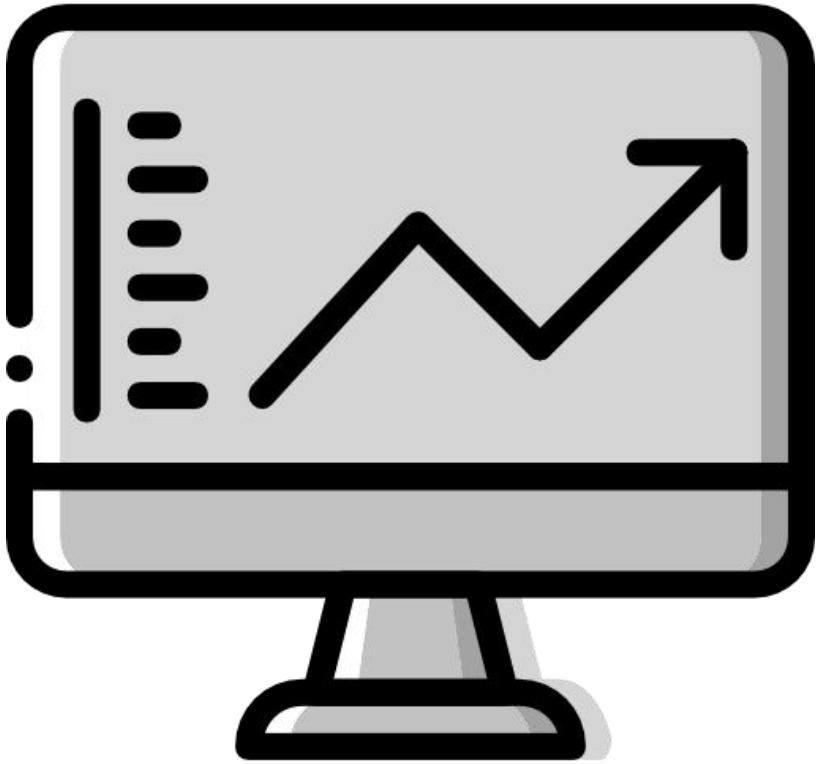
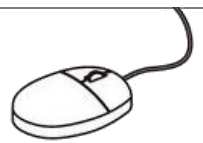
南通师范高等专科学校
Nantong Normal College



Python
数据分析与应用

执教：朱亚林

NumPy简介

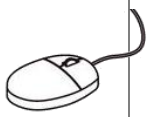


什么是NumPy

- NumPy 是 Python 的一个扩展程序库
- 官方网站:https://www.numpy.org/

- 支持高端大量的维度数组与矩阵运算
- 针对数组运算提供大量的数学函数库

- NumPy 是 SciPy、Matplotlib 等扩展程序库的基础组件
- 原作者:Travis Oliphant（特拉维斯·奥利芬特，美国数据科学家和商人）
- 初始版本　　Numeric, 1995年;NumPy, 2006年
- 最新版本:1.17(2019年4月23日)



NumPy的安装

安装方法

1. 使用已有的发行版本

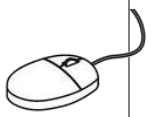
如：Anaconda

2. 使用pip安装

```
python -m pip install --user numpy
```

3. 如果是Linux

```
sudo apt install python3-numpy
```



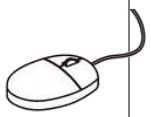
NumPy的安装

测试numpy是否安装成功

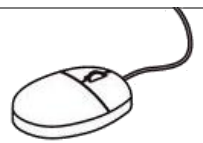
```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
```

```
>>> from numpy import *
>>> eye(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

from numpy import * 为导入 numpy 库。
eye(4) 生成对角矩阵。



NumPy基础知识



基础概念

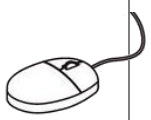
- NumPy的主要对象是**同构多维数组**。
- **同构多维数组**是一个元素表（通常是数字），所有类型都相同，由非负整数元组索引。
- 在NumPy中维度（dimension）称为轴（ axis）。

```
[1, 2, 1]
```

- 以上是一个一维数组（具有一个轴），包含三个元素（element），即长度为 3

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

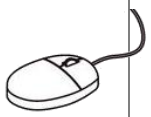
- 以上，数组有2个轴。第一轴的长度为2，第二轴的长度为3。



ndarray的属性

- NumPy的数组类被调用ndarray。它也有个叫做array的别名。
- 请注意，numpy.array与标准Python库类array.array不同，后者只处理一维数组并提供较少的功能。

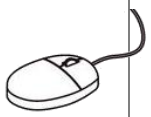
序号	属性名	作用
1	ndarray.ndim	数组的轴（维度）的个数。
2	ndarray.shape	数组每个维度的大小,例如一个 n 行 m 列的矩阵的大小为 (n, m)
3	ndarray.size	数组元素的总数。这等于 shape 的元素的乘积。
4	ndarray.dtype	一个描述数组中元素类型的对象。
5	ndarray.itemsize	数组中每个元素的字节大小。
6	ndarray.data	该缓冲区包含数组的实际元素（直接读取数组元素）。



属性举例

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
```

```
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```



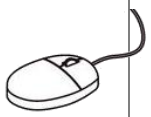
数组的创建

使用array函数从常规Python列表或元组中创建数组

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.shape
1,4
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

常见错误：调用array的时候传入序列，而不是提供单个数字的列表类型作为参数

```
>>> a = np.array(1,2,3,4)    # WRONG
>>> a = np.array([1,2,3,4])  # RIGHT
```



数组的创建

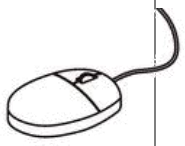
NumPy 提供可迅速生成数列的函数

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5,
 1.8])
```

由于小数精度问题,使用 arange 函数较难预测生成数组的元素数量

如果要指定生成的数组长度,可以使用 linspace 函数

```
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )
array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
>>> x = np.linspace( 0, 2*pi, 100 )
>>> f = np.sin(x)
```



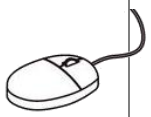
数组的创建

使用empty方法

numpy.empty 方法用来创建一个指定形状（shape）、数据类型（dtype）且未初始化的数组

```
numpy.empty(shape, dtype = float, order = 'C')
```

```
>>> import numpy as np
>>> x = np.empty([3,2], dtype = int)
>>> print (x)
[[ 6917529027641081856  5764616291768666155]
 [ 6917529027641081859 -5764598754299804209]
 [          4497473538      844429428932120]]
```



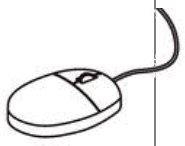
数组的创建

使用zeros方法

创建指定大小的数组，数组元素以 0 来填充

```
numpy.zeros(shape, dtype = float, order = 'C')
```

```
>>> import numpy as np
>>> x = np.zeros(5)
>>> print(x)
[0. 0. 0. 0. 0.]
>>> y = np.zeros((5,), dtype = np.int)
>>> print(y)
[0 0 0 0 0]
```



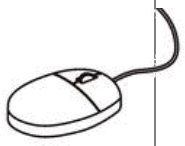
数组的创建

使用ones方法

创建指定形状的数组，数组元素以 1 来填充

```
numpy.ones(shape, dtype = None, order = 'C')
```

```
>>> import numpy as np
>>> x = np.ones(5)
>>> print(x)
[1. 1. 1. 1. 1.]
>>> x = np.ones([2,2], dtype = int)
>>> print(x)
[[1 1]
 [1 1]]
```



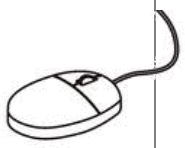
数组的创建

生成随机数组

使用NumPy提供的random函数可以生成随机数组

```
numpy.random.random(n)
Numpy.random.rand(low,high,size,dtype)
```

```
>>> import numpy as np
>>> x = np.random.random(20)
>>> y = np.random.randint(0,5,size=[3,4],dtype='l')
```



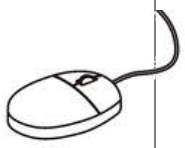
数组的创建

直接从文件中获得数组

使用NumPy提供的loadtxt函数将文件中的数据加载到数组。

```
numpy.loadtxt('file',delimiter=',', comments='#',dtype=float)
```

```
>>> import numpy as np
>>> x = np.loadtxt('louse.csv',delimiter=',',dtype=float)
```



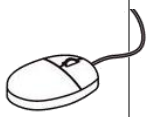
数组的创建

array 还可以将序列的序列转换成二维数组，将序列的序列的序列转换成三维数组

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

可以在创建时显式指定数组的类型

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

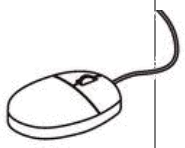


数组的打印

NumPy以与嵌套列表类似的方式显示它，显示顺序如下：

- 最后一个轴从左到右打印（行），
- 倒数第二个从上到下打印（列），
- 其余部分也从上到下打印，每个切片用空行分隔。

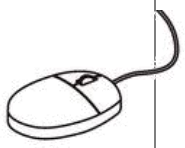
一维数组打印为行，将二维数据打印为矩阵，将三维数据打印为矩数组表



数组的打印

打印示例

```
>>> a = np.arange(6)                         # 1d array
>>> print(a)
[0 1 2 3 4 5]
>>>
>>> b = np.arange(12).reshape(4,3)           # 2d array
>>> print(b)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = np.arange(24).reshape(2,3,4)         # 3d array
>>> print(c)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```



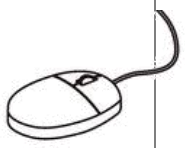
数组的打印

如果数组太大而无法打印，NumPy会自动跳过数组的中心部分并仅打印角点

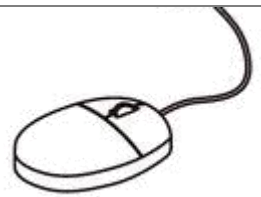
```
>>> print(np.arange(10000))
[   0    1    2 ..., 9997 9998 9999]
>>>
>>> print(np.arange(10000).reshape(100,100))
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

要禁用此行为并强制NumPy打印整个数组，可以使用更改打印选项set_printoptions

```
>>> np.set_printoptions(threshold=sys.maxsize)       # sys module should be imported
```



数组对象的操作

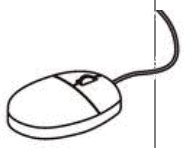


索引与切片

一维数组的索引

一维数组的索引方法与list索引方法一致。即用整数作为下标来获取数组指定位置的元素。

```
>>> a = np.arrange(8)
>>> a[6]
6
>>> a[2:5]
[2 3 4]
>>> a[:5]
[0 1 2 3 4]
>>> a[-1]
7
```

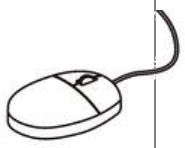


索引与切片

多维数组的索引

多维数组的每一个维度都有一个索引，各个维度的索引之间用逗号隔开。

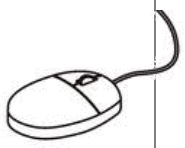
```
>>> arr = np.arange(10*10,dtype=int).reshape(10,10)
>>> arr
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
>>> arr[2:9:2,:]
array([[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89]])
```



布尔索引

可以通过一个布尔数组来索引目标数组，以此找出与布尔数组中值为True的对应的目标数组中的数据。需要注意的是，布尔数组的长度必须与目标数组对应的轴的长度一致。

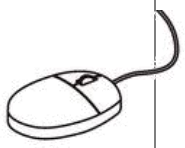
```
>>> arr = np.arange(7)
>>> booling1 = np.array([True,False,False,True,True,False,False])
>>> arr[booling1]
array([0, 3, 4])
# 思考：二维数组怎样索引
# 拓展：arr[arr>5] 的实现
```



花式索引

花式索引是利用元素所在的行、列组成的整数数组进行索引。

```
>>> arr=np.arange(10*10).reshape(10,10)
>>> row_index=[1,1,2,7]
>>> col_index=[0,2,4,8]
>>> arr[row_index,col_index]
```

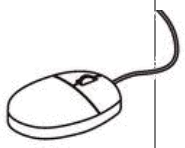


变换数组的形态

使用shape属性、reshape()函数改变数组形状

- 通过修改数组的shape属性，改变数组每个轴的长度。
- 通过使用reshape()函数改变数组的形状。
- 以上属性和函数只修改形状，不改变原始数值。shape修改自身并保存，reshape修改形态，但不保存。

```
>>> a = np.arrange(10)
>>> a.reshape(2,5)
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.shape=2,5
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

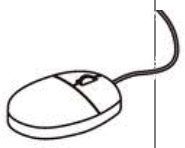


变换数组的形态

使用ravel()、flatten()函数展平数组

- 通过ravel()可以水平展平数组。
- 通过flatten()可以选择从水平或垂直角度展平数组。

```
>>> a = np.arrange(20).reshape(4,5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> a.ravel()
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
>>> a.flatten('F')
array([ 0,  5, 10, 15,  1,  6, 11, 16,  2,  7, 12, 17,  3,  8, 13, 18,  4,
        9, 14, 19])
```



堆叠数组

使用hstack()、column_stack()、conctrate(axis=1)可以完成水平叠放

0	1	2
3	4	5
6	7	8

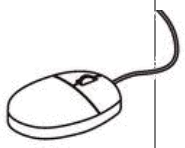
2	4	6
1	3	5
7	8	9

0	1	2	2	4	6
3	4	5	1	3	5
6	7	8	7	8	9

np.hstack((a,b))

np.column_stack((a,b))

np.conctrate((a,b),axis=1)



堆叠数组

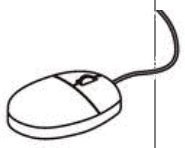
使用vstack()、row_stack()、conctrate(axis=0)可以完成垂直叠放

0	1	2
3	4	5
6	7	8

2	4	6
1	3	5
7	8	9

0	1	2
3	4	5
6	7	8
2	4	6
1	3	5
7	8	9

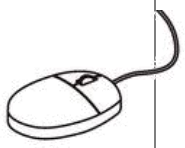
np.vstack((a,b))
np.row_stack((a,b))
np.conctrate((a,b),axis=0)



堆叠数组

使用dstack()完成深度叠放

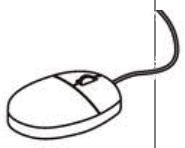
```
>>> a=np.arange(12).reshape(3,4)
>>> b=2*a
>>> np.dstack((a,b))
array([[[ 0,  0],
        [ 1,  2],
        [ 2,  4],
        [ 3,  6]],

       [[ 4,  8],
        [ 5, 10],
        [ 6, 12],
        [ 7, 14]],

       [[ 8, 16],
        [ 9, 18],
        [10, 20],
        [11, 22]]])
```



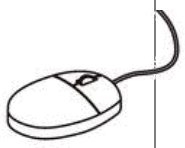
拆分数组

使用hsplit()、vsplit()、split()函数，可以将数组分割成相同大小的子数组。也可以指定原数组中需要分割的位置。

```
>>> a=np.arange(24).reshape(4,6)
>>> b=np.hsplit(a,2)
>>> c=np.vsplit(a,3)
>>> d=np.split(a,2,axis=1)
```



数组的基本运算



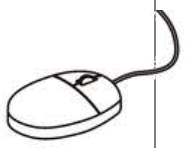
数组上的算术运算符会应用到元素级别

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

数组运算时的广播

一般数组运算时形状都会相同，当操作对象形状不同时，NumPy会尽力进行处理。如果一个数组要与一个标量相乘，那么标量需要根据数组的形状进行扩展，然后才可以执行乘法运算。这个过程称为广播。

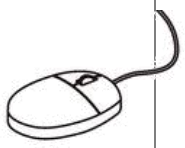
```
>>> a = np.array( [[1,1,1],[2,2,2],[3,3,3]] )
>>> b = np.array( [1,1,1] )
>>> a + b
array([[2, 2, 2],
       [3, 3, 3],
       [4, 4, 4]])
```



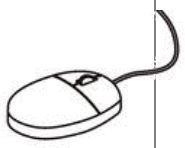
数组的基本运算

- 乘积运算符*在NumPy数组中按元素进行运算。
- 矩阵乘积可以使用@运算符（在python> = 3.5中）或dot函数或方法执行

```
>>> A = np.array( [[1,1],
...               [0,1]] )
>>> B = np.array( [[2,0],
...               [3,4]] )
>>> A * B                          # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B                          # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                       # another matrix product
array([[5, 4],
       [3, 4]])
```



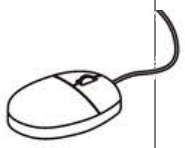
- 小练习
- 小明的本学期的成绩如下：平时成绩80分，期中成绩75分，期末成绩86分。总评分的标准时平时占30%，期中占30%，期末占40%。请使用数组运算的形式求出小明本学期的总评分。



数组的基本运算

某些操作（例如+=和 *=）会更直接更改被操作的矩阵数组而不会创建新矩阵数组。

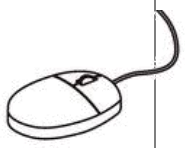
```
>>> a = np.ones((2,3), dtype=int)
>>> b = np.random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.417022  ,  3.72032449,  3.00011437],
       [ 3.30233257,  3.14675589,  3.09233859]])
>>> a += b                  # b is not automatically converted to integer
type
Traceback (most recent call last):
  ...
TypeError: Cannot cast ufunc add output from dtype('float64') to
dtype('int64') with casting rule 'same_kind'
```



数组的基本运算

当使用不同类型的数组进行操作时，结果数组的类型对应于更一般或更精确的数组（称为向上转换的行为）

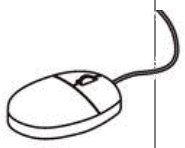
```
>>> a = np.ones(3, dtype=np.int32)
>>> b = np.linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = np.exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
       -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```



数组的基本运算

许多一元操作，例如计算数组中所有元素的总和，都是作为ndarray类的方法实现的。

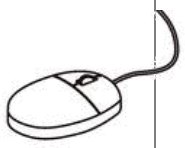
```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.1862602113776709
>>> a.max()
0.6852195003967595
```



数组的基本运算

默认情况下，这些操作适用于数组，就像它是一个数字列表一样，无论其形状如何。但是，通过指定axis 参数，可以沿数组的指定轴应用操作。

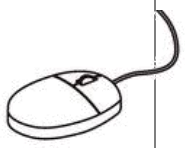
```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                       # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```



数组的通用函数

- NumPy 提供了一些常用的数学函数,例如 sin、cos、exp 等
- 此类数学函数被称为通用函数(universal functions)
- 与数学运算类似,通用函数通常是针对数组元素进行操作的

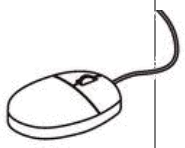
```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> np.sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([ 2.,  0.,  6.])
```



矩阵

在NumPy中，矩阵是ndarray的子类，可以使用数组对象中的mat()、matrix()以及bmat()函数来创建矩阵。

```
>>> a=np.mat(“1 2 3;4 5 6;7 8 9”)
>>> b=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> np.bmat(“a;b”)
```



矩阵属性

属性	说明
T	返回自身的转置
H	返回自身的共轭转置
I	返回自身的逆矩阵
A	返回自身数据的二维数组的一个视图