numpy-library

February 16, 2022

Table of contents

Starting with NumPy Array, top

Creating an array

[1]:
```
import numpy as np
a = np.array([2,4,6,8,10])
print(a)
```

```
[ 2  4  6  8 10]
```

[2]:
```
a = np.array([2,"a",1])
print(a)
```

```
['2' 'a' '1']
```

Creating an array using arange()

[3]:
```
import numpy as np
a = np.arange(1,11)
print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

[4]:
```
a=np.arange(3,20)
print(a)
```

```
[ 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Create an array of all zeros

[5]:
```
import numpy as np

p = np.zeros((2,4))
print(p)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Create an array of all ones

[6]:
```
q = np.ones((2,3))
print(q)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

Create a constant array

[7]:
```
r = np.full((2,2), 4)
print(r)
```

```
[[4 4]
 [4 4]]
```

Create an indentity matrix

[8]:
```
s=np.eye(4)
print(s)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Create a random matrix (uniform distribution on (0,1))

[9]:
```
import numpy as np

t = np.random.random((3,3))
print(t)
```

```
[[0.29857454 0.64971355 0.54179297]
 [0.40194791 0.27904836 0.89676051]
 [0.96472405 0.06345763 0.10786494]]
```

type() and dtype functions

[10]:
```
import numpy as np
a = np.arange(1,11)
print(type(a))
```

```
<class 'numpy.ndarray'>
```

[11]:
```
print(a.dtype)
```

```
int32
```

[12]:
```
t = np.random.random((3,3))
print(t.dtype)
```

```
float64
```

Shape of an array and getting specific elements

[13]:
```
a = np.array([[5,6],[7,8]])
print(a)
```

```
[[5 6]
 [7 8]]
```

[14]:
```
a.shape
```

[14]: (2, 2)

[15]:
```
a = np.array([[5,6],[7,8]])
print(a)
print(a[0,1])
```

```
[[5 6]
 [7 8]]
6
```

[16]:
```
a = np.arange(1,11)
print(a)
a.shape
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

[16]: (10,)

[17]:
```
print(a[8])
```

```
9
```

[18]:
```
print(a[0])
```

```
1
```

Manipulating Shape of NumPy Array top

Reshaping an array

[19]:
```
import numpy as np
arr = np.arange(12)
print(arr)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

[20]:
```
new_arr=arr.reshape(2,6)
print(new_arr)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

[21]:
```
new_arr2=arr.reshape(3,4)
print(new_arr2)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

flatten/transpose/ resize an array

[22]:
```
arr=np.arange(1,10).reshape(3,3)
print(arr)
```

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

[23]:
```
print(arr.flatten())
```

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

[24]:
```
print(arr.transpose())
```

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

[25]:
```
arr.resize(1,9)
print(arr)
```

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

Stacking of Numpy arrays top

[26]:
```
arr1 = np.arange(1,10).reshape(3,3)
print(arr1)
```

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

multiplying by a number

[27]:
```
arr2 = 2*arr1
print(arr2)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

Two arrays are stacked horizontally along the x axis..

[28]:
```
arr3=np.hstack((arr1, arr2))

print(arr3)
```

```
[[ 1  2  3  2  4  6]
 [ 4  5  6  8 10 12]
 [ 7  8  9 14 16 18]]
```

Horizontal stacking using concatenate() function

[29]:
```
arr4=np.concatenate((arr1, arr2), axis=1)
print(arr4)
```

```
[[ 1  2  3  2  4  6]
 [ 4  5  6  8 10 12]
 [ 7  8  9 14 16 18]]
```

Vertical stacking

[30]:
```
arr4=np.concatenate((arr1, arr2), axis=0)
print(arr4)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

Or we can proceed as following

[31]:
```
arr5=np.vstack((arr1, arr2))
print(arr5)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

Stack by columns

[32]:
```
arr7=np.dstack((arr1, arr2))
print(arr7)
```

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

 [[ 4  8]
  [ 5 10]
  [ 6 12]]

 [[ 7 14]
  [ 8 16]
  [ 9 18]]]
```

[33]:
```
arr1 = np.arange(4,7)
print(arr1)
```

```
[4 5 6]
```

Create column stack

Create 1-D array

[34]:
```
arr2 = 2 * arr1
print(arr2)
```

```
[ 8 10 12]
```

Create column stack

[35]:
```
arr_col_stack = np.column_stack((arr1,arr2))
print(arr_col_stack)
```

```
[[ 4  8]
 [ 5 10]
 [ 6 12]]
```

[36]:
```
# Create row stack
arr_row_stack = np.row_stack((arr1,arr2))
print(arr_row_stack)
```

```
[[ 4  5  6]
 [ 8 10 12]]
```

Partitioning Numpy Array top

Perform horizontal splitting

[37]:
```
arr=np.arange(1,10).reshape(3,3)
print(arr)
```

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

[38]:
```
arr_hor_split=np.hsplit(arr, 3)

print(arr_hor_split)
```

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

Vertical split

[39]:
```
arr_ver_split=np.vsplit(arr, 3)
print(arr_ver_split)
```

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

Split with axis=0

[40]:
```
arr_split=np.split(arr,3,axis=0)
print(arr_split)
```

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

[41]:
```
# split with axis=1
np.split(arr,3,axis=1)
```

[41]:
```
[array([[1],
        [4],
        [7]]),
 array([[2],
        [5],
        [8]]),
 array([[3],
        [6],
        [9]])]
```

Changing Datatype of NumPy Arrays top

[42]:
```
arr=np.arange(1,10).reshape(3,3)
print(arr)
```

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

[43]:
```
arr.dtype
```

[43]:
```
dtype('int32')
```

Change datatype of array

[44]:
```
arr=arr.astype(float)
```

Check new data type of array

[45]:
```
print(arr.dtype)
```

```
float64
```

Convert NumPy array to Python List

[46]:
```
arr=np.arange(1,10)
list1=arr.tolist()
print(list1)
```

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

[47]:
```
arr
```

[47]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Slicing NumPy Array top

[48]:
```
arr = np.arange(10)
print(arr)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

Coefficients from the 3rd to the 6th

[49]:
```
print(arr[3:6])
```

```
[3 4 5]
```

Coefficients from the 3rd

[50]:
```
print(arr[3:])
```

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

The last 3 coefficients

[51]:
```
print(arr[-3:])
```

```
[7 8 9]
```

Boolean and Fancy Indexing top

[52]:
```
arr = np.arange(21,41,2)
print("Orignial Array:\n",arr)
```

```
Orignial Array:
 [21 23 25 27 29 31 33 35 37 39]
```

Boolean Indexing

[53]:
```
print("After Boolean Condition:",arr[arr>30])
```

```
After Boolean Condition: [31 33 35 37 39]
```

[54]:
```
arr = np.arange(1,21).reshape(5,4)
print("Orignial Array:\n",arr)
```

```
Orignial Array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
```

Selecting 2nd and 3rd row

[55]:
```
indices = [1,2]
print("Selected 1st and 2nd Row:\n", arr[indices])
```

```
Selected 1st and 2nd Row:
 [[ 5  6  7  8]
 [ 9 10 11 12]]
```

Selecting 3nd and 4th row

[56]:
```
indices = [2,3]
print("Selected 3rd and 4th Row:\n", arr[indices])
```

```
Selected 3rd and 4th Row:
 [[ 9 10 11 12]
 [13 14 15 16]]
```

Create row and column indices

[57]:
```
row = np.array([1, 2])
print(row)
```

```
[1 2]
```

[58]:
```
col = np.array([2, 3])
print(col)
```

```
[2 3]
```

[59]:
```
print("Selected Sub-Array:", arr[row, col])
```

```
Selected Sub-Array: [ 7 12]
```

Broadcasting arrays top

[60]:
```
arr1 = np.arange(1,5).reshape(2,2)
print(arr1)
```

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

[61]:
```
arr2 = np.arange(5,9).reshape(2,2)
print(arr2)
```

```
[[5 6]
 [7 8]]
```

Sum two matrices

[62]:
```
print(arr1+arr2)
```

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

Multiply two matrices: $A = (a_{ij})$ and $B = (b_{ij})$, $AB = (a_{ij}b_{ij})$ (A and B are with same dimensions)

[63]:
```
print(arr1*arr2)
```

```
[[ 5 12]
 [21 32]]
```

Add a scaler value

[64]:
```
print(arr1 + 3)
```

```
[[4 5]
 [6 7]]
```

Multiply with a scalar value

[65]:
```
print(arr1 * 3)
```

```
[[ 3  6]
 [ 9 12]]
```

Multiply matrices

[66]:
```
a = np.array([[1, 0, 4],
              [0, 1, 2],
              [0, 0, 2]])
print(a)
```

```
[[1 0 4]
 [0 1 2]
 [0 0 2]]
```

[67]:
```
a.shape
```

[67]: (3, 3)

[68]:
```
b=np.array([[2, 4],
            [1, 1],
            [3, 2]])
print(b)
```

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

[69]:
```
b.shape
```

[69]: (3, 2)

$A = (a_{ij})$ is an $n \times p$ matrix and $B = (b_{ij})$ is an $p \times q$ matrix. The $C = (c_{ij})$, where $c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$, is an $n \times q$ matrix.

[70]:
```
c=np.matmul(a,b)
print(c)
```

```
[[14 12]
 [ 7  5]
 [ 6  4]]
```