

# NUMPY

**NUMPY**-Numpy stands for “Numeric Python” or “Numerical python”.Numpy is a package that contains several classes, functions, variables etc. to deal with scientific calculations in Python.

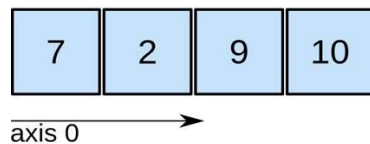
- Numpy is useful to create and process single and multi-dimensional arrays.
- In addition, numpy contains a large library of mathematics like linear algebra functions and Fourier transformations.
- Numpy installation:- `pip install numpy`

The arrays which are created using numpy are called n dimensional arrays where n can be any integer. If  $n = 1$  it represent a one dimensional array. If  $n = 2$ , it is a two dimensional array etc.

Numpy array can accept only one type of elements. We cannot store different data types into same arrays.

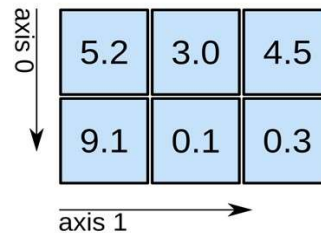
## ARRAY STRUCTURE IN NUMPY

1D array



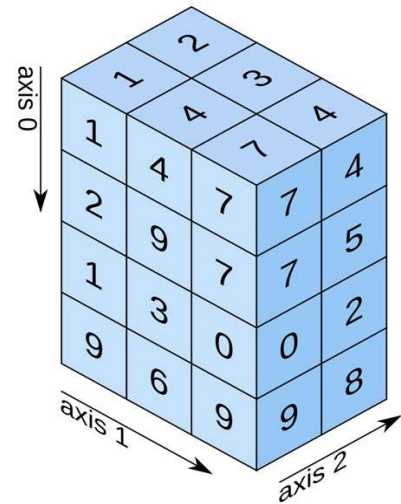
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

## PYTHON AND NUMPY

For working with numpy, we should first import numpy module into our Python program.

Following line of code is use to import numpy in the python programme.

```
import numpy or  
import numpy as <<name>>
```

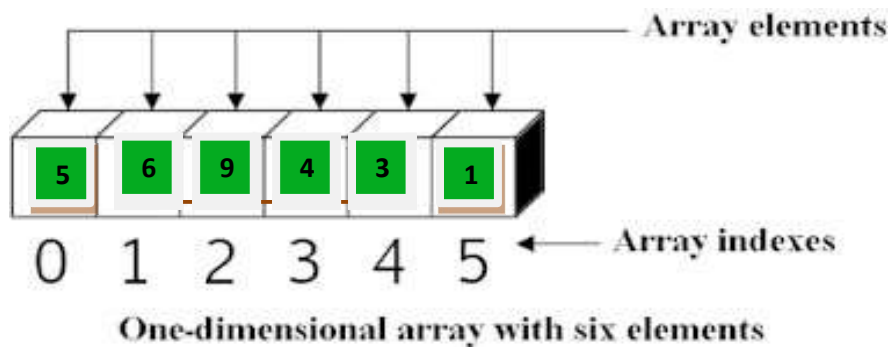
## TYPES OF ARRAY IN NUMPY

An array in numpy is of the following types-

1. 1D Array
2. 2D Array
3. N-Dimension Array

### 1D ARRAY

**1D Array-** One dimensional array contains elements only in one dimension. In other words, the shape of the **numpy array** should contain only one value in the tuple.



A simple program to implement one dimensional array using numpy

#### Example 1

```
import numpy
a = numpy.array([10,20,30,40,50])
print(a)
```

Output: [10,20,30,40,50]

```
import numpy as np  
  
a = np.array([10,20,30,40,50])  
  
print(a)
```

**Output: [10, 20, 30, 40, 50]**

Note: if we use the following statement then there is no need to add anything in front of array function.

```
from numpy import *  
  
a = array([10, 20,30,40,50])  
  
print(a)
```

**Output : [10, 20,30,40,50]**

## IMPLEMENTATION OF 1D ARRAY IN NUMPY

Creating array in numpy can be done in several ways. Some of the important ways are-

- i. Using array() function
- ii. Using linspace() function
- iii. Using arange() function
- iv. Using zeros() and ones() functions

## 1. Using array() function

Using this function we can create array of any data type, but if not data types is mentioned the default data type will be the "int"

For e.g :

```
from numpy import *  
  
Arr=array([10,20,30,40,50],int) is similar to  
  
arr = array([10,20,30,40,50])
```

While creating array if one of the values in the specified list belongs to float then all the values will be converted to float by default.

```
from numpy import *  
  
a = array([10,30,40.5, 50,100])  
  
print(a)
```

**Output : = [10.0,30.0,40.5,50.0,100.0]**

## 2. linspace() Function

The linspace() function is used to create an array with evenly spaced points between a starting and ending point. The following examples demonstrate the use of linspace() function.

Syntax- `linspace(start, stop, N)`

- Start represents the starting number.
- Stop represents the ending element.
- N represents the number of parts the elements should be divided

e.g.-

```
import numpy as np  
a = np.linspace(1,10,10)  
print(a)
```

Output : [ 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]

### 3. arange() Function

The arange() function in numpy is same as range() function in Python. The following format is used to create an array using the arange() function.

Syntax-

`arange(start,stop,stepsize)`

`arange(10)` - will create an array with values `[0,1,2,3,4,5,6,7,8,9]`

`arange(5,10)` - will create an array with values `[5, 6,7,8,9]`

`arange(10,1,-1)` will create an array with values `[10,9,8,7,6,5,4,3,2]`

Example

```
import numpy as np
a = np.arange(10)
b = np.arange(5,10)
c = np.arange(10,1,-1)
print(a)
print(b)
print(c)
```

Output-

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

`[5,6,7,8,9]`

`[10, 9, 8, 7, 6, 5, 4, 3, 2]`

We can use zeros() function to create an array with all zeros. The ones() function will is useful to create an array with all 1s. They are written in the following format-

```
zeros(n,datatype  
)
```

```
ones(n,datatype)
```

Note : if datatype is missing then the default value will be float.

Example 1

zeros(5) will create an array with five zero values.

ones(5) will create an array with five 1 values.

Example

```
import numpy as np
```

```
K = np.zeros(5)
```

```
R = np.ones(5)
```

```
print(K)
```

```
print(R)
```

Output :

```
[0.,0.,0.,0.,0.]
```

```
[1.,1.,1.,1.,1.]
```



## Mathematical Operations on Arrays

It is possible to perform various Mathematical operations like addition, subtraction, division etc. on the elements of any arrays. The functions of **math** module can be applied to the elements of any array.

Example

```
import numpy as np
K = np.array([10, 20, 30, 40,50])
K = K+5      —————→ Add 5 to the Array
print(k)
K = K-5      —————→ Subtract 5 from each value of Array
print(k)
K = K*5      —————→ Multiply array by 5
print(k)
K = K/5      —————→ divide Array by 5
print(k)
```

Output-

```
[15 25 35 45 55]
[10 20 30 40 50]
[ 50 100 150 200 250]
[10. 20. 30. 40. 50.]
```

## Aliasing the Arrays

Aliasing in arrays does not make any new copy of the array defined earlier. It means new array created only reference to the array k.

The following example demonstrates the use of aliasing.

```
import numpy as np
k = np.array ([3, 5, 6,7,8])
print(k)
h = k  —————→ Give another name h to array k
print(h)
print(k)
k[0] = 45
print(h)

print(k)
```

Output-

```
[3 5 6 7 8]
[3 5 6 7 8]
[3 5 6 7 8]
[45 5 6 7 8]
[45 5 6 7 8]
```

## copy() method

The copy() method is used to copy the contents of one array to another. The following function demonstrates the use of the copy method.

```
import numpy as np
```

```
k = np.array([3,5,6,7,8])
```

```
print(k)
```

```
h = k.copy() —————→ Create a copy of array k and call it h
```

```
print(h)
```

```
print(k)
```

```
k[0] = 45
```

```
print(h)
```

```
print(k)
```

Output-

```
[3 5 6 7 8]
```

```
[3 5 6 7 8]
```

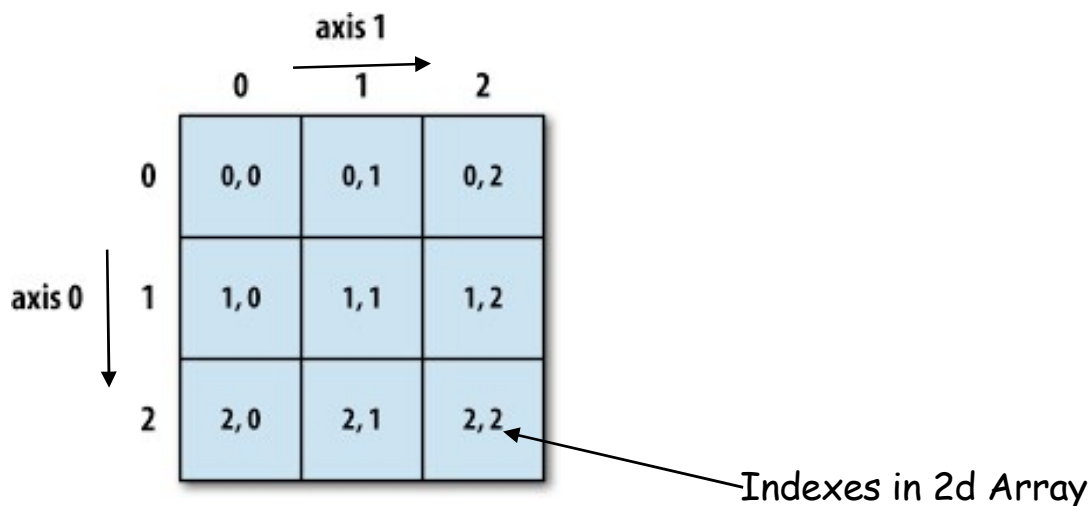
```
[3 5 6 7 8]
```

```
[3 5 6 7 8]
```

```
[45 5 6 7 8]
```

## 2-Dimensional Arrays in Numpy

2D Array-The dimension of an array represents the arrangement of elements in the array. If the elements are arranged horizontally, it is called the row and if the elements are arranged vertically, then it is called the column. When they contain only one row and one column of elements, it is called the Single dimensional array or one dimensional array. When an array contains more than one row and more than one column of elements, it is called the two dimensional array or 2-D array. The following example is used demonstrate how to declare the two dimensional array in using numpy.



2D Array structure

## Let us create a 2D array in numpy-

```
import numpy as np
```

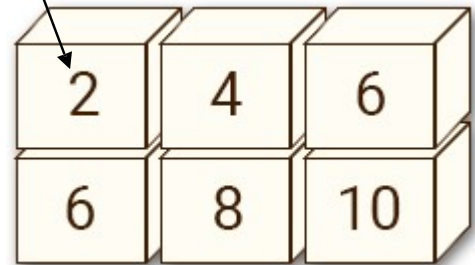
```
x = np.array([[2,4,6],[6,8,10]])
```

```
Print(x)
```

Output-

```
[[2 4 6]
 [6 8 10]]
```

2DArray elements



## ndim Attribute

ndim attribute is used to represent the number of dimensions of axes of the array. The number of dimensions is also known as 'rank'. The following example demonstrate the use of the ndim attribute

```
import numpy as np
```

```
A = np.array([5,6,7,8])
```

```
R = np.array([[4,5,6],[7,8,9]])
```

```
print(A.ndim)      —————>    Number of rows in array A
```

```
print(R.ndim)      —————>    Number of rows in array R
```

Answer :

1

2

## shape attribute

The 'shape' attribute gives the shape of an array. The shape is tuple listing the number of elements along each dimension. A dimension is called an axis. For one dimensional array it will display a single value and for two-dimensional array it will display two values separated by commas represent rows and columns.

For Example

```
import numpy as np
```

```
k = np.array([1,2,3,4,5])
```

```
print(k.shape)      →    Number of elements in array k
```

```
d = np.array([[5,6,7],[7,8,9]])
```

```
print(d.shape)      →    Number of rows and Columns in array d
```

Output-

```
(5,)
```

```
(2, 3)
```

## size Attribute

The size attribute gives the total number of elements in the array.  
For e.g.

```
import numpy as np  
a1 = np.array([1,2,3,4,5])  
print(a1.size) will result 5
```

In case of two dimensional array the result will be total rows\* total columns

```
import numpy as np  
k = np.array([[5,6,7],[7,8,9]])  
print(k.size) will result 6
```

## itemsizes Attribute

The itemsizes attribute gives the memory size of array elements in bytes. For e.g.

```
import numpy as np
```

```
a1 = np.array([1, 2, 3, 4, 5])
```

```
print(a1.itemsize) → Give memory size of array elements in bytes
```

Output-

4

## reshape() Method

The reshape() method is useful to change the shape of an array. The new array should have the same number of elements as in the original array. For e.g.

```
import numpy as np
```

```
d = np.array([[4, 5, 6, 7], [5, 6, 7, 8], [7, 8, 9, 6]])
```

```
print(d)
```

```
d = d.reshape(6, 2) → Reshape array d to 6 rows and 2 columns
```



```
print(d)
```

```
d = d.reshape(1,12)
```

Reshape array d to 1row and 12 columns

```
print(d)
```

```
d = d.reshape(12,1)
```

Reshape array d to 12rows and 1 column

```
print(d)
```

Output-

```
[[4 5 6 7]
```

```
 [5 6 7 8]
```

```
 [7 8 9 6]]
```

```
[[4 5]
```

```
 [6 7]
```

```
 [5 6]
```

```
 [7 8]
```

```
 [7 8]
```

```
 [9 6]]
```

```
[[4 5 6 7 5 6 7 8 7 8 9 6]]
```

```
[[4]
```

```
 [5]
```

```
 [6]
```

```
 [7]
```

```
 [5]
```

```
 [6]
```

```
 [7]
```

```
 [8]
```

```
 [7]
```

```
 [8]
```

```
 [9]
```

```
 [6]]
```

## empty() function

This function is used to create the empty array or an uninitialized array of specified data types and shape.

For e.g.

```
import numpy as np

x = np.empty([3,2], dtype = int)
y = np.empty([4,4], dtype = float)

print(x)

print(y)
```

Output-




```
[[0 0]
 [0 0]
 [0 0]]
[[6.23042070e-307 4.67296746e-307 1.69121096e-306 8.45593934e-307]
 [6.23058028e-307 2.22522597e-306 1.33511969e-306 1.37962320e-306]
 [9.34604358e-307 9.79101082e-307 1.78020576e-306 1.69119873e-306]
 [2.22522868e-306 1.24611809e-306 8.06632139e-308 2.29178686e-312]]
```

## Indexing in 2-D dimension array

Index represents the location number. The individual elements of an array can be accessed by specifying the location number of the row and column of the array element as follow-

$A[0][0]$  => represents 0<sup>th</sup> row and 0<sup>th</sup> column element in array A

$A[1][3]$  => represents 1<sup>st</sup> row and 3<sup>rd</sup> column element in the array A

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

## Slicing in 1D Array

Syntax-

- Arrayname[start:stop:stepsize]
- The default value of stepsize is "1"

-5	-4	-3	-2	-1
6	7	8	9	23
0	1	2	3	4

`A[:5]` will give `[6 7 8 9 23]`

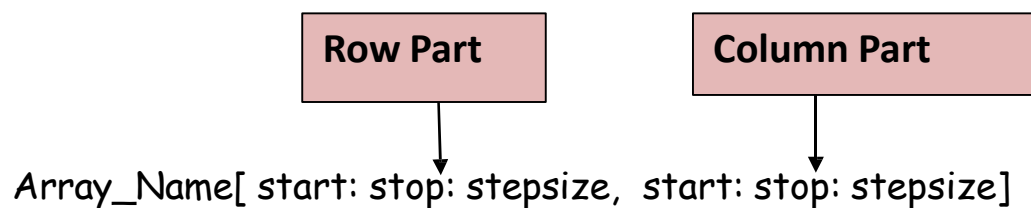
`A[::2]` will give `[6 8 23]`

`A[-1:-5:-1]` will give `[23 9 8 7]`

`A[2:-2]` will give `[8]`

## Slicing in 2-D Array

A slice represents a part or piece of the array.



0<sup>th</sup> row to 1<sup>st</sup> row, 0<sup>th</sup> column to 2<sup>nd</sup> column

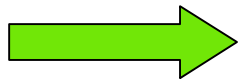
A[0:2, 0:3]

Or

A[:2, :3]

2<sup>ND</sup> row to 3<sup>RD</sup> row,  
3<sup>RD</sup> column to 4<sup>TH</sup> column

A[2:4, 3:]



11	2	3	56	14
40	52	16	12	20
70	8	9	32	22
18	30	17	44	49
25	55	66	78	82

Top 2 rows and Right 3 columns

A[0:2, 2:]

Or

A[:2, 2:]

Lower 3 rows and right 2 columns

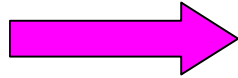
A[2:, 3:]



11	2	3	56	14
40	52	16	12	20
70	8	9	32	22
18	30	17	44	49
25	55	66	78	82

0<sup>th</sup> row and 0<sup>th</sup> column  
element

A[0:1, 0:1]



11	2	3	56	14
40	52	16	12	20
70	8	9	32	22
18	30	17	44	49
25	55	66	78	82

2<sup>nd</sup> row and 1<sup>th</sup> column  
element

A[2:3, 1:2]



0<sup>th</sup> row and 4<sup>th</sup> row as (0+4=4), 0<sup>th</sup> column and 3<sup>rd</sup> column as (0+3=3)

A[:, :4, :, :3]



11	2	3	56	14
40	52	16	12	20
70	8	9	32	22
18	30	17	44	49
25	55	66	78	82

Negative  
Index



-5

-4

-3

-2

-1



-5

-4

-3

-2

-1

11	2	3	56	14
40	52	16	12	20
70	8	9	32	22
18	30	17	44	49
25	55	66	78	82

(-2th row ),  
(-5<sup>th</sup> column  
and -3<sup>rd</sup>  
column, -1<sup>st</sup>  
column)

A[-2:-3, -5::2]



18

30

17

44

49

## eye() or identity() Function

The `eye()` function creates a 2D array and fills the elements in the diagonal with 1s.

Syntax-`eye(n, dtype=datatype)`

This function will create an array with `n` rows and `n` columns with diagonal elements as 1s. The default data type is float.

e-g-

```
import numpy
a=numpy.eye(3)
print(a)
```

output-

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```



## zeros() function in 2D array

This function is used to create two dimensional array with the 0 as default value and default data type is float.

```
import numpy
```

```
Q = numpy.zeros([3,2], dtype = int)
```

```
Z = numpy.zeros([4,4], dtype = float)
```

```
print(Q)
```

```
print(Z)
```

Output-

```
[[0 0]
```

```
 [0 0]
```

```
 [0 0]]
```

```
[[0. 0. 0. 0.]
```

```
 [0. 0. 0. 0.]
```

```
 [0. 0. 0. 0.]
```

```
 [0. 0. 0. 0.]]
```

## ones() function in 2D array

This function will be used to create the array with 1 as default value for each of individual defined element.

```
import numpy
```

```
Q = numpy.ones([3,2], dtype = int)
```

```
Z = numpy.ones([4,4], dtype = float)
```

```
print(Q)
```

```
print(Z)
```

Output-

```
[[1 1]
```

```
 [1 1]
```

```
 [1 1]]
```

```
[[1. 1. 1. 1.]
```

```
 [1. 1. 1. 1.]
```

```
 [1. 1. 1. 1.]
```

```
 [1. 1. 1. 1.]]
```

## Joins in Array

We can join array in numpy by following method-

1. Concatenate()
2. hstack()
3. vstack()

## 1. concatenate()

concatenate()- is used to join more than one array but the array must be of the same shape.

e.g.-

```
import numpy as np
a=np.array([2,3,4,50])
b=np.array([8, 9,10,11,15])
c=np.concatenate([a,b])
print (c)
```

```
import numpy as np
```

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

```
b=np.concatenate([a,a],axis=1) → concatenate array a  
with array a column wise
```

```
print (b)
```

Output-

```
[ [2 3 4 2 3 4]
```

```
 [4 5 6 4 5 6]
```

```
 [7 8 9 7 8 9]]
```

E.g-2

```
import numpy as np
```

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

```
b=np.concatenate([a,a],axis=0) → concatenate array a
```

with array a row wise

```
print (b)
```

Output-

[ [2 3 4]

[4 5 6]

[7 8 9]

[2 3 4]

[4 5 6]

[7 8 9]]

## 2.hstack()

hstack() - It is used to join more than one array horizontally or row wise.

e.g.-

```
import numpy as np
a=np.array([1,2,3])
b=np.array([10,11,12])
c=np.hstack((a,b))
print (c)
```

Output-

```
[1  2  3 10 11 12]
```

### 3.vstack()

vstack() - It is used to join more than one array vertically or column wise.

e.g.-

```
import numpy as np  
a=np.array([1,2,3])  
b=np.array([10,11,12])  
c=np.vstack((a,b))  
print (c)
```

Output-

```
[[1  2  3 ]
```

```
 [10 11 12]]
```

### Array subsets

We can get subsets of a numpy array by using any of the following-

1. split()
2. hsplit()
3. vsplit()

## split()

It is used to split a numpy array into equal or unequal parts.

```
import numpy as np
```

```
x = [1, 2, 3, 99, 99, 3, 2, 1]
```

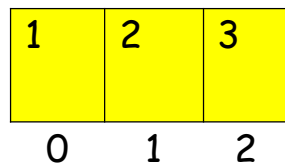
```
x1, x2, x3 = np.split(x, [3, 5])
```

split array into 3 subsets like-

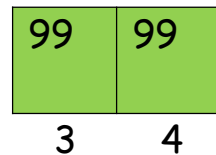
[0 : 3], [3 : 5] and [5 : ]

```
print(x1, x2, x3)
```

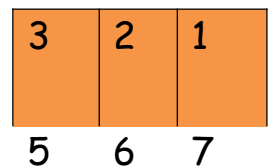
Output-[1 2 3] [99 99] [3 2 1]



1	2	3
0	1	2



99	99
3	4



3	2	1
5	6	7

## hsplit()

It is used to provide the subsets of an array after splitting it horizontally.

```
import numpy as np  
a= np.arange(16).reshape((4, 4))  
  
print( a)
```

Output-

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

```
       [ 4,  5,  6,  7],
```

```
       [ 8,  9, 10, 11],
```

```
       [12, 13, 14, 15]])
```

e.g.-

```
left, right = np.hsplit(a, 2) →
```

```
print(left)
```

```
print(right)
```

left

right

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Output-

```
[[ 0 1]
```

```
[ 4 5]
```

```
[ 8 9]
```

```
[12 13]]
```

```
[[ 2 3]
```

```
[ 6 7]
```

```
[10 11]
```

```
[14 15]]
```

## vsplit()

It is used to provide the subsets of an array after splitting it vertically.

Example:-

```
import numpy as np
```

```
a= np.arange(16).reshape((4, 4))
```

```
print (a)
```

Output-

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

*e.g.-*

```
top, bottom = np.vsplit(a, 2)
```

```
print(top)
```

```
print(bottom)
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Output-

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

```
 [4 5 6 7]]
```

```
[[ 8  9 10 11]
```

```
 [12 13 14 15]]
```

## Statistical Function in Numpy

Functions	Description
<code>np.mean()</code>	Compute the arithmetic mean along the specified axis.
<code>np.std()</code>	Compute the standard deviation along the specified axis.
<code>np.var()</code>	Compute the variance along the specified axis.
<code>np.sum()</code>	Sum of array elements over a given axis.
<code>np.prod()</code>	Return the product of array elements over a given axis.
<code>np.cumsum()</code>	Return the cumulative sum of the elements along a given axis.
<code>np.cumprod()</code>	Return the cumulative product of elements along a given axis.
<code>np.min()</code> , <code>np.max()</code>	Return the minimum / maximum of an array or minimum along an axis.

```
import numpy as np

array1 = np.array([[10, 20, 30], [40, 50, 60]])

print("Mean: ", np.mean(array1))

print("Std: ", np.std(array1))

print("Var: ", np.var(array1))

print("Sum: ", np.sum(array1))

print("Prod: ", np.prod(array1))
```

Sample output of above program.

```
Mean: 35.0

Std: 17.07825127659933

Var: 291.6666666666667

Sum: 210

Prod: 720000000
```

**Covariance**- Covariance is a measure of how two variables vary together (like the height of a person and the weight of a person in a population).

The covariance  $\sigma(x,y)$  of two random variables  $x$  and  $y$  is given by with  $n$  samples.

$$\sigma(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

e.g.-

```
import numpy as np
```

```
x = np.array([0,1,2])
```

```
y = np.array([2,1,0])
```

```
print("\nOriginal array1:")
```

```
print(x)
```

```
print("\nOriginal array2:")
```

```
print(y)
```

```
print("\nCovariance matrix of the said arrays:\n", np.cov(x, y))
```

Output-

Original array1:

[0 1 2]

Original array2:

[2 1 0]

Covariance matrix of the said arrays:

[ [ 1. - 1. ]

[ -1. 1. ] ]

## Basic Statistical Method for Understanding Data

### Mean / Average

Mean or Average is a central tendency of the data i.e. a number around which a whole data is spread out. In a way, it is a single number which can estimate the value of whole data set.

Let's calculate mean of the data set having 8 integers.

$$x = \frac{12+24+41+51+67+67+85+99}{8} = 55.75$$

### Median

Median is the value which divides the data in 2 equal parts i.e. number of terms on right side of it is same as number of terms on left side of it when data is arranged in either **ascending or descending order**.

Median will be a middle term, if number of terms is odd

Median will be average of middle 2 terms, if number of terms is even.

### Mode

Mode is the term appearing maximum time in data set i.e. term that has highest frequency.

### **Standard deviation**

Standard deviation is the measurement of average distance between each quantity and mean. That is, how data is spread out from mean. A low standard deviation indicates that the data points tend to be close to the mean of the data set, while a high standard deviation indicates that the data points are spread out over a wider range of values.

When we are asked to find SD of some part of a population, a segment of population; then we use sample Standard Deviation.

$$\text{S.D.} = \sqrt{\frac{1}{n-1} \sum_{i=0}^n (x - \bar{x})^2}$$

where  $\bar{x}$  is mean of a sample.

### **Variance**

Variance is a square of average distance between each quantity and mean. That is it is square of standard deviation.

$$\text{Variance} = (\text{S.D.})^2$$