1)numpy

September 24, 2022

[1]:
```
my_list=[1,2,3]
print(my_list)
```

```
[1, 2, 3]
```

[4]:
```
my_list=[1,2,3]
import numpy as np
arr=np.array(my_list)
arr
```

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

[6]:
```
#to get 2 dimensional array or a matrix
my_mat=[[1,2,3],[4,5,6],[7,8,9]]
np.array(my_mat)
```

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

[8]:
```
np.arange(0,10)
```

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

[9]:
```
 np.arange(0,11)
```

[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[10]:
```
np.arange(0,11,2)
```

[10]:
```
array([ 0,  2,  4,  6,  8, 10])
```

[11]:
```
np.zeros(3)
```

[11]:
```
array([0., 0., 0.])
```

[12]:
```
np.zeros((5,5))
```

[12]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[15]:
```
np.zeros((2,3))
```

[15]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[16]:
```
np.ones(4)
```

[16]:
```
array([1., 1., 1., 1.])
```

[17]:
```
np.ones((3,4))
```

[17]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[18]:
```
np.linspace(0,5,10)
```

[18]:
```
array([0.        , 0.55555556, 1.11111111, 1.66666667, 2.22222222,
       2.77777778, 3.33333333, 3.88888889, 4.44444444, 5.        ])
```

[19]:
```
#identity matrix
np.eye(4)
```

[19]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[20]:
```
np.random.rand(5)# it's one dimensional
```

[20]:
```
array([0.82087835, 0.59950802, 0.01534275, 0.80628244, 0.41698269])
```

[23]:
```
np.random.rand(5,5)# 2d
```

[23]:
```
array([[0.66538303, 0.5055271 , 0.4358297 , 0.10041152, 0.9768169 ],
       [0.40951643, 0.62797603, 0.56734164, 0.65632029, 0.88886153],
       [0.41257781, 0.8672054 , 0.8838875 , 0.44476975, 0.77657062],
       [0.50190291, 0.06349241, 0.92371077, 0.81706227, 0.41906752],
       [0.48172611, 0.94833442, 0.70782954, 0.07958076, 0.9394027 ]])
```

[24]:
```
np.random.randn(2)#if you want to return a sample or many sample from the
                  # standard normal distribution or a gaussian distribution
```

```
                    #we can use randn
```

[24]:
```
array([ 0.21719051, -0.40178241])
```

[25]:
```
np.random.randn(4,4)
```

[25]:
```
array([[-0.18027824,  0.1380421 ,  0.05536446,  0.77672393],
       [-1.39515525, -0.32005364, -1.21700122, -1.43233653],
       [ 0.08176997,  0.7257476 , -0.65440658,  0.6142217 ],
       [ 2.17805534,  0.3713876 , -2.67011826,  0.62212731]])
```

[26]:
```
# Randint this returns random integers from a low to high number
np.random.randint(1,100)
```

[26]:
```
44
```

[28]:
```
np.random.randint(1,100,10)#here we are printing 10 random integers from
                           #1 to 100
```

[28]:
```
array([97, 47, 47, 66, 26, 39, 18, 93, 48, 86])
```

[ ]:
```

```

[29]:
```
arr=np.arange(25)
```

[30]:
```
arr
```

[30]:
```
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])
```

[31]:
```
randrr = np.random.randint(0,50,10)
```

[32]:
```
randrr
```

[32]:
```
array([ 3, 19, 34, 18, 43, 47, 33, 28, 27, 16])
```

[33]:
```
#reshape method this gonna return the array containg the same data if in new
#shape.
arr.reshape(5,5)# here i have reshaper 0 to 24 numbers into matrix number of
                #rows you want number of columns you want.If you can't fillup
                #the matrix completely you will get error
```

[33]:
```
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]])
```

[35]:
```
array=np.arange(20)
array.reshape(4,5)
```

[35]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

[36]:
```
# if you want to know the max value
array.max()
```

[36]:
```
19
```

[37]:
```
array.min()
```

[37]:
```
0
```

[38]:
```
# if you want to know the index location of a max value
array.argmax()
```

[38]:
```
19
```

[39]:
```
array.argmin()
```

[39]:
```
0
```

[40]:
```
randrr = np.random.randint(0,20,10)
randrr
```

[40]:
```
array([ 2, 14,  3, 12, 14, 10,  1, 10, 12, 11])
```

[41]:
```
randrr.argmax()
```

[41]:
```
1
```

[42]:
```
randrr.argmin()# Here the o/p which we are getting are index number of min
               #value
```

[42]:
```
6
```

[46]:
```
#If you want to figure out the shape of the vector
arr.shape#Here in o/p we see (25,) bcoz it indicates the 1 dimensional vector
```

[46]:
```
(25,)
```

[47]:
```
arr=arr.reshape(5,5)
arr
```

[47]:
```
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]])
```

[48]:
```
arr.shape# now its two dimensional
```

[48]:
```
(5, 5)
```

[49]:
```
#If you want to know what data type you have an array
arr.dtype
```

[49]:
```
dtype('int32')
```

[50]:
```
#if you don't want to type np.random.randint what you can do is
from numpy.random import randint
randint(1,10)
```

[50]:
```
7
```

#Numpy indexing and selection in this lecture we are going to know how to select elements or groups of elements from numpy array

[2]:
```
import numpy as np
arr=np.arange(0,11)
arr
```

[2]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[3]:
```
arr[8]
```

[3]:
```
8
```

[4]:
```
arr[1:5]
```

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

[5]:
```
arr[0:5]
```

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

[6]:
```
arr[:6]
```

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

[7]:
```
arr[5:]
```

[7]:
```
array([ 5,  6,  7,  8,  9, 10])
```

[10]:
```
arr[0:5]=100
arr
```

[10]:
```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

[11]:
```
arr=np.arange(0,11)
arr
```

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

[14]:
```
slice_of_arr=arr[0:6]
slice_of_arr
```

[14]:
```
array([0, 1, 2, 3, 4, 5])
```

[15]:
```
slice_of_arr[:]#this means iam grabbing everything in the slice
```

[15]:
```
array([0, 1, 2, 3, 4, 5])
```

[17]:
```
#if i want to broadast this array to a number such as 99
slice_of_arr[:]=99
slice_of_arr
```

[17]:
```
array([99, 99, 99, 99, 99, 99])
```

[18]:
```
#now if i call back the arr it will be seen like this
arr
```

[18]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

[20]:
```
arr_copy=arr.copy()
arr
```

[20]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

[22]:
```
arr_copy[:]=100
arr_copy
```

[22]:
```
array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

[23]:
```
arr#here the original copy was not changed
```

[23]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

[29]:
```
import numpy as np
arr_2d=np.array([[5,10,15],[20,25,30],[35,40,45]])
arr_2d
```

[29]:
```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

[30]:
```
#There are two methods for grabbing elements from a 2d matrix
#Double grabbing format or method
arr_2d[0][0]# for grabbing element 5
```

[30]:
```
5
```

[32]:
```
#comma single bracket notation
arr_2d[2,1]
```

[32]:
```
40
```

[34]:
```
#to grab some section of a matrix
arr_2d[:2,1:]
```

[34]:
```
array([[10, 15],
       [25, 30]])
```

[36]:
```
arr_2d[:2,:2]
```

[36]:
```
array([[ 5, 10],
       [20, 25]])
```

[38]:
```
arr_2d[]
```

[38]:
```
array([[25]])
```

#Conditional Selection

[39]:
```
arr=np.arange(1,11)
arr
```

[39]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[40]:
```
arr>5
```

[40]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

[41]:
```
bool_arr=arr>6
bool_arr
```

[41]:
```
array([False, False, False, False, False, False,  True,  True,  True,
        True])
```

[42]:
```
arr[bool_arr]#here i keep it in brackets here i will get o/p which was true
```

[42]:
```
array([ 7,  8,  9, 10])
```

[43]:
```
arr[arr>5]
```

[43]:
```
array([ 6,  7,  8,  9, 10])
```

[44]:
```
arr[arr<3]
```

[44]:
```
array([1, 2])
```

[46]:
```
arr_2d=np.arange(50).reshape(5,10)
arr_2d
```

[46]:
```
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]])
```

[48]:
```
arr_2d[1:3,3:5]#here in 1:3 we have to include 1st row and not include 3rd row
               #In 3:5 we have to include 3rd column and not include 5th column
```

[48]:
```
array([[13, 14],
       [23, 24]])
```

#Numpy Operstions

[49]:
```
import numpy as np
arr=np.arange(0,11)
arr
```

[49]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[50]:
```
#If you wanted to add two arrays together on elements by elements basis
arr+arr
```

[50]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

[51]:
```
arr-arr
```

[51]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[52]:
```
arr*arr
```

[52]:
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100])
```

[53]:
```
arr +100
```

[53]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110])
```

[54]:
```
arr*100
```

[54]:
```
array([   0,  100,  200,  300,  400,  500,  600,  700,  800,  900, 1000])
```

[55]:
```
arr-100
```

[55]:
```
array([-100,  -99,  -98,  -97,  -96,  -95,  -94,  -93,  -92,  -91,  -90])
```

[56]:
```
arr**2
```

[56]:
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100], dtype=int32)
```

[57]:
```
#Universal array function
np.sqrt(arr)
```

[57]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766])
```

[58]:
```
np.exp(arr)
```

[58]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03, 2.20264658e+04])
```

[59]:
```
np.max(arr)
```

[59]:
```
10
```

[60]:
```
np.sin(arr)
```

[60]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
       -0.54402111])
```

[61]:
```
np.log(arr)
```

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_12608\3120950136.py:1:
RuntimeWarning: divide by zero encountered in log
  np.log(arr)
```

[61]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,
       2.30258509])
```

[ ]:
```

```

[ ]:
```

```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: