Numpy Tutorials

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python

What is an array

An array is a data structure that stores values of same data type. In Python, this is the main difference between arrays and lists. While python lists can contain values corresponding to different data types, arrays in python can only contain values corresponding to same data type

In [1]:

```
pip install numpy # To install numpy
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
ERROR: Invalid requirement: '#'
```

In [2]:

```
## initially Lets import numpy

import numpy as np
```

In [3]:

```
my_lst=[1,2,3,4,5] #list

arr=np.array(my_lst) #conveting list into array
```

In [4]:

```
print(arr)
```

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

In [5]:

```
type(arr)
```

Out[5]:

```
numpy.ndarray
```

In [6]:

```
arr # here array is one dimensional array
    #since it starts & close with one square bracket
```

Out[6]:

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

In [7]:

```
arr.shape #Shape is an inbuilt function, which gives us dimension of array in (row,column
) format.
          # for 1-D array it gives us just the no. of element present.
```

Out[7]:

```
(5,)
```

MULTI-DIMENSIONAL ARRAY(MULTI-NESTED ARRAY)

In [8]:

```
my_lst1=[1,2,3,4,5]
my_lst2=[2,3,4,5,6]
my_lst3=[9,7,6,8,9]

arr=np.array([my_lst1,my_lst2,my_lst3])
```

In [9]:

```
arr   #2-d array
      #2 opening and 2 closing bracket
```

Out[9]:

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

In [10]:

```
arr.dtype
```

Out[10]:

```
dtype('int32')
```

In [11]:

```
type(arr)
```

Out[11]:

```
numpy.ndarray
```

In [12]:

```
arr.shape  #(row,column)
```

Out[12]:

```
(3, 5)
```

In [13]:

```
my_lst1=[1,2,3,4,5]
my_lst2=[2,3,4,5,6]
my_lst3=[9,7,6,8,9]

arr=np.array([my_lst1,my_lst2,my_lst3],np.int64)
```

In [14]:

```
arr.dtype
```

Out[14]:

```
dtype('int64')
```

In [ ]:

```

```

RESHAPING OF ARRAY

In [15]:

```
arr.reshape(5,3) #the no. of elements should always remain same after  reshaping
```

```
                # we can't reshape these array into (5,4)
```

Out[15]:

```
array([[1, 2, 3],
       [4, 5, 2],
       [3, 4, 5],
       [6, 9, 7],
       [6, 8, 9]], dtype=int64)
```

In [16]:

```
arr.reshape(1,15) #look carefully, the printed array is 2-d array.
```

Out[16]:

```
array([[1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 9, 7, 6, 8, 9]], dtype=int64)
```

INDEXING IN ARRAYS

In [17]:

```
# accessing elements from array (array indexing)
arr=np.array(my_lst1)
```

In [18]:

```
arr
```

Out[18]:

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

In [19]:

```
arr[3]
```

Out[19]:

```
4
```

indexing rows and column

In [20]:

```
arr=np.array([my_lst1,my_lst2,my_lst3])
```

In [21]:

```
arr
```

Out[21]:

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

In [ ]:

```

```

In [22]:

```
arr[:2,0:]
```

Out[22]:

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

In [23]:

```
arr[1:2,0:5:2]
```

Out[23]:

```
array([[2, 4, 6]])
```

In [24]:

```
arr=np.array([my_lst1,my_lst2,my_lst3])
```

In [25]:

```
arr
```

Out[25]:

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

In [26]:

```
arr[2,3]= 7
```

In [27]:

```
arr
```

Out[27]:

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

In [28]:

```
arr.size
```

Out[28]:

```
15
```

creating object array

In [29]:

```
np.array({34,23,23})
```

Out[29]:

```
array({34, 23}, dtype=object)
```

LEARNING SOME IN-BUILT FUNCTIONS IN ARRAYS

1.ARRANGE

In [30]:

```
import numpy as np
```

In [31]:

```
#Arrange function creates 1-D Array
arr=np.arange(0,10)
```

In [32]:

In [32]:

```
arr #lower value toh aayegi,par higher  value include nhi hogi
```

Out[32]:

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

In [33]:

```
np.arange(15)
```

Out[33]:

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

In [34]:

```
arr=np.arange(1,10,2)
```

In [35]:

```
arr
```

Out[35]:

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

In [36]:

```
arr=np.arange(0,10,step=2)
```

In [37]:

```
arr
```

Out[37]:

```
array([0, 2, 4, 6, 8])
```

LINSPACE

In [38]:

```
# it gives equally divided points between lower and max value
np.linspace(1,10,50)
                    #print() function likhne ki jarurat nhi hai
```

Out[38]:

```
array([ 1.        ,  1.18367347,  1.36734694,  1.55102041,  1.73469388,
        1.91836735,  2.10204082,  2.28571429,  2.46938776,  2.65306122,
        2.83673469,  3.02040816,  3.20408163,  3.3877551 ,  3.57142857,
        3.75510204,  3.93877551,  4.12244898,  4.30612245,  4.48979592,
        4.67346939,  4.85714286,  5.04081633,  5.2244898 ,  5.40816327,
        5.59183673,  5.7755102 ,  5.95918367,  6.14285714,  6.32653061,
        6.51020408,  6.69387755,  6.87755102,  7.06122449,  7.24489796,
        7.42857143,  7.6122449 ,  7.79591837,  7.97959184,  8.16326531,
        8.34693878,  8.53061224,  8.71428571,  8.89795918,  9.08163265,
        9.26530612,  9.44897959,  9.63265306,  9.81632653, 10.        ])
```

COPY FUNCTION and BROAD-CASTING

In [39]:

```
arr=np.arange(0,10)
```

In [40]:

```
arr
```

Out[40]:

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

Broadcasting

In [41]:

```
arr[3:]=100  #array mai 3rd index se last index tak sabki value 100 kar do.
```

In [42]:

```
arr
```

Out[42]:

```
array([  0,   1,   2, 100, 100, 100, 100, 100, 100, 100])
```

In [43]:

```
arr1=arr  #we are assigning arr to arr1 to make a xerox
```

In [44]:

```
arr1[3:]=500
```

In [45]:

```
arr1
```

Out[45]:

```
array([  0,   1,   2, 500, 500, 500, 500, 500, 500, 500])
```

In [46]:

```
arr #hua kya yaha pr,
    # arr1 mai changes kiya toh arr bhi update ho gya
    # this is called REFERENCE-TYPE
    # array is refrence type
    # but in VALUE-TYPE,
    # arr does not changes due to change in arr1 , (EG:-a=9;a1=a;a1=10;print(a);9)
    # THIS IS BECOZ REFERENCE-TYPE PROPERTIES STORE THE NEW ASSIGNED VARIABLE AND OLD VAR
IABLE IN
    # IN SAME MEMORY BOX......
```

Out[46]:

```
array([  0,   1,   2, 500, 500, 500, 500, 500, 500, 500])
```

COPY FUNCTION

In [47]:

```
arr1=arr.copy()
```

In [48]:

```
arr1
```

Out[48]:

```
array([  0,   1,   2, 500, 500, 500, 500, 500, 500, 500])
```

In [49]:

```
arr
```

Out[49]:

```
array([  0,   1,   2, 500, 500, 500, 500, 500, 500, 500])
```

In [50]:

```
arr1[3:]=99
```

In [51]:

```
arr
```

Out[51]:

```
array([  0,   1,   2, 500, 500, 500, 500, 500, 500, 500])
```

In [52]:

```
arr1
```

Out[52]:

```
array([ 0,  1,  2, 99, 99, 99, 99, 99, 99, 99])
```

SOME CONDITIONS VERY USEFUL IN EXPLORATORY DATA ANALYSIS

In [53]:

```
val = 2
arr<2
```

Out[53]:

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

In [54]:

```
arr*2
```

Out[54]:

```
array([   0,    2,    4, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
```

In [55]:

```
arr/2
```

Out[55]:

```
array([  0. ,   0.5,   1. , 250. , 250. , 250. , 250. , 250. , 250. ,
       250. ])
```

In [56]:

```
arr%2
```

Out[56]:

```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

In [57]:

```
arr[arr<2]
```

Out[57]:

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

In [58]:

```
arr[arr<200]
```

Out[58]:

Out[58]:

```
array([0, 1, 2])
```

In [59]:

```
arr[arr<2]
arr[arr<200]
```

Out[59]:

```
array([0, 1, 2])
```

create arrays and reshaping practice

In [60]:

```
np.arange(0,10).reshape(2,5)
```

Out[60]:

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

In [61]:

```
arr1=np.arange(0,10).reshape(5,2)
```

In [62]:

```
arr2=np.arange(0,10).reshape(5,2)
```

In [63]:

```
arr1*arr2
```

Out[63]:

```
array([[ 0,  1],
       [ 4,  9],
       [16, 25],
       [36, 49],
       [64, 81]])
```

SOME MORE IN BUILT FUNCTIONS

np.zeros

In [64]:

```
np.zeros((2,5),dtype=int)
```

Out[64]:

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

np.ones

In [65]:

```
np.ones(4)  #it creates array whose all elements are one (1-D array)
```

Out[65]:

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

In [66]:

```
np.ones(4, dtype=int)
```

Out[66]:

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

In [67]:

```
np.ones((2,5),dtype=float) #IN 2-D ARRAY THIS FORMAT IS NEEDED, AND DTYPE IS MUST TO BE E
NTERED
```

Out[67]:

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

RANDOM DISTRIBUTION

In [68]:

```
np.random.rand(3,3) #it gives random uniform distribution from [0:1] & it is in 3x3 rows
and columns
```

Out[68]:

```
array([[0.60383722, 0.36625586, 0.42089185],
       [0.78687302, 0.952541  , 0.98074881],
       [0.9811554 , 0.12923877, 0.73476592]])
```

STANDARD NORMAL DISTRIBUTION

In [69]:

```
arr=np.random.randn(4,4) # every time we execute this code the values always changes bcoz
it is random
```

In [70]:

```
arr
```

Out[70]:

```
array([[-1.19791308, -0.37842771,  1.0753259 , -0.4369271 ],
       [ 0.20524418,  0.46979938, -0.55854436,  1.36503553],
       [-0.9496293 ,  0.59753117,  0.26682622, -0.56073917],
       [-1.19884708,  0.62220839,  1.26392488,  1.26335231]])
```

In [71]:

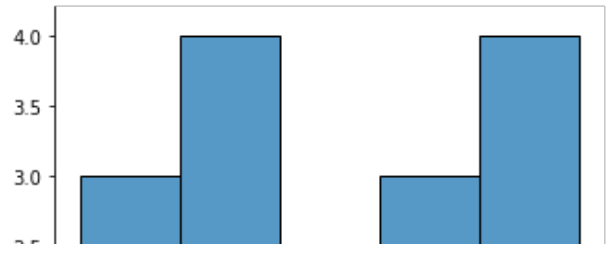
```
import seaborn as sns
import pandas as pd
```

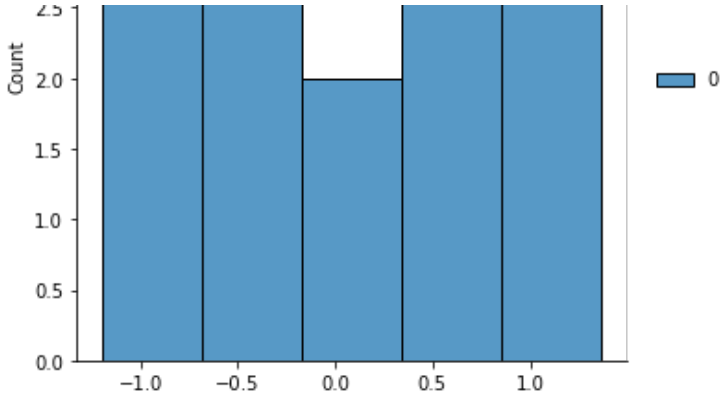
In [72]:

```
sns.displot(pd.DataFrame(arr.reshape(16,1))) #standard normal distribution ka graph hai
                                             # BELL-CURVE milna chahiye tha lekin kvh pr
blm
                                           # ke wajah se mila nhi
```

Out[72]:

```
<seaborn.axisgrid.FacetGrid at 0x9fd8400>
```





np.random.randint

In [73]:

```
np.random.randint(0,100,8).reshape(4,2)
                                        #randomly select 8 numbers from (0,100) and stor
e in (4,2)
```

Out[73]:

```
array([[49, 20],
       [59,  0],
       [94, 78],
       [98, 85]])
```

In [74]:

```
np.random.random_sample((1,5))    #select some random sample in(0.0:1.0)
```

Out[74]:

```
array([[0.30234018, 0.20245369, 0.27115746, 0.58273047, 0.4505639 ]])
```

IDENTITY MATRIX

In [75]:

```
ide = np.identity(4)
```

In [76]:

```
ide
```

Out[76]:

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

In [77]:

```
ide.size
```

Out[77]:

```
16
```

In [78]:

```
ide.shape
```

Out[78]:

```
(4, 4)
```

ARRAY RAVEL

In [79]:

```
arr = np.arange(99)
```

In [80]:

```
arr
```

Out[80]:

```
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, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98])
```

In [81]:

```
arr.reshape(3,33)
```

Out[81]:

```
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, 64,
        65],
       [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
        82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
        98]])
```

In [82]:

```
arr.ravel
```

Out[82]:

```
<function ndarray.ravel>
```

In [83]:

```
arr.ravel()  # it has converted 2d array into 1d array
```

Out[83]:

```
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, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98])
```

In [84]:

```
arr.shape
```

Out[84]:

```
(99,)
```

AXIS IN ARRAYS

1-D ARRAYS HAVE ONLY ONE AXIS, THAT IS AXIS ZERO 2-D ARRAYS HAVE TWO AXISES, THAT IS AXIS1 AND AXIS 2

REMEMBER EVERY AXIS ELEMENTS START WITH ELEMENT ZERO

REMEMBER EVERY AXIS ELEMENTS START WITH ELEMENT ZERO ....

In [85]:

```
x = [[1,2,3],[4,5,6],[7,1,0]]
```

In [86]:

```
ar = np.array(x)
```

In [87]:

```
ar
```

Out[87]:

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

In [88]:

```
ar.sum(axis=0)
```

Out[88]:

```
array([12,  8,  9])
```

In [89]:

```
ar.sum(axis=1)
```

Out[89]:

```
array([ 6, 15,  8])
```

ATTRIBUTES AND METHODS IN NUMPY

TRANSPOSE (ROWS INTO COLUMN)

In [90]:

```
ar.T
```

Out[90]:

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

FLAT (IT ITERATE THROUGH ARRAYS AND GIVES US ALL ELEMENTS)

In [91]:

```
ar.flat
```

Out[91]:

```
<numpy.flatiter at 0x3056ab0>
```

In [92]:

```
for item in ar.flat:
    print(item)
```

```
1
2
```

3
4
5
6
7
1
0

NDIM (GIVES US NO. OF DIMESNION)

In [93]:

```
ar.ndim
```

Out[93]:

2

NBYTES

In [94]:

```
ar.nbytes #it shows total bytes consumed
```

Out[94]:

36

METHODS

In [95]:

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

In [96]:

```
one.argmax() # it gives element which has maximum value
```

Out[96]:

4

In [97]:

```
one.argmin() # it gives element which has minimum value
```

Out[97]:

0

In [98]:

```
one.argsort() # it gives us array's indices, so that if arrange it in that way it will be sorted.
```

Out[98]:

```
array([0, 1, 2, 3, 4], dtype=int64)
```

In [99]:

```
one1 = np.array([1,12,33,4,5])
```

In [100]:

```
one1.argsort()
```

Out[100]:

```
array([0, 3, 4, 1, 2], dtype=int64)
```

In [101]:

```
ar
```

Out[101]:

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

In [102]:

```
ar.argmin()
```

Out[102]:

```
8
```

In [103]:

```
ar.argmax()
```

Out[103]:

```
6
```

In [104]:

```
ar.argmax(axis=0)
```

Out[104]:

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

In [105]:

```
ar.argmax(axis=1)
```

Out[105]:

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

In [106]:

```
ar
```

Out[106]:

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

In [107]:

```
ar.argsort() # see at column1,row3 = 2, means yeh array mai(3,3) pr minimum value hai
```

Out[107]:

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

In [108]:

```
ar.argsort(axis=0) # arraning elements index in assending order in axis zero.
```

Out[108]:

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

In [109]:

```
ar.argsort(axis=1) # arraning elements index in assending order in axis one.
```

Out[109]:

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

MATRIX OPERATIONS IN NUMPY

In [110]:

```
ar
```

Out[110]:

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

In [111]:

```
ar2=np.array([[1, 12, 3],
       [42, 5, 6],
       [7, 11, 0]])
```

In [112]:

```
ar2
```

Out[112]:

```
array([[ 1, 12,  3],
       [42,  5,  6],
       [ 7, 11,  0]])
```

In [113]:

```
ar + ar2
```

Out[113]:

```
array([[ 2, 14,  6],
       [46, 10, 12],
       [14, 12,  0]])
```

In [114]:

```
[321,456] + [898,909] #list mai append ho rha hai,na ki add
```

Out[114]:

```
[321, 456, 898, 909]
```

In [115]:

```
ar * ar2
```

Out[115]:

```
array([[  1,  24,   9],
       [168,  25,  36],
       [ 49,  11,   0]])
```

In [116]:

```
np.sqrt(ar)    # it gives sruare root
```

Out[116]:

```
array([[1.        , 1.41421356, 1.73205081],
```

```
       [2.        , 2.23606798, 2.44948974],
       [2.64575131, 1.        , 0.        ]])
```

In [117]:

```
ar.sum()
```

Out[117]:

```
29
```

In [118]:

```
ar.max()
```

Out[118]:

```
7
```

In [119]:

```
ar.min()
```

Out[119]:

```
0
```

In [120]:

```
ar
```

Out[120]:

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

In [121]:

```
np.where(ar>5) #it gives us arrays in form of tuple, where elemts are grater than 5.
```

Out[121]:

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

In [122]:

```
type(np.where(ar>5))
```

Out[122]:

```
tuple
```

In [123]:

```
np.count_nonzero(ar) #kitne non zero element present hai in this array
```

Out[123]:

```
8
```

In [124]:

```
np.nonzero(ar)   #  2,
                 #  2,   inke alawa sab milega,becoz (2,2) pe zero hai

'''Return the indices of the elements that are non-zero.'''
```

Out[124]:

```
'Return the indices of the elements that are non-zero.'
```

In [125]:

```
type(np.nonzero(ar))
```

```
type(np.nonzero(ar))
```

Out[125]:

```
tuple
```

In [126]:

```
ar[1,2]=0
```

In [127]:

```
np.nonzero(ar)  #(1,2) bhi hat chuka hai, becoz ab (1,2) pe zero hai
```

Out[127]:

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

REMEMBER PYTHON KI ARRAY, KI SIZE, NUMPY KI ARRAY KI SIZE, SE JYADA NBADI HOTI HAI IT MEANS IT TAKE MORE MEMORY STORAGE

In [128]:

```
ar.tolist()
```

Out[128]:

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

In [ ]:

```

```