In [1]:
```
# Introdution to Numpy ===>

# why Numpy is better than list ?
# (1). Same data dtype ==> Array
# (2). Memory consume ==> Numpy less , list high
# (3). Computation Power ===> Numpy high, list less
# # (4). Functions ==> Numpy high , list less.
```

In [2]:
```
import numpy as np
```

In [3]:
```
a = [1,23,56,67]
type(a)
```

Out[3]:
```
list
```

In [4]:
```
b = np.array(a)
b
```

Out[4]:
```
array([ 1, 23, 56, 67])
```

In [5]:
```
type(b)
```

Out[5]:
```
numpy.ndarray
```

In [6]:
```
a = []
size=int(input("enter the size:"))
for i in range(size):
    val =int(input("enter number:"))
    a.append(val)
a
```

```
enter the size:3
enter number:10
enter number:14
enter number:23
```

Out[6]:
```
[10, 14, 23]
```

In [7]:
```
a = []
size=int(input("enter the size:"))
for i in range(size):
    val =int(input("enter number:"))
    a.append(val)
b=np.array(a)
b
```

```
enter the size:4
enter number:12
enter number:25
enter number:54
enter number:45
```

Out[7]:
```
array([12, 25, 54, 45])
```

How to check shape and size of an array?

In [8]:
```
# esc+1 shift+enter
```

In [9]:
```
# shape = n(rows) , n(columns)

# size = total_elements ===> n(rows)*n(columns)
```

In [10]:
```
print("Total shape =", b.shape)
print("Total Elements =", b.size)
```

```
Total shape = (4,)
Total Elements = 4
```

In [11]:
```
a = [[1,2,3], [4,5,6] , [7,8,9]]
b = np.array(a)
b
```

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

In [12]:
```
print("Total shape =", b.shape)
print("Total Elements =", b.size)
```

```
Total shape = (3, 3)
Total Elements = 9
```

In [13]:
```
# r1 = [1,2,3]
# r2 = [4,5,6]
# r3 = [7,8,9]

# c1 = [1,4,7]
# c2 = [2,5,6]
# c3 = [3,6,9]
```

In [14]:
```
# Image ====> pixels ===> (0-255)px ===> 0px (complete black), 255px(white)

# Convert ===> grayscale Image ===>
# Image(Pixels) ===> Normalization(0-1) ===> opx black , 1px white

# 0 , 1 ===> Neoron System

# Matrix ===> rows , columns ===>
# Symmatric Matrix ===> n(rows) = n(columns)
#Asymmatric Matrix ===> n(rows) != n(columns)
# Diagonal Elements = [(1,1) , (2,2) ,(3,3), ........(n,n)]
```

(1). zeros() ===> It will create an array in which all the elements are zero.

In [15]:
```
a = np.zeros(4)
a
```

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

In [16]:
```
a = np.zeros((3,4))
a
```

Out[16]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

(2). Ones() ===> It will create an array in which all the values are one.

In [17]:
```
a = np.ones(3)
a
```

Out[17]:
```
array([1., 1., 1.])
```

In [18]:
```
a = np.ones((3,4))
a
```

Out[18]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

(3). eye() ===> This function will create an array in which diagonal position elements are 1 and rest all are 0.

In [19]:
```
a = np.eye(3,4)   ### Asymmatric Matrix
a
```

Out[19]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

In [20]:
```
a = np.eye(4)  ### Symmatric Matrix
a
```

Out[20]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```