NumPy

You can install numpy in your anaconda environment using command 'conda install numpy'. However it comes preinstalled with Anaconda Distribution

Using NumPy

Once you've installed NumPy you can import it as a library:

In [5]:
```
import numpy as np
```

Numpy Arrays

Used majorly for linear Algebra (Arrays).

Numpy arrays essentially come in two flavors: vectors and matrices. Vectors are strictly 1-d arrays and matrices are 2-d (but you should note a matrix can still have only one row or one column).

Creating NumPy Arrays

From a Python List

We can create an array by directly converting a list or list of lists:

In [10]:
```
my_list = [1,2,3]
```

In [11]:
```
my_list
```

Out[11]:
```
[1, 2, 3]
```

In [15]:
```
np.array(my_list)
```

Out[15]:
```
array([1, 2, 3])
```

In [16]:
```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

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

In [17]:
```
np.array(my_matrix)
```

Out[17]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Built-in Methods

There are lots of built-in ways to generate Arrays

arange

Return evenly spaced values within a given interval.

In [18]:
```
np.arange(0,10)
```

Out[18]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [19]:
```
np.arange(0,11,2)
```

Out[19]:
```
array([ 0,  2,  4,  6,  8, 10])
```

zeros and ones

Generate arrays of zeros or ones

In [20]:
```
np.zeros(3)
```

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

In [21]:
```
np.zeros((5,5))
```

Out[21]:
```
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 [22]:
```
np.ones(3)
```

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

In [23]:
```
np.ones((3,3))
```

Out[23]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

linspace

Return evenly spaced numbers over a specified interval.

In [24]:
```
np.linspace(0,10,3)
```

Out[24]:
```
array([ 0.,  5., 10.])
```

In [25]:
```
np.linspace(0,10,50)
```

Out[25]:
```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

eye

Creates an identity matrix

In [26]:
```
np.eye(4)
```

Out[26]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Random

Numpy also has lots of ways to create random number arrays:

rand

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)` .

In [44]:
```
np.random.rand(2)
```

Out[44]:
```
array([0.44445836, 0.20032311])
```

In [37]:
```
np.random.rand(5,5)
```

Out[37]:
```
array([[0.91466456, 0.89574313, 0.15426897, 0.86942249, 0.60027299],
       [0.73474219, 0.43703404, 0.66335686, 0.42269452, 0.4208263 ],
       [0.60805143, 0.18327169, 0.75504794, 0.63859542, 0.61812484],
       [0.28790036, 0.01151743, 0.46309261, 0.66482193, 0.87067187],
       [0.83414071, 0.02069827, 0.69557295, 0.57360033, 0.94485293]])
```

randn

Return a sample (or samples) from the "standard normal" distribution. Unlike rand which is uniform:

In [33]:
```
np.random.randn(2)
```

Out[33]:
```
array([ 1.13116617, -0.71405307])
```

In [30]:
```
np.random.randn(5,5)
```

Out[30]:
```
array([[-1.28609395, -0.28073716,  0.92554798,  1.81667736,  1.04219416],
       [-0.29962483, -0.22594639,  0.99003951, -0.7373743 ,  1.46813496],
       [ 1.14366488,  0.63664293, -1.77842913, -0.26089925,  0.14835158],
       [-1.23332858,  0.55516323, -0.04669131,  0.65939684,  0.97694828],
       [ 0.99719045, -0.55141605, -0.72725543,  0.237765  ,  1.30571147]])
```

randint

Return random integers from `low` (inclusive) to `high` (exclusive).

In [46]:
```
np.random.randint(1,100)
```

Out[46]:
```
73
```

In [47]:
```
np.random.randint(1,100,10)
```

Out[47]:
```
array([44, 22, 68, 52, 48, 20, 83, 30, 70, 40])
```

Array Attributes and Methods

Let's discuss some useful attributes and methods or an array:

In [48]:
```
arr = np.arange(25)
ranarr = np.random.randint(0,50,10)
```

In [49]:
```
arr
```

Out[49]:
```
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])
```

In [50]:
```
ranarr
```

Out[50]:
```
array([20, 35,  1, 41, 15, 10, 11,  5, 12, 35])
```

Reshape

Returns an array containing the same data with a new shape.

In [52]:
```
arr.reshape(5,5)
```

Out[52]:
```
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]])
```

max,min,argmax,argmin

These are useful methods for finding max or min values. Or to find their index locations using argmin or argmax

In [53]:
```
ranarr
```

Out[53]:
```
array([20, 35,  1, 41, 15, 10, 11,  5, 12, 35])
```

In [54]:
```
ranarr.max()
```

Out[54]:
```
41
```

In [55]:
```
ranarr.argmax()
```

Out[55]:
```
3
```

In [57]:
```
ranarr.min()
```

Out[57]:
```
1
```

In [58]:
```
ranarr.argmin()
```

Out[58]:
```
2
```

Shape

Shape is an attribute that arrays have (not a method):

In [69]:
```
arr
```

Out[69]:
```
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])
```

In [59]:
```
# Vector
arr.shape
```

Out[59]:
```
(25,)
```

In [70]:
```
# Notice the two sets of brackets
arr.reshape(1,25)
```

Out[70]:
```
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]])
```

In [73]:
```
arr.reshape(1,25).shape
```

Out[73]:
```
(1, 25)
```

In [74]:
```
arr.reshape(5,5)
```

Out[74]:
```
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]])
```

In [75]:
```
arr.reshape(25,1).shape
```

Out[75]:
```
(25, 1)
```

dtype

You can also grab the data type of the object in the array:

In [76]:
```
arr.dtype
```

Out[76]:
```
dtype('int32')
```