Day 7-Numpy Arrays

1-D Array

In [92]:
```
import numpy as np
a= np.array([[1,2,3]])
a
```

Out[92]:
```
array([[1, 2, 3]])
```

In [8]:
```
a=np.zeros(10) # here number in brackets define
#the number of zeroes
a
```

Out[8]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [9]:
```
b=np.ones(10)
b
```

Out[9]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [12]:
```
c=np.empty(0)
c
```

Out[12]:
```
array([], dtype=float64)
```

In [13]:
```
d=np.arange(6)
d
```

Out[13]:
```
array([0, 1, 2, 3, 4, 5])
```

In [15]:
```
e=np.arange (2,20,4)
e
```

Out[15]:
```
array([ 2,  6, 10, 14, 18])
```

In [22]:
```
f=np.linspace(0,20,num=9)
f
```

Out[22]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. , 12.5, 15. , 17.5, 20. ])
```

In [28]:
```
g=np.ones(10,dtype=np.float64)
g
```

Out[28]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [29]:
```
# int8, int32, float64; the number here shows the number of bits
```

2-D Array

In [14]:
```
d=np.arange(2,20)
d
```

Out[14]:
```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19])
```

In [4]:
```
a=np.array([[3,2,5],[2,4,6]])
a
```

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

a

In [32]:
```
e=np.ones((2,3))
e
```

Out[32]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [36]:
```
f=np.empty((2,4))
f
```

Out[36]:
```
array([[ 2.5,  5. ,  7.5, 10. ],
       [12.5, 15. , 17.5, 20. ]])
```

3-D Array

In 3-D Array the first variable shows the number of matrices say i, while remaining two shows rows (horizontal vectors) say j

In [35]:
```
g=np.zeros((4,2,3))
g
```

Out[35]:
```
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 [55]:
```
mat=np.arange(24).reshape(2,4,3)
mat
```

Out[55]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11]],

       [[12, 13, 14],
        [15, 16, 17],
        [18, 19, 20],
        [21, 22, 23]]])
```

N-D array

```
np.arange(number of elements).reshape(1st, 2nd, 3rd,..........,nth) such that
1st * 2nd * 3rd *,..............., * nth= number of elements (i.e rank=number of
elements)
```

It should be noted the the first dimension defines the uppermost sub-arrays following by 2nd, 3rd upto nth dimension following last in first out (LIFO) concept.

lets take an example

In below array the total number of elements or say items are 100, we have arranged them in five dimensions in a way that the multiplication of elements in each dimension i.e rank or total number of elements remain same. Here 1 * 2 * 5 * 2 * 5=100.

In [61]:
```
mat=np.arange(100).reshape(1,2,5,2,5)
mat
```

Out[61]:
```
array([[[[[ 0,  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, 26, 27, 28, 29]],

         [[30, 31, 32, 33, 34],
          [35, 36, 37, 38, 39]],

         [[40, 41, 42, 43, 44],
          [45, 46, 47, 48, 49]]],


        [[[50, 51, 52, 53, 54],
          [55, 56, 57, 58, 59]],

         [[60, 61, 62, 63, 64],
          [65, 66, 67, 68, 69]],

         [[70, 71, 72, 73, 74],
          [75, 76, 77, 78, 79]],

         [[80, 81, 82, 83, 84],
          [85, 86, 87, 88, 89]],

         [[90, 91, 92, 93, 94],
          [95, 96, 97, 98, 99]]]]])
```

For finding number of dimensions

In [63]:
```
mat.ndim
```

Out[63]:
```
5
```

So the above matrix is five dimensional.

One might thing that the world is three dimensional then how can more than three dimensional array is practical. So, for clarification here dimension is actually like the layers to reach the main elements that are arranged in a particular way. Practically objects mainly exist in three dimensions with time as fourth for a fan of Sir Albert Einstein.

For finding Shape

In [64]:
```
mat.shape
```

Out[64]:
```
(1, 2, 5, 2, 5)
```

For reshapping it

In [72]:
```
z=mat.reshape(2,10,5) # now its converted from 5 to 3 dimensions. Cooooooooool :)
z
```

Out[72]:
```
array([[[ 0,  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, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49]],

       [[50, 51, 52, 53, 54],
        [55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64],
        [65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74],
        [75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84],
        [85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94],
        [95, 96, 97, 98, 99]]])
```

In [73]:
```
z.ndim
```

Out[73]:
```
3
```

In [82]:
```
np.reshape(mat, newshape=(2,10,5), order="C")
```

Out[82]:
```
array([[[ 0,  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, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39],
        [40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49]],

       [[50, 51, 52, 53, 54],
        [55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64],
        [65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74],
        [75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84],
        [85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94],
        [95, 96, 97, 98, 99]]])
```

In [81]:
```
np.reshape(mat, newshape=(2,10,5), order="F")
```

Out[81]:
```
array([[[ 0,  1,  2,  3,  4],
        [10, 11, 12, 13, 14],
        [20, 21, 22, 23, 24],
        [30, 31, 32, 33, 34],
        [40, 41, 42, 43, 44],
        [ 5,  6,  7,  8,  9],
        [15, 16, 17, 18, 19],
        [25, 26, 27, 28, 29],
        [35, 36, 37, 38, 39],
        [45, 46, 47, 48, 49]],

       [[50, 51, 52, 53, 54],
        [60, 61, 62, 63, 64],
        [70, 71, 72, 73, 74],
        [80, 81, 82, 83, 84],
        [90, 91, 92, 93, 94],
        [55, 56, 57, 58, 59],
        [65, 66, 67, 68, 69],
        [75, 76, 77, 78, 79],
        [85, 86, 87, 88, 89],
        [95, 96, 97, 98, 99]]])
```

An Attempt to explain order C and F in Your Way

say we have (shoper, packet) elements = pakora, order C that is also by default will prioritise the shoppers for pakora distribution rather than packet like if there are 2 shoppers and each of it contain 3 small packets each containing 2 pakoras for distribution first put 2 packoras in each shopper like 2 in one packet of one shopper than 2 one paket of other shoper and repeat is process untill its over. While order F prioritise the packet over shopper so first it fill all the packets in one shopper than it do with other. So in other words 1st dimension here (n-1th dimension in general) that is rows here shoppers is prioritize in C order while F prioritise second dimension (nth dimension) here that is collumns/ packets. In place of it A just chose any of one of them keeping suitability under consideration.

I am not sure but it seems A order is used when one is not worry about which should be prioritise rows or columns.

In [101]:
```
a= np.array([1,2,3])
a
```
Out[101]:
```
array([1, 2, 3])
```

In [107]:
```
# (1,3)
b=a[np.newaxis,:]
b
```
Out[107]:
```
array([[1, 2, 3]])
```

In [103]:
```
# (3,1)
c=a[:, np.newaxis]
c
```
Out[103]:
```
array([[1],
       [2],
       [3]])
```

In [104]:
```
c.shape
```
Out[104]:
```
(3, 1)
```

In [105]:
```
b.shape
```
Out[105]:
```
(1, 3)
```

In [106]:
```
c
```
Out[106]:
```
array([[1],
       [2],
       [3]])
```

Simple way to recognise number columns is that you just count the items (most basic like here 1, 2 and 3) between two large brackets.

Indexing Slicing

```
In [108]: a
```

```
Out[108]: array([1, 2, 3])
```

```
In [109]: a[1:3]
```

```
Out[109]: array([2, 3])
```

```
In [ ]:
```