In [ ]:

1 Numpy ¶

Numpy is another module like matplotlib, csv, math that provides you with an array data structure that holds some benefits over Python lists, such as: being more compact, faster access in reading and writing items, being more convenient and more efficient.

Numpy = Numeric Python

NumPy is a Python library that is the core library for scientific computing in Python. It contains a collection of tools and techniques that can be used to solve on a computer mathematical models of problems in Science and Engineering. One of these tools is a high-performance multidimensional array object that is a powerful data structure for efficient computation of arrays and matrices. To work with these arrays, there's a huge amount of high-level mathematical functions operate on these matrices and arrays

1.0.1 What is a Numpy Array?

NumPy arrays are a like python lists. But they are different on following points.

An array is a grid of values of same type.

Eg.

```
[[1 2 3 4]
 [5 6 7 8]]
[[1 2 3 4]
 [5 6 7 8]]
[[[ 1  2  3  4]
  [ 5  6  7  8]]

 [[ 1  2  3  4]
  [ 9 10 11 12]]]
```

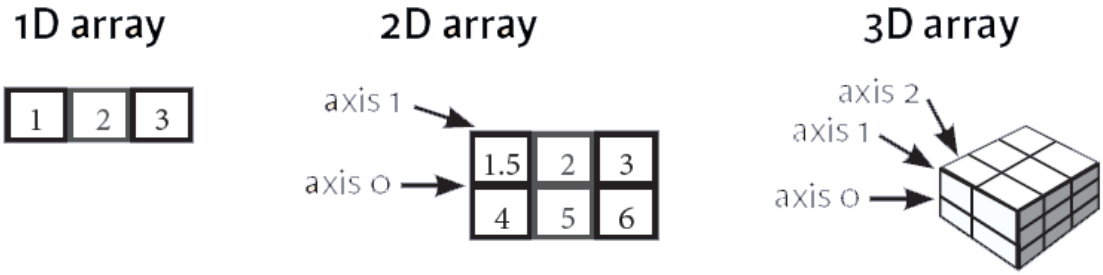
The arrays hold and represent any data in structured way.

Generally an array has few properties

1. data - where is the data? The memory location
2. datatype - what type of data ? (dtype)
3. shape - how much data?

```
my_array.data
my_array.dtype
my_array.shape
```

1.0.2 Axis



Rows are axis 0 and Columns are axis 1

1.0.3 How to make numpy arrays?

1. By Passing a list

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

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

In [2]:
```
print(a.data)
print(a.dtype)
print(a.shape)
print(type(a))
```

```
<memory at 0x10ff4d888>
int64
(3,)
<class 'numpy.ndarray'>
```

In [3]:
```
#Creating 2-D arrays
b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
print(b)
print(b.dtype)
print(b.shape)
```

```
[[ 1.5  2.   3. ]
 [ 4.   5.   6. ]]
float64
(2, 3)
```

In [4]:
```
#Creating 3D arrays
c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
dtype = float)
print(c)
print(c.dtype)
print(c.shape)
```

```
[[[ 1.5  2.   3. ]
  [ 4.   5.   6. ]]

 [[ 3.   2.   1. ]
  [ 4.   5.   6. ]]]
float64
(2, 2, 3)
```

What different datatypes can be taken. More info at the documentation (https://docs.scipy.org/doc/numpy-1.10.4/user/basics.types.html) , commonly used are

1. int64
2. int32
3. float64
4. float32
5. complex
6. bool

Many times we don't know the data that will be put inside the array so what we can do is create arrays with some value and then replace those values

```
# Create an array of ones
np.ones((3,4))

# Create an array of zeros
np.zeros((2,3,4),dtype=np.int16)

# Create an array with random values
np.random.random((2,2))

# Create an empty array
np.empty((3,2))

# Create a full array a constant value
np.full((2,2),7)

# Create an array of evenly-spaced values
np.arange(10,25,5)
```

In [5]:
```
np.ones((3,4))
```

Out[5]:
```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

In [11]:
```
np.zeros((2,3,4),dtype=np.int16)
```

Out[11]:
```
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]]], dtype=int16)
```

In [6]:
```
np.random.random((2,2))
```

Out[6]:
```
array([[ 0.80030682,  0.10652471],
       [ 0.87841342,  0.1430539 ]])
```

In [13]:
```
#what the hell this function does?

np.empty((3,3))
```

Out[13]:
```
array([[  1.72723371e-077,   1.73060100e-077,   2.30341581e-314],
       [  2.30346015e-314,   2.30346818e-314,   2.30346441e-314],
       [  2.25435565e-314,   0.00000000e+000,   2.23748262e-314]])
```

In [14]:
```
np.full((2,2),9)
```

Out[14]:
```
array([[9, 9],
       [9, 9]])
```

In [15]:
```
# Create an array of evenly-spaced values
np.arange(10,26,2)
```

Out[15]:
```
array([10, 12, 14, 16, 18, 20, 22, 24])
```

In [16]:
```
np.arange(10,26)
```

Out[16]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5])
```

In [17]:
```
np.arange(26,10,-1)
```

Out[17]:
```
array([26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 1
1])
```

In [18]:
```
#chnging the data type of the array

x = np.arange(10,25,3)
print(x)
x = x.astype(float)
print(x.dtype)
print(x)
```

```
[10 13 16 19 22]
float64
[ 10.  13.  16.  19.  22.]
```

2 Broadcasting

Allows numpy to work with arrays of different shapes for arithmetic operations

Rules for Broadcasting

The operations can be performed when the two dimension of two arrays are compatible.

1. The dimensions are compatible when they are same

eg. (3,4) and (3,4)

1. The dimensions are also compatible when one of them is 1.

In [20]:
```
## When the dimensions are equal

# Initialize `x`
x = np.ones((3,4))

# Check shape of `x`
print(x.shape)

# Initialize `y`
y = np.random.random((3,4))

# Check shape of `y`
print(y.shape)

# Add `x` and `y`
z = x + y

print(z)
print(z.shape)
```

```
(3, 4)
(3, 4)
[[ 1.52523631  1.47851231  1.03523357  1.02160921]
 [ 1.95811681  1.46177662  1.39491882  1.15447735]
 [ 1.54664765  1.06706567  1.23577024  1.21884514]]
(3, 4)
```

In [27]:
```
# Initialize `x`
x = np.ones((3,4))
x = x.T
# Check shape of `x`
print(x.shape)

# Initialize `y`
y = np.arange(3)

# Check shape of `y`
print(y.shape)
x-y
```

```
(4, 3)
(3,)
```

Out[27]:
```
array([[ 1.,  0., -1.],
       [ 1.,  0., -1.],
       [ 1.,  0., -1.],
       [ 1.,  0., -1.]])
```

In [22]:
```
# When one of the array dimension is 1
# Import `numpy` as `np`
import numpy as np

# Initialize `x`
x = np.ones((3,4))

# Check shape of `x`
print(x.shape)
print(x)
# Initialize `y`
y = np.arange(4)

# Check shape of `y`
print(y.shape)
print(y)

# Subtract `x` and `y`
z = x - y
print(z)
```

```
(3, 4)
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
(4,)
[0 1 2 3]
[[ 1.  0. -1. -2.]
 [ 1.  0. -1. -2.]
 [ 1.  0. -1. -2.]]
```

If the dimensions are not compatible then we get Value error

In [30]:
```
# Initialize `x`
x = np.ones((3,4))

# Check shape of `x`
print(x.shape)

# Initialize `y`
y = np.random.random((4,3))

# Check shape of `y`
print(y.shape)

# Subtract `x` and `y`
x - y
```

```
(3, 4)
(4, 3)

---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-30-2d3eb2d83a34> in <module>()
     12
     13 # Subtract `x` and `y`
---> 14 x - y

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

To test what the size of the resulting array is after you have done the computations! You'll see that the size is actually the maximum size along each dimension of the input arrays.

If you want to make use of broadcasting, you will rely a lot on the shape and dimensions of the arrays with which you're working. But what if the dimensions are not compatible? What if they are not equal or if one of them is not equal to 1? You'll have to fix this by manipulating your array!

2.1 Array Operations

We can use the simple operators but we have functions also

```
np.add()
np.subtract()
np.multiply()
np.divide()
np.remainder()
```

2.1.1 Mathematical operations

```
np.exp()
np.sin()
np.log()
np.cos()
np.sqrt()
np.dot()
```

2.1.2 Functions commonly used by us

function name	description
a.sum()	Array-wise sum
a.min()	Array-wise minimum value
b.max(axis=0)	Maximum value of an array row
b.cumsum(axis=1)	Cumulative sum of the elements
a.mean()	Mean
b.median()	Median
np.std(b)	Standard deviation

In [30]:
```
x = np.array([[1,2,3],[3,4,5]])
print(x)

x = x * 2
print(x)
```

```
[[1 2 3]
 [3 4 5]]
[[ 2  4  6]
 [ 6  8 10]]
```

In [33]:
```
x = np.array([[1,2,3],[3,4,5]])
y = np.array([[10,20,30],[30,40,50]])
print(x,'\n',y)

z = x * y
print(z)
```

```
[[1 2 3]
 [3 4 5]]
 [[10 20 30]
 [30 40 50]]
[[ 10  40  90]
 [ 90 160 250]]
```

In [34]:
```
x = np.array([[1,2,3],[3,4,5]])
y = np.array([[10,20,30],[30,40,50],[50,60,70]])
print(x,'\n',y)

z = np.dot(x,y)
print(z)
# np.dot(x,y)
```

```
[[1 2 3]
 [3 4 5]]
 [[10 20 30]
 [30 40 50]
 [50 60 70]]
[[220 280 340]
 [400 520 640]]
```

2.2 Subsets, Slicing and Indexing

In [35]:
```
# Regular Arrays index
# [][] [,]


y = np.array([[10,20,30],[30,40,50],[50,60,70]])

print(y)
```

```
[[10 20 30]
 [30 40 50]
 [50 60 70]]
```

In [43]:
```
# to get 40
print(y[1][1])

print( y[1,1] )

print(y[:2,1: ])
```

```
40
40
[[20 30]
 [40 50]]
```

In [47]:
```
my_filter = (y <= 40)
print(type(my_filter))
y[my_filter]
```

```
<class 'numpy.ndarray'>
```

Out[47]:
```
array([10, 20, 30, 30, 40])
```

In [41]:
```
# first column get

y[ : , 0]
```

Out[41]:
```
array([10, 30, 50])
```

In [42]:
```
#get first 2 columns
y [: ,:2]
```

Out[42]:
```
array([[10, 20],
       [30, 40],
       [50, 60]])
```

In [49]:
```
np.eye(3,3)
```

Out[49]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

In [53]:
```
# reshaping
print(y)
print(y.reshape(1,9))

#special case -1
print(y.reshape(-1))
print(y)
```

```
[[10 20 30]
 [30 40 50]
 [50 60 70]]
[[10 20 30 30 40 50 50 60 70]]
[10 20 30 30 40 50 50 60 70]
[[10 20 30]
 [30 40 50]
 [50 60 70]]
```

In [56]:
```
y = np.random.random((4,4))
y
```

Out[56]:
```
array([[ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517],
       [ 0.80189286,  0.31496101,  0.68242368,  0.46461197],
       [ 0.01226678,  0.29007112,  0.55545979,  0.02813327],
       [ 0.43018296,  0.71130723,  0.08064317,  0.52171489]])
```

In [57]:
```
y.reshape(-1)
```

Out[57]:
```
array([ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517,  0.801892
86,
        0.31496101,  0.68242368,  0.46461197,  0.01226678,  0.290071
12,
        0.55545979,  0.02813327,  0.43018296,  0.71130723,  0.080643
17,
        0.52171489])
```

In [58]:
```
y.reshape(2,2,2,2)
```

Out[58]:
```
array([[[[ 0.16158645,  0.5016914 ],
         [ 0.29069675,  0.83610517]],

        [[ 0.80189286,  0.31496101],
         [ 0.68242368,  0.46461197]]],


       [[[ 0.01226678,  0.29007112],
         [ 0.55545979,  0.02813327]],

        [[ 0.43018296,  0.71130723],
         [ 0.08064317,  0.52171489]]]])
```

In [59]:
```
y.reshape(2,2,4)
```

Out[59]:
```
array([[[ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517],
        [ 0.80189286,  0.31496101,  0.68242368,  0.46461197]],

       [[ 0.01226678,  0.29007112,  0.55545979,  0.02813327],
        [ 0.43018296,  0.71130723,  0.08064317,  0.52171489]]])
```

In [60]:
```
y.reshape(2,4,2)
```

Out[60]:
```
array([[[ 0.16158645,  0.5016914 ],
        [ 0.29069675,  0.83610517],
        [ 0.80189286,  0.31496101],
        [ 0.68242368,  0.46461197]],

       [[ 0.01226678,  0.29007112],
        [ 0.55545979,  0.02813327],
        [ 0.43018296,  0.71130723],
        [ 0.08064317,  0.52171489]]])
```

In [61]:
```
y
```

Out[61]:
```
array([[ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517],
       [ 0.80189286,  0.31496101,  0.68242368,  0.46461197],
       [ 0.01226678,  0.29007112,  0.55545979,  0.02813327],
       [ 0.43018296,  0.71130723,  0.08064317,  0.52171489]])
```

In [62]:
```
y.reshape(16,1)
```

Out[62]:
```
array([[ 0.16158645],
       [ 0.5016914 ],
       [ 0.29069675],
       [ 0.83610517],
       [ 0.80189286],
       [ 0.31496101],
       [ 0.68242368],
       [ 0.46461197],
       [ 0.01226678],
       [ 0.29007112],
       [ 0.55545979],
       [ 0.02813327],
       [ 0.43018296],
       [ 0.71130723],
       [ 0.08064317],
       [ 0.52171489]])
```

In [63]:
```
y.reshape(2,4,2).reshape(-1)
```

Out[63]:
```
array([ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517,  0.801892
86,
        0.31496101,  0.68242368,  0.46461197,  0.01226678,  0.290071
12,
        0.55545979,  0.02813327,  0.43018296,  0.71130723,  0.080643
17,
        0.52171489])
```

In [64]:
```
y.reshape(4,4).reshape(-1)
```

Out[64]:
```
array([ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517,  0.801892
86,
        0.31496101,  0.68242368,  0.46461197,  0.01226678,  0.290071
12,
        0.55545979,  0.02813327,  0.43018296,  0.71130723,  0.080643
17,
        0.52171489])
```

In [67]:
```
np.eye(5,3)*6
```

Out[67]:
```
array([[ 6.,  0.,  0.],
       [ 0.,  6.,  0.],
       [ 0.,  0.,  6.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

In [68]:
```
y
```

Out[68]:
```
array([[ 0.16158645,  0.5016914 ,  0.29069675,  0.83610517],
       [ 0.80189286,  0.31496101,  0.68242368,  0.46461197],
       [ 0.01226678,  0.29007112,  0.55545979,  0.02813327],
       [ 0.43018296,  0.71130723,  0.08064317,  0.52171489]])
```

In [69]:
```
y.sum()
```

Out[69]:
```
6.68374849467875
```

In [70]:
```
y.sum(axis=1)
```

Out[70]:
```
array([ 1.79007976,  2.26388952,  0.88593096,  1.74384825])
```

In [71]:
```
y.sum(axis = 0)
```

Out[71]:
```
array([ 1.40592905,  1.81803076,  1.60922339,  1.8505653 ])
```

In [72]:
```
y.mean(axis=1)[0]
```

Out[72]:
```
0.44751994052880983
```

In [74]:
```
y[0].sum()
```

Out[74]:
```
1.7900797621152393
```

In [75]:
```
y[:,0].sum()
```

Out[75]:
```
1.4059290477493025
```

In [76]:
```
from sklearn import datasets
```

In [77]:
```
iris_data = datasets.load_iris()
```

In [83]:
```
X = iris_data["data"]
X
```

Out[83]:
```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1],
       [ 5.4,  3.7,  1.5,  0.2],
       [ 4.8,  3.4,  1.6,  0.2],
       [ 4.8,  3. ,  1.4,  0.1],
       [ 4.3,  3. ,  1.1,  0.1],
       [ 5.8,  4. ,  1.2,  0.2],
       [ 5.7,  4.4,  1.5,  0.4],
       [ 5.4,  3.9,  1.3,  0.4],
       [ 5.1,  3.5,  1.4,  0.3],
       [ 5.7,  3.8,  1.7,  0.3],
       [ 5.1,  3.8,  1.5,  0.3],
```

In [85]:
```
X[:,-1]
```

Out[85]:
```
array([ 0.2,  0.2,  0.2,  0.2,  0.2,  0.4,  0.3,  0.2,  0.2,  0.1,
0.2,
        0.2,  0.1,  0.1,  0.2,  0.4,  0.4,  0.3,  0.3,  0.3,  0.2,
0.4,
        0.2,  0.5,  0.2,  0.2,  0.4,  0.2,  0.2,  0.2,  0.2,  0.4,
0.1,
        0.2,  0.1,  0.2,  0.2,  0.1,  0.2,  0.2,  0.3,  0.3,  0.2,
0.6,
        0.4,  0.3,  0.2,  0.2,  0.2,  0.2,  1.4,  1.5,  1.5,  1.3,
1.5,
        1.3,  1.6,  1. ,  1.3,  1.4,  1. ,  1.5,  1. ,  1.4,  1.3,
1.4,
        1.5,  1. ,  1.5,  1.1,  1.8,  1.3,  1.5,  1.2,  1.3,  1.4,
1.4,
        1.7,  1.5,  1. ,  1.1,  1. ,  1.2,  1.6,  1.5,  1.6,  1.5,
1.3,
        1.3,  1.3,  1.2,  1.4,  1.2,  1. ,  1.3,  1.2,  1.3,  1.3,
1.1,
        1.3,  2.5,  1.9,  2.1,  1.8,  2.2,  2.1,  1.7,  1.8,  1.8,
2.5,
        2. ,  1.9,  2.1,  2. ,  2.4,  2.3,  1.8,  2.2,  2.3,  1.5,
2.3,
        2. ,  2. ,  1.8,  2.1,  1.8,  1.8,  1.8,  2.1,  1.6,  1.9,
2. ,
        2.2,  1.5,  1.4,  2.3,  2.4,  1.8,  1.8,  2.1,  2.4,  2.3,
1.9,
        2.3,  2.5,  2.3,  1.9,  2. ,  2.3,  1.8])
```

In [ ]:
```

```