```
import numpy as np
#basic operations NUMPY
np_sqrt = np.sqrt([2,4,9,16])
print(np_sqrt)   #-> [1.41421356 2.         3.         4.        ]

from numpy import pi
print(np.cos(0)) #-> 1.0

print(np.sin(pi/2)) #-> 1.0

print(np.cos(pi)) #-> -1.0

print(np.floor([1.2,-2.8,6.9,9.8])) #-> [ 1. -3.  6.  9.]

print(np.exp([2,4,9])) #->[7.38905610e+00 5.45981500e+01 8.10308393e+03]


#Shape manipulation of ndarrays using numpy

collection = np.array([[10,15,17,26,13,19],[12,11,21,24,14,23]]) #declare
ndarray
#for flattening the dataset
print(collection.ravel()) #->[10 15 17 26 13 19 12 11 21 24 14 23]

#for reshaping the dataset into 3 rows and 4 columns
print(collection.reshape(3,4)) #->[[10 15 17 26]
                                  #[13 19 12 11]
                                  #[21 24 14 23]]


#for reshaping to 2 rows and 6 columns
print(collection.reshape(2,6)) #->[[10 15 17 26 13 19]
                                 #[12 11 21 24 14 23]]

#for splitting the array into 2
print(np.hsplit(collection,2)) #->[array([[10, 15, 17],
                                #[12, 11, 21]]), array([[26, 13, 19],
                                                       #[24, 14, 23]])]

#now i declare 2 new arrays chocolates and icecreams for stacking both of them
together
chocolates = np.array([10,15,17,26,13,19])
icecreams = np.array([12,11,21,24,14,23])
totalcandies= np.hstack((chocolates,icecreams))
print(totalcandies) #->[10 15 17 26 13 19 12 11 21 24 14 23]

# some more operations on numpy
#i create an array
myarray = np.array([2,3,8,5,6,99,7,9])
```

```
# for creating an array of 3 rows and 3 columns that are all zeros
zeroarray = np.zeros((3,3))
print(zeroarray) #-> [[0. 0. 0.]
                     #[0. 0. 0.]
                     #[0. 0. 0.]]

#similarly an array of ones
onearray = np.ones((3,3))
print(onearray) #->[[1. 1. 1.]
                   #[1. 1. 1.]
                   #[1. 1. 1.]]

#for creating an empty array, numpy automatically fills random values
emparray = np.empty((2,3))
print(emparray) #->[[4.9e-323 7.4e-323 8.4e-323]
                   #[1.3e-322 6.4e-323 9.4e-323]]

#for printing a  array range of 0 to 11
myrange = np.arange(12)
print(myrange) #->[ 0  1  2  3  4  5  6  7  8  9 10 11]

#reshaping the above array
print(myrange.reshape(3,4)) #->[[ 0  1  2  3]
                               #[ 4  5  6  7]
                               #[ 8  9 10 11]]

#linspace(equally spaced) takes 3 arguments firstarg is 1st element 2ndarg is
2nd ele and 3rd arg is no. of elements
exlinspace = np.linspace(1,99,10)
print(exlinspace) #->[ 1.         11.88888889 22.77777778 33.66666667
44.55555556 55.44444444
                                                    #66.33333333
77.22222222 88.11111111 99.        ]


#i create a one dimensional array and convert it into a 2 dimensional array
onedarray = np.array([132,4,24,24,4,5,23,56,34,32,424,43,3,99,98])
twodarray = onedarray.reshape(3,5)
print(twodarray) #->[[132   4  24  24   4]
                    #[  5  23  56  34  32]
                    #[424  43   3  99  98]]

#for creation a 3d array i first create a range of 1darray 27 elements and
then convert to 3darray using 3 arguments in reshape
threedarray = np.arange(27).reshape(3,3,3)
print(threedarray) #->[[[ 0  1  2]
                       #[ 3  4  5]
                       #[ 6  7  8]]
```

```
                      #[[ 9 10 11]
                        #[12 13 14]
                       #[15 16 17]]

                      #[[18 19 20]
                       #[21 22 23]
                       #[24 25 26]]]


#shape and ndim number of dimensions display wiyh a 3d array example
threedithmarray =
np.array([[[0,1,2],[3,4,5],[6,7,8]],[[9,10,11],[12,13,14],[15,16,17]],[[18,19,
20],[21,22,23],[24,25,26]]])
print(threedithmarray.shape)#->(3, 3, 3)
print(threedithmarray.ndim)#->3


#displaying array using x for x in range
newarray = np.array([x for x in range(1,10)])
print(newarray) #->[1 2 3 4 5 6 7 8 9]
#reshaping it into a 2d array
print(newarray.reshape(3,3))#->[[1 2 3]
                              #[4 5 6]
                              #[7 8 9]]

#2nd example for reshaping it into a 3d array
arraynew = np.array([x for x in range(1,13)])
print(arraynew.reshape(2,2,3))#->[[[ 1  2  3]
                                  #[ 4  5  6]]

                                 #[[ 7  8  9]
                                  #[10 11 12]]]

#reshaping again by changing arguments
print(arraynew.reshape(3,2,2))#->[[[ 1  2]
                                  #[ 3  4]]

                                 #[[ 5  6]
                                  #[ 7  8]]

                                 #[[ 9 10]
                                  #[11 12]]]

#when we do not know the 3rd argument use -1 it will automatically detect
print(arraynew.reshape(3,2,-1))

#flattening using -1
print(arraynew.reshape(-1))#->[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

```
#Arithmetic operations NUMPY
#declare arrays for calculation
A = np.array([[2,3,4],[2,4,3]])
B = np.array([8,9,7])
#Addition
print(np.add(A,B))#->[[10 12 11]
                    #[10 13 10]]
#Subtraction
print(np.subtract(B,A))#->[[6 6 3]
                         #[6 5 4]]
#Multiplication
print(np.multiply(A,B))#->[[16 27 28]
                         #[16 36 21]]
#Division
print(np.divide(B,A))#->[[4.         3.         1.75      ]
                       #[4.         2.25       2.33333333]]
#Power either manually pass 2nd arguments or can pass an array also
print(np.power(A,2))#->[[ 4  9 16]
                      #[ 4 16  9]]
print(np.power(A,B))#->[[   256  19683  16384]
                      #[   256 262144   2187]]


#Conditional Statements in numpy(Where and Select)
#Where
#Here if it finds x even it displays even else odd
x = np.array([i for i in range(10)])
print(x)
print(np.where(x%2==0,'Even','Odd'))#->['Even' 'Odd' 'Even' 'Odd' 'Even' 'Odd'
'Even' 'Odd' 'Even' 'Odd']
#Select statement uses three parameters called condition list,choice list and
default
#if condlist matches with array element the choice list will be executed else
default is executed
#here if x<5 then x is squared and if x>5 it is cubed
condlist = [x<5,x>5]
choicelist = [x**2,x**3]
print(np.select(condlist,choicelist,default=x))#->[  0   1   4   9  16   5 216
343 512 729]



#mathematical and statistical functions numpy
arr = np.array([[0,1,2],[3,4,5],[6,7,8]])
print(arr)#->[[0 1 2]
            #[3 4 5]
            #[6 7 8]]
# amin gives the minimum value from the whole array
```

```
print(np.amin(arr))#->0
#amin using axis
#if axis is 0 it gives minimum values vertically and if 1 then horizontally
print(np.amin(arr,axis=0))#->[0 1 2]
print(np.amin(arr,axis=1))#->[0 3 6]

#similarly maximum
print(np.amax(arr))#->8
print(np.amax(arr,axis=0))#->[6 7 8]
print(np.amax(arr,axis=1))#=>[2 5 8]

#similarly Mean
print(np.mean(arr))#->4.0
#median
print(np.median(arr))#->4.0
#standarddeviation
print(np.std(arr))#->2.581988897471611
#variance
print(np.var(arr))#->6.666666666666667
#percentile value
print(np.percentile(arr,50))#->4.0

#Trigonometric sin,cos,tan
deg = np.array([0,30,45,60,90])
print(np.sin(deg*np.pi/180))#->[0.         0.5        0.70710678
0.8660254  1.        ]
print(np.cos(deg*np.pi/180))#->[1.00000000e+00 8.66025404e-01 7.07106781e-01
5.00000000e-01 6.12323400e-17]
print(np.tan(deg*np.pi/180))#->[0.00000000e+00 5.77350269e-01 1.00000000e+00
1.73205081e+00 1.63312394e+16]

#floor and ceil
flrcl = np.array([1.7,8.4,7.5,8.1,4.2,9.7])
print(np.floor(flrcl))#->[1. 8. 7. 8. 4. 9.]
print(np.ceil(flrcl))#->[ 2.  9.  8.  9.  5. 10.]

#indexing
#create a one,two and three dimensional array
array1d = np.array([1,2,3,4,5,6])
array2d = np.array([[1,2,3],[4,5,6]])
array3d = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
#now to get the first element of array1d
print(array1d[0])#->1
print(array2d[0])#->[1 2 3]
print(array3d[0])#->[[1 2 3]
                    #[4 5 6]]
#getting elements from tail
print(array1d[-2])#->5
```

```
print(array2d[1,2])#->6
print(array3d[1,1,2])#->12

#Slicing
#slicing from first element till last of oned array
print(array1d[1:])#->[2 3 4 5 6]
#similarly
print(array1d[3:5])#->[4 5]
#negative parameter from 4th element till last element
print(array1d[-3:-1])#->[4 5]
#slicing 2d arrays
#slicing from the second element of 2d array, and now from the second element
slices from 1st position till last
print(array2d[1,1:])#->[5 6]
print(array2d[-2,-3:-1])#->[1 2]
#slicing 3d arrays
#slicing the first element of 3d array. from that selecting the second row and
slicing from the first position till last
print(array3d[0,1,1:])#->[5 6]
#slicing the second element of 3d array. from that selecting the second row
and slicing only last element
print(array3d[1,1,-1])#->12
```