第2章 科学计算库 NumPy

学习目标

- 认识 NumPy 数组对象，会创建 NumPy 数组
- 熟悉 ndarray 对象的数据类型，并会转换数据类型
- 掌握数组运算方式
- 掌握数组的索引和切片
- 会使用数组进行数据处理
- 熟悉线性代数模块和随机数模块的使用

NumPy 作为高性能科学计算和数据分析的基础包，它是本书介绍的其它重要数据分析工具的基础，掌握 NumPy 的功能及其用法，将有助于后续其他数据分析工具的学习。接下来，本章将带领大家学习 NumPy 的基本用法。

2.1 认识 NumPy 数组对象

NumPy 中最重要的一个特点就是其 N 维数组对象，即 ndarray（别名 array）对象，该对象具有矢量算术能力和复杂的广播能力，可以执行一些科学计算。不同于 Python 标准库，ndarray 对象拥有对高维数组的处理能力，这也是数值计算中缺一不可的重要特性。

ndarray 对象中定义了一些重要的属性，具体如表 2-1 所示。

表 2-1 ndarray 对象的常用属性

属性	具体说明
ndarray.ndim	维度个数，也就是数组轴的个数，比如一维、二维、三维等
ndarray.shape	数组的维度。这是一个整数的元组，表示每个维度上数组的大小。例如，一个 n 行和 m 列的数组，它的 shape 属性为(n,m)
ndarray.size	数组元素的总个数，等于 shape 属性中元组元素的乘积
ndarray.dtype	描述数组中元素类型的对象，既可以使用标准的 Python 类型创建或指定，也可以使用 NumPy 特有的数据类型来指定，比如 numpy.int32、numpy.float64 等
ndarray.itemsize	数组中每个元素的字节大小。例如，元素类型为 float64 的数组有 8（64/8）个字节，这相当于 ndarray.dtype.itemsize

值得一提的是，ndarray 对象中存储元素的类型必须是相同的。

为了让读者更好地理解 ndarray，接下来，通过一些示例来演示 ndarray 对象的使用，具体代码如下。

```
In [1]: import numpy as np                    # 导入 NumPy 工具包
In [2]: data = np.arange(12).reshape(3, 4)  # 创建一个 3 行 4 列的数组
In [3]: data
Out[3]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
In [4]: type(data)
Out[4]: numpy.ndarray
In [5]: data.ndim          # 数组维度的个数，输出结果 2，表示二维数组
Out[5]: 2
In [6]: data.shape         # 数组的维度，输出结果（3，4），表示 3 行 4 列
Out[6]: (3, 4)
In [7]: data.size          # 数组元素的个数，输出结果 12，表示总共有 12 个元素
Out[7]: 12
```

```
In [8]:  data.dtype # 数组元素的类型,输出结果 dtype('int64'),表示元素类型都是 int64
Out[8]: dtype('int64')
```

上述示例中，第 1 行代码使用 import...as 语句导入 numpy 库，并将其取别名为 np，表示后续会用 np 代替 numpy 执行操作。

第 2 行代码使用 arange()和 reshape()函数，创建了一个 3 行 4 列的数组 data。其中，arange()函数的功能类似于 range()，只不过 arange()函数生成的是一系列数字元素的数组；reshape()函数的功能是重组数组的行数、列数和维度。

第 4 行代码使用 type()函数查看了数组的类型，输出结果为 numpy.ndarray。

第 5 行代码获取了数组的维度个数，返回结果为 2，表示二维数组。

第 6 行代码获取了数组的维度，返回结果为(3,4)，表示数组有 3 行 4 列。

第 7 行代码获取了数组中元素的总个数，返回结果为 12，表示数组中一共有 12 个元素。

第 8 行代码获取了元素的具体类型，返回结果为 dtype('int64')，表示元素的类型为 int64。

2.2 创建 NumPy 数组

创建 ndarray 对象的方式有若干种，其中最简单的方式就是使用 array()函数，在调用该函数时传入一个 Python 现有的类型即可，比如列表、元组。例如，通过 array()函数分别创建一个一维数组和二维数组，具体代码如下。

```
 In [9]: import numpy as np
In [10]: data1 = np.array([1, 2, 3])                  # 创建一个一维数组
In [11]: data1
Out[11]: array([1, 2, 3])
In [12]: data2 = np.array([[1, 2, 3], [4, 5, 6]])   # 创建一个二维数组
In [13]: data2
Out[13]:
array([[1, 2, 3],
       [4, 5, 6]])
```

除了可以使用 array()函数创建 ndarray 对象外，还有其他创建数组的方式，

具体分为以下几种：

(1) 通过 zeros()函数创建元素值都是 0 的数组，示例代码如下。

```
In [14]: np.zeros((3, 4))

Out[14]:

array([[0., 0., 0., 0.],

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

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

(2) 通过调用 ones()函数创建元素值都为 1 的数组，示例代码如下。

```
In [15]: np.ones((3, 4))

Out[15]:

array([[1., 1., 1., 1.],

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

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

(3) 通过 empty()函数创建一个新的数组，该数组只分配了内存空间，它里面填充的元素都是随机的，且数据类型默认为 float64，示例代码如下。

```
In [16]: np.empty((5, 2))

Out[16]:

array([[-2.00000000e+000, -2.00390463e+000],

       [ 2.37663529e-312,  2.56761491e-312],

       [ 8.48798317e-313,  9.33678148e-313],

       [ 8.70018275e-313,  2.12199581e-314],

       [ 0.00000000e+000,  6.95335581e-309]])
```

(4) 通过 arange()函数可以创建一个等差数组，它的功能类似于 range()，只不过 arange()函数返回的结果是数组，而不是列表，示例代码如下。

```
In [17]: np.arange(1, 20, 5)

Out[17]: array([ 1,  6, 11, 16])
```

大家可能注意到，有些数组元素的后面会跟着一个小数点，而有些元素后面没有，比如 1 和 1.，产生这种现象，主要是因为元素的数据类型不同所导致的。

值得一提的是，在创建 ndarray 对象时，我们可以显式地声明数组元素的类

型，示例代码如下。

```
In [18]: np.array([1, 2, 3, 4], float)
Out[18]: array([1., 2., 3., 4.])
In [19]: np.ones((2, 3), dtype='float64')
Out[19]:
array([[1., 1., 1.],
       [1., 1., 1.]])
```

关于 ndarray 对象数据类型的更多介绍，将会在 2.3 小节中进行讲解。

2.3 ndarray 对象的数据类型

NumPy 支持比 Python 更多的数据类型。本节将为大家介绍一些常用的数据类型，以及这些数据类型之间的转换。

2.3.1 查看数据类型

如前面所述，通过“ndarray.dtype”可以创建一个表示数据类型的对象。要想获取数据类型的名称，则需要访问 name 属性进行获取，示例代码如下。

```
In [20]: data_one = np.array([[1, 2, 3], [4, 5, 6]])
In [21]: data_one.dtype.name
Out[21]: 'int32'
```

注意：

在默认情况下，64 位 windows 系统输出的结果为 int32，64 位 Linux 或 macOS 系统输出结果为 int64，当然也可以通过 dtype 来指定数据类型的长度。

上述代码中，使用 dtype 属性查看 data_one 对象的类型，输出结果是 int32。从数据类型的命名方式上可以看出，NumPy 的数据类型是由一个类型名（如 int、float）和元素位长的数字组成。

如果在创建数组时，没有显式地指明数据的类型，则可以根据列表或元组中的元素类型推导出来。默认情况下，通过 zeros()、ones()、empty()函数创建的数组中数据类型为 float64。

表 2-2 罗列了 NumPy 中常用的数据类型。

表 2-2 NumPy 的数据类型

数据类型	含义
bool	布尔类型，值为 True 或 False
int8、uint8	有符号和无符号的 8 位整数
int16、uint16	有符号和无符号的 16 位整数
int32、uint32	有符号和无符号的 32 位整数
int64、uint64	有符号和无符号的 64 位整数
float16	半精度浮点数（16 位）
float32	半精度浮点数（32 位）
float64	半精度浮点数（64 位）
complex64	复数，分别用两个 32 位浮点数表示实部和虚部
complex128	复数，分别用两个 64 位浮点数表示实部和虚部
object	Python 对象
string_	固定长度的字符串类型
unicode	固定长度的 unicode 类型

每一个 NumPy 内置的数据类型都有一个特征码，它能唯一标识一种数据类型，具体如表 2-3 所示。

表 2-3 NumPy 内置特征码

特征码	含义
b	布尔型
u	无符号整型
c	复数类型
S，a	字节字符串
V	原始数据
i	有符号整型
f	浮点型
O	Python 对象
U	unicode 字符串

2.3.2 转换数据类型

ndarray 对象的数据类型可以通过 astype()方法进行转换，示例代码如下：

```
In [22]: data = np.array([[1, 2, 3], [4, 5, 6]])

In [23]: data.dtype

Out[23]: dtype('int64')

In [24]: float_data = data.astype(np.float64) # 数据类型转换为 float64

In [25]: float_data.dtype

Out[25]: dtype('float64')
```

上述示例中，将数据类型 int64 转换为 float64，即整型转换为浮点型。若希望将数据的类型由浮点型转换为整型，则需要将小数点后面的部分截掉，具体示例代码如下。

```
In [26]: float_data = np.array([1.2, 2.3, 3.5])

In [27]: float_data

Out[27]: array([1.2, 2.3, 3.5])

In [28]: int_data = float_data.astype(np.int64) # 数据类型转换为 int64

In [29]: int_data

Out[29]: array([1, 2, 3], dtype=int64)
```

如果数组中的元素是字符串类型的，且字符串中的每个字符都是数字，则也可以使用 astype()方法将字符串转换为数值类型，具体示例如下。

```
In [30]: str_data = np.array(['1', '2', '3'])

In [31]: int_data = str_data.astype(np.int64)

In [32]: int_data

Out[32]: array([1, 2, 3], dtype=int64)
```

2.4 数组运算

NumPy 数组不需要循环遍历，即可对每个元素执行批量的算术运算操作，这个过程叫做矢量化运算。不过，如果两个数组的大小（ndarray.shape）不同，则它们进行算术运算时会出现广播机制。除此之外，数组还支持使用算术运算符

与标量进行运算，本节将针对数组运算的内容进行详细地介绍。

2.4.1 矢量化运算

在 NumPy 中，大小相等的数组之间的任何算术运算都会应用到元素级，即只用于位置相同的元素之间，所得的运算结果组成一个新的数组。接下来，通过一张示意图来描述什么是矢量化运算，具体如图 2-1 所示。

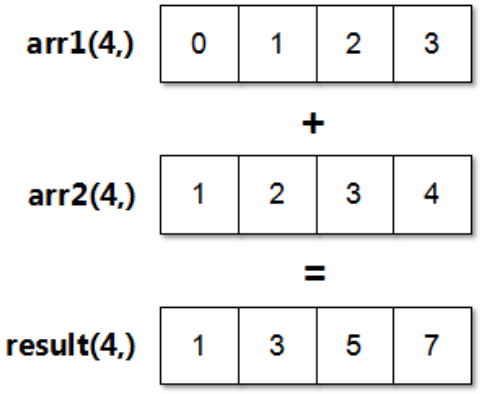

图2-1　形状相同的数组运算

由图 2-1 可知，数组 arr1 与 arr2 对齐以后，会让相同位置的元素相加得到一个新的数组 result。其中，result 数组中的每个元素为操作数相加的结果，并且结果的位置跟操作数的位置是相同的。

大小相等的数组之间的算术运算，示例代码如下。

```
In [33]: import numpy as np

In [34]: data1 = np.array([[1, 2, 3], [4, 5, 6]])

In [35]: data2 = np.array([[1, 2, 3], [4, 5, 6]])

In [36]: data1 + data2       # 数组相加

Out[36]:

array([[ 2,  4,  6],

       [ 8, 10, 12]])

In [37]: data1 * data2       # 数组相乘

Out[37]:

array([[ 1,  4,  9],

       [16, 25, 36]])
```

```
In [38]: data1 - data2        # 数组相减
Out[38]:
array([[0, 0, 0],
       [0, 0, 0]])
In [39]: data1 / data2       # 数组相除
Out[39]:
array([[1., 1., 1.],
       [1., 1., 1.]])
```

2.4.2 数组广播

数组在进行矢量化运算时，要求数组的形状是相等的。当形状不相等的数组执行算术计算的时候，就会出现广播机制，该机制会对数组进行扩展，使数组的 shape 属性值一样，这样就可以进行矢量化运算了。下面来看一个例子。

```
In [40]: import numpy as np
In [41]: arr1 = np.array([[0], [1], [2], [3]])
In [42]: arr1.shape
Out[42]: (4, 1)
In [43]: arr2 = np.array([1, 2, 3])
In [44]: arr2.shape
Out[44]: (3,)
In [45]: arr1 + arr2
Out[45]:
array([[1, 2, 3],
       [2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])
```

上述代码中，数组 arr1 的 shape 是（4，1），arr2 的 shape 是（3,），这两个数组要是进行相加，按照广播机制会对数组 arr1 和 arr2 都进行扩展，使得数组 arr1 和 arr2 的 shape 都变成（4，3）。

下面通过一张图来描述广播机制扩展数组的过程，具体如图 2-2 所示。

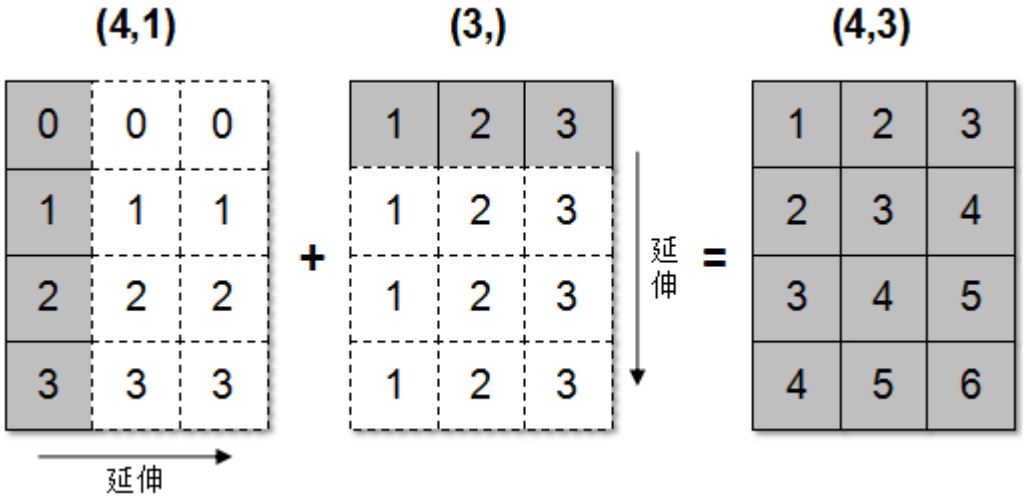

图2-2　数组广播机制

注意：

广播机制实现了对两个或两个以上数组的运算，即使这些数组的 shape 不是完全相同的，只需要满足如下任意一个条件即可：

（1）两个数组的某一维度等长。

（2）其中一个数组为一维数组。

广播机制需要扩展维度小的数组，使得它与维度最大的数组的 shape 值相同，以便使用元素级函数或者运算符进行运算。

2.4.3 数组与标量间的运算

大小相等的数组之间的任何算术运算都会将运算应用到元素级，同样，数组与标量的算术运算也会将那个标量值传播到各个元素。当数组进行相加、相减、乘以或除以一个数字时，这些称为标量运算。标量运算会产生一个与数组具有相同数量的行和列的新矩阵，其原始矩阵的每个元素都被相加、相减、相乘或者相除。

数组和标量之间的运算，示例代码如下：

```
In [46]: import numpy as np

In [47]: data1 = np.array([[1, 2, 3], [4, 5, 6]])

In [48]: data2 = 10
```

```
In [49]: data1 + data2      # 数组相加
Out[49]:
array([[11, 12, 13],
       [14, 15, 16]])
In [50]: data1 * data2      # 数组相乘
Out[50]:
array([[10, 20, 30],
       [40, 50, 60]])
In [51]: data1 - data2       # 数组相减
Out[51]:
array([[-9, -8, -7],
       [-6, -5, -4]])
In [52]: data1 / data2      # 数组相除
Out[52]:
array([[ 0.1,  0.2,  0.3],
       [ 0.4,  0.5,  0.6]])
```

2.5 ndarray 的索引和切片

ndarray 对象支持索引和切片操作，且提供了比常规 Python 序列更多的索引功能，除了使用整数进行索引以外，还可以使用整数数组和布尔数组进行索引。接下来，本节将针对 NumPy 的索引和切片进行详细地讲解。

2.5.1 整数索引和切片的基本使用

ndarray 对象的元素可以通过索引和切片来访问和修改，就像 Python 内置的容器对象一样。下面是一个一维数组，从表面上来看，该数组使用索引和切片的方式，与 Python 列表的功能相差不大，具体代码如下。

```
In [53]: import numpy as np
In [54]: arr = np.arange(8)    # 创建一个一维数组
In [55]: arr
```

```
Out[55]: array([0, 1, 2, 3, 4, 5, 6, 7])
In [56]: arr[5]                # 获取索引为 5 的元素
Out[56]: 5
In [57]: arr[3:5]              # 获取索引为 3~5 的元素，但不包括 5
Out[57]: array([3, 4])
In [58]: arr[1:6:2]            # 获取索引为 1~6 的元素，步长为 2
Out[58]: array([1, 3, 5])
```

不过，对于多维数组来说，索引和切片的使用方式与列表就大不一样了。在二维数组中，每个索引位置上的元素不再是一个标量了，而是一个一维数组，具体示例代码如下。

```
In [59]: import numpy as np
In [60]: arr2d = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]]) # 创建二维数组
In [61]: arr2d
Out[61]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
In [62]: arr2d[1]          # 获取索引为 1 的元素
Out[62]: array([4, 5, 6])
```

此时，如果我们想通过索引的方式来获取二维数组的单个元素，就需要通过形如“arr[x，y]”，以逗号分隔的索引来实现。其中，x 表示行号，y 表示列号。示例代码如下：

```
In [63]: arr2d[0, 1]       # 获取位于第 0 行第 1 列的元素
Out[63]: 2
```

接下来，通过一张图来描述数组 arr2d 的索引方式，如图 2-3 所示。

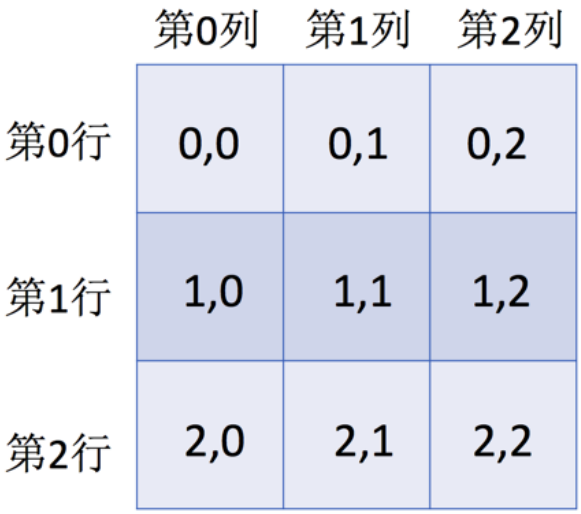

图2-3　arr2d 的索引方式

从图 2-3 中可以看出，arr2d 是一个 3 行 3 列的数组，如果我们想获取数组的单个元素，必须同时指定这个元素的行索引和列索引。例如，获取索引位置为第 1 行第 1 列的元素，我们可以通过 arr2d[1,1]来实现。

相比一维数组，多维数组的切片方式花样更多，多维数组的切片是沿着行或列的方向选取元素的，我们可以传入一个切片，也可以传入多个切片，还可以将切片与整数索引混合使用。

传入一个切片的示例代码：

```
In [64]: arr2d[:2]
Out[64]:
array([[1, 2, 3],
       [4, 5, 6]])
```

传入两个切片的示例代码：

```
In [65]: arr2d[0:2, 0:2]
Out[65]:
array([[1, 2],
       [4, 5]])
```

切片与整数索引混合使用的示例代码：

```
In [66]: arr2d[1, :2]
Out[66]: array([4, 5])
```

上述多维数组切片操作的相关示意图，如图 2-4 所示。

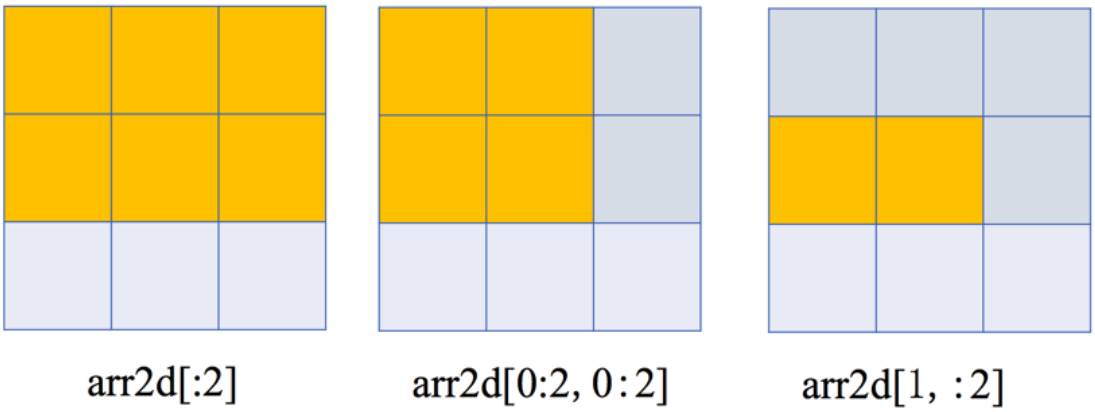

图2-4　多维数组切片图示

2.5.2 花式（数组）索引的基本使用

花式索引是 NumPy 的一个术语，是指用整数数组或列表进行索引，然后再将数组或列表中的每个元素作为下标进行取值。

当使用一个数组或列表作为索引时，如果使用索引要操作的对象是一维数组，则获取的结果是对应下标的元素；如果要操作的对象是一个二维数组，则获取的结果就是对应下标的一行数据。

例如，创建一个 4 行 4 列的二维数组，示例代码如下。

```
In [67]: import numpy as np
In [68]: demo_arr = np.empty((4, 4))                    # 创建一个空数组
         for i in range(4):
             demo_arr[i] = np.arange(i, i + 4)    # 动态地为数组添加元素
In [69]: demo_arr
Out[69]:
array([[ 0.,  1.,  2.,  3.],
       [ 1.,  2.,  3.,  4.],
       [ 2.,  3.,  4.,  5.],
       [ 3.,  4.,  5.,  6.]])
```

将[0,2]作为索引，分别获取 demo_arr 中索引 0 对应的一行数据以及索引 2 对应的一行数据，示例代码如下。

```
In [70]: demo_arr[[0, 2]]        # 获取索引为[0,2]的元素
```

```
Out[70]:

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

       [ 2.,  3.,  4.,  5.]])
```

上述操作的相关示意图如图 2-5 所示。

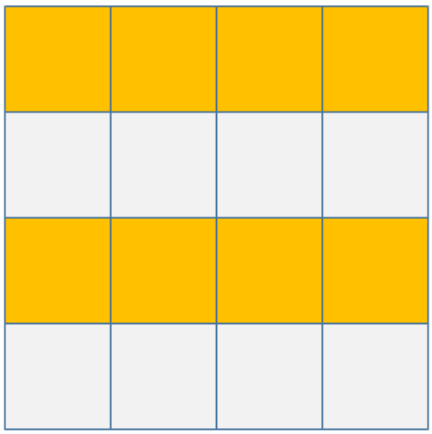

图2-5　花式索引图示（一个数组当索引）

如果使用两个花式索引操作数组时，即两个列表或数组，则会将第 1 个作为行索引，第 2 个作为列索引，通过二维数组索引的方式，选取其对应位置的元素，示例代码如下。

```
In [71]: demo_arr[[1, 3], [1, 2]]     # 获取索引为(1,1)和(3,2)的元素

Out[71]:

array([ 2.,  5.])
```

上述操作的相关示意图如图 2-6 所示。

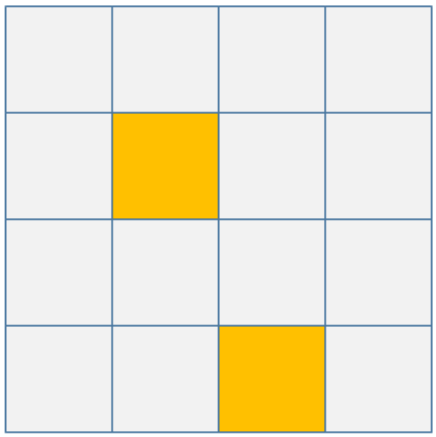

图2-6　花式索引图示（两个数组当索引）

2.5.3 布尔型索引的基本使用

布尔型索引指的是将一个布尔数组作为数组索引，返回的数据是布尔数组中True 对应位置的值。

假设现在有一组存储了学生姓名的数组，以及一组存储了学生各科成绩的数组，存储学生成绩的数组中，每一行成绩对应的是一个学生的成绩。如果我们想筛选某个学生对应的成绩，可以通过比较运算符，先产生一个布尔型数组，然后利用布尔型数组作为索引，返回布尔值 True 对应位置的数据。示例代码如下：

```
In [72]: # 存储学生姓名的数组
         student_name = np.array(['Tom', 'Lily', 'Jack', 'Rose'])
In [73]: student_name
Out[73]: array(['Tom', 'Lily', 'Jack', 'Rose'], dtype='<U4')
In [74]: # 存储学生成绩的数组
         student_score = np.array([[79, 88, 80], [89, 90, 92],
                                   [83, 78, 85], [78, 76, 80]])
In [75]: student_score
Out[75]:
array([[79, 88, 80],
       [89, 90, 92],
       [83, 78, 85],
       [78, 76, 80]])
In [76]: # 对 student_name 和字符串“Jack”通过运算符产生一个布尔型数组
         student_name == 'Jack'
Out[76]: array([False, False,  True, False])
In [77]: # 将布尔数组作为索引应用于存储成绩的数组 student_score，
         # 返回的数据是 True 值对应的行
         student_score[student_name=='Jack']
Out[77]: array([[83, 78, 85]])
```

布尔索引的相关示意图如图 2-7 所示。

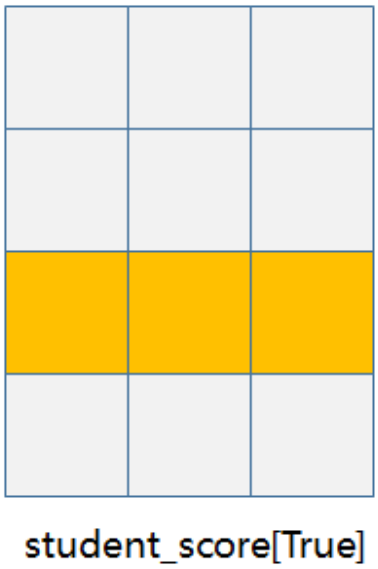

图2-7　布尔索引图示

需要注意的是，布尔型数组的长度必须和被索引的轴长度一致。

此外，我们还可以将布尔型数组跟切片混合使用，示例代码如下：

```
In [78]: student_score[student_name=='Jack', :1]

Out[78]: array([[83]])
```

值得一提的是，使用布尔型索引获取值的时候，除了可以使用“==”运算符，还可以使用诸如“!=”、“-”来进行否定，也可以使用“&”、“|”等符号来组合多个布尔条件。

2.6　数组的转置和轴对称

数组的转置指的是将数组中的每个元素按照一定的规则进行位置变换。NumPy 提供了 transpose()方法和 T 属性两种实现形式。其中，简单的转置可以使用 T 属性，它其实就是进行轴对换而已。例如，现在有个 3 行 4 列的二维数组，那么使用 T 属性对数组转置后，形成的是一个 4 行 3 列的新数组，示例代码如下。

```
In [79]: arr = np.arange(12).reshape(3, 4)

In [80]: arr

Out[80]:
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
In [81]: arr.T      # 使用 T 属性对数组进行转置
Out[81]:
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

对于高纬度的数组而言，transpose()方法需要得到一个由轴编号组成的元组，才能对这些轴进行转置。假设现在有个数组 arr，具体代码如下：

```
In [82]: arr = np.arange(16).reshape((2, 2, 4))
In [83]: arr
Out[83]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],
       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
In [84]: arr.shape
Out[84]: (2, 2, 4)
```

上述数组 arr 的 shape 是(2,2,4)，表示是一个三维数组，也就是说有三个轴，每个轴都对应着一个编号，分别为 0、1、2。

如果希望对 arr 进行转置操作，就需要对它的 shape 中的顺序进行调换。也就是说，当使用 transpose()方法对数组的 shape 进行调换时，需要以元组的形式传入 shape 的编号，比如(1,0,2)。如果调用 transpose()方法时传入“(0,1,2)”，则数组的 shape 不会发生任何变化。

下面是 arr 调用 transpose(1,0,2)的示例，具体代码如下。

```
In [85]: arr.transpose(1, 2, 0)   # 使用 transpose()方法对数组进行转置
Out[85]:
array([[[ 0,  8],
```

```
         [ 1,  9],
         [ 2, 10],
         [ 3, 11]],
        [[ 4, 12],
         [ 5, 13],
         [ 6, 14],
         [ 7, 15]]])
```

如果我们不输入任何参数，直接调用 transpose()方法，则其执行的效果就是将数组进行转置，作用等价于 transpose(2,1,0)。接下来，通过一张图比较使用所述两种方式的转置操作，具体如图 2-8 所示。

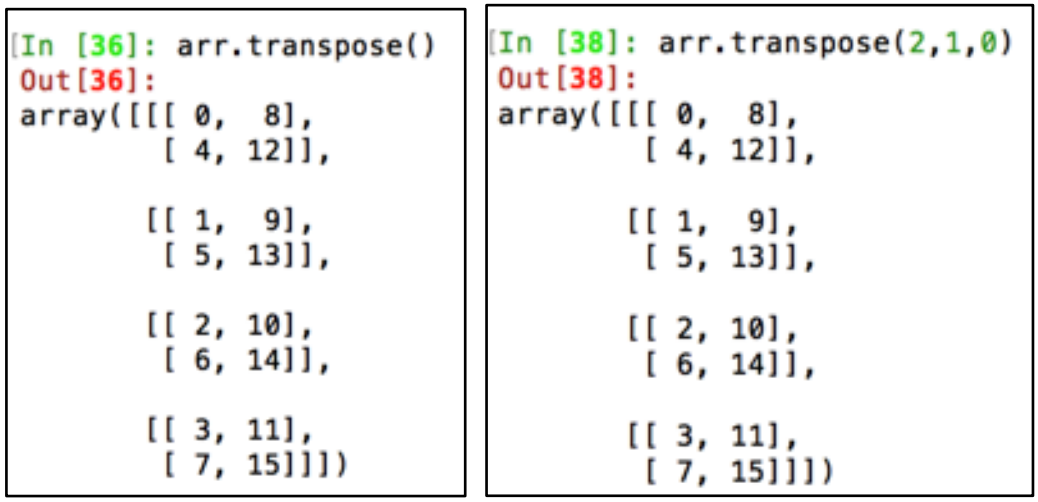

图2-8　transpose()方法传参对比

在某些情况下，我们可能只需要转换其中的两个轴，这时我们可以使用 ndarray 提供的 swapaxes()方法实现，该方法需要接受一对轴编号，示例代码如下。

```
In [86]: arr
Out[86]:
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],
       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
In [87]: arr.swapaxes(1, 0)    # 使用 swapaxes 方法对数组进行转置
Out[87]:
array([[[ 0,  1,  2,  3],
```

```
         [ 8,  9, 10, 11]],

        [[ 4,  5,  6,  7],

         [12, 13, 14, 15]]])
```

多学一招：轴编号

在 NumPy 中维度(dimensions)叫做轴(axes)，轴的个数叫做秩(rank)。例如，3D 空间中有个点的坐标[1, 2, 1]是一个秩为 1 的数组，因为它只有一个轴。这个轴有 3 个元素，所以我们说它的长度为 3。

在下面的示例中，数组有 2 个轴，第一个轴的长度为 2，第二个轴的长度为 3。

```
array([[ 1., 0., 0.],

       [ 0., 1., 2.]])
```

高维数据执行某些操作（如转置）时，需要指定维度编号，这个编号是从 0 开始的，然后依次递增 1。其中，位于纵向的轴（y 轴）的编号为 0，位于横向的轴（x 轴）的编号为 1，以此类推。

维度编号示意图如图 2-9 所示。

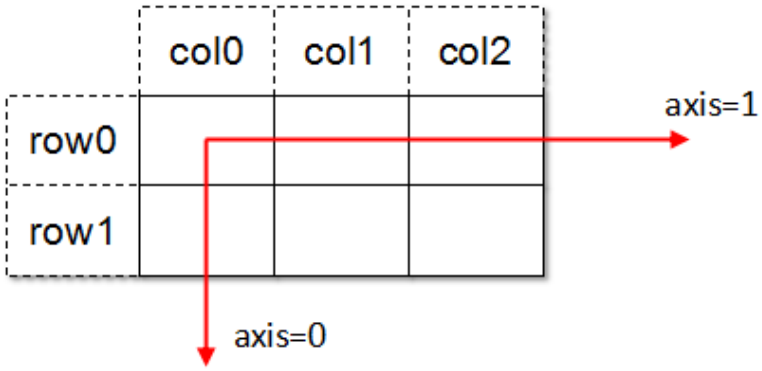

图2-9　维度编号图示

2.7 NumPy 通用函数

在 NumPy 中，提供了诸如“sin”、“cos”、“exp”等常见的数学函数，这些函数叫做通用函数（ufunc）。通用函数（ufunc）是一种针对 ndarray 中的数据执行元素级运算的函数，函数返回的是一个新的数组。通常情况下，我们将 ufunc 中接收一个数组参数的函数称为一元通用函数，而接受两个数组参数的则称为二

元通用函数。表 2-4 和表 2-5 列举了一些常见的一元和二元通用函数。

表 2-4 常见一元通用函数

函数	描述
abs、fabs	计算整数、浮点数或复数的绝对值
sqrt	计算各元素的平方根
square	计算各元素的平方
exp	计算各元素的指数 e^x
log、log10、log2、log1p	分别为自然对数（底数为 e），底数为 10 的 log，底数为 2 的 log，log(1+x)
sign	计算各元素的正负号：1（正数）、0（零）、-1（负数）
ceil	计算各元素的 ceilling 值，即大于或者等于该值的最小整数
floor	计算各元素的 floor 值，即小于等于该值的最大整数
rint	将各元素四舍五入到最接近的整数
modf	将数组的小数和整数部分以两个独立数组的形式返回
isnan	返回一个表示“哪些值是 NaN”的布尔型数组
isfinite、isinf	分别返回表示“哪些元素是有穷的”或“哪些元素是无穷”的布尔型数组
sin、sinh、cos、cosh、tan、tanh	普通型和双曲型三角函数
arcos、arccosh、arcsin	反三角函数

表 2-5 常见二元通用函数

函数	描述
add	将数组中对应的元素相加
subtract	从第一个数组中减去第二个数组中的元素
multiply	数组元素相乘

divide，floor_divide	除法或向下整除法（舍去余数）
maximum、fmax	元素级的最大值计算
minimum、fmin	元素级的最小值计算
mod	元素级的求模计算
copysign	将第二个数组中的值的符号赋值给第一个数组中的值
greater、greater_equal、less、less_equal、equal、not_equal、logical_and、logical_or、logical_xor	执行元素级的比较运算，最终产生布尔型数组，相当于运算符>、≥、<、≤、==、!=

为了让读者更好地理解，接下来，通过一些示例代码来演示上述部分函数的用法。有关一元通用函数的示例代码如下。

```
In [88]: arr = np.array([4, 9, 16])
# 计算数组元素的平方根
In [89]: np.sqrt(arr)
Out[89]: array([2., 3., 4.])
# 计算数组元素的绝对值
In [90]: np.abs(arr)
Out[90]: array([ 4,  9, 16])
# 计算数组元素的平方
In [91]: np.square(arr)
Out[91]: array([ 16,  81, 256])
```

有关二元通用函数的示例代码如下。

```
In [92]: x = np.array([12, 9, 13, 15])
In [93]: y = np.array([11, 10, 4, 8])
# 计算两个数组的和
In [94]: np.add(x, y)
Out[94]: array([23, 19, 17, 23])
# 计算两个数组的乘积
In [95]: np.multiply(x, y)
```

```
Out[95]: array([132,  90,  52, 120])

# 两个数组元素级最大值的比较

In [96]: np.maximum(x, y)

Out[96]: array([12, 10, 13, 15])

# 执行元素级的比较操作

In [97]: np.greater(x, y)

Out[97]: array([ True, False,  True,  True])
```

2.8 利用 NumPy 数组进行数据处理

NumPy 数组可以将许多数据处理任务转换为简洁的数组表达式，它处理数据的速度要比内置的 Python 循环快了至少一个数量级，所以，我们把数组作为处理数据的首选。接下来，本节将讲解如何利用数组来处理数据，包括条件逻辑、统计、排序、检索数组元素以及唯一化。

2.8.1 将条件逻辑转为数组运算

NumPy 的 where()函数是三元表达式 x if condition else y 的矢量化版本。

假设有两个数值类型的数组和一个布尔类型的数组，具体如下：

```
In [98]: arr_x = np.array([1, 5, 7])

In [99]: arr_y = np.array([2, 6, 8])

In [100]: arr_con = np.array([True, False, True])
```

现在提出一个需求，即当 arr_con 的元素值为 True 时，从 arr_x 数组中获取一个值，否则从 arr_y 数组中获取一个值。使用 where()函数实现的方式如下所示。

```
In [101]: result = np.where(arr_con, arr_x, arr_y)

In [102]: result

Out[102]: array([1, 6, 7])
```

上述代码中调用 np.where()时，传入的第 1 个参数 arr_con 表示判断条件，它可以是一个布尔值，也可以是一个数组，这里传入的是一个布尔数组。

当满足条件（从 arr_con 中取出的元素为 True）时，则会获取 arr_x 数组中对应位置的值。由于 arr_con 中索引为 0、2 的元素为 True，所以取出 arr_con 中

相应位置的元素 1、7。

当不满足条件（从 arr_con 中取出的元素为 False）时，则会获取 arr_y 数组中对应位置的值。由于 arr_con 中索引为 1 的元素为 False，所以取出 arr_con 中相应位置的元素 6。

从输出结果可以看出，使用 where()函数进行数组运算后，返回了一个新的数组。

2.8.2 数组统计运算

通过 NumPy 库中的相关方法，我们可以很方便地运用 Python 进行数组的统计汇总，比如计算数组极大值、极小值以及平均值等。表 2-6 列举了 NumPy 数组中与统计运算相关的方法。

表 2-6 NumPy 数组中与统计运算相关的方法

方法	描述
sum	对数组中全部或某个轴向的元素求和
mean	算术平均值
min	计算数组中的最小值
max	计算数组中的最大值
argmin	表示最小值的索引
argmax	表示最大值的索引
cumsum	所有元素的累计和
cumprod	所有元素的累计积

需要注意的是，当使用 ndarray 对象调用 cumsum()和 cumprod()方法后，产生的结果是一个由中间结果组成的数组。

为了能让读者更好地理解，接下来，通过一些示例来演示上述方法的具体用法，代码如下。

```
In [103]: arr = np.arange(10)

In [104]: arr.sum()      # 求和

Out[104]: 45

In [105]: arr.mean()     # 求平均值
```

```
Out[105]: 4.5

In [106]: arr.min()      # 求最小值

Out[106]: 0

In [107]: arr.max()       # 求最大值

Out[107]: 9

In [108]: arr.argmin()   # 求最小值的索引

Out[108]: 0

In [109]: arr.argmax()   # 求最大值的索引

Out[109]: 9

In [110]: arr.cumsum()   # 计算元素的累计和

Out[110]: array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])

In [111]: arr.cumprod()  # 计算元素的累计积

Out[111]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

2.8.3 数组排序

如果希望对 NumPy 数组中的元素进行排序，可以通过 sort()方法实现，示例代码如下。

```
In [112]: arr = np.array([[6, 2, 7], [3, 6, 2], [4, 3, 2]])

In [113]: arr

Out[113]:

array([[6, 2, 7],

       [3, 6, 2],

       [4, 3, 2]])

In [114]: arr.sort()

In [115]: arr

Out[115]:

array([[2, 6, 7],

       [2, 3, 6],

       [2, 3, 4]])
```

从上述代码可以看出，当调用 sort()方法后，数组 arr 中数据按行从小到大进行排序。需要注意的是，使用 sort()方法排序会修改数组本身。

如果希望对任何一个轴上的元素进行排序，只需要将轴的编号作为 sort()方法的参数传入即可。示例代码如下。

```
In [116]: arr = np.array([[6, 2, 7], [3, 6, 2], [4, 3, 2]])
In [117]: arr
Out[117]:
array([[6, 2, 7],
       [3, 6, 2],
       [4, 3, 2]])
In [118]: arr.sort(0)       # 沿着编号为 0 的轴对元素排序
In [119]: arr
Out[119]:
array([[3, 2, 2],
       [4, 3, 2],
       [6, 6, 7]])
```

2.8.4 检索数组元素

在 NumPy 中，all()函数用于判断整个数组中的元素的值是否全部满足条件，如果满足条件返回 True，否则返回 False。any()函数用于判断整个数组中的元素至少有一个满足条件就返回 True，否则就返回 False。

使用 all()和 any()函数检索数组元素的示例代码如下。

```
In [120]: arr = np.array([[1, -2, -7], [-3, 6, 2], [-4, 3, 2]])
In [121]: arr
Out[121]:
array([[ 1, -2, -7],
       [-3,  6,  2],
       [-4,  3,  2]])
In [122]: np.any(arr > 0)      # arr 的所有元素是否有一个大于 0
```

```
Out[122]: True

In [123]: np.all(arr > 0)      # arr 的所有元素是否都大于 0

Out[123]: False
```

2.8.5 唯一化及其他集合逻辑

针对一维数组，NumPy 提供了 unique()函数来找出数组中的唯一值，并返回排序后的结果，示例代码如下。

```
In [124]: arr = np.array([12, 11, 34, 23, 12, 8, 11])

In [125]: np.unique(arr)

Out[125]: array([ 8, 11, 12, 23, 34])
```

除此之外，还有一个 in1d()函数用于判断数组中的元素是否在另一个数组中存在，该函数返回的是一个布尔型的数组，示例代码如下。

```
In [126]: np.in1d(arr, [11, 12])

Out[126]: array([ True,  True, False, False,  True, False,  True])
```

NumPy 提供的有关集合的函数还有很多，表 2-7 列举了数组集合运算常见的函数。

表 2-7 数组集合运算的常见函数

函数	描述
unique(x)	计算 x 中的唯一元素，并返回有序结果
intersect1d(x,y)	计算 x 和 y 中的公共元素，并返回有序结果
union1d(x,y)	计算 x 和 y 的并集，并返回有序结果
in1d(x,y)	得到一个表示“x 的元素是否包含 y”的布尔型数组
setdiff1d(x,y)	集合的差，即元素在 x 中且不在 y 中
setxor1d(x,y)	集合的对称差，即存在于一个数组中但不同时存在于两个数组中的元素

2.9 线性代数模块

线性代数是数学运算中的一个重要工具，它在图形信号处理、音频信号处理

中起非常重要的作用。numpy.linalg 模块中有一组标准的矩阵分解运算以及诸如逆和行列式之类的东西。例如，矩阵相乘，如果我们通过“*”对两个数组相乘的话，得到的是一个元素级的积，而不是一个矩阵点积。

NumPy 中提供了一个用于矩阵乘法的 dot()方法，该方法的用法示例如下。

```
In [127]: arr_x = np.array([[1, 2, 3], [4, 5, 6]])

In [128]: arr_y = np.array([[1, 2], [3, 4], [5, 6]])

In [129]: arr_x.dot(arr_y)   # 等价于 np.dot(arr_x, arr_y)

Out[129]:

array([[22, 28],

       [49, 64]])
```

矩阵点积的条件是矩阵 A 的列数等于矩阵 B 的行数，假设 A 为 m*p 的矩阵，B 为 p*n 的矩阵，那么矩阵 A 与 B 的乘积就是一个 m*n 的矩阵 C，其中矩阵 C 的第 i 行第 j 列的元素可以表示为：

$$(A,B)_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ip}b_{pj}$$

上述矩阵 arr_x 与 arr_y 的乘积如图 2-10 所示：

1	2	3	•	1	2	=	1*1+2*3+3*5=22	1*2+2*4+3*6=28
4	5	6		3	4		4*1+5*3+6*5=49	4*2+5*4+6*6=64
				5	6			

图2-10　　矩阵 arr_x 与 arr_y 的乘积

除此之外，linalg 模块中还提供了其他很多有用的函数，具体如表 2-8 所示。

表 2-8 linalg 模块的常见函数

函数	描述
dot	矩阵乘法
diag	以一维数组的形式放回方阵的对角线，或将一维数组转为方阵
trace	计算对角线元素和
det	计算矩阵的行列式
eig	计算方阵的特征值和特征向量

inv	计算方阵的逆
qr	计算 qr 分解
svd	计算奇异值（SVD）
solve	解线性方程组 Ax=b，其中 A 是一个方阵
lstsq	计算 Ax=b 的最小二乘解

2.10 随机数模块

与 Python 的 random 模块相比，NumPy 的 random 模块功能更多，它增加了一些可以高效生成多种概率分布的样本值的函数。例如，通过 NumPy 的 random 模块随机生成一个 3 行 3 列的数组，示例代码如下。

```
In [130]: import numpy as np
In [131]: np.random.rand(3, 3)     # 随机生成一个二维数组
Out[131]:
array([[0.84507246, 0.69417139, 0.8596695 ],
       [0.65997549, 0.47116919, 0.82989148],
       [0.74321602, 0.06350157, 0.20833566]])
In [132]: np.random.rand(2, 3, 3) # 随机生成一个三维数组
Out[132]:
array([[[0.22736271, 0.57997499, 0.86616374],
        [0.19391042, 0.28925198, 0.66538324],
        [0.06265588, 0.27002459, 0.71791743]],
       [[0.67455806, 0.28524676, 0.26747945],
        [0.56214369, 0.32784243, 0.29093133],
        [0.56041467, 0.74910071, 0.99467489]]])
```

上述代码中，rand()函数隶属于 numpy.random 模块，它的作用是随机生成 N 维浮点数组。需要注意的是，每次运行代码后生成的随机数组都不一样。

除此之外，random 模块中还包括了可以生成服从多种概率分布随机数的其它函数。表 2-9 列举了 numpy.random 模块中用于生成大量样本值的函数。

表 2-9 random 模块的常见函数

函数	描述
seed	生成随机数的种子
rand	产生均匀分布的样本值
randint	从给定的上下限范围内随机选取整数
normal	产生正态分布的样本值
beta	产生 Beta 分布的样本值
uniform	产生在[0,1]中的均匀分布的样本值

在表 2-9 罗列的函数中，seed( )函数可以保证生成的随机数具有可预测性，也就是说产生的随机数相同，它的语法格式如下：

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

上述函数中只有一个 seed 参数，用于指定随机数生成时所用算法开始的整数值。当调用 seed()函数时，如果传递给 seed 参数的值相同，则每次生成的随机数都是一样的，如果传递这个参数值，则系统会根据时间来自己选择值，此时每次生成的随机数会因时间差异而不同。

使用 seed()函数的示例代码如下。

```
In [133]: import numpy as np
In [134]: np.random.seed(0)   # 生成随机数的种子
In [135]: np.random.rand(5)   # 随机生成包含 5 个元素的浮点数组
Out[135]:
array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
In [136]: np.random.seed(0)
In [137]: np.random.rand(5)
Out[137]:
array([ 0.5488135 ,  0.71518937,  0.60276338,  0.54488318,  0.4236548 ])
In [138]: np.random.seed()
In [139]: np.random.rand(5)
Out[139]:
array([0.9641088 , 0.75298789, 0.34224099, 0.43557176, 0.16201295])
```

由此可见，seed()函数使得随机数具有预见性。当传递的参数值不同或者不

传递参数时，则该函数的作用跟 rand()函数相同，即多次生成随机数且每次生成的随机数都不同。

2.11 案例—酒鬼漫步

通过前面对 NumPy 的学习，相信大家一定对 NumPy 这个科学计算包有了一定的了解，接下来，本节将通过酒鬼漫步的案例来介绍如何运用随机数模块与数据处理。

下面先为大家描述一下场景，在一片空旷的平地上（一个二维地面上）有一个醉汉，他最初停留在原点的位置，这个醉汉每走一步时，方向是不确定的，在经过时间 t 之后，我们希望计算出这个醉汉与原点的距离。

例如，这个醉汉走了 2000 步（每步为 0.5 米），向前走一步记为 1，向后走一步记为-1，当计算距原点的距离时，就是将所有的步数进行累计求和。因此，使用 random 模块来随机生成 2000 个“掷硬币值”（两个结果任选一个），具体代码如下：

```
In [140]: # 导入 numpy 包
          import numpy as np
          steps = 2000
          draws = np.random.randint(0, 2, size=steps)
          # 当元素为 1 时，direction_steps 为 1，
          # 当元素为 0 时，direction_steps 为-1
          direction_steps = np.where(draws > 0, 1, -1)
          # 使用 cumsum()计算步数累计和
          distance = direction_steps.cumsum()
```

有了步数的累计和之后，可以尝试计算醉汉距离原点最远的距离，即分别调用 max()与 min()计算向前走与向后走的最大值，具体代码如下。

```
In [141]: # 使用 max()计算向前走的最远距离
          distance.max()
Out[141]: 12
In [142]: # 使用 min()计算向后走的最远距离
```

```
            distance.min()

Out[142]: -31
```

从两次输出的结果中可以看出，这个醉汉走的最远的距离是朝后方距离原点15.5（31*0.5）米的位置。值得一提的是，由于这里使用的是随机数，所以每次运行的结果是随机的。

当醉汉距原点的距离大于或等于 15 米时，如果希望计算他总共走了多少步，则可以使用数学方程“|x * 0.5| >= 15”完成，其中 x 表示步数。要想计算一个数的绝对值，则需要调用 abs()函数进行实现，不过该函数返回的是一个布尔数组，即不满足条件的值均为 False，满足条件的值均为 True。为了从满足条件的结果中返回最大值的索引，则还需要通过调用 argmax()方法来实现，具体代码如下。

```
In [143]: # 15 米换算成步数

          steps = 15 / 0.5

          (np.abs(distance) >= steps).argmax()

Out[143]: 877
```

从计算结果可以看出，当醉汉走到 877 步时，此时距离原点的长度是大于或等于 15 米的。

2.12 本章小结

本章主要针对科学计算库 NumPy 进行了介绍，包括 ndarry 数组对象的属性和数据类型、数组的运算、索引和切片操作、数组的转置和轴对称、NumPy 通用函数、线性代数模块、随机数模块以及使用数组进行数据处理的相关操作。通过本章的学习，希望大家能熟练使用 NumPy 包，为后面章节的学习奠定基础。

2.13 本章习题

一、 填空题

1. 在 NumPy 中，可以使用数组对象_____执行一些科学计算。
2. 如果 ndarray.ndim 执行的结果为 2，则表示创建的是____维数组。
3. NumPy 的数据类型是由一个类型名和元素_____的数字组成。
4. 如果两个数组的大小（ndarray.shape）不同，则它们进行算术运算时会

出现_____机制。

5. 花式索引是 NumPy 的一个术语，是指用整数_____进行索引。

二、 判断题

1. 通过 empty()函数创建的数组，该数组中没有任何的元素。（ ）
2. 如果没有明确地指明数组中元素的类型，则默认为 float64。（ ）
3. 数组之间的任何算术运算都会将运算应用到元素级。（ ）
4. 多维数组操作索引时，可以将切片与整数索引混合使用。（ ）
5. 当通过布尔数组索引操作数组时，返回的数据是布尔数组中 False 对应位置的值。（ ）

三、 选择题

1. 下列选项中，用来表示数组维度的属性是（ ）。

A. ndim

B. shape

C. size

D. dtype

2. 下面代码中，创建的是一个 3 行 3 列数组的是（ ）。

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

3. 请阅读下面一段程序：

```
arr_2d = np.array([[11, 20, 13],[14, 25, 16],[27, 18, 9]])
print(arr_2d[1, :1])
```

执行上述程序后，最终输出的结果为（ ）。

A. [14]

B. [25]

C. [14, 25]

D. [20, 25]

4. 请阅读下面一段程序：

```
arr = np.arange(6).reshape(1, 2, 3)
```

```
print(arr.transpose(2, 0, 1))
```

执行上述程序后，最终输出的结果为（ ）。

A.

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

B.

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

C.

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

D.

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

5. 下列函数或方法中，用来表示矢量化三元表达式的是（ ）。

A. where()

B. cumsum()

C. sort()

D. unique()

四、 简答题

1. 什么是矢量化运算？

2. 实现数组广播机制需要满足哪些条件？

五、 程序题

1. 创建一个数组，数组的 shape 为(5,0)，元素都是 0。
2. 创建一个表示国际象棋棋盘的 8*8 数组，其中，棋盘白格用 0 填充，棋盘黑格用 1 填充。