```
# Libraries in Python:
'''
=> Python library is a collection of functions and methods that allows
you
   to perform many actions without writing your code.
'''
#! Python Numpy
'''
=> It consists of multidimensional array objects and a collection of
   routines for processing those arrays.
'''
```

```
' \n=> It consists of multidimensional array objects and a collection
of \n   routines for processing those arrays.\n'
```

```
# ! Single Dimensional Array:
import numpy as np
n1 = np.array([10,20,30,40])
n1
```

```
array([10, 20, 30, 40])
```

```
type(n1)
```

```
numpy.ndarray
```

```
''' .ndarray => n-dimensional array '''
```

```
#! Multidimensional Array (Matrix):
import numpy as np
n2 = np.array([[1,2,3,4],[10,20,30,40]])
n2
```

```
array([[ 1,  2,  3,  4],
       [10, 20, 30, 40]])
```

```
type(n2)
```

```
numpy.ndarray
```

```
# .shape => To check the rows and columns of matrix.
n2.shape
```

```
(2, 4)
```

```
''' (2, 4) => 2 Rows and 4 Columns '''
```

```
# Initializing NumPy array with zeros:  .zeros((row, column))
import numpy as np
n1 = np.zeros((1,2))      # 1 row and 2 columns
n1
```

```
array([[0., 0.]])
```

```
n2 = np.zeros((5,5))        # 5 rows and 5 columns
n2
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
# Initializing NumPy array with same number:   .full((rows, column),
number)
n1 = np.full((3,4),10)      # 3 rows and 4 columns of number 10
n1
```

```
array([[10, 10, 10, 10],
       [10, 10, 10, 10],
       [10, 10, 10, 10]])
```

```
# Initializing NumPy array within a range :    .arange(start, stop)
n1 = np.arange(15,25)            # 15 -> included, 25 -> excluded
n1
```

```
array([15, 16, 17, 18, 19, 20, 21, 22, 23, 24])
```

```
# Range with index (jump):       .arange(start,stop,index)
n2 = np.arange(15,25,3)           # 3 -> index (jump)
n2
```

```
array([15, 18, 21, 24])
```

```
n1 = np.arange(10,20)
n1
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
n2 = np.arange(0,50,5)
n2
```

```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])
```

```
# Initializing NumPy array with random
numbers:     .random.randint(start,stop, no.of elememt)
n1 = np.random.randint(1,100,10)
n1
```

```
array([76, 48, 86, 48, 51, 26, 55, 85, 72, 61])
```

```
''' Everytime it will exicuted, we will get different output. '''
```

```
# Checking and changing the shape of NumPy araray:
n1 = np.array([[1,2,3,4],[10,20,30,40]])
n1
```

```
array([[ 1,  2,  3,  4],
       [10, 20, 30, 40]])

n1.shape   # Checking the shape

(2, 4)

# Changing the shape:
n1.shape = (4,2)
n1

array([[ 1,  2],
       [ 3,  4],
       [10, 20],
       [30, 40]])

n1.shape

(4, 2)

n1.shape = (1,8)
n1

array([[ 1,  2,  3,  4, 10, 20, 30, 40]])

n2 = np.array([[1,2,3,],[10,20,30],[30,20,10],[3,2,1]])
n2

array([[ 1,  2,  3],
       [10, 20, 30],
       [30, 20, 10],
       [ 3,  2,  1]])

n2.shape

(4, 3)

n2.shape = (6,2)
n2

array([[ 1,  2],
       [ 3, 10],
       [20, 30],
       [30, 20],
       [10,  3],
       [ 2,  1]])

n2.shape = (2,6)
n2

array([[ 1,  2,  3, 10, 20, 30],
       [30, 20, 10,  3,  2,  1]])
```

```
# Joining NumPy Array:
#! .vstack() => Stacking Vertically

import numpy as np
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
np.vstack((n1,n2))

array([[10, 20, 30],
       [40, 50, 60]])

''' .stack() or .vstack() '''

np.stack((n1,n2))

array([[10, 20, 30],
       [40, 50, 60]])

np.vstack((n2,n1))

array([[40, 50, 60],
       [10, 20, 30]])


#! .hstack() => Stacking Horizontally
np.hstack((n1,n2))

array([10, 20, 30, 40, 50, 60])

np.hstack((n2,n1))

array([40, 50, 60, 10, 20, 30])


#! .column_stack() => Stacking Column-wise
np.column_stack((n1,n2))

array([[10, 40],
       [20, 50],
       [30, 60]])

np.column_stack((n2,n1))

array([[40, 10],
       [50, 20],
       [60, 30]])

# NumPy Intersection and Difference
#! Intersection => Common => .intersect1d()
#! Difference => Unique => .setdiff1d()

import numpy as np
```

```
n1 = np.array([10,20,30,40,50])
n2 = np.array([40,50,60,70])

np.intersect1d(n1,n2)

array([40, 50])

np.setdiff1d(n1,n2)       # Unique elements in array n1

array([10, 20, 30])

np.setdiff1d(n2,n1)       # Unique elements in array n2

array([60, 70])

# NumPy Array Mathematics:
#! Addition of Numpy Array:
import numpy as np
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])

# Sum of Matrix:
np.sum([n1,n2])         # 10+20+30+40+5+60 = 210

210

# Sum of rows' value:  => Set 'axis=1'
np.sum([n1,n2], axis=1)

array([ 60, 150])

'''
axis=1 => rows
Total Rows = 2
1. 10+20+30 = 60
2. 40+50+60 = 150
'''

# Sum of columns' values:  => Set 'axis=0'
np.sum([n1,n2],axis=0)

array([50, 70, 90])

'''
axis=0 => column,
Total Columns => 3
1. 10+40 = 50
2. 20+50 = 70
3. 30+60 = 90
'''

# Scalar value => I-D Value
# Basic Addition with Scalar Value:
import numpy as np
```

```
n1 = np.array([10,20,30,40])
sum = n1 + 1
sum
```

```
array([11, 21, 31, 41])
```

```
# Basic Subtraction with Scalar Value:
sub = n1 - 1
sub
```

```
array([ 9, 19, 29, 39])
```

```
# Basic Multiplication with Scalar Value:
mul = n1 * 2
mul
```

```
array([20, 40, 60, 80])
```

```
# Basic Division with Scalar Value:
div = n1 / 5
div
```

```
array([2., 4., 6., 8.])
```

```
# Basic Floor Division with Scalar Value:
fdiv = n1 // 5
fdiv
```

```
array([2, 4, 6, 8], dtype=int32)
```

```
# Basic Exponention(Power) with Scalar Value:
pow = n1 ** 3
pow
```

```
array([ 1000,  8000, 27000, 64000], dtype=int32)
```

```
# Numpy Math Function:

#! Mean => .mean()
import numpy as np
n1 = np.array([10,20,30,40,50])
np.mean(n1)
```

```
30.0
```

```
'''
mean = (Sum of the terms) / (Number of terms)
mean = (10+20+30+40+50) / 5
mean = 30.0
'''
```

```
#! Median => .median()
n1 = np.array([20,30,50,10,60,40])
np.median(n1)
```

```
35.0
```

```
'''
1. n = Number of terms
2. Arrange data in ascending or descending orer.

If n is odd.
Median = (n+1)/2 th term of arranged data in ascending or descending
order.

If n is even.
Median = Mean of (n/2)th term and ((n/2)+1)th term
or
Median = [(n/2)th term + ((n/2)+1)th term] / 2


'''

# Explanation:
'''
[20,30,50,10,60,40]
1. n = 6 (even)
2. arranged data: 10,20,30,40,50,60
3. (6/2)th term = 3rd term = 30
   ((6/2)+1)th term = (3+1)th term = 4th term = 40

Median = (30 + 40) / 2 = 35.0
Median = 35.0
'''


#! Standard Deviation =>  .std()
n1 = np.array([20,30,50,10,60,40])
np.std(n1)
```

```
17.07825127659933
```

```
n2 = np.array([10,20,30,40,50])
np.std(n2)
```

```
14.142135623730951
```

```
# NOTE :
'''
Standard Deviation explanation is in pdf notes.
or
```

```
You can learn from Google.
'''

# NumPy Save and Load:
#! Saving NumPy Array:     .save('name', data)

import numpy as np
n1 = np.array([2,4,9,7,10,12,11,18])
np.save('myarray',n1)           # Saving


#! Loading NumPy Array:    .load('name.npy') => use '.npy' extension.
newarray = np.load('myarray.npy')
newarray

array([ 2,  4,  9,  7, 10, 12, 11, 18])

'''
.npy => ,npy extension used to load the saved values.
'''

# Question : Take 10 random values in the range of 1 to 50 and
#            print all random values and its mean, median, standard
deviation.
#!           After that run that program for 2-3 times and observe it.

''' Solution:'''
import numpy as np
n1 = np.random.randint(1,50,10)
print(n1)

mean = np.mean(n1)
print("Mean:",mean)

med = np.median(n1)
print("Median:", med)

sd = np.std(n1)
print("Standard Deviation:", sd)

# NOTE : After every execution, printed values will be change,
#        because we used .random.randint() function.


[28 49 28 21 13 30 39 49 10 33]
Mean: 30.0
Median: 29.0
Standard Deviation: 12.609520212918492

# Matrix Multiplication:       .matmul()
import numpy as np
```

```
n1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
n2 = np.array([[5,2,7],[4,1,8],[11,17,25]])

n1

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

n2

array([[ 5,  2,  7],
       [ 4,  1,  8],
       [11, 17, 25]])

a = np.matmul(n1,n2)
a

array([[ 46,  55,  98],
       [106, 115, 218],
       [166, 175, 338]])

b = np.matmul(n2,n1)
b

array([[ 62,  76,  90],
       [ 64,  77,  90],
       [254, 307, 360]])

# Alternative method for .matmul() => .dot()
c = np.dot(n1,n2)
c

array([[ 46,  55,  98],
       [106, 115, 218],
       [166, 175, 338]])

d = np.dot(n2,n1)
d

array([[ 62,  76,  90],
       [ 64,  77,  90],
       [254, 307, 360]])

# Addition(Sum) of Matrix:
import numpy as np
v1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
v2 = np.array([[5,2,7],[4,1,8],[11,17,25]])

sum = np.add(v1,v2)
sum
```

```
array([[ 6,  4, 10],
       [ 8,  6, 14],
       [18, 25, 34]])

# Subtraction =>  .subtract()
sub1 = np.subtract(v1,v2)
sub1

array([[ -4,   0,  -4],
       [  0,   4,  -2],
       [ -4,  -9, -16]])

sub2 = np.subtract(v2,v1)
sub2

array([[ 4,  0,  4],
       [ 0, -4,  2],
       [ 4,  9, 16]])

# Division =>  .divide()
# Or True Divison =>  .true_divide()
div = np.divide(v1,v2)
div

array([[0.2       , 1.        , 0.42857143],
       [1.        , 5.        , 0.75      ],
       [0.63636364, 0.47058824, 0.36      ]])

div = np.divide(v2,v1)
div

array([[5.        , 1.        , 2.33333333],
       [1.        , 0.2       , 1.33333333],
       [1.57142857, 2.125     , 2.77777778]])

# Floor Division
fdiv = np.floor_divide(v1,v2)
fdiv

array([[0, 1, 0],
       [1, 5, 0],
       [0, 0, 0]])

fdiv = np.floor_divide(v2,v1)
fdiv

array([[5, 1, 2],
       [1, 0, 1],
       [1, 2, 2]])

# Linear Multiplication in NumPy =>  .multiply()
l_mul = np.multiply(v1,v2)
l_mul
```

```
array([[  5,   4,  21],
       [ 16,   5,  48],
       [ 77, 136, 225]])
```