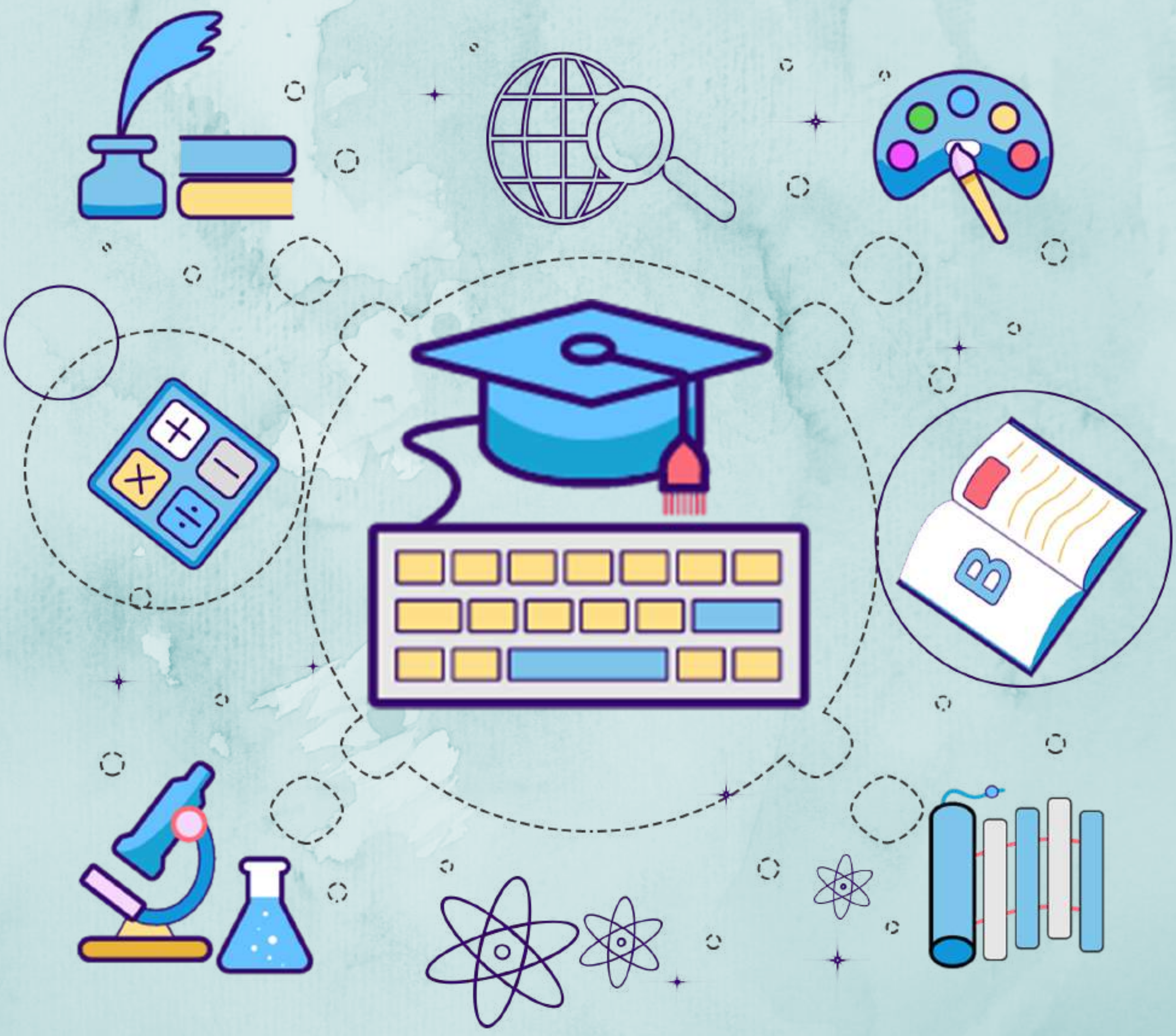


KTU STUDY MATERIALS

PROGRAMMING IN PYTHON

CST 362

Module 5

MODULE V

The os and sys modules, NumPy - Basics, Creating arrays, Arithmetic, Slicing, Matrix Operations, Random numbers. Plotting and visualization. Matplotlib - Basic plot, Ticks, Labels, and Legends. Working with CSV files. – Pandas - Reading, Manipulating, and Processing Data. Introduction to Micro services using Flask.

OS Module in Python

- The OS module in python provides functions for interacting with the operating system.
- OS, comes under Python's standard utility modules.
- This module provides a portable way of using operating system dependent functionality.

Following are some functions in OS module:

1. **os.name**

This function gives the name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'os2', 'ce', 'java' and 'riscos'

```
>>>import os
>>>os.name
'nt'
```

2. **os.getcwd()**

Function os.getcwd(), returns the Current Working Directory(CWD) of the file used to execute the code, can vary from system to system.

```
>>> os.getcwd()

'C:\\Users\\binuvp\\AppData\\Local\\Programs\\Python\\Python38-32'
```

3. os.listdir('.')

To print files and directories in the current directory on your system

```
>>> os.listdir('.')

['are.py', 'DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'mymodule.py', 'NEWS.txt', 'polygon.py', 'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe', 's.py', 'Scripts', 'student.py', 't.py', 'tcl', 'test.py', 'Tools', 'vcruntime140.dll', '__pycache__']
```

4. os.chdir('..')

This function is used to change the CWD

```
>>> os.getcwd()
'C:\\Users\\binuvp\\AppData\\Local\\Programs\\Python\\Python38-32'
>>> os.chdir('..')
>>> os.getcwd()
'C:\\Users\\binuvp\\AppData\\Local\\Programs\\Python'
```

5. os.mkdir(path)

This will create a test directory in C drive

6. **os.rmdir(path)**

Remove the directory temp

7. **os.remove(path)**

Remove a file

8. **os.rename(old.new)**

Renames the file or directory named old to new

Sys Module in Python

- The sys module provides functions and variables used to manipulate different parts of the Python runtime environment.

1. **sys.argv**

   Returns a list of command line arguments passed to a Python script. The item at index 0 in this list is always the name of the script. The rest of the arguments are stored at the subsequent indices.

2. **sys.exit**

   This causes the script to exit back to either the Python console or the command prompt. This is generally used to safely exit from the program in case of generation of an exception.

3. **sys.maxsize**

   Returns the largest integer a variable can take.

4. **sys.path**

   This is an environment variable that is a search path for all Python modules.

5. **sys.version**

- This attribute displays a string containing the version number of the current Python interpreter.

NumPy (Numerical Python)

- NumPy is a **library** consisting of **multidimensional array** objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Using NumPy, a developer can perform the following operations –

- ✓ Mathematical and logical operations on arrays.
- ✓ Fourier transforms and routines for shape manipulation.
- ✓ Operations related to linear algebra.
- ✓ NumPy has in-built functions for linea algebra and random number generation.

ndarray Object

- The most important object defined in NumPy is an **N-dimensional array type** called **ndarray.** It describes the collection of items of the same type. Items in the collection can be accessed using a **zero-based index**.
- Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of **data-type** object (called dtype).
- Any item extracted from ndarray object (by slicing) is represented by a **Python object** of one of **array scalar type**.
- The basic ndarray is created using an array function in NumPy as follows − numpy.array

Creating Arrays

Output:

```
import numpy as np
a = np.array([1,2,3,4])
```

print(a) → [1 2 3 4]

b = np.array([(1,2,3),(4,5,6)], dtype = float)

print(b) → [[1.2.3]
[4.5.6]]

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

print(c) → [[1 2 3]
[4 5 6]
[7 8 9]]

ndarray Object – Parameters

Some important attributes of ndarray object

(1) **ndarray.ndim**

ndim represents the number of dimensions (axes) of the ndarray.

(2) **ndarray.shape**

shape is a tuple of integers representing the size of the ndarray in each dimension.

(3) **ndarray.size**

size is the total number of elements in the ndarray. It is equal to the product of elements of the shape.

(4) **ndarray.dtype**

dtype tells the data type of the elements of a NumPy array. In NumPy array, all the elements have the same data type.

(5) **ndarray.itemsize**

itemsize returns the size (in bytes) of each element of a NumPy array.

Example:

```
import numpy as np
a = np.array([[[1,2,3],[4,3,5]],[[3,6,7],[2,1,0]]])
print("The dimension of array a is:", a.ndim)
print("The size of the array a is: ", a.shape)
print("The total no: of elements in array a is: ", a.size)
print("The datatype of elements in array a is: ", a.dtype)
print("The size of each element in array a is: ", a.itemsize)
```

Output:

```
The dimension of array a is: 3
The size of the array a is: (2, 2, 3)
The total no: of elements in array a is: 12
The datatype of elements in array a is: int32
The size of each element  in array a is: 4
```

Indexing and slicing

- One-dimensional arrays can be indexed, sliced and iterated over, much like lists and other Python sequences.

```
import numpy as np
A=np.arange(10)
print(A)
>>[0 1 2 3 4 5 6 7 8 9]
print(A[0])
>>0
```

```
print(A[-1])
>>9
print(A[0:3])
>>[0 1 2]
A[0:3]=100
A[3]=200
print(A)
>>[100 100 100 200 4 5 6 7 8 9]
```

When we assign a scalar value to a slice, as in A[0:3] = 100, the value is propagated (or broadcasted henceforth) to the entire selection. An important first distinction from lists is that array slices are views on the original array. This means that the data is not copied, and any modifications to the view will be reflected in the source array:

```
slice=A[5:9]
print(slice)
>>[5 6 7 8]

slice[:]=200
print(A)
>>[100 100 100 3 4 200 200 200 200 9]

B=np.arange(10)
print(B[0:8:2])
>>[0 2 4 6]
print(B[8:0:-2])
>>[8 6 4 2]
print(B[:4])
```

```
>>[0 1 2 3]
print(B[5:])
>>[5 6 7 8 9]
print(B[::-1])
>>[9 8 7 6 5 4 3 2 1 0]
```

Arithmetic Operations with NumPy Array

- The arithmetic operations with NumPy arrays perform element-wise operations, this means the operators are applied only between corresponding elements.
- Arithmetic operations are possible only if the array has the **same structure and dimensions.**

Basic operations : with scalars

```
import numpy as np
a = np.array([1,2,3,4,5])
b = a+1
print(b)
c = 2**a
print(c)
```

Output:

```
[2 3 4 5 6]
[ 2 4 8 16 32]
```

Matrix operations

Addition

```
import numpy as np
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B # element wise addition
```

```
print(C)
>>[[11 1]
  [ 8 0]]
```

Subtraction

```
import numpy as np
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A - B # element wise subtraction
print(C)
>>[[ -7 7]
  [ 2 -12]]
```

Multiplication (Element-wise matrix multiplication or the Hadamard product)

```
import numpy as np
A = np.array([[2, 4], [5, -6]])
print(A*A)
>>[[ 4 16]
  [25 36]]
```

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
print(A*2)
>>[[2 4]
  [6 8]]
```

Transpose

```
import numpy as np
A = np.array([[2, 4], [5, -6]])
```

```
print(A.T)
print(A.transpose())
>>[[ 2, 5],
   [ 4, -6]])
```

Inverse

Matrix inversion is a process that finds another matrix that when multiplied with the matrix, results in an identity matrix. Not all matrices are invertible. A square matrix that is not invertible is referred to as singular.

```
from numpy import array
from numpy.linalg import inv
# define matrix
A = array([[1.0, 2.0],[3.0, 4.0]])
print(A)
B = inv(A)    # invert matrix
print(B)
I = A.dot(B)    # multiply A and B
print(I)
```

Trace

A trace of a square matrix is the sum of the values on the main diagonal of the matrix (top-left to bottom-right).

```
# matrix trace
from numpy import array
from numpy import trace
# define matrix
A = array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
print(A)
```

```
# calculate trace
B = trace(A)
print(B)
```

Determinant

The determinant of a square matrix is a scalar representation of the volume of the matrix.

```
from numpy import array
from numpy.linalg import det
# define matrix
A = array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
print(A)
# calculate determinant
B = det(A)
print(B)
```

Matrix-Matrix Multiplication

- Matrix multiplication, also called the matrix dot product.
- It is more complicated than the previous operations and involves a rule as not all matrices can be multiplied together.
- The rule for matrix multiplication is as follows:

  The number of columns (n) in the first matrix (A) must equal the number of rows (m) in the second matrix (B).

Example:

```
from numpy import array
# define first matrix
A = array([[1, 2],[3, 4],[5, 6]])
```

```
print(A)
# define second matrix
B = array([[1, 2],[3, 4]])
print(B)
# multiply matrices
C = A.dot(B)
 print(C)
```

Random Numbers

- Random means something that cannot be predicted logically. Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.
- If there is a program to generate random number it can be predicted, thus it is not truly random. Random numbers generated through a generation algorithm are called pseudo random.
- In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc. Pseudo random number generation can be done with numpy **random module**.
- The random module's randint() method returns a **random number** from 0 to n.

```
import numpy as np
x = np.random.randint(100)
print(x)
>>64
```

- The **randint()** method takes a size parameter where you can specify the shape of an array. The following commands will generate 5 random numbers from 0 to 100.

```
import numpy as np
x = np.random.randint(100,size=5)
print(x)
>>[25 62 24 81 39]
```

- The following will Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
import numpy as np
x = np.random.randint(100,size=(3,5))
print(x)
>>[[ 2 96 40 43 85]
   [81 81  4 48 29]
   [80 31  6 10 24]]
```

- The random module's **rand()** method returns a **random float** between 0 and 1.

```
import numpy as np
x = np.random.rand()
print(x)
>>0.2733166576024767
```

- This will generate 10 random numbers

```
x = np.random.rand(10)
print(x)
>>[0.82536563  0.46789636  0.28863107  0.83941914  0.24424812  0.25816291
0.72567413 0.80770073 0.32845661 0.34451507]
```

- Generate an array with size (3,5)

```
x = np.random.rand(3,5)
print(x)
```

>>[[0.16220086 0.80935717 0.97331357 0.60975199 0.48542906] [0.68311884 0.27623475 0.73447814 0.29257476 0.27329666] [0.62625815 0.0069779 0.21403868 0.49191027 0.4116709]]

- The **choice()** method allows to get a random value from an array of values.

```
import numpy as np
x = np.random.choice([3,5,6,7,9,2])
print(x)
>>3
import numpy as np
x = np.random.choice([3,5,6,7,9,2],size=(3,5))
print(x)
>>[[3 2 5 2 6]
   [5 9 3 6 9]
   [5 6 9 3 3]]
```

<u>Random Permutations</u>

- A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.
- The NumPy Random module provides two methods for this: shuffle() and permutation().

<u>Shuffling Arrays</u>

- Shuffle means changing arrangement of elements in-place. i.e. in the array itself.

```
import numpy as np
x=np.array([1,2,3,4,5])
```

```
np.random.shuffle(x)
print(x)
>>[4 1 3 5 2]
```

Generating Permutation of Arrays

- The **permutation()** method returns a re-arranged array (and leaves the original array un-changed).

```
import numpy as np
x=np.array([1,2,3,4,5])
y=np.random.permutation(x)
print(y)
>>[3 1 5 2 4]
```

Matplotlib

- Matplotlib is one of the most popular Python packages used for data visualization.
- It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy.
- One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Let’s plot a simple sin wave using matplotlib

1. To begin with, the Pyplot module from Matplotlib package is imported

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

2. Next we need an array of numbers to plot.

   ```
   import numpy as np
   ```

```
import math
x = np.arange(0, math.pi*2, 0.05)
```

3. The ndarray object serves as values on x axis of the graph. The corresponding sine values of angles in x to be displayed on y axis are obtained by the following statement

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

4. The values from two arrays are plotted using the plot() function.

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

5. You can set the plot title, and labels for x and y axes.You can set the plot title, and labels for x and y axes.

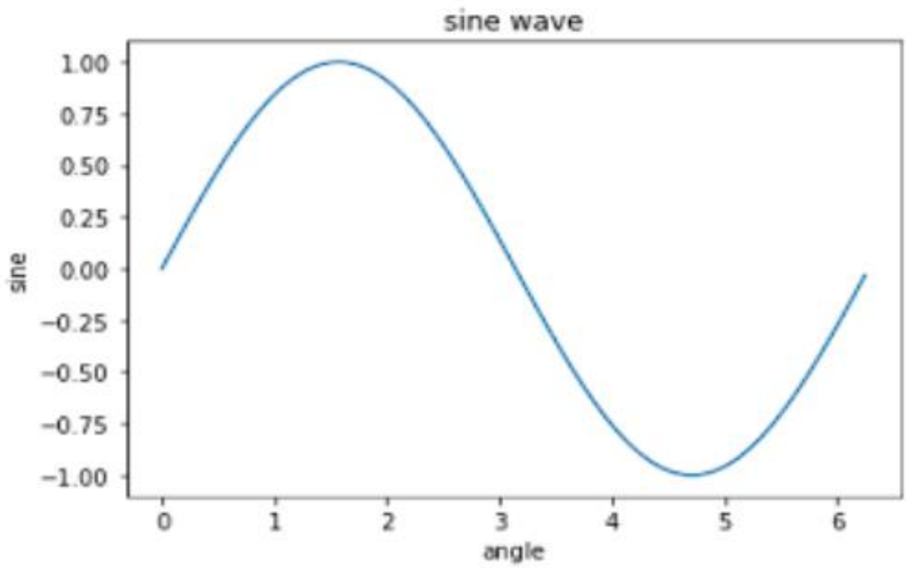
```
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

6. The Plot viewer window is invoked by the show() function

```
plt.show()
```

The complete program is as follows –

```
from matplotlib import pyplot as plt
import numpy as np
import math #needed for definition of pi
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```

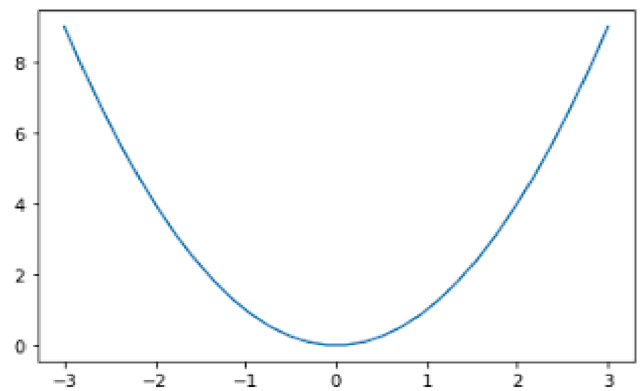


Matplotlib - PyLab module

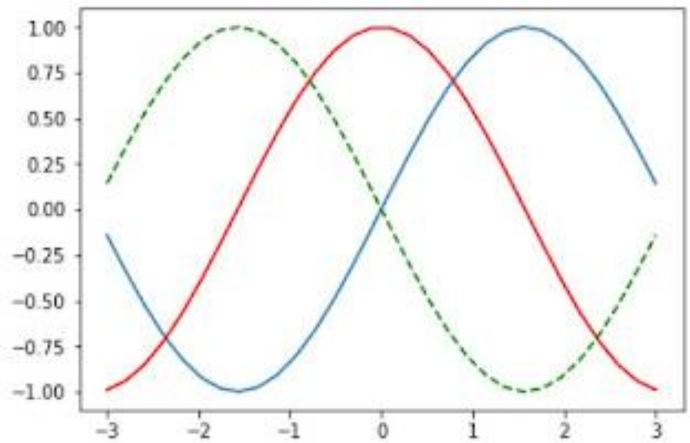
- PyLab is a procedural interface to the Matplotlib object-oriented plotting library.
- PyLab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and NumPy (for Mathematics and working with arrays) in a single name space.

basic plot

```
from numpy import *
from pylab import *
x = linspace(-3, 3, 30)
y = x**2
plot(x, y)
show()
```



```
from pylab import *
x = np.arange(0, math.pi*2, 0.05)
plot(x, sin(x))
plot(x, cos(x), 'r-')
plot(x, -sin(x), 'g--')
show()
```



Color code

Character	Color
‘b’	Blue
‘g’	Green
‘r’	Red
‘b’	Blue
‘c’	Cyan
‘m’	Magenta
‘y’	Yellow
‘k’	Black
‘b’	Blue
‘w’	White

Marker code

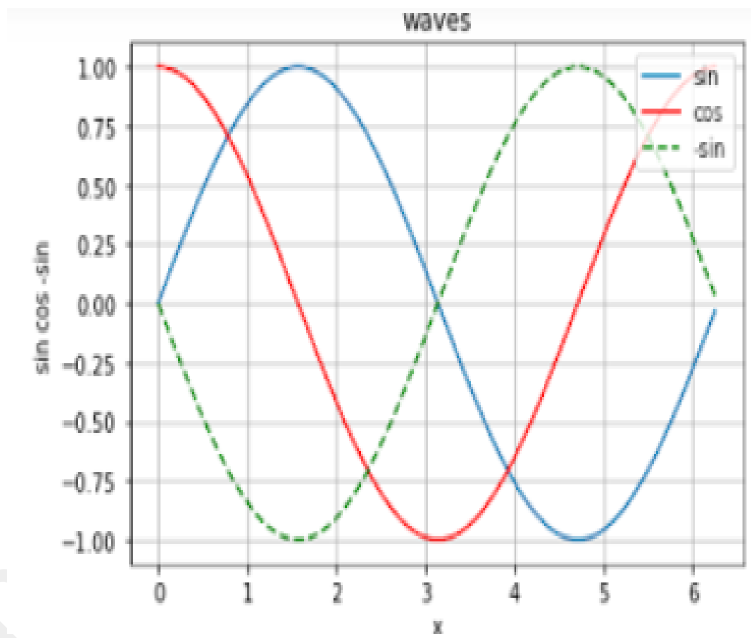
Character	Description
‘.’	Point marker
‘o’	Circle marker
‘x’	X marker
‘D’	Diamond marker
‘H’	Hexagon marker
‘s’	Square marker
‘+’	Plus marker

Line style

Character	Description
‘-‘	Solid line
‘—‘	Dashed line
‘-.’	Dash-dot line
‘:’	Dotted line
‘H’	Hexagon marker

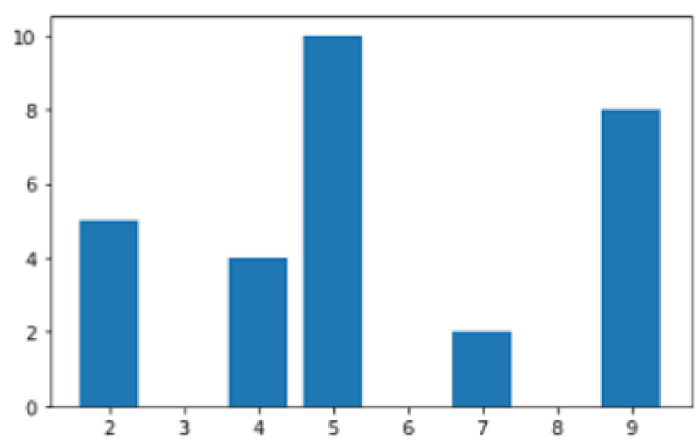
Adding Grids and Legend to the Plot

```
from pylab import *
x = np.arange(0, math.pi*2, 0.05)
plot(x, sin(x),label='sin')
plot(x, cos(x), 'r-',label='cos')
plot(x, -sin(x), 'g--',label='-sin')
grid(True)
title('waves')
xlabel('x')
ylabel('sin cos -sin')
legend(loc='upper right')
show()
```



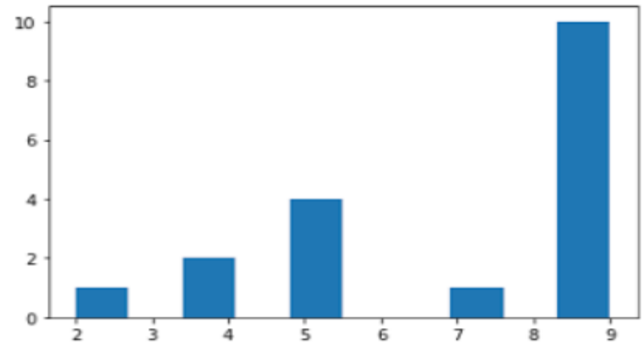
Creating a bar plot

```
from matplotlib import pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
# Function to plot the bar
plt.bar(x,y)
# function to show the plot
plt.show()
```



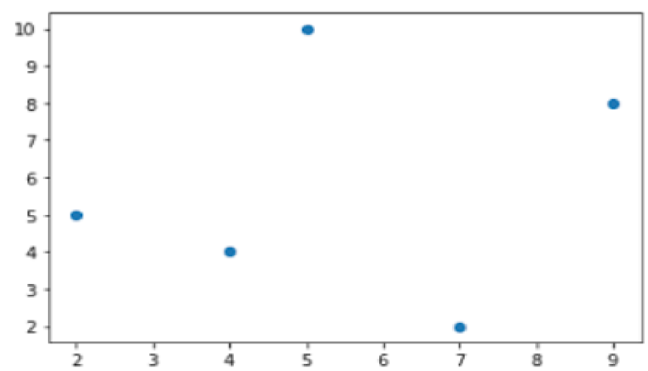
Creating a histogram

```
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7,5,5,5,4,9,9,9,9,9,9,9,9,9]
# Function to plot the histogram
plt.hist(x)
# function to show the plot
plt.show()
```



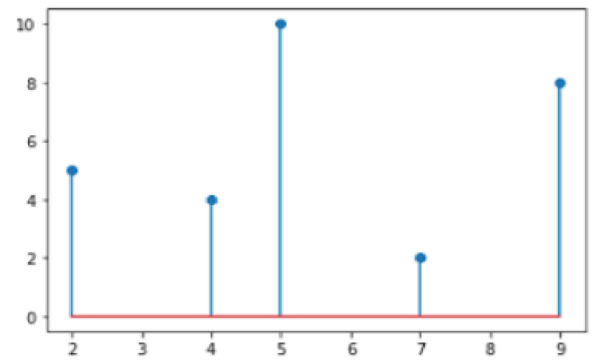
Scatter Plot

```
from matplotlib import pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
# Function to plot scatter
plt.scatter(x, y)
plt.show()
```



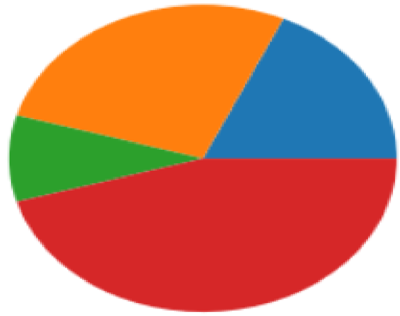
Stem plot

```
from matplotlib import pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
# Function to plot scatter
plt.stem(x, y,use_line_collection=True)
plt.show()
```



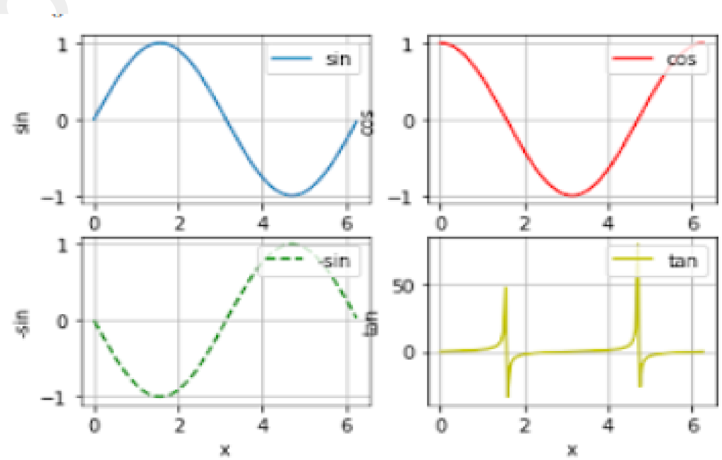
Pie Plot

```
data=[20,30,10,50]
from pylab import *
pie(data)
show()
```



Subplots with in the same plot

```
from pylab import *
x = np.arange(0, math.pi*2, 0.05)
subplot(2,2,1)
plot(x, sin(x),label='sin')
xlabel('x')
ylabel('sin')
legend(loc='upper right')
grid(True)
subplot(2,2,2)
plot(x, cos(x), 'r-',label='cos')
xlabel('x')
ylabel('cos')
legend(loc='upper right')
grid(True)
subplot(2,2,3)
xlabel('x')
ylabel('-sin')
plot(x, -sin(x), 'g--',label='-sin')
```



```
legend(loc='upper right')
grid(True)
subplot(2,2,4)
xlabel('x')
ylabel('tan')
plot(x, tan(x), 'y-',label='tan')
legend(loc='upper right')
grid(True)
show()
```

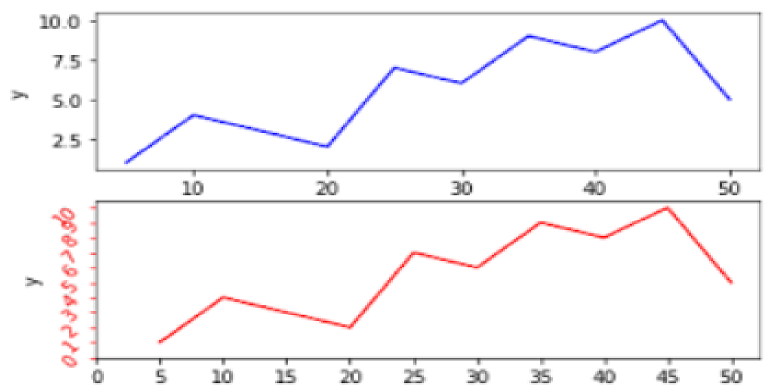
<u>Ticks in Plot</u>

- Ticks are the values used to show specific points on the coordinate axis. It can be a number or a string. Whenever we plot a graph, the axes adjust and take the default ticks.
- Matplotlib's default ticks are generally sufficient in common situations but are in no way optimal for every plot. Here, we will see how to customize these ticks as per our need.

<u>The following program shows the default ticks and customized ticks</u>

```
import matplotlib.pyplot as plt
import numpy as np
 # values of x and y axes
x = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
y = [1, 4, 3, 2, 7, 6, 9, 8, 10, 5]
figure(1)
```

```
plt.plot(x, y, 'b')
plt.xlabel('x')
plt.ylabel('y')
figure(2)
plt.plot(x, y, 'r')
plt.xlabel('x')
plt.ylabel('y')
# 0 is the initial value, 51 is the final value
# (last value is not taken) and 5 is the difference
# of values between two consecutive ticks
plt.xticks(np.arange(0, 51, 5))
plt.yticks(np.arange(0, 11, 1))
plt.tick_params(axis='y',colors='red',rotation=45)
plt.show()
```



PARAMETER	VALUE	USE
axis	x, y, both	Tells which axis to operate
reset	True, False	If True, set all parameters to default
direction	in, out, inout	Puts the ticks inside or outside or both
length	Float	Sets tick's length
width	Float	Sets tick's width
rotation	Float	Rotates ticks wrt the axis
colors	Color	Changes tick color
pad	Float	Distance in points between tick and label

Working with CSV Files

- CSV is a delimited data format that has fields/columns separated by the comma character and records/rows terminated by newlines.
- A CSV file does not require a specific character encoding, byte order, or line terminator format. A record ends at a line terminator.
- All records should have the same number of fields, in the same order. Data within fields is interpreted as a sequence of characters, not as a sequence of bits or bytes.

CSV File Characteristics

- One line for each record
- Comma separated fields
- Space-characters adjacent to commas are ignored
- Fields with in-built commas are separated by double quote characters.

- Fields with double quote characters must be surrounded by double quotes. Each inbuilt double quote must be represented by a pair of consecutive quotes.

Pandas-Panal Data and Python Data Analysis

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance and productivity for users.

Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN)
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets

Pandas generally provide two data structure for manipulating data, They are:

- Series
- DataFrame

Series

- Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
- The axis labels are collectively called index. Pandas Series is nothing but a column in an excel sheet.
- The simplest Series is formed from only an array of data.

  Example:

```
import pandas as pd
obj=pd.Series([3,5,-8,7,9])
print(obj)
0    3
1    5
2   -8
3    7
4    9
```

- Often it will be desirable to create a Series with an index identifying each data point:

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(obj2)
d    4
b    7
a   -5
c    3
```

- If you have data contained in a Python dict, you can create a Series from it by passing the dict:

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3=pd.Series(sdata)

print(obj3)
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
```

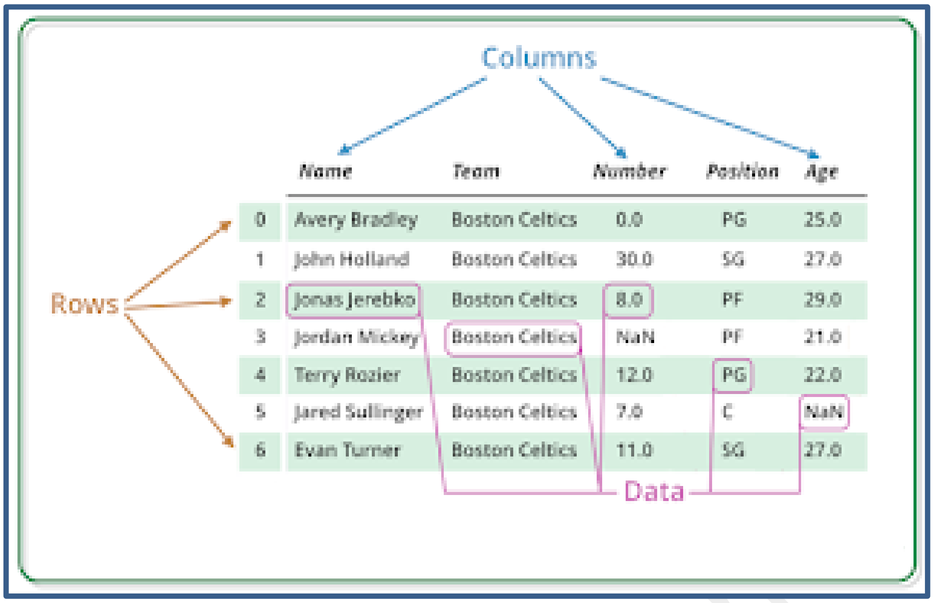
- The **isnull()** and **notnull()** functions in pandas should be used to detect missing data:

Series basic functionality

Sr.No.	Attribute or Method & Description
1	**axes** Returns a list of the row axis labels
2	**dtype** Returns the dtype of the object.
3	**empty** Returns True if series is empty.
4	**ndim** Returns the number of dimensions of the underlying data, by definition 1.
5	**size** Returns the number of elements in the underlying data.
6	**values** Returns the Series as ndarray.
7	**head()** Returns the first n rows.
8	**tail()** Returns the last n rows.

Pandas DataFrame

- Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- Pandas DataFrame consists of three principal components, the data, rows, and columns.



Basic operation which can be performed on Pandas DataFrame

- Creating a DataFrame
- Dealing with Rows and Columns
- Indexing and Selecting Data
- Working with Missing Data
- Iterating over rows and columns

Creating a DataFrame

- In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file.
- Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

```
import pandas as pd
 lst = ['mec', 'minor', 'stud', 'eee',  'bio']
 df = pd.DataFrame(lst)
```

```
print(df)
      0
0    mec
1  minor
2   stud
3    eee
4    bio
```

Creating DataFrame from dict of ndarray/lists:

```
import pandas as pd
# initialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
# Create DataFrame
df = pd.DataFrame(data)
 # Print the output.
print(df)
    Name  Age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

Dealing with Rows and Columns

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

Column Selection: In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

```
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

 # Convert the dictionary into DataFrame
df = pd.DataFrame(data)
 # select two columns
print(df)
print(df[['Name', 'Qualification']])
```

Row Selection: Pandas provide a unique method to retrieve rows from a Data frame. **DataFrame.loc[]** method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an **iloc[]** function.

```
print(data.loc['Jai'])
print(data.iloc[1])
```

Indexing and Selecting Data

Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns.

Working with Missing Data

- Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data can also refer to as NA(Not Available) values in pandas.
- In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull().
- Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

```
import pandas as pd
import numpy as np
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
df = pd.DataFrame(dict)
print(df.isnull())
```

```
       First Score  Second Score  Third Score
0        False         False         True
1        False         False        False
2         True         False        False
3        False          True        False
```

```
print(df.notnull())
```

```
       First Score  Second Score  Third Score
0         True          True        False
1         True          True         True
2        False          True         True
3         True         False         True
```

Filling missing values using fillna(), replace() and interpolate() :

```
import pandas as pd
```

```
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
print(df)
   First Score Second Score   Third Score
0      100.0        30.0             NaN
1       90.0        45.0            40.0
2        NaN        56.0            80.0
3       95.0         NaN            98.0

# filling missing value using fillna()
print(df.fillna(0))

   First Score  Second Score Third Score
0      100.0        30.0          0.0
1       90.0        45.0         40.0
2        0.0        56.0       80.0
3       95.0         0.0       98.0

#filling the NaN values by interpolation
print(df.interpolate())
```

	First Score	Second Score	Third Score
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	92.5	56.0	80.0
3	95.0	56.0	98.0

```
#replacing the nan values with -1
print(df.replace(np.nan,-1))
```

	First Score	Second Score	Third Score
0	100.0	30.0	-1.0
1	90.0	45.0	40.0
2	-1.0	56.0	80.0
3	95.0	-1.0	98.0

```
#dropping the rows containing null values
print(df.dropna())
```

	First Score	Second Score	Third Score
1	90.0	45.0	40.0

<u>DataFrame basic functionality</u>

Sr.No.	Attribute or Method & Description
1	**T** Transposes rows and columns.
2	**axes** Returns a list with the row axis labels and column axis labels as the only members.
3	**dtypes** Returns the dtypes in this object.
4	**empty** True if NDFrame is entirely empty [no items]; if any of the axes are of length 0.
5	**ndim** Number of axes / array dimensions.
6	**shape** Returns a tuple representing the dimensionality of the DataFrame.
7	**size** Number of elements in the NDFrame.
8	**values** Numpy representation of NDFrame.
9	**head()** Returns the first n rows.
10	**tail()** Returns last n rows

Pandas read_csv() and to_csv() Functions

- The process of creating or writing a CSV file through Pandas can be a little more complicated than reading CSV, but it's still relatively simple.
- We use the **to_csv()** function to perform this task.
- **read_csv()** function used to read data in a csv file.

Here is a simple example showing how to export a DataFrame to a CSV file via to_csv():

```
# importing pandas as pd
import pandas as pd
# dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
 'degree': ["MBA", "BCA", "M.Tech", "MBA"],
```

'score':[90, 40, 80, 98]}

creating a dataframe from a dictionary

df = pd.DataFrame(dict)

print(df)

df.to_csv('studdata.csv')

df.read_csv('studdata.csv')

Pandas Descriptive Statistics

sr.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value
9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

The following are the various functions you can do on this data file

importing pandas as pd

import pandas as pd

df=pd.read_csv('stud.csv',index_col='rollno')

print("data frame stud")

print(df)

```
data frame stud
          name     place  mark
rollno
101       binu  ernkulam    45
103      ashik  alleppey    35
102     faisal    kollam    48
105       biju   kotayam    25
106        ann   thrisur    25
107      padma      kylm    25
```

print("statistical info of numerical column")

print(df.describe())

```
statistical info of numerical column
            mark
count   6.000000
mean   33.833333
std    10.590877
min    25.000000
25%    25.000000
50%    30.000000
75%    42.500000
max    48.000000
```

print("coulmns")

print(df.columns)

```
coulmns
Index(['name', 'place', 'mark'], dtype='object')
```

print("size")

print(df.size)

```
size
18
```

print("data types")

print(df.dtypes)

```
data types
name      object
place     object
mark       int64
```

print("shapes")

print(df.shape)

```
shapes
(6, 3)
```

print("index and length of index")

print(df.index,len(df.index))

```
index and length of index
Int64Index([101, 103, 102, 105, 106, 107], dtype='int64', name='rollno') 6
```

print("statistical functions")

print("sum=",df['mark'].sum())

print("mean=",df['mark'].mean())

print("max=",df['mark'].max())

print("min=",df['mark'].min())

print("var=",df['mark'].var())

print("standard deviation=",df['mark'].std())

print(df.std())

```
statistical functions
sum= 203
mean= 33.833333333333336
max= 48
min= 25
var= 112.16666666666667
standard deviation= 10.59087657687817
mark    10.590877
```

print("top 2 rows")

print(df.head(2))

```
top 2 rows
          name     place  mark
rollno
101       binu  ernkulam    45
103      ashik  alleppey    35
```

print("last 2 rows")

print(df.tail(2))

```
last 2 rows
          name    place  mark
rollno
106        ann  thrisur    25
107      padma     kylm    25
```

print("data from rows 0,1,2")

print(df[0:3])

```
data from rows 0,1,2
          name     place  mark
rollno
101       binu  ernkulam    45
103      ashik  alleppey    35
102     faisal    kollam    48
```

print("mark column values")

print(df['mark'])

```
mark column values
rollno
101    45
103    35
102    48
105    25
106    25
107    25
```

print("rows where mark >40")

print(df[df['mark']>40])

print("rows 0,1,2 columns 0,2")

print(df.iloc[0:3,[0,2]])

```
rows where mark >40
          name     place  mark
rollno
101       binu  ernkulam    45
102     faisal    kollam    48
rows 0,1,2 columns 0,2
          name  mark
rollno
101       binu    45
103      ashik    35
102     faisal    48
```

print("sorting in the descending order of marks")

print(df.sort_values(by='mark',ascending=False))

```
sorting in the descending order of marks
          name     place  mark
rollno
102     faisal    kollam    48
101       binu  ernkulam    45
103      ashik  alleppey    35
105       biju   kotayam    25
106        ann   thrisur    25
107      padma      kylm    25
```

print("use agg function to compute all the values")

print(df['mark'].agg(['min','max','mean']))

```
use agg function to compute all the values

min   25.000000
max     48.000000
mean    33.833333
Name: mark, dtype: float64
```

print("median of marks")

```
print("Median",df.sort_values(by='mark',ascending=False).median())
print("mode of marks")
print("Mode",df.sort_values(by='mark',ascending=False)['mark'].mode())
print("count of marks")
print(df['mark'].value_counts())
```

```
median of marks
Median mark    30.0
dtype: float64

mode of marks
Mode 0    25
dtype: int64

count of marks
25    3
45    1
35    1
48    1
```

```
print("grouping data based on column value")
print(df.groupby('mark')['mark'].mean())
```

```
grouping data based on column value
mark
25    25
35    35
45    45
48    48
```

```
print("plotting the histogram")
import  matplotlib.pyplot as plt
figure(1)
plt.hist(df['mark'])
figure(2)
plt.scatter(df['name'],df['mark'])
figure(3)
```

plt.pie(df['mark'])