Importing NumPy

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

Creation of NumPy Arrays

1. Conversion from python structures (eg. lists, tuples and dictionaries)

In [2]:
```
myArr1 = np.array([1, 2, 3, 4])

myArr2 = np.array((1, 2, 3, 4))

myArr3 = np.array({1:"A", 2:"B", 3:"c"})
```

In [3]:
```
myArr1

myArr1.shape

myArr1.dtype

myArr1.size
```

Out[3]: 4

2. Intrinsic NumPy array creation (eg. arange, ones, zeros)

In [4]:
```
zeros = np.zeros((5, 5))
# gives a null matrix of provided shape

rng = np.arange(15)
# gives array of numbers from 0 to provided number

lspace = np.linspace(1,4,4)
# arg1, arg2 are starting no. and ending no. while arg3 is number of nos. wanted

emp = np.empty((4,6))
# makes an empty array and fills it with random values

emp_l = np.empty_like(lspace)
# takes a previous array and fills random values in the same shape

ide = np.identity(5)
# gives an identity matrix of provided shape
```

In [5]:
```
zeros
```

Out[5]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [6]:
```
rng
```

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

In [7]:
```
lspace
```

Out[7]:
```
array([1., 2., 3., 4.])
```

In [8]:
```
emp
```

Out[8]:
```
array([[ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
        -2.66192185e-310,  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, -2.71624486e-310,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000,  0.00000000e+000]])
```

In [9]:
```
emp_l
```

Out[9]:
```
array([2.12199579e-314, 4.67296746e-307, 2.68771711e-321, 7.93626426e-312])
```

In [10]:
```
ide
```

Out[10]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [11]:
```
rng_reshaped = rng.reshape(3, 5)
# reshapes the array to provided shape
rng_reshaped
```

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

In [12]:
```
rng_reshaped.ravel()
# it reshapes the reshaped array into original
```

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

NumPy Axes

In [13]:
```
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
lis = np.array(l)
lis
```

Out[13]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [14]:
```
lis.sum(axis = 0)
# axis = 0 stands for traversing all coloumns
```

Out[14]:
```
array([12, 15, 18])
```

In [15]:
```
lis.sum(axis = 1)
# axis = 1 stands for traversing all rows
```

Out[15]:
```
array([ 6, 15, 24])
```

NumPy Attributes

Attributes are functions without parenthesis

In [16]:
```
lis.T
# transposes the matrix
```

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

In [17]:
```
lis.flat
# gives an iterator for the numpy array

for item in lis.flat:
    print(item)
```

```
1
2
3
4
5
6
7
8
9
```

In [18]:
```
lis.ndim
# gives the dimension of numpy array
```

Out[18]: 2

In [19]:
```
lis.size
# gives the number of elements in the array
```

Out[19]: 9

In [20]:
```
lis.nbytes
# total bytes consumed by the array
```

Out[20]: 36

NumPy Functions

In [21]:
```
one = np.array([1, 2, 55, 4, 5])
```

In [22]:
```
one.argmax()
# returns the index of maximum element
```

Out[22]: 2

In [23]:
```
one.argmin()
# returns the index of minimum element
```

Out[23]: 0

In [24]:
```
one.argsort()
# returns the array of indices of sorted array
```

Out[24]: array([0, 1, 3, 4, 2], dtype=int64)

In [25]:
```
two = np.array([[1,7,3],
                [8,5,1],
                [3,9,2]])
```

In [26]:
```
two.argmax()
# first it straightens the array and then give index
print(two.argmax())

two.argmax(axis = 0)
# gives the array of indices of max elements from all  coloumns
```

```
7
```

Out[26]: array([1, 2, 0], dtype=int64)

In [27]:
```
two.argsort(axis = 0)
```

Out[27]:
```
array([[0, 1, 1],
       [2, 0, 2],
       [1, 2, 0]], dtype=int64)
```

NumPy Math Functions (Matrix Operations)

In [28]:
```
arr1 = np.array([[0, 1, 1],
                 [2, 0, 2],
                 [1, 2, 0]])

arr2 = np.array([[0, 3, 1],
                [9, 0, 7],
                [1, 5, 0]])
```

In [29]:
```
arr1 + arr2
```

Out[29]:
```
array([[ 0,  4,  2],
       [11,  0,  9],
       [ 2,  7,  0]])
```

In [30]:
```
arr1 * arr2
```

Out[30]:
```
array([[ 0,  3,  1],
       [18,  0, 14],
       [ 1, 10,  0]])
```

In [31]:
```
np.sqrt(arr1)
```

Out[31]:
```
array([[0.        , 1.        , 1.        ],
       [1.41421356, 0.        , 1.41421356],
       [1.        , 1.41421356, 0.        ]])
```

In [32]:
```
arr1.sum()
# gives sum of all elements of the array
```

Out[32]:
```
9
```

In [33]:
```
arr1.max()
```

Out[33]:
```
2
```

In [34]:
```
arr1.min()
```

Out[34]:
```
0
```

In [35]:
```
np.where(arr2>5)
# returns the tuple of corresponding indices where elements of arr2 are greater than 5
```

Out[35]:
```
(array([1, 1], dtype=int64), array([0, 2], dtype=int64))
```

In [36]:
```
np.nonzero(arr2)
# returns the tuple of corresponding indices where elements of arr2 are non-zero
```

Out[36]:
```
(array([0, 0, 1, 1, 2, 2], dtype=int64),
 array([1, 2, 0, 2, 0, 1], dtype=int64))
```

In [37]:
```
np.count_nonzero(arr2)
```

Out[37]:
```
6
```

How NumPy takes less memory

In [38]:
```
import sys

py_arr = [0, 1, 2, 3, 4, 5]

np_arr = np.array(py_arr)
```

In [39]:
```
sys.getsizeof(1) * len(py_arr)
# getting memory consumption of python list
```

Out[39]:
```
168
```

In [40]:
```
np_arr.itemsize * np_arr.size
# getting memory consumption of numpy array
```

Out[40]:
```
24
```

just feel the difference 😍😍😍