# 实验1：矩阵库numpy的使用（二）

**什么是numpy？**

NumPy是一个功能强大的Python库，主要用于对多维数组执行计算。NumPy这个词来源于两个单词-- Numerical和Python。NumPy提供了大量的库函数和操作，可以帮助程序员轻松地进行数值计算。这类数值计算广泛用于以下任务：

机器学习模型：在编写机器学习算法时，需要对矩阵进行各种数值计算。例如矩阵乘法、换位、加法等。NumPy提供了一个非常好的库，用于简单(在编写代码方面)和快速(在速度方面)计算。NumPy数组用于存储训练数据和机器学习模型的参数。

图像处理和计算机图形学：计算机中的图像表示为多维数字数组。NumPy成为同样情况下最自然的选择。实际上，NumPy提供了一些优秀的库函数来快速处理图像。例如，镜像图像、按特定角度旋转图像等。

## 1、NumPy 迭代数组

NumPy 迭代器对象 numpy.nditer 提供了一种灵活访问一个或者多个数组元素的方式。

迭代器最基本的任务的可以完成对数组元素的访问。

**1.1使用nditer迭代**

import numpy as np

a = np.arange(6).reshape(2,3)

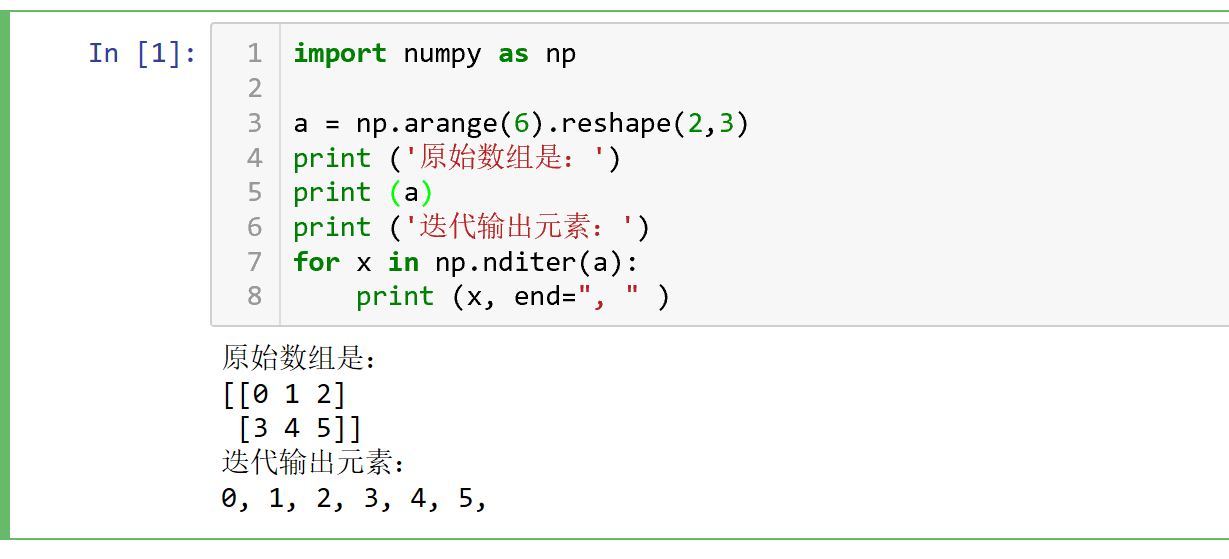
print ('原始数组是：')

print (a)

print ('迭代输出元素：')

for x in np.nditer(a):

print (x, end=", " )



以上实例不是使用标准 C 或者 Fortran 顺序，选择的顺序是和数组内存布局一致的，这样做是为了提升访问的效率，**默认是行序优先**（row-major order，或者说是 C-order）。

这反映了默认情况下只需访问每个元素，而无需考虑其特定顺序。我们可以通过迭代上述数组的转置来看到这一点，并与以 C 顺序访问数组转置的 copy 方式做对比。

*import numpy as np*

*a = np.arange(6).reshape(2,3)*

*for x in np.nditer(a.T):*

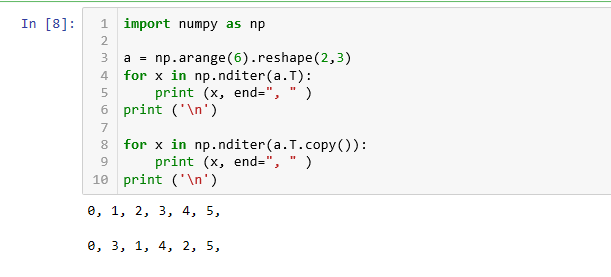
*print (x, end=", " )*

*print ('\n')*

*for x in np.nditer(a.T.copy()):*

*print (x, end=", " )*

*print ('\n')*



**1.2 控制遍历顺序**

for x in np.nditer(a, order='F'):Fortran order，即是列序优先；

for x in np.nditer(a.T, order='C'):C order，即是行序优先；

*import numpy as np*

*a = np.arange(0,60,5)*

*a = a.reshape(3,4)*

*print ('原始数组是：')*

*print (a)*

*print ('\n')*

*print ('以 C 风格顺序排序：')*

*for x in np.nditer(a, order = 'C'):*

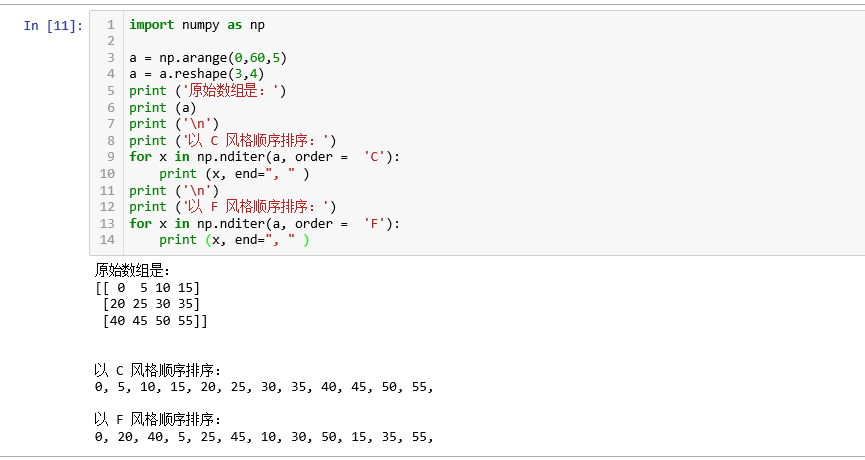
*print (x, end=", " )*

*print ('\n')*

*print ('以 F 风格顺序排序：')*

*for x in np.nditer(a, order = 'F'):*

*print (x, end=", " )*



**1.3 修改数组中元素的值**

import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)

print ('原始数组是：')

print (a)

print ('\n')

for x in np.nditer(a, op\_flags=['readwrite']):

x[...]=2\*x

print ('修改后的数组是：')

print (a)



**1.4 使用外部循环**

nditer类的构造器拥有flags参数，它可以接受下列值：



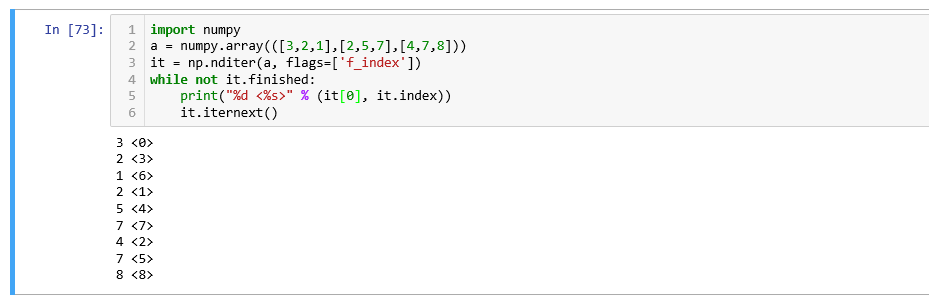
import numpy

a = numpy.array(([3,2,1],[2,5,7],[4,7,8]))

it = np.nditer(a, flags=['f\_index'])

while not it.finished:

print("%d <%s>" % (it[0], it.index))

it.iternext()

import numpy

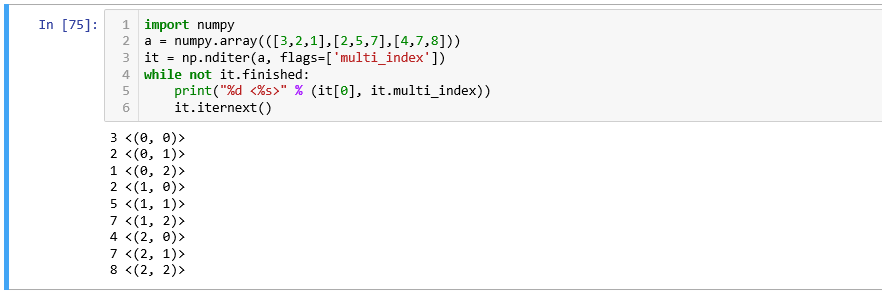
a = numpy.array(([3,2,1],[2,5,7],[4,7,8]))

it = np.nditer(a, flags=['multi\_index'])

while not it.finished:

print("%d <%s>" % (it[0], it.multi\_index))

it.iternext()



**1.4 使用外部循环**

如果两个数组是可广播的，nditer 组合对象能够同时迭代它们。 假设数组 a 的维度为 3X4，数组 b 的维度为 1X4 ，则使用以下迭代器（数组 b 被广播到 a 的大小）。

*import numpy as np*

*a = np.arange(0,60,5)*

*a = a.reshape(3,4)*

*print ('第一个数组为：')*

*print (a)*

*print ('\n')*

*print ('第二个数组为：')*

*b = np.array([1, 2, 3, 4], dtype = int)*

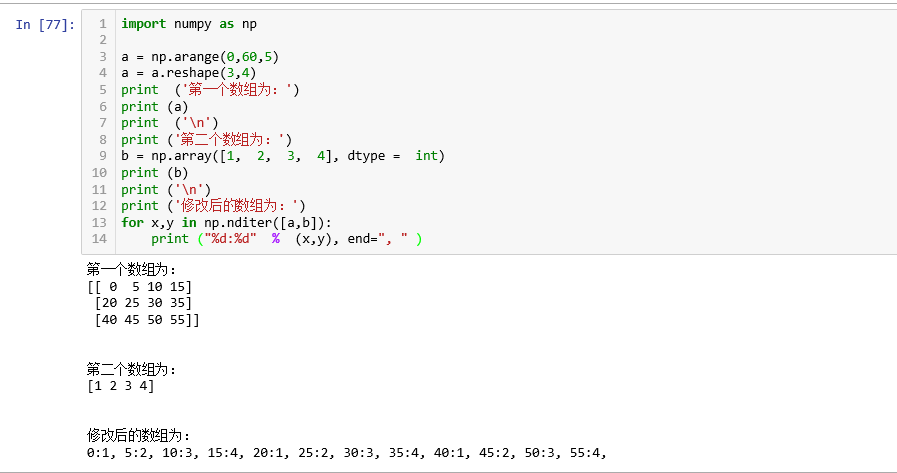
*print (b)*

*print ('\n')*

*print ('修改后的数组为：')*

*for x,y in np.nditer([a,b]):*

*print ("%d:%d" % (x,y), end=", " )*



## 2、NumPy 数组操作

Numpy 中包含了一些函数用于处理数组，大概可分为以下几类：

* 修改数组形状
* 翻转数组
* 修改数组维度
* 连接数组
* 分割数组
* 数组元素的添加与删除

**2.1 修改数组形状**



numpy.reshape 函数可以在不改变数据的条件下修改形状，格式如下： numpy.reshape(arr, newshape, order='C')

* arr：要修改形状的数组
* newshape：整数或者整数数组，新的形状应当兼容原有形状
* order：'C' -- 按行，'F' -- 按列，'A' -- 原顺序，'k' -- 元素在内存中的出现顺序。



numpy.ndarray.flat

numpy.ndarray.flat 是一个数组元素迭代器



numpy.ndarray.flatten

numpy.ndarray.flatten 返回一份数组拷贝，对拷贝所做的修改不会影响原始数组

*import numpy as np*

*a = np.arange(8).reshape(2,4)*

*print ('原数组：')*

*print (a)*

*print ('\n')*

*# 默认按行*

*print ('展开的数组：')*

*print (a.flatten())*

*print ('\n')*

*print ('以 F 风格顺序展开的数组：')*

*print (a.flatten(order = 'F'))*



numpy.ravel

numpy.ravel() 展平的数组元素，顺序通常是"C风格"，返回的是数组视图（view，有点类似 C/C++引用reference的意味），修改会影响原始数组。

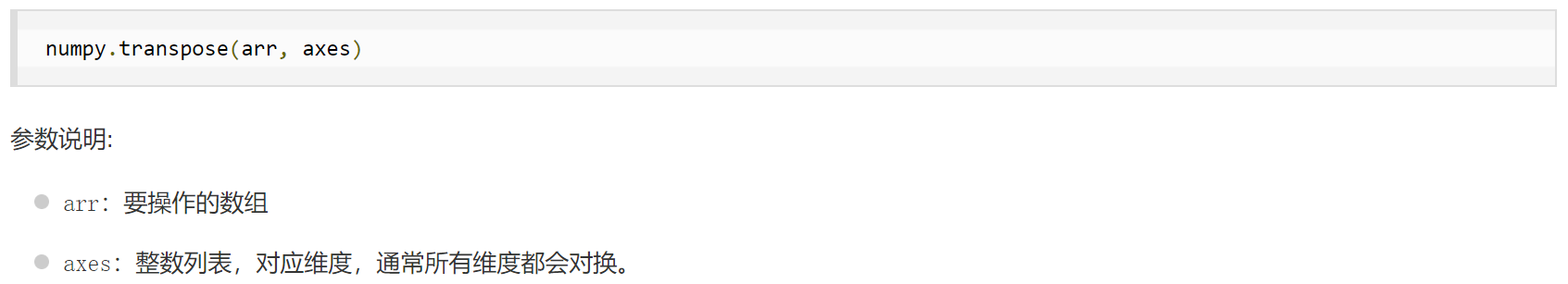
该函数接收两个参数：

numpy.ravel(a, order='C')

**2.2 翻转数组**



numpy.transpose



*import numpy as np*

*a = np.arange(24).reshape(2,3,4)*

*print ('原数组：')*

*print (a )*

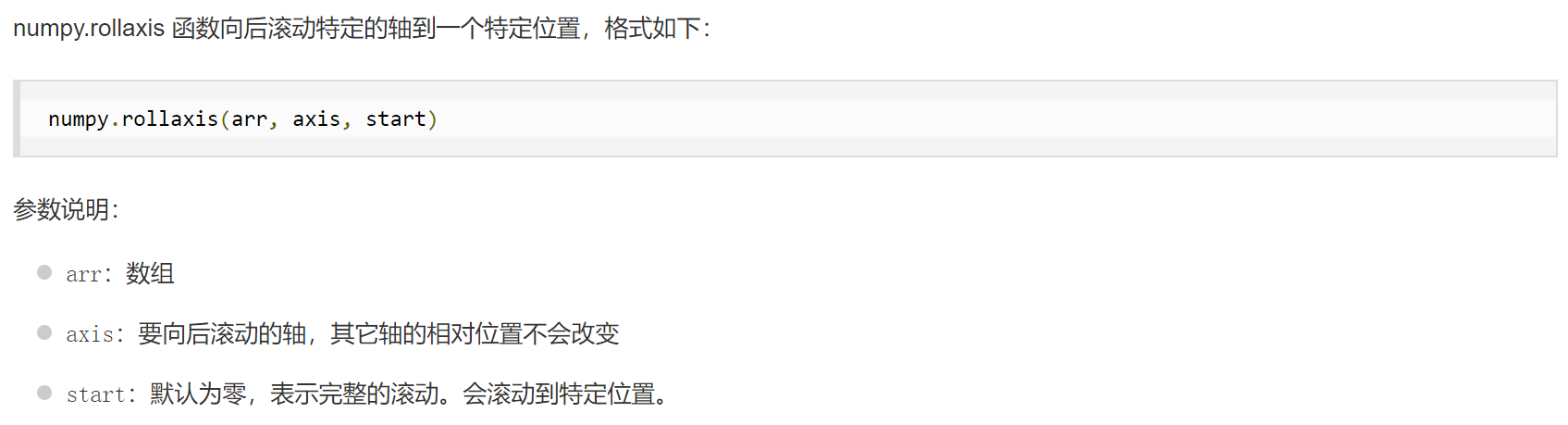
*print ('\n')*

*print ('对换数组：')*

*print (np.transpose(a,[0,2,1]))*



numpy.rollaxis



*import numpy as np*

*a = np.arange(6).reshape(1,2,3)*

*print ('原数组：')*

*print (a)*

*print ('\n')*

*# 将轴 2 滚动到轴 0（宽度到深度）*

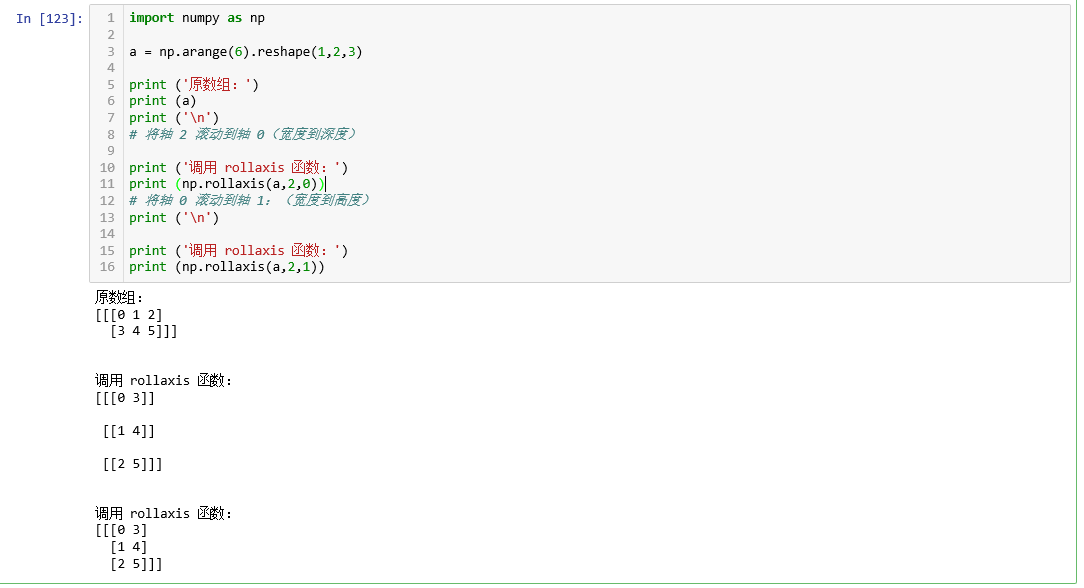
*print ('调用 rollaxis 函数：')*

*print (np.rollaxis(a,2,0))*

*# 将轴 0 滚动到轴 1：（宽度到高度）*

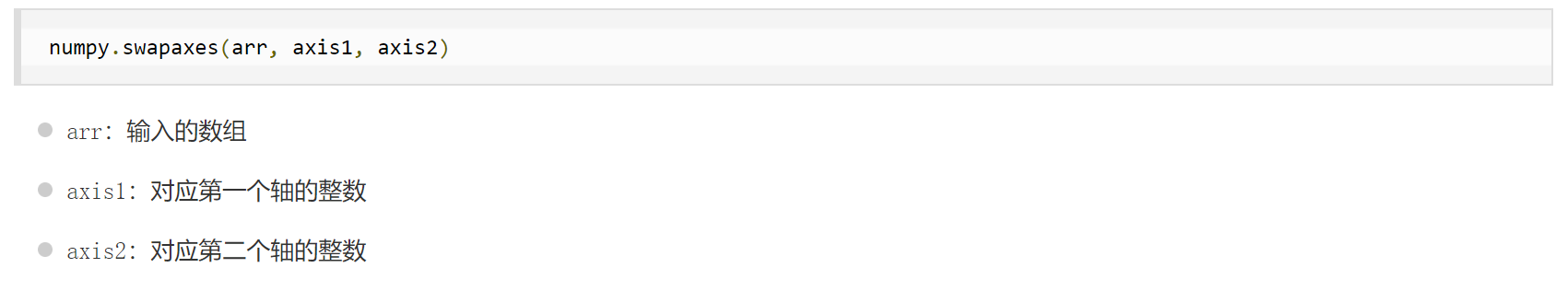
*print ('\n')*

*print ('调用 rollaxis 函数：')*

*print (np.rollaxis(a,2,1))*

numpy.swapaxes

numpy.swapaxes 函数用于交换数组的两个轴，格式如下：



可以参考<https://www.jb51.net/article/138340.htm>

import numpy as np

# 创建了三维的 ndarray

a = np.arange(8).reshape(2,2,2)

print ('原数组：')

print (a)

print ('\n')

# 现在交换轴 0（深度方向）到轴 2（宽度方向）

print ('调用 swapaxes 函数后的数组：')

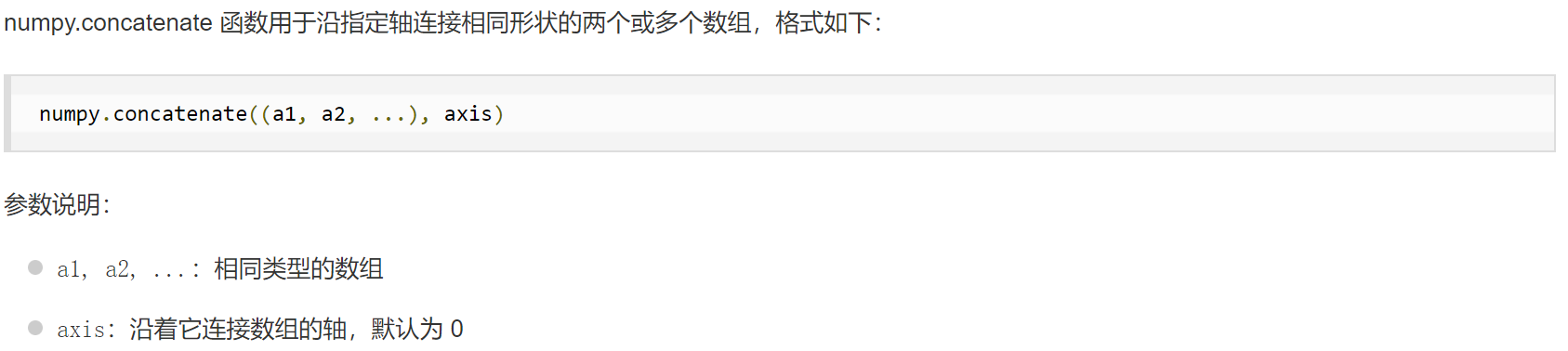
print (np.swapaxes(a, 2, 0))



**2.3 连接数组**



numpy.concatenate



*import numpy as np*

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

*print ('第一个数组：')*

*print (a)*

*print ('\n')*

*b = np.array([[5,6],[7,8]])*

*print ('第二个数组：')*

*print (b)*

*print ('\n')*

*# 两个数组的维度相同*

*print ('沿轴 0 连接两个数组：')*

*print (np.concatenate((a,b)))*

*print ('\n')*

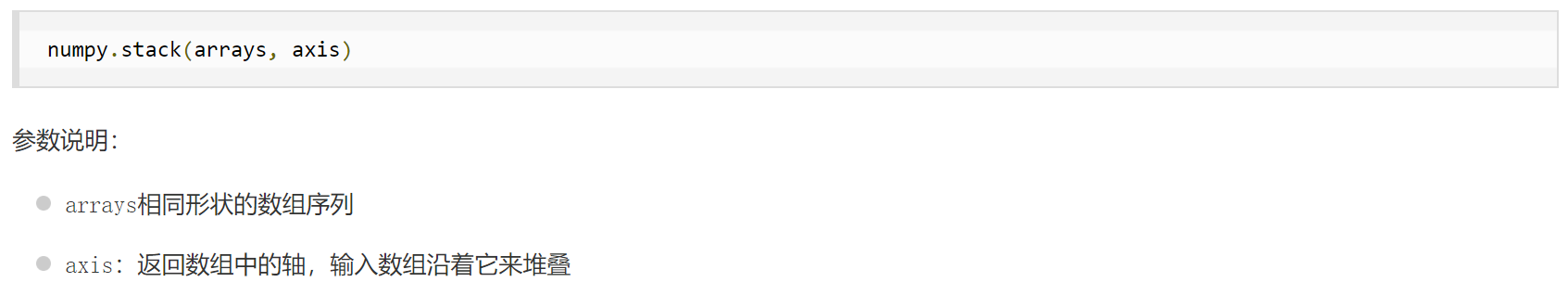
*print ('沿轴 1 连接两个数组：')*

*print (np.concatenate((a,b),axis = 1))*



numpy.stack

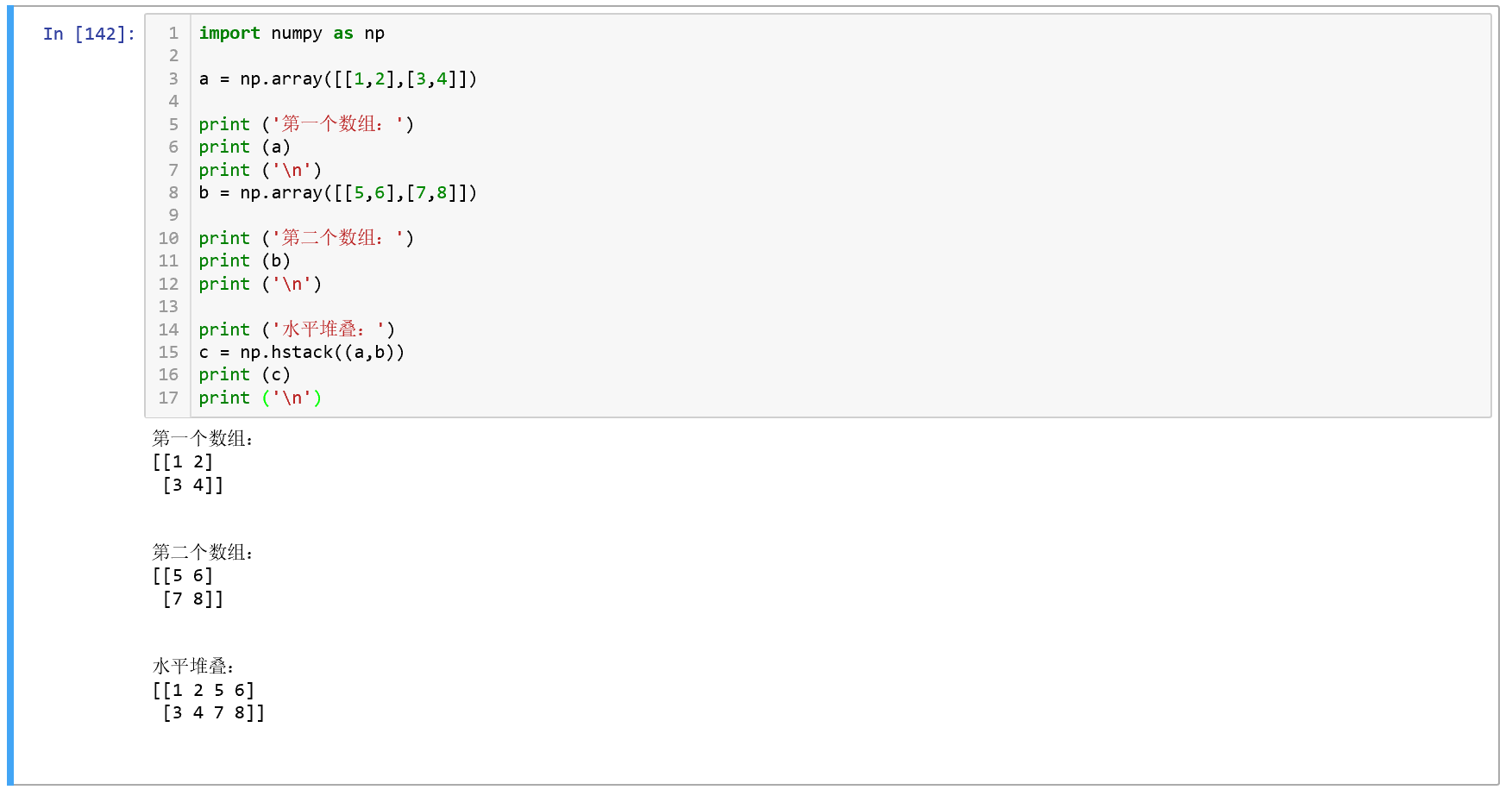
numpy.stack 函数用于沿新轴连接数组序列，格式如下：





numpy.hstack

numpy.hstack 是 numpy.stack 函数的变体，它通过水平堆叠来生成数组。



numpy.vstack

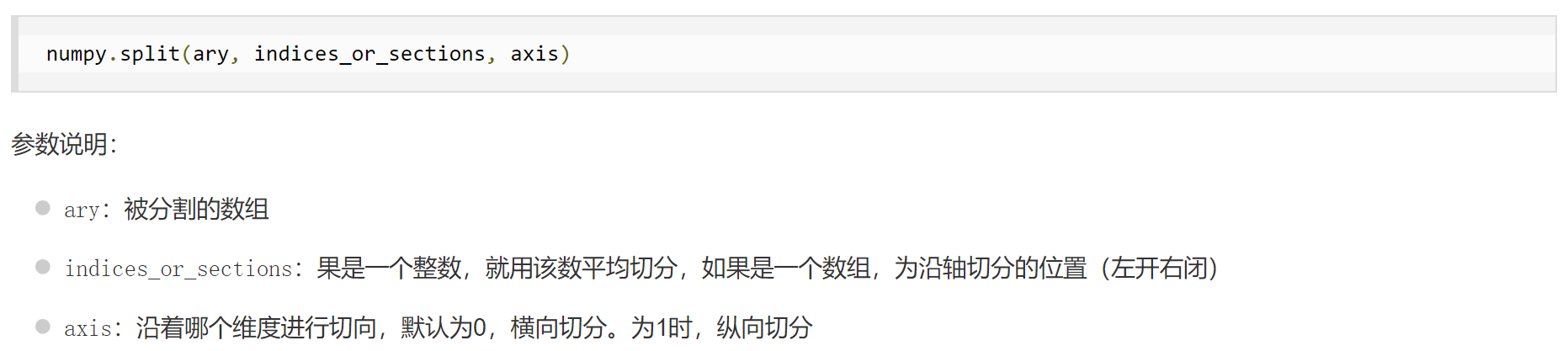
numpy.vstack 是 numpy.stack 函数的变体，它通过垂直堆叠来生成数组。

**2.4 分割数组**



numpy.split

numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：



*import numpy as np*

*a = np.arange(9)*

*print ('第一个数组：')*

*print (a)*

*print ('\n')*

*print ('将数组分为三个大小相等的子数组：')*

*b = np.split(a,3)*

*print (b)*

*print ('\n')*

*print ('将数组在一维数组中表明的位置分割：')*

*b = np.split(a,[4,7])*

*print (b)*



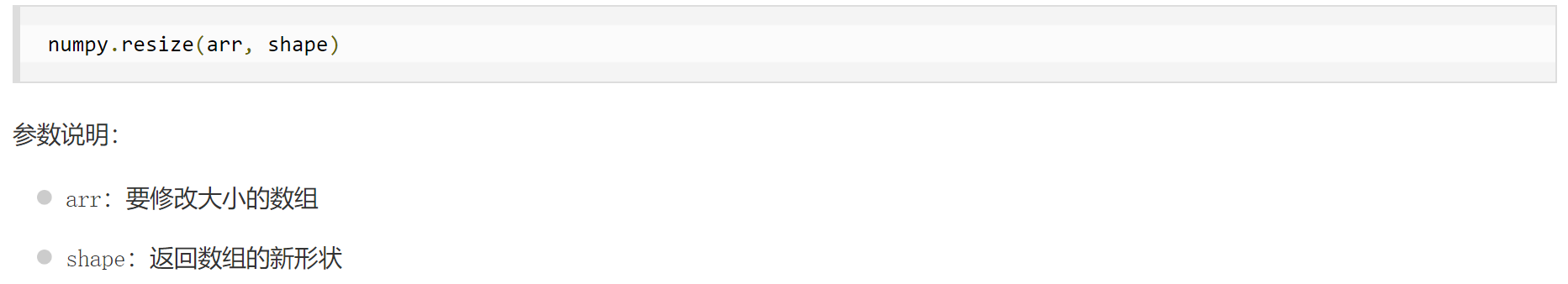
**2.5 数组元素的添加与删除**



numpy.resize

numpy.resize 函数返回指定大小的新数组。

如果新数组大小大于原始大小，则包含原始数组中的元素的副本。



*import numpy as np*

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

*print ('第一个数组：')*

*print (a)*

*print ('\n')*

*print ('第一个数组的形状：')*

*print (a.shape)*

*print ('\n')*

*b = np.resize(a, (3,2))*

*print ('第二个数组：')*

*print (b)*

*print ('\n')*

*print ('第二个数组的形状：')*

*print (b.shape)*

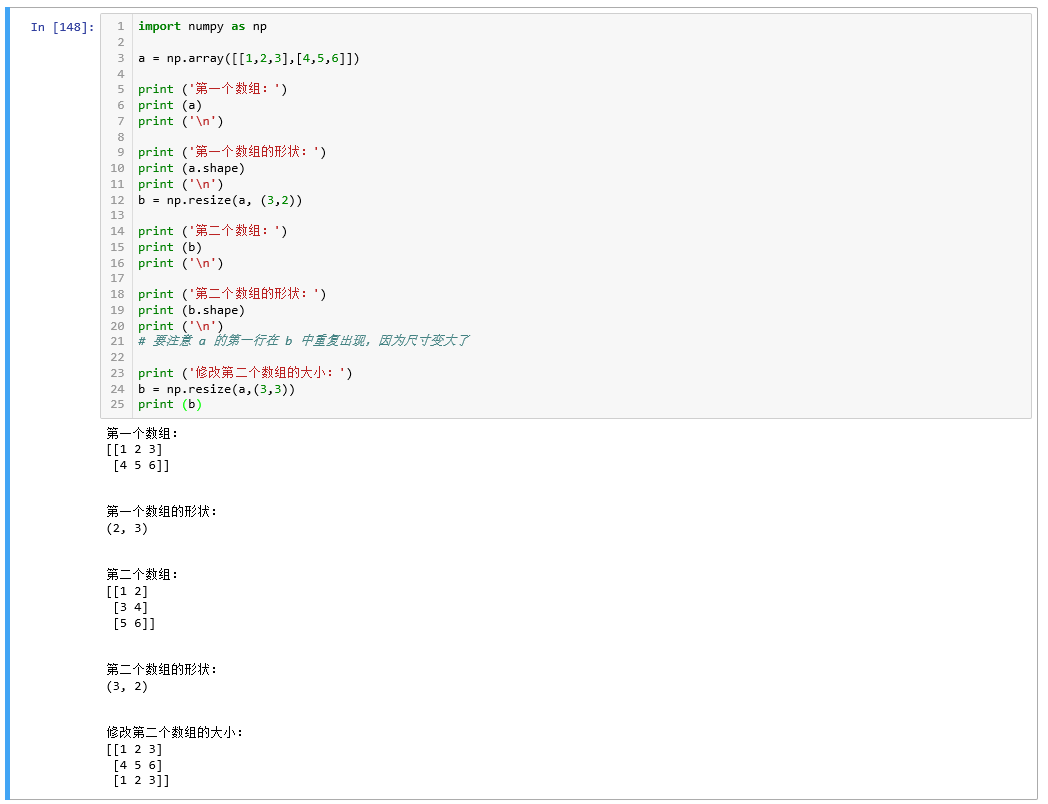
*print ('\n')*

*# 要注意 a 的第一行在 b 中重复出现，因为尺寸变大了*

*print ('修改第二个数组的大小：')*

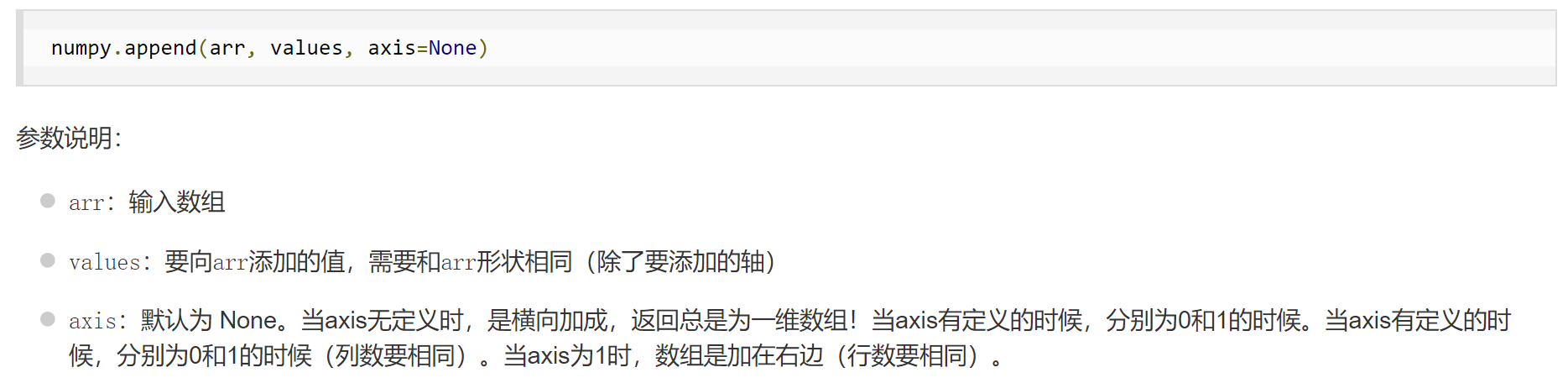
*b = np.resize(a,(3,3))*

*print (b)*



numpy.append

numpy.append 函数在数组的末尾添加值。 追加操作会分配整个数组，并把原来的数组复制到新数组中。 此外，输入数组的维度必须匹配否则将生成ValueError。



*import numpy as np*

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

*print ('第一个数组：')*

*print (a)*

*print ('\n')*

*print ('向数组添加元素：')*

*print (np.append(a, [7,8,9]))*

*print ('\n')*

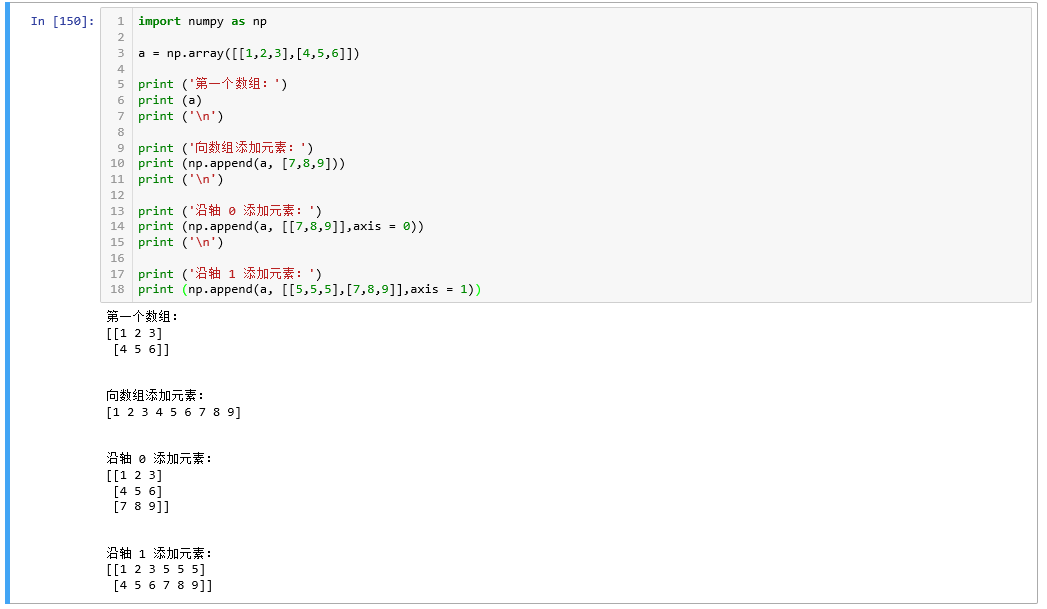
*print ('沿轴 0 添加元素：')*

*print (np.append(a, [[7,8,9]],axis = 0))*

*print ('\n')*

*print ('沿轴 1 添加元素：')*

*print (np.append(a, [[5,5,5],[7,8,9]],axis = 1))*



## 3、NumPy 数学函数

NumPy 包含大量的各种数学运算的函数，包括三角函数，算术运算的函数，复数处理函数等。

**3.1 三角函数**

NumPy 提供了标准的三角函数：sin()、cos()、tan()。arcsin，arccos，和 arctan 函数返回给定角度的 sin，cos 和 tan 的反三角函数。

*import numpy as np*

*a = np.array([0,30,45,60,90])*

*print ('不同角度的正弦值：')*

*# 通过乘 pi/180 转化为弧度*

*print (np.sin(a\*np.pi/180))*

*print ('\n')*

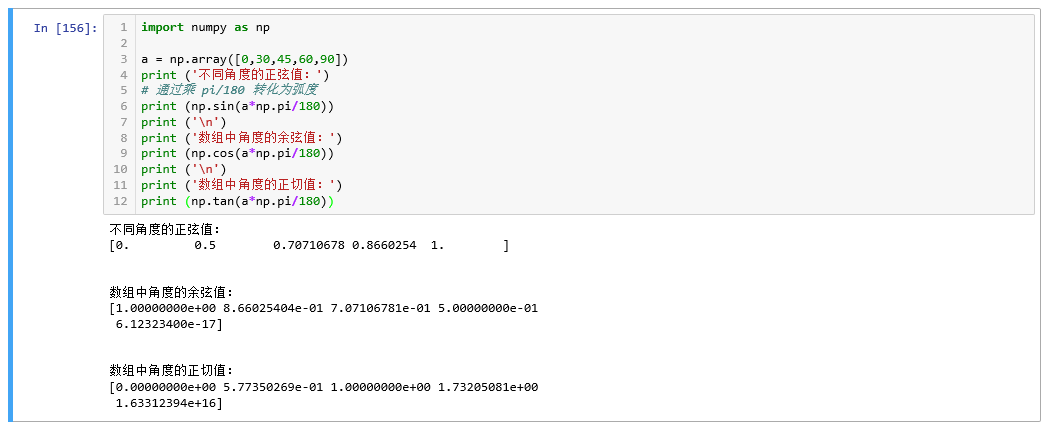
*print ('数组中角度的余弦值：')*

*print (np.cos(a\*np.pi/180))*

*print ('\n')*

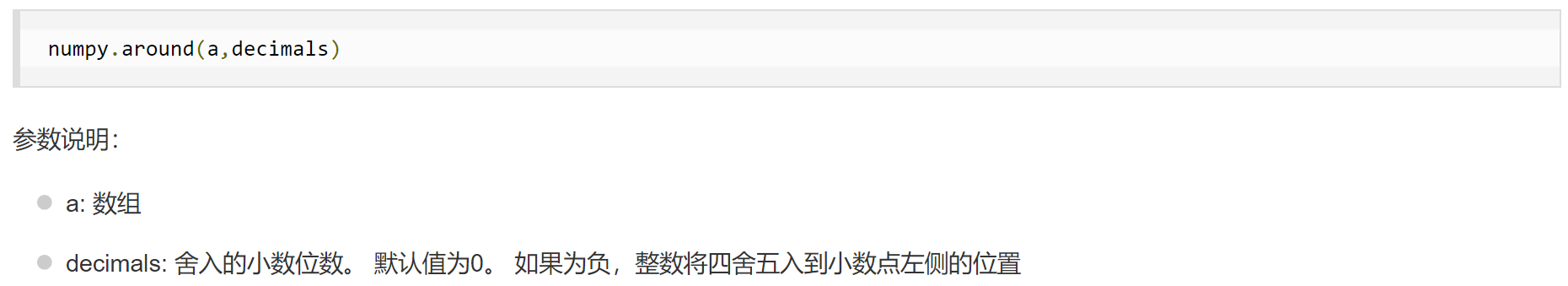
*print ('数组中角度的正切值：')*

*print (np.tan(a\*np.pi/180))*



**3.2 舍入函数**

numpy.around() 函数返回指定数字的四舍五入值。



import numpy as np

a = np.array([1.0,5.55, 123, 0.567, 25.532])

print ('原数组：')

print (a)

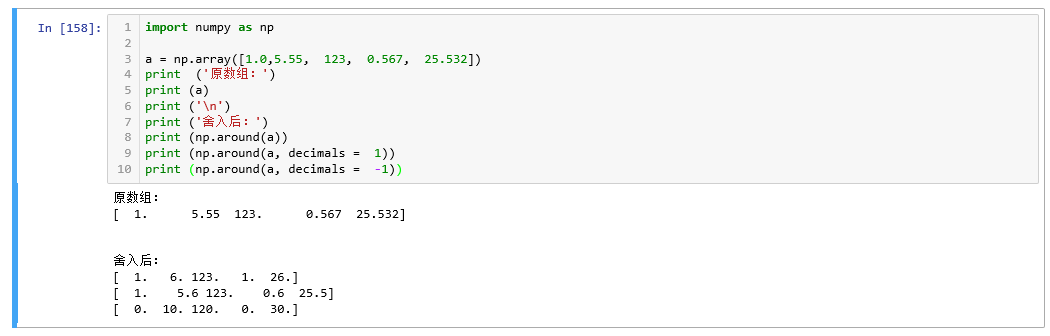
print ('\n')

print ('舍入后：')

print (np.around(a))

print (np.around(a, decimals = 1))

print (np.around(a, decimals = -1))



numpy.floor()

numpy.floor() 返回数字的下舍整数。

numpy.ceil()

numpy.ceil() 返回数字的上入整数。

import numpy as np

a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

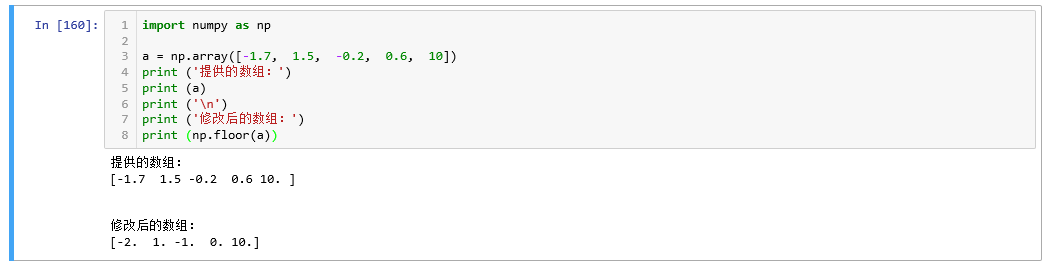
print ('提供的数组：')

print (a)

print ('\n')

print ('修改后的数组：')

print (np.floor(a))



## 4、NumPy 算术函数

**NumPy 算术函数包含简单的加减乘除: add()，subtract()，multiply() 和 divide()。**

import numpy as np

*a = np.arange(9, dtype = np.float\_).reshape(3,3)*

*print ('第一个数组：')*

*print (a)*

*print ('\n')*

*print ('第二个数组：')*

*b = np.array([10,10,10])*

*print (b)*

*print ('\n')*

*print ('两个数组相加：')*

*print (np.add(a,b))*

*print ('\n')*

*print ('两个数组相减：')*

*print (np.subtract(a,b))*

*print ('\n')*

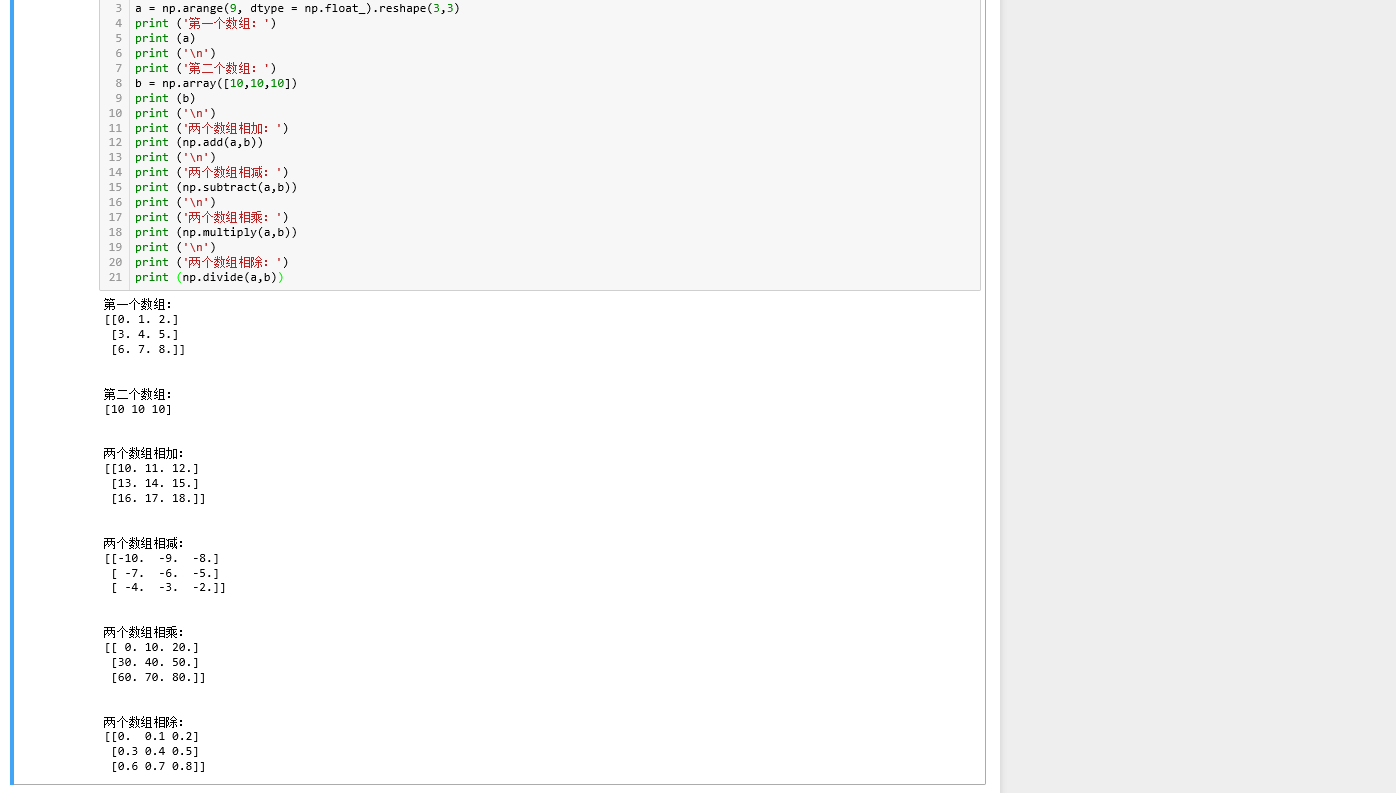
*print ('两个数组相乘：')*

*print (np.multiply(a,b))*

*print ('\n')*

*print ('两个数组相除：')*

*print (np.divide(a,b))*



**numpy.reciprocal()**

numpy.reciprocal() 函数返回参数逐元素的倒数。如 1/4 倒数为 4/1。

import numpy as np

a = np.array([0.25, 1.33, 1, 100])

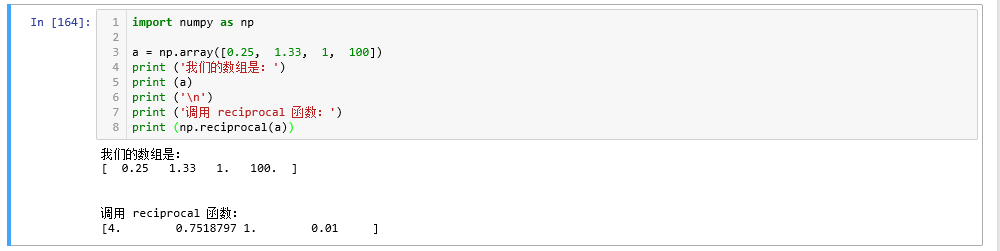
print ('我们的数组是：')

print (a)

print ('\n')

print ('调用 reciprocal 函数：')

print (np.reciprocal(a))



**numpy.power()**

numpy.power() 函数将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂。

import numpy as np

a = np.array([10,100,1000])

print ('我们的数组是；')

print (a)

print ('\n')

print ('调用 power 函数：')

print (np.power(a,2))

print ('\n')

print ('第二个数组：')

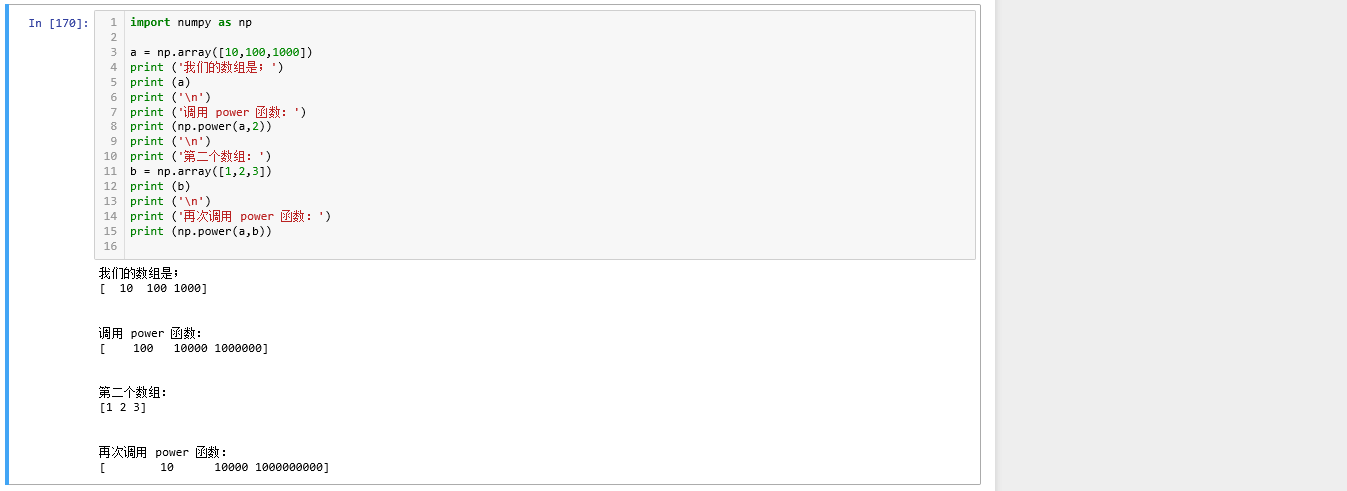
b = np.array([1,2,3])

print (b)

print ('\n')

print ('再次调用 power 函数：')

print (np.power(a,b))



**numpy.mod()**

numpy.mod() 计算输入数组中相应元素的相除后的余数。

## 5、NumPy 统计函数

NumPy 提供了很多统计函数，用于从数组中查找最小元素，最大元素，百分位标准差和方差等。 函数说明如下：

**numpy.min() 和 numpy.max()**

numpy.min() 用于计算数组中的元素沿指定轴的最小值。

numpy.max() 用于计算数组中的元素沿指定轴的最大值。

import numpy as np

a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print ('我们的数组是：')

print (a)

print ('调用 min() 函数：')

print (np.min(a,1))

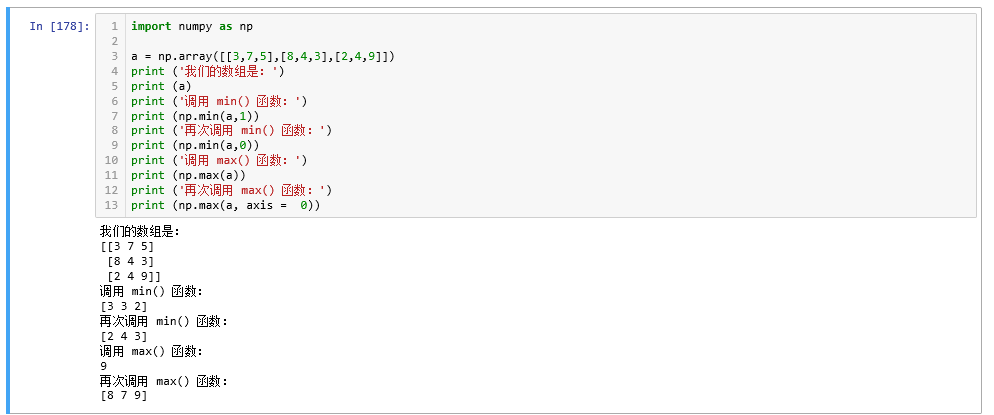
print ('再次调用 min() 函数：')

print (np.min(a,0))

print ('调用 max() 函数：')

print (np.max(a))

print ('再次调用 max() 函数：')

print (np.max(a, axis = 0))

**numpy.mean()**

numpy.mean() 函数返回数组中元素的算术平均值。 如果提供了轴，则沿其计算。算术平均值是沿轴的元素的总和除以元素的数量。

**numpy.sum()** 数返回数组中元素的算术和。

**numpy.median()**

numpy.median() 函数用于计算数组 a 中元素的中位数（中值）

**numpy.average()**

numpy.average() 函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值。

该函数可以接受一个轴参数。 如果没有指定轴，则数组会被展开。

加权平均值即将各数值乘以相应的权数，然后加总求和得到总体值，再除以总的单位数。

考虑数组[1,2,3,4]和相应的权重[4,3,2,1]，通过将相应元素的乘积相加，并将和除以权重的和，来计算加权平均值。

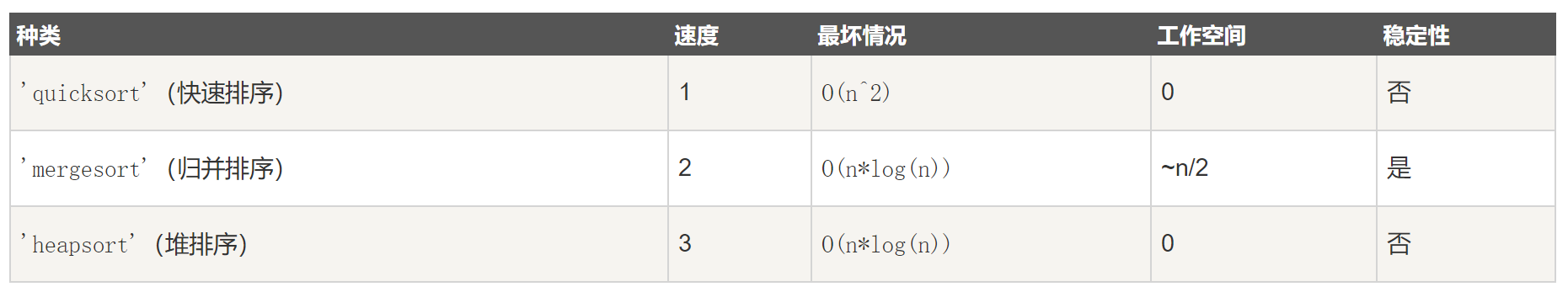
加权平均值 = (1\*4+2\*3+3\*2+4\*1)/(4+3+2+1)

numpy.std() 标准差

numpy.var() 方差

## 6、NumPy 排序、条件刷选函数

NumPy 提供了多种排序的方法。 这些排序函数实现不同的排序算法，每个排序算法的特征在于执行速度，最坏情况性能，所需的工作空间和算法的稳定性。 下表显示了三种排序算法的比较。



**numpy.sort()**

numpy.sort() 函数返回输入数组的排序副本。函数格式如下：

import numpy as np

a = np.array([[3,7],[9,1]])

print ('我们的数组是：')

print (a)

print ('\n')

print ('调用 sort() 函数：')

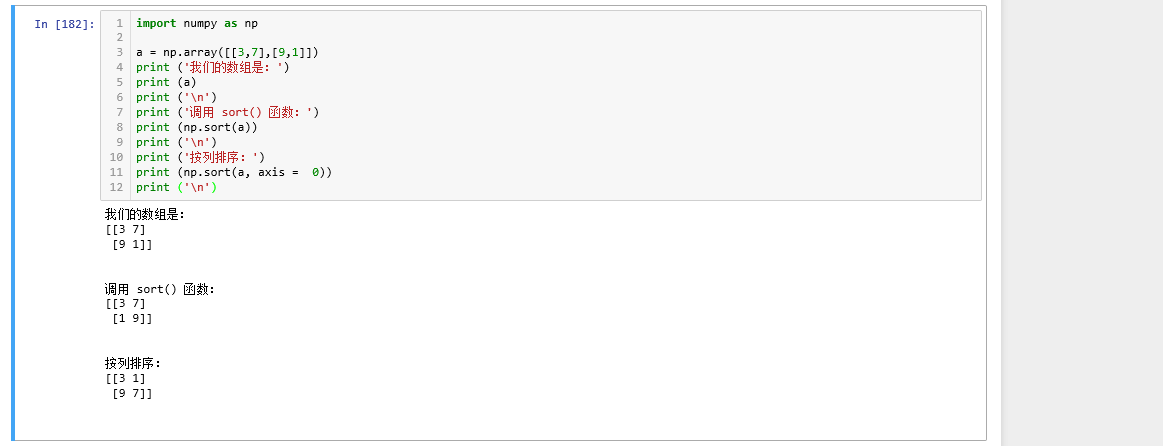
print (np.sort(a))

print ('\n')

print ('按列排序：')

print (np.sort(a, axis = 0))

print ('\n')



**numpy.argsort()**

numpy.argsort() 函数返回的是数组值从小到大的索引值。

import numpy as np

x = np.array([3, 1, 2])

print ('我们的数组是：')

print (x)

print ('\n')

print ('对 x 调用 argsort() 函数：')

y = np.argsort(x)

print (y)

print ('\n')

print ('以排序后的顺序重构原数组：')

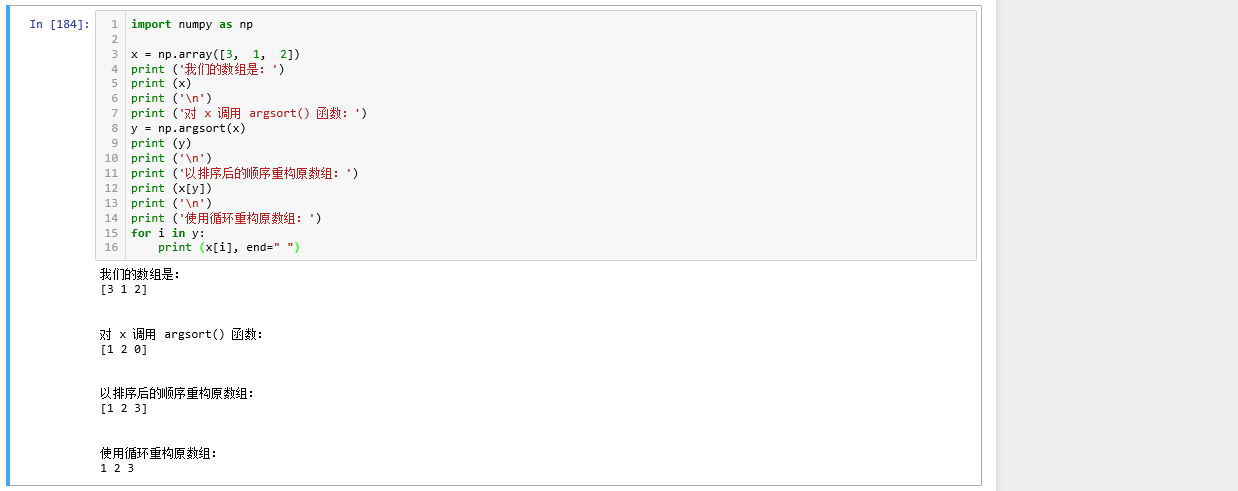
print (x[y])

print ('\n')

print ('使用循环重构原数组：')

for i in y:

print (x[i], end=" ")



**numpy.argmax() 和 numpy.argmin()**

import numpy as np

a = np.array([[30,40,70],[80,20,10],[50,90,60]])

print ('我们的数组是：')

print (a)

print ('调用 argmax() 函数：')

print (np.argmax(a))

print ('展开数组：')

print (a.flatten())

print ('沿轴 0 的最大值索引：')

maxindex = np.argmax(a, axis = 0)

print (maxindex)

print ('沿轴 1 的最大值索引：')

maxindex = np.argmax(a, axis = 1)

print (maxindex)

print ('调用 argmin() 函数：')

minindex = np.argmin(a)

print (minindex)

print ('展开数组中的最小值：')

print (a.flatten()[minindex])

print ('沿轴 0 的最小值索引：')

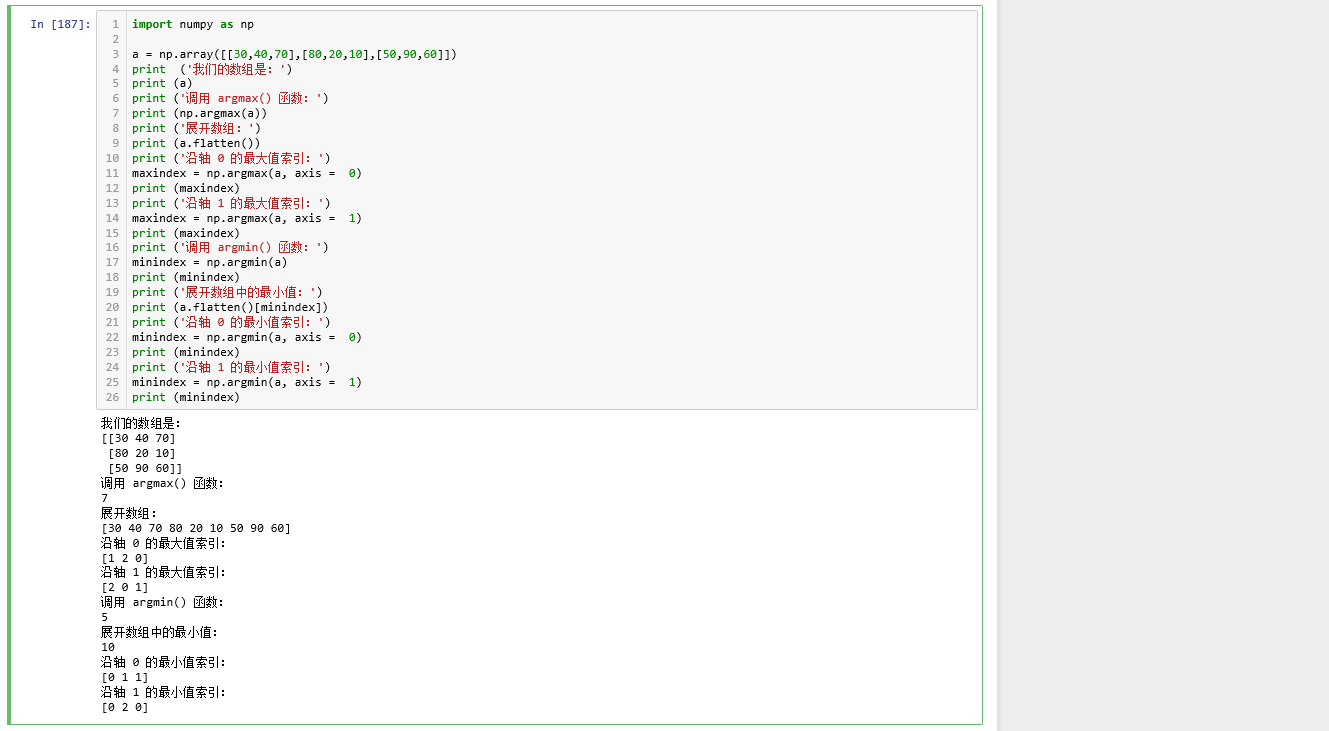
minindex = np.argmin(a, axis = 0)

print (minindex)

print ('沿轴 1 的最小值索引：')

minindex = np.argmin(a, axis = 1)

print (minindex)



numpy.nonzero()

numpy.nonzero() 函数返回输入数组中非零元素的索引。

numpy.where()

numpy.where() 函数返回输入数组中满足给定条件的元素的索引。

numpy.extract()

numpy.extract() 函数根据某个条件从数组中抽取元素，返回满条件的元素。

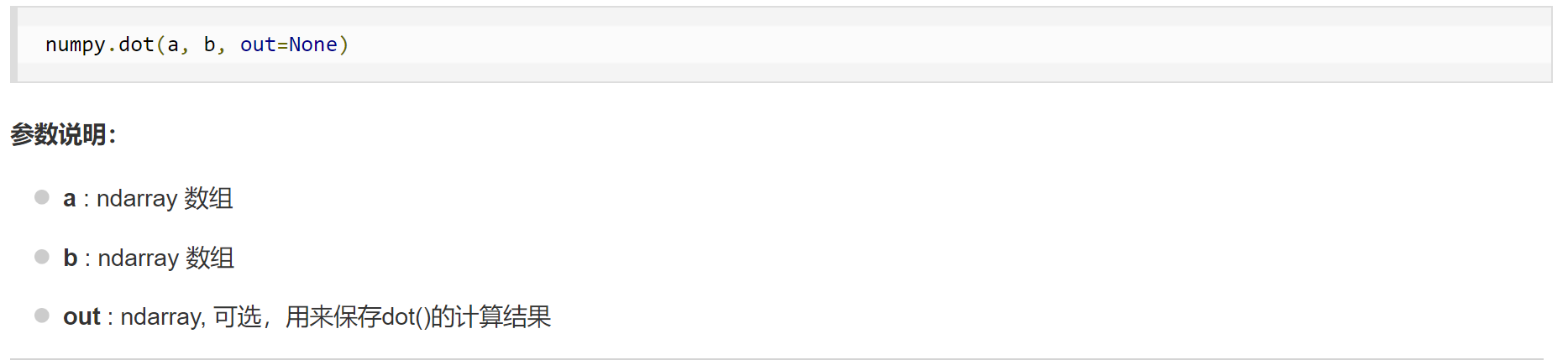
## 7、NumPy 线性代数

NumPy 提供了线性代数函数库 linalg，该库包含了线性代数所需的所有功能，可以看看下面的说明：



**numpy.dot()**

numpy.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；对于多维数组，它的通用计算公式如下，即结果数组中的每个元素都是：数组a的最后一维上的所有元素与数组b的倒数第二位上的所有元素的乘积和： dot(a, b)[i,j,k,m] = sum(a[i,j,:] \* b[k,:,m])。



import numpy.matlib

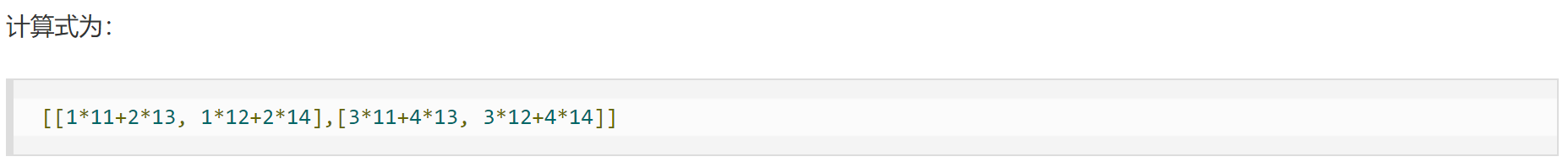
import numpy as np

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

b = np.array([[11,12],[13,14]])

print(np.dot(a,b))

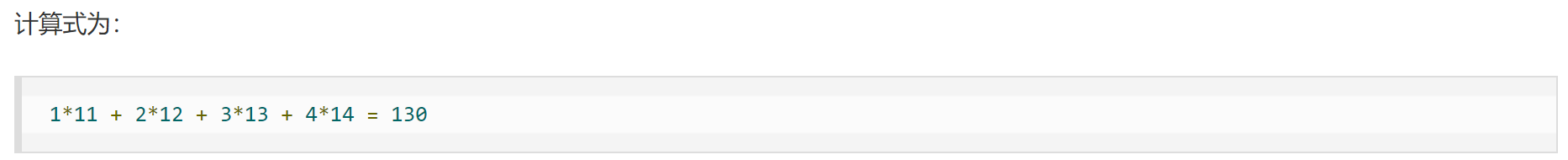




numpy.vdot()

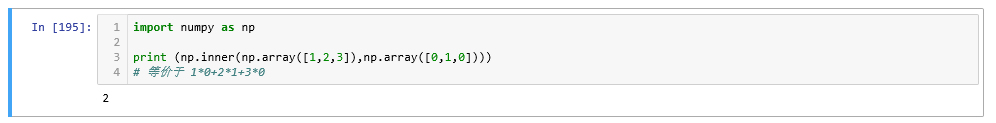
numpy.vdot() 函数是两个向量的点积。 如果第一个参数是复数，那么它的共轭复数会用于计算。 如果参数是多维数组，它会被展开。





**numpy.inner()**

numpy.inner() 函数返回一维数组的向量内积。对于更高的维度，它返回最后一个轴上的和的乘积。



import numpy as np

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

print ('数组 a：')

print (a)

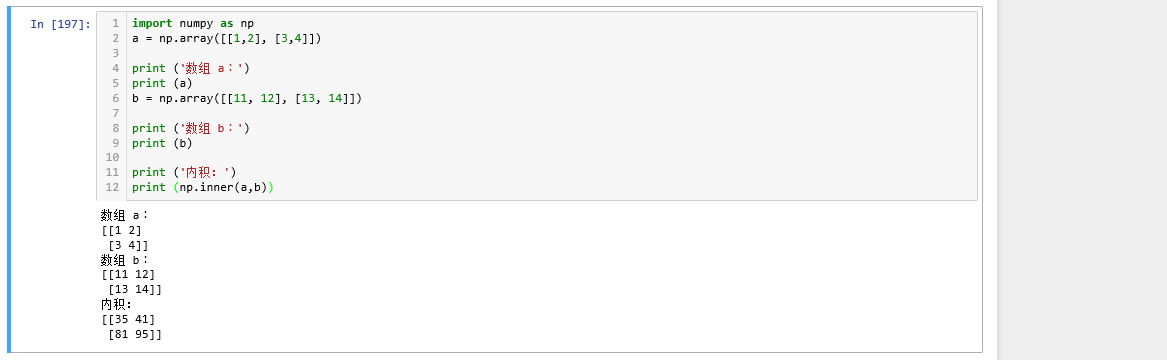
b = np.array([[11, 12], [13, 14]])

print ('数组 b：')

print (b)

print ('内积：')

print (np.inner(a,b))



numpy.matmul

numpy.matmul 函数返回两个数组的矩阵乘积。 虽然它返回二维数组的正常乘积，但如果任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。

import numpy as np

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

print ('数组 a：')

print (a)

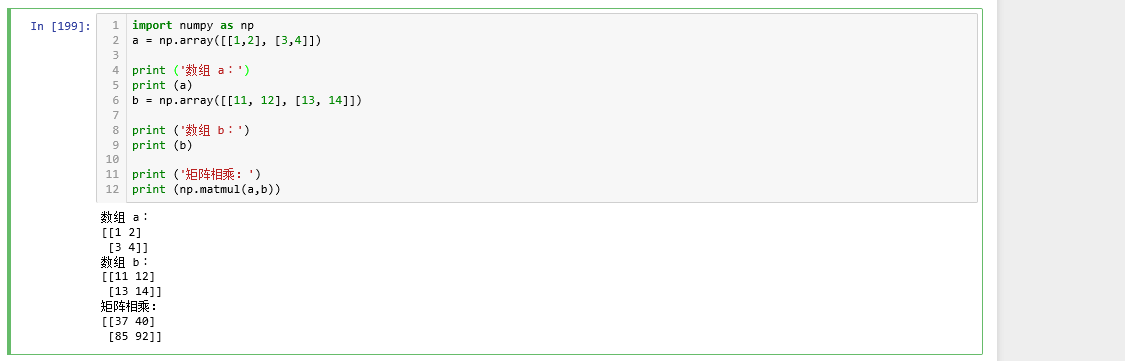
b = np.array([[11, 12], [13, 14]])

print ('数组 b：')

print (b)

print ('矩阵相乘：')

print (np.matmul(a,b))



## 练习：

1.随机生成一个3\*4的矩阵，计算每行的和；每列的平均值；每列的最大值的索引。