
NumPy

Difference Between List and Array:

List	Array
Can consist of elements belonging to different data types	Only consists of elements belonging to the same data type
No need to explicitly import a module for declaration	Need to explicitly import a module for declaration
Cannot directly handle arithmetic operations	Can directly handle arithmetic operations
Can be nested to contain different type of elements	Must contain either all nested elements of same size
Preferred for shorter sequence of data items	Preferred for longer sequence of data items
Greater flexibility allows easy modification (addition, deletion) of data	Less flexibility since addition, deletion has to be done element wise
The entire list can be printed without any explicit looping	A loop has to be formed to print or access the components of array
Consume larger memory for easy addition of elements	Comparatively more compact in memory size

In [1]:
```
#import numpy

import numpy as np
```

Numpy Array

In [2]:
```
np.array([1,2,3,4,5])
```

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

N-Dimensional Array Object

In [3]:
```
# 1_dimensional_array

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

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

In [4]:
```
# 2_dimensional_array

arr=np.array([[1,2,3,4,5],[201,202,203,204,205]])
arr
```

Out[4]:
```
array([[  1,   2,   3,   4,   5],
       [201, 202, 203, 204, 205]])
```

In [5]:
```
# 3_Dimensional_Array

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

print(arr)
```

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

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

In [6]:
```
#Higher Dimensional Array

arr = np.array([1, 2, 3], ndmin=5)

print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 3]]]]]
number of dimensions : 5
```

Differnt Method in numpy

In [7]:
```
arr=np.array([1,2,3,4,5])
arr
```

Out[7]:
```
array([1, 2, 3, 4, 5])
```

In [8]:
```
arr.dtype
```

Out[8]:
```
dtype('int32')
```

In [9]:
```
arr.max()
```

Out[9]:
```
5
```

In [10]:
```
arr.min()
```

Out[10]:
```
1
```

In [11]:
```
arr.mean()
```

Out[11]:
```
3.0
```

In [12]:
```
arr_1=([5,4,2,3,1])
np.sort(arr_1)
```

Out[12]:
```
array([1, 2, 3, 4, 5])
```

In [13]:
```
arr = np.array([4, 2, 7, 5, 5, 9, 4, 7, 8, 1])

x = np.where(arr == 7)

print(x)
```

```
(array([2, 7], dtype=int64),)
```

```
press tab after "arr." to see all method
```

Slicing Array

```
arr[start:end]
arr[start:end:step]
```

In [14]:
```
# 1st and 2nd value of colon respectively declares the start and end of array using slicing.
# If it is null then it defines start or end remain same

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

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

In [15]:
```
# step value to determine the step of the slicing:

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

```
[1 3 5 7]
```

In [16]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1::2])
```

```
[2 4 6]
```

In [17]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

```
[5 6]
```

In [18]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

```
[7 8 9]
```

In [19]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])
```

```
[3 8]
```

In [20]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

Genarating dummy variable

```
In [21]: np.zeros(3)
```

```
Out[21]: array([0., 0., 0.])
```

```
In [22]: np.ones(5)
```

```
Out[22]: array([1., 1., 1., 1., 1.])
```

```
In [23]: np.ones((5,3))
```

```
Out[23]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

Arrange array

In [24]:
```
np.arange(0,10,1)
```

Out[24]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [25]:
```
np.arange(2,20,.5)
```

Out[25]:
```
array([ 2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,  6.5,  7. ,
        7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5, 11. , 11.5, 12. , 12.5,
       13. , 13.5, 14. , 14.5, 15. , 15.5, 16. , 16.5, 17. , 17.5, 18. ,
       18.5, 19. , 19.5])
```

In [26]:
```
arr=np.arange(-10,11)
arr
```

Out[26]:
```
array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,
         3,   4,   5,   6,   7,   8,   9,  10])
```

In [27]:
```
np.absolute(arr) #absolute value
```

Out[27]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  2,  3,  4,  5,  6,
        7,  8,  9, 10])
```

Random Value

In [28]:
```
np.random.randn(20)
```

Out[28]:
```
array([ 0.55045899, -1.73320731,  1.04292378, -0.64483832, -1.23257841,
        1.45012857, -0.50404635, -1.04621427, -1.24322554, -0.23943463,
       -0.35919922,  0.90725438,  1.05659283, -0.09707496, -0.21249339,
        0.6046799 ,  1.02553678, -0.50478966,  0.29027958,  0.45341485])
```

In [29]:
```
np.random.randn(4,3)
```

Out[29]:
```
array([[-0.63729004,  2.68922551,  1.49876587],
       [ 1.0264687 , -1.29883781,  0.52067636],
       [ 1.08326875, -0.59906422, -0.27147891],
       [ 0.70885554,  1.26220367,  0.20647921]])
```

In [30]:
```
np.random.randn(4,3,2)
```

Out[30]:
```
array([[[ 0.27985757, -0.84631692],
        [-0.92257803, -0.51567284],
        [-1.96799993,  1.14042545]],

       [[ 0.96783359, -0.28441088],
        [ 0.33827437,  1.35139777],
        [ 0.42229746, -0.87312835]],

       [[-0.48192752, -1.02948696],
        [-0.52761622,  0.54187545],
        [ 0.72403859, -0.43639268]],

       [[-0.923002  , -0.86123717],
        [ 1.09573211,  0.23835872],
        [ 0.11076563,  0.95180322]]])
```

In [31]:
```
#create a random int value
np.random.randint(11,30)
```

Out[31]:
```
15
```

In [32]:
```
#create defined random int value

np.random.randint(11,30,3)
```

Out[32]:
```
array([24, 20, 14])
```

Numpy Operation

In [33]:
```
arr=np.arange(0,10,1)
arr
```

Out[33]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [34]:
```
arr+arr
```

Out[34]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [35]:
```
arr-arr
```

Out[35]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [36]:
```
arr*arr
```

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

In [37]:
```
arr/arr
```

```
e:\program files\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered
in true_divide
  """Entry point for launching an IPython kernel.
```

Out[37]:
```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
In [38]: arr**arr
```

```
Out[38]: array([       1,        1,        4,       27,      256,     3125,
               46656,   823543, 16777216, 387420489], dtype=int32)
```

```
In [39]: arr+10
```

```
Out[39]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
In [40]: arr**2
```

```
Out[40]: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```

```
In [41]: arr/0
```

```
e:\program files\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountere
d in true_divide
  """Entry point for launching an IPython kernel.
e:\program files\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered
in true_divide
  """Entry point for launching an IPython kernel.
```

```
Out[41]: array([nan, inf, inf, inf, inf, inf, inf, inf, inf, inf])
```

```
In [42]: np.sqrt(arr)
```

```
Out[42]: array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
                2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
In [43]: np.max(arr)
```

```
Out[43]: 9
```

```
In [44]: np.sin(arr)
```

```
Out[44]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
                -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

Reshape

In [45]:
```
arr=np.arange(1,16)
arr
```

Out[45]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [46]:
```
# 2 darray
new_arr = arr.reshape(5, 3)
print(new_arr)
```

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

In [47]:
```
# 3 darray
arr= np.arange(1,13)

new_arr = arr.reshape(2, 3, 2)

print(new_arr)
```

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

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

In [48]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
arr
```

Out[48]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [49]:
```
# 1 darray
arr_1d = arr.reshape(-1)
print(arr_1d)
```

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

Joining Numpy Array

In [50]:
```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

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

In [3]:
```
num1=np.array([[1,2,3],[4,5,6]])
num2=np.array([[21,22,23],[31,32,33]])
```

In [5]:
```
num=np.concatenate((num1,num2))
num
```

Out[5]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [21, 22, 23],
       [31, 32, 33]])
```

In [7]:
```
num=np.concatenate((num1,num2),axis=0)
num
```

Out[7]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [21, 22, 23],
       [31, 32, 33]])
```

In [8]:
```
num=np.concatenate((num1,num2),axis=1)
num
```

Out[8]:
```
array([[ 1,  2,  3, 21, 22, 23],
       [ 4,  5,  6, 31, 32, 33]])
```

Spliting Numpy Array

In [52]:
```
arr = np.arange(1,10)
new_arr = np.array_split(arr, 3)
new_arr
```

Out[52]:
```
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]
```

In [53]:
```
#Access the splitted arrays

print(new_arr[0])
print(new_arr[1])
print(new_arr[2])
```

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

In [54]:
```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.array_split(arr, 3)

print(newarr)
```

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

In [53]:
```
#Access the splitted arrays

print(new_arr[0])
print(new_arr[1])
print(new_arr[2])
```

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

In [54]:
```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.array_split(arr, 3)

print(newarr)
```

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