

Python 数据分析与科学计算

NumPy

第1章 NumPy 简介

1.1 学习 NumPy 前置知识点

- 1、Python 基础语言
- 2、重点 Python 内置数据结构：列表 (list)、字符串 (str)、元组 (tuple)、集合 (set) 和字典 (dict)。
- 3、序列的索引和切片操作。
- 4、列表推导式、集合推导式、字典推导式

`n_list = [x ** 2 for x in range(10) if x % 2 == 0]`

输出表达式 元素变量 输入序列 条件语句

输入: `n_list = [x for x in range(100) if x % 2 == 0 if x % 5 == 0]`
`print(n_list)`

输出:

- 5、lambda 表达式

- 6、Python 三个函数式编程基础的函数：filter()、map()和 reduce()。

输入: `users = ['Tony', 'Tom', 'Ben', 'Alex']`
`users_filter = filter(lambda u: u.startswith('T'), users)`
`print(list(users_filter))`

```
输出:  ['Tony', 'Tom']
```

1.2 NumPy 是什么？

NumPy (Numerical Python 的缩写) 是一个开源的 Python 数据分析和科学计算库。

1、NumPy 是 Pandas (数据分析)、SciPy (科学计算) 和 Matplotlib (绘图库) 基础。

2、图像处理 OpenCV for Python 中也大量使用 NumPy。

3、NumPy 官网: <http://www.numpy.org> (或 <http://www.scipy.org>)

4、NumPy 源代码: <https://github.com/numpy/numpy>

1.3 为什么选择 NumPy

1、Python 写出易读、整洁并且缺陷最少的代码。

2、NumPy 底层是用 C 语言实现速度快。

3、NumPy 提供数据结构 (数组) 比 Python 内置数据结构访问效率更高。

4、支持大量高维度数组与矩阵运算。

5、提供大量的数学函数库。

1.4 课后练习

1、访问 NumPy 官网。

- 2、使用《NumPy 用户指南文档》。
- 3、使用《NumPy API 文档》。

第2章 环境搭建

2.1 环境方案 1：手动安装

Python 要求 Python 3.5 以及以上版本。

2.1.1 使用 pip 安装 NumPy 等库

需要安装的库 numpy、scipy、matplotlib。

在 Windows 命令提示符中使用 pip 安装指令：

```
pip install numpy
```

2.1.2 安装 IPython

IPython 是一个加强版本的 Python 解释器。

在命令提示符或终端中使用 pip 安装 IPython 指令：

```
pip install ipython
```

2.2 环境方案 2：安装 Anaconda

Anaconda 指的是一个开源的 Python 发行版本，其包含了 conda、Python 等 1500 多个科学包。

1、Anaconda 官网：<https://www.anaconda.com>

2、清华大学开源软件镜像站：

<https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/>

❑ Anaconda 安装包：

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

❑ Miniconda 安装包，只包含了 python 和 conda：

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/>

2.2.1 Windows 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.2.2 Linux 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.2.3 macOS 安装 Anaconda

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

2.3 开发工具

Python shell、Python IDEL、IPython shell 、IDE 工具 (Pycharm、Eclipse Pydev 插件、Visual Studio Code、Spyder) 、Jupyter Notebook

2.3.1 IPython shell

1、启动

在命令提示符或终端中使用如下指令：

```
ipython
```

2、获得帮助

通过?或??获取获得帮助，示例如下：

3、用 Tab 键语法补全

示例如下：

4、IPython shell 中的快捷键

表 2-1 IPython shell 中常用的快捷键

快捷键	说明
Ctrl + p (或向上箭头)	获取前一个历史命令
Ctrl + n (或向下箭头)	获取后一个历史命令
Ctrl + r	对历史命令的反向搜索
Ctrl + L	清除终端屏幕的内容
Ctrl + d	退出 IPython 会话

示例如下：

5、IPython shell 魔法命令

表 2-2 IPython shell 常用魔法命令

魔法命令	说明
%run	执行外部代码
%timeit	计算代码运行时间
%magic	获得所有可用魔法命令的列表
?	某个魔法命令帮助

示例如下：

2.3.2 Jupyter Notebook

Jupyter Notebook 是 IPython shell 基于浏览器的图形界面。Jupyter Notebook 不仅可以执行 Python/IPython 语句，还允许用户编写科技文章。

1、安装

- ☐ Anaconda 完整安装包括 Jupyter Notebook。
- ☐ 手动使用 pip 安装指令：

```
pip install jupyter
```

2、启动

在命令提示符或终端中使用如下指令：

```
jupyter notebook
```

示例如下：

2.3.3 Spyder

Spyder(Scientific Python Development Environment)是一个强大的交互式 Python 语言 IDE 开发环境，支持包括 Windows、Linux 和 macOS 系统。Spyder 还集成了很多流行科学软件包，包括 NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy 等。

Spyder 官网: www.spyder-ide.org

1、安装

- ❑ Anaconda 完整安装包括 Spyder。
- ❑ 手动使用 pip 安装指令：

```
pip install spyder
```

~~手动使用安装 Spyder 不推荐！！！！~~

2、基本用法

2.4 课后练习

- 1、查看你的 Python 解释器版本。
- 2、查看你的 IPython 解释器版本。
- 3、如何使用 pip 或 conda 指令查询 NumPy 版本 。
- 4、如何在 IPython shell、Spyder、Jupyter Notebook 中查找 print 函数帮助。

第3章 编写 NumPy 程序

- ❑ NumPy 库中最重要的数据结构是多维数组对象 (ndarray)，它是一系列同类型数据的集合，下标索引从 0 开始。
- ❑ 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。

3.1 创建一维数组

```
输入:  import numpy as np
        a = np.array([1, 2, 3])
        print(a)
```

```
输出:  [1 2 3]
```

array 函数创建 ndarray 对象，其中参数可以是如下类型：

- ❑ Python 列表 (list)
- ❑ Python 元组 (tuple)

```
输入:  import numpy as np
        a = np.array((1, 2, 3))
        print(a)
```

```
输出:  [1 2 3]
```

提示 数组与Python列表的主要区别：数组只能保存相同数据类型,而Python列表可以是任何类型.

3.2 NumPy 数据类型

3.2.1 指定数组数据类型

NumPy 支持的数据类型比 Python 内置的数据类型要多。

```
输入:    # 浮点类型
         b = np.array([1., 2., 3.])
         print(b)
         print(b.dtype)
```

```
输出:    [1.  2.  3.]
         float64
```

数组的 dtype 属性返回数组的类型。

也可以在声明时指定数据类型。

```
输入:    # 指定dtype参数
         b = np.array((1, 2, 3, 4), dtype=float)
         print(b)
         print(b.dtype)
```

```
输出:    [1.  2.  3.  4.]
         float64
```

3.2.2 更多数据类型

```
输入:    # 复数类型
         d = np.array([1+2j, 3+4j, 5+6*1j])
         d.dtype
```

```
输出:    dtype('complex128')
```

输入: # 布尔类型
 e = np.array([True, False, False, True])
 e.dtype

输出: dtype('bool')

输入: # Unicode字符串
 f = np.array(['您好', 'Hello', 'Hallo'])
 f.dtype

输出: dtype('<U5')

输入: # ASCII字符串
 e = np.array([b'Bonjour', b'Hello', b'Hallo'])
 e.dtype

输出: dtype('S7')

表 3-1 numpy 数据类型

numpy 数据类型	类型代码	描述
np.bool_		布尔型数据类型 (True 或者 False)
int_		默认的整数类型 (类似于 C 语言中的 long, int32 或 int64)
intc		与 C 的 int 类型一样, 一般是 int32 或 int 64
int8	i1	字节 (-128 to 127)
int16	i2	整数 (-32768 to 32767)
int32	i4	整数 (-2147483648 to 2147483647)
int64	i8	整数 (-9223372036854775808 to 9223372036854775807)
uint8	u1	无符号整数 (0 to 255)
uint16	u2	无符号整数 (0 to 65535)

uint32	u4	无符号整数 (0 to 4294967295)
uint64	u8	无符号整数 (0 to 18446744073709551615)
float_		float64 类型的简写
float16	f2	半精度浮点数
float32	f4 或 f	单精度浮点数
float64	f8 或 d	双精度浮点数
complex_		complex128 类型的简写, 即 128 位复数
complex64	c8	复数, 表示双 32 位浮点数 (实数部分和虚数部分)
complex128	c16	复数, 表示双 64 位浮点数 (实数部分和虚数部分)
string_	S	ASCII 字符串, 例如: S7 表示长度为 7 的 ASCII 字符串
unicode_	U	Unicode 字符串, 例如: U5 表示长度为 5 的 Unicode 编码字符串

3.2.3 类型代码和字节序

```
输入:  # Unicode字符串
      f = np.array(['您好', 'Hello', 'Hallo'])
      f.dtype
-----
输出:  dtype('<U5')
```

<U5 表示长度为 5 的 Unicode 编码字符串。<表示小端字节序; >表示大端字节序。

提示 小端字节序: 数据低位在前, 高位在后; 大端字节序: 数据高位在前, 低位在后。

`int a = 0x08070605`



图 3-1 字节序

```
输入:  # 指定dtype参数
      a = np.array([1, 2, 3, 4], dtype='<u4')
      a.dtype
```

```
输出: dtype('uint32')
```

```
输入:  dt = np.dtype('<i4') # 创建一个dtype对象
      dt
```

```
输出: dtype('int32')
```

```
输入:  # 使用对象dtype指定dtype参数
      b = np.array([1, 2, 3, 4], dtype=dt)
      b.dtype
```

```
输出: dtype('int32')
```

3.2.4 类型转换

使用数组对象 (ndarray) 的 `astype(dtype)` 方法可以转换数组元素的数据类型。

```
输入:  a = np.array([1.0, 2.1, 3.9])
        print(a)
        print(a.dtype)

        b = a.astype(np.int32)
        print(b)
        print(b.dtype)
```

```
输出:  [1.  2.1 3.9]
        float64
        [1 2 3]
        int32
```

注意 `astype(dtype)`方法会创建一个新的对象。

可以使用类型代码：

```
输入:  c = np.array(["1.0", "-2.1", "3.9"])
        print(c)
        print(c.dtype)

        d = c.astype("f4")
        print(d)
        print(d.dtype)
```

```
输出:  ['1.0' '-2.1' '3.9']
        <U4
        [ 1. -2.1  3.9]
        float32
```

提示 如果类型不兼容则会抛出TypeError错误。

3.3 更多创建一维数组方式

使用 `array()` 函数是将 Python 内置的列表或元组转换为 NumPy 数组对象，这样做效率不高。为此 NumPy 提供了很多创建数组的函数。

- ❑ `arange`
- ❑ `linspace` (线性等分向量)
- ❑ `logspace` (对数等分向量)

注意 标量、向量 (矢量)、矩阵和张量。

3.3.1 使用 `arange` 函数

NumPy 包中的使用 `arange` 函数创建数值范围并返回数组对象，与 Python 中类型是 `range` 函数类似。`arange` 函数语法格式如下：

```
numpy.arange([start, ]stop, [step, ] dtype=None)
```

- ❑ `start` 是开始值，可以省略，默认值为 0，包含开始值。
- ❑ `stop` 是结束值，不包含结束值。
- ❑ `step` 是步长，默认值为 1。
- ❑ `dtype` 是数组元素类型。

注意 $\text{start} \leq \text{数组元素} < \text{stop}$, 步长 `step` 可以为负数, 可以创建递减序列。

```
输入:  a = np.arange(10)
       print(a)

       b = np.arange(1, 10, 2)
       print(b)

       c = np.arange(1, -10, -3)
       print(c)

       d = np.arange(1, -10, -1, dtype='f8')
       print(d)
       print(d.dtype)
```

```
输出:  [0 1 2 3 4 5 6 7 8 9]
       [1 3 5 7 9]
       [ 1 -2 -5 -8]
       [ 1.  0. -1. -2. -3. -4. -5. -6. -7. -8. -9.]
       float64
```

3.3.2 等差数列与 linspace 函数

`linspace`（线性等分向量）函数创建等差数列，语法格式如下：

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

- ❑ `start` 是开始值，包含开始值。
- ❑ `stop` 是结束值，默认包含结束值。
- ❑ `num` 是设置生成的元素个数。
- ❑ `endpoint` 是否包含结束值（`stop`），`False` 是不包含，`True` 是包含，默认值是 `True`。
- ❑ `retstep` 是否返回步长（或公差），`False` 是不返回，默认值是 `False`。`True` 是返回，当为 `True` 是返回值是二元组（数列数组，步长）。

```
输入:  a = np.linspace(0, 10, 10)
       print(a)

       b = np.linspace(0, 10, 10, endpoint=False)
```

```
print(b)

c = np.linspace(0, 10, 10, endpoint=False, retstep=True)
print(c)
mystep = c[1]
print("步长=", mystep)
```

输出: `[0. 1.11111111 2.22222222 3.33333333 4.44444444`
`5.55555556`
`6.66666667 7.77777778 8.88888889 10.]`
`[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]`
`(array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.]), 1.0)`
 步长= 1.0

3.3.3 等比数列与 logspace 函数

logspace（对数等分向量）函数创建等比数列，语法格式如下：

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

- ☐ start 是开始值 `base ** start`。
- ☐ stop 是结束值 `base ** stop`。
- ☐ base 是底数。

```
输入: a = np.logspace(0, 9, 10)
      print(a)

      b = np.logspace(0, 10, 10, endpoint=False)
      print(b)

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

输出: `[1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05 1.e+06 1.e+07 1.e+08`
`1.e+09]`
`[1.e+00 1.e+01 1.e+02 1.e+03 1.e+04 1.e+05 1.e+06 1.e+07 1.e+08`
`1.e+09]`

```
[ 1.  2.  4.  8. 16. 32. 64. 128. 256. 512.]
```

3.4 课后练习

1、写一个 NumPy 程序来获取 NumPy 版本。

参考答案：

```
import numpy as np
print(np.__version__)
```

2、编写一个 NumPy 程序来创建一个从 30 到 50 的所有偶数整数的数组，取值[30, 50)。

参考答案：

```
import numpy as np
array=np.arange(30,50,2)
print(array)
```

3、编写 NumPy 程序以获取有关 np.add 函数的帮助信息。

参考答案：

```
import numpy as np
print(np.info(np.add))
```

4、从给定数组中提取所有奇数。

参考答案:

```
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
arr[arr % 2 == 1]
```

5、编写 NumPy 程序创建介于 2.5 到 6.5 之间 30 个均匀间隔元素的一维数组，包括 6.5。

参考答案:

```
import numpy as np
x = np.linspace(2.5, 6.5, 30)
print(x)
```

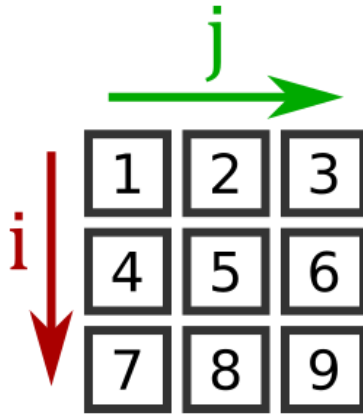
6、编写 NumPy 程序创建 20 元素的等比数列数组，指数介于 2 到 5，包括 5。

参考答案:

```
import numpy as np
x = np.logspace(2, 5, 20)
print(x)
```

第4章 二维数组

4.1 创建二维数组



```
输入:  L = [[1,2,3], [4,5,6], [7,8,9]]
      a = np.array(L) # 嵌套列表创建ndarray数组
      print(a)
      print(a.dtype)
```

```
输出:  [[1 2 3]
        [4 5 6]
        [7 8 9]]
      int32
```

```
输入:
      T1 = ([1.5,2,3], [4,5,6], [7,8,9])
      b = np.array(T1) # 嵌套元组创建ndarray数组
      print(b)
      print(b.dtype)
```

```
输出:  [[1.5  2.   3. ]
        [4.   5.   6. ]
        [7.   8.   9. ]]
float64
```

```
输入:  T2 = ((1,2,3), (4,5,6), (7,8,9))
c = np.array(T2, dtype='f8') # 嵌套元组创建ndarray数组
print(c)
print(c.dtype)
```

```
输出:  [[1.  2.  3.]
        [4.  5.  6.]
        [7.  8.  9.]]
float64
```

4.2 重新设置维度

数组的 **shape 属性**是数组的**形状**，返回值是一个元组。数组的 `reshape()` 方法可以重新设置数组的形状。

```
输入:  d = np.arange(1, 10)
print(d)
print("d的形状:", d.shape)
dd = d.reshape((3, 3)) # 从一维到二维

print(dd)
print("dd的形状:", dd.shape)
```

```
输出:  [1  2  3  4  5  6  7  8  9]
d的形状: (9,)
[[1  2  3]
 [4  5  6]
 [7  8  9]]
dd的形状: (3, 3)
```

4.3 更多创建二维数组方式

`arange`、`linspace` 和 `logspace` 函数都是创建一维数组。还有创建多维数组函数：

- ❑ `ones`
- ❑ `zeros`
- ❑ `empty`
- ❑ `full`
- ❑ `eye` 和 `identity`

4.3.1 使用 `ones` 函数

`ones` 函数可以根据指定的形状和数据类型生成全为 1 的数组。他的语法格式如下：

```
numpy.ones(shape, dtype=None)
```

示例：

```
输入： a = np.ones((2, 3))
        print(a)
        print(a.dtype)

        b = np.ones((2, 3), dtype='i4')
        print(b)
        print(b.dtype)
```

```
输出： [[1.  1.  1.]
        [1.  1.  1.]]
        float64
        [[1  1  1]
        [1  1  1]]
        int32
```

```
输入:  c = np.ones((3,)) # 生成一维数组
        print(c)
        print(c.dtype)

        d = np.ones((3, 1)) # 生成列向量数组
        print(d)
        print(d.dtype)
```

```
输出:  float64
        [[1.]
         [1.]
         [1.]]
        float64
```

4.3.2 使用 zeros 函数

`zeros` 函数可以根据指定的形状和数据类型生成全为 0 的数组。他的语法格式如下:

```
numpy.zeros(shape, dtype=float)
```

示例:

```
输入:  a = np.zeros((2, 3))
        print(a)
        print(a.dtype)

        b = np.zeros((2, 3), dtype='i4')
        print(b)
        print(b.dtype)
```

```
输出:  [[0. 0. 0.]
         [0. 0. 0.]]
```



```
float64
[[0 0 0]
 [0 0 0]]
int32
```

```
输入:  c = np.zeros((3,)) # 生成一维数组
        print(c)
        print(c.dtype)

        d = np.zeros((3, 1)) # 生成列向量数组
        print(d)
        print(d.dtype)
```

```
输出:  [0. 0. 0.]
        float64
        [[0.]
         [0.]
         [0.]]
        float64
```

4.3.3 使用 empty 函数

`empty` 函数可以根据指定的形状和数据类型生成数组, 其中的没有初始化。他的语法格式如下:

```
numpy.empty(shape, dtype=float)
```

示例:

```
输入:  e = np.empty([2, 2])
        print(e)
        print(e.dtype)
```

```
f = np.empty((2, 2), dtype='i4')
print(f)
print(f.dtype)
```

输出:

```
[[1.39737041e-311 1.39751574e-311]
 [1.39751574e-311 1.39751574e-311]]
float64
[[16843009 16843009]
 [16843009 16843009]]
int32
```

4.3.4 使用 full 函数

full 函数可以根据指定的形状和数据类型生成数组, 并用指定数组填充。他的语法格式如下:

```
numpy.full(shape, fill_value, dtype=None)
```

示例:

```
输入: a = np.full((2, 4), 10)
      print(a)
      print(a.dtype)

      b = np.full((2, 4), 10, dtype=float)
      print(b)
      print(b.dtype)

      c = np.full(5, 10)
      print(c)
      print(c.dtype)
```

```

输出:  [[10 10 10 10]
        [10 10 10 10]]
int32
[[10. 10. 10. 10.]
 [10. 10. 10. 10.]]
float64
[10 10 10 10 10]
int32

```

4.3.5 使用 identity 和 eye 函数

1、identity 函数可以创建单位矩阵，即：对角线元素为 1.0，其他元素为 0.0。他的语法格式如下：

```
numpy.identity(n, dtype=None)
```

1	0	0
0	1	0
0	0	1

图 4-1 3 阶单位矩阵

2、eye 函数可以创建二维数组，对角线元素为 1.0，其他元素为 0.0。他的语法格式如下：

```
numpy.eye(N, M=None, k=0, dtype=float)
```

- ❑ N 是行数。
- ❑ M 是列数，如果省略，则生成 N x N 矩阵。
- ❑ k 是对角线开始位置的索引，默认为 0，主对角线。
- ❑ dtype 指定元素数据类型，默认是 float。

示例：

```
输入:  a = np.identity(3)
      print(a)
      print(a.dtype)

      b = np.eye(3)
      print(b)
      print(b.dtype)
```

```
输出:  [[1.  0.  0.]
        [0.  1.  0.]
        [0.  0.  1.]]
float64
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
float64
```

```
输入:  c = np.eye(3,4)
      print(c)
      print(c.dtype)

      d = np.eye(3, 4, k=1)
      print(d)
      print(d.dtype)

      e = np.eye(3, 4, k=1, dtype="i4")
      print(e)
      print(e.dtype)
```

```
输出:  [[1.  0.  0.  0.]
        [0.  1.  0.  0.]
        [0.  0.  1.  0.]]
float64
```

```
[[0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 0. 1.]]  
float64  
[[0 1 0 0]  
 [0 0 1 0]  
 [0 0 0 1]]  
int32
```

4.4 数组的属性

表 4-1 常用数组属性

数组	说明
ndim	数组的维度
shape	数组的形状，每个维度的元素个数
size	数组的元素总个数
dtype	数组的类型
itemsize	每个元素的大小，以字节为单位
nbytes	数组总字节大小，以字节为单位。等于 size* itemsize

示例：

```
输入： a = np.ones((2, 3))  
  
print("a数组的维度：", a.ndim)  
print("a数组的形状:", a.shape)  
print("a数组的元素总个数:", a.size)  
print("a数组的类型:", a.dtype)  
print("每个元素的大小:", a.itemsize)  
  
b = np.ones(3)  
print("b数组的维度：", b.ndim)  
print("b数组的形状:", b.shape)  
print("b数组的元素总个数:", b.size)  
print("每个元素的大小:", a.itemsize)
```

输出: a数组的维度: 2
a数组的形状: (2, 3)
a数组的元素总个数: 6
a数组的类型: float64
每个元素的大小: 8
b数组的维度: 1
b数组的形状: (3,)
b数组的元素总个数: 3
每个元素的大小: 8

4.5 数组的轴

二维数组:

```
输入: # 创建二维数组
a2 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
print(a2)
```

输出: [[1 2 3]
[4 5 6]
[7 8 9]]

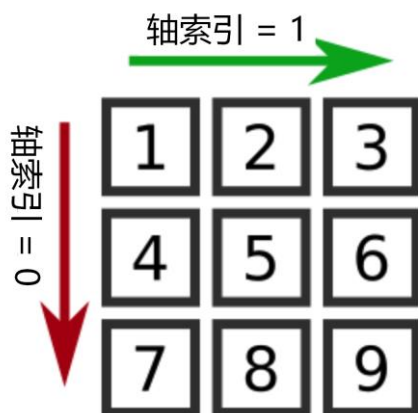


图 4-2 二维数组轴

三维数组：

```
输入： # 创建三维数组
a3 = np.array([[[10, 11, 12], [13, 14, 15], [16, 17, 18]],
               [[20, 21, 22], [23, 24, 25], [26, 27, 28]],
               [[30, 31, 32], [33, 34, 35], [36, 37, 38]]])

print(a3)
```

```
输出： [[10 11 12]
        [13 14 15]
        [16 17 18]]

        [[20 21 22]
        [23 24 25]
        [26 27 28]]

        [[30 31 32]
        [33 34 35]
        [36 37 38]]]
```

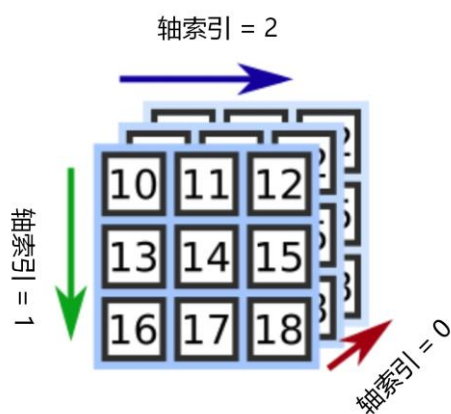


图 4-3 三维数组轴

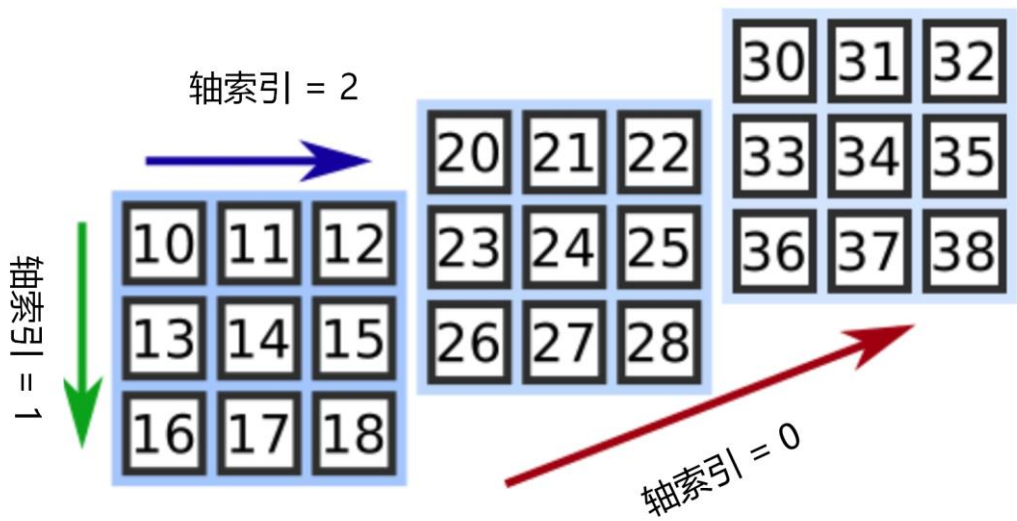


图 4-4 拉长的三维数组轴

4.6 数组转置

数组的 `T` 属性可以转置数组，将数组轴的索引倒置。

- ❑ 一维数组不能转置
- ❑ 数组形状为 (n, m) ，转置后的形状为 (m, n)
- ❑ 数组形状为 $(a_0, a_1, \dots, a_{n-1}, a_n)$ ，转置后的形状为 $(a_n, a_{n-1}, \dots, a_1, a_0)$



图 4-5 数组转置


```
输入:  # 创建二维数组
      a = np.array([[1, 2, 3],
                    [4, 5, 6]])
      print(a, a.shape)
      at = a.T
      print(at, at.shape)
```

```
输出:  [[1 2 3]
        [4 5 6]] (2, 3)
        [[1 4]
        [2 5]
        [3 6]] (3, 2)
```

```
输入:  # 创建一维数组
      b = np.array([1, 2, 3, 4])
      print(b)
      print(b.T) # 一维数组不能转置
```

```
输出:  [1 2 3 4]
        [1 2 3 4]
```

```
输入:  c = np.array([[1, 2, 3, 4]])
      print(c)
      print(c.T)
```

```
输出:  [[1 2 3 4]]
        [[1]
        [2]
        [3]
        [4]]
```

4.7 课后练习

1、编写 NumPy 程序将给定数组转换为列表。

参考答案：

```
import numpy as np
a = [[1, 2], [3, 4]]
x = np.array(a)
a2 = x.tolist()
print(a == a2)
```

2、编写一个 NumPy 程序来创建一个空数组。

参考答案：

```
import numpy as np
x = np.empty((3,4))
print(x)
```

3、编写一个 NumPy 程序来创建一个数组，并使用 full 函数填充。

参考答案：

```
import numpy as np
y = np.full((3,3),6)
print(y)
```

4、编写一个 NumPy 程序来创建一个大小为 2 x 3 的二维数组，并打印数组、数组形状和类型。

参考答案：

```
import numpy as np
x = np.array([[2, 4, 6], [6, 8, 10]], np.int32)
print(type(x))
print(x.shape)
print(x.dtype)
```

5、编写 NumPy 程序，为数组创建新形状。

参考答案：

```
import numpy as np
x = np.array([1, 2, 3, 4, 5, 6])
y = np.reshape(x,(3,2))
print("Reshape 3x2:")
print(y)
z = np.reshape(x,(2,3))
print("Reshape 2x3:")
print(z)
```

第5章 访问数组

5.1 索引访问

5.1.1 一维数组索引访问

NumPy 一维数组索引访问与 Python 内置序列类型索引访问一样，使用中括号+下标（[index]）。

正值索引	0	1	2	3	4	5
a数组	1	2	3	4	5	6
负值索引	0	-5	-4	-3	-2	-1

图 5-1 一维数组索引

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

```
print(a[5])  
print(a[-1])
```

```
输出: 6  
6
```

5.1.2 二维数组索引访问

多维数组索引访问有两种表达式：

- ❑ 表达式 1: `np.array[所在 0 轴索引][所在 1 轴索引]...[所在 n-1 轴索引]`

- 表达式 2: `np.array[所在 0 轴索引, 所在 1 轴索引, ..., 所在 n-1 轴索引]`

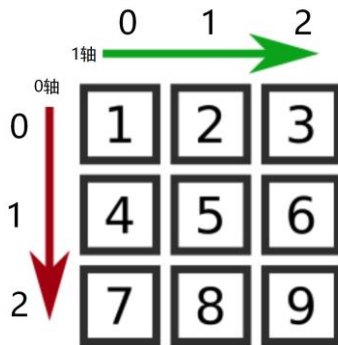


图 5-2 二维数组索引

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

```
print(b[2][1])
```

```
print(b[2, 1])
```

```
print(b[-1][-2])
```

```
print(b[-1, -2])
```

```
输出:  8
```

```
      8
```

```
      8
```

```
      8
```

5.2 切片访问

5.2.1 一维数组切片访问

NumPy 一维数组切片操作与 Python 内置序列切片操作一样。切片运算有两种形式：

- ❑ `[start:end]`。start 是开始索引，end 是结束索引。
 - ❑ `[start:end:step]`。start 是开始索引，end 是结束索引，step 是步长，步长是在切片时获取元素的间隔。步长可以为正整数，也可为负整数。
-

注意 切片包括start位置元素，但不包括end位置元素，start和end都可以省略。

```
In [10]: a = np.array([1, 2, 3, 4, 5, 6])
```

```
In [14]: a[1:3]
```

```
Out[14]: array([2, 3])
```

```
In [15]: a[:3]
```

```
Out[15]: array([1, 2, 3])
```

```
In [16]: a[0:]
```

```
Out[16]: array([1, 2, 3, 4, 5, 6])
```

```
In [17]: a[:]
```

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

```
In [18]: a[1:-1]
```

```
Out[18]: array([2, 3, 4, 5])
```

```
In [19]: a[0:3:2]
```

```
Out[19]: array([1, 3])
```

```
In [20]: a[::-1]
```

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

5.2.2 二维数组切片访问

多维数组切片访问表达式：

□ `np.array[所在 0 轴切片, 所在 1 轴切片, ..., 所在 n-1 轴切片]`

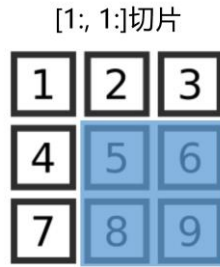


图 5-3 二维数组切片

```
In [21]: b = np.array([[1, 2, 3],
...:                  [4, 5, 6],
...:                  [7, 8, 9]])
...:

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

In [23]: b[1:, 1:]
Out[23]:
array([[5, 6],
       [8, 9]])

In [24]: b[1:, 1:] # 等价于b[1:, 1:3]
Out[24]:
array([[5, 6],
       [8, 9]])
```

多维数组切片与索引的访问表达式：

□ `np.array[所在 0 轴切片或索引, 所在 1 轴切片或索引, ..., 所在 n-1 轴切片或索引]`

[1:, 1]切片

1	2	3
4	5	6
7	8	9

图 5-4 二维数组切片与索引混合

```
In [21]: b = np.array([[1, 2, 3],
...:                  [4, 5, 6],
...:                  [7, 8, 9]])
...:
```

```
In [25]: b[1:, 1]
Out[25]: array([5, 8])
```

5.3 布尔索引

为了从数组中过滤我们想要的元素，可以使用布尔索引。

```
In [40]: # 创建一维数组
...: a1 = np.array([1, 2, 3, 4, 5, 6])
...: # 创建一维布尔数组b
...: b = np.array([False, True, False, True, False, True])
```

```
In [41]: a1[b]
Out[41]: array([2, 4, 6])
```

```
In [37]: # 创建二维数组
...: a2 = np.array([[1, 2, 3],
...:                [4, 5, 6],
```



```
...:             [7, 8, 9]))
...: # 创建与a2相同形状的布尔数组b
...: b = np.array([[True, False, False],
...:             [False, True, False],
...:             [False, False, True]])

In [38]: a2[b]
Out[38]: array([1, 5, 9])
```

注意：

1. 布尔数组必须与要索引的数组形状相同，否则引发IndexError错误。
 2. 布尔索引返回的新数组是原数组的副本，与原数组不共享相同的数据空间。新数组的修改不会影响原数组。这是所谓的**深层复制**。
-

```
输入:  # 创建一维数组
      a3 = np.array([1, 2, 3])
      # 创建一维布尔数组b
      b = np.array([False, True, True])

      c = a3[b]
      print(c)
      c[1] = 100
      print(c)
      print(a3)
```

```
输出:  [2 3]
      [ 2 100]
      [1 2 3]
```

5.4 花式索引

使用整数列表或整数数组作为数组索引，这称为“花式索引”。

5.4.1 一维数组花式索引

原始数组是一维数组, 索引数组可以是一维或多维的。

```
In [2]: # 创建一维数数组
...: a = np.array([1, 2, 3, 4, 5, 6])

In [3]: # 创建一维索引数组
...: i = np.array([1, 1, 3, 4])

In [4]: a[i] #通过索引数组i访问数组a
Out[4]: array([2, 2, 4, 5])

In [7]: # 整数列表作为索引
...: L = [1,1,3,4]

In [8]: a[L]
Out[8]: array([2, 2, 4, 5])
```

索引可以是整数数组或整数列表。

```
In [2]: # 创建一维数数组
...: a = np.array([1, 2, 3, 4, 5, 6])

In [5]: # 创建二维索引数组
...: j = np.array([[1, 1],
...:               [3, 4]])

In [6]: a[j] #通过索引数组j访问数组a
Out[6]:
array([[2, 2],
       [4, 5]])
```

索引数组可以是多维的，输出数组形状与索引数组形状相同。

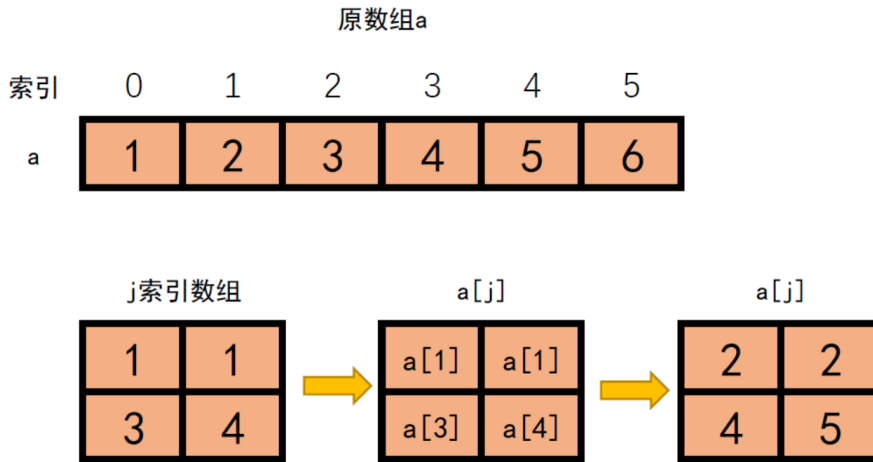


图 5-5 二维花式索引

注意：

1. 花式索引返回的新数组与花式索引数组形状相同。
2. 花式索引返回的新数组与布尔数组类似，属于深层复制。

5.4.2 二维数组花式索引

原始数组是二维数组，索引数组可以是一维或多维的。

```
In [9]:
...: b = np.arange(12).reshape(3, 4)
...: b
Out[9]:
```

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

In [10]: # 创建二维索引数组m,作为b数组的0轴索引
...: m = np.array([[1, 1],
...:               [2, 0]])

In [11]: # 创建二维索引数组n,作为b数组的1轴索引
...: n = np.array([[1, 0],
...:               [3, 2]])

In [12]: b[m, n]
Out[12]:
array([[ 5,  4],
       [11,  2]])
```

注意：每一个轴上索引数组形状相同。

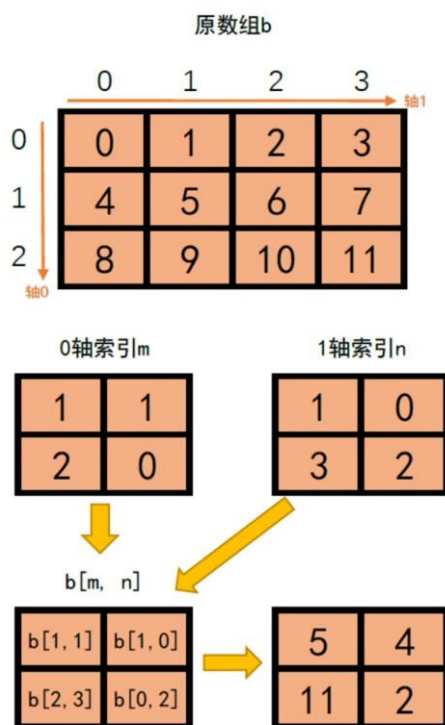


图 5-6 二维花式索引

可以混合使用整数索引和数组索引。

```
In [15]: b[m, 2] # 0轴索引使用二维数组m, 1轴索引使用整数
```

```
Out[15]:
```

```
array([[ 6,  6],
       [10,  2]])
```

```
In [16]: b[1, n] # 0轴索引使用整数, 1轴索引使用二维数组n
```

```
Out[16]:
```

```
array([[5, 4],
       [7, 6]])
```

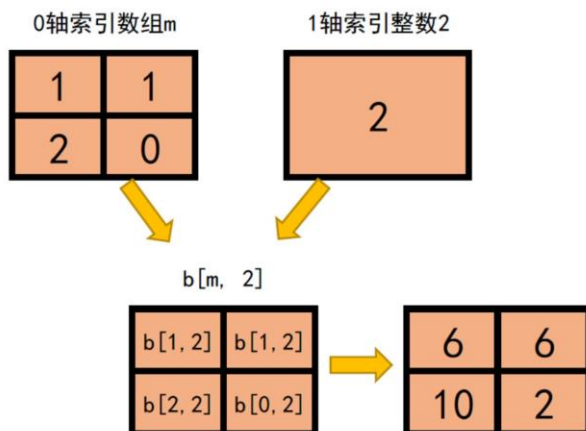


图 5-7 混合使用索引

5.5 迭代数组

通过数组的 `nditer` 类可以进行迭代。

```

输入:      # 创建二维数组
           a = np.arange(0,12).reshape(3,4)

           print('原始数组是: ')
           print(a)

           print ('迭代输出元素: ')
           for n in np.nditer(a):
               print(n, end=" ", )
    
```

```

输出:      原始数组是:
           [[ 0  1  2  3]
            [ 4  5  6  7]
            [ 8  9 10 11]]

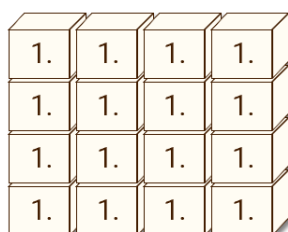
           迭代输出元素:
           0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    
```

5.6 课后练习

1、编写一个 NumPy 程序来创建一个 10x10 矩阵，其中边界上的元素是 1，内部元素是 0。

参考答案：

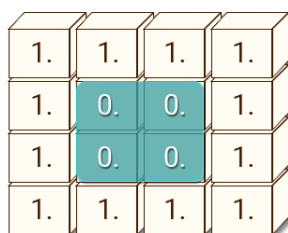
```
import numpy as np
x = np.ones((10, 10))
x[1:-1, 1:-1] = 0
print(x)
```



1.	1.	1.	1.
1.	1.	1.	1.
1.	1.	1.	1.
1.	1.	1.	1.

1 on the border and 0
inside in the array
using

$x[1:-1, 1:-1] = 0$

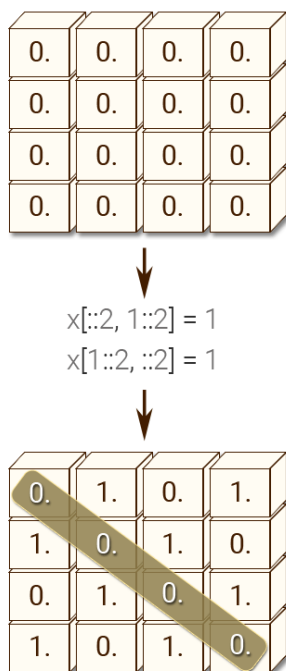


1.	1.	1.	1.
1.	0.	0.	1.
1.	0.	0.	1.
1.	1.	1.	1.

2、编写一个 NumPy 程序来创建一个 4x4 矩阵，其中 0 和 1 交错，主对角线都是 0。

参考答案：

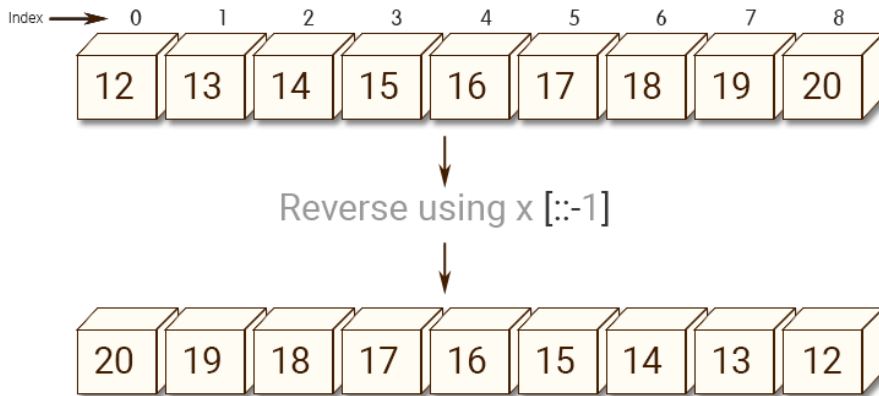
```
import numpy as np
x = np.zeros((4, 4))
x[::2, 1::2] = 1
x[1::2, ::2] = 1
print(x)
```



3、编写一个 NumPy 程序来反转一个数组（第一个元素成为最后一个）。

参考答案：

```
import numpy as np
x = np.arange(12, 38)
print("原始数组:")
print(x)
print("翻转数组:")
x = x[::-1]
print(x)
```

第6章 数组操作

6.1 连接数组

- ☐ concatenate
- ☐ hstack
- ☐ vstack

6.1.1 使用 concatenate 函数

concatenate 函数沿指定的轴连接多个数组，他的语法格式如下：

```
numpy.concatenate((a1, a2, ...), axis=0)
```

- ☐ a1, a2 是要连接的数组，除了连接指定轴外，其他轴大小（元素个数）相同。
- ☐ axis 是沿指定轴的索引，默认为 0。

```
In [38]: a = np.array([[1, 2],
...:                  [3, 4]])

In [39]: b = np.array([[5, 6]])

In [40]: ab = np.concatenate((a, b))

In [41]: ab
Out[41]:
array([[1, 2],
       [3, 4],
       [5, 6]])
```

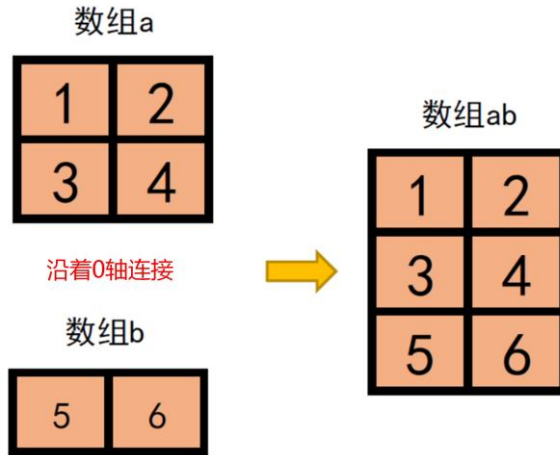


图 6-1 沿着 0 轴连接

```
In [44]: ab = np.concatenate((a, b.T), axis=1)
```

```
In [45]: ab
```

```
Out[45]:
```

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

6.1.2 使用 hstack 函数

hstack 函数沿水平堆叠多个数组, 相当于 concatenate 函数 axis=1 情况, 他的语法格式如下:

```
numpy.hstack(tup)
```

□ tup 是多个要连接数组的元组。

```
In [1]: import numpy as np

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

In [3]: b = np.array([[5],
...:                  [7]])

In [4]: ab = np.hstack((a, b))

In [5]: ab
Out[5]:
array([[1, 2, 5],
       [3, 4, 7]])
```

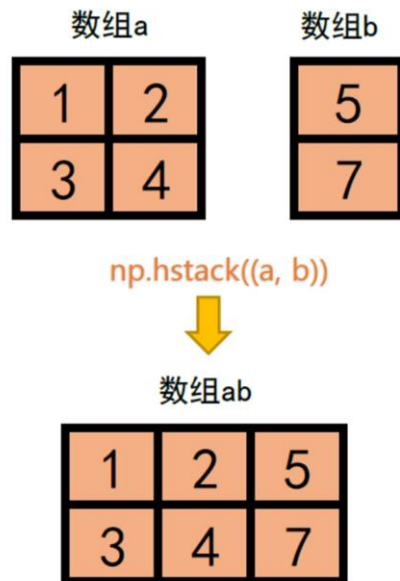


图 6-2 hstack 函数连接

注意：0轴上元素个数相同。

6.1.3 使用 vstack 函数

.vstack 函数沿垂直堆叠多个数组, 相当于 concatenate 函数 axis=0 情况, 他的语法格式如下:

```
numpy.vstack(tup)
```

□ tup 是多个要连接数组的元组。

```
In [1]: import numpy as np

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

In [3]: b = np.array([[5, 6]])

In [4]: ab = np.vstack((a, b))

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

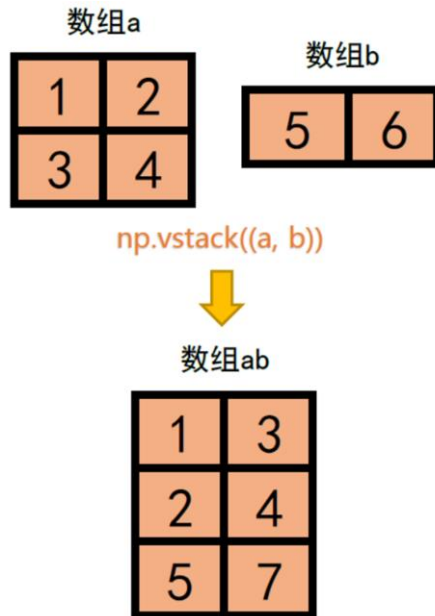


图 6-3 vstack 函数连接

注意：1轴上元素个数相同。

6.2 分割数组

- ❑ split
- ❑ hsplit
- ❑ vsplit

6.2.1 使用 split 函数

split 函数沿指定的分割多个数组，他的语法格式如下：

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

- ❑ ary 是要被分割的数组
- ❑ indices_or_sections 是一个整数或数组，如果是整数就用该数平均分割；如果是数组，则为沿指定轴切片操作（注意：包括开始元素，

不包括结束元素)

□ axis 是轴的分割方向，默认为 0。

```
In [11]: a = np.arange(9)

In [12]: b = np.split(a,3)

In [13]: b
Out[13]: [array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]

In [14]: c = np.split(a,[4,7])

In [15]: c
Out[15]: [array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]
```

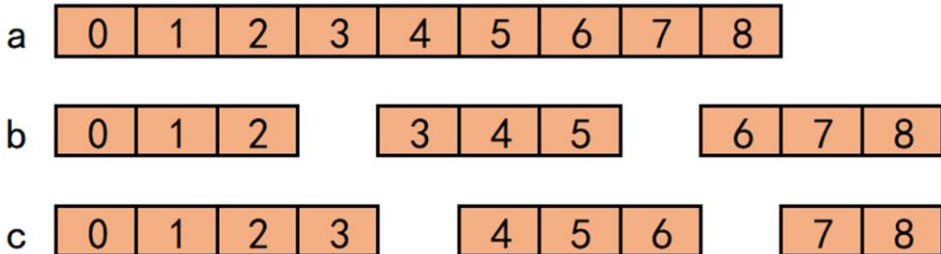


图 6-4 split 函数分割一维数组

```
In [16]: a = np.arange(9).reshape(3,3)

In [17]: b = np.split(a, 3, axis=0)

In [18]: b
Out[18]: [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]])]

In [19]: c = np.split(a, 3, axis=1)
```

```
In [20]: c
Out[20]:
[array([[0],
        [3],
        [6]]), array([[1],
        [4],
        [7]]), array([[2],
        [5],
        [8]])]
```

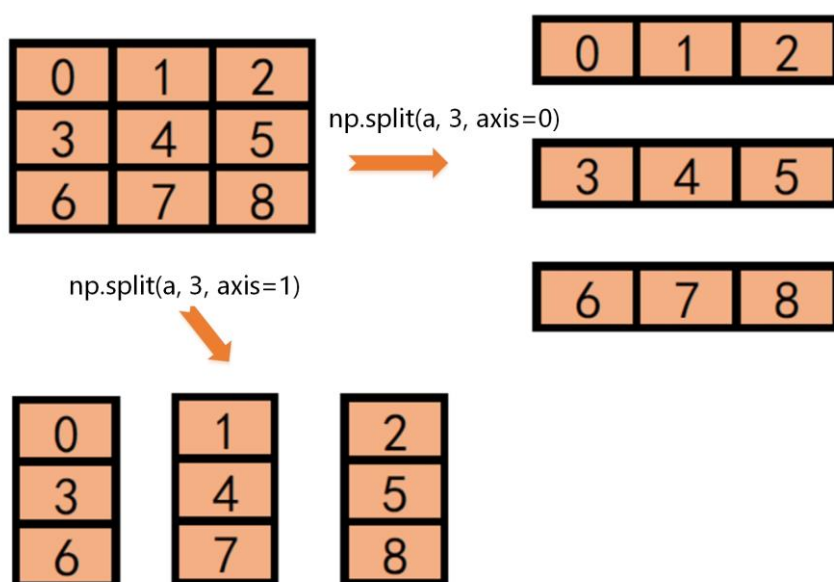


图 6-5 split 函数分割二维数组

6.2.2 使用 hsplit 函数

`hsplit` 函数沿水平方向分割数组，相当于 `split` 函数 `axis=1` 情况，他的语法格式如下：

```
numpy.hsplit(ary, indices_or_sections)
```



```
In [23]: a = np.arange(9).reshape(3,3)
```

```
In [24]: b = np.hsplit(a, 3)
```

```
In [25]: b
```

```
Out[25]:
```

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

6.2.3 使用 vsplit 函数

`vsplit` 函数沿垂直方向分割数组，相当于 `split` 函数 `axis=0` 情况，他的语法格式如下：

```
numpy.vsplit(ary, indices_or_sections)
```

```
In [26]: a = np.arange(9).reshape(3,3)
```

```
In [27]: b = np.vsplit(a, 3)
```

```
In [28]: b
```

```
Out[28]: [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7,  
8]])]
```

6.3 算术运算

NumPy 数组对象可以使用 Python 原生的算术运算符，加、减、乘、除都可以使用。

```
In [30]: a = np.array([[1, 2],  
...:                  [3, 4]])
```

```
In [31]: b = np.array([[5, 6],  
...:                  [7, 8]])
```

```
In [32]: a+b
```

```
Out[32]:
```

```
array([[ 6,  8],  
       [10, 12]])
```

```
In [33]: a+10
```

```
Out[33]:
```

```
array([[11, 12],  
       [13, 14]])
```

```
In [34]: a*b
```

```
Out[34]:
```

```
array([[ 5, 12],  
       [21, 32]])
```

```
In [35]: a*10
```

```
Out[35]:
```

```
array([[10, 20],  
       [30, 40]])
```

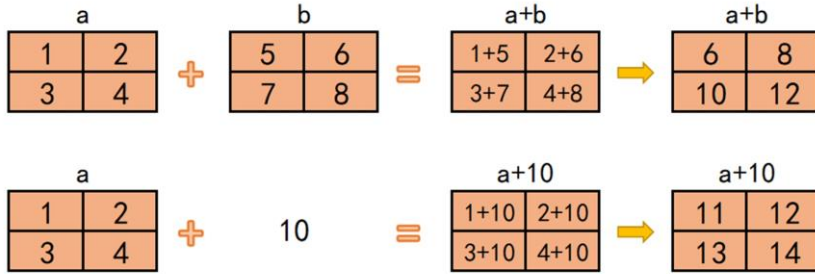


图 6-6 二维数组加法运算

6.4 广播

不同的形状数组或标量计算时发生广播。

6.4.1 标量广播

```
In [38]: a = np.array([1, 2, 3, 4])

In [39]: a*10    # 数组（向量）与标量计算
Out[39]: array([10, 20, 30, 40])

In [40]: b = np.array([[5, 6],
...:                  [7, 8]])

In [41]: b*10    # 数组（矩阵）与标量计算
Out[41]:
array([[50, 60],
       [70, 80]])
```

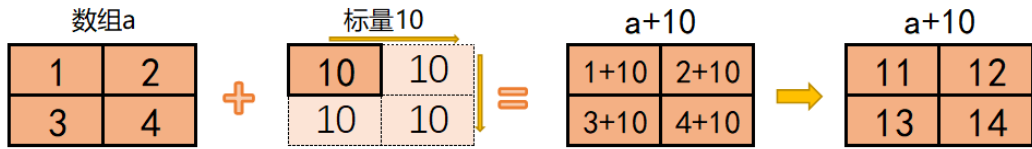


图 6-7 标量广播

6.4.2 数组广播

```
In [42]: a = np.array([1, 2])

In [43]: b = np.array([[5, 6],
...:                  [7, 8]])

In [44]: a + b
Out[44]:
array([[ 6,  8],
       [ 8, 10]])
```

广播规则：

- 1、如果两个数组维度不相等，维度较低的数组的形状（shape）会从左则开始填充 1，直到维度与高维数组相等。
- 2、如果两个数组维度相等时，要么对应轴的长度相同，要么其中一个长度为 1，则是兼容数组可以广播。长度为 1 的轴会被扩展。

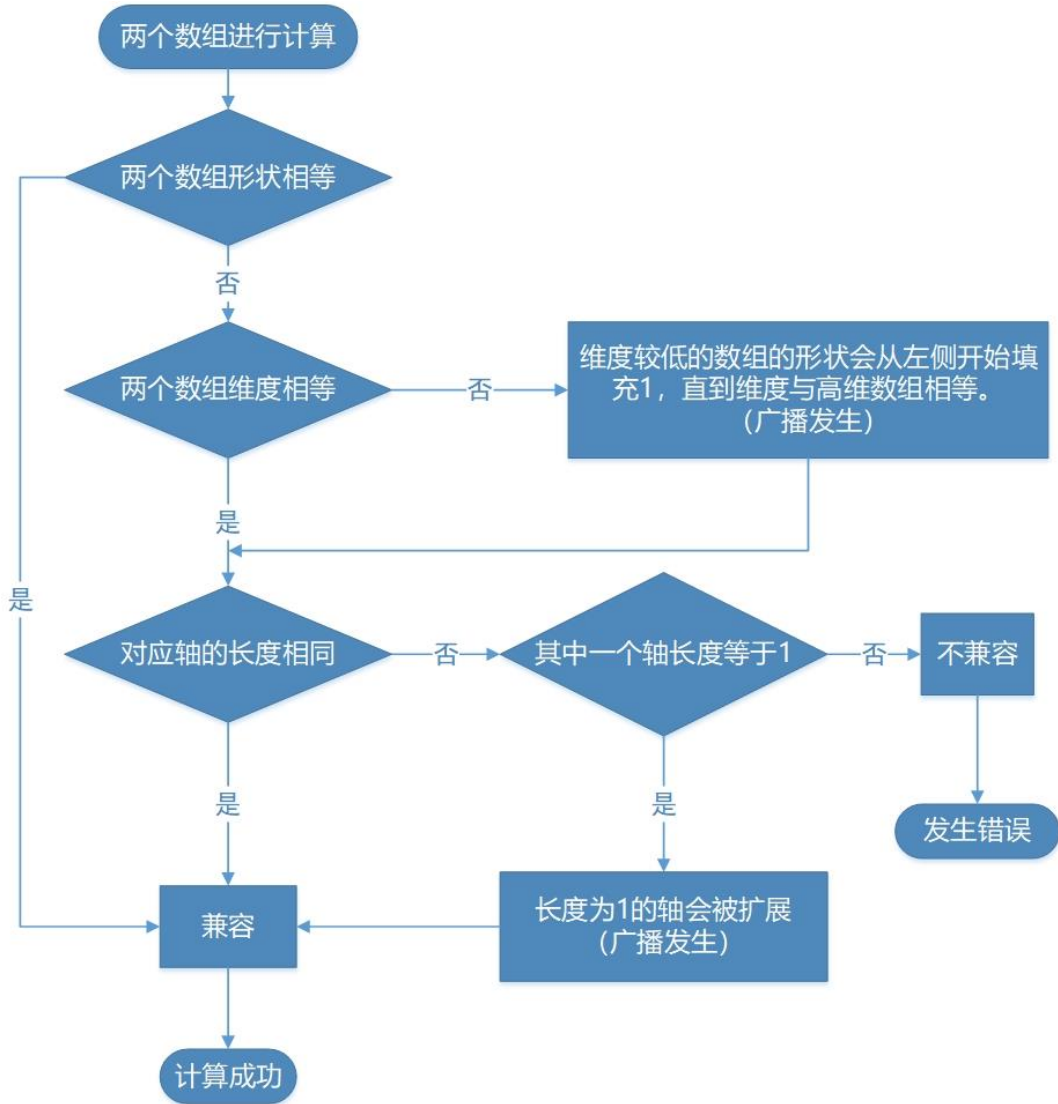


图 6-8 广播规则

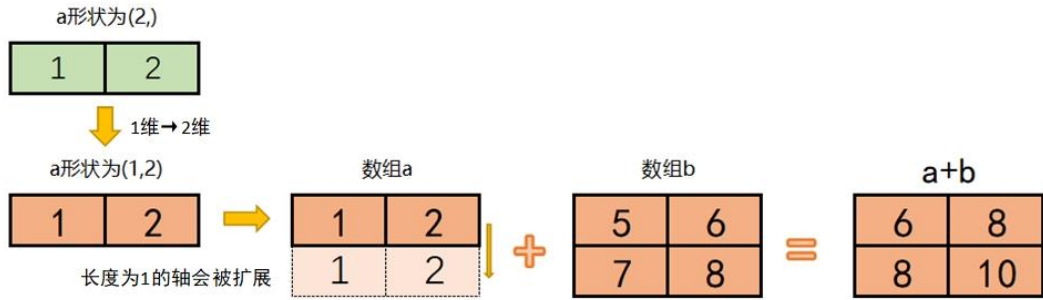


图 6-9 广播过程

6.5 课后练习

1、编写一个 NumPy 程序，将华氏度的值转换为摄氏度。华氏度值存储在 NumPy 数组中 `[0, 12, 45.21, 34, 99.91]`。注： $C = (5(F - 32)) / 9$ ，C 是摄氏度，F 是华氏度。

参考答案：

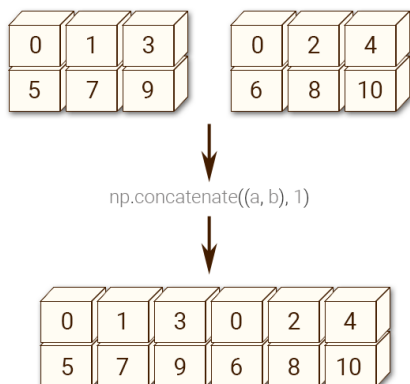
```
import numpy as np
fvalues = [0, 12, 45.21, 34, 99.91]
F = np.array(fvalues)
print("华氏度数值:")
print(F)
C = (5*F/9 - 5*32/9)
print("摄氏度数值:")
print(C)
```

2、编写一个 NumPy 程序来连接两个二维数组。两个二维数组：`[[0, 1, 3], [5, 7, 9]]`和`[[0, 2, 4], [6, 8, 10]]`。

参考答案：

```
import numpy as np
a = np.array([[0, 1, 3], [5, 7, 9]])
```

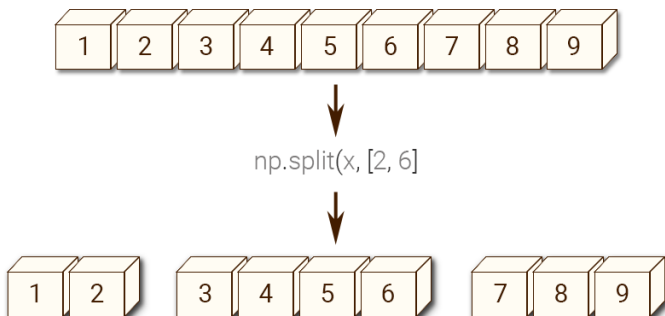
```
b = np.array([[0, 2, 4], [6, 8, 10]])
c = np.concatenate((a, b), 1)
print(c)
```



3、编写一个 NumPy 程序将 9 个元素数组拆分为 3 个数组，每个数组分别包含 2 个、4 个和 3 个元素。

参考答案：

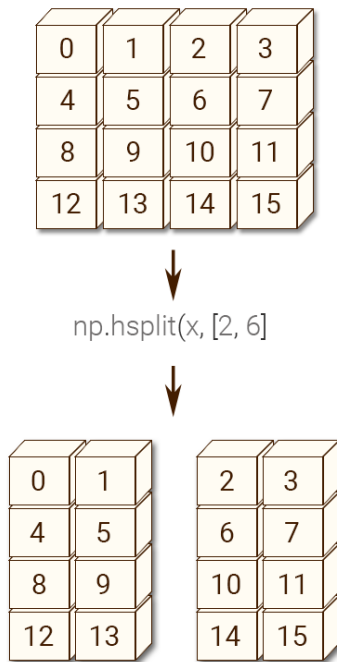
```
import numpy as np
x = np.arange(1, 10)
print("原始数组:",x)
print("分割后:")
print(np.split(x, [2, 6]))
```



4、编写一个 NumPy 程序，将形状 4x4 的数组拆分为沿第二轴的两个数组。

参考答案:

```
import numpy as np
x = np.arange(16).reshape((4, 4))
print("原始数组:",x)
print("分割后:")
print(np.hsplit(x, [2, 6]))
```



第7章 通用函数

通用函数（ufunc）使得 NumPy 数组操作作用于数组中的每一个元素。通用函数是 C 语言实现的，这样会取得更快的执行效率。

7.1 数学运算函数

7.1.1 算数运算函数

```
In [2]: import numpy as np

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

In [4]: b = np.array([[5, 6],
...:                  [7, 8]])

In [5]: a + b
Out[5]:
array([[ 6,  8],
       [ 8, 10]])

In [6]: np.add(a, b)
Out[6]:
array([[ 6,  8],
       [ 8, 10]])
```

表 7-1 算数运算

通用函数	说明
<code>add(x1, x2[, y])</code>	$y = x1 + x2$
<code>subtract(x1, x2[, y])</code>	$y = x1 - x2$

<code>multiply(x1, x2[, y])</code>	<code>y = x1 * x2</code>
<code>divide(x1, x2[, y])</code>	<code>y = x1 / x2</code>
<code>floor_divide(x1, x2[, y])</code>	<code>y = x1 // x2</code>
<code>power(x1, x2[, y])</code>	<code>y = x1 ** x2</code>

```
In [10]: np.power(a, b, c)
```

```
Out[10]:
```

```
array([[ 1, 64],
       [ 1, 256]])
```

```
In [11]: c
```

```
Out[11]:
```

```
array([[ 1, 64],
       [ 1, 256]])
```

7.1.2 关系运算函数

```
In [2]: import numpy as np
```

```
In [6]: a = np.array([1, 9])
```

```
...:
```

```
...: b = np.array([[5, 6],
```

```
...:               [7, 8]])
```

```
In [7]: a == b
```

```
Out[7]:
```

```
array([[False, False],
       [False, False]])
```

```
In [8]: a < b
```

```
Out[8]:
```

```
array([[ True, False],
       [ True, False]])
```

```

In [9]: np.equal(a, b)
Out[9]:
array([[False, False],
       [False, False]])

In [10]: np.less(a, b)
Out[10]:
array([[ True, False],
       [ True, False]])

```

表 7-2 关系运算

通用函数	说明
<code>equal(x1, x2[, y])</code>	$y = x1 == x2$
<code>not_equal(x1, x2[, y])</code>	$y = x1 != x2$
<code>less(x1, x2[, y])</code>	$y = x1 < x2$
<code>less_equal(x1, x2[, y])</code>	$y = x1 \leq x2$
<code>greater(x1, x2[, y])</code>	$y = x1 > x2$
<code>greater_equal(x1, x2[, y])</code>	$y = x1 \geq x2$

7.2 自定义通用函数

自定义通用函数 (ufunc) 数组操作将用于数组中的每一个元素，他的语法格式如下：

```
ufunc = numpy.frompyfunc(func, nin, nout)
```

- ❑ `func` 任何 Python 函数，可以是内置的也可以是自定义函数。
- ❑ `nin` 传入数组参数个数。
- ❑ `nout` 从返回数组个数。

该函数返回一个自定义的通用函数，他的类型是 `numpy.ufunc`。

1、输入 1 个参数，输出 1 个数组：

```
# 定义通用函数abs_ufunc, abs是Python内置函数
In [18]: abs_ufunc = np.frompyfunc(abs, 1, 1)
In [19]: a = np.array([[ -6,  8],
...:                  [ -8, -10]])
...:
...:

In [20]: b = abs_ufunc(a) # 调用通用函数abs_ufunc
In [21]: b
Out[21]:
array([[6, 8],
       [8, 10]], dtype=object)
```

2、输入 2 个参数，输出 2 个两个数组：

```
In [1]: import numpy as np

In [2]: # 定义add_sub函数实现两个数+和-运算，返回元组
...: def add_sub(x, y):
...:     return x + y, x -y
...:

In [3]:
...: # 定义通用函数add_sub_ufunc
...: add_sub_ufunc = np.frompyfunc(add_sub, 2, 2)

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

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

In [6]: # 调用通用函数add_sub_ufunc
...: add, sub = add_sub_ufunc(a, b)
```

```
In [7]: add
Out[7]:
array([[6, 8],
       [8, 10]], dtype=object)

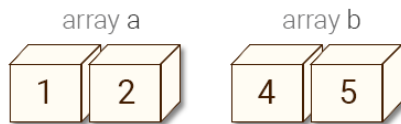
In [8]: sub
Out[8]:
array([[ -4,  -4],
       [ -6,  -6]], dtype=object)
```

7.3 课后练习

1、写一个 NumPy 程序比较两个数组。

参考答案：

```
import numpy as np
a = np.array([1, 2])
b = np.array([4, 5])
print("Array a: ",a)
print("Array b: ",b)
print("a > b")
print(np.greater(a, b))
print("a >= b")
print(np.greater_equal(a, b))
print("a < b")
print(np.less(a, b))
print("a <= b")
print(np.less_equal(a, b))
```



$a > b$
[False False]
 $a \geq b$
[False False]
 $a < b$
[True True]
 $a \leq b$
[True True]

2、写一个 NumPy 程序自定义一个通用函数，实现两个数组元素的翻倍计算。

参考答案：

```
import numpy as np

# 定义double函数
def double(x):
    return 2 * x

# 定义通用函数double_ufunc
double_ufunc = np.frompyfunc(double, 1, 1)
a = np.array([1, 2])
# 调用通用函数double_ufunc
b = double_ufunc(a)
print(b)
```

第8章 更多函数

8.1 随机数

8.1.1 常用随机数

- ❑ `numpy.random.rand(d0, d1, ..., dn)`。返回`[0.0, 1.0)`随机浮点数，即大于等于 0.0，且小于 1.0。
- ❑ `numpy.random.randint(low, high=None, size=None, dtype='l')`。返回`[low, high)`随机整数，如果 `high` 省略则返回`[0, low)`随机整数。

```
In [2]: a = np.random.rand(10)
```

```
In [3]: a
```

```
Out[3]:
```

```
array([0.33405906, 0.02429641, 0.03419599, 0.03095278,  
       0.25486166,  
       0.9252782 , 0.04851661, 0.77581319, 0.63205091,  
       0.88033074])
```

```
In [4]: b = np.random.rand(3,2)
```

```
In [5]: b
```

```
Out[5]:
```

```
array([[0.92538361, 0.92555885],  
       [0.62156198, 0.83121134],  
       [0.53577988, 0.66758233]])
```

```
In [6]: c = np.random.randint(5, size=(3,2))
```

```
In [7]: c
```

```
Out[7]:
```

```
array([[3, 0],
```



```
[0, 4],
[3, 2]])

In [8]: d = np.random.randint(5, 10, size=(3,2))

In [9]: d
Out[9]:
array([[7, 5],
       [5, 8],
       [6, 8]])
```

8.1.2 正态分布随机数

- ❑ `numpy.random.randn(d0, d1, ..., dn)`。返回标准正态分布随机数，参数与 `rand` 函数相同。
- ❑ `numpy.random.normal(loc=0.0, scale=1.0, size=None)`。返回正态分布随机数，`loc` 平均值，`scale` 标准差。

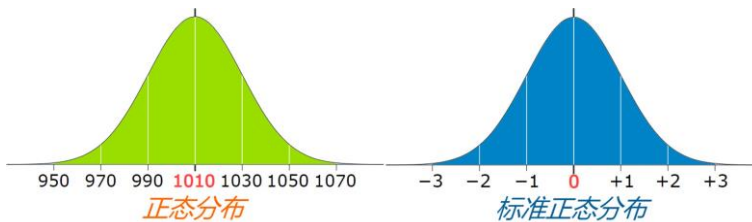


图 8-1 正态分布和标准正态分布

```
In [0]: a = np.random.randn(10)

In [1]: a
Out[1]:
array([-0.41209317,  0.68440156, -2.18403987,  0.51520253, -
        1.0088911 , ...])
```

```
-0.00747258, -1.33538674, -0.32828439, 0.07056826,  
1.01960763]])
```

```
In [2]: b = np.random.randn(3,2)
```

```
In [3]: b
```

```
Out[3]:
```

```
array([[ -0.90417267, -0.54450007],  
       [ -0.38576724,  1.22427304],  
       [ 0.05314471, -0.48759674]])
```

```
In [4]: c = np.random.normal(1010, 30, size=(3,2))
```

```
In [5]: c
```

```
Out[5]:
```

```
array([[1050.94428536, 1029.38285544],  
       [ 970.79769053, 1008.567652 ],  
       [1005.71159515, 1016.64185377]])
```

```
In [6]: c = np.random.normal(1010, 30, size=(3,2))
```

```
In [7]: c
```

```
Out[7]:
```

```
array([[1012.19502321, 989.26751417],  
       [1020.83296905, 1023.62452665],  
       [ 984.2371164 , 970.88248651]])
```

8.2 排序

8.2.1 轴排序

按照轴对数组进行排序函数 `sort`，他的语法格式如下：

```
numpy.sort(a, axis=-1, kind='quicksort', order=None)
```

- ❑ `a` 要排序的数组。
- ❑ `axis` 排序的轴索引，默认是 `-1` 表示最后一个轴。
- ❑ `kind` 排序类型。'quicksort'（快速排序）默认值最快。另外，'mergesort'（归并排序）和'heapsort'（堆排序）。
- ❑ `order` 排序字段。

```
In [3]: a = np.random.randint(0, 10, size=(3,4))
```

```
In [4]: a
```

```
Out[4]:
```

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

```
In [5]: b = np.sort(a)
```

```
In [6]: b
```

```
Out[6]:
```

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

```
In [7]: c = np.sort(a, axis=0)
```

```
In [8]: c
```

```
Out[8]:
```

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

8.2.2 轴排序索引

按照轴对数组进行排序索引函数 `argsort`，他的语法格式如下：

```
numpy.argsort(a, axis=-1, kind='quicksort', order=None)
```

```
In [37]: a = np.random.randint(0, 10, size=(3,4))
```

```
In [38]: a
```

```
Out[38]:
```

```
array([[1, 9, 4, 5],  
       [5, 9, 6, 3],  
       [7, 9, 6, 9]])
```

```
In [39]: bidx = np.argsort(a)
```

```
In [40]: bidx
```

```
Out[40]:
```

```
array([[0, 2, 3, 1],  
       [3, 0, 2, 1],  
       [2, 0, 1, 3]], dtype=int64)
```

```
In [41]: cidx = np.argsort(a, axis=0)
```

```
In [42]: cidx
```

```
Out[42]:
```

```
array([[0, 0, 0, 1],  
       [1, 1, 1, 0],  
       [2, 2, 2, 2]], dtype=int64)
```

8.3 聚合函数

可以对整个数组元素或对轴元素进行计算，获得单一值，这是聚合函数。如：sum、amin、amax、mean（平均值）、average（加权平均值）、var（方差）、std（标准偏差）等。

8.3.1 求和

求和函数可以使用 `numpy.sum` 函数或数组的 `numpy.ndarray.sum` 方法。

1、`numpy.sum` 函数语法格式如下：

```
numpy.sum(a, axis=None)
```

- ❑ `a` 要求和的数组。
- ❑ `axis` 指定轴索引，如果 `axis` 没有指定是所有元素之和，如果指定轴则是该轴上的所有元素之和。

2、`numpy.ndarray.sum` 方法，类似于语法 `numpy.sum` 函数，语法如下：

```
numpy.ndarray.sum(axis=None)
```

```
In [2]: a = np.array([[5, 6],  
...:                  [7, 8]])
```

```
In [3]: a  
Out[3]:  
array([[5, 6],  
       [7, 8]])
```

```
In [4]: np.sum(a)  
Out[4]: 26
```

```
In [5]: np.sum(a, axis=-1)  
Out[5]: array([11, 15])
```

```
In [6]: np.sum(a, axis=0)  
Out[6]: array([12, 14])
```

8.3.2 最大值

求最大值可以使用 `numpy.amax` 函数、`numpy.nanmax` 函数或数组的 `ndarray.max` 方法。

1、`numpy.amax` 函数语法如下：

```
numpy.amax(a, axis=None)
```

2、`numpy.nanmax` 函数忽略 NaN (Not a Number, 非数)，语法如下：

```
numpy.nanmax(a, axis=None)
```

3、`numpy.ndarray.max` 方法，类似于语法 `numpy.amax` 函数，语法如下：

```
numpy.ndarray.max(axis=None)
```

```
In [5]: a = np.array([[5, 6],  
...:                  [7, 8]])
```

```
In [6]: np.amax(a)  
Out[6]: 8
```

```
In [7]: np.amax(a, axis=-1)  
Out[7]: array([6, 8])
```

```
In [8]: np.amax(a, axis=0)  
Out[8]: array([7, 8])
```

```
In [9]: a.max()  
Out[9]: 8
```

```
In [10]: a.max(axis=-1)
```

```
Out[10]: array([6, 8])

In [11]: a.max(axis=0)
Out[11]: array([7, 8])

In [12]: b = np.array([[5, 6],
...:                   [7, np.nan]])

In [12]: np.nanmax(b)
Out[12]: 7.0

In [13]: np.nanmax(b, axis=-1)
Out[13]: array([6., 7.])

In [14]: np.nanmax(b, axis=0)
Out[14]: array([7., 6.] )
```

8.3.3 最小值

求最小值可以使用 `numpy.amin` 函数、`numpy.nanmin` 函数或数组的 `ndarray.min` 方法。

1、`numpy.amin` 函数语法如下：

```
numpy.amin(a, axis=None)
```

2、`numpy.nanmin` 函数忽略 NaN，语法如下：

```
numpy.nanmin(a, axis=None)
```

3、`numpy.ndarray.min` 方法，类似于语法 `numpy.amin` 函数，语法如下：

```
numpy.ndarray.min(axis=None)
```

```
In [20]: a = np.array([[5, 6],  
...:                  [7, 8]])
```

```
In [21]: np.amin(a)
```

```
Out[21]: 5
```

```
In [22]: np.amin(a, axis=-1)
```

```
Out[22]: array([5, 7])
```

```
In [23]: np.amin(a, axis=0)
```

```
Out[23]: array([5, 6])
```

```
In [24]: a.min()
```

```
Out[24]: 5
```

```
In [25]: a.min(axis=-1)
```

```
Out[25]: array([5, 7])
```

```
In [26]: a.min(axis=0)
```

```
Out[26]: array([5, 6])
```

```
In [27]: b = np.array([[5, 6],  
...:                  [7, np.nan]])  
...:
```

```
In [28]: np.nanmin(b)
```

```
Out[28]: 5.0
```

```
In [29]: np.nanmin(b, axis=-1)
```

```
Out[29]: array([5., 7.])
```

```
In [30]: np.nanmin(b, axis=0)
```

```
Out[30]: array([5., 6.])
```


8.3.4 mean (平均值)

平均值可以使用 `numpy.mean` 函数、`numpy.nanmean` 函数或数组的 `numpy.ndarray.mean` 方法。

1、`numpy.mean` 函数语法如下：

```
numpy.mean(a, axis=None)
```

2、`numpy.nanmean` 函数忽略 NaN，语法如下：

```
numpy.nanmean(a, axis=None)
```

3、`numpy.ndarray.mean` 方法，类似于语法 `numpy.mean` 函数，语法如下：

```
numpy.ndarray.mean(axis=None)
```

```
In [5]: a = np.array([[5, 6],  
...:                  [7, 8]])
```

```
In [6]: np.mean(a)  
Out[6]: 6.5
```

```
In [7]: np.mean(a, axis=-1)  
Out[7]: array([5.5, 7.5])
```

```
In [8]: np.mean(a, axis=0)  
Out[8]: array([6., 7.])
```

```
In [9]: a.mean()  
Out[9]: 6.5
```

```
In [10]: a.mean(axis=-1)
```

```
Out[10]: array([5.5, 7.5])

In [11]: a.mean(axis=0)
Out[11]: array([6., 7.])

In [12]: b = np.array([[5, 6],
...:                   [7, np.nan]])

In [13]: np.nanmean(b)
Out[13]: 6.0

In [14]: np.nanmean(b, axis=-1)
Out[14]: array([5.5, 7. ])

In [15]: np.nanmean(b, axis=0)
Out[15]: array([6., 6.])
```

8.3.5 average (加权平均值)

加权平均值可以使用 `numpy.average` 函数，语法如下：

```
numpy.average(a, axis=None, weights=None)
```

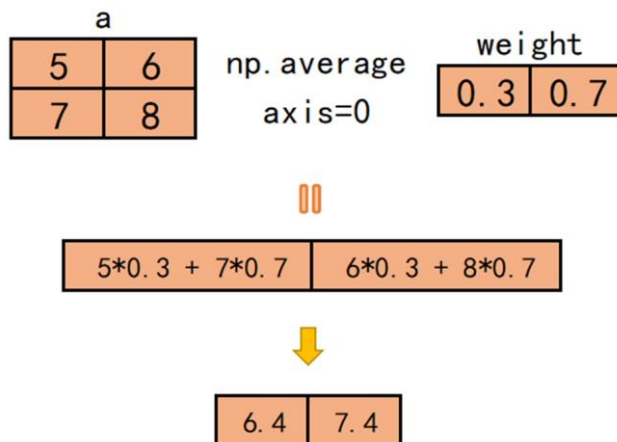


图 8-2 加权平均值

```
In [1]: a = np.array([[5, 6],
...:                  [7, 8]])
...:
...:

In [2]: np.average(a)
Out[2]: 6.5

In [3]: np.average(a, axis=0)
Out[3]: array([6., 7.])

In [4]: np.average(a, axis=-1)
Out[4]: array([5.5, 7.5])

In [5]: np.average(a, axis=0, weights=[0.3,0.7])
Out[5]: array([6.4, 7.4])

In [6]: np.average(a, axis=0, weights=[0.2,0.8])
Out[6]: array([6.6, 7.6])

In [7]: np.average(a, axis=-1, weights=[0.2,0.8])
Out[7]: array([5.8, 7.8])
```

8.4 unique 函数

`unique` 函数可以剔除数组中重复的元素，并按照从小到大的顺序排列。语法如下：

```
numpy.unique(ar, return_index=False, axis=None)
```

- ❑ `ar` 原始数组。
- ❑ `return_index` 设置为 `Ture`，返回原始数组中的索引数组。
- ❑ `axis` 指定轴索。如果没有指定，多维数组会平展进行排序。

```

In [3]: import numpy as np

[4]: L = [x for x in 'Hello']

In [5]: L
Out[5]: ['H', 'e', 'l', 'l', 'o']

In [6]: a = np.array(L)

In [7]: a
Out[7]: array(['H', 'e', 'l', 'l', 'o'], dtype='<U1')

In [8]: u = np.unique(a)

In [9]: u
Out[9]: array(['H', 'e', 'l', 'o'], dtype='<U1')

In [10]: u, idx = np.unique(a, return_index=True)

In [11]: u
Out[11]: array(['H', 'e', 'l', 'o'], dtype='<U1')

In [12]: idx
Out[12]: array([0, 1, 2, 4], dtype=int64)

```

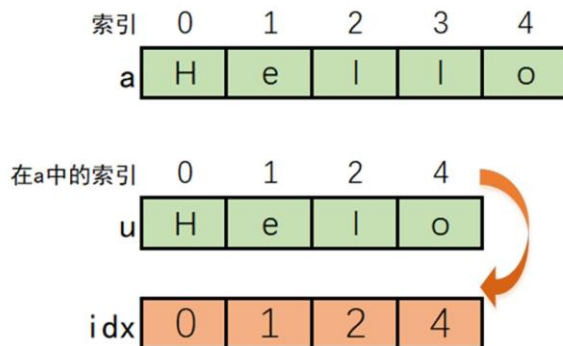


图 8-3 unique 函数

```
In [16]: a = np.array([[1, 0, 0],
                      [1, 0, 0],
                      [2, 1, 4]])
In [17]: u = np.unique(a) # 没有指定轴，展开数组进行处理

In [18]: u
Out[18]: array([0, 1, 2, 4])

In [19]: u = np.unique(a, axis=0) # 指定0轴，剔除0轴上重复的一维数组
In [20]: u
Out[20]:
array([[1, 0, 0],
       [2, 1, 4]])
```

8.5 where 函数

where 函数相当于三元运算符，他的语法格式如下：

```
numpy.where(condition[, x, y])
```

- ❑ condition 是条件，如果为 True 返回 x，否则返回 y。
- ❑ x 和 y 可以是标量或数组。

```
In [21]: a = np.arange(5)

In [22]: a
Out[22]: array([0, 1, 2, 3, 4])

In [23]: b = np.where(a < 3, a, a + 100)

In [24]: b
Out[24]: array([ 0,  1,  2, 103, 104])
```

```
In [25]: a = np.array([[1, 0, 0],
...:                  [1, 0, 0],
...:                  [2, 1, 4]])

In [26]: b = np.where(a == 0, a + 10, a)

In [27]: b
Out[27]:
array([[ 1, 10, 10],
       [ 1, 10, 10],
       [ 2,  1,  4]])
```

8.6 课后练习

1、编写 NumPy 程序以获取数组的唯一元素。

参考答案：

```
import numpy as np
x = np.array([10, 10, 20, 20, 30, 30])
print("原始数组:")
print(x)
print("唯一元素的数组:")
print(np.unique(x))
x = np.array([[1, 1], [2, 3]])
print("原始数组:")
print(x)
print("唯一元素的数组:")
print(np.unique(x))
```

2、编写 NumPy 程序从数组中沿给定轴取出最大值和最小值。

参考答案：

```
import numpy as np
x = np.array([1, 2, 3, 4, 5, 6])
print("原始数组: ",x)
print("最大值: ",np.amax(x))
print("最小值: ",np.amin(x))
```

3、编写一个 NumPy 程序，对数组[[2, 5], [4, 4]]的第一个轴和最后一个轴进行排序。

参考答案：

```
import numpy as np
a = np.array([[4, 6],[2, 1]])
print("原始数组:")
print(a)
print("沿第一轴排序:")
x = np.sort(a, axis=0)
print(x)
print("沿最后一个轴排序:")
y = np.sort(x, axis=-1)
print(y)
```

第9章 线性代数

NumPy 提供线性代数计算模块 `numpy.linalg`，该模块包含了线性代数所需的所有功能。

9.1 矩阵点乘

两个矩阵（二维数组）可以进行点乘积：

$$\begin{array}{c} \mathbf{A} \quad \times \quad \mathbf{B} \quad = \quad \mathbf{C} \\ \left(\begin{array}{cc} \mathbf{A0} & \mathbf{A1} \\ \mathbf{A2} & \mathbf{A3} \end{array} \right) \times \left(\begin{array}{cc} \mathbf{B0} & \mathbf{B1} \\ \mathbf{B2} & \mathbf{B3} \end{array} \right) = \left(\begin{array}{cc} \mathbf{C0} & \mathbf{C1} \\ \mathbf{C2} & \mathbf{C3} \end{array} \right) \\ \\ \mathbf{C0} = \mathbf{A0} \times \mathbf{B0} + \mathbf{A1} \times \mathbf{B2} \\ \mathbf{C1} = \mathbf{A0} \times \mathbf{B1} + \mathbf{A1} \times \mathbf{B3} \\ \\ \mathbf{C2} = \mathbf{A2} \times \mathbf{B0} + \mathbf{A3} \times \mathbf{B2} \\ \mathbf{C3} = \mathbf{A2} \times \mathbf{B1} + \mathbf{A3} \times \mathbf{B3} \end{array}$$

图 9-1 两个矩阵点乘积

矩阵点乘可以使用 `numpy.dot` 函数或数组的 `numpy.ndarray.dot` 方法。

1、`numpy.dot` 函数语法如下：

```
numpy.dot(a, b, out=None)
```

2、`numpy.ndarray.dot` 方法，类似于语法 `numpy.dot` 函数，语法如下：

```
ndarray.dot(b, out=None)
```

使用 `numpy.dot` 函数：


```
In [1]: import numpy as np
```

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

```
In [3]: b = np.array([[5,6],  
...:                  [7,8]])
```

```
In [4]: np.dot(a,b)
```

```
Out[4]:  
array([[19, 22],  
       [43, 50]])
```

```
In [5]: np.dot(a,b, out=a)
```

```
Out[5]:  
array([[19, 22],  
       [43, 50]])
```

```
In [6]: a
```

```
Out[6]:  
array([[19, 22],  
       [43, 50]])
```

使用 `numpy.ndarray.dot` 方法:

```
In [1]: import numpy as np
```

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

```
In [3]: b = np.array([[5,6],  
...:                  [7,8]])
```

```
In [4]: a.dot(b)
```

```
Out[4]:
array([[19, 22],
       [43, 50]])

In [5]: a.dot(b, out=a)
Out[5]:
array([[19, 22],
       [43, 50]])

In [6]: a
Out[6]:
array([[19, 22],
       [43, 50]])
```

9.2 矩阵行列式

2 阶行列式公式如下：

$$|A| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

矩阵行列式函数 `numpy.linalg.det`，语法如下：

```
numpy.linalg.det(a)
```

```
In [1]: import numpy as np

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

In [3]: np.linalg.det(a)
```

```
Out[3]: -2.0000000000000004
```

```
1*4 - 2*3 = -2
```

9.3 逆矩阵

逆矩阵： $AB=BA=E$ ，则称 B 是 A 的逆矩阵，而 A 则被称为可逆矩阵。 E 为单位矩阵。

逆矩阵函数 `numpy.linalg.inv`，语法如下：

```
numpy.linalg.inv(a)
```

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

```
In [3]: # A的逆矩阵B
...: B = np.linalg.inv(A)
```

```
In [4]: B
```

```
Out[4]:
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
```

```
In [5]: i = np.dot(A, B) # i是单位矩阵
```

```
In [6]: i
```

```
Out[6]:
array([[1.0000000e+00, 0.0000000e+00],
       [8.8817842e-16, 1.0000000e+00]])
```

```
In [7]: np.allclose(i, np.eye(2)) # 判断单位矩阵  
Out[7]: True
```

9.4 课后练习

1、编写一个 NumPy 程序来计算两个给定矩阵的乘法。两个矩阵如下：

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

参考答案：

```
import numpy as np  
p = [[1, 0], [0, 1]]  
q = [[1, 2], [3, 4]]  
print("原始数组矩阵:")  
print(p)  
print(q)  
result = np.dot(p, q)  
print("矩阵乘法的结果:")  
print(result)
```

$p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $q = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$$\begin{bmatrix} 1*1 + 0*3 & 1*2 + 0*4 \\ 0*1 + 1*3 & 0*2 + 1*4 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2、编写 NumPy 程序来计算给定方阵的行列式。

参考答案：

```
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 0], [1, 2]])
print("原始数组矩阵:")
print(a)
print("矩阵行列式:")
print(LA.det(a))
```

第10章 高维数组

10.1 创建高维数组

1、创建三维数组：

```
In [1]: import numpy as np

# 创建3维数组
In [2]: a = np.array([[[10, 11, 12],
...:                  [13, 14, 15],
...:                  [16, 17, 18]],
...:                  [[20, 21, 22],
...:                  [23, 24, 25],
...:                  [26, 27, 28]],
...:                  [[30, 31, 32],
...:                  [33, 34, 35],
...:                  [36, 37, 38]]])

In [3]: a
Out[3]:
array([[[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[20, 21, 22],
        [23, 24, 25],
        [26, 27, 28]],

       [[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38]]])
```

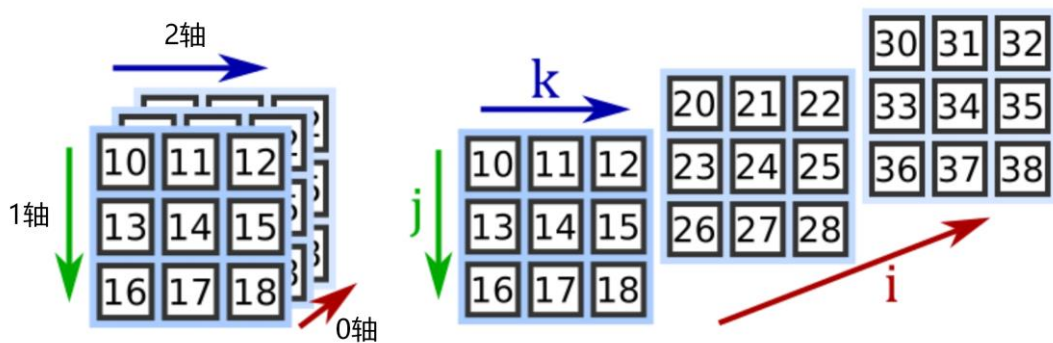


图 10-1 三维数组

2、创建四维数组：

```
In [1]: import numpy as np

In [2]: b = np.arange(81).reshape(3, 3, 3, 3)

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

        [[ 9, 10, 11],
          [12, 13, 14],
          [15, 16, 17]],

        [[18, 19, 20],
          [21, 22, 23],
          [24, 25, 26]]],

       ...
      ]])
```

10.2 访问高维数组元素

10.2.1 索引访问高维数组

索引访问高维数组元素 `a[i, j, k]`

- `a[i, j, k]` 返回单个数值（标量）
- `a[i, j]` 返回一维数组（向量）
- `a[i]` 返回二维数组（矩阵）

```
In [1]: import numpy as np
...:

In [2]: a = np.array([[[10, 11, 12],
...:                  [13, 14, 15],
...:                  [16, 17, 18]],
...:                  [[20, 21, 22],
...:                  [23, 24, 25],
...:                  [26, 27, 28]],
...:                  [[30, 31, 32],
...:                  [33, 34, 35],
...:                  [36, 37, 38]]])

In [3]: a[2, 0, 1]
Out[3]: 31

In [4]: a[1, 2]
Out[4]: array([26, 27, 28])

In [5]: a[2]
Out[5]:
```



```
array([[30, 31, 32],  
       [33, 34, 35],  
       [36, 37, 38]])
```

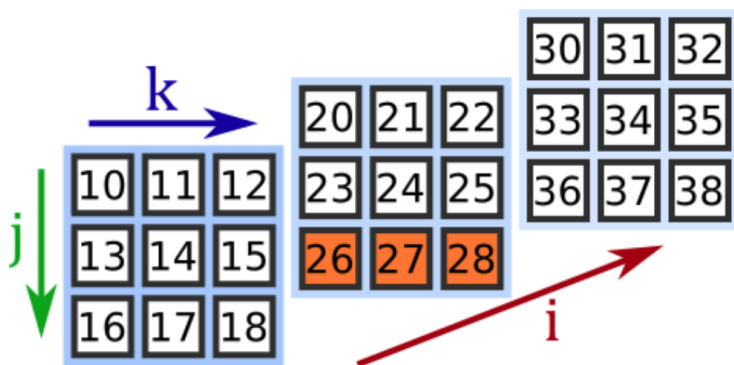


图 10-2 返回一维数组（向量）

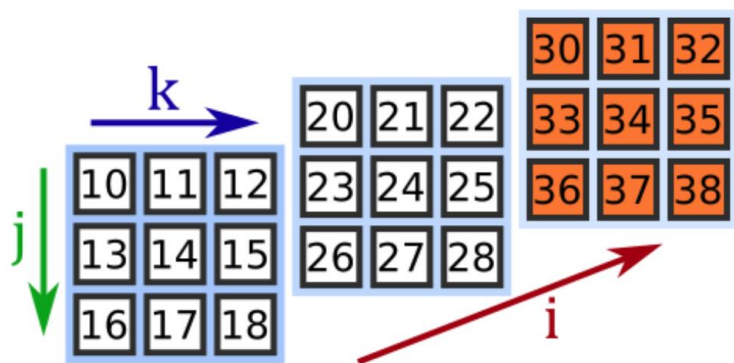


图 10-3 返回二维数组（矩阵）

10.2.2 切片访问高维数组

```
In [21]: a[:, 1, 2]
```

```
Out[21]: array([15, 25, 35])
```

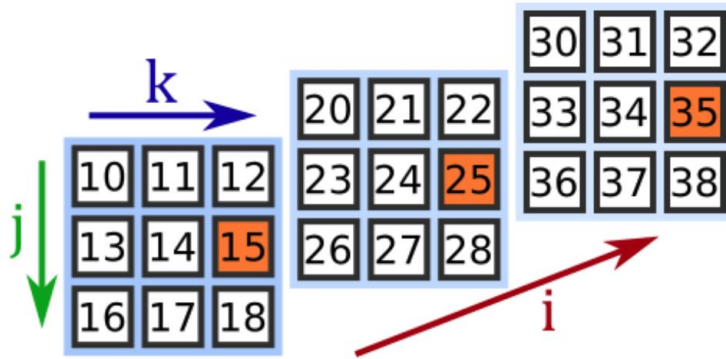


图 10-4 切片访问返回向量

```
In [22]: a[:, 1]
```

```
Out[22]:
```

```
array([[13, 14, 15],
       [23, 24, 25],
       [33, 34, 35]])
```

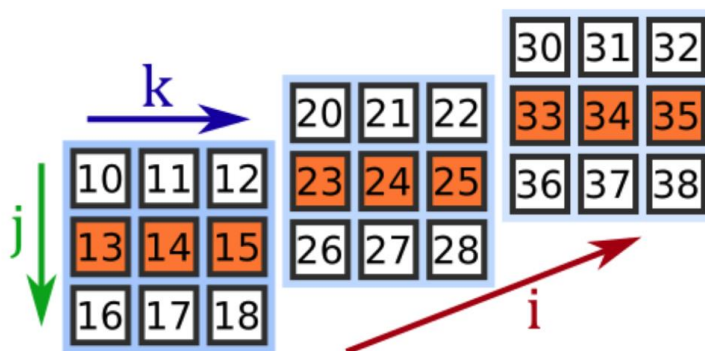


图 10-5 切片访问返回矩阵 1

```
In [2]: a[:, :, 0]
Out[2]:
array([[10, 13, 16],
       [20, 23, 26],
       [30, 33, 36]])
```

```
In [3]: a[...,0]
Out[3]:
array([[10, 13, 16],
       [20, 23, 26],
       [30, 33, 36]])
```

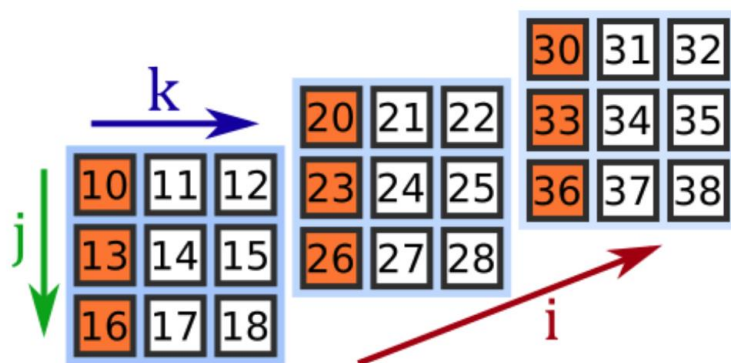


图 10-6 切片访问返回矩阵 2

```
In [27]: a[:2,1,:2]
Out[27]:
array([[[13, 14],
       [16, 17]],

       [[23, 24],
       [26, 27]]])
```

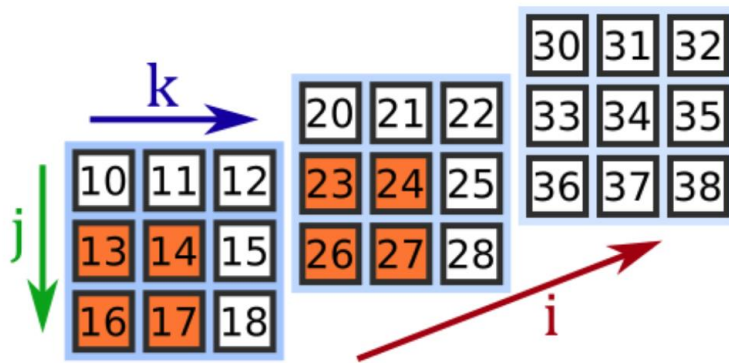


图 10-7 切片访问返回矩阵 3

10.3 课后练习

1、编写一个 NumPy 程序来创建一个 3x3x3 数组，并填充任意值。

参考答案：

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

第11章 数组的保存与读取

11.1 数组的保存

NumPy 数组可以保存到二进制或文本文件中，保存到二进制文件函数如下：

- ☐ `numpy.save`
- ☐ `numpy.savez`
- ☐ `numpy.savez_compressed`

1、`numpy.save` 函数，语法格式如下：

```
numpy.save(file, arr, allow_pickle=True, fix_imports=True)
```

- ☐ `file` 文件名/文件路径
- ☐ `arr` 要存储的数组
- ☐ `allow_pickle` 布尔值，允许使用 Python pickles 保存对象数组（可选参数，默认即可）
- ☐ `fix_imports` 为了方便 Python2 中读取 Python3 保存的数据（可选参数，默认即可）

```
In [28]: a = np.arange(9).reshape(3, 3)
```

```
In [29]: a
```

```
Out[29]:
```

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

```
In [30]: np.save('arry_save1', a)
```

2、numpy.savez 函数，语法格式如下：

```
numpy.savez(file, *args, **kwds)
```

□ file 文件名/文件路径

```
In [35]: a = np.arange(9).reshape(3, 3)
In [36]: b = np.arange(10)
In [37]: np.savez('arry_savez1', arry_a=a, arry_b=b)
```

3、numpy.savez_compressed 函数，语法格式如下：

```
numpy.savez_compressed(file, *args, **kwds)
```

□ file 文件名/文件路径

```
In [35]: a = np.arange(9).reshape(3, 3)
In [36]: b = np.arange(10)
In [38]: np.savez_compressed('arry_savez_compressed1', arry_a=a, arry_b=b)
```

11.2 数组的读取

NumPy 数组读取二进制文件函数 load，语法格式如下：

```
numpy.load(file, mmap_mode=None, allow_pickle=True,
fix_imports=True, encoding='ASCII')
```

```
In [1]: import numpy as np
...:
```

```
In [2]: a1 = np.load('arry_save1.npy')

In [3]: a1
Out[3]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

In [4]: data = np.load('arry_savez1.npz')

In [5]: data['arry_a']
Out[5]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

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

In [7]: data = np.load('arry_savez_compressed1.npz')

In [8]: data['arry_a']
Out[8]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

In [9]: data['arry_b']
Out[9]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

11.3 课后练习

1、编写一个 NumPy 程序，将两个给定的数组以压缩格式（. npz 格式）保存到一个文件中并加载它。

参考答案:

```
import numpy as np
import os
x = np.arange(10)
y = np.arange(11, 20)
print("原始数组:")
print(x)
print(y)
np.savez('temp_arra.npz', x=x, y=y)
print("加载数组从'temp_arra.npz'文件:")
with np.load('temp_arra.npz') as data:
    x2 = data['x']
    y2 = data['y']
    print(x2)
    print(y2)
```