WELCOM TO NUMPY TUTORIAL

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

In [2]:
```
my_arr = np.array([[1,2,3,4]],np.int8)
```

In [3]:
```
my_arr[0]
```

Out[3]:
```
array([1, 2, 3, 4], dtype=int8)
```

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

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

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

Out[5]:
```
dtype('int8')
```

In [6]:
```
my_arr[0,1]=50                  ### can change any number in array
```

In [7]:
```
my_arr
```

Out[7]:
```
array([[ 1, 50,  3,  4]], dtype=int8)
```

Array conversion from other python structurs

In [8]:
```
list_array = np.array([[1,2,3],[1,3,5],[2,5,6]])
```

In [9]:
```
list_array
```

Out[9]:
```
array([[1, 2, 3],
       [1, 3, 5],
       [2, 5, 6]])
```

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

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

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

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

In [12]:
```
zeros = np.zeros((2,5))            ### making array with zeros
```

In [13]:
```
zeros
```

Out[13]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [14]:
```
rng = np.arange(15)                ### making array of 15 elemnts
```

In [15]:
```
rng
```

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

In [16]:
```
lin_space = np.linspace(1,5,12)    ### mkaes an array of equaliy spaced 12 elemnts with range 1 to 5    [LINSPACE]
                                   ### 1 first number , 5 is last number, 12 is spacing
```

In [17]:
```
lin_space
```

Out[17]:
```
array([1.        , 1.36363636, 1.72727273, 2.09090909, 2.45454545,
       2.81818182, 3.18181818, 3.54545455, 3.90909091, 4.27272727,
       4.63636364, 5.        ])
```

In [18]:
```
emp = np.empty((4,6))              ### makes array of random veriable
                                   ### 4 is for rows and 6 is for columns (4 by 6)          [EMPTY]
```

In [19]:
```
emp
```

Out[19]:
```
array([[1.1e-321, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000],
       [0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000],
       [0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000],
       [0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000, 0.0e+000]])
```

In [20]:
```
emp_line = np.empty_like(lin_space)
```

In [21]:
```
emp_line
```

Out[21]:
```
array([1.        , 1.36363636, 1.72727273, 2.09090909, 2.45454545,
       2.81818182, 3.18181818, 3.54545455, 3.90909091, 4.27272727,
       4.63636364, 5.        ])
```

In [22]:
```
ide = np.identity(20)          ### mkaes identity array ( 20 by 20)              [IDENTITY]
ide
```

Out[22]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 1., 0.],
```

```
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1.]])
```

In [23]:
```
ide.shape
```

Out[23]:
```
(20, 20)
```

In [24]:
```
arr = np.arange(99)          ### mkaes array                                    [ARANGE]
```

In [25]:
```
arr
```

Out[25]:
```
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 [26]:
```
arr = arr.reshape(3,33)        ### rashaping array (3 by 33)              [RESHAPE]
```

In [27]:
```
arr
```

Out[27]:
```
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 [28]:
```
arr = arr.ravel()            ### convert array into one dimentional array          [RAVEL]
```

In [29]:
```
arr
```

Out[29]:
```
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 [30]:
```
arr.shape                       ### shape of array              [SHAPE]
```

Out[30]:
```
(99,)
```

Array axis

In [31]:
```
x = [[1,2,3],[4,5,6],[7,1,0]]
```

In [32]:
```
ar = np.array(x)
```

In [33]:
```
ar
```

Out[33]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [34]:
```
ar.sum(axis=0)                  ### additon of elemnts of axis 0 (1+4++7 , 2+5+1 , 3+6+0)
                                ### axis 0 is columns
```

Out[34]:
```
array([12,  8,  9])
```

In [35]:
```
ar.sum(axis=1)                  ### additon of elemnts of axis 0 (1+2+3, 4+5+6, 7+1+0)          [SUM]
                                ### axis 1 is rows
```

Out[35]:
```
array([ 6, 15,  8])
```

Atributes of numpy

In [36]:
```
ar.T                        ### transpose array ( interchange of rows and columns)
```

Out[36]:
```
array([[1, 4, 7],
       [2, 5, 1],
       [3, 6, 0]])
```

In [37]:
```
ar.flat
```

Out[37]:
```
<numpy.flatiter at 0x42db940>
```

In [38]:
```
for item in ar.flat:        ### gives all items in array
    print(item)
```

```
1
2
3
4
5
6
7
1
0
```

In [39]:
```
ar.ndim                     ### gives number of dimensions of array             [NDIM]
```

Out[39]:
```
2
```

In [40]:
```
ar.size                     ### gives number of elements in array
```

Out[40]:
```
9
```

In [41]:
```
ar.nbytes                   ### total bytes cinsume by array                    [NBYTES]
```

Out[41]:
```
36
```

In [42]:
```
one = np.array([1,23,112,12])
```

In [43]:
```
one.argmax()                ### gives index of maximum element                    [ARGMAX]
```

Out[43]:
```
2
```

In [44]:
```
one.argmin()                ### gives index of minimum element                    [ARGMIN]
```

Out[44]:
```
0
```

In [45]:
```
one.argsort()               ### rearnge array array as minimum to maximum
```

Out[45]:
```
array([0, 3, 1, 2], dtype=int64)
```

In [46]:
```
ar
```

Out[46]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [47]:
```
ar.argmin()                 ### gives index of minimum element in 2d array          [ARGMIN]
```

Out[47]:
```
8
```

In [48]:
```
ar.argmax()                 ### gives index of maximum element in 2d array          [ARGMAX]
```

Out[48]:
```
6
```

In [49]:
```
ar.argmax(axis=0)           ### gives index of max element in axis 0 (column)
```

Out[49]:
```
array([2, 1, 1], dtype=int64)
```

In [50]:
```
ar.argmax(axis=1)           ### gives index of max element in axis 1 (row)          [ARGMAX]
```

Out[50]:
```
array([2, 2, 0], dtype=int64)
```

In [51]:
```
ar.argsort(axis=0)          ### sort columns from min to max (show the indexes)        [ARGSORT]
```

Out[51]:
```
array([[0, 2, 2],
       [1, 0, 0],
       [2, 1, 1]], dtype=int64)
```

In [52]:
```
ar.argsort(axis=1)          ### sort rows from min to max (show the indexes)           [ARGSORT]
```

Out[52]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [2, 1, 0]], dtype=int64)
```

In [53]:
```
ar.ravel()                  ### array la saral karto                                   [RAVEL]
```

Out[53]:
```
array([1, 2, 3, 4, 5, 6, 7, 1, 0])
```

METRICS OPERATION

In [54]:
```
ar
```

Out[54]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [55]:
```
ar2 = np.array([[1,2,1],[4,0,6],[8,1,0]])
ar2
```

Out[55]:
```
array([[1, 2, 1],
       [4, 0, 6],
       [8, 1, 0]])
```

In [56]:
```
ar+ar2                          ### additon of array (matrics)
```

Out[56]:
```
array([[ 2,  4,  4],
       [ 8,  5, 12],
       [15,  2,  0]])
```

In [57]:
```
ar*ar2                          ### multiplication of array (matrics)
```

Out[57]:
```
array([[ 1,  4,  3],
       [16,  0, 36],
       [56,  1,  0]])
```

In [58]:
```
ar**ar2                         ### square
```

Out[58]:
```
array([[      1,       4,       3],
       [    256,       1,   46656],
       [5764801,       1,       1]], dtype=int32)
```

In [59]:
```
np.sqrt(ar)                     ### square root of array
```

Out[59]:
```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974],
       [2.64575131, 1.        , 0.        ]])
```

In [60]:
```
ar.sum()                         ### sum of all element
```

Out[60]:
```
29
```

In [61]:
```
ar
```

Out[61]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [62]:
```
np.where(ar>5)                   ### finds any element in array
```

Out[62]:
```
(array([1, 2], dtype=int64), array([2, 0], dtype=int64))
```

observation

(array([1, 2] = 1 index row , 2 index column & array([2, 0] = 2 index row , 0 index column

In [63]:
```
np.count_nonzero(ar)            ### number of zeros in array
```

Out[63]:
```
8
```

In [64]:
```
np.nonzero(ar)                  ### upper array is for axis 0 and lower for axis 1
                                ### gives location of non zero element
```

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

In [65]:
```
ar[1,2] = 0
ar
```

Out[65]:
```
array([[1, 2, 3],
       [4, 5, 0],
       [7, 1, 0]])
```

Cheking size of numpy array

In [66]:
```
import sys
```

In [67]:
```
py_ar = [0,4,55,2]
```

In [68]:
```
np_ar = np.array(py_ar)
```

In [69]:
```
sys.getsizeof(1) * len(py_ar)                ### size of pythin list      [GETSSIZEPF]
```

Out[69]:
```
112
```

In [70]:
```
np_ar.itemsize * np_ar.size                  ### size of numpy array      [ITEMSIZE]
```

Out[70]:
```
16
```

more on numpy array methods and atribute

In [71]:
```
x = np.sqrt([1+0j + 6+6j])
```

In [72]:
```
x.imag                                  ### imagnery part of array            [IMAG]
```

Out[72]:
```
array([1.05345727])
```

In [73]:
```
x.real                                  ### real part of array               [REAL]
```

Out[73]:
```
array([2.84776618])
```

In [74]:
```
x1 = np.array([[1,2,3,4,5,6],[12,1,32,5,54,5]])
```

In [75]:
```
x1.itemsize                             ### size pf array                     [ITEM_SIZE]
```

Out[75]:
```
4
```

In [76]:
```
np.all([[True,False],[True,True]])              ### Returns True if all elements evaluate to True.    [ALL]
```

Out[76]:
```
False
```

In [77]:
```
x3 = np.all([[True,False],[True,True]],axis=0)      ### Returns True if all elements evaluate to True.     [ALL]
```

In [78]:
```
np.any([[True,False],[True,True]])              ### return true if any alement evaluate to true     [ANY]
```

Out[78]:
```
True
```

In [79]:
```
np.any([[True,False],[False,False]],axis=1)         ### return true if any alement evaluate to true     [ANY]
```

Out[79]:
```
array([ True, False])
```

In [80]:
```
x3.dtype
```

Out[80]:
```
dtype('bool')
```

In [81]:
```
x = np.array([3, 4, 2, 1])                    ### Returns the indices that would partition this array.     [ARGPARTITI
x[np.argpartition(x, 3)]
```

Out[81]:
```
array([2, 1, 3, 4])
```

In [82]:
```
x4 = np.arange(10)                          ### 1 is minimum element and 8 is maximum elment in array          [CLIP]
np.clip(x4, 1, 8)
```

Out[82]:
```
array([1, 1, 2, 3, 4, 5, 6, 7, 8, 8])
```

In [83]:
```
x5 = np.array([[1, 2], [3, 4], [5, 6]])        ### Return selected slices of this array along given axis.    [COMPRESS]
print(x5)
np.compress([1,2],x5,axis=0)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

Out[83]:
```
array([[1, 2],
       [3, 4]])
```

In [84]:
```
x6 = np.array([[1, 4,2], [3,1,14]])           ### gives mean of array                    [MEAN]
np.mean(x6)
```

Out[84]:
```
4.166666666666667
```

In [85]:
```
x5.dot(x6)                                  ### dot product of two array                  [DOT]
```

Out[85]:
```
array([[ 7,  6, 30],
       [15, 16, 62],
       [23, 26, 94]])
```

In [86]:
```
x7 = np.array([[1,2,5],[21,3,21]])           ### goves standard deviation of an array      [STD]
print(np.std(x7,axis=0))
print(np.std(x7))
```

```
[10.   0.5  8. ]
8.6874750199481
```

END

In [ ]: