NumPy :(pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee)) is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Numpy Array :is a grid of values with same type, and is indexed by a tuple of non negative integers. The number of dimensions of it, is the rank of the array; the shape of an array depends upon a tuple of integers giving the size of the arraya long each dimension.

e.g.program

```
import numpy as np
a=np.array([500,200,300])        #Createa1DArray
print(type(a))                   #Prints"<class'numpy.ndarray'>"
print(a.shape)                   #Prints"(3,)"means size of an array
print(a.rank)                    # prints 1 as it is one dimensional
print(a[0],a[1],a[2])            #Prints"500200300"
a[0]=150                         #Change an element of the array
print(a)                         It will show array as [150 200 300]
p=np.empty(5)      #Create an array of 5elements with random values
p=np.empty([2,3])  #create a array of 2 rows with 3 values in each columns.
a1=np.zeros(5)      #Create an array of al lzeros float values
a2=np.zeros(5,dtype=np.int)     #Create an array of all zeros int values
b=np.ones(5)        #Create an array of all ones
c=np.full(5,7)       print(c)      #Prints"[77777]"
```

```
b = np.array([[1,2,3],[4,5,6]])     # Create a rank 2 array
print(b.shape)                      # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])    # Prints "1 2 4"
```

Creation of 2D array from 1D array We can create 2D array from 1d array using reshape() function.

e.g. program

```
import numpy as np
A = np.array([1,2,3,4,5,6])
B = np.reshape(A, (2, 3))
print(B)                          OUTPUT [[1 2 3],  [4 5 6]]
```

2 D ARRAY SLICES Slicing of numpy 2d array elements is just similar to slicing of list elements with 2 dimension.

```
import numpy as np
A = np.array([[7, 5, 9, 4], [ 7, 6, 8, 8], [ 1, 6, 7, 7]])
print(A[:2, :3]) #print elements of 0,1 rows and 0,1,2 columns
print(A[:3, ::2]) #print elements of 0,1,2 rows and alternate column position
```

print(A[::-1, ::-1]) #print elements in reverse order print(A[:, 0]) #print all elements of 0 column print(A[0, :]) #print all elements of 0 rows print(A[0]) #print all elements of 0 row

2 D ARRAY JOINING e.g.program

```
A = np.array([[7, 5], [1, 6]])                          OUTPUT [[7 5]
                                                               [1 6]
print(np.vstack([A, A]))                                       [7 5]
print(np.concatenate([A, A], axis=1))  [[7 5 7 5]              [1 6]]
                                        [1 6 1 6]]
print(np.concatenate([A, A.T], axis=1))  [[7 5 7 1]   as T is used as Transform, it will transform
Suppose an array is =[[1 2 3]     then transpose of this array will be [[1 4]
                       [4 5 6 ]]                                        [2 5 ]
                                                                        [3 6 ]
x = np.array([1, 2])                                          [[1 2]
print(np.vstack([x, A])) # horizontally stack the arrays       [7 5]
y = np.array([[99], [99]])                                     [1 6]]
print(np.hstack([A, y]))                [[ 7  5 99]
                                         [ 1  6 99]]
```

2 D ARRAY – ARITHMATIC OPERATION : Arithmetic operation over 2d array is possible with add,substract,multiply,divide () functions. E.G.PROGRAM import numpy as np

```
a = np.array([[7, 5, 9], [ 2, 6, 8]])  b = np.array([10,10,10])
c=np.add(a,b) # c=a+b,                        [[17 15 19]
                                               [12 16 18]]
similar print(c) c=np.subtract(a,b) # c=a-b,  [[-3 -5 -1] [-8 -4 -2]]
similar print(c) c=np.multiply(a,b) # c=a*b,  [[70 50 90] [20 60 80]]
similar print(c) c=np.divide(a,b) # c=a/b,    [[0.7 0.5 0.9] [0.2 0.6 0.8]]
```

Some Operations on Array Elements :

```
a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> a == b
Output : array([False,  True, False,  True], dtype=bool)
a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
Output : array([ True,  True,  True, False], dtype=bool)
>>> np.logical_and(a, b)
Output : array([ True, False, False, False], dtype=bool)
x = np.array([[1, 1], [2, 2]])
>>> x
Output : array([[1, 1],[2, 2]])
```

```
>>> x.sum(axis=0)   # columns (first dimension)
Output : array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
Output :  (3, 3)
>>> x.sum(axis=1)   # rows (second dimension)Output : array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
Output :  (2, 4)
```

```
x = np.array([1, 3, 2])
>>> x.argmin()   # index of minimum
Output : 0
>>> x.argmax()   # index of maximum
Output : 1
a = np.array([1, 2, 3, 2])
>>> b = np.array([2, 2, 3, 2])
>>> c = np.array([6, 4, 4, 5])
>>> ((a <= b) & (b <= c)).all()
Output : True
```

```
b = np.array([[1,2,3],[4,5,6]])     # Create a rank 2 array
print(b.shape)                  # Prints "(2, 3)" 2 columns and 3 rows
d = np.eye(2)           # Create a 2x2 identity matrix
print(d)                # Prints "[[ 1.  0.]
                        #          [ 0.  1.]]"
d = np.eye(5) it will create 5X5 matrix with 1 on diagonal positions
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = a[:2, 1:3]
Output : # [[2 3]
         #  [6 7]]
a = np.array([[1,2], [3, 4], [5, 6]])
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
b = np.array([0, 2, 0, 1])
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)
print(bool_idx)        # Prints "[[False False]
                       #          [ True  True]
                       #          [ True  True]]"

print(a[bool_idx])   # Prints "[3 4 5 6]"
print(a[a > 2])      # Prints "[3 4 5 6]"
x = np.array([[1,2],[3,4]])
print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column;
prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
x = np.array([[1,2], [3,4]])
print(x)     # Prints "[[1 2]
             #          [3 4]]"
print(x.T)   # Prints "[[1 3]
             #          [2 4]]"
```

Difference between Numpy array and list

NUMPY ARRAY	LIST
Numpy Array works on homogeneous types	Python list are made for heterogeneous types
Python list support adding and removing of elements	numpy.Array does not support adding and removing of elements
Can't contain elements of different types	can contain elements of different types
smaller memory consumption more memory	better runtime Runtime not

speedy consumption

e.g.program

```
import numpy as np
a = np.array([1, 2, 3]) b = np.array([5, 6]) c=np.concatenate([a,b,a])
print(c)                        #print [1 2 3 5 6 1 2 3]
```

2 D ARRAY – Mathematical Functions Maths functions like power,abs,ceil,floor,around and trigonometric functions like sin,cos,tan,asin etc are supported by numpy

E.G.PROGRAM import numpy as np a = np.array([[7.333, 5.223], [2.572, 6.119]])

```
print(np.power(a,2))        OUTPUT   [[53.772889 27.279729]
                                      [ 6.615184 37.442161]]
print(np.ceil(a))                     [[8. 6.]
                                      [3. 7.]]
print(np.floor(a))                    [[7. 5.]
                                      [2. 6.]]
print(np.around(a,1))                  [[7.3 5.2]
                                      [2.6 6.1]]
```

2 D ARRAY VARIANCE : The average of the squared differences from the Mean. STEPS 1. find the mean of values

2. Subtract value with mean value then square the result, sum all results of each value

3. Find the average of sum value

e.g. program

```
import numpy as np b = np.array([600,470,170,430,300])
print(b.mean()) # print 394.0
print(np.var(b,ddof=0)) # print 21704.0
```

NOTE :-Variance is calculated like (2062 + 762 + (−224)2 + 362 + (−94)2)/5 206=600-394 and so on for other values

COVARIANCE : Covariance is a statistical measure that shows whether two variables are related by measuring how the variables change in relation to each other. FORMULA (A is the Variance of Item 1 Price and B is the Variance of Item 2 Price)

YEAR	ITEM1 PRICE	ITEM 2 PRICE	A	B	AxB
2015	1000	130	-266.667	10	-2666.67
2016	1200	110	-66.6667	-10	666.6667
2017	1600	120	333.3333	10	3333.333
	Mean 1266.666667	Mean 120		Sum of AXB	1333.333
				covariance=	666.6667

Covariance = Sum of (A X B)/n-1 where n is the number of items

COVARIANCE e.g. program

```
import numpy as np a = np.array([1000,1200,1600]) b = np.array([130,110,120])
print(np.cov(a,b,bias=True)[0,1])
OUTPUT -666.6666666666666
```

2 D ARRAY CORRELATION of Coefficient : Correlation is the scaled measure of covariance. Besides, it is dimensionless. In other words, the correlation coefficient is always a pure value and not measured in any units.

Formula

Cov(X,Y) – the covariance between the variables X and Y

σX – the standard deviation of the X-variable

σY – the standard deviation of the Y-variable

Formula

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

import numpy as np a = np.array([1000,1200,1600])

b = np.array([130,110,120])

print(np.corrcoef(a, b))

```
OUTPUT              [[ 1.        -0.32732684]
                     [-0.32732684  1.        ]]
```

Linear Regression : In two sets of data when one is dependent on other then we use Linear Regression instead of Covariance or Coefficient of Correlation, like (x_n, y_n) where there are many values of x and y also y values are dependent on x, finding expected values is Regression, in y= ax +b equation, y is dependent on x, by using x and y values we plot a Regression Line and to check that all the point are near line, for linear regression we use polyfit() function as numpy.polyfit(x,y,deg) here x and y are array and deg – it specifies up to three degree of the fitting polynomial. Polyfit() generated vector of coefficient x=[1 2 3 4 5] and y= [3 4 0 -1 -3] then this numpy.polyfit(x,y,1) will give output as [-1.7,5.7]

Important Concepts :

1. By default Numpy array are created with NUmpy Data Type float

2. Size of Numpy Date types : int32bits – 4 bytes, int64bits-8 bytes, float32 bits- 4 byres, float64bits-8 bytes, string – 1 bytes per character, Boolean – 1 byte

Type(), dtype, itemsize commands are used to find the size of array, list or tuple as – na=np.array([2,5,2,1]) print (type(na) or print(na.dtype) or print(na.itemsize) will give 4 bytes as items are in 32 bits, if we prepare array like na=np.array([2,5,2,1], dtype=np.int64) it will give output on print as 8 bytes, if we use na=np.array([2,5,2,1], dtype=np.float16) it will give output on print as 2.

3. if we create array using dictionary or by using string then we can not access individual elements of array using indexes, as :

A= {1:'A', 2:,'B') then na=np.array(A) or by na=np.array("kendriya")

Print (A[0]) or Print(na[1]) both commands will give error.

If we use fromiter() command then we can make array using dictionary or String, as-

Na=np.fromiter(A,dtype=np.int32) or na=np.fromiter('kendriya' dtype=U2) then we can access element by using index and it will not show error, also fromiter can be used with list or tuple too.

4. linespace() command : it is used to make array like na=np.linespace(2.5,5,6), it will make a array of 6 elements in the range of 2.5 to 5 with appropriate differences.

Print(na) output=[2.5 3 3.5 4 4.5 5]

5. Obtainning subsets of array : For getting subsets we use two functions hsplit() and vsplit() and split() are used for breaking an array in to subsets.

na=np.arange(24.0).reshape(4,6)   it will create a 4X6 matrix taking items from 0 to 23

np.hsplit(na,2) it will split array in two matrixes of 4X3 and 4 X3

np.hsplit(na,3) it will split array in three matrixes of 4X2 and 4 X2

np.hsplit(na,4) it will give error message as 6 columns cannot be divided in 4 equal parts
np.vsplit(na,2) it will split array vertical base in two equal size matrixes of 2X6
(iii) split() – na= np.split([1, 2, 3, 4,5,6,7,8,9,10], [2,6])
it will break array in three parts as 2 = [0:2] and 6 = [2:6] and last parts as [6:]
[0:2] = [1.2], [2:6]=[3,4,5,6] and [6:] = [7,8,9,10]
Print(na) together they will show [1, 2, 3, 4,5,6,7,8,9,10] as output
Lets take example of 4X6 matrix named na of values 0 to 23
Print(np.split(na,[1,4]) it will break it in three sections [0:1], [1:4] and [4:]
It will show two matrixes as [0 1 2 3 4 5] and [[6 7 8 9 10 11]

```
                                                  [12 13 14 15 16 17]
                                                  [18 19 20 21 22 23]
```

a1,a2,a3=np.split(na,[1,4],axis=1) it will make three matrixes splitting column wise values

6. Extracting and compressing condition based Noncontiguous Subsets:

For extracting syntax is numpy.extract(<condition>,<array>), its example is as by taking 4X6 array names na of values from 0 to 23
Print(np.extract(np.mod(na,5)==0) it will give an array of elements [0 5 10 15 20]
If we print np.mod(na,5)==0 it will show 4X6 matrix with values of True and False only
Other mathematical functions which are used in extract commands are –
Around(x,2) – it will show values by rounding up to 2 decimals, others are floor(), ceil(), exp() as exponential, fabs() or absolute()
len(na) will show 4 as total rows but len(na.T) will show 6 as len() function counts rows
For compressing example is – cond1=np.mod(na.arange(4),2)==0
Cond2=np.mod(na.arange(6),2)==0
Np.compress(cond1,na, axis=0) it will show first and third rows element of index 0 and 2
Np.compress(cond2,na, axis=1)
it will show first, third and fifth columns element of index 0 and 2 and 4

Unsolved Application based Questions of Sumita Arora:

Q 2. If L=[3,4,5] and N= [1 2 3] then L*3 = [3,4,5,3,4,5,3,4,5]
N*3 = [1 4 9], L + L = will show [3,4,5,3,4,5] and N+N will show [2 4 6]
Q3 if a= [[2],[4],[6]] and na=np.array(a) will create array of (b) and so on
(a) Size 4 (b) 3 and (c) size 8
Q4. a= np.zeros(6, dtype=np.int) also a[2]=15 and a[4]=25
Q.5 a= np.arange(13,25) np.reshape(a,(3,4))
Q6. (a) [[1 2],[5 6]] (b) [30 60] (c) [80 90] (d) [[3],[7],[11],[15]]
(e) [[2 3],[6 7],[10 11]] (f) [[3],[7],[11]]
Q7. (a) [1 2 3 3 2 1] (b) [[1 2 3], [4 5 6], [1 2 3], [4 5 6]] (c) [1 2 3 1 2 3], [4 5 6 4 5 6]]
Q8. (a) [[1 2 3], [9 8 7], [6 5 4]] (b) [[9 8 7 99], [6 5 4 99]]
Q9. X1= [1 2 3] x2= [99 99] x3= [3 2 1]
Q10. X1,x2,x3=np.split(np.arange(18),[2,5], axis=1)
Q11. P=[[1 2 3 4], [5 6 7 8], [9 10 11 12], [13 14 15 16]] it's a 4X 4 matrix or array
Q= [[2 3 4 5], [6 7 8 9], [10 11 12 13], [14 15 16 17]]
(a) P+10 (b) P*Q (c) Q/7 (d) log(P) (e) around(P) (f) mod(P,7) (g) sqrt(Q)