numpy practice session

In [1]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-packages (1.
24.3)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
# import tgis library in j.notebook
import numpy as np
```

Creating an array using numpy

In [3]:
```
food=np.array (["pakora", "samosa", "raita"])
food
```

Out[3]:
```
array(['pakora', 'samosa', 'raita'], dtype='<U6')
```

In [4]:
```
price=np.array([5,5,5])
price
```

Out[4]:
```
array([5, 5, 5])
```

In [5]:
```
type(price)
```

Out[5]:
```
numpy.ndarray
```

ndarray..... n number of dimensional array

In [6]:
```
type(food)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
len(food)
```

Out[7]:
```
3
```

In [8]:
```
len(price)
```

Out[8]:
```
3
```

In [9]:
```
price[2]
```

Out[9]:
```
5
```

In [10]:
```
price[0]
```

Out[10]:
```
5
```

In [11]:
```
food[0]
```

Out[11]:
```
'pakora'
```

In [12]:
```
# 0 se start hy likin figure 3 he likhne hy
food[0:3]
```

Out[12]:
```
array(['pakora', 'samosa', 'raita'], dtype='<U6')
```

In [13]:
```
# Different function array
price.mean()
```

Out[13]:
```
5.0
```

In [14]:
```
# zeros method
np.zeros(6)
```

Out[14]:
```
array([0., 0., 0., 0., 0., 0.])
```

In [15]:
```
# ones
np.ones(4)
```

Out[15]:
```
array([1., 1., 1., 1.])
```

In [16]:
```
# empty
np.empty(5)
```

Out[16]:
```
array([6.23042070e-307, 4.67296746e-307, 1.69121096e-306, 9.12209163e-312,
       3.56050862e-307])
```

In [17]:
```
np.arange(10)
```

Out[17]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [18]:
```
# specify
np.arange(2,20)
```

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

In [19]:
```
# specific interval
np.arange(2,20,2)
```

Out[19]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [20]:
```
# specify interval
np.arange(4,100,10)
```

Out[20]:
```
array([ 4, 14, 24, 34, 44, 54, 64, 74, 84, 94])
```

In [21]:
```
# table
np.arange(0,55,5)
```

Out[21]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [22]:
```
# table
np.arange(0,44,4)
```

Out[22]:
```
array([ 0,  4,  8, 12, 16, 20, 24, 28, 32, 36, 40])
```

In [23]:
```
# line space
np.linspace(0,10, num=5)
```

Out[23]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

In [24]:
```
# line space
np.linspace(0,40, num=5)
```

Out[24]:
```
array([ 0., 10., 20., 30., 40.])
```

In [25]:
```
# linepsace (with same interval yni barabri ka gap)
np.linspace(0,50, num=5)
```

Out[25]:
```
array([ 0. , 12.5, 25. , 37.5, 50. ])
```

In [26]:
```
# line space( same distance between numbers)
np.linspace(0,10, num=5)
```

Out[26]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

In [27]:
```
#  specify your data type
np.ones(50, dtype=np.int64)
```

Out[27]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1], dtype=int64)
```

In [28]:
```
#  specify your data type
np.ones(50, dtype=np.int16)
```

Out[28]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1], dtype=int16)
```

Array FunctionFunctions

In [29]:
```
a=np.array([10,12,15,2,4,100,320,0.5,10.4])
a
```

Out[29]:
```
array([ 10. ,  12. ,  15. ,   2. ,   4. , 100. , 320. ,   0.5,  10.4])
```

In [30]:
```
# acesending order
a.sort()
a
```

Out[30]:
```
array([  0.5,   2. ,   4. ,  10. ,  10.4,  12. ,  15. , 100. , 320. ])
```

In [31]:
```
b=np.array([10.2, 3.4, 53.6, 91.4,45.4])
b
```

Out[31]:
```
array([10.2,  3.4, 53.6, 91.4, 45.4])
```

In [32]:
```
c=np.concatenate((a,b))
c
```

Out[32]:
```
array([  0.5,   2. ,   4. ,  10. ,  10.4,  12. ,  15. , 100. , 320. ,
        10.2,   3.4,  53.6,  91.4,  45.4])
```

In [33]:
```
c.sort()
c
```

Out[33]:
```
array([  0.5,   2. ,   3.4,   4. ,  10. ,  10.2,  10.4,  12. ,  15. ,
        45.4,  53.6,  91.4, 100. , 320. ])
```

2-d arrays

concatinate of same d or sizesm

In [34]:
```
   a=np.array([[1,2,3,4],[5,4,3,2]])
a
```

Out[34]:
```
array([[1, 2, 3, 4],
       [5, 4, 3, 2]])
```

In [35]:
```
b=np.array([[6,7,4,5],[8,5,5,9]])
b
```

Out[35]:
```
array([[6, 7, 4, 5],
       [8, 5, 5, 9]])
```

In [36]:
```
c=np.concatenate((a,b))
c
```

Out[36]:
```
array([[1, 2, 3, 4],
       [5, 4, 3, 2],
       [6, 7, 4, 5],
       [8, 5, 5, 9]])
```

concatenate of different D or sizes

In [37]:
```
a=np.array([[1,2,3,4],[5,4,3,2]])
a
```

Out[37]:
```
array([[1, 2, 3, 4],
       [5, 4, 3, 2]])
```

In [38]:
```
b=np.array([[5,6],[7,8]])
b
```

Out[38]:
```
array([[5, 6],
       [7, 8]])
```

In [39]:
```
c=np.concatenate((a,b), axis=1)
c
```

Out[39]:
```
array([[1, 2, 3, 4, 5, 6],
       [5, 4, 3, 2, 7, 8]])
```

concatinate of axis=0

In [40]:
```
a=np.array([[2,1],[7,3]])
a
```

Out[40]:
```
array([[2, 1],
       [7, 3]])
```

In [41]:
```
b=np.array([[4,2],[5,6]])
b
```

Out[41]:
```
array([[4, 2],
       [5, 6]])
```

In [42]:
```
c=np.concatenate((a,b), axis=0)
c
```

Out[42]:
```
array([[2, 1],
       [7, 3],
       [4, 2],
       [5, 6]])
```

In [43]:
```
c=np.concatenate((a,b), axis=1)
c
```

Out[43]:
```
array([[2, 1, 4, 2],
       [7, 3, 5, 6]])
```

concatenate of axis=1

In [44]:
```
a=np.array([[2,1,3],[7,3,4]])
a
```

Out[44]:
```
array([[2, 1, 3],
       [7, 3, 4]])
```

In [45]:
```
b=np.array([[4,2],[5,6]])
b
```

Out[45]:
```
array([[4, 2],
       [5, 6]])
```

In [46]:
```
c=np.concatenate((a,b), axis=1)
c
```

Out[46]:
```
array([[2, 1, 3, 4, 2],
       [7, 3, 4, 5, 6]])
```

In [47]:
```
# c=np.concatinate((a,b), axis=0) it can't be stack
```

3-D (2d in one etc is also called TASERFLOW

In [94]:
```
a=np.array([[[11,2,3,4],[3,4,5,6]],
            [[4,5,6,7],[5,2,3,1]],
```

```
          [[5,9,6,3],[9,6,5,3]]])
a
```

Out[94]:
```
array([[[11,  2,  3,  4],
        [ 3,  4,  5,  6]],

       [[ 4,  5,  6,  7],
        [ 5,  2,  3,  1]],

       [[ 5,  9,  6,  3],
        [ 9,  6,  5,  3]]])
```

In [49]:
```
a.size
```

Out[49]:
```
24
```

In [50]:
```
a.shape
```

Out[50]:
```
(3, 2, 4)
```

- (3,2,4) 3 means 3d, 2 means rows, 4 columns

In [51]:
```
# to find the number of dimensions
a.ndim
```

Out[51]:
```
3
```

In [52]:
```
type(a)
```

Out[52]:
```
numpy.ndarray
```

3 by 3 matrix is 2 dimension arraym

In [68]:
```
b=np.array([[5,4,3],[5,3,7],[4,7,5]])
b
```

Out[68]:
```
array([[5, 4, 3],
       [5, 3, 7],
       [4, 7, 5]])
```

In [65]:
```
b.size
```

Out[65]:
```
9
```

In [66]:
```
b.shape
```

Out[66]:
```
(3, 3)
```

In [54]:
```
b.ndim
```

Out[54]:
```
2
```

In [55]:
```
b=np.array([[[5,4,3],[5,3,7],[4,7,5]],
            [[5,4,3],[5,3,7],[4,7,5]]])
b
```

Out[55]:
```
array([[[5, 4, 3],
        [5, 3, 7],
        [4, 7, 5]],

       [[5, 4, 3],
        [5, 3, 7],
        [4, 7, 5]]])
```

In [56]:
```
b.size
```

Out[56]:
```
18
```

In [57]:
```
b.shape
```

Out[57]:
```
(2, 3, 3)
```

In [58]:
```
b.ndim
```

Out[58]:
```
3
```

In [59]:
```
b=np.array([4,5,6])
b.ndim
```

Out[59]:
```
1
```

HINT

- agr 3 blocks me bnd mtrix hoga to 3d
- agr 2 block me bnd mtrix hoga to b 3d
- age ak block me bnd hoga tb 2d
- agr simple ak he axis ho to 1D

In [60]:
```
#  size (number of elements)
a.size
```

Out[60]:
```
24
```

In [95]:
```
a=np.arange(6)
a
```

Out[95]:
```
array([0, 1, 2, 3, 4, 5])
```

In [97]:
```
#  reshape
a.reshape(2,3)
```

Out[97]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [133…
```
b=np.arange(9)   # 3*3=9
b.reshape(3,3)
```

Out[133]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [139…
```
b=np.arange(64)     #8*8=64
b.reshape(8,8)
```

Out[139]:
```
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, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47],
       [48, 49, 50, 51, 52, 53, 54, 55],
       [56, 57, 58, 59, 60, 61, 62, 63]])
```

In [165…
```
# reshape
a=np.array([[1,2,3],[6,7,8],[5,2,5]])
np.reshape(a,newshape=(1,9), order="C")
```

Out[165]:
```
array([[1, 2, 3, 6, 7, 8, 5, 2, 5]])
```

convert 1d into 2d

- [np.newaxis, :]

In [166…
```
a=np.array([1,2,3,4,5,6,7,8])
a
```

Out[166]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [167…
```
a.shape
```

Out[167]:
```
(8,)
```

shape change

In [171…
```
b=a[np.newaxis, :]
b
```

Out[171]:
```
array([[1, 2, 3, 4, 5, 6, 7, 8]])
```

In [172…
```
b.shape
```

Out[172]:
```
(1, 8)
```

Dimensional change

row wise 2d conversion

In [174…
```
c=a[np.newaxis,:]
c
```

Out[174]:
```
array([[1, 2, 3, 4, 5, 6, 7, 8]])
```

column wise 2d conversion

In [175…
```
d=a[:,np.newaxis]
d
```

Out[175]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8]])
```

indexes

In [179…
```
a
```

Out[179]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [181…
```
a[1:6]
```

Out[181]:
```
array([2, 3, 4, 5, 6])
```

In [182…
```
a[2:6]
```

Out[182]:
```
array([3, 4, 5, 6])
```

In [187…
```
a[2:8]
```

Out[187]:
```
array([3, 4, 5, 6, 7, 8])
```

In [188…
```
a*6
```

Out[188]:
```
array([ 6, 12, 18, 24, 30, 36, 42, 48])
```

In [189…
```
a
```

Out[189]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [190…
```
a*6
```

Out[190]:
```
array([ 6, 12, 18, 24, 30, 36, 42, 48])
```

In [191…
```
a+6
```

Out[191]:
```
array([ 7,  8,  9, 10, 11, 12, 13, 14])
```

In [194…
```
a.sum()
```

Out[194]:
```
36
```

In [195…
```
a.mean()
```

Out[195]:
```
4.5
```