

5

Data Types in Python

Python 数据类型

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



```
XXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXX
```



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。

注意，大小写问题，比如 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。

注意，对于 Python 初学者，请大家切记完全没有必要熟练掌握每一种数据类型。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

Python 有三种内置数字类型：

- ▶ 整数 (int)：表示整数，没有小数部分。例如，42、-123、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。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

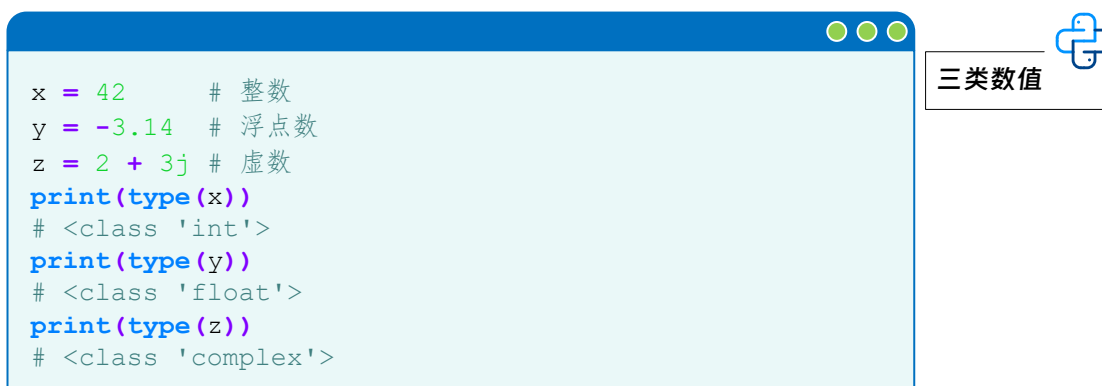
代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

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。

图 1 是一些示例，请大家在 JupyterLab 中自行练习。



```
x = 42      # 整数
y = -3.14   # 浮点数
z = 2 + 3j   # 虚数
print(type(x))
# <class 'int'>
print(type(y))
# <class 'float'>
print(type(z))
# <class 'complex'>
```

三类数值

图 1. Python 中三类数值

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，本书前文还介绍过的自加运算 (+=)、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。

图 2 是一些示例，请大家在 JupyterLab 中自行练习。

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，就会导致 `ValueError` 异常。

本书第 7 章将专门介绍如何处理异常。

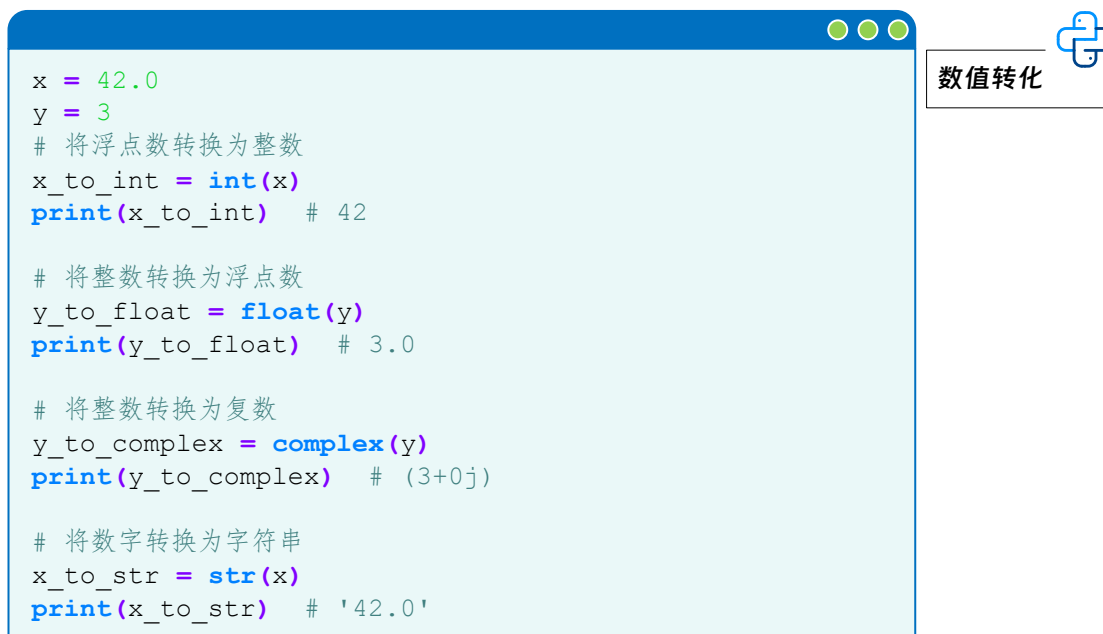


图 2. Python 中数值转换



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 π 、自然对数底数 e 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。

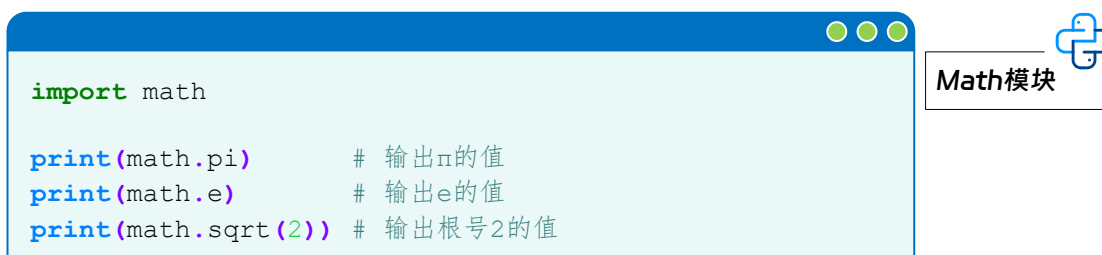


图 3. Math 模块中的特殊数值

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。

注意，大家日后会发现我们一般很少用到 Math 模块，会直接采用 NumPy、Pandas 中的运算函数。

5.3 字符串

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。请大家在 JupyterLab 中练习图 4 代码。三引号 `'''` 或 `"""` 一般用来创建多行字符串。

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。

请大家用 `len()` 函数获得图 4 每个字符串的长度，即字符串中字符个数。

注意，Python 中单字符也是字符串类型。



```

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

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

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

# 使用乘号*将一个字符串复制多次
str5 = 'Scikit-Learn is fun! ' #
字符串最后有一个空格
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
str7 = '123'
str8 = str7 * 3
print(str8)

str9 = '456'
str10 = str9 + str7
print(str10)
print(type(str10))

```

字符串定义

图 4. 字符串定义

索引、切片

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。

如图 5 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。

可以使用负数索引来从字符串的末尾开始计算位置。例如，-1 表示倒数第一个字符，-2 表示倒数第二个字符，依此类推。请大家自行在 JupyterLab 中练习图 7。

图 7 代码中使用了 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序号。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序号。

本书第 7 章将专门介绍 for 循环。

在代码中，f-字符串 (formatted string) 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序号信息。

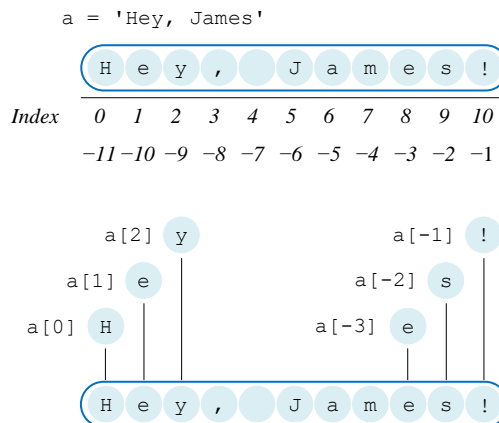


图 5. 字符串的索引

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` 序号对应的字符，但是不包括 `end` 位置的字符，相当于“左闭右开”区间。

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]`。

注意，复制字符串可以采用 `string_name[:]` 实现。

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！

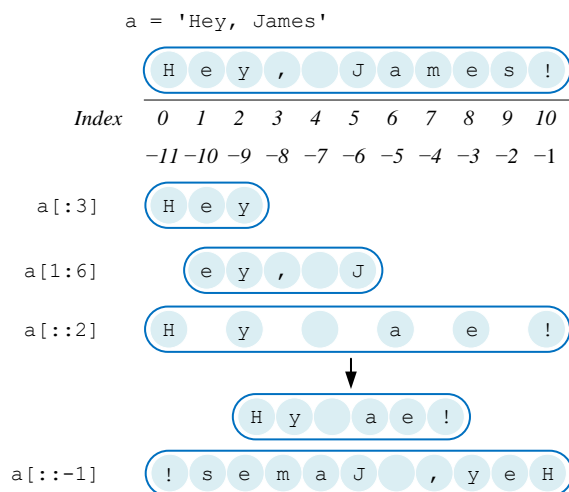


图 6. 字符串的切片

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。

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

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

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

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

# 切片
# 取出前3个字符，序号为0、1、2
print(greeting_str[:3])

# 取出序号1、2、3、4、5，不含0，不含6
print(greeting_str[1:6])

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

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

字符串索引、切片



图 7. 字符串索引和切片

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。

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!"
```

以下是一些常见的 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。

5.4 列表

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。

图 10 代码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 8 所示，这个列表中序号为 4 的元素 (从左到右第 5 个元素) 还是个列表，相当于嵌套。

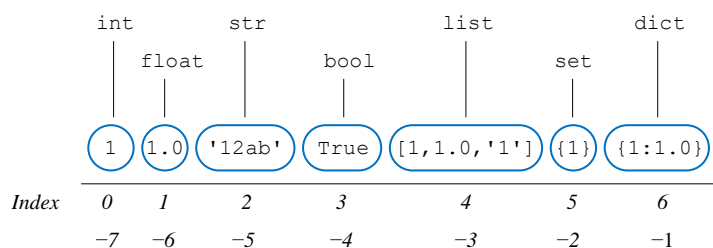


图 8. 混合列表

图 10 还给出 list 常用的索引方法，请大家在 JupyterLab 中练习。列表的索引、切片方式和字符串类似，我们不再展开。其中大家需要注意的是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，如图 9 所示。

请大家在 JupyterLab 中练习图 11 给出的 list 常见方法、操作。

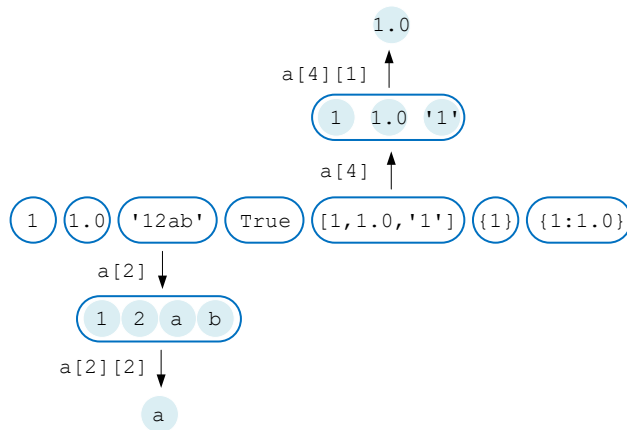


图 9. 混合列表的索引

```

# 创建一个混合列表

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
for index, item in enumerate(my_list):
    type_i = type(item)
    print(f"元素: {item}, 序号: {index}, 类型: {type_i}")

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

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

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

# 取出序号1、2、3, 不含0, 不含4
print(my_list[1:4])

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

# 指定步长-1, 倒序
print(my_list[::-1])

# 提取列表中的列表某个元素
print(my_list[4][1])

```

列表索引、
切片

图 10. 列表索引和切片

```

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

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

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

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

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

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

# 判断一个元素是否在列表中
if '123' in my_list:
    print("Yes")
else:
    print("No")

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

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

```

列表常见方
法和操作



图 11. 列表常用方法、操作

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 12 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 中练习图 13。

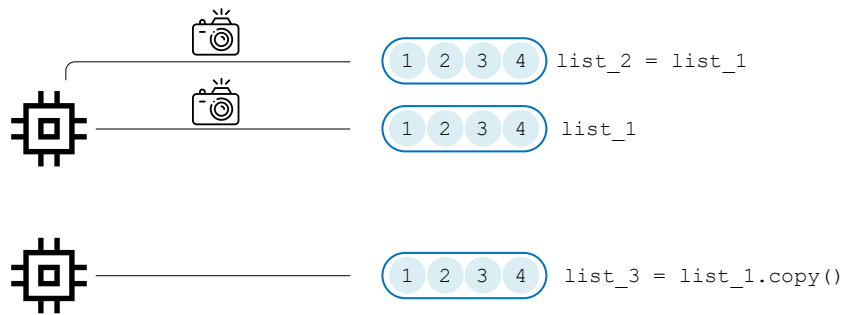



图 12. 视图，还是副本？

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

# 赋值，视图
list2 = list1
# 拷贝，副本（浅拷贝）
list3 = list1.copy()

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

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



视图 vs 副本

图 13. 视图 vs 副本

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 14 代码比较不同复制，请大家自行学习。



深复制

```

import copy

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

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

# 只深复制一层
list2 = list1.copy()
list3 = list1[:]
list4 = list(list1)
list5 = [*list1]

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

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

list3[3][0] = 'x1'
list3[2] = 'worked_2'

list4[3][0] = 'x2'
list4[2] = 'worked_3'

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 14. 浅复制、深复制

5.5 其他数据类型

元组

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

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。

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 15 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
```

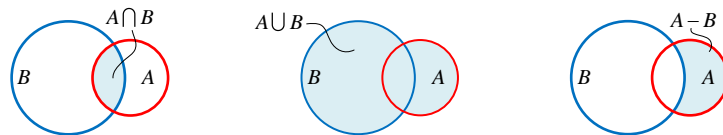


图 15. 交集、并集、差集

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，请大家自行学习图 16。

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。



图 16. 有关字典的常见操作

5.6 矩阵、向量：线性代数概念

抛开本章前文这些数据类型，数学上我们最关心的类型是——矩阵、向量。

简单来说，矩阵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 A 、 B 、 V 、 X 。

向量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 x 、 a 、 b 、 v 、 u 。

如图 17 所示，一个 $n \times D$ (n by capital D) 矩阵 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 X 的行索引就是 1、2、3、...、 n 。矩阵 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。

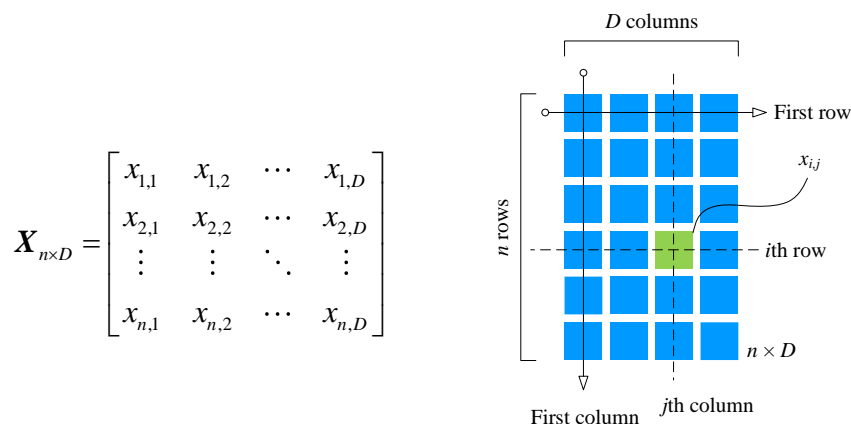



图 17. $n \times D$ 矩阵 X

从统计数据角度， n 是样本个数， D 是样本数据特征数。如图 18 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 $n = 150$ ， $D = 4$ 。



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	Setosa C_1
2	4.9	3	1.4	0.2	
3	4.7	3.2	1.3	0.2	
...	
49	5.3	3.7	1.5	0.2	
50	5	3.3	1.4	0.2	Versicolor C_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	
...	
99	5.1	2.5	3	1.1	Virginica C_3
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	
...	
149	6.2	3.4	5.4	2.3	
150	5.9	3	5.1	1.8	




图 18. 鸢尾花数据，数值数据单位为厘米 (cm)



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。



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。

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。

矩阵可以视作由一系列行向量、列向量构造而成。

如图 19 所示， 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 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。

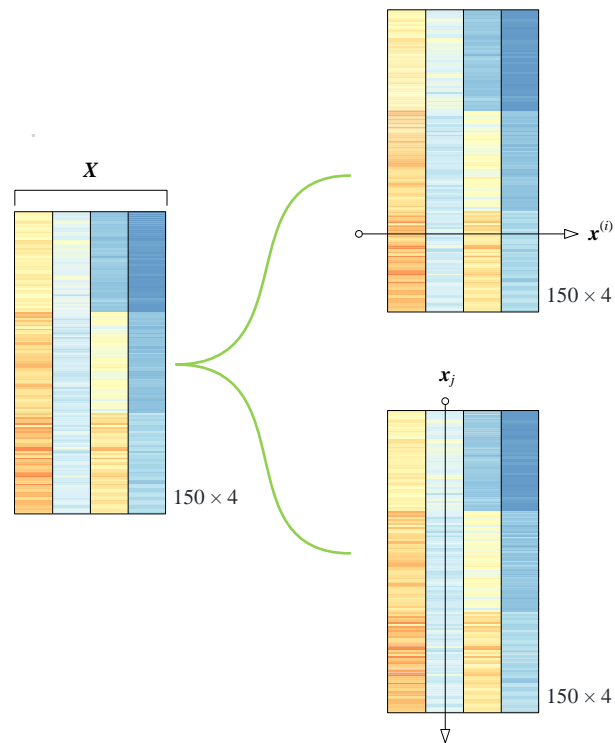


图 19. 矩阵可以分割成一系列行向量或列向量



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。