

Topic TWO

Python Programming Basics

Preparation:

Step 1: Go to <https://www.youtube.com/watch?v=OlAxTC7EqIg> or any other YouTube video about Python programming to catch up with the basic syntax.

Step 2: Follow the instructions as below to complete the lab exercises.

1. Learning Objectives

When you have completed this lab, you should be able to:

1. Setup Anaconda open-source distribution
2. Setup Jupyter notebook
3. Use Jupyter notebook to run Python programming examples
4. Install Python libraries
5. Import libraries in Python
6. Create python Classes / Objects
7. Use **Numpy** to perform manipulations, crunching vectors and matrices
8. Use **Pandas** to load a dataset, manipulate and analyze data
9. Use **Matplotlib** to create graphs, histogram and bar plot, etc.

2. Introduction to Python

Python is an interpreted, object-oriented, high-level programming language created by Guido van Rossum, and released in 1991. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc) and has syntax that allows developers to write programs with fewer lines than some other programming languages. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

3. Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment on Windows, Mac and Linux OS platforms.

Anaconda distribution comes with more than 1,500 packages as well as the conda package and virtual environment manager. It includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI). It also comes with Jupyter and Spyder Python Development Environments.

4. Jupyter Notebook

Jupyter Notebook is a web application that allows you to create and share documents that contain:

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

Jupyter Notebook is great for the following use cases:

- learn and try out Python
- data processing / transformation
- numeric simulation
- statistical modeling
- machine learning

5. Setting up Anaconda

The following steps will guide you through the Anaconda Installation process:

- Go to <https://www.anaconda.com/download>
- Click Download in Figure 1.
-

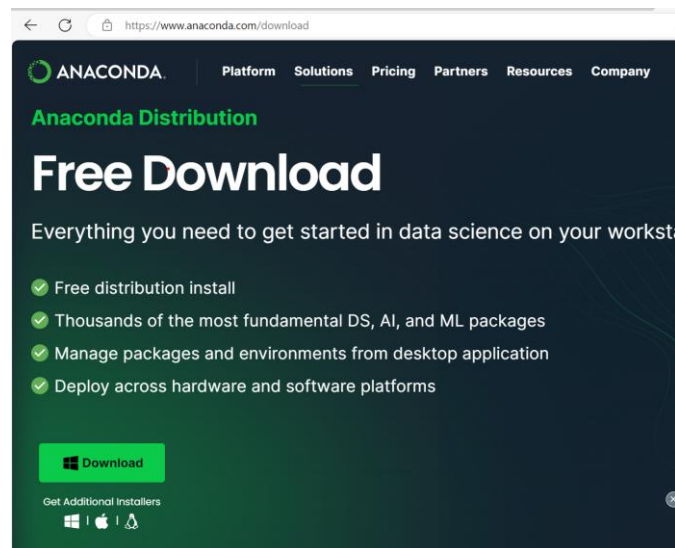


Figure 1 Anaconda Download

- After download is completed, you can install the Anaconda distribution by following on screen instructions.

6. Opening an Anaconda Navigator

To start Anaconda, you can go to windows search menu and type “anaconda navigator” and click open “Anaconda Navigator (Anaconda3)” as shown in figure 2.

Official (Closed), Non-Sensitive

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

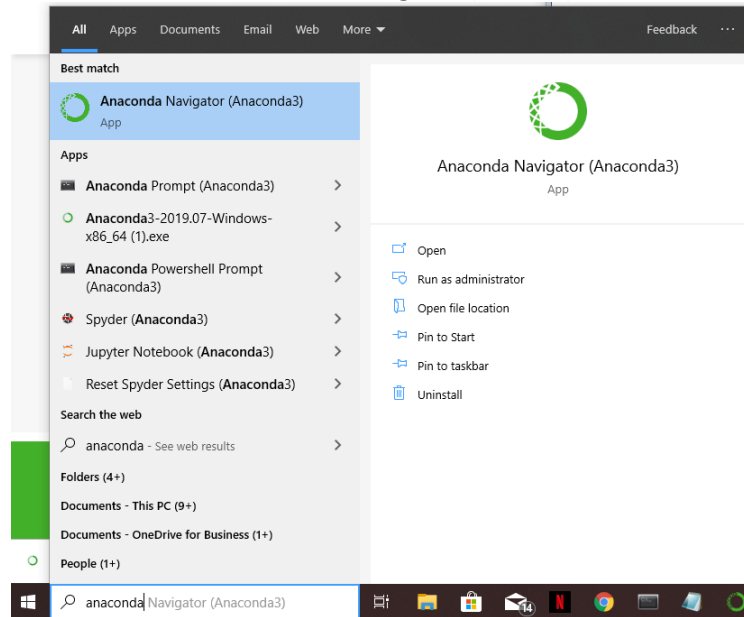


Figure 2 Navigating to “Anaconda Navigator”

Once Anaconda Navigator is launched, you will see the installed applications (JupyterLab, Jupyter Notebook, Spyder, Glueviz, etc) on it.

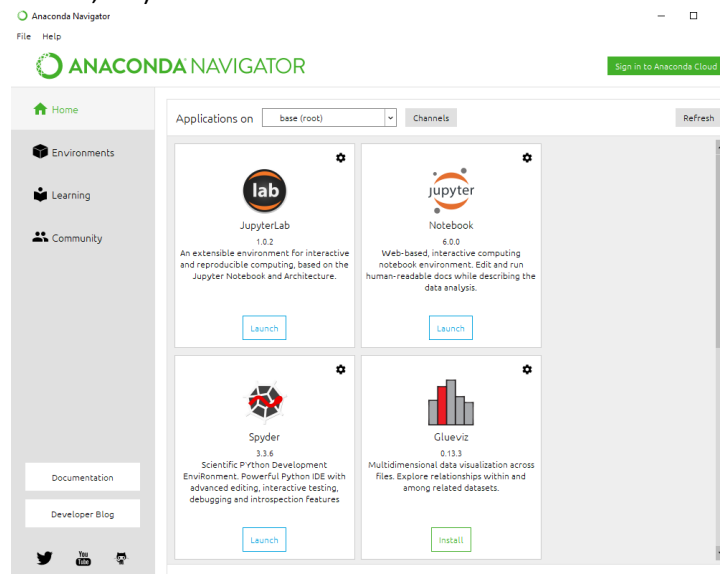


Figure 3 Anaconda Navigator Menu

7. Starting a Jupyter Notebook

To start a Jupyter Notebook application, you can click on Launch button located at Jupyter Notebook section.

Official (Closed), Non-Sensitive
School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

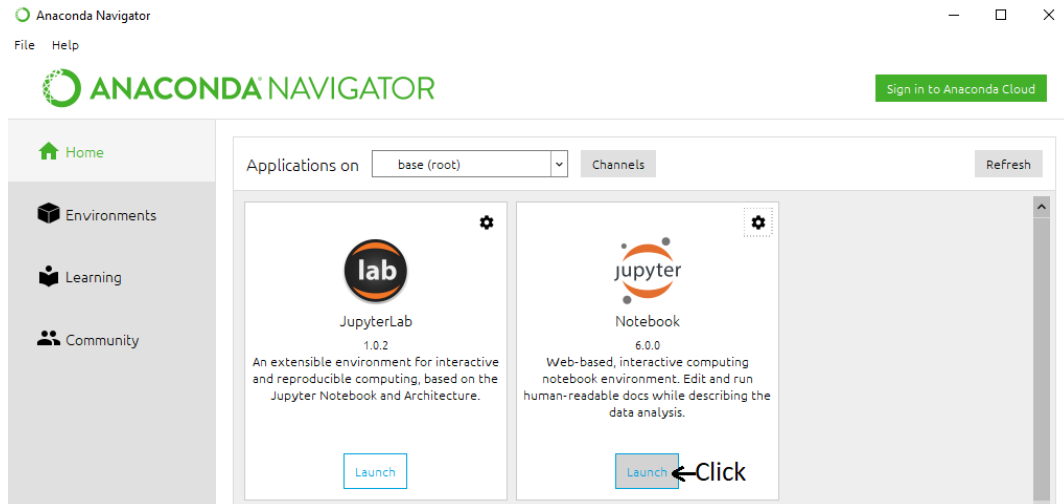


Figure 4 Launching a Jupyter Notebook

After click Launch, a web browser will be launched and will display Jupyter program main page.

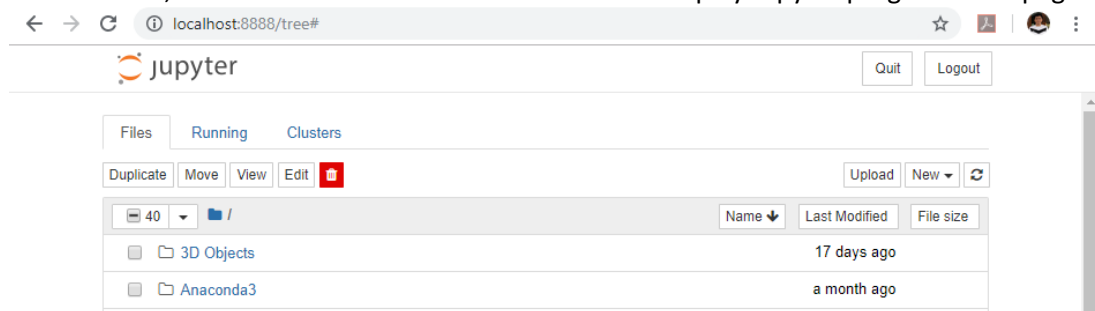


Figure 5 Jupyter Notebook Program main page

7.1 Create a new notebook

To create a new project, you can click on “New” button and choose Python 3. A new web page will be displayed. You can rename the project by clicking on “Untitled1”, type “Lab1” and click “Rename”.

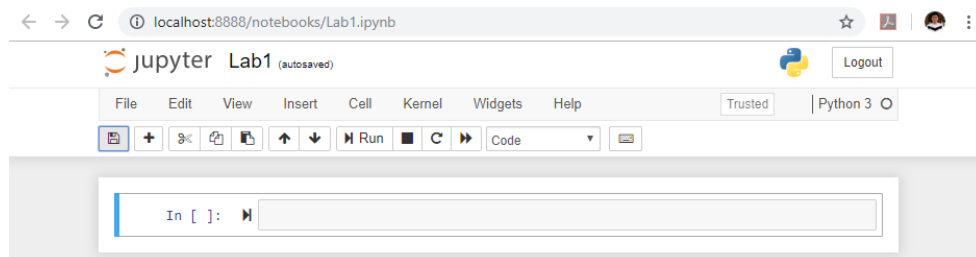


Figure 6 Jupyter Notebook

7.2 Save the notebook

The notebook is designed to save automatically but you can save it manually by clicking the save button or go to File menu → Select “Save and Checkpoint”. Lab1 will be saved under Jupyter Notebook Program Main Page.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

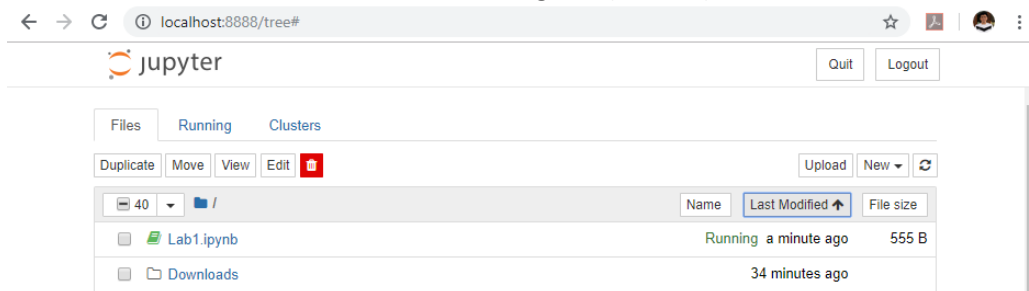


Figure 7 ipynb located on Jupyter Notebook Main Page

Do note that ipynb is the file extension for Jupyter notebooks.

7.3 Export the notebook

In the File menu, select Download as where you can download the notebook in multiple formats. You can choose the format based on what you intend to use it for.

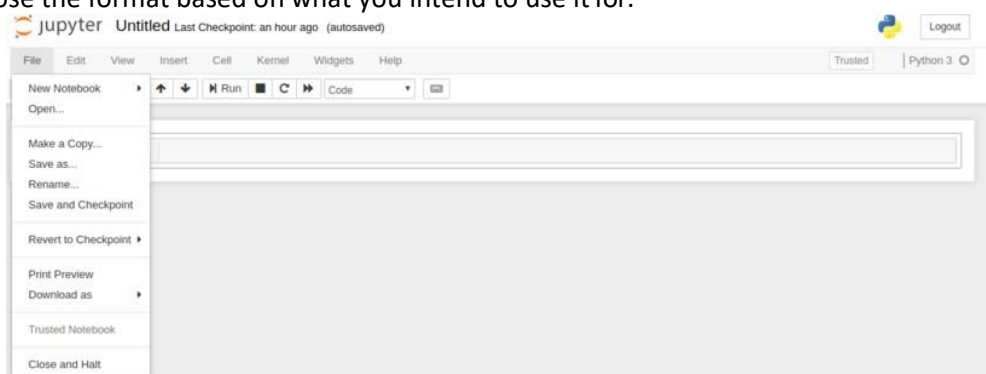


Figure 8 Exporting the notebook

7.4 Shutdown the notebook

You can shutdown the notebook by going to the Kernel menu and clicking on Shutdown. Make sure to save your work before you do this!

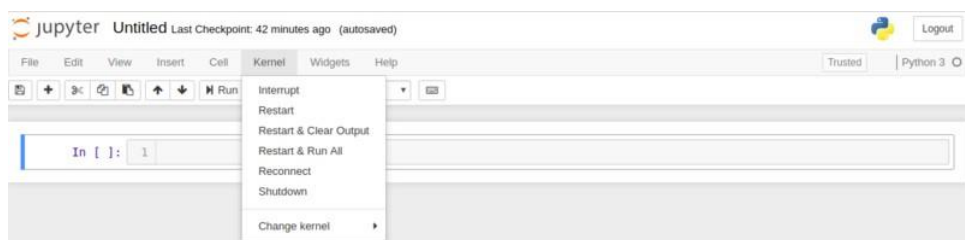


Figure 9 Shutting down the notebook

You can also shutdown the entire server by pressing Ctrl + c in the terminal where you started your Jupyter notebook server. Now, you can start performing the following Python exercises.

7.5 Jupyter Notebook Interface

You can see a box as shown in the image below. This is called a cell.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

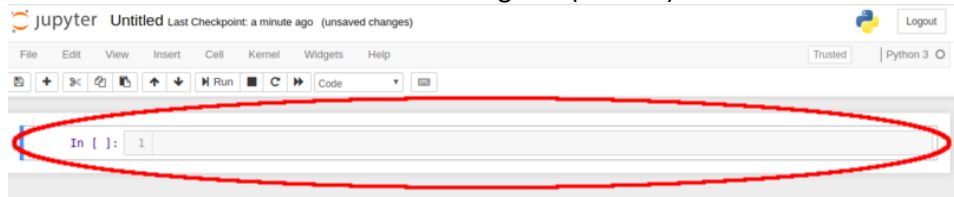


Figure 10 A cell

Cells are where you write and run your code or text. You can change the cell type from Code to Markdown and vice versa. Markdown is a popular formatting syntax for writing web content which can also be used to write a report. In the toolbar, click Code to change it to Markdown and back.

The little play button (indicated as Run) runs the cell, and the up and down arrows move cells up and down. When you run a code cell, the output is displayed below the cell. The cell also gets numbered, you see In [1]: on the left. This lets you know the code was run and the order if you run multiple cells. Running the cell in Markdown mode renders the Markdown as text.

You can also run the code in the cell by pressing Ctrl-Enter. Alt-Enter or Shift-Enter will run the code and create a new cell.

7.6 The Toolbar

Below briefly describes the toolbar buttons starting from left:

- The floppy disk symbol saves the notebook.
- The + button creates a new cell.
- The next three buttons are to cut, copy, and paste cells.
- The up and down arrows move cells up or down.
- The next four buttons are to run, stop, restart the kernel, and restart & re-run the entire notebook.
- The dropdown box is to choose the Cell type (eg. Code or Markdown).
- The keyboard symbol is the Command palette. This will bring up a panel with a search bar where you can search for various commands. This is helpful for speeding up your workflow. Just open the command palette and type in what you want to do.

8. Python Syntax Basics _Lab1

8.1 Python Indentations

The indentation in Python is very important as it indicates a block of code. See the example below.

```

> if 5 > 2:
    print("Five is greater than two!")

Five is greater than two!

```

The equivalent of the above code in C++ is below.

```

if 5 > 2
{
    cout<<"Five is greater than two!";
}

```

If you skip the indentation in the above Python example you will see the following error.

```

> if 5 > 2:
> print("Five is greater than two!")

File "<ipython-input-3-a314491c53bb>", line 2
    print("Five is greater than two!")
    ^
IndentationError: expected an indented block

```

8.2 Comments

You can use the # operator to start a comment. Just like the // in C++. See the example below.

```
# this is a comment. print("Hello, World!")
```

8.3 Variables

Unlike other programming languages, you do not have to declare a variable in Python. A variable is created the moment you first assign a value to it. See the example below.

```

a = 5                # a is a variable of type integer
b = 1.2              # b is a variable of type float
c = "ET0732"         # c is a variable of type string

```

The following rules apply for naming variables in Python:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

8.4 Casting

Casting in python is done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

See the example below.

```

x = int(1)           # x will be 1
y = int(2.8)         # y will be 2
z = int("3")         # z will be 3
w = float("4.2")     # w will be 4.2
s = str(3.0)         # s will be 3.0

```

8.5 Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

8.5.1 List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets. See the example below.

```
my_list = ["apple", "banana", "cherry"]
```

You access the list items by referring to the index number. See the example below.

```
print(my_list[1])
```

```
▶ my_list = ["apple", "banana", "cherry"]  
print(my_list[1])
```

The output would be _____.

You can loop through the list items by using a for loop. See the example below.

```
▶ for x in my_list: print(x)
```

The output would be _____

8.5.2 Tuple

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets. See the example below:

```
my_tuple = ("apple", "banana", "cherry")
```

8.5.3 Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets. See the example below.

```
my_set = {"apple", "banana", "cherry"}
```

8.5.4 Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values. See the example below.

```
my_dict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

You can access the items of a dictionary by referring to its key name, inside square brackets. See the example below.

```
x=my_dict["model"]
```

8.6 Conditions and if statements

An *if statement* is written by using the if keyword. See the example below.

```
▶ a = 100  
b = 200  
if b > a:  
    print("b is greater than a")  
  
b is greater than a
```


The **elif** keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

```

>>> a = 1
>>> b = 1
>>> if b > a:
>>>     print("b is greater than a")
>>> elif a == b:
>>>     print("a and b are equal")
a and b are equal

```

The **else** keyword catches anything which is not caught by the preceding conditions

```

>>> a = 200
>>> b = 100
>>> if b > a:
>>>     print("b is greater than a")
>>> elif a == b:
>>>     print("a and b are equal")
>>> else:
>>>     print("a is greater than b")
a is greater than b

```

The "and" keyword is a logical operator, and is used to combine conditional statements. See the example below.

```

if a > b and c > a:
    print("Both conditions are True")

```

The or keyword is a logical operator, and is used to combine conditional statements. See the example below.

```

if a > b or a > c:
    print("At least one of the conditions is True")

```

8.7 Loops

Python has two primitive loop commands: * while loops * for loops

8.7.1 The *while* Loop

With the while loop, we can execute a set of statements as long as a condition is true. See the example below.

```

>>> i = 0
>>> while i < 10:
>>>     print(i)
>>>     i += 1

```

The output would be _____.

8.7.2 The *break* Statement

With the break statement, we can stop the loop even if the while condition is true. See the example below.

```
➤ i = 0
while i < 10:
    print(i)
    if i == 3:
        break
    i += 1
```

The output would be _____.

8.7.3 The *continue* Statement

With the *continue* statement we can stop the current iteration, and continue with the next. See the example below.

```
➤ i = 0
while i < 10:
    i += 1
    if i == 3: continue
    print(i)
```

The output would be _____.

8.7.4 The *for* Loop

A *for* loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). With the *for* loop we can execute a set of statements, once for each item in a list, tuple, set etc. You can use *break* and *continue* statements in *for* loops.

```
➤ fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

The output would be _____.

The *range()* Function To loop through a set of code a specified number of times, we can use the *range()* function. It returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. See the example below.

```
➤ # values from 0 to 5
for x in range(6):
    print(x)
```

The output would be _____.

It is possible to change the *range()* default values. The example below iterates from 2 to 29 and increments by 3.

```
➤ for x in range(2, 30, 3):
    print(x)
```

The output would be _____.

8.8 Functions

8.8.1 Definition

A **function** is a block of code which only runs when it is called. You can pass data, known as parameters, into a **function**. A **function** can return data as a result.
 In Python a **function** is defined using the def keyword. See the example below.

```
def my_function():
    print("My function!")
```

To call a function, use the function name followed by parenthesis.

```
def my_function():
    print("My function!")

my_function()
```

The output would be _____.

Information can be passed to functions as parameter. Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma. The following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name.

```
def my_function(fname):
    print(fname + " Yang")

my_function("Joe")
my_function("Tracey")
```

The output would be _____

To let a function to return a value, you can use the return statement. See the example below.

```
def my_function(x):
    return 5 * x

print(my_function(3))
```

The output would be _____.

8.8.2 Function Customization

- (1) Define a function to return a sequence. Each number in the sequence is the sum of the former two numbers (Fibonacci sequence).

```
def fibs(num):
    result = [0,1]
    for i in range(2,num):
        a = result[i-1] + result[i-2]
        result.append(a)
    return result

fibs(5)
```

Position parameter.
 # Create a list to store the sequence value.
 # Cycle num-2 times.
 # Append the value to the list.
 # Return the list.

Output:

```
[0, 1, 1, 2, 3]
```

- (2) Define a function which can be customized to generate parameters transferred in different ways.

```
def hello(greeting='hello',name='world'):          # Default parameters.
    print('%s, %s!' % (greeting, name))           # Format the output.
hello()                                           # hello, world   Default parameter.
hello('Greetings')                               # Greetings, world   Position parameter.
hello ('Greetings', 'universe')                  # Greetings, universe   Position parameter.
hello (name= 'Gumby')                            # hello, Gumby   Keyword parameter.
```

Output:

```
hello, world!
Greetings, world!
Greetings, universe!
hello, Gumby!
```

9. Python – Object Oriented Programming (OOP)

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy. The concept of OOP in Python focuses on creating reusable code.

9.1 Class and Object

In general, Classes are a way of grouping together related data and functions which act upon that data. The simplest class possible is shown in the following example:

```
class MyClass:
    x = 5
```

9.2 Object

An object is a unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods. Here, we will create an object named obj1 and print the value of variable x.

```
obj1 = MyClass()
print(obj1.x)
```

Output would be _____ .

9.3 Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object. Let's see the example below.

```
class MyClass:
    def say_hi(self):
        print('Hello, how are you?')
p = MyClass()
p.say_hi()
Output would be _____.
```

9.4 The `__init__()` Function

All classes have a function called `__init__()`, which is always executed when the class is being initiated. Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created. This is similar to a constructor in C++.

Here, we will create class named Person, object p, execute `__init__()` function and print the name and age of the object.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("John", 36)

print(p.name)
print(p.age)
```

9.5 Create/use classes

Create the Dog class. Each instance created based on the Dog class stores the name and age. We will give each dog the `sit ()` and `roll_over ()` abilities.

```
class Dog():
    """A simple attempt to simulate a dog"""
    def __init__(self, name, age):
        """Initialize the name and age attributes."""
        self.name = name
        self.age = age
    def sit(self):
        """Simulate a dog sitting when ordered."""
        print(self.name.title()+"is now sitting")
    def roll_over(self):
        """Simulate a dog rolling over when ordered."""
        print(self.name.title()+"rolled over!")

# Instantiate a class.
dog = Dog ("Husky",2)
dog.sit()
dog.roll_over()

Output:
```

```
Husky is now sitting  
Husky rolled over!
```

9.6 Access attributes

```
class Employee:  
    'Base class of all employees'  
    empCount = 0  
    def __init__(self, name, salary):  
        self.name = name  
        self.salary = salary  
        Employee.empCount += 1  
    def displayCount(self):  
        print("Total Employee %d" % Employee.empCount )  
    def displayEmployee(self):  
        print("Name : ", self.name, " , Salary: ", self.salary)  
  
# Create the first object of the Employee class.  
emp1 = Employee("Zara", 2000)  
# Create the second object of the Employee class.  
emp2 = Employee("Manni", 5000)  
emp1.displayEmployee()  
emp2.displayEmployee()  
print("Total Employee %d" % Employee.empCount)
```

Output:

```
Name : Zara ,Salary: 2000  
Name : Manni ,Salary: 5000  
Total Employee 2
```

9.7 Inheritance

One of the main benefits of object-oriented programming is code reuse, which is achieved through inheritance. Inheritance can be understood as the type and subtype relationships between classes.

```
class Parent:    # Define the parent class.  
    parentAttr = 100  
    def __init__(self):  
        print("Invoke the parent class to construct a function.")  
    def parentMethod(self):  
        print('Invoke a parent class method.')  
    def setAttr(self, attr):  
        Parent.parentAttr = attr  
    def getAttr(self):  
        print("Parent attribute:", Parent.parentAttr)
```

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

```
class Child(Parent):      # Define a child class.
    def __init__(self):
        print("Invoke a child class to construct a method.")
    def childMethod(self):
        print('Invoke a child method.')

c = Child()               # Instantiate a child class.
c.childMethod()           # Invoke the method of a child class.
c.parentMethod()          # Invoke the method of a parent class.
c.setAttr(200)            # Invoke the method of the parent class again to set the
attribute value.
c.getAttr()               # Invoke the method of the parent class again to obtain the
attribute value.
```

Output:

```
Invoke a child class to construct a method.
Invoke the method of a child class.
Invoke the method of a parent class.
Parent attribute: 200
```

9.8 Class attributes and methods

```
class JustCounter:
    __secretCount = 0      # Private variable.
    publicCount = 0        # Public variable.
    def count(self):
        self.__secretCount += 1
        self.publicCount += 1
        print(self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
print(counter.publicCount)
print(counter.__secretCount)
# An error is reported, indicating that the instance cannot access private variables.
```

Output:

```
1
2
2
```

10. Python Libraries

One of the aspects that makes Python such a popular choice in general, is its abundance of libraries and frameworks that facilitate coding and save development time. Machine learning and deep learning are exceptionally well catered for.

Python libraries are collections of functions and methods that allows you to perform many actions without writing your code. Most of the libraries are available with free license for commercial use. In fact, most of them are even open source, but some of the more complex libraries that involves heavy computation are not as easy to use as other python modules. They are free in every sense but their optimization is left to the developer using it.

10.1 Numpy

Numpy is one of the most widely used library in Python for numeric computing. It greatly facilitates the handling of vectors and matrices. Mastering numpy will give you an edge when dealing and debugging with advanced use cases in these libraries.

10.2 Pandas

Pandas is a popular data manipulation library that provides data structures of high-level and a wide variety of tools for analysis. One of the great features of this library is the ability to translate complex operations with data using one or two commands. Pandas has many inbuilt methods for grouping, combining data, and filtering, as well as time-series functionality.

Pandas supports operations such as Re-indexing, Iteration, Sorting, Aggregations, Concatenations and Visualizations.

11. Python Library / Package Managers (Conda and PIP)

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. The conda package and environment manager is included in all versions of Anaconda.

PIP is the tool for installing packages from the Python Package Index, PyPI. PIP installs Python software packaged as wheels or source distributions. The latter may require that the system have compatible compilers, and possibly libraries, installed before invoking pip to succeed.

Conda and PIP are often considered as being nearly identical. Although some of the functionality of these two tools overlap, they were designed and should be used for different purposes.

In this lab, we will be using Conda package manager to manage python libraries.

	conda	pip
manages	binaries	wheel or source
can require compilers	no	yes
package types	any	Python-only
create environment	yes, built-in	no, requires virtualenv or venv
dependency checks	yes	no
package sources	Anaconda repo and cloud	PyPI

Figure 11 Comparison of conda and pip package managers

Source: <https://www.anaconda.com/understanding-conda-and-pip/>

12. Managing Python Libraries

In this section, you will learn the basics of managing packages in Anaconda.

The following steps will guide you through the process to launch Anaconda Prompt (CLI) to manage python libraries:

12.1 Opening an Anaconda Prompt

To start Anaconda Prompt, you can go to windows search menu and type “anaconda prompt” and click open “Anaconda Prompt as shown in figure 12.

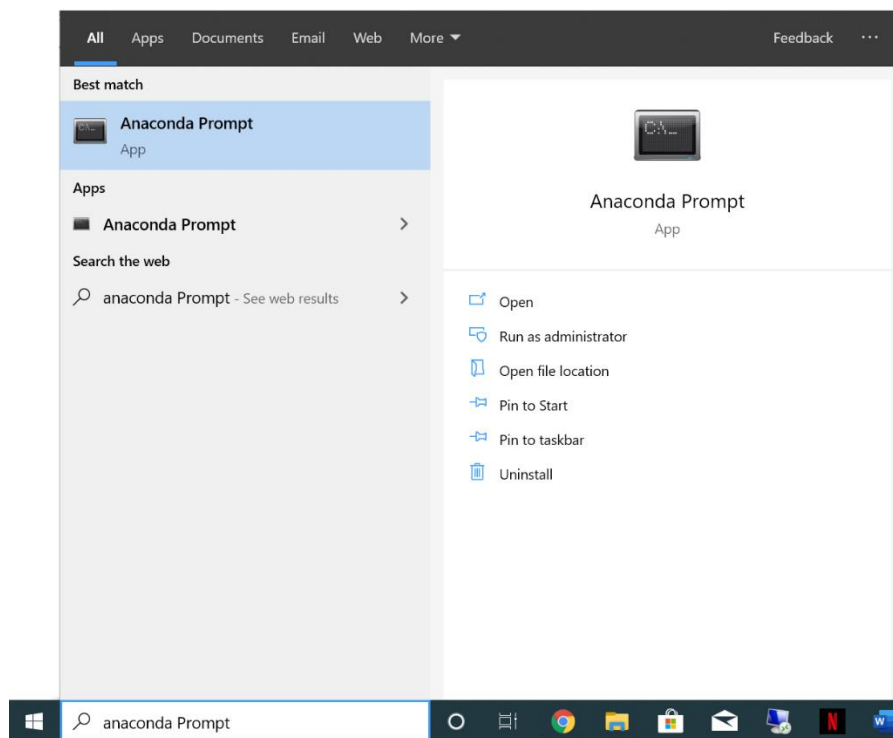


Figure 12 Navigating to “Anaconda Navigator”

Once Anaconda Prompt is launched, you will see CLI windows as shown in figure 13.

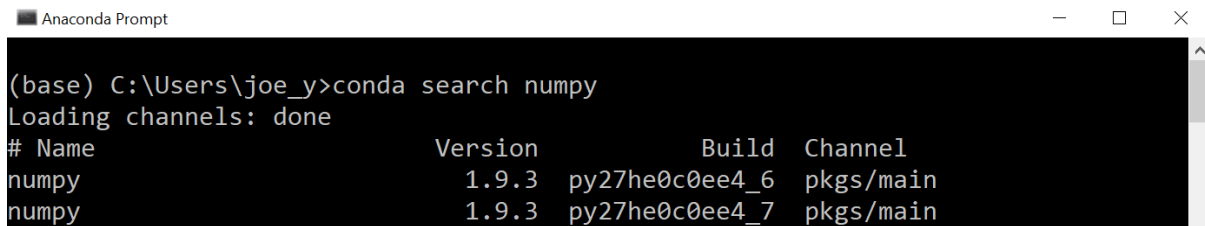


```
Anaconda Prompt
(base) C:\Users\joe_y>
```

Figure 13 Anaconda Prompt CLI Window

12.2 Searching for Packages

To see if a specific package, such as numpy, is available for installation:



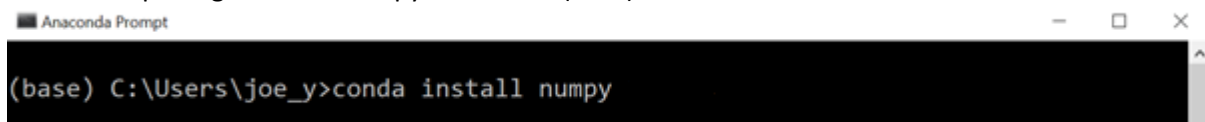
```
Anaconda Prompt
(base) C:\Users\joe_y>conda search numpy
Loading channels: done
# Name                Version          Build          Channel
numpy                 1.9.3            py27he0c0ee4_6 pkgs/main
numpy                 1.9.3            py27he0c0ee4_7 pkgs/main
```

To check that the packages installed:



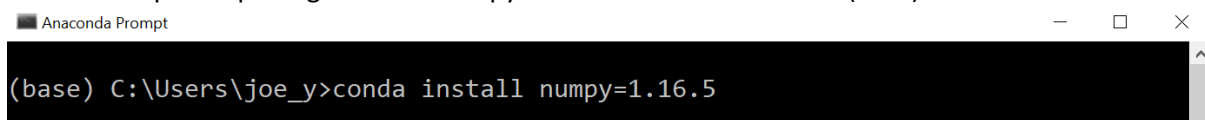
```
Anaconda Prompt
(base) C:\Users\joe_y>conda list
```

To install a package such as numpy into a root (base) environment.



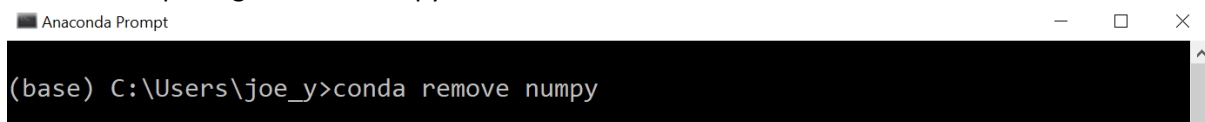
```
Anaconda Prompt
(base) C:\Users\joe_y>conda install numpy
```

To install a specific package such as numpy version 1.16.5 into an root (base) environment.



```
Anaconda Prompt
(base) C:\Users\joe_y>conda install numpy=1.16.5
```

To remove a package such as numpy in the current environment:



```
Anaconda Prompt
(base) C:\Users\joe_y>conda remove numpy
```

You may refer to the below conda docs website to learn more about managing packages.

<https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html>

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

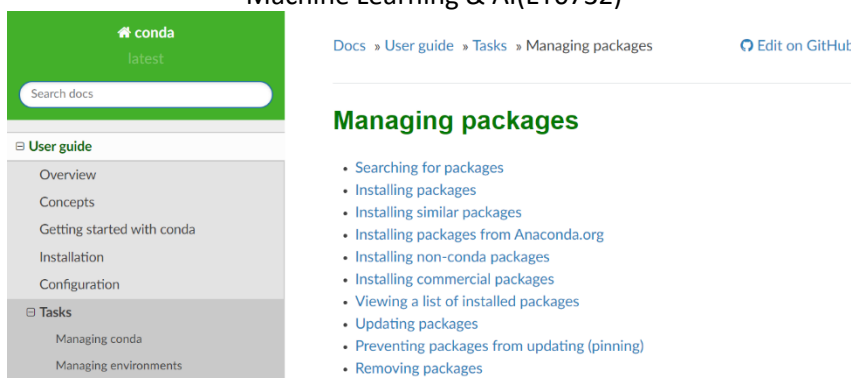


Figure 14 Conda Docs (Managing Packages)

12.3 Creating a Virtual Environment

Creating virtual environments in Anaconda is fairly simple. You can use the following template to create a virtual environment:

conda create -n env_name list_of_packages

Note the following points about creating environments:

It is a good practice to choose a descriptive name for your environment that would help you know which environment to use in the future. For example, if you intend you use an environment for Python version 3.3, you can name it as python33.

The list of packages is not necessary when creating an environment. You can install your packages after creating the environment.

Here are some examples for creating environments:

Create an environment that comes with numpy installed:

conda create -n my_env numpy

Create an environment that comes with Python version 3.3:

conda create -n my_env python=3.3

NOTE: Use unique names when creating environments. Replace the example environment names above with names of your preference but follow the suggested tips on naming schemes.

Using Virtual Environments

After creating an environment, you are ready to use it. You only need one step to start using your virtual environment. Use the following template to activate your virtual environment:

conda activate env_name

If you would like to deactivate an environment, just use the following template:

conda deactivate

If you do not remember your environment name, you can use the following command to see the list virtual environments:

conda env list

How many virtual environments have you created as part of this lab? List them down.

13. Starting a Jupyter Notebook

To start a Jupyter Notebook application, you can type “jupyter notebook” on the Anaconda Prompt as shown in figure 15.

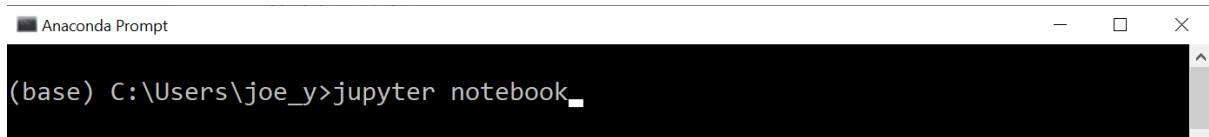


Figure 15 Starting a Jupyter Notebook from Anaconda Prompt

After Launching, a web browser will be launched and will display Jupyter program main page.

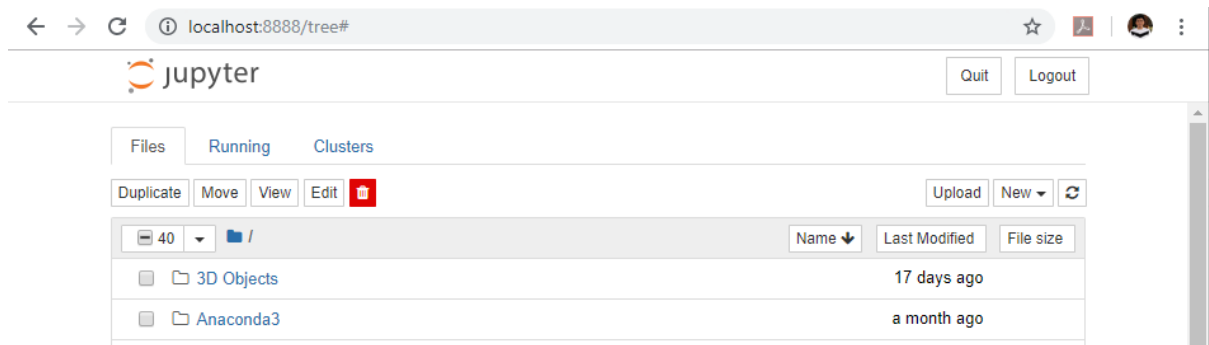


Figure 16 Jupyter Notebook Program main page

To create a new project, you can click on “New” button and choose Python 3. A new web page will be displayed. You can rename the project by clicking on “Untitled1”, type “Lab2” and click “Rename”.

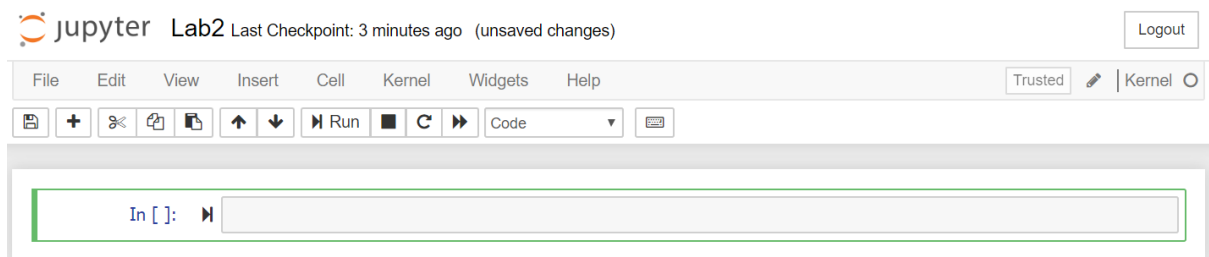


Figure 17 Jupyter Notebook

The notebook is designed to save automatically but you can save it manually by clicking the save button or go to File menu → Select “Save and Checkpoint”. Lab2 will be saved under Jupyter Notebook Program Main Page.

Now, you can start performing the following Python exercises.

14. Python Library Exercises _Lab2

14.1 Numpy

NumPy is a library that provides alternatives to mathematical operations in Python and is designed to work efficiently with groups of numbers such as matrices. NumPy is a large library and we are only going to cover the basics here.

14.1.1 Importing Numpy library

When importing the NumPy library, the convention you will see used most often is to name it np.

```
import numpy as np
```

14.1.2 Creating a numpy Array

A numpy array is a grid of values, all the same type, and is indexed by a tuple of integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

Type the following code inside the jupyter notebook and run it.

```
In [1]: ▶ import numpy as np  
arr = np.array([])  
type(arr)
```

```
Out[1]: numpy.ndarray
```

Observe that the type(arr) will return an output “numpy.ndarray”.

Next, we going to create a one-dimensional array and output the dimension and size of that array.

```
In [2]: ▶ one_d_array = np.array([1, 2, 3, 4, 5])  
# ndim attributes shows the number of dimension of an array  
one_d_array.ndim
```

```
Out[2]: 1
```

```
In [3]: ▶ # size attributes returns the size/length of the array  
one_d_array.size
```

```
Out[3]: 5
```

The dimension of the array is _____ and the size is _____.

14.1.3 Creating a Sequence Number

```
In [4]: ▶ # if a single parameter was passed then the sequence was start from 0.
print(np.arange(10))
```

[0 1 2 3 4 5 6 7 8 9]

```
In [5]: ▶ # first parameter denotes the starting point
# second parameter denotes the ending point
# if the third parameter was not specified then 1 is used as default step
print(np.arange(1, 10))
```

[1 2 3 4 5 6 7 8 9]

```
In [6]: ▶ # here 2 is for the steps
print(np.arange(1, 10, 2))
```

[1 3 5 7 9]

14.1.4 Reshaping and Flattening an Array

In the In[7] below, it shows that one-dimensional array can be reshaped into two-dimensional array.

```
In [7]: ▶ np.arange(10).reshape(2, 5)
```

Out[7]: array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9]])

In the In[8] below, it shows that two-dimensional array can be flattened into one-dimensional array.

```
In [8]: ▶ two_d_arr = np.arange(10).reshape(2, 5)
print(two_d_arr)
two_d_arr.ravel()
```

[[0 1 2 3 4]
[5 6 7 8 9]]

Out[8]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

14.1.5 Array List and Index

Numpy offers several ways to index into arrays. Look at an example below.

```
In [9]: ▶ a = np.array([1, 2, 3]) # Create a rank 1 array
print(type(a)) # Prints "<class 'numpy.ndarray'>"
print(a.shape) # Prints "(3,)"
print(a[0], a[1], a[2]) # Prints "1 2 3"
a[0] = 5 # Change an element of the array
print(a) # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
print(b.shape) # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0]) # Prints "1 2 4"
```

Observe the output from each print statement.

14.1.6 Array Slicing

Slicing is specified using the colon operator ':' with a 'from' and 'to' index before and after the column respectively. The slice extends from the 'from' index and ends one item before the 'to' index.

One-Dimensional Slicing

```
In [10]: # define array
data = np.array([11, 22, 33, 44, 55])
print(data[:])
print(data[0:1])
```

[11 22 33 44 55]
[11]

Two-Dimensional Slicing

```
In [11]: data = np.array([[11, 22, 33],[44, 55, 66],[77, 88, 99]])
# separate data
X, Y = data[:, :-1], data[:, -1]
print(X)
```

[[11 22]
[44 55]
[77 88]]

```
In [12]: print(Y)
```

[33 66 99]

14.1.7 Array Stacking

Stacking joins a sequence of arrays along an existing axis.

```
In [13]: a = np.arange(0, 20, 2)
b = np.arange(10)
vs = np.vstack([a, b])
print(vs)
```

[[0 2 4 6 8 10 12 14 16 18]
[0 1 2 3 4 5 6 7 8 9]]

```
In [14]: hs = np.hstack([a, b])
print(hs)
```

[0 2 4 6 8 10 12 14 16 18 0 1 2 3 4 5 6 7 8 9]

14.1.8 Matrices

You create matrices using NumPy's array function. However, instead of just passing in a list, you need to supply a list of lists, where each list represents a row. So to create a 3x3 matrix containing the numbers one through nine, you could do this:

You create matrices using NumPy's array function. However, instead of just passing in a list, you need to supply a list of lists, where each list represents a row. So to create a 3x3 matrix containing the numbers one through nine, you could do this:

```
m = np.array([[1,2,3], [4,5,6], [7,8,9]])
```

Checking its shape attribute (m.shape) would return the tuple (3, 3) to indicate it has two dimensions, each length 3. You can access elements of matrices just like vectors, but using additional index values. So to find the number 6 in the above matrix, you could do this:

```
n = m[1][2]  
print(n)
```

14.1.9 Matrices Multiplication

Matrix multiplication is an operation that takes two matrices as input and produces single matrix by multiplying rows of the first matrix to the column of the second matrix. In matrix multiplication make sure that the number of rows of the first matrix should be equal to the number of columns of the second matrix.

See example below:

```
In [15]: a = [[1, 0], [0, 1]]  
        b = [[4, 1], [2, 2]]  
        c = np.dot(a, b)  
        print(c)  
[[4 1]  
 [2 2]]
```

14.2 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

14.2.1 Importing Pandas library

When importing the pandas library, the convention you will see used most often is to name it pd.

```
import pandas as pd
```

14.2.2 Create a Pandas DataFrame

DataFrames is your first step in almost anything that you want to do when it comes to data munging in Python. Sometimes, you will want to start from scratch, but you can also convert other data structures, such as lists or NumPy arrays, to Pandas DataFrames.

Run the code below and observe the output from pd.DataFrame.


```

import numpy as np
import pandas as pd

data = np.array([[ '', 'Col1', 'Col2'],
                 ['Row1', 1, 2],
                 ['Row2', 3, 4]])

print(pd.DataFrame(data=data[1:,1:],
                   index=data[1:,0],
                   columns=data[0,1:]))

```

	Col1	Col2
Row1	1	2
Row2	3	4

Try it out the code below:

Remember that the Pandas library has already been imported for you as pd.

Take a 2D array as input to your DataFrame

```

my_2darray = np.array([[1, 2, 3], [4, 5, 6]])
print(pd.DataFrame(my_2darray))

```

The output would be _____.

Take a dictionary as input to your DataFrame

```

my_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}
print(pd.DataFrame(my_dict))

```

The output would be _____.

Take a DataFrame as input to your DataFrame

```

my_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4), columns=['A'])
print(pd.DataFrame(my_df))

```

The output would be _____.

Take a Series as input to your DataFrame

```

my_series = pd.Series({"United Kingdom":"London", "India":"New Delhi", "United States":"Washington", "Belgium":"Brussels"})
print(pd.DataFrame(my_series))

```

The output would be _____.

Using 'shape' and 'len()' properties functions with DataFrame

```
df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))
```

```
# Use the 'shape' property  
print(df.shape)
```

The output would be _____ .

```
# Or use the 'len()' function with the 'index' property  
print(len(df.index))
```

The output would be _____ .

14.2.3 Adding a Column to a DataFrame

In some cases, you want to make your index part of your DataFrame. You can easily do this by taking a column from your DataFrame or by referring to a column that you haven't made yet and assigning it to the `.index` property, just like this:

```
# df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns=['A', 'B', 'C'])  
  
# Use '.index'  
df['D'] = df.index  
  
# Print `df`  
print(df)
```

```
   A  B  C  D  
0  1  2  3  0  
1  4  5  6  1  
2  7  8  9  2
```

```
# Use '.index'  
df['D'] = df.index  
  
# Print `df`  
print(df)
```

```
   A  B  C  D  
0  1  2  3  0  
1  4  5  6  1  
2  7  8  9  2
```

14.2.4 Deleting a Column from a DataFrame

To get rid of (a selection of) columns from your DataFrame, you can use the `drop()` method:

```
# Drop the column with label 'A'  
df.drop('A', axis=1, inplace=True)  
print(df)
```

```
   B  C  D  
0  2  3  0  
1  5  6  1  
2  8  9  2
```

14.2.5 Removing a Row from a DataFrame

To get rid of row from your DataFrame, you can use the `drop()` method with selected index:

```
# Drop the index at position 1
print(df.drop(df.index[1]))
```

	A	B	C	D
0	1	2	3	0
2	7	8	9	2

14.2.6 Read data from CSV

The following example will show how simple to read a csv file to python environment.

You would need to download the `bmi_and_life_expectancy.csv` file (from Blackboard → Learning Resources → Datasets) and save it into a folder where you launch your jupyter notebook.

```
import pandas as pd

bmi_life_data = pd.read_csv('bmi_and_life_expectancy.csv')
print(bmi_life_data)
```

	Country	Life expectancy	BMI
0	Afghanistan	52.8	20.62058
1	Albania	76.8	26.44657
2	Algeria	75.5	24.59620
3	Andorra	84.6	27.63048
4	Angola	56.7	22.25083
5	Armenia	72.3	25.35542
6	Australia	81.6	27.56373

14.2.7 Write data to CSV

The following example will show how to write a DataFrame as a CSV.

```
import pandas as pd
import numpy as np

df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns=['A', 'B', 'C'])
df.to_csv('myDataFrame.csv')
print(df)
```

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

	A	B	C	D
1		A	B	C
2	0	1	2	3
3	1	4	5	6
4	2	7	8	9

myDataFrame.csv

14.2.8 Read CSV data using the csv library

Create the following file using a text editor for example, notepad. Save this file as csvtst.txt where your Jupyter notebooks are stored.

No, Name, Age
1, KK Chai, 59
2, S Pandian, 44
3, B Ross, 75

Enter the following program which uses the simpler csv library. Note how the data is converted from strings to lists then to numerical arrays.

```
import csv
import numpy as np

f = open('csvtst.txt')
csv_read = csv.reader(f, delimiter=',')

cols = next(csv_read)      # read column names

print(cols[0],cols[1], cols[2])# print them

col0 = []; col1=[]; col2=[]

for row in csv_read:      # read rest

    print(row[0],row[1], row[2])

    # create lists

    col0.append(row[0]); col1.append(row[1]); col2.append(row[2])

f.close()

# convert to numpy array
```

If you are loading data from a file located in other folder, i.e. desktop, you may refer to the example codes as below to open it:

```
f=open("/users/yourname/Desktop/csvtst.txt")
Or
f=open("C:\\users\\yourname\\Desktop\\csvtst.txt")
```

14.3 Matplotlib

Matplotlib is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits. There are several toolkits which are available that extend python matplotlib functionality.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

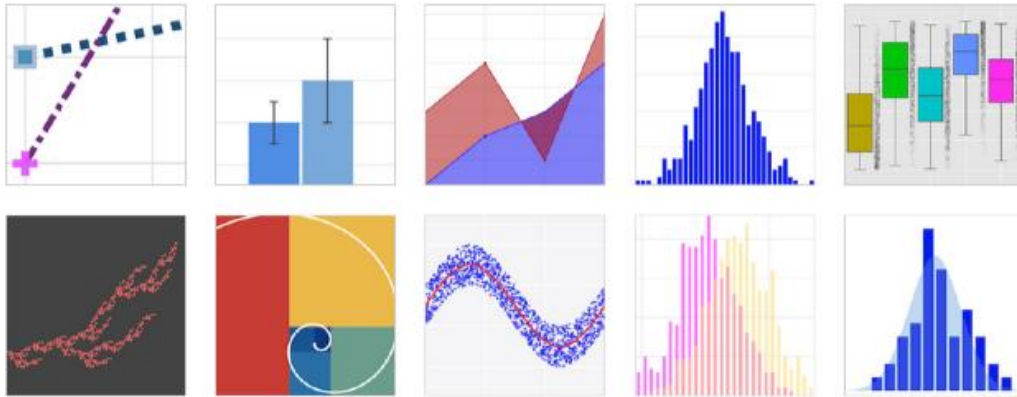


Figure 18 Plotting example using Matplotlib Library

14.3.1 Importing Matplotlib library

When importing the matplotlib library, the convention you will see used most often is to name it plt. Matplotlib is the whole package and matplotlib.pyplot is a module in Matplotlib.

```
import matplotlib.pyplot as plt
```

14.3.2 Plotting in Python using Matplotlib

In general, there are two big components that you need to consider when plotting in python:

- The Figure is the overall window or page that everything is drawn on. It's the top-level component of all the ones that you will consider in the following points. You can create multiple independent Figures. A Figure can have several other things as shown in figure 19.

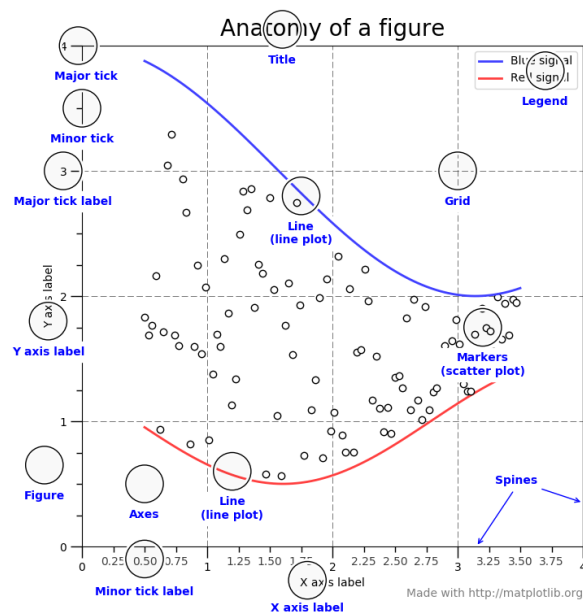


Figure 19 Anatomy of a figure (Source: matplotlib.org)

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)

- To the figure you add Axes. The Axes is the area on which the data is plotted with functions such as plot() and scatter() and that can have ticks, labels, etc. associated with it. This explains why Figures can contain multiple Axes.

Visualization with pyplot is quick. Let's try our first example.

```
import matplotlib.pyplot as plt  
plt.plot([1, 2, 3, 4])  
plt.ylabel('some numbers')  
plt.show()
```

The output of the plot will look like the figure 20 below.

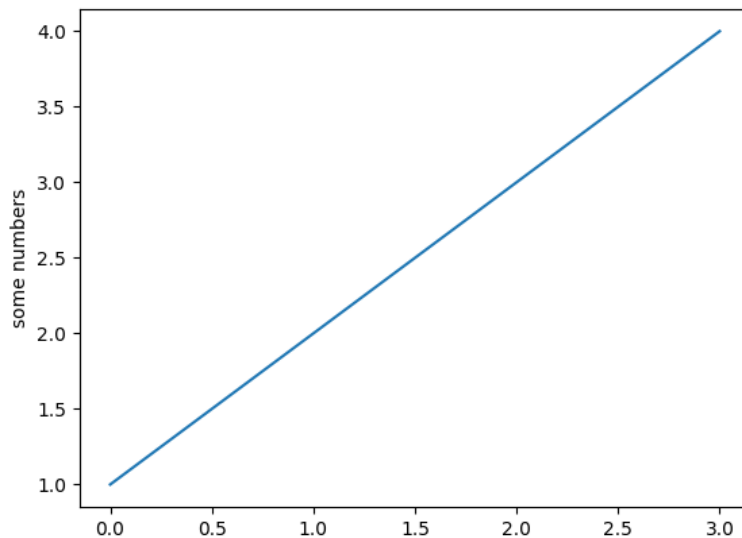


Figure 20 A simple line plot

You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

plot() is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the command:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

The output of the plot will look like the figure 21 below.

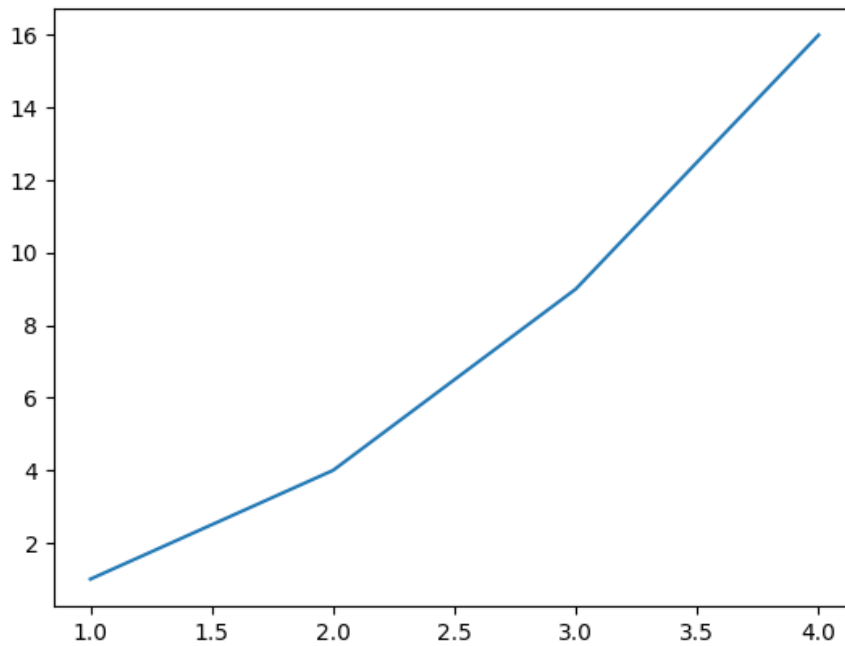


Figure 21 A simple line plot with x and y axes values

14.3.3 Formatting the style of the plot

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. For example, to plot the above with red circles, you could issue the code:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')  
plt.axis([0, 6, 0, 20])  
plt.show()
```

The output of the plot will look like the figure 22.

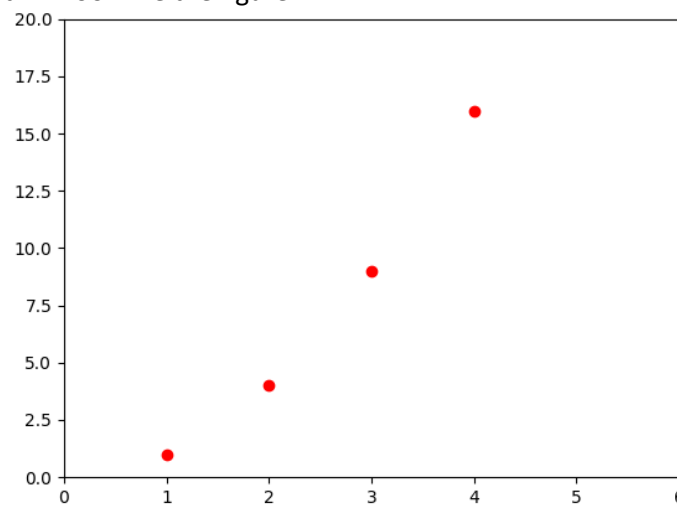


Figure 22 A simple plot with red circles

Next, the example below illustrates a plotting several lines with different format styles in one

command using arrays.

```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

The output of the plot will look like the figure 23 below.

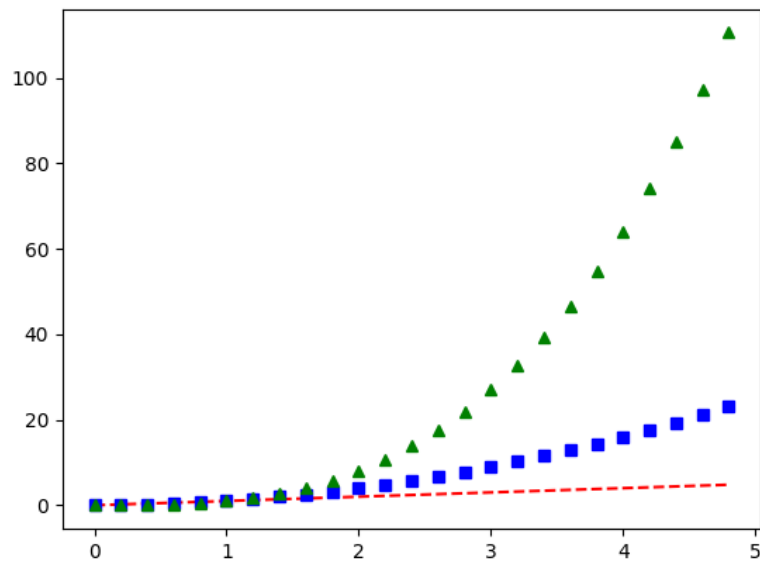


Figure 23 A simple plot with multiple lines and styles

14.3.4 Plotting with categorical variables

Matplotlib allows you to pass categorical variables directly to many plotting functions. For example:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
```

```
plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```

The output of the plot will look like the figure 24 below.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI(ET0732)
Categorical Plotting

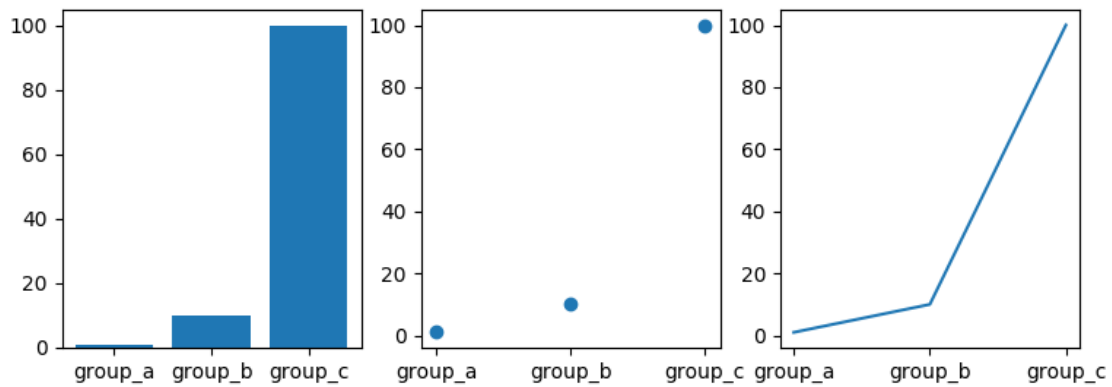


Figure 24 Plotting with categorical variables

14.3.5 Working with multiple figurers and axes

MATLAB, and pyplot, have the concept of the current figure and the current axes. All plotting commands apply to the current axes. Let's try the example below.

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
```

```
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
```

```
plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

The output of the plot will look like the figure 25 below.

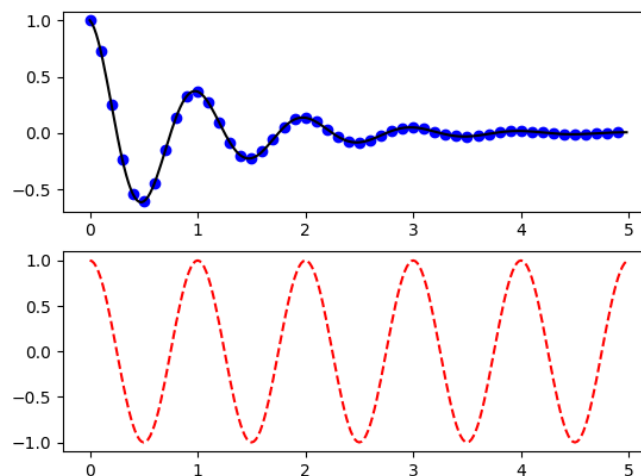


Figure 25 Plotting with multiple figures and axes

14.3.6 Working with text

The text() command can be used to add text in an arbitrary location, and the xlabel(), ylabel() and title() are used to add text in the indicated locations. For example:

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100, \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

The output of the plot will look like the figure 26.

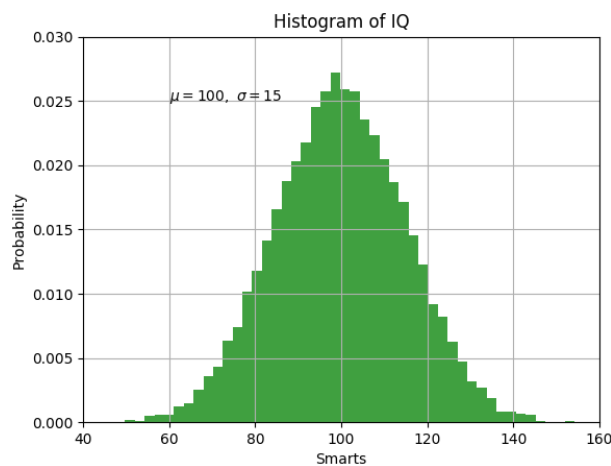


Figure 26 Adding text to a plot

14.3.7 Annotating text

The uses of the basic text() command above place text at an arbitrary position on the Axes. A common use for text is to annotate some feature of the plot, and the annotate() method provides helper functionality to make annotations easy. Let's try the example below.

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
```

```
plt.ylim(-2, 2)  
plt.show()
```

The output of the plot will look like the figure 27.

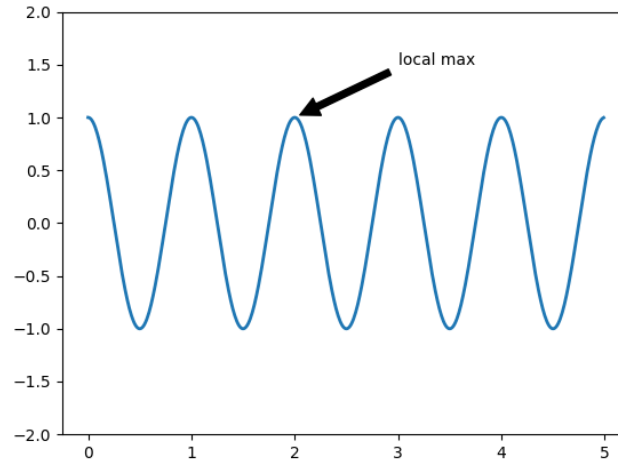


Figure 27 Annotating text in a plot

Additional Examples:

In the following example, the first call to `plt.plot` creates the axes, then subsequent calls to `plt.plot` add additional lines on the same axes, and `plt.xlabel`, `plt.ylabel`, `plt.title` and `plt.legend` set the axes labels and title and add a legend.

```
x = np.linspace(0, 2, 100)  
  
plt.plot(x, x, label='linear')  
plt.plot(x, x**2, label='quadratic')  
plt.plot(x, x**3, label='cubic')  
  
plt.xlabel('x label')  
plt.ylabel('y label')  
  
plt.title("Simple Plot")  
plt.legend()  
plt.show()
```

The output of the plot will look like the figure 28.

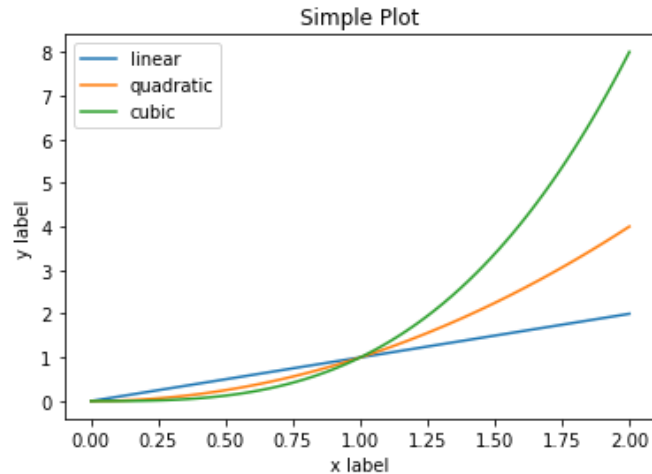


Figure 28 A Simple Plot with legends

15 Recommended Studies

- Python – Object Oriented Programming
 - https://www.tutorialspoint.com/python/python_classes_objects.htm
 - <https://python.swaroopch.com/oop.html>
- Numpy
 - <https://github.com/78526Nasir/Top-5-Machine-Learning-Libraries-in-Python/blob/master/source%20codes/Numpy.ipynb>
- Pandas
 - <https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python>
- Matplotlib
 - <https://matplotlib.org/index.html>