## 3-D or higher

*the term tensorflow is also used here tenserflow is a free software library for machine leraning and AI but also commonly used for 3-D or higher used operators k j i*

## Array attributes

- Dimenstions are called axis
- if single D then single axis
- if 2-d then 2 axis
- if 3-D then 3 axis

## 2-axis

- first axis has length = 2
- second axis has length = 3
- it means 2-by-3 matrix like example below

In [1]:
```
import numpy as np
c=np.array([[5,5,5],[4,4,6]])
c
```

Out[1]:
```
array([[5, 5, 5],
       [4, 4, 6]])
```

## how to create an array?

- All below are only examples of 1D but not explained by baba, he will explain in next days*

In [2]:
```
import numpy as np
a=np.array([1,2,3,4,5])
a
```

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

In [3]:
```
b=np.zeros(2)
b
```

Out[3]:
```
array([0., 0.])
```

In [4]:
```
c= np.ones(3)
c
```

Out[4]:
```
array([1., 1., 1.])
```

In [5]:
```
# creat an empty array with 2 elements
d=np.empty(3)
```

```
d
```

Out[5]:
```
array([1., 1., 1.])
```

In [8]:
```
# with range of elements
np.arange(6)
```

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

In [9]:
```
#  with specific range of element
np.arange(0,20)
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

## last digit will exclusive, means nikal dena

In [11]:
```
# with specific range of element
np.arange(3,20)
```

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

## continue array la mtlb hy k 2 se 20 tak jai

## likin har 2 number k bad (2,20,2)

In [13]:
```
# continue...
np.arange(2,20,2)
```

Out[13]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [17]:
```
# continue..(4,20,4) start with 4 to 20 but 4 gap
np.arange(4,20,4)
```

Out[17]:
```
array([ 4,  8, 12, 16])
```

## lineraly means 0 to 10 jai ga likin 5 number hongy

In [20]:
```
# lineraly spaced arrays
np.linspace(0,10, num=5)
```

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

In [21]:
```
# linearly spaced arrays
np.linspace(0,20, num=10)
```

Out[21]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [22]:
```
# specific data types in array
np.ones(5, dtype=np.int8)
```

Out[22]:
```
array([1, 1, 1, 1, 1], dtype=int8)
```

In [24]:
```
 np.ones(3,dtype=np.float64)
```

Out[24]:
```
array([1., 1., 1.])
```

## how to create an array?

## 2-D arrays

In [26]:
```
np.zeros((3,4))
```

Out[26]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [28]:
```
np.ones((5,4))
```

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

In [29]:
```
np.empty((3,4))
```

Out[29]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

## for 3-D array

In [32]:
```
# making and reshaping a 3D array
np.arange(24).reshape(2,3,4)
```

Out[32]:
```
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]]])
```

## reshape( 2 matrix, 3 rows, 4 coloums)

In [52]:
```
# making and reshaping a 3D array
#  (4,4,4) 4 means 4 matrix, 4 means rows, 4 coloumns
np.arange(64).reshape(4, 4, 4)
```

Out[52]:
```
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]]])
```

In [ ]: