

# 快速入门教程

## 先决条件

在阅读本教程之前，你应该了解一些Python的基础知识。如果你想复习一下，请回去看看[Python教程](#)。

如果您希望使用本教程中的示例，则还必须在计算机上安装某些软件。有关说明，请参阅<https://scipy.org/install.html>。

## 基础知识

NumPy的主要对象是同构多维数组。它是一个元素表（通常是数字），所有类型都相同，由非负整数元组索引。在NumPy维度中称为 轴。

例如，3D空间中的点的坐标 [1, 2, 1] 具有一个轴。该轴有3个元素，所以我们说它表示的例子中，数组有2个轴。第一轴的长度为2，第二轴的长度为3。

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

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

NumPy的数组类被调用 `ndarray`。它也被别名所知 `array`。请注意，`numpy.array` 这与标准Python库类不同 `array.array`，后者只处理一维数组并提供较少的功能。`ndarray` 对象更重要的属性是：

的矩阵，`shape` 将是 `(n,m)`。因此，`shape` 元组的长度就是rank或维度的个数 `ndim`。

- `ndarray.size` - 数组元素的总数。这等于 `shape` 的元素的乘积。

`dtype` - 一个描述数组中元素类型的对象。可以使用标准的Python类型创建或指定`dtype`。另外NumPy提供它自己的类型。例如`numpy.int32`、`numpy.int16`和`numpy.float64`。

- `ndarray.itemsize` - 数组中每个元素的字节大小。例如，元素为 `float64` 类型的数组的 `itemsize` 为8 ( $=64/8$ )，而 `complex32` 类型的数组的 `itemsize` 为4 ( $=32/8$ )。它等于 `ndarray.dtype.itemsize`。
- `ndarray.data` - 该缓冲区包含数组的实际元素。通常，我们不需要使用此属性，因为我们将使用索引访问数组中的元素。

## 一个例子

```
>>> 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)
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

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

## 数组创建

有几种方法可以创建数组。

例如，你可以使用array函数从常规Python列表或元组中创建数组。得到的数组的类型是从Python列表中元素的类型推导出来的。

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

一个常见的错误，就是调用 array 的时候传入多个数字参数，而不是提供单个数字参数。

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

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

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，等等。

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

显式指定数组的类型：

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

通常，数组的元素最初是未知的，但它的大小是已知的。因此，NumPy提供了几个函数来创建具有初始占位符内容的数组。这就减少了数组增长的必要，因为数组增长的操作花费很大。

函数 `zeros` 创建一个由0组成的数组，函数 `ones` 创建一个完整的数组，函数 `empty` 创建一个数组，其初始内容是随机的，取决于内存的状态。默认情况下，创建的数组的dtype是 `float64` 类型的。

```
>>> np.zeros( (3,4) )                                         py
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                         # dtype can also be specified
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]]], dtype=int16)
>>> np.empty( (2,3) )                                         # uninitialized, output may vary
array([[ 3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [ 5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

```
>>> np.arange( 10, 30, 5 )                                py
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )                      # it accepts float arguments
       3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

当 `arange` 与浮点参数一起使用时，由于有限的浮点精度，通常不可能预测所获得的元素的数量。出于这个原因，通常最好使用 `linspace` 函数来接收我们想要的元素数量的函数，而不是步长（step）：

```
>>> from numpy import pi                               py
>>> np.linspace( 0, 2, 9 )                  # 9 numbers from 0 to 2
array([ 0. ,  0.25,  0.5 ,  0.75,  1. ,  1.25,  1.5 ,  1.75,  2. ])
>>> x = np.linspace( 0, 2*pi, 100 )      # useful to evaluate function at lots of points
>>> f = np.sin(x)
```

#### 另见这些API

`array` ↗ , `zeros` ↗ , `zeros_like` ↗ , `ones` ↗ , `ones_like` ↗ , `empty` ↗ , `empty_like` ↗ , `arange` ↗ , `linspace` ↗ , `numpy.random.mtrand.RandomState.rand` ↗ ,  
`numpy.random.mtrand.RandomState.randn` ↗ , `fromfunction` ↗ , `fromfile` ↗

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## 打印数组

当您打印数组时，NumPy以与嵌套列表类似的方式显示它，但具有以下布局：

- 最后一个轴从左到右打印，
- 倒数第二个从上到下打印，

从口付一维数组丁印ノウ」，付—维数组丁印ノソ入印干，付二维数组丁印ノソ入印女丝日衣。

```
>>> a = np.arange(6)                                # 1d array          py
[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]]]
```

有关 `reshape` 的详情，请参阅下文。

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]]
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
[9800 9801 9802 ... , 9897 9898 9899]  
[9900 9901 9902 ... , 9997 9998 9999]]
```

限制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      py
```

## 基本操作

数组上的算术运算符会应用到 元素 级别。下面是创建一个新数组并填充结果的示例：

```
>>> a = np.array( [20,30,40,50] )      py  
>>> 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])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

与许多矩阵语言不同，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：

```
>>> B = np.array( [[2,0],  
...                 [3,4]] )  
>>> A * B  
# elementwise product
```

```
>>> A @ B  
# matrix product  
array([[5, 4],  
       [3, 4]])  
>>> A.dot(B)  
# another matrix product  
array([[5, 4],  
       [3, 4]])
```

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 c
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

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）。

```
>>> b.dtype.name  
'float64'  
>>> c = a+b  
  
          ,  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'
```

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))                                     py  
>>> 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
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

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，建议使用  
`axis` 参数，您可以沿数组的指定轴应用操作：

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

## 通函数

NumPy提供熟悉的数学函数，例如sin，cos和exp。在NumPy中，这些被称为“通函数”（ufunc）。在NumPy中，这些函数在数组上按元素进行运算，产生一个数组作为输出。

```
>>> 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.])
```

py

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

```
..., bincount ..., ceil ..., clip ..., conj ..., corrcoef ..., cov ..., cross ...,
cumprod ..., cumsum ..., diff ..., dot ..., floor ..., inner ..., INV, lexsort ...,
maximum ..., mean ..., median ..., min ..., minimum ..., nonzero ...,
outer ..., prod ..., re ..., round ..., sort ..., std ..., sum ..., trace ...,
transpose ..., var ..., vdot ..., vectorize ..., where ...
```

## 索引、切片和迭代

一维的数组可以进行索引、切片和迭代操作的，就像 [列表](#) 和其他 Python 序列类型一样。

```
>>> a = np.arange(10)**3  
py  
>>> a  
array([ 0,  1,  8, 27, 64, 125, 216, 343, 512, 729])  
>>> a[2]  
8  
>>> a[2:5]  
array([ 8, 27, 64])  
>>> a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; from start to position 6, exclusive, set every 2nd el  
>>> a  
array([-1000,      1, -1000,      27, -1000,     125,     216,     343,     512,     729])  
>>> a[ : :-1]           # reversed a  
array([ 729,   512,   343,   216,   125, -1000,    27, -1000,      1, -1000])  
>>> for i in a:  
...     print(i**(1/3.))  
...  
nan  
1.0  
nan  
3.0
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
7.0  
8.0  
9.0
```

多维的数组每个轴可以有一个索引。这些索引以逗号分隔的元组给出：

```
>>> def f(x,y):  
...     return 10*x+y  
...  
>>> b = np.fromfunction(f,(5,4),dtype=int)  
>>> b  
array([[ 0,  1,  2,  3],  
       [10, 11, 12, 13],  
       [20, 21, 22, 23],  
       [30, 31, 32, 33],  
       [40, 41, 42, 43]])  
>>> b[2,3]  
23  
>>> b[0:5, 1]                      # each row in the second column of b  
array([ 1, 11, 21, 31, 41])  
>>> b[ : ,1]                      # equivalent to the previous example  
array([ 1, 11, 21, 31, 41])  
>>> b[1:3, : ]                    # each column in the second and third row of b  
array([[10, 11, 12, 13],  
       [20, 21, 22, 23]])
```

当提供的索引少于轴的数量时，缺失的索引被认为是完整的切片：

```
>>> b[-1]                          # the last row. Equivalent to b[-1,:]
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

py

三个点 ( ... ) 表示产生完整索引元组所需的冒号。例如，如果 `x` 是rank为5的数组 (即，它具有5个轴)：

- `x[1,2,...]` 相当于 `x[1,2,:,:,:]`，
- `x[...,3]` 等效于 `x[:, :, :, :, 3]`
- `x[4,...,5,:]` 等效于 `x[4, :, :, 5, :]`。

```
>>> c = np.array( [[[ 0,  1,  2],           # a 3D array (two stacked 2D arrays)
...                   [ 10, 12, 13]],
...                   [[100,101,102],
...                   [110,112,113]]])
>>> c.shape
(2, 2, 3)
>>> c[1,...]                         # same as c[1,:,:] or c[1]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[...,2]                          # same as c[:, :, 2]
array([[ 2, 13],
       [102, 113]])
```

对多维数组进行 **迭代 (Iterating)** 是相对于第一个轴完成的：

```
>>> for row in b:
...     print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

¶ :

```
    in b.flat:  
        ement)  
...  
0  
1  
2  
3  
10  
11  
12  
13  
20  
21  
22  
23  
30  
31  
32  
33  
40  
41  
42  
43
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

## 另见

[Indexing, Indexing](#) (reference), [newaxis](#), [ndenumerate](#), [indices](#)

## 形状操纵

### 改变数组的形状

一个数组的形状是由每个轴的元素数量决定的：

```
>>> a = np.floor(10*np.random.random((3,4)))  
py  
>>> a  
array([[ 2.,  8.,  0.,  6.],  
       [ 4.,  5.,  1.,  1.],  
       [ 8.,  9.,  3.,  6.]])  
>>> a.shape  
(3, 4)
```

可以使用各种命令更改数组的形状。请注意，以下三个命令都返回一个修改后的数组，但不会更改原始数组：

```
>>> a.ravel() # returns the array, flattened  
py  
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])  
>>> a.reshape(6,2) # returns the array with a modified shape  
array([[ 2.,  8.],  
       [ 0.,  6.],  
       [ 4.,  5.],  
       [ 1.,  1.],  
       [ 8.,  9.],  
       [ 3.,  6.]])  
>>> a.T # returns the array, transposed  
array([[ 2.,  4.,  8.],  
       [ 8.,  5.,  9.]])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
(4, 3)
>>> a.shape
(3, 4)
```

由 `ravel()` 产生的数组中元素的顺序通常是“C风格”，也就是说，最右边的索引“变化最快”，因此[0,0]之后的元素是[0,1]。如果将数组重新整形为其他形状，则该数组将被视为“C风格”。NumPy通常创建按此顺序存储的数组，因此 `ravel()` 通常不需要复制其参数，但如果数组是通过获取另一个数组的切片或使用不常见的选项创建的，则可能需要复制它。还可以使用可选参数指示函数 `ravel()` 和 `reshape()`，以使用 FORTRAN样式的数组，其中最左边的索引变化最快。

该 `reshape` 函数返回带有修改形状的参数，而该 `ndarray.resize` 方法会修改数组本身：

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

如果在 `reshape` 操作中将 `size` 指定为-1，则会自动计算其他的 `size` 大小：

```
>>> a.reshape(3,-1)
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

ndarray.shape ↗ , reshape ↗ , resize ↗ , ravel ↗

## 将不同数组堆叠在一起

几个数组可以沿不同的轴堆叠在一起，例如：

```
>>> a = np.floor(10*np.random.random((2,2)))  
py  
>>> a  
array([[ 8.,  8.],  
       [ 0.,  0.]])  
>>> b = np.floor(10*np.random.random((2,2)))  
>>> b  
array([[ 1.,  8.],  
       [ 0.,  4.]])  
>>> np.vstack((a,b))  
array([[ 8.,  8.],  
       [ 0.,  0.],  
       [ 1.,  8.],  
       [ 0.,  4.]])  
>>> np.hstack((a,b))  
array([[ 8.,  8.,  1.,  8.],  
       [ 0.,  0.,  0.,  4.]])
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

该函数将 column\_stack ↗ 1D数组作为列堆叠到2D数组中。它仅相当于 hstack ↗。

```
>>> from numpy import newaxis  
py  
>>> np.column_stack((a,b))      # with 2D arrays  
array([[ 8.,  8.,  1.,  8.],  
       [ 0.,  0.,  0.,  4.]])
```

```
array([[ 4.,  3.],
       [ 2.,  8.]])
>>> np.hstack((a,b))          # the result is different
                                3., 8.])
...[...],...,...])
array([[ 4.],
       [ 2.]])
>>> np.column_stack((a[:,newaxis],b[:,newaxis]))
array([[ 4.,  3.],
       [ 2.,  8.]])
>>> np.hstack((a[:,newaxis],b[:,newaxis]))  # the result is the same
array([[ 4.,  3.],
       [ 2.,  8.]])
```

另一方面，该函数 `ma.row_stack` 等效 `vstack` 于任何输入数组。通常，对于具有两个以上维度的数组，`hstack` 沿其第二轴 `vstack` 堆叠，沿其第一轴堆叠，并 `concatenate` 允许可选参数给出连接应发生的轴的编号。

#### 注意

在复杂的情况下，`r_` 和 `c_` 于通过沿一个轴堆叠数字来创建数组很有用。它们允许使用范围操作符（“`:`”）。

```
>>> np.r_[1:4,0,4]
array([1, 2, 3, 0, 4])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

与数组一起用作参数时，`r_` 和 `c_` 在默认行为上类似于 `vstack` 和 `hstack`，但允许使用可选参数给出要连接的轴的编号。

hstack ↗ , vstack ↗ , column\_stack ↗ , concatenate ↗ , c\_ ↗ , r\_ ↗

## 将一个数组拆分成几个较小的数组

使用 hsplit ↗ , 可以沿数组的水平轴拆分数组 , 方法是指定要返回的形状相等的数组的数量 , 或者指定应该在其之后进行分割的列 :

```
>>> a = np.floor(10*np.random.random((2,12)))          py
>>> a
array([[ 9.,  5.,  6.,  3.,  6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 1.,  4.,  9.,  2.,  2.,  1.,  0.,  6.,  2.,  2.,  4.,  0.]])
>>> np.hsplit(a,3)    # Split a into 3
[array([[ 9.,  5.,  6.,  3.],
       [ 1.,  4.,  9.,  2.]]), array([[ 6.,  8.,  0.,  7.],
       [ 2.,  1.,  0.,  6.]]), array([[ 9.,  7.,  2.,  7.],
       [ 2.,  2.,  4.,  0.]])]
>>> np.hsplit(a,(3,4))    # Split a after the third and the fourth column
[array([[ 9.,  5.,  6.],
       [ 1.,  4.,  9.]]), array([[ 3.],
       [ 2.]]), array([[ 6.,  8.,  0.,  7.,  9.,  7.,  2.,  7.],
       [ 2.,  1.,  0.,  6.,  2.,  4.,  0.]])]
```

vsplit ↗ 沿垂直轴分割 , 并 array\_split ↗ 允许指定要分割的轴。

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## 拷贝和视图

## 完全不复制

简单分配不会复制数组对象或其数据。

```
>>> a = np.arange(12)                                     py
>>> b = a          # no new object is created
>>> b is a        # a and b are two names for the same ndarray object
True
>>> b.shape = 3,4    # changes the shape of a
>>> a.shape
(3, 4)
```

Python将可变对象作为引用传递，因此函数调用不会复制。

```
>>> def f(x):
...     print(id(x))
...
>>> id(a)          # id is a unique identifier of an object
148293216
>>> f(a)
148293216
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## 视图或浅拷贝

不同的数组对象可以共享相同的数据。该 `view` 方法创建一个查看相同数据的新数组对象。

```
False
>>> c.base is a                                # c is a view of the data owned by a
True
ata
>>>
>>> c.shape = 2,6                               # a's shape doesn't change
>>> a.shape
(3, 4)
>>> c[0,4] = 1234                             # a's data changes
>>> a
array([[ 0,  1,  2,  3],
       [1234,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

切片数组会返回一个视图：

```
>>> s = a[ : , 1:3]      # spaces added for clarity; could also be written "s = a[:,1:3]"      py
>>> s[:] = 10            # s[:] is a view of s. Note the difference between s=10 and s[:]=10
>>> a
array([[ 0, 10, 10,  3],
       [1234, 10, 10,  7],
       [ 8, 10, 10, 11]])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## 深拷贝

该 `copy` 方法生成数组及其数据的完整副本。

```
>>> d = a.copy()                                # a new array object with new data is created      py
>>> d is a
```

```
>>> d[0,0] = 9999
>>> a
array([[ 0, 10, 10,  3],
       [10, 10,  7,  0],
       [ 0, 10, 11]])
```

有时，如果不再需要原始数组，则应在切片后调用 `copy`。例如，假设`a`是一个巨大的中间结果，最终结果`b`只包含`a`的一小部分，那么在用切片构造`b`时应该做一个深拷贝：

```
>>> a = np.arange(int(1e8))
>>> b = a[:100].copy()
>>> del a # the memory of ``a`` can be released.
```

如果改为使用 `b = a[:100]`，则 `a` 由 `b` 引用，并且即使执行 `del a` 也会在内存中持久存在。

## 功能和方法概述

以下是按类别排序的一些有用的NumPy函数和方法名称的列表。有关完整列表，请参阅参考手册里的常用API。

- **数组的创建 ( Array Creation )** - `arange`, `array`, `copy`, `empty`, `empty_like`, `fromfunction`, `identity`, `linspace`, `logspace`, `mgrid`, `ogrid`, `ones`, `zeros`, `zeros_like`
- **转换和变换 ( Conversions )** - `ndarray.astype`, `atleast_1d`, `atleast_2d`, `atleast_3d`, `mat`
- **操纵术 ( Manipulations )** - `array_split`, `column_stack`, `concatenate`, `diagonal`, `dsplit`, `dstack`, `hsplit`, `hstack`, `ndarray.item`, `newaxis`, `ravel`, `repeat`, `reshape`, `resize`, `squeeze`

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

- **顺序 ( Ordering )** - `argmax`, `argmin`, `argsort`, `max`, `min`, `ptp`, `searchsorted`, `sort`
- **操作 ( Operations )** - `choose`, `compress`, `cumprod`, `cumsum`, `inner`, `ndarray.fill`, `imag`,  
`putmask`, `real`, `sum`
- **基本统计 ( Basic Statistics )** - `cov`, `mean`, `std`, `var`
- **基本线性代数 ( Basic Linear Algebra )** - `cross`, `dot`, `outer`, `linalg.svd`, `vdot`

## Less 基础

### 广播 ( Broadcasting ) 规则

广播允许通用功能以有意义的方式处理不具有完全相同形状的输入。

广播的第一个规则是，如果所有输入数组不具有相同数量的维度，则将“1”重复地预先添加到较小数组的形状，直到所有数组具有相同数量的维度。

广播的第二个规则确保沿特定维度的大小为1的数组表现为具有沿该维度具有最大形状的数组的大小。假定数组元素的值沿着“广播”数组的那个维度是相同的。

应用广播规则后，所有数组的大小必须匹配。更多细节可以在广播中找到。

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

### 花式索引和索引技巧

NumPy提供比常规Python序列更多的索引功能。除了通过整数和切片进行索引之外，正如我们之前看到的，数组可以由整数数组和布尔数组索引。

## 使用索引数组进行索引

```
e(12)**2                                # the first 12 square numbers      py
>>> i = np.array( [ 1,1,3,8,5 ] )
>>> a[i]                                 # an array of indices
array([ 1,   1,   9,  64, 25])
>>>
>>> j = np.array( [ [ 3, 4], [ 9, 7 ] ] )    # a bidimensional array of indices
>>> a[j]                                 # the same shape as j
array([[ 9, 16],
       [81, 49]])
```

当索引数组 `a` 是多维的时，单个索引数组指的是第一个维度 `a`。以下示例通过使用调色板将标签图像转换为彩色图像来显示此行为。

```
>>> palette = np.array( [ [0,0,0],                      # black
...                         [255,0,0],                     # red
...                         [0,255,0],                     # green
...                         [0,0,255],                    # blue
...                         [255,255,255] ] )            # white
>>> image = np.array( [ [ 0, 1, 2, 0 ],                  # each value corresponds to a color in
...                         [ 0, 3, 4, 0 ] ] )
>>> palette[image]                                     # the (2,4,3) color image
array([[[ 0,  0,  0],
       [255,  0,  0],
       [ 0, 255,  0],
       [ 0,  0, 255]],
      [[ 0,  0,  0],
       [ 0,  0, 255],
       [255, 255, 255],
       [ 0,  0,  0]]])
```

**这是一个最朴素的恰饭广告**

[点此购买优选课程赞助中文网](#)

```
>>> a = np.arange(12).reshape(3,4)
>>> a
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],
 [ 8,  9, 10, 11]]
>>> i = np.array( [ [0,1],           # indices for the first dim of a
...                  [1,2] ] )
>>> j = np.array( [ [2,1],           # indices for the second dim
...                  [3,3] ] )
>>>
>>> a[i,j]
array([[ 2,  5],
       [ 7, 11]])
>>>
>>> a[i,2]
array([[ 2,  6],
       [ 6, 10]])
>>>
>>> a[:,j]                         # i.e., a[ : , j ]
array([[[ 2,  1],
        [ 3,  3]],
       [[ 6,  5],
        [ 7,  7]],
       [[10,  9],
        [11, 11]]])
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

但是，我们不能通过放入 `i` 和 `j` 放入数组来实现这一点，因为这个数组将被解释为索引a的第一个维度

```
>>> s = np.array( [i,j] )                                     py
>>> a[s]                                                 # not what we want
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: index (3) out of range (0<=index<=2) in dimension 0
>>>
>>> a[tuple(s)]                                         # same as a[i,j]
array([[ 2,  5],
       [ 7, 11]])
```

使用数组索引的另一个常见用法是搜索与时间相关的系列的最大值：

```
>>> time = np.linspace(20, 145, 5)                         # time scale
>>> data = np.sin(np.arange(20)).reshape(5,4)      # 4 time-dependent series
>>> time
array([ 20. ,  51.25,  82.5 , 113.75, 145. ])
>>> data
array([[ 0.          ,  0.84147098,  0.90929743,  0.14112001],
       [-0.7568025 , -0.95892427, -0.2794155 ,  0.6569866 ],
       [ 0.98935825,  0.41211849, -0.54402111, -0.99999021],
       [-0.53657292,  0.42016704,  0.99060736,  0.65028784],
       [-0.28790332, -0.96139749, -0.75098725,  0.14987721]])
>>>
>>> ind = data.argmax(axis=0)                                # index of the maxima for each series
>>> ind
array([2, 0, 3, 1])
>>>
>>> time_max = time[ind]                                    # times corresponding to the maxima
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
>>> time_max  
array([ 82.5 ,  20. , 113.75,  51.25])  
>>> data_max  
25,  0.84147098,  0.99060736,  0.6569866 ])  
...  
>>> np.all(data_max == data.max(axis=0))  
True
```

您还可以使用数组索引作为分配给的目标：

```
>>> a = np.arange(5)                                         py  
>>> a  
array([0, 1, 2, 3, 4])  
>>> a[[1,3,4]] = 0  
>>> a  
array([0, 0, 2, 0, 0])
```

但是，当索引列表包含重复时，分配会多次完成，留下最后一个值：

```
>>> a = np.arange(5)                                         py  
>>> a[[0,0,2]]=[1,2,3]  
>>> a  
array([2, 1, 3, 3, 4])
```

这是合理的，但请注意是否要使用Python的 `+=` 构造，因为它可能不会按预期执行

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
>>> a = np.arange(5)                                         py  
>>> a[[0,0,2]]+=1  
>>> a  
array([1, 1, 3, 3, 4])
```

1"。

## 目进行索引

当我们使用（整数）索引数组索引数组时，我们提供了要选择的索引列表。使用布尔索引，方法是不同的；我们明确地选择我们想要的数组中的哪些项目以及我们不需要的项目。

人们可以想到的最自然的布尔索引方法是使用与原始数组具有相同形状的布尔数组：

```
>>> a = np.arange(12).reshape(3,4)                                     py
>>> b = a > 4
>>> b                                         # b is a boolean with a's shape
array([[False, False, False, False],
       [False, True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]                                         # 1d array with the selected elements
array([ 5,  6,  7,  8,  9, 10, 11])
```

此属性在分配中非常有用：

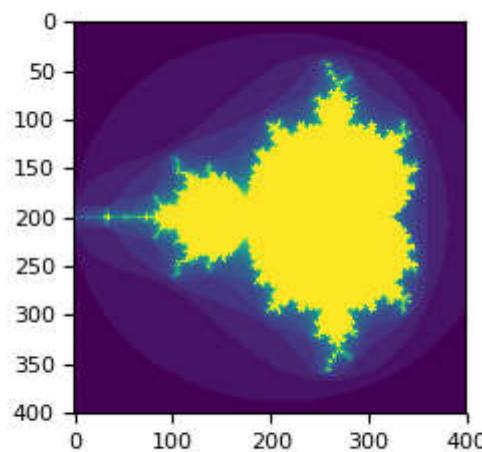
```
>>> a[b] = 0                                         # All elements of 'a' higher than 4 become 0
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  0,  0,  0],
       [ 0,  0,  0,  0]])
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

您可以查看以下示例，了解如何使用布尔索引生成Mandelbrot集的图像：

```
>>> def mandelbrot( h,w, maxit=20 ):
...     """Returns an image of the Mandelbrot fractal of size (h,w)."""
...     y,x = np.ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
...     1j
...
...     divtime = maxit + np.zeros(z.shape, dtype=int)
...
...     for i in range(maxit):
...         z = z**2 + c
...         diverge = z*np.conj(z) > 2**2           # who is diverging
...         div_now = diverge & (divtime==maxit)    # who is diverging now
...         divtime[div_now] = i                     # note when
...         z[diverge] = 2                          # avoid diverging too much
...
...     return divtime
>>> plt.imshow(mandelbrot(400,400))
>>> plt.show()
```



这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
>>> a = np.arange(12).reshape(3,4)                                py
...>>> b1 = np.array([False,True,True])          # first dim selection
...>>> b2 = np.array([True,False,True,False])    # second dim selection
>>>
>>> a[b1,:]
...          # selecting rows
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> a[b1]                                         # same thing
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> a[:,b2]                                         # selecting columns
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
>>>
>>> a[b1,b2]                                     # a weird thing to do
array([ 4, 10])
```

请注意，1D布尔数组的长度必须与要切片的尺寸（或轴）的长度一致。在前面的例子中，`b1` 具有长度为3（的数目的行中 `a`），和 `b2`（长度4）适合于索引的第二轴线（列）`a`。

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## ix\_()函数

`ix_` 函数可用于组合不同的向量，以便获得每个n-uplet的结果。例如，如果要计算从每个向量`a`，`b`和`c`中取得的所有三元组的所有 $a + b * c$ ：

```
>>> c = np.array([5,4,6,8,3])
>>> ax,bx,cx = np.ix_(a,b,c)
>>> ax
[[4],
 [[5]])
>>> bx
array([[[8],
 [5],
 [4]]])
>>> cx
array([[[5, 4, 6, 8, 3]]])
>>> ax.shape, bx.shape, cx.shape
((4, 1, 1), (1, 3, 1), (1, 1, 5))
>>> result = ax+bx*cx
>>> result
array([[[42, 34, 50, 66, 26],
 [27, 22, 32, 42, 17],
 [22, 18, 26, 34, 14]],
 [[43, 35, 51, 67, 27],
 [28, 23, 33, 43, 18],
 [23, 19, 27, 35, 15]],
 [[44, 36, 52, 68, 28],
 [29, 24, 34, 44, 19],
 [24, 20, 28, 36, 16]],
 [[45, 37, 53, 69, 29],
 [30, 25, 35, 45, 20],
 [25, 21, 29, 37, 17]]])
>>> result[3,2,4]
17
>>> a[3]+b[2]*c[4]
17
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
>>> def ufunc_reduce(ufct, *vectors):
...     vs = np.ix_(*vectors)
...     r = ufct.identity
...     for v in vs:
...         r = ufct(r, v)
...     return r
```

然后将其用作：

```
>>> ufunc_reduce(np.add,a,b,c)
array([[15, 14, 16, 18, 13],
       [12, 11, 13, 15, 10],
       [11, 10, 12, 14,  9]],
      [[16, 15, 17, 19, 14],
       [13, 12, 14, 16, 11],
       [12, 11, 13, 15, 10]],
      [[17, 16, 18, 20, 15],
       [14, 13, 15, 17, 12],
       [13, 12, 14, 16, 11]],
      [[18, 17, 19, 21, 16],
       [15, 14, 16, 18, 13],
       [14, 13, 15, 17, 12]]])
```

与普通的ufunc.reduce相比，这个版本的reduce的优点是它利用了广播规则，以避免输出的大小乘以向量的数量。

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

## 使用字符串建立索引

请参见结构化数组。

## 线性代数

这里包括基本线性代数。

### 简单数组操作

有关更多信息，请参阅numpy文件夹中的linalg.py.

```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]

>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]))

>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j      # matrix product
array([[-1.,  0.],
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
2.0

>>> y = np.array([[5.], [7.]])
      lve(a, y)
      [ 4.])

>>> np.linalg.eig(j)
(array([ 0.+1.j,  0.-1.j]), array([[ 0.70710678+0.j         ,  0.70710678-0.j         ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
```

Parameters:

square matrix

Returns

The eigenvalues, each repeated according to its multiplicity.

The normalized (unit "length") eigenvectors, such that the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`.

py

## 技巧和提示

这里我们列出一些简短有用的提示。

### “自动”整形

要更改数组的尺寸，您可以省略其中一个尺寸，然后自动推导出尺寸：

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)

```
>>> a.shape  
(2, 5, 3)  
>>> 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],  
 [24, 25, 26],  
 [27, 28, 29]]])
```

## 矢量堆叠

我们如何从同等大小的行向量列表中构造一个二维数组？在MATLAB这是很简单：如果 `x` 和 `y` 你只需要做两个相同长度的向量 `m=[x;y]`。在此NumPy的通过功能的工作原理 `column_stack`，`dstack`，`hstack` 和 `vstack`，视维在堆叠是必须要做的。例如：

```
x = np.arange(0,10,2)          # x=([0,2,4,6,8])  
y = np.arange(5)                # y=([0,1,2,3,4])  
m = np.vstack([x,y])           # m=([[0,2,4,6,8],  
#                  [0,1,2,3,4]])  
xy = np.hstack([x,y])           # xy =([0,2,4,6,8,0,1,2,3,4])
```

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

这些函数背后的逻辑在两个以上的维度上可能很奇怪。

另见

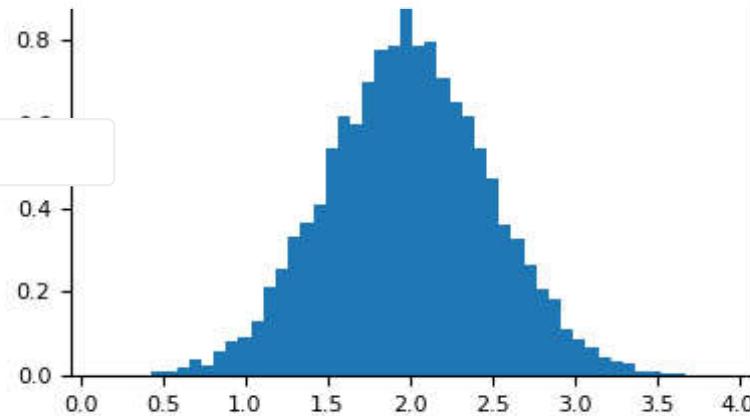
## 直方图

于数组的NumPy 函数返回一对向量：数组的直方图和bin的向量。注意：`matplotlib` 还有一个构建直方图的功能（`hist` 在Matlab中称为），与NumPy中的直方图不同。主要区别在于 `pylab.hist` 自动绘制直方图，而 `numpy.histogram` 只生成数据。

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # Build a vector of 10000 normal deviates with variance 0.5^2 and mean 2
>>> mu, sigma = 2, 0.5
>>> v = np.random.normal(mu,sigma,10000)
>>> # Plot a normalized histogram with 50 bins
>>> plt.hist(v, bins=50, density=1)      # matplotlib version (plot)
>>> plt.show()
```

这是一个最朴素的恰饭广告

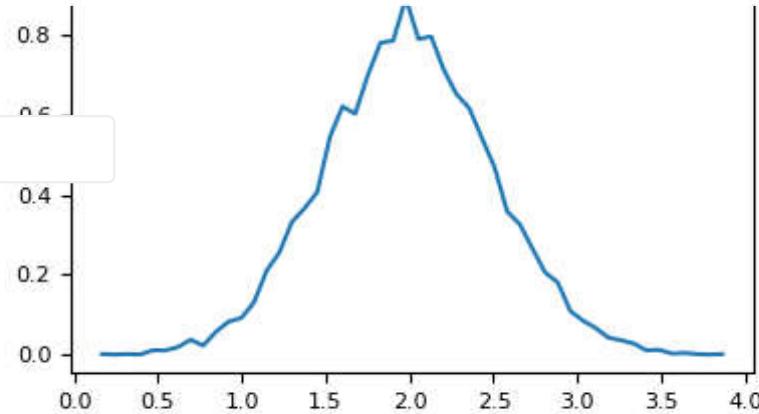
[点此购买优选课程赞助中文网](#)



```
>>> # Compute the histogram with numpy and then plot it
>>> (n, bins) = np.histogram(v, bins=50, density=True) # NumPy version (no plot)
>>> plt.plot(.5*(bins[1:]+bins[:-1]), n)
>>> plt.show()
```

这是一个最朴素的恰饭广告

[点此购买优选课程赞助中文网](#)



## 进一步阅读

- Python的教程 ↗
- NumPy参考
- SciPy教程 ↗
- SciPy讲义 ↗
- MATLAB , R , IDL , NumPy/SciPy 宝典 ↗

在 GitHub 上编辑此页 ↗

上次更新: 2020/4/5下午5:29:33

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

评论系统已退役，点击进入[NumPy中文社区](#)，体验功能更强大的交流社区。

Code -1: Request has been terminated

Possible causes: the network is offline, Origin is not allowed by Access-Control-Allow-Origin, the page is being unloaded,

这是一个最朴素的恰饭广告

点此购买优选课程赞助中文网

line

3.10