Author : Suleman Sayyed

Numpy Tutorial Series

NumPy (Numerical Python) is an open source Python library which is used in almost every field of science and engineering.

What is the difference between python list and numpy array?

- NumPy gives you efficient ways of creating arrays and manipulating numerical data.
- While a **Python list can contain different data types** within a single list, all of the elements in a **NumPy array should be homogeneous**.
- mathematical operations on arrays would be **extremely inefficient** if the **arrays weren't homogeneous**.

Benifits of Numpy Array -

- Are faster and more compact than Python lists.
- An array consumes less memory and is convenient to use.
- Thus it allows the code to be optimized even further.

How to create a basic array ?

```
# basic way to create array
import numpy as np
a = np.array([1,2,3,4,5,6])
print(a)
```

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

```
# creating an array with a range of elements
b = np.arange(4)
print(b)
print(b.dtype)   # to check data type of array
```

```
[0 1 2 3]
int64
```

```
#first number, last number, and the step size.
b= np.arange(7,71,2)
print(b)
```

```
[ 7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53
 55 57 59 61 63 65 67 69]
```

```
#spaced linearly in a specified interval
np.linspace(0,10, num=5)
```

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

Now creating numpy array using functions -

1. zeros
2. Ones
3. randint
4. diag
5. rand
6. randn
7. empty (this a update functions, you will get clear down below)
8. eye

1.zeros

```
# Instead of creating an array from a sequence of elements, you can easily create an array filled with 0’s:

c= np.zeros(4)
c
```

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

```
# or we can also create a 2d array in zeros
# i.e
c= np.zeros((2,3))
c
```

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

2. ones

```
# you can easily create an array filled with 1’s:

d= np.ones(4)
d
```

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

```
d=np.ones((2,3))
d
```

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

3. randint

```
# now generaing random values with randint function
# starting value , end value, Number of values/no. we want
e = np.random.randint(1,12,5)
e

# so here it generates random values between 1 & 12, and every time we run the cell we gonna have diff values
```

```
array([4, 3, 4, 4, 3])
```

So you will see that the main diff between arrange and randint is that arrange generates definite array values, whereas randint gives us new array values everytime.

4. rand

```
a = np.random.rand(7)
a
```

```
array([0.20415958, 0.78576246, 0.67930004, 0.20658083, 0.34408882,
       0.01761399, 0.67329377])
```

```
# here you can see that rand gives us an array in floating point values
# we can also create 2d array likewise
a = np.random.rand(7,5)
a
```

```
array([[0.09901032, 0.17132599, 0.33431886, 0.44284618, 0.86877645],
       [0.21295457, 0.09731151, 0.70679743, 0.48001264, 0.3133359 ],
```

```
[0.85703798, 0.27485   , 0.62087546, 0.38712611, 0.85753564],
[0.63100471, 0.48972739, 0.70968038, 0.11997148, 0.44744661],
[0.04287981, 0.86577903, 0.44182565, 0.82375665, 0.82902621],
[0.79352377, 0.99031721, 0.32395163, 0.76614189, 0.61064741],
[0.77705463, 0.76502688, 0.86861947, 0.17595692, 0.50442361]])
```

above we see 2d array of (7,5) size array is generated

Rand function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like "numpy.zeros" and "numpy.ones".

5. randn

```
# randn gives us -ve & +ve floating point values which are always closer to zero

b = np.random.randn(5)
b
```

```
array([ 1.50735547, -2.89777049,  0.18978439,  0.36026736,  0.74717705])
```

```
#If we wanna cross check values are zero or not in large range of array like values in hundreds
c= np.random.randn(785)
np.mean(c)
```

```
-0.026925270001872496
```

6. diag

```
# it creates a matrix that will be 2d or 3d and we have to pass the daigonal values to this function
```

```
# and we have to pass values in array form in list, we can't pass values in tuple form like other functions.
d = np.diag([1,2,6,8])
d
```

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 6, 0],
       [0, 0, 0, 8]])
```

7. **empty** array

```
# numpy.empty(shape, dtype = float, order = 'C')
e = np.empty(2, dtype=int)
print(e)

f = np.empty([2,2], dtype= int)
print(f)
```

```
[94843919996544        6647137]
[[ 4609148821603546619 -4635937011834716796]
 [ 4598944035368288412  4601523871703254145]]
```

The function empty creates an array whose initial content is random and depends on the state of the memory. The reason to **use empty over zeros** (or something similar) **is speed** - just make sure to fill every element afterwards!

8. eye

```
# it is similer to diag but here we have to pass a single value or shape of array
# and we get array with daigonal values as 1 only.
```

```
e = np.eye(5)
e
```

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

above you can see that all values are zero in array except daigonal values.

you can also pass shape of the array we want and same the daigonal values will be 1 only.

```
e = np.eye(3,5)
e
```

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

Thank you for being here!