Artificial Intelligence Experimental Manual

人工智能实验课程手册（上册）

浙江大学

内部资料，请勿传播!!!

目录

特别声明

欢迎选修人工智能实验课程！

本手册为内部资料，作为浙江大学《人工智能实验课程》的参考教材使用，**请勿传播！**本手册的内容大多来源于《Python 编程》以及互联网上大量的优秀资料，如维基百科、知乎等网站，在此表示感谢。

本手册分为上下两册。上册主要介绍 Python 语言以及常用的标准库，下册主要介绍常用的人工智能（机器学习）算法及实现过程。

此外，本手册可能存在不少错误，如果发现错误欢迎大家及时反馈至本人邮箱.

Yunlong Yu

前沿

本手册读者对象

选修人工智能实验课程的大三学生。

如果您是人工智能领域的初学者，不必担心，我们会由浅入深地向您介绍人工智能课程的基本方法和所需要的基础知识。

如果您是一位有人工智能或机器学习领域知识的人，可以不必在已掌握的知识上浪费时间，而是直接学习所需要的知识。

本手册主要内容

本手册主要介绍 **Python** 以及基于**Sklearn**的各种机器学习算法实现。本手册服务于人工智能实验课程，因此以实验为主。不过，在尽可能的情况下，本手册将提供人工智能及机器学习的基本的原理。

本手册结构

本手册首先介绍 Python 这一编程语言。从这里开始，会介绍 Python 的基本语法以及编程规则，进而利用 Python 这一工具实现机器学习的基本算法。本手册分为如下 3 个部分。

- 了解 **Python**。首先介绍 Python 的开发历史和优势，然后介绍 Python 的基本语言，包括它的语法和模块。
- 机器学习入门之 **Sklearn** 介绍。
- 机器学习算法及 Python 实验。

第 1 章 初步了解 Python

1.1 Python 的介绍

官方对 Python 的介绍： Python 是一个易于学习的、功能强大的编程语言。它具有高效的高级数据结构和能够简单有效地实现面向对象编程。Python 优美的语法和动态类型，连同解释型特性一起，使其在多个平台的许多领域都成为脚本处理以及快速应用开发的理想语言。

名字背后的故事： Python 是贵铎 · 范 · 罗萨姆 Guido van Rossum （图 1.1）在阿姆斯特丹于 1989 年圣诞节期间，为了打发圣诞节的无趣，开发的一个新的解释型脚本语言并以 BBC 的喜剧《Monty Python's Flying Circus》给这个语言命名。

Python 的设计思想： Python 被设计成“符合大脑思维习惯”的，采用极简主义的设计理念，加以统一规范的交互模式。这使得 Python 易于学习、理解和记忆。Python 开发者的哲学是“用一种方法，最好是只有一种方法来做一件事”。

Python 是完全面向对象的编程语言，函数、模块、数字、字符串等内置类型都是对象。它的类支持多态、操作符重载、和多重继承等高级 OOP 概念，并且 Python 特有的简洁的语法和类型使得 OOP 十分易于使用。当然 OOP 只是 Python 的一个选择而已，就像 C++ 一样，Python 既支持面向对象编程，也支持面向过程编程的模式。

Python 是一种解释型语言，目前 Python 的标准实现方式是将源代码的语句编译（转换）为字节码格式，然后通过解释器将字节码解释出来。Python 没有将代码编译成底层的二进制代码，所以相较于 C 和 C++ 等编译型语言，Python 的执行速度会慢一些。但是 Python 的解释型语言特性提高了开发者开发速度，同时使它拥有解释型语言易于编写和调试等优点。

Python 本身被设计为可扩展的，并非所有的特性和功能都集成到语言核心。Python 提供了丰富的 API 和工具，以便程序员能够轻松地使用 C/C++ 语言来编写扩充模块。Python 为我们提供了非常完善的基础代码库，覆盖了正则表达式、网络、多线程、GUI、数据库、等领域。除了内置的库外，Python 还有大量的第三方库，供你直接使用。

Python 编译器本身也可以被集成到其它需要脚本语言的程序内，因此，很多人还把 Python 作为一种“胶水语言”（glue language）使用，使用 Python 将其他语言编写的程序进行集成和

图 1.1: Python 语言的发明人 Guido van Rossum

封装。2004 年，Python 已在 Google 内部使用。Google Engine 使用 C++ 编写性能要求极高的部分，然后用 Python 或 Java/Go 调用相应的模块。他们的目的是 “Python where we can, C++ where we must"，在操控硬件的场合使用 C++，在快速开发时候使用 Python。

Python 的优点：

- 软件质量高

  Python 秉承了简洁、清晰的语法，以及高度一致的编程模式。始终如一的设计风格，可以保证开发出相当规范的代码。针对错误，Python 提供了 “安全合理” 的退出机制。Python 支持异常处理，能有效捕获和处理程序运行时发生的错误，使你能够监控这些错误并进行处理。Python 代码能打包成模块和包，方便管理和发布，很适合团队协同开发。

- 开发速度快

  Python 致力于开发速度的最优化：简洁的语法、动态的类型、无需编译、丰富的库支持等特性使得程序员可以快速的进行项目开发。Python 往往只要几十行代码就可以开发出需要几百行 C 代码的功能。Python 解析器能很方便地进行代码调试和测试，也可作为一个编程接口嵌入一个应用程序中。这就使的在开发过程中可以直接进行调试，而避免了耗时而又麻烦的编译过程，大大提高了开发的速度和效率。在 Python 中，由于内存管理是由 Python 解释器负责的，所以开发人员就可以从内存管理事务中解放出来，仅仅致力于开发计划中首要的应用程序设计。这使得 Python 编写的程序错误更少、更加健壮、开发周期更短。

- 功能强大

  Python 的功能足够强大，本身也足够强壮，它还有许多面向其他系统的接口，所以完全可以使用 Python 开发整个系统的原型。为了完成更多特定的任务，Python 内置了许多预编码的库工具，从正则表达式到网络编程，再到数据库编程都支持。在 web 领域、数据分析领域等，Python 还有强大的框架帮你快速开发你的服务。例如：Django、TruboGears、Pylons 等。

- 易于扩展

  Python 易于扩展，（对于 CPython）可以通过 C 或 C++ 编写的模块进行功能扩展，使其能够成为一种灵活的黏合语言，可以脚本化处理其他系统和组件的行为。

- 跨平台

  Python 是跨平台的。在各种不同的操作系统上（Linux、Windows、MacOS、Unix 等）都可以看到 Python 的身影。因为 Python 是用 C 写的，又由于 C 的可移植性，使得 Python 可以运行在任何带有 ANSI C 编译器的平台上。尽管有一些针对不同平台开发的特有模块，但是在任何一个平台上用 Python 开发的通用软件都可以稍事修改或者原封不动的在其他平台上运行。这种可移植性既适用于不同的架构，也适用于不同的操作系统。

1.2 准备工作

在开始用 Python 编程之前，首先要下载并安装 Python。打开https://www.python.org/downloads，进入到图1.2页面。然后下载自己电脑对应系统的 Python 版本，并进行安装。

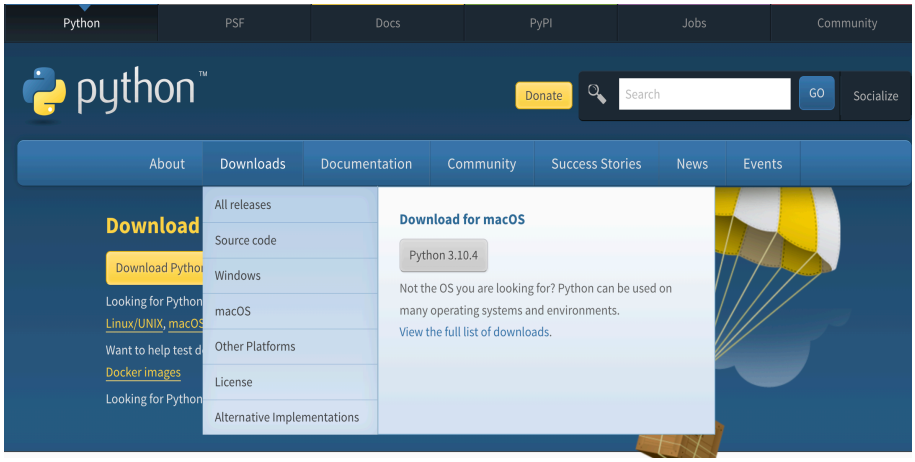

图 1.2: Python 下载界面

以 Windows 系统为例，下载结束后，双击以运行程序。最好接受 Python 的所有默认提示。安装过程大概需要几分钟。安装完毕后，测试 Python 是否被正确安装。单击 Windows 的 Start 菜单，找到 All Programs，选择 IDLE (Python GUI)，等待程序加载。

一旦 IDLE 启动，输入 “I Love Python”，并按 Enter 键。如果 Python 运行正常，则会返回：

```
'I Love Python'
```

恭喜你！你已经成功安装了 Python，并且正式步入了成为编程大师的征途。

注 在非 Windows 系统上安装 Python。如果使用 Mac 系统或者 Linux 系统，很幸运，它已经预安装了 Python。打开终端，输入 python 后就进入了编程模式。不过，它安装可能不是最新版本的 Python。如果想安装最新版本的 Python，可以去官网下载对应的版本并进行安装。

1.3 Sublime Text 简介

安装 Python 之后，你还需要安装一款文本编译器。Sublime Text 是一款简单的文本编译器，可以在任何现代操作系统中安装。你几乎能直接在 Sublime Text 中执行所有程序。在 Sublime Text 中执行程序时，代码将在其内嵌的终端会话中运行，让你能够轻松地看到输出。

Sublime Text 是一款适合初学者的编辑器，但很多专业编程人员也在使用它。在学习 Python 的过程中熟练掌握 Sublime Text 之后，可继续使用它来编写复杂的大型项目。Sublime Text 的许可条件非常宽松，可以一直免费使用，但如果你喜欢它并想长期使用，其开发者会要求你购买许可证。

安装 Sublime Text: 要下载 Sublime Text 安装程序，可访问Sublime Text网站主页，单击 Download 链接，并查找 Windows 安装程序。下载安装程序后运行它，并接受所有的默认设置。

1.4 Anaconda 简介

Anaconda 专门为了方便使用 Python 进行数据科学研究而建立的一组软件包，涵盖了数据科学领域常见的 Python 库，并且自带了专门用来解决软件环境依赖问题的 conda 包管理系统。主要是提供了包管理与环境管理的功能，可以很方便地解决多版本 python 并存、切换以及各

种第三方包安装问题。Anaconda 利用工具/命令 conda 来进行 package 和 environment 的管理，并且已经包含了 Python 和相关的配套工具。

conda 可以理解为一个工具，也是一个可执行命令，其核心功能是包管理与环境管理。包管理与 pip 的使用类似，环境管理则允许用户方便地安装不同版本的 python 并可以快速切换。

Anaconda 则是一个打包的集合，里面预装好了 conda、某个版本的 python、众多 packages、科学计算工具等等，所以也称为 Python 的一种发行版。其实还有 Miniconda，它只包含最基本的内容——python 与 conda，以及相关的必须依赖项，对于空间要求严格的用户，Miniconda 是一种选择。

conda 将几乎所有的工具、第三方包都当做 package 对待，甚至包括 python 和 conda 自身！因此，conda 打破了包管理与环境管理的约束，能非常方便地安装各种版本 python、各种 package 并方便地切换。

请参考Anaconda 配置环境。

1.5 Pycharm 简介

Pycharm 是开发工具，专业术语称作 IDE**。可以编写 Python 程序的 IDE 有很多，据统计唯独 Pycharm 用户最多，带有一整套可以帮助用户在使用 Python 语言开发时提高其效率的工具，比如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该 IDE 提供了一些高级功能，以用于支持 Django 框架下的专业 Web 开发。

1.6 Jupyter 简介

Jupyter Notebook (此前被称为 IPython notebook) 是一个交互式笔记本，支持运行 40 多种编程语言。

Jupyter Notebook 的本质是一个 Web 应用程序，便于创建和共享文学化程序文档，支持实时代码，数学方程，可视化和 Markdown。用途包括：数据清理和转换，数值模拟，统计建模，机器学习等等。

1.7 运行程序 hello_world.py

编写第一个程序前，在系统中创建一个名为 python_work 的文件夹，用于存储你开发的项目。文件名和文件夹名称最好使用小写字母，并使用下划线代替空格，因为 Python 采用了这些命名约定。

启动 Sublime Text，再选择菜单 File Save As 将 Sublime Text 创建的空文件存储到文件夹 python_work 中，并将其命名为 hello_world.py。文件扩展名.py 告诉 Sublime Text，文件中的代码是使用 Python 编写的，这能让它知道如何运行这个程序，并以有帮助的方式突出其中的代码。

保存这个文件后，在其中输入如下代码行：

hello_world.py

```
print ("Hello Python world!")
```

在你的系统中，如果能使用命令 python 来启动 Python 3，可以选择菜单 Tools Build 或按 Ctrl + B（在 macOS 系统中为 Command + B）来运行程序。如果需要像前一节那样配置 Sublime Text，请选择菜单 Tools Build System Python 3 来运行这个程序。从此以后，你就可以选择菜单 Tools Build 或按 Ctrl + B（或 Command + B）来运行程序了。

在 Sublime Text 的底部，将出现一个终端窗口，其中包含如下输出：

```
Hello Python world!
[Finished in 0.48ms]
```

如果看不到上述输出，可能是因为这个程序出了点问题。请检查你输入的每个字符。是否不小心将 print 的首字母大写了？是否遗漏了引号或圆括号？编程语言的语法非常严格，只要不满足要求，就会报错。

1.7.1 从终端运行 Python 程序

你编写的大多数程序将直接在文本编辑器中运行，但有时候从终端运行程序很有用。例如，你可能想直接运行既有的程序。在任何安装了 Python 的系统上都可以这样做，前提是你知道如何进入程序文件所在的目录。为尝试这样做，请确保将文件 hello_world.py 存储到了桌面的文件夹 python_work 中。

在 Windows 系统中从终端运行 Python 程序：

在命令窗口中，可以使用终端命令 cd （表示 change directory，即切换目录）在文件系统中导航。使用命令 dir （表示 directory，即目录）可以显示当前目录中的所有文件。为运行程序 hello_world.py，请打开一个新的终端窗口，并执行下面的命令：

```
C:\> cd Desktop\python_work
C:\Desktop\python_work> dir
hello_world.py
C:\Desktop\python_work> python hello_world.py
Hello Python world!
```

大多数程序可直接从编辑器运行，但待解决的问题比较复杂时，你编写的程序可能需要从终端运行。

在 Linux 和 macOS 系统中从终端运行 Python 程序：

在 Linux 和 macOS 系统中，从终端运行 Python 程序的方式相同。在终端会话中，可以使用终端命令 cd （表示 change directory，即切换目录）在文件系统中导航。使用命令 ls （表示 list，即列表）可以显示当前目录中所有未隐藏的文件。

为运行程序 hello_world.py，请打开一个新的终端窗口，并执行下面的命令：

```
$ cd Desktop/python_work/
/Desktop/python_work$ ls
hello_world.py
/Desktop/python_work$ python hello_world.py
Hello Python world!
```

第 2 章 变量和简单数据类型

在本章中，你将学习可在 Python 程序中使用的各种数据，还将学习如何在程序中使用变量来表示这些数据。

2.1 变量

下面来尝试在 hello_world.py 中使用一个变量。在这个文件开头添加一行代码，并对第二行代码进行修改，如下所示：

hello_world.py

```
message = "Hello Python world!"
print(message)
```

我们添加了一个名为 message 的变量。每个变量都指向一个值——与该变量相关联的信息。在这里，指向的值为文本"Hello Python world!" 。

```
message = "Hello Python world!"
print(message)
message = "Hello Python Crash Course world!"
print(message)
```

运行这个程序，输出两行：

```
Hello Python world!
Hello Python Crash Course world!
```

在程序中可随时修改变量的值，而 Python 将始终记录变量的最新值。

2.1.1 变量的命名和使用

在 Python 中使用变量时，需要遵守一些规则和指南。违反这些规则将引发错误，而指南旨在让你编写的代码更容易阅读和理解。请务必牢记下述有关变量的规则。

- 变量名只能包含字母、数字和下划线。变量名能以字母或下划线打头，但不能以数字打头。例如，可将变量命名为 message_1 ，但不能将其命名为 1_message 。
- 变量名不能包含空格，但能使用下划线来分隔其中的单词。例如，变量名 greeting_message 可行，但变量名 greeting message 会引发错误。
- 不要将 Python 关键字和函数名用作变量名，即不要使用 Python 保留用于特殊用途的单词，如 print。
- 变量名应既简短又具有 述性。例如，name 比 n 好，student_name 比 s_n 好，name_length 比 length_of_persons_name 好。
- 慎用小写字母 l 和大写字母 O ，因为它们可能被人错看成数字 1 和 0。

要创建良好的变量名，需要经过一定的实践，在程序复杂而有趣时尤其如此。随着编写的程序越来越多，并开始阅读别人编写的代码，你将越来越善于创建有意义的变量名。

注 就目前而言，应使用小写的 Python 变量名。虽然在变量名中使用大写字母不会导致错误，但是大写字母在变量名中有特殊含义。

2.1.2 使用变量时避免命名错误

我们将有意地编写一些引发错误的代码。请输入下面的代码，包括其中拼写不正确、以粗体显示的单词 mesage：

```
message = "Hello Python world!"
print(mesage)
```

程序存在错误时，Python 解释器将竭尽所能地帮助你找出问题所在。程序无法成功运行时，解释器将 供一个 traceback。traceback 是一条记录，指出了解释器尝试运行代码时，在什么地方陷入了困境。下面是你不小心错误地拼写了变量名时，Python 解释器提供的 traceback：

```
Traceback (most recent call last):
File "hello_world.py", line 2, in <module>
print(mesage)
NameError: name 'mesage' is not defined
```

2.2 字符串

大多数程序定义并收集某种数据，然后使用它们来做些有意义的事情。有鉴于此，对数据进行分类大有裨益。我们将介绍的第一种数据类型是字符串。字符串虽然看似简单，但能够以很多不同的方式使用。

字符串就是一系列字符。在 Python 中，用引号括起的都是字符串，其中的引号可以是单引号，也可以是双引号，如下所示：

```
"This is a string."
'This is also a string.'
```

这种灵活性让你能够在字符串中包含引号和撇号：

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

2.2.1 修改字符串的大小写

对于字符串，可执行的最简单的操作之一是修改其中单词的大小写。请看下面的代码，并尝试判断其作用：

name.py

```
name = "ada lovelace"
print(name.title())
```

将这个文件保存为 name.py，再运行它。你将看到如下输出：

```
Ada Lovelace
```

在这个示例中，变量 name 指向小写的字符串"ada lovelace" 。在函数调用 print() 中，方法 title() 出现在这个变量的后面。方法是 Python 可对数据执行的操作。在 name.title() 中，name 后面的句点（.）让 Python 对变量 name 执行方法 title() 指定的操作。每个方法后面都跟着一对圆括号，这是因为方法通常需要额外的信息来完成其工作。这种信息是在圆括号内 供的。函数 title() 不需要额外的信息，因此它后面的圆括号是空的。

方法 title() 以首字母大写的方式显示每个单词，即将每个单词的首字母都改为大写。这很有用，因为你经常需要将名字视为信息。例如，你可能希望程序将值 Ada、ADA 和 ada 视为同一个名字，并将它们都显示为 Ada 。

还有其他几个很有用的大小写处理方法。例如，要将字符串改为全部大写或全部小写，可以像下面这样做：

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

这些代码的输出如下：

```
ADA LOVELACE
ada lovelace
```

存储数据时，方法 lower() 很有用。很多时候，你无法依靠用户来 供正确的大小写，因此需要将字符串先转换为小写，再存储它们。以后需要显示这些信息时，再将其转换为最合适的大小写方式。

2.2.2 在字符串中使用变量

在有些情况下，你可能想在字符串中使用变量的值。例如，你可能想使用两个变量分别表示名和姓，然后合并这两个值以显示姓名：

full_name.py

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(full_name)
```

这种字符串名为 f 字符串。f 是 format（设置格式）的简写，因为 Python 通过把花括号内的变量替换为其值来设置字符串的格式。上述代码的输出如下：

```
ada lovelace
```

使用 f 字符串可完成很多任务，如利用与变量关联的信息来创建完整的消息，如下所示：

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(f"Hello, {full_name.title()}!")
```

在这里，一个问候用户的句子中使用了完整的姓名，并使用方法 title() 来将姓名设置为合适的格式。这些代码显示一条格式良好的简单问候语：

```
Hello, Ada Lovelace!
```

还可以使用 f 字符串来创建消息，再把整条消息赋给变量：

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
message = f"Hello, {full_name.title()}!"
print(message)
```

上述代码也显示消息 Hello, Ada Lovelace! ，但将这条消息赋给了一个变量，这让最后的函数调用 print() 变得简单得多。

注 f 字符串是 Python 3.6 引入的。如果你使用的是 Python 3.5 或更早的版本，需要使用 format() 方法，而非这种 f 语法。要使用方法 format() ，可在圆括号内列出要在字符串中使用的变量。对于每个变量，都通过一对花括号来引用。这样将按顺序将这些花括号替换为圆括号内列出的变量的值，如下所示：

```
full_name = "{} {}".format(first_name, last_name)
```

2.2.3 使用制表符或换行符来添加空白

在编程中，空白泛指任何非打印字符，如空格、制表符和换行符。你可以使用空白来组织输出，让用户阅读起来更容易。

要在字符串中添加制表符，可使用字符组合\t

```
print("Python")
Python
print("\tPython")
    Python
```

要在字符串中添加换行符，可使用字符组合\n

2.2.4 删除空白

在程序中，额外的空白可能令人迷惑。对程序员来说，'python' 和'python ' 看起来几乎没什么两样，但对程序来说，它们却是两个不同的字符串。Python 能够发现'python ' 中额外的空白，并认为它意义重大——除非你告诉它不是这样的。

空白很重要，因为你经常需要比较两个字符串是否相同。一个重要的示例是，在用户登录网站时检查其用户名。不过在非常简单的情形下，额外的空格也可能令人迷惑。所幸，在 Python 中删除用户输入数据中的多余空白易如反掌。

Python 能够找出字符串开头和末尾多余的空白。要确保字符串末尾没有空白，可使用方法 rstrip() 。

```
favorite_language = 'python '
favorite_language
'python '
favorite_language.rstrip()
'python'
favorite_language
'python '
```

你还可以剔除字符串开头的空白，或者同时剔除字符串两边的空白。为此，可分别使用方法 lstrip() 和 strip()：

```
favorite_language = ' python '
favorite_language.rstrip()
' python'
favorite_language.lstrip()
'python '
favorite_language.strip()
'python'
```

2.3 数

在编程中，经常使用数来记录得分、表示可视化数据、存储 Web 应用信息，等等。Python 能根据数的用法以不同的方式处理它们。鉴于整数使用起来最简单，下面就先来看看 Python 是如何管理它们的。

整数： 在 Python 中，可对整数执行加（+）减（-）乘（*）除（/）运算。

在终端会话中，Python 直接返回运算结果。Python 使用两个乘号表示乘方运算。

Python 还支持运算次序，因此可在同一个表达式中使用多种运算。还可以使用圆括号来修改运算次序，让 Python 按你指定的次序执行运算。

```
2 + 3*4
14
(2 + 3) * 4
20
```

在这些示例中，空格不影响 Python 计算表达式的方式。它们的存在旨在让你在阅读代码时能迅速确定先执行哪些运算。

浮点数： Python 将所有带小数点的数称为浮点数。大多数编程语言使用了这个术语，它指出了这样一个事实：小数点可出现在数的任何位置。每种编程语言都必须细心设计，以妥善地处理浮点数，确保不管小数点出现在什么位置，数的行为都是正常的。

从很大程度上说，使用浮点数时无须考虑其行为。你只需输入要使用的数，Python 通常会按你期望的方式处理它们。

整数和浮点数： 将任意两个数相除时，结果总是浮点数，即便这两个数都是整数且能整除：

```
4/2
2.0
```

在其他任何运算中，如果一个操作数是整数，另一个操作数是浮点数，结果也总是浮点数：

```
1 + 2.0
3.0
2 * 3.0
6.0
3.0 ** 2
9.0
```

无论是哪种运算，只要有操作数是浮点数，Python 默认得到的总是浮点数，即便结果原本为整数也是如此。

2.3.1 常量

常量类似于变量，但其值在程序的整个生命周期内保持不变。Python 没有内置的常量类型，但 Python 程序员会使用全大写来指出应将某个变量视为常量，其值应始终不变：

```
MAX_CONNECTIONS = 5000
```

在代码中，要指出应将特定的变量视为常量，可将其字母全部大写。

2.4 注释

在大多数编程语言中，注释是一项很有用的功能。本书前面编写的程序中都只包含 Python 代码，但随着程序越来越大、越来越复杂，就应在其中添加说明，对你解决问题的方法进行大致的阐述。注释让你能够使用自然语言在程序中添加说明。

2.4.1 如何编写注释

在 Python 中，注释用井号（#）标识。井号后面的内容都会被 Python 解释器忽略，如下所示：

comment.py

```
# 向大家问好。
print("Hello Python people!")
```

2.4.2 该编写什么样的注释

编写注释的主要目的是阐述代码要做什么，以及是如何做的。在开发项目期间，你对各个部分如何协同工作了如指掌，但过段时间后，有些细节你可能不记得了。当然，你总是可以通过研究代码来确定各个部分的工作原理，但通过编写注释以清晰的自然语言对解决方案进行概述，可节省很多时间。

要成为专业程序员或与其他程序员合作，就必须编写有意义的注释。当前，大多数软件是合作编写的，编写者可能是同一家公司的多名员工，也可能是众多致力于同一个开源项目的人员。训练有素的程序员都希望代码中包含注释，因此你最好从现在开始就在程序中添加描述述性注释。

作为新手，最值得养成的习惯之一就是在代码中编写清晰、简洁的注释。

如果不确定是否要编写注释，就问问自己：在找到合理的解决方案之前，考虑了多个解决方案吗？如果答案是肯定的，就编写注释对你的解决方案进行说明吧。相比回过头去再添加注释，删除多余的注释要容易得多。从现在开始，本书的示例都将使用注释来阐述代码的工作原理。

第 3 章 列表简介

内容提要

在本章中，你将学习列表是什么以及如何使用列表元素。列表让你能够在一个地方存储成组的信息，其中可以只包含几个元素，也可以包含数百万个元素。列表是新手可直接使用的最强大的 Python 功能之一，它融合了众多重要的编程概念。

3.1 列表是什么

列表由一系列按特定顺序排列的元素组成。你可以创建包含字母表中所有字母、数字 0 ~ 9 或所有家庭成员姓名的列表；也可以将任何东西加入列表中，其中的元素之间可以没有任何关系。列表通常包含多个元素，因此给列表指定一个表示复数的名称（如 letters 、digits 或 names ）是个不错的主意。

在 Python 中，用方括号（[] ）表示列表，并用逗号分隔其中的元素。下面是一个简单的列表示例，其中包含几种自行车：

bicycles.py

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
```

3.1.1 访问列表元素

列表是有序集合，因此要访问列表的任意元素，只需将该元素的位置（索引）告诉 Python 即可。要访问列表元素，可指出列表的名称，再指出元素的索引，并将后者放在方括号内。例如，下面的代码从列表 bicycles 中提取第一款自行车：

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])

trek
```

注 索引从 0 而不是 1 开始。

在 Python 中，第一个列表元素的索引为 0，而不是 1。多数编程语言是如此规定的，这与列表操作的底层实现相关。如果结果出乎意料，请看看你是否犯了简单的差一错误。第二个列表元素的索引为 1。根据这种简单的计数方式，要访问列表的任何元素，都可将其位置减 1，并将结果作为索引。例如，要访问第四个列表元素，可使用索引 3。

Python 为访问最后一个列表元素 供了一种特殊语法。通过将索引指定为-1，可让 Python 返回最后一个列表元素。

你可以像使用其他变量一样使用列表中的各个值。例如，可以使用 f 字符串根据列表中的值来创建消息。

下面尝试从列表中 取第一款自行车，并使用这个值创建一条消息：

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message = f"My first bicycle was a {bicycles[0].title()}."
print(message)
```

3.2 修改、添加和删除元素

你创建的大多数列表将是动态的，这意味着列表创建后，将随着程序的运行增删元素。例如，你创建一个游戏，要求玩家射杀从天而降的外星人。为此，可在开始时将一些外星人存储在列表中，然后每当有外星人被射杀时，都将其从列表中删除，而每次有新的外星人出现在屏幕上时，都将其添加到列表中。在整个游戏运行期间，外星人列表的长度将不断变化

3.2.1 修改列表元素

修改列表元素的语法与访问列表元素的语法类似。要修改列表元素，可指定列表名和要修改的元素的索引，再指定该元素的新值。

例如，假设有一个摩托车列表，其中的第一个元素为'honda' ，如何修改它的值呢?

motorcycles.py

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)

['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

3.2.2 在列表中添加元素

你可能出于众多原因要在列表中添加新元素。例如，你可能希望游戏中出现新的外星人、添加可视化数据或给网站添加新注册的用户。Python 提供了多种在既有列表中添加新数据的方式。

- 在列表末尾添加元素在列表中添加新元素时，最简单的方式是将元素附加（append）到列表。给列表附加元素时，它将添加到列表末尾。继续使用前一个示例中的列表，在其末尾添加新元素'ducati' ：

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)

['honda', 'yamaha', 'suzuki']
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
```

方法 append() 让动态地创建列表易如反掌。例如，你可以先创建一个空列表，再使用一系列函数调用 append() 来添加元素。下面来创建一个空列表，再在其中添加元素'honda'、'yamaha' 和'suzuki'：

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)

['honda', 'yamaha', 'suzuki']
```

这种创建列表的方式极其常见，因为经常要等程序运行后，你才知道用户要在程序中存储哪些数据。为控制用户，可首先创建一个空列表，用于存储用户将要输入的值，然后将用户 供的每个新值附加到列表中。

- 在列表中插入元素

  使用方法 insert() 可在列表的任何位置添加新元素。为此，你需要指定新元素的索引和值。

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)

['ducati', 'honda', 'yamaha', 'suzuki']
```

3.2.3 从列表中删除元素

你经常需要从列表中删除一个或多个元素。例如，玩家将空中的一个外星人射杀后，你很可能要将其从存活的外星人列表中删除；当用户在你创建的 Web 应用中注销账户时，你就需要将该用户从活动用户列表中删除。你可以根据位置或值来删除列表中的元素。

- 使用 del 语句删除元素

  如果知道要删除的元素在列表中的位置，可使用 del 语句。

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[0]
print(motorcycles)

['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

使用 del 可删除任意位置处的列表元素，条件是知道其索引。例如，下面演示了如何删除前述列表中的第二个元素'yamaha'：

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[1]
```

```
print(motorcycles)

['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

在这两个示例中，使用 del 语句将值从列表中删除后，你就无法再访问它了。

- 使用方法 pop() 删除元素

有时候，你要将元素从列表中删除，并接着使用它的值。例如，你可能需要获取刚被射杀的外星人的坐标和坐标，以便在相应的位置显示爆炸效果；在 Web 应用程序中，你可能要将用户从活跃成员列表中删除，并将其加入到非活跃成员列表中。方法 pop() 删除列表末尾的元素，并让你能够接着使用它。术语弹出（pop）源自这样的类比：列表就像一个栈，而删除列表末尾的元素相当于弹出栈顶元素。

下面来从列表 motorcycles 中弹出一款摩托车：

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)

['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

- 弹出列表中任何位置处的元素

实际上，可以使用 pop() 来删除列表中任意位置的元素，只需在圆括号中指定要删除元素的索引即可。

实际上，可以使用 pop() 来删除列表中任意位置的元素，只需在圆括号中指定要删除元素的索引即可。

```
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")
The first motorcycle I owned was a Honda.
```

每当你使用 pop() 时，被弹出的元素就不再在列表中了。

如果你不确定该使用 del 语句还是 pop() 方法，下面是一个简单的判断标准：如果你要从列表中删除一个元素，且不再以任何方式使用它，就使用 del 语句；如果你要在删除元素后还能继续使用它，就使用方法 pop() 。

- 根据值删除元素

有时候，你不知道要从列表中删除的值所处的位置。如果只知道要删除的元素的值，可使用方法 remove() 。

例如，假设要从列表 motorcycles 中删除值'ducati' 。

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
motorcycles.remove('ducati')
print(motorcycles

['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

使用 remove() 从列表中删除元素时，也可接着使用它的值。下面删除值'ducati' 并打印一条消息，指出要将其从列表中删除的原因：

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
print(motorcycles)
print(f"\nA {too_expensive.title()} is too expensive for me.")

['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
A Ducati is too expensive for me.
```

定义列表后，将值'ducati' 赋给变量 too_expensive 。接下来，使用这个变量来告诉 Python 将哪个值从列表中删除。最后，值'ducati' 已经从列表中删除，但可通过变量 too_expensive 来访问它。这让我们能够打印一条消息，指出将'ducati' 从列表 motorcycles 中删除的原因。

注 方法 remove() 只删除第一个指定的值。如果要删除的值可能在列表中出现多次，就需要使用循环来确保将每个值都删除。

第 3 章 练习

1. 嘉宾名单
   如果你可以邀请任何人一起共进晚餐（无论是在世的还是故去的），你会邀请哪些人？请创建一个列表，其中包含至少三个你想邀请的人，然后使用这个列表打印消息，邀请这些人来与你共进晚餐。
2. 修改嘉宾名单
   你刚得知有位嘉宾无法赴约，因此需要另外邀请一位嘉宾。
   - 在程序末尾添加一条 print 语句，指出哪位嘉宾无法赴约。
   - 修改嘉宾名单，将无法赴约的嘉宾的姓名替换为新邀请的嘉宾的姓名。
   - 再次打印一系列消息，向名单中的每位嘉宾发出邀请。
3. 添加嘉宾
   你刚找到了一个更大的餐桌，可容纳更多的嘉宾。请想想你还想邀请哪三位嘉宾。
   - 在程序末尾添加一条 print 语句，指出你找到了一个更大的餐桌。
   - 使用 insert() 将一位新嘉宾添加到名单开头。

- 使用 insert() 将另一位新嘉宾添加到名单中间。
- 使用 append() 将最后一位新嘉宾添加到名单末尾。
- 打印一系列消息，向名单中的每位嘉宾发出邀请。

4. 缩减名单

你刚得知新购买的餐桌无法及时送达，因此只能邀请两位嘉宾。

- 在程序末尾添加一行代码，打印一条你只能邀请两位嘉宾共进晚餐的消息。
- 使用 pop() 不断地删除名单中的嘉宾，直到只有两位嘉宾为止。每次从名单中弹出一位嘉宾时，都打印一条消息，
- 让该嘉宾知悉你很抱歉，无法邀请他来共进晚餐。对于余下两位嘉宾中的每一位，都打印一条消息，指出他依然在受邀人之列。
- 使用 del 将最后两位嘉宾从名单中删除，让名单变成空的。打印该名单，核实程序结束时名单确实是空的。

3.3 组织列表

在你创建的列表中，元素的排列顺序常常是无法预测的，因为你并非总能控制用户 供数据的顺序。这虽然在大多数情况下是不可避免的，但你经常需要以特定的顺序呈现信息。有时候，你希望保留列表元素最初的排列顺序，而有时候又需要调整排列顺序。Python 提供了很多组织列表的方式，可根据具体情况选用。

3.3.1 使用方法 sort() 对列表永久排序

Python 方法 sort() 让你能够较为轻松地对列表进行排序。假设你有一个汽车列表，并要让其中的汽车按字母顺序排列。为简化这项任务，假设该列表中的所有值都是小写的。

cars.py

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)

['audi', 'bmw', 'subaru', 'toyota']
```

方法 sort() 永久性地修改列表元素的排列顺序。现在，汽车是按字母顺序排列的，再也无法恢复到原来的排列顺序。

还可以按与字母顺序相反的顺序排列列表元素，只需向 sort() 方法传递参数 reverse=True 即可。下面的示例将汽车列表按与字母顺序相反的顺序排列：

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)

['toyota', 'subaru', 'bmw', 'audi']
```

3.3.2 使用函数 sorted() 对列表临时排序

要保留列表元素原来的排列顺序，同时以特定的顺序呈现它们，可使用函数 sorted() 。函数 sorted() 让你能够按特定顺序显示列表元素，同时不影响它们在列表中的原始排列顺序。

下面尝试来对汽车列表调用这个函数。

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)

Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

注 调用函数 sorted() 后，列表元素的排列顺序并没有变。如果要按与字母顺序相反的顺序显示列表，也可向函数 sorted() 传递参数 reverse=True 。

3.3.3 倒着打印列表

要反转列表元素的排列顺序，可使用方法 reverse() 。假设汽车列表是按购买时间排列的，可轻松地按相反的顺序排列其中的汽车：

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)

['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

注 reverse() 不是按与字母顺序相反的顺序排列列表元素，而只是反转列表元素的排列顺序。方法 reverse() 永久性地修改列表元素的排列顺序，但可随时恢复到原来的排列顺序，只需对列表再次调用 reverse() 即可。

3.3.4 确定列表的长度

使用函数 len() 可快速获悉列表的长度。在下面的示例中，列表包含四个元素，因此其长度为 4 ：

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
len(cars)
```

4

第 4 章 操作列表

内容提要

在本章中，你将学习如何遍历整个列表，这只需要几行代码，无论列表有多长。循环让你能够对列表的每个元素都采取一个或一系列相同的措施，从而高效地处理任何长度的列表，包括包含数千乃至数百万个元素的列表。

4.1 遍历整个列表

你经常需要遍历列表的所有元素，对每个元素执行相同的操作。例如，在游戏中，可能需要将每个界面元素平移相同的距离；对于包含数字的列表，可能需要对每个元素执行相同的统计运算；在网站中，可能需要显示文章列表中的每个标题。需要对列表中的每个元素都执行相同的操作时，可使用 Python 中的 for 循环。

假设我们有一个魔术师名单，需要将其中每个魔术师的名字都打印出来。为此，可以分别获取名单中的每个名字，但这种做法会导致多个问题。例如，如果名单很长，将包含大量重复的代码。另外，每当名单的长度发生变化时，都必须修改代码。通过使用 for 循环，可以让 Python 去处理这些问题。

下面使用 for 循环来打印魔术师名单中的所有名字：

magicians.py

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)

alice
david
carolina
```

4.2 创建数值列表

需要存储一组数的原因有很多。例如，在游戏中，需要跟踪每个角色的位置，还可能需要跟踪玩家的几个最高得分；在数据可视化中，处理的几乎都是由数（如温度、距离、人口数量、经度和纬度等）组成的集合。

列表非常适合用于存储数字集合，而 Python供了很多工具，可帮助你高效地处理数字列表。明白如何有效地使用这些工具后，即便列表包含数百万个元素，你编写的代码也能运行得很好。

4.2.1 使用函数 range()

Python 函数 range() 让你能够轻松地生成一系列数。例如，可以像下面这样使用函数 range() 来打印一系列数：

first_numbers.py

```
for value in range(1, 5):
    print(value)

1
2
3
4
```

在这个示例中，range() 只打印数 1 ~ 4。这是编程语言中常见的差一行为的结果。函数 range() 让 Python 从指定的第一个值开始数，并在到达你指定的第二个值时停止。因为它在第二个值处停止，所以输出不包含该值（这里为 5）。

4.2.2 对数字列表执行简单的统计计算

有几个专门用于处理数字列表的 Python 函数。例如，你可以轻松地找出数字列表的最大值、最小值和总和：

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
min(digits)
0
max(digits)
9
sum(digits)
45
```

4.3 使用列表的一部分

4.3.1 切片

要创建切片，可指定要使用的第一个元素和最后一个元素的索引。与函数 range() 一样，Python 在到达第二个索引之前的元素后停止。要输出列表中的前三个元素，需要指定索引 0 和 3，这将返回索引为 0、1 和 2 的元素。

players.py

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])

['charles', 'martina', 'michael']
```

如果没有指定第一个索引，Python 将自动从列表开头开始。如果要提取从第三个元素到列表末尾的所有元素，可将起始索引指定为 2 ，并省略终止索引：

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[2:])

['michael', 'florence', 'eli']
```

4.3.2 遍历切片

如果要遍历列表的部分元素，可在 for 循环中使用切片。下面的示例遍历前三名队员，并打印他们的名字：

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())

Here are the first three players on my team:
Charles
Martina
Michael
```

4.3.3 复制列表

我们经常需要根据既有列表创建全新的列表。下面来介绍复制列表的工作原理，以及复制列表可 供极大帮助的一种情形。

要复制列表，可创建一个包含整个列表的切片，方法是同时省略起始索引和终止索引（[:]）。这让 Python 创建一个始于第一个元素、终止于最后一个元素的切片，即整个列表的副本。

例如，假设有一个列表包含你最喜欢的四种食品，而你想再创建一个列表，并在其中包含一位朋友喜欢的所有食品。不过，你喜欢的食品，这位朋友也都喜欢，因此可通过复制来创建这个列表：

foods.py

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)

My favorite foods are:
['pizza', 'falafel', 'carrot cake']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake']
```

4.4 元组

列表非常适合用于存储在程序运行期间可能变化的数据集。列表是可以修改的，这对处理网站的用户列表或游戏中的角色列表至关重要。然而，有时候你需要创建一系列不可修改的元素，元组可以满足这种需求。Python 将不能修改的值称为不可变的，而不可变的列表被称为元组。

4.4.1 定义元组

元组看起来像列表，但使用圆括号而非中括号来标识。例如，如果有一个大小不应改变的矩形，可将其长度和宽度存储在一个元组中，从而确保它们是不能修改的：

dimensions.py

```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

笔记 下面来尝试修改元组 dimensions 的一个元素，看看结果如何：

```
dimensions = (200, 50)
dimensions[0] = 250
```

4.4.2 遍历元组中的所有值

像列表一样，也可以使用 for 循环来遍历元组中的所有值：

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

第 4 章 练习

1. 有一家自助式餐馆，只提供五种简单的食品。请想出五种简单的食品，并将其存储在一个元组中。
   - 使用一个 for 循环将该餐馆提供的五种食品都打印出来。
   - 尝试修改其中的一个元素，核实 Python 确实会拒绝你这样做。
   - 餐馆调整了菜单，替换了它提供的其中两种食品。请编写一个这样的代码块：给元组变量赋值，并使用一个 for 循环将新元组的每个元素都打印出来。

第 5 章 if 语句

内容提要

编程时经常需要检查一系列条件，并据此决定采取什么措施。在 Python 中，if 语句让你能够检查程序的当前状态，并采取相应的措施。

5.1 if 语句

理解条件测试后，就可以开始编写 if 语句了。if 语句有很多种，选择使用哪种取决于要测试的条件数。

5.1.1 简单的 if 语句

最简单的 if 语句只有一个测试和一个操作：

```
if conditional_test:
    do something
```

5.1.2 if-else 语句

我们经常需要在条件测试通过时执行一个操作，在没有通过时执行另一个操作。在这种情况下，可使用 Python 提供的 if-else 语句。if-else 语句块类似于简单的 if 语句，但其中的 else 语句让你能够指定条件测试未通过时要执行的操作。

```
age = 17
    if age >= 18:
        print("You are old enough to vote!")
        print("Have you registered to vote yet?")
    else:
        print("Sorry, you are too young to vote.")
        print("Please register to vote as soon as you turn 18!")
```

5.1.3 if-elif-else 结构

我们经常需要检查超过两个的情形，为此可使用 Python供的 ifelif-else 结构。Python 只执行 if-elif-else 结构中的一个代码块。它依次检查每个条件测试，直到遇到通过了的条件测试。测试通过后，Python 将执行紧跟在它后面的代码，并跳过余下的测试。

在现实世界中，很多情况下需要考虑的情形超过两个。例如，来看一个根据年龄段收费的游乐场：

- 4 岁以下免费；
- 418 岁收费 25 美元；
- 18 岁（含）以上收费 40 美元。

amusement_park.py

```
age = 12
if age < 4:
    print("Your admission cost is $0.")
elif age < 18:
    print("Your admission cost is $25.")
else:
    print("Your admission cost is $40.")
```

为了让代码更简洁，可不在 if-elif-else 代码块中打印门票价格，而只在其中设置门票价格，并在它后面添加一个简单的函数调用 print()：

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40
print(f"Your admission cost is ${price}.")
```

5.1.4 测试多个条件

if-elif-else 结构功能强大，但仅适合用于只有一个条件满足的情况：遇到通过了的测试后，Python 就跳过余下的测试。这种行为很好，效率很高，让你能够测试一个特定的条件。

然而，有时候必须检查你关心的所有条件。在这种情况下，应使用一系列不包含 elif 和 else 代码块的简单 if 语句。在可能有多个条件为 True 且需要在每个条件为 True 时都采取相应措施时，适合使用这种方法。

下面再来看看披萨店示例。如果顾客点了两种配料，就需要确保在其披萨中包含这些配料：

toppings.py

```
requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")
print("\nFinished making your pizza!")
```

第 5 章 练习

1. 人生的不同阶段设置变量 age 的值，再编写一个 if-elif-else 结构，根据 age 的值判断一个人处于人生的哪个阶段。
   - 如果年龄小于 2 岁，就打印一条消息，指出这个人是婴儿。

- 如果年龄为 2（含）~ 4 岁，就打印一条消息，指出这个人是幼儿。
- 如果年龄为 4（含）~ 13 岁，就打印一条消息，指出这个人是儿童。
- 如果年龄为 13（含）~ 20 岁，就打印一条消息，指出这个人是青少年。
- 如果年龄为 20（含）~ 65 岁，就打印一条消息，指出这个人是成年人。
- 如果年龄超过 65 岁（含），就打印一条消息，指出这个人是老年人。

第 6 章 字典

内容提要

在本章中，你将学习能够将相关信息关联起来的 Python 字典，以及如何访问和修改字典中的信息。字典可存储的信息量几乎不受限制，因此我们会演示如何遍历字典中的数据。另外，你还将学习存储字典的列表、存储列表的字典和存储字典的字典。

6.1 使用字典

在 Python 中，字典是一系列键值对。每个键都与一个值相关联，你可使用键来访问相关联的值。与键相关联的值可以是数、字符串、列表乃至字典。事实上，可将任何 Python 对象用作字典中的值。

在 Python 中，字典用放在花括号 {} 中的一系列键值对表示，如前面的示例所示：

```
alien_0 = {'color': 'green', 'points': 5}
```

键值对是两个相关联的值。指定键时，Python 将返回与之相关联的值。键和值之间用冒号分隔，而键值对之间用逗号分隔。在字典中，想存储多少个键值对都可以。

最简单的字典只有一个键值对，如下述修改后的字典 alien_0 所示：

```
alien_0 = {'color': 'green'}
```

这个字典只存储了一项有关 alien_0 的信息，具体地说是这个外星人的颜色。在该字典中，字符串'color' 是一个键，与之相关联的值为'green' 。

6.1.1 访问字典中的值

要获取与键相关联的值，可依次指定字典名和放在方括号内的键，如下所示：

alien.py

```
alien_0 = {'color': 'green'}
print(alien_0['color'])
```

现在，你可访问外星人 alien_0 的颜色和分数。如果玩家射杀了这个外星人，就可以使用下面的代码来确定应获得多少分：

```
alien_0 = {'color': 'green', 'points': 5}
new_points = alien_0['points']
print(f"You just earned {new_points} points!")
```

6.1.2 添加键值对

字典是一种动态结构，可随时在其中添加键值对。要添加键值对，可依次指定字典名、用方括号括起的键和相关联的值。

alien.py

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

6.1.3 先创建一个空字典

在空字典中添加键值对有时候可 供便利，而有时候必须这样做。为此，可先使用一对空花括号定义一个字典，再分行添加各个键值对。例如，下面演示了如何以这种方式创建字典 alien_0：

alien.py

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

6.1.4 修改字典中的值

要修改字典中的值，可依次指定字典名、用方括号括起的键，以及与该键相关联的新值。

alien.py

```
alien_0 = {'color': 'green'}
print(f"The alien is {alien_0['color']}.")
alien_0['color'] = 'yellow'
print(f"The alien is now {alien_0['color']}.")
```

对一个能够以不同速度移动的外星人进行位置跟踪。为此，我们将存储该外星人的当前速度，并据此确定该外星人将向右移动多远：

```
alien_0 = {'x_position': 0, 'y_position': 25, 'speed': 'medium'}
print(f"Original position: {alien_0['x_position']}")
# 向右移动外星人。
# 根据当前速度确定将外星人向右移动多远。
if alien_0['speed'] == 'slow':
x_increment = 1
elif alien_0['speed'] == 'medium':
x_increment = 2
else:
# 这个外星人的移动速度肯定很快。
x_increment = 3
```

```
# 新位置为旧位置加上移动距离。
alien_0['x_position'] = alien_0['x_position'] + x_increment
print(f"New position: {alien_0['x_position']}")
```

6.1.5 删除键值对

对于字典中不再需要的信息，可使用 del 语句将相应的键值对彻底删除。使用 del 语句时，必须指定字典名和要删除的键。

alien.py

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']
print(alien_0)
```

6.1.6 使用 get() 来访问值

使用放在方括号内的键从字典中获取感兴趣的值时，可能会引发问题：如果指定的键不存在就会出错。

```
alien_0 = {'color': 'green', 'speed': 'slow'}
print(alien_0['points'])
```

方法 get() 的第一个参数用于指定键，是必不可少的；第二个参数为指定的键不存在时要返回的值，是可选的：

```
alien_0 = {'color': 'green', 'speed': 'slow'}
point_value = alien_0.get('points', 'No point value assigned.')
print(point_value)
```

如果字典中有键'points' ，将获得与之相关联的值；如果没有，将获得指定的默认值。虽然这里没有键'points' ，但将获得一条清晰的消息，不会引发错误。

第 6 章 练习

使用一个字典来存储一个熟人的信息，包括名、姓、年龄和居住的城市。该字典应包含键 first_name 、last_name 、age 和 city 。将存储在该字典中的每项信息都打印出来。

6.2 遍历字典

个 Python 字典可能只包含几个键值对，也可能包含数百万个键值对。鉴于字典可能包含大量数据，Python 支持对字典进行遍历。字典可用于以各种方式存储信息，因此有多种遍历方式：可遍历字典的所有键值对，也可仅遍历键或值。

6.2.1 遍历所有键值对

探索各种遍历方法前，先来看一个新字典，它用于存储有关网站用户的信息。下面的字典存储一名用户的用户名、名和姓：

```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}
```

user.py

```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

for key, value in user_0.items():
    print(f"\nKey: {key}")
    print(f"Value: {value}")
```

for 语句的第二部分包含字典名和方法 items()，它返回一个键值对列表。接下来，for 循环依次将每个键值对赋给指定的两个变量。在本例中，使用这两个变量来打印每个键及其相关联的值。第一个函数调用 print() 中的"" 确保在输出每个键值对前都插入一个空行。

6.2.2 遍历字典中的所有键

在不需要使用字典中的值时，方法 keys() 很有用。下面来遍历字典 favorite_languages ，并将每个被调查者的名字都打印出来.

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
}
for name in favorite_languages.keys():
    print(name.title())
```

遍历字典时，会默认遍历所有的键。因此，如果将上述代码改为：

```
for name in favorite_languages:
```

在这种循环中，可使用当前键来访问与之相关联的值。下面来打印两条消息，指出两位朋友喜欢的语言。像前面一样遍历字典中的名字，但在名字为指定朋友的名字时，打印一条消息，指出其喜欢的语言：

```
favorite_languages = {
'jen': 'python',
```

```
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}

friends = ['phil', 'sarah']
for name in favorite_languages.keys():
print(f"Hi {name.title()}.")
if name in friends:
    language = favorite_languages[name].title()
    print(f"\t{name.title()}, I see you love {language}!")
```

还可使用方法 keys() 确定某个人是否接受了调查。下面的代码确定 Erin 是否接受了调查：

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}
if 'erin' not in favorite_languages.keys():
    print("Erin, please take our poll!")
```

方法 keys() 并非只能用于遍历：实际上，它返回一个列表，其中包含字典中的所有键。

6.2.3 按特定顺序遍历字典中的所有键

要以特定顺序返回元素，一种办法是在 for 循环中对返回的键进行排序。为此，可使用函数 sorted() 来获得按特定顺序排列的键列表的副本：

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}
for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")
```

6.2.4 遍历字典中的所有值

如果主要对字典包含的值感兴趣，可使用方法 values() 来返回一个值列表，不包含任何键。例如，假设我们想获得一个列表，其中只包含被调查者选择的各种语言，而不包含被调查者的名字，可以这样做：

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
```

```
'phil': 'python',
}
print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())
```

这种做法 取字典中所有的值，而没有考虑是否重复。涉及的值很少时，这也许不是问题，但如果被调查者很多，最终的列表可能包含大量重复项。为剔除重复项，可使用集合（set）。集合中的每个元素都必须是独一无二的。

```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}
print("The following languages have been mentioned:")
for language in set(favorite_languages.values()):
    print(language.title())
```

第 6 章 练习

1. 创建一个字典，在其中存储三条重要河流及其流经的国家。例如，一个键值对可能是'nile': 'egypt'
   - 使用循环为每条河流打印一条消息，下面是一个例子。
     The Nile runs through Egypt.
   - 使用循环将该字典中每条河流的名字打印出来。
   - 使用循环将该字典包含的每个国家的名字打印出来。

6.3 嵌套

有时候，需要将一系列字典存储在列表中，或将列表作为值存储在字典中，这称为嵌套。你可以在列表中嵌套字典、在字典中嵌套列表甚至在字典中嵌套字典。

6.3.1 字典列表

字典 alien_0 包含一个外星人的各种信息，但无法存储第二个外星人的信息，更别说屏幕上全部外星人的信息了。如何管理成群结队的外星人呢? 一种办法是创建一个外星人列表，其中每个外星人都是一个字典，包含有关该外星人的各种信息。

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}

aliens = [alien_0, alien_1, alien_2]
```

```
for alien in aliens:
    print(alien)
```

首先创建三个字典，其中每个字典都表示一个外星人。然后将这些字典都存储到一个名为 aliens 的列表中。最后，遍历这个列表，并将每个外星人都打印出来.

更符合现实的情形是，外星人不止三个，且每个外星人都是使用代码自动生成的。在下面的示例中，使用 range() 生成了 30 个外星人：

```
# 创建一个用于存储外星人的空列表。
aliens = []
# 创建30个绿色的外星人。
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
# 显示前5个外星人。
for alien in aliens[:5]:
    print(alien)
print("...")
# 显示创建了多少个外星人。
print(f"Total number of aliens: {len(aliens)}")
```

6.3.2 在字典中存储列表

有时候，需要将列表存储在字典中，而不是将字典存储在列表中。例如，你如何描述顾客点的比萨呢？如果使用列表，只能存储要添加的比萨配料；但如果使用字典，就不仅可在其中包含配料列表，还可包含其他有关比萨的描述。

pizza.py

```
# 存储所点比萨的信息。
pizza = {
'crust': 'thick',
'toppings': ['mushrooms', 'extra cheese'],
}
# 概述所点的比萨。

print(f"You ordered a {pizza['crust']}-crust pizza "
"with the following toppings:")
for topping in pizza['toppings']:
    print("\"f+topping)
```

每当需要在字典中将一个键关联到多个值时，都可以在字典中嵌套一个列表。

favorite_languages.py

```
favorite_languages = {
'jen': ['python', 'ruby'],
'sarah': ['c'],
'edward': ['ruby', 'go'],
```

```
'phil': ['python', 'haskell'],
}
for name, languages in favorite_languages.items():
    print(f"\n{name.title()}'s favorite languages are:")
    for language in languages:
        print(f"\t{language.title()}")
```

为进一步改进这个程序，可在遍历字典的 for 循环开头添加一条 if 语句，通过查看 len(languages) 的值来确定当前的被调查者喜欢的语言是否有多种。如果他喜欢的语言有多种，就像以前一样显示输出；如果只有一种，就相应修改输出的措辞，如显示 Sarah's favorite language is C 。

6.3.3 在字典中存储字典

可在字典中嵌套字典，但这样做时，代码可能很快复杂起来。例如，如果有多个网站用户，每个都有独特的用户名，可在字典中将用户名作为键，然后将每位用户的信息存储在一个字典中，并将该字典作为与用户名相关联的值。在下面的程序中，存储了每位用户的三项信息：名、姓和居住地。为访问这些信息，我们遍历所有的用户名，并访问与每个用户名相关联的信息字典：

many_users.py

```
users = {
'aeinstein': {
'first': 'albert',
'last': 'einstein',
'location': 'princeton',
},
'mcurie': {
'first': 'marie',
'last': 'curie',
'location': 'paris',
},
}
for username, user_info in users.items():
    print(f"\nUsername: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

请注意，表示每位用户的字典都具有相同的结构。虽然 Python 并没有这样的要求，但这使得嵌套的字典处理起来更容易。倘若表示每位用户的字典都包含不同的键，for 循环内部的代码将更复杂。

第 6 章 练习

1. 创建多个表示宠物的字典，每个字典都包含宠物的类型及其主人的名字。将这些字典存储在一个名为 pets 的列表中，再遍历该列表，并将有关每个宠物的所有信息都打印出来。

2. 创建一个名为 cities 的字典，将三个城市名用作键。对于每座城市，都创建一个字典，并在其中包含该城市所属的国家、人口约数以及一个有关该城市的事实。在表示每座城市的字典中，应包含 country 、population 和 fact 等键。将每座城市的名字以及有关信息都打印出来。