TURING 图灵程序设计丛书

no starch press

Python Crash Course

A Hands-On, Project-Based Introduction to Programming

Python编程

从入门到实践

【美】Eric Matthes 著　袁国忠 译

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

Eric Matthes

高中科学和数学老师，现居住在阿拉斯加，在当地讲授Python入门课程。他从5岁开始就一直在编写程序。

袁国忠

自由译者；2000年起专事翻译，主译图书，偶译新闻稿、软文；出版译著40余部，其中包括《C++ Prime Plus中文版》《CCNA学习指南》《CCNP　ROUTE学习指南》《面向模式的软件架构：模式系统》《Android应用UI设计模式》《风投的选择：谁是下一个十亿美元级公司》等，总计700余万字；专事翻译前，从事过三年化工产品分析和开发，做过两年杂志和图书编辑。

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

TURING 图灵程序设计丛书

Python编程
从入门到实践

【美】Eric Matthes 著　袁国忠 译

Python Crash Course
A Hands-On, Project-Based Introduction to Programming

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. -- 北京 : 人民邮电出版社, 2016.7
（图灵程序设计丛书）
ISBN 978-7-115-42802-8

Ⅰ. ①P… Ⅱ. ①埃… ②袁… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2016)第139461号

内 容 提 要

本书是一本针对所有层次的 Python 读者而作的 Python 入门书。全书分两部分：第一部分介绍用 Python 编程所必须了解的基本概念，包括 matplotlib、NumPy 和 Pygal 等强大的 Python 库和工具介绍，以及列表、字典、if 语句、类、文件与异常、代码测试等内容；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 Python 2D 游戏开发，如何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，并帮读者解决常见编程问题和困惑。

本书适合对 Python 感兴趣的任何层次的读者阅读。

◆ 著　　[美] Eric Matthes
译　　袁国忠
责任编辑　岳新欣
执行编辑　杨　琳　张　曼
责任印制　彭志环

◆ 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　http://www.ptpress.com.cn
北京　　印刷

◆ 开本：800×1000　1/16
印张：29.75
字数：703千字　　2016年 7 月第 1 版
印数：1－4 000册　　2016年 7 月北京第 1 次印刷
著作权合同登记号　图字：01-2016-1807号

定价：89.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第 8052 号

版 权 声 明

谨以此书献给我的父亲，以及儿子Ever。感谢父亲抽出时间来回答我提出的每个编程问题，而儿子Ever也开始向我提问了。

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。此情此景我永远都忘不了。

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。相比于孩提时期，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，以便能够编写能正确运行的程序——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，它不要求你有任何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学概念的理解程度，那么本书就是为你编写的。本书还可供初中和高中教师用来通过开发项目向学生介绍编程。

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将数据存储到列表和字典中的方式。你将学习如何创建数据集合以及如何高效地遍历这些集合。你将学习使用`while`和`if`语句来检查条件，并在条件满足时执行代码的一部分，而在条件不满足时执行代码的另一部分——这可为自动完成处理提供极大的帮助。

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重用，这样你编写执行特定任务的代码后，想使用它多少次都可以。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。你将学习编写妥善处理常见错误的程序。学习这些基本概念后，你就能编写一些简短的程序来解决一些明确的问题。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不用担心可能引入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。你可以根据自己的情况，以最合适的顺序完成这些项目；你也可以选择只完成其中的某些项目。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》，它包含多个难度不断增加的等级。完成这个项目后，你就能够自己动手开发2D游戏了。

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搞懂海量信息。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，你将能够编写能对大型数据集进行筛选的程序，并以可视化方式将筛选出来的数据呈现出来。

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概念记录下来。你将能够分别记录不同的主题，还可以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。

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：相比于众多其他的语言，使用Python编写时，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、解决商业问题以及供各类有趣的公司开发内部工具。Python还在科学领域被大量用于学术研究和应用研究。

我依然使用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。大多数程序员都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。需要有人帮助解决问题时，有一个联系紧密、互帮互

助的社区至关重要，而对于像你一样将Python作为第一门语言来学习的人而言，Python社区无疑是坚强的后盾。

Python是一门杰出的语言，值得你去学习，咱们现在就开始吧！

致　　谢

要是没有No Starch Press出色的专业人士的帮助，本书根本不可能出版。Bill Pollock邀请我编写一本入门图书，因此这里要深深感谢他给予我这样的机会。Tyler Ortman在我编写本书的早期帮助我理清思路。Liz Chadwick和Leslie Shen详细阅读了每一章，并提出了宝贵的意见，而Anne Marie Walker让本书的很多地方都更清晰。Riley Hoffman回答了我就图书出版过程提出的每个问题，并且耐心地将我的作品变成了漂亮的图书。

感谢技术审稿人Kenneth Love。我与Kenneth相识于一次PyCon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，还抱着让初学编程者对Python语言和编程有扎实认识的目的进行了审阅。即便如此，倘若书中有任何不准确的地方，责任都完全由我承担。

感谢我的父亲，感谢他在我很小的时候就向我介绍编程，而且一点都不担心我破坏他的设备。感谢妻子Erin在我编写本书期间对我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 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，你首先需要检查自己的计算机是否安装了Python；如果没有安装，你需要安装它。你还要安装一个文本编辑器，用于编写和运行Python程序。你输入Python代码时，这个文本编辑器能够识别它们并突出显示不同的部分，让你能够轻松地了解代码的结构。

1.1　搭建编程环境

在不同的操作系统中，Python存在细微的差别，因此有几点你需要牢记在心。这里将介绍大家使用的两个主要的Python版本，并简要介绍Python的安装步骤。

1.1.1　Python 2 和 Python 3

当前，有两个不同的Python版本：Python 2和较新的Python 3。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的开发者也一直致力于丰富和强化其功能。大多数修改都是逐步进行的，你几乎意识不到，但如果你的系统安装的是Python 3，那么有些使用Python 2编写的代码可能无法正确地运行。在本书中，我将指出Python 2和Python 3的重大差别，这样无论你安装的是哪个Python版本，都能够按书中的说明去做。

如果你的系统安装了这两个版本，请使用Python 3；如果没有安装Python，请安装Python 3；如果只安装了Python 2，也可直接使用它来编写代码，但还是尽快升级到Python 3为好，因为这样你就能使用最新的Python版本了。

1.1.2 运行 Python 代码片段

Python自带了一个在终端窗口中运行的解释器，让你无需保存并运行整个程序就能尝试运行Python代码片段。

本书将以如下方式列出代码片段：

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

加粗的文本表示需要你输入之后按回车键来执行的代码。本书的大多数示例都是独立的小程序，你将在编辑器中执行它们，因为大多数代码都是这样编写出来的。然而，为高效地演示某基本概念，需要在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尖括号（如❶所示），就意味着输出来自终端会话。稍后将演示如何在Python解释器中编写代码。

1.1.3 Hello World 程序

长期以来，编程界都认为刚接触一门新语言时，如果首先使用它来编写一个在屏幕上显示消息“Hello world!”的程序，将给你带来好运。

要使用Python来编写这种Hello World程序，只需一行代码：

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

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，你编写的任何Python程序都将如此。稍后将介绍如何在特定的系统中编写这样的程序。

1.2 在不同操作系统中搭建 Python 编程环境

Python是一种跨平台的编程语言，这意味着它能够运行在所有主要的操作系统中。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。

在这一节中，你将学习如何在自己的系统中安装Python和运行Hello World程序。你首先要检查自己的系统是否安装了Python，如果没有，就安装它；接下来，你需要安装一个简单的文本编辑器，并创建一个空的Python文件——hello_world.py。最后，你将运行Hello World程序，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，让你能够搭建一个对初学者友好的Python编程环境。

1.2.1 在 Linux 系统中搭建 Python 编程环境

Linux系统是为编程而设计的，因此在大多数Linux计算机中，都默认安装了Python。编写和维护Linux的人认为，你很可能会使用这种系统进行编程，他们也鼓励你这样做。鉴于此，要在

这种系统中编程，你几乎不用安装什么软件，也几乎不用修改设置。

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。为确定是否安装了Python，执行命令python（请注意，其中的p是小写的）。输出将类似下面这样，它指出了安装的Python版本；最后的>>>是一个提示符，让你能够输入Python命令。

```
$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:38)
[GCC 4.8.2] on linux2
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Python版本为Python 2.7.6。看到上述输出后，如果要退出Python并返回到终端窗口，可按Ctrl + D或执行命令exit()。

要检查系统是否安装了Python 3，可能需要指定相应的版本。换句话说，如果输出指出默认版本为Python 2.7，请尝试执行命令python3：

```
$ python3
Python 3.5.0 (default, Sep 17 2015, 13:05:18)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

上述输出表明，系统中也安装了Python 3，因此你可以使用这两个版本中的任何一个。在这种情况下，请将本书中的命令python都替换为python3。大多数Linux系统都默认安装了Python，但如果你的Linux系统不知什么原因没有安装Python或只安装了Python 2，而你要安装Python 3，请参见附录A。

2. 安装文本编辑器

Geany是一款简单的文本编辑器：它易于安装；让你能够直接运行几乎所有的程序（而无需通过终端来运行）；使用不同的颜色来显示代码，以突出代码语法；在终端窗口中运行代码，让你能够习惯使用终端。附录B介绍了其他一些文本编辑器，但我强烈建议你使用Geany，除非你有充分的理由不这样做。

在大多数Linux系统中，都只需执行一个命令就可以安装Geany：

```
$ sudo apt-get install geany
```

如果这个命令不管用，请参阅http://geany.org/Download/ThirdPartyPackages/的说明。

3. 运行Hello World程序

为编写第一个程序，需要启动Geany。为此，可按超级（Super）键（俗称Windows键），并在系统中搜索Geany。找到Geany后，双击以启动它；再将其拖曳到任务栏或桌面上，以创建一个快捷方式。接下来，创建一个用于存储项目的文件夹，并将其命名为python_work（在文件名

和文件夹名中，最好使用小写字母，并使用下划线来表示空格，因为这是Python采用的命名约定）。回到Geany，选择菜单File ▸ Save As，将当前的空Python文件保存到文件夹python_work，并将其命名为hello_world.py。扩展名.py告诉Geany，文件包含的是Python程序；它还让Geany知道如何运行该程序，并以有益的方式突出其中的代码。

保存文件后，在其中输入下面一行代码：

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

如果你的系统安装了多个Python版本，就必须对Geany进行配置，使其使用正确的版本。为此，可选择菜单Build（生成）▸ Set Build Commands（设置生成命令）；你将看到文字Compile（编译）和Execute（执行），它们旁边都有一个命令。默认情况下，这两个命令都是python，要让Geany使用命令python3，必须做相应的修改。

如果在终端会话中能够执行命令python3，请修改编译命令和执行命令，让Geany使用Python 3解释器。为此，将编译命令修改成下面这样：

```
python3 -m py_compile "%f"
```

你必须完全按上面的代码显示的那样输出这个命令，确保空格和大小写都完全相同。

将执行命令修改成下面这样：

```
python3 "%f"
```

同样，务必确保空格和大小写都完全与显示的相同。图1-1显示了该如何在Geany中配置这些命令。

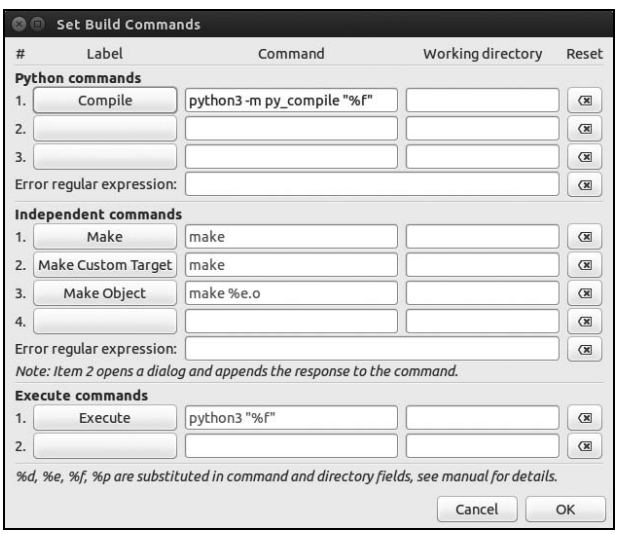


图1-1 在Linux中配置Geany，使其使用Python 3

现在来运行程序hello_world.py。为此，可选择菜单Build ▸ Execute、单击Execute图标（两个齿轮）或按F5。将弹出一个终端窗口，其中包含如下输出：

```
Hello Python world!

------------------
(program exited with code: 0)
Press return to continue
```

如果没有看到这样的输出，请检查你输入的每个字符。你是不是将print的首字母大写了？是不是遗漏了引号或括号？编程语言对语法的要求非常严格，只要你没有严格遵守语法，就会出错。如果代码都正确，这个程序也不能正确地运行，请参阅1.3节。

4. 在终端会话中运行Python代码

你可以打开一个终端窗口并执行命令python或python3，再尝试运行Python代码片段。检查Python版本时，你就这样做过。下面再次这样做，但在终端会话中输入如下代码行：

```
>>> 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()。

1.2.2　在 OS X 系统中搭建 Python 编程环境

大多数OS X系统都默认安装了Python。确定安装了Python后，你还需安装一个文本编辑器，并确保其配置正确无误。

1. 检查是否安装了Python

在文件夹Applications/Utilities中，选择Terminal，打开一个终端窗口；你也可以按Command + 空格键，再输入terminal并按回车。为确定是否安装了Python，请执行命令python（注意，其中的p是小写的）。输出将类似于下面这样，它指出了安装的Python版本；最后的>>>是一个提示符，让你能够输入Python命令。

```
$ python
Python 2.7.5 (default, Mar 9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
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Python版本为Python 2.7.5。看到上述输出后，如果要退出Python并返回到终端窗口，可按Ctrl + D或执行命令exit()。

要检查系统是否安装了Python 3，可尝试执行命令python3。可能会出现一条错误消息，但如果输出指出系统安装了Python 3，则无需安装就可使用它。如果在你的系统中能够执行命令

python3，则对于本书的所有命令python，都请替换为命令python3。如果不知道出于什么原因你的系统没有安装Python，或者只安装了Python 2，而你又想安装Python 3，请参阅附录A。

2. 在终端会话中运行Python代码

你可以打开一个终端窗口并执行命令python或python3，再尝试运行Python代码片段。检查Python版本时，你就这样做过。下面再次这样做，但在终端会话中输入如下代码行：

```
>>> 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. 安装文本编辑器

Sublime Text是一款简单的文本编辑器：它在OS X中易于安装；让你能够直接运行几乎所有程序（而无需通过终端）；使用不同的颜色来显示代码，以突出代码语法；在内嵌在Sublime Text窗口内的终端会话中运行代码，让你能够轻松地查看输出。附录B介绍了其他一些文本编辑器，但我强烈建议你使用Sublime Text，除非你有充分的理由不这样做。

要下载Sublime Text安装程序，可访问http://sublimetext.com/3，单击Download链接，并查找OS X安装程序。Sublime Text的许可策略非常灵活，你可以免费使用这款编辑器，但如果你喜欢它并想长期使用，建议你购买许可证。下载安装程序后，打开它，再将Sublime Text图标拖放到Applications文件夹。

4. 配置Sublime Text使其使用Python 3

如果你启动Python终端会话时使用的命令不是python，就需要配置Sublime Text，让它知道到系统的什么地方去查找正确的Python版本。要获悉Python解释器的完整路径，请执行如下命令：

```
$ type -a python3
python3 is /usr/local/bin/python3
```

现在，启动Sublime Text，并选择菜单Tools ▸ Build System ▸ New Build System，这将打开一个新的配置文件。删除其中的所有内容，再输入如下内容：

```
{
    "cmd": ["/usr/local/bin/python3", "-u", "$file"],
}
```

这些代码让Sublime Text使用命令python3来运行当前打开的文件。请确保其中的路径为你在前一步使用命令type -a python3获悉的路径。将这个配置文件命名为Python3.sublime-build，并将其保存到默认目录——你选择菜单Save时Sublime Text打开的目录。

5. 运行Hello World程序

为编写第一个程序，需要启动Sublime Text。为此，可打开文件夹Applications，并双击图标

Sublime Text；也可按Command + 空格键，再在弹出的搜索框中输入sublime text。

创建一个用于存储项目的文件夹，并将其命名为python_work（在文件名和文件夹名中，最好使用小写字母，并使用下划线来表示空格，因为这是Python采用的命名约定）。在Sublime Text中，选择菜单File ▸ Save As，将当前的空Python文件保存到文件夹python_work，并将其命名为hello_world.py。扩展名.py告诉Sublime Text，文件包含的是Python程序；它还让Sublime Text知道如何运行该程序，并以有益的方式突出其中的代码。

保存文件后，在其中输入下面一行代码：

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

如果在系统中能够运行命令python，就可选择菜单Tools ▸ Build或按Ctrl + B来运行程序。如果你对Sublime Text进行了配置，使其使用的命令不是python，请选择菜单Tools ▸ Build System，再选择Python 3。这将把Python 3设置为默认使用的Python版本；此后，你就可选择菜单Tools ▸ Build或按Command+ B来运行程序了。

Sublime Text窗口底部将出现一个终端屏幕，其中包含如下输出：

```
Hello Python world!
[Finished in 0.1s]
```

如果没有看到这样的输出，请检查你输入的每个字符。你是不是将print的首字母大写了？是不是遗漏了引号或括号？编程语言对语法的要求非常严格，只要你没有严格遵守语法，就会出错。如果代码都正确，这个程序也不能正确地运行，请参阅1.3节。

1.2.3　在 Windows 系统中搭建 Python 编程环境

Windows系统并非都默认安装了Python，因此你可能需要下载并安装它，再下载并安装一个文本编辑器。

1. 安装Python

首先，检查你的系统是否安装了Python。为此，在“开始”菜单中输入command并按回车以打开一个命令窗口；你也可按住Shift键并右击桌面，再选择“在此处打开命令窗口”。在终端窗口中输入python并按回车；如果出现了Python提示符（>>>），就说明你的系统安装了Python。然而，你也可能会看到一条错误消息，指出python是无法识别的命令。

如果是这样，就需要下载Windows Python安装程序。为此，请访问http://python.org/downloads/。你将看到两个按钮，分别用于下载Python 3和Python 2。单击用于下载Python 3的按钮，这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1-2所示），这让你能够更轻松地配置系统。

图1-2 确保选中复选框Add Python to PATH

2. 启动Python终端会话

通过配置系统，让其能够在终端会话中运行Python，可简化文本编辑器的配置工作。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 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 22:15:05) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

如果是这样，就可以直接跳到下一部分——“在终端会话中运行Python”。

然而，输出可能类似于下面这样：

```
C:\> python
'python' is not recognized as an internal or external command, operable
program or batch file.
```

在这种情况下，你就必须告诉Windows如何找到你刚安装的Python版本。命令python通常存储在C盘，因此请在Windows资源管理器中打开C盘，在其中找到并打开以Python打头的文件夹，再找到文件python。例如，在我的计算机中，有一个名为Python35的文件夹，其中有一个名为python的文件，因此文件python的路径为C:\Python35\python。如果找不到这个文件，请在Windows资源管理器的搜索框中输入python，这将让你能够准确地获悉命令python在系统中的存储位置。

如果认为已知道命令python的路径，就在终端窗口中输入该路径进行测试。为此，打开一个命令窗口，并输入你确定的完整路径：

```
C:\> C:\Python35\python
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 22:15:05) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

如果可行，就说明你已经知道如何访问Python了。

3. 在终端会话中运行Python

在Python会话中执行下面的命令，并确认看到了输出“Hello Python world!”。

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

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，可按Ctrl + Z，再按回车键，也可执行命令exit()。

4. 安装文本编辑器

Geany是一款简单的文本编辑器：它易于安装；让你能够直接运行几乎所有的程序（而无需通过终端）；使用不同的颜色来显示代码，以突出代码语法；在终端窗口中运行代码，让你能够习惯使用终端。附录B介绍了其他一些文本编辑器，但我强烈建议你使用Geany，除非你有充分的理由不这样做。

要下载Windows Geany安装程序，可访问http://geany.org/，单击Download下的Releases，找到安装程序geany-1.25_setup.exe或类似的文件。下载安装程序后，运行它并接受所有的默认设置。

为编写第一个程序，需要启动Geany。为此，可按超级（Super）键（俗称Windows键），并在系统中搜索Geany。找到Geany后，双击以启动它；再将其拖曳到任务栏或桌面上，以创建一个快捷方式。接下来，创建一个用于存储项目的文件夹，并将其命名为python_work（在文件名和文件夹名中，最好使用小写字母，并使用下划线来表示空格，因为这是Python采用的命名约定）。回到Geany，选择菜单File ▸ Save As，将当前的空Python文件保存到文件夹python_work，并将其命名为hello_world.py。扩展名.py告诉Geany，文件包含的是Python程序；它还让Geany知道如何运行该程序，并以有益的方式突出其中的代码。

保存文件后，在其中输入下面一行代码：

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

如果能够在系统中执行命令python，就无需配置Geany，因此你可以跳过下一部分，直接进入“运行Hello World程序”部分。如果启动Python解释器时必须指定路径，如C:\Python35\python，请按下面的说明对Geany进行配置。

5. 配置Geany

要配置Geany，请选择菜单Build ▸ Set Build Commands；你将看到文字Compile和Execute，它们旁边都有一个命令。默认情况下，编译命令和执行命令的开头都是python，但Geany不知道命

令python存储在系统的什么地方，因此你需要在其中添加你在终端会话中使用的路径。

为此，在编译命令和执行命令中，加上命令python所在的驱动器和文件夹。其中编译命令应类似于下面这样：

```
C:\Python35\python -m py_compile "%f"
```

在你的系统中，路径可能稍有不同，但请务必确保空格和大小写与这里显示的一致。

执行命令应类似于下面这样：

```
C:\Python35\python "%f"
```

同样，指定执行命令时，务必确保空格和大小写与这里显示的一致。图1-3显示了该如何在Geany中配置这些命令。

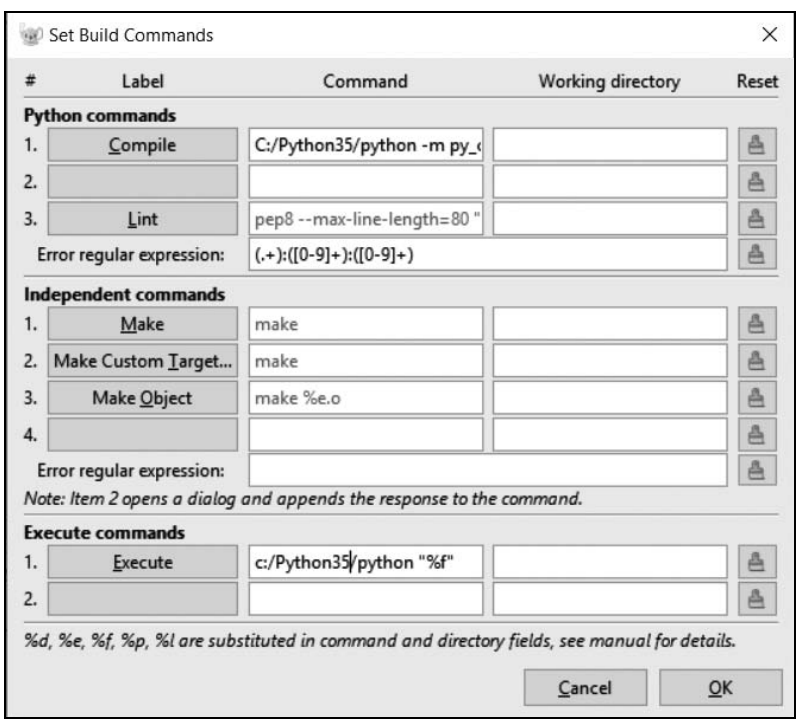


图1-3 在Windows中配置Geany，使其使用Python 3

正确地设置这些命令后，单击OK按钮。

6. 运行Hello World程序

现在应该能够成功地运行程序了。请运行程序hello_world.py；为此，可选择菜单Build ▸ Execute、单击Execute图标（两个齿轮）或按F5。将弹出一个终端窗口，其中包含如下输出：

```
Hello Python world!

------------------
(program exited with code: 0)
Press return to continue
```

如果没有看到这样的输出，请检查你输入的每个字符。你是不是将print的首字母大写了？是不是遗漏了引号或括号？编程语言对语法的要求非常严格，只要你没有严格遵守语法，就会出错。如果代码都正确，这个程序也不能正确地运行，请参阅下一节。

1.3　解决安装问题

如果你按前面的步骤做，应该能够成功地搭建编程环境。但如果你始终无法运行程序hello_world.py，可尝试如下几个解决方案。

- 程序存在严重的错误时，Python将显示traceback。Python会仔细研究文件，试图找出其中的问题。trackback可能会提供线索，让你知道是什么问题让程序无法运行。
- 离开计算机，先休息一会儿，再尝试。别忘了，在编程中，语法非常重要，即便是少一个冒号、引号不匹配或括号不匹配，都可能导致程序无法正确地运行。请再次阅读本章相关的内容，再次审视你所做的工作，看看能否找出错误。
- 推倒重来。你也许不需要把一切都推倒重来，但将文件hello_world.py删除并重新创建它也许是合理的选择。
- 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。
- 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。
- 本章的安装说明在网上也可以找到，其网址为https://www.nostarch.com/pythoncrash-course/。对你来说，在线版也许更合适。
- 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，你可以前往这些地方，请求解决过你面临的问题的人提供解决方案。

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，而大多数程序员都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，就很可能有人能够帮到你。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，如果你遇到了困难，请想办法寻求帮助吧。前期的问题可能令人沮丧，但很值得你花时间去解决。能够运行hello_world.py后，你就可以开始学习Python了，而且编程工作会更有趣，也更令人愉快。

1.4 从终端运行 Python 程序

你编写的大多数程序都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，请确保已将文件hello_world.py存储到了桌面的python_work文件夹中。

1.4.1 在 Linux 和 OS X 系统中从终端运行 Python 程序

在Linux和OS X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（表示切换目录，change directory）在文件系统中导航。命令ls（list的简写）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：

```
❶ ~$ cd Desktop/python_work/
❷ ~/Desktop/python_work$ ls
  hello_world.py
❸ ~/Desktop/python_work$ python hello_world.py
  Hello Python world!
```

这里使用了命令cd来切换到文件夹Desktop/python_work（见❶）。接下来，使用命令ls来确认这个文件夹中包含文件hello_world.py（见❷）。最后，使用命令python hello_world.py来运行这个文件（见❸）。

就这么简单。要运行Python程序，只需使用命令python（或python3）即可。

1.4.2 在 Windows 系统中从终端运行 Python 程序

在命令窗口中，要在文件系统中导航，可使用终端命令cd；要列出当前目录中的所有文件，可使用命令dir（表示目录， directory）。

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：

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

这里使用了命令cd来切换到文件夹Desktop\python_work（见❶）。接下来，使用命令dir来确认这个文件夹中包含文件hello_world.py（见❷）。最后，使用命令python hello_world.py来运行这个文件（见❸）。

如果你没有对系统进行配置以使用简单命令python，就可能需要指定这个命令的路径：

```
C:\$ cd Desktop\python_work
C:\Desktop\python_work$ dir
hello_world.py
C:\Desktop\python_work$ C:\Python35\python hello_world.py
Hello Python world!
```

大多数程序都可以直接从编辑器运行，但需要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 章开始将要求你用那一章学到的知识来解决问题。

1-1 python.org：浏览 Python 主页（http://python.org/），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。

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？你能添加一个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？

1-3 无穷的技艺：如果你编程技艺无穷，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，就能立即将新学到的技能付诸应用；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。现在请花点时间描绘三个你想创建的程序。

1.5　小结

在本章中，你大致了解了Python，并在自己的系统中安装了Python。你还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，并运行了第一个货真价实的程序——hello_world.py。你还大致了解了如何解决安装问题。

在下一章，你将学习如何在Python程序中使用各种数据和变量。

第 12 章

武装飞船 12

我们来开发一个游戏吧！我们将使用Pygame，这是一组功能强大而有趣的模块，可用于管理图形、动画乃至声音，让你能够更轻松地开发复杂的游戏。通过使用Pygame来处理在屏幕上绘制图像等任务，你不用考虑众多烦琐而艰难的编码工作，而是将重点放在程序的高级逻辑上。

在本章中，你将安装Pygame，再创建一艘能够根据用户输入而左右移动和射击的飞船。在接下来的两章中，你将创建一群作为射杀目标的外星人，并做其他的改进，如限制可供玩家使用的飞船数以及添加记分牌。

从本章开始，你还将学习管理包含多个文件的项目。我们将重构很多代码，以提高代码的效率，并管理文件的内容，以确保项目组织有序。

创建游戏是趣学语言的理想方式。看别人玩你编写的游戏让你很有满足感，而编写简单的游戏有助于你明白专业级游戏是怎么编写出来的。在阅读本章的过程中，请动手输入并运行代码，以明白各个代码块对整个游戏所做的贡献，并尝试不同的值和设置，这样你将对如何改进游戏的交互性有更深入的认识。

注意 游戏《外星人入侵》将包含很多不同的文件，因此请在你的系统中新建一个文件夹，并将其命名为alien_invasion。请务必将这个项目的所有文件都存储到这个文件夹中，这样相关的`import`语句才能正确地工作。

12.1 规划项目

开发大型项目时，做好规划后再动手编写项目很重要。规划可确保你不偏离轨道，从而提高项目成功的可能性。

下面来编写有关游戏《外星人入侵》的描述，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飞船。玩家可以使用箭头键左右移动飞船，还可使用空格键进行射击。游戏开始时，一群外星人出现在天空中，他们在屏幕中向下移动。玩家的任务是射杀这些外星人。玩家将所有外星人都消灭干净后，将出现一群新的外星人，他们移动的速度更快。只要有外星人撞到了玩家的飞船或到达了屏幕底部，玩家就损失一艘飞船。玩家损失三艘飞船后，游戏结束。

在第一个开发阶段，我们将创建一艘可左右移动的飞船，这艘飞船在用户按空格键时能够开火。设置好这种行为后，我们就能够将注意力转向外星人，并提高这款游戏的可玩性。

12.2　安装 Pygame

开始编码前，先来安装Pygame。下面介绍如何在Linux、OS X和Microsoft Windows中安装Pygame。

如果你使用的是Linux系统和Python 3，或者是OS X系统，就需要使用pip来安装Pygame。pip是一个负责为你下载并安装Python包的程序。接下来的几小节介绍如何使用pip来安装Python包。

如果你使用的是Linux系统和Python 2.7，或者是Windows，就无需使用pip来安装Pygame；在这种情况下，请直接跳到12.2.2节或12.2.4节。

注意　接下来的部分包含在各种系统上安装pip的说明，因为数据可视化项目和Web应用程序项目都需要pip。这些说明也可在https://www.nostarch.com/pythoncrashcourse/在线资源中找到。如果安装时遇到麻烦，看看在线说明是否管用。

12.2.1　使用 pip 安装 Python 包

大多数较新的Python版本都自带pip，因此首先可检查系统是否已经安装了pip。在Python 3中，pip有时被称为pip3。

1. 在Linux和OS X系统中检查是否安装了pip

打开一个终端窗口，并执行如下命令：

```
$ pip --version
❶ pip 7.0.3 from /usr/local/lib/python3.5/dist-packages (python 3.5)
$
```

如果你的系统只安装了一个版本的Python，并看到了类似于上面的输出，请跳到12.2.2节或12.2.3节。如果出现了错误消息，请尝试将pip替换为pip3。如果这两个版本都没有安装到你的系统中，请跳到“安装pip”。

如果你的系统安装了多个版本的Python，请核实pip关联到了你使用的Python版本，如python

3.5（见❶）。如果pip关联到了正确的Python版本，请跳到12.2.2节或12.2.3节。如果pip没有关联到正确的Python版本，请尝试将pip替换为pip3。如果执行这两个命令时，输出都表明没有关联到正确的Python版本，请跳到“安装pip”。

2. 在Windows系统中检查是否安装了pip

打开一个终端窗口，并执行如下命令：

```
$ python -m pip --version
❶ pip 7.0.3 from C:\Python35\lib\site-packages (python 3.5)
$
```

如果你的系统只安装了一个版本的Python，并看到了类似于上面的输出，请跳到12.2.4节。如果出现了错误消息，请尝试将pip替换为pip3。如果执行这两个命令时都出现错误消息，请跳到“安装pip”。

如果你的系统安装了多个版本的Python，请核实pip关联到了你使用的Python版本，如python 3.5（见❶）。如果pip关联到了正确的Python版本，请跳到12.2.4节。如果pip没有关联到正确的Python版本，请尝试将pip替换为pip3。如果执行这两个命令时都出现错误消息，请跳到“安装pip”。

3. 安装pip

要安装pip，请访问https://bootstrap.pypa.io/get-pip.py。如果出现对话框，请选择保存文件；如果get-pip.py的代码出现在浏览器中，请将这些代码复制并粘贴到文本编辑器中，再将文件保存为get-pip.py。将get-pip.py保存到计算机中后，你需要以管理员身份运行它，因为pip将在你的系统中安装新包。

注意 如果你找不到get-pip.py，请访问https://pip.pypa.io/，单击左边面板中的Installation，再单击中间窗口中的链接get-pip.py。

4. 在Linux和OS X系统中安装pip

使用下面的命令以管理员身份运行get-pip.py：

```
$ sudo python get-pip.py
```

注意 如果你启动终端会话时使用的是命令python3，那么在这里应使用命令sudo python3 get-pip.py。

这个程序运行后，使用命令pip --version（或pip3 --version）确认正确地安装了pip。

5. 在Windows系统中安装pip

使用下面的命令运行get-pip.py：

```
$ python get-pip.py
```

如果你在终端中运行Python时使用的是另一个命令，也请使用这个命令来运行get-pip.py。例如，你可能需要使用命令python3 get-pip.py或C:\Python35\python get-pip.py。

这个程序运行后，执行命令python -m pip --version以确认成功地安装了pip。

12.2.2　在 Linux 系统中安装 Pygame

如果你使用的是Python 2.7，请使用包管理器来安装Pygame。为此，打开一个终端窗口，并执行下面的命令，这将下载Pygame，并将其安装到你的系统中：

```
$ sudo apt-get install python-pygame
```

执行如下命令，在终端会话中检查安装情况：

```
$ python
>>> import pygame
>>>
```

如果没有任何输出，就说明Python导入了Pygame，你可以跳到12.3节。

如果你使用的是Python 3，就需要执行两个步骤：安装Pygame依赖的库；下载并安装Pygame。

执行下面的命令来安装Pygame依赖的库（如果你开始终端会话时使用的是命令python3.5，请将python3-dev替换为python3.5-dev）：

```
$ sudo apt-get install python3-dev mercurial
$ sudo apt-get install libsdl-image1.2-dev libsdl2-dev libsdl-ttf2.0-dev
```

这将安装运行《外星人入侵》时需要的库。如果你要启用Pygame的一些高级功能，如添加声音的功能，可安装下面这些额外的库：

```
$ sudo apt-get install libsdl-mixer1.2-dev libportmidi-dev
$ sudo apt-get install libswscale-dev libsmpeg-dev libavformat-dev libavcode-dev
$ sudo apt-get install python-numpy
```

接下来，执行下面的命令来安装Pygame（如有必要，将pip替换为pip3）：

```
$ pip install --user hg+http://bitbucket.org/pygame/pygame
```

告知你Pygame找到了哪些库后，输出将暂停一段时间。请按回车键，即便有一些库没有找到。你将看到一条消息，说明成功地安装了Pygame。

要确认安装成功，请启动一个Python终端会话，并尝试执行下面的命令来导入Pygame：

```
$ python3
>>> import pygame
>>>
```

如果导入成功，请跳到12.3节。

12.2.3 在 OS X 系统中安装 Pygame

要安装Pygame依赖的有些包，需要Homebrew。如果你没有安装Homebrew，请参阅附录A的说明。

为安装Pygame依赖的库，请执行下面的命令：

```
$ brew install hg sdl sdl_image sdl_ttf
```

这将安装运行游戏《外星人入侵》所需的库。每安装一个库后，输出都会向上滚动。

如果你还想启用较高级的功能，如在游戏中包含声音，可安装下面两个额外的库：

```
$ brew install sdl_mixer portmidi
```

使用下面的命令来安装Pygame（如果你运行的是Python 2.7，请将`pip3`替换为`pip`）：

```
$ pip3 install --user hg+http://bitbucket.org/pygame/pygame
```

启动一个Python终端会话，并导入Pygame以检查安装是否成功（如果你运行的是Python 2.7，请将`python3`替换为`python`）：

```
$ python3
>>> import pygame
>>>
```

如果导入成功，请跳到12.3节。

12.2.4 在 Windows 系统中安装 Pygame

Pygame项目托管在代码分享网站Bitbucket中。要在Windows系统中安装Pygame，请访问https://bitbucket.org/pygame/pygame/downloads/，查找与你运行的Python版本匹配的Windows安装程序。如果在Bitbucket上找不到合适的安装程序，请去http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame看看。

下载合适的文件后，如果它是.exe文件，就运行它。

如果该文件的扩展名为.whl，就将它复制到你的项目文件夹中。再打开一个命令窗口，切换到该文件所在的文件夹，并使用pip来运行它：

```
> python -m pip install --user pygame-1.9.2a0-cp35-none-win32.whl
```

12.3 开始游戏项目

现在来开始开发游戏《外星人入侵》。首先创建一个空的Pygame窗口，供后面用来绘制游戏元素，如飞船和外星人。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 窗口以及响应用户输入

首先，我们创建一个空的Pygame窗口。使用Pygame编写的游戏的基本结构如下：

alien_invasion.py

```
import sys

import pygame

def run_game():
    # 初始化游戏并创建一个屏幕对象
❶   pygame.init()
❷   screen = pygame.display.set_mode((1200, 800))
    pygame.display.set_caption("Alien Invasion")

    # 开始游戏的主循环
❸   while True:

        # 监视键盘和鼠标事件
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()

run_game()
```

首先，我们导入了模块sys和pygame。模块pygame包含开发游戏所需的功能。玩家退出时，我们将使用模块sys来退出游戏。

游戏《外星人入侵》的开头是函数run_game()。❶处的代码行pygame.init()初始化背景设置，让Pygame能够正确地工作。在❷处，我们调用pygame.display.set_mode()来创建一个名为screen的显示窗口，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。通过将这些尺寸值传递给pygame.display.set_mode()，我们创建了一个宽1200像素、高800像素的游戏窗口（你可以根据自己的显示器尺寸调整这些值）。

对象screen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。我们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。

这个游戏由一个while循环（见❸）控制，其中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。为让程序响应事件，我们编写一个事件循环，以侦听事件，并根据发生的事件执行相应的任务。❹处的for循环就是一个事件循环。

为访问Pygame检测到的事件，我们使用方法pygame.event.get()。所有键盘和鼠标事件都将促使for循环运行。在这个循环中，我们将编写一系列的if语句来检测并响应特定的事件。例如，玩家单击游戏窗口的关闭按钮时，将检测到pygame.QUIT事件，而我们调用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。在这里，它在每次执行while循环时都绘制一个空屏幕，并擦去旧屏幕，使得只有新屏幕可见。在我们移动游戏元素时，pygame.display.flip()将不断更新屏幕，以显示元素的新位置，并在原来的位置隐藏元素，从而营造平滑移动的效果。

在这个基本的游戏结构中，最后一行调用run_game()，这将初始化游戏并开始主循环。

如果此时运行这些代码，你将看到一个空的Pygame窗口。

12.3.2 设置背景色

Pygame默认创建一个黑色屏幕，这太乏味了。下面来将背景设置为另一种颜色：

alien_invasion.py

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

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

    # 开始游戏主循环。
    while True:

        # 监听键盘和鼠标事件
        --snip--

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

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

run_game()
```

首先，我们创建了一种背景色，并将其存储在bg_color中（见❶）。该颜色只需指定一次，因此我们在进入主while循环前定义它。

在Pygame中，颜色是以RGB值指定的。这种颜色由红色、绿色和蓝色值组成，其中每个值的可能取值范围都为0~255。颜色值(255, 0, 0)表示红色，(0, 255, 0)表示绿色，而(0, 0, 255)表示蓝色。通过组合不同的RGB值，可创建1600万种颜色。在颜色值(230, 230, 230)中，红色、蓝色和绿色量相同，它将背景设置为一种浅灰色。

在❷处，我们调用方法screen.fill()，用背景色填充屏幕；这个方法只接受一个实参：一种颜色。

12.3.3　创建设置类

每次给游戏添加新功能时，通常也将引入一些新设置。下面来编写一个名为settings的模块，其中包含一个名为Settings的类，用于将所有设置存储在一个地方，以免在代码中到处添加设置。这样，我们就能传递一个设置对象，而不是众多不同的设置。另外，这让函数调用更简单，且在项目增大时修改游戏的外观更容易：要修改游戏，只需修改settings.py中的一些值，而无需查找散布在文件中的不同设置。

下面是最初的Settings类：

settings.py

```
class Settings():
    """存储《外星人入侵》的所有设置的类"""

    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)
```

为创建Settings实例并使用它来访问设置，将alien_invasion.py修改成下面这样：

alien_invasion.py

```
--snip--
import pygame

from settings import Settings

def run_game():
    # 初始化pygame、设置和屏幕对象
    pygame.init()
❶   ai_settings = Settings()
❷   screen = pygame.display.set_mode(
        (ai_settings.screen_width, ai_settings.screen_height))
    pygame.display.set_caption("Alien Invasion")

    # 开始游戏主循环
    while True:
        --snip--
        # 每次循环时都重绘屏幕
❸       screen.fill(ai_settings.bg_color)

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

run_game()
```

在主程序文件中，我们导入Settings类，调用pygame.init()，再创建一个Settings实例，并将其存储在变量ai_settings中（见❶）。创建屏幕时（见❷），使用了ai_settings的属性screen_width和screen_height；接下来填充屏幕时，也使用了ai_settings来访问背景色（见❸）。

12.4 添加飞船图像

下面将飞船加入到游戏中。为了在屏幕上绘制玩家的飞船，我们将加载一幅图像，再使用Pygame方法blit()绘制它。

为游戏选择素材时，务必要注意许可。最安全、最不费钱的方式是使用http://pixabay.com/等网站提供的图形，这些图形无需许可，你可以对其进行修改。

在游戏中几乎可以使用任何类型的图像文件，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。大多数图像都为.jpg、.png或.gif格式，但可使用Photoshop、GIMP和Paint等工具将其转换为位图。

选择图像时，要特别注意其背景色。请尽可能选择背景透明的图像，这样可使用图像编辑器将其背景设置为任何颜色。图像的背景色与游戏的背景色相同时，游戏看起来最漂亮；你也可以将游戏的背景色设置成与图像的背景色相同。

就游戏《外星人入侵》而言，你可以使用文件ship.bmp（如图12-1所示），这个文件可在本书的配套资源（https://www.nostarch.com/pythoncrashcourse/）中找到。这个文件的背景色与这个项目使用的设置相同。请在主项目文件夹（alien_invasion）中新建一个文件夹，将其命名为images，并将文件ship.bmp保存到这个文件夹中。

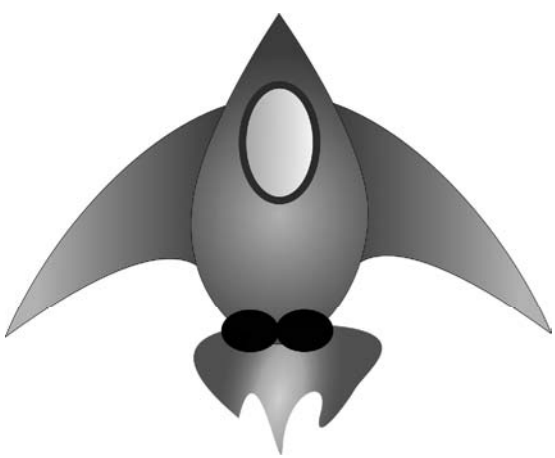


图12-1 游戏《外星人入侵》中的飞船

12.4.1　创建 Ship 类

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

ship.py

```
import pygame

class Ship():

    def __init__(self, screen):
        """初始化飞船并设置其初始位置"""
        self.screen = screen

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

        # 将每艘新飞船放在屏幕底部中央
❹       self.rect.centerx = self.screen_rect.centerx
        self.rect.bottom = self.screen_rect.bottom

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

首先，我们导入了模块pygame。Ship的方法__init__()接受两个参数：引用self和screen，其中后者指定了要将飞船绘制到什么地方。为加载图像，我们调用了pygame.image.load()（见❶）。这个函数返回一个表示飞船的surface，而我们将这个surface存储到了self.image中。

加载图像后，我们使用get_rect()获取相应surface的属性rect（见❷）。Pygame的效率之所以如此高，一个原因是它让你能够像处理矩形（rect对象）一样处理游戏元素，即便它们的形状并非矩形。像处理矩形一样处理游戏元素之所以高效，是因为矩形是简单的几何形状。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。

处理rect对象时，可使用矩形四角和中心的*x*和*y*坐标。可通过设置这些值来指定矩形的位置。

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，可使用属性top、bottom、left或right；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*和*y*坐标。这些属性让你无需去做游戏开发人员原本需要手工完成的计算，你经常会用到这些属性。

注意　在Pygame中，原点(0, 0)位于屏幕左上角，向右下方移动时，坐标值将增大。在1200×800的屏幕上，原点位于左上角，而右下角的坐标为(1200, 800)。

我们将把飞船放在屏幕底部中央。为此，首先将表示屏幕的矩形存储在self.screen_rect中（见❸），再将self.rect.centerx（飞船中心的*x*坐标）设置为表示屏幕的矩形的属性centerx（见❹），并将self.rect.bottom（飞船下边缘的*y*坐标）设置为表示屏幕的矩形的属性bottom。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。

在❺处，我们定义了方法blitme()，它根据self.rect指定的位置将图像绘制到屏幕上。

12.4.2 在屏幕上绘制飞船

下面来更新alien_invasion.py，使其创建一艘飞船，并调用其方法blitme()：

alien_invasion.py

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

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

    # 创建一艘飞船
❶   ship = Ship(screen)

    # 开始游戏主循环
    while True:
        --snip--
        # 每次循环时都重绘屏幕
        screen.fill(ai_settings.bg_color)
❷       ship.blitme()

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

run_game()
```

我们导入Ship类，并在创建屏幕后创建一个名为ship的Ship实例。必须在主while循环前面创建该实例（见❶），以免每次循环时都创建一艘飞船。填充背景后，我们调用ship.blitme()将飞船绘制到屏幕上，确保它出现在背景前面（见❷）。

现在如果运行alien_invasion.py，将看到飞船位于空游戏屏幕底部中央，如图12-2所示。

图12-2　游戏《外星人入侵》屏幕底部中央有一艘飞船

12.5　重构：模块 game_functions

在大型项目中，经常需要在添加新代码前重构既有代码。重构旨在简化既有代码的结构，使其更容易扩展。在本节中，我们将创建一个名为game_functions的新模块，它将存储大量让游戏《外星人入侵》运行的函数。通过创建模块game_functions，可避免alien_invasion.py太长，并使其逻辑更容易理解。

12.5.1　函数 check_events()

我们将首先把管理事件的代码移到一个名为check_events()的函数中，以简化run_game()并隔离事件管理循环。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。

将check_events()放在一个名为game_functions的模块中：

game_functions.py

```
import sys

import pygame

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

这个模块中导入了事件检查循环要使用的sys和pygame。当前，函数check_events()不需要任何形参，其函数体复制了alien_invasion.py的事件循环。

下面来修改alien_invasion.py，使其导入模块game_functions，并将事件循环替换为对函数check_events()的调用：

alien_invasion.py

```
import pygame

from settings import Settings
from ship import Ship
import game_functions as gf

def run_game():
    --snip--
    # 开始游戏主循环
    while True:
        gf.check_events()

        # 让最近绘制的屏幕可见
        --snip--
```

在主程序文件中，不再需要直接导入sys，因为当前只在模块game_functions中使用了它。出于简化的目的，我们给导入的模块game_functions指定了别名gf。

12.5.2 函数 update_screen()

为进一步简化run_game()，下面将更新屏幕的代码移到一个名为update_screen()的函数中，并将这个函数放在模块game_functions.py中：

game_functions.py

```
--snip--

def check_events():
    --snip--

def update_screen(ai_settings, screen, ship):
    """更新屏幕上的图像，并切换到新屏幕"""
    # 每次循环时都重绘屏幕
    screen.fill(ai_settings.bg_color)
    ship.blitme()

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

新函数update_screen()包含三个形参：ai_settings、screen和ship。现在需要将alien_invasion.py的while循环中更新屏幕的代码替换为对函数update_screen()的调用：

alien_invasion.py

```
--snip--
    # 开始游戏主循环
    while True:
        gf.check_events()
        gf.update_screen(ai_settings, screen, ship)

run_game()
```

这两个函数让while循环更简单，并让后续开发更容易：在模块game_functions而不是run_game()中完成大部分工作。

鉴于我们一开始只想使用一个文件，因此没有立刻引入模块game_functions。这让你能够了解实际的开发过程：一开始将代码编写得尽可能简单，并在项目越来越复杂时进行重构。

对代码进行重构使其更容易扩展后，可以开始处理游戏的动态方面了！

动手试一试

12-1 蓝色天空：创建一个背景为蓝色的 Pygame 窗口。

12-2 游戏角色：找一幅你喜欢的游戏角色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背景色，或将屏幕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。为此，我们将编写代码，在用户按左或右箭头键时作出响应。我们将首先专注于向右移动，再使用同样的原理来控制向左移动。通过这样做，你将学会如何控制屏幕图像的移动。

12.6.1　响应按键

每当用户按键时，都将在Pygame中注册一个事件。事件都是通过方法pygame.event.get()获取的，因此在函数check_events()中，我们需要指定要检查哪些类型的事件。每次按键都被注册为一个KEYDOWN事件。

检测到KEYDOWN事件时，我们需要检查按下的是否是特定的键。例如，如果按下的是右箭头键，我们就增大飞船的rect.centerx值，将飞船向右移动：

game_ functions.py

```
def check_events(ship):
    """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               ship.rect.centerx += 1
```

我们在函数check_events()中包含形参ship，因为玩家按右箭头键时，需要将飞船向右移动。在函数check_events()内部，我们在事件循环中添加了一个elif代码块，以便在Pygame 检测到KEYDOWN事件时作出响应（见❶）。我们读取属性event.key，以检查按下的是否是右箭头键（pygame.K_RIGHT）（见❷）。如果按下的是右箭头键，就将ship.rect.centerx的值加1，从而将飞船向右移动（见❸）。

在alien_invasion.py中，我们需要更新调用的check_events()代码，将ship作为实参传递给它：

alien_invasion.py

```
# 开始游戏主循环
while True:
    gf.check_events(ship)
    gf.update_screen(ai_settings, screen, ship)
```

如果现在运行alien_invasion.py，则每按右箭头键一次，飞船都将向右移动1像素。这是一个开端，但并非控制飞船的高效方式。下面来改进控制方式，允许持续移动。

12.6.2 允许不断移动

玩家按住右箭头键不放时，我们希望飞船不断地向右移动，直到玩家松开为止。我们将让游戏检测pygame.KEYUP事件，以便玩家松开右箭头键时我们能够知道这一点；然后，我们将结合使用KEYDOWN和KEYUP事件，以及一个名为moving_right的标志来实现持续移动。

飞船不动时，标志moving_right将为False。玩家按下右箭头键时，我们将这个标志设置为True；而玩家松开时，我们将这个标志重新设置为False。

飞船的属性都由Ship类控制，因此我们将给这个类添加一个名为moving_right的属性和一个名为update()的方法。方法update()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。每当需要调整飞船的位置时，我们都调用这个方法。

下面是对Ship类所做的修改：

ship.py

```
class Ship():

    def __init__(self, screen):
        --snip--
        # 将每艘新飞船放在屏幕底部中央
```

```
        self.rect.centerx = self.screen_rect.centerx
        self.rect.bottom = self.screen_rect.bottom

        # 移动标志
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.centerx += 1

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

在方法__init__()中，我们添加了属性self.moving_right，并将其初始值设置为False（见❶）。接下来，我们添加了方法update()，它在前述标志为True时向右移动飞船（见❷）。

下面来修改check_events()，使其在玩家按下右箭头键时将moving_right设置为True，并在玩家松开时将moving_right设置为False：

game_functions.py

```
def check_events(ship):
    """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               ship.moving_right = True

❷       elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                ship.moving_right = False
```

在❶处，我们修改了游戏在玩家按下右箭头键时响应的方式：不直接调整飞船的位置，而只是将moving_right设置为True。在❷处，我们添加了一个新的elif代码块，用于响应KEYUP事件：玩家松开右箭头键（K_RIGHT）时，我们将moving_right设置为False。

最后，我们需要修改alien_invasion.py中的while循环，以便每次执行循环时都调用飞船的方法update()：

alien_invasion.py

```
    # 开始游戏主循环
    while True:
        gf.check_events(ship)
        ship.update()
        gf.update_screen(ai_settings, screen, ship)
```

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，玩家输入时，飞船的位置将更新，从而确保使用更新后的位置将飞船绘制到屏幕上。

如果你现在运行alien_invasion.py并按住右箭头键，飞船将不断地向右移动，直到你松开为止。

12.6.3 左右移动

飞船能够不断地向右移动后，添加向左移动的逻辑很容易。我们将再次修改Ship类和函数check_events()。下面显示了对Ship类的方法__init__()和update()所做的相关修改：

ship.py

```
    def __init__(self, screen):
        --snip--
        # 移动标志
        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.centerx += 1
        if self.moving_left:
            self.rect.centerx -= 1
```

在方法__init__()中，我们添加了标志self.moving_left；在方法update()中，我们添加了一个if代码块而不是elif代码块，这样如果玩家同时按下了左右箭头键，将先增大飞船的rect.centerx值，再降低这个值，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，右箭头键将始终处于优先地位。从向左移动切换到向右移动时，玩家可能同时按住左右箭头键，在这种情况下，前面的做法让移动更准确。

我们还需对check_events()作两方面的调整：

game_functions.py

```
def check_events(ship):
    """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:
                ship.moving_right = True
            elif event.key == pygame.K_LEFT:
                ship.moving_left = True

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

如果因玩家按下K_LEFT键而触发了KEYDOWN事件，我们就将moving_left设置为True；如果因玩家松开K_LEFT而触发了KEYUP事件，我们就将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；如果玩家同时按下了左右箭头键，将检测到两个不同的事件。

如果此时运行alien_invasion.py，将能够不断地左右移动飞船；如果你同时按左右箭头键，飞船将纹丝不动。

下面来进一步优化飞船的移动方式：调整飞船的速度；限制飞船的移动距离，以免它移到屏幕外面去。

12.6.4　调整飞船的速度

当前，每次执行while循环时，飞船最多移动1像素，但我们可以在Settings类中添加属性ship_speed_factor，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少距离。下面演示了如何在settings.py中添加这个新属性：

settings.py

```
class Settings():
    """一个存储游戏《外星人入侵》的所有设置的类"""

    def __init__(self):
        --snip--

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

我们将ship_speed_factor的初始值设置成了1.5。需要移动飞船时，我们将移动1.5像素而不是1像素。

通过将速度设置指定为小数值，可在后面加快游戏的节奏时更细致地控制飞船的速度。然而，rect的centerx等属性只能存储整数值，因此我们需要对Ship类做些修改：

ship.py

```
class Ship():

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

        # 将每艘新飞船放在屏幕底部中央
        --snip--

        # 在飞船的属性center中存储小数值
❸       self.center = float(self.rect.centerx)

        # 移动标志
```

```
        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center值，而不是rect
        if self.moving_right:
❹           self.center += self.ai_settings.ship_speed_factor
        if self.moving_left:
            self.center -= self.ai_settings.ship_speed_factor

        # 根据self.center更新rect对象
❺       self.rect.centerx = self.center

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

在❶处，我们在__init__()的形参列表中添加了ai_settings，让飞船能够获取其速度设置。接下来，我们将形参ai_settings的值存储在一个属性中，以便能够在update()中使用它（见❷）。鉴于现在调整飞船的位置时，将增加或减去一个单位为像素的小数值，因此需要将位置存储在一个能够存储小数值的变量中。可以使用小数来设置rect的属性，但rect将只存储这个值的整数部分。为准确地存储飞船的位置，我们定义了一个可存储小数值的新属性self.center（见❸）。我们使用函数float()将self.rect.centerx的值转换为小数，并将结果存储到self.center中。

现在在update()中调整飞船的位置时，将self.center的值增加或减去ai_settings.ship_speed_factor的值（见❹）。更新self.center后，我们再根据它来更新控制飞船位置的self.rect.centerx（见❺）。self.rect.centerx将只存储self.center的整数部分，但对显示飞船而言，这问题不大。

在alien_invasion.py中创建Ship实例时，需要传入实参ai_settings：

alien_invasion.py

```
--snip--
def run_game():
    --snip--
    # 创建飞船
    ship = Ship(ai_settings, screen)
    --snip--
```

现在，只要ship_speed_factor的值大于1，飞船的移动速度就会比以前更快。这有助于让飞船的反应速度足够快，能够将外星人射下来，还让我们能够随着游戏的进行加快游戏的节奏。

12.6.5 限制飞船的活动范围

当前，如果玩家按住箭头键的时间足够长，飞船将移到屏幕外面，消失得无影无踪。下面来修复这种问题，让飞船到达屏幕边缘后停止移动。为此，我们将修改Ship类的方法update()：

ship.py

```
    def update(self):
        """根据移动标志调整飞船的位置"""
        # 更新飞船的center值，而不是rect
❶       if self.moving_right and self.rect.right < self.screen_rect.right:
            self.center += self.ai_settings.ship_speed_factor
❷       if self.moving_left and self.rect.left > 0:
            self.center -= self.ai_settings.ship_speed_factor

        # 根据self.center更新rect对象
        self.rect.centerx = self.center
```

上述代码在修改self.center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.center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。

12.6.6　重构 check_events()

随着游戏开发的进行，函数check_events()将越来越长，我们将其部分代码放在两个函数中：一个处理KEYDOWN事件，另一个处理KEYUP事件：

game_functions.py

```
def check_keydown_events(event, ship):
    """响应按键"""
    if event.key == pygame.K_RIGHT:
        ship.moving_right = True
    elif event.key == pygame.K_LEFT:
        ship.moving_left = True

def check_keyup_events(event, ship):
    """响应松开"""
    if event.key == pygame.K_RIGHT:
        ship.moving_right = False
    elif event.key == pygame.K_LEFT:
        ship.moving_left = False

def check_events(ship):
    """响应按键和鼠标事件"""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            check_keydown_events(event, ship)
        elif event.type == pygame.KEYUP:
            check_keyup_events(event, ship)
```

我们创建了两个新函数：check_keydown_events()和check_keyup_events()，它们都包含形参event和ship。这两个函数的代码是从check_events()中复制而来的，因此我们将函数check_events中相应的代码替换成了对这两个函数的调用。现在，函数check_events()更简单，代码结构更清晰。这样，在其中响应其他玩家输入时将更容易。

12.7 简单回顾

下一节将添加射击功能，这需要新增一个名为bullet.py的文件，并对一些既有文件进行修改。当前，我们有四个文件，其中包含很多类、函数和方法。添加其他功能之前，为让你清楚这个项目的组织结构，先来回顾一下这些文件。

12.7.1 alien_invasion.py

主文件alien_invasion.py创建一系列整个游戏都要用到的对象：存储在ai_settings中的设置、存储在screen中的主显示surface以及一个飞船实例。文件alien_invasion.py还包含游戏的主循环，这是一个调用check_events()、ship.update()和update_screen()的while循环。

要玩游戏《外星人入侵》，只需运行文件alien_invasion.py。其他文件（settings.py、game_functions.py、ship.py）包含的代码被直接或间接地导入到这个文件中。

12.7.2 settings.py

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

12.7.3 game_functions.py

文件game_functions.py包含一系列函数，游戏的大部分工作都是由它们完成的。函数check_events()检测相关的事件，如按键和松开，并使用辅助函数check_keydown_events()和check_keyup_events()来处理这些事件。就目前而言，这些函数管理飞船的移动。模块game_functions还包含函数update_screen()，它用于在每次执行主循环时都重绘屏幕。

12.7.4 ship.py

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

动手试一试

12-3 火箭：编写一个游戏，开始时屏幕中央有一个火箭，而玩家可使用四个方向键上下左右移动火箭。请务必确保火箭不会移到屏幕外面。

12-4 按键：创建一个程序，显示一个空屏幕。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，并按各种键，看看Pygame如何响应。

12.8 射击

下面来添加射击功能。我们将编写玩家按空格键时发射子弹（小矩形）的代码。子弹将在屏幕中向上穿行，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，在其方法__init__()末尾存储新类Bullet所需的值：

settings.py

```
    def __init__(self):
        --snip--
        # 子弹设置
        self.bullet_speed_factor = 1
        self.bullet_width = 3
        self.bullet_height = 15
        self.bullet_color = 60, 60, 60
```

这些设置创建宽3像素、高15像素的深灰色子弹。子弹的速度比飞船稍低。

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):
    """一个对飞船发射的子弹进行管理的类"""

    def __init__(self, ai_settings, screen, ship):
        """在飞船所处的位置创建一个子弹对象"""
        super(Bullet, self).__init__()
        self.screen = screen

        # 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, ai_settings.bullet_width,
            ai_settings.bullet_height)
❷       self.rect.centerx = ship.rect.centerx
❸       self.rect.top = ship.rect.top
```

```
        #存储用小数表示的子弹位置
❹       self.y = float(self.rect.y)

❺       self.color = ai_settings.bullet_color
        self.speed_factor = ai_settings.bullet_speed_factor
```

Bullet类继承了我们从模块pygame.sprite中导入的Sprite类。通过使用精灵，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。为创建子弹实例，需要向__init__()传递ai_settings、screen和ship实例，还调用了super()来继承Sprite。

注意 代码super(Bullet, self).__init__()使用了Python 2.7语法。这种语法也适用于Python 3，但你也可以将这行代码简写为super().__init__()。

在❶处，我们创建了子弹的属性rect。子弹并非基于图像的，因此我们必须使用pygame.Rect()类从空白开始创建一个矩形。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，但接下来的两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ai_settings中获取的。

在❷处，我们将子弹的centerx设置为飞船的rect.centerx。子弹应从飞船顶部射出，因此我们将表示子弹的rect的top属性设置为飞船的rect的top属性，让子弹看起来像是从飞船中射出的（见❸）。

我们将子弹的*y*坐标存储为小数值，以便能够微调子弹的速度（见❹）。在❺处，我们将子弹的颜色和速度设置分别存储到self.color和self.speed_factor中。

下面是bullet.py的第二部分——方法update()和draw_bullet()：

bullet.py

```
    def update(self):
        """向上移动子弹"""
        #更新表示子弹位置的小数值
❶       self.y -= self.speed_factor
        #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)
```

方法update()管理子弹的位置。发射出去后，子弹在屏幕中向上移动，这意味着*y*坐标将不断减小，因此为更新子弹的位置，我们从self.y中减去self.speed_factor的值（见❶）。接下来，我们将self.rect.y设置为self.y的值（见❷）。属性speed_factor让我们能够随着游戏的进行或根据需要提高子弹的速度，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地往上穿行。

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

12.8.3　将子弹存储到编组中

定义Bullet类和必要的设置后，就可以编写代码了，在玩家每次按空格键时都射出一发子弹。首先，我们将在alien_invasion.py中创建一个编组（group），用于存储所有有效的子弹，以便能够管理发射出去的所有子弹。这个编组将是pygame.sprite.Group类的一个实例；pygame.sprite.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，我们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：

alien_invasion.py

```
import pygame
from pygame.sprite import Group
--snip--

def run_game():
    --snip--
    # 创建一艘飞船
    ship = Ship(ai_settings, screen)
    # 创建一个用于存储子弹的编组
❶   bullets = Group()

    # 开始游戏主循环
    while True:
        gf.check_events(ai_settings, screen, ship, bullets)
        ship.update()
❷       bullets.update()
        gf.update_screen(ai_settings, screen, ship, bullets)

run_game()
```

我们导入了pygame.sprite中的Group类。在❶处，我们创建了一个Group实例，并将其命名为bullets。这个编组是在while循环外面创建的，这样就无需每次运行该循环时都创建一个新的子弹编组。

注意　如果在循环内部创建这样的编组，游戏运行时将创建数千个子弹编组，导致游戏慢得像蜗牛。如果游戏停滞不前，请仔细查看主while循环中发生的情况。

我们将bullets传递给了check_events()和update_screen()。在check_events()中，需要在玩家按空格键时处理bullets；而在update_screen()中，需要更新要绘制到屏幕上的bullets。

当你对编组调用update()时，编组将自动对其中的每个精灵调用update()，因此代码行bullets.update()将为编组bullets中的每颗子弹调用bullet.update()。

12.8.4 开火

在game_functions.py中，我们需要修改check_keydown_events()，以便在玩家按空格键时发射一颗子弹。我们无需修改check_keyup_events()，因为玩家松开空格键时什么都不会发生。我们还需修改update_screen()，确保在调用flip()前在屏幕上重绘每颗子弹。下面是对game_functions.py所做的相关修改：

game_functions.py

```
--snip--
from bullet import Bullet

❶ def check_keydown_events(event, ai_settings, screen, ship, bullets):
    --snip--
❷     elif event.key == pygame.K_SPACE:
        # 创建一颗子弹，并将其加入到编组bullets中
        new_bullet = Bullet(ai_settings, screen, ship)
        bullets.add(new_bullet)
      --snip--

❸ def check_events(ai_settings, screen, ship, bullets):
    """响应按键和鼠标事件"""
    for event in pygame.event.get():
        --snip--
        elif event.type == pygame.KEYDOWN:
            check_keydown_events(event, ai_settings, screen, ship, bullets)
        --snip--

❹ def update_screen(ai_settings, screen, ship, bullets):
    --snip--
    # 在飞船和外星人后面重绘所有子弹
❺     for bullet in bullets.sprites():
        bullet.draw_bullet()
    ship.blitme()
    --snip--
```

编组bulltes传递给了check_keydown_events()（见❶）。玩家按空格键时，创建一颗新子弹（一个名为new_bullet的Bullet实例），并使用方法add()将其加入到编组bullets中（见❷）；代码bullets.add(new_bullet)将新子弹存储到编组bullets中。

在check_events()的定义中，我们需要添加形参bullets（见❸）；调用check_keydown_events()时，我们也需要将bullets作为实参传递给它。

在❹处，我们给在屏幕上绘制子弹的update_screen()添加了形参bullets。方法bullets.sprites()返回一个列表，其中包含编组bullets中的所有精灵。为在屏幕上绘制发射的所有子弹，我们遍历编组bullets中的精灵，并对每个精灵都调用draw_bullet()（见❺）。

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。子弹在屏幕上向上穿行，抵达屏幕顶部后消失，如图12-3所示。可在settings.py中修改子弹的尺寸、颜色和速度。

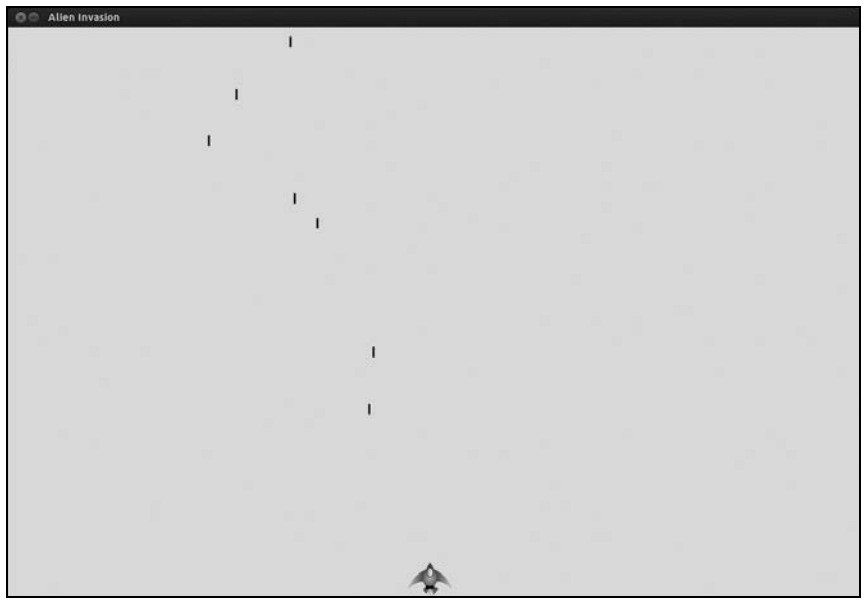


图12-3　飞船发射一系列子弹后的《外星人入侵》游戏

12.8.5　删除已消失的子弹

当前，子弹抵达屏幕顶端后消失，这仅仅是因为Pygame无法在屏幕外面绘制它们。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，我们需要检测这样的条件，即表示子弹的rect的bottom属性为零，它表明子弹已穿过屏幕顶端：

alien_invasion.py

```
    # 开始游戏主循环
    while True:
        gf.check_events(ai_settings, screen, ship, bullets)
        ship.update()
        bullets.update()

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

        gf.update_screen(ai_settings, screen, ship, bullets)
```

在for循环中，不应从列表或编组中删除条目，因此必须遍历编组的副本。我们使用了方法copy()来设置for循环（见❶），这让我们能够在循环中修改bullets。我们检查每颗子弹，看看它是否已从屏幕顶端消失（见❷）。如果是这样，就将其从bullets中删除（见❸）。在❹处，我们使用了一条print语句，以显示当前还有多少颗子弹，从而核实已消失的子弹确实删除了。

如果这些代码没有问题，我们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顶端消失，子弹数将逐渐降为零。运行这个游戏并确认子弹已被删除后，将这条print语句删除。如果你留下这条语句，游戏的速度将大大降低，因为将输出写入到终端而花费的时间比将图形绘制到游戏窗口花费的时间还多。

12.8.6 限制子弹数量

很多射击游戏都对可同时出现在屏幕上的子弹数量进行限制，以鼓励玩家有目标地射击。下面在游戏《外星人入侵》中作这样的限制。

首先，在settings.py中存储所允许的最大子弹数：

settings.py

```
        # 子弹设置
        self.bullet_width = 3
        self.bullet_height = 15
        self.bullet_color = 60, 60, 60
        self.bullets_allowed = 3
```

这将未消失的子弹数限制为3颗。在game_functions.py的check_keydown_events()中，我们在创建新子弹前检查未消失的子弹数是否小于该设置：

game_functions.py

```
def check_keydown_events(event, ai_settings, screen, ship, bullets):
    --snip--
    elif event.key == pygame.K_SPACE:
        # 创建新子弹并将其加入到编组bullets中
        if len(bullets) < ai_settings.bullets_allowed:
            new_bullet = Bullet(ai_settings, screen, ship)
            bullets.add(new_bullet)
```

玩家按空格键时，我们检查bullets的长度。如果len(bullets)小于3，我们就创建一个新子弹；但如果已有3颗未消失的子弹，则玩家按空格键时什么都不会发生。如果你现在运行这个游戏，屏幕上最多只能有3颗子弹。

12.8.7 创建函数 update_bullets()

编写并检查子弹管理代码后，可将其移到模块game_functions中，以让主程序文件alien_invasion.py尽可能简单。我们创建一个名为update_bullets()的新函数，并将其添加到

game_functions.py的末尾：

game_functions.py

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

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

update_bullets()的代码是从alien_invasion.py剪切并粘贴而来的，它只需要一个参数，即编组bullets。

alien_invasion.py中的while循环又变得很简单了：

alien_invasion.py

```
    # 开始游戏主循环
    while True:
❶       gf.check_events(ai_settings, screen, ship, bullets)
❷       ship.update()
❸       gf.update_bullets(bullets)
❹       gf.update_screen(ai_settings, screen, ship, bullets)
```

我们让主循环包含尽可能少的代码，这样只要看函数名就能迅速知道游戏中发生的情况。主循环检查玩家的输入（见❶），然后更新飞船的位置（见❷）和所有未消失的子弹的位置（见❸）。接下来，我们使用更新后的位置来绘制新屏幕（见❹）。

12.8.8　创建函数 fire_bullet()

下面将发射子弹的代码移到一个独立的函数中，这样，在check_keydown_events()中只需使用一行代码来发射子弹，让elif代码块变得非常简单：

game_functions.py

```
def check_keydown_events(event, ai_settings, screen, ship, bullets):
    """响应按键"""
    --snip--
    elif event.key == pygame.K_SPACE:
        fire_bullet(ai_settings, screen, ship, bullets)

def fire_bullet(ai_settings, screen, ship, bullets):
    """如果还没有到达限制，就发射一颗子弹"""
    #创建新子弹，并将其加入到编组bullets中
    if len(bullets) < ai_settings.bullets_allowed:
        new_bullet = Bullet(ai_settings, screen, ship)
        bullets.add(new_bullet)
```

函数fire_bullet()只包含玩家按空格键时用于发射子弹的代码；在check_keydown_events()中，我们在玩家按空格键时调用fire_bullet()。

请再次运行alien_invasion.py，确认发射子弹时依然没有错误。

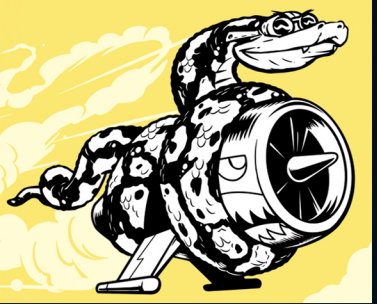
动手试一试

12-5 侧面射击：编写一个游戏，将一艘飞船放在屏幕左边，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穿行的子弹，并在子弹离开屏幕而消失后将其删除。

12.9 小结

在本章中，你学习了：游戏开发计划的制定；使用Pygame编写的游戏的基本结构；如何设置背景色，以及如何将设置存储在可供游戏的各个部分访问的独立类中；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；如何创建自动移动的元素，如在屏幕中向上飞驰的子弹，以及如何删除不再需要的对象；如何定期重构项目的代码，为后续开发提供便利。

在第13章中，我们将在游戏《外星人入侵》中添加外星人。在第13章结束时，你将能够击落外星人——但愿是在他们撞到飞船前！



亚马逊读者评论

“我读过很多本为Python初学者所写的入门书，到目前为止，这是我最爱的一本。这本Python编程书内容全面，讲解详细，编排合理，真是太棒了！”

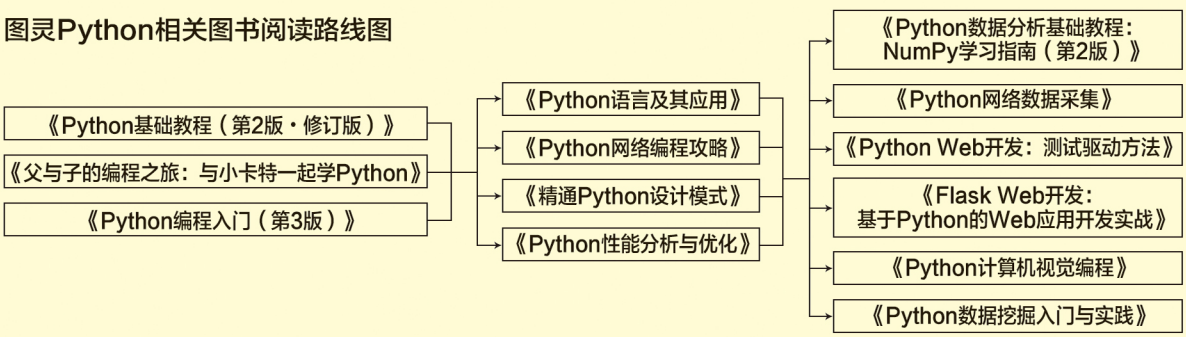
“这本书用平实的语言向初学者解释各种概念，没有过早引入隐晦难懂的技术术语。我至少有8本介绍Python的书，但大多数没读多少页就读不下去了；而阅读这本书的时候，我能更轻松地理解其中的概念并且一直坚持读完。”

上到有编程基础的程序员，下到10岁少年，想入门Python并达到可以开发实际项目的水平，本书是最佳选择！

本书是一本全面的从入门到实践的Python编程教程，带领读者快速掌握编程基础知识，编写出能解决实际问题的代码并开发复杂项目。

书中内容分为基础篇和实战篇两部分。基础篇介绍基本的编程概念，如列表、字典、类和循环，并指导读者编写整洁且易于理解的代码。另外还介绍了如何让程序能够与用户交互，以及如何在代码运行前进行测试。实战篇介绍如何利用新学到的知识开发功能丰富的项目：2D游戏《外星人入侵》，数据可视化实战，Web应用程序。

图灵Python相关图书阅读路线图



no starch press

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/程序设计/Python

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-42802-8

ISBN 978-7-115-42802-8
定价：89.00元