numpy

September 23, 2024

1 Welcome to Numpy

```
[1]: import numpy as np
```

```
[2]: myarr=np.array([3,6,44,3,4,])
```

```
[3]: myarr
     #print(myarr)
```

```
[3]: array([ 3,  6, 44,  3,  4])
```

```
[4]: myarr=np.array([3,6,44,3,4,],np.int8)
```

```
[5]: myarr
```

```
[5]: array([ 3,  6, 44,  3,  4], dtype=int8)
```

```
[6]: print(myarr)
```

```
[ 3  6 44  3  4]
```

```
[7]: myarr[2]
```

```
[7]: np.int8(44)
```

```
[8]: print(myarr[2])
```

```
44
```

```
[9]: myarr2=np.array([[3,34,66,67,223,334,33]],np.int64)
```

```
[10]: myarr2
```

```
[10]: array([[  3,  34,  66,  67, 223, 334,  33]])
```

```
[11]: type(myarr2)
```

```
[11]: numpy.ndarray
```

```
[12]: myarr2.dtype
```

```
[12]: dtype('int64')
```

```
[13]: myarr2[0][3]
```

```
[13]: np.int64(67)
```

```
[14]: print(myarr2[0][3])
```

```
67
```

```
[15]: myarr2.dtype
```

```
[15]: dtype('int64')
```

```
[16]: myarr2[0,1]=1000
```

```
[17]: myarr2
```

```
[17]: array([[   3, 1000,   66,   67,  223,  334,   33]])
```

Converting Python sequences to NumPy arrays

```
[18]: a1D = np.array([1, 2, 3, 4])
      a2D = np.array([[1, 2], [3, 4]])
      a3D = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

```
[30]: a1D
```

```
[30]: array([1, 2, 3, 4])
```

```
[31]: a2D
```

```
[31]: array([[1, 2],
             [3, 4]])
```

```
[32]: a3D
```

```
[32]: array([[[1, 2],
              [3, 4]],

             [[5, 6],
              [7, 8]]])
```

```
[51]: a1d =np.array([1,2,3,4],np.int8)
      a2d=np.array([[12,11,10],[1,2,3]],np.int32)
      a3d=np.array([[[1,2,3,4],[4,3,2,1]],[[1,4,2,3],[2,3,4,5]]],np.int64)
```

```
print(a1d," \n")
print(a2d,"  \n")
print(a3d,"  \n")
print(a1d.dtype)
print(a2d.dtype)
print(a3d.dtype)
```

```
[1 2 3 4]

[[12 11 10]
 [ 1  2  3]]

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

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

int8
int32
int64
```

2 Array Creation method 2 Conversion from other Python structures

[52]:

```
listarr = np.array([[1, 2], [3, 4], [5, 6], [7, 8]], np.int64)
```

[53]:

```
listarr.dtype
```

[53]:

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

[54]:

```
listarr
```

[54]:

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

[45]:

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

[45]:

```
np.int64(2)
```

[55]:

```
listarr.shape
```

[55]:

```
(4, 2)
```

[56]:

```
listarr.size
```

[56]:

```
8
```

[59]:

```
dic=np.array({1,2,3})
print(dic)
dic.dtype
#problem is it will give object type
```

```
{1, 2, 3}
```

[59]:

```
dtype('O')
```

3 Intrinsic NumPy array creation functions (e.g. arange, ones, zeros)

[61]:

```
zeroarr = np.zeros((2,4))
```

[62]:

```
zeroarr
```

[62]:

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

[63]:

```
zeroarr.dtype
```

[63]:

```
dtype('float64')
```

[64]:

```
zeroarr.size
```

[64]:

```
8
```

[65]:

```
zeroarr.shape
```

[65]:

```
(2, 4)
```

arange

[66]:

```
arr = np.arange(2,30)
```

[68]:

```
arr
#it will give 1d array
```

[68]:

```
array([ 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])
```

linspace

```
[72]: arrlin=np.linspace(1,10,5)
                      #( start , end , how many elements )
      # it will give equally spaced 5 num
```

```
[73]: arrlin
```

```
[73]: array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

empty

```
[80]: #it will give zero array
      emp=np.empty((2,7))
                  #(no of row , no of col)
      #print(emp)
```

emp

empty_like

```
[85]: emp_lik=np.empty_like(arrlin)
      emp_lik
      #use for manupulation
```

```
[85]: array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
```

Identity matrix

```
[88]: ide=np.identity(10)
      ide
      #it will give 10x10 identity matrix
```

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

```
[89]: ide.shape
```

```
[89]: (10, 10)
```

Reshape (reshape)

```
[91]: arr= np.arange(99)
```

```
[92]: arr
```

```
[92]: 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])
```

```
[95]: arr=arr.reshape(3,33)
      #it will give 3 row and 33 each one row
      # but (2,33) not possible blc elemets is 99
      arr
```

```
[95]: 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]])
```

Ravel (ravel)

```
[98]: arr= arr.ravel()
      arr
      #it will covert in to normal 1D
```

```
[98]: 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])
```

4 Axis in array

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

```
[101]: arr
```

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

SUM

[103]:

```
arr.sum(axis=0)
#it will add columns wise 1+4+7 , 2+5+8 ,...
```

[103]:

```
array([12, 15, 18])
```

[106]:

```
#BUT
arr.sum(axis=1)
#add in rows wise
```

[106]:

```
array([ 6, 15, 24])
```

Transpose(T)

[110]:

```
arr.T
```

[110]:

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

.flat (Used for iteration in an array)

[111]:

```
arr
```

[111]:

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

[112]:

```
arr.flat
```

[112]:

```
<numpy.flatiter at 0x1f9d7254300>
```

[113]:

```
for i in arr.flat:
    print(i)
```

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

num of dimention

[115]:
```
arr.ndim
#for 2d
```

[115]:
```
2
```

[116]:
```
arr.size
```

[116]:
```
9
```

[118]:
```
arr.nbytes
#total bytes consumed\
```

[118]:
```
72
```

Max value index (.argmax)

[120]:
```
one=np.array([1,3,5,22,45,4444,99])
one
```

[120]:
```
array([   1,    3,    5,   22,   45, 4444,   99])
```

[122]:
```
one.argmax()
#index 5
```

[122]:
```
np.int64(5)
```

.argsort()

[125]:
```
# sorted array index
one.argsort()
```

[125]:
```
array([0, 1, 2, 3, 4, 6, 5])
```

[127]:
```
#for 2d array
arr
```

[127]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[128]:
```
arr.argsort()
```

[128]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [0, 1, 2]])
```

[130]:
```
arr.argmin()
#it will give sortest number index value
```

```
# 0th number index
```

```
[130]: np.int64(0)
```

```
[132]: arr.argmax()
       #index 8 having maximum value
```

```
[132]: np.int64(8)
```

```
[135]: arr.argmax(axis=1)
```

```
[135]: array([2, 2, 2])
```

```
[136]: arr.argmax(axis=0)
```

```
[136]: array([2, 2, 2])
```

```
[139]: arr.argsort(axis=1)
       #in already in sorted manner
```

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

```
[140]: arr.argsort(axis=0)
```

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

```
[141]: arr.ravel()
```

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

```
[143]: arr.reshape(9,1)
       # 9 rows and 1 columns
```

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

5 Arithmatic operations

[144]:
```
arr
```

[144]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[145]:
```
arr2=np.array([[1, 4, 3],
       [2, 5, 7],
       [7, 11, 9]])
```

[147]:
```
arr+arr2
#add two matrix or array
```

[147]:
```
array([[ 2,  6,  6],
       [ 6, 10, 13],
       [14, 19, 18]])
```

[149]:
```
# but we can't do it in python
[223,434]+[22,466]
#it will just concadinate the two list values
```

[149]:
```
[223, 434, 22, 466]
```

[151]:
```
arr*arr2
#element wise multiplication
```

[151]:
```
array([[ 1,  8,  9],
       [ 8, 25, 42],
       [49, 88, 81]])
```

[152]:
```
arr/arr2
```

[152]:
```
array([[1.        , 0.5       , 1.        ],
       [2.        , 1.        , 0.85714286],
       [1.        , 0.72727273, 1.        ]])
```

[154]:
```
arr%arr2
#find out the modulo
```

[154]:
```
array([[0, 2, 0],
       [0, 0, 6],
       [0, 8, 0]])
```

[156]:
```
#square root of the elements present in an array
np.sqrt(arr)
```

```
[156]: array([[1.        , 1.41421356, 1.73205081],
              [2.        , 2.23606798, 2.44948974],
              [2.64575131, 2.82842712, 3.        ]])
```

```
[157]: arr.sum()
```

```
[157]: np.int64(45)
```

Finding elements from the array (np.where())

```
[160]: arr
```

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

```
[164]: np.where(arr>7)
       #index value (row,col)
```

```
[164]: (array([2, 2]), array([1, 2]))
```

```
[165]: type(np.where(arr>7))
```

```
[165]: tuple
```

count zero and non zeros

```
[166]: arr
```

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

```
[167]: np.count_nonzero(arr)
```

```
[167]: 9
```

```
[169]: np.nonzero(arr)
```

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

```
[171]: #changes
       arr[1,1]=0
       np.nonzero(arr)
       #[1,1] has removed
```

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

Size compare

[172]:

```
import sys
```

[177]:

```
np_arr=np.array(py_arr)
```

[176]:

```
sys.getsizeof(1) * len(py_arr)
#one int size * length
```

[176]:

```
112
```

[179]:

```
np_arr.itemsize * np_arr.size
#size of one ele * size of arr(length)
```

[179]:

```
32
```

Array Convert list(.tolist)

[181]:

```
np_arr.tolist()
```

[181]:

```
[2, 3, 45, 33]
```

[ ]:

```

```