5 Python 数据类型

Data Types in Python

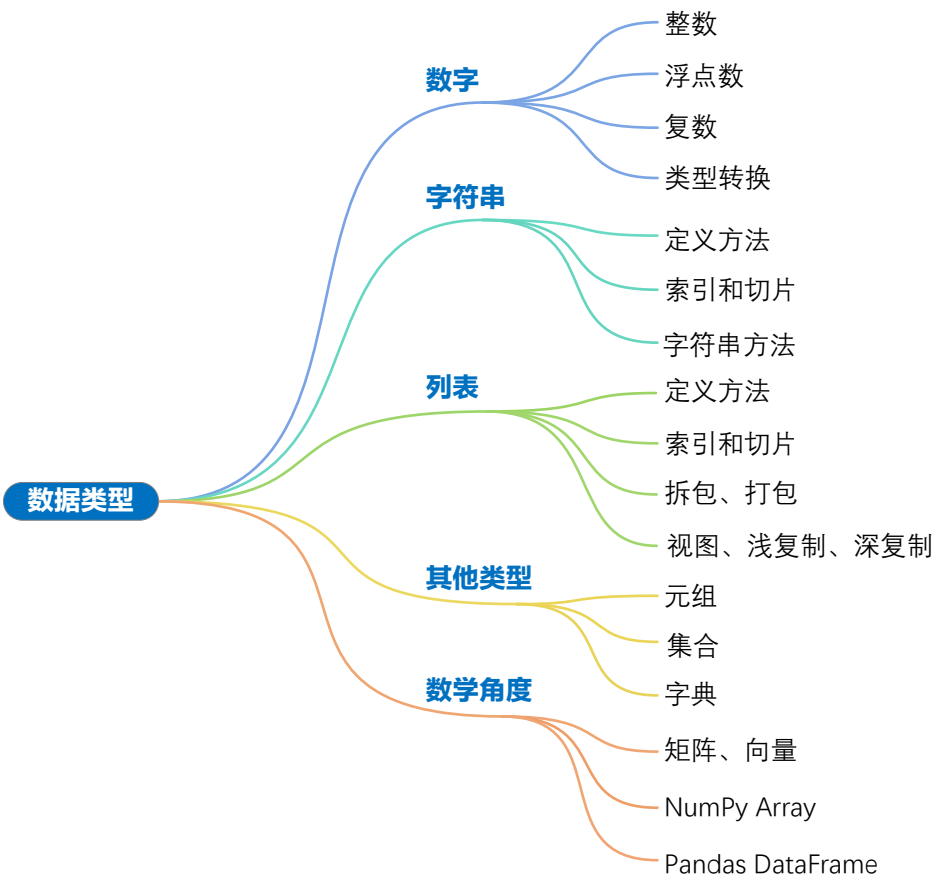
字符串、列表、元组、字典...蜻蜓点水，了解就好

每个人都是天才。但是，如果您以爬树的能力来判断一条鱼，那么那条鱼终其一生都会相信自己是愚蠢的。

Everybody is a genius. But if you judge a fish by its ability to climb a tree, it will live its whole life believing that it is stupid.

—— 阿尔伯特·爱因斯坦 (Albert Einstein) | 理论物理学家 | 1879 ~ 1955

- `copy.deepcopy()` 创建指定对象的深拷贝
- `dict()` Python 内置函数，创建一个字典数据结构
- `emunerate()` Python 内置函数，返回索引和元素，可用于在循环中同时遍历序列的索引和对应的元素
- `float()` Python 内置函数，将指定的参数转换为浮点数类型，如果无法转换则会引发异常
- `int()` Python 内置函数，用于将指定的参数转换为整数类型，如果无法转换则会引发异常
- `len()` Python 内置函数，返回指定序列，字符串、列表、元组等等，的长度，即其中元素的个数
- `list()` Python 内置函数，将元组、字符串等等转换为列表
- `math.ceil()` 将给定数值向上取整，返回不小于该数值的最小整数
- `math.e` math 模块提供的常量，表示数学中的自然常数 e 的近似值
- `math.exp()` 计算以自然常数 e 为底的指数幂
- `math.floor()` 将给定数值向下取整，返回不大于该数值的最大整数
- `math.log()` 计算给定数值的自然对数
- `math.log10()` 计算给定数值的以 10 为底的对数
- `math.pi` math 模块提供的常量，表示数学中的圆周率的近似值
- `math.pow()` 计算一个数的乘幂
- `math.round()` 将给定数值进行四舍五入取整
- `math.sqrt()` 计算给定数值的平方根
- `print()` Python 内置函数，将指定的内容输出到控制台或终端窗口，方便用户查看程序的运行结果或调试信息
- `set()` Python 内置函数，创建一个无序且不重复元素的集合，可用于去除重复元素或进行集合运算
- `str()` Python 内置函数，用于将指定的参数转换为字符串类型
- `type()` Python 内置函数，返回指定对象的数据类型



5.1 数据类型有哪些？

通过上一章学习，我们知道 Python 是一种动态类型语言，它支持多种数据类型。以下是 Python 中常见的数据类型：

- **数字**（number）类型：整数、浮点数、复数等。
- **字符串**（string）类型：表示文本的一系列字符。
- **列表**（list）类型：表示一组有序的元素，可以修改。
- **元组**（tuple）类型：表示一组有序的元素，不能修改。
- **集合**（set）类型：表示一组无序的元素，不允许重复。
- **字典**（dictionary）类型：表示键-值对，其中键必须是唯一的。
- **布尔**（Boolean）类型：表示 True 和 False 两个值。
- None 类型：表示空值或缺失值。

⚠ 再次强调大小写问题，True、False、None 都是首字母大写。此外，注意 Python 代码都是半角字符，只有注释、Markdown 才能出现全角字符。

Python 还支持一些高级数据类型，如**生成器**（Generator）、**迭代器**（Iterator）、**函数**（Function）、**类**（Class）等。

本章最后还要从数学角度介绍矩阵、向量，这两种数据类型。然后再介绍本书最常用的两种数据类型——NumPy Array 和 Pandas DataFrame。

对于 Python 初学者，完全没有必要死记硬背每一种数据类型的操作方法。对于数据类型等 Python 语法细节，希望大家蜻蜓点水，轻装上阵，边用边学。

5.2 数字：整数、浮点数、复数

Python 有三种内置数字类型：

- **整数**（int）：表示整数值，没有小数部分。例如，88、-88、0 等。
- **浮点数**（float）：表示实数值，可以有小数部分。例如，3.14、-0.5、2.0 等。
- **复数**（complex）：表示由实数和虚数构成的数字。

什么是复数？

复数是数学中的一个概念，由实部和虚部组成。它可以表示为 $a + bi$ 的形式，其中 a 是实部，b 是虚部，而 i 是虚数单位，满足 $i^2 = -1$。复数在数学和物理等领域中有广泛的应用。

复数扩展了实数域，使得可以处理平面上的向量运算、波动和振荡等问题。它在电路分析、信号处理、量子力学、调频通信等领域具有重要作用。复数还能用于描述周期性事件、解析函数和几何形状等。

通过复数的运算，我们可以进行加法、减法、乘法和除法等操作，同时也可以求解方程、解析函数和变换等数学问题。复数的使用使得我们能够更好地描述和理解许多实际问题，扩展了数学的应用范围。

代码 1 中 a 将整数值 88 赋值给变量 x。用 type(x)，我们可以得到 x 的数据类型为 int。

b 将浮点数值-8.88 赋值给变量 y。用 type(y)，我们知道 y 的数据类型为 float。

c 构造表示一个实部为 8，虚部为 8 的复数。

注意，8 + 8j 可以写成 8 + 8J，不可以写成 8 + 8*j 或 8 + 8*J。

此外，我们可以用 Python 内置函数 complex() 创建复数。比如，complex(8,8) 也创建了一个实部为 8、虚部为 8 的复数。

complex(real=8, imag=8) 是另一种创建复数的方式，其中 real 参数代表实部，imag 参数代表虚部。complex(real=8, imag=8)与 complex(8, 8) 效果相同。

另外，complex(real=8, imag=8)可以写成 complex(imag=8,real=8)。

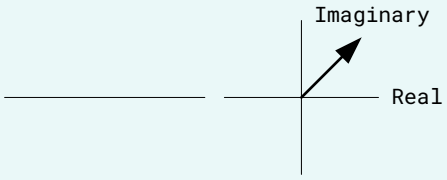
大家还需要注意，在 Python 中，8.8e3 表示 8.8 × 10^3，即 8800.0；8.8e-3 表示 8.8 × 10^{-3}，即 0.0088。

通过 type(8.8e3)，大家可以发现这是一个浮点数 float。

8.8e3 还可以写成 8.8E3，也是没有*号。

请大家在 JupyterLab 中自行练习代码 1。

```
a  x = 88      # 整数
b  y = -8.88   # 浮点数
c  z = 8 + 8j # 复数
print(type(x))
# <class 'int'>
print(type(y))
# <class 'float'>
print(type(z))
# <class 'complex'>
```



代码 1. Python 中三类数值 | Bk1_Ch05_01.ipynb

在 Python 中，数字类型可以进行基本的算术操作，例如加法 (+)、减法 (-)、乘法 (*)、除法 (/)、取余数 (%)、乘幂 (**) 等。数字类型还支持比较运算符，如等于 (==)、不等于 (!=)、大于 (>)、小于 (<)、大于等于 (>=)、小于等于 (<=)。此外，本书后文还会介绍自加运算 (+=)、自减运算 (-=)、自乘运算 (*=)、自除运算 (/=) 等。

本书第 6 章将专门介绍 Python 常见运算符。

类型转换

在 Python 中，可以使用内置函数将一个数字类型转换为另一个类型。下面是常用的数字类型转换函数：

- `int(x)`：将 x 转换为整数类型。如果 x 是浮点数，则会向下取整；如果 x 是字符串，则字符串必须表示一个整数。
- `float(x)`：将 x 转换为浮点数类型。如果 x 是整数，则会转换为相应的浮点数；如果 x 是字符串，则字符串必须表示一个浮点数。
- `complex(x)`：将 x 转换为复数类型。如果 x 是数字，则表示实部，虚部为 0；如果 x 是字符串，则字符串必须表示一个复数；如果 x 是两个参数，则分别表示实部和虚部。
- `str(x)`：将 x 转换为字符串类型。如果 x 是数字，则表示为字符串；如果 x 是布尔类型，则返回 `'True'`或`'False'`字符串。

下面聊一聊代码 2。

a 用 `int()` 将浮点数 88.8 转化为整数 88。我们也可以用 `int()` 把字符串整数`'88'`转换为整数 88。

但是，目前 `int()`不能把浮点数字符串`'88.8'`转化为整数。`int()` 可以把布尔值 (`True` 和 `False`) 转化为整数，比如 `int(True)` 结果为 1，`int(False)` 结果为 0。

`int()` 还可以把二进制字符串转化为十进制整数，比如 `int("1011000", 2)` 的结果为 88。

b 用 `float()` 将整数 8 转换为浮点数 8.0。`float()`还可以将浮点数字符串转化成浮点数，比如 `float('8.8')`的结果为浮点数 8.8。

c 用 `complex()` 将整数转化为复数。

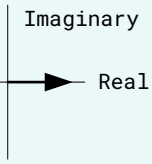
d 用 `str()` 将浮点数转化为字符串。

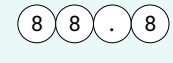
请大家在 JupyterLab 中自行练习代码 2。

需要注意的是，如果在类型转换过程中出现了不合理的转换，例如将一个非数字字符串转换为数字类型，`int('xyz')`，就会导致 `ValueError` 异常。

本书第 7 章将专门介绍如何处理异常。

```python
x = 88.8
y = 8
# 将浮点数转换为整数
x_to_int = int(x)
print(x_to_int)  # 88

# 将整数转换为浮点数
y_to_float = float(y)
print(y_to_float)  # 8.0

# 将整数转换为复数
y_to_complex = complex(y)
print(y_to_complex)  # (8+0j)

# 将数字转换为字符串
x_to_str = str(x)
print(x_to_str)  # '88.8'
```





代码 2. Python 中数值转换 | Bk1_Ch05_02.ipynb

什么是异常？

在 Python 中，异常（exception）是指在程序执行期间出现的错误或异常情况。当出现异常时，程序的正常流程被中断，转而执行异常处理的代码块，以避免程序崩溃或产生不可预知的结果。

Python 中有许多不同类型的异常，每种异常都代表了特定类型的错误。以下是一些常见的异常类型：ValueError（数值错误）：当函数接收到一个不合法的参数值时引发。TypeError（类型错误）：当使用不兼容的类型进行操作或函数调用时引发。IndexError（索引错误）：当尝试访问列表、元组或字符串中不存在的索引时引发。FileNotFoundError（文件未找到错误）：当尝试打开不存在的文件时引发。ZeroDivisionError（零除错误）：当尝试将一个数除以零时引发。

可以使用 try-except 语句来捕获并处理这些异常，以便在程序出现问题时执行适当的操作或提供错误信息。

特殊数值

有很很多场合还需要用到特殊数值，比如圆周率 pi（3.1415926535…）、自然对数底数 e（2.7182818284…）等等。在 Python 中，可以使用 Math 模块来引入这些特殊值，请大家在 JupyterLab 中练习代码 3。

```python
import math

print(math.pi)       # 输出π的值
print(math.e)        # 输出e的值
print(math.sqrt(2)) # 输出根号2的值
```

代码 3. Math 模块中的特殊数值 | Bk1_Ch05_03.ipynb

除了这些特殊数值外，Math 模块还提供了许多其他数学函数，比如四舍五入 round()、上入取整数 ceil()、下舍取整数 floor()、乘幂运算 pow()、指数函数 exp()、以 e 为底数的对数 log()、以 10 为底数的对数 log10()等等。

本书下一章将简单介绍 Python 中的 Math、Statistics、Random 模块。

注意，大家日后会发现我们一般很少用到 Math 模块，为了方便向量化运算我们会直接采用 NumPy、Pandas 中的运算函数。

5.3 字符串：用引号定义的文本

Python 中**字符串**（string）是一个常见的数据类型，常常用于表示文本信息。本节介绍一些常用的字符串用法。

字符串定义

使用单引号'、双引号"、三引号'''或"""将字符串内容括起来即可定义字符串。三引号'''或"""一般用来创建多行字符串。

代码 4 中 a 用一对单引号定义字符串。请大家用 len(str1) 计算字符串的长度。空格、标点符号、数字都是字符串的一部分。

b 用一对双引号定义字符串。

c 用加号 + 将两个字符串相连，如图 1 所示。

d 定义的字符串最后的一个元素是个空格。

e 利用*3 将字符串复制 3 次。

f 定义的是整数字符串，并不能直接进行算术运算。

g 用*3 也是将字符串复制 3 次，并非计算 3 倍，具体如图 2 所示。请大家利用 int()将 f 定义的整数字符串转化为整数，然后再*3 并查看结果。

h 定义了另外一个整数字符串。

i 利用加号 + 将两个整数字符串相连，具体如图 3 所示。请大家调换 + 左右字符串的顺序，再查看结果。

请大家在 JupyerLab 中练习代码 4。请大家用 len() 函数获得代码 4 每个字符串的长度，即字符串中字符个数。

再次请大家注意，空格、标点符号都是字符串的一部分。使用加号 + 将多个字符串连接起来，使用乘号 * 复制字符串。数字字符串仅仅是文本，不能直接完成算数运算，需要转化成整数、浮点数之后才能进行算数运算。

注意，Python 中长度为 0 的字符串也是字符串类型，比如 str_test = ''; type(str_test) 的结果还是 str。

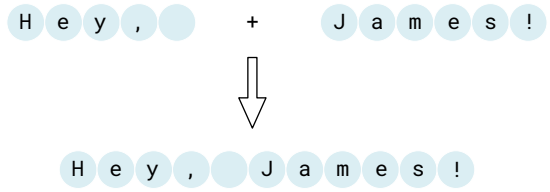


图 1．字符串相加

图 2．数字字符串乘法

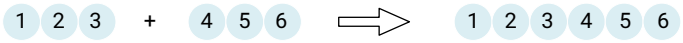


图 3．数字字符串加法

```
a  str1 = 'I am learning Python 101!'
   print(str1)
   # 打印

b  str2 = "Python is fun. Machine learning is fun too."
   print(str2)
   # 打印

   # 使用加号 + 将多个字符串连接起来
c  str4 = 'Hey, ' + 'James!'
   print(str4)
   # 'Hey, James!'

   # 使用乘号*将一个字符串复制多次
d  str5 = 'Python is FUN! '  # 字符串最后有一个空格
e  str6 = str5 * 3
   print(str6)
   # 'Python is FUN! Python is FUN! Python is FUN!'

   # 字符串中的数字仅仅是字符
f  str7 = '123'
g  str8 = str7 * 3
   print(str8)

h  str9 = '456'
i  str10 = str7 + str9
   print(str10)
   print(type(str10))
```

代码 4．字符串定义和操作 | Bk1_Ch05_04.ipynb

索引

在 Python 中，可以通过**索引**（indexing）和**切片**（slicing）来访问和操作字符串中的单个字符、部分字符。

代码 5 中ⓐ用单引号定义了字符串。ⓑ用 len() 计算字符串长度，并用 print() 打印出来。

代码 5 中ⓒ使用了 for 循环来遍历字符串中的每个字符，并打印出字符及其对应的索引。enumerate() 函数来同时获取字符和它们的索引位置。

enumerate() 函数会返回一个迭代器，包含每个字符及其对应的索引。然后，通过 for 循环遍历迭代器，依次打印出每个字符和它们的索引。

本书第 7 章将专门介绍 for 循环。

在ⓒ中，**f-字符串** (formatted string, f-strings) 是一种用于格式化字符串的语法。它以字母 "f" 开头，并使用花括号{}来插入变量或表达式的值。

在ⓒ这个特定的例子中，f-字符串用于构建一个带有变量值的字符串。通过在字符串中使用花括号和变量名，可以在字符串中插入变量的值。在这种情况下，使用了两个变量 {char} 和 {index}。

当代码执行时，{char} 会被替换为当前循环迭代的字符，{index} 会被替换为对应字符的索引值。这样就创建了一个字符串，包含了字符及其对应的索引信息。

本章后文将介绍包括在 f-字符串在内的其他格式化字符串方法。

如图 4 所示，字符串中的每个字符都有一个对应的索引位置，索引从 0 开始递增。可以使用方括号 [] 来访问指定索引位置的字符。

代码 5 中ⓓ提取字符串中索引为 0 的元素，即第 1 个元素。

ⓔ提取索引为 1 的元素。

可以使用负数索引来从字符串的末尾开始计算位置。例如，ⓕ用索引-1 提取字符串倒数第一个字符，ⓖ用索引-2 提取倒数第二个字符，依此类推。

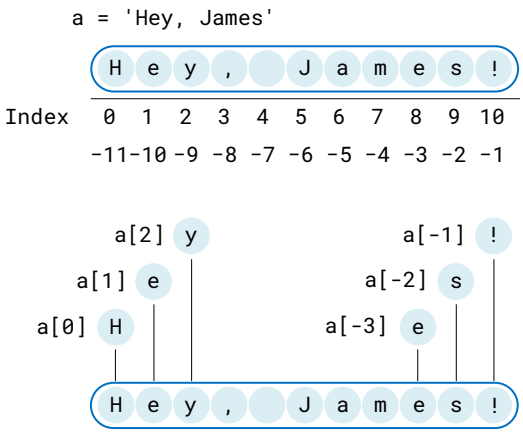


图 4. 字符串的索引

切片

如图 5 所示，切片是指从字符串中提取出一部分子字符串。可以使用半角冒号 : 来指定切片的起始位置 start 和结束 end 位置。语法为 string[start:end]，包括 start 索引对应的字符，但是不包括 end 位置的字符，相当于数学中的“左闭右开”区间。

请大家参考代码 5 中的 h 和 i 。

切片还可以指定步长 (step)，用于跳过指定数量的字符。语法为 string[start:end:step]。请大家参考代码 5 中的 j 和 k 。

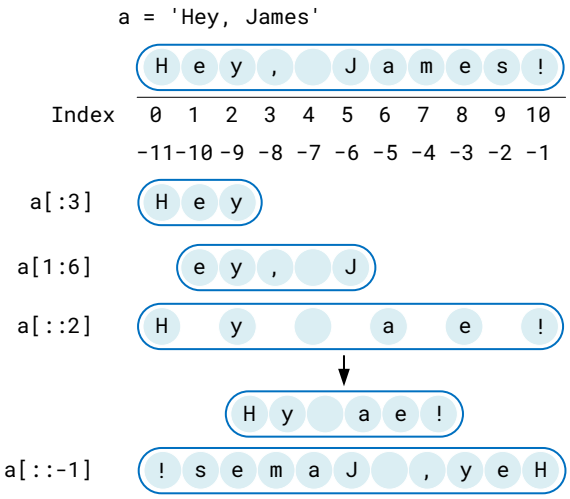


图 5. 字符串的切片

⚠ 注意，复制字符串可以采用 string_name[:] 实现。还需要注意的是，索引和切片操作不会改变原始字符串，而是返回一个新的字符串。

Python 中还有很多字符串“花式”切片方法，大家没有必要花大力气去“精雕细琢”。大概知道字符串有哪些常见的索引、切片方法就足够了，等到用到时再去特别学习。还是那句话，别死磕 Python 语法！

请大家自行在 JupyterLab 中练习代码 5。

```
a greeting_str = 'Hey, James!'
  # 打印字符串长度
  print('字符串的长度为：')
b print(len(greeting_str))

  # 打印每个字符和对应的索引
c for index, char in enumerate(greeting_str):
      print(f"字符：{char}，索引：{index}")

  # 单个字符索引
d print(greeting_str[0])
e print(greeting_str[1])

f print(greeting_str[-1])
g print(greeting_str[-2])

  # 切片
  # 取出前3个字符，索引为0、1、2
h print(greeting_str[:3])

  # 取出索引1、2、3、4、5，不含0，不含6
i print(greeting_str[1:6])

  # 指定步长2，取出第0、2、4 ...
j print(greeting_str[::2])

  # 指定步长-1，倒序
k print(greeting_str[::-1])
```

a: H e y , ␣ J a m e s !
d: H
e: e
f: !
g: s
h: H e y
i: e y , ␣ J
j: H y ␣ a e !
k: ! s e m a J ␣ , y e H

代码 5. 字符串索引和切片 | Bk1_Ch05_05.ipynb

从 0 计数 vs 从 1 计数

从 0 计数和从 1 计数是在数学和编程中常见的计数方式。

从 0 计数（zero-based counting）将第一个元素的索引或位置标记为 0，即从 0 开始计数。例如，对于一个包含 n 个元素的序列，它们的索引分别为 0、1、2、...、n - 1。在计算机科学和编程中，Python 使用从 0 计数的方式。

⚠注意，在绘图中，大家会经常碰到一幅图中有若干子图，这时 Python 对子图的编号一般从 1 开始。

从 1 计数（one-based counting）将第一个元素的索引或位置标记为 1，即从 1 开始计数。例如，对于一个包含 n 个元素的序列，它们的索引分别为 1、2、3、...、n。MATLAB 使用从 1 计数方式；统计学（样本）、线性代数（矩阵、向量）等通常使用从 1 计数的方式。

相比来看，从 1 计数更符合人类直观理解的习惯。从 1 计数在数学、统计学、数值计算等领域中较为常见。编程角度来看，从 0 计数在计算机科学中更常见，因为它与计算机内存和数据结构的底层表示方式相匹配。它使得处理数组、列表和字符串等数据结构更加高效和一致。

在实际编程中，理解和适应使用不同的计数方式是重要的。需要根据具体情况选择适当的计数方式，以确保正确地处理索引、循环和算法等操作。同时，注意在不同的领域和语境中遵循相应的计数习惯和规则。

字符串方法

Python 提供了许多用于字符串处理的常见方法。下面是一些常见的字符串方法及其示例。

len() 返回字符串的长度，比如下例。

```
string = "Hello, James!"
length = len(string)
print(length)
```

lower() 和 upper() 将字符串转换为小写或大写，比如下例。

```
string = "Hello, James!"
lower_string = string.lower()
upper_string = string.upper()
print(lower_string)  # 输出 "hello, james!"
print(upper_string)  # 输出 "HELLO, JAMES!"
```

以下是一些常见 Python 字符串方法及其作用。

capitalize():将字符串的第一个字符转换为大写，其他字符转换为小写。

count() 统计字符串中指定子字符串的出现次数。

find() 在字符串中查找指定子字符串的第一次出现，并返回索引值。

isalnum() 检查字符串是否只包含字母和数字。

isalpha() 检查字符串是否只包含字母。

isdigit() 检查字符串是否只包含数字。

join() 将字符串列表或可迭代对象中的元素连接为一个字符串。

replace() 将字符串中的指定子字符串替换为另一个字符串。

split() 将字符串按照指定分隔符分割成子字符串，并返回一个列表。

⚠注意，这些方法大家也不需要死记硬背！了解就好，轻装上阵。数据分析、机器学习中更常用的 NumPy 数组、Pandas 数据帧，这都是本书后续要重点介绍的内容。

将数据插入字符串

很多场合需要将数据插入特定字符串。

比如以下几种情形，使用 print() 时，图片中插入**图例**（legend），图片中插入**标题**（title）、打印日期时间、打印统计量（均值、方差、标准差、四分位）等等，都可能需要将特定数据插入到字符串中。

代码 6 给出四种常用方法。

c 使用 + 运算符可以将字符串与其他数据类型连接在一起。这是最简单的方法之一，但不够灵活。其中，str(height) 将浮点数转化为字符串。

d 使用 % 将占位符插入字符串中，并使用 % 运算符的右侧的数据来替换这些**占位符**(placeholder)。

其中，%s 是一个字符串占位符，表示要插入一个字符串值。%.3f 是一个浮点数占位符，表示要插入一个浮点数值，并指定了小数点后保留三位小数。表 1 总结常用%占位符类型。

这是一种相对来说比较旧式的字符串格式化方法，不太推荐在新代码中使用。

e 使用 str.format() 方法在字符串中指定占位符，并使用 .format() 方法的参数来替换这些占位符。其中，{:.3f} 是一个浮点数占位符，表示要插入一个浮点数值，并指定了小数点后保留三位小数。

f 使用 f-strings。前文提过，这是从 Python 3.6 版本开始引入的一种方式。f-strings 允许在字符串前面加上 f 或 F，并在字符串中使用大括号 {} 插入变量。

```
a name = 'James'
b height = 181.18
  # 使用 + 运算符
c str_1 = name + ' has a height of ' + str(height) + ' cm.'
  print(str_1)

  # 使用 %
d str_2 = '%s has a height of %.3f cm.' %(name, height)
  print(str_2)

  # 使用 str.format()
e str_3 = '{} has a height of {:.3f} cm.'.format(name, height)
  print(str_3)

  # 使用f-strings
f str_4 = f'{name} has a height of {height:.3f} cm.'
  print(str_4)
```

代码 6. 将数据插入字符串 | Bk1_Ch05_06.ipynb

表 1. 常用%占位符类型 | Bk1_Ch05_07.ipynb

%占位符	解释	例子
%c	单个字符	'The first letter of Python is %c' % 'P'
%s	字符串	'Welcome to the world of %s!' % 'Python'
%i	整数	'Python has %i letters.' % len('Python')
%f	浮点数	number = 1.8888

		print("Rounding %.4f to 2 decimal places is %.2f" % (number, number))
%o	八进制整数	number = 12 print("%i in octal is %o" % (number, number))
%e	科学计数	number = 12000 print("%i is %.2e" % (number, number))

表 2. 常用.format() 示例 | Bk1_Ch05_08.ipynb

样式	解释	例子
:.2f	浮点数后两位	"{:.2f}".format(3.1415926) # '3.14'
:%	百分数	"{:%}".format(3.1415926) #'314.159260%'
:.2%	百分数，小数点后两位	"{:.2%}".format(3.1415926) #'314.16%'
:.2e	科学计数	"{:.2e}".format(3.1415926) #'3.14e+00'
:,	千位加逗点	"{:,}".format(3.1415926*1000) #'3,141.5926'

表 3. 常用 f-strings 示例 | Bk1_Ch05_09.ipynb

解释	示例
日期和时间	import datetime now = datetime.datetime.now() print(f'{now:%Y-%m-%d %H:%M}')不要写成 f'{now}:%Y-%m-%d %H:%M:%S' print(f'{now:%d/%m/%y %H:%M:%S}')
小数点后两位	pi = 3.14159265358979323846264 3 f'{pi:.2f}'
科学计数	pi = 3.14159265358979323846264 3 f'{pi * 1000:.2e}'
2 进制	a = 18 print(f"{a:b}")
16 进制	a = 68 print(f"{a:x}")
8 进制	a = 88 print(f"{a:o}")

计算机领域常用的是**二进制** (binary numerical system)。**八进制** (octal numeral system) 和**十六进制** (hexadecimal numeral system 或 hexadecimal 或 hex) 也常用。十六进制在十进制的基础上增加了 A、B、C、D、E 和 F。

举个例子，在 RGB（Red, Green, and Blue）色彩模型颜色定义中，我们会用到十六进制。比如，纯红色为 '#FF0000'，纯绿色为 '#00FF00'，纯蓝色为 '#0000FF'。

表 4. 比较十进制、二进制、八进制、十六进制

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	1B
28	11100	34	1C
29	11101	35	1D
30	11110	36	1E
31	11111	37	1F
32	100000	40	20

5.4 列表：存储多个元素的序列

在 Python 中，**列表**（list）是一种非常常用的数据类型，可以存储多个元素，并且可以进行增删改查等多种操作。

代码 7 中 a 生成的是一个特殊的列表，我们称之为混合列表，原因是这个列表中每个元素都不同。如图 6 所示，这个列表中索引为 4 的元素（从左到右第 5 个元素）还是个列表，相当于嵌套。

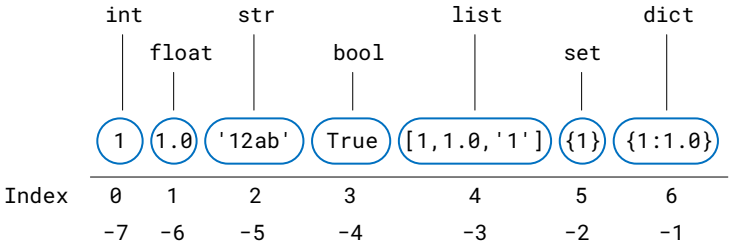


图 6. 混合列表

b 也是用 for 循环和 enumerate() 遍历混合列表元素，并返回索引。

c 用 type() 提取列表每个元素的数据类型。

d 用 f-string 打印列表元素、索引、数据类型。

类似前文字符串索引，我们可以用同样的方法索引列表中元素。

e 和 f 提取列表索引为 0 和 1 的元素。

g 和 h 分别提取列表倒数第 1、2 元素。

列表切片的方法和前文字符串切片方法一致。

i 提取列表前 3 个元素，结果依然是个列表。

j 提取列表索引为 1、2、3 的元素，不包含索引为 0 的元素，即第 1 个元素。

k 通过指定步长（2），提取索引为 0、2、4、6 元素切片。

l 通过指定步长，将列表倒序。

如图 7 所示，如果列表中的某个元素也是列表，我们可以通过二次索引来进一步索引、切片。

比如，m 先从嵌套列表中提取索引为 4 的元素，这个元素还是一个列表。然后进一步再提取子列表中索引为 1 的元素，这个元素是个浮点数 float。

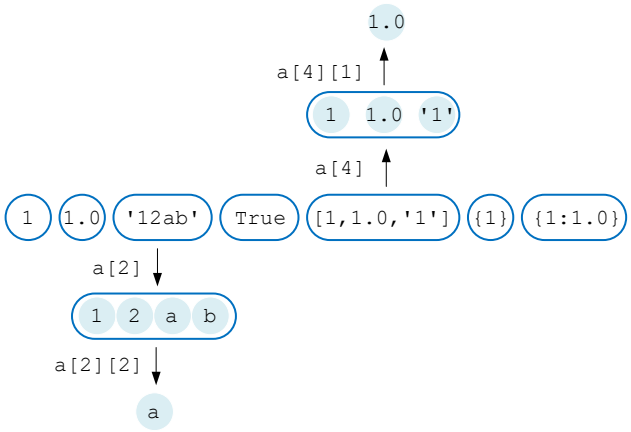
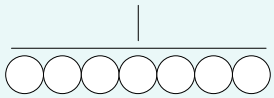
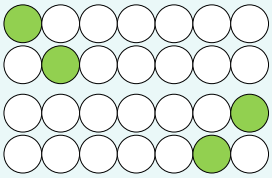
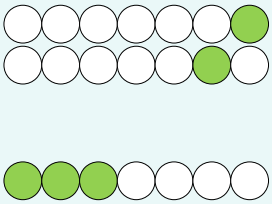


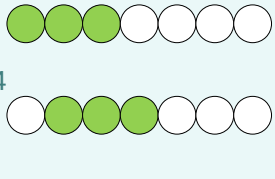
图 7. 混合列表的索引

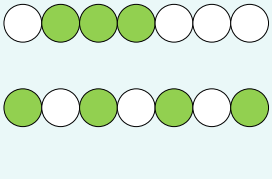
```
# 创建一个混合列表
a my_list = [1, 1.0, '1', True, [1, 1.0, '1'], {1}, {1:1.0}]
print('列表长度为')
print(len(my_list))

# 打印每个元素和对应的序号
b for index, item in enumerate(my_list):
c     type_i = type(item)
d     print(f"元素：{item}，索引：{index}，类型：{type_i}")

# 列表索引
e print(my_list[0])
f print(my_list[1])

g print(my_list[-1])
h print(my_list[-2])

# 列表切片
# 取出前3个元素，索引为0、1、2
i print(my_list[:3])

# 取出索引1、2、3，不含0，不含4
j print(my_list[1:4])

# 指定步长2，取出第0、2、4、6
k print(my_list[::2])

# 指定步长-1，倒序
l print(my_list[::-1])

# 提取列表中的列表某个元素
m print(my_list[4][1])
```







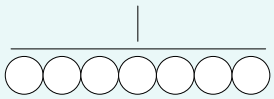




代码 7. 列表索引和切片 | Bk1_Ch05_10.ipynb

代码 8 给出的 list 常见方法、操作，请大家在 JupyterLab 中练习，本章不展开讲解。

```
# 创建一个混合列表
a my_list = [1, 1.0, '12ab', True, [1, 1.0, '1'], {1}, {1:1.0}]
print(my_list)

# 修改某个元素
b my_list[2] = '123'
print(my_list)

# 在列表指定位置插入元素
c my_list.insert(2, 'inserted')
print(my_list)

# 在列表尾部插入元素
d my_list.append('tail')
print(my_list)

# 通过索引删除
e del my_list[-1]
print(my_list)

# 删除某个元素
f my_list.remove('123')
print(my_list)

# 判断一个元素是否在列表中

g if '123' in my_list:
    print("Yes")
else:
    print("No")

# 列表翻转
h my_list.reverse()
print(my_list)

# 将列表用所有字符连接，连接符为下划线 _
i letters = ['J', 'a', 'm', 'e', 's']
j word = '_'.join(letters)
print(word)
```



代码 8. 列表常用方法、操作 | Bk1_Ch05_11.ipynb

拆包、打包

星号 * 可以用来将一个列表**拆包**（unpacking）成单独元素，也可以将若干元素打包放入另一个列表中。这种操作对于列表拆解、合并非常有用。代码 9 举几个例子介绍这种方法。

a 定义了一个包含整数的列表 list_0。

b 将 list_0 中索引为 0 元素赋给变量 first，而其余的元素将被收集到一个列表 list_rest 中。这是使用 * 操作符来实现的，它用于收集多余的元素。

类似地，c 将 list_0 中索引为 0、1 元素分别赋给变量 first 和 second，而其余的元素将被收集到一个列表 list_rest 中。

d 在 c 基础上，又将 list_0 最后一个元素赋给变量 last，其余元素也是被收集在 list_rest。

e 利用下划线_表示一个占位符，，通常用于表示一个不需要使用的元素。

f 使用 * 操作符将两个列表 list1 和 list2 中的所有元素先拆包，然后合并到一个新的列表 combined_list 中。请大家对比 [list1, list2] 的结果。

```
# 定义列表
a list_0 = [0, 1, 2, 3, 4, 5, 6, 7, 8]

b first, *list_rest = list_0
  print(list_rest) # [1, 2, 3, 4, 5, 6, 7, 8]

c first, second, *list_rest = list_0
  print(list_rest) # [2, 3, 4, 5, 6, 7, 8]

d first, second, *list_rest, last = list_0
  print(list_rest) # [2, 3, 4, 5, 6, 7]

e first, *list_rest, _ = list_0
  print(list_rest) # [1, 2, 3, 4, 5, 6, 7]

  list1 = [1, 2, 3, 4, 5]
  list2 = [6, 7, 8]
  # 合并
f combined_list = [*list1, *list2]
  print(combined_list) # [1, 2, 3, 4, 5, 6, 7, 8]
```

代码 9. 列表拆包、打包 | Bk1_Ch05_12.ipynb

用星号 * 拆包、打包也适用于元组和字符串，请大家自行学习代码 10。注意，元组和字符串打包之后的结果为列表。

```
# 定义字符串
a string_0 = 'abcd'
b first, *str_rest, last = string_0
  print(str_rest) # ['b', 'c']

  # 定义元组
c tuple_0 = (1,2,3,4)
d first, *tuple_rest, last = tuple_0
  print(tuple_rest) # [2, 3]
```

代码 10. 字符串和元组拆包、打包 | Bk1_Ch05_13.ipynb

视图 vs 浅复制 vs 深复制

如果用 = 直接赋值，是非拷贝方法，结果是产生一个**视图**（view）。这两个列表是等价的，修改其中任何（原始列表、视图）一个列表都会影响到另一个列表。

如图 8 所示，用等号 = 赋值得到的 list_2 和 list_1 共享同一地址，这就是我们为什么称 list_2 为视图。视图这个概念是借用自 NumPy。

我们在本书后续还要聊到 NumPy Array 的视图和副本这两个概念。

而通过 copy() 获得的 list_3 和 list_1 地址不同。请大家自行在 JupyterLab 中练习代码 11。

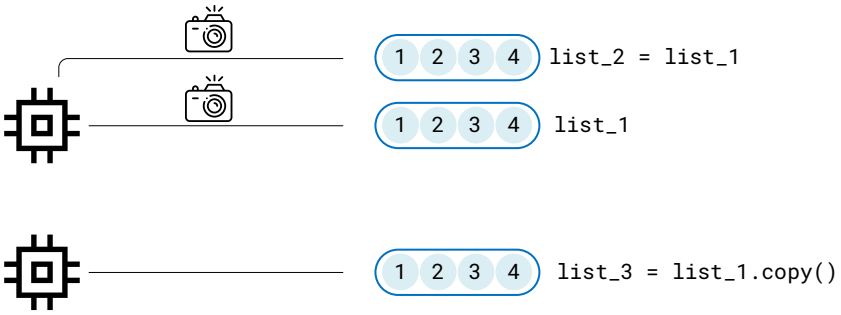


图 8. 视图，还是副本？

```
a  list1 = [1, 2, 3, 4]

   # 赋值，视图
b  list2 = list1

   # 拷贝，副本（浅拷贝）
c  list3 = list1.copy()

d  list2[0] = 'a'
   list2[1] = 'b'
e  list3[2] = 'c'
   list3[3] = 'd'

   print(list1)
   print(list2)
   print(list3)
```

代码 11. 列表视图 vs 副本 | Bk1_Ch05_14.ipynb

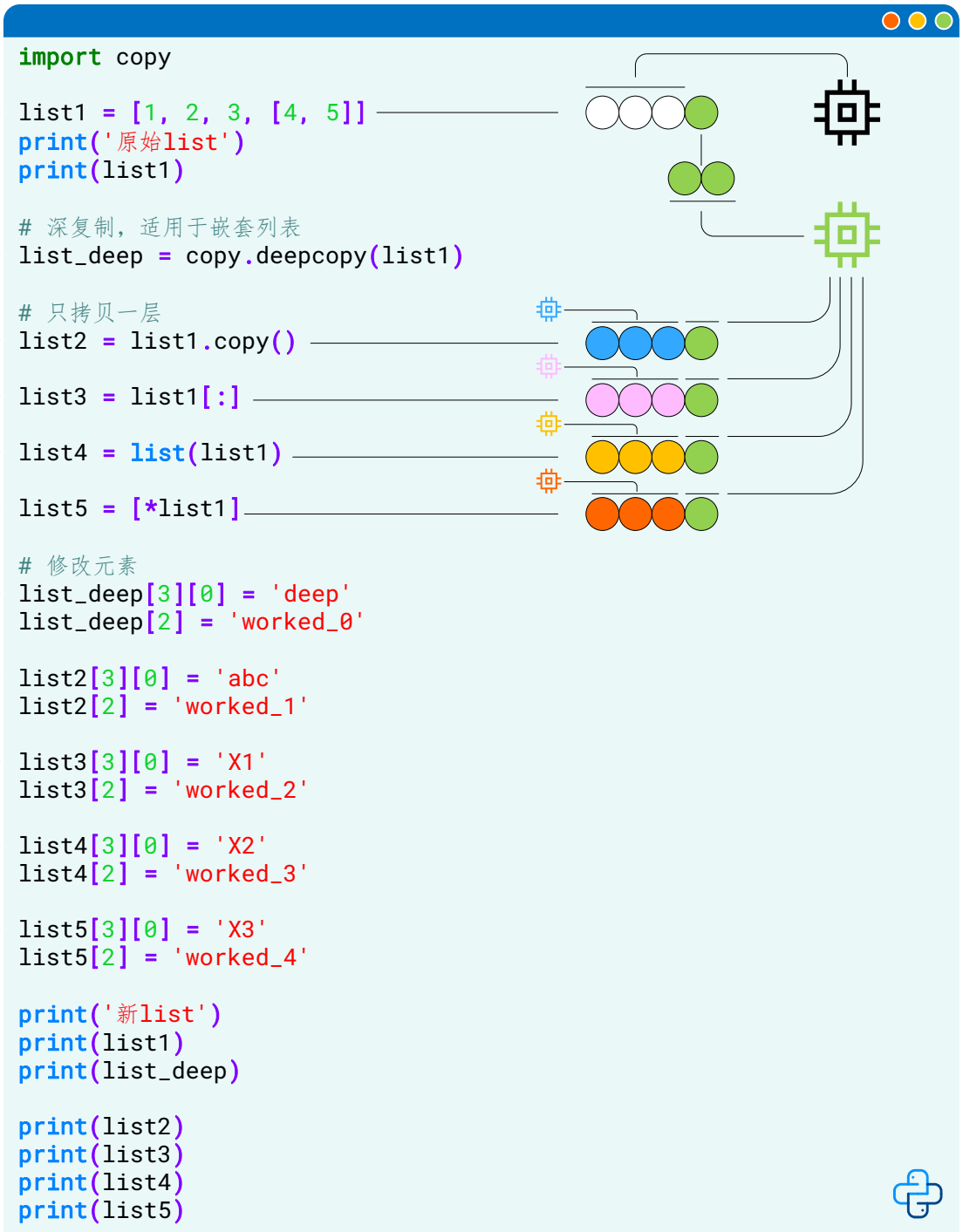
可惜事情并没有这么简单。在 Python 中，列表是可变对象，因此在复制列表时会涉及到深复制和浅复制的概念。

浅复制（shallow copy）只对 list 的第一层元素完成拷贝，深层元素还是和原 list 共用。

深复制（deep copy）是创建一个完全独立的列表对象，该对象中的元素与原始列表中的元素是不同的对象。

⚠注意，特别是对于嵌套列表，建议大家采用 copy.deepcopy() 深复制。

图 9 代码比较不同复制，请大家自行学习。



```
import copy

a list1 = [1, 2, 3, [4, 5]]
print('原始list')
print(list1)

# 深复制，适用于嵌套列表
b list_deep = copy.deepcopy(list1)

# 只拷贝一层
c list2 = list1.copy()

d list3 = list1[:]

e list4 = list(list1)

f list5 = [*list1]

# 修改元素
g list_deep[3][0] = 'deep'
list_deep[2] = 'worked_0'

h list2[3][0] = 'abc'
list2[2] = 'worked_1'

i list3[3][0] = 'X1'
list3[2] = 'worked_2'

j list4[3][0] = 'X2'
list4[2] = 'worked_3'

k list5[3][0] = 'X3'
list5[2] = 'worked_4'

print('新list')
print(list1)
print(list_deep)

print(list2)
print(list3)
print(list4)
print(list5)
```

图 9. 嵌套列表浅复制 vs 深复制 | Bk1_Ch05_15.ipynb

5.5 其他数据类型：元组、集合、字典

元组

在 Python 中，**元组**（tuple）是一种不可变的序列类型，用圆括号 () 来表示。元组一旦创建就不能被修改，这意味着你不能添加或删除其中的元素。

tuple 和 list 都是序列类型，可以存储多个元素，它们都可以通过索引访问和修改元素，支持切片操作。

但是，两者有明显区别，元组使用圆括号 () 表示，而列表使用方括号 [] 表示。元组是不可变的，而列表是可变的。这意味着元组的元素不能被修改、添加或删除，而列表可以进行这些操作。

元组的优势在于它们比列表更轻量级，这意味着在某些情况下，它们可以提供更好的性能和内存占用。本书不展开介绍元组，感兴趣的读者可以参考。

```
https://docs.python.org/3/tutorial/datastructures.html
```

集合

在 Python 中，**集合**（set）是一种无序的、可变的数据类型，可以用来存储多个不同的元素。使用花括号 {} 或者 set() 函数创建集合，或者使用一组元素来初始化一个集合。

```
number_set = {1, 2, 3, 4, 5}
word_set = set(["apple", "banana", "orange"])
```

可以使用 add() 方法向集合中添加单个元素，使用 update() 方法向集合中添加多个元素。

```
fruit_set = set(["apple", "banana"])
fruit_set.add("orange")
fruit_set.update(["grape", "kiwi"])
```

删除元素：使用 remove() 或者 discard() 方法删除集合中的元素，如果元素不存在，remove() 方法会引发 KeyError 异常，而 discard() 方法则不会。

```
fruit_set.remove("banana")
fruit_set.discard("orange")
```

集合的好处是可以用交集、并集、差集等集合操作来操作集合，如图 10 所示。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
set3 = set1 & set2  # 交集
set4 = set1 | set2  # 并集
set5 = set1 - set2  # 差集
```

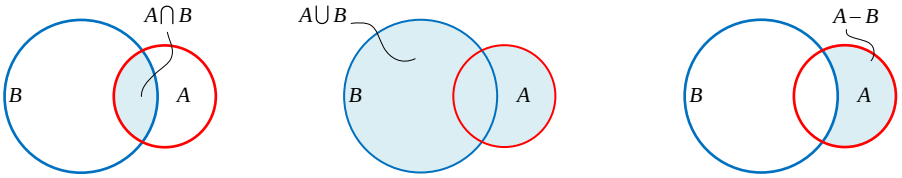


图 10. 交集、并集、差集

字典

在 Python 中，字典是一种无序的**键值对** (key-value pair) 集合。

可以使用大括号 {} 或者 dict() 函数创建字典，**键** (key) **值** (value) 对之间用冒号 : 分隔。有关字典这种数据类型本书不做展开，请大家自行学习代码 12。

⚠注意，使用大括号 {} 创建字典时，字符串键 key 用引号；而使用 dict() 创建字典时，字符串键不使用引号。

再次强调，数据分析、机器学习实践中，我们更关注的数据类型是 NumPy 数组、Pandas 数据帧，这是本书后续要着重讲解的内容。

```
# 使用大括号创建字典
a person = {'name': 'James', 'age': 88, 'gender': 'male'}

# 使用 dict() 函数创建字典
b fruits = dict(apple=88, banana=888, cherry=8888)

# 访问字典中的值
c print(person['name'])
d print(fruits['cherry'])

# 修改字典中的值
e person['age'] = 28
  print(person)

# 添加键值对
f person['city'] = 'Toronto'
  print(person)

# 删除键值对
g del person['gender']
  print(person)

# 获取键、值、键值对列表
h print(person.keys())
i print(person.values())
j print(person.items())
```

key	value
name	James
age	88
gender	male

key	value
apple	88
banana	888
cherry	8888

代码 12. 有关字典的常见操作 | Bk1_Ch05_16.ipynb

5.6 矩阵、向量：线性代数概念

矩阵、向量

抛开本章前文这些数据类型，数学上我们最关心的数据类型是--矩阵、向量。

简单来说，**矩阵**（matrix）是一个由数值排列成的矩形阵列，其中每个数值都称为该矩阵的元素。矩阵通常使用大写、斜体、粗体字母来表示，比如 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{V}$、$\boldsymbol{X}$。

向量（vector）是一个有方向和大小的量，通常表示为一个由数值排列成的一维数组。向量通常使用小写字母加粗体来表示，例如 $\boldsymbol{x}$、$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{v}$、$\boldsymbol{u}$。

如图 11 所示，一个 $n \times D$（n by capital D）矩阵 $\boldsymbol{X}$。

n 是**矩阵行数**（number of rows in the matrix）。

D 是**矩阵列数**（number of columns in the matrix）。

矩阵 $\boldsymbol{X}$ 的行索引就是 1、2、3、...、n。矩阵 $\boldsymbol{X}$ 的列索引就是 1、2、3、...、D。

$x_{1,1}$ 代表矩阵第 1 行、第 1 列元素，$x_{i,j}$ 代表矩阵第 i 行、第 j 列元素。

$$\boldsymbol{X}_{n\times D} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix}$$

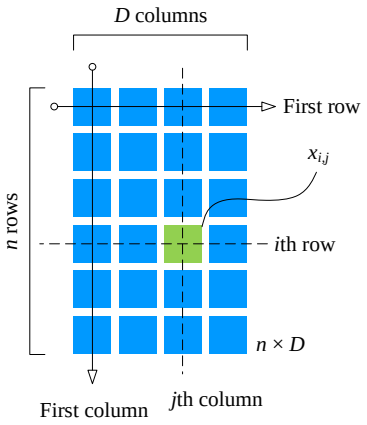


图 11. $n \times D$ 矩阵 $\boldsymbol{X}$

从数据、统计、线性代数、几何角度解释，什么是矩阵？

矩阵是一个由数字或符号排列成的矩形阵列。简单来说，矩阵就是个表格。矩阵在数据、统计、线性代数和几何学中扮演着重要的角色。

从数据的角度来看，矩阵可以表示为一个包含行和列的数据表。每个单元格中的数值可以代表某种测量结果、观察值或特征。数据科学家和分析师使用矩阵来存储和处理数据，从中提取有用的信息。比如，一张黑白照片中的数据就可以看做是个矩阵。

从统计学的角度来看，矩阵可以用于描述多个变量之间的关系。例如，协方差矩阵用于衡量变量之间的相关性，而相关矩阵则提供了变量之间的线性相关性度量。统计学家使用这些矩阵来推断模式、关联和依赖性，以及进行数据分析和建模。

从线性代数的角度来看，矩阵可以用于表示线性方程组的系数矩阵。通过矩阵运算，例如矩阵乘法、求逆和特征值分解，可以解决线性方程组、求解特征向量和特征值等问题。线性代数中的矩阵理论提供了处理线性关系的强大工具。

从几何学的角度来看，矩阵可以用于表示几何变换。通过将向量表示为矩阵的列或行，可以应用平移、旋转、缩放等几何变换。矩阵乘法用于组合多个变换，从而实现更复杂的几何操作。在计算机图形学和计算机视觉中，矩阵在处理和表示二维或三维对象的位置、方向和形状方面起着重要作用。

总而言之，矩阵是一个在数据、统计、线性代数和几何学中广泛应用的数学工具，它能够表示和处理多个变量之间的关系、解决线性方程组、进行几何变换等。

几何视角看：行向量、列向量

行向量（row vector）是由一系列数字或符号排列成的一行序列。**列向量**（column vector）是由一系列数字或符号排列成的一列序列。

如图 12 所示，矩阵 $\boldsymbol{A}$ 可以视作由一系列行向量、列向量构造而成。这相当于硬币的正反两面，即一体两面。这幅图中，我们还看到了如何用几何方式展示向量。比如 $\boldsymbol{A}$ 的行向量可以看成是平面中的三个箭头，而 $\boldsymbol{A}$ 的列向量可以看成是三维空间中的两个箭头。

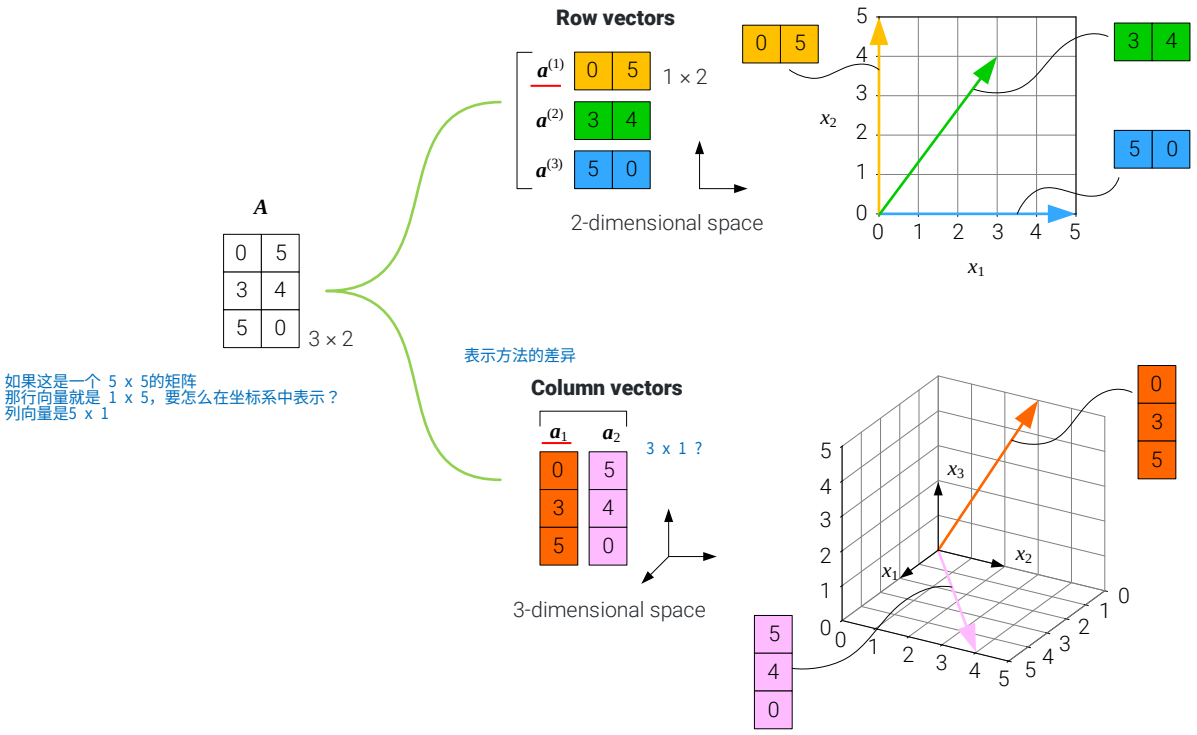


图 12．行向量和列向量

图 13 所示为矩阵转置 $\boldsymbol{A}^{\mathrm{T}}$ 的行列向量。而 $\boldsymbol{A}^{\mathrm{T}}$ 的行向量是三维空间的两个箭头，$\boldsymbol{A}^{\mathrm{T}}$ 的列向量是平面中的三个箭头。

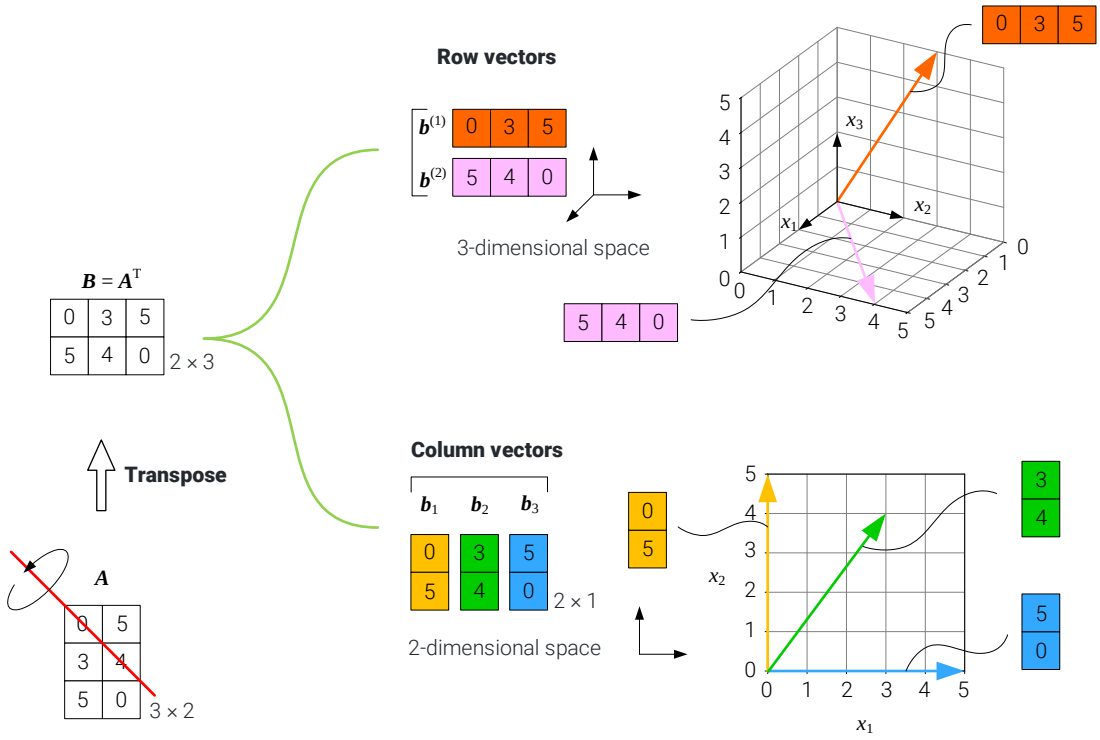
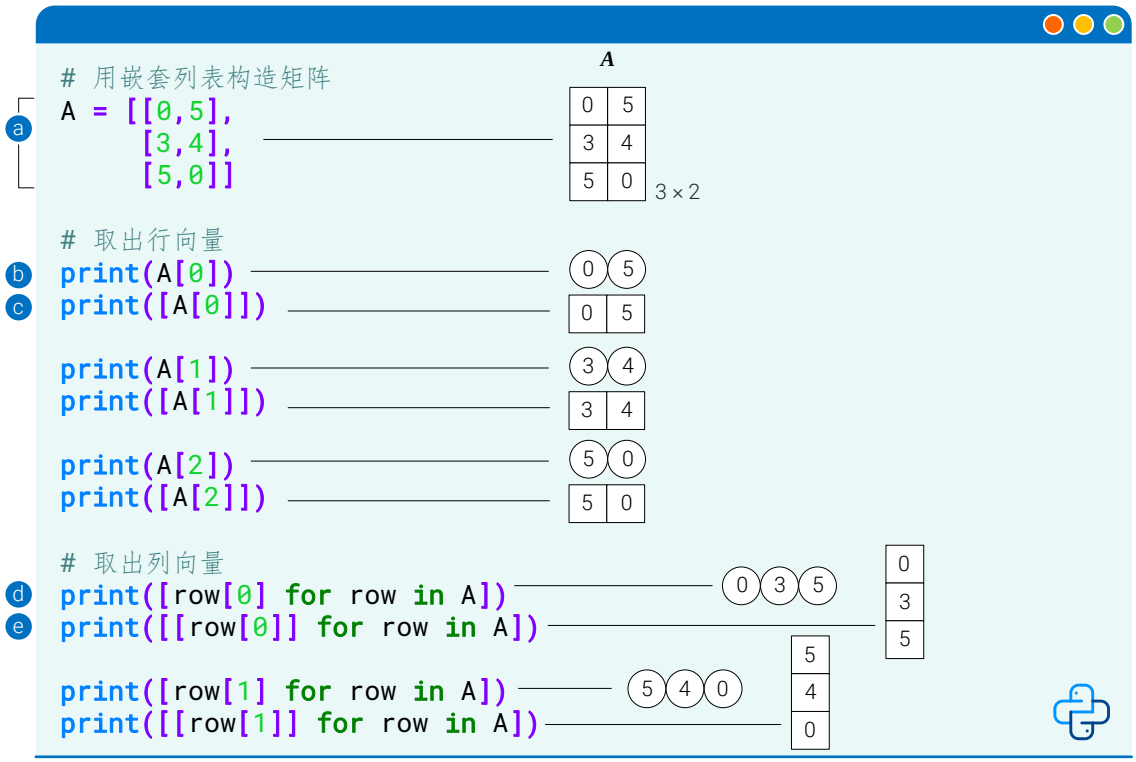


图 13．转置之后矩阵的行向量和列向量

什么是矩阵转置？

矩阵转置是指将矩阵的行和列对调，得到一个新的矩阵。原矩阵的第 i 行会变成新矩阵的第 i 列，原矩阵的第 j 列会变成新矩阵的第 j 行。这个操作不改变矩阵的元素值，只是改变了它们的排列顺序。

我们可以用嵌套列表方式来表达矩阵，如代码 13 所示，请大家自行学习这段代码。



代码 13．用嵌套列表构造矩阵 | Bk1_Ch05_17.ipynb

鸢尾花数据

从统计数据角度，n 是样本个数，D 是样本数据特征数。如图 14 所示，鸢尾花数据集，不考虑标签（即鸢尾花三大类 setosa、versicolor、virginica），数据集本身 $n = 150$，$D = 4$。

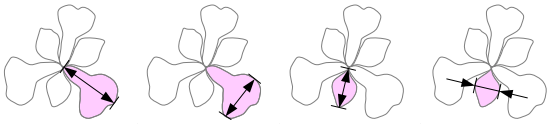
什么是鸢尾花数据集？

鸢尾花数据集是一种经典的用于机器学习和模式识别的数据集。数据集的全称为安德森鸢尾花卉数据集（Anderson's Iris data set），是植物学家埃德加·安德森（Edgar Anderson）在加拿大魁北克加斯帕半岛上的采集的鸢尾花样本数据。它包含了 150 个样本，分为三个不同品种的鸢尾花（山鸢尾、变色鸢尾和维吉尼亚鸢尾），每个品种 50 个样本。每个样本包含了四个特征：花萼长度、花萼宽度、花瓣长度和花瓣宽度。

鸢尾花数据集由统计学家罗纳德·费舍尔（Ronald Fisher）在 1936 年引入，并被广泛用于模式识别和机器学习的教学和研究。这个数据集是机器学习领域的一个基准测试数据集，被用来评估分类算法的性能。

鸢尾花数据集在机器学习应用中有很多用途。它经常被用来进行分类任务，即根据花的特征将其分为不同的品种。许多分类算法和模型，如 K 近邻、决策树、支持向量机和神经网络等，都可以使用鸢尾花数据集进行训练和测试。

由于鸢尾花数据集是一个相对简单的数据集，它也常用于机器学习的入门教学和实践。通过对这个数据集的分析和建模，学习者可以了解特征工程、模型选择和评估等机器学习的基本概念和技术。矩阵是一个由数字或符号排列成的矩形阵列。简单来说，矩阵就是个表格。矩阵在数据、统计、线性代数和几何学中扮演着重要的角色。



Index	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Species C
1	5.1	3.5	1.4	0.2	
2	4.9	3	1.4	0.2	
3	4.7	3.2	1.3	0.2	Setosa C_1
...	...	...	...	...	
49	5.3	3.7	1.5	0.2	
50	5	3.3	1.4	0.2	
51	7	3.2	4.7	1.4	
52	6.4	3.2	4.5	1.5	
53	6.9	3.1	4.9	1.5	Versicolor C_2
...	...	...	...	...	
99	5.1	2.5	3	1.1	
100	5.7	2.8	4.1	1.3	
101	6.3	3.3	6	2.5	
102	5.8	2.7	5.1	1.9	
103	7.1	3	5.9	2.1	Virginica C_3
...	...	...	...	...	
149	6.2	3.4	5.4	2.3	
150	5.9	3	5.1	1.8	

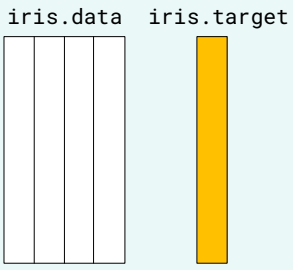
图 14. 鸢尾花数据，数值数据单位为厘米（cm）

对于鸢尾花数据，或其他大得多的数据集，我们则需要用 NumPy Array 或 Pandas DataFrame 这两种数据类型来保存、调用、运算。NumPy Array 或 Pandas DataFrame 是本书中最常见的数据类型。

以鸢尾花数据集为例，如代码 14 所示，我们可以从 Scikit-Learn 中导入鸢尾花数据集。我们可以发现数据类型是 numpy.ndarray，即 Numpy 多维数组。特别地，用 X.ndim 可以计算得到 X 的维度为 2，即二维数组，相当于一个矩阵。

如图 15 所示，从 Seaborn 导入的鸢尾花数据集保存类型为 pandas.core.frame.DataFrame，即 Pandas 数据帧。而这个数据帧整体相当于一个表格，有行索引和列标签。

```
# 导入包
from sklearn.datasets import load_iris

# 使用load_iris函数加载Iris数据集
iris = load_iris()

# Iris数据集的特征存储在iris.data中
X = iris.data
type(X) # numpy.ndarray

# Iris数据集的目标（标签）存储在iris.target中
y = iris.target
type(y) # numpy.ndarray
```



代码 14. 从 Scikit-learn 导入鸢尾花数据集 | Bk1_Ch05_18.ipynb

```
# 导入包
import seaborn as sns

# 使用seaborn.load_dataset函数加载Iris数据集
iris_df = sns.load_dataset("iris")

# 查看数据集的前5行
iris_df.head()
type(iris_df) # pandas.core.frame.DataFrame
```

图 15. 从 Seaborn 导入鸢尾花数据集 | Bk1_Ch05_19.ipynb

如图 16 所示，$\boldsymbol{X}$ 任一行向量代表一朵特定鸢尾花样本花萼长度、花萼宽度、花瓣长度和花瓣宽度测量结果。而 $\boldsymbol{X}$ 某一列向量为鸢尾花某个特征（花萼长度、花萼宽度、花瓣长度、花瓣宽度）的样本数据。

也是从几何角度来看，$\boldsymbol{X}$ 行向量相当于是 4 维空间中的 150 个箭头；$\boldsymbol{X}$ 列向量相当于是 150 维空间中的 4 个箭头。

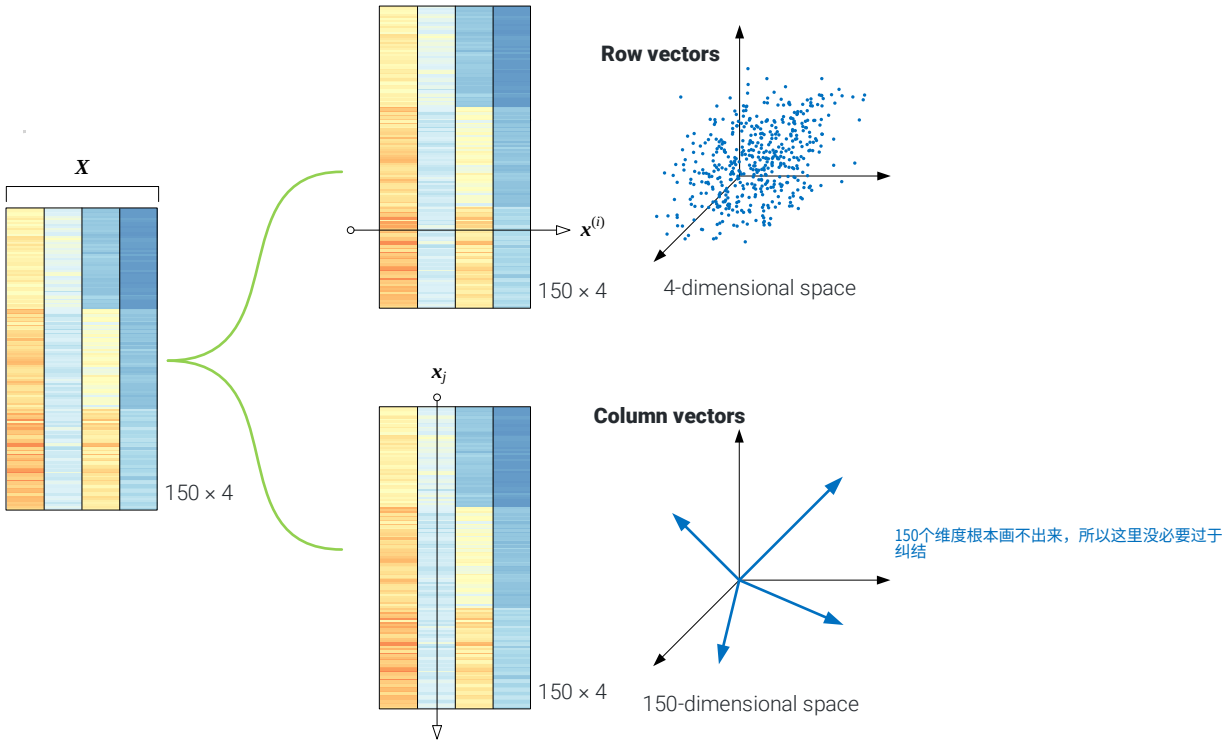
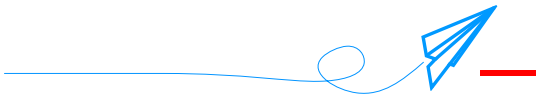


图 16. 矩阵可以分割成一系列行向量或列向量

请大家完成下面 1 道题目。

Q1. 本章的唯一的题目就是请大家在 JupyterLab 中练习本章正文给出的示例代码。

* 不提供答案。



鸢尾花书的读者很快就会发现，线性代数是整套书各种数学工具的核心，也是机器学习绕不过的五指山、必须走的独木桥。这也是为什么《编程不难》急不可耐、不遗余力、见缝插针地在各个角落引入线性代数概念。

线性代数看上去很抽象、恐怖，实际上非常简洁、优雅。我相信鸢尾花书的读者中，很多人曾经被线性代数折磨的伤痕累累。即便如此，不管大家是线性代数的初识，还是发誓老死不相往来的宿敌，请大家张开双臂，敞开胸怀。很快你就会发现线性代数的伟力，甚至还会惊叹于她的美。

特别是和数据、几何、微积分、统计结合起来之后，线性代数就会脱胎换骨，瞬间变得亭亭玉立、楚楚动人。