Machine Learning using Python 1- Installation & Programming Basics

Neelotpal Chakraborty

Department of Computer Science and Engineering

Jadavpur University

Introduction To Python

- Python is an **open source, interpreted, high-level**, **general-purpose** programming language.
- Python's design philosophy emphasizes **code readability** with its notable use of significant **whitespace**.
- Python is **dynamically typed** and **garbage-collected** language**.**
- Python was conceived in the late **1980s** as a successor to the **ABC language**.
- Python was Created by **Guido van Rossum** and first released in **1991**.
- **Python 2.0**, released in **2000**,
  - introduced features like list comprehensions and a garbage collection system with reference counting.
- **Python 3.0** released in **2008** and current version of python is **3.8.3** (as of June-2020).
  - The Python 2 language was officially discontinued in 2020

Why Python?

- Python has many advantages
  - Easy to learn
  - Less code
  - Syntax is easier to read
  - Open source
  - Huge amount of additional open-source libraries

    Some libraries listed below.

1. Performing fundamental scientific computing using **NumPy**
2. Performing data analysis using **pandas**
3. Plotting the data using **matplotlib**
4. Accessing scientific tools using **SciPy**
5. Implementing machine learning using **Scikit-learn**
6. Going for deep learning with **Keras** and **TensorFlow**
7. Creating graphs with **NetworkX**
8. Parsing HTML documents using **Beautiful Soup**
   - And many more..

1) NumPy

- NumPy is used to perform fundamental scientific computing.
- NumPy library provides the means for performing n-dimensional array manipulation, which is critical for data science work.
- NumPy provides functions that include support for linear algebra, Fourier transformation, random-number generation and many more..

  Explore listing of functions at https://numpy.org/doc/stable/reference/routines.html

NumPy

2) pandas

- pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- it offers data structures and operations for manipulating numerical tables and time series.
- The library is optimized to perform data science tasks especially fast and efficiently.
- The basic principle behind pandas is to provide data analysis and modelling support for Python that is similar to other languages such as R.

3) matplotlib

- The matplotlib library gives a MATLAB like interface for creating data presentations of the analysis.
- The library is initially limited to 2-D output, but it still provide means to express analysis graphically.
- Without this library we can not create output that people outside the data science community could easily understand.

4) SciPy

- The SciPy stack contains a host of other libraries that we can also download separately.
- These libraries provide support for mathematics, science and engineering.
- When we obtain SciPy, we get a set of libraries designed to work together to create applications of various sorts, these libraries are
  - NumPy
  - Pandas
  - matplotlib
  - Jupeter
  - Sympy
  - Etc…..

5) Scikit-learn

- The Scikit-learn library is one of many Scikit libraries that build on the capabilities provided by NumPy and SciPy to allow Python developers to perform domain specific tasks.
- Scikit-learn library focuses on data mining and data analysis, it provides access to following sort of functionality:
  - Classification
  - Regression
  - Clustering
  - Dimensionality reduction
  - Model selection
  - Pre-processing
- Scikit-learn is the most important library we are going to learn in this subject

scikits learn
machine learning in Python

6) Keras and TensorFlow

- Keras is an application programming interface (API) that is used to train deep learning models.
- An API often specifies a model for doing something, but it doesn’t provide an implementation.
- TensorFlow is an implementation for the keras, there are many other implementations for the keras like
  - Microsoft’s cognitive Toolkit, CNKT
  - Theano

7) NetworkX

- NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks (For example GPS setup to discover routes through city streets).
- NetworkX also provides the means to output the resulting analysis in a form that humans understand.
- Main advantage of using NetworkX is that nodes can be anything (including images) and edges can hold arbitrary data.

NetworkX
Network Analysis in Python

8) Beautiful Soup

- Beautiful Soup is a Python package for parsing HTML and XML documents.
- It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

Installing Python

- For Windows & Mac:
  - To install python in windows you need to download installable file from https://www.python.org/downloads/
  - After downloading the installable file you need to execute the file.
- For Linux :
  - For ubuntu 16.10 or newer
    - sudo apt-get update
    - sudo apt-get install python3.8
- To verify the installation
  - Windows :
    - python --version
  - Linux :
    - **python3** --version (linux might have python2 already installed, you can check python 2 using **python --version**)
- Alternatively we can use anaconda distribution for the python installation
  - http://anaconda.com/downloads
  - Anaconda comes with many useful inbuilt libraries.

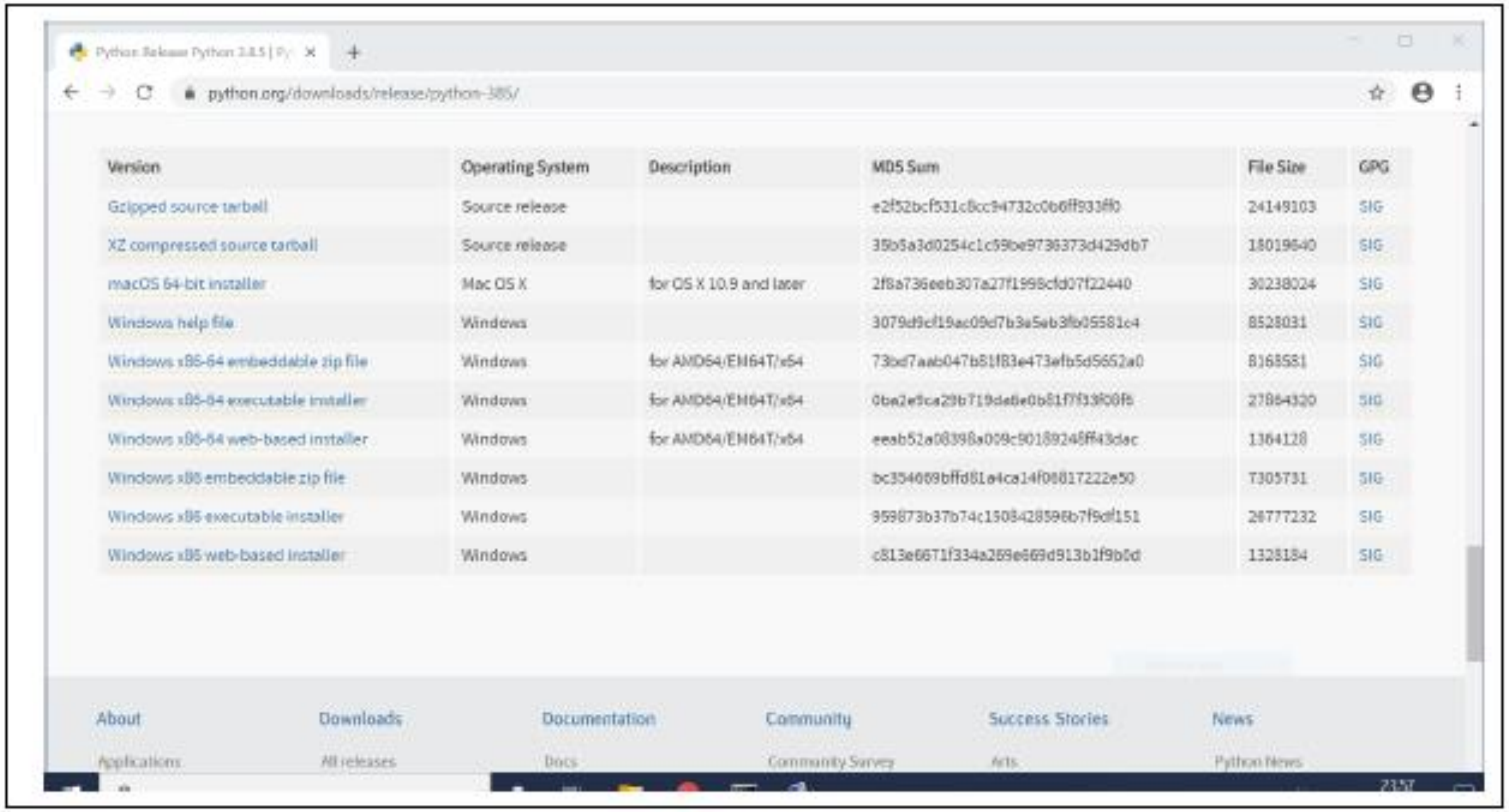
Intro to Python

- There are two versions of Python:
  - *Python 2.x* . and
  - *Python 3.x* .

Windows

Installation : 3.8.5

https://www.python.org/downloads/release/python-385/

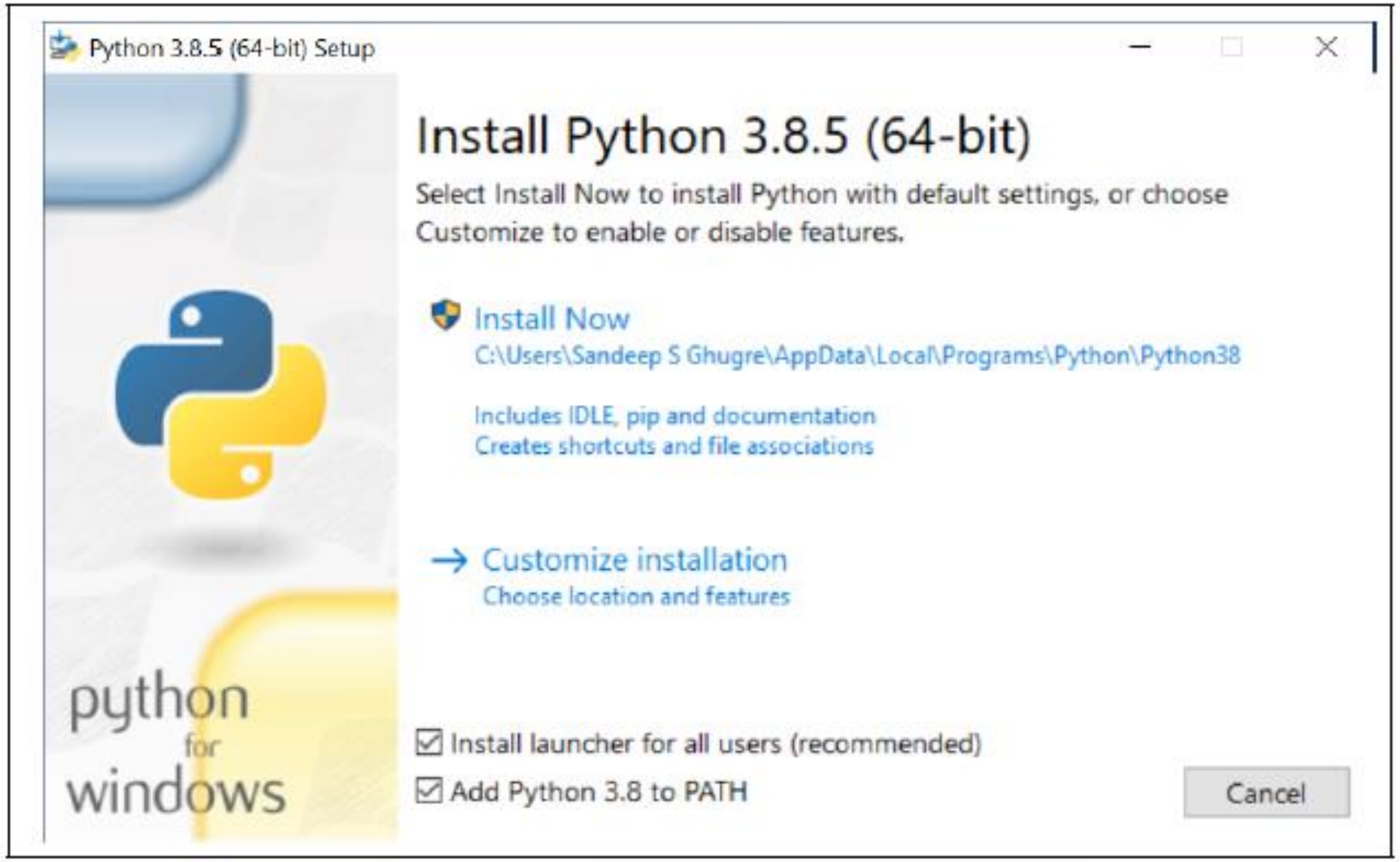


Python Release Python 3.8.5 | Py

python.org/downloads/release/python-385/

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e2f52bcf531c8cc94732c0b6ff933ff0	24149103	SIG
XZ compressed source tarball	Source release		35b5a3d0254c1c59be9736373d429db7	18019640	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	2f8a736eeb307a27f1998cfd07f22440	30238024	SIG
Windows help file	Windows		3079d9cf19ac09d7b3a5eb3fb05581c4	8528031	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	73bd7aab047b81f83e473efb5d5652a0	8168581	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	0ba2e9ca29b719da6e0b81f7f33f08f6	27864320	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	eeab52a08398a009c90189248ff43dac	1364128	SIG
Windows x86 embeddable zip file	Windows		bc354669bffd81a4ca14f06817222e50	7305731	SIG
Windows x86 executable installer	Windows		959873b37b74c1508428596b7f9df151	26777232	SIG
Windows x86 web-based installer	Windows		c813e6671f334a269e669d913b1f9b0d	1328184	SIG

Installation : 3.8.5



Python 3.8.5 (64-bit) Setup

Install Python 3.8.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\Sandeep S Ghugre\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

Figure: Click *Add..... & Install Now*.

The Python interpreter can be run (activated) by clicking on an icon (which starts the IDE) , or by typing *python* on a command line.

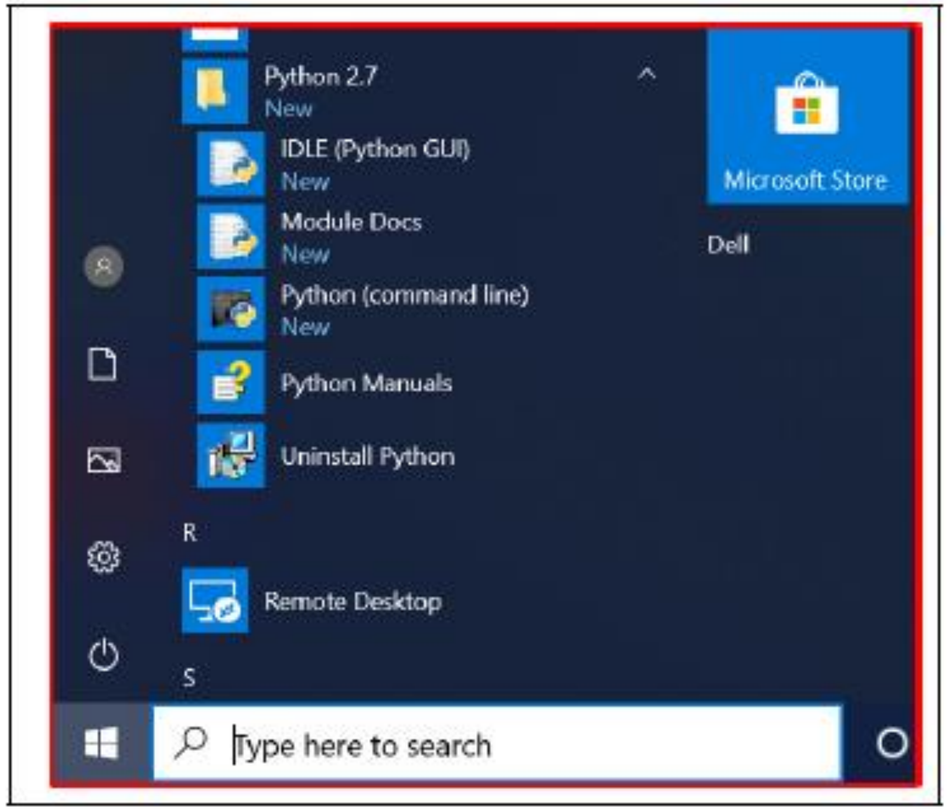
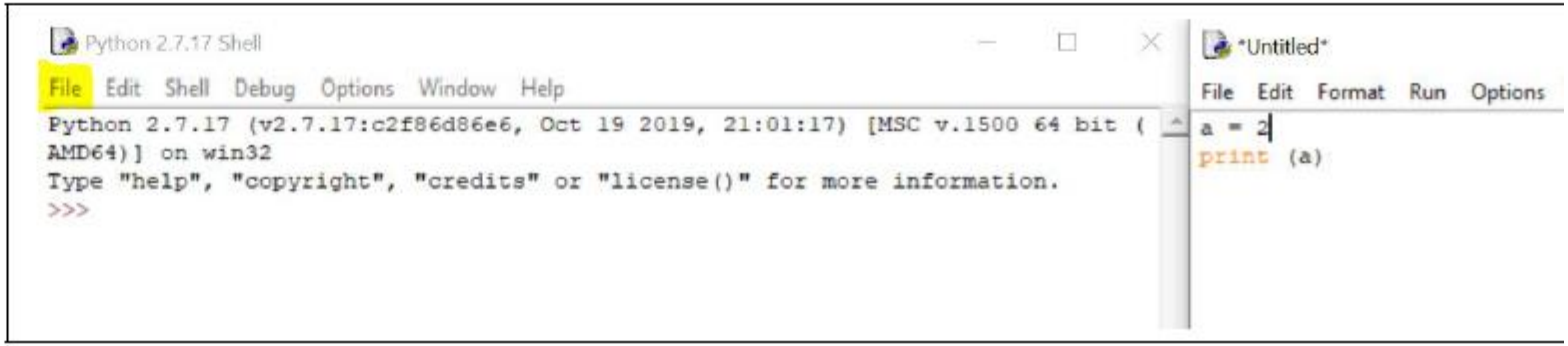


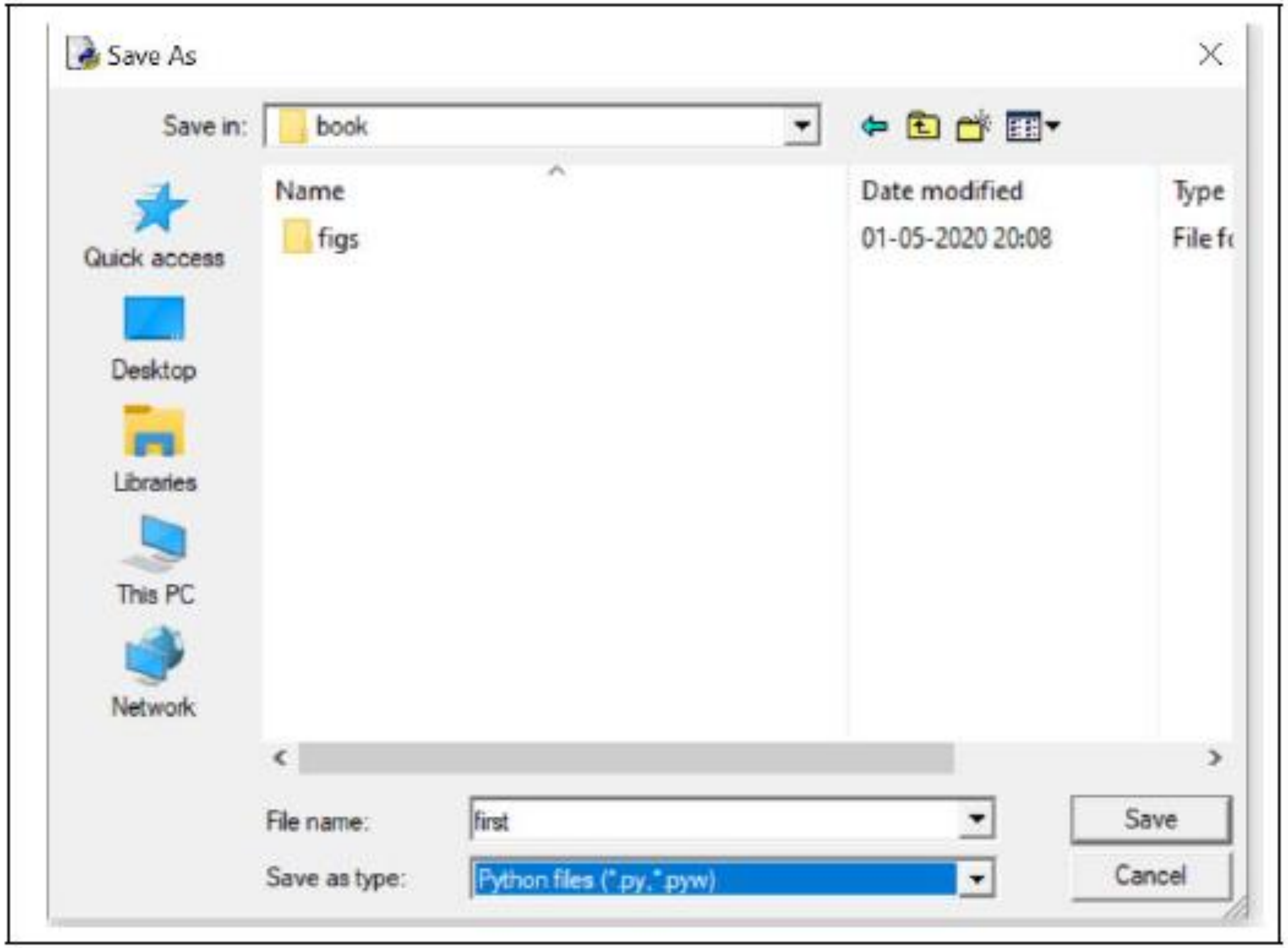
Figure: Getting started.


Python 2.7.17 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options

```
Python 2.7.17 (v2.7.17:c2f86d86e6, Oct 19 2019, 21:01:17) [MSC v.1500 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

```
a = 2
print (a)
```

Figure: Invoking the Python IDE.

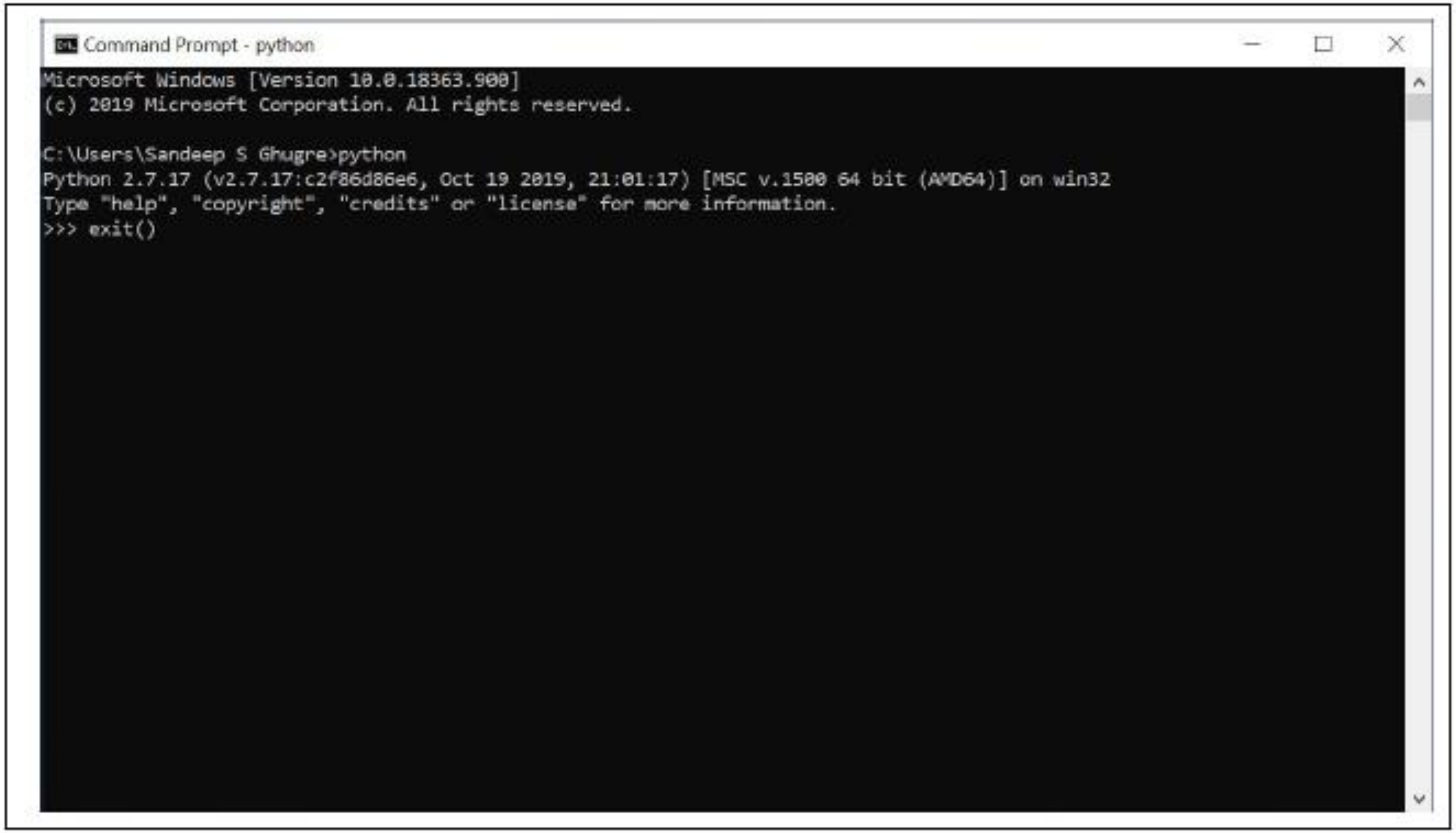


figs 01-05-2020 20:08 File fc

File name: first

Save as type: Python files (*.py,*.pyw)

Figure: Save your work with **.py** extension.


Command Prompt - python

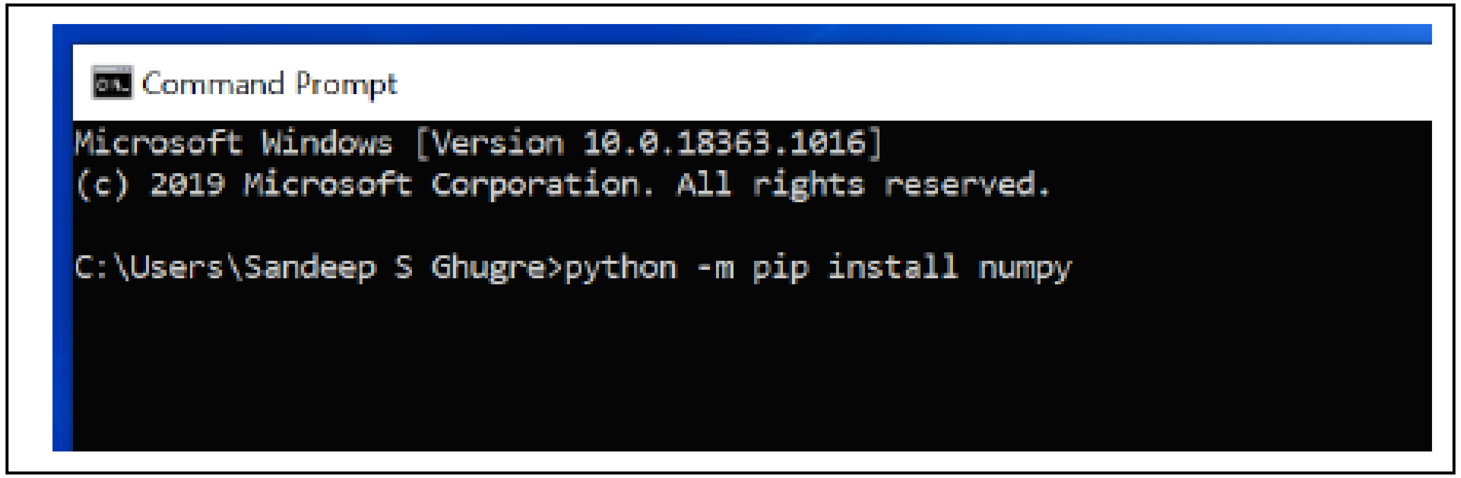
```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep S Ghugre>python
Python 2.7.17 (v2.7.17:c2f86d86e6, Oct 19 2019, 21:01:17) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

Figure: Using the command prompt.

We need to install *packages*, which essentially refer to a set of softwares, which have been specifically developed for a certain use.

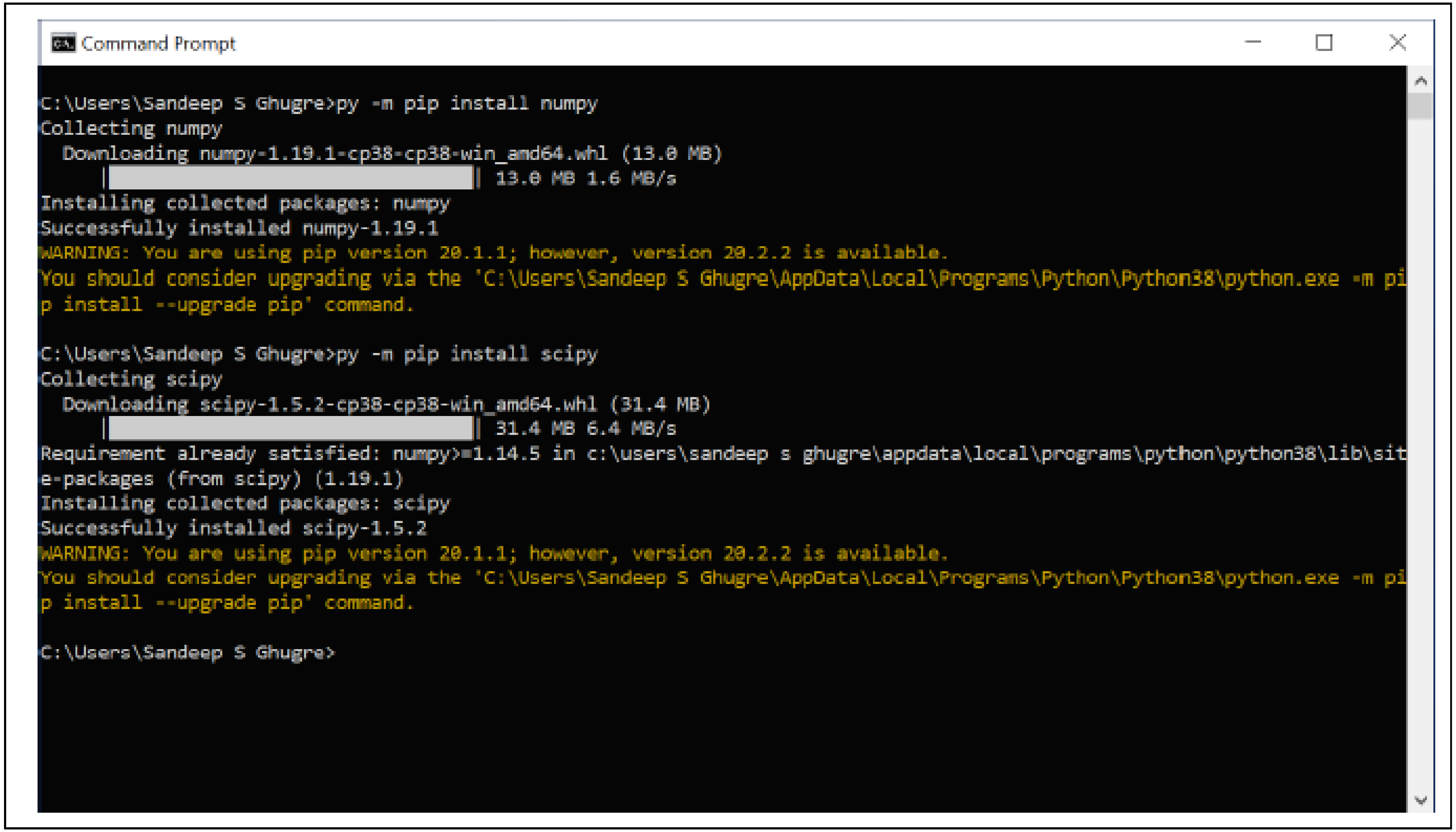
For example, we shall require *NumPy*, fundamental package for scientific computing with Python. It can be installed using the command *python -m pip install numpy*, from a terminal.



```
Microsoft Windows [Version 10.0.18363.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sandeep S Ghugre>python -m pip install numpy
```

Figure: The command in Python 3.8.5 is **python** *-m pip install numpy*.



```
C:\Users\Sandeep S Ghugre>py -m pip install numpy
Collecting numpy
  Downloading numpy-1.19.1-cp38-cp38-win_amd64.whl (13.0 MB)
     |████████████████████████████████| 13.0 MB 1.6 MB/s
Installing collected packages: numpy
Successfully installed numpy-1.19.1
WARNING: You are using pip version 20.1.1; however, version 20.2.2 is available.
You should consider upgrading via the 'C:\Users\Sandeep S Ghugre\AppData\Local\Programs\Python\Python38\python.exe -m pi
p install --upgrade pip' command.

C:\Users\Sandeep S Ghugre>py -m pip install scipy
Collecting scipy
  Downloading scipy-1.5.2-cp38-cp38-win_amd64.whl (31.4 MB)
     |████████████████████████████████| 31.4 MB 6.4 MB/s
Requirement already satisfied: numpy>=1.14.5 in c:\users\sandeep s ghugre\appdata\local\programs\python\python38\lib\sit
e-packages (from scipy) (1.19.1)
Installing collected packages: scipy
Successfully installed scipy-1.5.2
WARNING: You are using pip version 20.1.1; however, version 20.2.2 is available.
You should consider upgrading via the 'C:\Users\Sandeep S Ghugre\AppData\Local\Programs\Python\Python38\python.exe -m pi
p install --upgrade pip' command.

C:\Users\Sandeep S Ghugre>
```

Figure: In Python 3.8.5 use **py** *-m pip install *****.

Install scikit-learn

NOTE: More information on installing scikit-learn at the link provided above (http://scikit-learn.org/stable/install.html)

On Windows: use pip to install scikit-learn:
pip install scikit-learn

On Linux: Use the package manager or follow the build instructions at http://www.bogotobogo.com/python/scikit-learn/scikit-learn_install.php

Test Installation

Now we must see if everything installed correctly. Open up a command line terminal and type:

```
Python
```

This will open a python interpreter. You will know this because there will be some text and three chevrons, “>>>”, prompting input. Type:

```
import sklearn
```

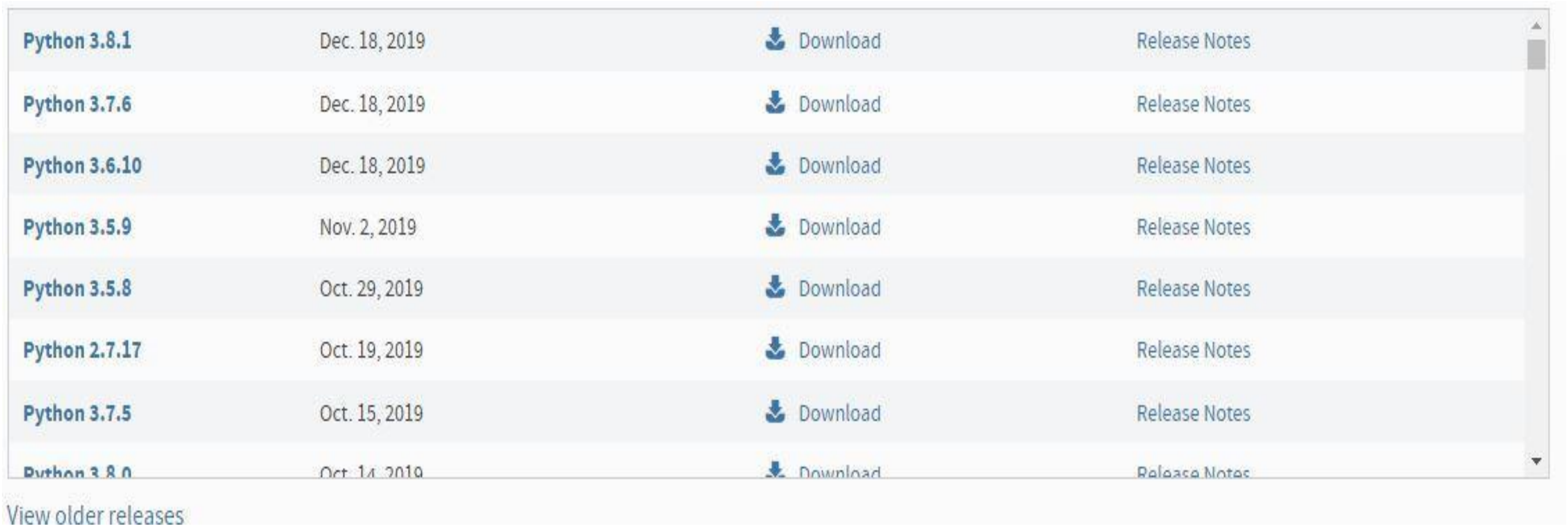
If nothing happens and another prompt appears scikit-learn has been installed correctly.

LINUX

Download and Install Python:

Before starting with the installation process, you need to download it. For that all versions of Python for Linux are available on python.org.

Download the required version and follow the further instructions for the installation process.



Python 3.8.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 29, 2019	Download	Release Notes
Python 2.7.17	Oct. 19, 2019	Download	Release Notes
Python 3.7.5	Oct. 15, 2019	Download	Release Notes
Python 3.8.0	Oct. 14, 2019	Download	Release Notes

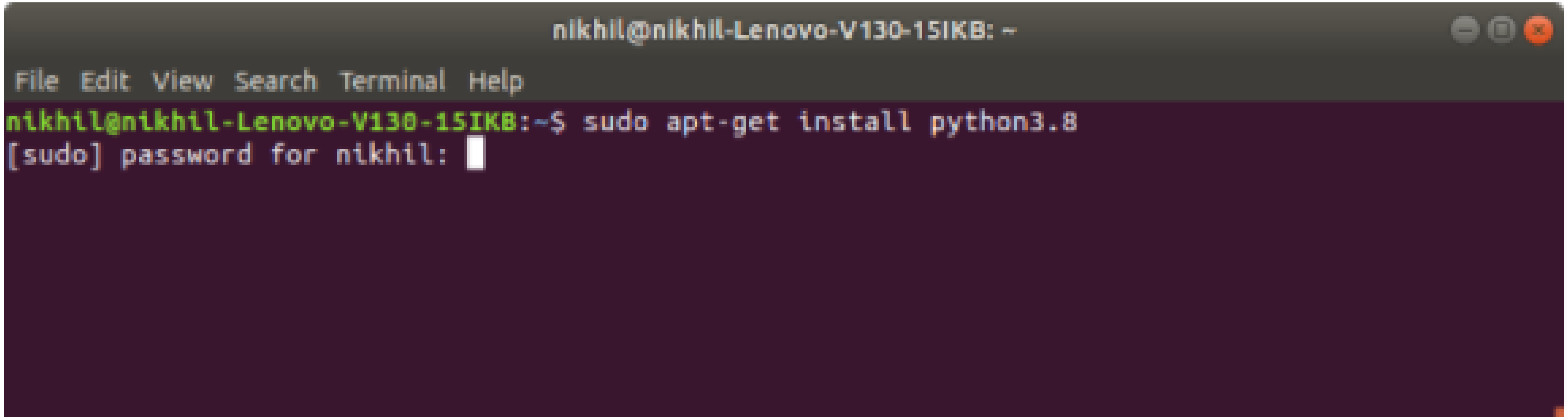
View older releases

Beginning the installation.

For almost every Linux system, the following command could be used to install Python directly:

```
$ sudo apt-get install python3.8
```

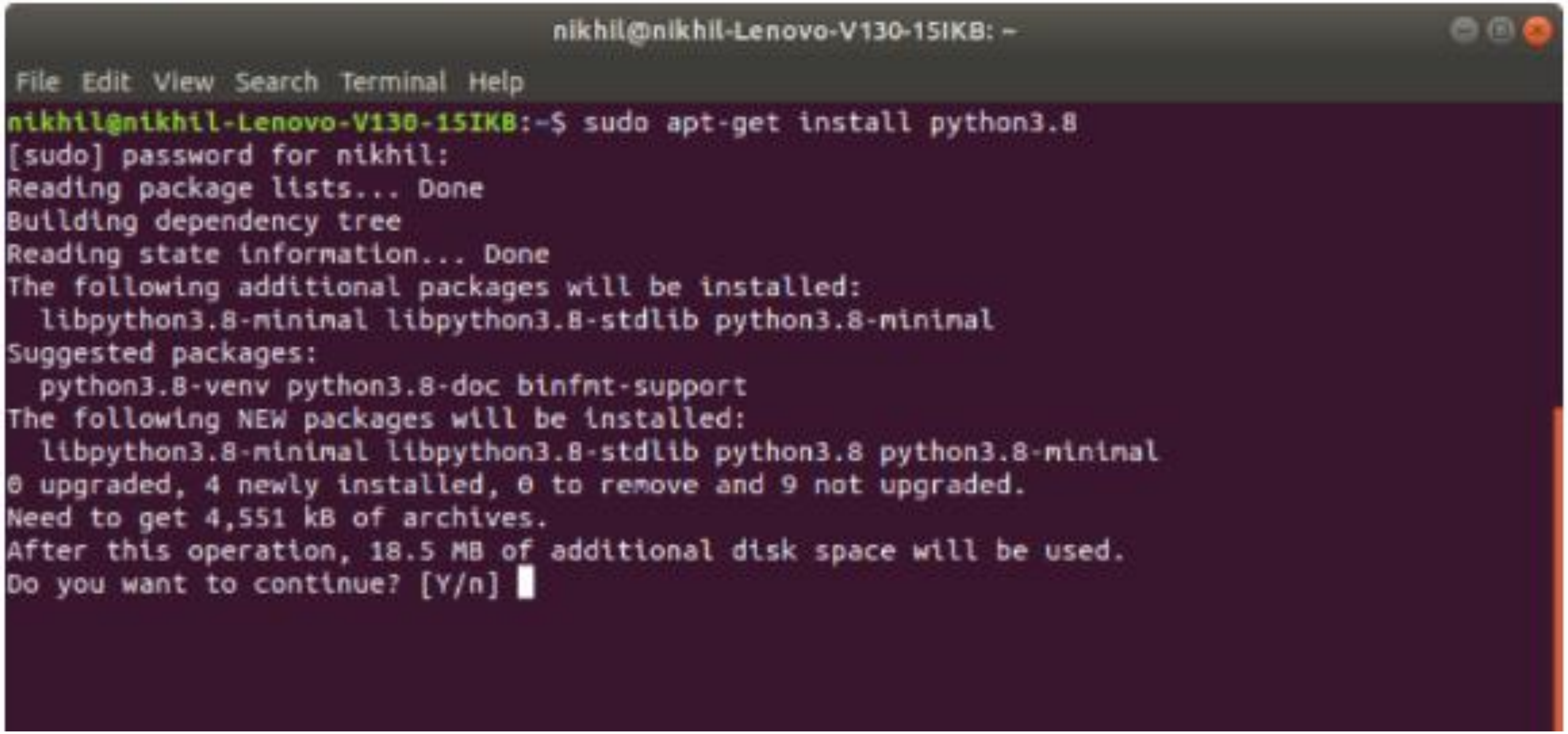
Getting Started:


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

```
nikhil@nikhil-Lenovo-V130-15IKB:~$ sudo apt-get install python3.8
[sudo] password for nikhil: 
```

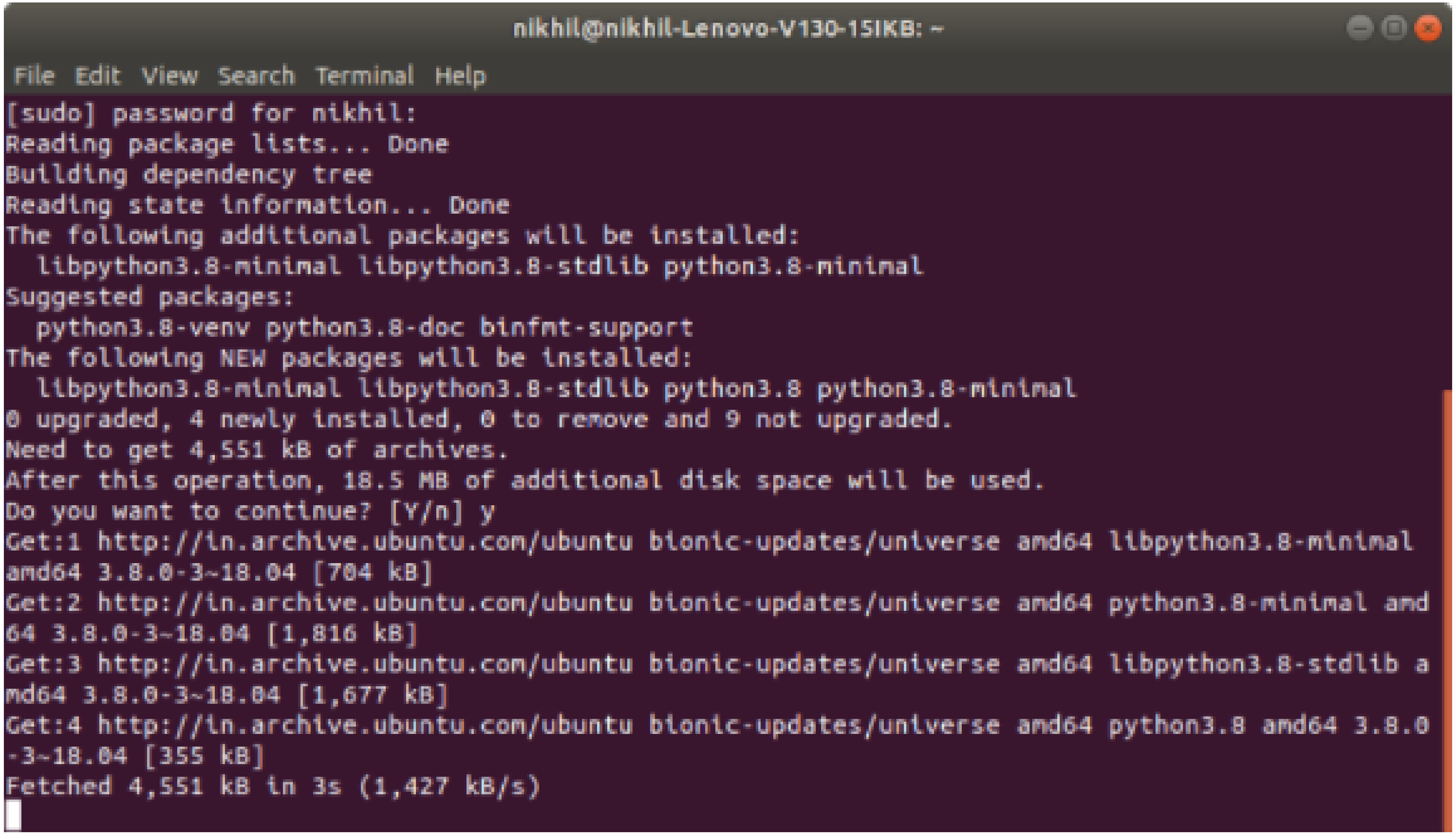
SOURCE: https://www.geeksforgeeks.org/how-to-install-python-on-linux/

Assigning DiskSpace:


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

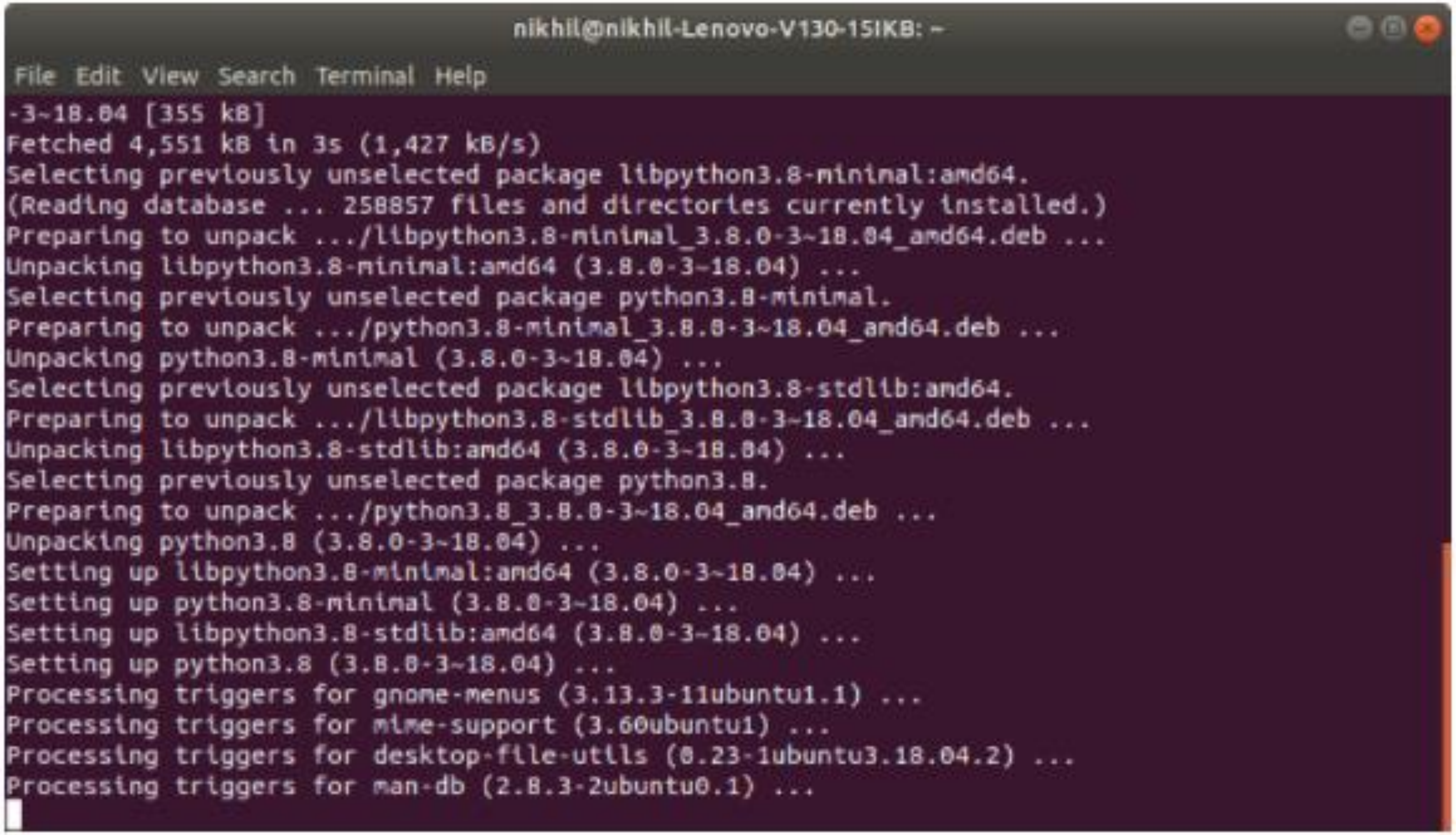
```
nikhil@nikhil-Lenovo-V130-15IKB:~$ sudo apt-get install python3.8
[sudo] password for nikhil:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython3.8-minimal libpython3.8-stdlib python3.8-minimal
Suggested packages:
  python3.8-venv python3.8-doc binfmt-support
The following NEW packages will be installed:
  libpython3.8-minimal libpython3.8-stdlib python3.8 python3.8-minimal
0 upgraded, 4 newly installed, 0 to remove and 9 not upgraded.
Need to get 4,551 kB of archives.
After this operation, 18.5 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Fetching and Installing Packages:


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

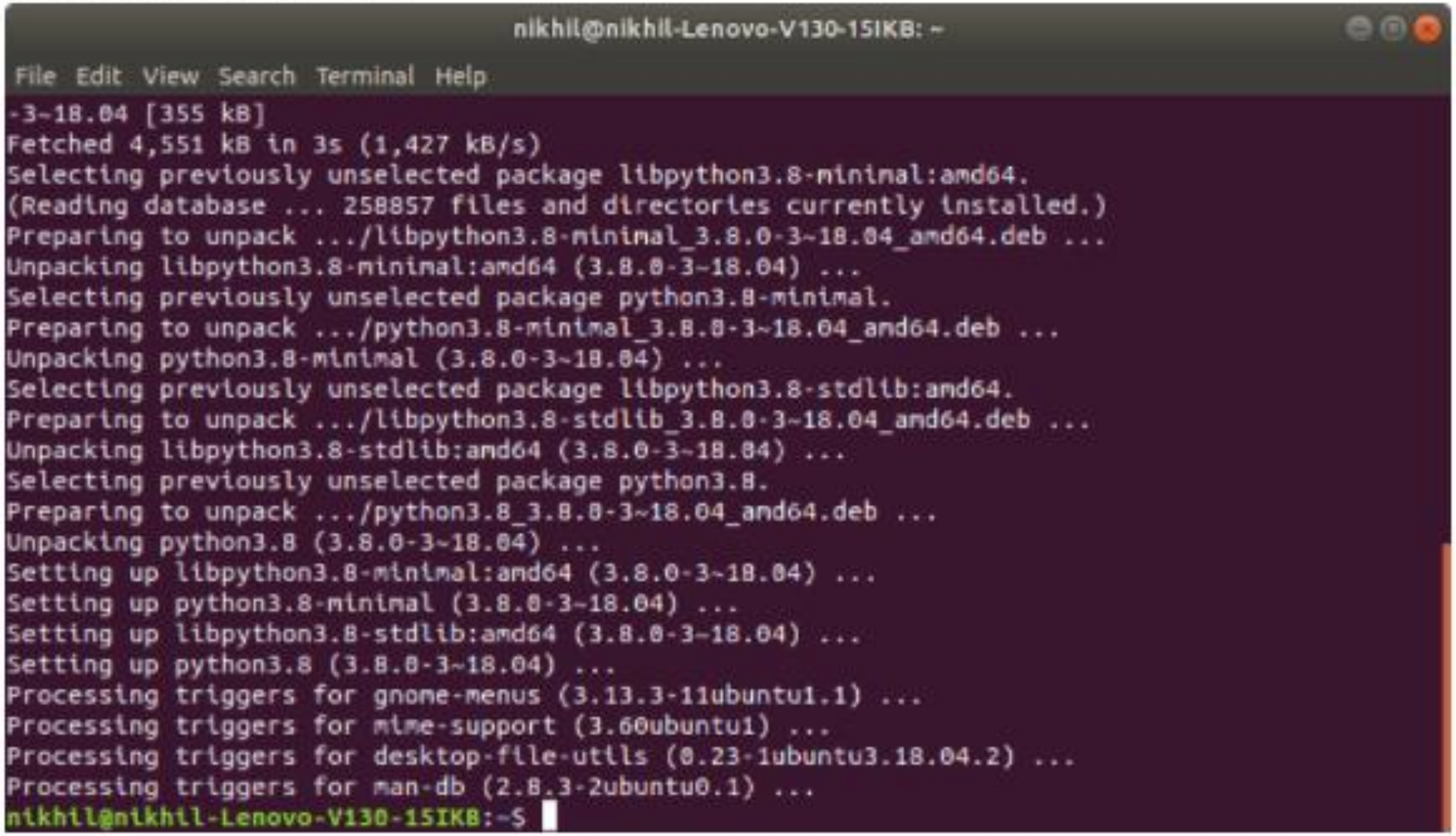
```
[sudo] password for nikhil:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython3.8-minimal libpython3.8-stdlib python3.8-minimal
Suggested packages:
  python3.8-venv python3.8-doc binfmt-support
The following NEW packages will be installed:
  libpython3.8-minimal libpython3.8-stdlib python3.8 python3.8-minimal
0 upgraded, 4 newly installed, 0 to remove and 9 not upgraded.
Need to get 4,551 kB of archives.
After this operation, 18.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 libpython3.8-minimal
amd64 3.8.0-3~18.04 [704 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 python3.8-minimal amd
64 3.8.0-3~18.04 [1,816 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 libpython3.8-stdlib a
md64 3.8.0-3~18.04 [1,677 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 python3.8 amd64 3.8.0
-3~18.04 [355 kB]
Fetched 4,551 kB in 3s (1,427 kB/s)
```

Getting through the installation process:


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

```
-3~18.04 [355 kB]
Fetched 4,551 kB in 3s (1,427 kB/s)
Selecting previously unselected package libpython3.8-minimal:amd64.
(Reading database ... 258857 files and directories currently installed.)
Preparing to unpack .../libpython3.8-minimal_3.8.0-3~18.04_amd64.deb ...
Unpacking libpython3.8-minimal:amd64 (3.8.0-3~18.04) ...
Selecting previously unselected package python3.8-minimal.
Preparing to unpack .../python3.8-minimal_3.8.0-3~18.04_amd64.deb ...
Unpacking python3.8-minimal (3.8.0-3~18.04) ...
Selecting previously unselected package libpython3.8-stdlib:amd64.
Preparing to unpack .../libpython3.8-stdlib_3.8.0-3~18.04_amd64.deb ...
Unpacking libpython3.8-stdlib:amd64 (3.8.0-3~18.04) ...
Selecting previously unselected package python3.8.
Preparing to unpack .../python3.8_3.8.0-3~18.04_amd64.deb ...
Unpacking python3.8 (3.8.0-3~18.04) ...
Setting up libpython3.8-minimal:amd64 (3.8.0-3~18.04) ...
Setting up python3.8-minimal (3.8.0-3~18.04) ...
Setting up libpython3.8-stdlib:amd64 (3.8.0-3~18.04) ...
Setting up python3.8 (3.8.0-3~18.04) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

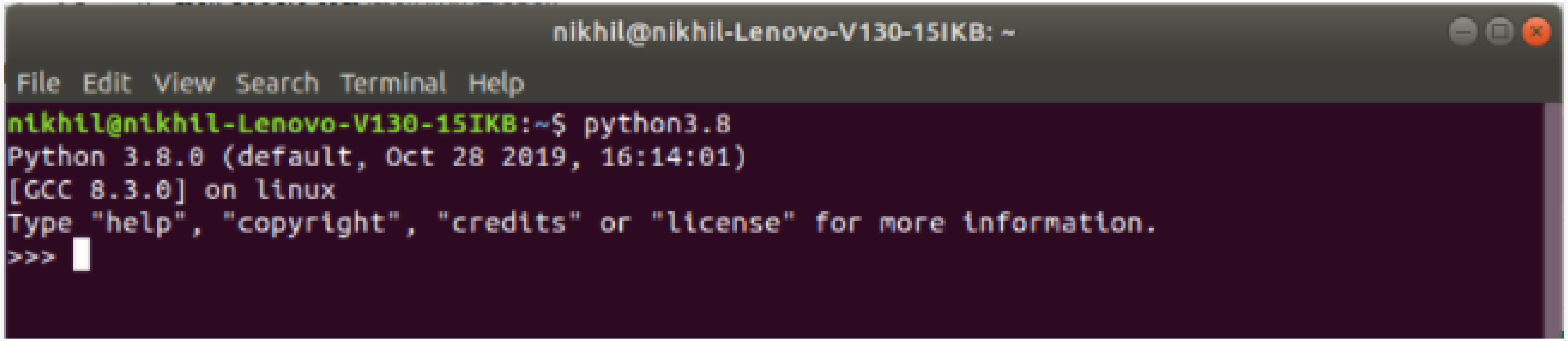
Finished Installation:


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

```
-3~18.04 [355 kB]
Fetched 4,551 kB in 3s (1,427 kB/s)
Selecting previously unselected package libpython3.8-minimal:amd64.
(Reading database ... 258857 files and directories currently installed.)
Preparing to unpack .../libpython3.8-minimal_3.8.0-3~18.04_amd64.deb ...
Unpacking libpython3.8-minimal:amd64 (3.8.0-3~18.04) ...
Selecting previously unselected package python3.8-minimal.
Preparing to unpack .../python3.8-minimal_3.8.0-3~18.04_amd64.deb ...
Unpacking python3.8-minimal (3.8.0-3~18.04) ...
Selecting previously unselected package libpython3.8-stdlib:amd64.
Preparing to unpack .../libpython3.8-stdlib_3.8.0-3~18.04_amd64.deb ...
Unpacking libpython3.8-stdlib:amd64 (3.8.0-3~18.04) ...
Selecting previously unselected package python3.8.
Preparing to unpack .../python3.8_3.8.0-3~18.04_amd64.deb ...
Unpacking python3.8 (3.8.0-3~18.04) ...
Setting up libpython3.8-minimal:amd64 (3.8.0-3~18.04) ...
Setting up python3.8-minimal (3.8.0-3~18.04) ...
Setting up libpython3.8-stdlib:amd64 (3.8.0-3~18.04) ...
Setting up python3.8 (3.8.0-3~18.04) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
nikhil@nikhil-Lenovo-V130-15IKB:~$
```

To verify the installation enter the following commands in your Terminal.

```
python3.8
```


nikhil@nikhil-Lenovo-V130-15IKB: ~
File Edit View Search Terminal Help

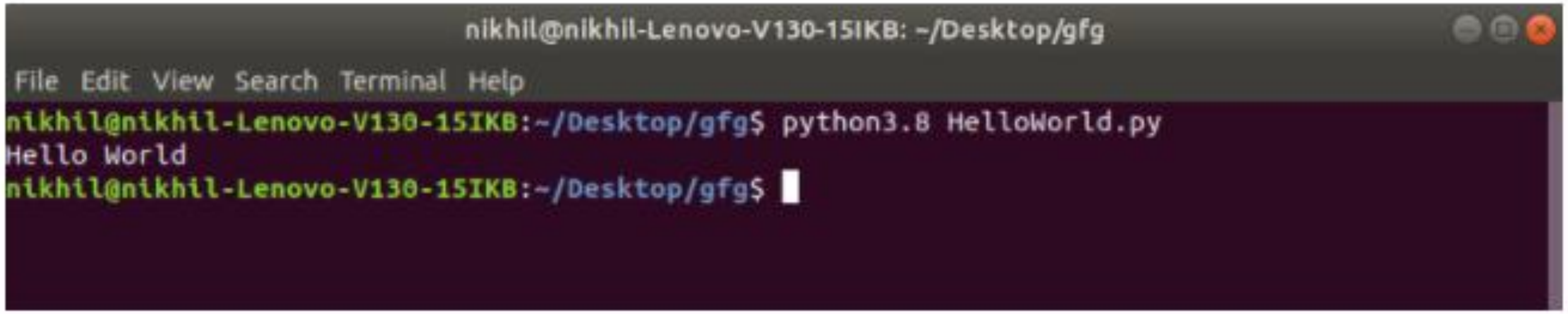
```
nikhil@nikhil-Lenovo-V130-15IKB:~$ python3.8
Python 3.8.0 (default, Oct 28 2019, 16:14:01)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Let’s consider a simple Hello World Program.

```
# Python program to print
# Hello World

print("Hello World")
```

Output:



nikhil@nikhil-Lenovo-V130-15IKB: ~/Desktop/gfg
File Edit View Search Terminal Help

```
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$ python3.8 HelloWorld.py
Hello World
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$ 
```

Other Guides to Installation in Linux

https://opensource.com/article/20/4/install-python-linux

https://docs.python-guide.org/starting/install3/linux/

https://linuxize.com/post/how-to-install-python-3-8-on-ubuntu-18-04/

Data Types in Python

Name	Type	Description
Integer	int	Whole number such as 0,1,5, -5 etc..
Float	float	Numbers with decimal points such as 1.5, 7.9, -8.2 etc..
String	str	Sequence of character (Ordered) such as “pawan”, ‘college’, etc..
Boolean	bool	Logical values indicating Ture or False (T and F here are capital in python)
		Data Structures
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, “pawan”, True, 102.3]
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “pawan”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “pawan”, True, 102.3 }
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { “college”: “pawan”, “code”: “054” }

Variables in Python

- A Python variable is a reserved memory location to store values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Python uses Dynamic Typing so,
  - We need not to specify the data types to the variable as it will internally assign the data type to the variable according to the value assigned.
  - we can also reassign the different data type to the same variable, variable data type will change to new data type automatically.
  - We can check the current data type of the variable with **type(variablename)** in-built function.
- Rules for variable name
  - Name can not start with digit
  - Space not allowed
  - Can not contain special character
  - Python keywords not allowed
  - **Should** be in lower case

Example of Python Variables

- Example :

demo.py

```
x = 10
print(x)
print(type(x))

y = 123.456
print(y)

x = “Information technology"
print(x)
print(type(x))
```

Reassign same variable to hold different data type

Run in terminal

```
python demo.py
```

Output

```
10
int
123.456
Information technology
str
```

Simple program

```
#Add two user i/p nos:
a = int(input("enter first number: "))
b = int(input("enter second number: "))

sum = a + b

print("sum:", sum)
```

OR

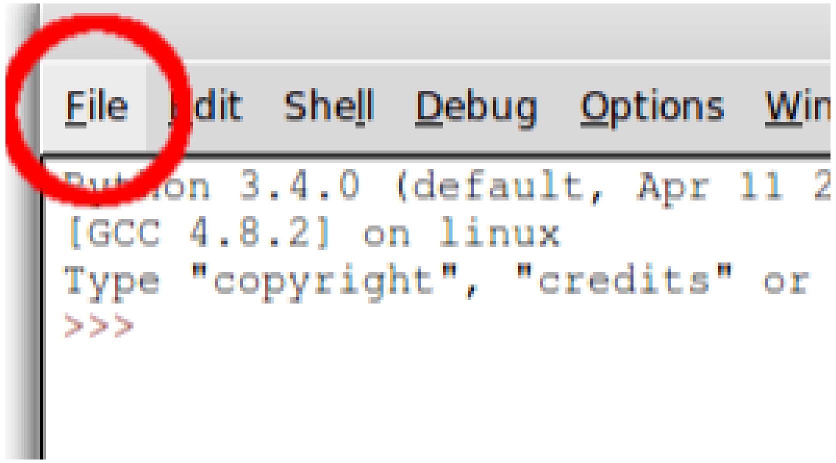
```
# Python3 program to add two numbers

num1 = 15
num2 = 12

# Adding two nos
sum = num1 + num2

# printing values
print("Sum of {0} and {1} is {2}" .format(num1, num2, sum))
```

Saving the code

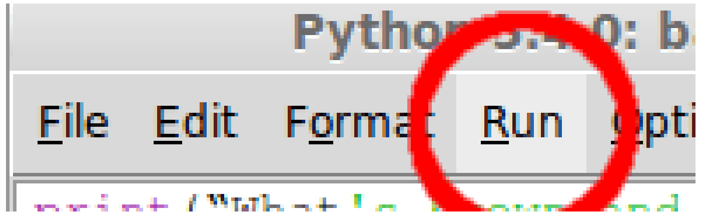


```
Python 3.4.0 (default, Apr 11 2
[GCC 4.8.2] on linux
Type "copyright", "credits" or
>>>
```

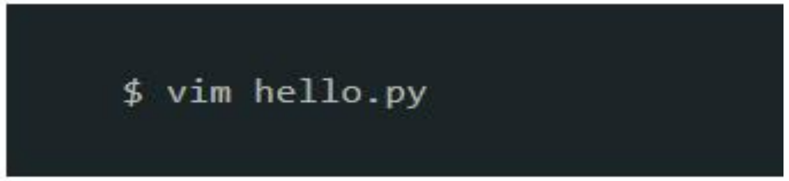
SAVE AS: <filename>.py

Eg: AdditionNum.py

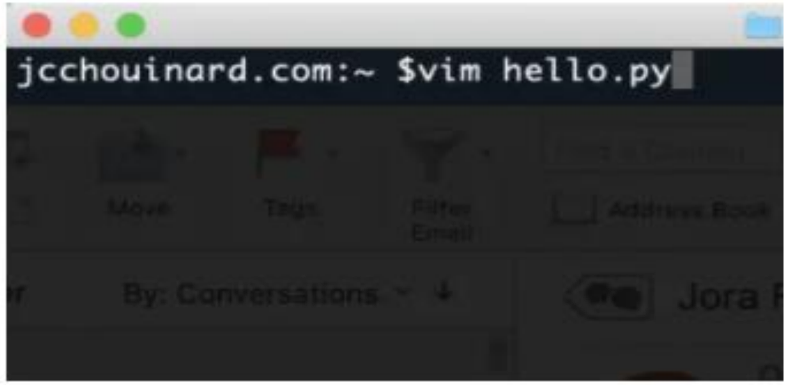
Running the code



Create the Python File in LINUX

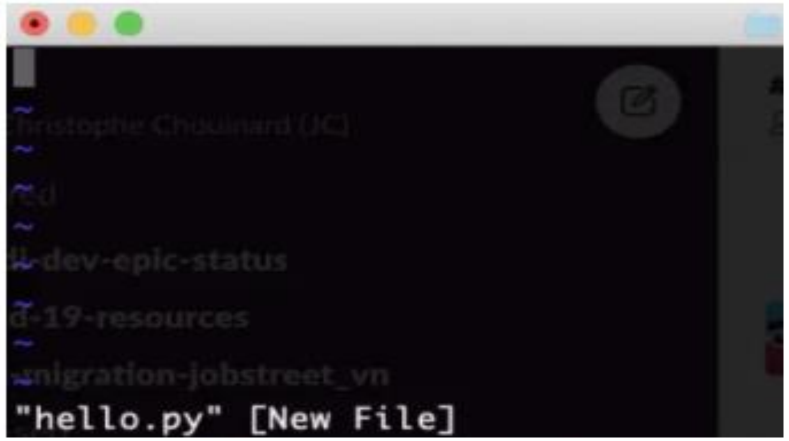


```
$ vim hello.py
```



```
jcchouinard.com:~ $vim hello.py
```

This will open the `vim` editor.



```
~
~
~
~
~
~
"hello.py" [New File]
```

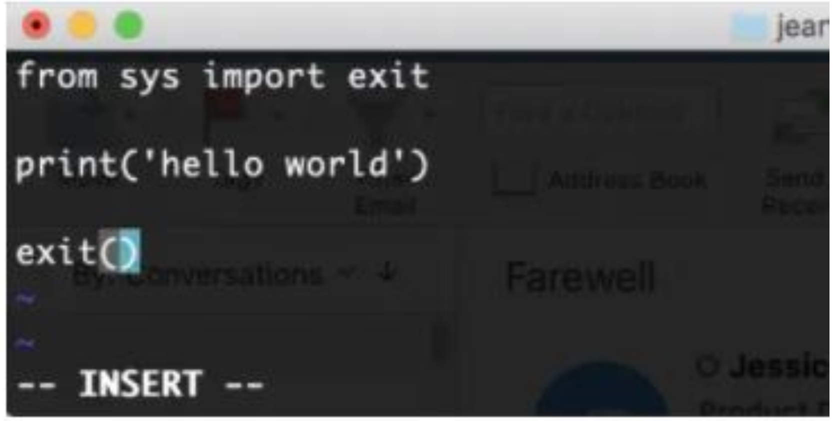
Vi Editor Commands

First, let's look at the commands you can use in the vi editor.

i	Switch to Insert mode (editing mode)
esc	Exit the editing mode
dd	Delete the current line
u	Undo the last change
:q!	Close the editor without saving changes.
:wq	Save the text and close the editor
→ + `Shift`	Move Cursor Faster
$	Move to end of line

```
-- INSERT --
```

Write the best python script in the world.



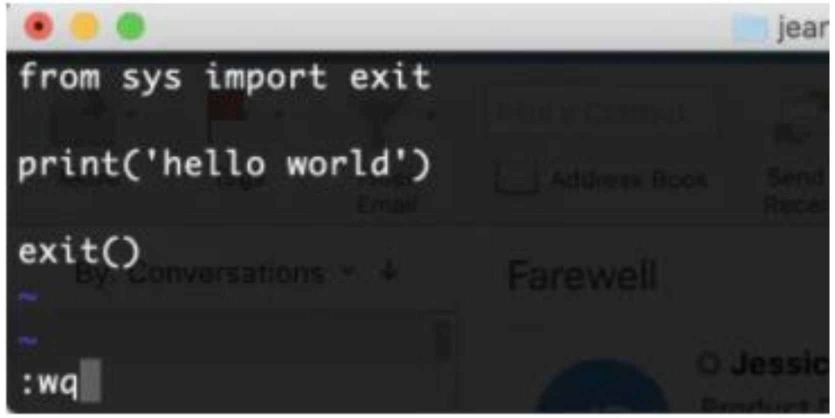
```
from sys import exit

print('hello world')

exit()
~
~
-- INSERT --
```

Press `esc` to leave the editing mode.

Write the command `:wq`to save and quite t



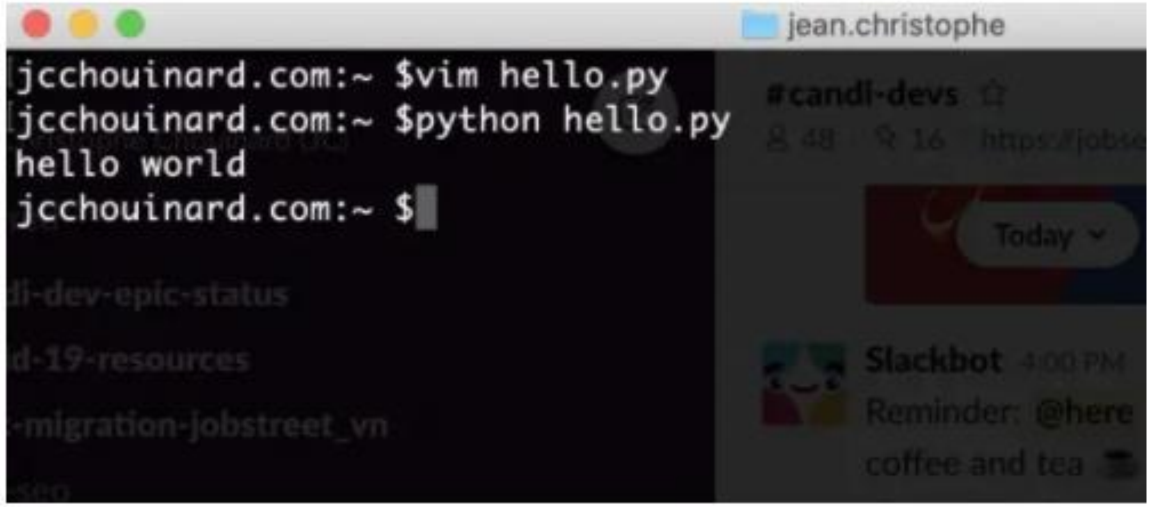
```
from sys import exit

print('hello world')

exit()
~
~
:wq
```

Run Your Script

Run your script by typing python `hello.py` in the Terminal.



```
jcchouinard.com:~ $vim hello.py
jcchouinard.com:~ $python hello.py
hello world
jcchouinard.com:~ $
```

Data Structures in Python

- There are four built-in data structures in Python - *list, dictionary, tuple and set*.

Name	Type	Description
List	list	**Ordered** Sequence of objects, will be represented with **square** brackets **[ ]** Example : [18, “darshan”, True, 102.3]
Dictionary	dict	**Unordered key : value** pair of objects , will be represented with **curly** brackets **{ }** Example : { “college”: “darshan”, “code”: “054” }
Tuple	tup	**Ordered immutable** sequence of objects, will be represented with **round** brackets **( )** Example : (18, “darshan”, True, 102.3)
Set	set	**Unordered** collection of **unique** objects, will be represented with the **curly** brackets **{ }** Example : { 18, “darshan”, True, 102.3 }

- Lets explore all the data structures in detail…

Operators in python

- We can segregate python operators in the following groups
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators

- We will discuss some of the operators from the given list in detail in some of next slides.

Arithmetic Operators

- Note : consider A = 10 and B = 3

Operator	Description	**Example**	Output
+	Addition	A + B	13
-	Subtraction	A - B	7
/	Division	A / B	3.3333333333333335
*	Multiplication	A * B	30
%	Modulus return the remainder	A % B	1
//	Floor division returns the quotient	A // B	3
**	Exponentiation	A ** B	10 * 10 * 10 = 1000

Logical Operators

- Note : consider A = 10 and B = 3

Operator	Description	**Example**	Output
and	Returns True if both statements are true	A > 5 **and** B < 5	True
or	Returns True if one of the statements is true	A > 5 **or** B > 5	True
not	Negate the result, returns True if the result is False	**not** (A > 5)	False

Identity & Member Operators

- Identity Operator
- Note : consider A = [1,2], B = [1,2] and C=A

Operator	Description	**Example**	Output
is	Returns True if both variables are the same object	A is B A is C	FALSE TRUE
is not	Returns True if both variables are **different** object	A is not B	TRUE

- Member Operator
- Note : consider A = 2 and B = [1,2,3]

Operator	Description	**Example**	Output
in	Returns True if a sequence with the specified value is present in the object	A in B	TRUE
not in	Returns True if a sequence with the specified value is not present in the object	A not in B	FALSE

If statement

- if statement is written using the **if** keyword followed by **condition** and **colon**(:) .
- Code to execute when the condition is true will be ideally written in the next line with **Indentation** (white space).
- Python relies on indentation to define scope in the code (Other programming languages often use curly-brackets for this purpose).

Syntax

```
if some_condition :
  # Code to execute when condition is true
```

if statement ends with :

Indentation (tab/whitespace) at the beginning

ifdemo.py

```python
x = 10

if x > 5 :
    print("X is greater than 5")
```

Run in terminal

```
python ifdemo.py
```

Output

```
X is greater than 5
```

If else statement

Syntax

```
if some_condition :
   # Code to execute when condition is true
else :
   # Code to execute when condition is false
```

ifelsedemo.py

```python
x = 3

if x > 5 :
    print("X is greater than 5")
else :
    print("X is less than 5")
```

Run in terminal

```
python ifelsedemo.py
```

Output

```
X is less than 5
```

If, elif and else statement

Syntax

```
if some_condition_1 :
   # Code to execute when condition 1 is true
elif some_condition_2 :
   # Code to execute when condition 2 is true
else :
   # Code to execute when both conditions are false
```

ifelifdemo.py

```
x = 10

if x > 12 :
    print("X is greater than 12")
elif x > 5 :
    print("X is greater than 5")
else :
    print("X is less than 5")
```

Run in terminal

```
python ifelifdemo.py
```

Output

```
X is greater than 5
```

For loop in python

- Many objects in python are **iterable,** meaning we can iterate over every element in the object.
  - such as every elements from the List, every characters from the string etc..
- We can use for loop to execute block of code for each element of iterable object.

Syntax

```
for temp_item in iterable_object :
    # Code to execute for each object in iterable
```

For loop ends with :

Indentation (tab/whitespace) at the beginning

fordemo1.py

```
my_list = [1, 2, 3, 4]
for list_item in my_list :
    print(list_item)
```

Output :

```
1
2
3
4
```

fordemo2.py

```
my_list = [1,2,3,4,5,6,7,8,9]
for list_item in my_list :
    if list_item % 2 == 0 :
        print(list_item)
```

Output :

```
2
4
6
8
```

For loop (tuple unpacking)

- Sometimes we have nested data structure like List of tuples, and if we want to iterate with such list we can use tuple unpacking.

withouttupleunapacking.py

```python
my_list = [(1,2,3), (4,5,6), (7,8,9)]
for list_item in my_list :
    print(list_item[1])
```

Output :

```
2
5
8
```

withtupleunpacking.py

```python
my_list = [(1,2,3), (4,5,6), (7,8,9)]
for a,b,c in my_list :
    print(b)
```

This technique is known as tuple unpacking

Output :

```
2
5
8
```

- range() function will create a list from 0 till (not including) the value specified as a

rangedemo.py

```python
my_list = range(5)
for list_item in my_list :
    print(list_item)
```

Output :

```
0
1
2
3
4
```

While loop

- While loop will continue to execute block of code until some condition remains True.
- For example,
  - while felling hungry, keep eating
  - while have internet pack available, keep watching videos

Syntax

```
while some_condition :
    # Code to execute in loop
```

while loop ends with **:**

Indentation (tab/whitespace) at the beginning

whiledemo.py

```
x = 0
while x < 3 :
    print(x)
    x += 1    # x++ is valid in python
```

Output :

```
0
1
2
```

withelse.py

```
x = 5
while x < 3 :
    print(x)
    x += 1    # x++ is valid in python
else :
    print("X is greater than 3")
```

Output :
X is greater than 3

break, continue & pass keywords

- break : Breaks out of the current closest enclosing loop.

breakdemo.py

```
for temp in range(5) :
    if temp == 2 :
        break

    print(temp)
```

Output :
```
0
1
```

- continue : Goes to the top of the current closest enclosing loop.

continuedemo.py

```
for temp in range(5) :
    if temp == 2 :
        continue

    print(temp)
```

Output :
```
0
1
3
4
```

- Pass : Does nothing at all, will be used as a placeholder in conditions where you don't want to write anything.

passdemo.py

```
for temp in range(5) :
    pass
```

Output : (nothing)

Functions in python

- Creating clean repeatable code is a key part of becoming an effective programmer.
- A function is a block of code which only runs when it is called.
- In Python a function is defined using the def keyword:

Syntax

```
def function_name() :
    #code to execute when function is called
```

ends with :

Indentation (tab/whitespace) at the beginning

functiondemo.py

```
def seperator() :
    print('================================')

print("hello world")
seperator()
print("from JU")
seperator()
print("Kolkata")
```

Output :

```
hello world
================================
from JU
================================
Kolkata
```

Function (cont.) (DOCSTRIGN & return)

- Doc string helps us to define the documentation about the function within the function itself.

Syntax

```
def function_name() :
    '''
        DOCSTRING: explains the function
            INPUT: explains input
            OUTPUT: explains output
    '''
    #code to execute when function is called
```

Enclosed within triple quotes

- **return statement** : return allows us to assign the output of the function to a new variable, return is use to send back the result of the function, instead of just printing it out.

whiledemo.py

```
def add_number(n1,n2) :
    return n1 + n2

sum1 = add_number(5,3)
sum2 = add_number(6,1)
print(sum1)
print(sum2)
```

Output :

```
8
7
```