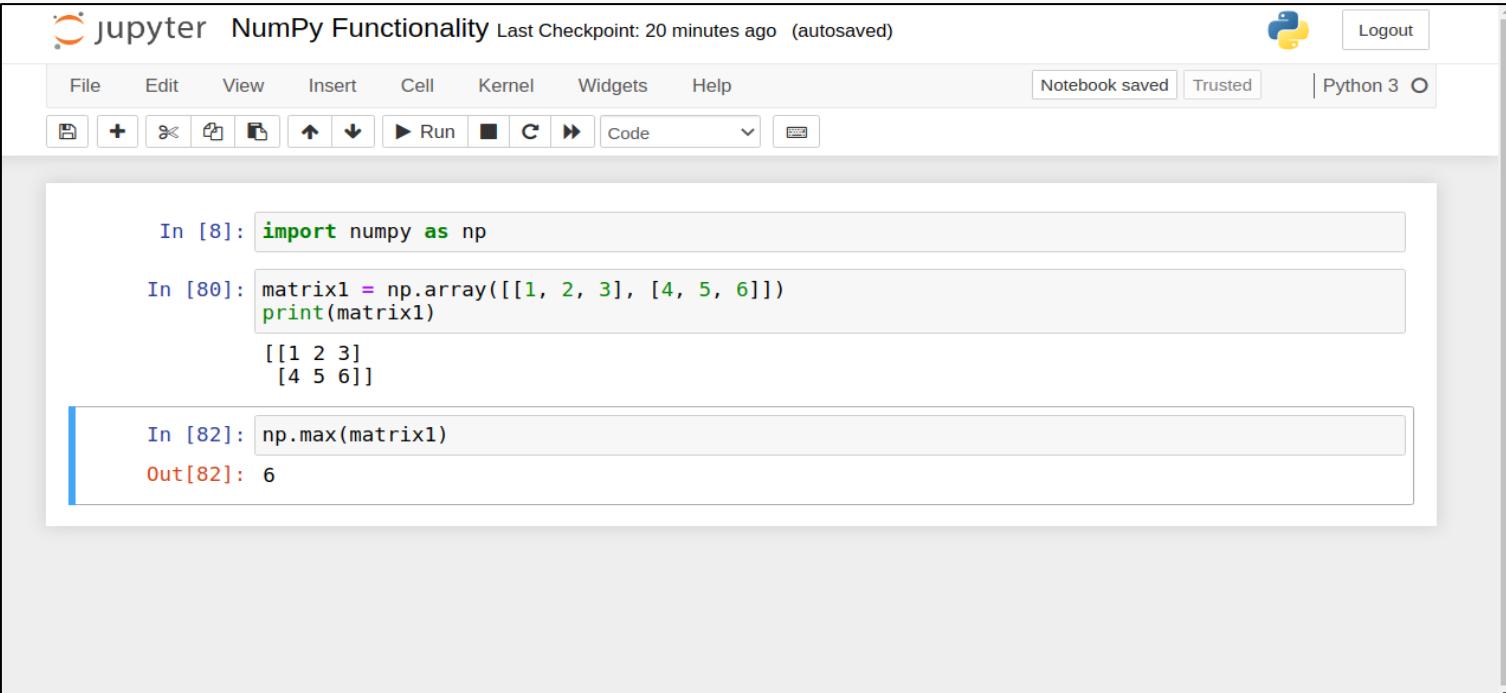
The notebook has three cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [81]: `np.min(matrix1)`
Out[81]: 1

Statistics (2/7)

.max()

- .max() function gives us the maximum value in a NumPy array.
- This function can also be applied on Python lists.



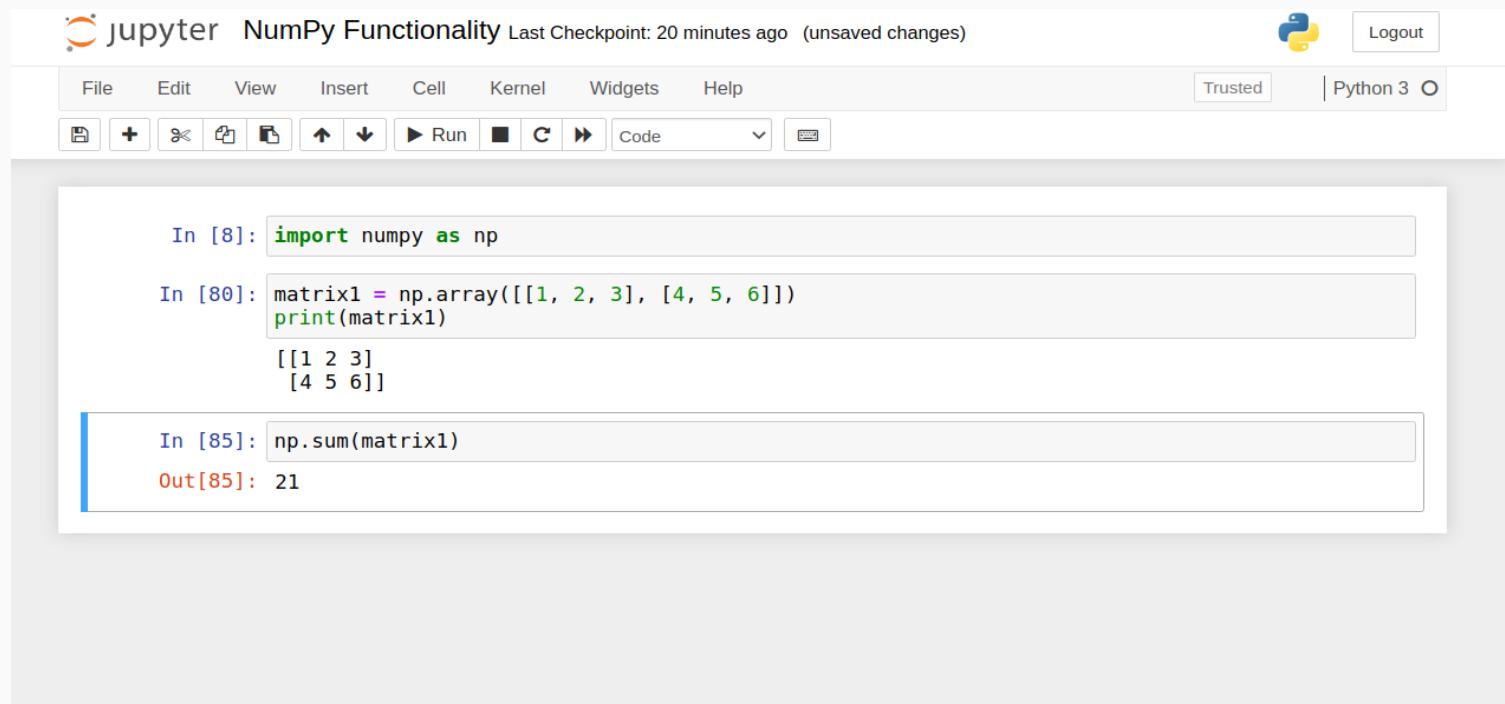
The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and a status bar indicating "Last Checkpoint: 20 minutes ago (autosaved)". The toolbar includes standard notebook operations like File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 logo. Below the toolbar, a code editor displays three cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [82]: `np.max(matrix1)`
Output:
`6`

Statistics (3/7)

.sum()

- .sum() function gives us the sum of all the values in a NumPy array.
- This function can also be applied on Python lists.



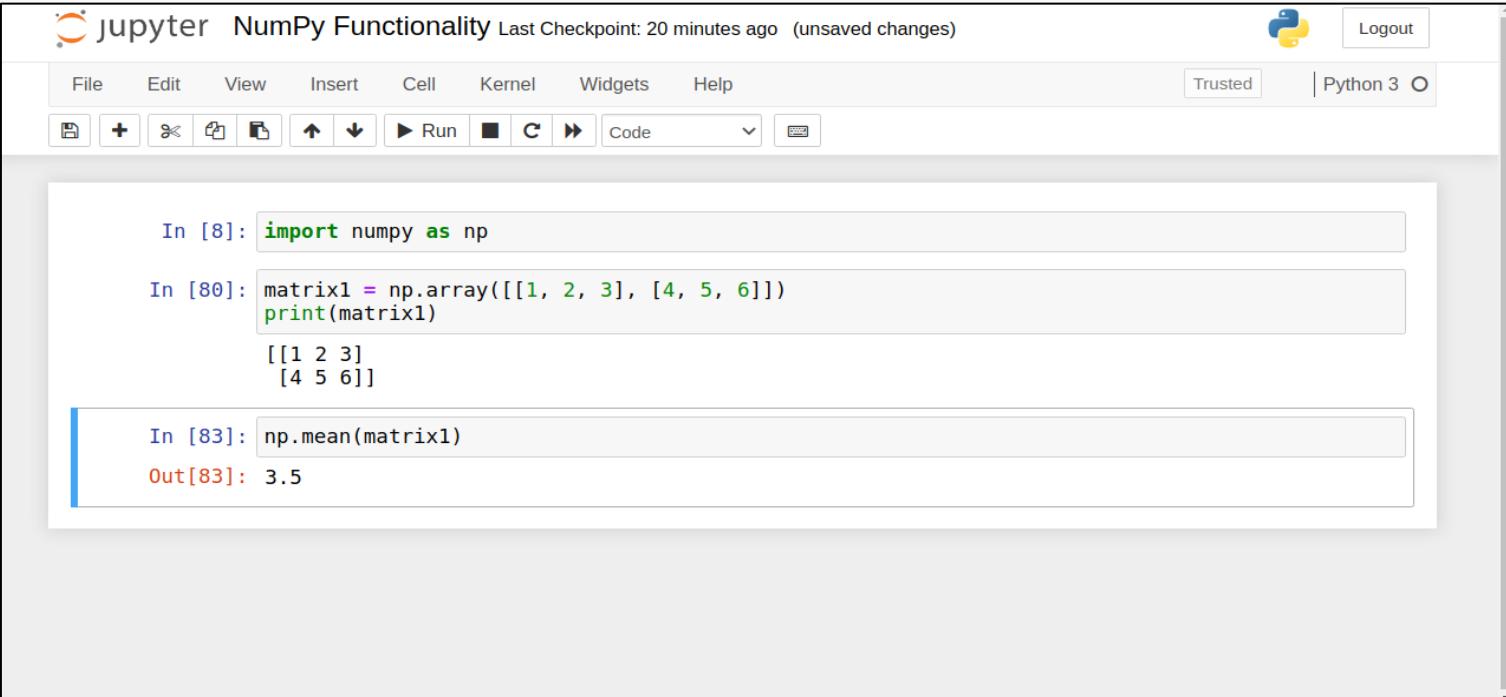
The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 20 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. The code editor contains three cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
[[1 2 3]
[4 5 6]]
- In [85]: `np.sum(matrix1)`
Out[85]: 21

Statistics (4/7)

.mean()

- .mean() function gives us the mean of all the values in a NumPy array.
- This function can also be applied on Python lists.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 20 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are standard notebook controls for cell selection, running, and code input. The main area displays three code cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
[[1 2 3]
 [4 5 6]]
- In [83]: `np.mean(matrix1)`
Out[83]: 3.5

Statistics (5/7)

.std()

- .std() function gives us the standard deviation of a NumPy array.
- This function can also be applied on Python lists.



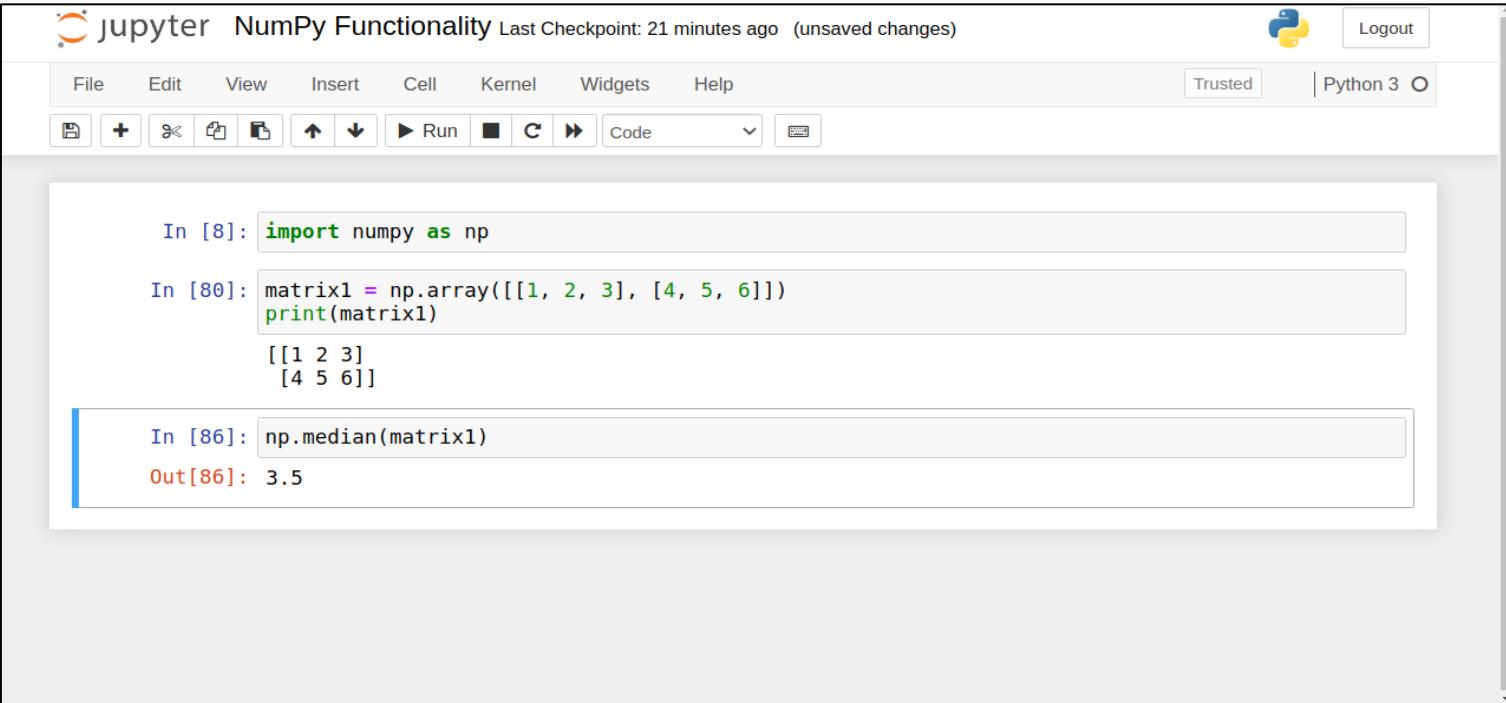
The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 20 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3. Below the toolbar are buttons for cell operations like Run, Cell, and Kernel. The notebook area contains three code cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [84]: `np.std(matrix1)`
Out[84]: `1.707825127659933`

Statistics (6/7)

.median()

- .median() function gives us the median of a NumPy array.
- This function can also be applied on Python lists.



The screenshot shows a Jupyter Notebook interface with the title "jupyter NumPy Functionality" and "Last Checkpoint: 21 minutes ago (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Run, and Cell options. The Python kernel is set to "Python 3".

The notebook contains the following code cells:

- In [8]: `import numpy as np`
- In [80]: `matrix1 = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix1)`
Output:
`[[1 2 3]
 [4 5 6]]`
- In [86]: `np.median(matrix1)`
Out[86]: 3.5

Statistics (7/7)

- A detailed list of NumPy statistical functions can be found at the link below;
<https://numpy.org/doc/stable/reference/routines.statistics.html>

Resources

- https://www.w3schools.com/python/numpy/numpy_intro.asp
- <https://www.tutorialspoint.com/numpy/index.htm>

PANDAS DATA STRUCTURES

Data Structures in Pandas

- Pandas has two main data structures;
 - DataFrame, which is two dimensional
 - Series, which is one dimensional



	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813



0	70
1	120
2	70
3	50
4	110
5	110
6	110
7	130
8	90
9	90

Name: calories, dtype: int64

What is Pandas DataFrame

- Pandas provides a two dimensional data structure called DataFrame.
- A row is represented by row labels, also called index, which may be numerical or strings.
- A column is represented by column labels which may be numerical or strings.
- Following DataFrame contains 10 rows (0-9) and 5 columns (name, calories, protein, vitamins, rating)

The diagram illustrates the structure of a Pandas DataFrame. An orange bracket on the left side, labeled "index", spans the first column from top to bottom, indicating that each row has an index. An orange box on the right side, labeled "column labels", spans the entire width of the table, centered over the column headers (name, calories, protein, vitamins, rating), indicating that each column has a label.

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

What is Pandas Series

- A Series in Pandas is a one dimensional data structure.
- It consists of a single row or column.
- Following Series contains 10 rows (0-9) and 1 column called calories.

The diagram illustrates a Pandas Series as a vertical list of 10 data points, each consisting of an index (0-9) and a value (calories). The series is enclosed in a light gray box. An orange bracket on the left is labeled 'index' and spans the indices 0 through 9. A green bracket on the right is labeled 'column values' and spans the values 70, 120, 70, 50, 110, 110, 110, 130, 90, and 90. Below the series, the text 'Name: calories, dtype: int64' is displayed. Two orange arrows point from the text 'column name' and 'column data type' to the word 'calories' and 'int64' respectively.

0	70
1	120
2	70
3	50
4	110
5	110
6	110
7	130
8	90
9	90

Name: calories, dtype: int64

DataFrame and Series

- A Pandas DataFrame is just a collection of one or more Series.
- The Series in the previous example was extracted from the DataFrame.

DataFrame

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

Series

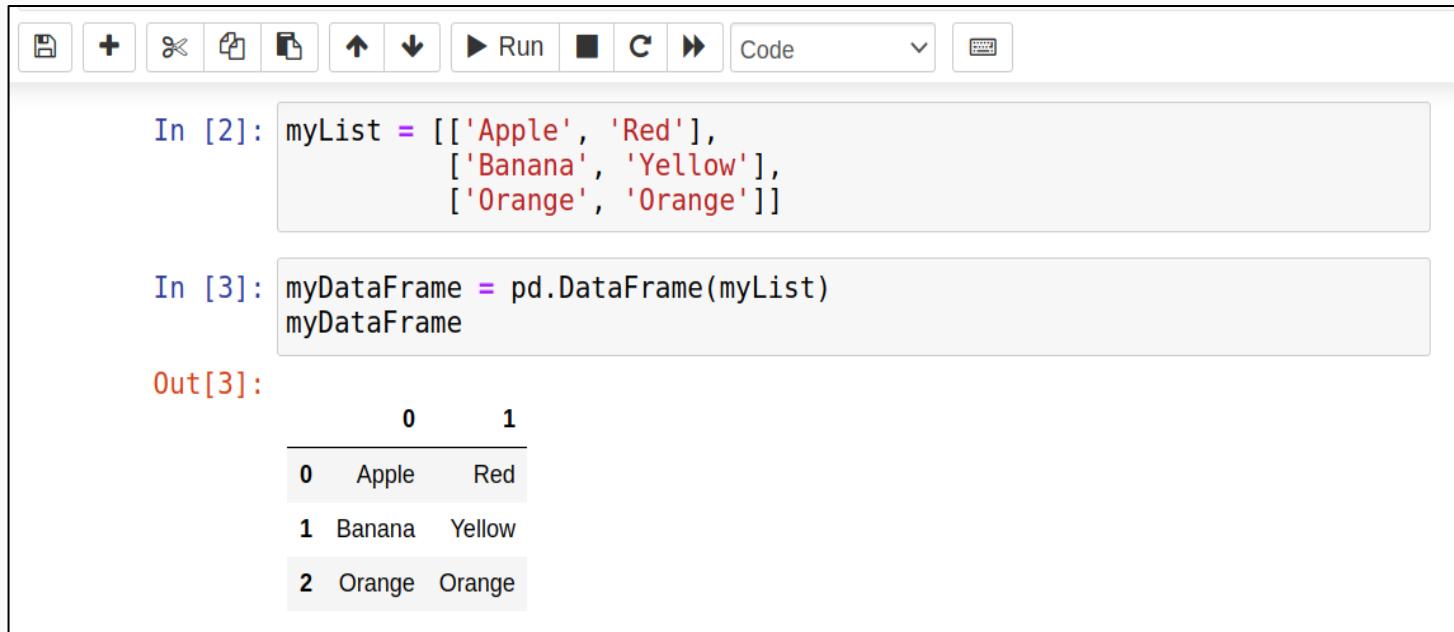
0	70
1	120
2	70
3	50
4	110
5	110
6	110
7	130
8	90
9	90

Name: calories, dtype: int64

Identical

Creating a DataFrame Using Lists (1/3)

- We can create a DataFrame using lists.
- We pass the list as an argument to the pandas.DataFrame() function which returns us a DataFrame.
- Pandas automatically assigns numerical row labels to each row of the DataFrame.
- Since, we did not provide column labels, Pandas automatically assigned numerical column labels to each column as well.



The screenshot shows a Jupyter Notebook interface with the following content:

In [2]:

```
myList = [['Apple', 'Red'],
          ['Banana', 'Yellow'],
          ['Orange', 'Orange']]
```

In [3]:

```
myDataFrame = pd.DataFrame(myList)
myDataFrame
```

Out[3]:

	0	1
0	Apple	Red
1	Banana	Yellow
2	Orange	Orange

Creating a DataFrame Using Lists (2/3)

- Let's create another DataFrame using the same list, but this time with custom column labels.
- Pandas.DataFrame() takes another optional argument called 'columns' which takes a list of custom column names to be set as columns' labels.

```
In [2]: myList = [['Apple', 'Red'],
                 ['Banana', 'Yellow'],
                 ['Orange', 'Orange']]
```

```
In [4]: myDataFrame = pd.DataFrame(myList, columns=['Fruit', 'Color'])
myDataFrame
```

Out[4]:

	Fruit	Color
0	Apple	Red
1	Banana	Yellow
2	Orange	Orange

Creating a DataFrame Using Lists (3/3)

- As we know that a NumPy Array is similar to a Python list with added functionality, we can also convert a NumPy Array to a DataFrame using the same method.

```
In [9]: myList = np.array([[0, 1],  
                         [2, 3],  
                         [4, 5]])
```

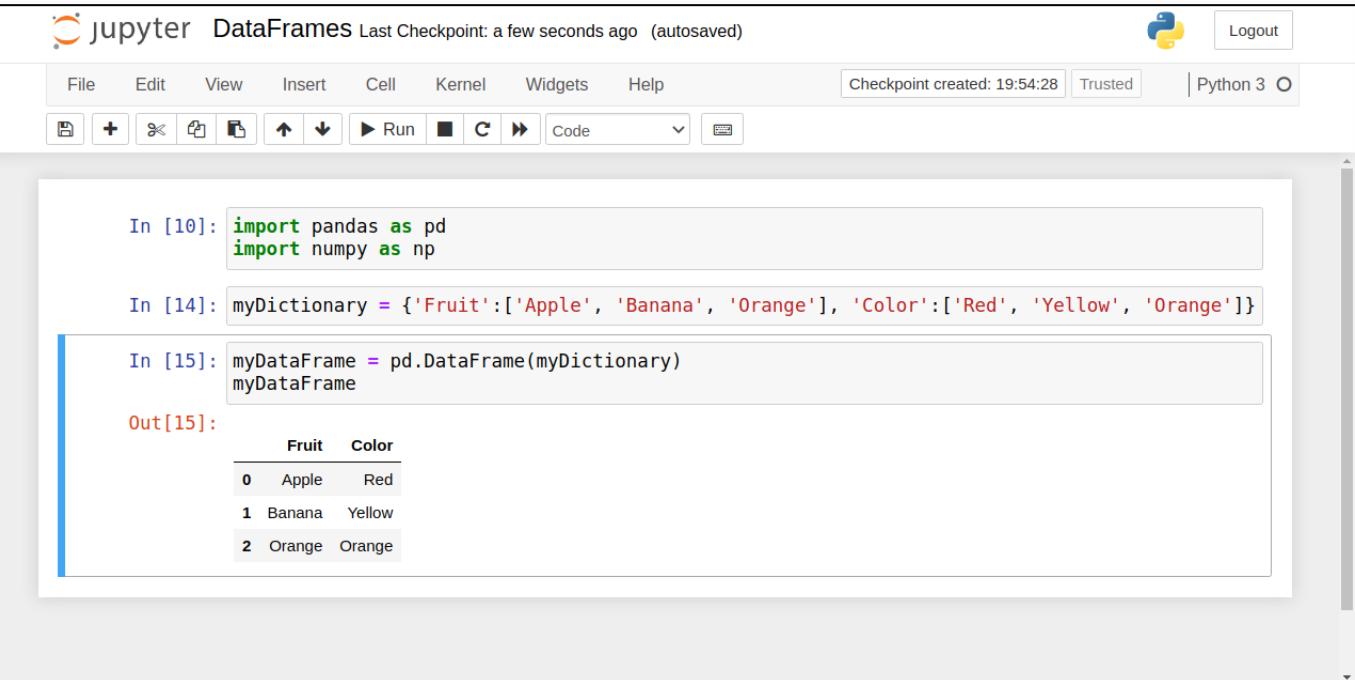
```
In [10]: myDataFrame = pd.DataFrame(myList, columns=['even', 'odd'])  
myDataFrame
```

Out[10]:

	even	odd
0	0	1
1	2	3
2	4	5

Creating a DataFrame Using Dictionary

- We can also pass a dictionary to the pandas.DataFrame() function to create a DataFrame.
- Each key of the array should have a list of one or more values associated with it.
- The keys of the dictionary become column labels.
- Pandas automatically assigns numerical row labels to each row of the DataFrame.



The screenshot shows a Jupyter Notebook interface with the following content:

```
In [10]: import pandas as pd
import numpy as np

In [14]: myDictionary = {'Fruit':['Apple', 'Banana', 'Orange'], 'Color':['Red', 'Yellow', 'Orange']}

In [15]: myDataFrame = pd.DataFrame(myDictionary)
myDataFrame
```

Out[15]:

	Fruit	Color
0	Apple	Red
1	Banana	Yellow
2	Orange	Orange

Loading csv File as a DataFrame

- We can also load a csv (comma separated values) file as a DataFrame in Pandas using the `pandas.read_csv()` function.
- Each value of the first row of the csv file becomes a column label.
- Pandas automatically assigns numerical row labels to each row of the DataFrame.

```
In [11]: df = pd.read_csv('cereals.csv')
df
```

Out[11]:

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

Changing the Index Column

- We can set one of the existing columns as the new index column of the DataFrame using `.set_index()` function.

```
In [13]: df
```

```
Out[13]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

```
In [14]: df.set_index('name')
```

```
Out[14]:
```

	calories	protein	vitamins	rating
name				
100% Bran	70	4	25	68.402973
100% Natural Bran	120	3	0	33.983679
All-Bran	70	4	25	59.425505
All-Bran with Extra Fiber	50	4	25	93.704912
Almond Delight	110	2	25	34.384843

Inplace (1/2)

- Remember that most of the functions in Pandas do not change the original DataFrame.
- In the previous section we changed the index column of our DataFrame. If we print our DataFrame again, we see that the original DataFrame is unchanged.

```
In [15]: df
```

```
Out[15]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

Inplace (2/2)

- We can use the inplace argument to make changes to the original DataFrame.
- In the following example we use the .set_index() function to change the index of our DataFrame, and set inplace = True.
- As shown in the figure, our original DataFrame has been changed.

```
In [16]: df.set_index('name', inplace=True)
```

```
In [17]: df
```

Out[17]:

		calories	protein	vitamins	rating
	name				
	100% Bran	70	4	25	68.402973
	100% Natural Bran	120	3	0	33.983679
	All-Bran	70	4	25	59.425505
	All-Bran with Extra Fiber	50	4	25	93.704912
	Almond Delight	110	2	25	34.384843

Examining the Data (1/2)

head()

- head() function gives us the **first** 5 rows of the DataFrame/Series by default.
- To get more rows, we can pass the desired number as an argument to the head() function.

		name	calories	protein	vitamins	rating
0		100% Bran	70	4	25	68.402973
1		100% Natural Bran	120	3	0	33.983679
2		All-Bran	70	4	25	59.425505
3		All-Bran with Extra Fiber	50	4	25	93.704912
4		Almond Delight	110	2	25	34.384843
5		Apple Cinnamon Cheerios	110	2	25	29.509541
6		Apple Jacks	110	2	25	33.174094

Examining the Data (2/2)

`tail()`

- `tail()` function gives us the **last** 5 rows of the DataFrame/Series by default.
- To get more rows, we can pass the desired number as an argument to the `tail()` function.

In [22]: `df.tail(7)`

Out[22]:

		name	calories	protein	vitamins	rating
3	All-Bran with Extra Fiber		50	4	25	93.704912
4	Almond Delight		110	2	25	34.384843
5	Apple Cinnamon Cheerios		110	2	25	29.509541
6	Apple Jacks		110	2	25	33.174094
7	Basic 4		130	3	25	37.038562
8	Bran Chex		90	2	25	49.120253
9	Bran Flakes		90	3	25	53.313813

Statistical Summary

- We can use the `describe()` function to get a quick statistical summary of each column of the DataFrame.

```
In [25]: df.describe()
```

```
Out[25]:
```

	calories	protein	vitamins	rating
count	5.000000	5.000000	5.000000	5.000000
mean	84.000000	3.400000	20.000000	57.980382
std	29.664794	0.894427	11.18034	25.097570
min	50.000000	2.000000	0.000000	33.983679
25%	70.000000	3.000000	25.000000	34.384843
50%	70.000000	4.000000	25.000000	59.425505
75%	110.000000	4.000000	25.000000	68.402973
max	120.000000	4.000000	25.000000	93.704912

[] Operator for Row Slicing (1/2)

- We can use the brackets ([]) operator to slice rows of the DataFrame.
- We pass a start index (inclusive) and an end index (exclusive) to the bracket operator ([]) to slice the rows of the DataFrame.

```
In [26]: df[1:4]
```

```
Out[26]:
```

		name	calories	protein	vitamins	rating
1	100% Natural Bran		120	3	0	33.983679
2	All-Bran		70	4	25	59.425505
3	All-Bran with Extra Fiber		50	4	25	93.704912

[] Operator for Row Slicing (2/2)

- Remember that [] operator works on row position and not row labels.
- For example, in the following case row labels are strings. But we pass positions of the rows that we want to slice.

```
In [30]: df
```

Out[30]:

	name	calories	protein	vitamins	rating
	100% Bran	70	4	25	68.402973
	100% Natural Bran	120	3	0	33.983679
	All-Bran	70	4	25	59.425505
	All-Bran with Extra Fiber	50	4	25	93.704912
	Almond Delight	110	2	25	34.384843

```
In [31]: df[1:4]
```

Out[31]:

	name	calories	protein	vitamins	rating
	100% Natural Bran	120	3	0	33.983679
	All-Bran	70	4	25	59.425505
	All-Bran with Extra Fiber	50	4	25	93.704912

Quiz Time

1. Consider the given DataFrame called df. Which of the following will give us rows 5-10 of the DataFrame?
 - a) df[5:10]
 - b) df[5:11]
 - c) df[4:10]
 - d) df[4:11]

Quiz Time

1. Consider the given DataFrame called df. Which of the following will give us rows 5-10 of the DataFrame?
 - a) df[5:10]
 - b) df[5:11]
 - c) df[4:10]
 - d) df[4:11]

[] Operator for Column Indexing

- We can also use the brackets ([]) operator to index column of the DataFrame.
- Indexing a single column returns a Series.
- Indexing a list of columns returns a DataFrame.
- Remember that for indexing columns, we pass their labels to the [] operator and not their positions.

```
In [34]: df[['name', 'rating']]
```

```
Out[34]:
```

	name	rating
0	100% Bran	68.402973
1	100% Natural Bran	33.983679
2	All-Bran	59.425505
3	All-Bran with Extra Fiber	93.704912
4	Almond Delight	34.384843

Boolean List

- We can also pass a list of booleans to the [] operator.
- We get all the rows of the DataFrame for which the corresponding element in the list is True.
- Rows of the DataFrame for which the corresponding element in the list is False are ignored.
- Note: Original DataFrame remains unchanged.

```
In [71]: df
```

```
Out[71]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

```
In [75]: thirdRow = [False, False, True, False, False]
df[thirdRow]
```

```
Out[75]:
```

	name	calories	protein	vitamins	rating
2	All-Bran	70	4	25	59.425505

Filtering Rows (1/3)

- We can also use the [] operator to apply conditions on one or more columns of the DataFrame.
- Rows of the DataFrame which satisfy those conditions are filtered out.

```
In [36]: condition = df['calories'] > 70  
df[condition]
```

Out[36]:

	name	calories	protein	vitamins	rating
1	100% Natural Bran	120	3	0	33.983679
4	Almond Delight	110	2	25	34.384843

```
In [37]: df[ df['calories'] > 70]
```

Out[37]:

	name	calories	protein	vitamins	rating
1	100% Natural Bran	120	3	0	33.983679
4	Almond Delight	110	2	25	34.384843

Filtering Rows (2/3)

and (&)

- We can also group conditions using the and operator.
- Symbol for and in pandas is `&`. It works the same way as `and` in Python.
- Note: Each condition should be in parentheses.

```
In [38]: df[ (df['calories'] > 70) & (df['protein'] < 4) ]
```

```
Out[38]:
```

		name	calories	protein	vitamins	rating
1	100% Natural Bran		120	3	0	33.983679
4	Almond Delight		110	2	25	34.384843

Filtering Rows (3/3)

or (|)

- We can also group conditions using the or operator.
- Symbol for and in pandas is | It works the same way as `or` in Python.
- Note: Each condition should be in parentheses.

```
In [39]: df[ (df['calories'] > 70) | (df['protein'] > 3) ]
```

Out[39]:

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

loc (1/4)

Indexing

- loc is used to index/slice a group of rows and columns based on their labels.
- The first argument is the row label and the second argument is the column label.
- In the following example we index the first row and the first column.

```
In [79]: df
```

```
Out[79]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

```
In [88]: df.loc[0, 'name']
```

```
Out[88]: '100% Bran'
```

loc (2/4)

Indexing

- If we pass a list of row and column labels, we get a DataFrame.
- In the following example, we index first row and first column, but we pass the labels as lists. We get a DataFrame.

```
In [79]: df
```

```
Out[79]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843


```
In [89]: df.loc[[0], ['name']]
```

```
Out[89]:
```

	name
0	100% Bran

loc (3/4)

Slicing

- We can also slice rows and/or columns using the loc method.
- Both the start and stop index of a slice with loc are inclusive.
- In the following example, we slice the first 5 rows and the first 3 columns of the DataFrame. The result is a DataFrame.

```
In [41]: df
```

```
Out[41]:
```

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813

```
In [42]: df.loc[0:4, 'name':'protein']
```

```
Out[42]:
```

	name	calories	protein
0	100% Bran	70	4
1	100% Natural Bran	120	3
2	All-Bran	70	4
3	All-Bran with Extra Fiber	50	4
4	Almond Delight	110	2

loc (4/4)

Indexing and Slicing

- We can index and slice simultaneously as well.
- In the following example we index rows and slice columns. The opposite is also possible.

```
In [43]: df.loc[[5, 8], 'name':'protein']
```

```
Out[43]:
```

		name	calories	protein
	5	Apple Cinnamon Cheerios	110	2
	8	Bran Chex	90	2

Quiz Time

1. Consider the given DataFrame called df. What will be the result of the following loc command?

```
df.loc[[0, 1], ['name']]
```

- a) DataFrame
- b) Series
- c) Cell

Quiz Time

1. Consider the given DataFrame called df. What will be the result of the following loc command?

```
df.loc[[0, 1], ['name']]
```

- a) DataFrame
- b) Series
- c) Cell

iloc (1/4)

Indexing

- iloc is used to index/slice a group of rows and columns.
- iloc takes row and column positions as arguments and not their labels.
- The first argument is the row position and the second argument is the column position.
- In the following example we index the 10th row and the third column. The result is a Series.

In [41]:

```
df
```

Out[41]:

		name	calories	protein	vitamins	rating
0		100% Bran	70	4	25	68.402973
1		100% Natural Bran	120	3	0	33.983679
2		All-Bran	70	4	25	59.425505
3		All-Bran with Extra Fiber	50	4	25	93.704912
4		Almond Delight	110	2	25	34.384843
5		Apple Cinnamon Cheerios	110	2	25	29.509541
6		Apple Jacks	110	2	25	33.174094
7		Basic 4	130	3	25	37.038562
8		Bran Chex	90	2	25	49.120253
9		Bran Flakes	90	3	25	53.313813

In [44]:

```
df.iloc[9, 2]
```

Out[44]:

```
3
```

iloc (2/4)

Indexing

- If we pass a list of row and column positions, we get a DataFrame.
- In the following example, we index 10th row and third column, but we pass the positions as lists. We get a DataFrame.

```
In [45]: df.iloc[[9], [2]]
```

```
Out[45]:
```

protein

9	3
---	---

iloc (3/4)

Slicing

- We can also slice rows and/or columns using the iloc method.
- We provide row and column positions for slicing using iloc.
- The start index of a slice with iloc is inclusive. However, the end index is exclusive.
- In the following example, we slice the first 5 rows and the first 3 columns of the DataFrame. The result is a DataFrame.

```
In [46]: df.iloc[0:5, 0:3]
```

```
Out[46]:
```

	name	calories	protein
0	100% Bran	70	4
1	100% Natural Bran	120	3
2	All-Bran	70	4
3	All-Bran with Extra Fiber	50	4
4	Almond Delight	110	2

iloc (4/4)

Indexing and Slicing

- We can index and slice simultaneously as well.
- In the following example we index rows and slice columns. The opposite is also possible.

```
In [47]: df.iloc[[0, 2, 4], 0:3]
```

```
Out[47]:
```

	name	calories	protein
0	100% Bran	70	4
2	All-Bran	70	4
4	Almond Delight	110	2

Quiz Time

1. Consider the given DataFrame called df. What will be the result of the following iloc command?
`df.loc[[0, 2], [2]]`
 - a) DataFrame
 - b) Series
 - c) Cell
1. The stop index in iloc slice is inclusive. Is this statement True or False?
 - a) True
 - b) False

Quiz Time

1. Consider the given DataFrame called df. What will be the result of the following iloc command?
`df.loc[[0, 2], [2]]`
 - a) DataFrame
 - b) Series
 - c) Cell
1. The stop index in iloc slice is inclusive. Is this statement True or False?
 - a) True
 - b) False

Adding and Deleting Rows and Columns (1/4)

Adding Rows

- We can add more rows to our DataFrame using the loc method.
- If the row label does not exist, a new row with the specified label will be added at the end of the row.

```
In [24]: df
```

```
Out[24]:
```

		name	calories	protein	vitamins	rating
0		100% Bran	70	4	25	68.402973
1		100% Natural Bran	120	3	0	33.983679
2		All-Bran	70	4	25	59.425505
3		All-Bran with Extra Fiber	50	4	25	93.704912
4		Almond Delight	110	2	25	34.384843

```
In [68]: df.loc[6] = ['Trix', 110, 1, 25, 27.753301]  
df
```

```
Out[68]:
```

		name	calories	protein	vitamins	rating
0		100% Bran	70	4	25	68.402973
1		100% Natural Bran	120	3	0	33.983679
2		All-Bran	70	4	25	59.425505
3		All-Bran with Extra Fiber	50	4	25	93.704912
4		Almond Delight	110	2	25	34.384843
6		Trix	110	1	25	27.753301

Adding and Deleting Rows and Columns (2/4)

Deleting Rows

- We can delete rows from the DataFrame using drop() function by specifying axis=0 for rows.
- Provide the labels of the rows to be deleted as argument to the drop() function.
- Don't forget to use inplace=True, otherwise the original DataFrame will remain unchanged.

```
In [69]: df.drop(2, axis=0, inplace=True)
```

```
In [70]: df
```

Out[70]:

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
6	Trix	110	1	25	27.753301

Adding and Deleting Rows and Columns (3/4)

Adding Columns

- To add a column to the DataFrame, we use the same notation as adding a key, value pair to a dictionary.
- Instead of the key, we provide column name in the square brackets, and then provide a list of values for that column.
- If no column with the given name exists, a new column with the specified name and values will be added to the DataFrame.

```
In [71]: df['My Column'] = ['A', 'B', 'C', 'D', 'E']
df
```

Out[71]:

	name	calories	protein	vitamins	rating	My Column
0	100% Bran	70	4	25	68.402973	A
1	100% Natural Bran	120	3	0	33.983679	B
3	All-Bran with Extra Fiber	50	4	25	93.704912	C
4	Almond Delight	110	2	25	34.384843	D
6	Trix	110	1	25	27.753301	E

Adding and Deleting Rows and Columns (4/4)

Deleting Columns

- We can also delete columns of the DataFrame using drop() function by specifying axis=1 for columns.
- Provide the column names to be deleted as argument to the drop() function.
- Don't forget to use inplace=True, otherwise the original DataFrame will remain unchanged.

```
In [72]: df.drop('My Column', axis=1, inplace=True)
```

```
In [73]: df
```

Out[73]:

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
6	Trix	110	1	25	27.753301

Sorting Values (1/2)

Ascending

- We can sort the values of a DataFrame with respect to a column using the `sort_values()` function, which sorts the values in ascending order by default.
- If the values of the column are alphabets, they are sorted alphabetically.
- If the values of the column are numbers, they are sorted numerically.

```
In [74]: df.sort_values(by='calories')
```

Out[74]:

		name	calories	protein	vitamins	rating
3	All-Bran with Extra Fiber		50	4	25	93.704912
0	100% Bran		70	4	25	68.402973
4	Almond Delight		110	2	25	34.384843
6	Trix		110	1	25	27.753301
1	100% Natural Bran		120	3	0	33.983679

Sorting Values (2/2)

Descending

- To sort the values in descending order, we set ascending = False in the sort_values() function.

```
In [75]: df.sort_values(by='calories', ascending=False)
```

Out[75]:

		name	calories	protein	vitamins	rating
1	100% Natural Bran		120	3	0	33.983679
4	Almond Delight		110	2	25	34.384843
6	Trix		110	1	25	27.753301
0	100% Bran		70	4	25	68.402973
3	All-Bran with Extra Fiber		50	4	25	93.704912

Exporting and Saving Pandas DataFrame

- To export a DataFrame as a csv file, use `to_csv()` function.
- If a file with the specified filename exists, it will be modified. Otherwise, a new file with the specified filename will be created.
- If you do not want to store index column in the csv file, you can set `index_label=False` in the `to_csv()` function.

```
In [76]: df.to_csv('myFile.csv', index_label=False)
```

```
In [77]: newDf = pd.read_csv('myFile.csv')
newDf
```

Out[77]:

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
6	Trix	110	1	25	27.753301

Concatenating DataFrames (1/3)

- We can concatenate two or more DataFrames together using pandas.concat() function.

First Data Frame

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843

Second Data Frame

	name	calories	protein	vitamins	rating
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813
10	Cap'n'Crunch	120	1	25	18.042851

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505
3	All-Bran with Extra Fiber	50	4	25	93.704912
4	Almond Delight	110	2	25	34.384843
5	Apple Cinnamon Cheerios	110	2	25	29.509541
6	Apple Jacks	110	2	25	33.174094
7	Basic 4	130	3	25	37.038562
8	Bran Chex	90	2	25	49.120253
9	Bran Flakes	90	3	25	53.313813
10	Cap'n'Crunch	120	1	25	18.042851

Resultant Data Frame

Concatenating DataFrames (2/3)

- We can also concatenate two or more DataFrames side-by-side each other.

First Data Frame

	name	calories	protein	vitamins	rating
0	Apple Cinnamon Cheerios	110	2	25	29.509541
1	Apple Jacks	110	2	25	33.174094
2	Basic 4	130	3	25	37.038562

Second Data Frame

	name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973
1	100% Natural Bran	120	3	0	33.983679
2	All-Bran	70	4	25	59.425505

Resultant Data Frame

	name	calories	protein	vitamins	rating		name	calories	protein	vitamins	rating
0	100% Bran	70	4	25	68.402973		Apple Cinnamon Cheerios	110	2	25	29.509541
1	100% Natural Bran	120	3	0	33.983679		Apple Jacks	110	2	25	33.174094
2	All-Bran	70	4	25	59.425505		Basic 4	130	3	25	37.038562

Concatenating DataFrames (3/3)

- To join two or more DataFrames side-by-side, use axis = 1 in the pandas.concat() function.

```
In [26]: df
Out[26]:
      name  calories  protein  vitamins  rating
0    100% Bran       70        4         25  68.402973
1  100% Natural Bran     120        3         0  33.983679
2      All-Bran       70        4         25  59.425505
```



```
In [36]: df2
Out[36]:
      name  calories  protein  vitamins  rating
0  Apple Cinnamon Cheerios     110        2         25  29.509541
1          Apple Jacks       110        2         25  33.174094
2            Basic 4       130        3         25  37.038562
```



```
In [37]: pd.concat([df, df2], axis=1)
Out[37]:
      name  calories  protein  vitamins  rating      name  calories  protein  vitamins  rating
0    100% Bran       70        4         25  68.402973  Apple Cinnamon Cheerios     110        2         25  29.509541
1  100% Natural Bran     120        3         0  33.983679          Apple Jacks       110        2         25  33.174094
2      All-Bran       70        4         25  59.425505          Basic 4       130        3         25  37.038562
```

groupby() (1/7)

- groupby() function is used to group DataFrame based on Series.
 - The DataFrame is splitted into groups.
 - An aggregate function is applied to each column of the splitted DataFrame.
 - Results are combined together.
- Consider the following DataFrame.

	Gender	Score
0	female	85
1	male	88
2	female	95
3	male	80

groupby() (2/7)

- The 'Gender' column contains two values, male and female.
- Let's split our DataFrame into two parts based on 'Gender' column;
 - First part will contain the rows where Gender = male
 - Second part will contain the rows where Gender = female

	Gender	Score
0	female	85
2	female	95

	Gender	Score
1	male	88
3	male	80

groupby() (3/7)

- If we find the mean score of both the genders, this is what we get.

Gender	Score
female	90

Gender	Score
male	84

groupby() (4/7)

- Let's combine the two results together. This is what we get.

Gender	Score
female	90
male	84

groupby() (5/7)

- The groupby() function works exactly the same way, except that it makes things easier for us.
- In the given example, we group our DataFrame on the basis of 'Gender' column, and then apply the aggregate function mean() on it.

```
In [55]: df
```

```
Out[55]:
```

	Gender	Score
0	female	85
1	male	88
2	female	95
3	male	80

```
In [56]: df.groupby(x['Gender']).mean()
```

```
Out[56]:
```

Gender	Score
female	90
male	84

groupby() (6/7)

- Note that aggregate functions are applied automatically on all the columns of the DataFrame except the one used to group the DataFrame.

```
In [72]: df
```

```
Out[72]:
```

	Gender	Math	English
0	female	85	80
1	male	88	88
2	female	95	92
3	male	80	95

```
In [73]: df.groupby(x['Gender']).mean()
```

```
Out[73]:
```

Gender	Math	English
female	90.0	86.0
male	84.0	91.5

groupby() (7/7)

- The common aggregate functions are;
 - mean()
 - sum()
 - max()
 - min()
 - median()
 - count()
 - std() (standard deviation)

Resources

- <https://www.geeksforgeeks.org/python-pandas-dataframe/?ref=lbp>
- https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html
- https://www.w3schools.com/python/pandas/pandas_dataframes.asp

DATA CLEANING

Introduction to Data Cleaning

- Data extracted from real world contains incorrect, incomplete, irrelevant or missing values which need to be cleaned.
- Cleaning can be done by modifying, replacing or deleting such values.
- Data cleaning is a fundamental part of data science lifecycle.



<https://www.educative.io/blog/what-is-data-cleaning>

Quality of Data

- Today's world is all about data-driven decision making, hence the quality of our data will ultimately impact the quality of the decision that we make based on that data.
- Data is generally considered high quality if it is "**fit for [its] intended uses in operations, decision making and planning.**"
- Through different data cleaning techniques, we can improve the quality of our data extracted from the real world.

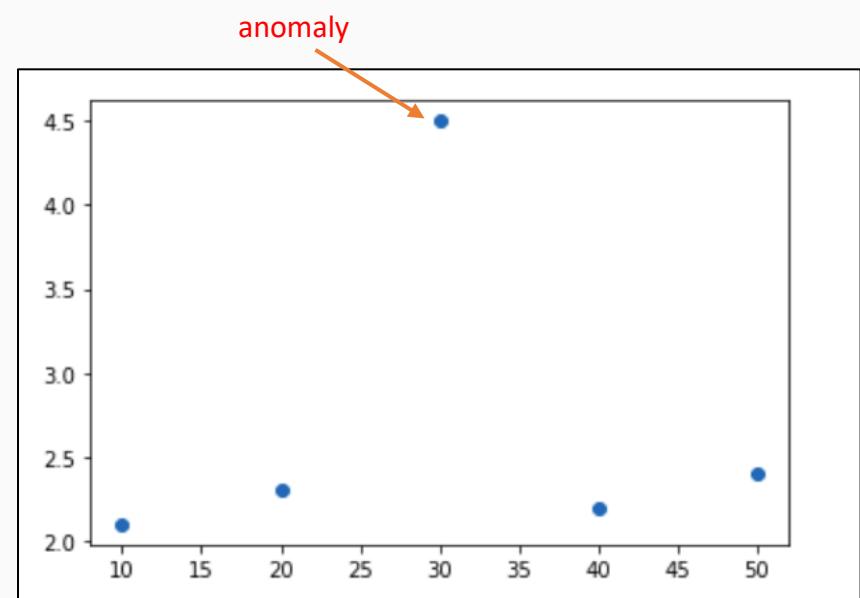
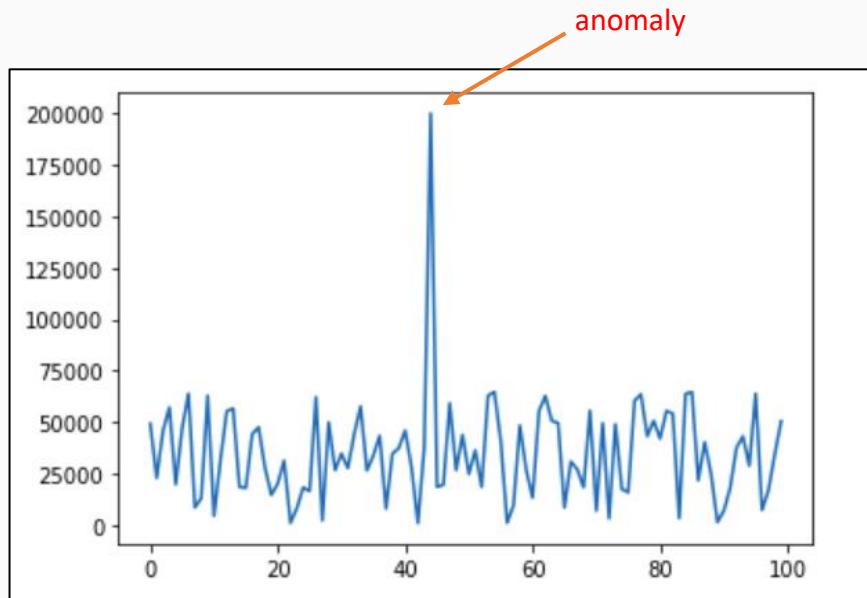
Examples of Anomalies (1/2)

- An anomaly in data, also known as an outlier, is the observation which differs significantly from the standard pattern.
- Anomalies in data could arise for a number of reasons, such as human error.
- Consider the given series containing width of a table measured by 5 different students.
 - The third value is significantly different than the rest – an anomaly!

```
0    2.1
1    2.3
2    4.5
3    2.2
4    2.4
dtype: float64
```

Examples of Anomalies (2/2)

- Data extracted from real world often contains such anomalies;
 - Temperature data over a period of 6 months might contain some irregular values due to tremendous shifts in weather conditions.
- It is important to detect such anomalies in the data and deal with them appropriately.



Median-based Anomaly Detection

- One way to detect anomalies is using the median value.
- We set a reasonable threshold and if for a certain value, $|value - \text{median}| > \text{threshold}$, then the value is considered an anomaly.

```
x = pd.Series([2.1, 2.3, 4.5, 2.2, 2.4])

median = np.median(x)
threshold = 2
outliers = []
for item in x:
    if abs(item - median) > threshold:
        outliers.append(item)

print(outliers)
```

```
[4.5]
```

Quiz Time

A certain dataset has a median value 3, and the threshold for anomaly detection is 2. Third value of the dataset is 7. According to median-based anomaly detection, is the third value an anomaly?

- Yes
- No

Quiz Time

A certain dataset has a median value 3, and the threshold for anomaly detection is 2. Third value of the dataset is 7. According to median-based anomaly detection, is the third value an anomaly?

- Yes
- No

Mean-based Anomaly Detection

- Another way to detect anomalies is using mean and standard deviation of the data.
- We define the range as $(\text{mean} - \text{standard deviation}) \leq \text{value} \leq (\text{mean} + \text{standard deviation})$
 - i.e., any value less than $(\text{mean} - \text{standard deviation})$ or greater than $(\text{mean} + \text{standard deviation})$ is considered an anomaly.

```
x = pd.Series([2.1, 2.3, 4.5, 2.2, 2.5])

mean = np.mean(x)
std = np.std(x)
outliers = []
for item in x:
    if (item < mean - std) or (item > mean + std):
        outliers.append(item)

outliers
[4.5]
```

Quiz Time

1. A certain dataset has a mean value 20 and a standard deviation of 2.5. A certain value x in the dataset is 16. According to mean-based anomaly detection, is x an anomaly?

- Yes
- No

Quiz Time

1. A certain dataset has a mean value 20 and a standard deviation of 2.5. A certain value x in the dataset is 16. According to mean-based anomaly detection, is x an anomaly?

- Yes
- No

Z-score-based Anomaly Detection

- Another technique for anomaly detection is the Z-score.
- Z-score is a statistical measure showing how many standard deviations a certain value is from the mean.
- It is defined as;
 - $Z = (\text{value} - \text{mean}) / \text{standard deviation}$
- If the Z-score of a value is greater than a reasonable threshold, it is considered an anomaly.

```
x = pd.Series([2.1, 2.3, 4.5, 2.2, 2.4])

mean = np.mean(x)
std = np.std(x)
outliers = []
for item in x:
    z_score = (item - mean) / std
    if z_score > 1.5:
        outliers.append(item)

print(outliers)
```

[4.5]

Quiz Time

1. A certain dataset has a mean value 12 and a standard deviation of 2. A certain value x in the dataset is 8. Compute the Z-score of the value x?

- 4
- 20
- 2
- 10

Quiz Time

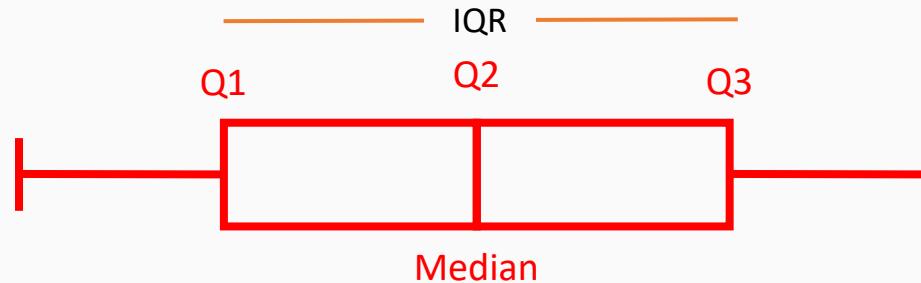
1. A certain dataset has a mean value 12 and a standard deviation of 2. A certain value x in the dataset is 8. Compute the Z-score of the value x?

- 4
- 20
- 2
- 10

Interquartile Range for Anomaly Detection (1/3)

IQR

- We can also use interquartile range (IQR) for detecting anomalies in the data.
- A quartile divides the dataset (sorted from smallest to largest) into 3 points and 4 intervals.
- The interquartile range (IQR) is the difference between the 3rd quartile and 1st quartile ($IQR = Q3 - Q1$).



Interquartile Range for Anomaly Detection (2/3)

- Consider the series of widths from the previous slide as a list as shown below;
 - Widths = [2.3, 2.2, 4.5, 2.1, 2.5]
- We first sort this list from smallest to largest.
 - Widths = [2.1, 2.2, 2.3, 2.5, 4.5]
- The first quartile is at 25%, which is 2.2.
- The second quartile is 50%, which is 2.3 (median).
- The third quartile is 75%, which is 2.5.
- Use the `numpy.percentile()` function to get the quartiles of a dataset.
 - Automatically sorts the data from smallest to largest.

Interquartile Range for Anomaly Detection (3/3)

Anomaly

- Any value less than $(Q1 - 1.5 \times IQR)$ or greater than $(Q3 + 1.5 \times IQR)$ is considered an anomaly.

```
x = pd.Series([2.1, 2.3, 4.5, 2.2, 2.5])

Q1, Q3 = np.percentile(x, [25, 75])
IQR = Q3 - Q1
outliers = []
for item in x:
    if item < (Q1 - 1.5 * IQR) or item > (Q3 + 1.5 * IQR):
        outliers.append(item)

outliers
[4.5]
```

Quiz Time

1. Consider the given dataset $x=[1,2,3,4,5]$. What is the second quartile (Q2) of the dataset?

- 2
- 3
- 4

2. Compute the interquartile range for the dataset $x=[1,2,3,4,5]$.

- 1
- 2
- 3

Quiz Time

1. Consider the given dataset $x=[1,2,3,4,5]$. What is the second quartile (Q2) of the dataset?

- 2
- 3
- 4

2. Compute the interquartile range for the dataset $x=[1,2,3,4,5]$.

- 1
- 2
- 3

Dealing with Missing values (1/6)

- Apart from outliers/anomalies, data also often contains missing values.
- A missing value means loss of information.
- In pandas, an NaN indicates a missing value.
- Consider the following dataframe called data with one missing value in the 'Age' column.

	Name	Age
0	Edison	28
1	Edward	27
2	James	NaN
3	Neesham	36

Dealing with Missing values (2/6)

Finding Missing Values in Pandas

- The `.isnull()` function tells us if a cell is empty or not.
- Use the `.sum()` function with the `.isnull()` function to find total number of missing values in the data.

The screenshot shows two code cells in a Jupyter Notebook. The first cell contains the code `data.isnull()` and displays a DataFrame with columns `Name` and `Age`. The second cell contains the code `data.isnull().sum()` and displays a Series with the count of missing values for each column.

	Name	Age
0	False	False
1	False	False
2	False	True
3	False	False

	0
Name	0
Age	1
dtype:	int64

Dealing with Missing values (3/6)

Dealing with Missing Values in Pandas

- There are a number of ways to deal with these missing values.
- Which method to use depends upon the kind of data and the task that the data is supposed to accomplish.
- Different Methods Used are;
 - Deleting rows with missing values.
 - Replacing missing values with mean/median/mode.

Dealing with Missing values (4/6)

Deleting Rows with Missing Values

- One way to deal with the missing values is to delete the rows containing missing values.
- Use the `.dropna()` with inplace set to True to remove missing values from the dataset.

```
data.dropna(inplace=True)  
data
```

	Name	Age
0	Edison	28
1	Edward	27
3	Neesham	36

Dealing with Missing values (5/6)

Replacing Missing Values with Mean/Median/Mode

- We can also replace the missing values in each column with one of the statistical measures (mean/median/mode) of that column.
- Use the `.fillna()` method to fill the missing values with mean, median, or mode.

```
data.fillna(data.mean(), inplace=True)  
data
```

	Name	Age
0	Edison	28.000000
1	Edward	27.000000
2	James	30.333333
3	Neesham	36.000000

Dealing with Missing values (6/6)

Replacing Missing Values with Mean/Median/Mode

- In this example, we replace the missing value in the 'Age' column with the mode of the 'Age' column.

```
data
```

```
      0      1
0   Edison  28.0
1   Edward  27.0
2   James   NaN
3  Neesham 36.0
4   Stuart  27.0
```

```
data['Age'].mode()
```

```
0    27.0
dtype: float64
```

```
data['Age'].fillna(data['Age'].mode()[0], inplace=True)
data
```

	Name	Age
0	Edison	28.0
1	Edward	27.0
2	James	27.0
3	Neesham	36.0
4	Stuart	27.0

Feature Scaling (1/5)

- Sometimes, we might wish to normalize the range of each feature (column) of the given dataset.
- For example, consider the given dataframe where the range of each feature (column) is different.
- We would like to scale all the features to the same range, e.g. 0-1, 1-100, etc.

df		
	Age	Salary
0	28.0	10000
1	27.0	15000
2	30.0	11000
3	36.0	11000
4	27.0	13000

Feature Scaling (2/5)

Normalization

- One simple method of feature scaling is normalization, also called min-max scaling.
- For every value in a feature (column), we subtract the minimum value of the particular feature (column) from it and divide it by the difference of maximum and minimum value of that feature (column).
 - Normalized value = $(\text{original value} - \text{minimum}) / (\text{maximum} - \text{minimum})$
- This method scales the features in the range [0, 1]

Feature Scaling (3/5)

Normalization

- For min-max scaling, use the following line of code;
 - `normalized_df=(df-df.min())/(df.max()-df.min())`
- Pandas will automatically use the feature (column) min-max values for each feature.

```
df = (df - df.min()) / (df.max() - df.min())
df
```

	Age	Salary
0	0.111111	0.0
1	0.000000	1.0
2	0.333333	0.2
3	1.000000	0.2
4	0.000000	0.6

Feature Scaling (4/5)

Standardization

- In standardization, for each value of a feature (column), we subtract the mean of that feature (column) from the value and divide the result by the standard deviation of that feature (column).
 - Standardized value = $(\text{original value} - \text{mean}) / \text{standard deviation}$
- As a result, the standard deviation of the feature (column) becomes 1.

Feature Scaling (5/5)

Standardization

- For standardization, use the following line of code;
 - `standardized_df=(df-df.mean())/df.std()`
- Once again, Pandas will automatically use the feature (column) mean and std values for each feature.

```
df = (df - df.mean()) / df.std()  
df
```

	Age	Salary
0	-0.423109	-1.0
1	-0.687552	1.5
2	0.105777	-0.5
3	1.692435	-0.5
4	-0.687552	0.5

```
df.std()  
Age      1.0  
Salary   1.0  
dtype: float64
```

Quiz Time

1. Min-max scaling is also known as;

- Normalization
- Standardization

Quiz Time

1. Min-max scaling is also known as;

- Normalization
- Standardization

Regular Expressions (1/8)

- Regular Expression or RegEx is an expression containing a sequence of characters for matching patterns in strings.
- Almost all the major programming languages have implementation for RegEx.
- The 're' module of Python is used for pattern matching using RegEx in Python.
- Following functions are available in the 're' module;
 - `.findall()`
 - `search()`
 - `sub()`

Regular Expressions (2/8)

re.findall()

- `re.findall()` is used to match all the occurrences of a pattern in a string.
- A list with all the matches is returned.
- In the given figure, we find how many times does the word 'Python' appear in the given string.

```
import re

txt = 'Python is my favorite programming language. I love Python.'
x = re.findall('Python', txt)
x

['Python', 'Python']
```

```
len(x)
```

2

Regular Expressions (3/8)

re.findall()

- In the given figure, we check if the string x starts with the word 'Python' or not.
- The `^` character returns a match only if the string starts with the pattern given after `^` symbol.
- The string y contains the word Python but does not begin with it, hence we get an empty list.

```
import re

x = 'Python is my favorite programming language.'
re.findall('^Python', x)
```

```
['Python']
```

```
y = 'I love Python.'
re.findall('^Python', y)
```

```
[]
```

Regular Expressions (4/8)

re.findall()

- To match numbers in a string, we use the \d sequence.
- A + sign at the end of \d makes sure that number such as 50 is treated as 50 and not as 5 and 0.
- You can find a list of sequences at https://www.w3schools.com/python/python_regex.asp

```
import re
```

```
txt = 'Python was released in 1991.'  
re.findall('\d', txt)
```

```
['1', '9', '9', '1']
```

```
txt = 'Python was released in 1991.'  
re.findall('\d+', txt)
```

```
['1991']
```

Regular Expressions (5/8)

re.findall()

- To find matches in a Series, we first convert the Series into a string using the `.to_string()` method.

```
import pandas as pd
import re
```

```
txtList = ['Pakistan', 'Indonesia', 'Jordan', 'Pakistan']
txt = pd.Series(txtList)
txt
```

```
0      Pakistan
1      Indonesia
2       Jordan
3      Pakistan
dtype: object
```

```
re.findall('Pakistan', txt.to_string())
```

```
['Pakistan', 'Pakistan']
```

Regular Expressions (6/8)

re.search()

- `re.search()` returns a Match Object in case of a pattern match in the string.
- We can get the position of the match using the `.span()` method of the Match Object.

```
import pandas as pd  
import re
```

```
txt = 'Hello World'  
match_object = re.search('World', txt)  
match_object
```

```
<re.Match object; span=(6, 11), match='World'>
```

```
match_object.span()
```

```
(6, 11)
```

Regular Expressions (7/8)

re.sub()

- To replace text in a string with a different text, use the `re.sub()` method.

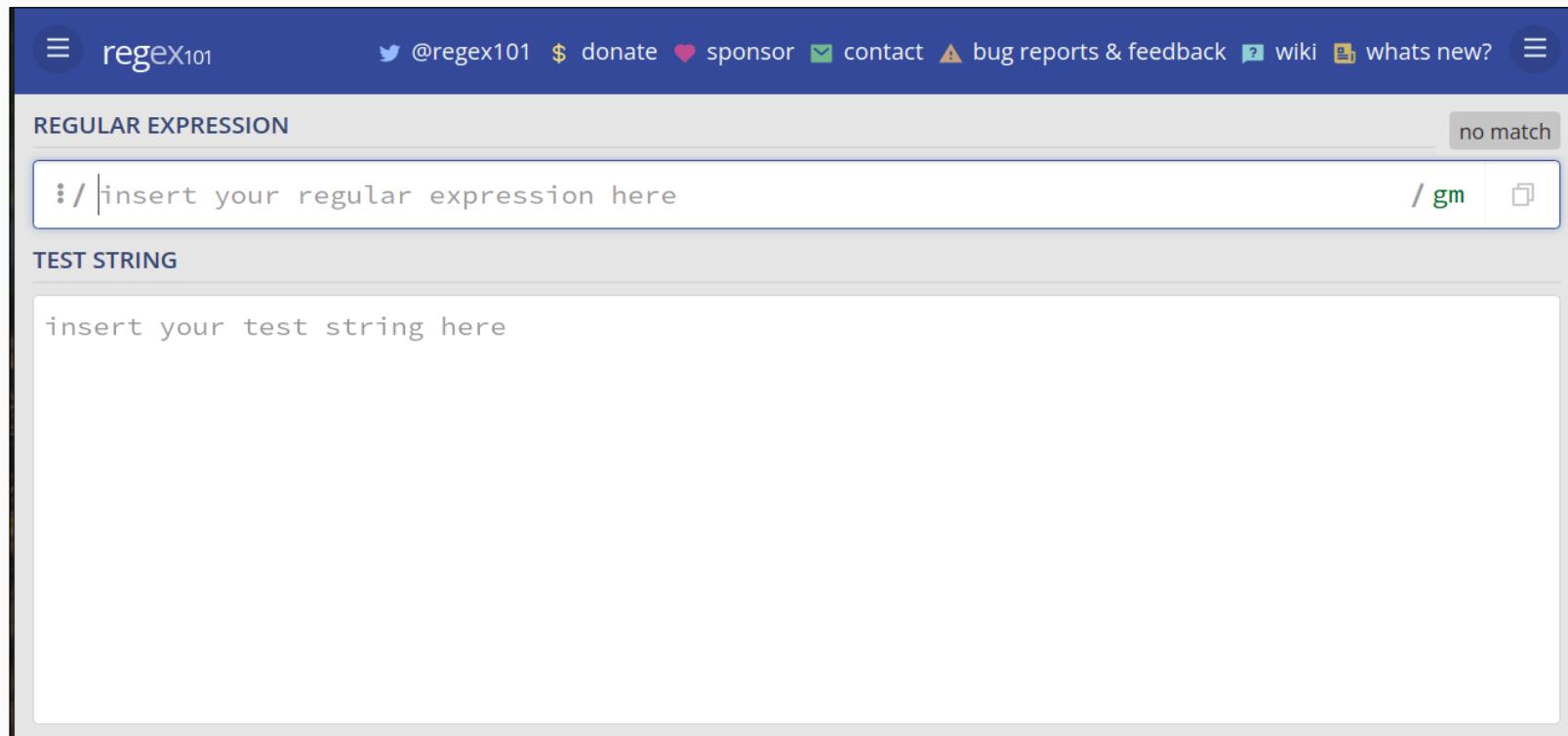
```
import pandas as pd  
import re
```

```
txt = 'C is my favorite programming language.'  
re.sub(pattern='C', repl='Python', string=txt)
```

```
'Python is my favorite programming language.'
```

Regular Expressions (8/8)

- <https://regex101.com/> is a very good website to create and test regular expressions.
- We have also added some resources to learn more about regular expression in Python in the 'Resources slide'.



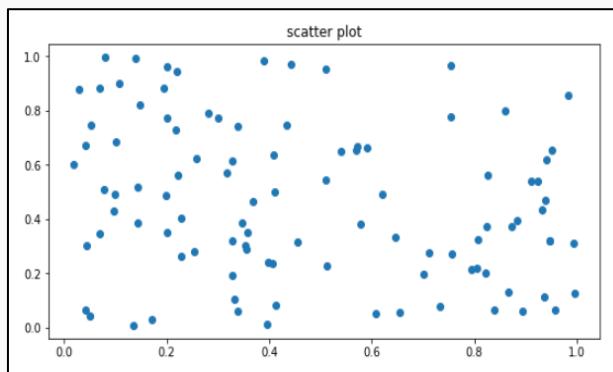
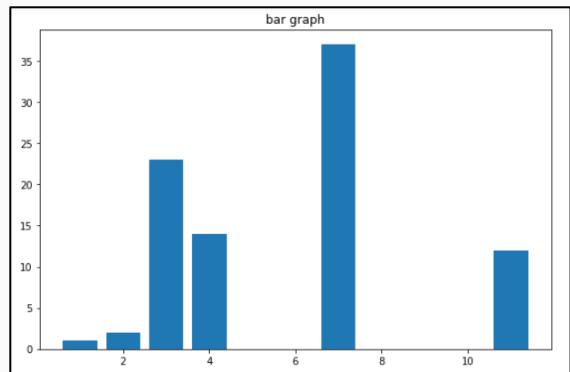
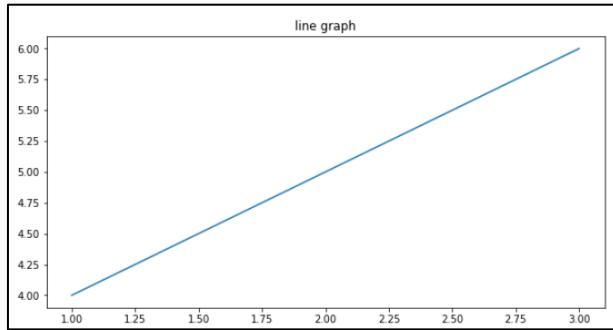
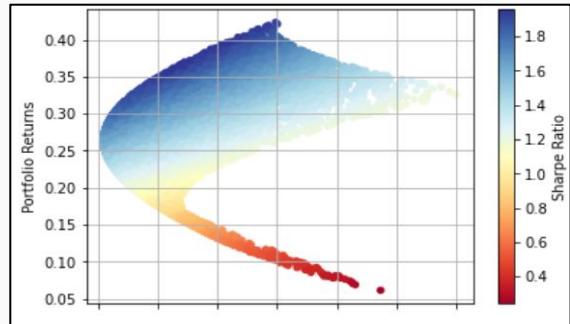
Resources

- https://www.w3schools.com/python/pandas/pandas_cleaning.asp
- <https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/>
- <https://www.kdnuggets.com/2021/04/data-science-101-normalization-standardization-regularization.html>
- https://www.w3schools.com/python/python_regex.asp

DATA VISUALIZATION USING PYTHON

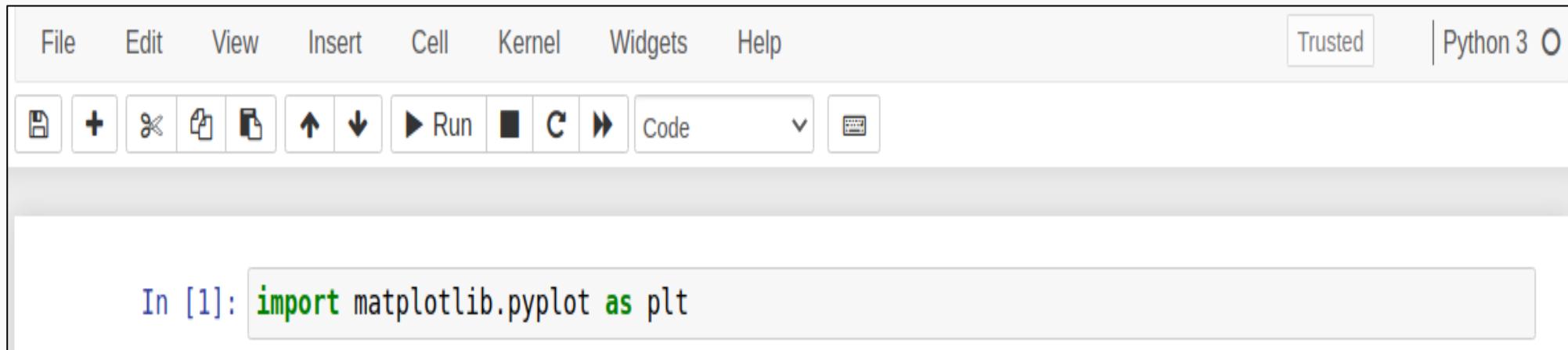
About Matplotlib

- Matplotlib is the most popular python library for plotting different kinds of graphs.
- The Pyplot module inside the Matplotlib makes it work like Matlab.



Importing Matplotlib

- To import matplotlib.pyplot, type ‘import matplotlib.pyplot’ in Jupyter Notebook and run the cell.
- The common abbreviation used for matplotlib.pyplot is plt.



The image shows a screenshot of a Jupyter Notebook interface. At the top, there is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu, there are two status indicators: "Trusted" and "Python 3". Below the menu is a toolbar with various icons: a file icon, a plus sign, a delete icon, a copy icon, a paste icon, up and down arrow icons, a "Run" button, a cell type icon, and a "Code" dropdown menu. The main area of the notebook is a code cell labeled "In [1]:" containing the Python code: `import matplotlib.pyplot as plt`.

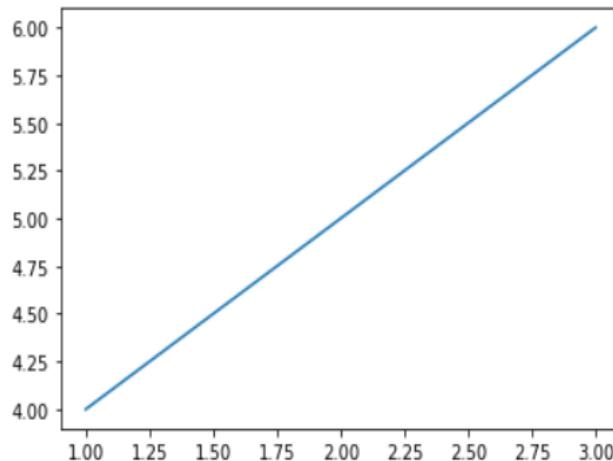
Plotting Line Plots (1/2)

- We can plot line plots using matplotlib using the `.plot()` function.
 - The first argument in the `.plot()` function specifies the x-axis.
 - The second argument in the `.plot()` function specifies the y-axis.

```
In [3]: x_axis = [1,2,3]
y_axis = [4,5,6]
```

```
plt.plot(x_axis, y_axis)
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x7f186e0faf70>]
```



Plotting Line Plots (2/2)

Changing Color

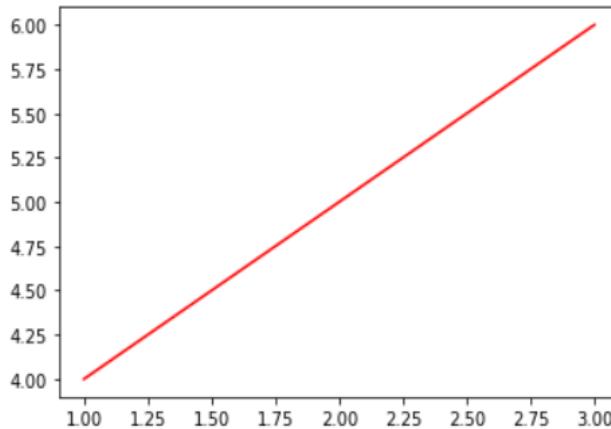
- We can also change the color of the line by providing the color as third argument in the `plot()` function.
- A list of the color abbreviations can be found at:

https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

```
In [4]: x_axis = [1,2,3]
y_axis = [4,5,6]

plt.plot(x_axis, y_axis, 'r')

Out[4]: [<matplotlib.lines.Line2D at 0x7f186d85eb80>]
```



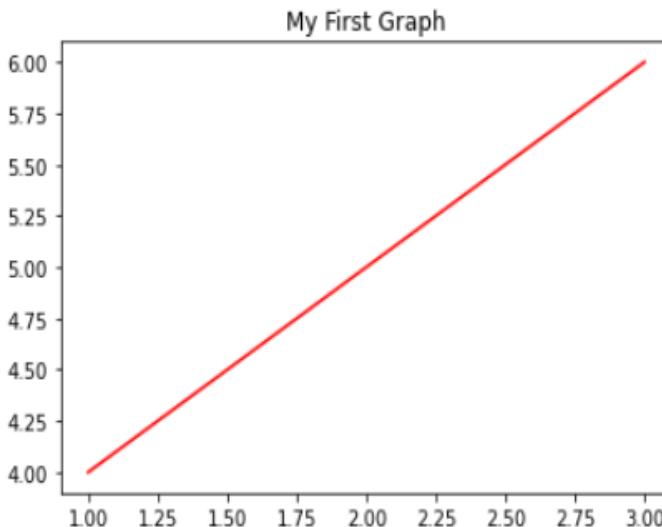
Title

- To set the title of the plot, use the `.title()` function.

```
In [4]: x_axis = [1,2,3]
y_axis = [4,5,6]

plt.title('My First Graph')
plt.plot(x_axis, y_axis, 'r')
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x7efef8336700>]
```



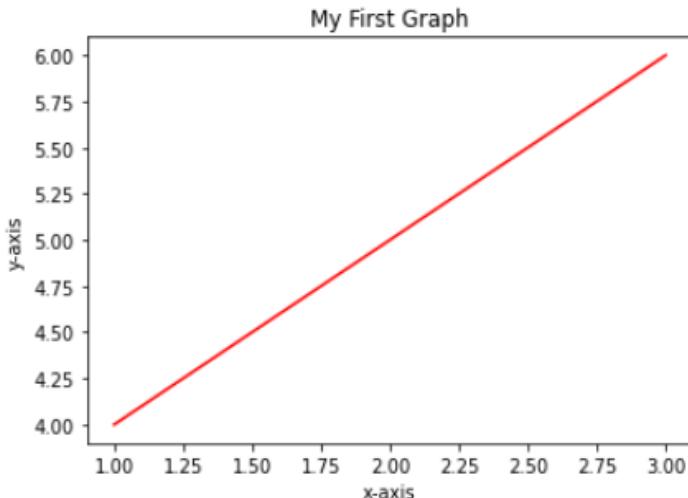
Labels

- To assign labels to x and y-axis, use `.xlabel()` and `.ylabel()` respectively.

```
In [5]: x_axis = [1,2,3]
y_axis = [4,5,6]

plt.title('My First Graph')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.plot(x_axis, y_axis, 'r')
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x7efef82a4340>]
```



Legend (1/2)

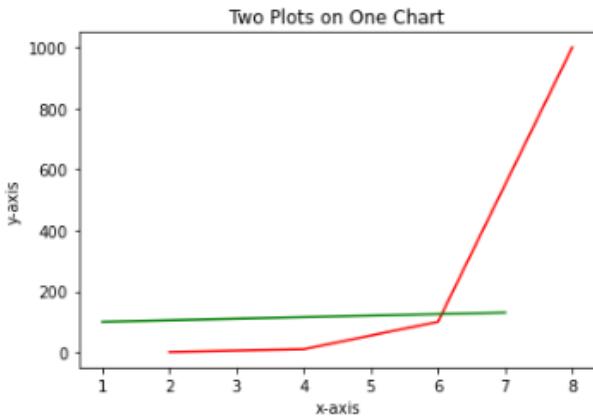
- We can plot multiple plots on the same chart simply by plotting them one by one as in the given example.

```
In [10]: x1_axis = [2, 4, 6, 8]
y1_axis = [1, 10, 100, 1000]
x2_axis = [1, 3, 5, 7]
y2_axis = [100, 110, 120, 130]

plt.title('Two Plots on One Chart')
plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.plot(x1_axis, y1_axis, 'r')
plt.plot(x2_axis, y2_axis, 'g')
```

```
Out[10]: [<matplotlib.lines.Line2D at 0x7efef81c7850>]
```



Legend (2/2)

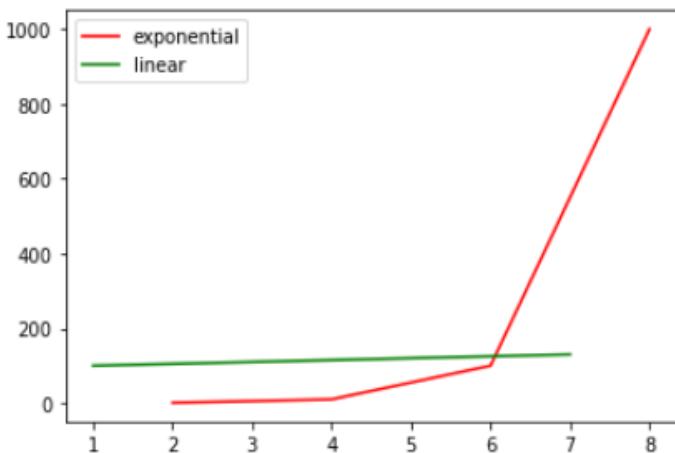
- If plotting more than one plots, it is a good idea to add a legend in your figure using the `.legend()` function.

```
In [12]: x1_axis = [2, 4, 6, 8]
y1_axis = [1, 10, 100, 1000]
x2_axis = [1, 3, 5, 7]
y2_axis = [100, 110, 120, 130]

plt.plot(x1_axis, y1_axis, 'r')
plt.plot(x2_axis, y2_axis, 'g')

plt.legend(['exponential', 'linear'])
```

Out[12]: <matplotlib.legend.Legend at 0x7efef81037c0>

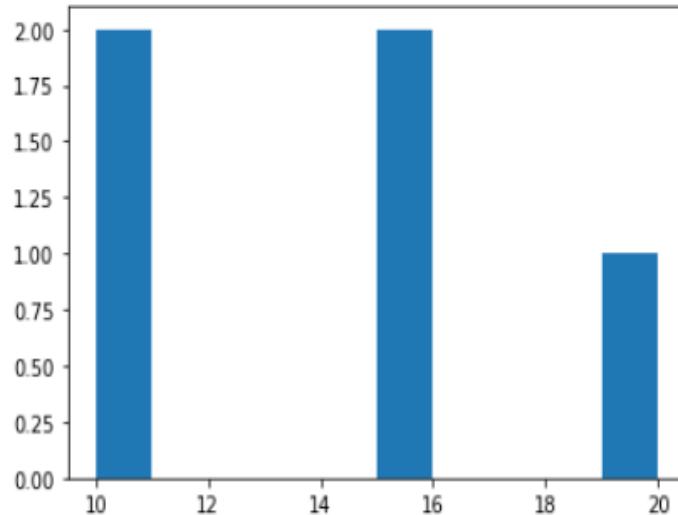


Plotting Histograms (1/3)

- We can also plot histograms using `.hist()` function.
- A histogram is generally used to plot frequency which helps identify distribution of data.

```
In [20]: values = [10, 15, 20, 10, 15]
plt.hist(values)
```

```
Out[20]: (array([2., 0., 0., 0., 0., 2., 0., 0., 0., 1.]),
           array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
           <BarContainer object of 10 artists>)
```



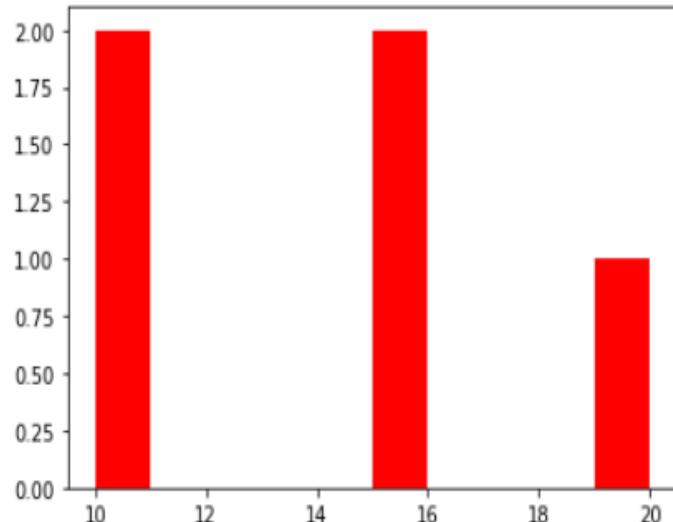
Plotting Histograms (2/3)

Changing Color

- We can also change color of the bars using the 'color' parameter inside the hist() function.

```
In [21]: values = [10, 15, 20, 10, 15]
plt.hist(values, color='r')
```

```
Out[21]: (array([2., 0., 0., 0., 0., 2., 0., 0., 0., 1.]),
           array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
           <BarContainer object of 10 artists>)
```



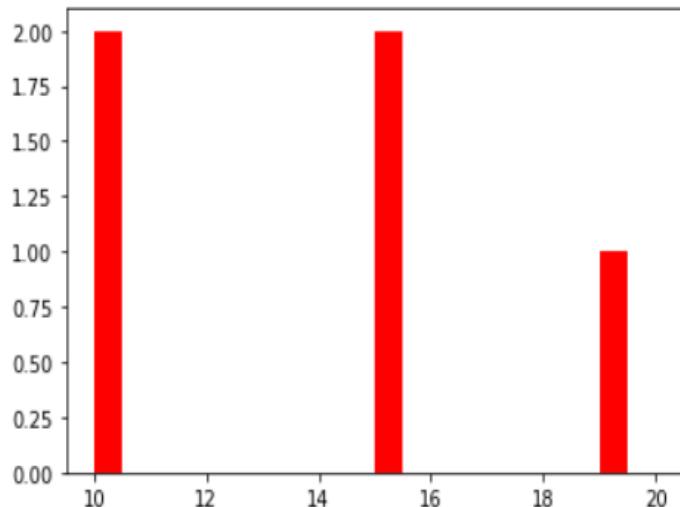
Plotting Histograms (3/3)

Changing Width

- We can also change width of the bars using the ‘width’ parameter inside the hist() function.

```
In [26]: values = [10, 15, 20, 10, 15]
plt.hist(values, color='r', width=0.5)

Out[26]: (array([2., 0., 0., 0., 2., 0., 0., 0., 1.]),
array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
<BarContainer object of 10 artists>)
```

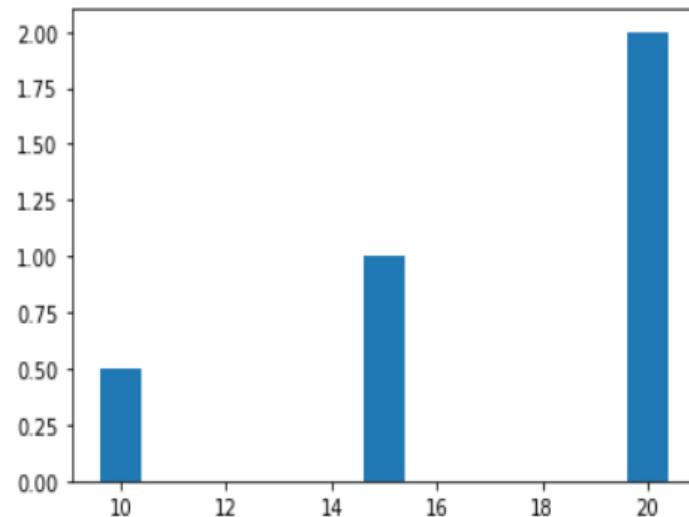


Plotting Bar Charts (1/2)

- To plot a bar graph, use the `.bar()` function of the `matplotlib.pyplot`.
 - First argument in the `bar()` function is the x-label.
 - Second argument in the `bar()` function is the height of each bar, which can be a list of values or a single value.

```
In [33]: values = [10, 15, 20]
plt.bar(values, [0.5, 1, 2])
```

```
Out[33]: <BarContainer object of 3 artists>
```



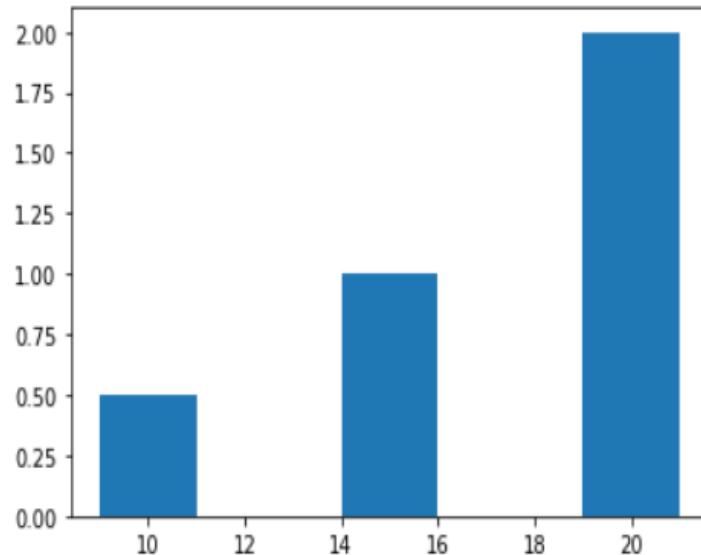
Plotting Bar Charts (2/2)

Changing Width

- We can also change the width of bars in the bar plot using the 'width' parameter.

```
In [34]: values = [10, 15, 20]
plt.bar(values, [0.5, 1, 2], width=2)
```

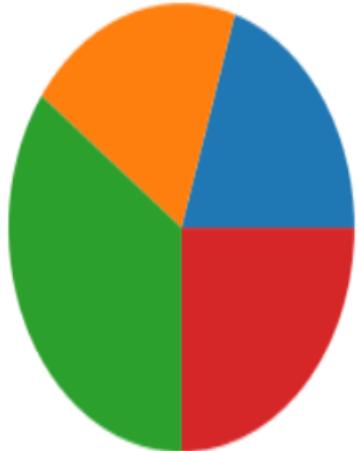
```
Out[34]: <BarContainer object of 3 artists>
```



Plotting Pie Charts (1/4)

- `.pie()` function is used to create a pie chart in `matplotlib.pyplot`.

```
In [42]: values = [20, 20, 35, 25]
plt.pie(values)
```

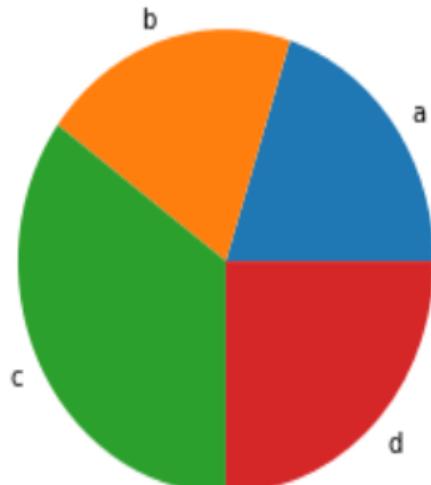


Plotting Pie Charts (2/4)

Labels

- To add labels in your pie chart, use the 'labels' parameter which takes a list of labels.

```
In [43]: values = [20, 20, 35, 25]
plt.pie(values, labels=['a', 'b', 'c', 'd'])
```



Plotting Pie Charts (3/4)

Explode

- If you want one or more wedges of your pie chart to stand out, you can use the 'explode' parameter.
 - Provide a list containing distance of each wedge from the center to the 'explode' parameter.

```
In [44]: values = [20, 20, 35, 25]
plt.pie(values, labels=['a', 'b', 'c', 'd'], explode=[0, 0.2, 0, 0])
```

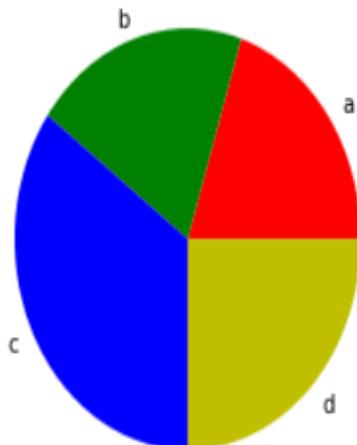


Plotting Pie Charts (4/4)

Colors

- We can also change the colors of wedges by providing a list of colors to the 'colors' parameter.

```
In [45]: values = [20, 20, 35, 25]
plt.pie(values, labels=['a', 'b', 'c', 'd'], colors=['r', 'g', 'b', 'y'])
```



Plotting Scatter Plot (1/5)

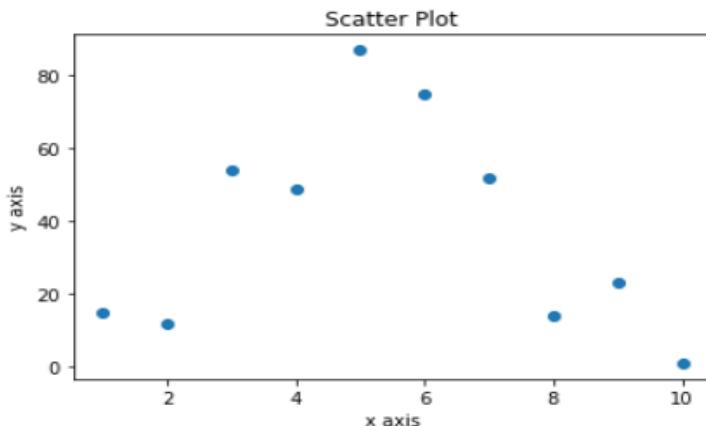
- We can also plot scatter plots using the `.scatter()` function inside `matplotlib.pyplot`.
- It takes two lists as arguments;
 - First list specifies the values for the x-axis
 - Second list specifies the values for the y-axis

```
In [48]: x_axis = [1,2,3,4,5,6,7,8,9,10]
y_axis = [15,12,54,49,87,75,52,14,23,1]

plt.title('Scatter Plot')
plt.xlabel('x axis')
plt.ylabel('y axis')

plt.scatter(x_axis, y_axis)

Out[48]: <matplotlib.collections.PathCollection at 0x7efef1670a00>
```



Plotting Scatter Plot (2/5)

Colors

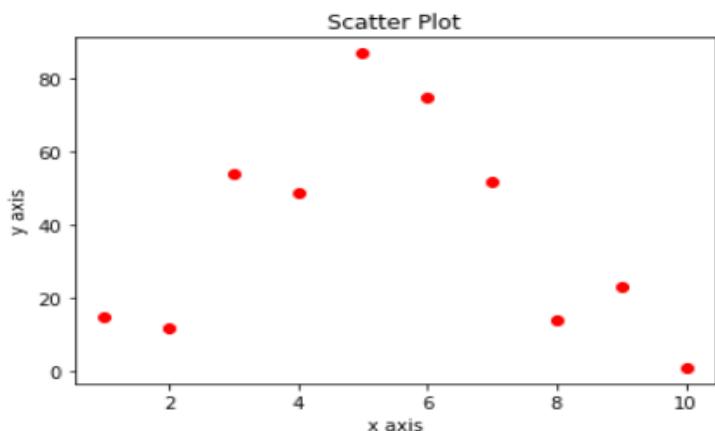
- We can also change color of the dots using the 'color' parameter.

```
In [49]: x_axis = [1,2,3,4,5,6,7,8,9,10]
y_axis = [15,12,54,49,87,75,52,14,23,1]

plt.title('Scatter Plot')
plt.xlabel('x axis')
plt.ylabel('y axis')

plt.scatter(x_axis, y_axis, color='r')

Out[49]: <matplotlib.collections.PathCollection at 0x7efef1b2dd00>
```



Plotting Scatter Plot (3/5)

Colormap

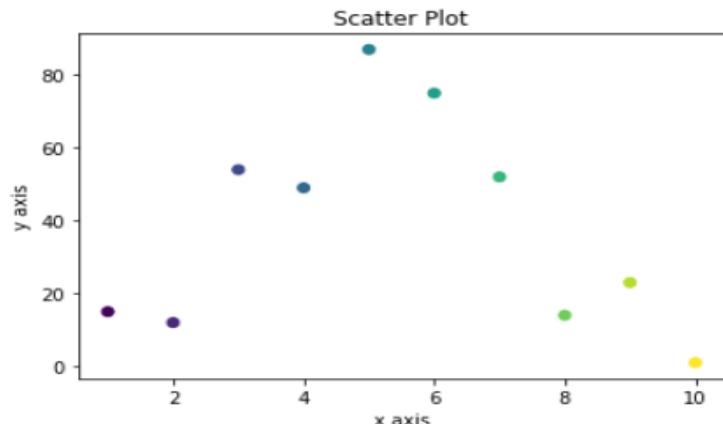
- If you would like to give each dot a different color, you can provide a list of colors or integers to the 'c' parameter.

```
In [52]: x_axis = [1,2,3,4,5,6,7,8,9,10]
y_axis = [15,12,54,49,87,75,52,14,23,1]

plt.title('Scatter Plot')
plt.xlabel('x axis')
plt.ylabel('y axis')

plt.scatter(x_axis, y_axis, c=[1,2,3,4,5,6,7,8,9,10])

Out[52]: <matplotlib.collections.PathCollection at 0x7efef19bf610>
```



Plotting Scatter Plot (4/5)

Colormap

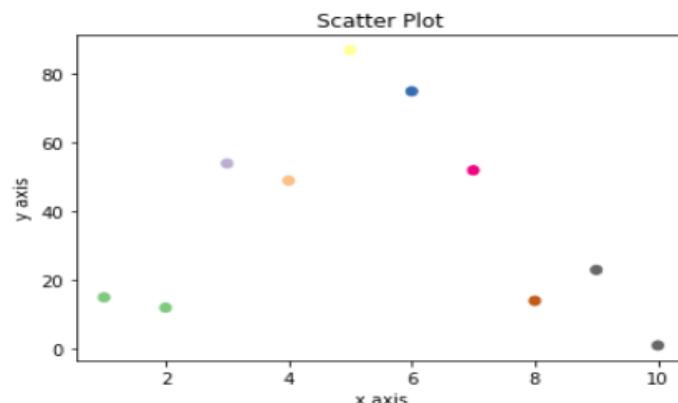
- You can also set the colormap using the ‘cmap’ parameter.
- For this, you will need to provide a list of integers that will be mapped to colors.
- We have used ‘Accent’ colormap, which is one of the many built-in colormaps in matplotlib.

```
In [53]: x_axis = [1,2,3,4,5,6,7,8,9,10]
y_axis = [15,12,54,49,87,75,52,14,23,1]

plt.title('Scatter Plot')
plt.xlabel('x axis')
plt.ylabel('y axis')

plt.scatter(x_axis, y_axis, c=[1,2,3,4,5,6,7,8,9,10], cmap='Accent')

Out[53]: <matplotlib.collections.PathCollection at 0x7efef188d190>
```



Plotting Scatter Plot (5/5)

Colormap

- To see the available colormaps in matplotlib, import ‘cm’ module from matplotlib.
- Give the ‘dir(cm)’ command and run the cell.
- A list of all the available colormaps will be displayed.

```
In [54]: from matplotlib import cm
```

```
In [55]: dir(cm)
```

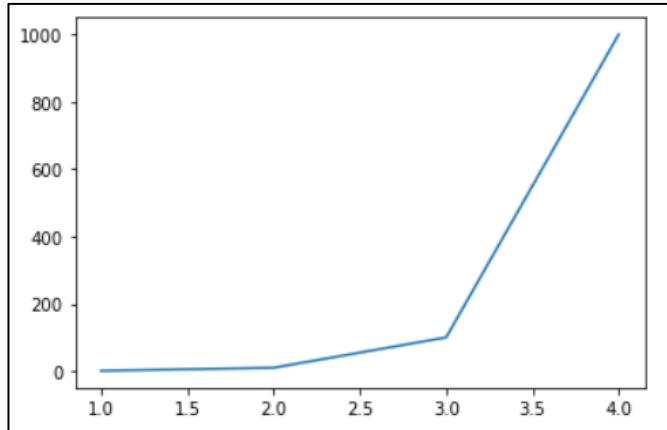
```
'__builtins__',  
'__cached__',  
'__doc__',  
'__file__',  
'__loader__',  
'__name__',  
'__package__',  
'__spec__',  
'_cmap_registry',  
'_gen_cmap_registry',  
'_reverser',  
'afmhot',  
'afmhot_r',  
'autumn',  
'autumn_r',  
'binary',  
'binary_r',  
'bone',  
'bone_r'.
```

Plotting Log Plots

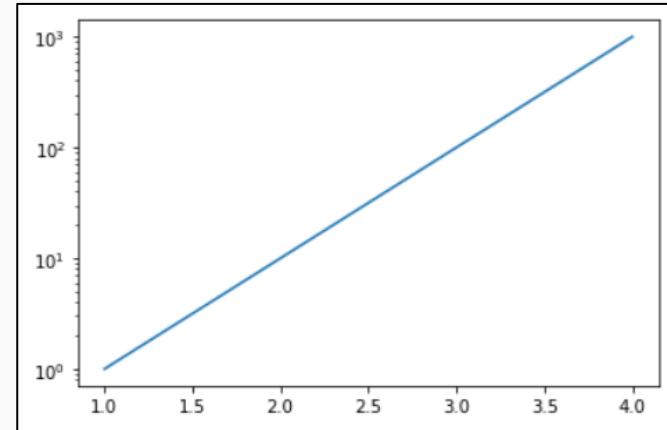
- We can also plot on logarithmic y-axis scale by using `.yscale()` function and passing 'log' as argument.
- Logarithmic scale is generally used if one or more data points are way bigger than bulk of the data, resulting in a skewed graph.

```
In [60]: x_axis = [1,2,3,4]
y_axis = [1, 10, 100, 1000]

plt.yscale('log')
plt.plot(x_axis, y_axis)
```



Linear Scale



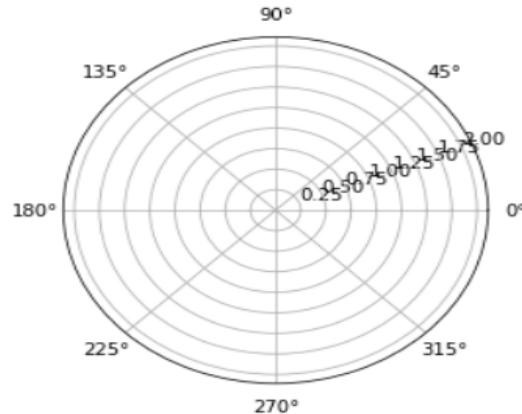
Log Scale

Plotting Polar Plots

- Matplotlib also allows us to plot a polar plot.
- A point in a polar plot is represented as (r, theta);
 - r is the distance from the origin.
 - theta is the angle along which r is measured.
- Use the .polar() function to plot a polar plot.

```
In [78]: theta = np.arange(0, (2 * np.pi), 0.01) # generating an array of evenly spaced floats
r = 2

for radian in theta:
    plt.polar(radian, r)
```



Handling Dates (1/2)

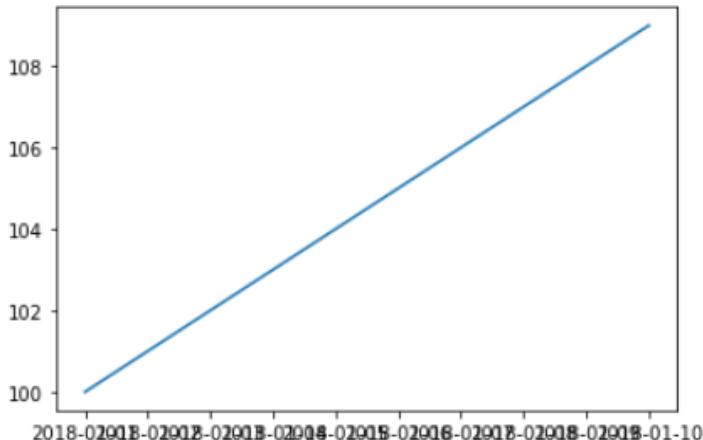
- Sometimes, there are too many values on the x-axis and it becomes difficult to distinguish them in the graph.
- This also happens when you have dates on the x-axis and they overlap as shown in the figure.

```
In [94]: dates = ['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04', '2018-01-05',
              '2018-01-06', '2018-01-07', '2018-01-08', '2018-01-09', '2018-01-10']
```

```
values = [100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
```

```
plt.plot(dates, values)
```

```
Out[94]: [<matplotlib.lines.Line2D at 0x7efef13c0d60>]
```



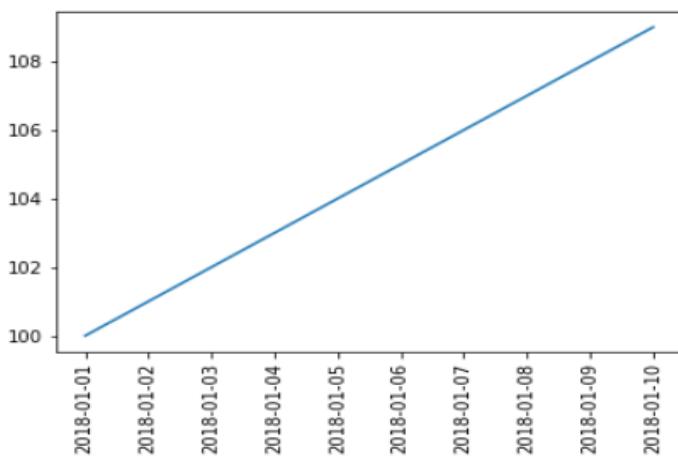
Handling Dates (2/2)

- We can avoid this by changing the orientation of the values on the x-axis using the `.xticks()` function.
 - `.xticks()` has a rotation parameter which can be used to rotate the values on the x-axis by a suitable angle.

```
In [96]: dates = ['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04', '2018-01-05',
               '2018-01-06', '2018-01-07', '2018-01-08', '2018-01-09', '2018-01-10']
values = [100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

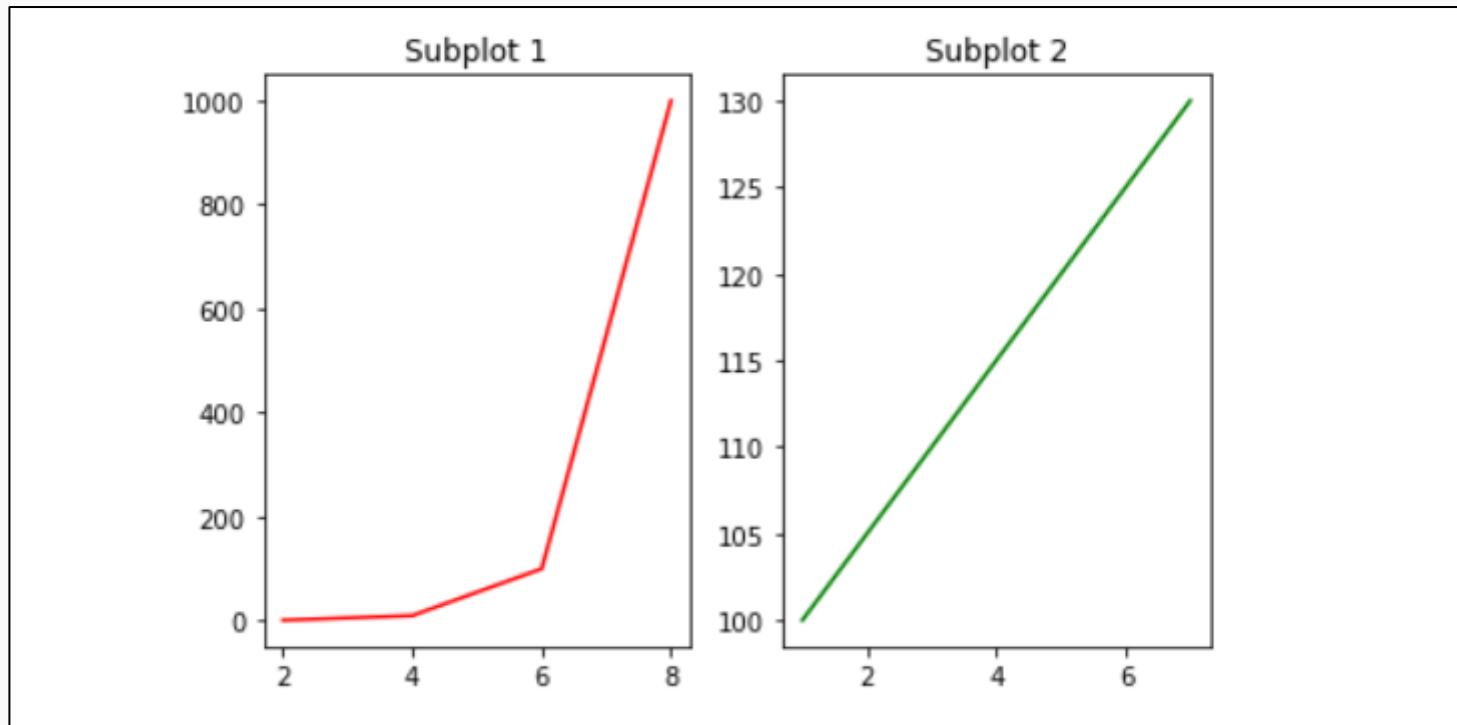
plt.xticks(rotation=90)
plt.plot(dates, values)
```

```
Out[96]: [<matplotlib.lines.Line2D at 0x7efef0dae430>]
```



Creating Multiple Subplots in One Figure (1/3)

- We saw how we can plot multiple plots on the same chart, but sometimes we would like to have different charts for each plot.
- What we want to have actually is multiple subplots in the same figure, as shown in the given figure.



Creating Multiple Subplots in One Figure (2/3)

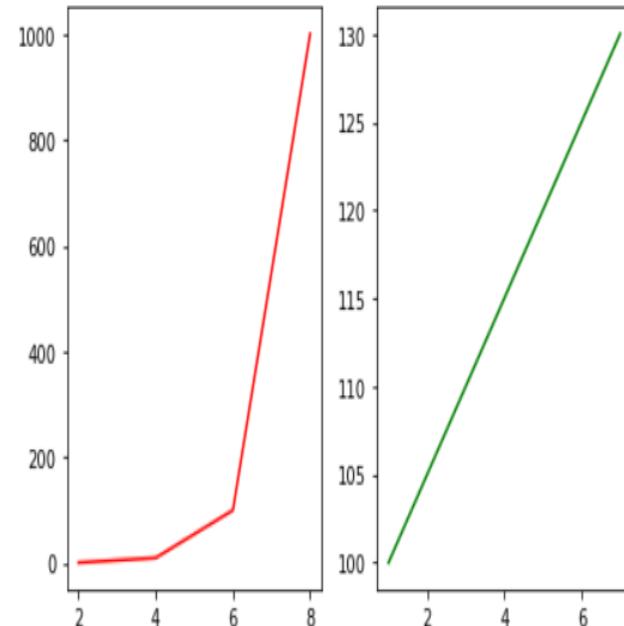
- To create subplots, we use the `.subplot()` function before plotting each graph.
- `.subplot()` takes 3 parameters;
 - First parameter is the number of rows you want to have in your figure.
 - Second parameter is the number of columns you want to have in your figure.
 - Third parameter is the id/position of the plot in the figure.

```
In [101]: x1_axis = [2, 4, 6, 8]
y1_axis = [1, 10, 100, 1000]
x2_axis = [1, 3, 5, 7]
y2_axis = [100, 110, 120, 130]

plt.subplot(1, 2, 1)
plt.plot(x1_axis, y1_axis, 'r')

plt.subplot(1, 2, 2)
plt.plot(x2_axis, y2_axis, 'g')

plt.tight_layout()
```



Creating Multiple Subplots in One Figure (3/3)

- We can have separate x and y labels as well as title for each subplot in the figure.

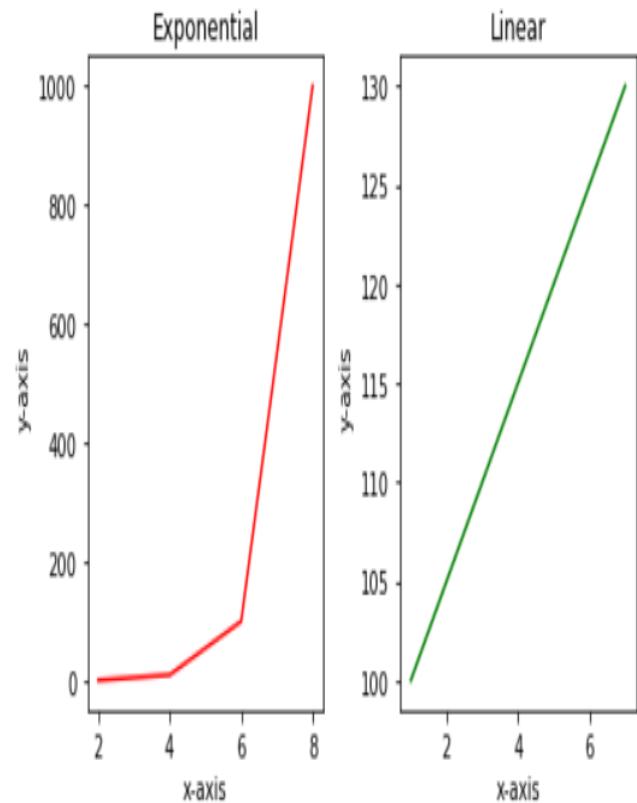
In [103]:

```
x1_axis = [2, 4, 6, 8]
y1_axis = [1, 10, 100, 1000]
x2_axis = [1, 3, 5, 7]
y2_axis = [100, 110, 120, 130]

plt.subplot(1, 2, 1)
plt.title('Exponential')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.plot(x1_axis, y1_axis, 'r')

plt.subplot(1, 2, 2)
plt.title('Linear')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.plot(x2_axis, y2_axis, 'g')

plt.tight_layout()
```



Resources

- https://www.w3schools.com/python/matplotlib_intro.asp

EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis

- Exploratory Data Analysis, or EDA is a way to analyze data sets, often using data visualization methods, to summarize the main characteristics of the data.
- EDA helps in better understanding the different features of the data, the relationship between them, and in determining the statistical techniques appropriate for the data set.
- matplotlib and seaborn libraries are pretty good for EDA.

Univariate Analysis (1/7)

- A dataset may contain one or more features/variables/columns.
- Univariate Analysis provides summary statistics only on one variable.
- Univariate Analysis only describes the data and helps identify any patterns in the data.

Categorical vs Continuous Data

- There are two types of data;
 - Categorical/Discrete, e.g., spam vs no spam, male vs female
 - Continuous, e.g., Age of population
- EDA is performed differently on the two types of data.

Univariate Analysis (2/7)

- Consider the following dataframe containing 5 features containing information about iris plants.
- The last column is categorical, while the rest are continuous.

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

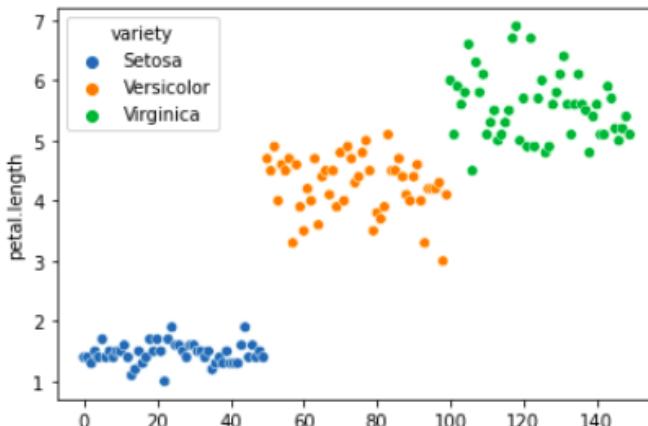
150 rows × 5 columns

Univariate Analysis – Continuous Data (3/7)

Scatter Plot

- We can use a number of graphs to perform EDA on continuous data.
- In the given figure, we use the `.scatterplot()` function of the seaborn library to plot the ‘petal.length’ column of the dataframe.
- The hue parameter assigns different color to the data points based on the category they belong to in the column specified in the hue parameter.

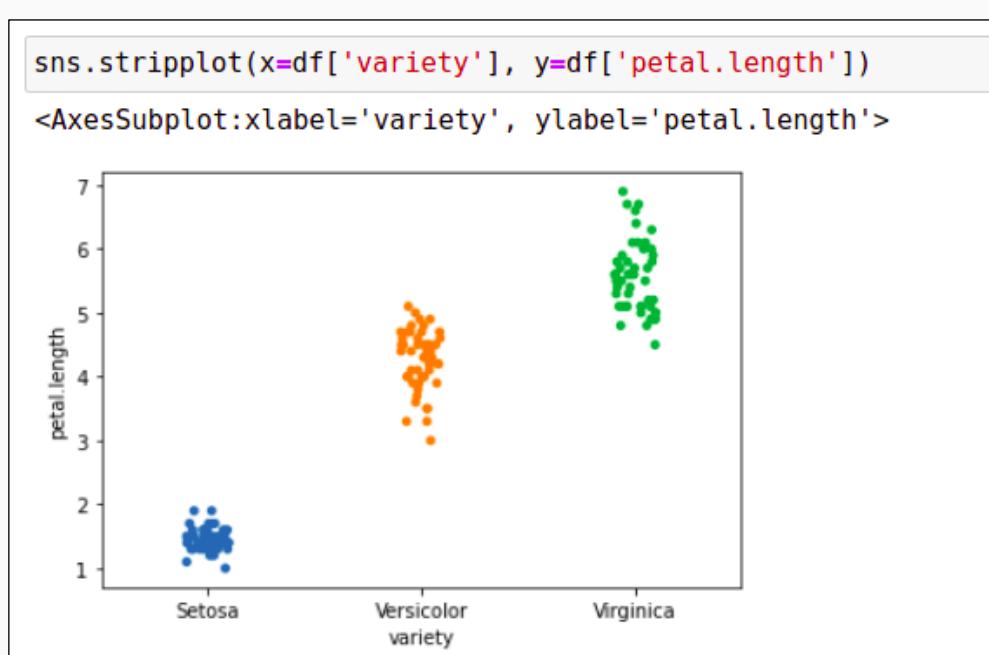
```
x_axis = df.index  
y_axis = df['petal.length']  
sns.scatterplot(x=x_axis, y=y_axis, hue=df['variety'])  
  
<AxesSubplot:ylabel='petal.length'>
```



Univariate Analysis – Continuous Data (4/7)

Strip Plot

- Strip plots are also a good way to analyze the distribution of the variables for each category.
- In the given figure, we have plotted ‘petal.length’ column for each category in the ‘variety’ column.
- On the Y-axis, we have the distribution of each category, and on the X-axis we have the categories.



Univariate Analysis – Continuous Data (5/7)

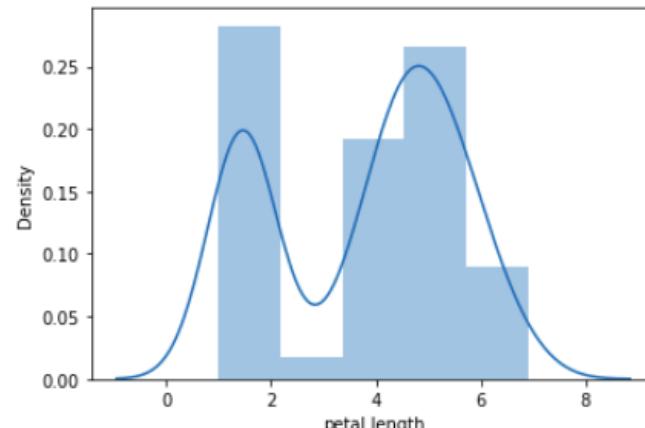
Distribution Plot

- To find the distribution of a variable/feature/column, use the `.distplot()` function of the seaborn library.
- In this figure, we plot the distribution of the ‘petal.length’ column.

```
sns.distplot(df['petal.length'])
```

```
/home/waqar/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
warnings.warn(msg, FutureWarning)
```

```
<AxesSubplot:xlabel='petal.length', ylabel='Density'>
```

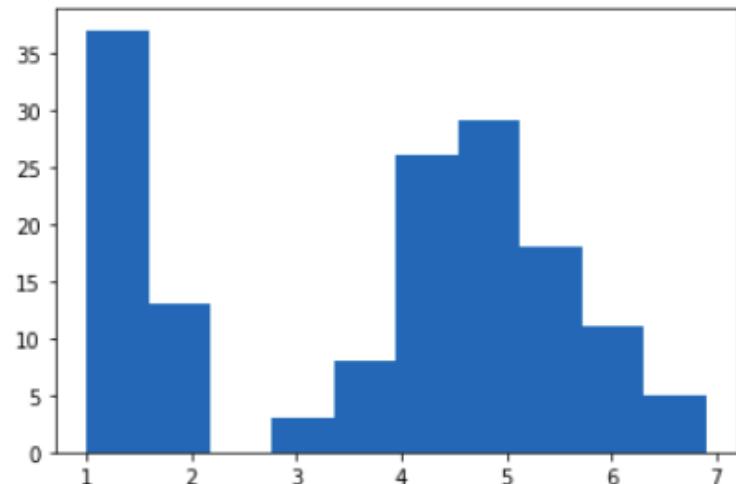


Univariate Analysis – Continuous Data (6/7)

Histograms

- To analyze the frequency of values, we can plot a histogram using the `.hist()` method of the `matplotlib` library.
- In this figure, we plot the frequency of values in the ‘petal.length’ column.

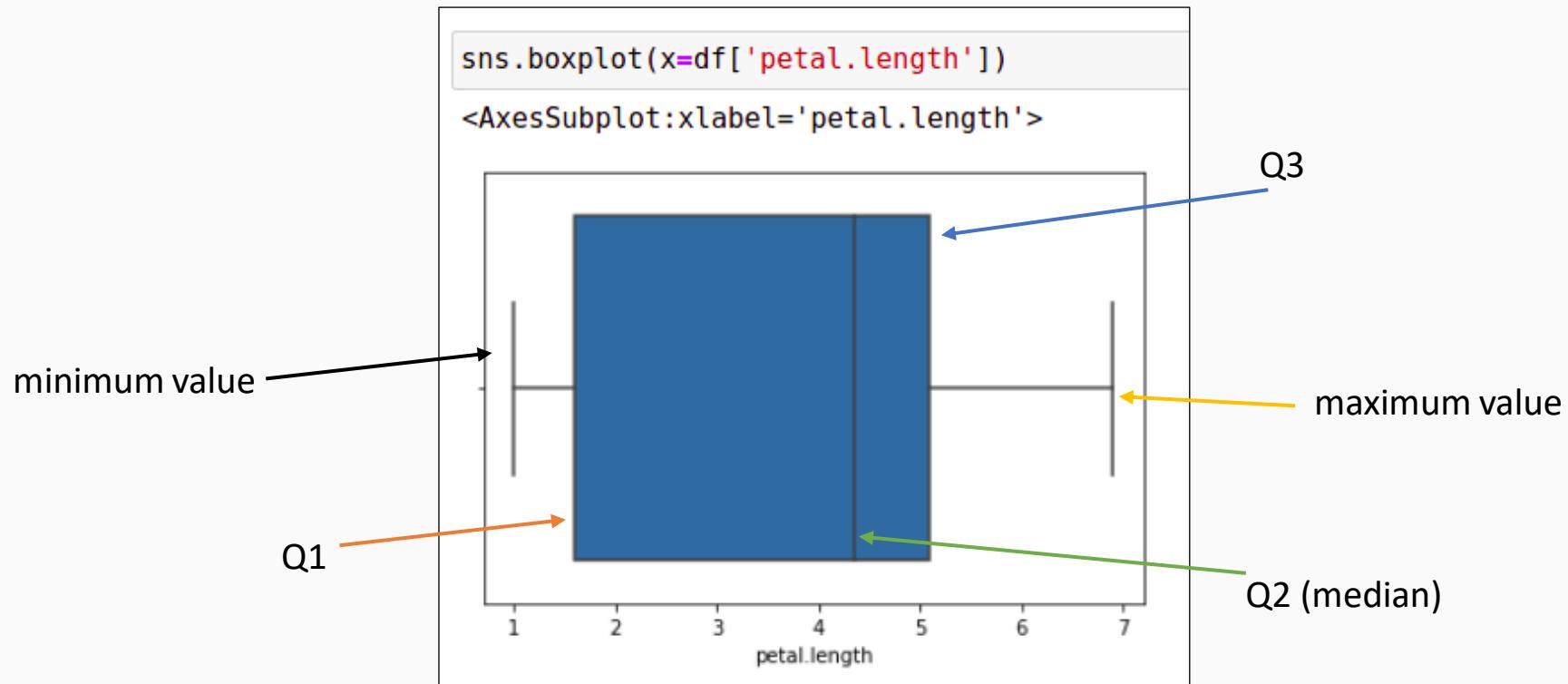
```
plt.hist(df['petal.length'])  
  
(array([37., 13., 0., 3., 8., 26., 29., 18., 11., 5.]),  
 array([1. , 1.59, 2.18, 2.77, 3.36, 3.95, 4.54, 5.13, 5.72, 6.31, 6.9 ]),  
 <BarContainer object of 10 artists>)
```



Univariate Analysis – Continuous Data (7/7)

Box Plot

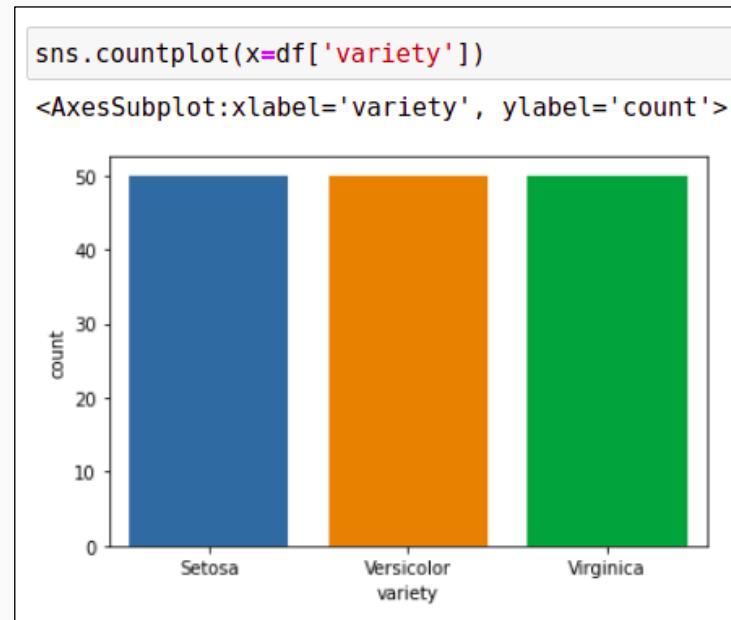
- A box plot provides great insights into the data using 5-number summary; minimum value, first quartile, second quartile, third quartile, and maximum value.
- We can use the `.boxplot()` function of either matplotlib or seaborn.



Univariate Analysis – Categorical Data (1/2)

Count Plot

- We can use count plots to perform EDA on categorical data.
- The `.countplot()` function of the seaborn library plots the total count of each value as a bar chart.
- In the given figure, we see that we have equal instances of each category in the dataframe.



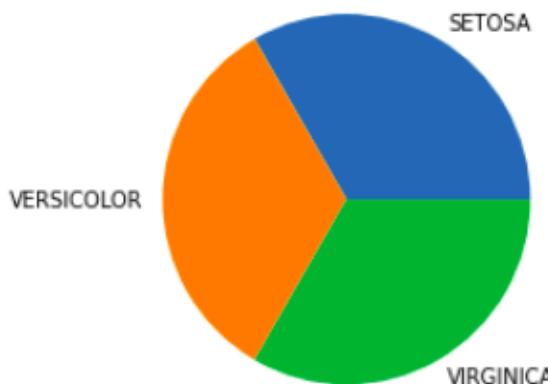
Univariate Analysis – Categorical Data (2/2)

Pie Chart

- We can also visualize the proportion of each category using a pie chart as shown in the figure.

```
: _labels = ['SETOSA', 'VERSICOLOR', 'VIRGINICA']
plt.pie(df['variety'].value_counts(), labels = _labels)

([<matplotlib.patches.Wedge at 0x7f18c505e5b0>,
 <matplotlib.patches.Wedge at 0x7f18c505ea90>,
 <matplotlib.patches.Wedge at 0x7f18c505ef10>],
 [Text(0.5499999702695115, 0.9526279613277875, 'SETOSA'),
 Text(-1.0999999999999954, -1.0298943258065002e-07, 'VERSICOLOR'),
 Text(0.5500001486524352, -0.9526278583383436, 'VIRGINICA')])
```



Bivariate Analysis – Continuous & Continuous (1/4)

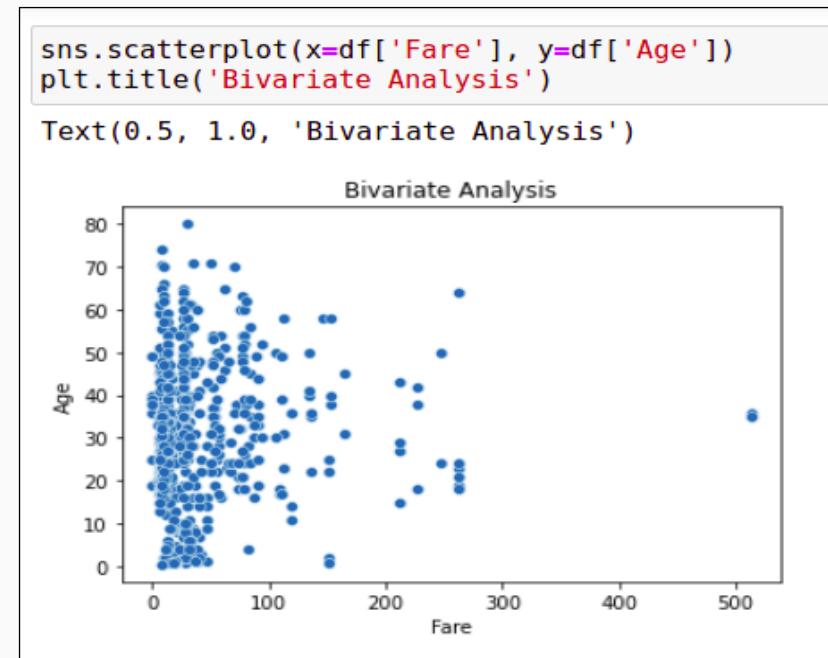
- Bivariate Analysis is used to study the relationship between exactly two variables/features/columns of the data set.
 - Consider the following dataframe.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W.C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

Bivariate Analysis – Continuous & Continuous (2/4)

Scatter Plot

- If the two variables in question are both continuous, we can plot a scatter plot between them to get an idea of their distribution.
- In the given figure, we plot the ‘Age’ column of the dataframe against the ‘Fare’ column, both of which are continuous.
- It is evident that the two features are independent of each other.



Bivariate Analysis – Continuous & Continuous (3/4)

Correlation

- We can also find the correlation between two continuous features.
- If the correlation is high, the two features are significantly related.
- Use the `.corr()` function to compute correlation between two features.

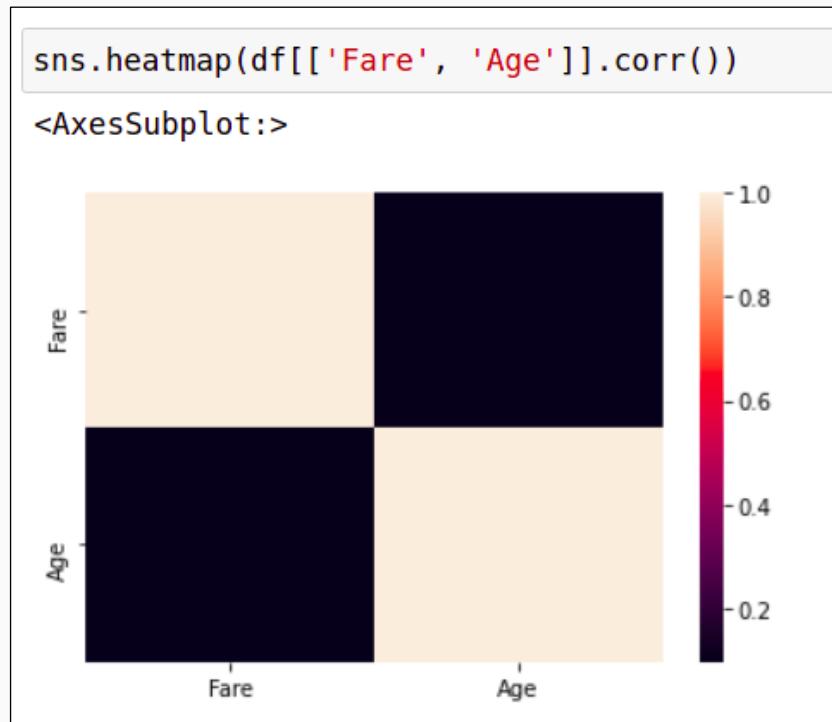
```
df[['Fare', 'Age']].corr()
```

	Fare	Age
Fare	1.000000	0.096067
Age	0.096067	1.000000

Bivariate Analysis – Continuous & Continuous (4/4)

Heatmp

- We can also use a heatmap to graphically visualize the correlation between something.
- Use the `.heatmap()` method of the seaborn library to create a heatmap.
- Provide the correlation dataframe that we created in the previous slide as an argument to the `.heatmap()`



Bivariate Analysis – Categorical & Categorical (1/3)

- Consider the following dataframe from the previous slide.
 - We will see how we can use bar plot to identify any relation between two categorical variables, namely ‘Pclass’ and ‘Survived’.

Bivariate Analysis – Categorical & Categorical (2/3)

- We first select the two columns that we are interested in, namely ‘Pclass’ and ‘Survived’.
- We then group this new dataframe based on ‘Pclass’ column and apply the `.sum()` function to find total number of survivors from each category in the Pclass.
- The output of this part is shown below.

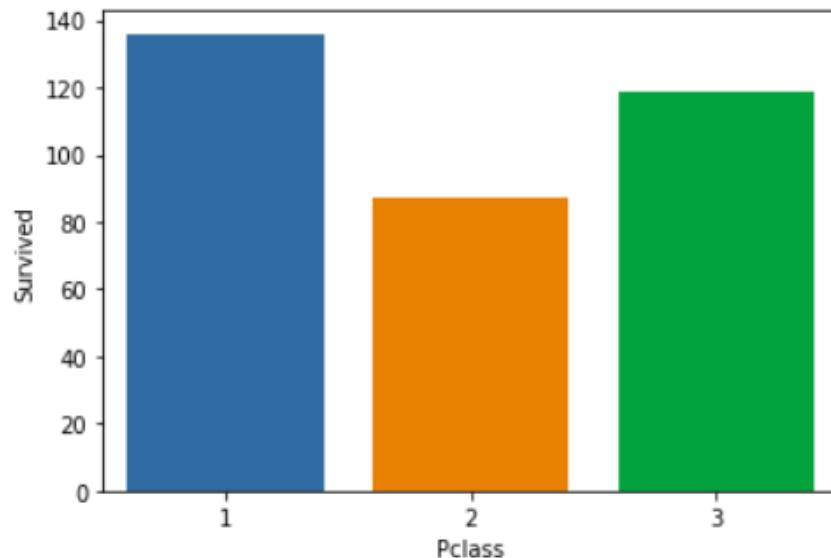
```
survived_ratio = df[['Pclass', 'Survived']].groupby('Pclass').sum()  
survived_ratio
```

Survived	
Pclass	
1	136
2	87
3	119

Bivariate Analysis – Categorical & Categorical (3/3)

- We then plot a bar plot, with ‘Pclass’ on the X-axis.
- As evident, more passengers in ‘Pclass’ 1 were able to survive than those from ‘Pclass’ 2 or 3.

```
sns.barplot(x=survived_ratio.index, y=survived_ratio['Survived'])  
<AxesSubplot:xlabel='Pclass', ylabel='Survived'>
```



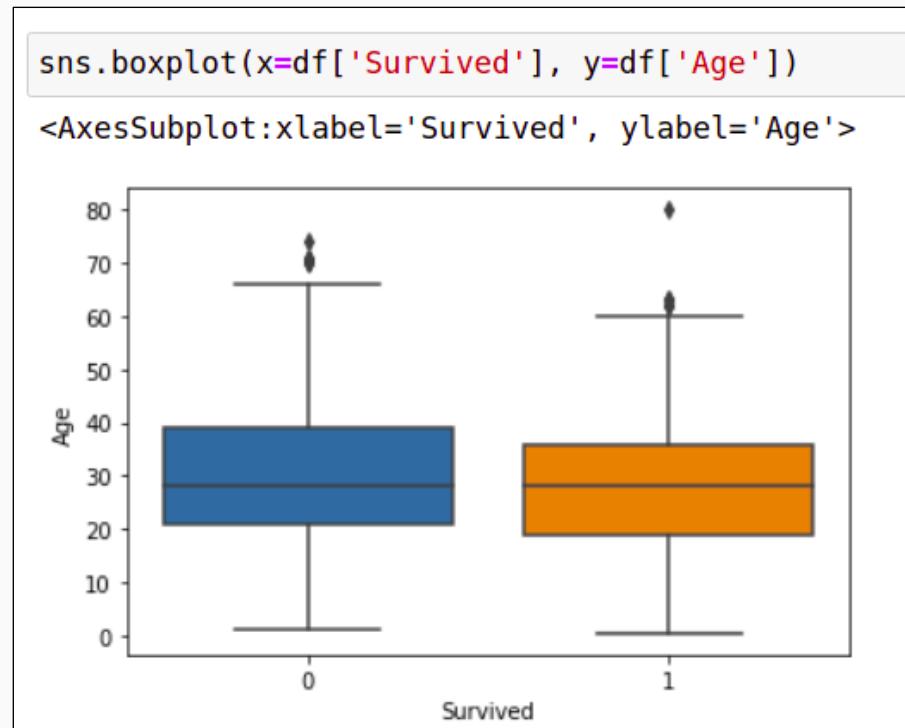
Bivariate Analysis – Continuous & Categorical (1/3)

- Sometimes, we want to find out the relationship between a continuous and a categorical variable.
 - Consider the following dataframe from the previous slide.

Bivariate Analysis – Continuous & Categorical (2/3)

Box Plot

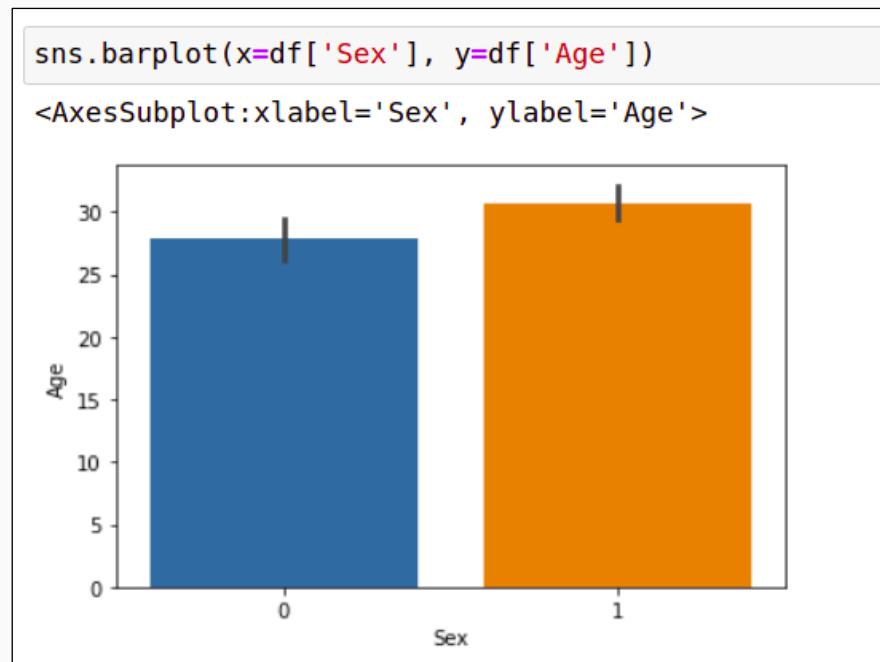
- We can use a box plot to perform EDA on continuous and categorical data.
- In the given figure, we plot 'Age' (continuous feature) against 'Survived' (categorical feature).
- The plot suggests that the younger people had a greater chance of survival.



Bivariate Analysis – Continuous & Categorical (3/3)

Bar Plot

- Bar plots can also be used for bivariate analysis of a continuous and a categorical feature.
- In the given figure we plot 'Age' (continuous feature) against 'Sex' (categorical feature). 1 in the 'Sex' column represents male and 0 represents 'female' passengers.
- The bar plot suggests that elderly passengers were mostly male.



Detecting Outliers

- Outliers/anomalies in the data are the observations that do not fit into the standard pattern of the data.
- In chapter 4, we discussed major techniques to detect outliers/anomalies e.g.;
 - Median-based anomaly detection
 - Mean-based anomaly detection
 - Z-score-based anomaly detection
 - IQR-based anomaly detection
- In this chapter, we will learn different ways to treat such outliers/anomalies in the data.

Outliers Treatment (1/3)

Trimming Outliers

- One naïve way of treating outliers is to remove them from the data. However, this approach is not very good.
- Outliers can be removed from the data in several ways.
- Consider the given Series. It is safe to say that the value 150 is an outlier in the data.

0	1
1	2
2	3
3	6
4	7
5	8
6	150
dtype: int64	

Outliers Treatment (2/3)

Trimming Outliers

- We delete the rows of the Series for which the absolute value of the Z-score is bigger than 1.5, which means that the values lie outside of 1.5 standard deviations of the data.

```
x = pd.Series([1, 2, 3, 6, 7, 8, 150])
mean = x.mean()
std = x.std()
z_scores = abs((x-mean)/std)
z_scores
```

0	0.441104
1	0.422941
2	0.404778
3	0.350288
4	0.332125
5	0.313962
6	2.265198

dtype: float64

```
outliers_removed = x[z_scores <= 1.5]
outliers_removed
```

0	1
1	2
2	3
3	6
4	7
5	8

dtype: int64

Outliers Treatment (3/3)

Mean/Median Imputation

- We can also replace outliers with either mean or median.
- In the given figure, we detect outliers in the data using Z-score and replace them with the median value of the Series.

```
x = pd.Series([1, 2, 3, 6, 7, 8, 150])
mean = x.mean()
std = x.std()
median = np.median(x)
z_scores = abs((x-mean)/std)
median

6.0
```

```
x[z_scores > 1.5] = median
x

0    1
1    2
2    3
3    6
4    7
5    8
6    6
dtype: int64
```

Categorical Variable Transformation (1/3)

- We discussed about numerical variable transformation in chapter 4 using normalization and standardization.
- In this chapter we will discuss categorical variable transformation.
- There are several ways to transform categorical variable to make them more meaningful for machines, we will discuss only some of them.
- Consider the following dataframe. We will transform the 'Sex' variable.

	Name	Sex
0	Braund, Mr. Owen Harris	male
1	Cumings, Mrs. John Bradley (Florence Briggs Th... e	female
2	Heikkinen, Miss. Laina	female
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female
4	Allen, Mr. William Henry	male

Categorical Variable Transformation (2/3)

Label Encoding

- In label encoding, we replace categorical data with numbers.
- We replace male with 1 and female with 0.

```
df['Sex'].replace({'male':1, 'female':0}, inplace=True)  
df
```

	Name	Sex
0	Braund, Mr. Owen Harris	1
1	Cumings, Mrs. John Bradley (Florence Briggs Th... ...	0
2	Heikkinen, Miss. Laina	0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0
4	Allen, Mr. William Henry	1

Categorical Variable Transformation (3/3)

Frequency Encoding

- In this encoding method, we replace each value in the categorical variable by its frequency.
- In the given figure, we replaced male and female labels in the 'Sex' column with their frequencies.

```
freq = df['Sex'].value_counts()/len(df['Sex'])
freq
female    0.6
male      0.4
Name: Sex, dtype: float64
```

```
df['Sex'].replace({'male':freq['male'], 'female':freq['female']}, inplace=True)
df
```

	Name	Sex
0	Braund, Mr. Owen Harris	0.4
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0.6
2	Heikkinen, Miss. Laina	0.6
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0.6
4	Allen, Mr. William Henry	0.4

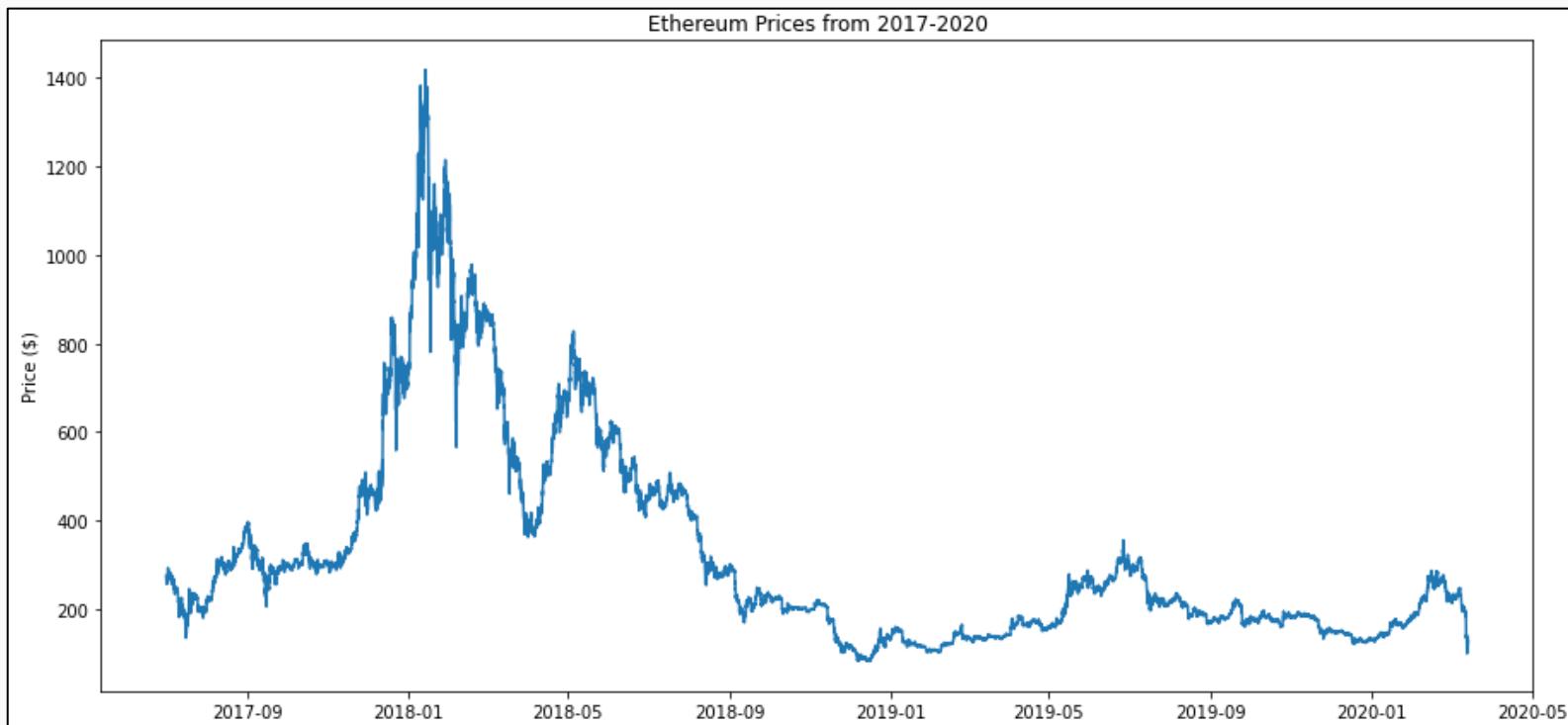
Resources

- <https://www.kaggle.com/residentmario/univariate-plotting-with-pandas>
- <https://purnasaigudikandula.medium.com/exploratory-data-analysis-beginner-univariate-bivariate-and-multivariate-haberman-dataset-2365264b751>

TIME SERIES

What is Time Series

- A time series is a sequence of data measured over regular time intervals.
- By analyzing a time series, we can identify the pattern in the data and develop a forecasting model.
- Python provides a lot of functionality to analyze a time series.



Python Libraries for Time Series Analysis

- Python has a lot of libraries for time series analysis.
- Some of these libraries are used in Machine Learning and hence are out of the scope of this course.
- We will be using the following libraries for time series analysis;
 - Pandas
 - NumPy
 - datetime
 - Matplotlib

How to Get Time Series?

- As mentioned earlier, a time series is just a set of values measured over regular intervals of time.
- Generally, any dataset with values measured over time can be treated as a time series.
- Examples of such dataset could include but is not limited to;
 - Stocks data
 - Weather data
 - Patient health evolution data
- In Python, the data type of dates and times in a time series is either datetime or Timestamp.
- In this course, we will see how can we download a time series and how can we convert a dataset with values measured over time into a time series.

Getting Stocks Data using yfinance (1/5)

- yfinance is a Python library that allows downloading stocks data from Yahoo Finance for a specified period of time as a time series.
- You can also download a dataset from their website directly <https://finance.yahoo.com/>



Getting Stocks Data using yfinance (2/5)

- To use yfinance Python library, we first have to install it via the following command;
pip install yfinance
- Once the library has been installed, we can import it into the Jupyter Notebook.
- yfinance is commonly imported as `yf`.

```
pip install yfinance
Collecting yfinance
  Note: you may need to restart the kernel to use updated packages.

    Downloading yfinance-0.1.70-py2.py3-none-any.whl (26 kB)
      Requirement already satisfied: lxml>=4.5.1 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (4.6.3)
      Requirement already satisfied: numpy>=1.15 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (1.20.3)
      Requirement already satisfied: requests>=2.26 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (2.26.0)
      Collecting multitasking>=0.0.7
        Downloading multitasking-0.0.10.tar.gz (8.2 kB)
          Requirement already satisfied: pandas>=0.24.0 in c:\users\22100199\anaconda3\lib\site-packages (from yfinance) (1.3.4)
          Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\22100199\anaconda3\lib\site-packages (from pandas>=0.24.0->yfinance) (2.8.2)
          Requirement already satisfied: pytz>=2017.3 in c:\users\22100199\anaconda3\lib\site-packages (from pandas>=0.24.0->yfinance) (2021.3)
          Requirement already satisfied: six>=1.5 in c:\users\22100199\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.16.0)
          Requirement already satisfied: idna<4,>=2.5 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (3.2)
          Requirement already satisfied: certifi>=2017.4.17 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2021.10.8)
          Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2.0.4)
          Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\22100199\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (1.26.7)
          Building wheels for collected packages: multitasking
            Building wheel for multitasking (setup.py): started
            Building wheel for multitasking (setup.py): finished with status 'done'
            Created wheel for multitasking: filename=multitasking-0.0.10-py3-none-any.whl size=8500 sha256=7d5ad1da18be483c99d9f99c43e6f458099dd09b5ec65cf632200300198faee6
            Stored in directory: c:\users\22100199\appdata\local\pip\cache\wheels\f2\b5\2c\59ba95dcf854e542944c75fe3da584e4e3833b319735a0546c
            Successfully built multitasking
            Installing collected packages: multitasking, yfinance
            Successfully installed multitasking-0.0.10 yfinance-0.1.70

import yfinance as yf
```

Getting Stocks Data using yfinance (3/5)

- After importing the library, we can download stock data for different companies using the `.download()` method.
- We pass a start and an end date in the format ‘YYYY-MM-DD’.
- If there is no data available for a certain date, that date would be skipped.

Getting Stocks Data using yfinance (4/5)

- In the given example, we download the stocks data for Apple (abbreviated as AAPL in Yahoo Finance) from 1st Jan, 2022 to 31st Jan, 2022.

```
df = yf.download('AAPL', start='2022-01-01', end='2022-01-31')
df.head(10)

[*****100*****] 1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-12-31	178.089996	179.229996	177.259995	177.570007	177.344055	64062300
2022-01-03	177.830002	182.880005	177.710007	182.009995	181.778397	104487900
2022-01-04	182.630005	182.940002	179.119995	179.699997	179.471344	99310400
2022-01-05	179.610001	180.169998	174.639999	174.919998	174.697418	94537600
2022-01-06	172.699997	175.300003	171.639999	172.000000	171.781143	96904000
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800

Getting Stocks Data using yfinance (5/5)

- As you can see, the values of the index column (Date) of the dataset in the previous slide have the type Timestamp, indicating that it is a Python time series.

```
type(df.index[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

Quiz Time

1. Stocks data over a period of regular time intervals is a time series?

- True
- False

Quiz Time

1. Stocks data over a period of regular time intervals is a time series?

- True
- False

Converting a Dataset To Time Series (1/6)

- Time Series data is generally stored as csv file having a column representing date or time or both.
- It can be imported in Python using the `.read_csv()` function of Pandas library.
- We can use the `to_datetime()` function of the Pandas library to convert the values of the date column from string to Timestamp.

Converting a Dataset To Time Series (2/6)

- Consider the following time series. The type of each value of 'date' (index) is a string as shown in the figure on the right.

df	
	value
date	
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740
204 rows × 1 columns	

```
type(df.index[0])  
str
```

Converting a Dataset To Time Series (3/6)

- To be able to use the time series functionality available in Python, we have to convert the values of 'date' to Timestamp.
- This can be done using the `to_datetime()` function of Pandas.

```
df.index = pd.to_datetime(df.index)
df
```

	value
date	
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

204 rows × 1 columns

```
type(df.index[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

Converting a Dataset To Time Series (4/6)

- In the previous example, the data in the ‘date’ column was in a format that Pandas could easily convert. However, this is not always the case.
- Consider the following dataframe for example. ‘Date’ values are strings.

Date	Symbol	Open	High	Low	Close	Volume
2020-03-13 08-PM	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
2020-03-13 07-PM	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2020-03-13 06-PM	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
2020-03-13 05-PM	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
2020-03-13 04-PM	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...
2017-07-01 03-PM	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
2017-07-01 02-PM	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
2017-07-01 01-PM	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
2017-07-01 12-PM	ETHUSD	275.01	275.01	271.00	274.83	824362.87
2017-07-01 11-AM	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 6 columns

Converting a Dataset To Time Series (5/6)

- Converting the values of ‘Date’ (index) results in a ParserError.

```
df.index = pd.to_datetime(df.index)
df

-----
TypeError                                 Traceback (most recent call last)
File ~\Anaconda3\lib\site-packages\pandas\core\arrays\datetimes.py:2187, in objects_to_datetime64ns(data, dayfirst, yearfirst, utc, errors, require_iso8601, allow_object, allow_mixed)
    2186     try:
-> 2187         values, tz_parsed = conversion.datetime_to_datetime64(data.ravel("K"))
    2188         # If tzaware, these values represent unix timestamps, so we
    2189         # return them as i8 to distinguish from wall times

File ~\Anaconda3\lib\site-packages\pandas\_libs\tslibs\conversion.pyx:359, in pandas._libs.tslibs.conversion.datetime_to_datetime64()
TypeError: Unrecognized value type: <class 'str'>

During handling of the above exception, another exception occurred:

ParserError                                Traceback (most recent call last)
Input In [32], in <module>
----> 1 df.index = pd.to_datetime(df.index)
      2 df
```

Converting a Dataset To Time Series (6/6)

- The error arises because Pandas does not know the format of the string values.
- We can use the ‘format’ parameter to resolve this issue.
- You can find a list of all format codes at:

<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior>

```
df.index = pd.to_datetime(df.index, format='%Y-%m-%d %I-%p')
print(type(df.index[0]))
df
```

	Symbol	Open	High	Low	Close	Volume
Date						
2020-03-13 20:00:00	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
2020-03-13 19:00:00	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2020-03-13 18:00:00	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
2020-03-13 17:00:00	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
2020-03-13 16:00:00	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...
2017-07-01 15:00:00	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
2017-07-01 14:00:00	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
2017-07-01 13:00:00	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
2017-07-01 12:00:00	ETHUSD	275.01	275.01	271.00	274.83	824362.87
2017-07-01 11:00:00	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 6 columns

Working With Time Series (1/7)

- In this section we will learn to use Pandas time series functionality.
- We will use the following time series containing stock data of Apple from 1st of Jan, 2022 to 4th Feb, 2022.

```
df = yf.download('AAPL', start='2022-01-01', end='2022-02-04')
df.head(10)
```

```
[*****100*****] 1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-12-31	178.089996	179.229996	177.259995	177.570007	177.344055	64062300
2022-01-03	177.830002	182.880005	177.710007	182.009995	181.778397	104487900
2022-01-04	182.630005	182.940002	179.119995	179.699997	179.471344	99310400
2022-01-05	179.610001	180.169998	174.639999	174.919998	174.697418	94537600
2022-01-06	172.699997	175.300003	171.639999	172.000000	171.781143	96904000
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800

Working With Time Series (2/7)

.day_name()

- The first thing you might have noticed is that there is no data available for 1st and 2nd Jan, 2022.
- Let's see whether these two were business days or not.
- We will use the `.day_name()` function to find out what day was on 1st Jan.
- Remember that this function can only be applied on a value of time datetime or Timestamp.
- 1st Jan, 2022 was a Saturday and hence 2nd Jan, 2022 was a Sunday. No business!

```
jan1 = pd.to_datetime('2022, 1, 1')
jan1.day_name()
```

```
'Saturday'
```

Working With Time Series (3/7)

Max/Min Date

- Sometimes, our timeseries data might not be in order.
- In such a case, we would be interested in what is the maximum and the minimum datetime (or Timestamp) in the dataset.
- This can be done using the .max() and .min() functions.

```
df.index.max()
```

```
Timestamp('2022-02-03 00:00:00')
```

```
df.index.min()
```

```
Timestamp('2021-12-31 00:00:00')
```

Working With Time Series (4/7)

datetime Range

- Next, let's see what is the datetime range in our dataset.
- This is as simple as subtracting minimum value of datetime (Timestamp) from its maximum value.
- Note that this will only work if you have datetime or Timestamp values in your dataset and not strings.

```
timePeriod = df.index.max() - df.index.min()  
timePeriod
```

```
Timedelta('34 days 00:00:00')
```

Working With Time Series (5/7)

Time-based Indexing

- Pandas provides time-based indexing for time series, i.e., accessing data using datetime.
- We can select data on specific dates and times using `.loc`.
- In the given example, we get the data on 7th Jan, 2022.

```
df.loc['2022-01-07']

Open          1.728900e+02
High          1.741400e+02
Low           1.710300e+02
Close          1.721700e+02
Adj Close      1.719509e+02
Volume         8.658010e+07
Name: 2022-01-07 00:00:00, dtype: float64
```

Working With Time Series (6/7)

Time-based Indexing

- Let's select multiple dates using slicing with `.loc`.

```
df.loc['2022-01-07':'2022-01-14']
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-01-07	172.889999	174.139999	171.029999	172.169998	171.950928	86580100
2022-01-10	169.080002	172.500000	168.169998	172.190002	171.970901	106765600
2022-01-11	172.320007	175.179993	170.820007	175.080002	174.857224	76138300
2022-01-12	176.119995	177.179993	174.820007	175.529999	175.306641	74805200
2022-01-13	175.779999	176.619995	171.789993	172.190002	171.970901	84505800
2022-01-14	171.339996	173.779999	171.089996	173.070007	172.849792	80355000

Working With Time Series (7/7)

Time-based Indexing

- We can also partially match a datetime using `.loc`.
- In the given example, we get the data from the time series for the month of February only.
- We pass the string ‘2022-02’ to get the data of February.

```
df.loc['2022-02']
```

Date	Open	High	Low	Close	Adj Close	Volume
2022-02-01	174.009995	174.839996	172.309998	174.610001	174.387817	86213900
2022-02-02	174.750000	175.880005	173.330002	175.839996	175.616257	84914300
2022-02-03	174.479996	176.240005	172.119995	172.899994	172.679993	89418100

Quiz Time

1. Consider the given dataframe called df. What value will the following command return?

```
df.loc['2008-02-01']  
a) 29.665356  
b) 21.654285  
c) 18.264945  
d) Error
```

date	value
2008-01-01	29.665356
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

Quiz Time

1. Consider the given dataframe called df. What value will the following command return?

```
df.loc['2008-02-01']
```

- a) 29.665356
- b) 21.654285
- c) 18.264945
- d) Error

date	value
2008-01-01	29.665356
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

Visualizing a Time Series (1/3)

- Using Matplotlib and seaborn, we can visualize a time series.
- Let's see how the stock prices of Apple varied over the last 5 years (2017-2021).
- We will begin by downloading the stock data from 2017 to 2021.

```
df = yf.download('AAPL', start='2017-01-01', end='2021-12-31')
df
```

[*****100%*****] 1 of 1 completed

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	28.950001	29.082500	28.690001	29.037500	27.297691	115127600
2017-01-04	28.962500	29.127501	28.937500	29.004999	27.267141	84472400
2017-01-05	28.980000	29.215000	28.952499	29.152500	27.405800	88774400
2017-01-06	29.195000	29.540001	29.117500	29.477501	27.711329	127007600
2017-01-09	29.487499	29.857500	29.485001	29.747499	27.965153	134247600
...
2021-12-23	175.850006	176.850006	175.270004	176.279999	176.055695	68356600
2021-12-27	177.089996	180.419998	177.070007	180.330002	180.100540	74919600
2021-12-28	180.160004	181.330002	178.529999	179.289993	179.061859	79144300
2021-12-29	179.330002	180.630005	178.139999	179.380005	179.151749	62348900
2021-12-30	179.470001	180.570007	178.089996	178.199997	177.973251	59773000

1258 rows × 6 columns

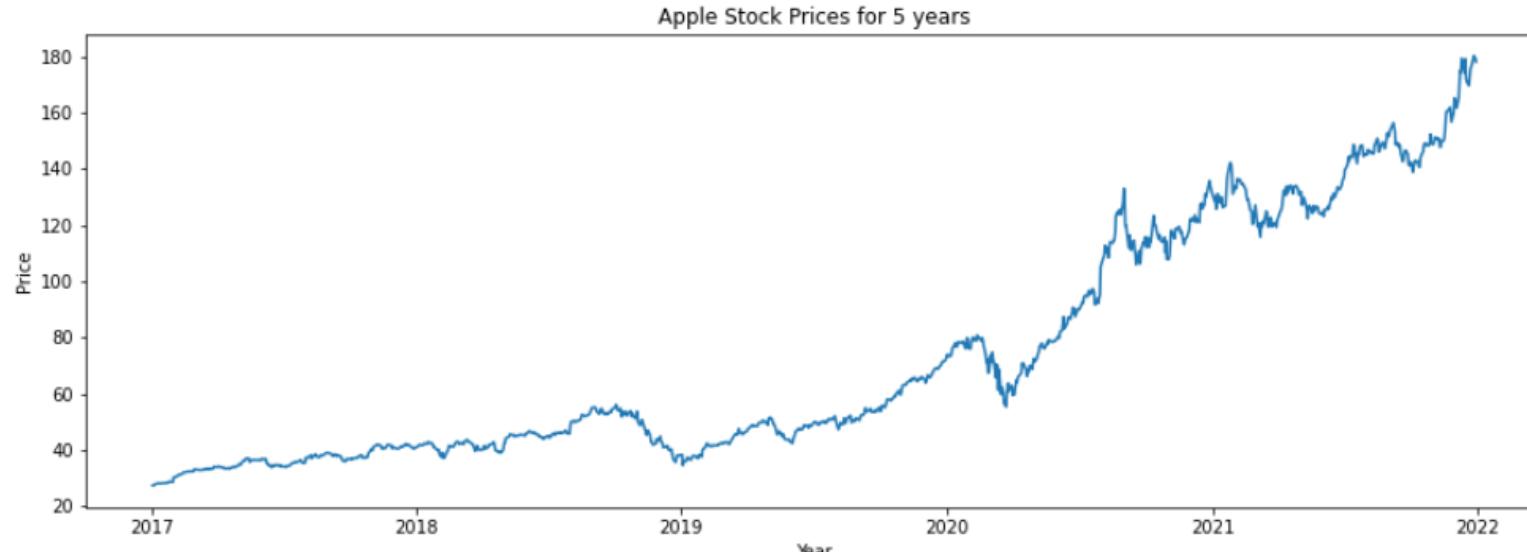
Visualizing a Time Series (2/3)

- Next, let's use the `.plot()` function of the `matplotlib.pyplot` to plot the 'Adj Close' time series.
- We can see that the stock prices of Apple have increased drastically since 2017.

```
plt.figure(figsize=(15, 5))
plt.title('Apple Stock Prices for 5 years')
plt.xlabel('Year')
plt.ylabel('Price')

plt.plot(df['Adj Close'])

[<matplotlib.lines.Line2D at 0x2639bfe75b0>]
```



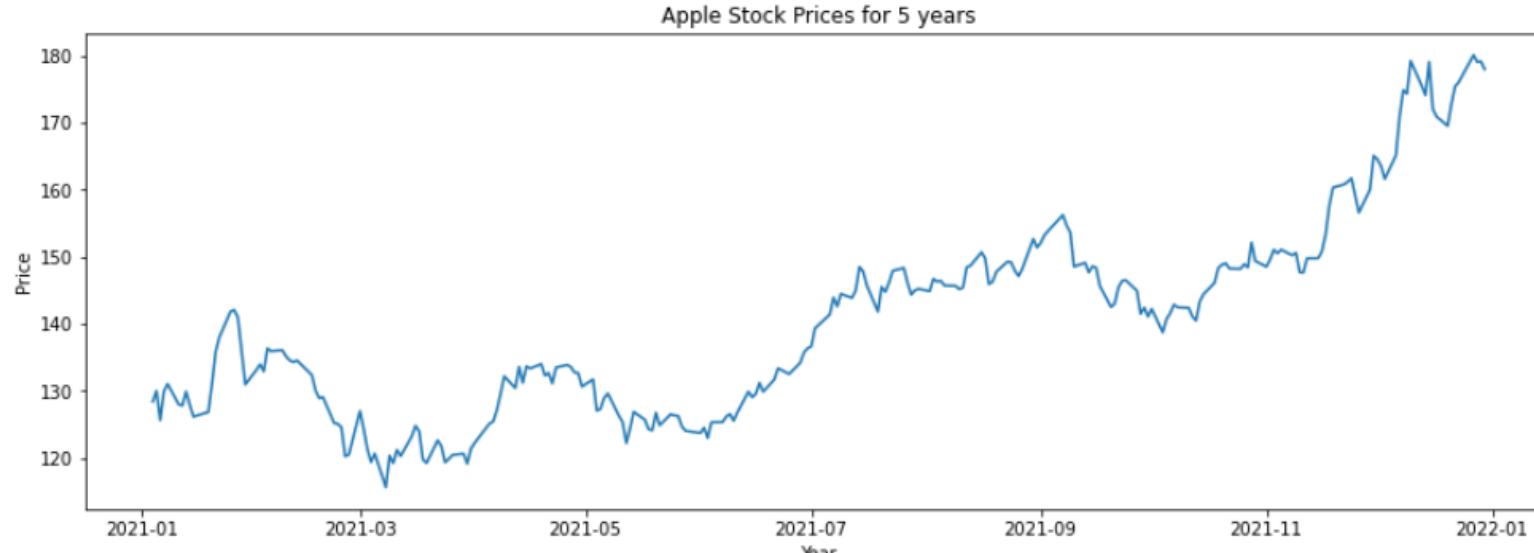
Visualizing a Time Series (3/3)

- We can also see how the Apple stock prices have varied over 2021 alone.
- We use the time-based indexing to get the year 2021 from the dataset.
- We then plot the ‘Adj Close’ time series.

```
plt.figure(figsize=(15, 5))
plt.title('Apple Stock Prices for 5 years')
plt.xlabel('Year')
plt.ylabel('Price')

plt.plot(df.loc['2021']['Adj Close'])

[<matplotlib.lines.Line2D at 0x2639c3010d0>]
```



Thank You!