What is numpy?

- Is a scientific computing package used for mathematics operations.
- Provides a multidimensional array object.

What is an array?

- Is a grid of values and it contains information about the raw data.

What is dimension in array?

- Tells you the number of axes in the array. Like 1D, 2D, 3D.

Creating Numpy Arrays

```
import numpy as np        #import the numpy package and use the alias np.

#create 1-dimension array.
_1D= np.array([1,2,3])
_1D

array([1, 2, 3])

#create 2-dimension array.
_2D= np.array([[1,0,6], [5,8,7]])
_2D

array([[1, 0, 6],
       [5, 8, 7]])

#create 3-dimension array.
_3D= np.array([[[1,2,3]],[[4,5,6]],[[7,8,9]]])
_3D

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

       [[4, 5, 6]],

       [[7, 8, 9]]])
```

Check dimension using ndim function

```
_1D= np.array( [1,2,3] )
_1D.ndim

1

_2D= np.array([[1,0,6], [5,8,7]])
_2D.ndim

2
```

```
_3D= np.array([[[1,2,3]],[[4,5,6]],[[7,8,9]]])
_3D.ndim
```

```
3
```

```
#create array with 10 dimension
x=np.array([1,2,2,4], ndmin=10)
x
```

```
array([[[[[[[[[[1, 2, 2, 4]]]]]]]]]])
```

Special Numpy array

```
#create an array with 7 zeros.
ar_zero= np.zeros(7)
ar_zero
```

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

```
#create an 0 elements array with 3 rows and 4 columns.
ar_zero1= np.zeros((3,4)) #create an 0 elements array with 3 rows
and 4 columns.
ar_zero1
```

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

```
#create an array with 4 ones
ar_one= np.ones(4)
ar_one
```

```
array([1., 1., 1., 1.])
```

```
#create an array with 0 to 3 elements
ar_rn= np.arange(4)
ar_rn
```

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

```
#create an diagonal array with 1 in the diagonal
ar_dig= np.eye(3)
ar_dig
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
#Array with value, spaced linearly in a specific interval
ar_lin= np.linspace(1,10, num=5)
ar_lin
```

```
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

Create numpy array with random numbers

```
# rand() : Generate random value between 0 to 1.
var1= np.random.rand(2,5)
var1
```

```
array([[0.37447722, 0.57985748, 0.15500739, 0.52435939, 0.66825833],
       [0.72831378, 0.75411082, 0.58616151, 0.58332412,
0.98819327]])
```

```
# randn() : Generate random value close to zero. May return positive
or negative.
var2= np.random.randn(5)
var2
```

```
array([ 1.00627015, -0.9312942 ,  0.98867271,  0.79262174, -
0.03312416])
```

```
# ranf() : Do random sampling, return an array of specific shape and
fill it with random float in the half open interval [0.0, 1.0)
var3= np.random.ranf(4)
var3
```

```
array([0.70095396, 0.0143614 , 0.99869006, 0.92094161])
```

```
# randit() : Generate random numbers between a given range.
var4= np.random.randint(5,20,5) #create 5 elements array with min
value 5 and max 20.
var4
```

```
array([11,  5, 19,  5,  8], dtype=int32)
```

Shape and reshaping

Shape means how many rows and columns in an array.

```
# 1D array with 4 columns
var1= np.array([1,2,3,4])
var1.shape
```

```
(4,)
```

```
# 2D array with 2 rows and 2 columns
var= np.array ([[1,2],[1,2]])
var.shape
```

```
(2, 2)
```

Reshape means change the no. of rows and columns.

```
# reshaping the 6 columns array into 2 rows and 3 columns.
var= np.array([1,2,5,4,7,9])
var.reshape(2,3)
```

```
array([[1, 2, 5],
       [4, 7, 9]])
```

Arithmetic operations

Addition

```
#add 3 in every element of an array.
x= np.array([1,2,3,4])
y=x+3
y
```

```
array([4, 5, 6, 7])
```

```
# add corresponding elements to each other
var1= np.array ([1,2,3,4])
var2= np.array ([1,2,3,4])
np.add(var1,var2)
```

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

Subtraction

```
#subtract 3 in every element of an array.
x= np.array([1,2,3,4])
y=x-3
y
```

```
array([-2, -1,  0,  1])
```

```
# subtract corresponding elements to each other
var1= np.array ([1,2,3,4])
var2= np.array ([1,2,3,4])
np.subtract(var1,var2)
```

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

Multiplication

```
 #multiply 3 in every element of an array.
x= np.array([1,2,3,4])
y=x*3
y
```

```
array([ 3,  6,  9, 12])
```

```
# multiply corresponding elements to each other
var1= np.array ([1,2,3,4])
var2= np.array ([1,2,3,4])
np.multiply(var1,var2)
```

```
array([ 1,  4,  9, 16])
```

Divide

```
#divide every element with 3.
x= np.array([1,2,3,4])
y=x/3 #divide every element with 3.
y
```

```
array([0.33333333, 0.66666667, 1.        , 1.33333333])
```

```
# divide corresponding elements to each other
var1= np.array ([1,2,3,4])
var2= np.array ([1,2,3,4])
np.divide(var1,var2)
```

```
array([1., 1., 1., 1.])
```

Modulos

```
# divide array with 3 and give remainder.
a= np.array([5,6,7,8])
b=a%3
b
```

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

```
#divide array with each othe and gives remainder.
a= np.array([5,6,7,8])
var1= np.array ([1,2,3,4])
mod= np.mod(a,var1)
mod
```

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

Exponent

```
#give exponent to every element of an array.
x= np.array([1,2,3,4])
y=x**2
y
```

```
array([ 1,  4,  9, 16])
```

```
a= np.array([5,6,7,8])
var1= np.array ([1,2,3,4])
mod= np.power(a,var1)
mod
```

```
array([   5,   36,  343, 4096])
```

Indexing and slicing

- Indexing in NumPy is a way to access individual elements or a group of elements from an array, starting from 0. And negative indexing starts with -1.

```
# return the value of 0 index.
arr = np.array([10, 20, 30, 40, 50])
print(arr[0])
```

```
10
```

2D array indexing

```
# return element of 0th row ana 1st column
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr[0,1])
```

```
2
```

3D Array indexing

```
#return element of 0th depth, 0th row and 0 column.
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0,0,0])
```

```
1
```

- Slicing means to get the particular data set from an array.

```
 #return elements from 1 to 2nd index.
arr = np.array([10, 20, 30, 40, 50])
print(arr[1:3])
```

```
[20 30]
```

2D slicing

```
#Slice the first two rows and first two columns.
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr[:2, :2])
```

```
[[1 2]
 [4 5]]
```

3D slicing

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[:2, :2, :2])
```

```
[[[ 1  2]
  [ 4  5]]

 [[ 7  8]
  [10 11]]]
```

Iterating numpy array

Iterating means do repeating.

```
#iterate every element of an array.
var= np.array([1,2,3,4,5,6])
for i in var:
 print(i)
```

```
1
2
3
4
5
6
```

Iterating Over 2D Arrays

```
#iterate every element.
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
for row in arr_2d:
    for element in row:
        print(element)
```

```
1
2
3
4
5
6
7
8
9
```

```
# iterate through rows
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
for row in arr_2d:
 print(row)
```

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

Iterating Over 3D Arrays

```
#iterate every element.
arr_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for matrix in arr_3d:
    for row in matrix:
        for element in row:
             print(element)
```

```
1
2
3
4
5
6
7
8
```

```
arr_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for matrix in arr_3d:
 print(matrix)
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
```

Join and split

Joining means putting contents of two or more arrays in a single array.

```
var = np.array([1,2,3,4])
var1=np.array([5,6,7,8])
ar=np.concatenate((var, var1))
ar
```

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

2D arrays joining

```
#join array along the row side.
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
result_rows = np.concatenate((arr1, arr2), axis=0)
result_rows
```

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

```
#join array along the column side.
result_cols = np.concatenate((arr1, arr2), axis=1)
result_cols
```

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

3D arrays joining

```
#joining along depth
arr1 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
arr2 = np.array([[[9, 10], [11, 12]], [[13, 14], [15, 16]]])
result_depth = np.concatenate((arr1, arr2), axis=0)
result_depth
```

```
array([[[ 1,  2],
        [ 3,  4]],

       [[ 5,  6],
        [ 7,  8]],

       [[ 9, 10],
        [11, 12]],

       [[13, 14],
        [15, 16]]])
```

```
#joining along row
arr1 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
arr2 = np.array([[[9, 10], [11, 12]], [[13, 14], [15, 16]]])
result_rows = np.concatenate((arr1, arr2), axis=1)
result_rows
```

```
array([[[ 1,  2],
        [ 3,  4],
        [ 9, 10],
        [11, 12]],

       [[ 5,  6],
        [ 7,  8],
        [13, 14],
        [15, 16]]])
```

```
#joining along column
arr1 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
arr2 = np.array([[[9, 10], [11, 12]], [[13, 14], [15, 16]]])
result_columns = np.concatenate((arr1, arr2), axis=2)
result_columns
```

```
array([[[ 1,  2,  9, 10],
        [ 3,  4, 11, 12]],

       [[ 5,  6, 13, 14],
        [ 7,  8, 15, 16]]])
```

Split array:- Break one array into multiple array.

```
#split array into 3
var= np.array([1,2,3,4,5,6])
np.array_split(var,3)
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

2D array split

```
#split from row side
var= np.array([[1,2],[3,4],[5,6]])
np.array_split(var,3,axis=0)
```

```
[array([[1, 2]]), array([[3, 4]]), array([[5, 6]])]
```

```
#split from column side
var= np.array([[1,2],[3,4],[5,6]])
np.array_split(var,3,axis=1)
```

```
[array([[1],
        [3],
        [5]]),
 array([[2],
        [4],
        [6]]),
 array([], shape=(3, 0), dtype=int64)]
```

3D array split

```
#split array along depth into 2 new array
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
np.array_split(arr3,2,axis=0)
```

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

```
#split array along rows into 3 new array
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
np.array_split(arr3,2,axis=1)
```

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

        [[5, 6]]]),
 array([[[3, 4]],
```

```
        [[7, 8]]])]
```

```
#split array along columns into 2 new array
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
np.array_split(arr3,2,axis=2)
```

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

        [[5],
         [7]]]),
 array([[[2],
         [4]],

        [[6],
         [8]]])]
```

Search, sort, search sorted and functions

- Search:- Search an array for a certain value, and return the indexes that get a match.

```
var=np.array([1,2,3,4,5,6,7,8])
x=np.where(var==2)
x
```

```
(array([1]),)
```

- Search sorted array:- Perform a binary search in the array, and return the index where the specific value would be inserted to maintain the search order.

```
var= np.array([1,2,3,4,6,7,8,9,10])
x= np.searchsorted(var, 5, side="right")
x
```

```
np.int64(4)
```

- Sort:- Sort the array in ascending or descending.

```
var=([5,3,4,2,8,1,9,3,5,3,54])
np.sort(var)
```

```
array([ 1,  2,  3,  3,  3,  4,  5,  5,  8,  9, 54])
```

Insert and delete

```
#insert 40 at index 2
var= np.array([1,2,3,4])
v= np.insert(var, 2, 40)
v
```

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

```
#delete value from the 2 index.
var= np.array([1,2,3,4])
```

```
d= np.delete(var, 2)
d
```

```
array([1, 2, 4])
```