TURING 图灵程序设计丛书

PYTHON CRASH COURSE, 3RD EDITION

A HANDS-ON, PROJECT-BASED INTRODUCTION TO PROGRAMMING

Python编程

从入门到实践

（第3版）

[美] 埃里克·马瑟斯（Eric Matthes）著

袁国忠 译　陶俊杰 审

人民邮电出版社

北　京

图书在版编目（CIP）数据

Python编程 : 从入门到实践 / (美) 埃里克·马瑟斯 (Eric Matthes) 著 ; 袁国忠译. -- 3版. -- 北京 : 人民邮电出版社, 2023.5
（图灵程序设计丛书）
ISBN 978-7-115-61363-9

Ⅰ. ①P… Ⅱ. ①埃… ②袁… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国国家版本馆CIP数据核字(2023)第042770号

内 容 提 要

本书是针对所有层次的 Python 读者而作的 Python 入门书。全书分为两部分：第一部分介绍使用 Python 编程所必须了解的基本概念，包括强大的 Python 库和工具，以及列表、字典、if 语句、类、文件和异常、测试代码等内容；第二部分将理论付诸实践，讲解如何开发三个项目，包括简单的 2D 游戏、利用数据生成交互式的信息图以及创建和定制简单的 Web 应用程序，并帮助读者解决常见编程问题和困惑。第 3 版进行了全面修订：使用了文本编辑器 VS Code，新增了介绍 `removeprefix()` 方法和 `removesuffix()` 方法的内容，并且在项目中利用了 Matplotlib 和 Plotly 的最新特性，等等。

本书适合对 Python 感兴趣的所有读者阅读。

◆ 著　[美] 埃里克·马瑟斯（Eric Matthes）
译　袁国忠
审　陶俊杰
责任编辑　岳新欣
责任印制　胡　南

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　https://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：29.75　2023年 5 月第 1 版
字数：703千字　2023年 5 月北京第 1 次印刷
著作权合同登记号　图字：01-2022-5039号

定价：109.80元

读者服务热线：(010)84084456-6009　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东市监广登字 20170147 号

版 权 声 明

对本书前两版的赞誉

“No Starch Press 革故鼎新，不断推出堪与传统编程图书比肩的未来经典，而本书就是其中之一。”

——Greg Laden，ScienceBlogs

“对复杂的项目娓娓道来，逻辑合理、赏心悦目，令人欲罢不能。”

——*Full Circle* 杂志

“清晰地阐述代码片段，引领你每次前进一小步，逐步编写出复杂的代码，并对其中的原理了如指掌。”

——FlickThrough Reviews

“美妙的 Python 学习体验，Python 新手的不二选择。”

——Mikke Goes Coding

“名副其实，出色地完成了引领读者从入门到实践的任务。三个项目富有挑战性又寓教于乐，还有大量极具帮助的练习题。”

——RealPython

“简明而全面的 Python 编程入门读物，助你最终掌握 Python，是一本值得拥有的杰出作品。”

——TutorialEdge

“编程小白的明智之选。化繁为简，一步一个脚印地带领你进入 Python 这门深奥语言的殿堂。”

——WhatPixel

“面面俱到，初学者需要知道的 Python 知识应有尽有。”

——FireBearStudio

“主要介绍如何使用 Python 编写代码，同时讲授了编写整洁代码的技巧，可应用于其他大部分语言。”

——Great Lakes Geek

推　荐　语

编程教学之道，一是重在实践，二是循序渐进——通过巧妙的实战项目，激发和保持学习的热情，让学习渐入佳境。在这两方面，这本书无疑都是非常出色的。无论是初次尝试编程，还是打算拥抱人工智能，相信这本书都会成为你的最佳起点。

——爱可可-爱生活，北京邮电大学副教授陈光老师

很高兴看到这本书的第 3 版更新，这是一本实操性很强的 Python 语言零基础入门和起步教材。它最大的特色在于，在为初学者构建完整的 Python 语言知识体系的同时，面向实际应用情境编写代码样例，而且许多样例还是后续实践项目部分的伏笔。实践项目部分的选题经过精心设计，生动详尽又面面俱到。相信这本书能够得到更多 Python 初学者的喜爱。

——陈斌，北京大学地球与空间科学学院教授、北京市高等学校教学名师

这本书的前两版已经广销全球，而且稳居 Python 图书的各大销量榜首，这足以证明它的内容有多么出色！这本书简明又全面地阐述了入门 Python 需要掌握的各方面知识，可以说是学习 Python 的不二选择。

——崔庆才，《Python 3 网络爬虫开发实战》作者、微软（中国）软件工程师

Python Crash Course 从 2016 年出版，到现在刚刚 7 年就已经增补到了第 3 版，可见作者是认真的、市场是认可的、内容是靠谱的。关键是，这本书在图灵的 Python 技术图书中的核心地位难以撼动。为什么呢？因为其他入门书没这本全面，而其他专业领域图书又没这本好读，它基本上可以作为将其他所有 Python 技术图书串联起来的总线。这本书唯一的缺点可能就是太厚，读者怕读不完。其实不必，第一部分看过后，其余内容就可以当成工程辞典，有需要时查阅即可。注意原书副标题“*A Hands-On, Project-Based Introduction to Programming*”点出了关键：这是以一个个小项目为线索来阐述如何用 Python 进行具体编程的书。它的每一个版本都紧跟 Python 的进步而增补，值得收藏。

——大妈，CPyUG 联合创始人、蟒营®创始人

本书注重用户体验，列举了大量易于理解的例子和各种练习来帮助读者掌握 Python，非常适合初学者以及有一定编程经验的人学习 Python。

——廖雪峰，知名技术专家

从这本书第 1 版起，我就开始把它推荐给身边正在学 Python 的朋友，因为作为 Python 入门的第一本书，它对初学者非常友好。如今这本书已经更新到了第 3 版，内容与时俱进且更加精练，现在依然是最好的 Python 入门读物之一。

——刘志军，公众号“Python 之禅”主理人

说实话，这本书可能不太需要那么多推荐。近十年来，这本书引导着包括我在内的无数 Python 开发者进入了 Python 世界。在我心中，放眼全球，它在 Python 入门书中应该是“天花板”般的存在。而更令人惊喜的是，中文版的翻译水平也是引进图书的“天花板”。因此，请放心，这本书一定能将你带入令人陶醉的 Python 开发世界！

——Manjusaka，PyCon China 负责人、Python 播客“捕蛇者说”联合创始人、微软 MVP

这是一本让你轻松掌握 Python 的绝佳教材。这本书用简练的文字阐述 Python 知识，已成为百万读者信赖的“编程圣经”。它包含三个实战项目：《外星人入侵》游戏、数据可视化、Web 开发，方便读者迅速学以致用。渴望学习 Python 的朋友，这本书无疑是你不容错过的入门必读之作！

——彭涛，“涛哥聊 Python”博主、字码网络科技创始人

编程语言很快就要成为大家母语之外应该掌握的第二语言，而 Python 是学习编程的优选语言。这本书内容循序渐进、基础与实战相结合，非常适合 Python 初学者，是新手入门的最佳选择。

——豌豆花下猫，自媒体“Python 猫”主理人

如果你想学习 Python 编程，那么这是一本非常适合初学者和有经验的程序员的入门书。本书将 Python 编程的基本概念和相关工具讲解得深入浅出，通过三个实际项目的开发帮助读者更好地理解如何应用所学的概念和技巧，同时解决实际编程中遇到的问题和困惑。第 3 版进行了全面修订，采用了流行的编程软件，并且新增了一些内容。这是一本值得放在你桌上的书。

——翁恺，浙江大学计算机学院教授

最理想的新人入门书应该满足两个特点：第一就是内容通俗易懂；第二就是要有实战，能够让读者在学完之后知道具体怎么用。这本书刚好满足了这两点，而且销量也是一个很好的证明。不管你是要入门还是精进 Python，都建议你读一读这本经典著作。

——张俊红，《对比 Excel，轻松学习 Python 数据分析》作者

“蟒蛇书”是我最常向朋友们推荐的 Python 入门书之一。全书内容安排合理，既有通俗易懂的技术概念讲解，又包含大量有趣的项目实战，可谓面面俱到，尤其适合初学者。

——朱雷（@piglei），腾讯公司高级工程师、《Python 工匠：案例、技巧与工程实践》作者

这本书是我的 Python 启蒙老师，它不仅有详细的语法讲解，还配有大量项目案例，第 3 版的实践项目更加丰富。所谓“输出是最好的输入”，通过 Python 来实现数据分析、数据可视化、Web 开发等，帮助自己提升工作效率，是非常有成就感的事。

——朱卫军，公众号“Python 大数据分析”主理人

谨以此书献给我的父亲和儿子。

感谢父亲抽出时间回答我提出的每个编程问题，感谢儿子 Ever 开始向我提问了。

第 3 版修订说明

本书前两版出版后反响强烈，被翻译成了 12 种语言，仅中文版销量就超过了 100 万册。我收到了众多读者的来信和电子邮件，有小到 10 岁的孩童，还有利用闲暇学习编程的退休人员。有一些初中、高中和大学将本书作为教材，有使用高级教材的学生将其作为补充材料，还有人通过阅读它来提高工作技能或者开发自己的项目。总而言之，本书的用途之广远远超出了我最初的预期。

第 3 版的编写过程从始至终令人愉悦。Python 虽是一门成熟的语言，但也像其他语言一样在不断发展。我对本书的主要修订目标依然是确保精练、简单易懂。本书能让读者具备动手开发项目所需的一切知识，同时为进一步学习打下坚实的基础。为此，我修订了部分章节，以反映如何利用 Python 中的新方式更简单地完成任务，还澄清了对 Python 语言的某些细节描述得不太准确的地方。所有的项目都做了全面修订，采用得到良好维护的流行库，让你能够充满信心地用它们来开发自己的项目。

下面概述第 3 版的具体修订。

- 第 1 章推荐使用文本编辑器 VS Code（Visual Studio Code），它深受初学者和专业程序员的欢迎，适用于各种操作系统。
- 第 2 章新增了介绍 `removeprefix()`方法和 `removesuffix()`方法的内容，这两个方法可以在处理文件和 URL 时提供极大的帮助。这一章还介绍了改进后的 Python 错误消息，它们提供了非常具体的信息，有助于找出并修复代码中的错误。
- 第 10 章改用模块 `pathlib` 来处理文件，这是一种更加简单的文件读写方法。
- 第 11 章改用 `pytest` 来为代码编写自动化测试。`pytest` 库目前已成为编写 Python 测试的行业标准工具，能让初学者轻松地编写测试。如果你的目标是成为 Python 程序员，将来也会在职业生涯中用到它。
- 第 12 ~ 14 章的“外星人入侵”项目新增了控制帧率的设置，让这款游戏在不同操作系统中的运行情况更加一致。我还使用了更简单的方法来创建外星舰队，同时让整个项目的结构更简洁。
- 第 15 ~ 17 章的可视化项目利用了 Matplotlib 和 Plotly 的最新特性。对于 Matplotlib 可视化项目，我更新了样式设置。在随机游走项目中，我通过细微的改进提高了图表的准确度，让新生成的随机游走呈现更多不同的模式。在所有的 Plotly 可视化项目中，使用的都是模

块 Plotly Express，让你只需编写几行代码就能够生成初始的可视化形式。这样，你可以轻松地探索各种图表，并从中选择最合适的，再专注于改进其中的各个元素。

- 第 18 ~ 20 章使用最新版的 Django 创建“学习笔记”项目，并使用最新版的 Bootstrap 设置样式。我重命名了该项目的一些部分，让你能够更轻松地明白该项目的总体组织结构。另外，我将这个项目部署到了 Platform.sh 上，这是一个新兴的 Django 项目托管服务。部署过程由 YAML 配置文件控制，让你对如何部署项目有更大的控制权。这种做法与专业程序员部署现代 Django 项目的方式是一致的。
- 附录 A 做了全面修订，推荐你采用 Python 在主流操作系统中的最佳安装方法。附录 B 提供了详尽的 VS Code 安装说明，并简要介绍了大部分主流文本编辑器和 IDE。附录 C 引导你访问更新、更流行的在线资源以寻求帮助。附录 D 提供了 Git 版本控制的简明教程。附录 E 是新增的。即便本书对如何部署应用程序做了详尽的说明，你也可能在很多地方遇到问题。因此，附录 E 提供了详尽的故障排除指南，以便你在部署过程中遇到问题时参考。

感谢购买本书，如果有任何反馈或问题，请务必通过 Twitter（@ehmatthes）与我联系。

中文版审读致谢

对于一本过往影响了无数读者，未来还要继续影响更多读者的图书，第 3 版邀请了陶俊杰担纲审校，并公开招募业内 20 位专家进行审读，力求品质更上一层楼。感谢各位专家对译文提出了大量宝贵建议，感谢 Manjusaka、蔡琛承担了更多章节的审读，感谢陶叶港（@Scruel）、姜子龙承担了审读之后的统筹工作。专家姓名列在了下表中（按姓氏字母排序）。

审读章号	审读专家
第 1 ~ 4 章	蔡琛、陈骜、张鑫明
第 5 ~ 8 章	蔡琛、陈栋、江志强、金圣凯
第 9 ~ 11 章	姜子龙、Manjusaka、朱雷
第 12 ~ 14 章	艾凌风、陈翔（@翔翔的学习频道）、陶叶港（@Scruel）
第 15 ~ 17 章	胡屹、柳佳龙、卢震、杨双龙
第 18 ~ 20 章	大妈、Manjusaka
附录 A ~ E	陈少辉、Kyle C、周鹤龄

前　言

如何学习编写第一个程序，每个程序员都有不同的故事。我在还是个孩子时就开始学习编程了，当时我父亲在计算时代的先锋之一——数字设备公司（Digital Equipment Corporation）工作。我使用一台简陋的计算机编写了第一个程序，这台计算机是父亲在家里的地下室组装而成的，它没有机箱，裸露的主板与键盘相连，显示器是裸露的阴极射线管。我编写的这个程序是一款简单的猜数字游戏，其输出类似于下面这样：

```
I'm thinking of a number! Try to guess the number I'm thinking of: 25
Too low! Guess again: 50
Too high! Guess again: 42
That's it! Would you like to play again? (yes/no) no
Thanks for playing!
```

看到家人玩着我编写的游戏，而且它完全按我预期的方式运行，我心里不知有多满足。此情此景我永远也忘不了。

儿童时期的这种体验一直影响我至今。现在，每当我通过编写程序解决了一个问题时，心里都会感到非常满足。相比于年少时，我现在编写的软件满足了更大的需求，但通过编写程序获得的满足感几乎与从前一样。

读者对象

本书旨在让你尽快学会 Python，以便编写出能正确运行的程序——游戏、数据可视化和 Web 应用程序，同时掌握让你终身受益的基本编程知识。本书适合任何年龄的读者阅读，它不要求你有 Python 编程经验，甚至不要求你有编程经验。如果你想快速掌握基本的编程知识以便专注于开发感兴趣的项目，并想通过解决有意义的问题来检查你对新概念的理解程度，那么本书就是为你编写的。本书可供 Python 教师通过开发项目向学生介绍编程。如果你是刚开始学习 Python 的大学生，觉得指定的教材不那么容易理解，那么阅读本书将让学习过程变得更轻松。如果你想

转行当程序员，本书可帮助你走上更满意的职业道路。总而言之，本书适合目标各异的各类读者阅读。

本书内容

本书旨在让你成为优秀的程序员，具体地说，是优秀的 Python 程序员。通过阅读本书，你将迅速掌握编程概念，打下坚实的基础，并养成良好的习惯。阅读本书后，你就可以开始学习 Python 高级技术，并能够更轻松地掌握其他编程语言。

在本书的第一部分，你将学习编写 Python 程序时需要熟悉的基本编程概念，你在刚接触几乎任何编程语言时都需要学习这些概念。你将学习各种数据以及在程序中存储数据的方式。你将学习如何创建数据集合（如列表和字典），以及如何高效地遍历它们。你将学习使用 `while` 循环和 `if` 语句来检查条件，并在满足条件时执行代码的一部分，而在不满足条件时执行代码的另一部分——这可为流程自动化提供极大的帮助。

你将学习获取用户输入，让程序能够与用户交互，并在用户没停止输入时保持运行状态。你将探索如何编写函数来让程序的各个部分可复用，这样在编写好执行特定任务的代码后，可以无限制地多次使用。然后，你将学习使用类来扩展这种概念以实现更复杂的行为，从而让非常简单的程序也能处理各种不同的情形。你将学习编写能妥善处理常见错误的程序。学习这些基本概念后，你将使用学到的知识编写大量越来越复杂的程序。最后，你将向中级编程迈出第一步，学习如何为代码编写测试，以便在进一步改进程序时免于担心可能引入 bug。第一部分介绍的知识让你能够开发更大、更复杂的项目。

在第二部分，你将利用在第一部分学到的知识来开发三个项目。你既可以根据自己的情况，以最合适的顺序完成这些项目，也可以选择只完成其中的某个项目。在第一个项目（第 12 ~ 14 章）中，你将创建一个类似于《太空入侵者》的射击游戏，这个游戏名为《外星人入侵》，包含多个难度不断增加的等级。完成这个项目后，你就完全能够自己动手开发 2D 游戏了。就算你无意成为游戏程序员，也应该完成这个项目，因为它以寓教于乐的方式综合应用了第一部分介绍的很多知识点。

第二个项目（第 15 ~ 17 章）介绍数据可视化。数据科学家的目标是通过各种可视化技术来理解海量信息。你将使用通过代码生成的数据集、已经从网络下载下来的数据集以及程序自动下载的数据集。完成这个项目后，你将能编写出对大型数据集进行筛选的程序，并以可视化方式将各种数据呈现出来。

在第三个项目（第 18 ~ 20 章）中，你将创建一个名为“学习笔记”的小型 Web 应用程序。这个项目能够让用户将学到的与特定主题相关的知识记录下来。你将能够分别记录不同的主题，还可让其他人建立账户并开始记录自己的学习笔记。你还将学习如何部署这个项目，让任何人都能够通过网络访问它，而不管他身处何方。

在线资源

要获取以下及更多补充材料，可访问 iturin g.cn/book/3038。

- **安装说明**：与书中的安装说明相同，在遇到安装问题时，可参阅这些材料。
- **更新**：与其他编程语言一样，Python 也是在不断发展变化的。我提供了详尽的更新记录，每当遇到问题时，你都可参阅它看看是否需要调整操作。
- **练习答案**：你应该花大量时间独立完成“动手试一试”中的练习。如果卡壳了，无法独立完成，可查看部分练习的答案。
- **数据下载方法**：在完成第 16 章中的项目和部分练习时，需要额外下载一些数据集。可参阅这些材料中的网址和步骤进行下载。
- **速查表**：这一整套速查表可作为主要概念的参考指南。

为何使用 Python

继续使用 Python，还是转而使用其他语言——也许是编程领域里较新的语言？我每年都会考虑这个问题。可我依然专注于 Python，其中的原因很多。Python 是一种效率极高的语言：相比于众多其他语言，使用 Python 编写的程序包含的代码行更少。Python 的语法也有助于创建整洁的代码：相比于使用其他语言，使用 Python 编写的代码更容易阅读、调试和扩展。

大家将 Python 用于众多方面：编写游戏、创建 Web 应用程序、解决商业问题以及开发内部工具。Python 还在科学领域被大量用于学术研究和应用研究。

我坚持使用 Python 的一个最重要的原因是，Python 社区有形形色色充满激情的人。对程序员来说，社区非常重要，因为编程绝非孤独的修行。大多数程序员需要向解决过类似问题的人寻求建议，经验最为丰富的程序员也不例外。当需要他人帮助解决问题时，有一个联系紧密、互帮互助的社区至关重要，而对于将 Python 作为第一门编程语言的人或从其他语言转向 Python 的人来说，Python 社区无疑是坚强的后盾。

Python 是一门出色的语言，值得你去学习。现在就开始吧！

配套视频

扫码观看随书配套视频。

致　　谢

如果没有 No Starch Press 出色的专业人士的帮助，本书根本不可能付梓。是 Bill Pollock 邀请我编写这样一本入门书，深深感谢他给予我这样的机会。Liz Chadwick 参与了本书全部三版的出版工作，正是她持之以恒的投入让本书越来越好。Eva Morrow 从全新的角度审视本书，用真知灼见改善了新版本的质量。感谢 Doug McNair 就如何妥善地使用语法提供指导，避免了本书学究气过浓。感谢 Jennifer Kepler 监督整体的制作流程，将众多文件变成散发淡淡墨香的书本。

在 No Starch Press 里，还有很多人为本书的成功付出了心血，但是我没有机会与他们打交道。No Starch Press 拥有出色的市场营销团队，他们的工作不仅是卖书，还包括帮助读者找到适合自己的书，进而达成学习目标。No Starch Press 还拥有强大的版权销售团队，他们的勤奋工作让本书得以被翻译成众多不同的语言，供全球各地的读者阅读。感谢我没有见过面，却帮助我将本书送到读者手里的所有人。

感谢 Kenneth Love 对全部三版所做的技术审阅工作。我与 Kenneth 相识于一次 PyCon 年度大会，他对 Python 和 Python 社区充满热情，一直是我获取专业灵感的源泉。Kenneth 不仅检查了本书介绍的知识是否正确，在审阅中还始终抱着这样一个目的：让编程初学者对 Python 语言和编程获得扎实的认识。此外，他还帮我找到了之前版本中有改进空间之处，让我有机会重写这些内容。不过，倘若本书有任何不准确的地方，责任完全在我。

感谢所有分享本书阅读经验的读者。学习基本编程知识可能改变世界观，而这可能给人带来深远的影响。听到这些故事让人深感谦卑，感谢各位读者如此开诚布公地分享阅读体验。

感谢父亲在我很小的时候就教我编程，一点儿也不担心我破坏他的设备。感谢妻子 Erin 在我编写本书期间一如既往的鼓励和支持。还要感谢儿子 Ever，他的好奇心每天都会给我带来灵感。

目　录

第一部分　基础知识

第二部分　项　目

Part 1

第一部分

基础知识

本书的第一部分介绍编写 Python 程序所需要熟悉的基本概念，其中很多适用于所有编程语言，因此它们在你的整个程序员生涯中都很有用。

第 1 章介绍如何在计算机中安装 Python，并运行第一个程序——在屏幕上打印消息“Hello world!”。

第 2 章论述如何将信息赋给变量以及如何使用文本和数值。

第 3 章和第 4 章介绍列表。列表让你能够在一个地方存储任意数量的信息，从而高效地处理这些数据：只需几行代码，你就能够处理数百、数千乃至数百万个值。

第 5 章讲解如何使用 `if` 语句来编写这样的代码：在满足特定条件时采取一种措施，而在不满足该条件时采取另一种措施。

第 6 章演示如何使用 Python 字典，将不同的信息关联起来。与列表一样，你也可以根据需要在字典中存储任意数量的信息。

第 7 章讲解如何从用户那里获取输入，让程序变成交互式的。你还将学习 `while` 循环，它重复地运行代码块，直到指定的条件不再满足为止。

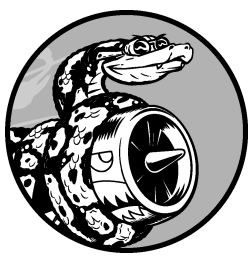
第 8 章介绍如何编写函数。函数是执行特定任务的具名代码块，你可以根据需要随时运行它。

第 9 章介绍类，它能够让你模拟实物。你将编写代码来表示小狗、小猫、人、火箭等。

第 10 章介绍如何使用文件，以及如何处理错误以免程序意外崩溃。你将在程序关闭前保存数据，并在程序再次运行时读取它们。你将学习 Python 异常，以便未雨绸缪，让程序妥善地处理错误。

第 11 章讲解如何为代码编写测试，以核实程序是否像你期望的那样工作。这样，在扩展程序时，就不用担心引入新 bug。要想脱离初级程序员，跻身于中级程序员的行列，测试代码是你必须掌握的基本技能之一。

第 1 章 起　步



在本章中，你将运行自己的第一个程序——hello_world.py。为此，你需要检查自己的计算机是否安装了较新版本的 Python；如果没有，就要进行安装。你还将安装一个用于编写和运行 Python 程序的文本编辑器。当你输入 Python 代码时，这个文本编辑器能够识别它们并高亮不同的部分，让你能够轻松地了解代码的结构。

1.1　编程环境简介

在不同的操作系统中，Python 存在细微的差别，因此有一些要点你需要牢记在心。本节将确保在你的系统上正确地安装 Python。

1.1.1　Python 版本

每种编程语言都会随着新概念和新技术的推出而不断发展，Python 开发者也一直致力于丰富和强化其功能。本书编写期间的最新版本为 Python 3.11，但只要你安装了 Python 3.9 或更高的版本，就能运行本书的所有代码。[①]在本节中，你将确认系统上是否安装了 Python，以及是否需要安装更新的版本。附录 A 提供了详尽的指南，指导你如何在各种主流操作系统中安装最新版本的 Python。

1.1.2　运行 Python 代码片段

Python 自带在终端窗口中运行的解释器，让你无须保存并运行整个程序就能尝试运行 Python 代码片段。

本书都将以如下方式列出代码片段：

① 本书示例代码所用的 Python 版本为 3.11，但在内容上不涉及 3.9 以上版本的新特性。Python 一直致力于优化错误提示信息，你在运行代码过程中得到的错误提示信息可能与书中提供的不同，这不会影响你的学习，无须担心。对新特性感兴趣的读者，请阅读《流畅的 Python（第 2 版・上下册）》。——编者注

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
```

提示符>>>表明正在使用终端窗口，而加粗的文本表示需要你输入并按回车键来执行的代码。本书的大多数示例是独立的小程序，将在编辑器中执行，因为大多数代码就是这样编写出来的。然而，为了高效地演示一些基本概念，还会在 Python 终端会话中执行一系列代码片段。只要代码清单中包含三个右尖括号，就意味着代码是在终端会话中执行的，而输出也来自终端会话。稍后将演示如何在 Python 解释器中编写代码。

此外，你还将安装一款文本编辑器，并使用它来完成学习编程的标准操作——编写一个简单的 Hello World 程序。长期以来，编程界都认为在刚接触一门新语言时，首先使用它来编写一个在屏幕上显示消息“Hello world!”的程序将带来好运。这种程序虽然简单，却有其用途：如果它能够在你的系统上正确地运行，那么你编写的任何 Python 程序也都将正确运行。

1.1.3 编辑器 VS Code 简介

VS Code 是一款功能强大的专业级文本编辑器，免费且适合初学者使用。无论是简单还是复杂的项目，使用 VS Code 来开发都是非常不错的选择。因此，在学习 Python 的过程中熟练掌握 VS Code 后，还可以继续使用它来编写复杂的大型项目。无论你使用的是哪种现代操作系统，都可安装 VS Code，它支持包含 Python 在内的大多数编程语言。

附录 B 介绍了其他几种文本编辑器，如果你想知道还有哪些编辑器可用，现在就应读一读。如果你想马上动手编程，可先使用 VS Code，等有了一些编程经验后再考虑使用其他编辑器。本章稍后将引导你在当前使用的操作系统中安装 VS Code。

注意：如果你已经安装了其他文本编辑器，并且知道如何通过配置使其自动运行 Python 程序，也可使用其他编辑器。

1.2 在各种操作系统中搭建 Python 编程环境

Python 是一种跨平台的编程语言，这意味着它能够在所有主流操作系统中运行。在所有安装了 Python 的现代计算机上，都能够运行你编写的任何 Python 程序。然而，在不同的操作系统中，安装 Python 的方法存在细微的差别。

在本节中，你将学习如何在自己的系统中安装 Python。首先检查系统是否安装了较新的 Python 版本，如果没有就进行安装，然后安装 VS Code。在各种操作系统中搭建 Python 编程环境时，只有这两步存在差别。

在接下来的两节中，你将运行程序 Hello World，并排除各种故障。我将详细介绍如何在各种操作系统中完成这些任务，让你能够搭建出一个可靠的 Python 编程环境。

1.2.1　在 Windows 系统中搭建 Python 编程环境

Windows 系统通常没有默认安装 Python，因此你可能需要安装它，再安装 VS Code。

1. 安装 Python

首先，检查你的系统是否安装了 Python。在“开始”菜单的搜索框中输入“命令”并按回车键，再单击程序“命令提示符”打开一个命令窗口。在终端窗口中输入 python（全部小写）并按回车键。如果出现 Python 提示符（>>>），就说明系统安装了 Python；如果出现一条错误消息，指出 python 是无法识别的命令，就说明没有安装 Python；如果系统自动启动了 Microsoft Store，也说明没有安装 Python，此时请关闭 Microsoft Store，因为相比于使用 Microsoft 提供的 Python 版本，下载官方安装程序是更好的选择。

如果没有安装 Python 或安装的版本低于 3.9，就需要下载 Windows Python 安装程序。为此，请访问 Python 官方网站主页。将鼠标指向链接 Downloads，你将看到一个用于下载 Python 最新版本的按钮。单击这个按钮，就会根据你的系统自动下载正确的安装程序。下载安装程序后，运行它。请务必选中复选框 Add Python ... to PATH（如图 1-1 所示），这让你能够更轻松地配置系统。

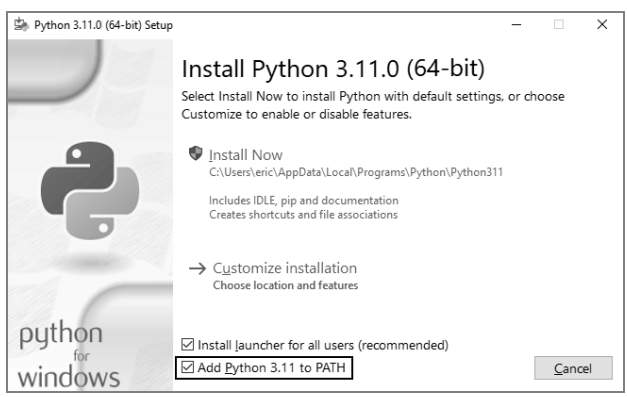


图 1-1　务必选中复选框 Add Python ... to PATH

2. 在终端会话中运行 Python

打开一个命令窗口，并在其中执行命令 python。如果出现了 Python 提示符（>>>），就说明 Windows 找到了你刚安装的 Python 版本。

```
C:\> python
Python 3.x.x (main, Jun . . . , 13:29:14) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

注意：如果没有看到类似的输出，请参阅附录 A，其中有更详尽的安装说明。

在 Python 会话中执行下面的命令：

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

应该会出现输出 `Hello Python interpreter!`。每当要运行 Python 代码片段时，都请打开一个命令窗口并启动 Python 终端会话。要关闭该终端会话，可先按 Ctrl + Z 再按回车键，也可执行命令 `exit()`。

3. 安装 VS Code

要下载 VS Code 安装程序，可访问 Visual Studio Code 官方网站主页，单击按钮 Download for Windows 下载安装程序，再运行它。然后跳到 1.3 节，并按那里的说明继续。

1.2.2 在 macOS 系统中搭建 Python 编程环境

最新的 macOS 版本默认不安装 Python，因此需要你自行安装。在本节中，你将安装最新的 Python 版本，再安装 VS Code 并确保其配置正确无误。

注意：较旧的 macOS 版本默认安装了 Python 2，但你应使用较新的 Python 版本。

1. 检查是否安装了 Python 3

在文件夹 Applications/Utilities 中，选择 Terminal，打开一个终端窗口；也可以按 Command + 空格键，再输入 `terminal` 并按回车键。为确定是否安装了较新的 Python 版本，请执行命令 `python3`。很可能会出现一个消息框，询问你是否要安装命令行开发者工具。最好先安装 Python，再安装这些工具，因此请关闭该消息框。

如果输出表明已经安装了 Python 3.9 或更高的版本，可跳过下一小节，直接阅读“在终端会话中运行 Python 代码”。如果安装的是 Python 3.9 之前的版本，请按下一小节的说明安装最新的版本。

请注意，如果你使用的是 macOS，请将本书中所有的命令 `python` 都替换为 `python3`，以确保你使用的是 Python 3。在大多数 macOS 系统中，命令 `python` 要么指向供内部系统工具使用的过期 Python 版本，要么没有指向任何程序（在这种情况下，执行命令 `python` 将引发错误）。

2. 安装最新的 Python 版本

要下载 Python 安装程序，可访问 Python 官方网站主页。将鼠标指向链接 Downloads，将出现一个用于下载最新版本 Python 的按钮。单击这个按钮，就会根据你的系统自动下载正确的安装程序。下载安装程序后运行它。

运行安装程序后，将出现一个 Finder 窗口。双击其中的文件 Install Certificates.command，运行它能让你在开发实际项目（包括本书第二部分中的项目）时更轻松地安装所需的额外库。

3. 在终端会话中运行 Python 代码

现在可以尝试运行 Python 代码片段了。为此，需要先打开一个终端窗口并执行命令 python3：

```
$ python3
Python 3.x.x (v3.11.0:eb0004c271, Jun . . . , 10:03:01)
[Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

这个命令会启动 Python 终端会话。应该会出现 Python 提示符（>>>），这意味着 macOS 找到了你刚安装的 Python 版本。

请在终端会话中输入如下代码行并按回车键：

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

应该会出现消息“Hello Python interpreter!”，它被直接打印到了当前终端窗口中。要关闭 Python 解释器，可按 Ctrl + D 或执行命令 exit()。

注意：在较新的 macOS 系统中，终端提示符为百分号（%），而不是美元符号（$）。

4. 安装 VS Code

要安装编辑器 VS Code，需要下载安装程序。为此，可访问 Visual Studio Code 官方网站主页，并单击链接 Download。然后，打开 Finder 窗口并切换到文件夹 Downloads，将其中的安装程序 Visual Studio Code 拖到文件夹 Applications 中，再双击这个安装程序以运行它。安装 VS Code 后，可跳过 1.2.3 节，直接阅读 1.3 节并按其中的说明继续。

1.2.3　在 Linux 系统中搭建 Python 编程环境

Linux 系统是为编程而设计的，因此大多数 Linux 计算机默认安装了 Python。编写和维护 Linux 的人认为，你肯定会使用该系统进行编程，他们也鼓励你这样做。因此，要在这种系统中编程，几乎不用安装什么软件，只需要修改一些设置。

1. 检查 Python 版本

在你的系统中运行应用程序 Terminal（如果你使用的是 Ubuntu，可按 Ctrl + Alt + T），打开一个终端窗口。为确定安装的是哪个 Python 版本，请执行命令 python3（请注意，其中的 p 是小

写的）。如果安装了 Python，这个命令将启动 Python 解释器。输出指出了安装的 Python 版本，还将显示 Python 提示符（>>>），让你能够输入 Python 命令。

```
$ python3
Python 3.10.4 (main, Apr  . . . , 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

上述输出表明，当前计算机默认使用的版本为 Python 3.10.4。看到上述输出后，如果要退出 Python 并返回终端窗口，可按 Ctrl + D 或执行命令 `exit()`。请将本书中的命令 `python` 都替换为 `python3`。

要运行本书的代码，必须使用 Python 3.9 或更高的版本。如果你的系统中安装的版本低于 Python 3.9，请参阅附录 A，了解如何安装最新版。

2. 在终端会话中运行 Python 代码

现在可打开终端窗口并执行命令 `python3`，再尝试运行 Python 代码片段。在检查 Python 版本时，你就这样做过。下面再次这样做，然后在终端会话中输入如下代码并按回车：

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

消息将直接打印到当前终端窗口中。别忘了，要关闭 Python 解释器，可按 Ctrl + D 或执行命令 `exit()`。

3. 安装 VS Code

在 Ubuntu Linux 系统中，可通过 Ubuntu Software Center 来安装 VS Code。为此，单击菜单中的 Ubuntu Software 图标并查找 vscode。在查找结果中，单击应用程序 Visual Studio Code（有时称为 code），再单击 Install 按钮。安装完毕后，在系统中查找 VS Code 并启动它即可。

1.3 运行 Hello World 程序

安装较新版本的 Python 和 VS Code 后，就可以编写并运行你的第一个 Python 程序了。但是在编写程序之前，还需要给 VS Code 安装 Python 扩展。

1.3.1 给 VS Code 安装 Python 扩展

虽然 VS Code 支持很多种不同的编程语言，但要让它给 Python 程序员提供尽可能多的帮助，必须安装 Python 扩展。这个扩展让你能够在 VS Code 中编写、编辑和运行 Python 程序。

要安装 Python 扩展，可单击 VS Code 应用程序窗口右下角的 Manage 图标，它看起来像个齿轮。在出现的菜单中，单击 Extensions，在搜索框中输入 python，再单击 Python extension（如果出现多个名为 Python 的扩展，请选择 Microsoft 提供的扩展）。单击 Install，安装所有需要的额外工具。如果出现一条消息提示需要安装 Python，而你已经安装了，可忽略这条消息。

注意：在 macOS 系统中，如果出现一个弹出框，询问你是否要安装命令行开发者工具，请单击 Install。可能会出现一条消息，指出需要很长时间才能安装完成，但只要你的网络连接不是很慢，实际上只需要不到 20 分钟。

1.3.2　运行程序 hello_world.py

在编写第一个程序前，在桌面上创建一个名为 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英文字母，并使用下划线代替空格，因为 Python 采用了这些命名约定。虽然完全可以将文件夹 python_work 放在其他地方，但将它放在桌面上可更轻松地完成后续步骤。

启动 VS Code，并关闭选项卡 Get Started（如果它是打开的）。选择菜单 File ▶ New File 或按 Ctrl + N（在 macOS 系统中为 Command + N）新建一个文件，再将这个文件保存到文件夹 python_work 中，并命名为 hello_world.py。文件扩展名.py 告诉 VS Code，文件中的代码是使用 Python 编写的，从而让它知道如何运行这个程序，并以有帮助的方式高亮其中的代码。

保存这个文件后，在其中输入如下代码行：

hello_world.py
```
print("Hello Python world!")
```

要运行这个程序，选择菜单 Run ▶ Run Without Debugging 或按 Ctrl + F5。在 VS Code 应用程序窗口的底部将出现一个终端窗口，其中包含程序的输出：

```
Hello Python world!
```

可能还有其他输出，提示使用了 Python 解释器来运行程序。如果想精简显示的信息，以便只看到程序的输出，请参阅附录 B。附录 B 还提供了有关如何更高效地使用 VS Code 的建议。

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？是否遗漏了引号或括号？编程语言的语法非常严格，只要不满足要求就会报错。如果你无法运行这个程序，请参阅下一节的建议。

1.4　排除安装问题

如果无法运行程序 hello_world.py，可尝试如下几个解决方法。这些通用方法适用于任何编

程问题。

- 当程序存在严重错误时，Python 将显示 traceback，即错误报告。Python 会仔细研究文件，试图找出其中的问题。traceback 可能会提供线索，让你知道是什么问题让程序无法运行。
- 先离开计算机，休息一会儿再尝试。别忘了，语法在编程中非常重要，即便是引号不匹配或括号不匹配，也可能导致程序无法正确运行。请再次阅读本章的相关内容，并重新审视你编写的代码，看看能否找出错误。
- 推倒重来。不需要卸载任何软件，删除文件 hello_world.py 并重新创建也许是个合理的选择。
- 让别人在你的计算机或其他计算机上按本章的步骤重做一遍，并仔细观察。你可能遗漏了一小步，而别人刚好没有遗漏。
- 参阅附录 A 中的详尽安装说明，其中的一些细节可能可以帮助你解决问题。
- 请懂 Python 的人帮忙。当你有这样的想法时，可能会发现在你认识的人当中就有人使用 Python。
- 本章的安装说明可在本书主页上下载。对你来说，在线版也许更合适，因为可以复制并粘贴其中的代码，还可以点击指向资源的链接。
- 到网上寻求帮助。附录 C 提供了很多在线资源，如论坛或在线聊天网站，你可以在这些地方请教解决过相同问题的人。

不要担心这会打扰经验丰富的程序员。每个程序员都遇到过问题，大多数程序员会乐意帮助你正确地设置系统。只要能清晰地说明你要做什么、尝试了哪些方法及其结果，就很可能有人能帮到你。正如前言中指出的，Python 社区对初学者非常友好。

任何现代计算机都能够运行 Python。前期的配置问题可能会令人沮丧，但很值得花时间去解决。能够运行 hello_world.py 后，你就可以开始学习 Python 了，编程工作会更有趣，也更令人愉快。

1.5 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，但是从终端运行程序有时候很有用。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 存储到了桌面上的文件夹 python_work 中。

1.5.1 在 Windows 系统中从终端运行 Python 程序

可以使用终端命令 `cd`（表示 change directory，即**切换目录**）在命令窗口中浏览文件系统。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!
```

首先，使用命令 cd 来切换到文件夹 Desktop\python_work。接下来，使用命令 dir 来确认这个文件夹中包含文件 hello_world.py。最后，使用命令 python hello_world.py 来运行这个文件。

大多数程序可直接从编辑器运行，但在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。

1.5.2　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，可使用终端命令 cd 浏览文件系统。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$ python3 hello_world.py
```

首先，使用命令 cd 来切换到文件夹 Desktop/python_work。接下来，使用命令 ls 来确认这个文件夹中包含文件 hello_world.py。最后，使用命令 python3 hello_world.py 来运行这个文件。

大多数程序可直接从编辑器运行，但当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。

动手试一试

本章的练习都是探索性的，但从第 2 章开始将要求你应用学到的知识来解决问题。

练习 1.1：Python 官网　浏览 Python 官网主页，寻找你感兴趣的主题。你对 Python 越熟悉，这个网站对你来说就越有用。

练习 1.2：输入错误　打开你刚创建的文件 hello_world.py，在代码中添加一个输入错误，再运行这个程序。输入错误会引发错误吗？你能理解显示的错误消息吗？你能添加不会导致错误的输入错误吗？你凭什么认为它不会导致错误？

练习 1.3：无穷的技能 如果你有无穷多种编程技能，你打算开发什么样的程序呢？你就要开始学习编程了。如果心中有目标，就能立即应用新学到的技能，现在正是草拟目标的大好时机。将想法记录下来是个不错的习惯，这样每当需要开始新项目时，都可参考它们。现在请花点儿时间描述三个你想创建的程序。

1.6 小结

在本章中，你首先大致了解了 Python，并在自己的系统中安装了 Python。然后安装了一个文本编辑器，以简化 Python 代码的编写工作。你学习了如何在终端会话中运行 Python 代码片段，并运行了第一个程序——hello_world.py。最后大致地了解了如何排除安装问题。

在下一章中，你将学习如何在 Python 程序中使用各种数据和变量。

Part 2 第二部分

项　　目

祝贺你！你现在已经对 Python 有足够的认识，可以开始开发有意思的交互式项目了。通过动手开发项目，你不仅能学到新技能，还能更深入地理解第一部分中介绍的概念。

第二部分包含三个不同类型的项目，你可以选择完成其中的任意或全部项目，完成这些项目的顺序无关紧要。下面简要地描述每个项目，帮助你决定先完成哪一个。

外星人入侵

在项目“外星人入侵”（第 12 ~ 14 章）中，你将使用 Pygame 包开发一款 2D 游戏，它在玩家每消灭一个向下移动的外星舰队后，让玩家提高一个等级。等级越高，游戏的节奏越快，难度越大。完成这个项目后，你将获得自己动手使用 Pygame 开发 2D 游戏所需的技能。

数据可视化

“数据可视化”项目始于第 15 章，你将在这一章中学习如何使用 Matplotlib 和 Plotly 来生成数据，以及根据这些数据创建实用而漂亮的图形。第 16 章介绍如何从网上获取数据，并将其提供给可视化包以创建天气图和世界地震活动散点图。最后，第 17 章介绍如何编写自动下载数据并对其进行可视化的程序。学习可视化让你能够探索数据科学领域，这是当前最热门的编程技能应用领域之一。

Web 应用程序

在“Web 应用程序”项目（第 18 ~ 20 章）中，你将使用 Django 包来创建一个简单的 Web 应用程序，让用户能够记录所学的不同主题。用户将通过指定用户名和密码来创建账户，输入主题，并编写条目来记录学习的内容。你还将把该应用程序部署到远程服务器上，让所有人都能够访问它。

完成这个项目后，你将能够自己动手创建简单的 Web 应用程序，并能够深入学习其他有关如何使用 Django 开发应用程序的资源。

项目 1　外星人入侵

第 12 章

武装飞船 12

我们来开发一个名为《外星人入侵》的游戏吧！为此，我们将使用 Pygame 这个功能强大而且非常有趣的模块，它可以管理游戏制中用到的图像、动画甚至声音，让你能够更轻松地开发复杂的游戏。使用 Pygame 来处理在屏幕上绘制图像等任务，有助于你将重心放在设计游戏的高级逻辑上。

在本章中，你将安装 Pygame，然后创建一艘能够根据用户输入左右移动和射击的武装飞船。在接下来的两章中，你将创建一个作为射击目标的外星舰队，并改进这款游戏：限制玩家可使用的飞船数，以及添加记分牌。

在开发这款游戏的过程中，你还将学习如何管理包含多个文件的大型项目。你将学习如何通过重构代码和管理文件内容，来创建整洁、代码高效的项目。

开发游戏是趣学语言的一种理想方式。看别人玩你编写的游戏能获得满足感，编写简单的游戏也有助于你明白专业人员是如何开发游戏的。在阅读本章的过程中，请动手输入并运行代码，理解各个代码块对整个游戏的贡献。另外，请尝试不同的值和设置，以便更好地理解如何提升游戏的交互性。

注意：游戏《外星人入侵》包含很多不同的文件，因此请在系统中新建一个名为 alien_invasion 的文件夹，并将这个项目的所有文件都存储到该文件夹中。这样，相关的 import 语句才能正确工作。

如果你熟悉版本控制，可以将其用于这个项目；如果你没有使用过版本控制，请参阅附录 D 的概述。

12.1 规划项目

在开发大型项目时，先制定好规划再动手编写代码很重要。规划可确保你不偏离轨道，提高项目成功的可能性。

下面来为游戏《外星人入侵》编写大概的玩法说明，其中虽然没有涵盖这款游戏的所有细节，但能让你清楚地知道该如何动手开发它：

> 在游戏《外星人入侵》中，玩家控制着一艘最初出现在屏幕底部中央的武装飞船。玩家可以使用方向键左右移动飞船，使用空格键进行射击。当游戏开始时，一个外星舰队出现在天空中，并向屏幕下方移动。玩家的任务是消灭这些外星人。玩家将外星人消灭干净后，将出现一个新的外星舰队，其移动速度更快。只要有外星人撞到玩家的飞船或到达屏幕下边缘，玩家就损失一艘飞船。玩家损失三艘飞船后，游戏结束。

在开发的第一个阶段，我们将创建一艘飞船，这艘飞船在用户按方向键时能够左右移动，并在用户按空格键时开火。设置这种行为后，就可以创建外星人以提高游戏的可玩性了。

12.2　安装 Pygame

开始写程序前，需要安装 Pygame。这里将像第 11 章安装 `pytest` 那样安装 Pygame——使用 pip。如果你跳过了第 11 章，或者需要复习 pip 的用法，请参阅 11.1 节。

通过如下终端命令即可安装 Pygame：

```
$ python -m pip install --user pygame
```

如果你在运行程序或启动终端会话时使用的命令不是 `python`，而是 `python3`，务必将上述命令中的 `python` 替换为 `python3`。

12.3　开始游戏项目

现在开始开发游戏《外星人入侵》。首先创建一个空的 Pygame 窗口，稍后将在其中绘制游戏元素，如飞船和外星人。之后，我们还将让这个游戏响应用户输入，设置背景色，以及加载飞船图像。

12.3.1　创建 Pygame 窗口及响应用户输入

下面创建一个表示游戏的类，以创建空的 Pygame 窗口。在文本编辑器中新建一个文件，将其保存为 alien_invasion.py，再在其中输入如下代码：

alien_invasion.py

```
import sys

import pygame

class AlienInvasion:
    """管理游戏资源和行为的类"""
```

```
    def __init__(self):
        """初始化游戏并创建游戏资源"""
❶       pygame.init()

❷       self.screen = pygame.display.set_mode((1200, 800))
        pygame.display.set_caption("Alien Invasion")

    def run_game(self):
        """开始游戏的主循环"""
❸       while True:
            # 侦听键盘和鼠标事件
❹           for event in pygame.event.get():
❺               if event.type == pygame.QUIT:
                    sys.exit()

            # 让最近绘制的屏幕可见
❻           pygame.display.flip()

if __name__ == '__main__':
    # 创建游戏实例并运行游戏
    ai = AlienInvasion()
    ai.run_game()
```

首先，导入模块 sys 和 pygame。pygame 模块包含开发游戏所需的功能。当玩家退出时，我们将使用 sys 模块中的工具来退出游戏。

为开发游戏《外星人入侵》，首先创建一个名为 AlienInvasion 的类。在这个类的__init__()方法中，调用 pygame.init()函数来初始化背景，让 Pygame 能够正确地工作（见❶）。然后，调用 pygame.display.set_mode()创建一个显示窗口（见❷），这个游戏的所有图形元素都将在其中绘制。实参(1200，800)是一个元组，指定了游戏窗口的尺寸——宽 1200 像素、高 800 像素（你可以根据自己的显示器尺寸调整）。将这个显示窗口赋给属性 self.screen，让这个类的所有方法都能够使用它。

赋给属性 self.screen 的对象是一个 surface。在 Pygame 中，**surface** 是屏幕的一部分，用于显示游戏元素。在这个游戏中，每个元素（如外星人或飞船）都是一个 surface。display.set_mode()返回的 surface 表示整个游戏窗口，激活游戏的动画循环后，每经过一次循环都将自动重绘这个 surface，将用户输入触发的所有变化都反映出来。

这个游戏由 run_game()方法控制。该方法包含一个不断运行的 while 循环（见❸），而这个循环包含一个事件循环以及管理屏幕更新的代码。**事件**是用户玩游戏时执行的操作，如按键或移动鼠标。为了让程序能够响应事件，可编写一个**事件循环**，以**侦听**事件并根据发生的事件类型执行适当的任务。嵌套在 while 循环中的 for 循环（见❹）就是一个事件循环。

我们使用 pygame.event.get()函数来访问 Pygame 检测到的事件。这个函数返回一个列表，其中包含它在上一次调用后发生的所有事件。所有键盘和鼠标事件都将导致这个 for 循环运行。在这个循环中，我们将编写一系列 if 语句来检测并响应特定的事件。例如，当玩家单击游戏窗

口的关闭按钮时，将检测到 pygame.QUIT 事件，进而调用 sys.exit()来退出游戏（见❺）。

❻处调用了 pygame.display.flip()，命令 Pygame 让最近绘制的屏幕可见。这里，它在每次执行 while 循环时都绘制一个空屏幕，并擦去旧屏幕，使得只有新的空屏幕可见。我们在移动游戏元素时，pygame.display.flip()将不断更新屏幕，以显示新位置上的元素并隐藏原来位置上的元素，从而营造平滑移动的效果。

在这个文件末尾，创建一个游戏实例并调用 run_game()。这些代码被放在一个 if 代码块中，仅当直接运行该文件时，它们才会执行。如果此时运行 alien_invasion.py，将看到一个空的 Pygame 窗口。

12.3.2　控制帧率

理想情况下，游戏在所有的系统中都应以相同的速度（**帧率**）运行。对于可在多种系统中运行的游戏，控制帧率是个复杂的问题，好在 Pygame 提供了一种相对简单的方式来达成这个目标。我们将创建一个时钟（clock），并确保它在主循环每次通过后都进行计时（tick）。当这个循环的通过速度超过我们定义的帧率时，Pygame 会计算需要暂停多长时间，以便游戏的运行速度保持一致。

我们在__init__()方法中定义这个时钟：

alien_invasion.py

```
    def __init__(self):
        """初始化游戏并创建游戏资源"""
        pygame.init()
        self.clock = pygame.time.Clock()
        --snip--
```

初始化 pygame 后，创建 pygame.time 模块中的 Clock 类的一个实例，然后在 run_game()的 while 循环末尾让这个时钟进行计时：

```
    def run_game(self):
        """开始游戏的主循环"""
        while True:
            --snip--
            pygame.display.flip()
            self.clock.tick(60)
```

tick()方法接受一个参数：游戏的帧率。这里使用的值为 60， Pygame 将尽可能确保这个循环每秒恰好运行 60 次。

注意：在大多数系统中，使用 Pygame 提供的时钟有助于确保游戏的运行速度保持一致。如果在你的系统中，使用时钟导致游戏运行速度的一致性变差，可尝试不同的帧率值。如果找不到合适的帧率值，可不使用时钟，直接通过调整游戏的设置来让游戏在你的系统中平稳地运行。

12.3.3 设置背景色

Pygame 默认创建一个黑色屏幕，这太乏味了。下面在__init__()方法末尾将背景设置为另一种颜色：

alien_invasion.py

```
    def __init__(self):
        --snip--
        pygame.display.set_caption("Alien Invasion")

        # 设置背景色
❶       self.bg_color = (230, 230, 230)

    def run_game(self):
        --snip--
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()

            # 每次循环时都重绘屏幕
❷           self.screen.fill(self.bg_color)

            # 让最近绘制的屏幕可见
            pygame.display.flip()
            self.clock.tick(60)
```

在 Pygame 中，颜色是以 RGB 值指定的。这种色彩模式由红色（R）、绿色（G）和蓝色（B）值组成，其中每个值的可能取值范围都是 0 ~ 255。颜色值(255, 0, 0)表示红色，(0, 255, 0)表示绿色，(0, 0, 255)表示蓝色。通过组合不同的 RGB 值，可创建超过 1600 万种颜色。在颜色值(230, 230, 230)中，红色、绿色和蓝色的量相同，呈现出一种浅灰色。我们将这种颜色赋给 self.bg_color（见❶）。

在❷处，调用 fill()方法用这种背景色填充屏幕。fill()方法用于处理 surface，只接受一个表示颜色的实参。

12.3.4 创建 Settings 类

每次给游戏添加新功能时，通常会引入一些新设置。下面来编写一个名为 settings 的模块，其中包含一个名为 Settings 的类，用于将所有设置都存储在一个地方，以免在代码中到处添加设置。这样，每当需要访问设置时，只需使用一个 settings 对象。在项目规模增大时，这还让游戏的外观和行为修改起来更加容易：在（接下来将创建的）settings.py 中修改一些相关的值即可，无须查找散布在项目中的各种设置。

在文件夹 alien_invasion 中，新建一个名为 settings.py 的文件，并在其中添加如下 Settings 类：

settings.py

```
class Settings:
    """存储游戏《外星人入侵》中所有设置的类"""
```

```
    def __init__(self):
        """初始化游戏的设置"""
        # 屏幕设置
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)
```

为了在项目中创建 Settings 实例，并使用它来访问设置，需要将 alien_invasion.py 修改成下面这样：

alien_invasion.py

```
--snip--
import pygame

from settings import Settings

class AlienInvasion:
    """管理游戏资源和行为的类"""

    def __init__(self):
        """初始化游戏并创建游戏资源"""
        pygame.init()
        self.clock = pygame.time.Clock()
❶       self.settings = Settings()

❷       self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Alien Invasion")

    def run_game(self):
            --snip--
            # 每次循环时都重绘屏幕
❸           self.screen.fill(self.settings.bg_color)

            # 让最近绘制的屏幕可见
            pygame.display.flip()
            self.clock.tick(60)
--snip--
```

在主程序文件中，首先导入 Settings 类，并在调用 pygame.init()后创建一个 Settings 实例，这个案例被赋给 self.settings（见❶）。在创建屏幕时（见❷），使用了 self.settings 的属性 screen_width 和 screen_height 来获取屏幕的宽度和高度；在接下来填充屏幕时，也使用了 self.settings 来获取背景色（见❸）。

如果此时运行 alien_invasion.py，结果不会有任何不同，因为我们只是将设置移到了不同的地方。现在可以在屏幕上添加新元素了。

12.4　添加飞船图像

下面将飞船加入游戏。为了在屏幕上绘制玩家的飞船，需要先加载一幅图像，再使用 Pygame

blit()方法绘制它。

在为游戏选择素材时，务必注意是否有版权许可。最安全、成本最低的方式是使用 OpenGameArt 等网站提供的免费图形，这些素材无须授权许可即可使用和修改。

在游戏中，可以使用几乎任意类型的图像文件，但使用位图（.bmp）文件最为简单，因为 Pygame 默认加载位图。虽然可配置 Pygame 以使用其他文件类型，但有些文件类型要求你在计算机上安装相应的图像库。网上的大多数图像是.jpg 和.png 格式的，不过可以使用 Photoshop、GIMP 和 Paint 等工具将其转换为位图。

在选择图像时，要特别注意背景色。请尽可能选择背景为透明或纯色的图像，以便使用图像编辑器将背景改成任意颜色。当图像的背景色与游戏的背景色一致时，游戏看起来最漂亮。简单起见，也可以直接将游戏的背景色设置成图像的背景色。

就游戏《外星人入侵》而言，飞船图像可使用文件 ship.bmp（如图 12-1 所示），它可在本书的源代码文件中找到（chapter_12/adding_ship_image/images/ship.bmp）。这个文件的背景色与项目使用的设置相同。请在项目文件夹（alien_invasion）中新建一个名为 images 的文件夹，并将文件 ship.bmp 保存在其中。

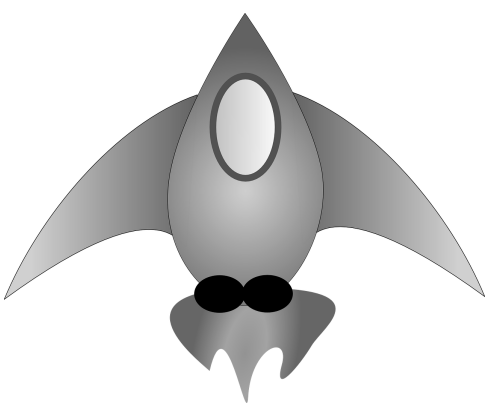


图 12-1 游戏《外星人入侵》中的飞船

12.4.1 创建 Ship 类

选择好用于表示飞船的图像后，需要将其显示到屏幕上。我们创建一个名为 ship 的模块，其中包含 Ship 类，负责管理飞船的大部分行为：

ship.py

```
import pygame

class Ship:
    """管理飞船的类"""
```

```
    def __init__(self, ai_game):
        """初始化飞船并设置其初始位置"""
❶       self.screen = ai_game.screen
❷       self.screen_rect = ai_game.screen.get_rect()

        # 加载飞船图像并获取其外接矩形
❸       self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()

        # 每艘新飞船都放在屏幕底部的中央
❹       self.rect.midbottom = self.screen_rect.midbottom

❺   def blitme(self):
        """在指定位置绘制飞船"""
        self.screen.blit(self.image, self.rect)
```

Pygame 之所以高效，是因为它让你能够把所有的游戏元素当作矩形（rect 对象）来处理，即便它们的形状并非矩形也一样。而把游戏元素当作矩形来处理之所以高效，是因为矩形是简单的几何形状。例如，通过将游戏元素视为矩形，Pygame 能够更快地判断出它们是否发生了碰撞。这种做法的效果通常很好，游戏玩家几乎注意不到我们处理的不是游戏元素的实际形状。在这个类中，我们将把飞船和屏幕作为矩形进行处理。

定义这个类之前，导入模块 pygame。Ship 的__init__()方法接受两个参数：除了 self 引用，还有一个指向当前 AlienInvasion 实例的引用。这让 Ship 能够访问 AlienInvasion 中定义的所有游戏资源。在❶处，将屏幕赋给 Ship 的一个属性，这样可在这个类的所有方法中轻松地访问它。在❷处，使用 get_rect()方法访问屏幕的 rect 属性，并将其赋给 self.screen_rect，这让我们能够将飞船放到屏幕的正确位置上。

为了加载图像，我们调用 pygame.image.load()，并将飞船图像的位置传递给它（见❸）。这个函数返回一个表示飞船的 surface，而我们将这个 surface 赋给了 self.image。加载图像后，调用 get_rect()获取相应 surface 的属性 rect，以便将来使用它来指定飞船的位置。

在处理 rect 对象时，可使用矩形的四个角及中心的 x 坐标和 y 坐标，通过设置这些值来指定矩形的位置。如果要将游戏元素居中，可设置相应 rect 对象的属性 center、centerx 或 centery；要让游戏元素与屏幕边缘对齐，可设置属性 top、bottom、left 或 right。除此之外，还有一些组合属性，如 midbottom、midtop、midleft 和 midright。要调整游戏元素的水平或垂直位置，可使用属性 x 和 y，它们分别是相应矩形左上角的 x 坐标和 y 坐标。这些属性让你无须去做游戏开发人员原本需要手动完成的计算，因此很常用。

注意：在 Pygame 中，原点(0, 0)位于屏幕的左上角，当一个点向右下方移动时，它的坐标值将增大。在 1200×800 的屏幕上，原点位于左上角，右下角的坐标为(1200, 800)。这些坐标对应的是游戏窗口，而不是物理屏幕。

因为我们要将飞船放在屏幕底部的中央，所以将 self.rect.midbottom 设置为表示屏幕的矩形的属性 midbottom（见❹）。Pygame 将使用这些 rect 属性来放置飞船图像，使其与屏幕下边缘对齐并水平居中。

最后，定义 blitme()方法（见❺），它会将图像绘制到 self.rect 指定的位置。

12.4.2 在屏幕上绘制飞船

下面更新 alien_invasion.py，创建一艘飞船并调用其方法 blitme()：

alien_invasion.py

```
--snip--
from settings import Settings
from ship import Ship

class AlienInvasion:
    """管理游戏资源和行为的类"""

    def __init__(self):
        --snip--
        pygame.display.set_caption("Alien Invasion")

❶       self.ship = Ship(self)

    def run_game(self):
            --snip--
            # 每次循环时都重绘屏幕
            self.screen.fill(self.settings.bg_color)
❷           self.ship.blitme()

            # 让最近绘制的屏幕可见
            pygame.display.flip()
            self.clock.tick(60)
--snip--
```

我们导入 Ship 类，并在创建屏幕后创建一个 Ship 实例（见❶）。在调用 Ship()时，必须提供一个参数：一个 AlienInvasion 实例。在这里，self 指向的是当前的 AlienInvasion 实例。这个参数让 Ship 能够访问游戏资源，如对象 screen。我们将这个 Ship 实例赋给了 self.ship。

填充背景后，调用 ship.blitme()将飞船绘制到屏幕上，确保它出现在背景的前面（见❷）。

现在运行 alien_invasion.py，将看到飞船位于游戏屏幕底部的中央，如图 12-2 所示。

图 12-2　游戏《外星人入侵》屏幕底部的中央有一艘飞船

12.5　重构：_check_events()方法和_update_screen()方法

在大型项目中，经常需要在添加新代码前重构既有的代码。重构旨在简化既有代码的结构，使其更容易扩展。本节将把越来越长的 run_game()方法拆分成两个辅助方法。**辅助方法**（helper method）一般只在类中调用，不会在类外调用。在 Python 中，辅助方法的名称以单下划线打头。

12.5.1　_check_events()方法

我们将把管理事件的代码移到一个名为_check_events()的方法中，以简化 run_game()并隔离事件循环。通过隔离事件循环，可将事件管理与游戏的其他方面（如更新屏幕）分离。

下面是新增_check_events()方法后的 AlienInvasion 类，只有 run_game()的代码受到了影响：

alien_invasion.py

```
    def run_game(self):
        """开始游戏的主循环"""
        while True:
❶           self._check_events()

            # 每次循环时都重绘屏幕
            --snip--

❷   def _check_events(self):
        """响应按键和鼠标事件"""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

我们新增了_check_events()方法（见❷），并将检查玩家是否单击了关闭窗口按钮的代码移到这个方法中。

要调用当前类的方法，可使用点号，并指定变量名 self 和要调用的方法的名称（见❶）。我们在 run_game()的 while 循环中调用了这个新增的方法。

12.5.2 _update_screen()方法

为了进一步简化 run_game()，我们把更新屏幕的代码移到一个名为_update_screen()的方法中：

alien_invasion.py

```
    def run_game(self):
        """开始游戏的主循环"""
        while True:
            self._check_events()
            self._update_screen()
            self.clock.tick(60)

    def _check_events(self):
        --snip--

    def _update_screen(self):
        """更新屏幕上的图像，并切换到新屏幕"""
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()

        pygame.display.flip()
```

这将绘制背景和飞船以及切换屏幕的代码移到了_update_screen()方法中。现在，run_game()中的主循环简单多了，很容易看出在每次循环中都会检测新发生的事件、更新屏幕并让时钟计时。

如果你开发过很多游戏，可能早就开始像这样将代码放到不同的方法中了；但如果你从未开发过这样的项目，可能在一开始不知道应该如何组织代码。这里采用的做法是，先编写尽量简单、可行的代码，再在代码越来越复杂时进行重构——这就是现实中的开发过程。

对代码进行重构使其更容易扩展后，就可以开始让游戏“动起来”了！

动手试一试

练习 12.1：蓝色的天空 创建一个背景为蓝色的 Pygame 窗口。

练习 12.2：游戏角色 找一幅你喜欢的游戏角色的位图图像或将一幅图像转换为位图。创建一个类，将该角色绘制到屏幕中央，并将该图像的背景色设置为屏幕的背景色或将屏幕的背景色设置为该图像的背景色。

12.6　驾驶飞船

下面来让玩家能够左右移动飞船。你将编写代码，在用户按左右方向键时做出响应。先看看如何向右移动，再以同样的方式控制飞船向左移动。通过这样做，你将学会移动屏幕上的图像以及响应用户输入。

12.6.1　响应按键

在 Pygame 中，事件都是通过 pygame.event.get()方法获取的，因此需要在_check_events()方法中指定要检查的事件类型。每当用户按键时，都将在 Pygame 中产生一个 KEYDOWN 事件。

在检测到 KEYDOWN 事件时，需要检查按下的是否是触发行动的键。如果玩家按下的是右方向键，就增大飞船的 rect.x 值，使飞船向右移动：

alien_invasion.py

```
    def _check_events(self):
        """响应按键和鼠标事件"""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
❶           elif event.type == pygame.KEYDOWN:
❷               if event.key == pygame.K_RIGHT:
                    # 向右移动飞船
❸                   self.ship.rect.x += 1
```

在_check_events()方法中，为事件循环添加一个 elif 代码块，以便在 Pygame 检测到 KEYDOWN 事件时做出响应（见❶）。我们检查按下的键（event.key）是否是右方向键（pygame.K_RIGHT）（见❷）。如果是，就将 self.ship.rect.x 的值加 1，从而使飞船向右移动（见❸）。

如果现在运行 alien_invasion.py，那么每按一次右方向键，飞船都将向右移动 1 像素。这只是一个开端，并非控制飞船的高效方式。下面来改进控制方式，允许飞船持续移动。

12.6.2　允许持续移动

当玩家按住右方向键不放时，我们希望飞船持续向右移动，直到玩家释放该键为止。我们将让游戏检测 pygame.KEYUP 事件，以便知道玩家何时释放右方向键。然后，将结合使用 KEYDOWN 和 KEYUP 事件以及一个名为 moving_right 的标志来实现持续移动。

当标志 moving_right 为 False 时，飞船不会移动。当玩家按下右方向键时，我们将这个标志设置为 True；当玩家释放该键时，将这个标志重新设置为 False。

飞船的属性都由 Ship 类控制，因此要给这个类添加一个名为 moving_right 的属性和一个名为 update()的方法。update()方法检查标志 moving_right 的状态。如果这个标志为 True，就调整飞船的位置。我们将在每次通过 while 循环时调用一次这个方法，以更新飞船的位置。

下面是对 Ship 类所做的修改：

ship.py

```
class Ship:
    """管理飞船的类"""

    def __init__(self, ai_game):
        --snip--
        # 每艘新飞船都放在屏幕底部的中央
        self.rect.midbottom = self.screen_rect.midbottom

        # 移动标志（飞船一开始不移动）
❶       self.moving_right = False

❷   def update(self):
        """根据移动标志调整飞船的位置"""
        if self.moving_right:
            self.rect.x += 1

    def blitme(self):
        --snip--
```

在__init__()方法中，添加属性 self.moving_right，并将其初始值设置为 False（见❶）。接下来，添加 update()方法，它在前述标志为 True 时向右移动飞船（见❷）。update()方法将被从类外调用，因此不是辅助方法。

接下来，需要修改_check_events()，使其在玩家按下右方向键时将 moving_right 设置为 True，并在玩家释放时将 moving_right 设置为 False：

alien_invasion.py

```
    def _check_events(self):
        """响应按键和鼠标事件"""
        for event in pygame.event.get():
            --snip--
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT:
❶                   self.ship.moving_right = True
❷           elif event.type == pygame.KEYUP:
                if event.key == pygame.K_RIGHT:
                    self.ship.moving_right = False
```

在❶处，修改游戏在玩家按下右方向键时响应的方式：不直接调整飞船的位置，只是将 moving_right 设置为 True。在❷处，添加一个新的 elif 代码块，用于响应 KEYUP 事件：当玩家释放右方向键（K_RIGHT）时，将 moving_right 设置为 False。

最后，需要修改 run_game()中的 while 循环，以便在每次执行循环时都调用飞船的 update()方法：

alien_invasion.py

```
    def run_game(self):
        """开始游戏的主循环。"""
```

```
        while True:
            self._check_events()
            self.ship.update()
            self._update_screen()
            self.clock.tick(60)
```

飞船的位置将在检测到键盘事件后（但在更新屏幕前）更新。这样能让飞船的位置根据玩家输入进行更新，并确保使用更新后的位置将飞船绘制到屏幕上。

如果现在运行 alien_invasion.py 并按下右方向键，飞船将持续向右移动，直到释放右方向键为止。

12.6.3　左右移动

在飞船能够持续向右移动后，添加向左移动的逻辑很容易。我们再次修改 Ship 类和_check_events()方法。下面显示了对 Ship 类的__init__()方法和 update()方法所做的相关修改：

ship.py

```
    def __init__(self, ai_game):
        --snip--
        # 移动标志（飞船一开始不移动）
        self.moving_right = False
        self.moving_left = False

    def update(self):
        """根据移动标志调整飞船的位置"""
        if self.moving_right:
            self.rect.x += 1
        if self.moving_left:
            self.rect.x -= 1
```

在__init__()方法中，添加标志 self.moving_left。在 update()方法中，添加一个 if 代码块，而不是 elif 代码块。这样，如果玩家同时按下左右方向键，将先增大再减小飞船的 rect.x 值，即飞船的位置保持不变。如果使用一个 elif 代码块来处理向左移动的情况，右方向键将始终处于优先地位。在改变飞船的移动方向时，玩家可能会同时按住左右方向键，此时使用两个 if 块能让移动更准确。

还需对_check_events()做两处调整：

alien_invasion.py

```
    def _check_events(self):
        """响应按键和鼠标事件"""
        for event in pygame.event.get():
            --snip--
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT:
                    self.ship.moving_right = True
                elif event.key == pygame.K_LEFT:
                    self.ship.moving_left = True
```

```
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                self.ship.moving_right = False
            elif event.key == pygame.K_LEFT:
                self.ship.moving_left = False
```

如果因玩家按下 K_LEFT 键而触发了 KEYDOWN 事件，就将 moving_left 设置为 True；如果因玩家释放 K_LEFT 键而触发了 KEYUP 事件，就将 moving_left 设置为 False。这里之所以可以使用 elif 代码块，是因为每个事件都只与一个键相关联——如果玩家同时按下左右方向键，将检测到两个不同的事件。

如果此时运行 alien_invasion.py，将能够持续地左右移动飞船。如果同时按住左右方向键，飞船将纹丝不动。

下面来进一步优化飞船的移动方式：一是调整飞船的速度，二是限制飞船的移动距离，以免因移到屏幕外而消失。

12.6.4 调整飞船的速度

当前，每次执行 while 循环时，飞船都移动 1 像素。但是，可以在 Settings 类中添加属性 ship_speed，用于控制飞船的速度。我们将根据这个属性决定飞船在每次循环时最多移动多远。下面演示了如何在 settings.py 中添加这个新属性：

settings.py

```
class Settings:
    """存储游戏《外星人入侵》中所有设置的类"""

    def __init__(self):
        --snip--

        # 飞船的设置
        self.ship_speed = 1.5
```

这里将 ship_speed 的初始值设置成 1.5。现在在移动飞船时，每次循环将移动 1.5 像素而不是 1 像素。

通过将速度设置指定为浮点数，可在稍后加快游戏的节奏时更细致地控制飞船的速度。然而，rect 的 x 等属性只能存储整数值，因此需要对 Ship 类做些修改：

ship.py

```
class Ship:
    """管理飞船的类"""

    def __init__(self, ai_game):
        """初始化飞船并设置其初始位置"""
        self.screen = ai_game.screen
❶       self.settings = ai_game.settings
        --snip--
```

```
        # 每艘新飞船都放在屏幕底部的中央
        self.rect.midbottom = self.screen_rect.midbottom

        # 在飞船的属性 x 中存储一个浮点数
❷       self.x = float(self.rect.x)

        # 移动标志（飞船一开始不移动）
        self.moving_right = False
        self.moving_left = False

    def update(self):
        """根据移动标志调整飞船的位置"""
        # 更新飞船的属性 x 的值，而不是其外接矩形的属性 x 的值
        if self.moving_right:
❸           self.x += self.settings.ship_speed
        if self.moving_left:
            self.x -= self.settings.ship_speed

        # 根据 self.x 更新 rect 对象
❹       self.rect.x = self.x

    def blitme(self):
        --snip--
```

在❶处，给 Ship 类添加属性 settings，以便能够在 update()中便捷地使用它。鉴于在调整飞船的位置时，将增减小数像素，因此需要将位置赋给一个能够存储浮点数的变量。虽然可以使用浮点数来设置 rect 的属性，但 rect 将只保留这个值的整数部分。为了准确地存储飞船的位置，定义一个可存储浮点数的新属性 self.x（见❷）。我们使用 float()函数将 self.rect.x 的值转换为浮点数，并将结果赋给 self.x。

现在在 update()中调整飞船的位置，self.x 的值会增减 settings.ship_speed 的值（见❸）。更新 self.x 后，再根据它来更新控制飞船位置的 self.rect.x（见❹）。self.rect.x 只存储 self.x 的整数部分，但对于显示飞船而言，问题不大。

现在可以修改 ship_speed 的值了。只要它的值大于 1，飞船的移动速度就会比以前更快。这有助于让飞船有足够快的反应速度，以便消灭外星人，还让我们能够随着游戏的进行加快节奏。

12.6.5　限制飞船的活动范围

当前，如果玩家按住方向键的时间足够长，飞船将移到屏幕之外，消失得无影无踪。下面来修复这个问题，让飞船到达屏幕边缘后停止移动。为此，将修改 Ship 类的 update()方法：

ship.py

```
    def update(self):
        """根据移动标志调整飞船的位置"""
        # 更新飞船而不是 rect 对象的 x 值
❶       if self.moving_right and self.rect.right < self.screen_rect.right:
            self.x += self.settings.ship_speed
```

```
❷       if self.moving_left and self.rect.left > 0:
            self.x -= self.settings.ship_speed

        # 根据 self.x 更新 rect 对象
        self.rect.x = self.x
```

上述代码在修改 self.x 的值之前检查飞船的位置。self.rect.right 返回飞船外接矩形的右边缘的 x 坐标，如果这个值小于 self.screen_rect.right 的值，就说明飞船未触及屏幕右边缘（见❶）。左边缘的情况与此类似：如果 rect 的左边缘的 x 坐标大于零，就说明飞船未触及屏幕左边缘（见❷）。这确保仅当飞船在屏幕内时，才调整 self.x 的值。

如果此时运行 alien_invasion.py，飞船将在触及屏幕左边缘或右边缘后停止移动。真是太神奇了！只在 if 语句中添加一个条件测试，就能让飞船在到达屏幕左右边缘后像被墙挡住一样。

12.6.6 重构_check_events()

随着游戏的开发，_check_events()方法将越来越长。因此我们将其部分代码放在两个方法中，其中一个处理 KEYDOWN 事件，另一个处理 KEYUP 事件：

alien_invasion.py

```
    def _check_events(self):
        """响应鼠标和按键事件"""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                self._check_keydown_events(event)
            elif event.type == pygame.KEYUP:
                self._check_keyup_events(event)

    def _check_keydown_events(self, event):
        """响应按下"""
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = True
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = True

    def _check_keyup_events(self, event):
        """响应释放"""
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = False
```

这里创建两个新的辅助方法：_check_keydown_events()和_check_keyup_events()，它们都包含形参 self 和 event。这两个方法的代码是从_check_events()中复制而来的，因此_check_events()方法中相应的代码被替换成了对这两个方法的调用。现在，_check_events()方法更简单了，代码结构也更清晰了，使程序能更容易地对玩家输入做出进一步的响应。

12.6.7　按 Q 键退出

能够高效地响应按键后，我们来添加一种退出游戏的方式。当前，每次测试新功能时，都需要单击游戏窗口顶部的 X 按钮来结束游戏，实在是太麻烦了。因此，我们来添加一个结束游戏的键盘快捷键——Q 键：

alien_invasion.py

```
    def _check_keydown_events(self, event):
        --snip--
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = True
        elif event.key == pygame.K_q:
            sys.exit()
```

在_check_keydown_events()中添加一个 elif 代码块，用于在玩家按 Q 键时结束游戏。现在测试这款游戏时，你可以直接按 Q 键来结束游戏，无须使用鼠标关闭窗口了。

12.6.8　在全屏模式下运行游戏

Pygame 支持全屏模式，相比于常规窗口，你可能更喜欢在这种模式下运行游戏。有些游戏在全屏模式下看起来更舒服，而且在一些系统中，游戏在全屏模式下可能有性能上的提升。

要在全屏模式下运行这款游戏，可在__init__()中做如下修改：

alien_invasion.py

```
    def __init__(self):
        """初始化游戏并创建游戏资源"""
        pygame.init()
        self.settings = Settings()

❶       self.screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
❷       self.settings.screen_width = self.screen.get_rect().width
        self.settings.screen_height = self.screen.get_rect().height
        pygame.display.set_caption("Alien Invasion")
```

在创建屏幕时，传入尺寸(0，0)以及参数 pygame.FULLSCREEN（见❶），这让 Pygame 生成一个覆盖整个显示器的屏幕。由于无法预先知道屏幕的宽度和高度，要在创建屏幕后更新这些设置（见❷）：使用屏幕的 rect 的属性 width 和 height 来更新对象 settings。

如果你喜欢这款游戏在全屏模式下的外观和行为，请保留这些设置；如果你更喜欢这款游戏在独立的窗口中运行，可恢复成原来的方法——将屏幕尺寸设置为特定的值。

注意：在全屏模式下运行这款游戏前，请确认能够按 Q 键退出，因为 Pygame 不提供在全屏模式下退出游戏的默认方式。

12.7 简单回顾

下一节将添加射击功能，为此需要新增一个名为 bullet.py 的文件，并修改一些既有的文件。当前有三个文件，其中包含很多类和方法。在添加其他功能前，先来回顾一下这些文件，以便对这个项目的组织结构有清楚的认识。

12.7.1 alien_invasion.py

主文件 alien_invasion.py 包含 AlienInvasion 类，这个类创建在游戏的很多地方会用到的一系列属性：赋给 settings 的设置，赋给 self.screen 的主显示 surface，以及一个飞船实例。这个模块还包含游戏的主循环，即一个调用_check_events()、ship.update()和_update_screen()的 while 循环。它还在每次通过循环后让时钟按键计时。

_check_events()方法检测相关的事件(如按下和释放),并通过调用_check_keydown_events()方法和_check_keyup_events()方法处理这些事件。当前，这些方法负责管理飞船的移动。AlienInvasion 类还包含_update_screen()方法，这个方法在每次主循环中重绘屏幕。

要开始游戏《外星人入侵》，只需运行文件 alien_invasion.py，其他文件（ settings.py 和 ship.py ）包含的代码会被导入这个文件。

12.7.2 settings.py

文件 settings.py 包含 Settings 类，这个类只包含__init__()方法，用于初始化控制游戏外观和飞船速度的属性。

12.7.3 ship.py

文件 ship.py 包含 Ship 类，这个类包含__init__()方法、管理飞船位置的 update()方法和在屏幕上绘制飞船的 blitme()方法。表示飞船的图像 ship.bmp 存储在文件夹 images 中。

动手试一试

练习 12.3：Pygame 文档 经过一段时间的游戏开发实践，你可能想看看 Pygame 的文档（ 可在 Pygame 主页中找到)。目前，只需大致浏览一下文档即可。在完成本章项目的过程中，不需要参阅这些文档，但如果你想修改游戏《外星人入侵》或编写自己的游戏，这些文档会有所帮助。

练习 12.4：火箭 编写一个游戏，它在屏幕中央显示一艘火箭，而玩家可使用上下左右四个方向键移动火箭。务必确保火箭不会移动到屏幕之外。

练习 12.5：按键 编写一个创建空屏幕的 Pygame 文件。在事件循环中，每当检测到 pygame.KEYDOWN 事件时都打印属性 event.key。运行这个程序并按下不同的键，看看控制台窗口的输出，以便了解 Pygame 会如何响应。

12.8 射击

下面来添加射击功能。我们将编写在玩家按空格键时发射子弹（用小矩形表示）的代码。子弹将在屏幕中直线上升，并在抵达屏幕上边缘后消失。

12.8.1 添加子弹设置

首先，更新 settings.py，在__init__()方法末尾存储新类 Bullet 所需的值：

settings.py

```
    def __init__(self):
        --snip--
        # 子弹设置
        self.bullet_speed = 2.0
        self.bullet_width = 3
        self.bullet_height = 15
        self.bullet_color = (60, 60, 60)
```

这些设置创建了宽 3 像素、高 15 像素的深灰色子弹。子弹的速度比飞船稍快。

12.8.2 创建 Bullet 类

下面来创建存储 Bullet 类的文件 bullet.py，其前半部分如下：

bullet.py

```
import pygame
from pygame.sprite import Sprite

class Bullet(Sprite):
    """管理飞船所发射子弹的类"""

    def __init__(self, ai_game):
        """在飞船的当前位置创建一个子弹对象"""
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # 在(0,0)处创建一个表示子弹的矩形，再设置正确的位置
❶       self.rect = pygame.Rect(0, 0, self.settings.bullet_width,
            self.settings.bullet_height)
❷       self.rect.midtop = ai_game.ship.rect.midtop
```

```
        # 存储用浮点数表示的子弹位置
❸       self.y = float(self.rect.y)
```

Bullet 类继承了从模块 pygame.sprite 导入的 Sprite 类。通过使用精灵（sprite），可将游戏中相关的元素编组，进而同时操作编组中的所有元素。为了创建子弹实例，__init__()需要当前的 AlienInvasion 实例，因此调用 super()来继承 Sprite。另外，还定义了用于存储屏幕和设置对象以及表示子弹颜色的属性。

在❶处，创建子弹的属性 rect。子弹并非基于图像文件的，因此必须使用 pygame.Rect()类从头开始创建一个矩形。在创建这个类的实例时，必须提供矩形左上角的 x 坐标和 y 坐标，还有矩形的宽度和高度。我们在(0, 0)处创建这个矩形，而下一行代码将其移到了正确的位置，因为子弹的初始位置取决于飞船当前的位置。子弹的宽度和高度是从 self.settings 中获取的。

在❷处，将子弹的 rect.midtop 设置为飞船的 rect.midtop，这样子弹将出现在飞船顶部，看起来像是飞船发射出来的。我们将子弹的 y 坐标存储为浮点数，以便能够微调子弹的速度（见❸）。

下面是 bullet.py 的后半部分，包括 update()方法和 draw_bullet()方法：

bullet.py

```
    def update(self):
        """向上移动子弹"""
        # 更新子弹的准确位置
❶       self.y -= self.settings.bullet_speed
        # 更新表示子弹的 rect 的位置
❷       self.rect.y = self.y

    def draw_bullet(self):
        """在屏幕上绘制子弹"""
❸       pygame.draw.rect(self.screen, self.color, self.rect)
```

update()方法管理子弹的位置。发射之后，子弹向上移动，这意味着 y 坐标将不断减小。为了更新子弹的位置，从 self.y 中减去 settings.bullet_speed 的值（见❶）。接下来，将 self.rect.y 设置为 self.y 的值（见❷）。

属性 bullet_speed 让我们能够随着游戏的进行或根据需要加快子弹的速度，以调整游戏的行为。子弹发射后，其 x 坐标始终不变，因此子弹将沿直线垂直上升。

在需要绘制子弹时，我们调用 draw_bullet()。draw.rect()使用存储在 self.color 中的颜色值，填充表示子弹的 rect 占据的那部分屏幕（见❸）。

12.8.3 将子弹存储到编组中

在定义 Bullet 类和必要的设置后，便可编写代码在玩家每次按空格键时都发射一颗子弹了。我们将在 AlienInvasion 中创建一个编组（group），用于存储所有有效的子弹，以便管理发射出去的所有子弹。这个编组是 Group 类（来自 pygame.sprite 模块）的一个实例。Group 类类似于列

表，但提供了有助于开发游戏的额外功能。在主循环中，将使用这个编组在屏幕上绘制子弹，以及更新每颗子弹的位置。

首先，导入新的 Bullet 类：

alien_invasion.py
```
--snip--
from ship import Ship
from bullet import Bullet
```

接下来，在__init__()中创建用于存储子弹的编组：

alien_invasion.py
```
    def __init__(self):
        --snip--
        self.ship = Ship(self)
        self.bullets = pygame.sprite.Group()
```

然后在 while 循环中更新子弹的位置：

alien_invasion.py
```
    def run_game(self):
        """开始游戏的主循环。"""
        while True:
            self._check_events()
            self.ship.update()
            self.bullets.update()
            self._update_screen()
            self.clock.tick(60)
```

在对编组调用 update()时，编组会自动对其中的每个精灵调用 update()，因此 self.bullets.update()将为 bullets 编组中的每颗子弹调用 bullet.update()。

12.8.4　开火

在 AlienInvasion 中，需要修改_check_keydown_events()，以便在玩家按空格键时发射一颗子弹。无须修改_check_keyup_events()，因为在玩家释放空格键时不需要做任何操作。还需修改_update_screen()，确保在调用 flip()前在屏幕上重绘每颗子弹。

为了发射子弹，需要做的工作不少，因此编写一个新方法_fire_bullet()来完成这项任务：

alien_invasion.py
```
    def _check_keydown_events(self, event):
        --snip--
        elif event.key == pygame.K_q:
            sys.exit()
❶       elif event.key == pygame.K_SPACE:
            self._fire_bullet()

    def _check_keyup_events(self, event):
        --snip--
```

```
    def _fire_bullet(self):
        """创建一颗子弹，并将其加入编组 bullets """
❷       new_bullet = Bullet(self)
❸       self.bullets.add(new_bullet)

    def _update_screen(self):
        """更新屏幕上的图像，并切换到新屏幕"""
        self.screen.fill(self.settings.bg_color)
❹       for bullet in self.bullets.sprites():
            bullet.draw_bullet()
        self.ship.blitme()

        pygame.display.flip()
--snip--
```

当玩家按空格键时，我们调用_fire_bullet()（见❶）。在_fire_bullet()中，创建一个 Bullet 实例并将其赋给 new_bullet（见❷），再使用 add()方法将其加入编组 bullets（见❸）。add()方法类似于列表的 append()方法，不过是专门为 Pygame 编组编写的。

bullets.sprites()方法返回一个列表，其中包含 bullets 编组中的所有精灵。为了在屏幕上绘制发射出的所有子弹，遍历 bullets 编组中的精灵，并对每个精灵都调用 draw_bullet()（见❹）。我们将这个循环放在绘制飞船的代码行前面，以防子弹出现在飞船上。

如果此时运行 alien_invasion.py，将能够左右移动飞船，并发射任意数量的子弹。子弹在屏幕上直线上升，抵达屏幕上边缘后消失，如图 12-3 所示。子弹的尺寸、颜色和速度可以在 settings.py 中修改。

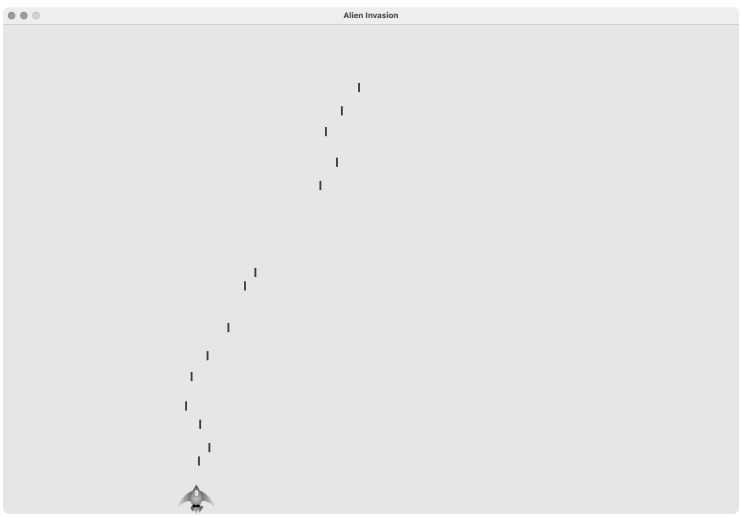


图 12-3 飞船发射一系列子弹后的《外星人入侵》游戏

12.8.5　删除已消失的子弹

当前，虽然子弹会在抵达屏幕上边缘后消失，但这仅仅是因为 Pygame 无法在屏幕外绘制它们。这些子弹实际上依然存在，它们的 y 坐标为负数且越来越小。这是个问题，因为它们将继续消耗系统的内存和处理能力。

我们需要将这些已消失的子弹删除，否则游戏所做的无谓工作将越来越多，进而变得越来越慢。为此，需要检测表示子弹的 rect 的 bottom 属性是否为零。如果是，就表明子弹已飞过屏幕上边缘：

alien_invasion.py

```
    def run_game(self):
        """开始游戏的主循环"""
        while True:
            self._check_events()
            self.ship.update()
            self.bullets.update()

            # 删除已消失的子弹
❶           for bullet in self.bullets.copy():
❷               if bullet.rect.bottom <= 0:
❸                   self.bullets.remove(bullet)
❹           print(len(self.bullets))

            self._update_screen()
            self.clock.tick(60)
```

在使用 for 循环遍历列表（或 Pygame 编组）时，Python 要求该列表的长度在整个循环中保持不变。这意味着不能从 for 循环遍历的列表或编组中删除元素，因此必须遍历编组的副本。使用方法 copy()来作为 for 循环的遍历对象（见❶），让我们能够在循环中修改原始编组 bullets。我们检查每颗子弹，看看它是否从屏幕上边缘消失了（见❷）。如果是，就将其从 bullets 中删除（见❸）。在❹处，使用函数调用 print()显示当前还有多少颗子弹，以核实确实删除了已消失的子弹。

如果这些代码没有问题，我们在发射子弹后查看终端窗口时，将发现随着子弹一颗颗地在屏幕上边缘消失，子弹数将逐渐降为零。运行这个游戏并确认子弹被正确地删除后，请将 print() 删除。如果不删除，游戏的速度将大大减慢，因为将输出写入终端的时间比将图形绘制到游戏窗口的时间还多。

12.8.6　限制子弹数量

很多射击游戏对可同时出现在屏幕上的子弹数量进行了限制，以鼓励玩家有目标地射击。在游戏《外星人入侵》中也可以做这样的限制。

首先，将允许同时出现的子弹数存储在 settings.py 中：

settings.py

```
        # 子弹设置
        --snip--
        self.bullet_color = (60, 60, 60)
        self.bullets_allowed = 3
```

这将未消失的子弹数限制为三颗。在 AlienInvasion 的_fire_bullet()中，会在创建新子弹前检查未消失的子弹数是否小于该设置：

alien_invasion.py

```
    def _fire_bullet(self):
        """创建新子弹并将其加入编组 bullets"""
        if len(self.bullets) < self.settings.bullets_allowed:
            new_bullet = Bullet(self)
            self.bullets.add(new_bullet)
```

在玩家按空格键时，我们检查 bullets 的长度。如果 len(self.bullets)小于 3，就创建一颗新子弹；但如果已经有三颗未消失的子弹，则什么都不做。现在运行这个游戏，屏幕上最多只能有三颗子弹。

12.8.7 创建_update_bullets()方法

编写并检查子弹管理代码后，可将这些代码移到一个独立的方法中，以确保 AlienInvasion 类整洁。为此，创建一个名为_update_bullets()的新方法，并将其放在_update_screen()前面：

alien_invasion.py

```
    def _update_bullets(self):
        """更新子弹的位置并删除已消失的子弹"""
        # 更新子弹的位置
        self.bullets.update()

        # 删除已消失的子弹
        for bullet in self.bullets.copy():
            if bullet.rect.bottom <= 0:
                self.bullets.remove(bullet)
```

_update_bullets()的代码是从 run_game()剪切粘贴而来的，这里只是添加了清晰注释。

run_game()中的 while 循环又变得简单了：

alien_invasion.py

```
        while True:
            self._check_events()
            self.ship.update()
            self._update_bullets()
            self._update_screen()
            self.clock.tick(60)
```

我们让主循环包含尽可能少的代码，这样只要看方法名就能迅速知道游戏中发生的情况了。主循环检查玩家的输入，并更新飞船的位置和所有未消失的子弹的位置。然后，在每次循环末尾，

都使用更新后的位置来绘制新屏幕，并让时钟计时。

请再次运行 alien_invasion.py，确认发射子弹时没有错误。

动手试一试

练习 12.6：《横向射击》　编写一个游戏，将一艘飞船放在屏幕左侧，并允许玩家上下移动飞船。在玩家按空格键时，让飞船发射一颗在屏幕中向右飞行的子弹，并在子弹从屏幕中消失后将其删除。

12.9　小结

在本章中，你首先学习了游戏开发计划的制定以及使用 Pygame 编写的游戏的基本结构。接着学习了如何设置背景色，以及如何将设置存储在独立的类中，以便将来可以轻松地调整。然后学习了如何在屏幕上绘制图像，以及如何让玩家控制游戏元素的移动。你不仅创建了能自动移动的元素，如在屏幕中直线上升的子弹，还删除了不再需要的对象。最后，你学习了经常性重构是如何为项目的后续开发提供便利的。

在第 13 章中，我们将在游戏《外星人入侵》中添加外星人。学完这一章，你将能够击落外星人——但愿是在其撞到飞船之前！