



Functions & Modules

UNIT - IV

Unit-4: Functions & Modules

CO4 - Apply modular programming approach to solve given problems using user-defined functions.

- 🐘 Develop programs using Functions
- 🐘 Develop program using Recursion
- 🐘 Use Built-in functions in Programs
- 🐘 Develop programs using rand, math, datetime modules

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 certain task.
- To define a function, Python provides the **def** keyword. The following is the syntax of defining a function.

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

```
def greet():  
    """This function displays  
    'Hello World!'"""  
    print('Hello World!')
```

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)
```

- 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.

```
Hello Apple  
Hello 123
```

Multiple Parameters

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

```
def greet(name1, name2, name3):  
    print ('Hello ', name1, ' , ', name2 , ' , and ', name3)
```

```
greet('Ramesh', 'Suresh', 'Mahesh')  
# calling function with string argument
```

```
Hello Ramesh, Suresh, and Mahesh
```

- We can also have Variable Length Arguments:

```
def greet(*name):  
    print ('Hello ', name[0], ' , ', name[1] , ' , and ', name[2])
```

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.

```
def sum(a, b):  
    return a + b  
  
total = sum(10, 20)  
print(total) #30  
total = sum(5, sum(10, 20))  
print(total) #35
```

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.

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

Recursive Call




Function Call

Certain rules to be followed while using recursion.






Advantages & Disadvantages of Recursion

Advantages:

-  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:

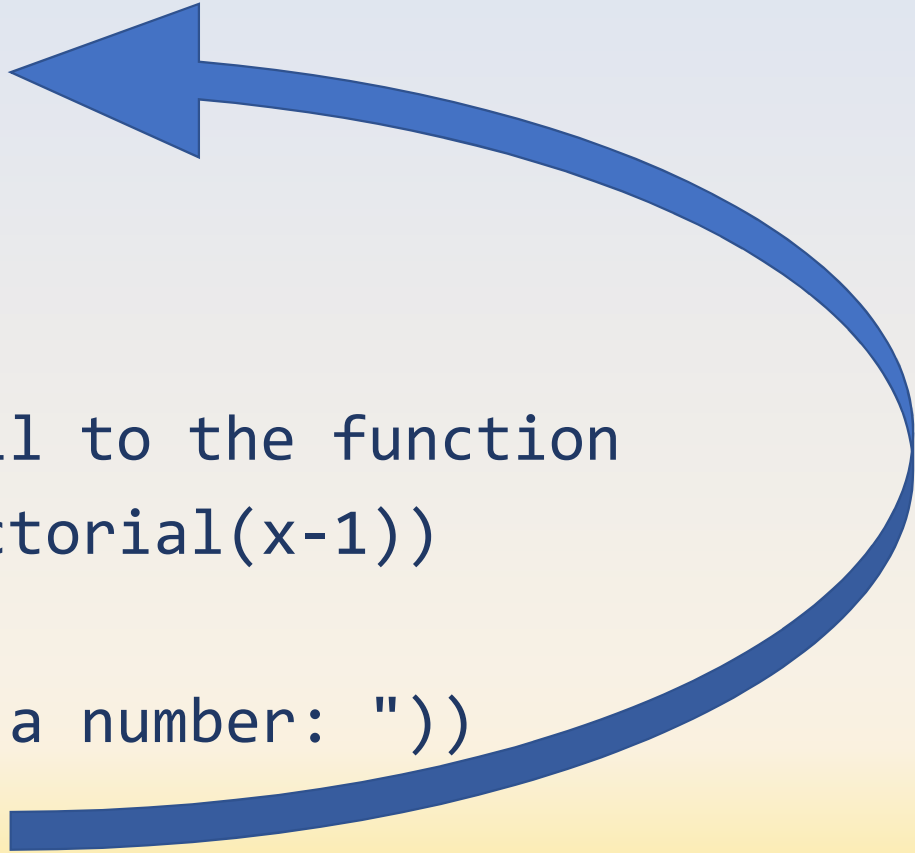
-  A lot of memory and time is taken through recursive calls which makes it expensive for use.
-  Recursive functions are challenging to debug.
-  The reasoning behind recursion can sometimes be tough to think through.

Factorial Number : Recursion Example

```
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)
```

**Recursive
Call**



Standard Library Functions: Built-in

- 🐍 A Python library is collection of modules.
- 🐍 A package is library that can be installed in package manager like numpy, nltk, keras.
- 🐍 Python standard libraries make the programming process simpler and remove the need commonly used commands again and again.
- 🐍 Packages calling/importing is at the beginning of python script.
- 🐍 Package make programming very easy at developer's end.

```
>>>help("modules")
```

Standard Library Functions: Built-in

(1) Input/output functions

 input()

 print()

(2) Data functions type conversion

 int()

 float()

 str()

 complex()

 list()

 tuple()

 set()

 dict()

 bool()

(3) Mathematical functions

 abs()

 divmod()

 min()

 max()

 pow()

 sum()

 round()

Modules

- 🐍 A Python module is a (.py extension) file containing Python definitions and statements. A module can define functions, classes, and variables.
- 🐍 A module can also include runnable code.
- 🐍 Module provides grouping of code. Which provides us with some logical structuring of the program.
- 🐍 **import** keyword to do this.
- 🐍 Modules in Python can be of two types:
 1. Built-in Module
 2. User-defined Modules

Module: Example

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

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).
- 🐍 **Conceptually**, it's a **namespace**.
- 🐍 This simply means that a package's modules are bound together by a package name, by which they may be referenced.
- 🐍 Syntax:

```
import <package_name>

>>> import math
>>> math.factorial(3)
6
>>> from math import factorial
>>> from math import *
>>> math.log(1)
0.0
```

Third Party Package Installation

- 🐍 The official repository for finding and downloading such third-party packages is the Python Package Index, usually referred to simply as **PyPI**.
- 🐍 To install packages from PyPI, use the package installer **pip** from **command shell**.

```
$ pip install gensim
```

- 🐍 **pip** can install Python packages **from any source, not just PyPI**. If you installed Python using Anaconda or Miniconda, you can also use the **conda** command to install Python packages.

```
$ conda install gensim
```

Build-in Modules

 random Module

 math Module

 datetime Module

 matplotlib Module

Module - random

- Python offers random module that can generate psuedo-random numbers between 0 to 1.
- These are pseudo-random number as the sequence of number generated depends on the seed.

```
>>> import random
>>> random.seed(2)
>>> print(random.random())
0.9560342718892494
>>> print(random.random())
0.9478274870593494
>>> print(random.random())
0.05655136772680869
```

Module - random

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
triangular()	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters

Module - math

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

Module - math

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

Module math Example

```
>>>import math
>>>print(math.acos(0.55))
>>>0.9884320889261531
>>>print (math.degrees(8.90))
>>>509.9324376664327
>>>print(math.factorial(9))
>>>362880
>>>print(math.pow(9, 3))
729.0
```

Module – math Constants

Constant	Description
<u>math.e</u>	Returns Euler's number (2.7182...)
<u>math.inf</u>	Returns a floating-point positive infinity
<u>math.nan</u>	Returns a floating-point NaN (Not a Number) value
<u>math.pi</u>	Returns PI (3.1415...)
<u>math.tau</u>	Returns tau (6.2831...)

Module - datetime

- Python provides the **datetime module** work with real dates and times. In real-world applications, we need to work with the date and time. Python enables us to **schedule our Python script to run at a particular 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, time, and calendar**.

```
import datetime
var1 = datetime.date(year, month, day)
#This will convert the numeral date to the date object
```

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

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

Module – datetime : Example 1 :

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

```
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'>
```


Module – datetime : Example 2 :

```
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.
```

```
import datetime  
userdatetime = datetime.datetime(2019, 5, 10, 17, 30, 20,  
154236)
```

Output

```
userdatetime: 2019-05-10 14:30:20.154236
```

Module – datetime : Example 3 :

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

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

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

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

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

```
=> 00:00:00
```

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

```
=>12:55:50
```

Module – datetime : Example 4 :

```
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.

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

datetime Example 5

🐍 `today = date.today()`

🔗 To return the current local date `today()` function of `date` class is used.

🐍 `today.year => Current Year`

🐍 `today.month => Current Month`

🐍 `today.day => Current Day`

datetime Example 6

🐍 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
```

Convert Date to String

🐘 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'>
```

Other Functions

Function Name	Description
ctime()	Return a string representing the date
fromisocalendar()	Returns a date corresponding to the ISO calendar
fromisoformat()	Returns a date object from the string representation of the date
fromordinal()	Returns a date object from the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1
fromtimestamp()	Returns a date object from the POSIX timestamp
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Returns the string representation of the date

Other Functions

Function Name	Description
isoweekday()	Returns the day of the week as integer where Monday is 1 and Sunday is 7
replace()	Changes the value of the date object with the given parameter
strftime()	Returns a string representation of the date with the given format
timetuple()	Returns an object of type time.struct_time
today()	Returns the current local date
toordinal()	Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1
weekday()	Returns the day of the week as integer where Monday is 0 and Sunday is 6

Time Object Representation

```
from datetime import time #calling the constructor
my_time = time(13, 24, 56)
```

```
print("Entered time", my_time)
# calling constructor with 1 argument
my_time = time(minute=12)
print("\nTime with one argument", my_time)
```

```
# Calling constructor with 0 argument
my_time = time()
print("\nTime without argument", my_time)
```

Output

Entered time 13:24:56

Time with one argument
00:12:00

Time without argument
00:00:00

Module - matplotlib

- matplotlib is a plotting library for Python.
- It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab.
- 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.

pyplot()

Here **pyplot()** is the most important function in matplotlib library, which is used to plot 2D data.

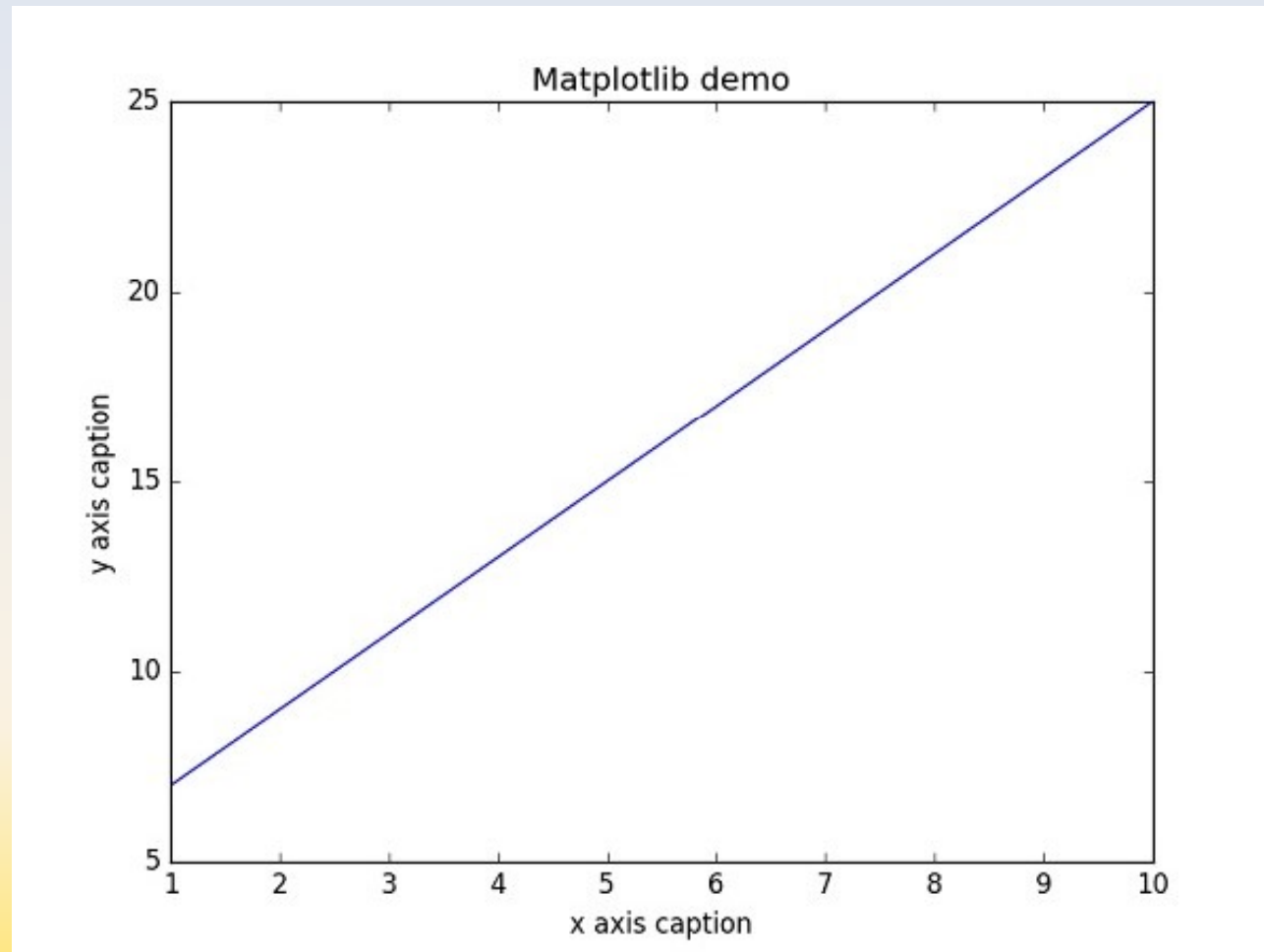
The following script plots the equation $y = 2x + 5$

```
>>> 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)
>>> plt.show()
```

- An ndarray object **x** is created from **np.arange()** function as the values on the **x axis**.
- The corresponding values on the **y axis** are stored in another **ndarray object y**.
- These values are plotted using **plot()** function of pyplot submodule of matplotlib package.
- The graphical representation is displayed by **show()** function.

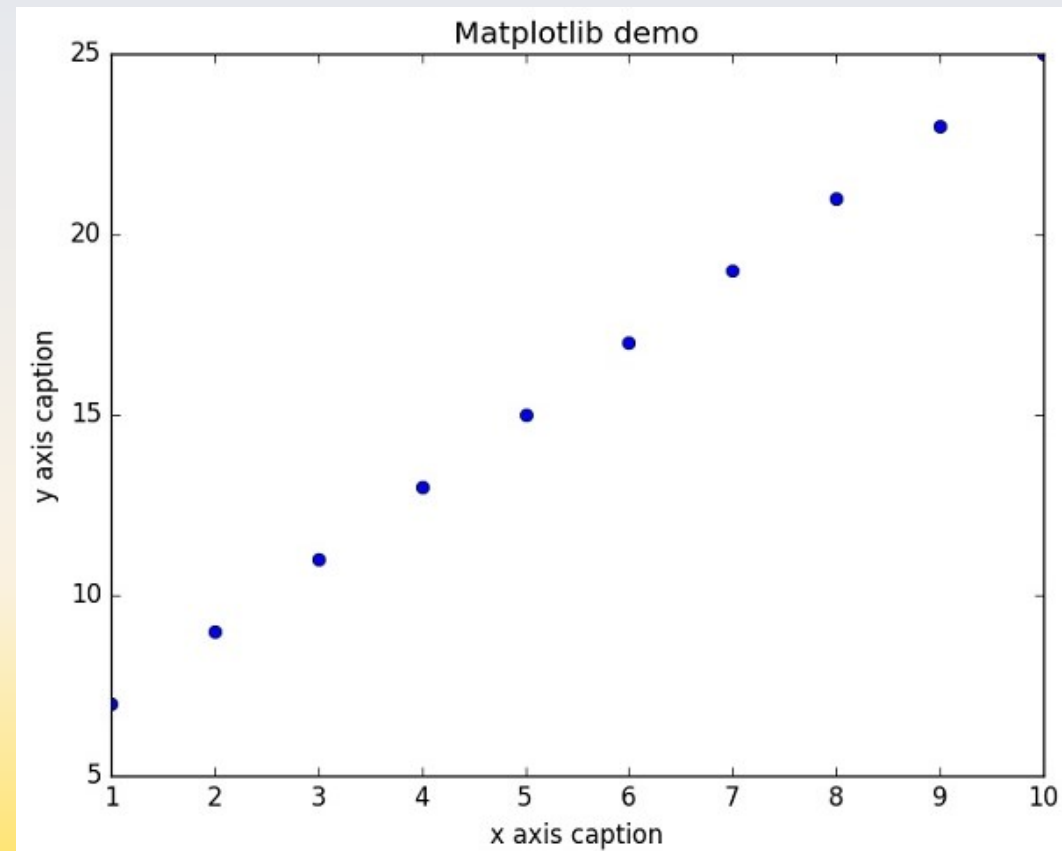
Output



Example - 2

```
>>> 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()
```



Plot 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)
```

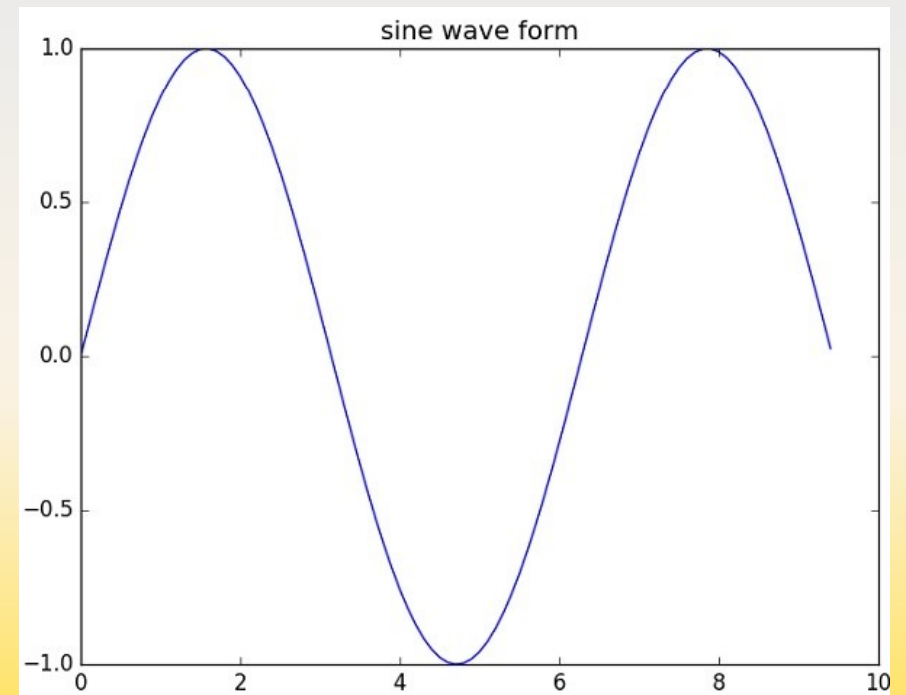
```
y = np.sin(x)
```

```
plt.title("sine wave form")
```

```
# Plot the points using matplotlib
```

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

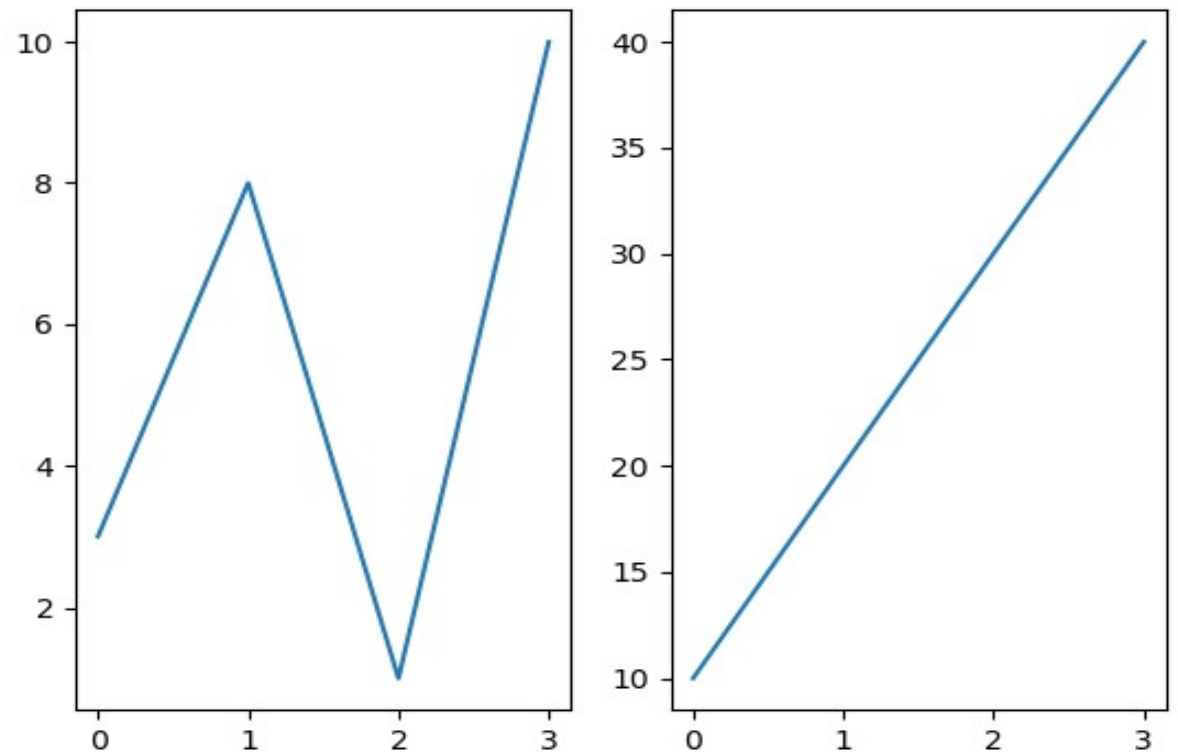
```
plt.show()
```



subplot() function

```
import numpy as np
import matplotlib.pyplot as plt
# Compute the x and y coordinates
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)

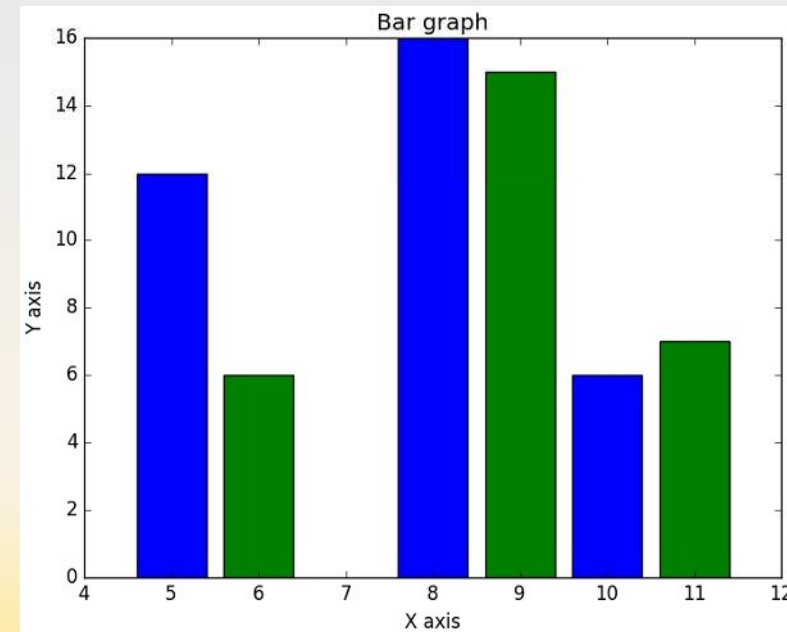
plt.show()
```



Plotting Bar charts – bar()

✚ The **pyplot submodule** provides **bar()** function to generate bar graphs. The following example produces the bar graph of two sets of **x** and **y** arrays.

```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]
x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center')
plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

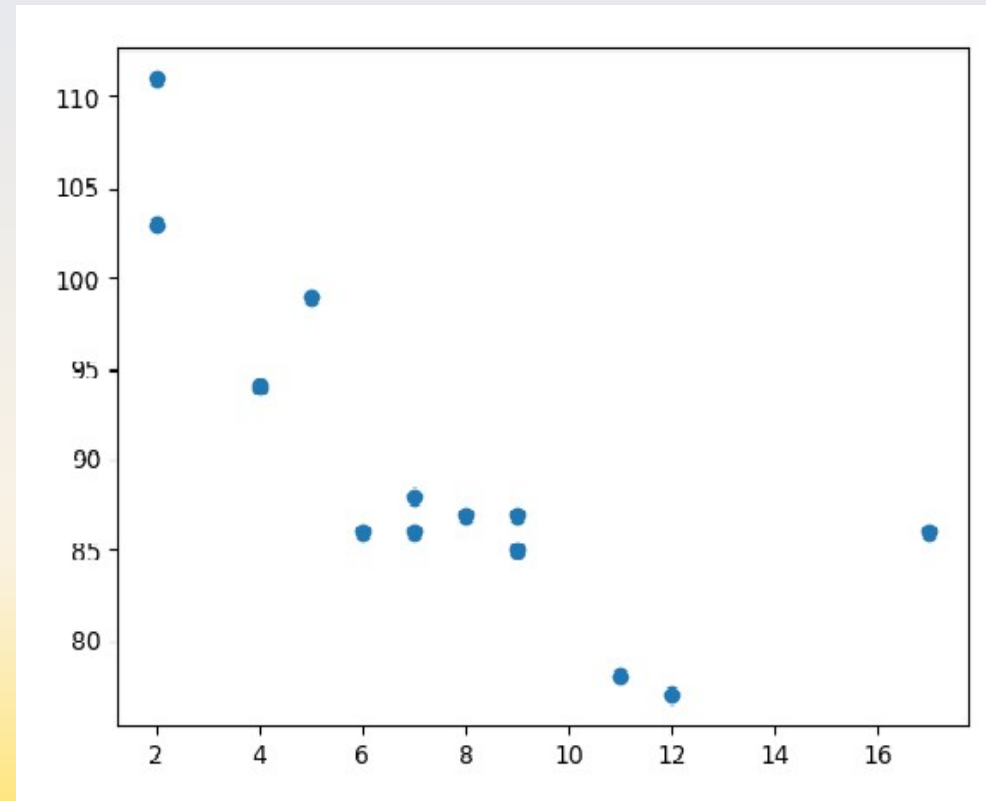


Plotting scatter charts – scatter()

✚ The **pyplot submodule** provides **scatter()** function to generate scatter graphs. The following example produces the bar graph of two sets of **x** and **y** arrays.

```
from matplotlib import pyplot as plt  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
plt.scatter(x, y)  
plt.show()
```



Plotting pie charts – pie()

✚ The **pyplot submodule** provides **pie()** function to generate pie graphs. The following example produces the pie graph of list **y**. Labels are given in list **mylabels** which is optional.

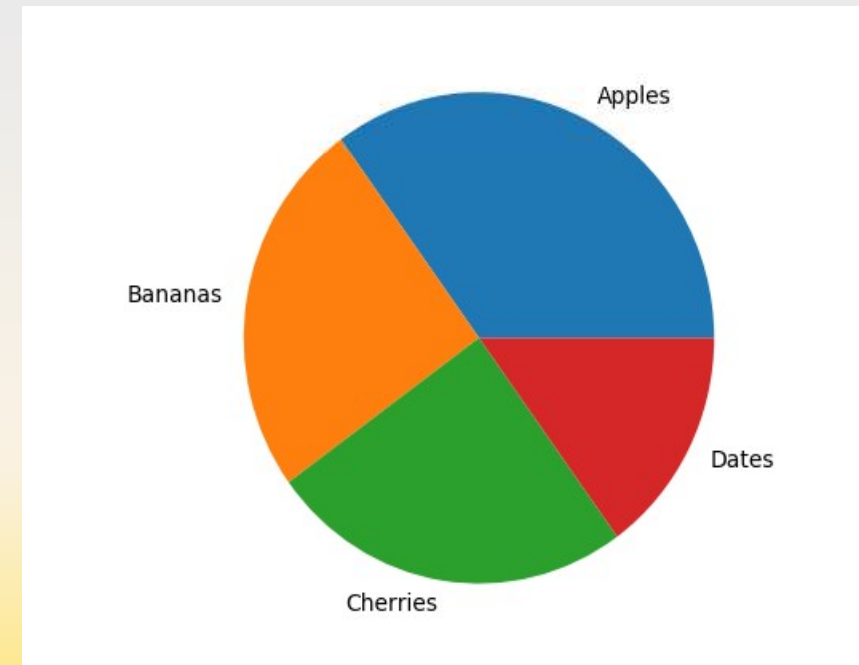
```
from matplotlib import pyplot as plt
```

```
y = [35, 25, 25, 15]
```

```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
```

```
plt.pie(y, labels = mylabels)
```

```
plt.show()
```



Plotting histogram – hist()

✚ The **pyplot submodule** provides **hist()** function to create histograms. The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

```
from matplotlib import pyplot as plt
```

```
# data to display on plots
```

```
x = [1, 2, 3, 4, 5, 6, 7, 4]
```

```
# This will plot a simple histogram
```

```
plt.hist(x, bins = [1, 2, 3, 4, 5, 6, 7])
```

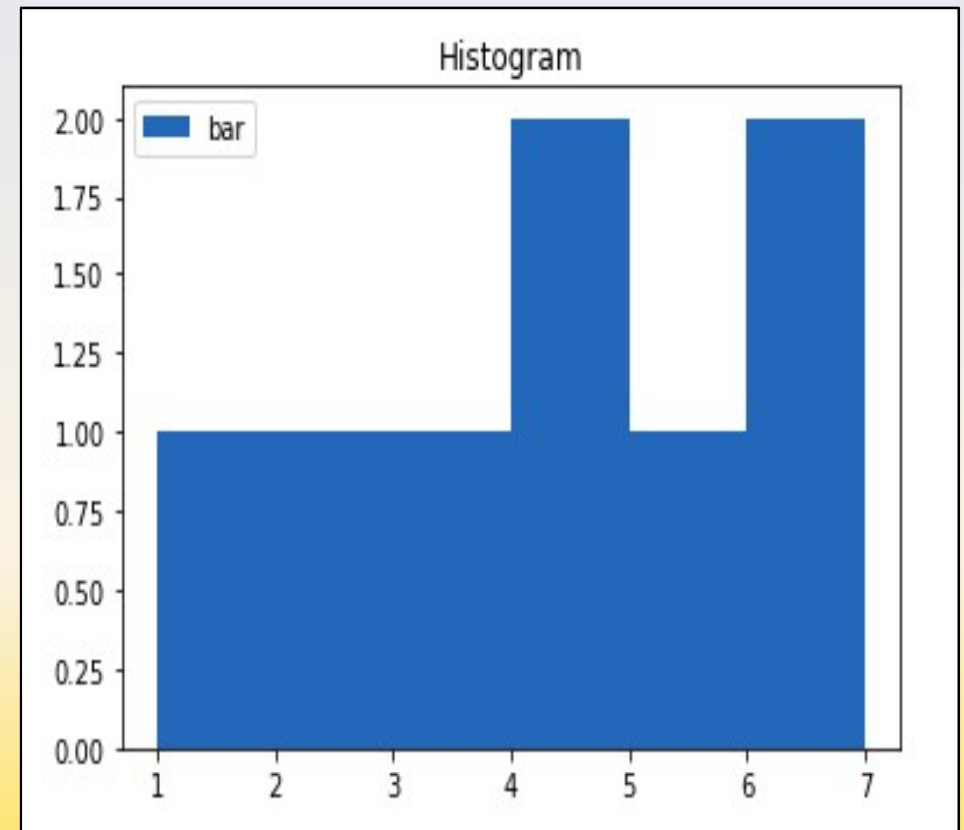
```
# Title to the plot
```

```
plt.title("Histogram")
```

```
# Adding the legends
```

```
plt.legend(["bar"])
```

```
plt.show()
```



Create and import custom user defined module

- 🐍 `__init__.py` helps the Python interpreter to recognize the folder as package.
- 🐍 We must create a file named `__init__.py` (*could be an empty file*) in a directory to make it a package in python.
- 🐍 It also specifies the resources to be imported from the modules.
- 🐍 If the `__init__.py` is empty this means that all the functions of the modules will be imported.
- 🐍 We can also specify the functions from each module to be made available

Create and import custom user defined module

- 🐍 Programmers can import the entire code and definition of any **previously-written user-defined** module using the import statement.
- 🐍 To perform this, programmers have to use the import keyword followed by the module name.
- 🐍 Programmers can also import multiple modules at a time using the comma punctuator.
import module_name
import module1, module2,, module

Renaming Modules - Alias

- Programmers can give any name to the module file, but the module should have the file extension .py.
- To rename a module, programmers need to use the alias while importing a module.
- The 'as' keyword is used for renaming the module with the aliasing concept.

```
import newmodule as mn  
sal = mn.data["salary"]  
print(sal)
```

Example

newmodule.py

```
import numpy as np
```

```
def print_text():  
    print("This message is from an external  
module")
```

```
def find_log(num):  
    return np.log(num)
```

```
def find_exp(num):  
    return np.exp(num)
```

program1.py

```
import newmodule
```

```
newmodule.print_text()
```

```
print("This message is from the  
main file")
```

```
log16 = newmodule.find_log(16)  
print(log16)
```

```
exp16 = newmodule.find_exp(16)  
print(exp16)
```