



Functions & Modules

UNIT - IV

Unit-4: Functions & Modules

C04 - Use built-in functions, user-defined functions and modules in a program.

4.1 Introduction to Python User defined Function

4.2 Passing parameters to a function and returning values from a function

4.3 Recursion

4.4 Standard Library: Built-in Functions

4.5 Modules and Packages

- rand module - Random numbers generators
- math module - Mathematical functions
- datetime module - Date and time functions
- matplotlib module - Plotting functions

4.6 Create and import custom user defined module

Python User Defined Functions

- A function is a reusable block of programming statements designed to perform a specific task.
- To define a function, Python provides the **def** keyword.
- The following is the syntax of defining a function.

```
def function_name (<parameters>):  
    statement1  
    statement2  
    ...  
    ...  
    return <expr>
```

Example:

```
def my_function():  
    print("Hello from a function")  
  
my_function()
```

Output: Hello from a function

Passing Parameters to a Function

- It is possible to define a function to receive one or more parameters (also called arguments) and use them for processing inside the function block.

- Parameters/Arguments may be given suitable formal names.

```
def greet(name):  
    print ('Hello ', name)  
greet('Apple') #Calling function with argument  
greet(123) #Calling function with argument
```

- The **greet()** function is now defined to receive a string parameter called name. Inside the function, the **print()** statement is modified to display the greeting message addressed to the received parameter.

Output:

```
Hello Apple  
Hello 123
```

Passing Multiple Parameters to a Function

- A function can have multiple parameters.
- The following function takes three arguments.

```
def greet(name1, name2, name3):  
    print ('Hello ', name1, ', ', name2, ', ', name3)  
greet('Ramesh', 'Suresh', 'Mahesh') # calling function with string argument
```

Output:

Hello Ramesh , Suresh , Mahesh

- We can also have Variable Length Arguments:

```
def greet(*name):  
    print ('Hello ', name[0], ' ', name[1])  
greet('AVPT', 'COMPUTER')
```

Output:

Hello AVPT , COMPUTER

Returning Values from a Function

- A user-defined function can also be made to return a value to the calling environment by putting an expression in front of the return statement.
- Unlike C/C++ it is not mandatory to declare the return method in function declaration.

Syntax:

```
def sum(a, b):  
    return a + b
```

Example:

```
def sum(a,b):  
    return a + b  
  
print(sum(3,5))
```

Output:8

Note: By default, all the functions return None if the return statement does not exist.

Recursion

- **Recursion** is the process of calling the same function from inside a function.
- Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

```
def recursive():  
    . . .  
    recursive()  
    . . .  
recursive()
```

The diagram shows a function definition `def recursive():` followed by three lines of code: `. . .`, `recursive()`, and `. . .`. A yellow arrow originates from the `recursive()` call inside the function and points back to the `def recursive():` line, labeled "Recursive Call". Below the function definition, the text `recursive()` is shown again, with a yellow arrow pointing to the `def recursive():` line, labeled "Function Call".

- Advantages of Recursion:

- A complicated function can be split down into smaller sub-problems utilizing recursion.
- Sequence creation is simpler through recursion than utilizing any nested iteration.
- Recursive functions render the code look simple and effective.

- Disadvantages of Recursion:

- A lot of memory and time is taken through recursive calls which makes it expensive for use.
- Recursive functions are challenging to debug.
- The reasoning behind recursion can sometimes be tough to think through.

Example: Factorial

```
def factorial(x):  
    if x == 1:  
        return 1  
    else:  
        # recursive call to the function  
        return (x * factorial(x-1))  
num = int(input("Enter a number: "))  
result = factorial(num)  
print("The factorial of", num, "is", result)
```

Output:

Enter a number: 5

The factorial of 5 is 120

Standard Library : Built-In Functions

- Python supports large set of standard libraries which provides lots of modules and built in functions.
- **Built-in Functions:**
 - This functions is already defined in standard library.
 - Following are some common used built-in functions
 - **Input/Output Functions**
`Input(), print()`
 - **Data Type Conversion Functions**
`int(), float(), str(), list(), tuple(), set(), dict()`
 - **Mathematical Functions**
`abs(), min(), max(), sum(), pow(), round(), divmod()`

Example:

| | |
|-----------------------------------|---------------------|
| <code>print(max([12,4,5]))</code> | <code>#12</code> |
| <code>print(min([12,4,5]))</code> | <code>#4</code> |
| <code>print(sum([12,4,5]))</code> | <code>#21</code> |
| <code>print(abs(-5))</code> | <code>#5</code> |
| <code>print(divmod(5,2))</code> | <code>#(2,1)</code> |
| <code>print(pow(2,3))</code> | <code>#8</code> |
| <code>print(round(5.6))</code> | <code>#6</code> |
| <code>print(len(1,2,3))</code> | <code>#3</code> |

Modules

- Module is a python file which contains a python code including functions, class or variables.
- Python Module has a .py extension.
- Python Module Provides Flexibility to organize the code in logical way.
- **import** keyword is used to do this.
- Listing of Modules:

```
help("modules")
```
- Modules in Python can be of two types:
 - Built-in Module (Standard Library Module)
 - User-Defined Modules

Example of a Module Working:

File: example.py

```
# Python Module example
```

```
def add(a, b):  
    result = a + b  
    return result
```

```
# Python Module import
```

```
import example
```

```
print(example.add(4,5.5))
```

Output: 9.5

Build-in Modules

- A Module which is already created in standard library of python is known as Built-in Modules.
- Syntax to Use it:

```
Import Module_Name
```

Example:

```
import math
```

- random Module (Random Number Generators)
- math Module (Mathematical Functions)
- datetime Module (Date and Time Functions)
- matplotlib Module (Plotting Functions)

random module

- It contains functions to generate random numbers.

| Method | Description |
|----------------------|---|
| seed() | Initialize the random number generator |
| getstate() | Returns the current internal state of the random number generator |
| setstate() | Restores the internal state of the random number generator |
| getrandbits() | Returns a number representing the random bits |
| randrange() | Returns a random number between the given range |
| randint() | Returns a random number between the given range |
| choice() | Returns a random element from the given sequence |
| choices() | Returns a list with a random selection from the given sequence |
| shuffle() | Takes a sequence and returns the sequence in a random order |
| sample() | Returns a given sample of a sequence |
| random() | Returns a random float number between 0 and 1 |
| uniform() | Returns a random float number between two given parameters |

Example:

```
import random
```

```
print(random.random())
```

```
0.9560342718892494
```

```
print(random.randint(1,100))
```

```
45
```

```
print(random.randrange(35))
```

```
23
```


math module

- It contains functions and constant to perform mathematics related operations

| Method | Description |
|-------------------------------|--|
| <code>math.degrees()</code> | Converts an angle from radians to degrees |
| <code>math.dist()</code> | Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point |
| <code>math.exp()</code> | Returns E raised to the power of x |
| <code>math.factorial()</code> | Returns the factorial of a number |
| <code>math.floor()</code> | Rounds a number down to the nearest integer |
| <code>math.fmod()</code> | Returns the remainder of x/y |
| <code>math.log()</code> | Returns the natural logarithm of a number, or the logarithm of number to base |
| <code>math.ceil()</code> | Rounds a number up to the nearest integer |
| <code>math.pow()</code> | Returns the value of x to the power of y |
| <code>math.sqrt()</code> | Returns the square root of a number |

| Method | Description |
|---------------------------|--|
| <code>math.acos()</code> | Returns the arc cosine of a number |
| <code>math.acosh()</code> | Returns the inverse hyperbolic cosine of a number |
| <code>math.asin()</code> | Returns the arc sine of a number |
| <code>math.asinh()</code> | Returns the inverse hyperbolic sine of a number |
| <code>math.atan()</code> | Returns the arc tangent of a number in radians |
| <code>math.atan2()</code> | Returns the arc tangent of y/x in radians |
| <code>math.atanh()</code> | Returns the inverse hyperbolic tangent of a number |
| <code>math.cos()</code> | Returns the cosine of a number |
| <code>math.cosh()</code> | Returns the hyperbolic cosine of a number |
| <code>math.tan()</code> | Returns the tangent of a number |
| <code>math.tanh()</code> | Returns the hyperbolic tangent of a number |
| <code>math.sin()</code> | Returns the sine of a number |
| <code>math.sinh()</code> | Returns the hyperbolic sine of a number |

| constants | Description |
|-----------|------------------------------------|
| math.e | Returns Euler's number (2.7182...) |
| math.pi | Returns PI (3.1415...) |

Example:

```
import math
```

| | |
|-------------------------------------|--------------------------|
| <pre>print(math.ceil(2.3))</pre> | <pre>#3</pre> |
| <pre>print(math.floor(2.3))</pre> | <pre>#2</pre> |
| <pre>print(math.exp(2))</pre> | <pre>#7.3890560989</pre> |
| <pre>print(math.fabs(-4))</pre> | <pre>#4</pre> |
| <pre>print(math.factorial(4))</pre> | <pre>#24</pre> |
| <pre>print(math.gcd(12,8))</pre> | <pre>#4</pre> |
| <pre>print(math.pow(2,3))</pre> | <pre>#8</pre> |

datetime module

- Python `datetime` module having class named `datetime`, that provides various functions to deal with date and time.
- In Python, the date is `not a data type`, but we can work with the date objects by importing the `module` named with `datetime`, so using that we can fetch current date and time as well as performs various calculations on date and time.

```
var1 = datetime.date(YYYY, MM, DD)  
# This will convert the numeral date to the date object
```

Example:

```
import datetime  
userdate = datetime.date(2021, 10, 20)  
print('userdate: ', userdate)  
print('type of userdate: ', type(userdate))
```

Output:

```
userdate: 2021-10-20  
type of userdate: <class 'datetime.date'>
```

```
# returns the time with all it's value as 0 like 00:00:00
```

```
var1 = datetime.time()
```

Example:

```
time1 = datetime.time()
```

```
00:00:00
```

```
# returns the time with the value which are specified by the user in  
the attributes
```

```
var2 = datetime.time(hour=?, minute=?, second=?, microsecond=?)
```

Example:

```
time2 = datetime.time(hour=12, minute=55, second=50)
```

```
12:55:50
```

```
var1 = datetime.datetime(YYYY, MM, DD, hr, min, s, ms)
# This will convert the numeral date to the datetime object in the
form of YYYY-MM-DD hr:min:s:ms.
```

Example:

```
import datetime
userdatetime = datetime.datetime(2021, 9, 15, 20, 55, 20, 562789)
userdatetime: 2021-09-15 20:55:20.562789
```


- We can create date objects from timestamps by using the `fromtimestamp()` method.
- The timestamp is the number of seconds from 1st January 1970 at UTC to a particular date.

Getting Datetime from timestamp

```
date_time = datetime.fromtimestamp(1887639468)
print("Datetime from timestamp:", date_time)
```

Output:

```
Datetime from timestamp: 2029-10-25 16:17:48
```

- We can convert date object to a string representation using two functions `isoformat()` and `strftime()`.

```
# function of date class
```

```
today = date.today()
```

```
# Converting the date to the string
```

```
Str = date.isoformat(today)
```

```
print("String Representation", Str)
```

```
print(type(Str))
```

Output:

```
String Representation 2021-08-19
```

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

| Function Name | Description |
|--------------------------------|---|
| <code>ctime()</code> | Return a string representing the date |
| <code>fromisocalendar()</code> | Returns a date corresponding to the ISO calendar |
| <code>fromisoformat()</code> | Returns a date object from the string representation of the date |
| <code>fromordinal()</code> | Returns a date object from the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 |
| <code>fromtimestamp()</code> | Returns a date object from the POSIX timestamp |
| <code>isocalendar()</code> | Returns a tuple year, week, and weekday |
| <code>isoformat()</code> | Returns the string representation of the date |
| <code>isoweekday()</code> | Returns the day of the week as integer where Monday is 1 and Sunday is 7 |
| <code>replace()</code> | Changes the value of the date object with the given parameter |
| <code>strftime()</code> | Returns a string representation of the date with the given format |
| <code>timetuple()</code> | Returns an object of type <code>time.struct_time</code> |
| <code>today()</code> | Returns the current local date |
| <code>toordinal()</code> | Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1 |
| <code>weekday()</code> | Returns the day of the week as integer where Monday is 0 and Sunday is 6 |

matplotlib module

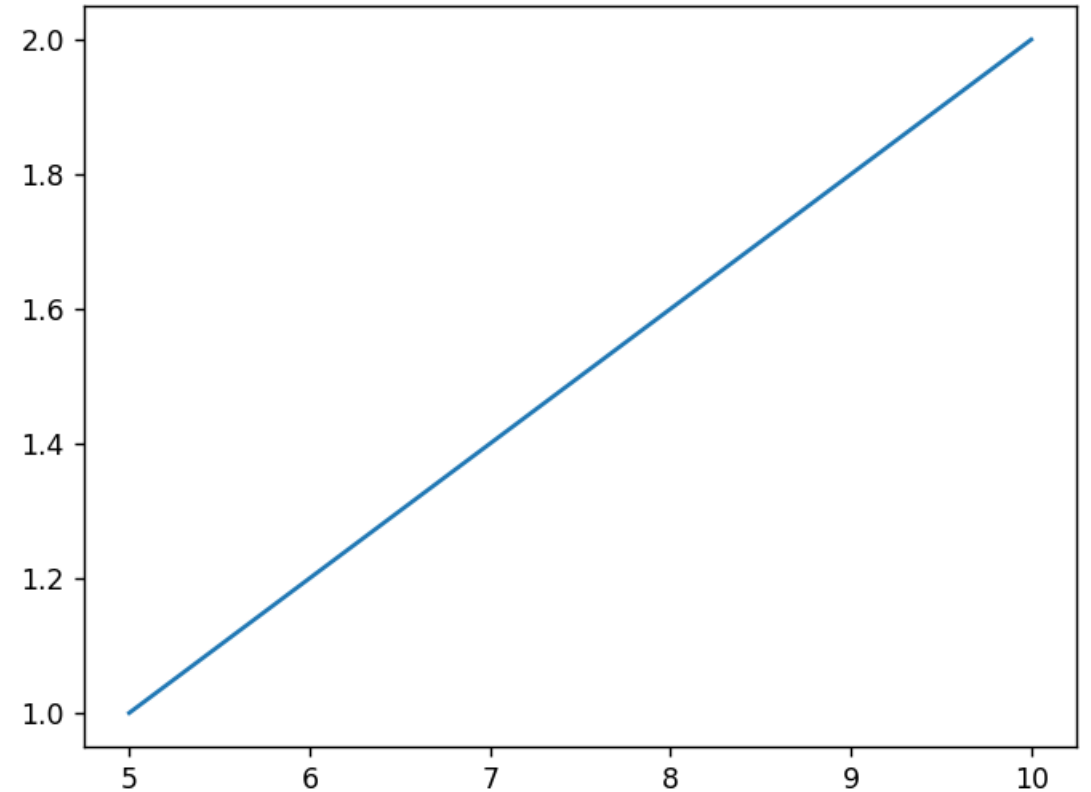
- matplotlib is a plotting library for Python, having a sub module `pyplot`, that contains various built in graph plotting functions.
- Along with matplotlib `numpy` module is also used to represent array of points for graph.
- Conventionally, the package is imported into the Python script by adding the following statement:

```
from matplotlib import pyplot as plt
```
- Here keyword `as` is used to give an alias to matplotlib, now you can call it as just `plt`.

Plotting Line:

```
import numpy as np
from matplotlib import
pyplot as plt

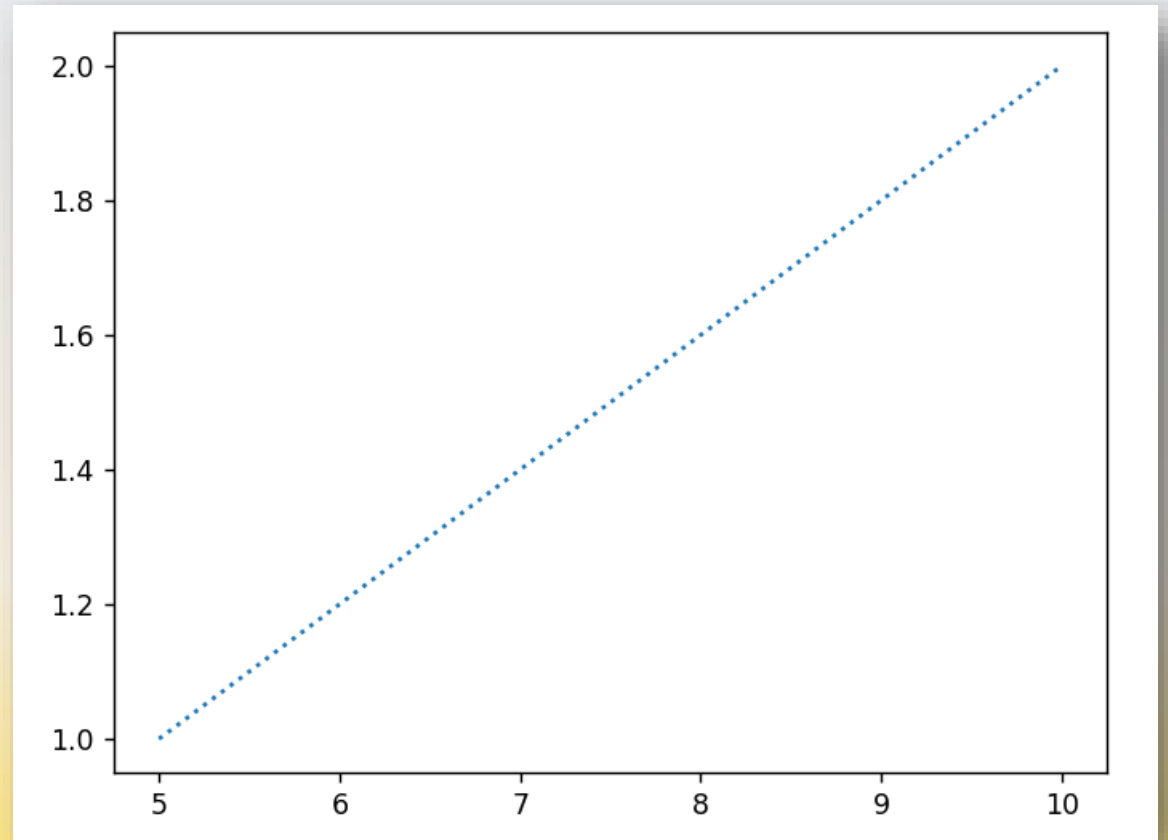
xpoint = np.array([5,10])
ypoint = np.array([1,2])
plt.plot(xpoint,ypoint)
plt.show()
```



Plotting Line with Specific Style:

```
import numpy as np
from matplotlib import
pyplot as plt

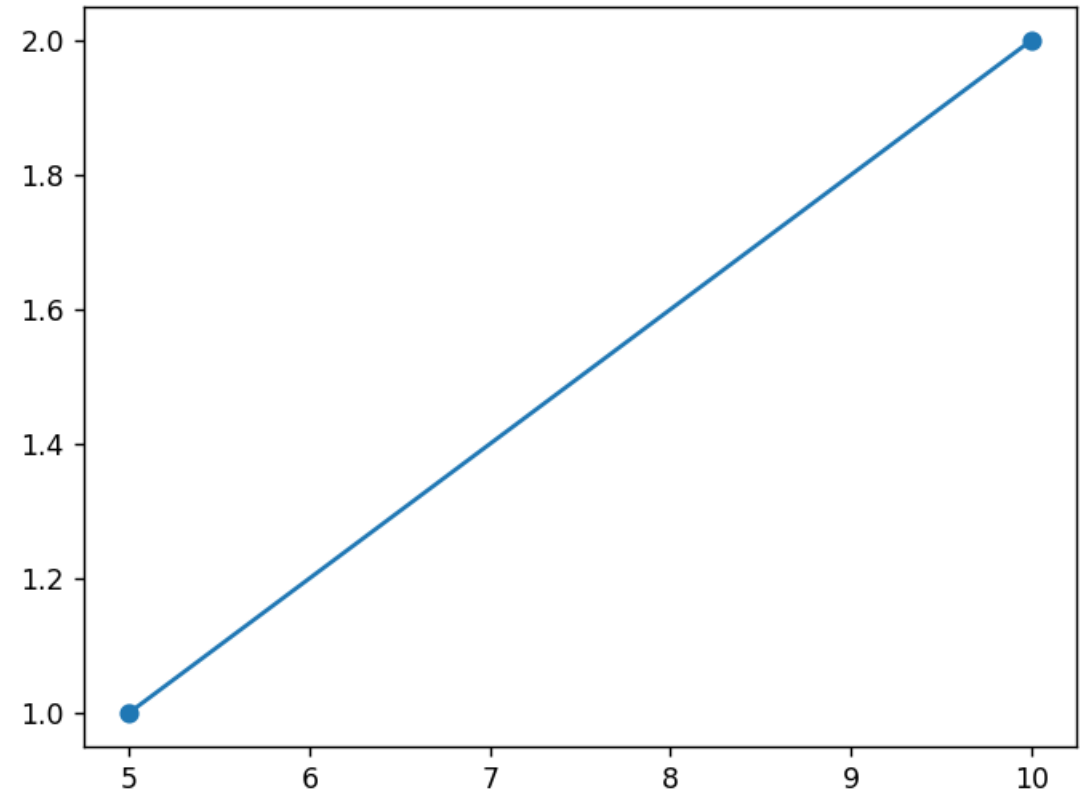
xpoint = np.array([5,10])
ypoint = np.array([1,2])
plt.plot(xpoint,ypoint,line
style='dotted')plt.show()
```



Plotting Line with end marker:

```
import numpy as np
from matplotlib import
pyplot as plt

xpoint = np.array([5,10])
ypoint = np.array([1,2])
plt.plot(xpoint,ypoint,mark
er='o')
plt.show()
```

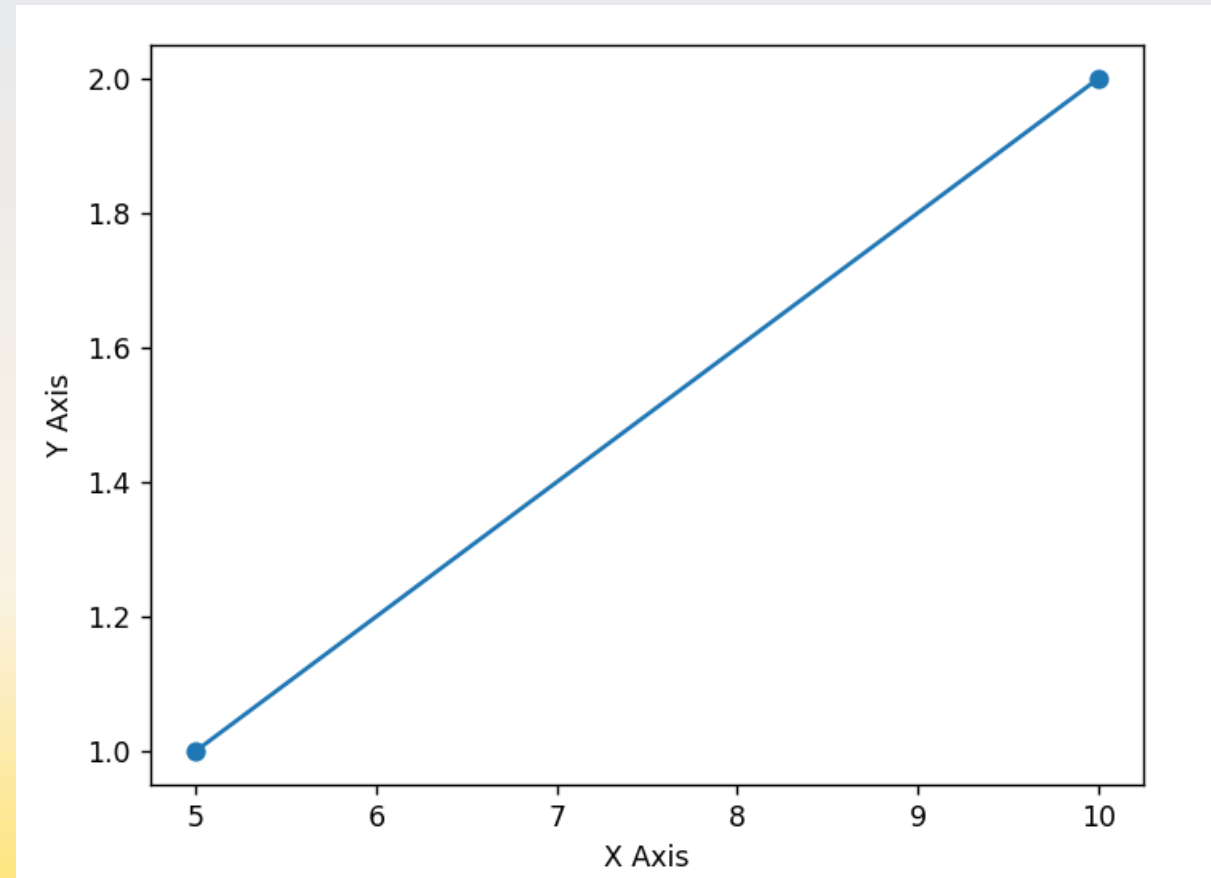


Plotting Line with Label for X axis and Y axis:

```
import numpy as np
from matplotlib import pyplot
as plt

xpoint = np.array([5,10])
ypoint = np.array([1,2])
plt.plot(xpoint,ypoint,marker=
'o')

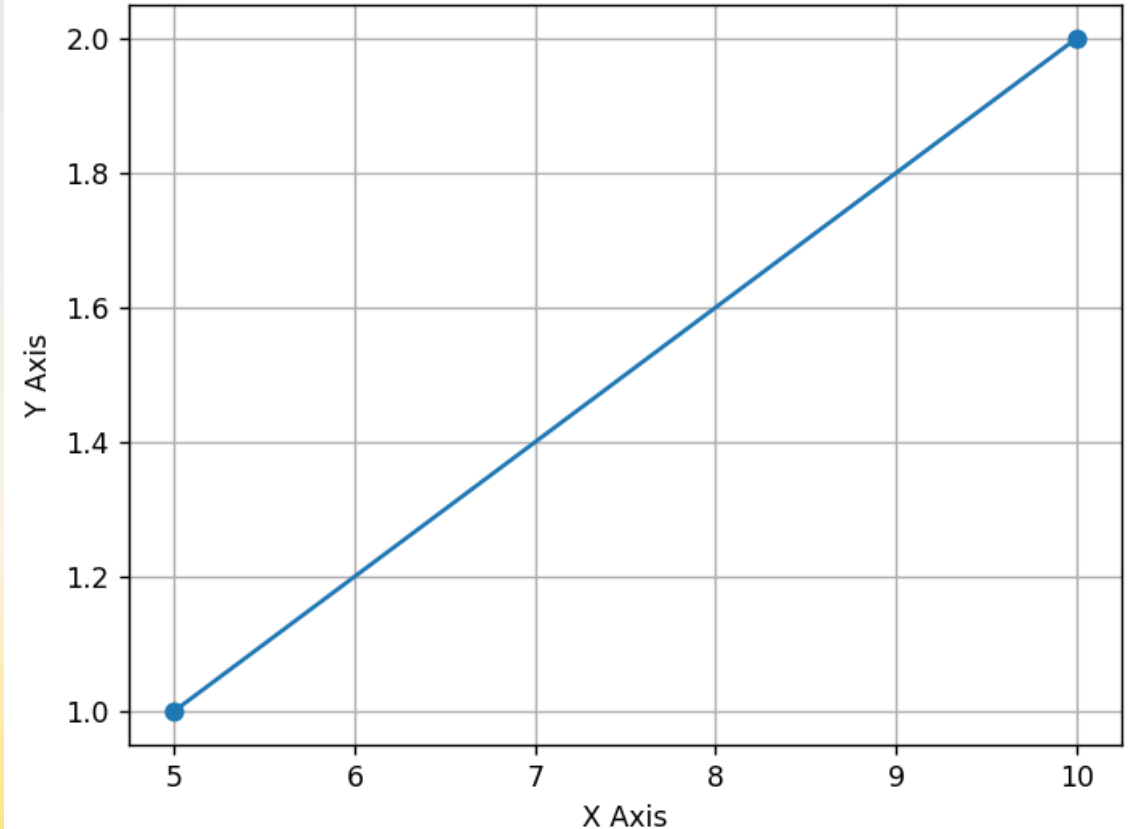
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.show()
```



Plotting Line with Grid:

```
import numpy as np
from matplotlib import pyplot
as plt

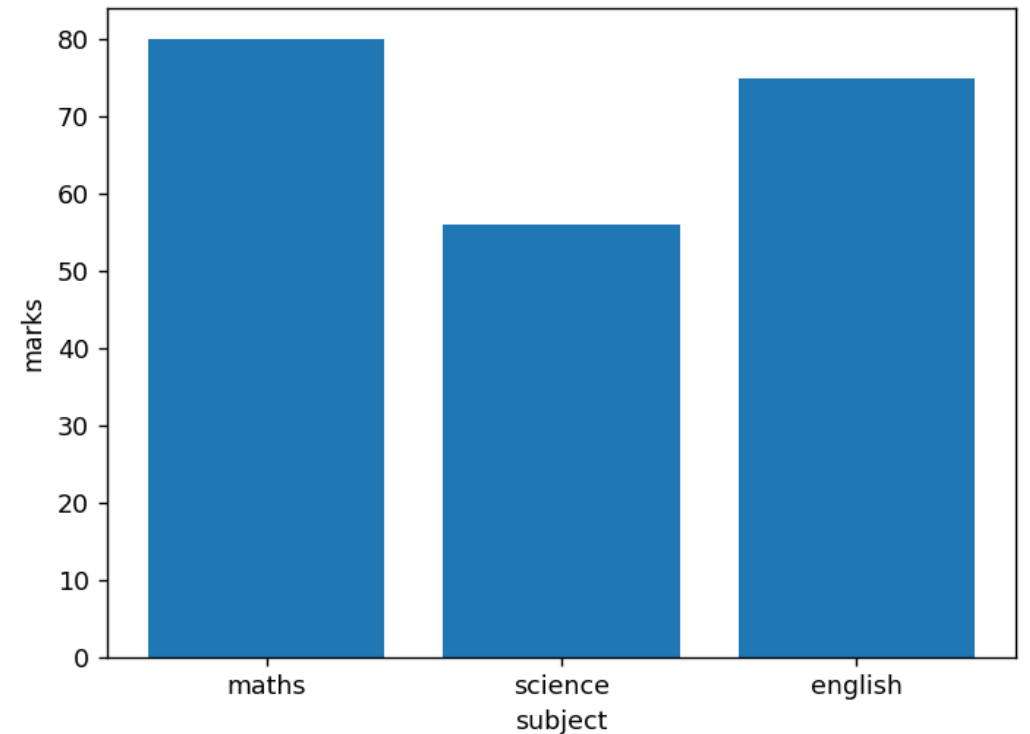
xpoint = np.array([5,10])
ypoint = np.array([1,2])
plt.plot(xpoint,ypoint,marker=
'o')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.grid()
plt.show()
```



Plotting Bar Chart:

```
import numpy as np
from matplotlib import pyplot as plt

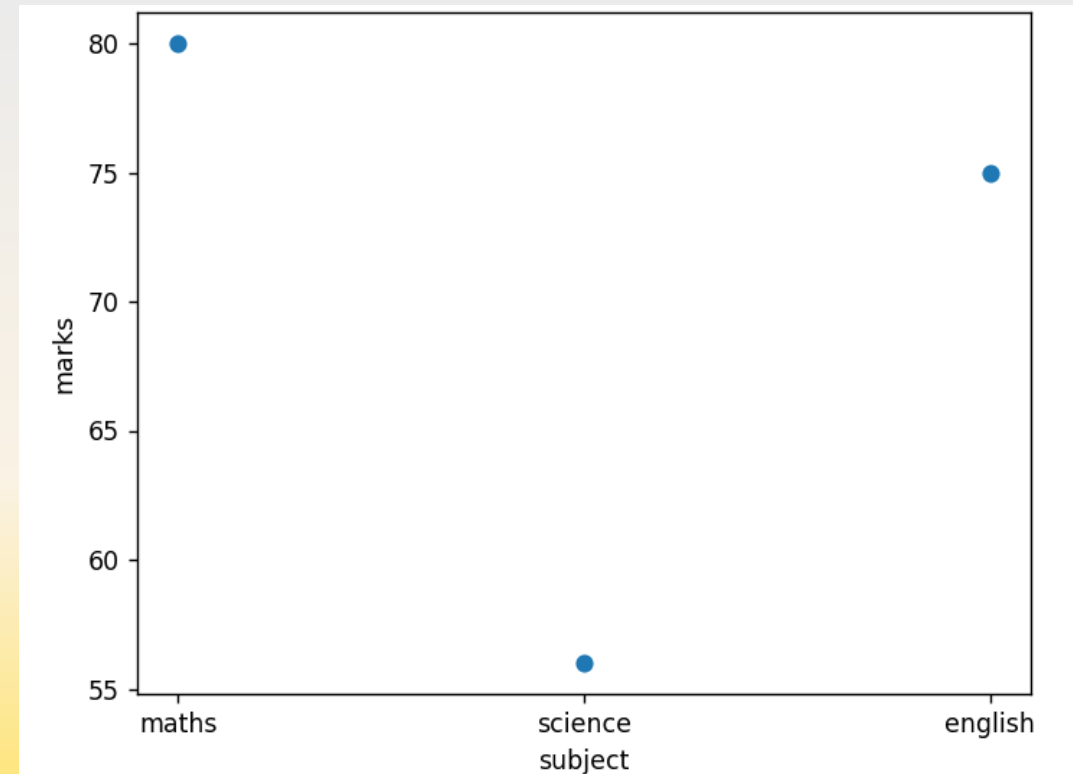
subject =
np.array(["maths", "science", "english"])
marks = np.array([80, 56, 75])
plt.bar(subject, marks)
plt.xlabel("subject")
plt.ylabel("marks")
plt.show()
```



Plotting Scatter Chart:

```
import numpy as np
from matplotlib import pyplot as plt

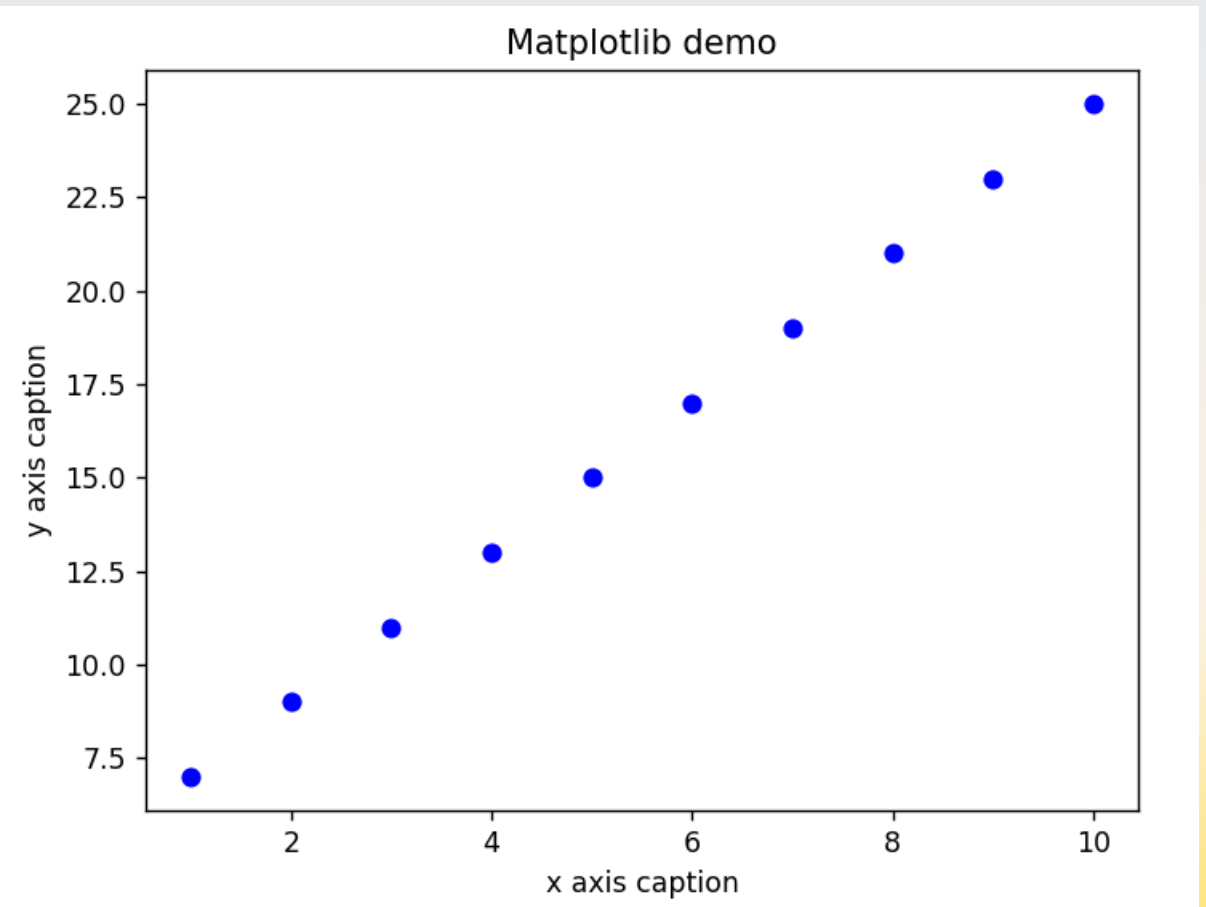
subject =
np.array(["maths", "science", "english"])
marks = np.array([80, 56, 75])
plt.scatter(subject, marks)
plt.xlabel("subject")
plt.ylabel("marks")
plt.show()
```



Plotting Line with Given Equation:

```
import numpy as np
from matplotlib import
pyplot as plt

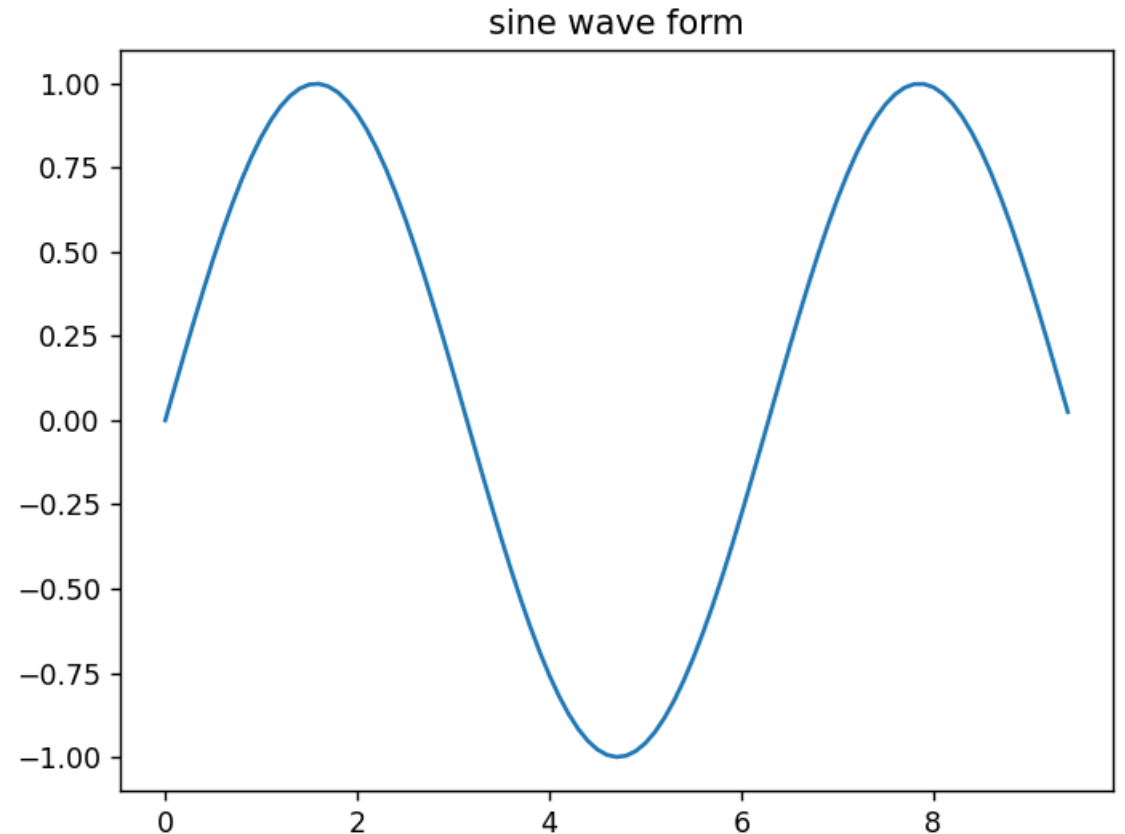
x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x, y, "ob")
plt.show()
```



Plotting Sinusoidal waves:

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y
coordinates for points on a
sine curve
x = np.arange(0, 3 * np.pi,
0.1)
y = np.sin(x)
plt.title("sine wave form")
plt.plot(x, y)
plt.show()
```



subplot() Function

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Compute the x and y coordinates for points on sine and cosine
curves
```

```
x = np.arange(0, 3 * np.pi, 0.1)
```

```
y_sin = np.sin(x)
```

```
y_cos = np.cos(x)
```

```
# Set up a subplot grid that has height 2 and width 1, and
first such subplot as active.
```

```
plt.subplot(2, 1, 1)
```

```
# Make the first plot
```

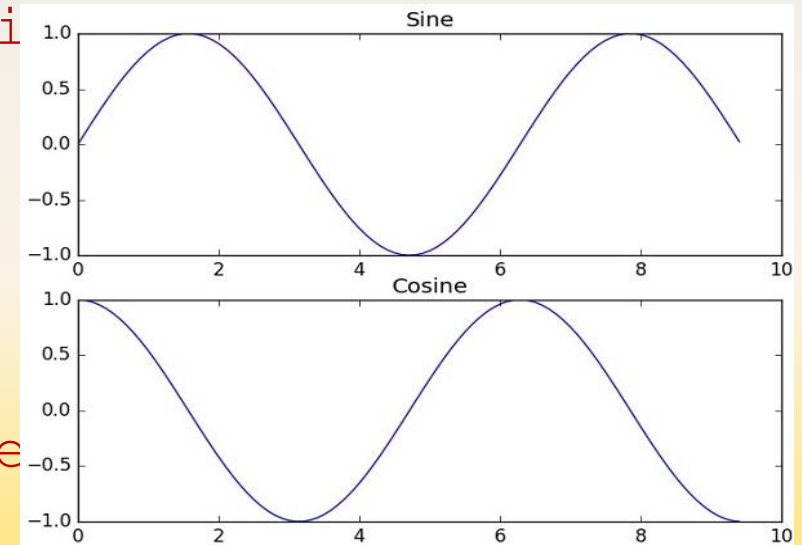
```
plt.plot(x, y_sin)
```

```
plt.title('Sine')
```

```
# Set the second subplot as active, and make the second plot
```

```
plt.subplot(2, 1, 2)
```

```
plt.plot(x, y_cos)
```



Create user defined module

- Module which is created by user is known as User Defined Module.
- Module can be created by `.py` file.
- You can define various functions, classes and variables in this file.

Example:

Step1: Create file Named `myModule.py`

Step2: Define Variable in file

```
student = {"Name": "Yagnik"}
```

Step3: Define Functions to display value of variable

```
def DisplayStudent():  
    print("Name:" + student["Name"])
```

Step4: Save file

Import user defined module

- Once Module is Created you can use that module in another python file by Importing it by using import statement.

```
import myModule  
myModule.DisplayStudent()
```

Output:

Name:Yagnik

Renaming user defined module

- You can rename module at the time of importing it using `as` keyword.

```
import myModule as mm  
mm.DisplayStudent()
```

Output:

```
Name:Yagnik
```

Packages

- A Python Package usually consists of several modules.
- Physically, a Package is a folder containing modules and maybe other folders that themselves may contain more folders and modules (nested in the form of a hierarchy).
- Using the concept of package we can organize a large python application in a structured way where each package contains modules of relevant type.
- This simply means that a package's modules are bound together by a package name, by which they may be referenced.
- Syntax for import packages:

```
import <package_name>
```

Example:

```
import math
```

```
print(math.factorial(3))          #6
```

```
print(math.log(1))               #0.0
```