Python Programming

by Narendra Allam

Chapter 10.1

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 [1]:

```
x = 20
```

In [2]:

```
import sys
sys.getsizeof(x)
```

Out[2]:

```
28
```

In [3]:

```
x.bit_length()
```

Out[3]:

```
5
```

In [4]:

```
l = [2, 3, 4 ,5]
print(l)
```

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

In [5]:

```
from array import array
```

In [6]:

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

In [7]:

```
a
```

Out[7]:

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

In [8]:

```
l[2]
```

Out[8]:

```
4
```

In [9]:

```
sys.getsizeof(l)
```

Out[9]:

```
96
```

In [10]:

```
sys.getsizeof(a)
```

Out[10]:

```
72
```

In [11]:

```
a[2]
```

Out[11]:

```
4
```

In [12]:

```
a[2:4] # slicing works
```

Out[12]:

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

In [13]:

```
import numpy as np
```

In [14]:

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

In [15]:

```
a
```

Out[15]:

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

In [16]:

```
a.shape
```

Out[16]:

```
(5,)
```

In [17]:

```
a.dtype
```

Out[17]:

```
dtype('int64')
```

In [18]:

```
a.ndim
```

Out[18]:

```
1
```

In [19]:

```
a.size
```

Out[19]:

```
5
```

In [20]:

```
a.nbytes
```

Out[20]:

```
40
```

In [21]:

```
a = np.array(range(10))
```

In [22]:

```
a.dtype
```

Out[22]:

```
dtype('int64')
```

In [23]:

```
a
```

Out[23]:

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

In [24]:

```
a = np.array(range(10), dtype=float)
a.dtype
```

Out[24]:

```
dtype('float64')
```

In [25]:

```
a
```

Out[25]:

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

In [26]:

```
a.dtype, a.nbytes
```

Out[26]:

```
(dtype('float64'), 80)
```

In [27]:

```
a = np.array((3, 5, 7, 1))
```

In [28]:

```
a.dtype, a.nbytes
```

Out[28]:

```
(dtype('int64'), 32)
```

In [29]:

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

Out[29]:

```
32
```

In [30]:

```
a = np.array([True, False, True])
a.nbytes
```

Out[30]:

```
3
```

In [31]:

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

Out[31]:

```
(dtype('<U14'), 168)
```

In [32]:

```
a
```

Out[32]:

```
array(['Apple', 'Banana', 'Tender Coconut'], dtype='<U14')
```

In [33]:

```
print(a)
a.dtype
```

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

Out[33]:

```
dtype('<U14')
```

In [34]:

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

In [35]:

```
a
```

Out[35]:

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

Multi-dimension arrays:

In [36]:

```
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
```

Out[36]:

```
(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 ASCII characters,i.e, 'S20', means string with width 20 characters
- UN: String with N UNICODE characters,i.e, 'U20', 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 [37]:

```
a = np.array(range(1, 11, 3))
print(a)
```

```
[ 1  4  7 10]
```

In [38]:

```
a = np.arange(1,11,3, dtype='uint32')
print(a)
```

```
[ 1  4  7 10]
```

In [39]:

```
a = np.empty(4)
print (a.dtype)
print (a)
```

```
float64
[4.9e-324 2.0e-323 3.5e-323 4.9e-323]
```

In [40]:

```
a = np.empty((4, 3))
print (a.dtype)
print (a)
```

```
float64
[[-3.10503618e+231 -2.68677853e+154  2.37663529e-312]
 [ 2.14321575e-312  2.37663529e-312  2.56761491e-312]
 [ 8.48798317e-313  9.33678148e-313  1.08221785e-312]
 [ 8.70018274e-313  3.99910963e+252  8.34404849e-309]]
```

In [41]:

```
a = np.empty((4, 3), dtype='int32')
print (a.dtype)
print (a)
```

```
int32
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

In [42]:

```
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 [43]:

```
np.ones((4, 3))
```

Out[43]:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [44]:

```
np.identity(4)
```

Out[44]:

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

In [45]:

```
a = np.arange(1, 25)
a
```

Out[45]:

```
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 [46]:

```
a.reshape((4, 3, 2))
```

Out[46]:

```
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 [47]:

```
a.reshape((2, 12))
```

Out[47]:

```
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 [48]:

```
a
```

Out[48]:

```
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 [49]:

```
a.resize(4, 6)
```

In [50]:

```
a
```

Out[50]:

```
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 [51]:

```
import numpy as np
a = np.arange(24).reshape(4, 6)
a
```

Out[51]:

```
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 [52]:

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

In [53]:

```
a.reshape(6, 4)
```

Out[53]:

```
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 [54]:

```
a.reshape(2,3,4)
```

Out[54]:

```
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 [55]:

```
a
```

Out[55]:

```
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 [56]:

```
a = np.arange(24)
a.resize(4, 3) # modifies actual array
```

In [57]:

```
a
```

Out[57]:

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

In [58]:

```
a.flatten()
```

Out[58]:

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

In [59]:

```
a
```

Out[59]:

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

In [60]:

```
a.resize(a.size)
```

In [61]:

```
a
```

Out[61]:

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

In [62]:

```
b = a.ravel()
print (b) # produces view of the array
c = a.flatten() # takes a copy of the array
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 [63]:

```
# traversing array using ravel
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 [64]:

```
import numpy as np
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 [65]:

```
a
```

Out[65]:

```
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 [66]:

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

Out[66]:

```
10
```

In [67]:

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

Out[67]:

```
10
```

In [68]:

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

Out[68]:

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

In [69]:

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

Out[69]:

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

In [70]:

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

Out[70]:

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

In [71]:

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

Out[71]:

```
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 [72]:

```
a
```

Out[72]:

```
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 [73]:

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

Out[73]:

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

In [74]:

```
a
```

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]])
```

In [75]:

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

Out[75]:

```
array([[  0,   1,   2,   3,   4,   5],
       [  6,   7,   8,   9, 999,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23]])
```

Replacing a slice with a value

In [76]:

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

Out[76]:

```
array([[  0,   1,   2,   3,   4,   5],
       [999, 999, 999, 999, 999,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23]])
```

Replacing a slice with a sequence

In [77]:

```
a[1, :4] = [55, 88, 77, 66]
a
```

Out[77]:

```
array([[  0,   1,   2,   3,   4,   5],
       [ 55,  88,  77,  66, 999,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23]])
```

Replacing a slice with another slice

In [78]:

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

Out[78]:

```
array([[  0,   1,   2,   3,   4,   5],
       [ 14,  15,  16,  17, 999,  11],
       [ 12,  13,  14,  15,  16,  17],
       [ 18,  19,  20,  21,  22,  23]])
```

Swapping:

In [79]:

```
a = np.arange(1, 25).reshape(4, 6)
a
```

Out[79]:

```
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 [80]:

```
a[1, 2] , a[2, 5] =   a[2, 5], a[1, 2]
```

In [81]:

```
a
```

Out[81]:

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8, 18, 10, 11, 12],
       [13, 14, 15, 16, 17,  9],
       [19, 20, 21, 22, 23, 24]])
```

In [82]:

```
# Applying math function sin()
c = np.sin(a)
c
```

Out[82]:

```
array([[ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.9589242
7,
        -0.2794155 ],
       [ 0.6569866 ,  0.98935825, -0.75098725, -0.54402111, -0.9999902
1,
        -0.53657292],
       [ 0.42016704,  0.99060736,  0.65028784, -0.28790332, -0.9613974
9,
         0.41211849],
       [ 0.14987721,  0.91294525,  0.83665564, -0.00885131, -0.8462204
,
        -0.90557836]])
```

In [83]:

```
c
```

Out[83]:

```
array([[ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.9589242
7,
        -0.2794155 ],
       [ 0.6569866 ,  0.98935825, -0.75098725, -0.54402111, -0.9999902
1,
        -0.53657292],
       [ 0.42016704,  0.99060736,  0.65028784, -0.28790332, -0.9613974
9,
         0.41211849],
       [ 0.14987721,  0.91294525,  0.83665564, -0.00885131, -0.8462204
,
        -0.90557836]])
```

In [84]:

```
c < 0
```

Out[84]:

```
array([[False, False, False,  True,  True,  True],
       [False, False,  True,  True,  True,  True],
       [False, False, False,  True,  True, False],
       [False, False, False,  True,  True,  True]])
```

In [85]:

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

Out[85]:

```
array([-0.7568025 , -0.95892427, -0.2794155 , -0.75098725, -0.5440211
1,
       -0.99999021, -0.53657292, -0.28790332, -0.96139749, -0.0088513
1,
       -0.8462204 , -0.90557836])
```

In [86]:

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

In [87]:

```
c
```

Out[87]:

```
array([[0.84147098, 0.90929743, 0.14112001, 0.        , 0.        ,
        0.        ],
       [0.6569866 , 0.98935825, 0.        , 0.        , 0.        ,
        0.        ],
       [0.42016704, 0.99060736, 0.65028784, 0.        , 0.        ,
        0.41211849],
       [0.14987721, 0.91294525, 0.83665564, 0.        , 0.        ,
        0.        ]])
```

In [88]:

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

Out[88]:

```
array([[0.84147098, 0.        , 0.14112001, 0.        , 0.        ,
        0.        ],
       [0.6569866 , 0.        , 0.        , 0.        , 0.        ,
        0.        ],
       [0.42016704, 0.        , 0.65028784, 0.        , 0.        ,
        0.41211849],
       [0.14987721, 0.        , 0.83665564, 0.        , 0.        ,
        0.        ]])
```

In [89]:

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

Out[89]:

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

Where exactly this happend ?

In [90]:

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

Out[90]:

```
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 [91]:

```
np.where(c > 17)
```

Out[91]:

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

In [92]:

```
x, y = np.where(c > 17)
print ("List of indices where c > 17:")
[(i, j) for i, j in zip(x, y)]
```

```
List of indices where c > 17:
```

Out[92]:

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

In [93]:

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

Out[93]:

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

In [94]:

```
y
```

Out[94]:

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

In [95]:

```
b = (x%2 == 0)
```

In [96]:

```
b
```

Out[96]:

```
array([[False,  True, False,  True],
       [False,  True, False,  True],
       [False,  True, False,  True]])
```

In [97]:

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

In [98]:

```
x
```

Out[98]:

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

Scalar multiplcation

In [99]:

```
x * -1
```

Out[99]:

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

In [100]:

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

In [101]:

```
print(a)
print(b)
```

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

Array addition

In [102]:

```
a + b
```

Out[102]:

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

Element wise multiplication

In [103]:

```
a * b
```

Out[103]:

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

In [104]:

```
print (a)
print (b)
```

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

In [105]:

```
a.dot(b) # matrix multiplication
```

Out[105]:

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

In [106]:

```
a.transpose()
```

Out[106]:

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

In [107]:

```
a.T # transpose
```

Out[107]:

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

In [108]:

```
a.std()
```

Out[108]:

```
2.581988897471611
```

In [109]:

```
a.var()
```

Out[109]:

```
6.666666666666667
```

In [110]:

```
a.mean()
```

Out[110]:

```
5.0
```

In [111]:

```
np.average(a)
```

Out[111]:

```
5.0
```

In [112]:

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

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

Out[112]:

```
True
```

In [113]:

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

Out[113]:

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

In [114]:

```
a.any()
```

Out[114]:

```
True
```

In [115]:

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

In [116]:

```
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]]
```

Out[116]:

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

In [117]:

```
c = np.vstack((a, b))
```

In [118]:

```
c
```

Out[118]:

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

Matrices

In [119]:

```
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
```

Out[119]:

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

In [120]:

```
b
```

Out[120]:

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

In [121]:

```
c
```

Out[121]:

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

In [122]:

```
b.I
```

Out[122]:

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

In [123]:

```
b.T
```

Out[123]:

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

In [124]:

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

In [125]:

```
a + b
```

Out[125]:

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

In [126]:

```
a * b
```

Out[126]:

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

numpy random module

In [127]:

```
np.random.rand(10)
```

Out[127]:

```
array([0.07479079, 0.72665947, 0.70042678, 0.77335042, 0.1155922 ,
       0.8048485 , 0.93547217, 0.70002197, 0.19946388, 0.38868979])
```

In [128]:

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

Out[128]:

```
array([13, 82, 41, 52, 43, 82, 66, 58, 14, 13])
```

In [129]:

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

Out[129]:

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

In [130]:

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

Out[130]:

```
array([[31, 43, 23, 51, 18],
       [67, 91, 31, 87, 35]])
```

In [131]:

```
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)
```

```
[[1 5 8 9]
 [5 4 1 7]
 [5 5 6 2]]
[[6 1 8]
 [7 3 8]
 [9 3 2]
 [8 2 7]]
[[185  58 127]
 [123  34 123]
 [135  42 106]]
```

In [132]:

```
np.random.sample((3, 4))
```

Out[132]:

```
array([[0.68980796, 0.02076431, 0.93742826, 0.81879762],
       [0.57583921, 0.45666754, 0.18365687, 0.983695  ],
       [0.98617154, 0.38839879, 0.63944443, 0.80156979]])
```

In [133]:

```
np.random.random_sample((3, 4))
```

Out[133]:

```
array([[0.51031749, 0.21900293, 0.41421075, 0.01706868],
       [0.51677009, 0.43820286, 0.38699853, 0.60941428],
       [0.45946998, 0.05787105, 0.74831082, 0.01195137]])
```

In [134]:

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

```
[[0.6244031913275435, 0.6443901998016525, 0.6734318602030748, 0.929253
3340000765], [0.08985269810854213, 0.378120153640894, 0.54022794103574
61, 0.9572109765617419], [0.1095517450379383, 0.09347802495263291, 0.1
66817717516749, 0.25583261658587464]]
```

In [135]:

```
c.shape
```

Out[135]:

```
(3, 4)
```

In [136]:

```
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 [137]:

```
c * [5, 3, 1] # Broadcasting
```

Out[137]:

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

In [138]:

```
[5, 3, 1] * c # Broadcasting
```

Out[138]:

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

In [139]:

```
c
```

Out[139]:

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

In [140]:

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

Out[140]:

```
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 [141]:

```
np.linspace(1, 100, 15)
```

Out[141]:

```
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.        ])
```