Numpy

- we have different modules in numpy to creare an arrays

by using ¶

- array()
- arange()
- zeros()
- ones()
- empty()
- linspace()
- random.rand()

By using array()

In [7]:

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8,9,10])
print(a)
```

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

In [8]:

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

Out[8]:

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

In [13]:

```
np.array([1,2,3,4,5,6,7,8,9,10],"d")
```

Out[13]:

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

In [94]:

```
np.array(['a','b','c','d','e','f','g','h','i'],"c")
```

Out[94]:

```
array([b'a', b'b', b'c', b'd', b'e', b'f', b'g', b'h', b'i'], dtype='|S1')
```

arange()

- creates array of evenly spaced values within a given interval #### syntax ##### arange([start],stop, [step],dtype=None)

In [14]:

```
np.arange(10)
```

Out[14]:

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

In [15]:

```
np.arange(1,11)
```

Out[15]:

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

In [16]:

```
np.arange(0,101,10)
```

Out[16]:

```
array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

In [17]:

```
np.arange(15.0)
```

Out[17]:

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
       13., 14.])
```

In [18]:

```
np.arange(5,dtype='complex')
```

Out[18]:

```
array([0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j])
```

In [20]:

```
np.arange(1,11,2,dtype='f')
```

Out[20]:

```
array([1., 3., 5., 7., 9.], dtype=float32)
```

zeros()

- creates an array filled with zeros #### syntax ##### zeros(shape,dtype=float,order='C')======>shape is int or tuple of ints,float==> is default type,C--->column major

In [21]:

```
np.zeros(10)
```

Out[21]:

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

In [22]:

```
np.zeros(5,dtype=int)
```

Out[22]:

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

In [25]:

```
np.zeros((4,3),dtype=int)
```

Out[25]:

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

ones()

- creates an array filled with ones #### syntax ##### ones(shape,dtype=None,order='C')

In [27]:

```
np.ones(10)
```

Out[27]:

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

In [28]:

```
np.ones(10,dtype=int)
```

Out[28]:

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

In [27]:

```
np.ones((3,4))
```

Out[27]:

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

In [28]:

```
np.ones((3,4),dtype=int)
```

Out[28]:

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

empty()

In [29]:

```
np.empty(6)
```

Out[29]:

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

In [30]:

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

Out[30]:

```
array([[-317039331, 1688006725],
       [ 282372808,  335739381]])
```

In [31]:

```
np.empty([3,3])
```

Out[31]:

```
array([[0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
       [0.00000000e+000, 0.00000000e+000, 6.81810591e-321],
       [8.10602392e-312, 0.00000000e+000, 0.00000000e+000]])
```

- linspace

- create an array filled with evenly spaced values #### syntax #####
  linspace(start,stop,num=50,endpoint=True,retstep=False,dtype=None,axis=0)

In [45]:

```
np.linspace(2.0,3.0,num=5)
```

Out[45]:

```
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
```

In [50]:

```
np.linspace(2.0,3.0,num=10,endpoint=False)
```

Out[50]:

```
array([2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

In [34]:

```
np.linspace(1,100,num=4,retstep=True)
```

Out[34]:

```
(array([  1.,  34.,  67., 100.]), 33.0)
```

In [38]:

```
np.linspace(1,1000,num=5,retstep=True)
```

Out[38]:

```
(array([   1.  ,  250.75,  500.5 ,  750.25, 1000.  ]), 249.75)
```

In [53]:

```
np.linspace(1,100,dtype=int,retstep=True) # default num is 50
```

Out[53]:

```
(array([  1,   3,   5,   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,  71,  73,  75,  77,
         79,  81,  83,  85,  87,  89,  91,  93,  95,  97, 100]),
 2.020408163265306)
```

random()

- rand()------> uniformly distributed values
- randn()------> Normally distributed values
- ranf()------> Uniformly distribute floating point values
- randint()-----> Uniformly distribute integers in a given range

In [59]:

```
help(np.random.rand)
```

```
Help on built-in function rand:

rand(...) method of numpy.random.mtrand.RandomState instance
    rand(d0, d1, ..., dn)

    Random values in a given shape.

    .. note::
        This is a convenience function for users porting code from Matlab,
        and wraps `random_sample`. That function takes a
        tuple to specify the size of the output, which is consistent with
        other NumPy functions like `numpy.zeros` and `numpy.ones`.

    Create an array of the given shape and populate it with
    random samples from a uniform distribution
    over ``[0, 1)``.

    Parameters
    ----------
    d0, d1, ..., dn : int, optional
        The dimensions of the returned array, must be non-negative.
        If no argument is given a single Python float is returned.

    Returns
    -------
    out : ndarray, shape ``(d0, d1, ..., dn)``
        Random values.

    See Also
    --------
    random

    Examples
    --------
    >>> np.random.rand(3,2)
    array([[ 0.14022471,  0.96360618],  #random
           [ 0.37601032,  0.25528411],  #random
           [ 0.49313049,  0.94909878]]) #random
```

In [63]:

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

Out[63]:

```
array([[0.13232915, 0.96371818, 0.75804319],
       [0.36663544, 0.99703371, 0.04864712],
       [0.72540174, 0.50578407, 0.09996913]])
```

In [64]:

```
np.random.rand(10)
```

Out[64]:

```
array([0.34812377, 0.80450047, 0.07969156, 0.50833441, 0.10716961,
       0.59107138, 0.27696453, 0.43016025, 0.22322489, 0.06086106])
```

In [65]:

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

Out[65]:

```
array([ 0.99430983,  0.83500875,  0.77292154, -1.0078165 , -0.13683652])
```

In [71]:

```
np.random.randint(3,size=10)
```

Out[71]:

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

In [72]:

```
np.random.ranf(10)
```

Out[72]:

```
array([0.91135684, 0.37123118, 0.44938885, 0.05555783, 0.32238656,
       0.31025001, 0.87489049, 0.36186433, 0.76430141, 0.2723183 ])
```

Attributes in Arrays

In [74]:

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

Out[74]:

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

In [75]:

```
a.ndim
```

Out[75]:

```
1
```

In [79]:

```
b=np.array([[3,3,3],[4,4,4],[5,5,5]])
b
```

Out[79]:

```
array([[3, 3, 3],
       [4, 4, 4],
       [5, 5, 5]])
```

In [80]:

```
b.ndim
```

Out[80]:

```
2
```

In [81]:

```
b.shape
```

Out[81]:

```
(3, 3)
```

In [82]:

```
a.shape
```

Out[82]:

```
(10,)
```

In [83]:

```
b.size
```

Out[83]:

```
9
```

In [84]:

```
c=np.zeros((2,3,4))
c
```

Out[84]:

```
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.]]])
```

In [85]:

```
c.shape
```

Out[85]:

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

In [86]:

```
c.size
```

Out[86]:

```
24
```

In [87]:

```
c.dtype
```

Out[87]:

```
dtype('float64')
```

In [ ]:

```

```