## CONVERSTION : FROM OTHER PYHTON STRUCTURES TO NUMPY STRUCTURE

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

In [2]:
```
import sklearn
```

In [3]:
```
myarr1 = np.array([1,2,3,4,5,6,7,8,9,100], np.int32)
```

In [4]:
```
myarr1
```

Out[4]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9, 100], dtype=int32)
```

In [5]:
```
myarr1.shape
```

Out[5]:
```
(10,)
```

In [6]:
```
myarr = np.array([[1,2,3,4,5,6,7,8,9,101]], np.int32)
```

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

Out[7]:
```
array([[  1,   2,   3,   4,   5,   6,   7,   8,   9, 101]], dtype=int32)
```

In [8]:
```
myarr.shape
```

Out[8]:
```
(1, 10)
```

In [9]:
```
myarr2 = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15],[16,17,18,19
```

In [10]:
```
myarr2
```

Out[10]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

In [11]:
```
myarr2.shape
```

Out[11]:
```
(4, 5)
```

In [12]:
```
 myarr.dtype
```

Out[12]:
```
dtype('int32')
```

In [13]:
```
myarr2[3,4] = 28
```

In [14]:
```
myarr2
```

Out[14]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 28]])
```

In [16]:
```
  myarr
```

Out[16]:
```
array([[  1,   2,   3,   4,   5,   6,   7,   8,   9, 101]], dtype=int32)
```

In [17]:
```
myarr2.size
```

Out[17]:
```
20
```

In [24]:
```
  myarr3 = np.array({34,35,36,37,38})
```

In [29]:
```
np.array({34,34,34})
```

Out[29]:
```
array({34}, dtype=object)
```

Data type Description bool_ Boolean (True or False) stored as a byte int_ Default integer type (same as C long; normally either int64 or int32) intc Identical to C int (normally int32 or int64) intp Integer used for indexing (same as C ssize_t; normally either int32 or int64) int8 Byte (-128 to 127) int16 Integer (-32768 to 32767) int32 Integer (-2147483648 to 2147483647) int64 Integer (-9223372036854775808 to 9223372036854775807) uint8 Unsigned integer (0 to 255) uint16 Unsigned integer (0 to 65535) uint32 Unsigned integer (0 to 4294967295) uint64 Unsigned integer (0 to 18446744073709551615) float_ Shorthand for float64. float16 Half precision float: sign bit, 5 bits exponent, 10 bits mantissa float32 Single precision float: sign bit, 8 bits exponent, 23 bits mantissa float64 Double precision float: sign bit, 11 bits exponent, 52 bits mantissa complex_ Shorthand for complex128. complex64 Complex number, represented by two 32-bit floats (real and imaginary components) complex128 Complex number, represented by two 64-bit floats (real and imaginary components)

https://numpy.org/doc/stable/user/absolute_beginners.html#basic-array-operations

**Some Basic Functions which we can perform in NUMPY**

In [41]:
```
zero = np.zeros((2,5)) #this will create a matrix whos size is given by u
```

In [42]:
```
zero
```

Out[42]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [36]:
```
rang = np.arange(15) #this will give us a matrix which have numbers in ra
```

In [37]:
```
rang
```

Out[37]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [44]:
```
lspace = np.linspace(1,5,8)
```

In [45]:
```
lspace
```

Out[45]:
```
array([1.        , 1.57142857, 2.14285714, 2.71428571, 3.28571429,
       3.85714286, 4.42857143, 5.        ])
```

In linspace (1,5,8) this function will divide it and the starting is from = 1 and the ending is from = 5 and the number of elements which it will give is = 8

In [56]:
```
ide = np.identity(50) # As we know the matrix identity function create a
```

In [47]:
```
print(ide)
```

```
[[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 [48]:
```
ide
```

Out[48]:
```
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 [55]:
```
emp = np.empty((11,12))#this will create an empty array which can have so
                       #on that array we can assign any sepcific values.
```

In [54]:
```
emp
```

Out[54]:
```
array([[0.00000000e+000, 3.01380044e-322, 0.00000000e+000,
        2.17213117e-314, 2.38247220e-314, 4.94065646e-324,
                    nan,             nan, 2.38233847e-314,
                    nan, 2.17213116e-314, 2.38241688e-314],
       [2.33419537e-313, 0.00000000e+000, 0.00000000e+000,
        2.38233847e-314, 2.12199579e-314, 2.15488522e-314,
        2.38241688e-314, 1.48539705e-313, 2.76676762e-322,
        0.00000000e+000, 2.38241688e-314, 2.12199579e-314],
       [2.15488522e-314, 2.38241688e-314, 6.36598737e-314,
        5.53353523e-322, 0.00000000e+000, 4.24399158e-314,
        2.12199579e-314, 2.15488525e-314, 2.38241688e-314,
        1.90979621e-313, 8.30030285e-322, 1.48219694e-323],
       [2.38233847e-314, 2.12199579e-314, 2.15488526e-314,
        2.38241689e-314, 1.90979621e-313, 1.10670705e-321,
        2.38241674e-314, 2.38233847e-314, 6.36598738e-314,
        2.15488530e-314, 2.38241689e-314, 1.90979621e-313],
       [1.38338381e-321, 9.88131292e-323, 2.38233847e-314,
        6.36598738e-314, 2.15488530e-314, 2.38241690e-314,
        2.33419537e-313, 1.66006057e-321, 2.38241674e-314,
        0.00000000e+000, 0.00000000e+000, 2.15488531e-314],
       [0.00000000e+000, 2.12199579e-313, 1.93673733e-321,
        0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
        2.38241674e-314, 0.00000000e+000, 2.38241674e-314,
        0.00000000e+000, 2.38241674e-314, 0.00000000e+000],
       [2.38241674e-314, 0.00000000e+000, 0.00000000e+000,
        2.38241674e-314,             nan, 2.19663287e-314,
        2.38241674e-314, 4.24399158e-314, 1.93673733e-321,
        2.38241674e-314, 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, 2.38241674e-314, 0.00000000e+000,
        2.38241674e-314, 0.00000000e+000, 2.38241674e-314],
       [0.00000000e+000, 2.38241674e-314, 4.94065646e-324,
        2.38233847e-314,             nan, 2.19663289e-314,
        2.38241674e-314, 1.90979621e-313, 2.92486862e-321,
        2.38241674e-314, 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, 2.38241674e-314, 0.00000000e+000,
        2.38241674e-314, 0.00000000e+000, 2.38241674e-314],
       [0.00000000e+000, 2.38241674e-314, 2.38241674e-314,
        0.00000000e+000, 0.00000000e+000,             nan,
        2.19663286e-314, 0.00000000e+000, 4.24399158e-314,
        3.87347466e-321, 0.00000000e+000, 0.00000000e+000]])
```

In [51]:
```
emp_like = np.empty_like(lspace)
```

In [52]:
```
emp_like
```

Out[52]:
```
array([1.        , 1.57142857, 2.14285714, 2.71428571, 3.28571429,
       3.85714286, 4.42857143, 5.        ])
```

In [57]:
```
arr = np.arange(99)
```

In [58]:
```
arr
```

Out[58]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5
0,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 6
7,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 8
4,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98])
```

In [60]:
```
arr.reshape(3,33) #this'll reshape the array as per the demand.
```

Out[60]:
```
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]])
```

Axis : on dimensional array have only one axis which is axis 0. If the array is two dimensional array then there are two axis - axis : 0[for row], axis : 1[for column].

# Axis

In [93]:
```
x = [[1,2,3],[4,5,6],[7,1,0]]
```

In [94]:
```
arr = np.array(x)
```

In [95]:
```
x
```

Out[95]:
```
[[1, 2, 3], [4, 5, 6], [7, 1, 0]]
```

In [96]:
```
arr
```

Out[96]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [97]:
```
arr.sum(axis = 0)
```

Out[97]:
```
array([12,  8,  9])
```

In [98]:
```
arr.sum(axis = 1)
```

Out[98]:
```
array([ 6, 15,  8])
```

In [99]:
```
arr.flat
for i in arr.flat:
    print(i)
```

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

In [100…
```
arr.T
```

Out[100]:
```
array([[1, 4, 7],
       [2, 5, 1],
       [3, 6, 0]])
```

In [101…
```
arr.ndim
```

Out[101]:
```
2
```

In [102…
```
arr.size
```

Out[102]:
```
9
```

In [103…
```
arr.shape
```

Out[103]:
```
(3, 3)
```

In [104…
```
arr.nbytes #this will show how much the bits does consume by the array.
```

Out[104]:
```
72
```

WE HAVE TO PREPARE OURSELEF WITH THE GOOD KNOWLEDGE OF *ATTRIBUTES* AND *ARGUMENTS*

In [105…
```
one = np.array([1,2,3,44,5,66,777])
```

In [106…
```
one
```

Out[106]:
```
array([  1,   2,   3,  44,   5,  66, 777])
```

In [107…
```
one.argmin()#Output will come one the behalf of indexing
```

Out[107]:
```
0
```

In [108…
```
one.argmax()#Output will come one the behalf of indexing
```

Out[108]:
```
6
```

In [109…
```
one.argsort()#Output will come one the behalf of indexing
```

Out[109]:
```
array([0, 1, 2, 4, 3, 5, 6])
```

In [110…
```
one.argmin()
```

Out[110]:
```
0
```

In [111…
```
arr.argmax(axis = 0)
```

Out[111]:
```
array([2, 1, 1])
```

In [112…
```
 arr.argmax(axis = 1)
```

Out[112]:
```
array([2, 2, 0])
```

In [115…
```
arr.argsort(axis = 1) #this will sort on the behalf of column and show th
```

Out[115]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [2, 1, 0]])
```

In [114…
```
print(arr)
```

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

In [116…
```
arr.argsort(axis = 1)
```

Out[116]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [2, 1, 0]])
```

In [117…
```
 arr.ravel()
```

Out[117]:
```
array([1, 2, 3, 4, 5, 6, 7, 1, 0])
```

In [118…
```
arr.reshape((9,1))
```

Out[118]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [1],
       [0]])
```

In [130…
```
arr
```

Out[130]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 1, 0]])
```

In [140…
```
arr1 = np.array([[1,1,1],[2,2,2],[3,3,3]])
```

In [141…
```
arr1
```

Out[141]:
```
array([[1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

In [142…
```
arr + arr1
```

Out[142]:
```
array([[ 2,  3,  4],
       [ 6,  7,  8],
       [10,  4,  3]])
```

In [143…
```
[200,100]+[5,4]
```

Out[143]:
```
[200, 100, 5, 4]
```

In [144…
```
arr*arr1
```

Out[144]:
```
array([[ 1,  2,  3],
       [ 8, 10, 12],
       [21,  3,  0]])
```

In [146…
```
arr.sum()
```

Out[146]:
```
29
```

In [148…
```
arr.min()
```

Out[148]:
```
0
```

In [149…
```
arr.max()
```

Out[149]:
```
7
```

In [150…
```
np.where(arr>5)
```

Out[150]:
```
(array([1, 2]), array([2, 0]))
```

In [152…
```
type(np.where(arr>5))
```

Out[152]:
```
tuple
```

# Just focus on the ATTRIBUTES and METHODS.