

Numpy

NumPy(Numerical Python) 是科学计算基础库，提供大量科学计算相关功能，比如数据统计，随机数生成等。其提供最核心类型为多维数组类型（ndarray），支持大量的维度数组与矩阵运算，Numpy 支持向量处理 ndarray 对象，提高程序运算速度。

安装

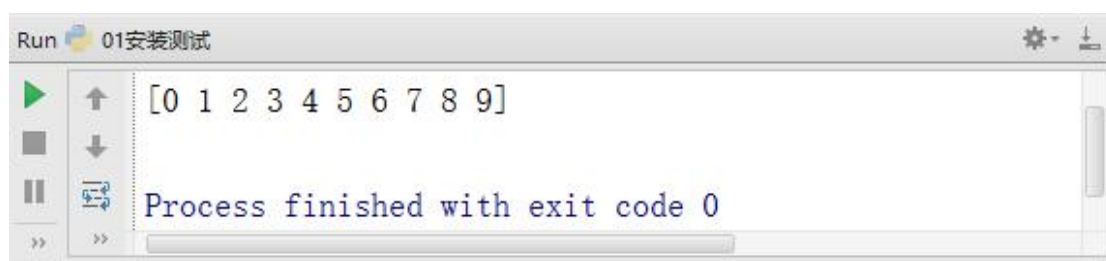
安装 NumPy 最简单的方法就是使用 pip 工具，语法格式如下：

```
pip install numpy
```

【示例】arange 函数测试环境安装

```
import numpy as np
a=np.arange(10)
print(a)
```

执行结果如图所示：



在上面的程序中只涉及 numpy 模块中的一个 arange 函数，该函数可以传入一个整数类型的参数 n，函数返回值看着像一个列表，其实返回值类型是 numpy.ndarray。这是 Numpy 中特有的数组类型。如果传入 arange 函数的参数值是 n，那么 arange 函数会返回 0 到 n-1 的 ndarray 类型的数组。

数组的创建

array 创建

numpy 模块的 array 函数可以生成多维数组。例如，如果要生成一个二维数组，需要向 array 函数传递一个列表类型的参数。每一个列表元素是一维的 ndarray 类型数组，作为二维数组的行。另外，通过 ndarray 类的 shape 属性可以获得数组每一维的元素个数（元组形式），也可以通过 shape[n]形式获得每一维的元素个数，其中 n 是维度，从 0 开始。

语法格式如下：

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

array 参数说明

名称	描述
----	----

object	数组或嵌套的数列
dtype	数组元素的数据类型，可选
copy	对象是否需要复制，可选
order	创建数组的样式，C 为行方向，F 为列方向，A 为任意方向（默认）
subok	默认返回一个与基类类型一致的数组
ndmin	指定生成数组的最小维度

【示例】创建一维

```
b=np.array([1,2,3,4,5,6])
print(b)
print('b 数组的维度: ',b.shape)
```

执行结果如图所示：



【示例】创建二维

```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print('a 数组的维度:',a.shape)
```

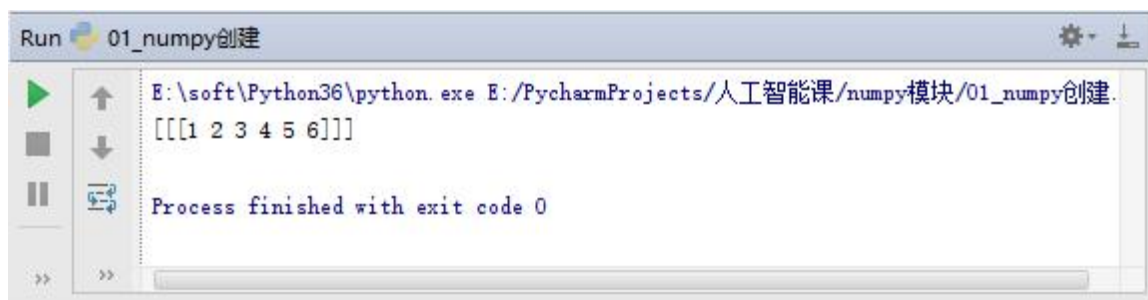
执行结果如图所示：



【示例】ndmin 参数的使用

```
import numpy as np
a=np.array([1,2,3,4,5,6],ndmin=3)
print(a)
```

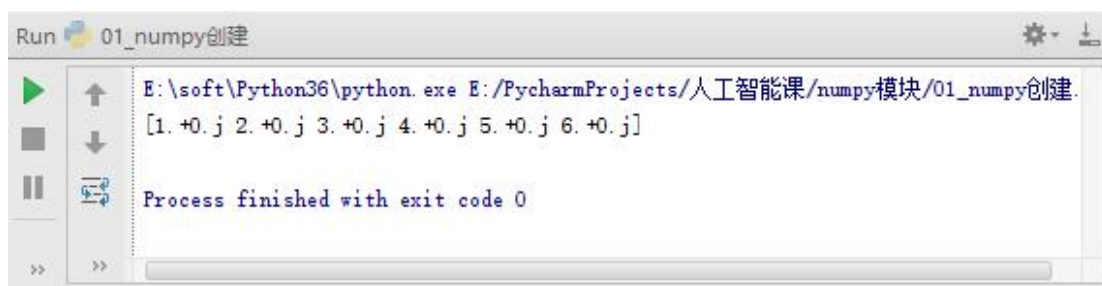
执行结果如图所示：



【示例】dtype 参数的使用

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

执行结果如图所示：



arange 创建

使用 arange 函数创建数值范围并返回 ndarray 对象，函数格式如下：

```
numpy.arange(start, stop, step, dtype)
```

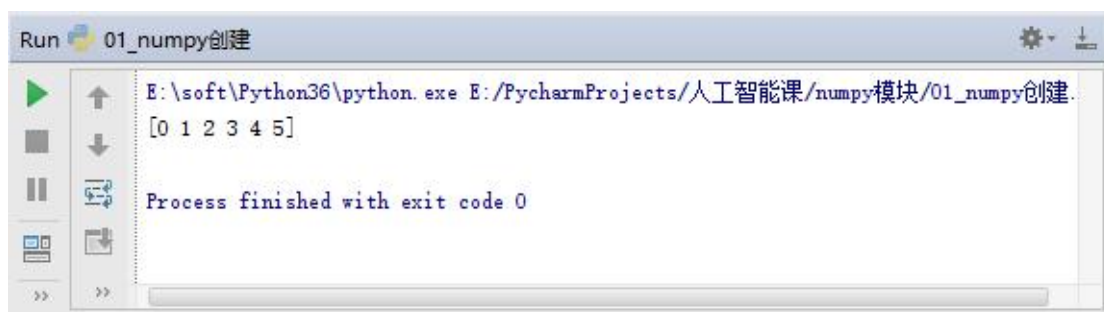
arange 参数说明

参数	描述
start	起始值，默认为 0
stop	终止值（不包含）
step	步长，默认为 1
dtype	返回 ndarray 的数据类型，如果没有提供，则会使用输入数据的类型。

【示例】arange 生成 0 到 5 的数组

```
x=np.arange(0,6,dtype=int)
print(x)
```

执行结果如图所示：

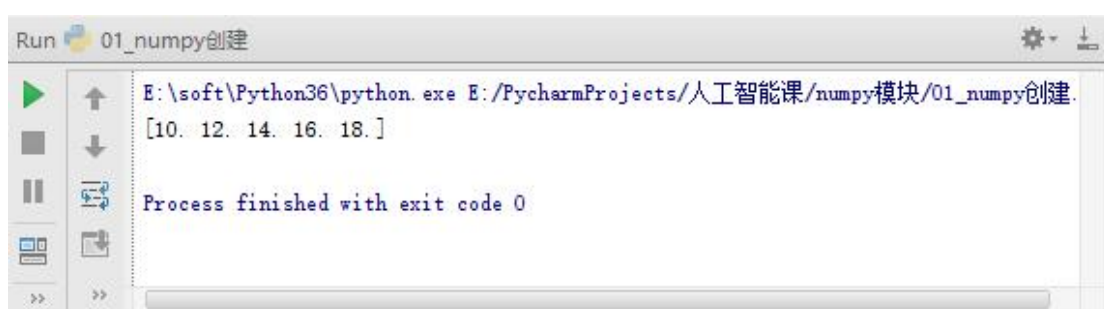


```
Run 01_numpy创建
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/01_numpy创建.
[0 1 2 3 4 5]
Process finished with exit code 0
```

【示例】arange 设置了起始值、终止值及步长

```
x=np.arange(10,20,2,dtype=float)
print(x)
```

执行结果如图所示：



```
Run 01_numpy创建
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/01_numpy创建.
[10. 12. 14. 16. 18.]
Process finished with exit code 0
```

【示例】arange 创建二维数组

```
b=np.array([np.arange(1,4),np.arange(4,7),np.arange(7,10)])
print(b)
print('b 数组的维度: ',b.shape)
```

执行结果如图所示：



```
Run 03二维数组的创建
[[1 2 3]
 [4 5 6]
 [7 8 9]]
b数组的维度: (3, 3)
Process finished with exit code 0
```

随机数创建

```
numpy.random.random(size=None)
```

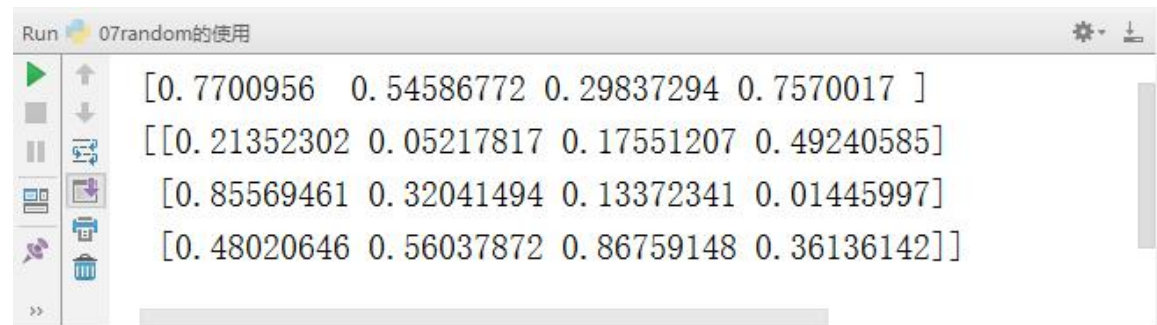
该方法返回[0.0, 1.0)范围的随机数。

【示例】numpy.random.random(size=None)的使用

```
#numpy.random.random(size=None)
#返回[0.0, 1.0)范围的随机数
```

```
import numpy as np
x=np.random.random(size=4)
y=np.random.random(size=(3,4))
print(x)
print(y)
```

执行结果如图所示：



```
Run 07random的使用
[0.7700956 0.54586772 0.29837294 0.7570017 ]
[[0.21352302 0.05217817 0.17551207 0.49240585]
 [0.85569461 0.32041494 0.13372341 0.01445997]
 [0.48020646 0.56037872 0.86759148 0.36136142]]
```

`numpy.random.randint()`

该方法有三个参数 `low`、`high`、`size` 三个参数。默认 `high` 是 `None`，如果只有 `low`，那范围就是 `[0,low)`。如果有 `high`，范围就是 `[low,high)`。

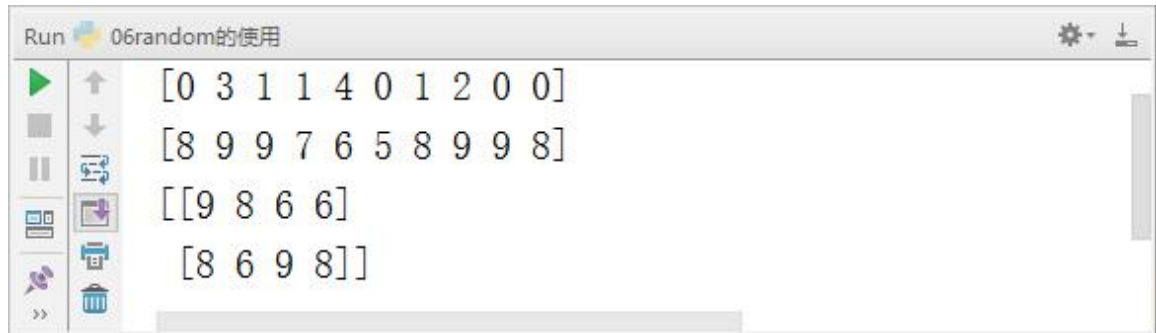
【示例】`numpy.random.randint()`的使用

```
import numpy as np
#numpy.random.randint()的使用
#生成 [0,low)范围的随机整数
x=np.random.randint(5,size=10)
print(x)

#生成[low,high)范围的随机整数
y=np.random.randint(5,10,size=10)
print(y)

#生成[low,high)范围的 2*4 的随机整数
z=np.random.randint(5,10,size=(2,4))
print(z)
```

执行结果如图所示：



```
Run 06random的使用
[0 3 1 1 4 0 1 2 0 0]
[8 9 9 7 6 5 8 9 9 8]
[[9 8 6 6]
 [8 6 9 8]]
```

`numpy.random.randn(d0,d1,...,dn)`

`randn` 函数返回一个或一组样本，具有标准正态分布（期望为 0，方差为 1）。

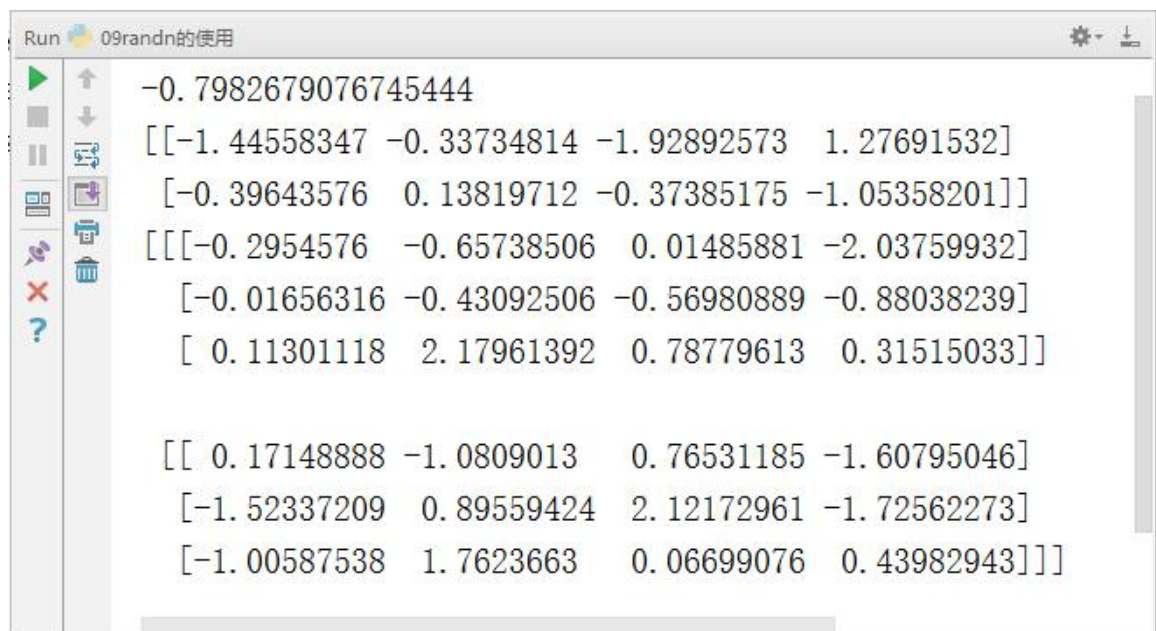
`dn` 表格每个维度

返回值为指定维度的 `array`

【示例】`numpy.random.randn()` 的使用

```
# randn 函数返回一个或一组样本，具有标准正态分布
import numpy as np
x=np.random.randn()
print(x)
y=np.random.randn(2,4)
print(y)
z=np.random.randn(2,3,4)
print(z)
```

执行结果如图所示：



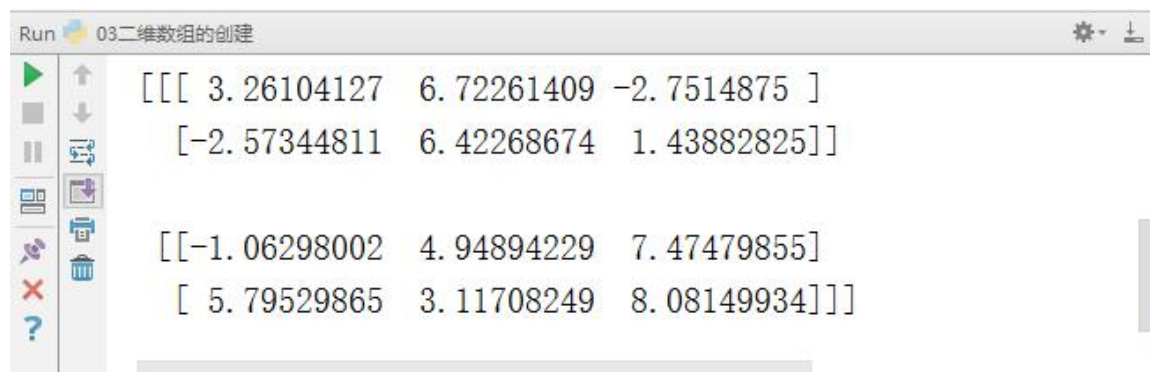
```
Run 09randn的使用
-0.7982679076745444
[[-1.44558347 -0.33734814 -1.92892573  1.27691532]
 [-0.39643576  0.13819712 -0.37385175 -1.05358201]]
[[[-0.2954576 -0.65738506  0.01485881 -2.03759932]
 [-0.01656316 -0.43092506 -0.56980889 -0.88038239]
 [ 0.11301118  2.17961392  0.78779613  0.31515033]]

[[ 0.17148888 -1.0809013  0.76531185 -1.60795046]
 [-1.52337209  0.89559424  2.12172961 -1.72562273]
 [-1.00587538  1.7623663  0.06699076  0.43982943]]]
```

【示例】`np.random.normal` 指定期望和方差的正太分布


```
#正太分布（高斯分布）loc：期望 scale：方差 size 形状
print(np.random.normal(loc=3,scale=4,size=(2,2,3)))
```

执行结果如图所示：



```
Run 03二维数组的创建
[[[ 3.26104127  6.72261409 -2.7514875 ]
  [-2.57344811  6.42268674  1.43882825]]

 [[-1.06298002  4.94894229  7.47479855]
  [ 5.79529865  3.11708249  8.08149934]]]
```

ndarray 对象

NumPy 最重要的一个特点是其 N 维数组对象 `ndarray`，它是一系列同类型数据的集合，以 0 下标为开始进行集合中元素的索引。

`ndarray` 对象是用于存放同类型元素的多维数组。

`ndarray` 中的每个元素在内存中都有相同存储大小的区域。

`ndarray` 内部由以下内容组成：

- 一个指向数据（内存或内存映射文件中的一块数据）的指针。
- 数据类型或 `dtype`，描述在数组中的固定大小值的格子。
- 一个表示数组形状（`shape`）的元组，表示各维度大小的元组。

NumPy 的数组中比较重要 `ndarray` 对象属性有：

ndarray 对象属性

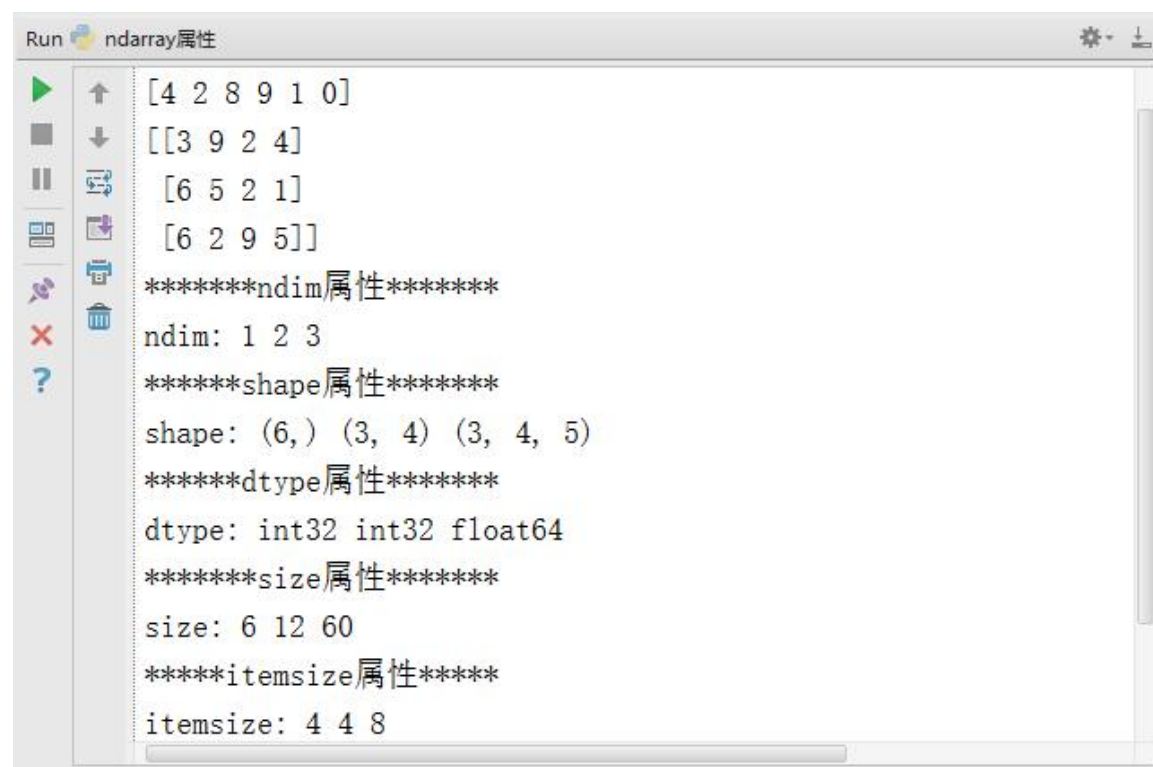
属性	说明
<code>ndarray.ndim</code>	秩，即轴的数量或维度的数量
<code>ndarray.shape</code>	数组的维度，对于矩阵，n 行 m 列
<code>ndarray.size</code>	数组元素的总个数，相当于 <code>.shape</code> 中 <code>n*m</code> 的值
<code>ndarray.dtype</code>	<code>ndarray</code> 对象的元素类型
<code>ndarray.itemsize</code>	<code>ndarray</code> 对象中每个元素的大小，以字节为单位
<code>ndarray.flags</code>	<code>ndarray</code> 对象的内存信息
<code>ndarray.real</code>	<code>ndarray</code> 元素的实部
<code>ndarray.imag</code>	<code>ndarray</code> 元素的虚部
<code>ndarray.data</code>	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所

以通常不需要使用这个属性。

【示例】ndarray 属性测试

```
import numpy as np
x1=np.random.randint(10,size=6)
print(x1)
x2=np.random.randint(10,size=(3,4))
print(x2)
x3=np.random.randn(3,4,5)
print('ndim 属性'.center(20,'*'))
print('ndim:',x1.ndim,x2.ndim,x3.ndim)
print('shape 属性'.center(20,'*'))
print('shape:',x1.shape,x2.shape,x3.shape)
print('dtype 属性'.center(20,'*'))
print('dtype:',x1.dtype,x2.dtype,x3.dtype)
print('size 属性'.center(20,'*'))
print('size:',x1.size,x2.size,x3.size)
print('itemsize 属性'.center(20,'*'))#一个字节是 8 位
print('itemsize:',x1.itemsize,x2.itemsize,x3.itemsize)
```

执行结果如图所示：



```
Run ndarray属性
[4 2 8 9 1 0]
[[3 9 2 4]
 [6 5 2 1]
 [6 2 9 5]]
*****ndim属性*****
ndim: 1 2 3
*****shape属性*****
shape: (6,) (3, 4) (3, 4, 5)
*****dtype属性*****
dtype: int32 int32 float64
*****size属性*****
size: 6 12 60
*****itemsize属性*****
itemsize: 4 4 8
```


其他方式创建

ndarray 数组除了可以使用底层 ndarray 构造器来创建外，也可以通过以下几种方式来创建。

zeros 创建指定大小的数组，数组元素以 0 来填充：

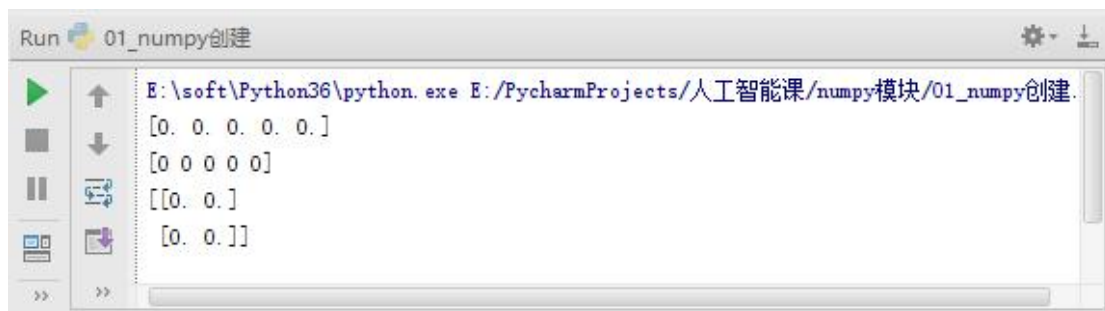
```
numpy.zeros(shape, dtype = float, order = 'C')
```

【示例】zeros 创建

```
x=np.zeros(5)
print(x)
#设置类型为整数
y=np.zeros((5,),dtype=int)
print(y)

z=np.zeros((2,2))
print(z)
```

执行结果如图所示：



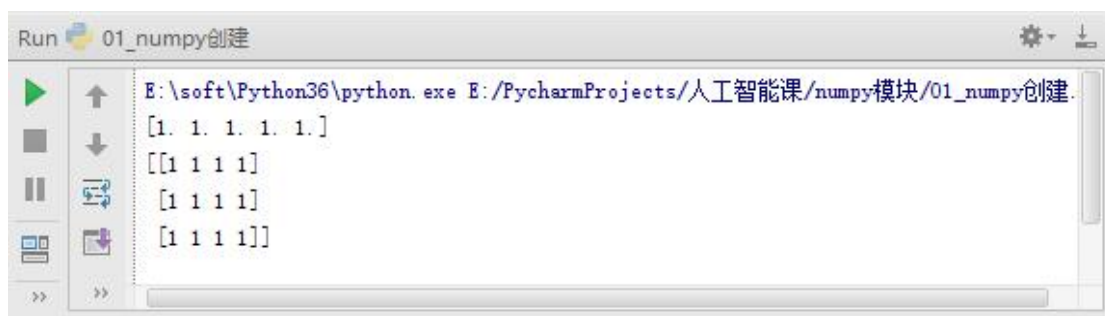
numpy.ones 创建指定形状的数组，数组元素以 1 来填充：

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

【示例】ones 创建

```
x=np.ones(5)
print(x)
y=np.ones((3,4),dtype=int)
print(y)
```

执行结果如图所示：



`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，里面的元素的值是之前内存的值：

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

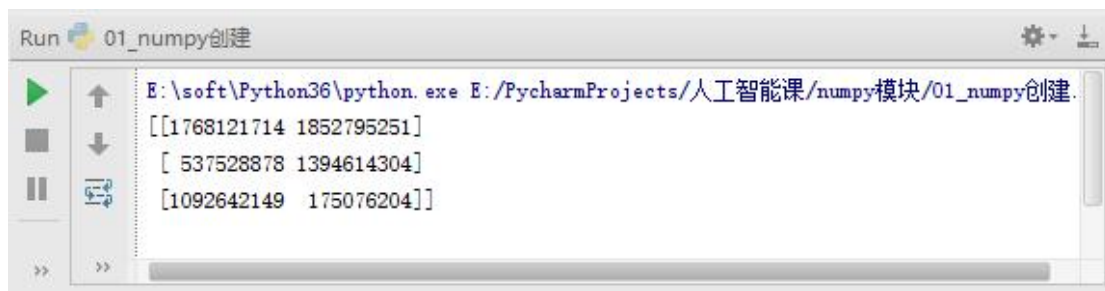
表 10-3 empty 参数说明

参数	描述
shape	数组形状
dtype	数据类型，可选
order	有"C"和"F"两个选项,分别代表，行优先和列优先，在计算机内存中的存储元素的顺序。

【示例】empty 创建

```
x=np.empty([3,2],dtype=int)
print(x)
```

执行结果如图所示：



`linspace` 函数用于创建一个一维数组，数组是一个等差数列构成的，格式如下：

```
np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
```

表 10-6 linspace 参数说明

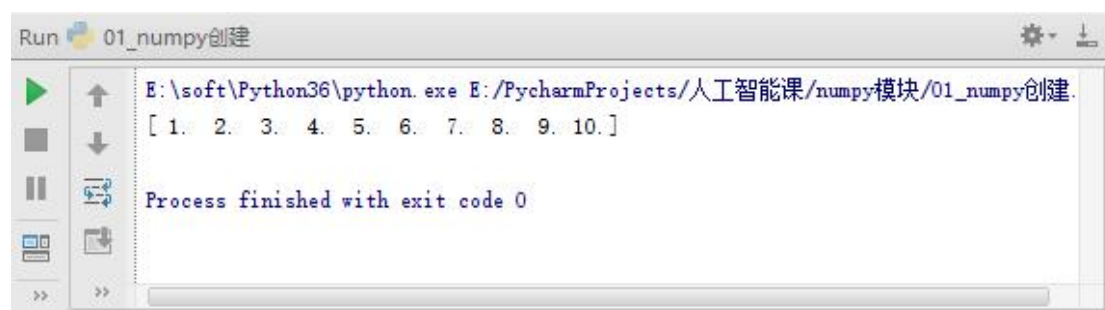
参数	描述
start	序列的起始值
stop	序列的终止值，如果 endpoint 为 true，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50

endpoint	该值为 true 时, 数列中包含 stop 值, 反之不包含, 默认是 True。
retstep	如果为 True 时, 生成的数组中会显示间距, 反之不显示。
dtype	ndarray 的数据类型

【示例】linspace 创建等差数列

```
x=np.linspace(1,10,10)
print(x)
```

执行结果如图所示:

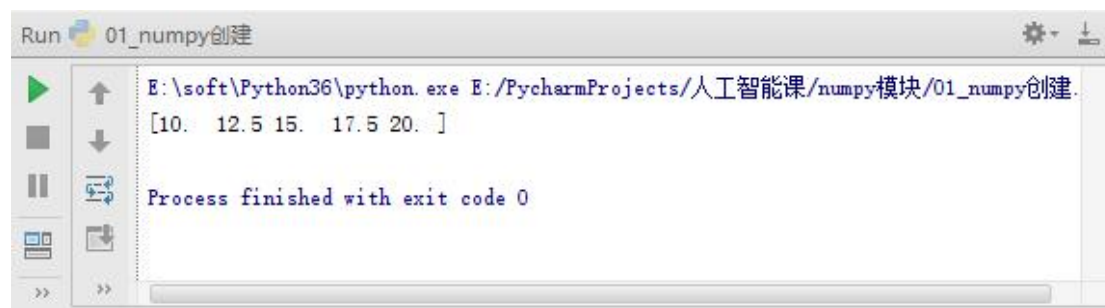


```
Run 01_numpy创建
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/01_numpy创建.
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
Process finished with exit code 0
```

【示例】linspace 创建等差数列 endpoint 设为 true

```
x=np.linspace(10,20,5,endpoint=True)
print(x)
```

执行结果如图所示:

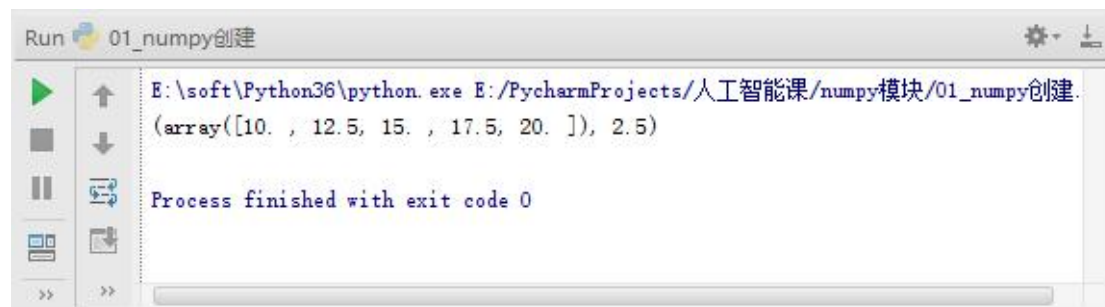


```
Run 01_numpy创建
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/01_numpy创建.
[10.  12.5 15.  17.5 20.]
Process finished with exit code 0
```

【示例】linspace 创建等差数列 endpoint 设为 true, retstep=True

```
x=np.linspace(10,20,5,endpoint=True,retstep=True)
print(x)
```

执行结果如图所示:



```
Run 01_numpy创建
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/01_numpy创建.
(array([10. , 12.5, 15. , 17.5, 20. ]), 2.5)
Process finished with exit code 0
```

numpy.logspace 函数用于创建一个等比数列。格式如下：

```
np.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
```

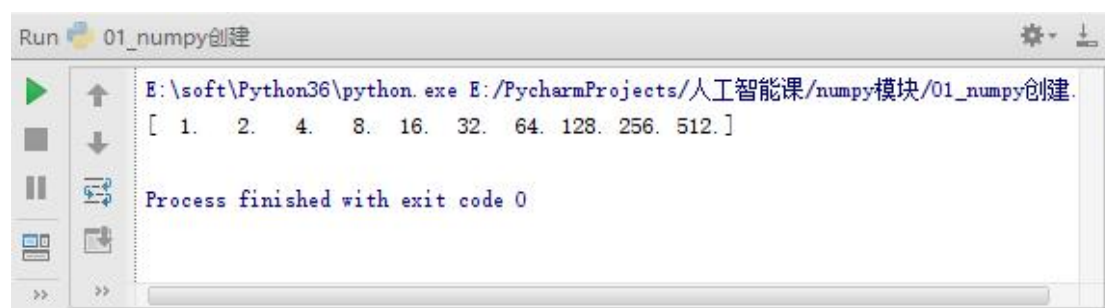
表 10-7 logspace 参数说明

参数	描述
start	序列的起始值为：base ** start
stop	序列的终止值为：base ** stop。如果 endpoint 为 true，该值包含于数列中
num	要生成的等步长的样本数量，默认为 50
endpoint	该值为 true 时，数列中包含 stop 值，反之不包含，默认是 True。
base	对数 log 的底数。
dtype	ndarray 的数据类型

【示例】logspace 创建等比数列

```
x=np.logspace(0,9,10,base=2)
print(x)
```

执行结果如图所示：



切片和索引

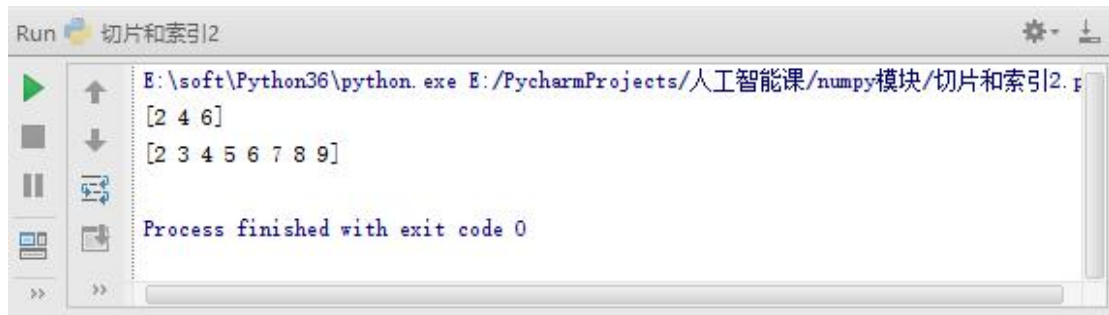
ndarray 对象的内容可以通过索引或切片来访问和修改，与 Python 中 list 的切片操作一样。

ndarray 数组可以基于 0 - n 的下标进行索引，并设置 start, stop 及 step 参数进行，从原数组中切割出一个新数组。

【示例】一维数组切片和索引的使用

```
x=np.arange(10)
y=x[2:7:2]
z=x[2:]
print(y)
print(z)
```

执行结果如图所示：

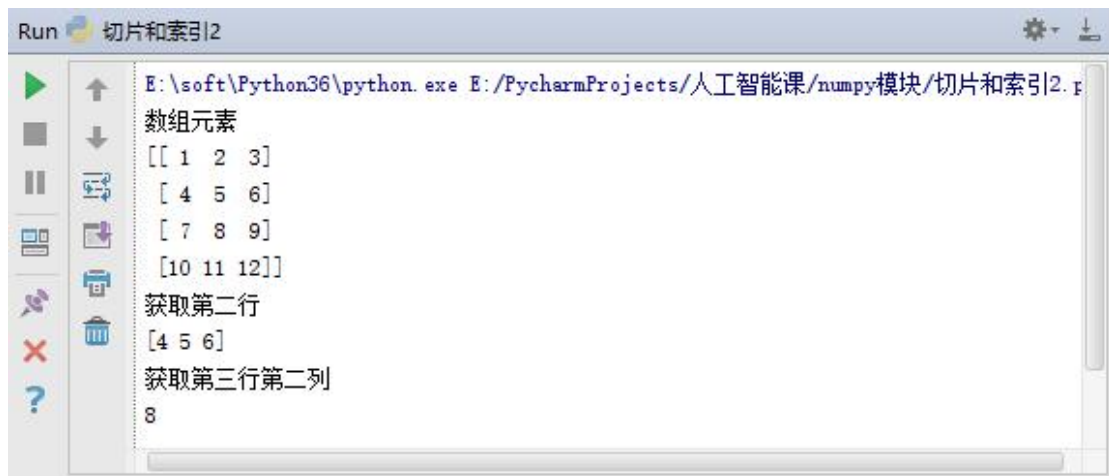


```
Run 切片和索引2
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/切片和索引2.py
[2 4 6]
[2 3 4 5 6 7 8 9]
Process finished with exit code 0
```

【示例】根据索引直接获取

```
x=np.arange(1,13)
a=x.reshape(4,3)
print('数组元素')
print(a)
print('获取第二行')
print(a[1])
print('获取第三行第二列')
print(a[2][1])
```

执行结果如图所示：



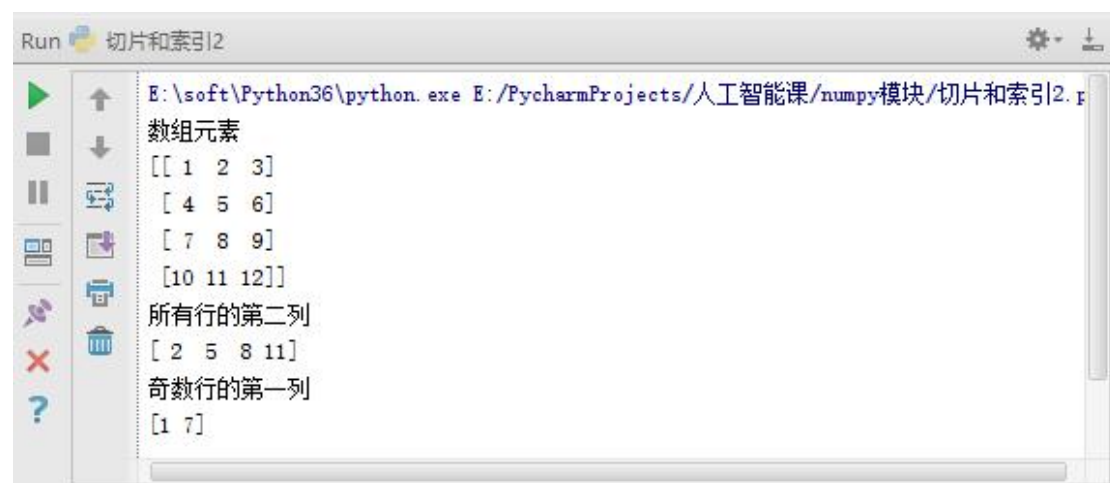
```
Run 切片和索引2
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/切片和索引2.py
数组元素
[[ 1 2 3]
 [ 4 5 6]
 [ 7 8 9]
 [10 11 12]]
获取第二行
[4 5 6]
获取第三行第二列
8
```

【示例】二维数组切片的使用

```
x=np.arange(1,13)
a=x.reshape(4,3)
print('数组元素')
print(a)
#使用索引获取
print('所有行的第二列')
print(a[:,1])
print('奇数行的第一列')
```

```
print(a[:,2,0])
```

执行结果如图所示：



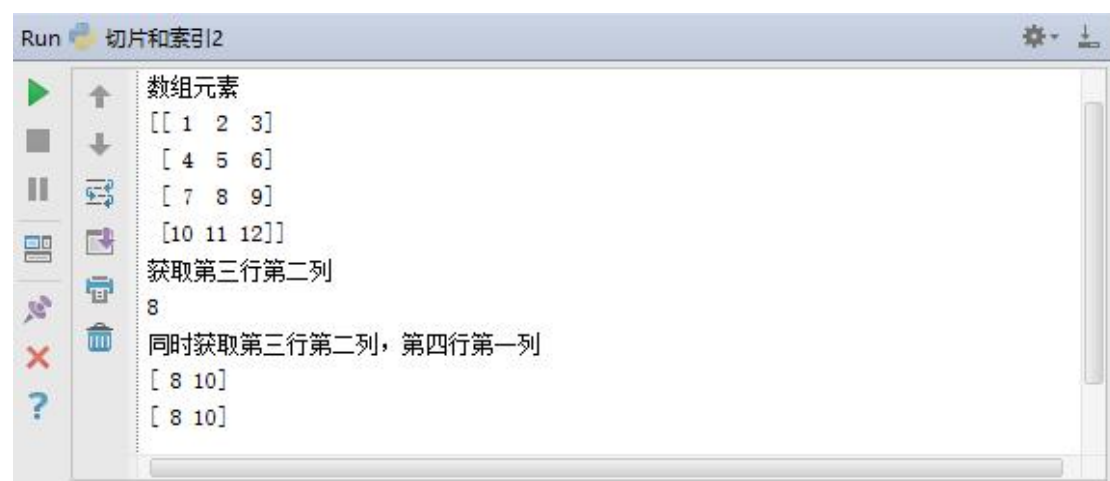
【示例】使用坐标获取数组[x,y]

```

print('获取第三行第二列')
print(a[2,1])
print('同时获取第三行第二列，第四行第一列')
print(np.array([a[2,1],a[3,0]]))
print(a[(2,3),(1,0)])

```

执行结果如图所示：



【示例】索引为负数来获取

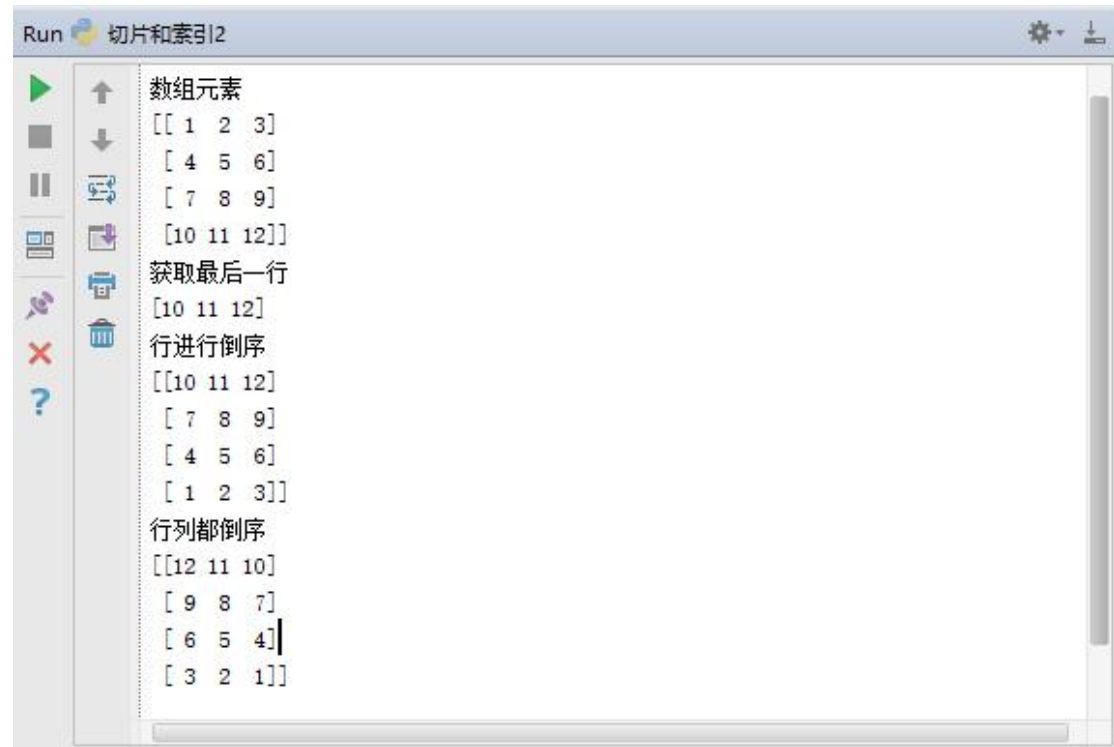
```

print('获取最后一行')
print(a[-1])
print('行进行倒序')

```

```
print(a[::-1])
print('行列都倒序')
print(a[::-1,:-1])
```

执行结果如图所示：



所有切片取出来的数组，即使你把它赋值给了新的变量，它仍有全都是原来数组的视图。

【示例】切片数组的复制

```
a=np.arange(1,13).reshape(3,4)
sub_array=a[:2,:2]
sub_array[0][0]=1000
print(a)
print(sub_array)
print('copy'*20)
sub_array=np.copy(a[:2,:2])
sub_array[0][0]=2000
print(a)
print(sub_array)
```

执行结果如图所示：



改变数组的维度

处理数组的一项重要工作就是改变数组的维度，包含提高数组的维度和降低数组的维度，还包括数组的转置。Numpy 提供的大量 API 可以很轻松地完成这些数组的操作。例如，通过 `reshape` 方法可以将一维数组变成二维、三维或者多维数组。通过 `ravel` 方法或 `flatten` 方法可以将多维数组变成一维数组。改变数组的维度还可以直接设置 Numpy 数组的 `shape` 属性（元组类型），通过 `resize` 方法也可以改变数组的维度。

【示例】切片数组的复制

```
import numpy as np

#创建一维的数组

a=np.arange(24)

print(a)

print('数组 a 的维度: ',a.shape)

print('-'*30)


#使用 reshape 将一维数组变成三维数组

b=a.reshape(2,3,4)

print(b)

print('数组 b 的维度: ',b.shape)

print('-'*30)
```

```

#将 a 变成二维数组
c=a.reshape(3,8)
print(c)
print('数组 c 的维度: ',c.shape)
print('-'*30)

#使用 ravel 函数将三维的 b 变成一维的数组
a1=b.ravel()
print(a1)
print('-'*30)

#使用 flatten 函数将二维的 c 变成一维的数组
a2=c.flatten()
print(a2)
print('-'*30)

```

数组的拼接

水平数组组合

通过 `hstack` 函数可以将两个或多个数组水平组合起来形成一个数组,那么什么叫做数组的水平组合。现在有两个 2×3 的数组 A 和 B。

数组 A

```

0  1  2
3  4  5

```

数组 B

```

6  7  8
9 10 11

```

使用 `hstack` 函数将两个数组水平组合的代码如下:

```
hstack(A,B)
```

返回的结果:

```

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

```

可以看到,数组 A 和数组 B 在水平方向首尾连接了起来,形成了一个新的数组。这就

是数组的水平组合。多个数组进行水平组合的效果类似。但数组水平组合必须要满足一个条件，就是所有参与水平组合的数组的行数必须相同，否则进行水平组合会抛出异常。

垂直数组组合

通过 `vstack` 函数可以将两个或多个数组垂直组合起来形成一个数组，那么什么叫数组的垂直组合呢？现在以两个 2×3 的数组 A 和 B 为例

数组 A

0 1 2

3 4 5

数组 B

6 7 8

9 10 11

使用 `vstack` 函数将两个数组垂直组合代码：`vstack(A,B)`

运行的结果是：

0 1 2

3 4 5

6 7 8

9 10 11

表 10-6 数组的拼接

函数	描述
<code>concatenate</code>	连接沿现有轴的数组序列
<code>hstack</code>	水平堆叠序列中的数组（列方向）
<code>vstack</code>	竖直堆叠序列中的数组（行方向）

`numpy.concatenate` 函数用于沿指定轴连接相同形状的两个或多个数组，格式如下：

```
numpy.concatenate((a1, a2, ...), axis)
```

参数说明：

`a1, a2, ...`：相同类型的数组

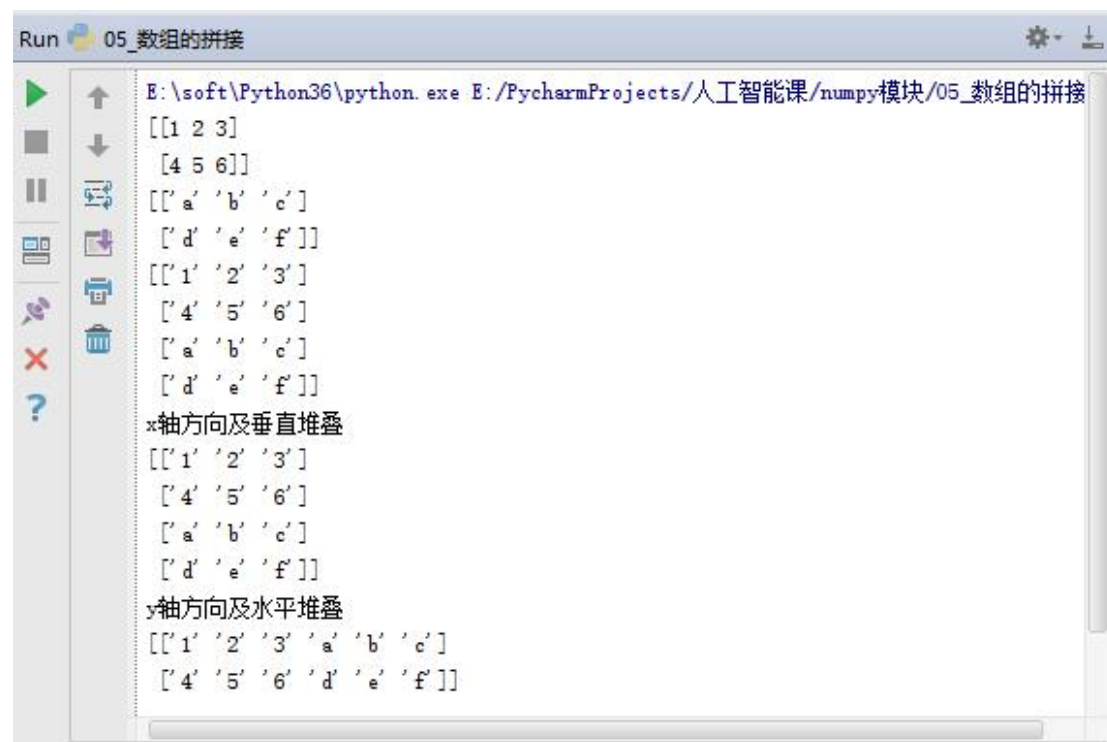
`axis`：沿着它连接数组的轴，默认为 0

【示例】`concatenate` 实现数组的拼接

```
a=np.array([[1,2,3],[4,5,6]])
print(a)
b=np.array([[ 'a','b','c'],[ 'd','e','f']])
print(b)
print(np.concatenate([a,b]))
```

```
print('垂直方向拼接 相当于 vstack')
print(np.concatenate([a,b],axis=0))
print('水平方向拼接 相当于 hstack')
print(np.concatenate([a,b],axis=1))
```

执行结果如图所示：



```
Run 05_数组的拼接
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/05_数组的拼接
[[1 2 3]
 [4 5 6]]
[['a' 'b' 'c']
 ['d' 'e' 'f']]
[['1' '2' '3']
 ['4' '5' '6']]
[['a' 'b' 'c']
 ['d' 'e' 'f']]
x轴方向及垂直堆叠
[['1' '2' '3']
 ['4' '5' '6']
 ['a' 'b' 'c']
 ['d' 'e' 'f']]
y轴方向及水平堆叠
[['1' '2' '3' 'a' 'b' 'c']
 ['4' '5' '6' 'd' 'e' 'f']]
```

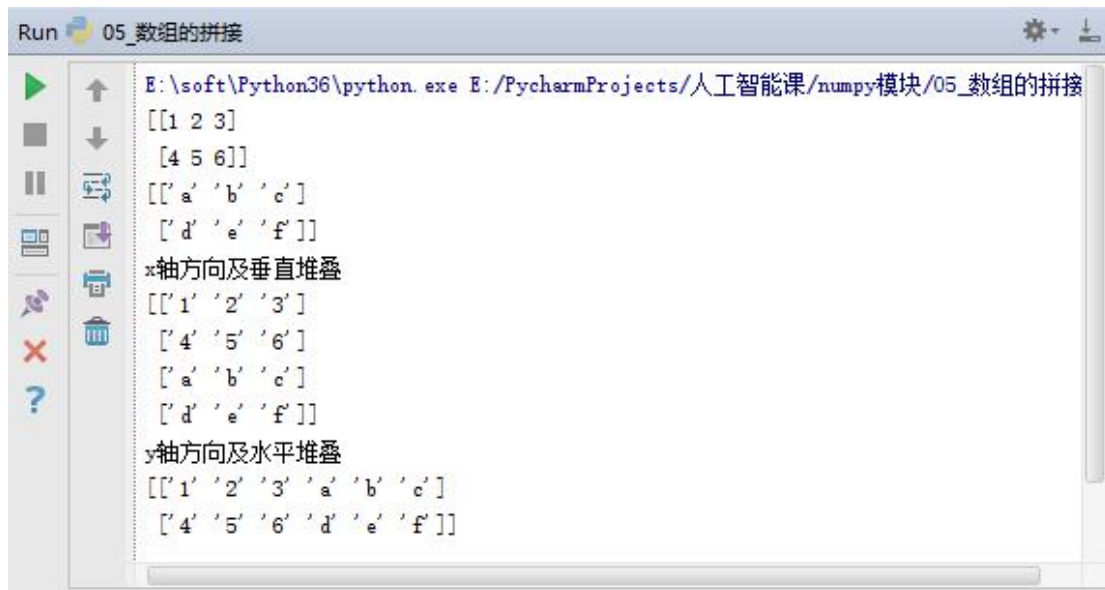
numpy.hstack 它通过水平堆叠来生成数组。

numpy.vstack 它通过垂直堆叠来生成数组。

【示例】vstack 与 hstack 实现数组的拼接

```
print('x 轴方向及垂直堆叠')
print(np.vstack([a,b]))
print('y 轴方向及水平堆叠')
print(np.hstack([a,b]))
```

执行结果如图所示：



The image shows a PyCharm Run window titled '05_数组的拼接'. The output text is as follows:

```

E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/05_数组的拼接
[[1 2 3]
 [4 5 6]]
[['a' 'b' 'c']
 ['d' 'e' 'f']]
x轴方向及垂直堆叠
[['1' '2' '3']
 ['4' '5' '6']
 ['a' 'b' 'c']
 ['d' 'e' 'f']]
y轴方向及水平堆叠
[['1' '2' '3' 'a' 'b' 'c']
 ['4' '5' '6' 'd' 'e' 'f']]

```

注意如果拼接的行和列数目不同，则会报错

【示例】三维数组的拼接

```

aa=np.arange(1,37).reshape(3,4,3)
print(aa)
bb=np.arange(101,137).reshape(3,4,3)
print(bb)
print('axis=0'*10)
print(np.concatenate((aa,bb),axis=0))  #6 4 3
print('axis=1'*10)
print(np.concatenate((aa,bb),axis=1))  #3,8,3
print('axis=2'*10)
print(np.concatenate((aa,bb),axis=2))  #3,4,6

```

axis=0 可以使用 vstack 替换

axis=1 可以使用 hstack 替换

axis=2 可以使用 dstack 替换

数组的分隔

split 分隔

numpy.split 函数沿特定的轴将数组分割为子数组，格式如下：

```
numpy.split(ary, indices_or_sections, axis)
```

参数说明：

ary: 被分割的数组

indices_or_sections: 如果是一个整数，就用该数平均切分，如果是一个数组，为沿轴切分的位置。

axis: 沿着哪个维度进行切向，默认为 0，横向切分。为 1 时，纵向切分。

【示例】split 分隔一维数组

```
import numpy as np
x=np.arange(1,9)
a=np.split(x,4)
print(a)
print(a[0])
print(a[1])
print(a[2])
print(a[3])

#传递数组进行分隔
b=np.split(x,[3,5])
print(b)
```

运行结果图：

```
Run 06数组的分隔
[array([1, 2]), array([3, 4]), array([5, 6]), array([7, 8])]
[1 2]
[3 4]
[5 6]
[7 8]
[array([1, 2, 3]), array([4, 5]), array([6, 7, 8])]
```

【示例】split 分隔二维数组

```
#导入 numpy
```

```
import numpy as np
#创建两个数组
a=np.array([[1,2,3],[4,5,6],[11,12,13],[14,15,16]])
print('axis=0 垂直方向 平均分')
r=np.split(a,2,axis=0)
print(r[0])
print(r[1])

print('axis=1 水平方向 按位置分隔')
r=np.split(a,[2],axis=1)
print(r)
```

水平分隔

分隔数组是组合数组的逆过程，与组合数组一样，分隔数组也分为水平分隔数组和垂直分隔数组。水平分隔数组与水平组合数组对应。水平组合数组是将两个或多个数组水平进行收尾相接，而水平分隔数组是将已经水平组合到一起的数组再分开。

使用 `hsplit` 函数可以水平分隔数组，该函数有两个参数，第 1 个参数表示待分隔的数组，第 2 个参数表示要将数组水平分隔成几个小数组，现在先来看一个例子。下面是一个 2*6 的二维数组 X

数组 X

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

使用如下代码对 X 数组进行分隔

```
np.hsplit(X,2)
```

分隔的结果如下：

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

很明显，将数组 X 分隔成了列数相同的两个数组。现在使用下面的代码重新对数组 X 进行分隔。

```
np.hsplit(X,3)
```

分隔的结果如下：

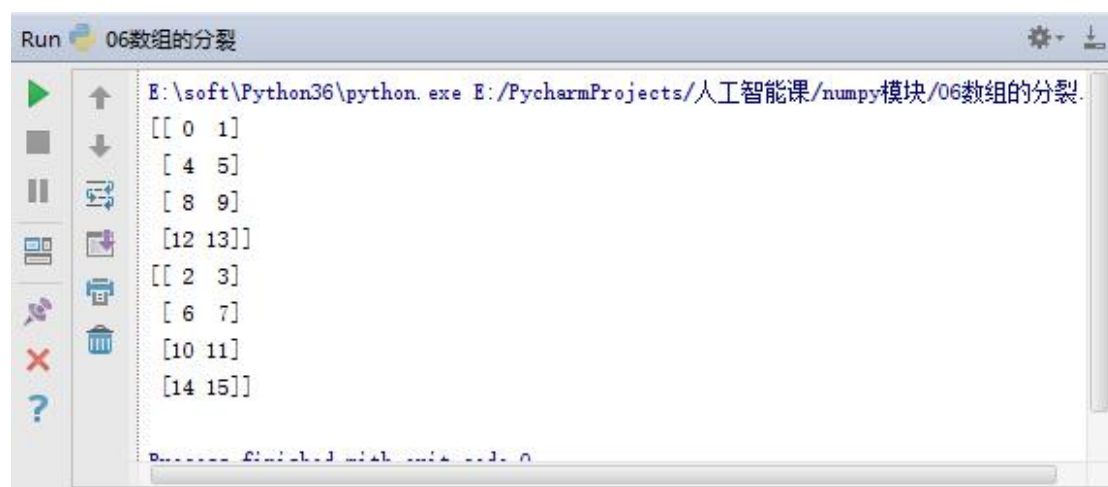
```
1  2  3  4      5  6
7  8  9 10      11 12
```


现在讲数组 X 分隔成了 3 个列数都为 2 的数组，但要是使用 `hsplit(X,4)` 分隔数组 X 就会抛出异常，这是因为数组 X 是没有办法被分隔成列数相同的 4 个数组的，所以使用 `hsplit` 函数分隔数组的一个规则就是第 2 个参数值必须可以整除待分隔数组的列数。

【示例】hsplit

```
grid=np.arange(16).reshape(4,4)
a,b=np.hsplit(grid,2)
print(a)
print(b)
```

运行结果图：



垂直分隔数组

垂直分隔数组是垂直组合数组的逆过程。垂直组合数组是将两个或多个数组垂直进行首尾相接，而垂直分隔数组是将已经垂直组合到一起的数组再分开。

使用 `vsplit` 函数可以垂直分隔数组，该函数有两个参数，第 1 个参数表示待分隔的数组，第 2 个参数表示将数组垂直分隔成几个小数组。示例如下，一个 4*3 的二维数组 X。

数组 X

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

现在使用如下的代码对数组 X 进行分隔：

```
vsplit(X,2)
```

分隔后的结果如下：

```
1  2  3
4  5  6
```

7 8 9

10 11 12

很明显，将数组 X 分隔成了行数相同的两个数组。现在使用下面的代码重新对数组 X 进行分隔。

```
vsplit(X,4)
```

分隔后的结果：

0 1 2

1 2 3

2 5 6

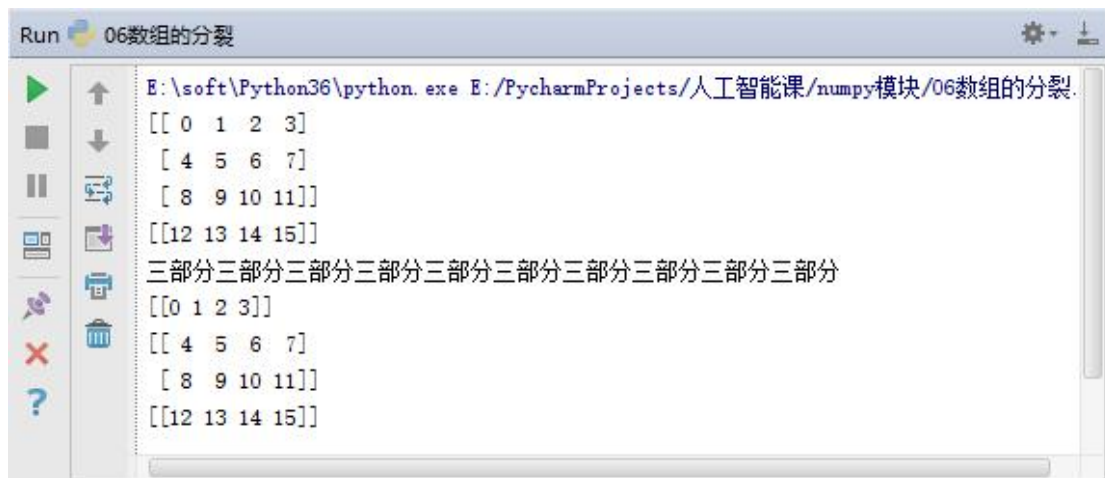
7 8 9

10 11 12

【示例】vsplit

```
grid=np.arange(16).reshape(4,4)
a,b=np.vsplit(grid,[3])
print(a)
print(b)
print('三部分'*10)
a,b,c=np.vsplit(grid,[1,3])
print(a)
print(b)
print(c)
```

运行结果图：



```
Run 06数组的分裂
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/06数组的分裂.
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
三部分三部分三部分三部分三部分三部分三部分三部分三部分
[[0 1 2 3]]
[[ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

【示例】transpose 进行转换

```
#transpose 进行转置
```

```
#二维转置
a=np.arange(1,13).reshape(2,6)
print('原数组 a')
print(a)
print('转置后的数组')
print(a.transpose())

#多维数组转置
aaa=np.arange(1,37).reshape(1,3,3,4)
#将 1,3,3,4 转换为 3,3,4,1
print(np.transpose(aaa,[1,2,3,0]).shape)
```

算术函数

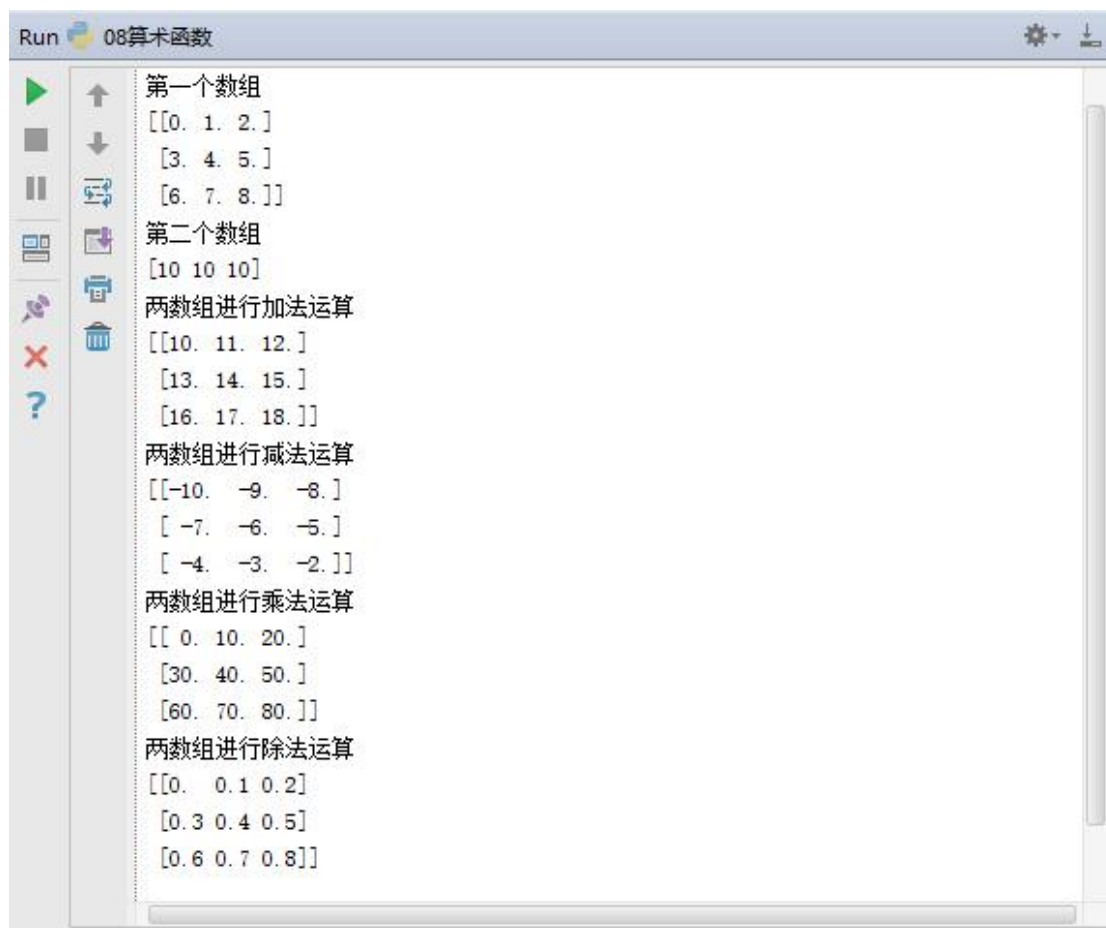
如果参与运算的两个对象 都是 `ndarray`，并且形状相同，那么会对位彼此之间进行（+ - * /）运算。NumPy 算术函数包含简单的加减乘除：`add()`，`subtract()`，`multiply()` 和 `divide()`。

【示例】算术函数的使用

```
a=np.arange(9,dtype=np.float).reshape(3,3)
b=np.array([10,10,10])
print('第一个数组')
print(a)
print('第二个数组')
print(b)
print('两数组进行加法运算 add')
print(np.add(a,b))
print('两数组进行加法运算+')
print(a+b)
print('两数组进行减法运算 subtract')
print(np.subtract(a,b))
print('两数组进行减法运算-')
print(a-b)
print('两数组进行乘法运算 multiply')
print(np.multiply(a,b))
print('两数组进行乘法运算*')
print(a*b)
```

```
# print('两数组进行除法运算')
# print(np.divide(a,b))
```

运行结果图：



【示例】通用函数指定输出结果的用法

```
# 通用函数指定输出结果的用法
x = np.arange(5)
y = np.empty(5)
print(y)
np.multiply(x, 10, out=y)
print(y)
```

运行结果图：

```
[ 0. 10. 20. 30. 40.]
```

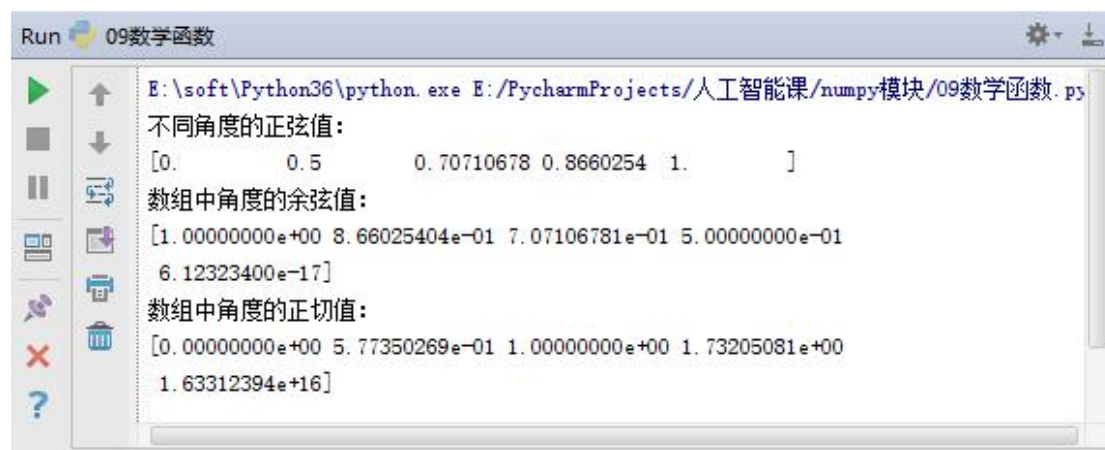
数学函数

NumPy 提供了标准的三角函数：sin()、cos()、tan()。

【示例】三角函数的使用

```
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 ('数组中角度的余弦值: ')
print (np.cos(a*np.pi/180))
print ('数组中角度的正切值: ')
print (np.tan(a*np.pi/180))
```

运行结果图：



```
Run 09数学函数
E:\soft\Python36\python.exe E:/PycharmProjects/人工智能课/numpy模块/09数学函数.py
不同角度的正弦值:
[0.          0.5         0.70710678 0.8660254  1.         ]
数组中角度的余弦值:
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01
 6.12323400e-17]
数组中角度的正切值:
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00
 1.63312394e+16]
```

numpy.around() 函数返回指定数字的四舍五入值。

```
numpy.around(a,decimals)
```

参数说明：

a: 数组

decimals: 舍入的小数位数。默认值为 0。如果为负，整数将四舍五入到小数点左侧的位置

numpy.floor() 返回数字的下舍整数。

numpy.ceil() 返回数字的上入整数。

【示例】around、floor、ceil 函数的使用

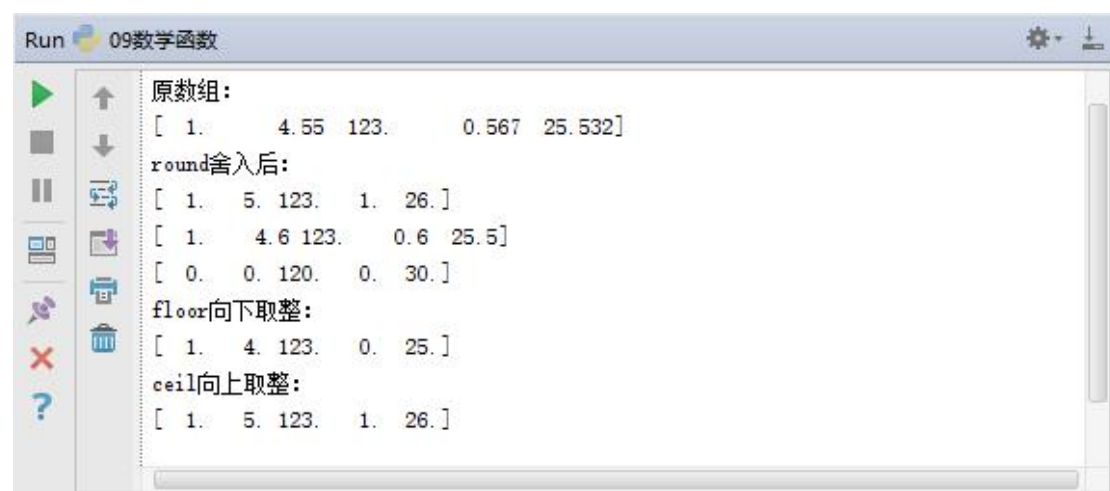
```
a = np.array([1.0,4.55, 123, 0.567, 25.532])
```

```

print ('原数组: ')
print (a)
print ('round 舍入后: ')
print (np.around(a))
print (np.around(a, decimals = 1))
print (np.around(a, decimals = -1))
print('floor 向下取整: ')
print(np.floor(a))
print('ceil 向上取整: ')
print(np.ceil(a))

```

运行结果图：



```

Run 09数学函数
原数组:
[ 1.    4.55 123.    0.567 25.532]
round舍入后:
[ 1.    5. 123.    1.  26.]
[ 1.    4.6 123.    0.6 25.5]
[ 0.    0. 120.    0.  30.]
floor向下取整:
[ 1.    4. 123.    0.  25.]
ceil向上取整:
[ 1.    5. 123.    1.  26.]

```

聚合函数

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。具体如下：

函数名	说明
np.sum()	求和
np.prod()	所有元素相乘
np.mean()	平均值
np.std()	标准差
np.var()	方差
np.median()	中数
np.power()	幂运算
np.sqrt()	开方
np.min()	最小值

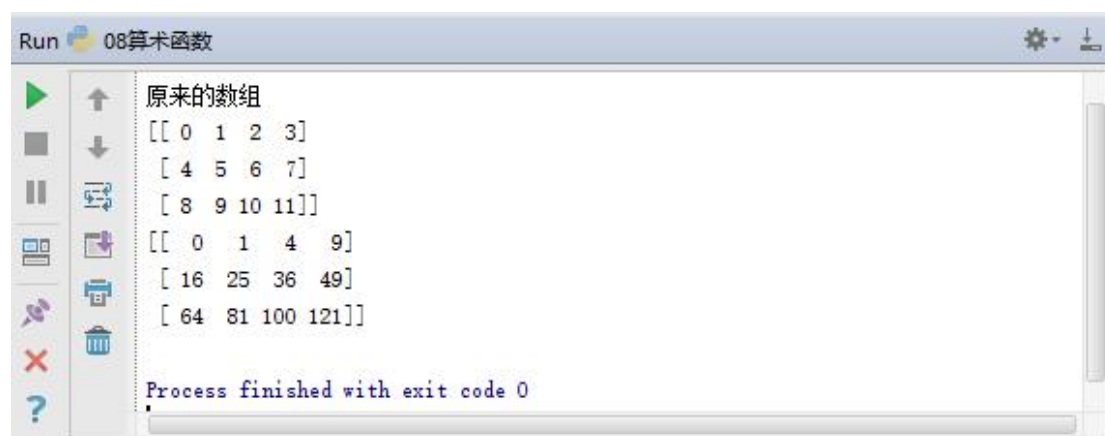
np. max()	最大值
np. argmin()	最小值的下标
np. argmax()	最大值的下标
np. inf	无穷大
np. exp(10)	以 e 为底的指数
np. log(10)	对数

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。

【示例】power 函数的使用

```
a=np.arange(12).reshape(3,4)
print('原来的数组')
print(a)
print(np.power(a,2))
```

运行结果图：



【示例】power 函数指定输出结果的用法

```
x=np.arange(5)
y=np.zeros(10)
np.power(2,x,out=y[:5])
print('power:',x)
print(y)
power: [0 1 2 3 4]
[ 1.  2.  4.  8. 16.  0.  0.  0.  0.  0.]
```

运行结果图：

```
power: [0 1 2 3 4]
```



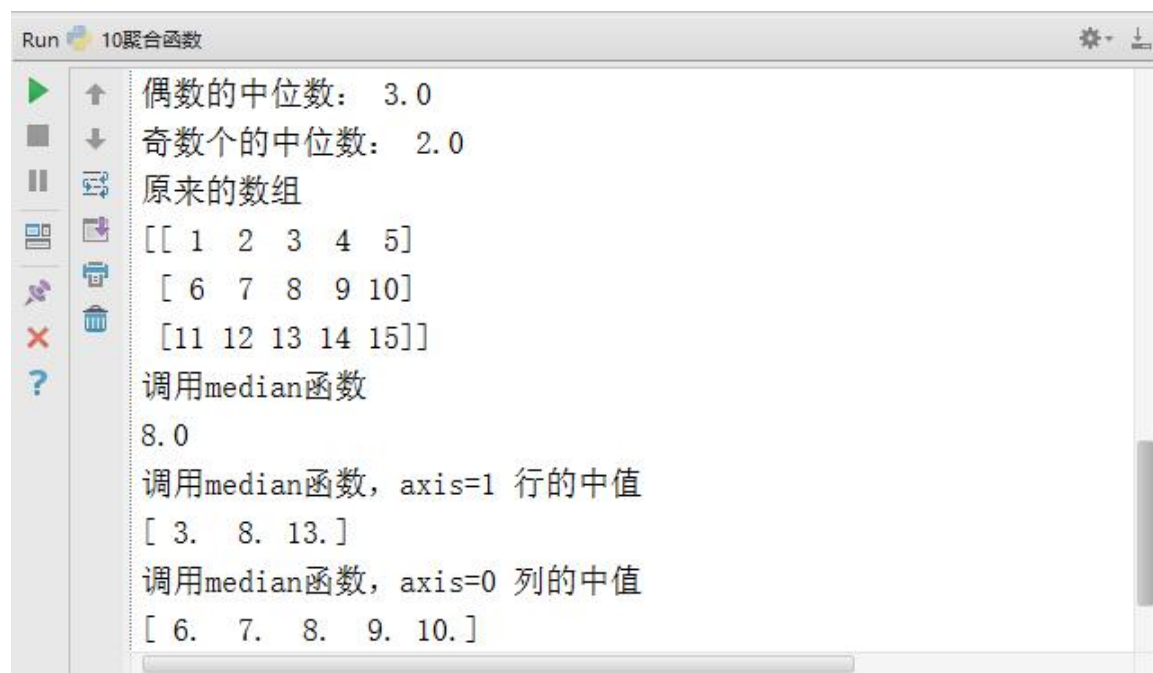
```
[ 1.  2.  4.  8. 16.  0.  0.  0.  0.  0.]
```

【示例】numpy.median ()函数的使用

```
a=np.array([4,2,1,5])
#计算偶数的中位数
print('偶数的中位数: ',np.median(a))
a=np.array([4,2,1])
print('奇数个的中位数: ',np.median(a))

a=np.arange(1,16).reshape(3,5)
print('原来的数组')
print(a)
print('调用 median 函数')
print(np.median(a))
print('调用 median 函数, axis=1 行的中值')
print(np.median(a,axis=1))
print('调用 median 函数, axis=0 列的中值')
print(np.median(a,axis=0))
```

运行结果图：



```
Run 10 聚合函数
↑ 偶数的中位数: 3.0
↓ 奇数个的中位数: 2.0
|| 原来的数组
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
调用median函数
8.0
调用median函数, axis=1 行的中值
[ 3.  8. 13.]
调用median函数, axis=0 列的中值
[ 6.  7.  8.  9. 10.]
```

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总和除以元素的数量。

【示例】numpy.mean ()函数的使用

```
a=np.arange(1,11).reshape(2,5)
print('原来的数组')
print(a)
print('调用 mean 函数')
print(np.mean(a))
print('调用 mean 函数 axis=0 列')
print(np.mean(a,axis=0))
print('调用 mean 函数 axis=1 行')
print(np.mean(a,axis=1))
```

运行结果图：



The screenshot shows a Jupyter Notebook interface with the following output:

```
Run 14 算术运算
原来的数组
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
调用mean函数
5.5
调用mean函数 axis=0 列
[3.5 4.5 5.5 6.5 7.5]
调用mean函数 axis=1 行
[3. 8.]
```