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
```

1 nddaray常用属性

1.1 np.random.seed()

np.random.seed(Num)可以设置一组种子值，确保每次程序执行的时候都可以生成同样的随机数组。

In [2]:
```
#不设定随机种子，每次随机生成的数组都不一样
x1 = np.random.randint(10, size=6) # 一维数组
x2 = np.random.randint(10, size=(3, 4)) # 二维数组

x1
x2
```

Out[2]:
```
array([4, 7, 8, 9, 7, 7])
```

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

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
a=np.random.seed(0)

x1 = np.random.randint(10, size=6) # 一维数组
x2 = np.random.randint(10, size=(3, 4)) # 二维数组

x1
x2
```

Out[3]:
```
array([5, 0, 3, 3, 7, 9])
```

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

1.2 ndarray.shape

现在我们想要查看数组是什么形状

In [4]:
```
x1.shape
x2.shape
```

Out[4]:
```
(6,)
```

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

1.3 ndarray.ndim

想要查看数组的维度

In [5]:
```
x1
x2

x1.ndim  #n-dimension的意思
x2.ndim
```

Out[5]:
```
array([5, 0, 3, 3, 7, 9])
```

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

Out[5]:
```
1
```

Out[5]:
```
2
```

1.4 ndarray.size

查看数组元素个数

In [6]:
```
x1.size
x2.size
```

Out[6]: 6

Out[6]: 12

2 数组的索引和切片

Python 中原生的数组就支持使用方括号（[ ]）进行索引和切片操作，Numpy 也同样具有这个强大的特性。

2.1 单个元素索引

数组的单元素索引操作的工作原理与其他标准Python序列一样。它是从0开始的，并且接受负索引来从数组的结尾进行索引。

In [7]:
```
x = np.arange(10)

x
```

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

In [8]:
```
x[0]

x[0:5]

x[-1:-5:-1]
```

Out[8]: 0

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

Out[8]: array([9, 8, 7, 6])

2.2 多维数组索引

与Python原生的列表、元组不同的是，Numpy数组支持多维数组的多维索引。

每一个逗号,代表索引的一个维度

In [9]:
```
x2=np.array([[3, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])
```

In [10]:
```
x2[0]

x2[0, 0]

x2[1, 1]
```

Out[10]:
```
array([3, 5, 2, 4])
```

Out[10]:
```
3
```

Out[10]:
```
6
```

也可以按照以往嵌套列表取元素的方法，一层一层取：

In [19]:
```
x2[0]

x2[0][0]

x2[1][1]
```

Out[19]:
```
array([3, 5, 2, 4])
```

Out[19]:
```
3
```

Out[19]:
```
6
```

二维以上的如何取？

In [21]:
```
x3=np.array([[[8, 1, 5, 9, 8],
              [9, 4, 3, 0, 3],
              [5, 0, 2, 3, 8],
              [1, 3, 3, 3, 7]],

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

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

如何取中间那块[0,0]？

In [22]:
```
x3[1]
x3[1][2]
x3[1][2][1:3]
```

Out[22]:
```
array([[0, 1, 9, 9, 0],
       [4, 7, 3, 2, 7],
       [2, 0, 0, 4, 5],
       [5, 6, 8, 4, 1]])
```

Out[22]:
```
array([2, 0, 0, 4, 5])
```

Out[22]:
```
array([0, 0])
```

In [23]:
```
x3[1]
x3[1,2]
x3[1,2,1:3]
```

Out[23]:
```
array([[0, 1, 9, 9, 0],
       [4, 7, 3, 2, 7],
       [2, 0, 0, 4, 5],
       [5, 6, 8, 4, 1]])
```

Out[23]:
```
array([2, 0, 0, 4, 5])
```

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

2.3 习题1

如何取出x3中的：

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

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

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

In [ ]:

2.4 习题2

如果需要截取的行和列数，并不是并排的，该怎么办？

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

如果你想取出上面数组的第二行，同时取出第一和第四列，该怎么取？

如何取出x2的第一行和第三行的第一列和第四列值？

In [ ]:

2.5 修改数组元素值

In [11]:
```
x=np.array([5, 0, 3, 3, 7, 9])

x[0] = 1999
x
```

Out[11]:
```
array([1999,    0,    3,    3,    7,    9])
```

In [12]:
```
x2=np.array([[ 3,  5,  2,  4],
             [ 7,  6,  8,  8],
             [ 1,  6, 77, 77]])

x2[2,3]=88.1  #数据类型会转化

x2
```

Out[12]:
```
array([[ 3,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6, 77, 88]])
```

2.6 切片

可以使用切片和步长来截取不同长度的数组，使用方式与Python原生的对列表和元组的方式相同。

语法和之前学过的列表的切片是一样的

```
x[start:stop:step]
```

In [27]:
```
x=np.array([[3, 5, 2, 4],
            [7, 6, 8, 8],
            [1, 6, 7, 7]])
x
```

Out[27]:
```
array([[3, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
```

In [28]:
```
#取出第一行
x[0,:]
```

Out[28]:
```
array([3, 5, 2, 4])
```

In [29]:
```
#取出第一列
x[:,0]
```

Out[29]:
```
array([3, 7, 1])
```

注意，二维数组切片的取法，下面两种方法的差异:

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

In [31]:
```
#x[第一维度,第二维度]
x[:2, :2]      #先取行，后取列

x[:2]
x[:2][:2]      #取两次都是取行数
```

Out[31]:
```
array([[3, 5],
       [7, 6]])
```

Out[31]:
```
array([[3, 5, 2, 4],
       [7, 6, 8, 8]])
```

Out[31]:
```
array([[3, 5, 2, 4],
       [7, 6, 8, 8]])
```

2.7 习题3

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

如何通过索引和切片的方式，分别取出：

```
[[3,5]
 [7,6]]
```

和

```
[[3,2]
[1,7]]
```

In [ ]:
```

```

3 复制和视图

3.1 简单赋值

简单赋值不会创建数组对象或其数据的拷贝。

In [35]:
```
a = np.arange(6)
b = a

id(a)
id(b) # id(a)和id(b)结果相同
```

Out[35]: 2574643468048

Out[35]: 2574643468048

In [36]:
```
b[1] =  22

b
a
```

Out[36]: array([0, 22, 2, 3, 4, 5])

Out[36]: array([0, 22, 2, 3, 4, 5])

3.2 视图

a.view()

- 不同的数组对象可以共享相同的数据。view方法创建一个新数组对象，该对象看到相同的数据。
- 新数组数据更改后，会影响原始数据。

In [37]:
```
a = np.arange(0, 12, 1).reshape(6, 2)
c = a.view()

c
```

Out[37]:
```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

In [38]:
```
id(a)
id(c)
```

Out[38]: 2574643470208

Out[38]: 2574643469888

新数组数据更改后，会影响原始数据。类似于原生python中的浅复制：

In [41]:
```
a[0,1]=11

a
c  #c = a.view()
```

Out[41]:
```
array([[ 0, 11],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

Out[41]:
```
array([[ 0, 11],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

3.3 数组副本

也可以通过.copy()方法创建一个副本，类似于列表知识章节中提到的深复制。

In [42]:
```
#复原x3
a = np.random.randint(2, size=(3, 4)) # 二维数组
b = a.copy()

a
b
```

Out[42]:
```
array([[0, 0, 1, 1],
       [1, 1, 0, 1],
       [0, 1, 0, 1]])
```

Out[42]:
```
array([[0, 0, 1, 1],
       [1, 1, 0, 1],
       [0, 1, 0, 1]])
```

In [43]:
```
id(a)
id(b)
```

Out[43]:
```
2574643470048
```

Out[43]:
```
2574643509808
```

In [44]:
```
a[0, 0]= 8888

a
b
```

Out[44]:
```
array([[8888,    0,    1,    1],
       [   1,    1,    0,    1],
       [   0,    1,    0,    1]])
```

Out[44]:
```
array([[0, 0, 1, 1],
       [1, 1, 0, 1],
       [0, 1, 0, 1]])
```

In [14]:
```
from copy import deepcopy
a=[1,2,3]

b=deepcopy(a)
a.append(4)

a
b
```

Out[14]: [1, 2, 3, 4]

Out[14]: [1, 2, 3]