Numpy :-

⇒ Numpy provides efficient storage.

It provides an Array Data Structure, that is dif. from native Python List in terms that in a Numpy Array, data of only same data type can be stored.

⇒ Access to data in Numpy Array is faster than Python List.

⇒ In Numpy, we can use as much memory as we need. Thus, it gives a better control over memory management.

⇒ Importing :-

```
import numpy as np
```

⇒ Creating array :-

```
my_arr = np.array([3, 4, 14])
```

we here pass any sequence (ex:- list) that we want to get converted to array.

⇒ Setting datatype of array :-

⇒ Pass the datatype explicitly in the 2nd param of the np.array() method.

```
myarr = np.array([44,3,2], np.int8)
```

np.int8 : Each element of array will be of integer datatype, occupying 8 bytes each.

Options available :- np.int32 , np.int64

⇒ Accessing Array Elements :-

⇒ Elements are accessed using indexes in same way as in lists.

[my_arr[2] ⇒ 3rd element.

⇒ Array elements can be changed dynamically by assigning new value to any index :-

[my_arr[2] = 45 ⇒ 2nd index value will be changed to 45.

⇒ Creating & Accessing 2D Array :-

```
arr_2d = np.array([[1,4,3],[2,4,1]])
print(arr_2d[1][0])
```

o/p

2

⇒ Getting Dimension of Array :-

```
arr = np.array([[4,2,3,1],[2,1,1,-4]])
arr.shape
```

o/p ⟺ means 2 rows, 4 columns

(2, 4)

shape is a member variable of the NP array object that contains a tuple, in which from right-to-left the number denotes dimension of array, and size of each level of array.

≥ Checking DataType of Array Elements :-

≥ dtype member variable of Numpy array object contains datatype of the array elements.

```
print(my_arr.dtype)
```

o/p

```
int32
```

≥ Getting No. of Elements of an Array :-

```
my_arr.size
```

o/p

```
5
```

"size" member variable contains an integer equal to the no. of elements in the array.

When we create an Array from a Python SET or DICTIONARY, then array of 'object' datatype will be formed.
Entire SET/DICTIONARY will be at 0th index of the numpy array.

Intrinsic Array Creation Objects:

1) Creating array of 0's [zero]:-

```
zero_arr = np.zeros((2,6))
print(zero_arr)
```

Here, we pass a tuple that contains the dimension of the array that we want.

op

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

⇒ zeros method of the Numpy array object is used.

Syntax

```
np.zeros(<shape>, <datatype>)
```

numpy module datatypes are supported.
Optional parameter :- "float64" is the default datatype.

2> Creating Array from a Range of Integers : arange() method :-

Syntax :-

```
range_arr = np.arange(beg, end, steps)
```

beg — compulsory

steps — optional

"end" is excluded.

=> Takes the same parameter as the Python range() function, and creates a 1-D array out of the integers satisfying the parameter range.

3> linspace() method :- Getting equally spaced floating numbers :-

Syntax :-

```
arr = np.linspace(beg, end, no_of_elements)
```

end — inclusive

Ex:-

```
arr = np.linspace(2, 4, 9)
print(arr)
```

o/p

```
[2.  , 2.25, 2.50, 2.75, 3.00,
 3.25, 3.50, 3.75, 4.00]
```

⇒ Gives an array containing floats equal to the 3rd param, that are equally spaced with lower & upper bound as 1st and 2nd param, respec, both inclusive.

⇒ empty() method :- Getting an empty array of fixed dimensions :-

```
em_arr = np.empty((2,6))
```

Gives an empty array having random elements of the dimensions passed in the parameter.

5> Creating Identity Matrix :- identity()

```
id_arr = np.identity(3)
```

op

```
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1]]
```

will give a 3x3 identity matrix.

Reshaping an existing Array : reshape() method :-

```
arr = np.arange(10)
re_arr = arr.reshape((2,5))
```

Reshapes caller obj array into an array of dimension passed in the parameter.

⇒ Dimension passed in reshape() must fit all the elements in the original array exactly (not more or less), else will throw an error.

⇒ Converting Multi-D array to 1D array : ravel()

```
arr = np.array([[4,2,6],[1,8,7]])

re_arr = arr.ravel()
print(re_arr)
```

o/p

```
[4,2,6,1,8,7]
```

32:10

⇒ NUMPY AXIS :-

⇒ Dimension of an array is called Axis.

Counting of Axis starts from 0.

Sample Array

AXES (plural)

① 1D → [1, 2, 3, 4, 5] → 1 [Axis 0]

② 2D → [[1, 2, 3, 4, 5],
[6, 7, 8, 9, 10],
[1, 9, 5, 4, 2]] → 2 [Axis 0, Axis 1]

⇒ The lowest level of array will be at the highest axis, i.e, at Axis (n-1) for an n-Dimensional Array.

↓

For ex:- For a 2D array.

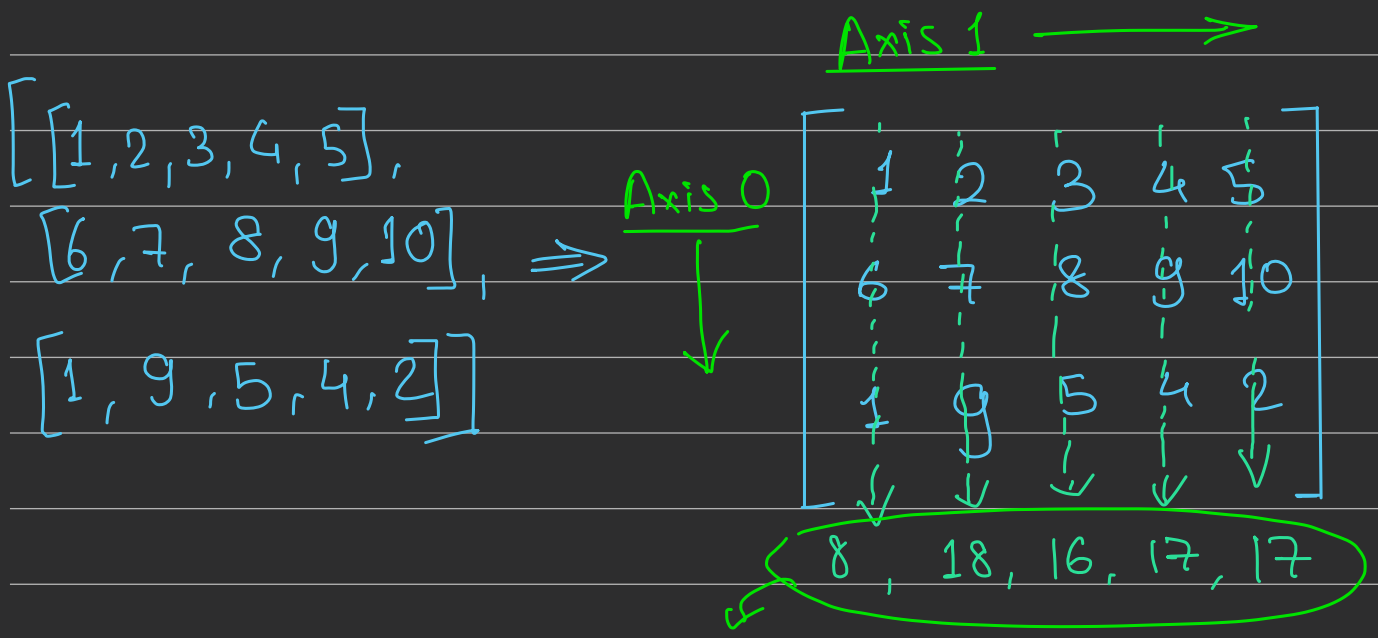
Row → Axis 0 → (elements of this axis array contains array elements)

Column → Axis 1 (lowest level, containing data of other datatype)

Suppose app is a 2D array :-

app [1] [4]

Axis 0 Axis 1.



Sum on Axis 0.

This means, elements intersecting Axis 0 will be summed up.

Operation on Any Axis :-

Those elements will be operated which intersects that particular axis. (being given)

For a 2D Array :-

Operation on Axis 0 :- ↓ ↓ ↓ ↓

Op. on axis 1 :- → → → →

Operation on Axis :-

sum(), max(), min()

Ex:-

```
n = [[1,2,3],[4,5,6],[7,1,0]]
ar = np.array(n)
```

`n.sum(axis=0)` ⟹ `[12, 8, 9]`

sum along axis 0

`n.sum(axis=1)` ⟹ `[6, 15, 8]`

sum along axis 1.

If no parameter is passed, performs the operation will all the elements of the array.

Getting Transpose of Array :-

I attribute / member variable of the array object contains transpose matrix of the numpy array.

```
arr = np.array([1,4,2],[3,5,6])
```

$$\begin{bmatrix} 1 & 4 & 2 \\ 3 & 5 & 6 \end{bmatrix}$$

```
print(arr.T)
```

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \\ 2 & 6 \end{bmatrix}$$

✳ Looping through all items in any Dimensional Array :-

- **flat** attribute of Np array contains an iterator that gives all the elements of an array sequentially (even without handling indexes using variable)

```
arr = np.array([[1,2,3],[4,5,6]])
for i in arr.flat:
    print(i)
```

o/p

1
2
3
4
5
6

⊛ ndim attribute - contains no. of dimensions :-

`arr.ndim` → o/p `2` as it is a 2-D array.

⊛ nbytes attribute :-

⇒ Contains the size of array in terms of number of bytes it occupies.

`arr.nbytes`

⊛ sqrt() Numpy function :-

`np.sqrt(arr)`

Returns an array in which each element is a square root of the elements in the array passed in the parameter.

✱ argmax() & argmin() methods :-

⇒ Returns the index of that element which is maximum, minimum respectively.

```
arr = np.array([8, 2, 63 4, 17, 24, 1, 19])
print (arr.argmax())
print (arr.argmin())
```

0 1 2 3 4 5 6

o/p

2

5

✱ When applied to 2D or multi-dim array (without any parameter), then it ravel()'s the array and then gives the index of min/max element.

✱ Applied on multi-dimensional array with axis parameter :-

⇒ When applied on multi-dimensional array by applying an axis parameter, then it returns an array of dimension n-1 (n being dimension of original array), with each element giving min/max elements' index along the axis specified.

Ex:

```
arr = np.array([
        [1, 2, 3],
        [4, 5, 6],
        [7, 1, 0]
    ])
arr.argmax(axis=0)
```

[2, 1, 1] → index of max elements across axis 0

op

2	1	1

Here, vertically the numbers are considered as array, as axis is set to 0, and in that each index of max element is given in the o/p array.

⊛ argsort() method :-

⇒ Returns an array which contains the indexes of original array, in a sequence of ascending sort of the original array.

```
arr = np.array([4, 1, 3, 7])
print (arr.argsort())
```

indexes: 0 1 2 3

op: [1, 2, 0, 3]

indexes of elements in sorted order.

⇒ argsort() can also be used for multi-dim array by passing axis parameter, and along that axis, we will get index of elements in sorted order.

MATRIX Operations:

⇒ In Numpy, we can take multiple 2D array, and can perform matrix operations on them.

```
arr1 = np.array([[...],[...],[...]])
arr2 =      "

print (arr1 + arr2)  → addition
print (arr1 - arr2)  → subtraction

print (arr1 * arr2)  → gives element by element multiplication, & not matrix multiplication.
```

Numpy Methods :-

1) np.sqrt().

2) np.where()

any condition.

syntax:- `np.where(arr_name > 5)`

Returns a tuple of indexes, where the elements matching the particular condition occurs.

3) np.count_nonzero(<arr>)

Returns the no. of non-zero elements in the array.

Converting Numpy array to List :-

`np_arr.tolist()`

converts numpy array to Python list and returns it.