NumPy

Table of Contents

```
import numpy as np
```

1. Coverting a List to Array

```
lst = [1,2,3,4]

print(type(lst))

arr = np.array(lst)

print(type(arr))

arr = np.array(lst, ndmin = 2)

print(arr)
```

```
<class 'list'>
<class 'numpy.ndarray'>
[[1 2 3 4]]
```

2. Arrange function

The arange([start,] stop[, step,][, dtype]) : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)

```
arr = np.arange(1,10,2)
```

3. Multidimensional Array

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

arr
```

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

4. Size Function

In Python, numpy.size() function count the number of elements along a given axis (0 - Rows | 1 - Columns)

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

```
print('Total : ', arr.size)
print('Rows  : ', np.size(arr,0))
print('Cols  : ', np.size(arr,1))

arr
```

```
Total :  12
Rows  :  4
Cols  :  3

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

5. Shape Function

The shape of an array can be defined as the number of elements in each dimension. Dimension is the number of indices or subscripts, that we require in order to specify an individual element of an array.

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

print((np.size(arr,0),np.size(arr,1)))

print(arr.shape)
```

```
(4, 3)
(4, 3)
```

6. Dtype Function

Every ndarray has an associated data type (dtype) object. This data type object (dtype) informs us about the layout of the array. This means it gives us information about:

- Type of the data (integer, float, Python object, etc.)
- Size of the data (number of bytes)
- The byte order of the data (little-endian or big-endian)
- If the data type is a sub-array, what is its shape and data type?

```
arr1 = np.array([1,2,3,1,2,3])
arr2 = np.array([1.2,3.1,2.3])


print(arr1.dtype)
print(arr2.dtype)
```

```
int64
float64
```

7. Ndim Function

numpy.ndarray.ndim() function return the number of dimensions of an array.

```
arr1 = np.array([1,2,3,1,2,3])
arr2 = np.array([[1,2,3],[1,2,3]])

print(arr1.ndim)
print(arr2.ndim)

arr3 = np.array(arr1, ndmin = 3)
print(arr3.ndim)
```

```
1
2
3
```

8. Zeros Function

The numpy.zeros() function returns a new array of given shape and type, with zeros. Syntax:

numpy.zeros(shape, dtype = None)

```
arr = np.zeros(shape = (3,5), dtype = int)

print(arr)
```

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

Ones Function

The numpy.ones() function returns a new array of given shape and type, with ones.

Syntax: numpy.ones(shape, dtype = None)

```
arr = np.ones(shape = (3,5), dtype = int)

print(arr)
```

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

9. Eye Function

The eye tool returns a 2-D array with 1's as the diagonal and 0's elsewhere. The diagonal can be main, upper, or lower depending on the optional parameter k.

```
np.eye(4, dtype = int)

array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

10. Empty Function

Return am empty array of given shape and type

```
np.empty(shape = (5,5))

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.]])
```

11. Random Functions

a) np.random.rand

The numpy.random.rand() function creates an array of specified shape and fills it with random values.

```
np.random.rand(3,4)

array([[0.5868123 , 0.99141347, 0.26751161, 0.39300277],
       [0.78515141, 0.71565733, 0.23264335, 0.28439002],
       [0.27487689, 0.13062022, 0.54136802, 0.95905724]])
```

b) np.random.randint

numpy.random.randint() is one of the function for doing random sampling in numpy. It returns an array of specified shape and fills it with random integers from low (inclusive) to high (exclusive), i.e. in the interval [low, high).

```
np.random.randint(low = 1, high = 10, size = 10)

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

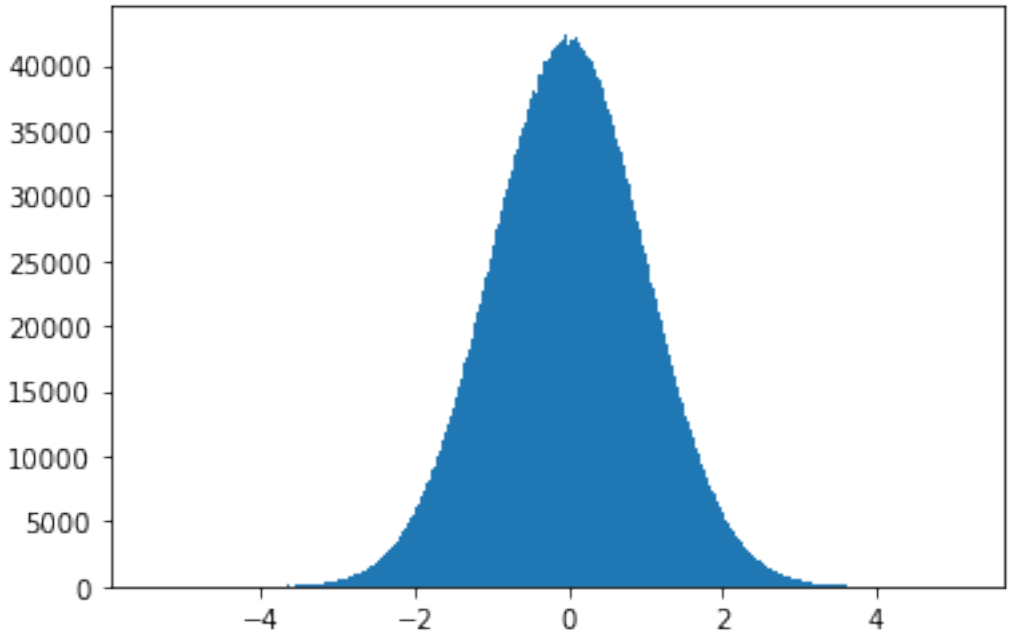
c) np.random.randn

The numpy.random.randn() function creates an array of specified shape and fills it with random values as per standard normal distribution.

```
import matplotlib.pyplot as plt

%matplotlib inline

plt.hist(np.random.randn(10000000), bins = 1000);
```



12. Reshape function - will not alter the original data

The numpy.reshape() function shapes an array without changing the data of the array.
 It should be noted that the product if the rows and columns should be same

```
arr = np.random.randint(1,15,(3,4))

print(arr.shape)
arr = arr.reshape(4,3)
print(arr.shape)
arr = arr.reshape(6,2)
print(arr.shape)
```

```
(3, 4)
(4, 3)
(6, 2)
```

13. Linspace Function

The numpy.linspace() function returns number spaces evenly w.r.t interval. Similar to numpy.arange() function but instead of step it uses sample number.

Parameters :

- -> start : [optional] start of interval range. By default start = 0
- -> stop : end of interval range
- -> restep : If True, return (samples, step). By default restep = False
- -> num : [int, optional] No. of samples to generate
- -> dtype : type of output array

```
np.linspace(1,10,5)
```

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

14. Flatten Function

ndarray.flatten() function return a copy of the array collapsed into one dimension.

```
arr = np.random.randint(1,15,(3,4))

print(arr.shape)
print(arr)
print('-'*20)
arr = arr.flatten()
print(arr.shape)
print(arr)
```

```
(3, 4)
[[ 6  3  8  3]
 [ 8  3  1  3]
 [11  5  9  3]]
--------------------
(12,)
[ 6  3  8  3  8  3  1  3 11  5  9  3]
```

15. Logspace Function

The numpy.logspace() function returns number spaces evenly w.r.t interval on a log scale.

Parameters :

- -> start : [float] start(base ** start) of interval range.
- -> stop : [float] end(base ** stop) of interval range
- -> endpoint : [boolean, optional]If True, stop is the last sample. By default, True
- -> num : [int, optional] No. of samples to generate
- -> base : [float, optional] Base of log scale. By default, equals 10.0
- -> dtype : type of output array

```
print(np.logspace(2,4,10))
print('-'*20)
print(np.logspace(2,4,10, base = 2))
```

```
[  100.           166.81005372   278.25594022   464.15888336
   774.26368268  1291.54966501  2154.43469003  3593.8136638
  5994.84250319 10000.        ]
--------------------
[ 4.          4.66611616  5.44316     6.34960421  7.4069977
8.64047791
 10.0793684  11.75787594 13.71590373 16.        ]
```

16. Copy function

If we normally use the equals sign then the **id** of both the numpy array will be same, so we need to use to **copy** function

```
arr1 = np.random.randint(1,15,10)

arr2 = arr1

print(arr1, id(arr1))
print(arr2, id(arr2))

print('-'*50)

arr1 = np.random.randint(1,15,10)
arr2 = np.copy(arr1)

print(arr1, id(arr1))
print(arr2, id(arr2))
```

```
[ 9  4  4  9 12 12 14  8  9  5] 140634733676464
[ 9  4  4  9 12 12 14  8  9  5] 140634733676464
--------------------------------------------------
[14  1  3  9  5 11 14 12 10 12] 140634733676368
[14  1  3  9  5 11 14 12 10 12] 140634952342288
```

17. arr.max() , arr.min() , arr.sum() functions

- arr.max() will give us the maximum element present in the array
- arr.min() will give us the minimun element present in the array
- arr.sum() will give us the sum of all the elements present in the array

```
arr = np.random.randint(1,30,(3,5))

print('-'*20)
print(arr)
print('-'*20)

print('MAX : ', arr.max())
```

```
print('MIN : ', arr.min())
print('SUM : ', arr.sum())
```

```
--------------------
[[19 29 28 18 24]
 [13 29  1 11 21]
 [ 3 11 28 20  7]]
--------------------
MAX :  29
MIN :  1
SUM :  262
```

We can use the axis parameter to get the max, min and sum of a particular column or row

- axis=0 is for the column
- axis=1 is for the row

```
arr = np.random.randint(1,30,(3,5))

print('-'*20)
print(arr)
print('-'*20)
```

```
--------------------
[[12 21 24  5  9]
 [16 18 17  7  7]
 [15  9  8 17 16]]
--------------------
```

For columns

```
print('MAX : ', arr.max(axis = 0))
print('MIN : ', arr.min(axis = 0))
print('SUM : ', arr.sum(axis = 0))
```

```
MAX :  [16 21 24 17 16]
MIN :  [12  9  8  5  7]
SUM :  [43 48 49 29 32]
```

Fow rows

```
print('MAX : ', arr.max(axis = 1))
print('MIN : ', arr.min(axis = 1))
print('SUM : ', arr.sum(axis = 1))
```

```
MAX :  [24 18 17]
MIN :  [5 7 8]
SUM :  [71 65 65]
```

18. Seed Function

random() function is used to generate random numbers in Python. Not actually random, rather this is used to generate pseudo-random numbers. That implies that these randomly generated numbers can be determined. random() function generates numbers for some values. This value is also called seed value.

Syntax : random.seed(l, version)

Parameter :

- l : Any seed value used to produce a random number.
- version : A integer used to specify how to convert l in a integer.

```
np.random.seed(3)

arr = np.random.randint(1,10,(3,5))

arr
```

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

19. Sorting Function

numpy.sort() : This function returns a sorted copy of an array.

Parameters :

- arr : Array to be sorted.
- axis : Axis along which we need array to be started.
- order : This argument specifies which fields to compare first.
- kind : [‘quicksort’{default}, ‘mergesort’, ‘heapsort’]Sorting algorithm.

```
np.random.seed(3)
arr = np.random.randint(1,10,(3,5))


print('-'*15)
print(arr)
print('-'*15)
print(np.sort(arr, axis = 1))
print('-'*15)
print(np.sort(arr, axis = 0))
print('-'*15)

arr = np.reshape(np.sort(arr.flatten()), arr.shape)
print(arr)
print('-'*15)
```

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

We can choose which type of sort we want to perform

```
print(np.sort(arr, axis = 0, kind = 'mergesort'))
```

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

`20. Mathematical Operations`

a) Addition

```
np.random.seed(3)

arr = np.random.randint(1,10,(3,5))

print('-' * 15)
print(arr)
print('-' * 15)
print(arr + 1)
print('-' * 15)
```

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

b) Substraction

```
print('-' * 15)
print(arr)
print('-' * 15)
print(arr - 1)
print('-' * 15)
```

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

c) Multiplication

```
print('-' * 15)
print(arr)
print('-' * 15)
print(arr * 2)
print('-' * 15)
```

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

d) Division

```
print('-' * 15)
print(arr)
print('-' * 15)
print(arr / 2)
print('-' * 15)
```

```
---------------
[[9 4 9 9 1]
 [6 4 6 8 7]
 [1 5 8 9 2]]
---------------
[[4.5 2.  4.5 4.5 0.5]
 [3.  2.  3.  4.  3.5]
```

```
 [0.5 2.5 4.  4.5 1. ]]
---------------
```

e) Matrix Multiplication

`For Matrix Multiplication we either use the`

- dot function or
- the @ symbol

```
np.random.seed(3)

arr1 = np.random.randint(1,10,(2,2))
arr2 = np.random.randint(1,10,(2,2))

print(arr1.dot(arr2))
print('-'*12)
print(arr1@arr2)
```

```
[[ 25  78]
 [ 45 108]]
------------
[[ 25  78]
 [ 45 108]]
```

f) Power

```
print(arr)
print('-'*15)
print(arr ** 2)
```

```
[[9 4 9 9 1]
 [6 4 6 8 7]
 [1 5 8 9 2]]
---------------
[[81 16 81 81  1]
 [36 16 36 64 49]
 [ 1 25 64 81  4]]
```

`21. Percentile Function`

`numpy.percentile()function used to compute the nth percentile of the given data (array elements) along the specified axis`.

```
np.random.seed(3)

arr = np.random.randint(1,10,10)
print(arr)

arr = np.sort(arr)
```

```
print(arr)

print(np.percentile(arr, 50))
```

```
[9 4 9 9 1 6 4 6 8 7]
[1 4 4 6 6 7 8 9 9 9]
6.5
```

22. Mean, Variance and Standard deviation

In NumPy, we can compute the mean, standard deviation, and variance of a given array along the second axis by two approaches first is by using inbuilt functions and second is by the formulas of the mean, standard deviation, and variance.

Using numpy.mean(), numpy.std(), numpy.var()

```
arr = np.random.randint(1,10,10)

print(arr)
print(arr.mean())
print(arr.var())
print(arr.std())
```

```
[5 4 1 1 1 1 3 3 8 1]
2.8
4.96
2.227105745132009
```

23. Filtering an numpy array

```
arr = np.arange(1,10)

print(arr)
print(arr < 5)
print(arr[arr < 5])
```

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

- We can store this into a new variable if we need this filtered array

```
arr2 = arr[arr < 5]
print(arr2)
```

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

- If we want to replace all the values above 5 with 10 we can even do that

```
arr[arr > 5] = 10

arr

array([ 1,  2,  3,  4,  5, 10, 10, 10, 10])
```

24 Transposing an array

With the help of Numpy ndarray.T object, we can make a Transpose of an array having dimension greater than or equal to 2.

```
arr = np.random.randint(1,20,(3,5))

print(arr)
print('-'*20)
print(arr.T)

[[11 18 10  1 14]
 [13  8  5 11  9]
 [13 18  1  8  1]]
--------------------
[[11 13 13]
 [18  8 18]
 [10  5  1]
 [ 1 11  8]
 [14  9  1]]
```

25. Where function

The numpy.where() function returns the indices of elements in an input array where the given condition is satisfied.

Syntax :numpy.where(condition[, x, y]) Parameters: condition : When True, yield x, otherwise yield y. x, y : Values from which to choose. x, y and condition need to be broadcastable to some shape.

Returns: out : [ndarray or tuple of ndarrays] If both x and y are specified, the output array contains elements of x where condition is True, and elements from y elsewhere.

If only condition is given, return the tuple condition.nonzero(), the indices where condition is True.

```
arr = np.random.randint(1,100,10)
arr

array([12, 36, 86, 93, 50, 73, 90, 35, 12, 90])
```

So now if we are having an even number let's convert it into 100 and the ones which are odd we leave it to the way it is

```
print(arr)

print(np.where(arr%2==0, 'even',arr))
print(np.where(arr%2!=0, 'odd',arr))

arr1 = np.where(arr%2==0, 'even',arr)
arr1 = np.where(arr%2!=0, 'odd',arr1)

print(arr1)
```

```
[12 36 86 93 50 73 90 35 12 90]
['even' 'even' 'even' '93' 'even' '73' 'even' '35' 'even' 'even']
['12' '36' '86' 'odd' '50' 'odd' '90' 'odd' '12' '90']
['even' 'even' 'even' 'odd' 'even' 'odd' 'even' 'odd' 'even' 'even']
```

26. Merging Arrays

a) Concatenate

We can perform the concatenation operation using the concatenate() function. With this function, arrays are concatenated either row-wise or column-wise, given that they have equal rows or columns respectively. Column-wise concatenation can be done by equating axis to 1 as an argument in the function.

```
arr1 = np.random.randint(1,10,(2,4))
arr2 = np.random.randint(1,10,(2,4))

print(arr1)
print('-'*15)
print(arr2)

print('-'*30)
print(np.concatenate((arr1,arr2), axis = 0))
print('-'*30)
print(np.concatenate((arr1,arr2), axis = 1))
```

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

b) Vstack and Hstack

numpy.hstack() function is used to stack the sequence of input arrays horizontally (i.e. column wise) to make a single array.

```
arr1 = np.random.randint(1,10,(2,4))
arr2 = np.random.randint(1,10,(2,4))

print(arr1)
print('-'*15)
print(arr2)
print('-'*15)

print(np.hstack((arr1,arr2)))
```

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

numpy.vstack() function is used to stack the sequence of input arrays vertically to make a single array.

```
print(arr1)
print('-'*15)
print(arr2)
print('-'*15)

print(np.vstack((arr1,arr2)))
```

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

27. Splitting Arrays

```
arr = np.random.randint(1,10,(4,4))

print(arr)
```

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

numpy.vsplit() function split an array into multiple sub-arrays vertically (row-wise). vsplit is equivalent to split with axis=0 (default), the array is always split along the first axis regardless of the array dimension.

```
np.vsplit(arr, 2)

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

numpy.hsplit() function split an array into multiple sub-arrays horizontally (column-wise). hsplit is equivalent to split with axis=1, the array is always split along the second axis regardless of the array dimension.

```
np.hsplit(arr, 2)

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

28. Turning Pics into Numpy arrays

This is picture we are having right here

Once we can turn this picture into an numpy array we can perform operations on that and pass the array even for some machine learning algorithms which will be benefitial for us

We will be using **imread** function here which belongs from the matplotlib class

```
from matplotlib.image import imread

img = imread('GFG.png')

img.shape
```

```
(607, 1000, 3)
```