Task35-TOTAL NUMPY PRACTICE

In [ ]:

```
!pip install numpy
import numpy as np
```

In [ ]:

```
#Numpy:- It is a python library.It stands for numerical python.It is a library consi
'''
Python list:-
> Able to store different data types in the same list
> Storing each item in random location in the memory
> Good for the scenario where list can grow dynamically
> Inbuilt data type
> It has more inbuilt functions
> Appending will take less 0(1) time

Numpy array:-
> Can store only one data type in a array at any time
> Storing each item is sequential which makes array more effective in processing
> Good for the scenario where the items are fixed size and same data time
> Need to install external library numpy
> No extra functions, so it will not more memory store
> Appending elements will take more time in 0(N) time

Dimensions in arrays:-
> Numpy array can be of n dimension
> Lets create array of different dimension
> a = A numpy with one single integer 10
> b = A numpy with passing a list having a list [1,2,3]
> c = A numpy with passing a nested list having [[1,2,3],[4,5,6]] as elements
> d = A numpy with passing a nested list having [[1,2,3][4,5,6][1,2,3][4,5,6]] as el

'''
```

In [9]:

```
#Define a,b,c,d as instructed data
import numpy as np
a=np.array(11)
b=np.array([1,2,3])
c=np.array([[1,2,3],[4,5,6]])
d=np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])
#printing dimension of above data
print('a dimension: ', a.ndim)
print('b dimension: ', b.ndim)
print('c dimension: ', c.ndim)
print('d dimension: ', d.ndim)
```

```
a dimension:  0
b dimension:  1
c dimension:  2
d dimension:  3
```

In [10]:

```
#printing shape of above data
print('shape of a: ', a.shape)
print('shape of b: ', b.shape)
print('shape of c: ', c.shape)
print('shape of d: ', d.shape)
```

```
shape of a:  ()
shape of b:  (3,)
shape of c:  (2, 3)
shape of d:  (2, 2, 3)
```

In [13]:

```
#printing datatypes
print(c.dtype)
print(a.dtype)
print(d.dtype)
```

```
int32
int32
int32
```

In [14]:

```
#printing type of varibles
print(type(a))
print(type(b))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

In [18]:

```
#checking Length of array using Len() function
print(len(b))
len(d)
```

```
3
```

Out[18]:

```
2
```

In [20]:

```
s1=[21,32,24]
s2=[32,65,76]
s3=np.array(s1)
s4=np.array(s2)
print(type(s3))
print(s4)
```

```
<class 'numpy.ndarray'>
[32 65 76]
```

In [25]:

```
#Creating Numpy array
#1.Using arrange()function

print(np.arange(0,10)) # create of numpy array
np.arange(1,11,2) # create with start stop step
```

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

Out[25]:

```
array([1, 3, 5, 7, 9])
```

In [30]:

```
#2.Using Eye function
np.eye(9,dtype=float)  # it is written as 2d array
```

Out[30]:

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

In [32]:

```
#3.Using Zeros function
np.zeros((4,3),dtype=int) #written zeros in a array and deflaut dtype is float
```

Out[32]:

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

In [33]:

```
#4.Using Ones function
np.ones((4,3),dtype=int)
```

Out[33]:

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

In [34]:

```
#5.Using full function
np.full((3,3),9)
```

Out[34]:

```
array([[9, 9, 9],
       [9, 9, 9],
       [9, 9, 9]])
```

In [35]:

```
#6.Using dilag function
x=[1,2,3,4,5,6,7,8,9]
np.diag(x)
```

Out[35]:

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

In [42]:

```
#Numpy random numbers : It generates the numbers  randomly
np.random.rand(1)[0]
```

Out[42]:

```
0.31194228679345015
```

In [51]:

```
np.random.rand(3,4)
```

Out[51]:

```
array([[0.89892763, 0.95058203, 0.77608525, 0.56862867],
       [0.66907592, 0.66935241, 0.18729434, 0.07962013],
       [0.71881369, 0.96671938, 0.78528091, 0.72711842]])
```

In [52]:

```
#Numpy Reshape
'''
> Reshaping means changing the shape of an array
> The shape of an array is the number of elements in each dimension
> By reshaping we can add or remove dimensions or change number of elements in
each dimension
'''
```

Out[52]:

```
'\n> Reshaping means changing the shape of an array\n> The shape of an array is the number of elements in each dimension\n> By reshaping we can add or remove dimensions or change number of elements in\neach dimension\n'
```

In [57]:

```
x=np.arange(1, 22)
print(x)
print(x.shape)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
(21,)
```

In [61]:

```
d=x.reshape(7,3)
print(d)
print(d.shape)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]
 [19 20 21]]
(7, 3)
```

In [62]:

```
f=d.ravel()  # convert to original dimension
print(d.shape)
print(f)
```

```
(7, 3)
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
```

In [64]:

```
x.reshape9((3,7),order='F') # order-rows  for reshape-cloumn
```

Out[64]:

```
array([[ 1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19, 20, 21]])
```

Numpy Array indexing

> Array indexing is the same as accessing an array element.You can access an array element by referring to its index numbers The indexes in numpy arrays starts with(), meaning that the first element has index(), and the second index 1 so on...

In [65]:

```
x.flatten()  #flattening x convert into 1D array
```

Out[65]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21])
```

In [66]:

```
z=np.arange(1,12,3)
z
```

Out[66]:

```
array([ 1,  4,  7, 10])
```

In [68]:

```
z[3]
```

Out[68]:

```
10
```

In [69]:

```
z[0:2:1]
```

```
Out[69]: array([1, 4])
```

```
In [70]: z=np.array([[1,2,3],[4,5,6],[7,8,9]])
         z
```

```
Out[70]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [73]: z[1,1]
```

```
Out[73]: 5
```

```
In [74]: z[2,2]
```

```
Out[74]: 9
```

```
In [75]: z[0,0]
```

```
Out[75]: 1
```

```
In [76]: #Numpy copy vs view
         z1=np.arange(12)
         z1
```

```
Out[76]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
In [78]: z2=z1
         print(z1)
         print(z2)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
In [84]: z2[0]=12
         print(z1)
         print(z2)
```

```
[12 22  2  3  4  5  6  7  8  9 10 11]
[12 22  2  3  4  5  6  7  8  9 10 11]
```

```
In [86]: print(np.shares_memory(z1,z2))
         print(id(z1))
         print(id(z2))
```

```
True
2139866490128
2139866490128
```

```
In [87]: z
```

```
Out[87]: array([[1, 2, 3],
                [4, 5, 6],
        [7, 8, 9]])
```

In [88]:

```
z>3
```

Out[88]:

```
array([[False, False, False],
       [ True,  True,  True],
       [ True,  True,  True]])
```

In [90]:

```
z[z>2]
```

Out[90]:

```
array([3, 4, 5, 6, 7, 8, 9])
```

In [97]:

```
z[(z>2)&(z<8)]
```

Out[97]:

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

In [101…

```
#Transpose array
print(z)
print('transpose of z')
print(z.transpose())
```

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

In [109…

```
#Horizontal and Vertical Stack
print(z1)
print(z2)
print('Vertical stack')
print(np.vstack((z1,z2)))
print('Horizontal stack')
np.hstack((z1,z2))
```

```
[12 22  2  3  4  5  6  7  8  9 10 11]
[12 22  2  3  4  5  6  7  8  9 10 11]
Vertical stack
[[12 22  2  3  4  5  6  7  8  9 10 11]
 [12 22  2  3  4  5  6  7  8  9 10 11]]
Horizontal stack
```

Out[109…

```
array([12, 22,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 22,  2,  3,  4,
        5,  6,  7,  8,  9, 10, 11])
```

In [110…

```
#Add,Insert & delete numpy array
print(z1)
print(z2)
np.insert(z1,5,z2)
```

```
[12 22  2  3  4  5  6  7  8  9 10 11]
[12 22  2  3  4  5  6  7  8  9 10 11]
```

Out[110…

```
array([12, 22,  2,  3,  4, 12, 22,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11,
        5,  6,  7,  8,  9, 10, 11])
```

In [113…

```
print(z2)
np.delete(z2,0)
```

```
[12 22  2  3  4  5  6  7  8  9 10 11]
```

Out[113…

```
array([22,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [114…

```
#Mathematical Operations in Numpy
a=([[1,2,3],[4,5,6],[7,8,9]])
np.sin(a)
```

Out[114…

```
array([[ 0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ],
       [ 0.6569866 ,  0.98935825,  0.41211849]])
```

In [115…

```
np.cos(a)
```

Out[115…

```
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029],
       [ 0.75390225, -0.14550003, -0.91113026]])
```

In [116…

```
(np.sin(a)/np.cos(a))
```

Out[116…

```
array([[ 1.55740772, -2.18503986, -0.14254654],
       [ 1.15782128, -3.38051501, -0.29100619],
       [ 0.87144798, -6.79971146, -0.45231566]])
```

In [117…

```
np.exp(a)
```

Out[117…

```
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01],
       [5.45981500e+01, 1.48413159e+02, 4.03428793e+02],
       [1.09663316e+03, 2.98095799e+03, 8.10308393e+03]])
```

In [118…

```
np.sum(a)
```

Out[118…

```
45
```

In [119…

```
np.average(a)
```

Out[119…

```
5.0
```

In [123…

```
np.median(a)
```

Out[123…

```
5.0
```

In [124…

```
#Searching arrays:- To search an array use Where() method
w=np.where(z1==5)
w
```

Out[124…

```
(array([5], dtype=int64),)
```

In [125…

```
z2
```

Out[125…

```
array([12, 22,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [126…

```
np.where(z2%2==0)
```

```
Out[126…  (array([ 0,  1,  2,  4,  6,  8, 10], dtype=int64),)
```

```
In [127…  np.where(z1>5,z1,0)
```

```
Out[127…  array([12, 22,  0,  0,  0,  0,  6,  7,  8,  9, 10, 11])
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```