Table of Contents

In [1]:
```
#全部行都能输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import numpy as np
```

Numpy官网：http://www.numpy.org/ (http://www.numpy.org/)

1 数组的变形

数组变形最灵活的实现方式是通 过 reshape() 函数来实现。例如，如果你希望将数字 1~9 放入一个 3×3 的矩阵中，可以采用如下方法：

In [2]:
```
a=np.arange(1,10)
a.reshape(3,3)
```

Out[2]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

1.1 ndarray.reshape()

In [3]:
```
# 为了确保大家都能生成一样的数组，我们先设置随机数种子
np.random.seed(100)
x1 = np.random.randint(10, size=6) # 一维数组
x2 = np.random.randint(10, size=(3, 4)) # 二维数组
x3 = np.random.randint(10, size=(3, 4, 5)) # 三维数组

x1
x2
x3
```

Out[3]:
```
array([8, 8, 3, 7, 7, 0])
```

Out[3]:
```
array([[4, 2, 5, 2],
       [2, 2, 1, 0],
       [8, 4, 0, 9]])
```

Out[3]:
```
array([[[6, 2, 4, 1, 5],
        [3, 4, 4, 3, 7],
        [1, 1, 7, 7, 0],
        [2, 9, 9, 3, 2]],

       [[5, 8, 1, 0, 7],
        [6, 2, 0, 8, 2],
        [5, 1, 8, 1, 5],
        [4, 2, 8, 3, 5]],

       [[0, 9, 3, 6, 3],
        [4, 7, 6, 3, 9],
        [0, 4, 4, 5, 7],
        [6, 6, 2, 4, 2]]])
```

注意：使用ndarray.reshape()变形不会改变x1原有的数组形状，而是生成一个新的对象：

In [4]:
```
x1.reshape((3, 2))
```

Out[4]:
```
array([[8, 8],
       [3, 7],
       [7, 0]])
```

In [5]:
```
x1
```

Out[5]:
```
array([8, 8, 3, 7, 7, 0])
```

同样地，可以对3维数组进行变形，此时注意数组中元素的总和即可：

In [6]:
```
x3=np.array([[[6, 2, 4, 1, 5],
              [3, 4, 4, 3, 7],
              [1, 1, 7, 7, 0],
              [2, 9, 9, 3, 2]],

             [[5, 8, 1, 0, 7],
              [6, 2, 0, 8, 2],
              [5, 1, 8, 1, 5],
              [4, 2, 8, 3, 5]],

             [[0, 9, 3, 6, 3],
              [4, 7, 6, 3, 9],
              [0, 4, 4, 5, 7],
              [6, 6, 2, 4, 2]]])
x3.size
```

Out[6]:
```
60
```

In [7]:
```
#既然x3的元素个数是60，三维的三个数字相乘等于60即可
x3.reshape(2,3,10)
```

Out[7]:
```
array([[[6, 2, 4, 1, 5, 3, 4, 4, 3, 7],
        [1, 1, 7, 7, 0, 2, 9, 9, 3, 2],
        [5, 8, 1, 0, 7, 6, 2, 0, 8, 2]],

       [[5, 1, 8, 1, 5, 4, 2, 8, 3, 5],
        [0, 9, 3, 6, 3, 4, 7, 6, 3, 9],
        [0, 4, 4, 5, 7, 6, 6, 2, 4, 2]]])
```

1.2 小技巧

技巧：在使用 reshape 时，可以将其中的一个维度指定为 -1，Numpy 会自动计算出它的真实值

In [8]:
```
x3
```

Out[8]:
```
array([[[6, 2, 4, 1, 5],
        [3, 4, 4, 3, 7],
        [1, 1, 7, 7, 0],
        [2, 9, 9, 3, 2]],

       [[5, 8, 1, 0, 7],
        [6, 2, 0, 8, 2],
        [5, 1, 8, 1, 5],
        [4, 2, 8, 3, 5]],

       [[0, 9, 3, 6, 3],
        [4, 7, 6, 3, 9],
        [0, 4, 4, 5, 7],
        [6, 6, 2, 4, 2]]])
```

In [10]:
```
x3.reshape((2, 3,-1))    #当其中一个维度你不知道是多少时，可以用-1来代替
```

Out[10]:
```
array([[[6, 2, 4, 1, 5, 3, 4, 4, 3, 7],
        [1, 1, 7, 7, 0, 2, 9, 9, 3, 2],
        [5, 8, 1, 0, 7, 6, 2, 0, 8, 2]],

       [[5, 1, 8, 1, 5, 4, 2, 8, 3, 5],
        [0, 9, 3, 6, 3, 4, 7, 6, 3, 9],
        [0, 4, 4, 5, 7, 6, 6, 2, 4, 2]]])
```

In [14]:
```
x3.reshape((-1, 2,10))    #当其中一个维度你不知道是多少时，可以用-1来代替
```

Out[14]:
```
array([[[6, 2, 4, 1, 5, 3, 4, 4, 3, 7],
        [1, 1, 7, 7, 0, 2, 9, 9, 3, 2]],

       [[5, 8, 1, 0, 7, 6, 2, 0, 8, 2],
        [5, 1, 8, 1, 5, 4, 2, 8, 3, 5]],

       [[0, 9, 3, 6, 3, 4, 7, 6, 3, 9],
        [0, 4, 4, 5, 7, 6, 6, 2, 4, 2]]])
```

1.3 ndarray.shape

ndarray.shape有两种功能：

- 第一种是查看数组形状：

In [15]:
```
x_01=np.array([8, 8, 3, 7, 7, 0])
x_01.shape
```

Out[15]:
```
(6,)
```

第二种是改变数组形状：

In [16]:
```
x_01

x_01.shape=(3,2)
x_01
```

Out[16]:
```
array([8, 8, 3, 7, 7, 0])
```

Out[16]:
```
array([[8, 8],
       [3, 7],
       [7, 0]])
```

1.4 ndarray.resize()

- 使用 resize 方法可以直接修改数组本身
- 作用和shape改变数组形状是一样的，即改变数组本身。

In [17]:
```
x2=np.array([[3, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])

x2.resize((2, 6))
x2
```

Out[17]:
```
array([[3, 5, 2, 4, 7, 6],
       [8, 8, 1, 6, 7, 7]])
```

1.5 ndarray.ravel()

- 数组的平铺。
- 不管多少维，全部铺开变成一维。

In [18]:
```
x2=np.array([[3, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])

x2.ravel()
```

Out[18]:
```
array([3, 5, 2, 4, 7, 6, 8, 8, 1, 6, 7, 7])
```

In [19]:
```
x3=np.array([[[6, 2, 4, 1, 5],
              [3, 4, 4, 3, 7],
              [1, 1, 7, 7, 0],
              [2, 9, 9, 3, 2]],

             [[5, 8, 1, 0, 7],
              [6, 2, 0, 8, 2],
              [5, 1, 8, 1, 5],
              [4, 2, 8, 3, 5]],

             [[0, 9, 3, 6, 3],
              [4, 7, 6, 3, 9],
              [0, 4, 4, 5, 7],
              [6, 6, 2, 4, 2]]])


x3.ravel()
```

Out[19]:
```
array([6, 2, 4, 1, 5, 3, 4, 4, 3, 7, 1, 1, 7, 7, 0, 2, 9, 9, 3, 2, 5, 8,
       1, 0, 7, 6, 2, 0, 8, 2, 5, 1, 8, 1, 5, 4, 2, 8, 3, 5, 0, 9, 3, 6,
       3, 4, 7, 6, 3, 9, 0, 4, 4, 5, 7, 6, 6, 2, 4, 2])
```

1.6 ndarry.T

In [20]:
```
x2=np.array([[3, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])

x2.T
```

Out[20]:
```
array([[3, 7, 1],
       [5, 6, 6],
       [2, 8, 7],
       [4, 8, 7]])
```

In [21]:
```
x2.shape
x2.T.shape
```

Out[21]:
```
(3, 4)
```

Out[21]:
```
(4, 3)
```

无论是ravel、reshape、T，它们都不会更改原有的数组形状，都是返回一个新的数组。

In [ ]:
```
```

2 数组的拼接

拼接或连接 NumPy 中的两个数组主要由np.concatenate、np.vstack 和 np.hstack 实现,此外还有dstack()

2.1 np.concatenate()

concatenate((a1, a2, ...), axis=0, out=None) 沿现有轴加入一系列数组。

- a1，a2，...：array_like的序列
  - 除尺寸外，阵列必须具有相同的形状
- axis：int，可选
  - 数组将连接的轴。如果axis为None，数组在使用前是扁平的。默认值为0。
- out：ndarray，可选
  - 如果提供，则放置结果的目的地。形状必须是正确，匹配连接将返回的。

In [25]:
```
x = np.array([34, 43, 12])
y = np.array([343, 34, 676])

np.concatenate((x, y))
```

Out[25]:
```
array([ 34,  43,  12, 343,  34, 676])
```

In [26]:
```
x = np.array([[1, 2, 3],[4,5,6]])
y = np.array([[11,22,33],[44,55,66]])

np.concatenate((x, y))  #默认按照第一个轴合并，即axis=0
# np.concatenate((x, y),axis=0)
```

Out[26]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [11, 22, 33],
       [44, 55, 66]])
```

In [30]:
```
np.concatenate([x, y], axis=0)
```

Out[30]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [11, 22, 33],
       [44, 55, 66]])
```

同样地,我们观察一下三维数组合并规律:

In [23]:
```
a1=np.ones((2,2,2),dtype='int')
a2=np.zeros((2,2,2),dtype='int')
a1
a2
```

Out[23]:
```
array([[[1, 1],
        [1, 1]],

       [[1, 1],
        [1, 1]]])
```

Out[23]:
```
array([[[0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0]]])
```

In [24]:
```
np.concatenate([a1,a2],axis=0)  #axis=1、2有什么区别?
```

Out[24]:
```
array([[[1, 1],
        [1, 1]],

       [[1, 1],
        [1, 1]],

       [[0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0]]])
```

2.2 np.vstack()

- 垂直堆叠数组。
- 这相当于1-D数组后沿第一轴的连接。
- 对于最多3维的数组，此函数最有意义。
- np.vstack（tup）
  - tup：ndarrays的序列
  - 除了第一个轴之外，阵列必须具有相同的形状。
  - 1-D阵列必须具有相同的长度。

In [39]:
```
x = np.array([[1, 2, 3],[4,5,6]])
y = np.array([[11,22,33],[44,55,66],[44,55,66]])

np.vstack((x, y))
```

Out[39]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [11, 22, 33],
       [44, 55, 66],
       [44, 55, 66]])
```

In [40]:
```
#作用和concatenate((...),axis=0)一样
np.concatenate([x,y],axis=0)
```

Out[40]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [11, 22, 33],
       [44, 55, 66],
       [44, 55, 66]])
```

2.3 np.hstack()

- 水平堆叠数组。
- 这相当于1-D数组后沿第二轴的连接。
- 对于最多3维的数组，此函数最有意义。
- np.hstack（tup）
  - tup：ndarrays的序列
  - 除了第二个轴之外，阵列必须具有相同的形状。
  - 1-D阵列必须具有相同的长度。

In [41]:
```
x = np.array([[1, 2, 3,66],[4,5,6,66]])
y = np.array([[11,22,33],[44,55,66]])

np.hstack((x, y))
```

Out[41]:
```
array([[ 1,  2,  3, 66, 11, 22, 33],
       [ 4,  5,  6, 66, 44, 55, 66]])
```

In [42]:
```
#作用和concatenate((...),axis=1)一样
np.concatenate([x,y],axis=1)
```

Out[42]:
```
array([[ 1,  2,  3, 66, 11, 22, 33],
       [ 4,  5,  6, 66, 44, 55, 66]])
```

2.4 np.dstack()

- np.dstack(tup)
  - 按顺序深度（沿第三轴）堆叠阵列。
  - 对于最多3维的数组，此函数最有意义。
  - tup：数组序列
    - 除了第三个轴之外，阵列必须具有相同的形状。
    - 1-D或2-D阵列必须具有相同的形状。

In [43]:
```
x = np.array([[1, 2, 3],[4,5,6]])
y = np.array([[11,22,33],[44,55,66]])

x
y

np.dstack((x, y))
```

Out[43]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Out[43]:
```
array([[11, 22, 33],
       [44, 55, 66]])
```

Out[43]:
```
array([[[ 1, 11],
        [ 2, 22],
        [ 3, 33]],

       [[ 4, 44],
        [ 5, 55],
        [ 6, 66]]])
```

3 数据的分裂

- 将一个数组分成几个较小的数组
- 既然可以将多个数组进行对堆叠，自然也可以将一个数组拆分成多个小数组。
- 和拼接有以下对应关系：
  - split --->concatenate
  - hsplit--->hstack
  - vsplit--->vstack
- 使用hsplit，可以沿其水平轴拆分数组，通过指定要返回的均匀划分的数组数量，或通过指定要在其后进行划分的列：

3.1 np.split()

- 语法：np.split（ary，indices_or_sections，axis = 0）
- 作用：将数组拆分为多个子数组。

- ary：ndarray。
- indices_or_sections：int或1-D数组。
  - 如果indices_or_sections是一个整数N，则数组将被分割沿着'轴'进入N个相等的数组。
  - 如果indices_or_sections是排序整数的1-D数组，则为条目指示数组被分割的“轴”的位置。
  - 如果索引超过沿“轴”的数组维度，相应地返回一个空的子数组。

- axis：int，可选。要拆分的轴，默认为0。

如果indices_or_sections是一个整数N，则数组将被分割沿着'轴'进入N个相等的数组。

In [49]:
```
a=np.ones((2,2),dtype='int')
b=np.zeros((2,2),dtype='int')

a
b
```

Out[49]:
```
array([[1, 1],
       [1, 1]])
```

Out[49]:
```
array([[0, 0],
       [0, 0]])
```

首先，我们用np.concatenate()按照第一个轴进行合并：

In [50]:
```
c=np.concatenate([a,b],axis=0)
c
```

Out[50]:
```
array([[1, 1],
       [1, 1],
       [0, 0],
       [0, 0]])
```

然后我们用np.split()按照相同的第一个轴进行拆分:

In [55]:
```
np.split(c,2,axis=0)  #返回的对象其实是子阵列表
```

Out[55]:
```
[array([[1, 1],
        [1, 1]]), array([[0, 0],
        [0, 0]])]
```

返回的对象其实是子阵列表,因此是可以进行索引操作的:

In [56]:
```
np.split(c,2,axis=0)[0]
```

Out[56]:
```
array([[1, 1],
       [1, 1]])
```

如果indices_or_sections是排序整数的1-D数组，则为条目指示数组被分割的“轴”的位置。

In [31]:
```
d=np.array([[1, 1],
            [1, 1],
            [2, 2],
            [2, 2],
            [3, 3],
            [3, 3],
            [4, 4],
            [4, 4]])
```

In [32]:
```
np.split(d,[5],axis=0)
```

Out[32]:
```
[array([[1, 1],
        [1, 1],
        [2, 2],
        [2, 2],
        [3, 3]]), array([[3, 3],
        [4, 4],
        [4, 4]])]
```

如果要将数组切分成三份呢?

In [33]:
```
np.split(d,[3,6],axis=0)  #注意这里不要写切片的冒号
```

Out[33]:
```
[array([[1, 1],
        [1, 1],
        [2, 2]]), array([[2, 2],
        [3, 3],
        [3, 3]]), array([[4, 4],
        [4, 4]])]
```

如果要将数组切分成四份呢?

In [34]:
```
np.split(d,[2,4,6],axis=0)
```

Out[34]:
```
[array([[1, 1],
        [1, 1]]), array([[2, 2],
        [2, 2]]), array([[3, 3],
        [3, 3]]), array([[4, 4],
        [4, 4]])]
```

如果按照axis=1来划分呢? 会有什么现象?

In [35]:
```
d
np.split(d,2,axis=1)
```

Out[35]:
```
array([[1, 1],
       [1, 1],
       [2, 2],
       [2, 2],
       [3, 3],
       [3, 3],
       [4, 4],
       [4, 4]])
```

Out[35]:
```
[array([[1],
        [1],
        [2],
        [2],
        [3],
        [3],
        [4],
        [4]]), array([[1],
        [1],
        [2],
        [2],
        [3],
        [3],
        [4],
        [4]])]
```

3.2 np.vsplit()

- 语法：np.vsplit（ary，indices_or_sections）
- 作用：将数组垂直拆分为多个子数组（按行）。
- 'vsplit'相当于'split'用'axis = 0'来'拆分'，数组总是沿着第一个轴分开,无论数组尺寸如何。

In [73]:
```
c

np.vsplit(c,2)
np.split(c,2,axis=0)
```

Out[73]:
```
array([[1, 1],
       [1, 1],
       [0, 0],
       [0, 0]])
```

Out[73]:
```
[array([[1, 1],
        [1, 1]]), array([[0, 0],
        [0, 0]])]
```

Out[73]:
```
[array([[1, 1],
        [1, 1]]), array([[0, 0],
        [0, 0]])]
```

3.3 np.hsplit()

- 语法：np.hsplit（ary，indices_or_sections）
- 作用：将数组水平拆分为多个子数组（按列）。
- 'hsplit'相当于'split'用'axis = 1'来'拆分'，数组总是沿着第二个分开,轴与阵列尺寸无关。

In [76]:
```
c
np.hsplit(c,2)   #返回的对象其实是子阵列表
np.split(c,2,axis=1)
```

Out[76]:
```
array([[1, 1],
       [1, 1],
       [0, 0],
       [0, 0]])
```

Out[76]:
```
[array([[1],
        [1],
        [0],
        [0]]), array([[1],
        [1],
        [0],
        [0]])]
```

Out[76]:
```
[array([[1],
        [1],
        [0],
        [0]]), array([[1],
        [1],
        [0],
        [0]])]
```

3.4 练习

In [91]:
```
np.random.seed(0)
ss=np.random.randint(1,30,(7,7))
ss
```

Out[91]:
```
array([[13, 16, 22,  1,  4, 28,  4],
       [ 8, 10, 20, 22, 19,  5, 24],
       [ 7, 25, 25, 13, 27,  2,  7],
       [ 8, 24, 15, 25, 18,  6, 26],
       [14,  9, 10, 21, 20, 17, 20],
       [ 6, 16, 16,  1, 19,  4, 25],
       [18, 20, 20, 20, 15,  8,  1]])
```

将上面的数组按照以下规则切割：

- 横向切，切割成三部分，比例：2：2：3
- 纵向切，切割成两部分，比例：3：4

In [ ]:
```

```