In [1]:
```
import numpy as np
```

This is two dimensional array.

In [2]:
```
myarr = np.array([[3,6,37,7]] , np.int64)
```

In [3]:
```
myarr[0,1]
```

Out[3]:
```
6
```

In [4]:
```
myarr.shape
```

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

In [5]:
```
myarr.dtype
```

Out[5]:
```
dtype('int64')
```

In [6]:
```
myarr[0,1]= 45
```

In [7]:
```
myarr
```

Out[7]:
```
array([[ 3, 45, 37,  7]], dtype=int64)
```

Method to create arrays in Numpy:

1) Creating arrays using python structures such as lists and tuples etc.

In [8]:
```
listarray = np.array([[1,2,3],[5,8,5],[0,3,1]])
```

In [9]:
```
listarray
```

Out[9]:
```
array([[1, 2, 3],
       [5, 8, 5],
       [0, 3, 1]])
```

In [10]:
```
listarray.dtype
```

Out[10]:
```
dtype('int32')
```

In [11]:
```
listarray.shape
```

Out[11]:
```
(3, 3)
```

In [12]:
```
listarray.size
```

Out[12]:
```
9
```

In [13]:
```
np.array({34,23,23})
```

Out[13]:
```
array({34, 23}, dtype=object)
```

2) Intrinsic NumPy array creation functions (e.g. arange, ones, zeros, etc.)

In [14]:
```
zeros = np.zeros((2,5))
```

In [15]:
```
zeros
```

Out[15]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [16]:
```
zeros.dtype
```

Out[16]:
```
dtype('float64')
```

In [17]:
```
zeros.shape
```

Out[17]:
```
(2, 5)
```

In [18]:
```
zeros.size
```

Out[18]:
```
10
```

In [19]:
```
rng = np.arange(15)
```

In [20]:
```
rng
```

Out[20]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [21]:
```
rng.dtype
```

Out[21]:
```
dtype('int32')
```

In [22]:
```
rng.shape
```

Out[22]:
```
(15,)
```

In [23]:
```
rng.size
```

Out[23]:
```
15
```

In [24]:
```
lspace = np.linspace(1,50,10)
```

In [25]:
```
lspace
```

Out[25]:
```
array([ 1.        ,  6.44444444, 11.88888889, 17.33333333, 22.77777778,
       28.22222222, 33.66666667, 39.11111111, 44.55555556, 50.        ])
```

In [26]:
```
lspace.dtype
```

Out[26]:
```
dtype('float64')
```

In [27]:
```
lspace.shape
```

Out[27]:
```
(10,)
```

In [28]:
```
lspace.size
```

Out[28]:
```
10
```

In [29]:
```
emp = np.empty((4,6))
```

In [30]:
```
emp
```

Out[30]:
```
array([[ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  2.12199579e-314,  0.00000000e+000],
       [ 0.00000000e+000, -4.58394422e-311,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000,  0.00000000e+000]])
```

In [31]:
```
emp_like = np.empty_like(lspace)
```

In [32]:
```
emp_like
```

Out[32]:
```
array([ 1.        ,  6.44444444, 11.88888889, 17.33333333, 22.77777778,
       28.22222222, 33.66666667, 39.11111111, 44.55555556, 50.        ])
```

empty_like is used for efficency.

In [33]:
```
ide = np.identity(45)
```

In [34]:
```
ide
```

Out[34]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

In [35]:
```
ide.shape
```

Out[35]:
```
(45, 45)
```

In [36]:
```
arr = np.arange(99)
```

In [37]:
```
arr
```

Out[37]:
```
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])
```

This reshape is used to change in some other type of arrays of an existing array.

In [38]:
```
arr = arr.reshape(3,33)
```

In [39]:
```
arr
```

Out[39]:
```
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]])
```

For creating the 1D array use ravel()--

In [40]:
```
arr = arr.ravel()
```

In [41]:
```
arr
```

Out[41]:
```
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])
```

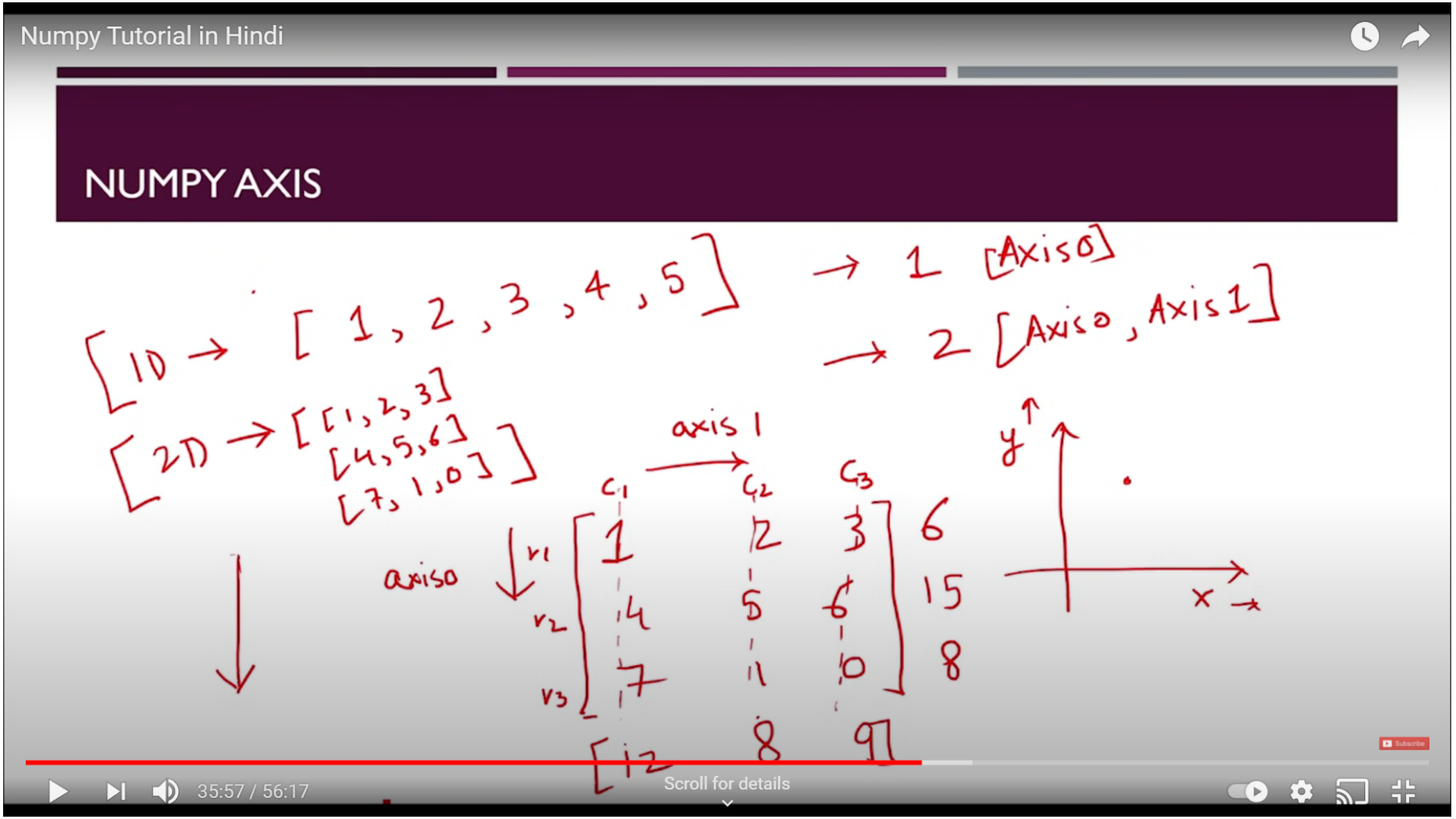
In [42]:
```
arr.shape
```

Out[42]:
```
(99,)
```

Numpy Axis

In 2D: axis0 represent the row side. axis1 represent the column side.

In 1D: There is a single axis i.e. axis0.



In [43]:
```
x = [[1,2,3],[4,5,6],[7,1,0]]
```

In [44]:
```
ar = np.array(x)
```

In [45]:
```
ar
```

```
array([[1, 2, 3],
```

Out[45]:
```
       [4, 5, 6],
       [7, 1, 0]])
```

In [46]:
```
ar.sum(axis = 0)
```

Out[46]:
```
array([12,  8,  9])
```

In [47]:
```
ar.sum(axis = 1)
```

Out[47]:
```
array([ 6, 15,  8])
```

Numpy Attributes and Methods:

1) Transpose:

In [48]:
```
ar.T
```

Out[48]:
```
array([[1, 4, 7],
       [2, 5, 1],
       [3, 6, 0]])
```

2) flat: It print the items by using the for loop

In [49]:
```
ar.flat
```

Out[49]:
```
<numpy.flatiter at 0x23bf1a88ab0>
```

In [50]:
```
for item in ar.flat:
    print(item)
```

```
1
2
3
4
5
6
7
```

```
1
0
```

3) ndim = to find the number of dimension.

In [51]:
```
ar.ndim
```

Out[51]:
```
2
```

In [52]:
```
ar.size
```

Out[52]:
```
9
```

In [53]:
```
ar.shape
```

Out[53]:
```
(3, 3)
```

In [54]:
```
ar.nbytes
```

Out[54]:
```
36
```

argmax: It gives the index of highest value in an array.

argmin: It gives the index of least value in an array.

argsort: It sort the array by give the index of the values in which order it should be sorted.

For 1D:

In [55]:
```
one = np.array([1,3,4,643,2])
```

In [56]:
```
one.argmax()
```

Out[56]:
```
3
```

In [57]:
```
one.argmin()
```

Out[57]:
```
0
```

In [58]:
```
one.argsort()
```

Out[58]:
```
array([0, 4, 1, 2, 3], dtype=int64)
```

For 2D:

In [59]:
```
ar
```

Out[59]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

For these try to imagine the array in the horizontal form by arranging the values.

In [60]:
```
ar.argmin()
```

Out[60]:
```
8
```

In [61]:
```
ar.argmax()
```

Out[61]:
```
6
```

In [62]:
```
ar.argmax(axis=0)
```

Out[62]:
```
array([2, 1, 1], dtype=int64)
```

In [63]:
```
ar.argmax(axis=1)
```

Out[63]:
```
array([2, 2, 0], dtype=int64)
```

In [64]:
```
ar.argsort(axis=1)
```

Out[64]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [2, 1, 0]], dtype=int64)
```

In [65]:
```
ar.argsort(axis=0)
```

Out[65]:
```
array([[0, 2, 2],
       [1, 0, 0],
       [2, 1, 1]], dtype=int64)
```

In [66]:
```
ar.ravel()
```

Out[66]:
```
array([1, 2, 3, 4, 5, 6, 7, 1, 0])
```

In [67]:
```
ar.reshape((9,1))
```

Out[67]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [1],
       [0]])
```

Mathematical Operations:

In [70]:
```
ar
```

Out[70]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [72]:
```
ar2 = np.array([[1,2,1], [4,0,6], [8,1,0]])
```

In [73]:
```
ar2
```

Out[73]:
```
array([[1, 2, 1],
       [4, 0, 6],
       [8, 1, 0]])
```

In [74]:
```
ar + ar2
```

Out[74]:
```
array([[ 2,  4,  4],
       [ 8,  5, 12],
       [15,  2,  0]])
```

This above operation can't be happen with list in python, if we try to add the list then it will become an extend list:

In [75]:
```
[34,53]+[55,22]
```

Out[75]:
```
[34, 53, 55, 22]
```

In [77]:
```
ar
```

Out[77]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [78]:
```
ar2
```

Out[78]:
```
array([[1, 2, 1],
       [4, 0, 6],
       [8, 1, 0]])
```

In [79]:
```
ar * ar2
```

Out[79]:
```
array([[ 1,  4,  3],
       [16,  0, 36],
       [56,  1,  0]])
```

In [80]:
```
np.sqrt(ar)
```

Out[80]:
```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974],
       [2.64575131, 1.        , 0.        ]])
```

In [81]:
```
ar.sum()
```

Out[81]:
```
29
```

In [82]:
```
ar.max()
```

Out[82]:
```
7
```

In [83]:
```
ar.min()
```

Out[83]:
```
0
```

In [84]:
```
ar
```

Out[84]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [85]:
```
np.where(ar>5)
```

Out[85]:
```
(array([1, 2], dtype=int64), array([2, 0], dtype=int64))
```

In [86]:
```
type(np.where(ar>5))
```

Out[86]:
```
tuple
```

In [87]:
```
np.count_nonzero(ar)
```

Out[87]:
```
8
```

In [88]:
```
np.nonzero(ar)
```

Out[88]:
```
(array([0, 0, 0, 1, 1, 1, 2, 2], dtype=int64),
```

```
array([0, 1, 2, 0, 1, 2, 0, 1], dtype=int64))
```

For checking is Numpy take less space:

In [89]:
```
import sys
```

In [92]:
```
py_ar = [0,4,55,2]
```

In [93]:
```
np_ar = np.array(py_ar)
```

In [95]:
```
sys.getsizeof(1) * len(py_ar)
```

Out[95]:
```
112
```

In [96]:
```
np_ar.itemsize * np_ar.size
```

Out[96]:
```
16
```

In [ ]:
```

```