Numpy基础-聚类算法 C06

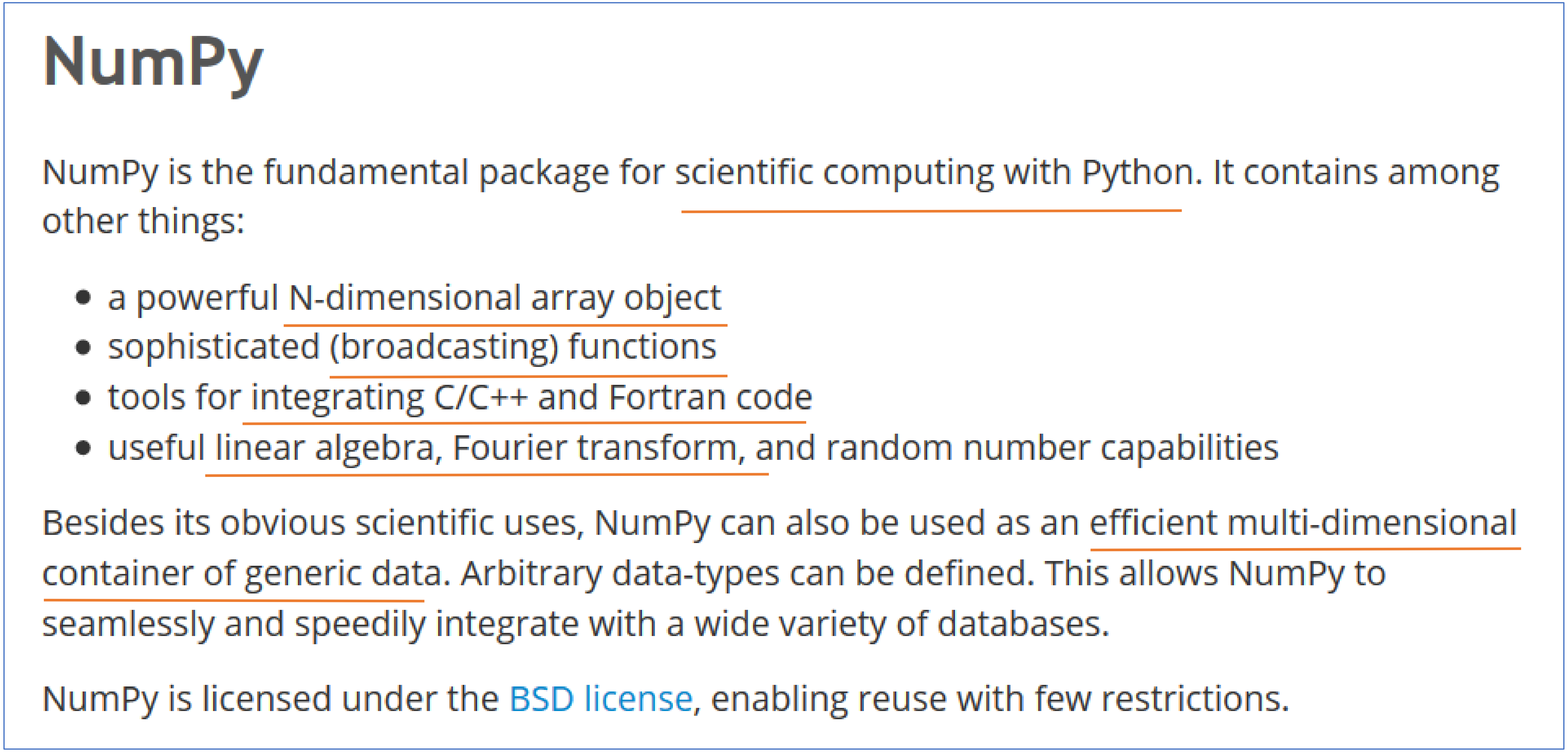
信息科学技术学院

胡俊峰

本次课内容

- Numpy及矩阵运算基础:
- 向量计算
- 线性代数运算
- K-means聚类算法

Numpy基础



NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

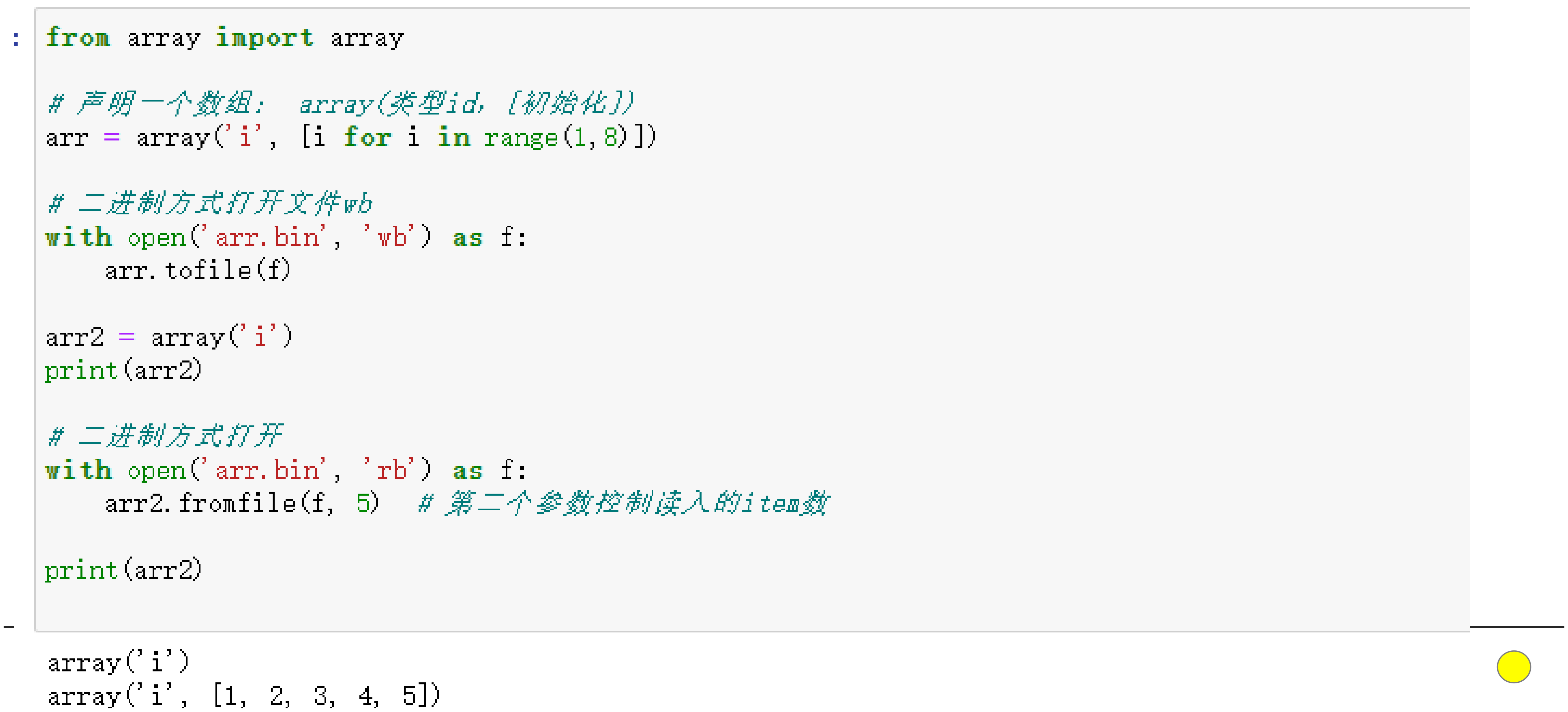
Numpy中的ndarray类型

- Python中的array
- ndarray的类型、访问、切片与shape转换
- ndarray内置计算

Python中的array

array： class array.array(typecode[initializer ,])

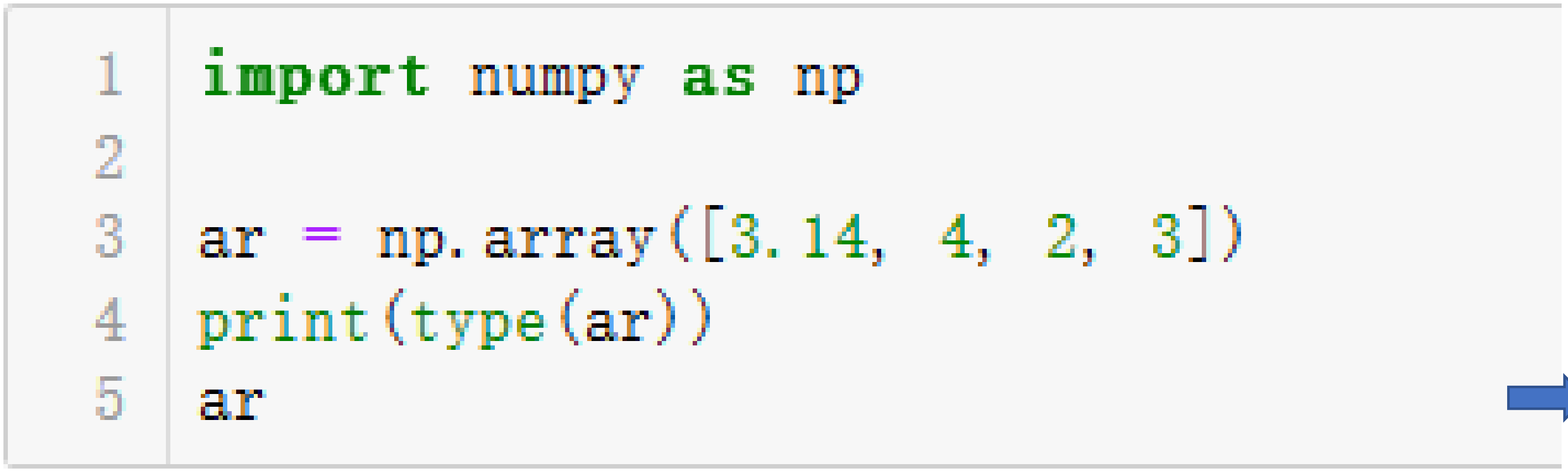
提供更高效率的数值计算数组.可用于C++的数组兼容接口。 提供类似list的访问和增删改操作



```
from array import array

# 声明一个数组： array(类型id, [初始化])
arr = array('i', [i for i in range(1,8)])

# 二进制方式打开文件wb
with open('arr.bin', 'wb') as f:
    arr.tofile(f)

arr2 = array('i')
print(arr2)

# 二进制方式打开
with open('arr.bin', 'rb') as f:
    arr2.fromfile(f, 5)  # 第二个参数控制读入的item数

print(arr2)
```

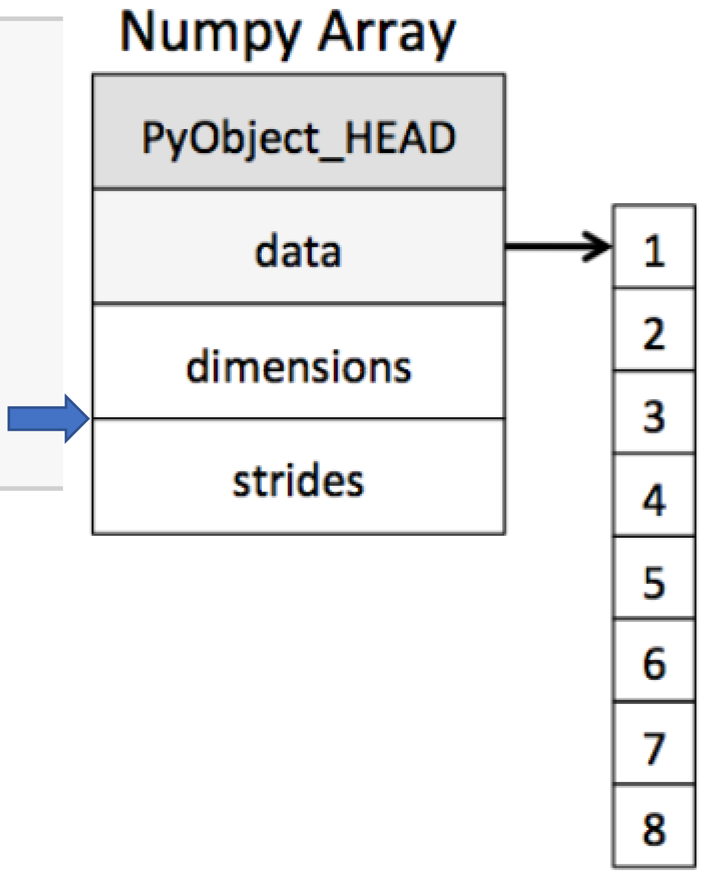
```
array('i')
array('i', [1, 2, 3, 4, 5])
```

高效的固定类型的矩阵数据处理方案

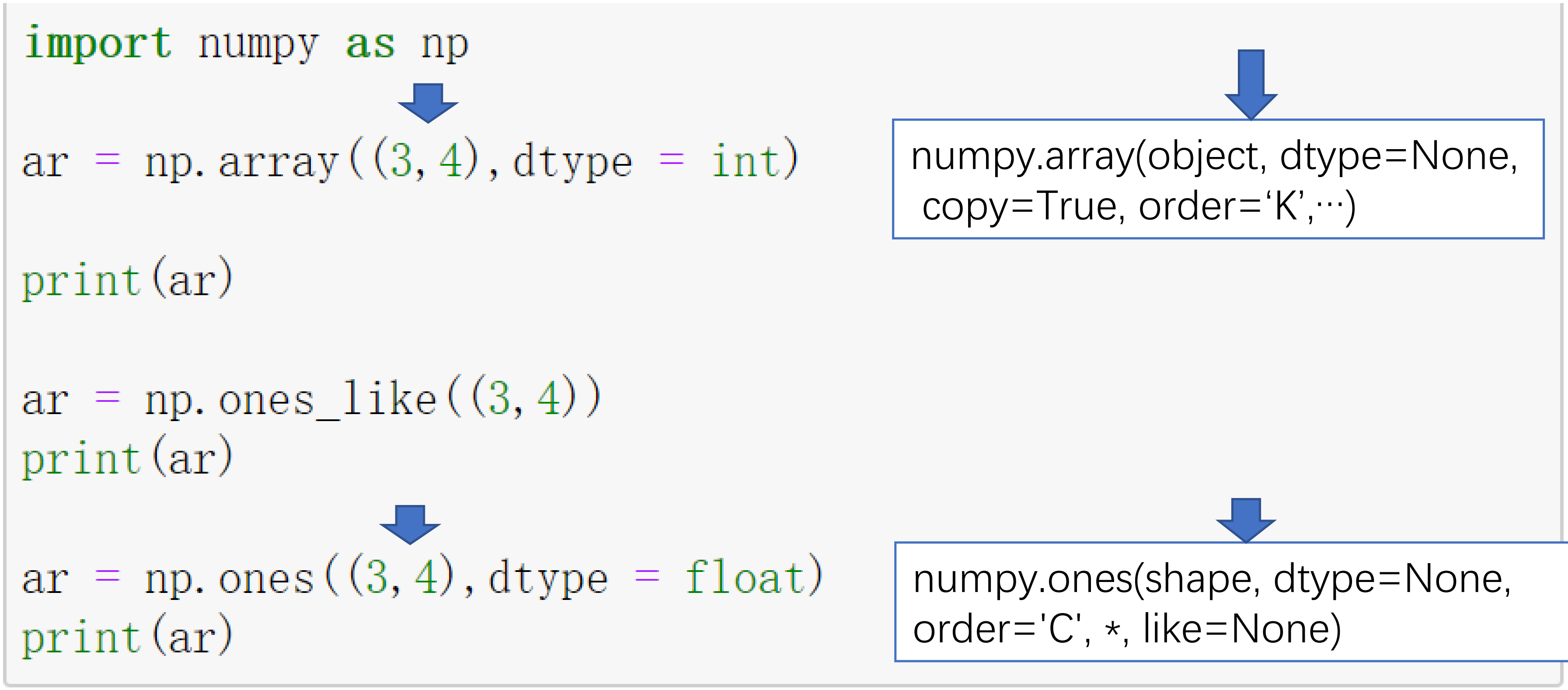


```
import numpy as np

ar = np.array([3.14, 4, 2, 3])
print(type(ar))
ar
```

```
<class 'numpy.ndarray'>

array([3.14, 4.  , 2.  , 3.  ])
```



声明/初始化 一个ndarray对象



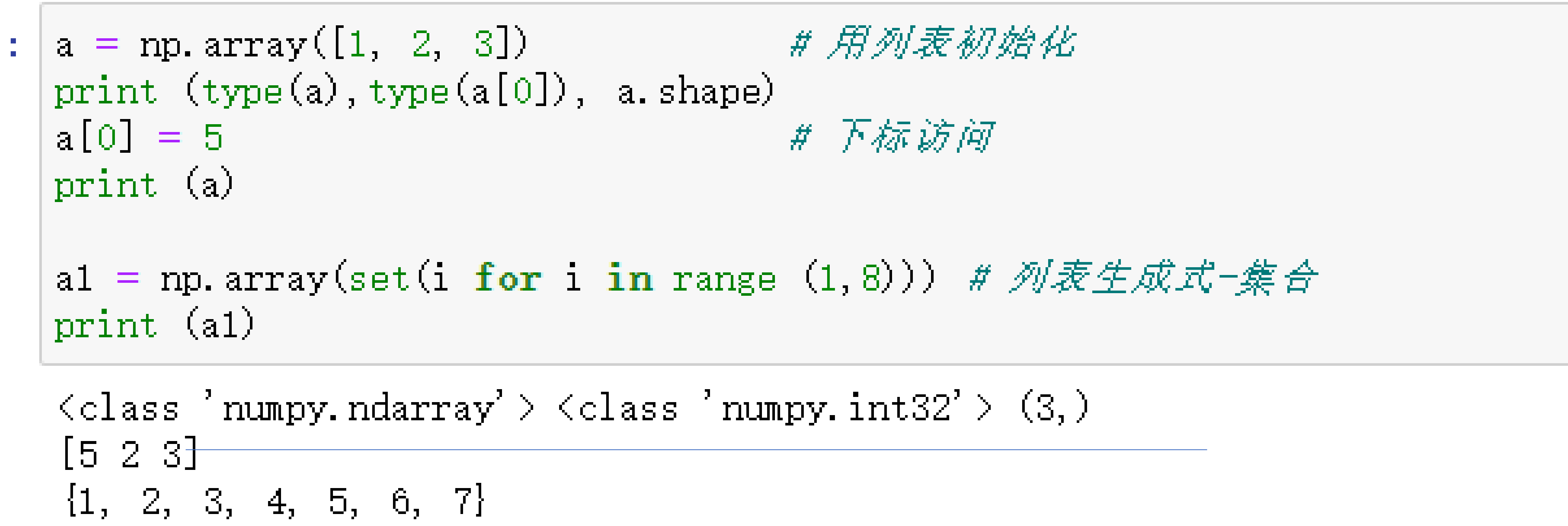
```
import numpy as np

ar = np.array((3,4),dtype = int)

print(ar)

ar = np.ones_like((3,4))
print(ar)

ar = np.ones((3,4),dtype = float)
print(ar)
```

numpy.array(object, dtype=None, copy=True, order='K',…)

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

```
[3 4]
[1 1]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

用python的容器（迭代器）类型来初始化ndarray



```
a = np.array([1, 2, 3])                  # 用列表初始化
print (type(a),type(a[0]), a.shape)
a[0] = 5                                 # 下标访问
print (a)

a1 = np.array(set(i for i in range (1,8))) # 列表生成式-集合
print (a1)
```

```
<class 'numpy.ndarray'> <class 'numpy.int32'> (3,)
[5 2 3]
{1, 2, 3, 4, 5, 6, 7}
```

```python
b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print (b)
```

```
[[1 2 3]
 [4 5 6]]
```

一些常见的初始化矩阵方法

```python
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)
```

```python
np.empty( (2,3) )          # uninitialized, output may vary np.zeros((3,4))
```

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

```
c = np.full((2,2), 7) # Create a constant array
print (c)
```

```
[[7 7]
 [7 7]]
```

一些常见的初始化矩阵方法（续）

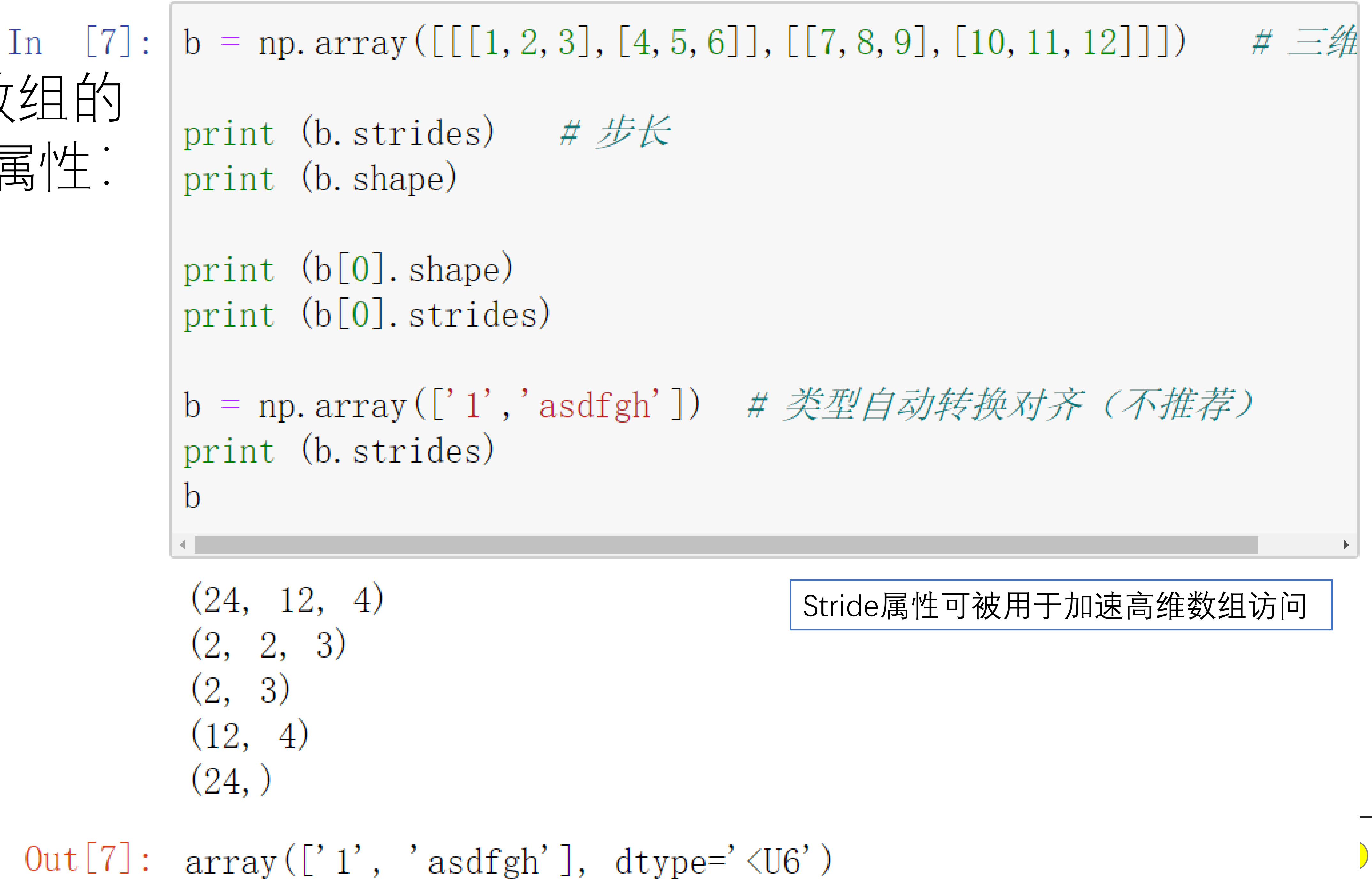
```
d = np.eye(3)          # Create a 3x3 identity matrix
print (d)
```

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

```
e = np.random.random((2,2)) # Create an array filled with random values
print (e)
```

```
[[0.90184008 0.19514327]
 [0.8609894  0.80850789]]
```

高维数组的Stride属性：

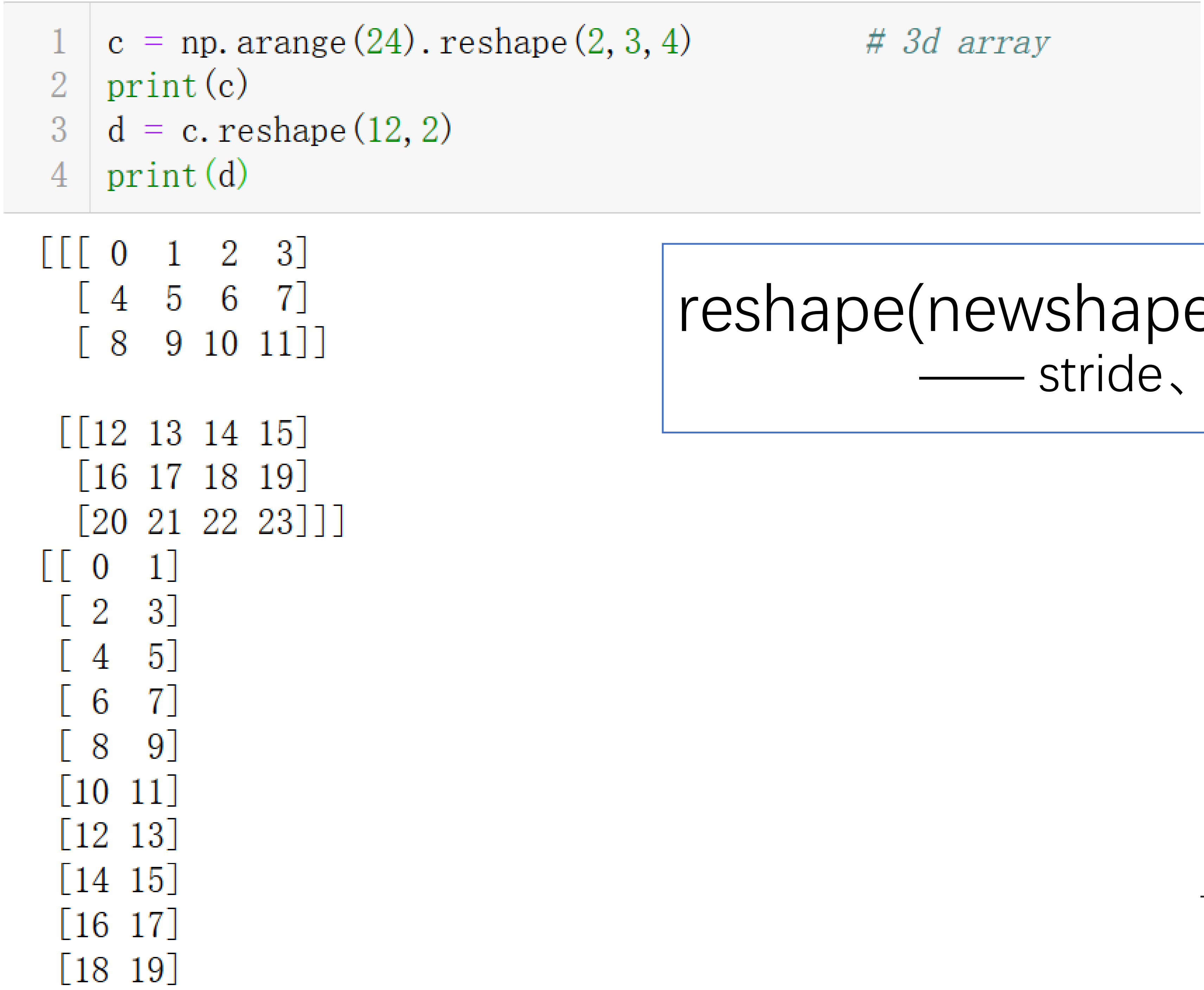


```
In [7]: b = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])   # 三维

        print (b.strides)   # 步长
        print (b.shape)

        print (b[0].shape)
        print (b[0].strides)

        b = np.array(['1','asdfgh'])   # 类型自动转换对齐（不推荐）
        print (b.strides)
        b
```

```
(24, 12, 4)
(2, 2, 3)
(2, 3)
(12, 4)
(24,)
```

```
Out[7]: array(['1', 'asdfgh'], dtype='<U6')
```

Stride属性可被用于加速高维数组访问



```
c = np.arange(24).reshape(2,3,4)          # 3d array
print(c)
d = c.reshape(12,2)
print(d)
```

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

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

reshape(newshape),like(b)：

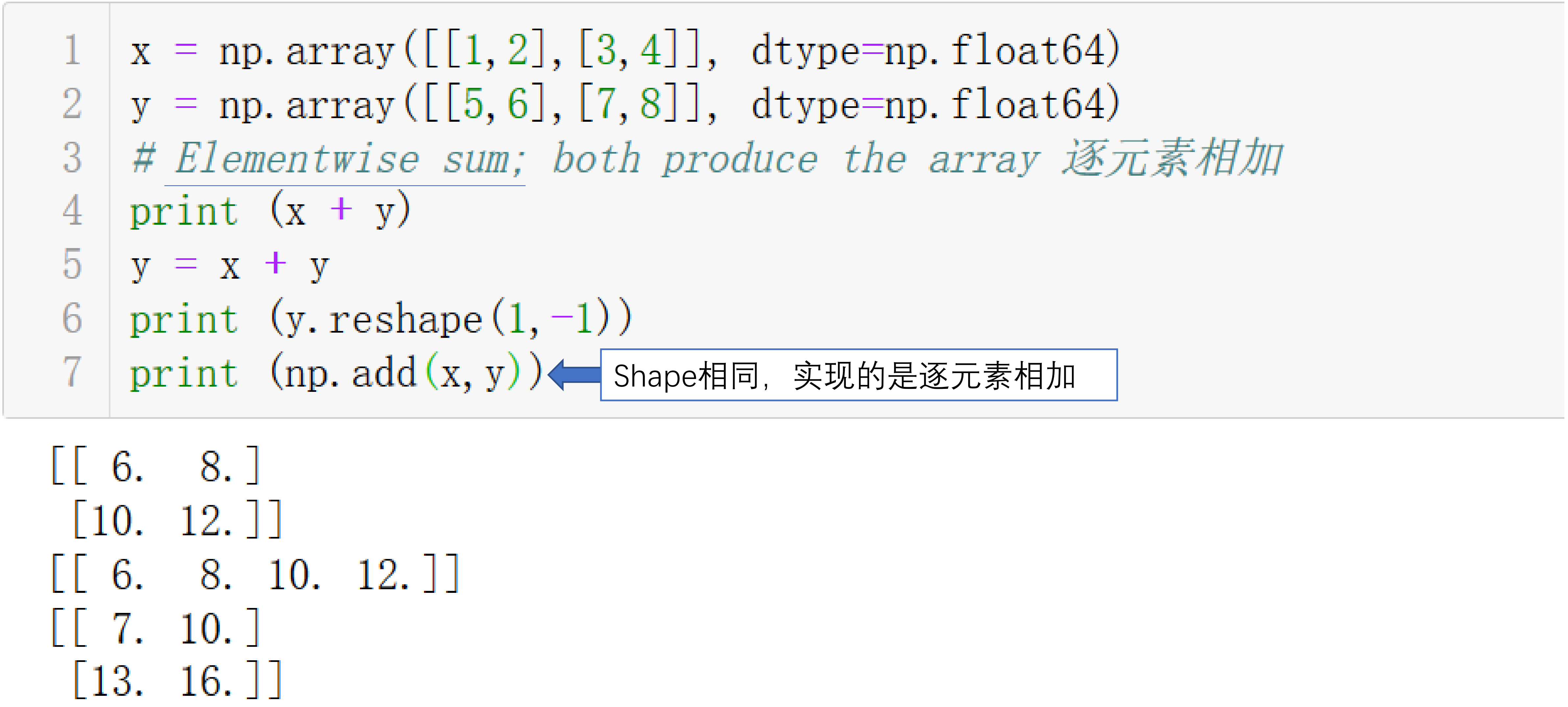
—— stride、shape属性调整：

Reshape与base-data：

```python
b = np.array([[1,2,3],[4,5,6]])     # Create a rank 2 array
print(b.reshape(6))   # 等价 np.reshape(b, 6)
print (b)

c = b.reshape(1,6) # c是两维数组
print (c)
c[0][1] = 9          # c是一个view
print(b.reshape(1,3,-1))    # -1 是自动计算出未标明的维度参数
```

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

Array math (数组间的算术运算， 逐元素对应计算)



```python
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# Elementwise sum; both produce the array 逐元素相加
print (x + y)
y = x + y
print (y.reshape(1,-1))
print (np.add(x,y))
```

Shape相同，实现的是逐元素相加

```
[[ 6.   8.]
 [10.  12.]]
[[ 6.   8.  10.  12.]]
[[ 7.  10.]
 [13.  16.]]
```

```python
# Elementwise difference; both produce the array
print (y - x)
y = np.subtract(y,x) - x
print (y)
```

```
[[5.  6.]
 [7.  8.]]
[[4.  4.]
 [4.  4.]]
```

```python
# Elementwise product; both produce the array
print (x * y)
print (np.multiply(x,  y))
```

Add, subtract, multiply, divide

```
[[ 4.   8.]
 [12.  16.]]
[[ 4.   8.]
 [12.  16.]]
```

```python
# Elementwise division; both produce the array
print (x / y)
print (np.divide(x, y))
```

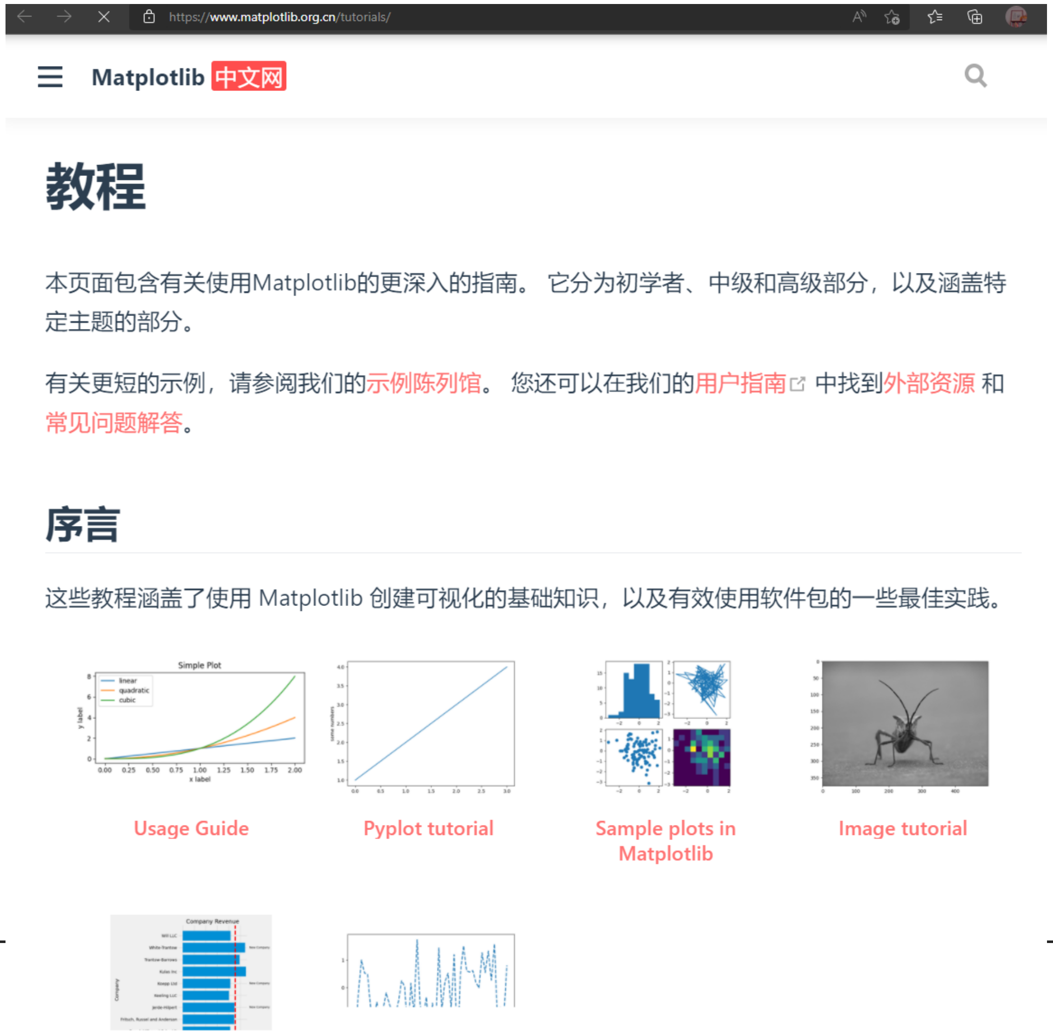
```
[[0.25 0.5 ]
 [0.75 1.  ]]
[[0.25 0.5 ]
 [0.75 1.  ]]
```

```python
# Elementwise square root; produces the array
y = np.sqrt(y)
print (np.sqrt(y))
```

```
[[1.41421356 1.41421356]
 [1.41421356 1.41421356]]
```

可视化软件包matplotlib

https://www.matplotlib.org.cn/tutorials



https://www.matplotlib.org.cn/tutorials/

Matplotlib 中文网

教程

本页面包含有关使用Matplotlib的更深入的指南。 它分为初学者、中级和高级部分，以及涵盖特定主题的部分。

有关更短的示例，请参阅我们的示例陈列馆。 您还可以在我们的用户指南 中找到外部资源 和常见问题解答。

序言

这些教程涵盖了使用 Matplotlib 创建可视化的基础知识，以及有效使用软件包的一些最佳实践。

Usage Guide

Pyplot tutorial

Sample plots in Matplotlib

Image tutorial

Importing Matplotlib

Just as we use the `np` shorthand for NumPy and the `pd` shorthand for Pandas, we will use some standard shorthands for Matplotlib imports:

```python
import matplotlib as mpl
import matplotlib.pyplot as plt
```

The `plt` interface is what we will use most often, as we shall see throughout this chapter.

Setting Styles

We will use the `plt.style` directive to choose appropriate aesthetic styles for our figures. Here we will set the `classic` style, which ensures that the plots we create use the classic Matplotlib style:

```python
plt.style.use('classic')
```

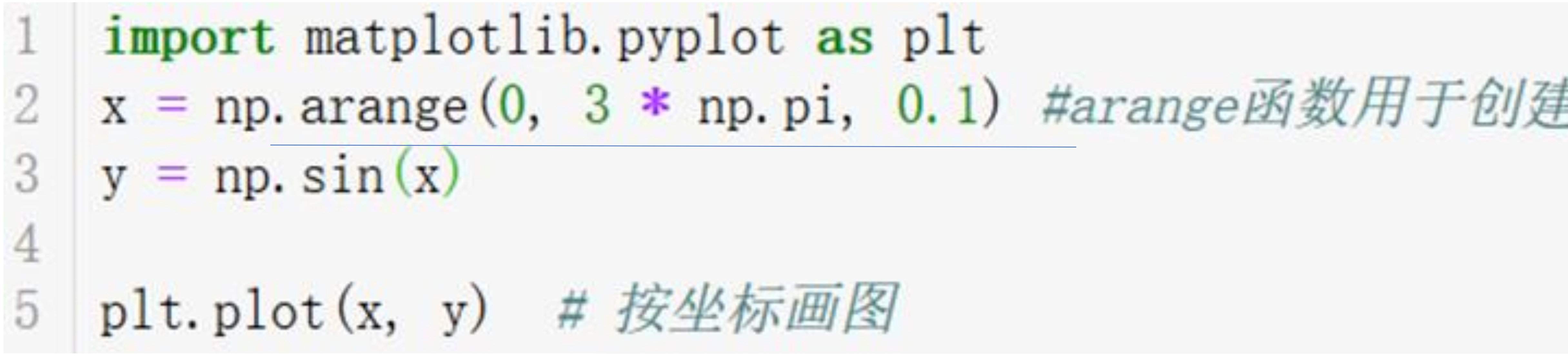
Plotting from a script

If you are using Matplotlib from within a script, the function `plt.show()` is your friend. `plt.show()` starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure or figures.

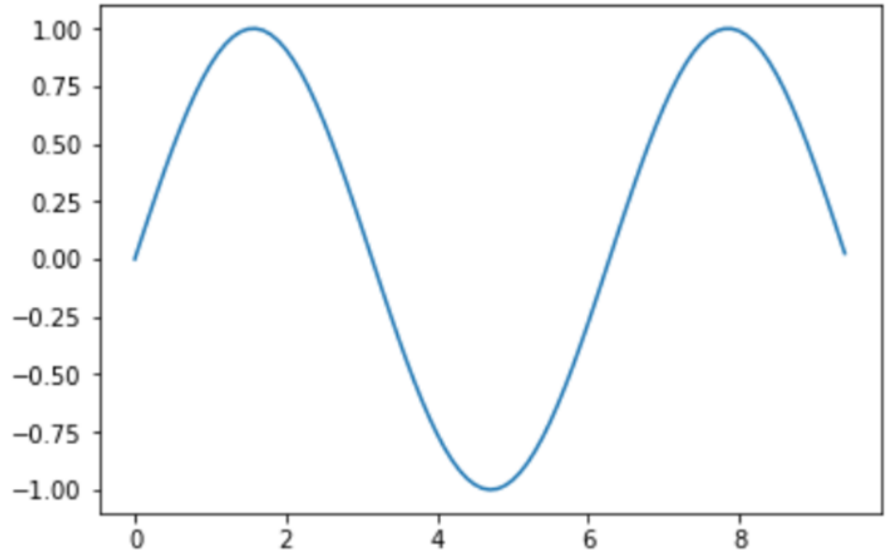
So, for example, you may have a file called *myplot.py* containing the following:

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

Plotting —— 二维画图软件包

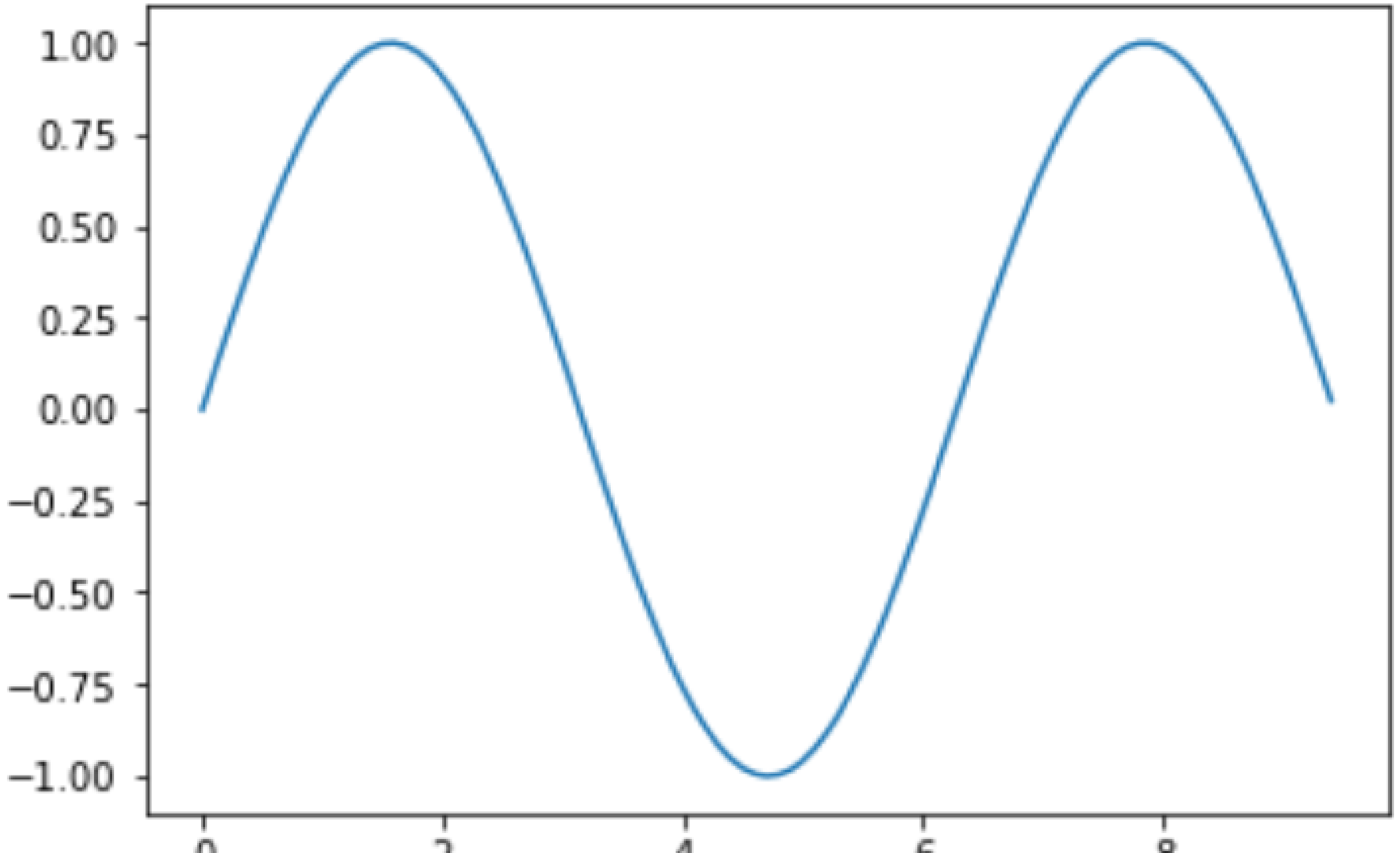


```
import matplotlib.pyplot as plt
x = np.arange(0, 3 * np.pi, 0.1) #arange函数用于创建
y = np.sin(x)

plt.plot(x, y)  # 按坐标画图
```



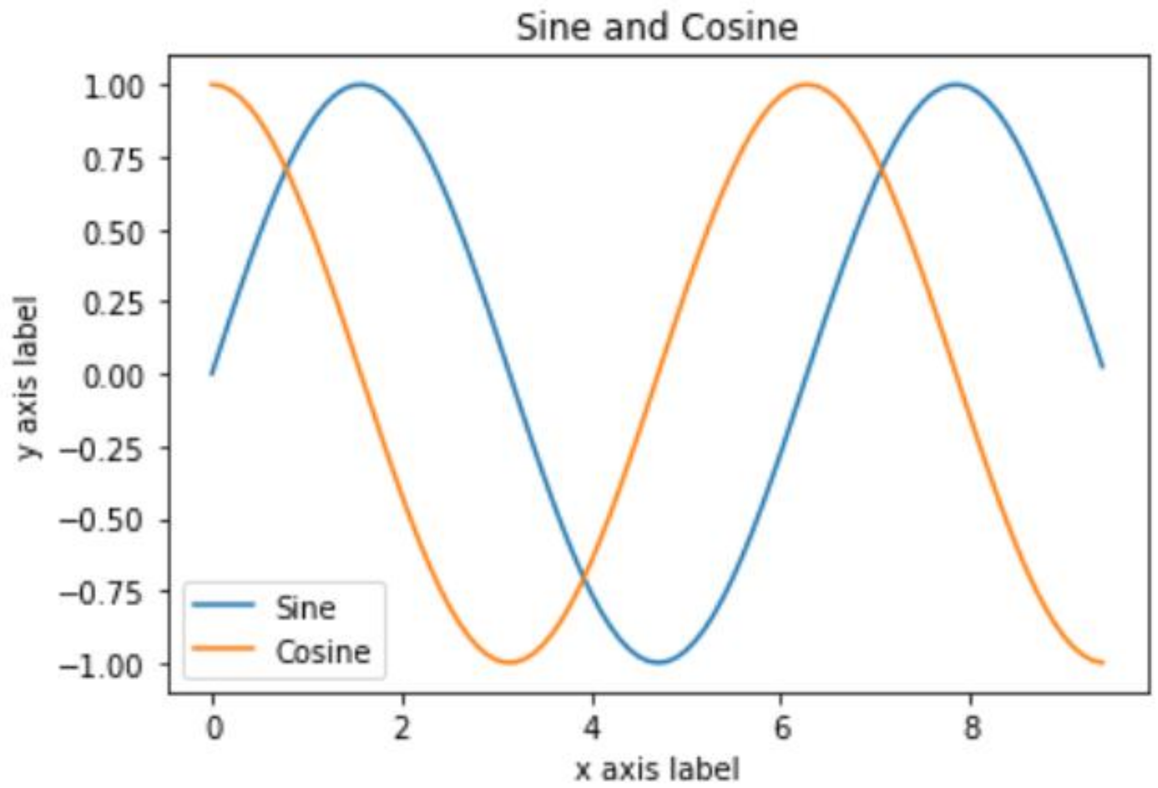
```python
import matplotlib.pyplot as plt
x = np.arange(0, 3 * np.pi, 0.1) #arange函数用于创建等差数组
y = np.sin(x)

plt.plot(x, y)  # 按坐标画图
```

```
[<matplotlib.lines.Line2D at 0x1b20b857128>]
```

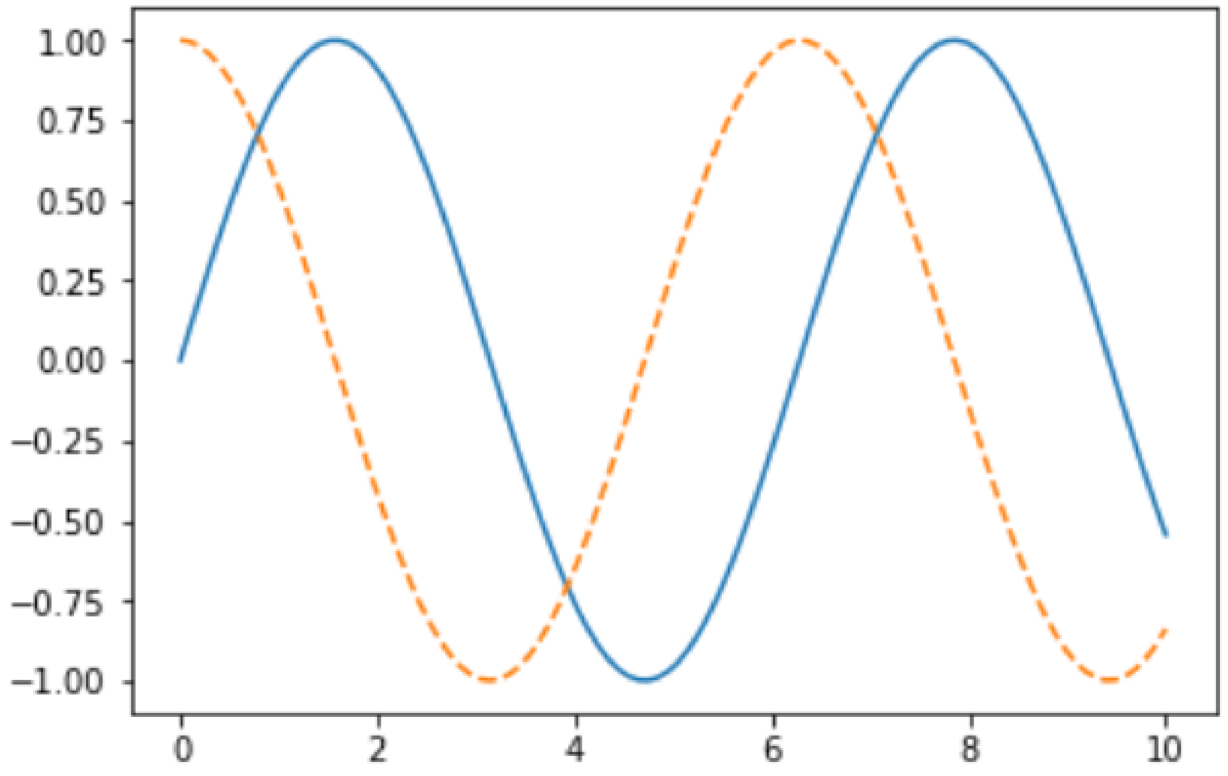


```
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
```

```
<matplotlib.legend.Legend at 0x1d7e537b0b8>
```



```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 100)

#fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```



Saving Figures to File

One nice feature of Matplotlib is the ability to save figures in a wide variety of formats. Saving a figure can be done using the `savefig()` command. For example, to save the previous figure as a PNG file, you can run this:

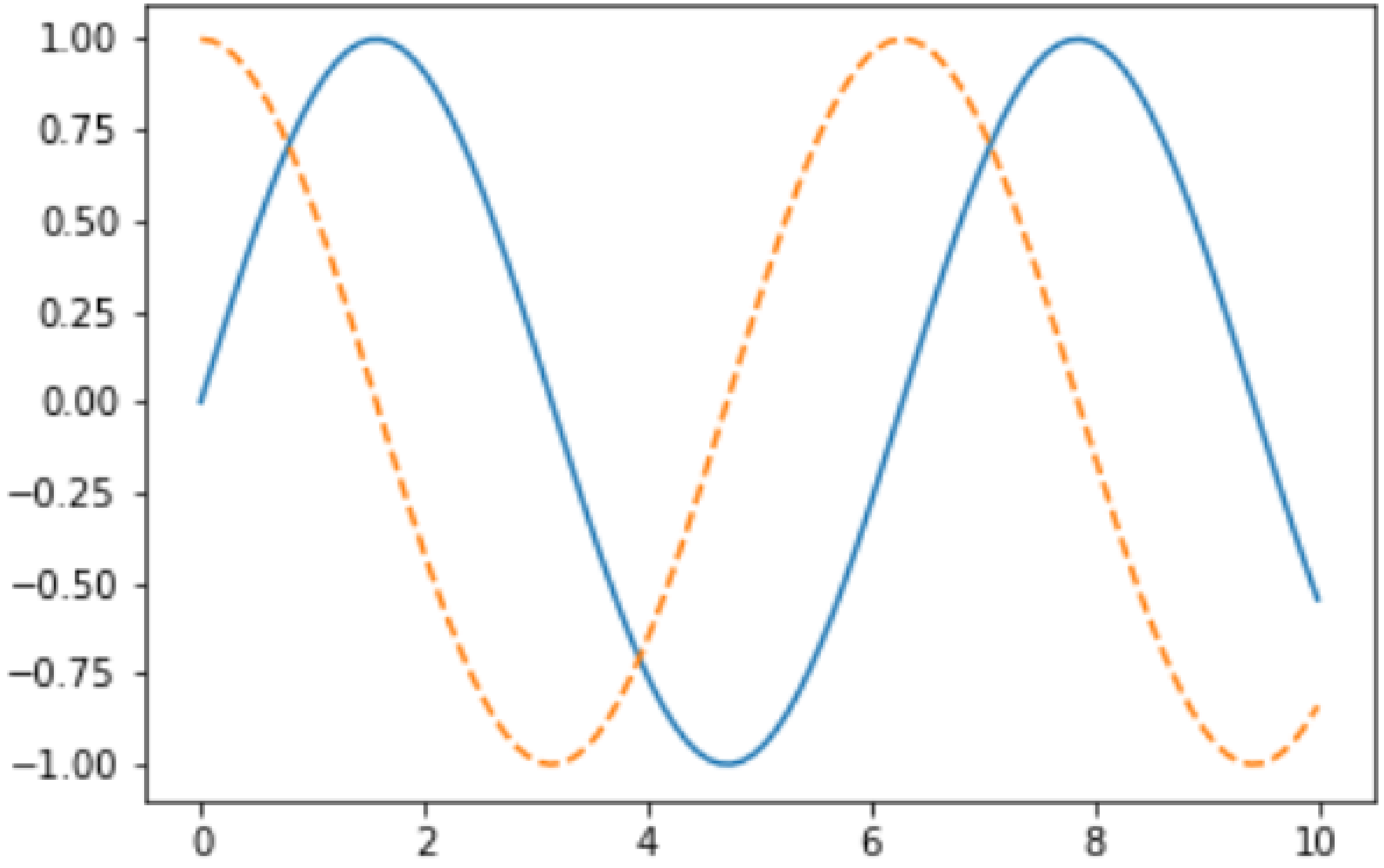
```
fig.savefig('my_figure.png')
```

We now have a file called `my_figure.png` in the current working directory:

```
!dir  my_figure.png
```

```
 驱动器 C 中的卷是 Windows
 卷的序列号是 368A-7FED

 C:\Users\hjf_p\2019python\notebooks 的目录

2019/03/27  07:09            22,604 my_figure.png
               1 个文件         22,604 字节
               0 个目录 138,808,619,008 可用字节
```

```python
from IPython.display import Image
Image('my_figure.png')
```



Numpy的向量计算

- 运算的广播机制
- 向量点积与矩阵乘法

Broadcasting

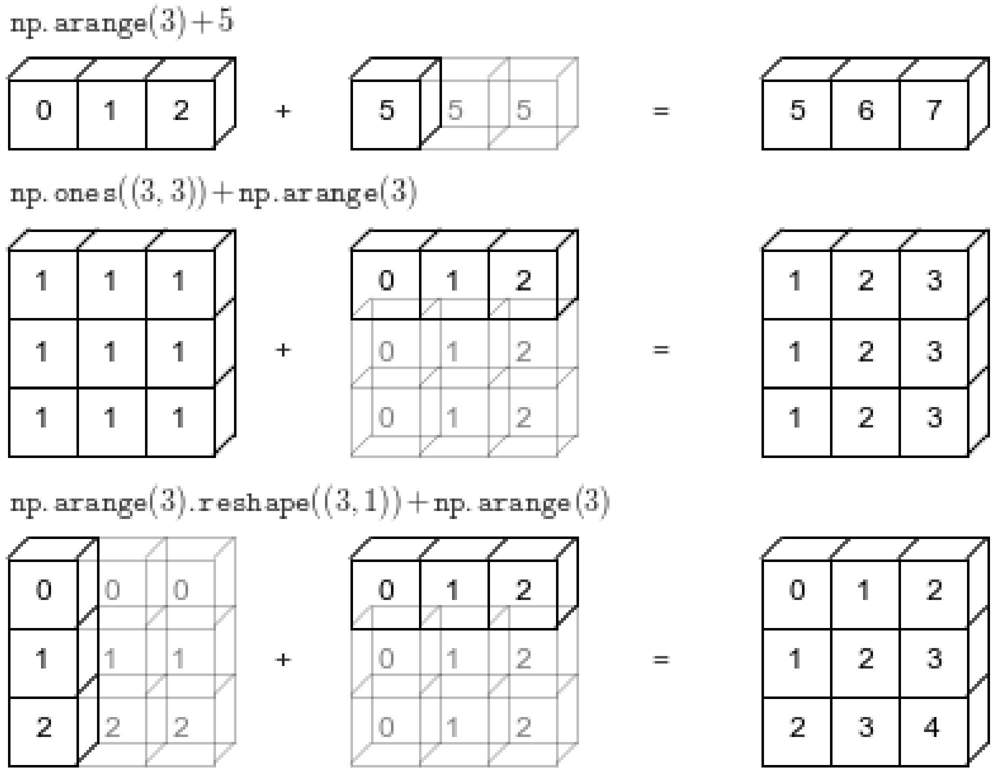
Numpy中 常量-向量-矩阵运算中的广播机制

Each universal function takes array inputs and produces array outputs by performing the core function element-wise on the inputs (where an element is generally a scalar, but can be a vector or higher-order sub-array for generalized ufuncs). Standard broadcasting rules are applied so that inputs not sharing exactly the same shapes can still be usefully operated on. Broadcasting can be understood by four rules:

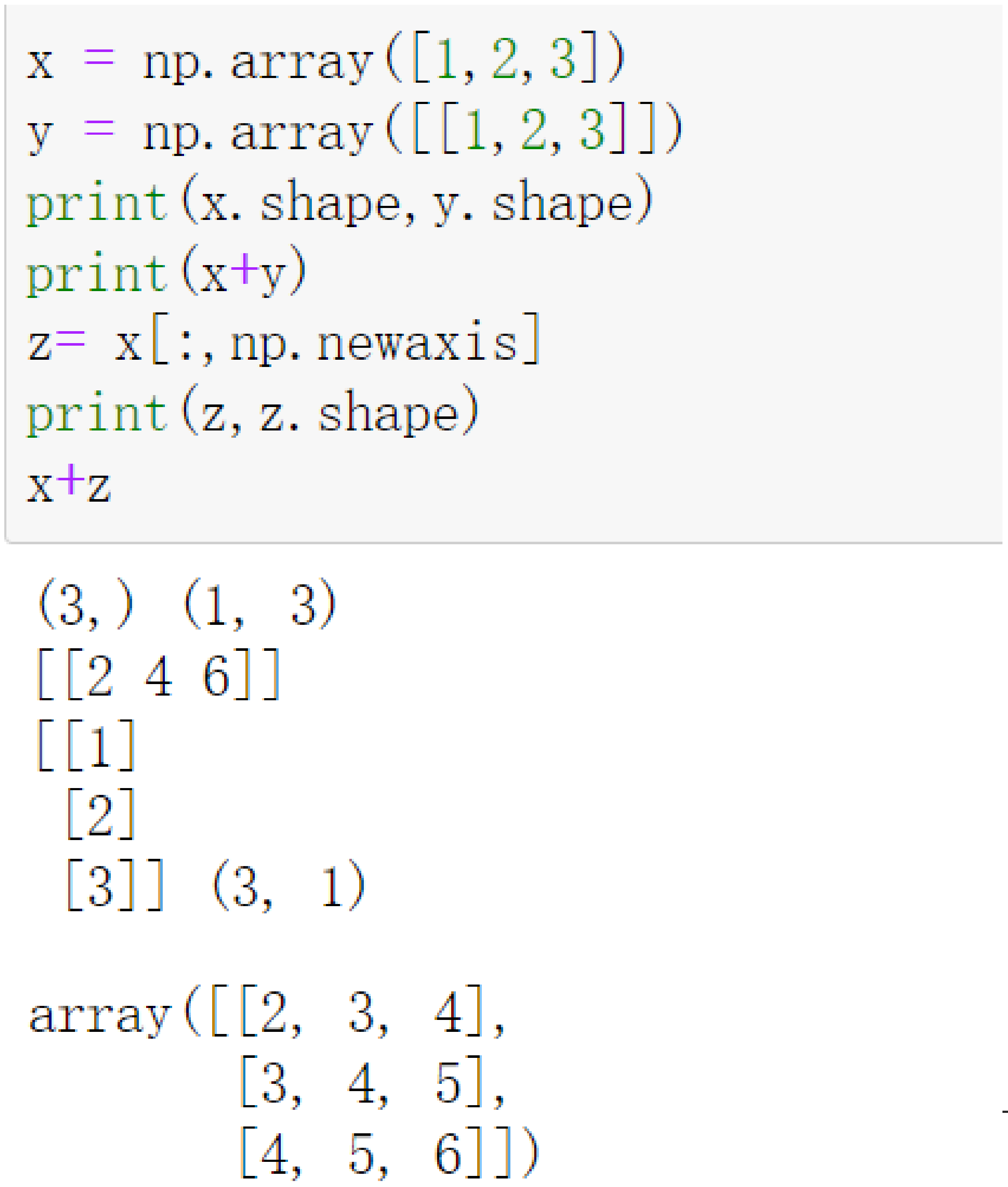
```
a = np.array([[1, 2, 3, 4], [10, 20, 30, 40]])
b = np.array([100, 100, 100, 100])
print(a + b)        # 行 对矩阵广播

c = np.array([3])
a * c               # 元素 对矩阵广播
```

```
[[101 102 103 104]
 [110 120 130 140]]
```

```
array([[  3,   6,   9,  12],
       [ 30,  60,  90, 120]])
```



行-列交叉广播

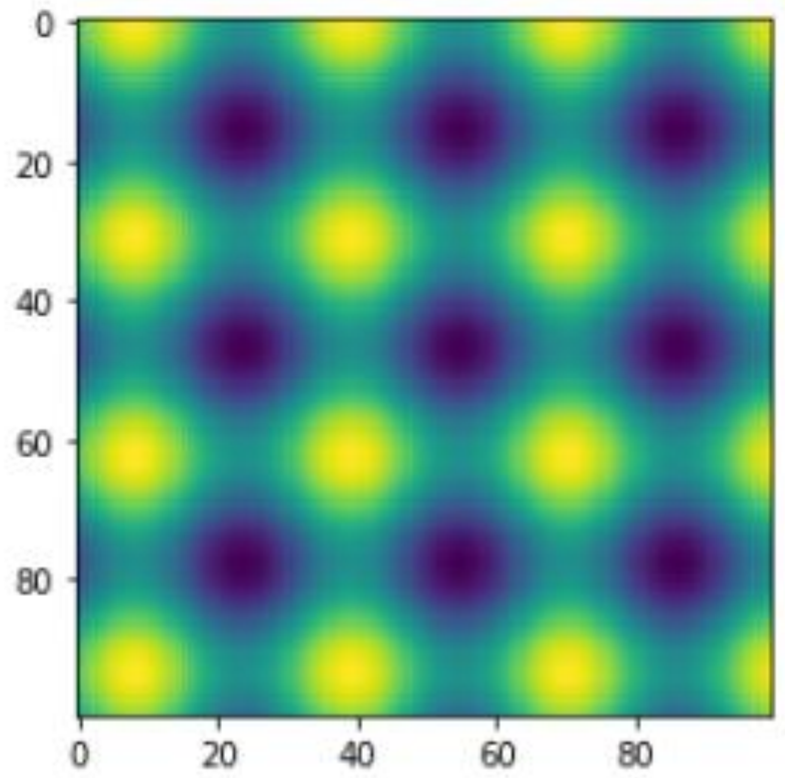


```python
x = np.array([1,2,3])
y = np.array([[1,2,3]])
print(x.shape,y.shape)
print(x+y)
z= x[:,np.newaxis]
print(z,z.shape)
x+z
```

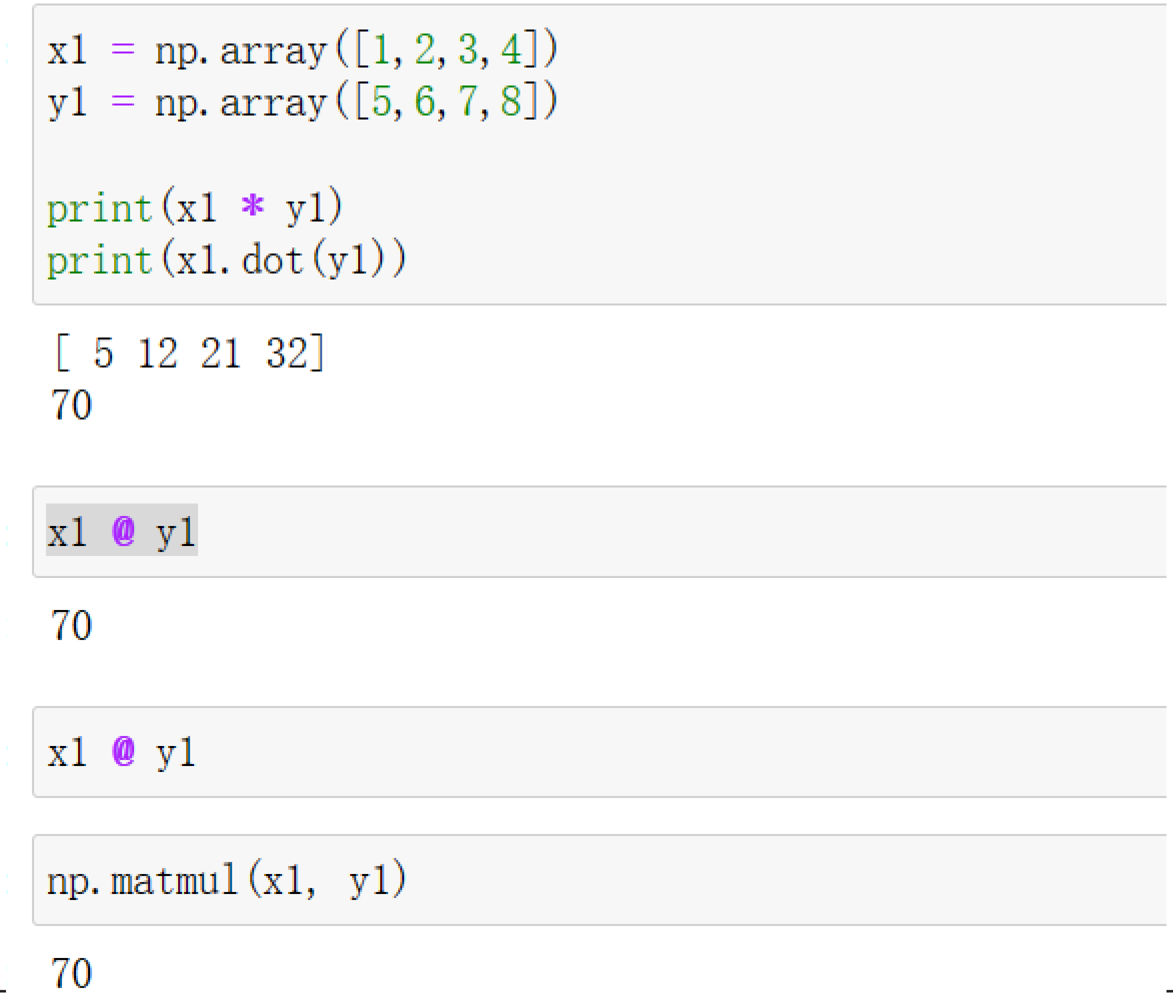
```
(3,) (1, 3)
[[2 4 6]]
[[1]
 [2]
 [3]] (3, 1)

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

```
# define a function z = f(x, y)
# x and y have 100 steps from 0 to 10

x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)[:, np.newaxis]  # 增加一个维度

z = np.sin(2*x) + np.cos(2*y) # 广播合成2维度矩阵
plt.imshow(z)
```

```
<matplotlib.image.AxesImage at 0x23bca086c88>
```



在numpy中 * 代表算数乘法， .dot() 或 @ 用于表示向量乘法



```
x1 = np.array([1,2,3,4])
y1 = np.array([5,6,7,8])

print(x1 * y1)
print(x1.dot(y1))
```

```
[ 5 12 21 32]
70
```

```
x1 @ y1
```

```
70
```

```
x1 @ y1
```

```
np.matmul(x1, y1)
```

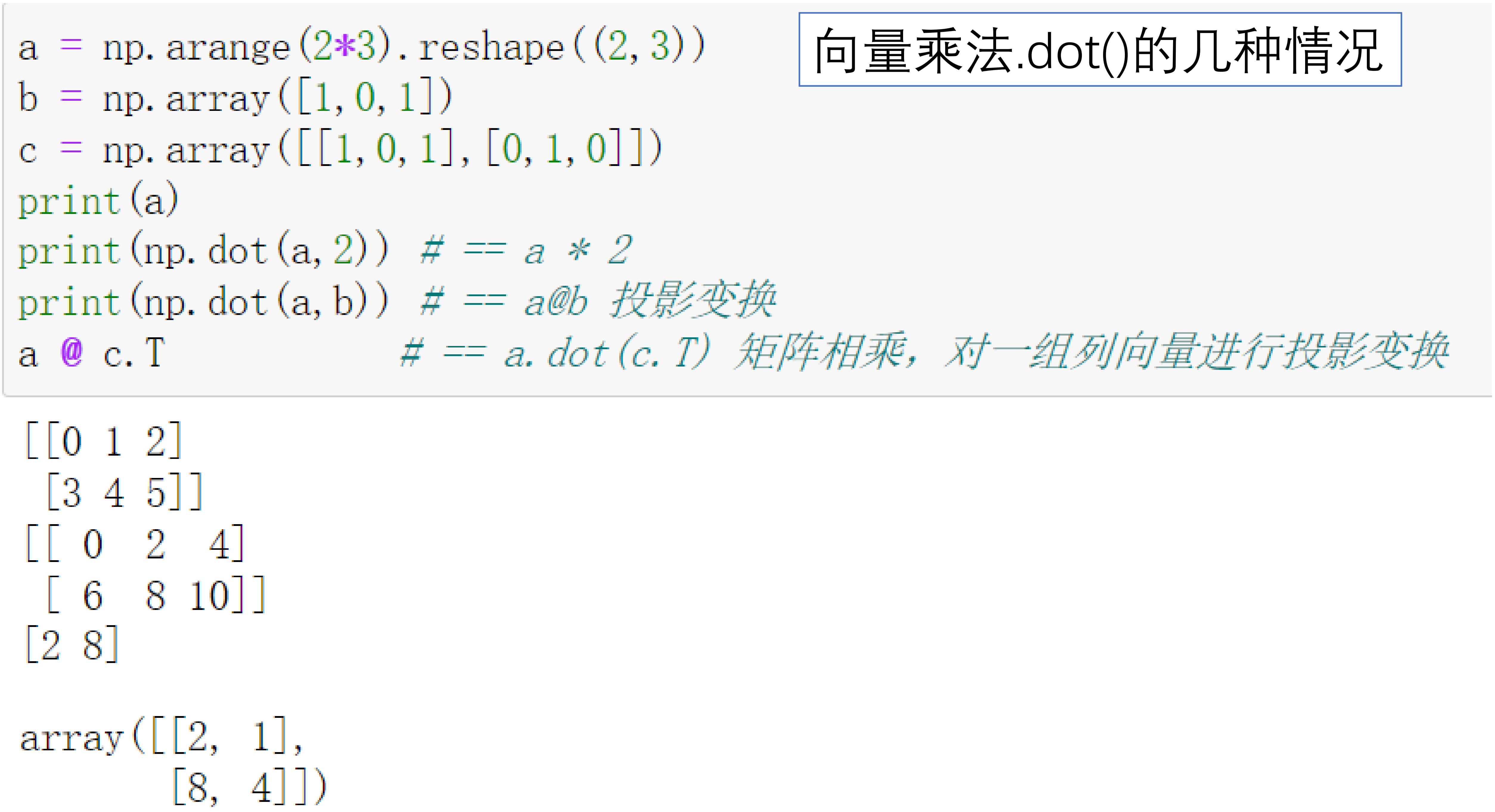
```
70
```

矩阵和向量乘法

向量乘法函数.dot() 计算会因为操作数类型的不同有进阶的推广：

- np.dot(a,b,out=None),如果a和b是1维数组，就相当于两个向量的点积
- 如果a是N维数组，b是1维数组，就是b的向量乘法对a的最后一维进行广播。
- 如果a和b是2维数组，就表达为a和b的矩阵乘法。可以理解为二维向量乘法广播。建议最好用a@b 或 matmul(a,b)
- 如果a是N维数组，b是M维数组，是沿着a的最后一个轴向量和b的倒数第二个轴的向量乘法广播。当然，这两个维度的向量的维度必须是相同的。

向量乘法.dot()的几种情况



```
a = np.arange(2*3).reshape((2,3))
b = np.array([1,0,1])
c = np.array([[1,0,1],[0,1,0]])
print(a)
print(np.dot(a,2)) # == a * 2
print(np.dot(a,b)) # == a@b 投影变换
a @ c.T           # == a.dot(c.T) 矩阵相乘，对一组列向量进行投影变换
```

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

array([[2, 1],
       [8, 4]])
```

```
a = np.arange(8).reshape((2,2,2))
b = np.array(['a','b','c','d','e','f'], dtype=object).reshape((2,3))
a, b
```

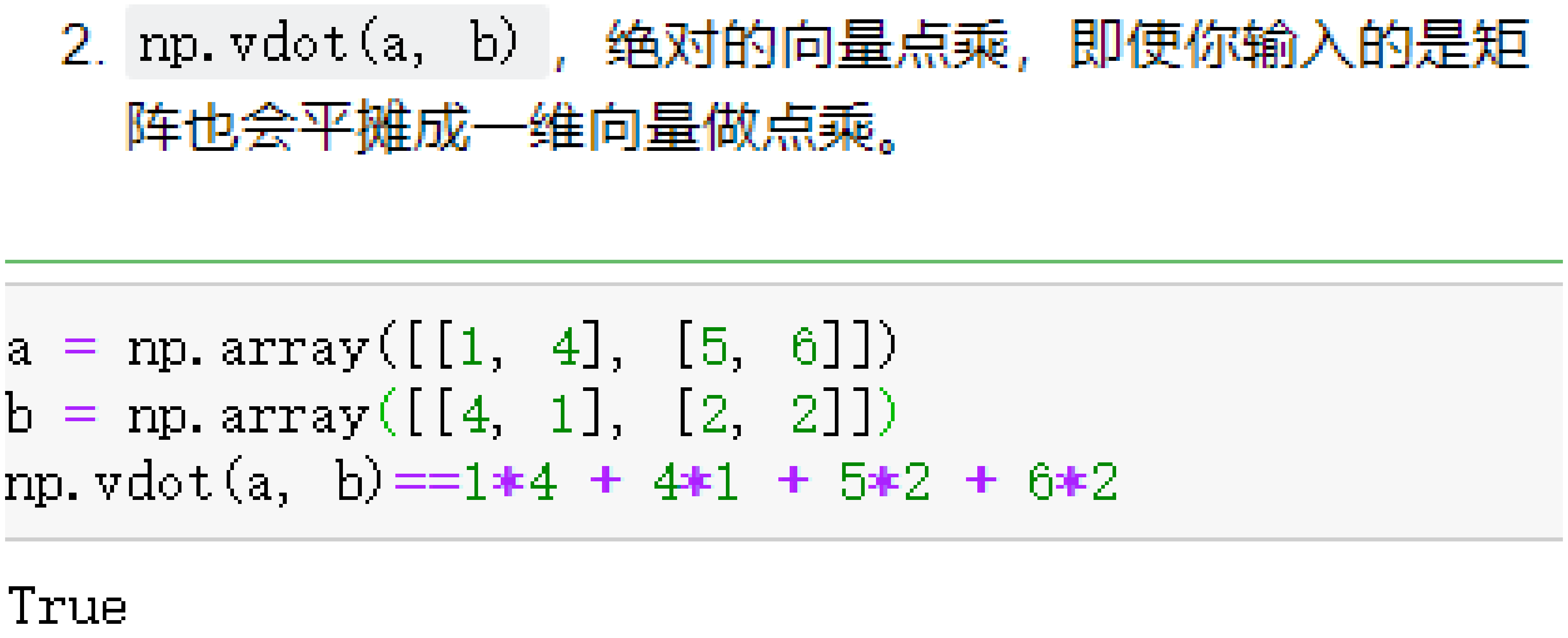
```
(array([[[0, 1],
         [2, 3]],

        [[4, 5],
         [6, 7]]]),
 array([['a', 'b', 'c'],
        ['d', 'e', 'f']], dtype=object))
```

N维数组 dot M维数组：沿着a的最后一个轴和b的倒数第二个轴的乘积和作为元素的广播

```
c = a @ b
print(c, c.shape)
```

```
[[['d' 'e' 'f']
  ['aaddd' 'bbeee' 'ccfff']]

 [['aaaadddd' 'bbbbeeeee' 'ccccfffff']
  ['aaaaaadddddd' 'bbbbbbeeeeeee' 'ccccccffffff']]] (2, 2, 3)
```

计算高维矩阵的逐元素投影量

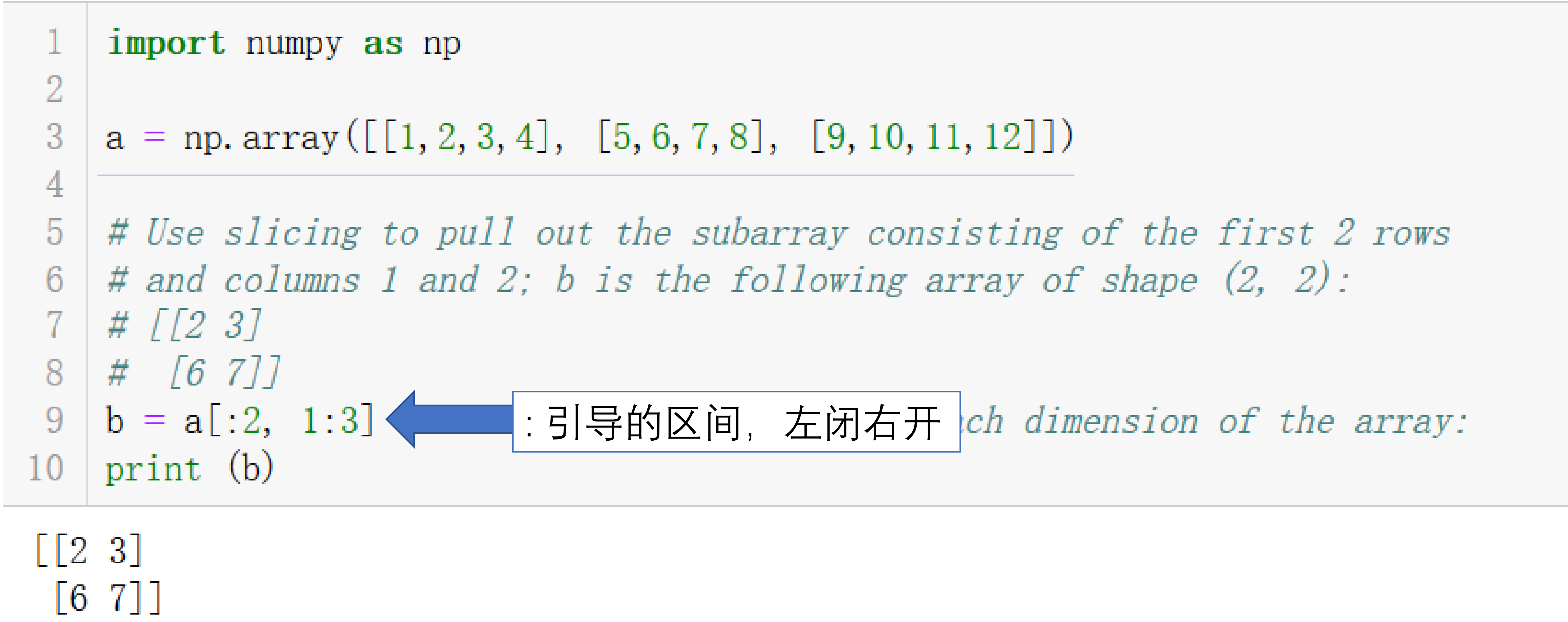


2. `np.vdot(a, b)`，绝对的向量点乘，即使你输入的是矩阵也会平摊成一维向量做点乘。

```
a = np.array([[1, 4], [5, 6]])
b = np.array([[4, 1], [2, 2]])
np.vdot(a, b)==1*4 + 4*1 + 5*2 + 6*2
```

```
True
```

矩阵的下标访问与切片 :Array indexing and Slicing

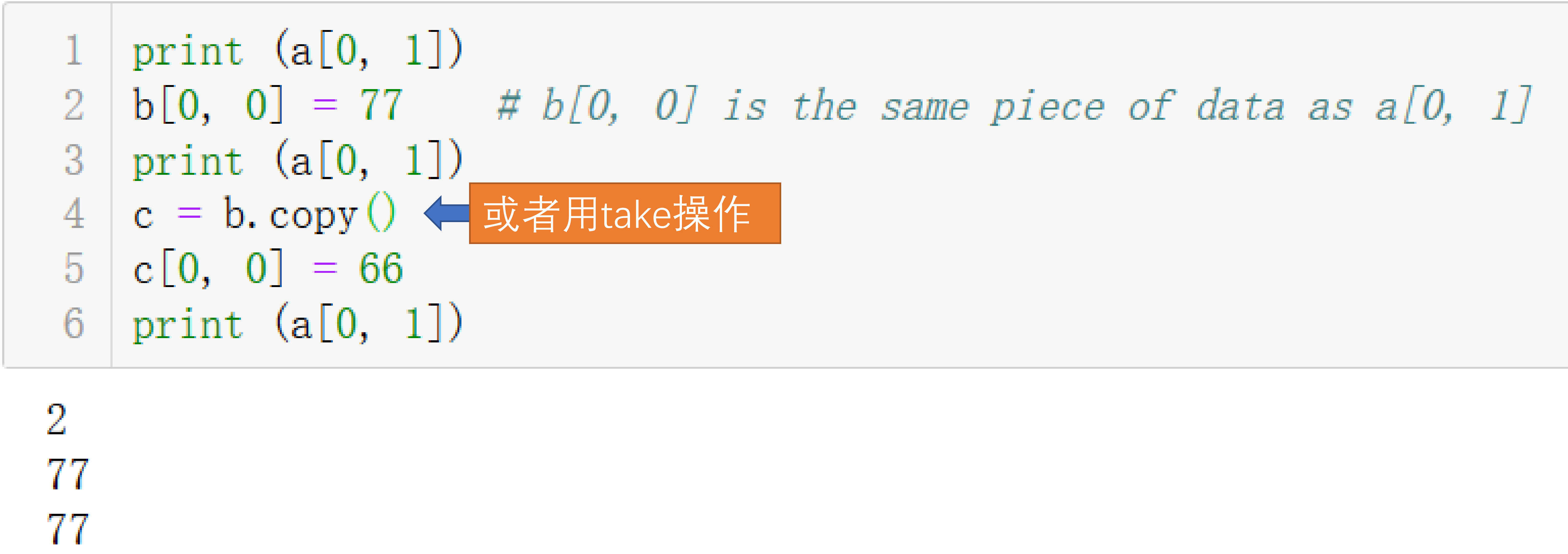


```python
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]                                ch dimension of the array:
print (b)
```

: 引导的区间，左闭右开

```
[[2 3]
 [6 7]]
```

矩阵切片视图： A slice of an array is a view(视图) into the same data, modifying it will modify the original array

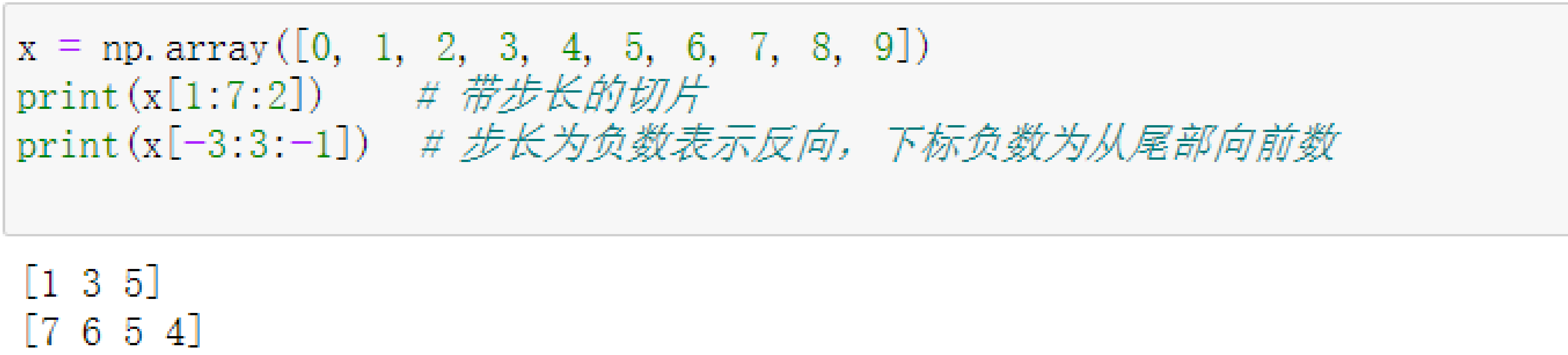


```
print (a[0, 1])
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print (a[0, 1])
c = b.copy()
c[0, 0] = 66
print (a[0, 1])
```

或者用take操作

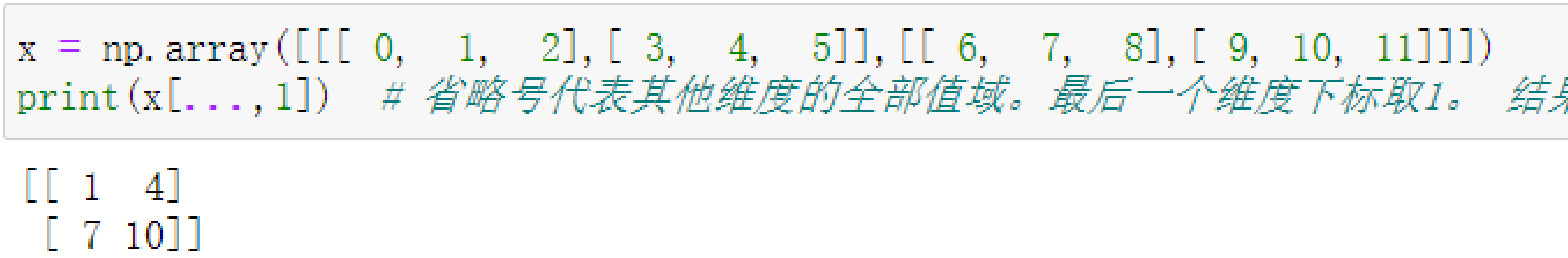
```
2
77
77
```

矩阵切片的步长与区间方向:



```python
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(x[1:7:2])     # 带步长的切片
print(x[-3:3:-1])   # 步长为负数表示反向, 下标负数为从尾部向前数
```

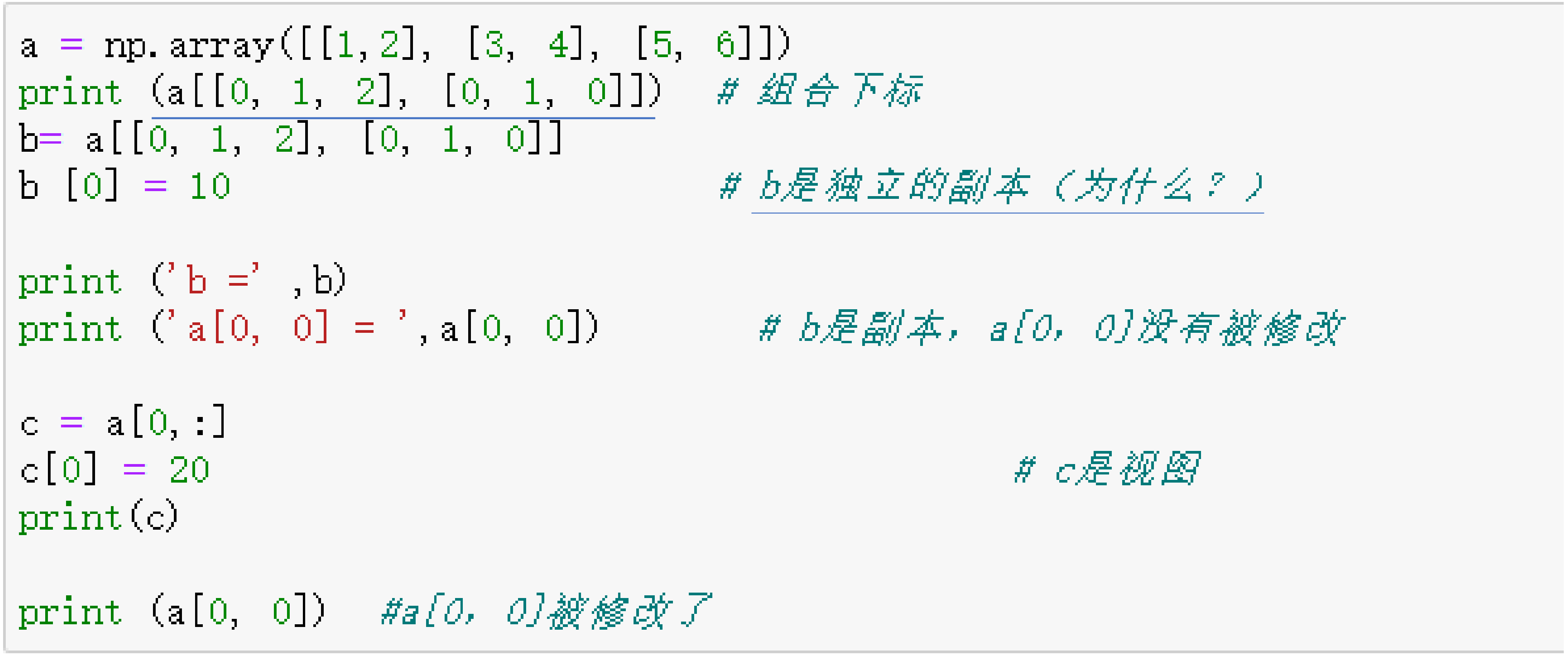
```
[1 3 5]
[7 6 5 4]
```



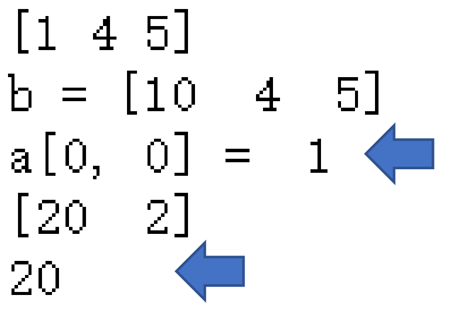
```python
x = np.array([[[ 0,  1,  2],[ 3,  4,  5]],[[ 6,  7,  8],[ 9, 10, 11]]])
print(x[...,1])  # 省略号代表其他维度的全部值域。最后一个维度下标取1。 结
```

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

advance indexing： 通过下标元组集合来标定元素

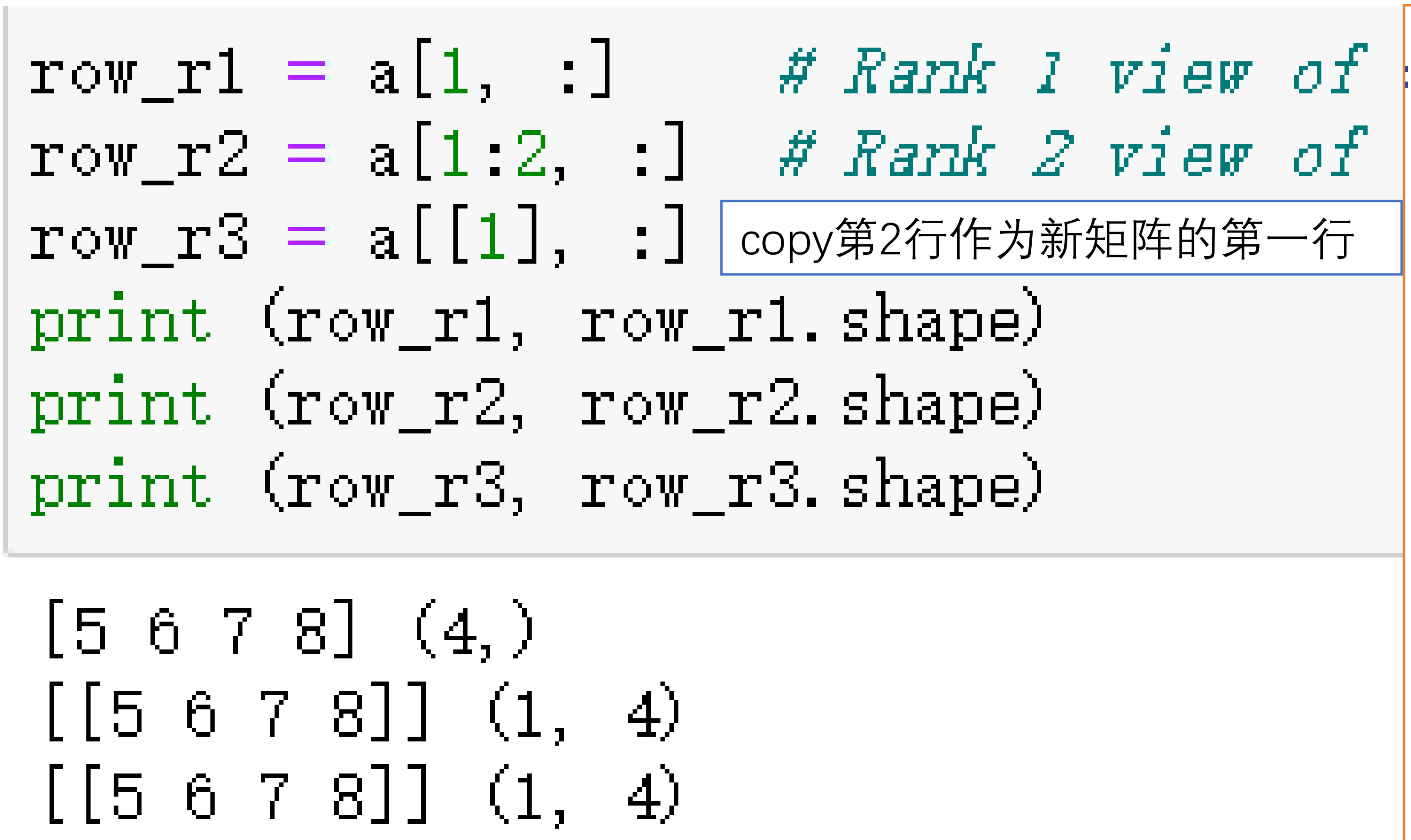


```
a = np.array([[1,2], [3, 4], [5, 6]])
print (a[[0, 1, 2], [0, 1, 0]])    # 组合下标
b= a[[0, 1, 2], [0, 1, 0]]
b [0] = 10                         # b是独立的副本（为什么？）

print ('b =' ,b)
print ('a[0, 0] = ',a[0, 0])         # b是副本，a[0, 0]没有被修改

c = a[0,:]
c[0] = 20                                    # c是视图
print(c)

print (a[0, 0])  #a[0, 0]被修改了
```



```
[1 4 5]
b = [10  4  5]
a[0, 0] =  1
[20  2]
20
```

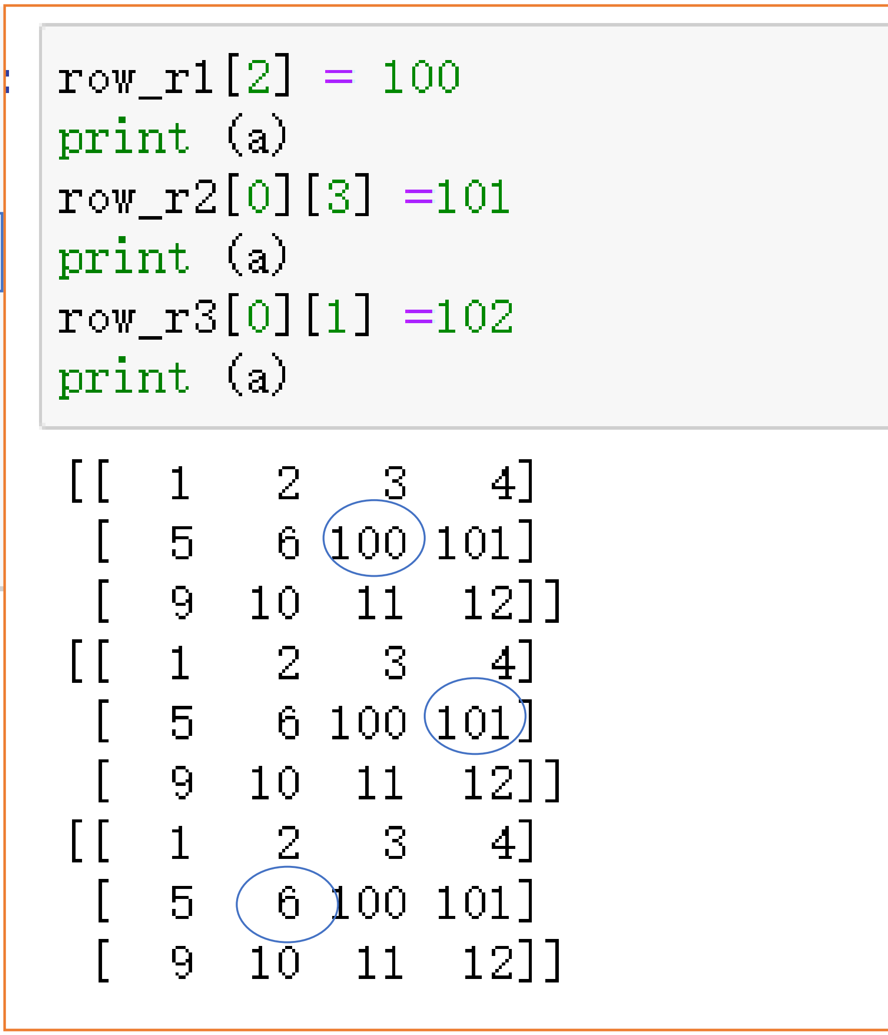
- 切片操作如果直接标定一个维度，会得到一个低一阶的矩阵
- 每个维度都是区段，哪怕区段里只有一个下标
- 如果切片参数为一个list，则默认采用高级索引模式，copy



```
row_r1 = a[1, :]      # Rank 1 view of
row_r2 = a[1:2, :]    # Rank 2 view of
row_r3 = a[[1], :]
print (row_r1, row_r1.shape)
print (row_r2, row_r2.shape)
print (row_r3, row_r3.shape)
```

copy第2行作为新矩阵的第一行

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[[5 6 7 8]] (1, 4)
```

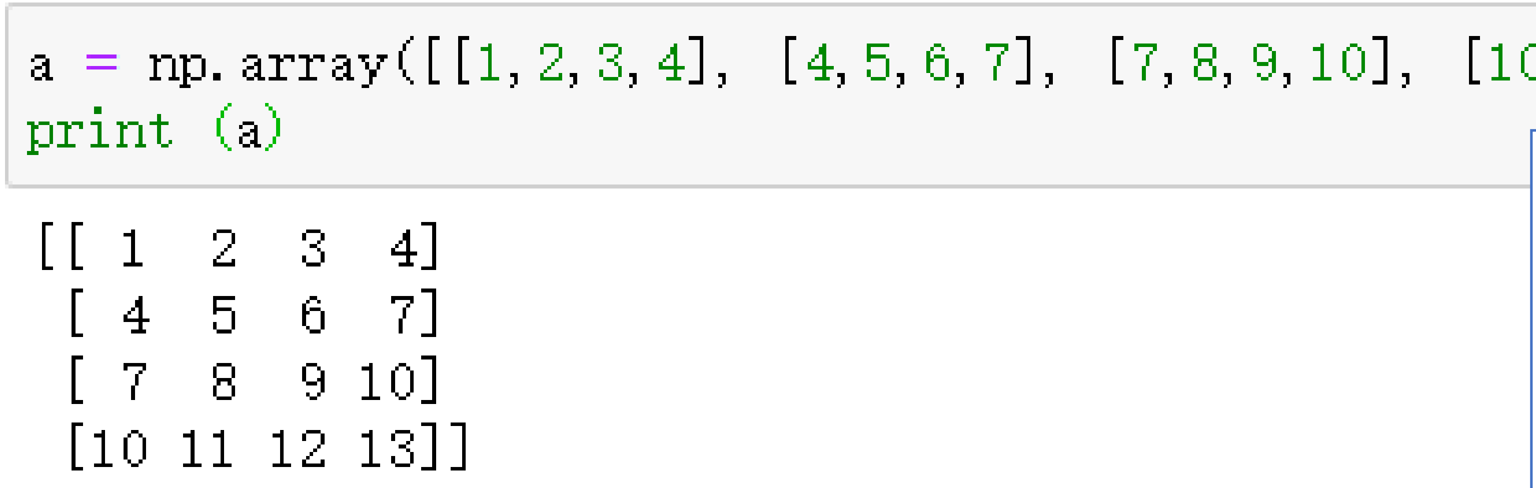


```
row_r1[2] = 100
print (a)
row_r2[0][3] =101
print (a)
row_r3[0][1] =102
print (a)
```

```
[[  1   2   3   4]
 [  5   6 100 101]
 [  9  10  11  12]]
[[  1   2   3   4]
 [  5   6 100 101]
 [  9  10  11  12]]
[[  1   2   3   4]
 [  5   6 100 101]
 [  9  10  11  12]]
```

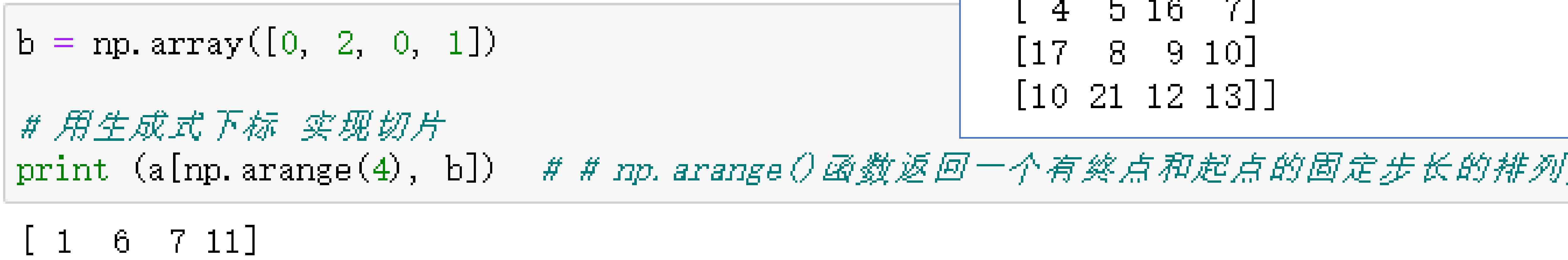
是否是独立的副本可以通过 .base属性
和 .flags.owndata属性来确认

生成式也可以用来做组合下标



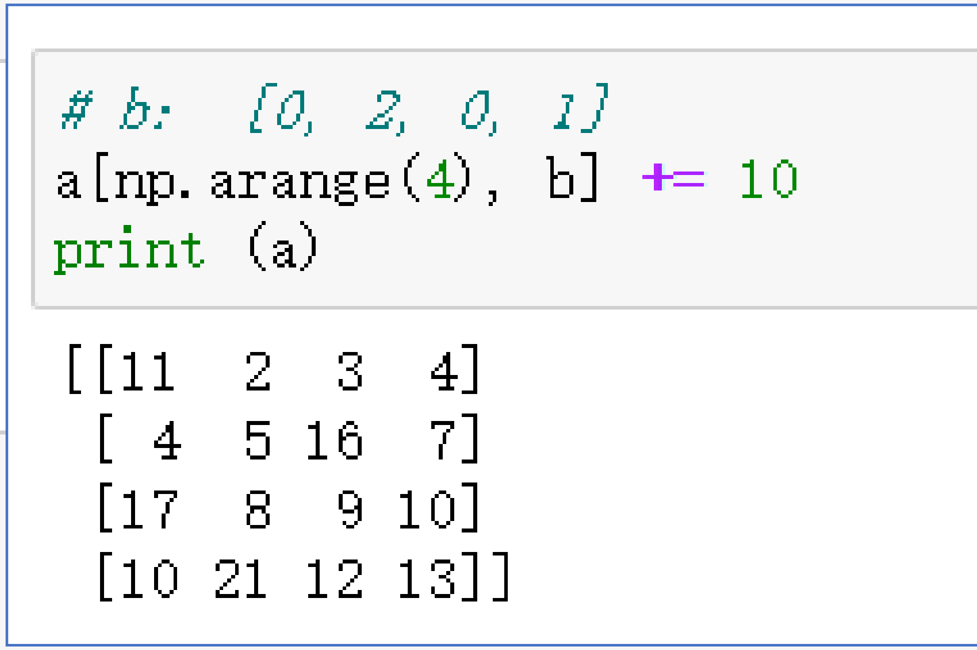
```
a = np.array([[1,2,3,4], [4,5,6,7], [7,8,9,10], [10, 11, 12,13]])
print (a)
```

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



```
b = np.array([0, 2, 0, 1])

# 用生成式下标 实现切片
print (a[np.arange(4), b])   # # np.arange()函数返回一个有终点和起点的固定步长的排列
```

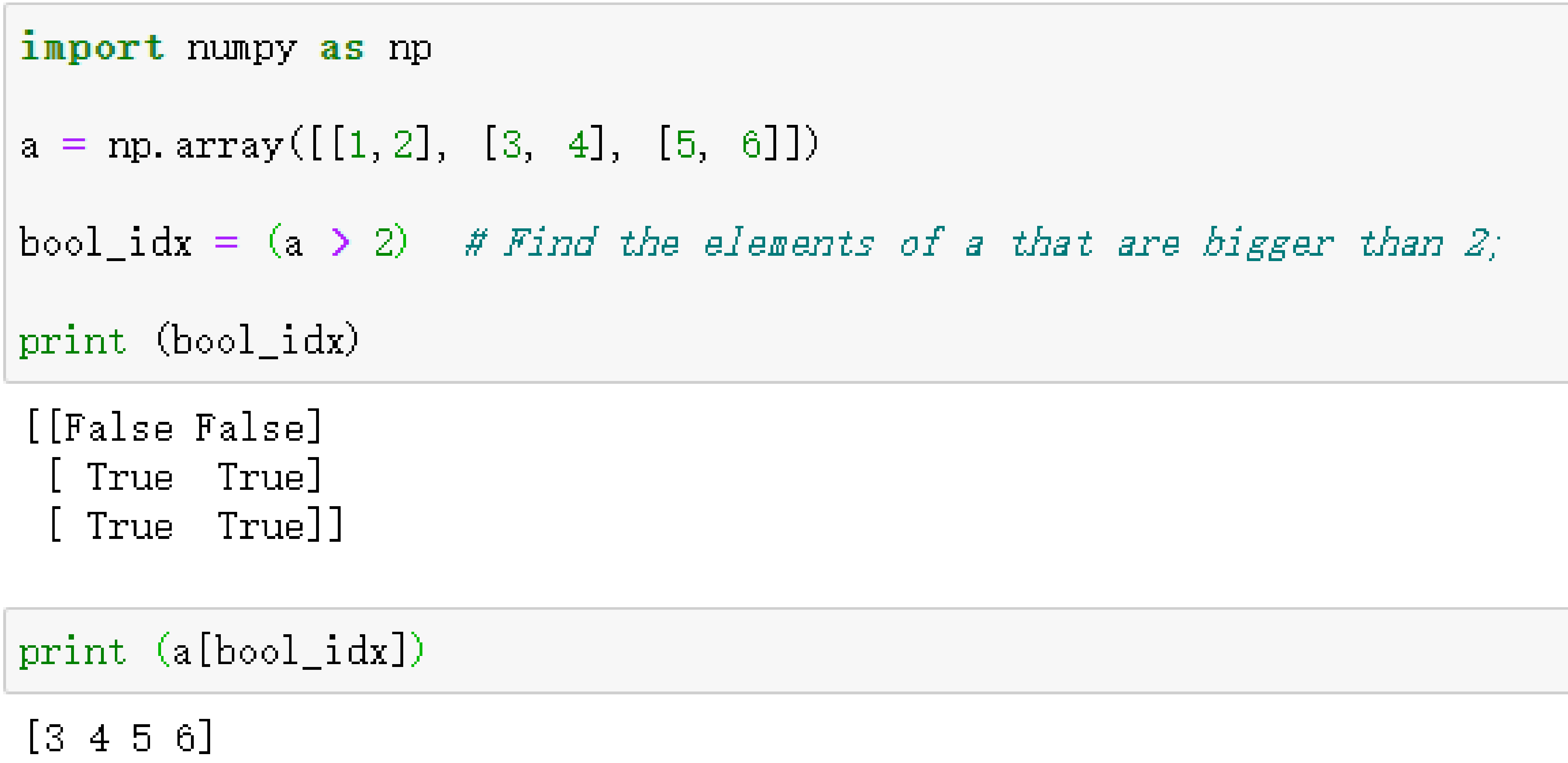
```
[ 1  6  7 11]
```



```
# b:  [0, 2, 0, 1]
a[np.arange(4), b] += 10
print (a)
```

```
[[11  2  3  4]
 [ 4  5 16  7]
 [17  8  9 10]
 [10 21 12 13]]
```

通过条件约束获得布尔下标矩阵



```python
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;

print (bool_idx)
```

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

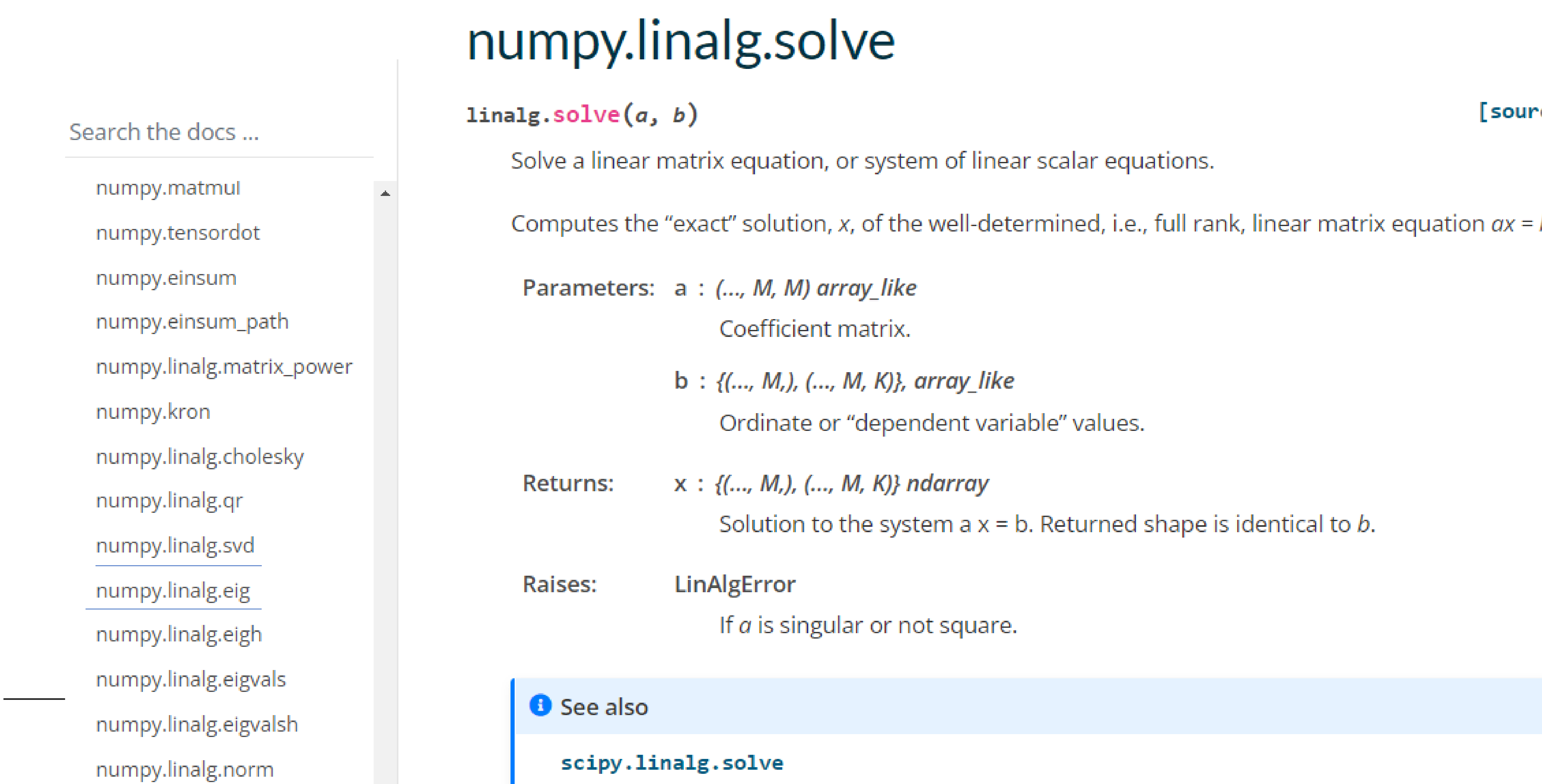
```python
print (a[bool_idx])
```

```
[3 4 5 6]
```

numpy的线性代数运算

- 矩阵乘法
- 矩阵求逆
- 矩阵的特征值与特征向量

numpy的线性代数包提供了矩阵计算功能



Search the docs ...

- numpy.matmul
- numpy.tensordot
- numpy.einsum
- numpy.einsum_path
- numpy.linalg.matrix_power
- numpy.kron
- numpy.linalg.cholesky
- numpy.linalg.qr
- numpy.linalg.svd
- numpy.linalg.eig
- numpy.linalg.eigh
- numpy.linalg.eigvals
- numpy.linalg.eigvalsh
- numpy.linalg.norm

numpy.linalg.solve

`linalg.solve(a, b)` [sour

Solve a linear matrix equation, or system of linear scalar equations.

Computes the “exact” solution, *x*, of the well-determined, i.e., full rank, linear matrix equation *ax* =

Parameters: a : *(..., M, M) array_like*

Coefficient matrix.

b : *{(..., M,), (..., M, K)}, array_like*

Ordinate or “dependent variable” values.

Returns: x : *{(..., M,), (..., M, K)} ndarray*

Solution to the system a x = b. Returned shape is identical to *b*.

Raises: LinAlgError

If *a* is singular or not square.

See also

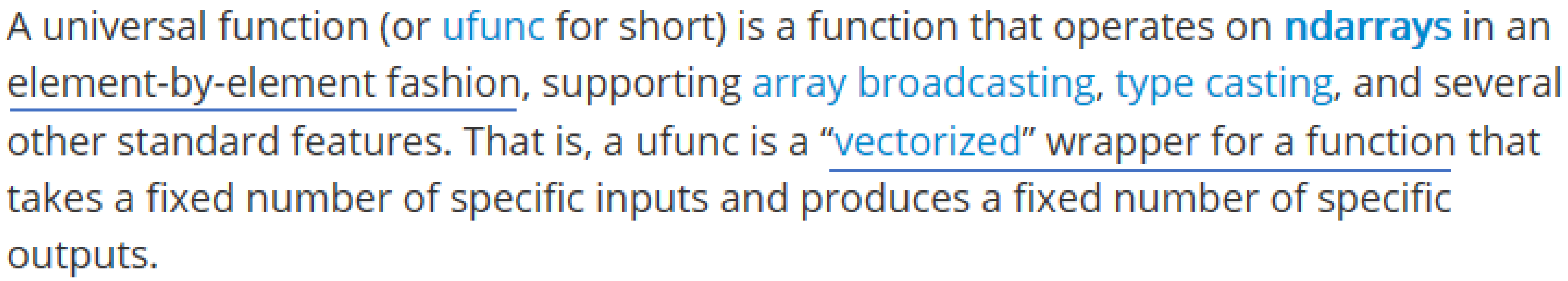
`scipy.linalg.solve`

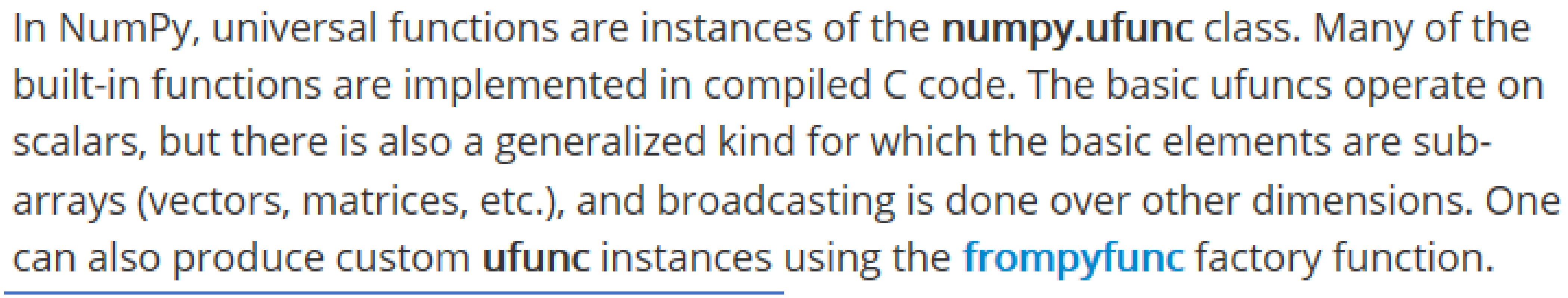
2 计算$Bx = C$的解。

```python
def func1():
    x = np.linalg.inv(B) @ C

def func2():
    x = np.linalg.solve(B, C)



t = timeit('func1()', 'from __main__ import func1', number=100)
print('func1=',t)

t = timeit('func2()', 'from __main__ import func2', number=100)
print('func2=',t)


```

```
func1= 0.8877446270053042
func2= 0.29727534799894784
```

Universal functions (ufunc)：逐元素操作函数


A universal function (or ufunc for short) is a function that operates on **ndarrays** in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features. That is, a ufunc is a “vectorized” wrapper for a function that takes a fixed number of specific inputs and produces a fixed number of specific outputs.


In NumPy, universal functions are instances of the **numpy.ufunc** class. Many of the built-in functions are implemented in compiled C code. The basic ufuncs operate on scalars, but there is also a generalized kind for which the basic elements are sub-arrays (vectors, matrices, etc.), and broadcasting is done over other dimensions. One can also produce custom **ufunc** instances using the **frompyfunc** factory function.

Math operations

add(x1, x2, /[, out, where, casting, order, ...])	Add arguments element-wise.
subtract(x1, x2, /[, out, where, casting, ...])	Subtract arguments, element-wise.
multiply(x1, x2, /[, out, where, casting, ...])	Multiply arguments element-wise.
matmul(x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
divide(x1, x2, /[, out, where, casting, ...])	Returns a true division of the inputs, element-wise.
logaddexp(x1, x2, /[, out, where, casting, ...])	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2, /[, out, where, casting, ...])	Logarithm of the sum of exponentiations of the inputs in base-2.
true_divide(x1, x2, /[, out, where, ...])	Returns a true division of the inputs, element-wise.

numpy.add

```
numpy.add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None,
subok=True[, signature, extobj]) = <ufunc 'add'>
```

Add arguments element-wise.

Parameters: **x1, x2** : *array_like*

The arrays to be added. If `x1.shape != x2.shape`, they must be broadcastable to a common shape (which becomes the shape of the output).

out : *ndarray, None, or tuple of ndarray and None, optional*

A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned. A tuple (possible only as a keyword argument) must have length equal to the number of outputs.

where : *array_like, optional*

This condition is broadcast over the input. At locations where the condition is True, the *out* array will be set to the ufunc result. Elsewhere, the *out* array will retain its original value. Note that if an uninitialized *out* array is created via the default `out=None`, locations within it where the condition is False will remain uninitialized.

****kwargs**

For other keyword-only arguments, see the ufunc docs.

Returns: **add** : *ndarray or scalar*

The sum of *x1* and *x2*, element-wise. This is a scalar if both *x1* and *x2* are scalars.

ufunc使用例子：

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
zz = np.add.outer(x,y)
zz
```

```
array([[5, 6, 7],
       [6, 7, 8],
       [7, 8, 9]])
```

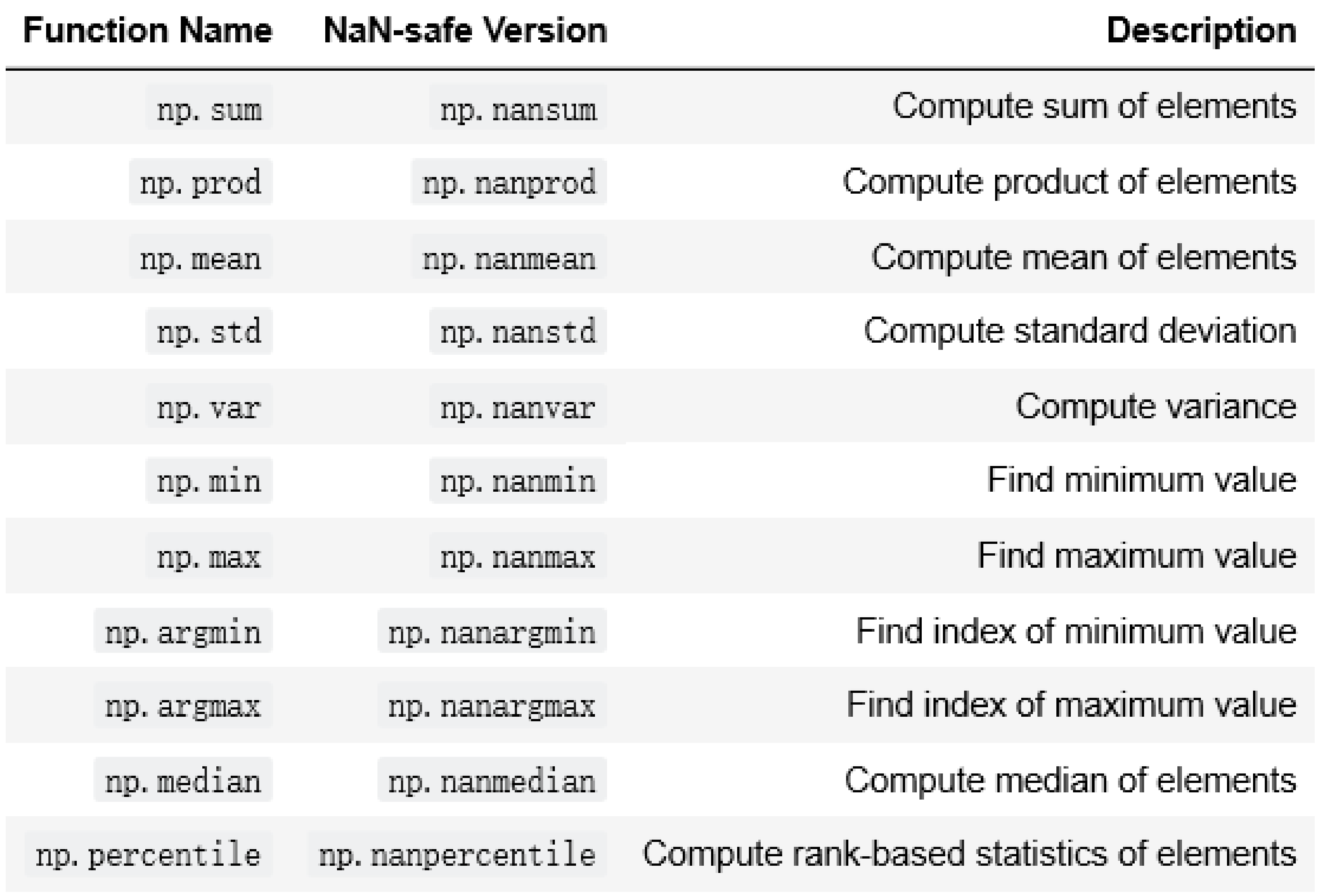
```
zz[1:] += [2,2,2]  # 第二行开始，广播计算
zz
```

```
array([[ 5,  6,  7],
       [ 8,  9, 10],
       [ 9, 10, 11]])
```

```
np.add.at(zz,np.s_[1,[1,2]],[2]) # 生成一个index_exp
zz
```

```
array([[ 5,  6,  7],
       [ 8, 11, 12],
       [ 9, 10, 11]])
```

Aggregation functions（聚合函数 group by）



Function Name	NaN-safe Version	Description
`np.sum`	`np.nansum`	Compute sum of elements
`np.prod`	`np.nanprod`	Compute product of elements
`np.mean`	`np.nanmean`	Compute mean of elements
`np.std`	`np.nanstd`	Compute standard deviation
`np.var`	`np.nanvar`	Compute variance
`np.min`	`np.nanmin`	Find minimum value
`np.max`	`np.nanmax`	Find maximum value
`np.argmin`	`np.nanargmin`	Find index of minimum value
`np.argmax`	`np.nanargmax`	Find index of maximum value
`np.median`	`np.nanmedian`	Compute median of elements
`np.percentile`	`np.nanpercentile`	Compute rank-based statistics of elements

对特定维度进行的聚合类计算（给定轴参数）：

```
a = np.array([i for i in range (24)]).reshape(2,3,4)
a
```

```
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]]])
```

```
a.mean()
```

```
11.5
```

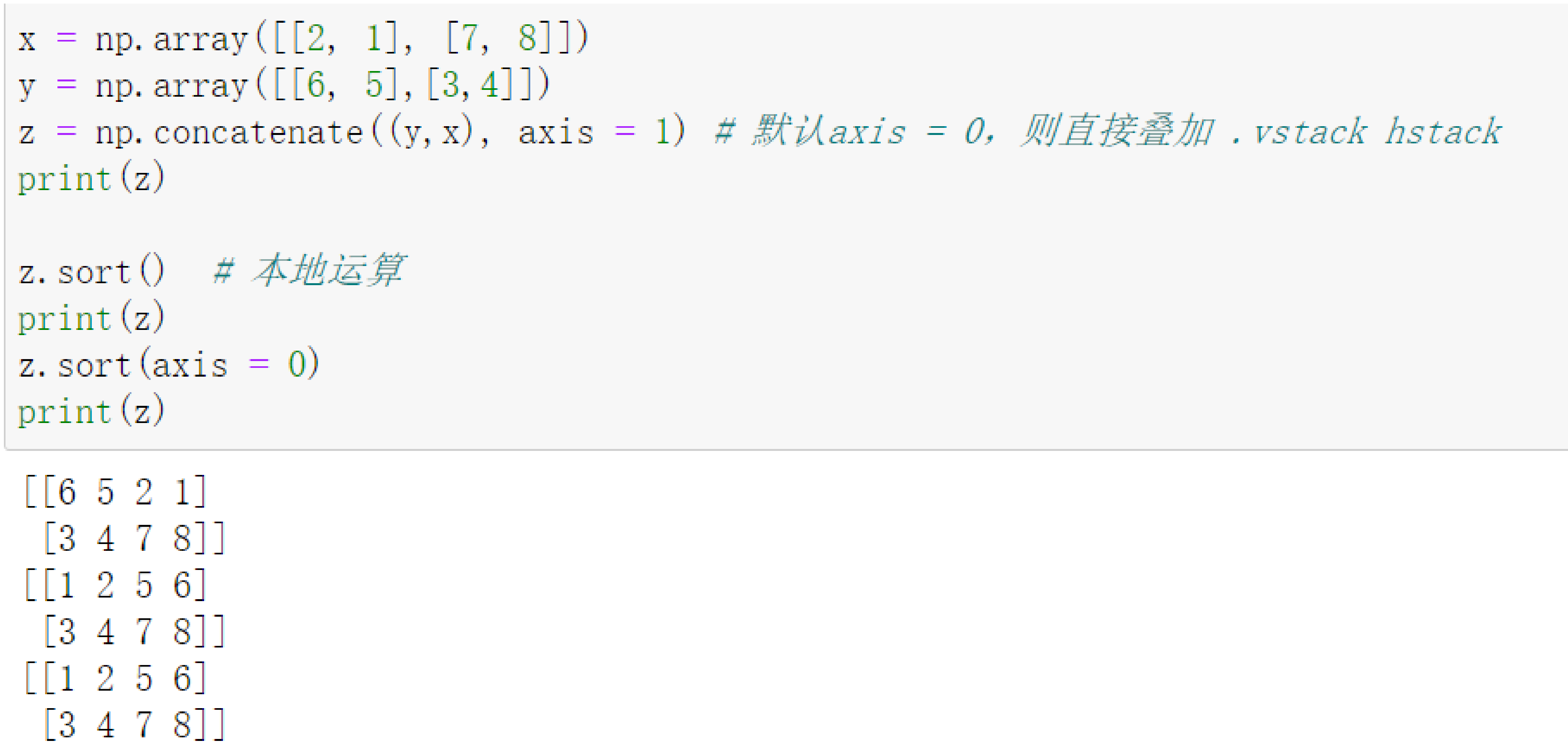
```
a.mean(0)
```

```
array([[ 6.,  7.,  8.,  9.],
       [10., 11., 12., 13.],
       [14., 15., 16., 17.]])
```

```
a.mean((1,2))
```

```
array([ 5.5, 17.5])
```

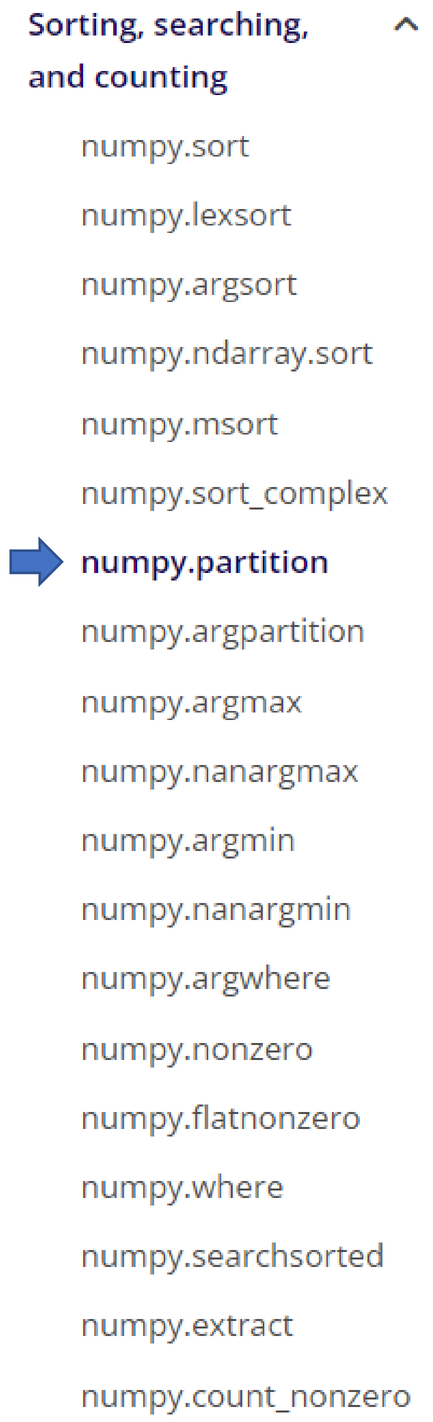
ndarray的sort()方法



```
x = np.array([[2,  1],  [7,  8]])
y = np.array([[6,  5],[3,4]])
z = np.concatenate((y,x),  axis = 1) # 默认axis = 0, 则直接叠加 .vstack hstack
print(z)

z.sort()  # 本地运算
print(z)
z.sort(axis = 0)
print(z)
```

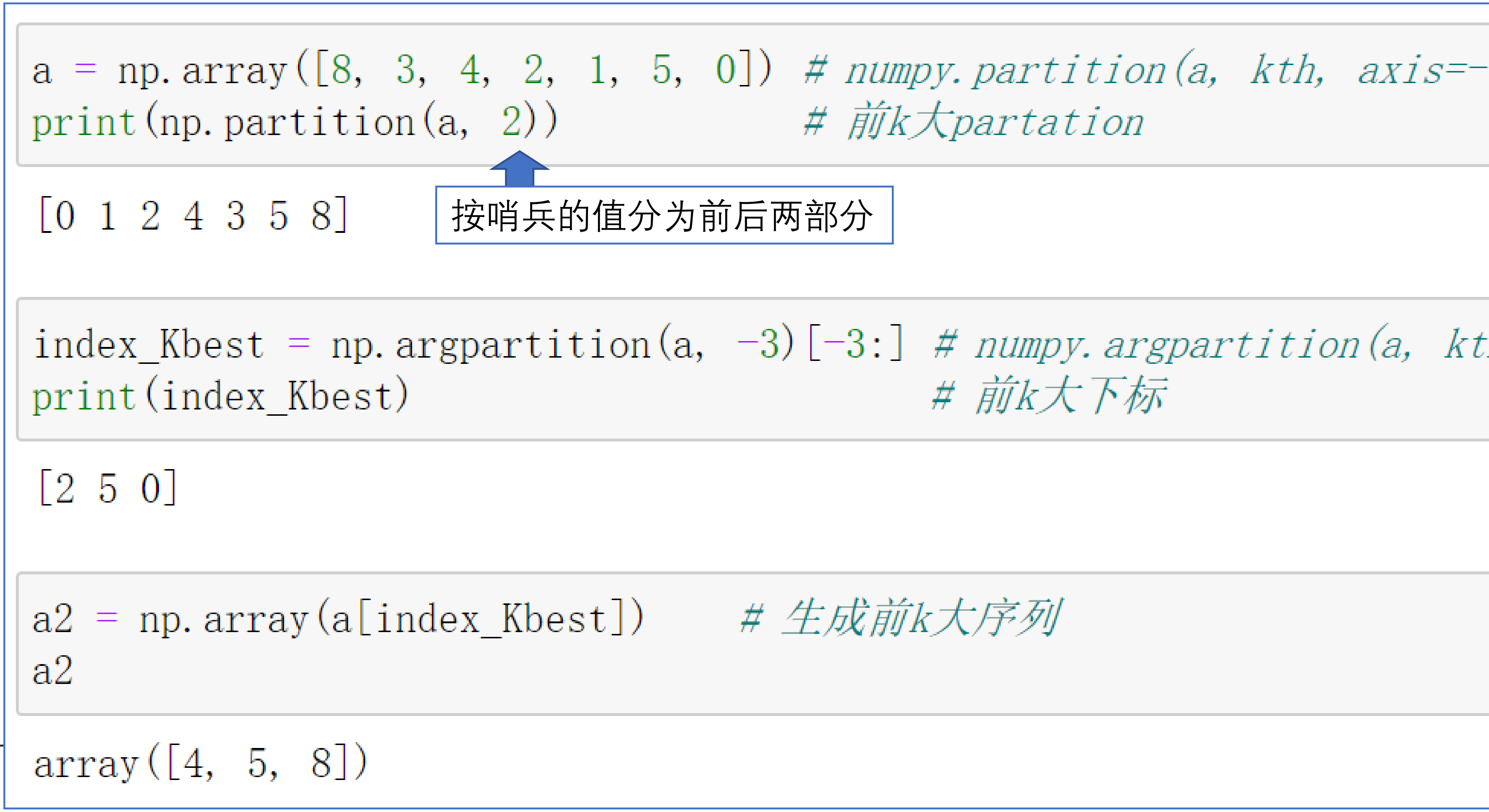
```
[[6 5 2 1]
 [3 4 7 8]]
[[1 2 5 6]
 [3 4 7 8]]
[[1 2 5 6]
 [3 4 7 8]]
```

排序与计数函数:



Sorting, searching, and counting

- numpy.sort
- numpy.lexsort
- numpy.argsort
- numpy.ndarray.sort
- numpy.msort
- numpy.sort_complex
- **numpy.partition**
- numpy.argpartition
- numpy.argmax
- numpy.nanargmax
- numpy.argmin
- numpy.nanargmin
- numpy.argwhere
- numpy.nonzero
- numpy.flatnonzero
- numpy.where
- numpy.searchsorted
- numpy.extract
- numpy.count_nonzero



```
a = np.array([8, 3, 4, 2, 1, 5, 0]) # numpy.partition(a, kth, axis=-
print(np.partition(a, 2))           # 前k大partation
```

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

按哨兵的值分为前后两部分

```
index_Kbest = np.argpartition(a, -3)[-3:] # numpy.argpartition(a, kt
print(index_Kbest)                        # 前k大下标
```

```
[2 5 0]
```

```
a2 = np.array(a[index_Kbest])     # 生成前k大序列
a2
```

```
array([4, 5, 8])
```

- numpy.sort
- numpy.lexsort
- numpy.argsort
- numpy.ndarray.sort
- numpy.msort
- numpy.sort_complex
- numpy.partition
- numpy.argpartition
- **numpy.argmax**
- numpy.nanargmax
- numpy.argmin
- numpy.nanargmin
- numpy.argwhere
- numpy.nonzero
- numpy.flatnonzero
- numpy.where
- numpy.searchsorted
- numpy.extract
- numpy.count_nonzero

Statistics

Test Support (`numpy.testing`)

Window functions

`ndarray.argmax`, `argmin`

`amax`

The maximum value along a given axis.

`unravel_index`

Convert a flat index into an index tuple.

`take_along_axis`

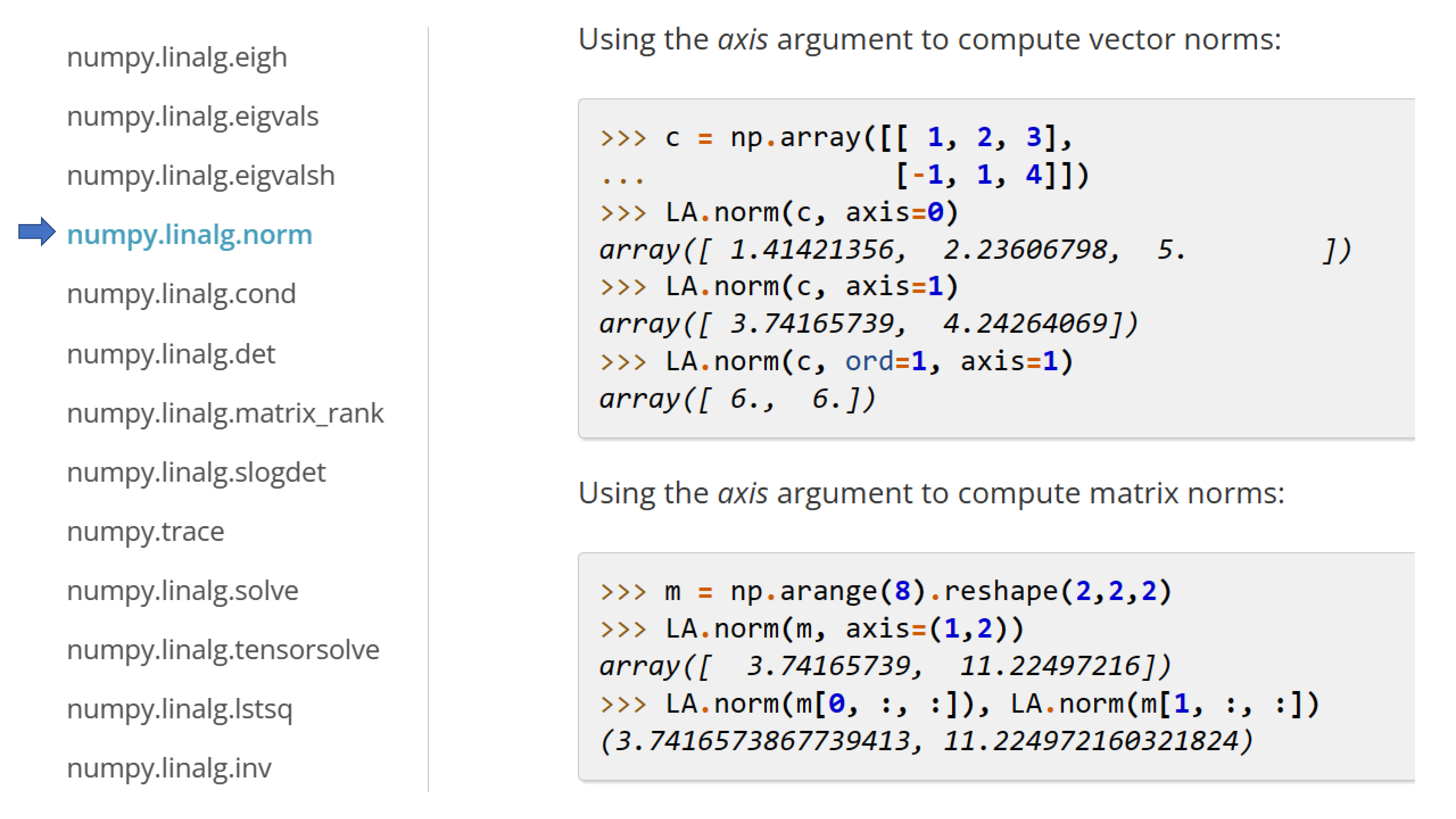
Apply `np.expand_dims(index_array, axis)` from argmax to an array as if by calling max.

Notes

In case of multiple occurrences of the maximum values, the indices corresponding to the first occurrence are returned.

Examples

```
>>> a = np.arange(6).reshape(2,3) + 10
>>> a
array([[10, 11, 12],
       [13, 14, 15]])
>>> np.argmax(a)
5
>>> np.argmax(a, axis=0)
array([1, 1, 1])
>>> np.argmax(a, axis=1)
array([2, 2])
```



numpy.linalg.eigh

numpy.linalg.eigvals

numpy.linalg.eigvalsh

numpy.linalg.norm

numpy.linalg.cond

numpy.linalg.det

numpy.linalg.matrix_rank

numpy.linalg.slogdet

numpy.trace

numpy.linalg.solve

numpy.linalg.tensorsolve

numpy.linalg.lstsq

numpy.linalg.inv

Using the *axis* argument to compute vector norms:

```
>>> c = np.array([[ 1, 2, 3],
...               [-1, 1, 4]])
>>> LA.norm(c, axis=0)
array([ 1.41421356,  2.23606798,  5.        ])
>>> LA.norm(c, axis=1)
array([ 3.74165739,  4.24264069])
>>> LA.norm(c, ord=1, axis=1)
array([ 6.,  6.])
```

Using the *axis* argument to compute matrix norms:

```
>>> m = np.arange(8).reshape(2,2,2)
>>> LA.norm(m, axis=(1,2))
array([  3.74165739,  11.22497216])
>>> LA.norm(m[0, :, :]), LA.norm(m[1, :, :])
(3.7416573867739413, 11.224972160321824)
```

k-means聚类算法简介

- 输入:类的数目k，包含n个文本的特征向量。
- 输出: k个类，使平方误差准则最小。
- 步骤:

  1)任意选择k个对象作为初始的类中心;

  2) repeat;

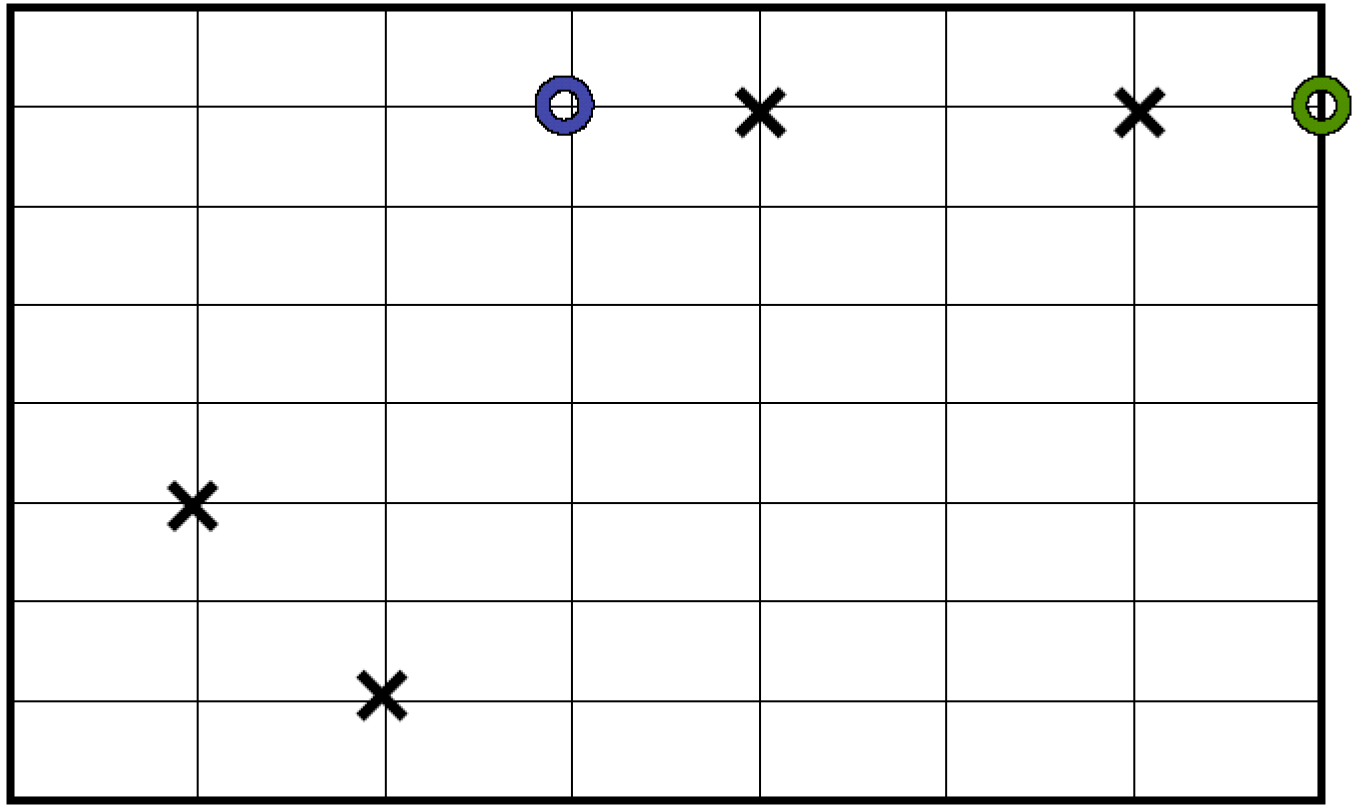
  3)根据类中对象的平均值，将每个对象(重新)赋给最类似的类;

- 4)更新类的平均值;
- 5) until不再发生变化或残差小于阈值

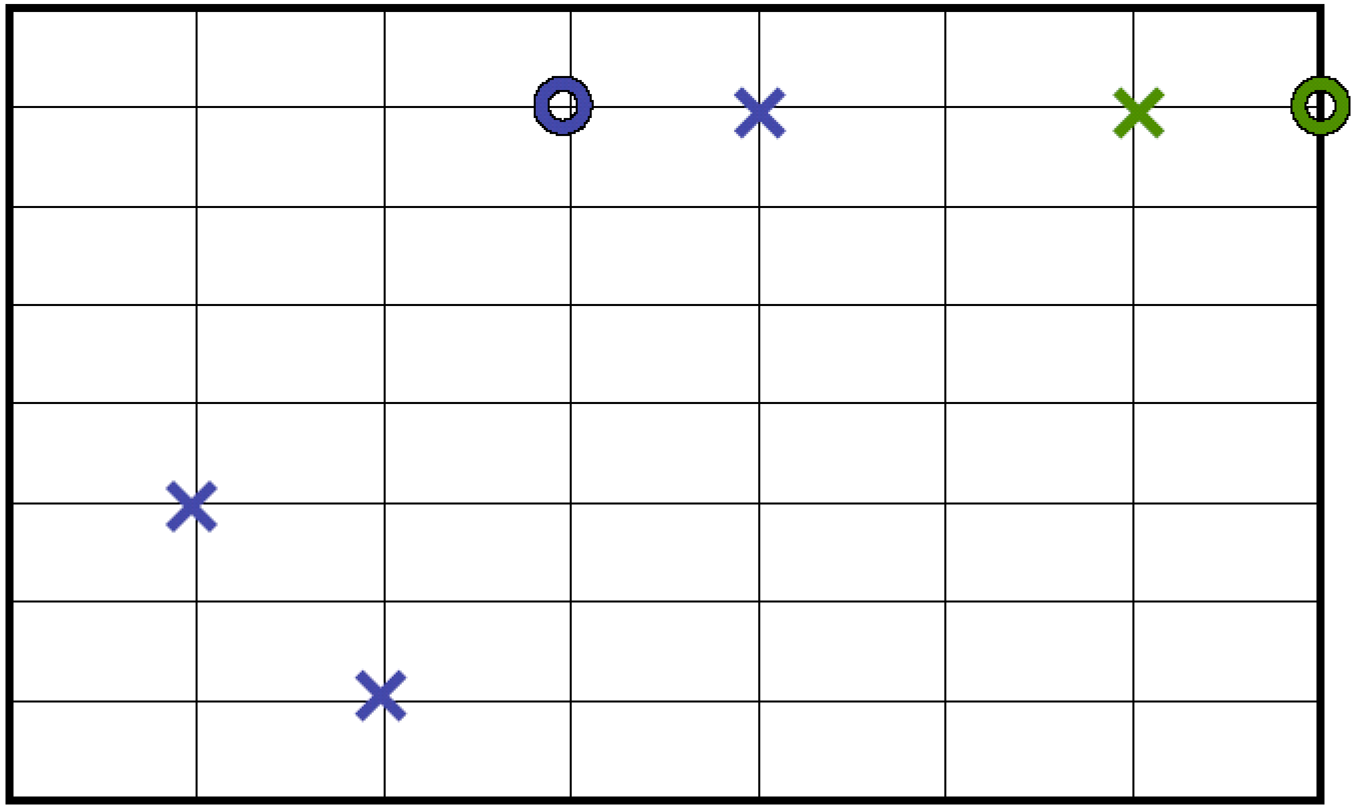
K-均值举例

- 初始化n个数据向量
  - $x_1=(2,1)$ $x_2=(1,3)$ $x_3=(6,7)$ $x_4=(4,7)$
- 随机初始化k个类中心
  - $f_1=(4,3)$ $f_2=(5,5)$
- 计算与质心的距离 $d(x_i,x_j)=\sqrt{\sum_{k=1}^{n}(x_{i_k}-x_{j_k})^2}$
- 对每个类计算均值 $\mu(x_1,\ldots,x_n)=(\frac{\sum_{i=1}^{n}x_{i_1}}{n},\ldots,\frac{\sum_{i=1}^{n}x_{i_m}}{n})$

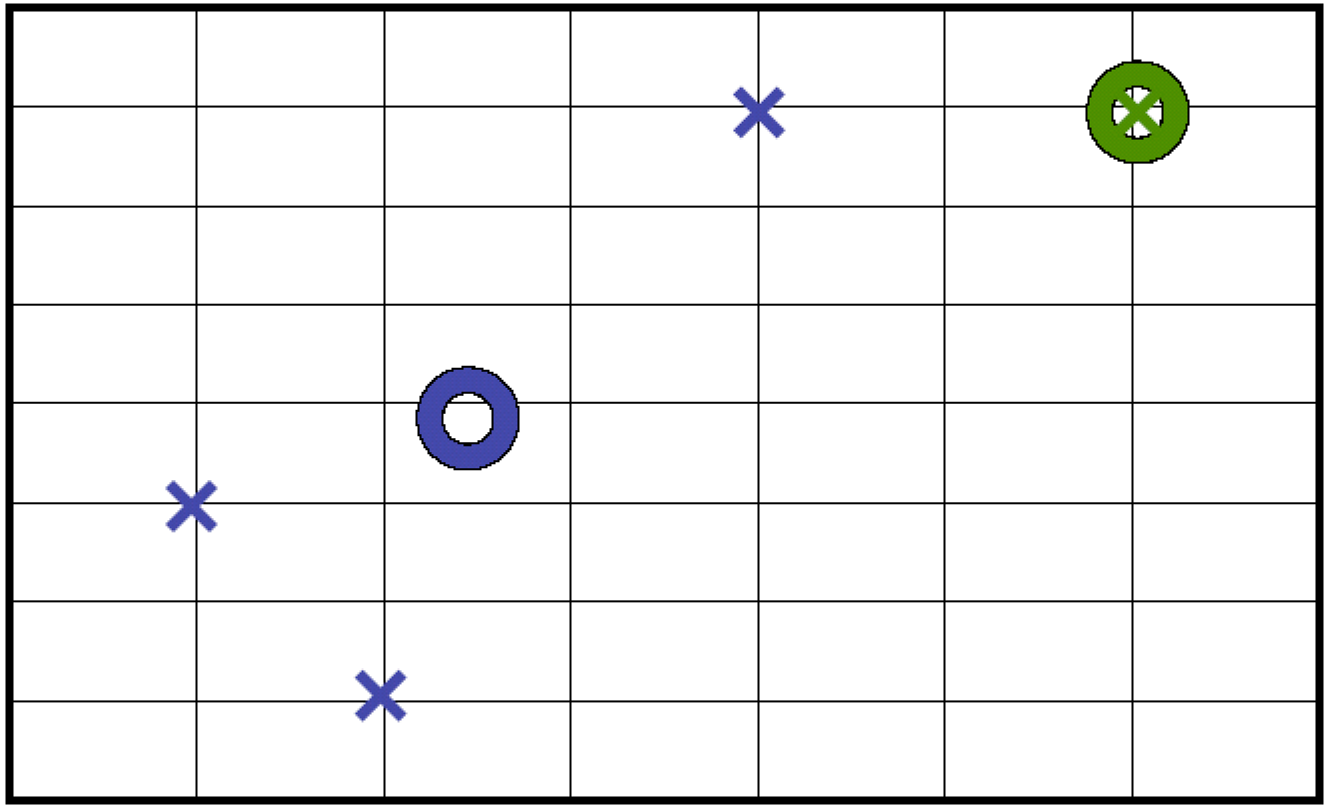
随机两个kernel



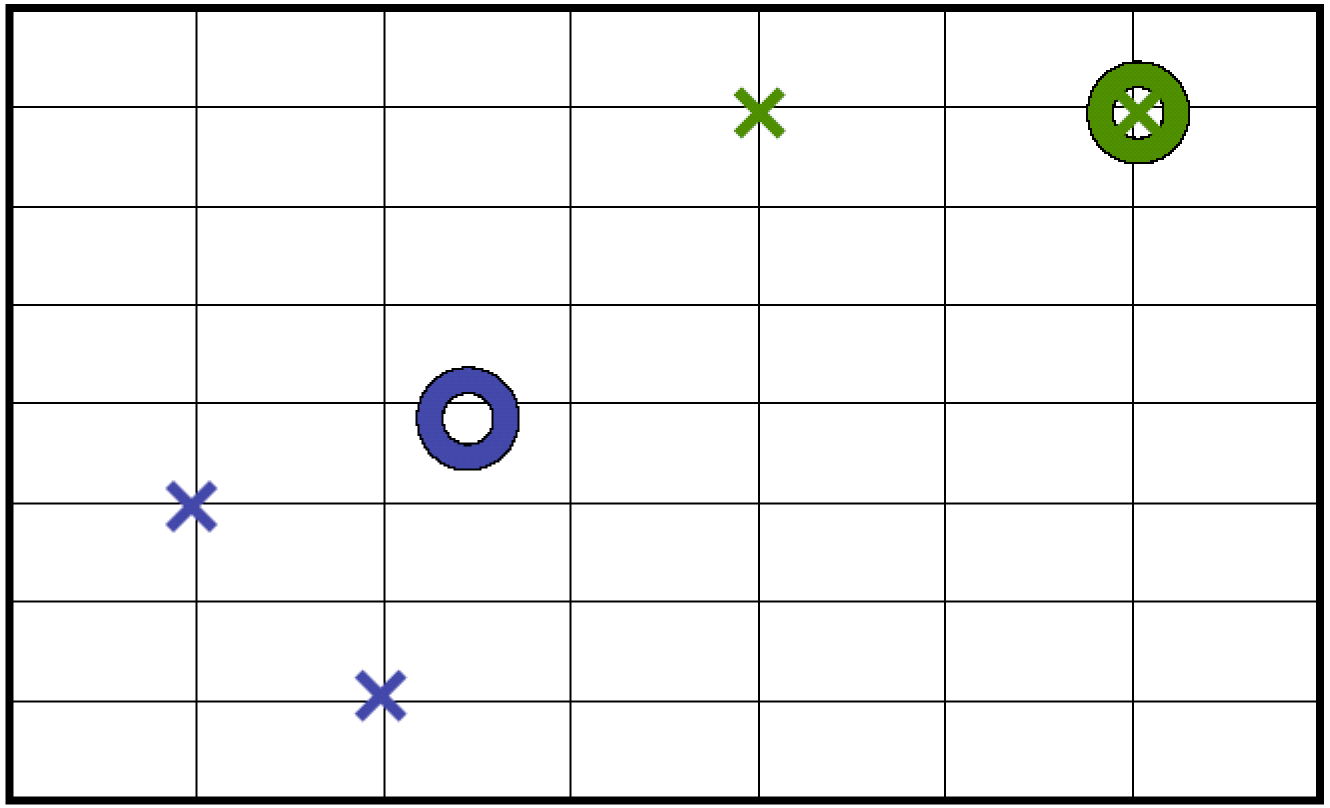
完成按距离的划分



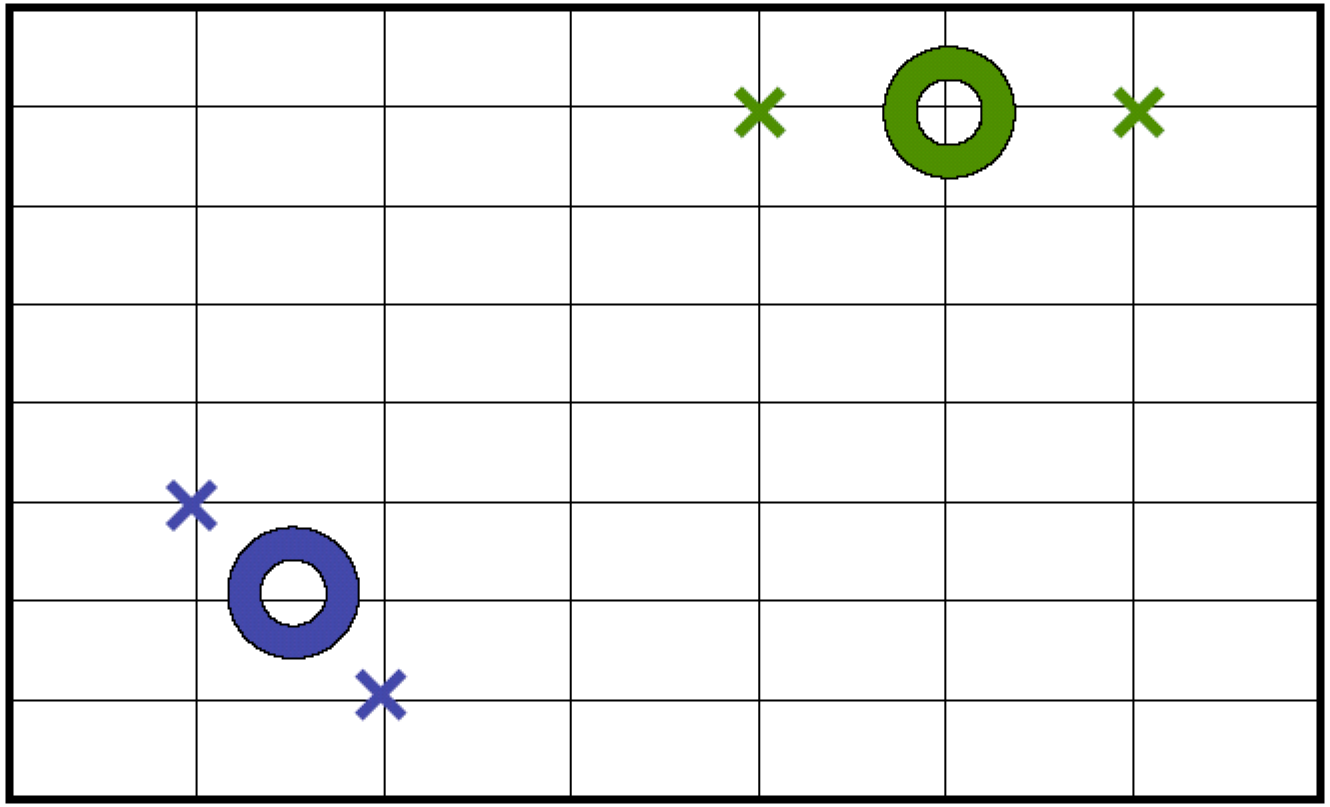
根据划分计算出每个类新的kernel



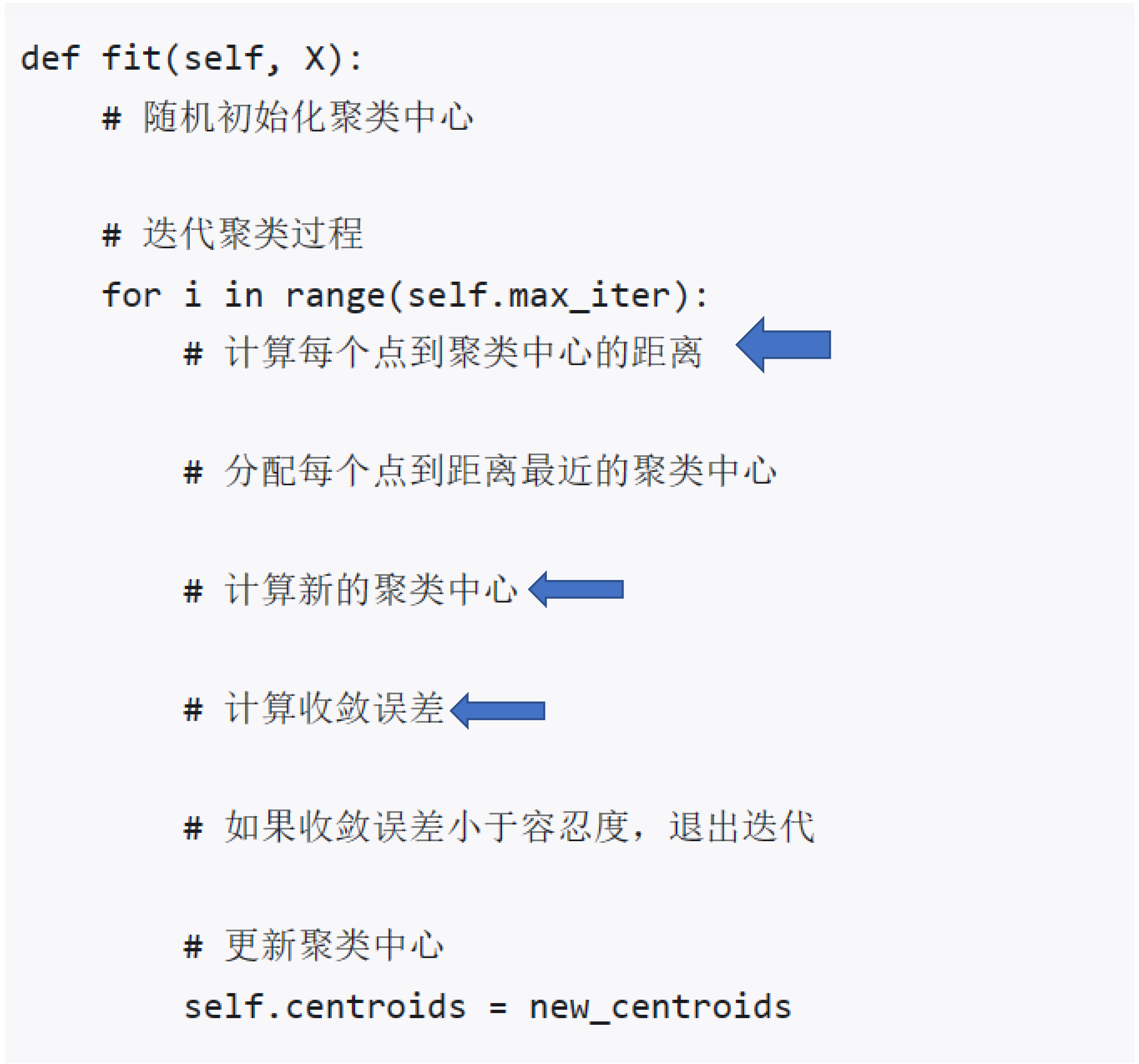
按新的kernel重新对样本进行划分



迭代到收斂

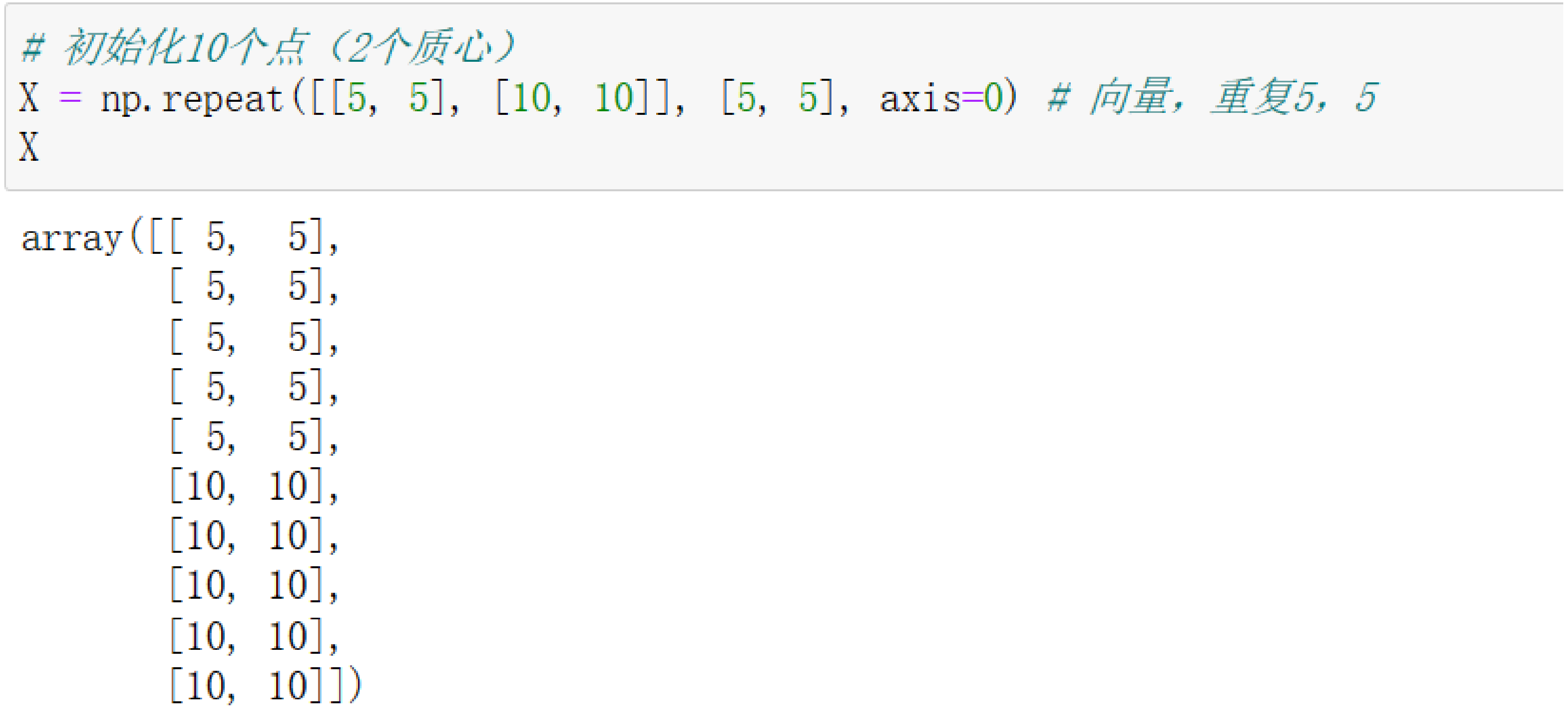


算法复杂度分析:



```
def fit(self, X):
    # 随机初始化聚类中心

    # 迭代聚类过程
    for i in range(self.max_iter):
        # 计算每个点到聚类中心的距离

        # 分配每个点到距离最近的聚类中心

        # 计算新的聚类中心

        # 计算收敛误差

        # 如果收敛误差小于容忍度，退出迭代

        # 更新聚类中心
        self.centroids = new_centroids
```

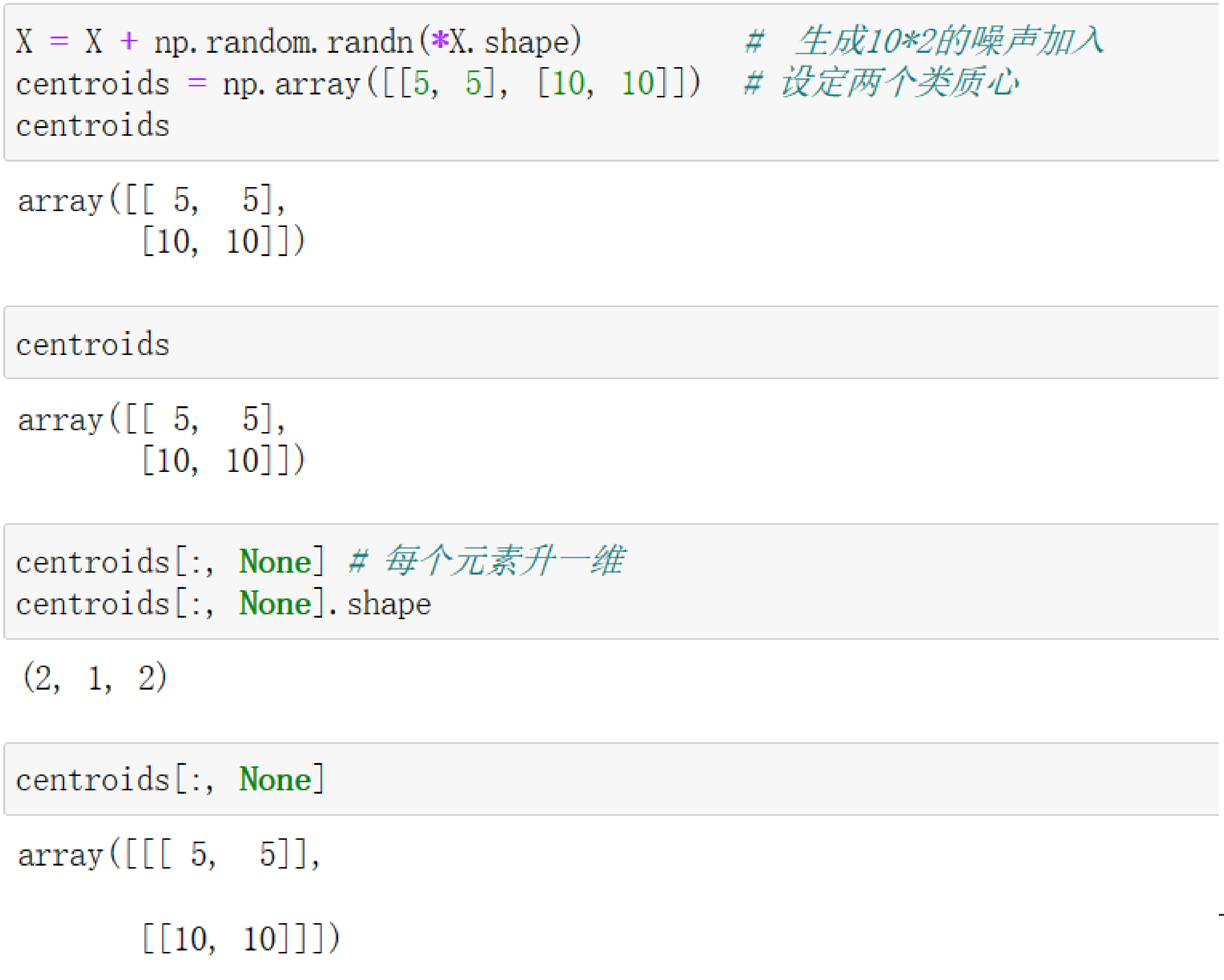
Numpy算法实现（step1）



```
# 初始化10个点（2个质心）
X = np.repeat([[5, 5], [10, 10]], [5, 5], axis=0) # 向量，重复5，5
X
```

```
array([[ 5,  5],
       [ 5,  5],
       [ 5,  5],
       [ 5,  5],
       [ 5,  5],
       [10, 10],
       [10, 10],
       [10, 10],
       [10, 10],
       [10, 10]])
```

Step2：



```
X = X + np.random.randn(*X.shape)          #  生成10*2的噪声加入
centroids = np.array([[5, 5], [10, 10]])   # 设定两个类质心
centroids
```

```
array([[ 5,  5],
       [10, 10]])
```

```
centroids
```

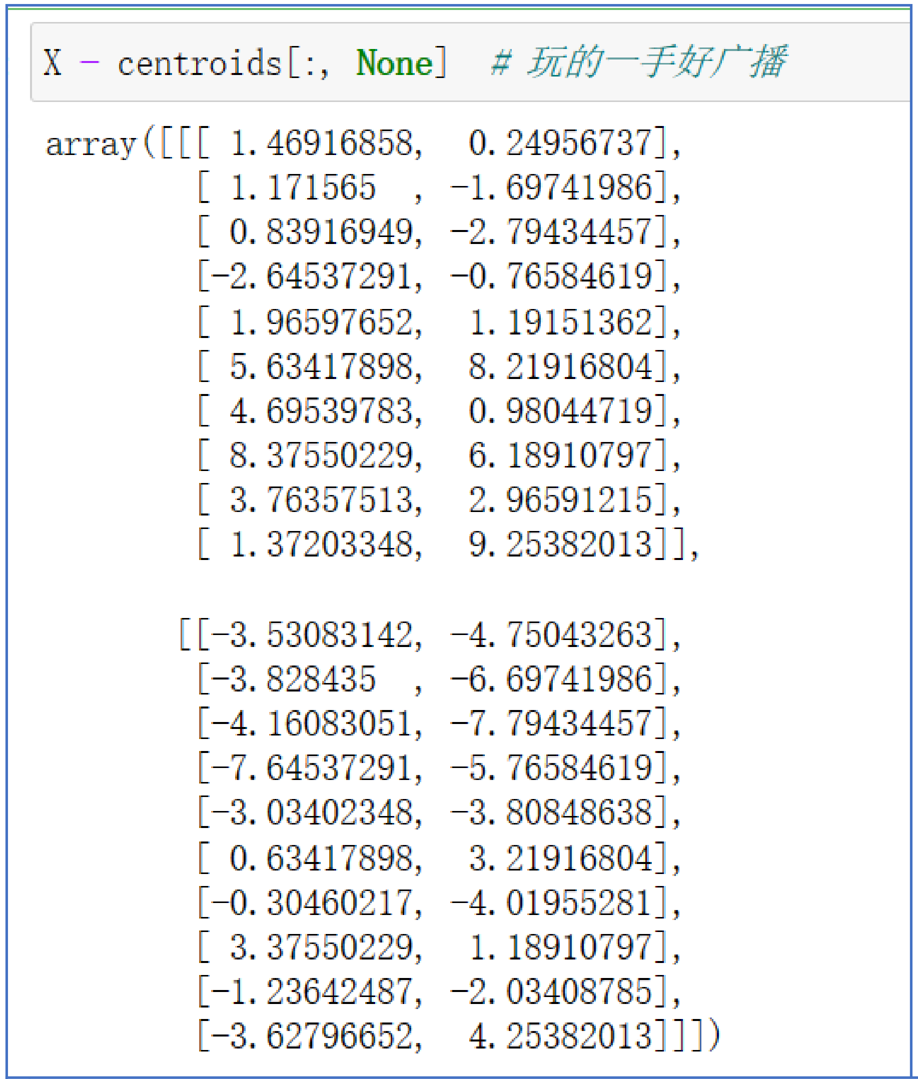
```
array([[ 5,  5],
       [10, 10]])
```

```
centroids[:, None] # 每个元素升一维
centroids[:, None].shape
```

```
(2, 1, 2)
```

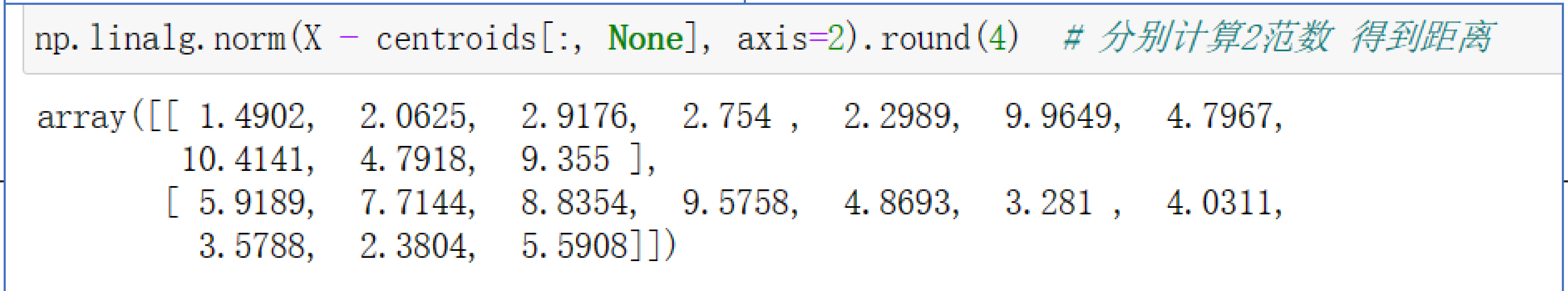
```
centroids[:, None]
```

```
array([[[ 5,  5]],

       [[10, 10]]])
```



```
X - centroids[:, None]  # 玩的一手好广播
```

```
array([[[ 1.46916858,  0.24956737],
        [ 1.171565  , -1.69741986],
        [ 0.83916949, -2.79434457],
        [-2.64537291, -0.76584619],
        [ 1.96597652,  1.19151362],
        [ 5.63417898,  8.21916804],
        [ 4.69539783,  0.98044719],
        [ 8.37550229,  6.18910797],
        [ 3.76357513,  2.96591215],
        [ 1.37203348,  9.25382013]],

       [[-3.53083142, -4.75043263],
        [-3.828435  , -6.69741986],
        [-4.16083051, -7.79434457],
        [-7.64537291, -5.76584619],
        [-3.03402348, -3.80848638],
        [ 0.63417898,  3.21916804],
        [-0.30460217, -4.01955281],
        [ 3.37550229,  1.18910797],
        [-1.23642487, -2.03408785],
        [-3.62796652,  4.25382013]]])
```



```
np.linalg.norm(X - centroids[:, None], axis=2).round(4)  # 分别计算2范数 得到距离
```

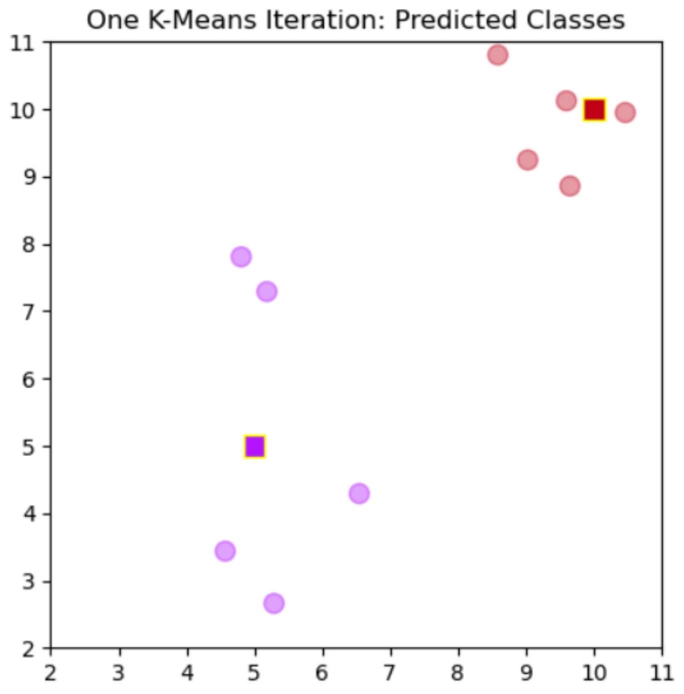
```
array([[ 1.4902,  2.0625,  2.9176,  2.754 ,  2.2989,  9.9649,  4.7967,
        10.4141,  4.7918,  9.355 ],
       [ 5.9189,  7.7144,  8.8354,  9.5758,  4.8693,  3.281 ,  4.0311,
         3.5788,  2.3804,  5.5908]])
```

```
np.argmin(np.linalg.norm(X - centroids[:, None], axis=2), axis=0)
```

```
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1], dtype=int64)
```

```
def get_labels(X, centroids) -> np.ndarray:
    return np.argmin(np.linalg.norm(X - centroids[:, None], axis=2),axis=0)

labels = get_labels(X, centroids)
```

```
c1, c2 = ['#b213fe', '#be0119']  # rgb十六进制配色

llim, ulim  = np.trunc([X.min() * 0.9, X.max() * 1.1])#设定边界

_, ax = plt.subplots(figsize=(5, 5))
ax.scatter(*X.T, c=np.where(labels, c2, c1), alpha=0.4, s=80)
ax.scatter(*centroids.T, c=[c1, c2], marker='s', s=95,edgecolor='yellow')
ax.set_ylim([llim, ulim])
ax.set_xlim([llim, ulim])
ax.set_title('One K-Means Iteration: Predicted Classes')
```



```
import timeit

setup = '''
import numpy as np
'''


snippet = 'arr = np.arange(10000)'

num_runs = 10000

time_elapsed = timeit.timeit(setup = setup, stmt = snippet, number = num_runs)

print("Time Elapsed: ", time_elapsed / num_runs)
```

```
Time Elapsed:  4.706920031458139e-06
```

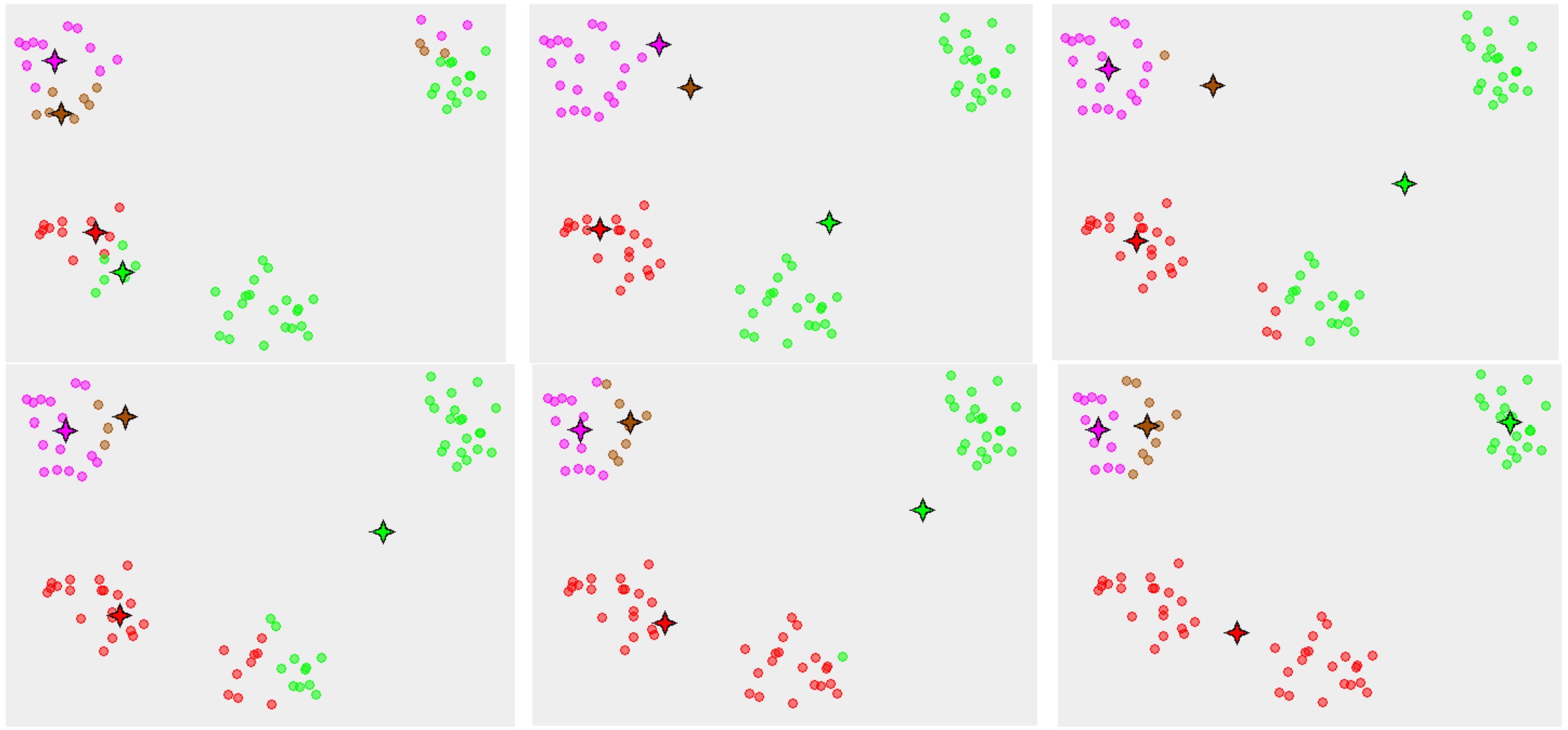
```
def fn():
    return np.arange(10000)

num_runs = 10000

time_elapsed = timeit.timeit(setup = setup, stmt = fn, number = num_runs)

print("Time Elapsed: ", time_elapsed / num_runs)
```

```
Time Elapsed:  5.026069981977343e-06
```

K-均值算法分析

- 算法复杂度为O (kln)， 其中l为迭代次数， n为文档个数， k为类别个数
- K-均值算法最后一定是可以收敛的
- 该算法本质上是一种贪心算法。
    - 可以保证局部最优，但是很难保证全局最优。
- 需要预先指定k值和初始划分

K-means convergence to a local minimum



K-means的物理意义:

- VQ下平方和误差最小
- 能否采用其他的误差方案?

聚类的质量评价

- 指标：纯度（Purity）和F值（F-measure）
- 标准答案：一般是人工分好类的文档集合

纯度

- 对于聚类后形成的任意类别r，聚类的纯度定义为

$$P(S_r) = \frac{1}{n_r}\max(n_r^i)$$

- 整个聚类结果的纯度定义为

$$\text{Purity} = \sum_{r=1}^{k}\frac{n_r}{n}P(S_r)$$

- n_r^i：属于预定义类i且被分配到第r个聚类的文档个数
- n_r：第r个聚类类别中的文档个数

F值

- F值：准确率（precision）和召回率（recall）的调和平均数
- $\mathrm{precision(i,r)} = n_r^i / n_r$
- $\mathrm{recall(i,r)} = n_r^i / n_i$
- n_r^i：属于预定义类i且被分配到第r个聚类的文档个数
- n_r：第r个聚类类别中的文档个数
- n_i：预定义类别i中的文档个数

F值

- 聚类r和类别i之间的f值计算如下：
- $\mathrm{f(i,r)} = \frac{2 \times recall(i,r) \times precision(i,r)}{precision(i,r) + recall(i,r)}$

- 最终聚类结果的评价函数为
- $\mathrm{F} = \sum_i \frac{n_i}{n} \max\{\mathrm{f(i,r)}\}$，$n$是所有文档的个数

K-means 的优化改进

- 确定K
  - 对于不同的K都尝试聚类，取效果最好的
  - Bisectional K-means

- 初始种子选定
  - 排除明显是“噪声”的文档向量
  - 尝试多种初始向量的组合，取效果最好的
  - 通过其他方法（如层次聚类）确定初始文档向量