NUMPY

Why Numpy is better than list

1. Numpy provides efficient storage
2. Provides better ways of handling data for preprocessing
3. It is fast
4. Uses relatively less memory to store data
5. Numpy uses contiguous memory while lists don't
6. No type checking while looping in numpy

Helps to increase speed by vectorization,also this array is homogenous

a=[1,2,3] , b=[1,4,2] , a*b =ERROR (gives error)

a=np.array([1,2,3]) , b=np.array([1,4,2]) , a*b=np.array([1,8,6])

```
import numpy as np
```

```
#create np array
myarr=np.array([3,4,37,7])
print(myarr)
print("data type of values is : ", myarr.dtype)      #print data type of values
print("size of array is :",myarr.size)               #print count of elements
print("shape of array is :" ,myarr.shape)            #print shape of arrray
print("Dimension of array is :",myarr.ndim)          #print dimesnsion of array
```

```
[ 3  4 37  7]
data type of values is :  int64
size of array is : 4
shape of array is : (4,)
Dimension of array is : 1
```

```
#This array is homogenous but list is heterogenous
#so hetrogeneous list to array
p=[1,2,4,5,2,5,"pal"]
np_p=np.array(p)        #All will get converted to string
print(np_p)
print(np_p.dtype)
print(np_p.nbytes)      #total size of np array
```

```
['1' '2' '4' '5' '2' '5' 'pal']
<U21
588
```

```
#can even assign dtype
np_t=np.array([1,2,3,4],np.float32)
np_t
```

```
array([1., 2., 3., 4.], dtype=float32)
```

Other methods to make np array

```
a=np.zeros((3,3))
a
```

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

```
a2=np.ones((3,3))
a2
```

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

```
#for random values
a3=np.random.random((3,4))
a3
```

```
array([[0.60791597, 0.83531929, 0.29252787, 0.7314454 ],
       [0.70049141, 0.69210195, 0.91767332, 0.01128485],
       [0.24483708, 0.74961796, 0.50603566, 0.99685925]])
```

```
#np array with random integers in range [2,7) also if 2 would not be given default 0 would ha
np.random.randint(2,7,size=(3,3))
```

```
array([[5, 3, 5],
       [5, 5, 4],
       [2, 3, 5]])
```

```
a4=np.full((3,3),24,dtype="int16")        #all elements =24
a4
```

```
array([[24, 24, 24],
       [24, 24, 24],
       [24, 24, 24]], dtype=int16)
```

```
#array with values 0.....14
a5=np.arange(15)
```

```
a5
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
#Array with n equally spaced values between a and b  -> np.linspace(a,b,n)
a6=np.linspace(1,4,5)
a6
```

```
array([1.  , 1.75, 2.5 , 3.25, 4.  ])
```

```
#Question :- Generate given array pattern
a=np.ones((5,5),dtype="int")
a[1:-1,1:-1]=0
a[a.shape[0]//2,a.shape[1]//2]=9
print(a)
```

```
[[1 1 1 1 1]
 [1 0 0 0 1]
 [1 0 9 0 1]
 [1 0 0 0 1]
 [1 1 1 1 1]]
```

Indexing

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

print(b[0])
print(b[0,0] , b[0][0])       #Two ways to handle 2D array
print(b[-1,-1])               #negative based indexing
```

```
[1 2 3]
1 1
9
```

```
print(b[1,:])                 #return second row
```

```
[4 5 6]
```

```
print(b)

print("\n second column")
print(b[:,1])                 #return second column

print("\n first and 3rd columns")
print(b[:,0:3:2])         #0->start col  3->end col(not included)   2->step size(jump)
```

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

```
 [7 8 9]]

 second column
[2 5 8]

 first and 3rd columns
[[1 3]
 [4 6]
 [7 9]]
```

Be Careful while copying array

```
#if b=a so this means that b points where a points, so changes in b will be displayed on a
a=np.array([1,2,3])
b=a
b[0]=100
print(a)#changes will be shown here also
```

```
[100   2   3]
```

```
#better way is
a=np.array([1,2,3])
b=a.copy()
b[0]=100
print(a)#no change occurs here
```

```
[1 2 3]
```

Reshaping

```
a=np.array([1,2,3,4,5,6,7,8,9])
a2=np.reshape(a,(3,3))
a2
```

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

```
a3=np.reshape(a2,(1,3,3))
a3
```

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

```
#vertical , horizontal stacking two arrays
```

```
p=np.array([1,2,4])
q=np.array([4,5,6])
print(np.vstack((p,q)))
print(np.hstack((q,p)))
```

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

Appending and changing data

```
b=np.array([[1,2,3],[4,3,1],[1,2,4]])

b[1,2]=22
print(b)       #after changes
```

```
[[ 1  2  3]
 [ 4  3 22]
 [ 1  2  4]]
```

In np 1D array only one axis ->axis 0, so data appends in one way only.

In 2D array rows->axis 0 and columns->axis 1

so to append as row -> append as axis 0

to append as columns ->append as axis 1

```
#APPENDING
print(b)

temp=np.array([10,11,12])

print("\n Row sum")
row_sum=np.append(b,temp.reshape(1,3),axis=0)     #for appending dimension of both arrays be
print(row_sum)

print("\n Column sum")
col_sum=np.append(b,temp.reshape(3,1),axis=1)     #for appending dimension of both arrays be
print(col_sum)
```

```
[[ 1  2  3]
 [ 4  3 22]
 [ 1  2  4]]

 Row sum
[[ 1  2  3]
 [ 4  3 22]
 [ 1  2  4]
```

```
 [10 11 12]]

 Column sum
[[ 1  2  3 10]
 [ 4  3 22 11]
 [ 1  2  4 12]]
```

MATHS

b.sum() like functions are methods

b.size,b.dtype are attributes

```
#element wise maths
a=np.array([1,2,3],dtype="int")
print(a+3)
print(a*3)
print(a/2)
print(np.sin(a))    #take sin values of numbers
```

```
[4 5 6]
[3 6 9]
[0.5 1.  1.5]
[0.84147098 0.90929743 0.14112001]
```

```
#maths between two arrays
b=np.array([4,5,6],int)
print(a+b)
print(a-b)
print(a*b)
```

```
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
```

```
b=np.array([[1,2,3],[4,5,6]])
b
```

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

```
#sum along rows
print(b.sum(axis=0),"\n")

#sum along columns
print(b.sum(axis=1),"\n")

#total sum
```

```
#total sum
print(b.sum())
```

```
[5 7 9]

[ 6 15]

21
```

```
print(b.cumsum(),"\n")     #first flatten array then find cummulative sum

print(b.prod())            #product of all values
```

```
[ 1  3  6 10 15 21]

720
```

```
print(b.max())
print(b.max(axis=0))
print(b.min())
```

```
6
[4 5 6]
1
```

```
#index of max value
print(b.argmax(axis=0))   #max values index along row
```

```
[1 1 1]
```

Matrix

```
x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])

#sum of two arrays
print(x+y)
```

```
[[ 6  8]
 [10 12]]
```

```
#product of two arrays
print(x*y)
```

```
[[ 5 12]
```

```
       [21 32]]

#matrix multiplication
print(np.matmul(x,y))

     [[19 22]
      [43 50]]


#Find transpose (a[i][j]=a[j][i])
print(x.T)

     [[1 3]
      [2 4]]
```

Extra Features

Load data from file

a=np.genfromtxt(' data.csv ', delimiter=',')

```
#convert to python list
l=x.tolist()
l

     [[1, 2], [3, 4]]


#you can index with list in numpy
z=np.array([1,2,3,4,5,6,7,8,9])
z=z[[0,1,8]]
print(z)

     [1 2 9]


#list as index for 2D array -> pass 2 list [x indexs],[y indexs]
z1=np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
print(z1[[0,1,2],[1,2,3]])
print("\n",z1)

#get 4,5,14,15
print("\n",z1[[0,2],3:])

     [ 2  8 14]

      [[ 1  2  3  4  5]
      [ 6  7  8  9 10]
      [11 12 13 14 15]]
```

```
[[ 4  5]
 [14 15]]
```

```
#boolena to check if value satisfied
bool_y= y>5         #values in y >5 have True
print(bool_y)
```

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

```
print(np.where(y>5))    #return tuples of array where condition satisfied
```

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

Notebook tip:- without being on typing part of cell

type a -> add cell above

type b-> add cell below