Numpy library functions P23CS005 Shashank Varnekar

In [2]:
```
import numpy as np
```

In [3]:
```
#creating arrays using numpy
arr = np.array([1, 2, 3, 4, 5, 6])
print("Arr: ", arr)

two_arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("Two dimentional Arr: \n", two_arr)
```

```
Arr:  [1 2 3 4 5 6]
Two dimentional Arr:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

In [4]:
```
print(two_arr[1])
```

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

np.zeros() creates array containing zeros of size given

In [5]:
```
arr0 = np.zeros(5)
print(arr0)
```

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

np.ones() creates array containing ones of size given

In [6]:
```
arr1 = np.ones(10)
print(arr1)
```

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

In [38]:
```
mat = np.ones((5, 6))
print(mat)
```

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

In [55]:
```
np.ones((4,4), dtype = bool)
```

Out[55]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

In [7]:
```
arr2 = np.arange(5)
print(arr2)

arr3 = np.arange(2, 11, 2)
print(arr3)
```

```
[0 1 2 3 4]
[ 2  4  6  8 10]
```

np.sort() will return sorted array

In [8]:
```
arr4 = np.array([4, 3, 6, 7, 2, 8, 1, 0, 9, 5])
arr4 = np.sort(arr4)
print(arr4)
```

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

concatenate function is used to attach one are to another

In [9]:
```
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
np.concatenate((a, b))
```

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

In [10]:
```
type(arr)
type(two_arr)
```

Out[10]:
```
numpy.ndarray
```

ndarray.size will return the total number of elements of the array

In [11]:
```
print(arr.size)
print(two_arr.size)
```

```
6
12
```

Shape function will return dimentions of array i.e. (rows,cols)

In [12]:
```
print(arr.shape)
print(arr1.shape)
print(two_arr.shape)
```

```
(6,)
(10,)
(3, 4)
```

reshape((rows, cols)) to reshape your array to x * y

In [39]:
```
arr5 = np.arange(10)
print(arr5)
```

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

In [40]:
```
b = arr5.reshape(5, 2)
print(b)
```

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

In [41]:
```
c = arr5.reshape(2, 5)
print(c)
```

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

dtype returns data type of the elements in array

In [42]:
```
print(arr.dtype)
print(c.dtype)
```

```
int32
int32
```

transpose() will return transpose of the array

In [43]:
```
c.transpose()
```

Out[43]:
```
array([[0, 5],
       [1, 6],
       [2, 7],
       [3, 8],
       [4, 9]])
```

np.empty() will generate a random array of size row * col containing of dtype values

syntax : np.empty((row,col), dtype = data_type)

In [47]:
```
np.empty((3,4), dtype = int)
```

Out[47]:
```
array([[ 1670598114, -2147483303, -2125552816,  -998675401],
       [ 1752669196,         345,  1752669220,         345],
       [         11,           0,           7,           0]])
```

Reshaping and flattening multidimensional arrays:

In [59]:
```
arr7 = np.empty((4,4), dtype = int)
print(arr7)
```

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

There are two popular ways to flatten an array: .flatten() and .ravel(). The primary difference between the two is that flatten() does not make changes in same array instead it creates new copy of the array while ravel() does operations on original array itself.

In [60]:
```
arr7.flatten()
```

Out[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [61]:
```
arr7.reshape((4,4))
```

Out[61]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [62]:
```
arr7.ravel()
```

Out[62]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Indexing and slicing

In [85]:
```
data = np.arange(1,26)
data = data.reshape((5,5))
print(data)
```

```
[[ 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]]
```

arr[row] will return row at intex [row]

In [86]:
```
data[0]
```

Out[86]:
```
array([1, 2, 3, 4, 5])
```

arr[i][j] will return element at location row [i] and col [j]

In [87]:
```
data[3][4]
```

Out[87]:
```
20
```

arr[r1:r2,c1:c2] to slice array at locations row1 to row2 and col1 to col2

In [95]:
```
data[0:4,2:5]
```

Out[95]:
```
array([[ 3,  4,  5],
       [ 8,  9, 10],
       [13, 14, 15],
       [18, 19, 20]])
```

Useful array operations

1. max() gives maximum in array
2. min() gives minimum in array
3. sum() gives addition of all array elements
4. mean() returns mean of array elements

In [96]:
```
data.max()
```

Out[96]:
```
25
```

In [97]:
```
data.min()
```

Out[97]:
```
1
```

In [98]:
```
data.sum()
```

Out[98]:
```
325
```

In [99]:
```
data.mean()
```

Out[99]: 13.0

In [100…
```
data.sum(axis = 1)
```

Out[100… array([15, 40, 65, 90, 115])