ndarray

for this data strucutre we need to use numpy!

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

In [3]:
```
#when we define import with 'as' then we need to use 'np'
```

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

In [5]:
```
a
```

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

The array is not restricted to numbers (integers or float) but it can be made of strings as well

In [7]:
```
b=np.array(['dude','!','this','is','also','an','array'])
```

In [8]:
```
b
```

Out[8]:
```
array(['dude', 'this', 'is', 'also', 'an', 'array'],
      dtype='|S5')
```

The major difference from list is that arrays can contain always one type of values. They can be eather integers, or all floats, or all strings!

an array defned like this:

In [10]:
```
c=np.array([1,2,3,fox])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-25f0defd3a5b> in <module>()
----> 1 c=np.array([1,2,3,fox])

NameError: name 'fox' is not defined
```

In [11]:
```
c=np.array([1,2,3,'fox'])
```

In [12]:
```
type(c)
```

Out[12]:
```
numpy.ndarray
```

In [15]:
```
c.dtype
```

Out[15]:
```
dtype('S3')
```

The c is an array with the values in it as 'S3' which means strings of max length of 3 characters

In [18]:
```
a.dtype
```

Out[18]:
```
dtype('int64')
```

In [19]:
```
b.dtype
```

Out[19]:
```
dtype('S5')
```

This distinguishing feature of ndarrays gives speed to the program.

ndarrays are used for mathematic as well

In [21]:
```
a
```

Out[21]:
```
array([1, 2, 3, 4])
```

In [48]:
```
k=np.array([7,8,9])
```

In [49]:
```
k
```

Out[49]:
```
array([7, 8, 9])
```

In [50]:
```
a+k
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-50-13aa4dfdabbf> in <module>()
----> 1 a+k

ValueError: operands could not be broadcast together with shapes (4) (3)
```

In [51]:
```
np.append(k,10)
```

Out[51]:
```
array([ 7,  8,  9, 10])
```

In [52]:
```
len(a)
```

Out[52]:
```
4
```

In [53]:
```
len(k)
```

Out[53]:
```
3
```

In [54]:
```
k
```

Out[54]:
```
array([7, 8, 9])
```

In [55]:
```
k=np.append(k,10)
```

In [56]:
```
k
```

Out[56]:
```
array([ 7,  8,  9, 10])
```

In [57]:
```
len(k)
```

Out[57]:
```
4
```

In [58]:
```
a*k
```

Out[58]:
```
array([ 7, 16, 27, 40])
```

In [59]:
```
a+k
```

Out[59]:
```
array([ 8, 10, 12, 14])
```

In [60]:
```
a.shape
```

Out[60]:
```
(4,)
```

Matrixes

arrays are treated as matrixes, to use matrix algebra just use Dot, .T, inv, qr etc....

We can always reshape the array in any form.

In [70]:
```
a1=a.reshape(2,2)
a1
```

Out[70]:
```
array([[1, 2],
       [3, 4]])
```

In [71]:
```
k1=k.reshape(2,2)
k1
```

Out[71]:
```
array([[ 7,  8],
       [ 9, 10]])
```

In [72]:
```
np.dot(a1,k1)
```

Out[72]:
```
array([[25, 28],
       [57, 64]])
```

In [74]:
```
a1.T
```

Out[74]:
```
array([[1, 3],
       [2, 4]])
```

In [107]:
```
from numpy.linalg import inv,qr,det,eig
```

In [108]:
```
inv(k1)
```

Out[108]:
```
array([[-5. ,  4. ],
       [ 4.5, -3.5]])
```

In [109]:
```
q,r=qr(k1)
```

In [110]:
```
q
```

Out[110]:
```
array([[-0.61394061, -0.78935222],
       [-0.78935222,  0.61394061]])
```

In [111]:
```
r
```

Out[111]:
```
array([[-11.40175425, -12.80504708],
       [  0.        ,  -0.1754116 ]])
```

In [112]:
```
det(k1)
```

Out[112]:
```
-2.0000000000000089
```

We can define a matrix matrix from array, but this is slover cumputationall and matrixes can only be of two dimensions, while arrays can be as many dimensional as one wishes

In [113]:
```
matrixxx=np.matrix(np.zeros((5,5)))
```

In [114]:
```
matrixxx
```

Out[114]:
```
matrix([[ 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 [115]:
```
type(matrixxx)
```

Out[115]:
```
numpy.matrixlib.defmatrix.matrix
```

In [116]:
```
k1
```

Out[116]:
```
array([[ 7,  8],
       [ 9, 10]])
```

In [117]:
```
type(k1)
```

Out[117]:
```
numpy.ndarray
```

In [119]:
```
v,m=eig(k1)
```

In [120]:
```
v
```

Out[120]:
```
array([ -0.11684397,  17.11684397])
```

In [121]:
```
m
```

Out[121]:
```
array([[-0.7471434 , -0.62026588],
       [ 0.66466288, -0.78439164]])
```

In [ ]:
```

```