Numpy, pandas, matplotlib,scipy,seaborn

In [ ]:

```

```

Numpy (Numerical Python)

In [ ]:

```

```

Installing Numpy

In [3]:

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.1
9.2)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import numpy as np
```

In [7]:

```
a=5
print(a)
b=6
print(b)
c=7
print(c)
print(type(a),type(b),type(c))
```

```
5
6
7
<class 'int'> <class 'int'> <class 'int'>
```

In [ ]:

```

```

Array is a collection of similar type (homogeneous) of elements at continuous(contiguous) memory location unlike lists, tuple dictionary all are hetrogeneous data structure

In [ ]:

```
numpy contains ndarray
ndarray means: n dimensional array
```

In [2]:

```
#defining 1D array using array function of numpy
import numpy as np
a=np.array([1,2,3,4,5]) #passing a list homogeneous data
print(a)
print(type(a))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [11]:

```python
#defining 1D array with different type of elements
b=np.array([1,2,3.5,4,5])
print(b)
print(type(b))
```

```
[1.  2.  3.5 4.  5. ]
<class 'numpy.ndarray'>
```

In [16]:

```python
# checking type of elements in an array
print(a.dtype)
print(b.dtype)
```

```
int32
float64
```

In [17]:

```python
#changing the datatype of elements
print(a.dtype)
a=np.array([1,2,3,4,5],dtype="float")
print(a)
print(a.dtype)
```

```
int32
[1. 2. 3. 4. 5.]
float64
```

In [19]:

```python
#creating an array by using seq of numbers through arange function
a=np.arange(10)
print(a)
b=np.arange(20,30)
print(b)
c=np.arange(20,30,3)
print(c)
print(type(a),type(b),type(c))
```

```
[0 1 2 3 4 5 6 7 8 9]
[20 21 22 23 24 25 26 27 28 29]
[20 23 26 29]
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Accessing Array Elements through Indexing and Slicing

In [20]:

```python
#indexing
print(b)
print(b[0])
print(b[5])
print(b[-1])
```

```
[20 21 22 23 24 25 26 27 28 29]
20
25
29
```

In [26]:

```python
#slicing (sub array)
print(b[:5]) #first five elements
print(b[5:]) #elements from index 5
print(b[4:7]) #from 4 to 6
print(b[::2]) #alternate elements
print(b[::-1]) #reverse
print(b[5::-2])# reverse alternate from index 5
```

```
[20 21 22 23 24]
[25 26 27 28 29]
[24 25 26]
[20 22 24 26 28]
[29 28 27 26 25 24 23 22 21 20]
[25 23 21]
```

In [27]:

```
#creating an array using tuple elements
x=np.array((1,2,3,4,5)) # tuple as parameter
print(x)
print(type(x))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

In [33]:

```
print(sum(x))
print(len(x))
lt=[1,2,3,4]
print(sum(lt))
```

```
15
5
10
```

Multi dimensional array(2-D)

In [3]:

```
#creating 2-d array
ar=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(ar)
print(type(ar))
print(ar.ndim) #ndim is used to get dimension of the array
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
<class 'numpy.ndarray'>
2
```

In [42]:

```
#indexing in 2-D
print(ar[2][1])
print(ar[0,2])
```

```
8
3
```

In [6]:

```
#slicing in 2-D
r1=ar[:2,:2] #first 2 rows and cols
print(r1)
r2=ar[::-1,::-1]#array will be reversed
print(r2)
r3=ar[::-1,:3]#rows are reversed but not the cols
print(r3)
print(ar[:,0]) #first col of matrix
print(ar[0,:]) #first row of matrix
print(ar[(0,1,2),(0,1,2)])
```

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

attributes in numpy

ndim: provides deimensions of the array

size: no of elements in the array

shape: provides order of the matrix

nbytes: no of bytes occupied by an array

itemsize: bytes occupied by each element of the matrix

dtype: gives type of element

In [7]:

```
a=np.array([1,2,3,4,5])
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print(b)
```

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

In [8]:

```
#ndim
print(a.ndim)
print(b.ndim)
```

```
1
2
```

In [9]:

```
#size
print(a.size)
print(b.size)
```

```
5
9
```

In [12]:

```
#shape
print(a.shape)
print(b.shape)
c=np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(c.ndim)
print(c.size)
print(c.shape)
```

```
(5,)
(3, 3)
3
8
(2, 2, 2)
```

In [13]:

```
#nbytes
print(a.nbytes)
print(b.nbytes)
print(c.nbytes)
```

```
20
36
32
```

In [14]:

```
#itemsize
print(a.itemsize)
print(b.itemsize)
print(c.itemsize)
```

```
4
4
4
```

In [17]:

```
#dtype
print(a.dtype)
a=np.array([1,2,3,4,5],dtype='int16')
print(a.dtype)
```

```
int16
int16
```

In [18]:

```
print(a.itemsize)
```

```
2
```

creating some usual matrices

In [22]:

```
#creating an array of zeros
a=np.zeros((2,4),dtype='int16')
print(a)
print(a.dtype)
print(a.itemsize)
```

```
[[0 0 0 0]
 [0 0 0 0]]
int16
2
```

In [25]:

```
#creating an array of ones
a=np.ones((3,4))
print(a)
print(a.dtype)
print(a.itemsize)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
float64
8
```

In [28]:

```
#creating an array having all the elemets same
a=np.full((3,4),44)
print(a)
```

```
[[44 44 44 44]
 [44 44 44 44]
 [44 44 44 44]]
```

In [30]:

```
#creating a matrix of nan
b=np.full((3,3),np.nan)
print(b)
```

```
[[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
```

In [31]:

```
#creating a diagonal matrix
c=np.diag([10,20,30,40,50])
print(c)
```

```
[[10  0  0  0  0]
 [ 0 20  0  0  0]
 [ 0  0 30  0  0]
 [ 0  0  0 40  0]
 [ 0  0  0  0 50]]
```

In [33]:

```
#creating an identity matrix
d=np.eye(5,dtype='int16')
print(d)
```

```
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

In [38]:

```
#changing type of elements in an array
a=np.array([1,2,3,4,5])
print(a)
print(a.dtype)
b=a.astype(float)#astype is used to change data type of elements
print(b)
print(a)
```

```
[1 2 3 4 5]
int32
[1. 2. 3. 4. 5.]
[1 2 3 4 5]
```

reshaping a matrix: changing the order of a matrix

In [39]:

```
c=np.arange(1,13)
print(c)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

In [54]:

```
#reshaping an array
x=c.reshape((3,4))
print(x)
print(c)
y=c.reshape((2,3,2))
print(y)
z=x.reshape((4,3))
print(z)
w=z.reshape((1,12))
print(w)
v=z.reshape((12))
print(v)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[ 1  2  3  4  5  6  7  8  9 10 11 12]
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[ 1  2  3  4  5  6  7  8  9 10 11 12]]
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

In [3]:

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

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

In [6]:

```
#flatten and ravel methods
a=ar.ravel() # it creates 1 dimensional array from any multidim
print(a)
print(ar)
b=ar.flatten() # it creates 1 dimensional array from any multidim
print(b)
```

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

In [12]:

```
# repeat: it repeats the element of an existing array to form new arrays
x=np.array([[10,20,30]])
print(x)
r=np.repeat(x,10,axis=0)
print(r)
s=np.repeat(x,10,axis=1)
print(s)
```

```
[[10 20 30]]
[[10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]
 [10 20 30]]
[[10 10 10 10 10 10 10 10 10 10 20 20 20 20 20 20 20 20 20 20 30 30 30 30
  30 30 30 30 30 30]]
```

In [14]:

```
#creating empty matrics
e=np.empty((3,3),dtype='int16') #try it with float64 as well
print(e)
```

```
[[ -9360 -32762    595]
 [     0      0      0]
 [     0      0      1]]
```

random numbers:

In [15]:

```
#random.rand() is used to create a matrics of random numbers between 0 and 1.
d=np.random.rand(3,4) # provide the order
print(d)
```

```
[[0.78146601 0.65532222 0.97807561 0.46578427]
 [0.9754159  0.89626486 0.79751148 0.80555687]
 [0.46268769 0.6170843  0.33949579 0.33992345]]
```

In [19]:

```
#random.randint() is used to create matrics of random integers within  specified range
c=np.random.randint(-4,10,size=(2,3))
print(c)
```

```
[[-3 -1 -1]
 [ 2  0  8]]
```

copy and view methods

In [23]:

```
#view method: it will generate an array same as existing one
# if we change data in view then change will reflect in the original one as well
a=np.array([10,20,30,40,50])
print(a)
b=a.view()
print(b)
b[1]=200
print(b)
print(a)
```

```
[10 20 30 40 50]
[10 20 30 40 50]
[ 10 200  30  40  50]
[ 10 200  30  40  50]
```

In [26]:

```
x=np.arange(30,41)
print(x)
y=x
print(y)
y[1]=100
print(y)
print(x)
```

```
[30 31 32 33 34 35 36 37 38 39 40]
[30 31 32 33 34 35 36 37 38 39 40]
[ 30 100  32  33  34  35  36  37  38  39  40]
[ 30 100  32  33  34  35  36  37  38  39  40]
```

In [29]:

```
#copy method creates a copy of array where changes made in
#copy will not be reflected in the original array
c=a.copy()
print(c)
c[0]=100
print(c)
print(a)
```

```
[ 10 200  30  40  50]
[100 200  30  40  50]
[ 10 200  30  40  50]
```

In [33]:

```
x=np.arange(11,23).reshape(3,4)
print(x)
y=x.ravel() # creates a view of original array
z=x.flatten() # creates a copy of original array
print(y)
print(z)
y[1]=100
```

```
z[1]=200
print(y)
print(z)
print(x)
```

```
[[11 12 13 14]
 [15 16 17 18]
 [19 20 21 22]]
[11 12 13 14 15 16 17 18 19 20 21 22]
[11 12 13 14 15 16 17 18 19 20 21 22]
[ 11 100  13  14  15  16  17  18  19  20  21  22]
[ 11 200  13  14  15  16  17  18  19  20  21  22]
[[ 11 100  13  14]
 [ 15  16  17  18]
 [ 19  20  21  22]]
```

Mathematical Operations(with scalar quantity)

In [34]:

```
#all arithematic operators can be applied on arrays with a scalar value
a=np.arange(1,9)
print(a)
```

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

In [35]:

```
print(a+2)
print(a-2)
print(a*2)
print(a/2)
print(a//2)
print(a%2)
print(a**2)
```

```
[ 3  4  5  6  7  8  9 10]
[-1  0  1  2  3  4  5  6]
[ 2  4  6  8 10 12 14 16]
[0.5 1.  1.5 2.  2.5 3.  3.5 4. ]
[0 1 1 2 2 3 3 4]
[1 0 1 0 1 0 1 0]
[ 1  4  9 16 25 36 49 64]
```

In [36]:

```
a=a.reshape(2,4)
print(a)
```

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

In [38]:

```
print(a+2)
print(a-2)
print(a*2)
print(a/2)
print(a//2)
print(a%2)
print(a**2)
print(a)
```

```
[[ 3  4  5  6]
 [ 7  8  9 10]]
[[-1  0  1  2]
 [ 3  4  5  6]]
[[ 2  4  6  8]
 [10 12 14 16]]
[[0.5 1.  1.5 2. ]
 [2.5 3.  3.5 4. ]]
[[0 1 1 2]
 [2 3 3 4]]
[[1 0 1 0]
 [1 0 1 0]]
[[ 1  4  9 16]
 [25 36 49 64]]
[[1 2 3 4]
 [5 6 7 8]]
```

In [39]:

```
a=a+5
print(a)
```

```
[[ 6  7  8  9]
 [10 11 12 13]]
```

In [40]:

```
np.sin(a)
```

Out[40]:

```
array([[-0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849],
       [-0.54402111, -0.99999021, -0.53657292,  0.42016704]])
```

In [41]:

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

Out[41]:

```
array([[ 0.96017029,  0.75390225, -0.14550003, -0.91113026],
       [-0.83907153,  0.0044257 ,  0.84385396,  0.90744678]])
```

linear algebra

In [43]:

```
A=np.array([[2,0,3],[4,1,8],[7,5,9]])
print(A)
B=np.array([[10,20,30],[50,40,70],[90,60,80]])
print(B)
```

```
[[2 0 3]
 [4 1 8]
 [7 5 9]]
[[10 20 30]
 [50 40 70]
 [90 60 80]]
```

In [44]:

```
#Transpose of a matrix
A.transpose()
```

Out[44]:

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

In [45]:

```
B.T # alternate way to transpose
```

Out[45]:

```
array([[10, 50, 90],
       [20, 40, 60],
       [30, 70, 80]])
```

In [46]:

```
#trace: it will give sum of diagonal elements
print(np.trace(A))
print(np.trace(B))
```

```
12
130
```

In [47]:

```
#A+B: Addition of two matrices
print(A+B)
```

```
[[12 20 33]
 [54 41 78]
 [97 65 89]]
```

In [48]:

```
#A*B: Multiplication of two matrices
print(A.dot(B))#or
print(A@B)#or
print(np.matmul(A,B))
```

```
[[ 290  220  300]
 [ 810  600  830]
 [1130  880 1280]]
[[ 290  220  300]
 [ 810  600  830]
 [1130  880 1280]]
[[ 290  220  300]
 [ 810  600  830]
 [1130  880 1280]]
```

In [1]:

```
import numpy as np
A=np.array([[2,0,3],[4,1,8],[7,5,9]])
print(A)
B=np.array([[10,20,30],[50,40,70],[90,60,80]])
print(B)
```

```
[[2 0 3]
 [4 1 8]
 [7 5 9]]
[[10 20 30]
 [50 40 70]
 [90 60 80]]
```

In [3]:

```
# Determent of a matrix
print(np.linalg.det(A))
print(np.linalg.det(B))
```

```
-23.0
18000.000000000007
```

In [4]:

```
#inverse of a matrix
print(np.linalg.inv(A))
print(np.linalg.inv(B))
```

```
[[ 1.34782609 -0.65217391  0.13043478]
 [-0.86956522  0.13043478  0.17391304]
 [-0.56521739  0.43478261 -0.08695652]]
[[-0.05555556  0.01111111  0.01111111]
 [ 0.12777778 -0.10555556  0.04444444]
 [-0.03333333  0.06666667 -0.03333333]]
```

In [8]:

```
#statstics function
print(A.sum())  #sum of elements
print(A.sum(axis=0))# for colwise
print(A.sum(axis=1))#rowwise
print(A.mean(axis=0))# mean value
print(A.mean(axis=1))
print(A.mean())
print(A.cumsum(axis=0))# cumulative Sum
print(A.cumsum(axis=1))
print(A.cumsum())
print(A.std()) # standard deviation we can calculate row and colwise as well
```

```
39
[13  6 20]
[ 5 13 21]
[4.33333333 2.         6.66666667]
[1.66666667 4.33333333 7.        ]
4.333333333333333
[[ 2  0  3]
 [ 6  1 11]
 [13  6 20]]
[[ 2  2  5]
 [ 4  5 13]
 [ 7 12 21]]
[ 2  2  5  9 10 18 25 30 39]
2.9814239699997196
```

Use of Boolean expressions with array

where clause

In [9]:

```
a=np.arange(12).reshape(3,4)
print(a)
```

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

In [10]:

```
b=a>4 # it creates boolean array
print(b)
```

```
[[False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
```

In [11]:

```
print(a[b])
```

```
[ 5  6  7  8  9 10 11]
```

In [12]:

```
a[b]=15
print(a)
```

```
[[ 0  1  2  3]
 [ 4 15 15 15]
 [15 15 15 15]]
```

In [13]:

```
#using where
print(np.where(a<3))#provides index for which the condition is true
                     #it returns tuple corresponding to the index
```

```
(array([0, 0, 0], dtype=int64), array([0, 1, 2], dtype=int64))
```

In [14]:

```
x=np.array([2,1,5,7,3,4,8,9])
print(x)
print(np.where(x<5))
```

```
[2 1 5 7 3 4 8 9]
(array([0, 1, 4, 5], dtype=int64),)
```

In [16]:

```
y=np.where(x<5,x,25) #values are replaced by 25 for which condition is false
print(y)
print(x)
```

```
[ 2  1 25 25  3  4 25 25]
[2 1 5 7 3 4 8 9]
```

Reading and writing from/in files

In [17]:

```
#writing in a file
#savetxt
np.savetxt(r"C:\users\admin\desktop\example.txt",a,fmt='%d',delimiter=',',header="my fi
```

In [18]:

```
#reading from file
#loadtxt
y=np.loadtxt(r"C:\users\admin\desktop\example.txt",delimiter=',',dtype=int,usecols=(0,1
```

In [19]:

```
print(y)
```

```
[[ 0  1]
 [ 4 15]
 [15 15]]
```

In [20]:

```
a.tofile(r"C:\users\admin\desktop\example1.txt",sep=',',format="%d")
```

In [21]:

```
b=np.fromfile(r"C:\users\admin\desktop\example1.txt",sep=',',dtype=int)
```

In [22]:

```
print(b)
```

```
[ 0  1  2  3  4 15 15 15 15 15 15 15]
```

In [23]:

```
#genfromtxt: for reading the data from file
ar_csv=np.genfromtxt(r"C:\users\admin\desktop\abc.csv",delimiter=';')
```

In [25]:

```
#print(ar_csv)
```

In [1]:

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

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

Iterating Through numpy arrays using for loop*

In [3]:

```
for item in a:
    print(item)
```

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

In [4]:

```
ar=np.arange(12).reshape(3,4)
print(ar)
```

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

In [6]:

```
for row in ar: #external loop: row wise
    for item in row: # internal loop: elements of row
        print(item,end=' ')
    print()
```

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

In [ ]:

```

```

In [7]:

```
#nditer(): numpy iterator
for item in np.nditer(ar):
    print(item)
```

```
0
1
```

```
2
3
4
5
6
7
8
9
10
11
```

In [8]:

```
for item in np.nditer(ar,order='C'): #Row major order
    print(item)
```

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

In [9]:

```
for item in np.nditer(ar,order='F'): #Col major order
    print(item)
```

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

In [14]:

```
for item in np.nditer(ar,order='F',flags=['external_loop']): #Col major order
    #external loop provides col elements as one dim array
    print(item)
```

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

In [16]:

```
for item in np.nditer(ar,op_flags=['readwrite']):#it will modify array elements
    item[...]=item*item
    print(item)
```

```
0
```

```
1
4
9
16
25
36
49
64
81
100
121
```

In [17]:

```
print(ar)
```

```
[[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]]
```

In [19]:

```
#iterating two arrays simultaneously using nditer
ar1=np.arange(12).reshape(3,4)
ar2=np.arange(12,24).reshape(3,4)
print(ar1)
print(ar2)
for x,y in np.nditer([ar1,ar2]):
    print(x,y)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
0 12
1 13
2 14
3 15
4 16
5 17
6 18
7 19
8 20
9 21
10 22
11 23
```

In [23]:

```
#we can broadcast more than one array through nditer
#when all of the arrays has equal trailing size
ar1=np.arange(12).reshape(3,4)
ar2=np.array([11,12,13,14])
ar3=np.array([10,20,30,40])
print(ar1)
print(ar2)
for x,y,z in np.nditer([ar1,ar2,ar3]):
    print(x,y,z)
```

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

```
[11 12 13 14]
0 11 10
1 12 20
2 13 30
3 14 40
4 11 10
5 12 20
6 13 30
7 14 40
8 11 10
9 12 20
10 13 30
11 14 40
```

In [27]:

```
for i in ar1.flat: #it is an iterator, it will access one
                     #element at a time in a flatten way
    print(i)
```

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

In [28]:

```
for item in ar1.flatten():
    print(item)
```

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

In [29]:

```
#split() function: split an array into multiple sub arrays of equal size
x=np.array([0,1,2,3,4,5,6,7])
print(x)
```

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

In [32]:

```
print(np.split(x,4))# in case of np.split(x,5) will raise an error
```

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

In [33]:

```
np.split(x,[2,4])# we provide index: x[:2],x[2:4],x[4:]
```

Out[33]:

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

In [34]:

```
np.split(x,[2,4,6])
```

Out[34]:

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

In [36]:

```
y=np.arange(11,27).reshape(4,4)
print(y)
```

```
[[11 12 13 14]
 [15 16 17 18]
 [19 20 21 22]
 [23 24 25 26]]
```

In [39]:

```
np.split(y,2) #split in equal sizes
```

Out[39]:

```
[array([[11, 12, 13, 14],
        [15, 16, 17, 18]]),
 array([[19, 20, 21, 22],
        [23, 24, 25, 26]])]
```

In [40]:

```
np.split(y,[1,3])
```

Out[40]:

```
[array([[11, 12, 13, 14]]),
 array([[15, 16, 17, 18],
        [19, 20, 21, 22]]),
 array([[23, 24, 25, 26]])]
```

In [ ]:

```
#numpy ends here
```