- 创建矩阵

```
>>> import numpy as np
>>> a = np.array([1,2,3,4])
>>> b=np.array([[1,2,3,4],[5,6,7,8],[8,9,0,10]])
>>> b
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 8,  9,  0, 10]])
>>> b.dtype          #数据类型
dtype('int32')

>>> a.shape
(4,)
>>> b.shape
(3, 4)
>>> b.shape = 4,3    #强行改变行列大小并不是转置，顺序是不变的
>>> b
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  8],
       [ 9,  0, 10]])

>>> b.shape = 2,-1  #-1表示该轴自动计算
>>> b
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  8,  9,  0, 10]])

>>> a
array([1, 2, 3, 4])
>>> c = a.reshape(2,2)  #reshape之后是共享内存空间，改变一个另一个也会变化
>>> c
array([[1, 2],
       [3, 4]])
>>> a[1] = 100
>>> a
array([  1, 100,   3,   4])
>>> c
array([[  1, 100],
       [  3,   4]])

>>> np.array([1,2,3,4],dtype = np.float)
array([ 1.,  2.,  3.,  4.])
```

```
>>> a = np.arange(0, 1, 0.1) #注意数组不包括终值
>>> a
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])

>>> np.linspace(0, 1, 12) #可以通过endpoint关键字指定是否包括终值，缺省设置是包括终值
array([ 0.        ,  0.09090909,  0.18181818,  0.27272727,  0.36363636,
        0.45454545,  0.54545455,  0.63636364,  0.72727273,  0.81818182,
        0.90909091,  1.        ])

>>> s = "qwertrtyu"
>>> np.fromstring(s,dtype=np.int8)
array([113, 119, 101, 114, 116, 114, 116, 121, 117], dtype=int8)

>>> def func(i,j):
...     return (i+1)*(j+1)
... a = np.fromfunction(func,(9,9))
>>> a
array([[  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.],
       [  2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.,  18.],
       [  3.,   6.,   9.,  12.,  15.,  18.,  21.,  24.,  27.],
       [  4.,   8.,  12.,  16.,  20.,  24.,  28.,  32.,  36.],
       [  5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.],
       [  6.,  12.,  18.,  24.,  30.,  36.,  42.,  48.,  54.],
       [  7.,  14.,  21.,  28.,  35.,  42.,  49.,  56.,  63.],
       [  8.,  16.,  24.,  32.,  40.,  48.,  56.,  64.,  72.],
       [  9.,  18.,  27.,  36.,  45.,  54.,  63.,  72.,  81.]])
```

- 数据存取：和Python的列表序列不同，通过下标范围获取的新的数组是原始数组的一个视图。它与原始数组共享同一块数据空间；当使用整数序列对数组元素进行存取时，将使用整数序列中的每个元素作为下标，整数序列可以是列表或者数组。使用整数序列作为下标获得的数组不和原始数组共享数据空间。：

```
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[5]
5
>>> a[3:5] #包括3但是不包括5
array([3, 4])
>>> a[:5] #表示从0开始
array([0, 1, 2, 3, 4])
>>> a[:-1] #-1表示到最后一个（不包括）
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> a[2:4] = 100,101 #修改元素
>>> a
array([  0,   1, 100, 101,   4,   5,   6,   7,   8,   9])
>>> a[1:-1:2] #步长2
```

```
array([  1, 101,   5,   7])
>>> a[::-1] #步长负数表示倒着来
array([  9,   8,   7,   6,   5,   4, 101, 100,   1,   0])
>>> a[5:1:-1]
array([  5,   4, 101, 100])


>>> x = np.arange(10,1,-1)
>>> x
array([10,  9,  8,  7,  6,  5,  4,  3,  2])
>>> x[[3,3,1,8]]        # 获取x中的下标为3, 3, 1, 8的4个元素，组成一个新的数组
array([7, 7, 9, 2])
>>> b = x[np.array([3,3,-3,8])]
>>> b
array([7, 7, 4, 2])
>>> b[2] = 100
>>> b
array([  7,   7, 100,   2])
>>> x                    # 由于b和x不共享数据空间，因此x中的值并没有改变
array([10,  9,  8,  7,  6,  5,  4,  3,  2])
>>> x[[3,5,1]] = -1,-2,-3
>>> x
array([10, -3,  8, -1,  6, -2,  4,  3,  2])
```

- 使用bool数组访问

```
>>> x = np.arange(5,0,-1)
>>> x
array([5, 4, 3, 2, 1])
>>> x[np.array([True,False,True,False,False])]  #取出对应位置为True的元素
array([5, 3])
>>> x[[True,False,True,False,False]]    #如果是列表会把True当作1 反复取下标为1的元素
array([4, 5, 4, 5, 5])
>>> x[np.array([True, False, True, True])]  #缺省值默认是False
array([5, 3, 2])
>>> x[np.array([True, False, True, True])] = -1, -2, -3
>>> x
array([-1,  4, -2, -3,  1])
```

- bool数组的用法：

```
>>> import numpy as np
>>> x = np.random.rand(10)
>>> x
array([ 0.99417532,  0.62634283,  0.18214893,  0.70119425,  0.91433575,
```

```
       0.78750317,  0.45406477,  0.21615546,  0.10961145,  0.54021769])
>>> x>0.5
array([ True,  True, False,  True,  True,  True, False, False, False,  True],
dtype=bool)
>>> x[x>0.5]
array([ 0.99417532,  0.62634283,  0.70119425,  0.91433575,  0.78750317,
       0.54021769])
```

- 矩阵的存取：类似matlab

```
>>> a = np.arange(0,60,10).reshape(-1,1)+np.arange(0,6)
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
>>> a[0,3:5]
array([3, 4])
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
>>> a[:,2]
array([ 2, 12, 22, 32, 42, 52])
>>> a[2::2,::2]          #行从2开始，步长2，列从0开始，步长2
array([[20, 22, 24],
       [40, 42, 44]])
>>> a[(0,1,2,3,4),(1,2,3,4,5)]      #相当于(0,1)(1,2)(2,3)对应的坐标
array([ 1, 12, 23, 34, 45])
>>> a[3:,[0,2,5]]                   #行从3开始，列为0,2,5
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
>>> a[np.array([1,0,1,0,0,1],dtype=np.bool),2] #用bool数组取对应元素
array([ 2, 22, 52])
```

- 结构数组：类似json的语法

```
>>> persontype = np.dtype({
...     'names':['name','age','weight'],
...     'formats':['S32','i','f']
...      })
... a = np.array([('zhang',32,75.5),('wang',24,65.2)],dtype=persontype)
>>> a
array([(b'zhang', 32, 75.5), (b'wang', 24, 65.19999694824219)],
```

```
      dtype=[('name', 'S32'), ('age', '<i4'), ('weight', '<f4')])
```

- ufunc：ufunc是universal function的缩写，它是一种能对数组的每个元素进行操作的函数

```
>>> x = np.linspace(0,2*np.pi,10)
>>> x
array([ 0.        ,  0.6981317 ,  1.3962634 ,  2.0943951 ,  2.7925268 ,
        3.4906585 ,  4.1887902 ,  4.88692191,  5.58505361,  6.28318531])
>>> y = np.sin(x)
>>> y
array([  0.00000000e+00,   6.42787610e-01,   9.84807753e-01,
         8.66025404e-01,   3.42020143e-01,  -3.42020143e-01,
        -8.66025404e-01,  -9.84807753e-01,  -6.42787610e-01,
        -2.44929360e-16])
>>> a = np.arange(0,4)
>>> a
array([0, 1, 2, 3])
>>> b=np.arange(1,5)
>>> b
array([1, 2, 3, 4])
>>> np.add(a, b)
array([1, 3, 5, 7])
```

- 广播

```
>>> a = np.arange(0, 60, 10).reshape(-1, 1)
>>> a
array([[ 0],
       [10],
       [20],
       [30],
       [40],
       [50]])
>>> b = np.arange(0, 5)
>>> b
array([0, 1, 2, 3, 4])
>>> a + b
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24],
       [30, 31, 32, 33, 34],
       [40, 41, 42, 43, 44],
       [50, 51, 52, 53, 54]])

>>> x, y = np.ogrid[0:5, 0:5]
```

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


>>> np.add.reduce([1,2,3]) # 1 + 2 + 3
6
>>> np.add.reduce([[1,2,3],[4,5,6]], axis=1) # 1,4 + 2,5 + 3,6
array([ 6, 15])
>>> np.add.reduce([[1,2,3],[4,5,6]], axis=0)
array([5, 7, 9])
```

- 矩阵运算：NumPy和Matlab不一样，对于多维数组的运算，缺省情况下并不使用矩阵运算；numpy库提供了matrix类，使用matrix类创建的是矩阵对象，它们的加减乘除运算缺省采用矩阵方式计算，因此用法和matlab十分类似；矩阵的乘积可以使用dot函数进行计算。对于二维数组，它计算的是矩阵乘积，对于一维数组，它计算的是其点积。当需要将一维数组当作列矢量或者行矢量进行矩阵运算时，推荐先使用reshape函数将一维数组转换为二维数组
    1. dot : 对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和(数学上称之为内积)；对于二维数组，计算的是两个数组的矩阵乘积
    2. outer : 只按照一维数组进行计算，如果传入参数是多维数组，则先将此数组展平为一维数组之后再进行运算。
    3. 矩阵中更高级的一些运算可以在NumPy的线性代数子库linalg中找到。例如inv函数计算逆矩阵，solve函数可以求解多元一次方程组。

```
>>> a = array([1, 2, 3])
>>> a.reshape((-1,1))
array([[1],
       [2],
       [3]])
>>> a.reshape((1,-1))
array([[1, 2, 3]])

>>> np.outer([1,2,3],[4,5,6,7])
    array([[ 4,  5,  6,  7],
```

```
           [ 8, 10, 12, 14],
           [12, 15, 18, 21]])


>>> a = np.random.rand(10,10)
>>> b = np.random.rand(10)
>>> x = np.linalg.solve(a,b)
>>> np.sum(np.abs(np.dot(a,x) - b))
3.1433189384699745e-15
```

- 文件存取：

```
>>> a = np.arange(0,12)
>>> a.shape = 3,4
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a.tofile("a.bin")
>>> b = np.fromfile("a.bin", dtype=np.float) # 按照float类型读入数据
>>> b # 读入的数据是错误的
array([  2.12199579e-314,   6.36598737e-314,   1.06099790e-313,
         1.48539705e-313,   1.90979621e-313,   2.33419537e-313])
>>> a.dtype # 查看a的dtype
dtype('int32')
>>> b = np.fromfile("a.bin", dtype=np.int32) # 按照int32类型读入数据
>>> b # 数据是一维的
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> b.shape = 3, 4 # 按照a的shape修改b的shape
>>> b # 这次终于正确了
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])


>>> np.save("a.npy", a)
>>> c = np.load( "a.npy" )
>>> c
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])


>>> a = np.array([[1,2,3],[4,5,6]])
>>> b = np.arange(0, 1.0, 0.1)
```

```
>>> c = np.sin(b)
>>> np.savez("result.npz", a, b, sin_array = c)
>>> r = np.load("result.npz")
>>> r["arr_0"] # 数组a
array([[1, 2, 3],
       [4, 5, 6]])
>>> r["arr_1"] # 数组b
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
>>> r["sin_array"] # 数组c
array([ 0.        ,  0.09983342,  0.19866933,  0.29552021,  0.38941834,
        0.47942554,  0.56464247,  0.64421769,  0.71735609,  0.78332691])


>>> a = np.arange(0,12,0.5).reshape(4,-1)
>>> np.savetxt("a.txt", a) # 缺省按照'%.18e'格式保存数据，以空格分隔
>>> np.loadtxt("a.txt")
array([[  0. ,   0.5,   1. ,   1.5,   2. ,   2.5],
       [  3. ,   3.5,   4. ,   4.5,   5. ,   5.5],
       [  6. ,   6.5,   7. ,   7.5,   8. ,   8.5],
       [  9. ,   9.5,  10. ,  10.5,  11. ,  11.5]])
>>> np.savetxt("a.txt", a, fmt="%d", delimiter=",") #改为保存为整数，以逗号分隔
>>> np.loadtxt("a.txt",delimiter=",") # 读入的时候也需要指定逗号分隔
array([[  0.,   0.,   1.,   1.,   2.,   2.],
       [  3.,   3.,   4.,   4.,   5.,   5.],
       [  6.,   6.,   7.,   7.,   8.,   8.],
       [  9.,   9.,  10.,  10.,  11.,  11.]])


>>> a = np.arange(8)
>>> b = np.add.accumulate(a)
>>> c = a + b
>>> f = file("result.npy", "wb")
>>> np.save(f, a) # 顺序将a,b,c保存进文件对象f
>>> np.save(f, b)
>>> np.save(f, c)
>>> f.close()
>>> f = file("result.npy", "rb")
>>> np.load(f) # 顺序从文件对象f中读取内容
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> np.load(f)
array([ 0,  1,  3,  6, 10, 15, 21, 28])
>>> np.load(f)
array([ 0,  2,  5,  9, 14, 20, 27, 35])
```