Python Programming

by Narendra Allam

Chapter 10

Numpy

Topics Covering

- Numpy Arrays
  - double dimension arrays
  - resizing, reshaping
  - vector multiplication
  - boolean filtering
  - querying using where() function
  - indexing
  - slicing
  - mean, median, stndard deviation, average
  - Transpose
  - Broadcasting
- Nimpy matrix
  - addition, multiplication
  - trnaspose, inverse
- Numpy random module

What's NumPy?

NumPy is a Python extension to add support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions.

In[]
```
x = 20
```

In[]
```
import sys
```

In[]
```
sys.getsizeof(x)
```

```
Output: 24
```

```
In[] l = [2, 3, 4 ,5]
     print l
```

```
[2, 3, 4, 5]
```

```
In[] from array import array
```

```
In[] a = array('H', [2, 3, 4, 5])
```

```
In[] a
```

```
Output: array('H', [2, 3, 4, 5])
```

```
In[] l[2]
```

```
Output: 4
```

```
In[] sys.getsizeof(l)
```

```
Output: 104
```

```
In[] sys.getsizeof(a)
```

```
Output: 64
```

```
In[] a[2]
```

```
Output: 4
```

```
In[] a[2:4]
```

```
Output: array('H', [4, 5])
```

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

```
In[] a = np.array([2,3,4,5,7])
```

```
In[] a
```

```
Output: array([2, 3, 4, 5, 7])
```

```
In[] a.shape
```

```
Output: (5,)
```

```
In[] a.dtype
```

```
Output: dtype('int64')
```

In[]
```
a.ndim
```
Output: 1

In[]
```
a.size
```
Output: 5

In[]
```
a.nbytes
```
Output: 40

In[]
```
a = np.array(range(10))
```

In[]
```
a.dtype
```
Output: dtype('int64')

In[]
```
a
```
Output: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In[]
```
a = np.array(range(10), dtype=float)
a.dtype
```
Output: dtype('float64')

In[]
```
a
```
Output: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

In[]
```
a = np.array((3, 5))
a.dtype, a.nbytes
```
Output: (dtype('int64'), 16)

In[]
```
a = np.array([2+3j, 4+5j])
a.nbytes
```
Output: 32

In[]
```
a = np.array([True, False, True])
a.nbytes
```
Output: 3

In[]
```
a = np.array(['Apple', 'Banana', 'Tender Coconut'], dtype='S20')
a.dtype, a.nbytes
```

```
Output: (dtype('S20'), 60)
```

In[]
```
print(a)
a.dtype
```

```
['Apple' 'Banana' 'Tender Coconut']
```

Output: dtype('S20')

In[]
```
import numpy as np
a = np.array(range(10), dtype='uint64')
```

In[]
```
a
```

Output: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint64)

Multi-dimension arrays:

In[]
```
l = [2, 3, 4]
ll = [[[2, 3, 4],
       [4, 5, 6]],
      [[2, 3, 4],
       [4, 5, 6]],
      [[2, 3, 4],
       [4, 5, 6]]
     ]

a = np.array(ll)
a.shape
```

Output: (3, 2, 3)

datatypes

- bool: Boolean (True or False) stored as a bit
- inti: Platform integer (normally either int32 or int64)
- int8: Byte (-128 to 127)
- int16: Integer (-32768 to 32767)
- int32: Integer (-2 **31 to 2** 31 -1)
- int64: Integer (-2 **63 to 2** 63 -1)
- uint8: Unsigned integer (0 to 255)
- uint16: Unsigned integer (0 to 65535)
- uint32: Unsigned integer (0 to 2 ** 32 - 1)
- uint64: Unsigned integer (0 to 2 ** 64 - 1)
- float16: Half precision float: sign bit, 5 bits exponent, and 10 bits mantissa
- float32: Single precision float: sign bit, 8 bits exponent, and 23 bits mantissa
- float64 or float: Double precision float: sign bit, 11 bits exponent, and 52 bits mantissa
- complex64 Complex number, represented by two 32-bit floats (real and imaginary components)
- complex128 or complex: Complex number, represented by two 64-bit floats (real and imaginary components)
- SN: String with N characters,i.e, 'S20', means string with width 20 characters

dtype Character codes

Type code	C Type	Minimum size in bytes
'b'	signed integer	1
'B'	unsigned integer	1
'u'	Unicode character	2 (see note)
'h'	signed integer	2
'H'	unsigned integer	2
'i'	signed integer	2
'I'	unsigned integer	2
'l'	signed integer	4
'L'	unsigned integer	4
'q'	signed integer	8 (see note)
'Q'	unsigned integer	8 (see note)
'f'	floating point	4
'd'	floating point	8

```
In[] a = np.array(range(10))
```

In[]
```
a
```

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

In[]
```
a = np.arange(1,11,3, dtype='uint32')
```

In[]
```
a
```

Output:
```
array([ 1,  4,  7, 10], dtype=uint32)
```

In[]
```
a = np.empty(4)
print a.dtype
print a
```

```
float64
[ 0.  0.  0.  0.]
```

In[]
```
a = np.empty((4, 3, 2))
print a.dtype
print a
```

```
float64
[[[ -2.68156159e+154  -2.68156159e+154]
  [  6.95212893e-310   2.15027482e-314]
  [  2.16272729e-314   6.95212893e-310]]

 [[  2.15094736e-314   2.15615136e-314]
  [ -2.68156159e+154  -2.68156159e+154]
  [ -2.68156159e+154   1.27319747e-313]]

 [[  1.27319747e-313   1.27319747e-313]
  [  1.27319747e-313   1.27319747e-313]
  [ -2.68156159e+154  -2.68156159e+154]]

 [[  2.59270999e-313   2.15027482e-314]
  [  2.12199580e-314   3.81959242e-313]
  [ -2.68679809e+154   1.67315047e-308]]]
```

In[]
```
a = np.zeros((3, 5), dtype='uint64')
print a.dtype
print a
```

```
uint64
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

```
In[]
```

```
In[] np.ones((4, 2, 3))
```

```
Output: array([[[ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],

               [[ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],

               [[ 1.,  1.,  1.],
                [ 1.,  1.,  1.]],

               [[ 1.,  1.,  1.],
                [ 1.,  1.,  1.]]])
```

```
In[] np.identity(4)
```

```
Output: array([[ 1.,  0.,  0.,  0.],
               [ 0.,  1.,  0.,  0.],
               [ 0.,  0.,  1.,  0.],
               [ 0.,  0.,  0.,  1.]])
```

```
In[] a = np.arange(1, 25)
     a
```

```
Output: array([ 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[] a.reshape((4, 3, 2))
```

```
Output: array([[[ 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[] a.reshape((2, 12))
```

```
Output: array([[ 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[]
```
a
```

Output:
```
array([ 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[]
```
a.resize(4, 6)
```

In[]
```
a
```

Output:
```
array([[ 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[]
```
import numpy as np
a = np.arange(24).reshape(4, 6)
```

In[]
```
import numpy as np
a = np.arange(24)
```

In[]
```
print a.reshape(6, 4)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

In[]
```
print a.reshape(2,3,4)
```

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

In[]
```
print a
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
22 23]
```

In[]
```
a = np.arange(24)
a.resize(4, 3)
```

```
In[] print a
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
In[] a.flatten()
```

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

```
In[] a
```

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

```
In[] a.resize(a.size)
```

```
In[] print a
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
In[] a.resize(a.size)
```

```
In[] print a
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
In[] b = a.ravel()
print b
c = a.flatten()
print c
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

In[]
```
for x in a.ravel():
    print x
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

Note:

The difference is that flatten always returns a copy and ravel returns a view of the original array whenever possible.

This isn't visible in the printed output, but if you modify the array returned by ravel, it may modify the entries in the

original array. If you modify the entries in an array returned from flatten this will never happen. ravel will often be faster

since no memory is copied, but you have to be more careful about modifying the array it returns.

In[]
```
l = [
    ( 0,  1,  2,  3,  4,  5),
    ( 6,  7,  8,  9, 10, 11),
    [12, 13, 14, 15, 16, 17],
    [18, 19, 20, 21, 22, 23]
    ]
a = np.array(l)
```

In[]
```
a
```

Output:
```
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]])
```

In[]
```
a[1][4]
```

Output:
```
10
```

```
In[] a[1, 4]
```

```
Output: 10
```

```
In[]
```

```
In[] a[1, :5]
```

```
Output: array([ 6,  7,  8,  9, 10])
```

```
In[] a[:, :4]
```

```
Output: array([[ 0,  1,  2,  3],
               [ 6,  7,  8,  9],
               [12, 13, 14, 15],
               [18, 19, 20, 21]])
```

```
In[] a[:, -1]
```

```
Output: array([ 5, 11, 17, 23])
```

```
In[] a[:, ::-1]
```

```
Output: array([[ 5,  4,  3,  2,  1,  0],
               [11, 10,  9,  8,  7,  6],
               [17, 16, 15, 14, 13, 12],
               [23, 22, 21, 20, 19, 18]])
```

```
In[] a
```

```
Output: 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]])
```

```
In[] a[:, ::-2]
```

```
Output: array([[ 5,  3,  1],
               [11,  9,  7],
               [17, 15, 13],
               [23, 21, 19]])
```

```
In[] a
```

```
Output: 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]])
```

```
In[] a[1, 4] = 999
     print a
```

```
[[  0   1   2   3   4   5]
 [  6   7   8   9 999  11]
 [ 12  13  14  15  16  17]
```

```
[ 18  19  20  21  22  23]]
```

In[]
```
a = np.arange(1, 25).reshape(4, 6)
print a
```

```
[[ 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[]
```
a[1, :4] = 999
print a
```

```
[[  1   2   3   4   5   6]
 [999 999 999 999  11  12]
 [ 13  14  15  16  17  18]
 [ 19  20  21  22  23  24]]
```

In[]
```
a[1, :4] = [1, 2, 3, 4]
print a
```

```
[[ 1  2  3  4  5  6]
 [ 1  2  3  4 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

In[]
```
a[1, :4] = a[2, 2:]
print a
```

```
[[ 1  2  3  4  5  6]
 [15 16 17 18 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

In[]
```
a[:2, :4] = a[2:, 2:]
```

In[]
```
a
```

Output:
```
array([[15, 16, 17, 18,  5,  6],
       [21, 22, 23, 24, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24]])
```

In[]
```
c = np.sin(a)
print c
```

```
[[ 0.65028784 -0.28790332 -0.96139749 -0.75098725 -0.95892427  -0.2
794155 ]
 [ 0.83665564 -0.00885131 -0.8462204  -0.90557836 -0.99999021 -0.5
3657292]
 [ 0.42016704  0.99060736  0.65028784 -0.28790332 -0.96139749 -0.7
5098725]
 [ 0.14987721  0.91294525  0.83665564 -0.00885131 -0.8462204  -0.9
0557836]]
```

In[]
```
c
```

Output:
```
array([[ 0.65028784, -0.28790332, -0.96139749, -0.75098725, -0.958
92427,
        -0.2794155 ],
       [ 0.83665564, -0.00885131, -0.8462204 , -0.90557836, -0.999
99021,
        -0.53657292],
       [ 0.42016704,  0.99060736,  0.65028784, -0.28790332, -0.961
39749,
        -0.75098725],
       [ 0.14987721,  0.91294525,  0.83665564, -0.00885131, -0.846
2204 ,
        -0.90557836]])
```

In[]
```
b = c < 0
print b
```

```
[[False  True  True  True  True  True]
 [False  True  True  True  True  True]
 [False False False  True  True  True]
 [False False False  True  True  True]]
```

In[]
```
c[b]
```

Output:
```
array([-0.28790332, -0.96139749, -0.75098725, -0.95892427, -0.2794
155 ,
       -0.00885131, -0.8462204 , -0.90557836, -0.99999021, -0.5365
7292,
       -0.28790332, -0.96139749, -0.75098725, -0.00885131, -0.8462
204 ,
       -0.90557836])
```

In[]
```
c[c < 0]
```

Output:
```
array([-0.28790332, -0.96139749, -0.75098725, -0.95892427, -0.2794
155 ,
       -0.00885131, -0.8462204 , -0.90557836, -0.99999021, -0.5365
7292,
       -0.28790332, -0.96139749, -0.75098725, -0.00885131, -0.8462
204 ,
       -0.90557836])
```

In[]
```
c[c < 0] = 0
```

In[]
```
print c
```

```
[[ 0.65028784  0.          0.          0.          0.          0.
]
 [ 0.83665564  0.          0.          0.          0.          0.
]
 [ 0.42016704  0.99060736  0.65028784  0.          0.          0.
]
 [ 0.14987721  0.91294525  0.83665564  0.          0.          0.
]]
```

In[]
```
c[(c < 0) | (c > 0.9)] = 0
print c
```

```
[[ 0.65028784  0.          0.          0.          0.          0.
]
 [ 0.83665564  0.          0.          0.          0.          0.
]
 [ 0.42016704  0.          0.65028784  0.          0.          0.
]
 [ 0.14987721  0.          0.83665564  0.          0.          0.
]]
```

In[]
```
c[(c > 0) & (c <= 0.9)] = 1
print c
```

```
[[ 1.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.]
 [ 1.  0.  1.  0.  0.  0.]
 [ 1.  0.  1.  0.  0.  0.]]
```

In[]
```
c = np.arange(1, 25).reshape(4, 6)
```

In[]
```
np.where(c > 17)
```

Output:
```
(array([2, 3, 3, 3, 3, 3, 3]), array([5, 0, 1, 2, 3, 4, 5]))
```

In[]
```
x, y = np.where(c > 17)
zip(x, y)
```

Output:
```
[(2, 5), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5)]
```

In[]
```
x = np.arange(1, 13).reshape(3, 4)
y = np.arange(13, 25).reshape(3,  4)
```

In[]
```
x
```

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

In[]
```
y
```

Output:
```
array([[13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In[]
```
b = (x%2 == 0)
```

In[]
```
x[b] = y[b]
```

In[]
```
x
```

Output:
```
array([[ 1, 14,  3, 16],
       [ 5, 18,  7, 20],
       [ 9, 22, 11, 24]])
```

In[]
```
x * -1
```

Output:
```
array([[ -1, -14,  -3, -16],
       [ -5, -18,  -7, -20],
       [ -9, -22, -11, -24]])
```

In[]
```
a = np.arange(1, 10).reshape(3, 3)
b = np.arange(10, 19).reshape(3,  3)
```

In[]
```
print a
print b
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[10 11 12]
 [13 14 15]
```

```
[16 17 18]]
```

In[]
```
a + b
```

Output:
```
array([[11, 13, 15],
       [17, 19, 21],
       [23, 25, 27]])
```

In[]
```
a * b
```

Output:
```
array([[ 10,  22,  36],
       [ 52,  70,  90],
       [112, 136, 162]])
```

In[]
```
print a
print b
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

In[]
```
a.dot(b)
```

Output:
```
array([[ 84,  90,  96],
       [201, 216, 231],
       [318, 342, 366]])
```

In[]
```
a.transpose()
```

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

In[]
```
a.T
```

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

In[]
```
a.std()
```

Output:
```
2.5819888974716112
```

In[]
```
a.var()
```

Output:
```
6.666666666666667
```

In[]
```
a.mean()
```

Output: 5.0

In[]
```
np.average(a)
```

Output: 5.0

In[]
```
a = np.arange(1,10).reshape(3, 3)
print a
a.all()
```

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

Output: True

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

Output:
```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

In[]
```
a.any()
```

Output: True

In[]
```
a = np.arange(9).reshape(3, 3)
b = np.arange(9, 18).reshape(3, 3)
```

In[]
```
print "----a-----"
print a
print "----b-----"
print b
np.hstack((a, b))
```

```
----a-----
[[0 1 2]
 [3 4 5]
 [6 7 8]]
----b-----
[[ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

Output:
```
array([[ 0,  1,  2,  9, 10, 11],
       [ 3,  4,  5, 12, 13, 14],
       [ 6,  7,  8, 15, 16, 17]])
```

In[]
```
c = np.vstack((a, b))
```

```
In[] c
```

```
Output: array([[ 0,  1,  2],
               [ 3,  4,  5],
               [ 6,  7,  8],
               [ 9, 10, 11],
               [12, 13, 14],
               [15, 16, 17]])
```

Matrices

```
In[] a = np.mat('4 3 1; 2 1 8; 6 5 4')
     b = np.mat([[2, 5, 6], [2, 1, 5], [7, 8, 9]])
     c = np.mat(np.arange(9).reshape(3, 3))
     a
```

```
Output: matrix([[4, 3, 1],
                [2, 1, 8],
                [6, 5, 4]])
```

```
In[] b
```

```
Output: matrix([[2, 5, 6],
                [2, 1, 5],
                [7, 8, 9]])
```

```
In[] c
```

```
Output: matrix([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])
```

```
In[] b.I
```

```
Output: matrix([[-0.4025974 ,  0.03896104,  0.24675325],
                [ 0.22077922, -0.31168831,  0.02597403],
                [ 0.11688312,  0.24675325, -0.1038961 ]])
```

```
In[] b.T
```

```
Output: matrix([[2, 2, 7],
                [5, 1, 8],
                [6, 5, 9]])
```

```
In[] a=np.mat('4 3; 2 1')
     b=np.mat('1 2; 3 4')
```

```
In[] a + b
```

```
Output: matrix([[5, 5],
                [5, 5]])
```

```
In[] a * b
```

```
Output: matrix([[13, 20],
                [ 5,  8]])
```

numpy random module

```
In[] np.random.rand(10)
```

```
Output: array([ 0.72224113,  0.74252841,  0.94850873,  0.60095589,  0.5044
        5489,
                0.28350123,  0.0888207 ,  0.76315723,  0.16674291,  0.8490
        3059])
```

```
In[] np.random.randint(1,101, size=10)
```

```
Output: array([66, 42, 41, 96, 92, 45, 42, 45, 96, 19])
```

```
In[] np.random.randint(1,11, size=20)
```

```
Output: array([ 4,  9,  7,  5,  4,  6,  9,  5,  7,  9, 10,  4,  2,  2,  7,
        8,  9,
                2,  6,  4])
```

```
In[] np.random.randint(1,101, size=10).reshape(2,5)
```

```
Output:  array([[19, 94, 54,  8, 42],
                [82, 75, 13, 92, 31]])
```

```
In[] a = np.mat(np.random.randint(1, 10, size=12).reshape(3,  4))
     b = np.mat(np.random.randint(1, 10, size=12).reshape(4,  3))
     print a
     print b
     print a * b
```

```
[[8 2 1 4]
 [9 4 6 5]
 [2 8 5 1]]
[[8 1 7]
 [6 8 1]
 [2 2 7]
 [9 8 1]]
```

```
[[114  58  69]
 [153  93 114]
 [ 83  84  58]]
```

In[]
```
np.random.sample((3, 4))
```

Output:
```
array([[ 0.62064205,  0.65436972,  0.92891408,  0.85935699],
       [ 0.59320707,  0.36582383,  0.02786175,  0.51467625],
       [ 0.70468277,  0.59664169,  0.55583458,  0.74305521]])
```

In[]
```
np.random.random_sample((3, 4))
```

Output:
```
array([[ 0.15756279,  0.54173593,  0.41419547,  0.80864341],
       [ 0.47666514,  0.99995442,  0.18649099,  0.86560101],
       [ 0.52619223,  0.2976447 ,  0.51789312,  0.1841071 ]])
```

In[]
```
c = np.random.random_sample((3, 4))
l = c.tolist()
print l
```

```
[[0.7776321165664242, 0.5711208312405456, 0.08763610390430387,  0.5
709511923795006], [0.33579726554314704, 0.6574265226117838,  0.5255
651327549744, 0.05165223582886724], [0.9302852340785452,  0.2367638
508841542, 0.7656118749678794, 0.976596013428897]]
```

In[]
```
c.shape
```

Output:
```
(3, 4)
```

In[]
```
import numpy as np
c = np.array([[0, 1, 2],
              [3, 4, 5],
              [6, 7, 8]])
print c
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In[]
```
c * [5, 3, 1]
```

Output:
```
array([[ 0,  3,  2],
       [15, 12,  5],
       [30, 21,  8]])
```

In[]
```
[5, 3, 1] * c
```

Output:
```
array([[ 0,  3,  2],
       [15, 12,  5],
       [30, 21,  8]])
```

```
In[] c
```

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

```
In[] c * [[5],
          [3],
          [1]]
```

```
Output:  array([[ 0,  5, 10],
                [ 9, 12, 15],
                [ 6,  7,  8]])
```

Broadcasting Rules

In order for an operation to broadcast, the size of all the trailing dimensions for both arrays must either: be equal OR one of them must be one

> A (1d array): 3
> B (2d array): 2 x 3

Result (2d array): 2 x 3

> A (2d array): 6 x 1
> B (3d array): 1 x 6 x 4

Result (3d array): 1 x 6 x 4

> A (4d array): 3 x 1 x 6 x 1
> B (3d array): 2 x 1 x 4

Result (4d array): 3 x 2 x 6 x 4

np.linspace() devides the range into n equal partitions and retuns list of points.

```
In[] np.linspace(1, 100, 15)
```

```
Output:  array([   1.        ,    8.07142857,   15.14285714,   22.21428571,
                  29.28571429,   36.35714286,   43.42857143,   50.5       ,
                  57.57142857,   64.64285714,   71.71428571,   78.78571429,
                  85.85714286,   92.92857143,  100.        ])
```