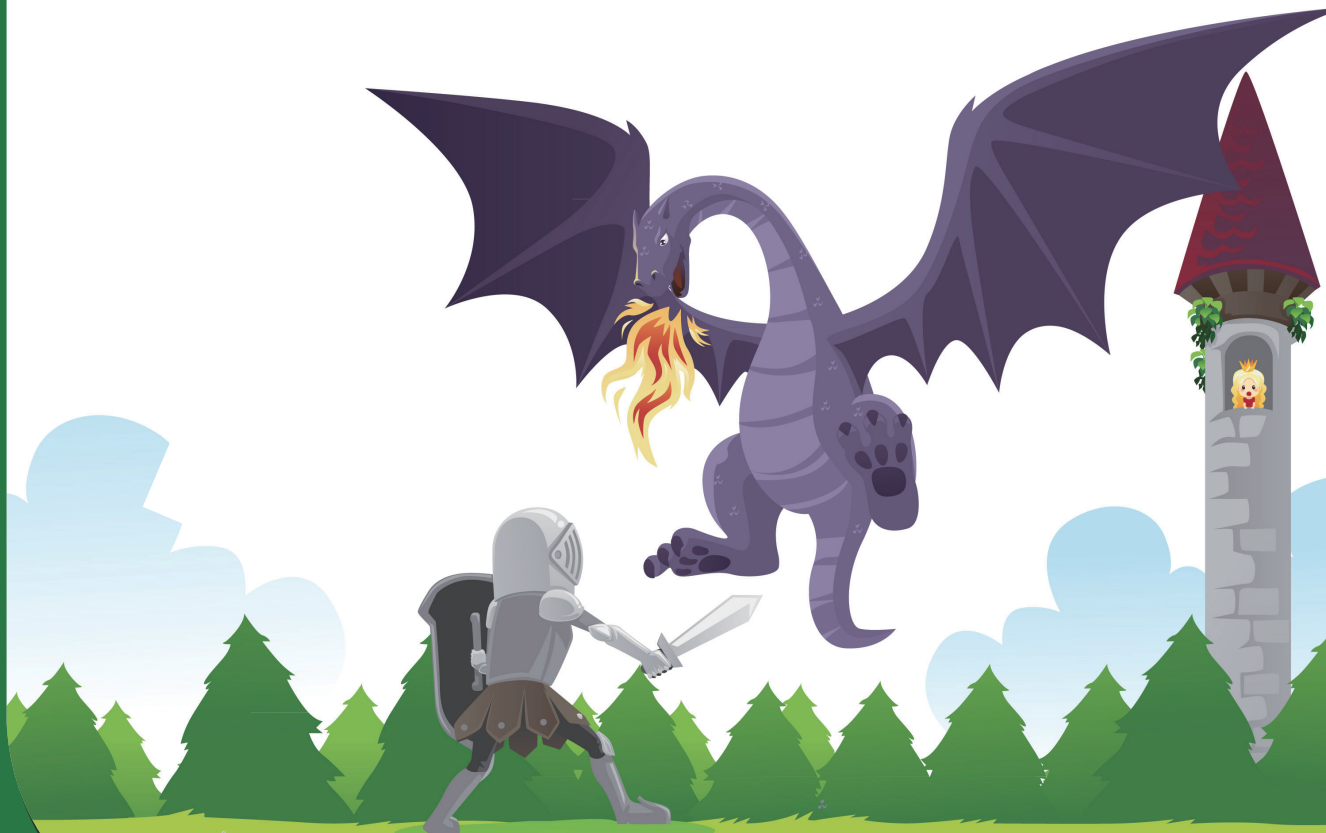
Python游戏编程的初学者指南

Python
游戏编程快速上手

[美] Al Sweigart 著
李强 译

Invent Your Own Computer Games with Python, 3rd Edition



中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Python

游戏编程快速上手

[美] Al Sweigart 著
李强 译

Invent Your Own Computer Games with Python, 3rd Edition

人民邮电出版社
北京

图书在版编目（CIP）数据

Python游戏编程快速上手 / (美) 斯维加特
(Al Sweigart) 著 ; 李强译. -- 北京 : 人民邮电出版
社, 2016.8
ISBN 978-7-115-42903-2

Ⅰ. ①P… Ⅱ. ①斯… ②李… Ⅲ. ①游戏程序－程序
设计 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2016)第157317号

版 权 声 明

◆ 著　　[美] Al Sweigart
译　　李 强
责任编辑　陈冀康
责任印制　焦志炜

◆ 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
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：21.5
字数：472 千字　　2016 年 8 月第 1 版
印数：1 – 2 500 册　　2016 年 8 月河北第 1 次印刷

著作权合同登记号　图字：01-2015-5086 号

定价：59.00 元

读者服务热线：(010)81055410　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

内容提要

Python 是一种高级程序设计语言，因其简洁、易读及可扩展性日渐成为程序设计领域备受推崇的语言。

本书通过编写一个个小巧、有趣的游戏来教授 Python 编程，并且采用了直接展示游戏的源代码并通过实例来解释编程的原理的方式。全书共 21 章，14 个游戏程序和示例贯穿其中，介绍了 Python 基础知识、数据类型、函数、流程控制、程序调试、流程图设计、字符串操作、列表和字典、图形和动画、碰撞检测、声音和图像等方方面面的程序设计知识。本书可以帮助读者在轻松有趣的过程中，掌握 Python 游戏编程的基本技能。

本书适合不同年龄和层次的 Python 编程初学者阅读。

前言

感谢您阅读本书。编写本书最初的动机是，我看到目前很少有能够激发孩子学习编程的兴趣的图书。我最初使用 BASIC 编程语言开始编程，使用的是一本像本书一样的书。

在写作本书的过程中，我意识到，像 Python 这样的现代语言使得编程更加容易，并且为新一代的程序员提供了更多的功能。Python 拥有平缓的学习曲线，而且是供专业程序员使用的一种正规语言。

目前的编程书籍大多分为两种类型。第一种，与其说是教编程的书，倒不如说是在教“游戏制作软件”，或教授使用一种呆板的语言，使得编程“简单”到不再是编程。而第二种，它们就像是教数学课一样教编程：所有的原理和概念都以小的应用程序的方式呈现给读者。本书采用了不同的方式直接展示了游戏的源代码，并且通过实例来解释编程的原理。

本书第 3 版有哪些新的内容？

第 3 版与第 2 版相比，没有增加新的内容。但是，第 3 版精简了 20%的篇幅，却介绍了相同的内容。根据需要，增加了解释，并且讲解得更加清晰。

为了防止混淆，源代码故意与第 2 版保持相同。如果你已经阅读了第 2 版，就没有理由再阅读本书。然而，如果你正打算编程或介绍朋友学习编程，第 3 版会让学习过程更加简单、顺利和有趣。

本书的目标读者

编程并不难。但是教你通过编程来做有趣事情的学习材料却很难找到。有些计算机书籍会介绍很多大部分新手程序员都不需要了解的话题。本书将介绍如何编写自己的计算机游戏。你将学习到有用的技巧和可以展示的有趣游戏。本书的目标读者是：

- 想要学习计算机编程的完全初学者，他们甚至之前没有任何编程经验；
- 想要通过创建游戏来学习编程的儿童和青少年；
- 想要教其他人编程的成年人和教师；
- 任何想要通过学习专业编程语言来学习如何编程的人，无论是年轻人还是老年人。

译者序

对于编程初学者来说，Python 是一种理想的语言。Python 的语法并不复杂，学习起来相对比较简单。没有太多编程经验的人，也可以较为轻松地读懂一个简单的 Python 脚本并理解它在做什么。一旦掌握了 Python 的基础知识，也比较容易从简单的程序转移到更为高级的项目。而且，Python 可以用于很多的领域，从科学计算到游戏开发。即使是程序员新手，也总是能够找到自己感兴趣的项目和应用领域。

市面上已经有很多介绍编程和 Python 语言的图书了。然而，在本书作者的眼里，这些书大致分为两种类型。而这两种类型，似乎与作者心目中教授编程的图书相去甚远，也都不太适合青少年阅读和学习。正因为很少有能够激发青少年学习编程兴趣的图书，作者才萌发了编写本书的想法。

本书专门针对任何想要学习 Python 或初次接触编程的人，特别适合作为不同年龄的初学者的第一本编程学习图书。本书的内容浅显易懂，具有很强的操作性，所有的理论知识都是在游戏实例中循序展开。书中的示例，都是一个一个简单但生动有趣的小游戏。本书对这些游戏程序的设计和代码进行深入细致的讲解和分析，使读者在不知不觉之中就能够掌握 Python 编程的基本知识和方法技巧。一路下来，读者将通过创建游戏来扩展自己的编程技能，并且将所学的知识付诸应用。

我的儿子若瑜上小学一年级，和大多数同龄小朋友一样，他酷爱各种电子游戏。在翻译这本关于游戏编程的书籍的过程中，若瑜表现出极大的热情，他成为第一位读者，每完成一章，他都会通读一次，还会反复地玩书中的游戏；尽管这些游戏很简单，但是对他的吸引力还是很大。他甚至和我约定，今年的生日礼物，就是教会他自己来编写本书中的游戏。感谢若瑜对本书翻译工作的支持和帮助。

我想这样一本能够吸引小朋友的图书，也一定能够让读者朋友感到编程的乐趣，并能从中受益。特别对于打算学习编程的青少年儿童，这是一本不可多得的好书。

虽然付出了很大的精力和心血翻译本书，但难免有不足之处，恳请广大读者谅解，或通过邮件 reejohn@sohu.com 赐教。

目录

第 1 章　安装 Python

本章主要内容：

- 下载并且安装 Python 解释器；
- 如何使用本书；
- 本书的网址：http://inventwithpython.com。

你好！本书介绍了如何编写电子游戏。一旦你了解了本书中的游戏是如何工作的，就能够创建自己的游戏了。你只需要一台计算机、一款叫做 Python 解释器的软件以及这本书。Python 解释器可以从网上免费下载。

当我还是一个孩子时，就是像这样的一本书教会了我如何编写第一个程序和游戏。那本书既有趣又简单。现在，作为一名成年人，我仍然能够享受到编程的乐趣，并且能从中获到回报。但是，即便当你长大成人后并没有成为一名程序员，编程也还是一种有用而且有趣的技能。

计算机是不可思议的机器，并且学习编写计算机程序并不像人们想象中的那样难。如果你能够阅读本书，那么就可以编写计算机程序。计算机程序就是计算机所能够理解的一堆指令，这就像一本故事书就是读者可以读懂的一堆句子一样。既然电子游戏就是计算机程序，那么它们也是由指令组成的。

要对计算机发号施令，就使用计算机能够理解的语言来编写一个程序。本书介绍的是一种叫做 Python 的编程语言。有很多种不同的编程语言，如 BASIC、Java、JavaScript、PHP 和 C++等。

当我还是一个孩子的时候，通常，BASIC 是作为第一门编程语言来学习。然而，此后出现了像 Python 这样的新的编程语言。Python 学起来甚至比 BASIC 还要简单！但是 Python 也是供专业程序员使用的一种正规语言。很多成年人在工作中使用 Python 并以此为乐趣。

在本书中，我们将要创建的游戏看上去比 Xbox、PlayStation 或者 Nintendo 的游戏简单。这些游戏没有绚丽的图案，因为我们要用它们来教授基本的编程知识。我们有意选择这些简单的游戏，以便你可以专注于学习编程。游戏并非复杂才有趣。

1.1　下载和安装 Python

我们需要安装一个叫做 Python 解释器的软件。解释器程序理解我们用 Python 语言编写的指令。从现在开始，我把“Python 解释器软件”直接简称为“Python”。

特别提示！

一定要安装 Python 3 而不是 Python 2。本书中的程序使用的是 Python 3，如果试图使用 Python 2 来运行这些程序，就会出错。这一点非常重要，我在图 1-1 中给出了一个卡通企鹅的形象，来提醒你要安装 Python 3，所以请一定留意这一点。

在 Windows 操作系统中，下载 Python 安装程序（文件名的后缀是.msi），并且双击它。按照安装程序在屏幕上显示的说明来安装 Python，如下所示：

1．选择 Install for All Users，然后单击 Next 按钮。

2．单击 Next 按钮，将程序安装在 C:\Python34 文件夹下。

3．单击 Next 按钮，跳过 Customize Python 部分。

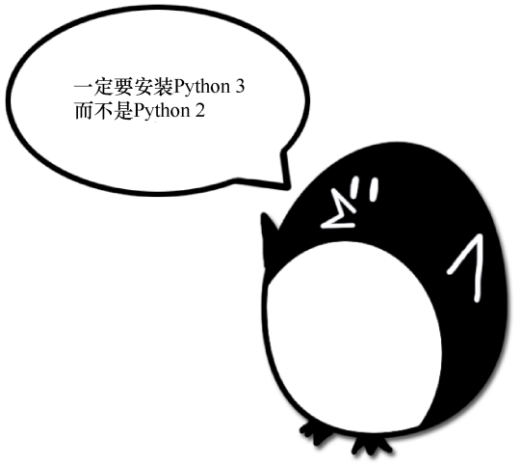


图 1-1　特别显眼的企鹅提醒你要安装 Python3

在 Mac OS X 操作系统中，从网上下载适合你的 OS X 版本的.dmg 文件，并且双击它。按照安装程序在屏幕上显示的说明安装 Python，如下所示：

1．在一个新的窗口中打开 DMG 包时，双击 Python.mpkg 文件。可能需要输入计算机管理员的密码。

2．在 Welcome 部分，单击 Continue 按钮，并且单击 Agree 按钮以接受许可协议。

3．选择 HD Macintosh（或者任意硬盘驱动器名称），并且点击 Install 按钮。

如果你使用的是 Ubuntu 操作系统，可以通过 Ubuntu Software Center，按照如下步骤安装 Python：

1．打开 Ubuntu Software Center。

2．在窗口右上角的搜索框中输入 Python。

3．选择 IDLE（使用 Python 3.4），或者任何最新的版本。

4．单击 Install 按钮。可能需要输入计算机管理员的密码来完成安装。

1.2　启动 IDLE

IDLE 表示交互式开发环境（Interactive DeveLopment Environment）。对于编写 Python 程序来说，这个开发环境就像是字处理软件一样。在不同的操作系统上，启动 IDLE 的方式有所不同。

在 Windows 操作系统中，单击左下角的启动按钮，输入“IDLE”并且选择 IDLE

(Python GUI)。

在 Mac OS X 操作系统中,打开 Finder 窗口,点击 Applications。接下来单击 Python 3.4。然后单击 IDLE 图标。

在 Ubuntu 或者 Linux 操作系统中，打开一个终端窗口，然后输入“idle3”。也可以单击屏幕上端的 Applications。然后单击 Programming 和 IDLE 3。

当第一次运行 IDLE 时，出现的窗口是交互式的 shell，如图 1-2 所示。你可以在交互式 shell 的>>>提示符后输入 Python 指令，Python 就会执行这些指令。显示完执行指令的结果之后，会出现一个新的>>>提示符，并等待下一条指令。

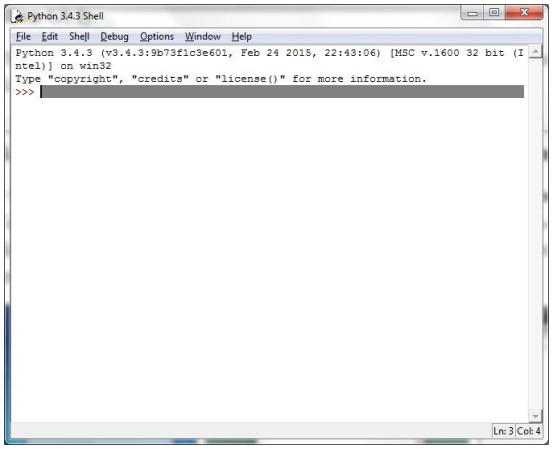


图 1-2 Windows、OS X 和 Ubuntu Linux 操作系统的 IDLE 程序的交互式 shell

1.3 如何使用本书

本书的大多数的章，一开始都会给出一个程序示例，并展示运行这个示例程序的样子。程序中用户输入的部分会用粗体字表示。

请在 IDLE 文件编辑器中自行输入代码，而不是下载或者复制/粘贴代码。花些时间录入代码，这会帮助你更好地记住如何编程。

行号和空格

当按照本书中内容录入源代码时，不要录入每一行开头处的行号。例如，你会看到书中代码如下所示：

```
9. number = random.randint(1, 20)
```

不要录入左边的“9.”，以及紧随其后的空格。只需要录入如下内容即可：

```
number = random.randint(1, 20)
```

这些编号只是为了便于在书中引用特定的代码行。它们并非实际程序的一部分。

除了行号之外，确保严格按照代码的样子进行录入。注意，其中一些代码行缩进了 4 个或 8 个空格。IDLE 中的每个字符是等宽的，所以你可以通过数一数上一行和下一行的字符数来统计空格的数量。

例如，用一个点表示缩进空格，看上去如下所示：

```
while guesses < 10:
....if number == 42:
........print('Hello')
```

图书中的文本折行

一些代码行太长了，无法在图书页面中放到一行之中，那么，代码的文本会换到新的一行中。当录入这些代码行的时候，要将所有代码输入到同一行，并且不要按下回车键。当开始新的一行的时候，你可以从代码左边的行号来区分。例如，如下的代码只有两行：

```
1. print('This is the first instruction! xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxx')
2. print('This is the second instruction, not the third instruction.')
```

第一行代码折行了，这使得代码段看上去好像是一共有 3 行。注意，这只是因为图书的页面不够宽了，无法把第一条指令印刷在一行中。

1.4　寻求在线帮助

本书的网址是 http://inventwithpython.com。你可以从那里找到许多与本书相关的资源。本书中的几个链接都使用 invpy.com 域名以缩短 URL。

网站 http://reddit.com/r/inventwithpython 是就本书相关的编程问题提问的一个好地方。一般性的 Python 问题可以去网站 LearnProgramming 和 LearnPython 发贴子，它们的网址分别是 http://reddit.com/r/learnprogramming 和 http://reddit.com/r/learnpython。

你还可以把编程问题发送到作者的邮箱 al@inventwithpython.com。

请记住，聪明的提问方式对自己和别人都有帮助。一定要阅读这些网站上“Frequently Asked Questions”部分，这些网站都会介绍如何以正确的方式提问。当询问编程问题时，注意如下几点：

- 如果录入了本书中的程序，但是发现一个错误，请首先通过 http://invpy.com/diff

的在线 diff 工具检查录入错误。复制代码并将其粘贴到 diff 工具中，以查看你的程序和本书中的代码之间的任何差异。

- 当描述错误的时候，说明你想要做什么。这会让帮助你的人知道你是否完全走错了路。
- 复制并粘贴完整的错误消息和代码。
- 从网上查找是否有人也问过（或者回答）和你同样的问题。
- 说明你已经尝试了哪些方法去解决问题。这就会告诉人们，你已经做了一些工作来试图自己解决问题。
- 要有礼貌。不要命令帮助你的人或者给帮助你的人施压以求他们快速做出回答。

问别人“为什么我的程序不能工作？”，这样的问题并不能告诉他们任何有用的信息。应该告诉他们你想要做什么、确切的错误是什么以及你的操作系统和版本。

第 2 章　交互式 shell

本章主要内容：

- 整数和浮点数；
- 表达式；
- 值；
- 操作符；
- 计算表达式；
- 在变量中存储值。

在开始创建游戏之前，我们需要先介绍一些基本的编程概念。本章不会创建游戏，但是学习这些概念是编写电子游戏程序的第一步。我们先来学习如何使用 Python 的交互式 shell。

2.1　一些简单的数学知识

按照第 1 章中介绍的步骤打开 IDLE，然后使用 Python 来求解一些简单的数学问题。交互式 shell 可以像计算器一样工作。在交互式 shell 的>>>提示符之后，输入 2+2，然后按下回车键（有些键盘上显示为 RETURN 键）。

图 2-1 展示了交互式 shell 给出的响应是数字 4。

这道数学题就是一个简单的编程指令。加号（+）告诉计算机把数字 2 和 2 相加。表 2-1 列出了 Python 中其他可用的数学符号。减号（-）是数字相减。乘号（*）是数字相乘。除号（/）是数字相除。

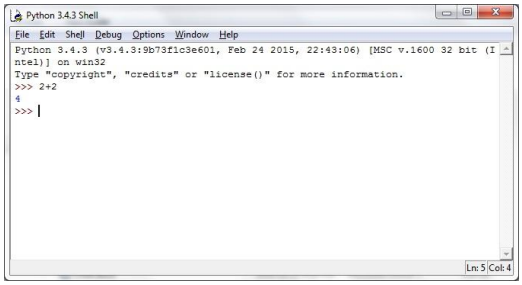


图 2-1　在交互式 shell 中输入 2+2

表 2-1　Python 中的各种数学操作符

操作符	运算
+	加法
-	减法
*	乘法
/	除法

当以这种方式使用时，+、-、*和/叫做操作符。操作符告诉 Python 要对它们旁边的数字进行何种运算。

2.1.1 整数和浮点数

整数就是诸如 4、99 或者 0 这样的数。浮点数（简写为 float）就是诸如 3.5、42.1 或者 5.0 这样的分数或小数。在 Python 中，数字 5 是整数，但是 5.0 是浮点数。这些数字都称为值。

2.1.2 表达式

下面这些数学问题就是表达式的示例。计算机可以在几秒钟之内解决数百万道这样的数学题。表达式是由操作符（数学符号）连接的值（数字）组成的。尝试在交互式 shell 中输入一些这样的数学问题，每输入完一道题后按一下回车键。

```
2+2+2+2+2
8*6
10-5+6
2   +       2
```

当输入上面的指令后，交互式 shell 将如图 2-2 所示。

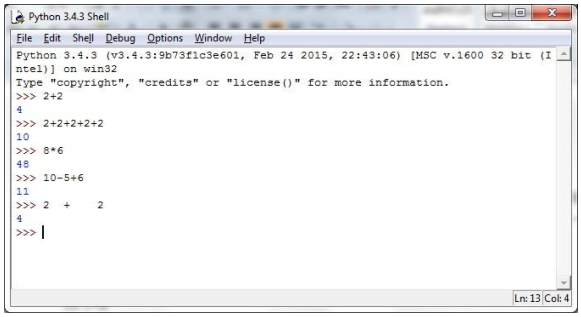


图 2-2 输入指令后 IDLE 窗口的样子

图 2-3 表达式由值和操作符组成

注意，在 2+ 2 的示例中，值和操作符之间可以有任意多个空格。然而，当把指令输入到交互式 shell 中时，总是从一行的开头开始执行（即指令之前不能有空格）。

2.2 计算表达式

当计算机求解表达式 10 + 5 并且得到值 15 的时候，它就已经计算了这个表达式。计算一个表达式就是把表达式规约为一个数字，就像解答一道数学题而把问题简化成一个数字一样：数字就是答案。表达式 10 + 5 和表达式 10 + 3 + 2 的计算结果都是 15。

表达式可以是任意大小的，但它们总是会求解得到一个数字。即便单个值也可以是表达

式：表达式 15 的计算结果就是值 15。例如，表达式 8 * 3 / 2 + 2 + 7 -9，通过如下步骤，其计算结果是值 12.0。

```
8 * 3 / 2 + 2 + 7-9
          ▼
  24 / 2 + 2 + 7-9
          ▼
    12.0  + 2 + 7-9
          ▼
       14.0  + 7-9
          ▼
            21.0-9
              ▼
               12.0
```

在交互式 shell 中看不到所有这些步骤。交互式 shell 对表达式进行运算，并且只把结果展示给我们。

```
>>> 8 * 3 / 2 + 2 + 7 -9
12.0
```

需要注意的是，除法操作符（/）的运算结果是一个浮点数，所以 24/2 的结果是 12.0。使用浮点数的数学运算，其结果也是浮点数，所以 12.0 +2 的结果是 14.0。

语法错误

如果在交互式 shell 中输入 5+，将会得到一条错误消息。

```
>>> 5 +
SyntaxError: invalid syntax
```

产生这个错误，是因为 5+不是一个表达式。表达式通过操作符来连接值。但是加法操作符期待+后边有一个值。当漏掉这个值时，就会出现一个错误。

SyntaxError 表示 Python 不理解这条指令，因为你的输入不正确。很多时候，从事计算机编程不只是告诉计算机要做什么，还要知道如何告诉它。

但是，不要担心出错。错误并不会对计算机造成危害。只要在交互式 shell 中的下一个>>>提示符处，重新输入正确的指令即可。

2.3　在变量中存储值

也可以把表达式的计算结果的值存储到变量中，以便后面可以使用它。可以把变量当做

是一个可以保存值的盒子。

一条赋值语句指令会把一个值保存到一个变量中。输入变量的名称，后边跟着等号（=称为赋值操作符），然后是要存储到这个变量中的值。例如，在交互式 shell 中输入 spam = 15：

```
>>> spam = 15
>>>
```

这将把值 15 存储到 spam 变量的盒子中，如图 2-4 所示。名字“spam”是盒子的标签（Python 由此可以识别变量），而值写在盒子中的一张便签上。

当按下回车键时，你不会看到任何响应。在 Python 中，如果没有出现错误，就表示成功地执行了指令。然后将会出现>>>提示符，你可以输入下一条指令了。

和表达式不同，语句是不会计算为任何值的指令。正因为如此，在 spam = 15 之后，交互式 shell 的下一行中没有显示任何的值。如果你不清楚哪些指令是表达式，哪些指令是语句，那么请记住：表达式会得到一个值，而任何其他类型的指令都是一条语句。

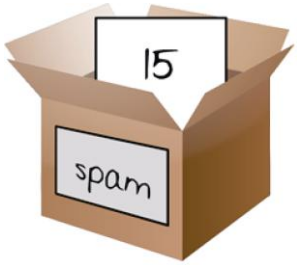


图 2-4 变量就像是可以容纳值的盒子

变量保存的是值而不是表达式。例如，考虑一下语句 spam = 10 + 5 和 spam = 10 + 7 - 2 中的表达式。它们的运算结果都是 15。最终结果是相同的：两条赋值语句都把值 15 保存到了变量 spam 中。

第一次在赋值语句中使用一个变量的时候，Python 将会创建该变量。要查看变量中的值，在交互式 shell 中输入该变量的名称：

```
>>> spam = 15
>>> spam
15
```

表达式 spam 得到了变量 spam 中的值，即 15。可以在表达式中使用变量。尝试在交互式 shell 中输入如下指令：

```
>>> spam = 15
>>> spam + 5
20
```

我们已经把变量 spam 的值设置为 15，所以输入 spam + 5 就像是输入表达式 15 + 5 一样。下面是 spam + 5 的运算步骤：

```
spam + 5
     ▼
  15 + 5
```

```
      ▼
    20
```

在赋值语句创建变量之前，不能使用该变量。否则，Python 将会给出一个 NameError 的错误，因为尚不存在该名称的变量。输错了变量名称也会得到这样一个错误：

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    spma
NameError: name 'spma' is not defined
```

出现这个错误，是因为虽然有 spam 变量，但是并没有名为 spma 的变量。

可以通过输入另一条赋值语句来修改变量中存储的值。例如，尝试在交互式 shell 中输入如下语句：

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

当第一次输入 spam + 5 时，表达式的计算结果是 20，因为我们把 15 存储在 spam 中。然而，当输入 spam = 3 时，用值 3 替代（或覆盖）了值 15。现在，当我们输入 spam + 5 时，表达式的计算结果是 8，因为现在 spam 的值是 3。覆盖的过程如图 2-5 所示。

图 2-5　用值 3 覆盖了 spam 中的值 15

甚至可以使用 spam 变量中的这个值，来给 spam 赋一个新的值：

```
>>> spam = 15
>>> spam = spam + 5
20
```

赋值语句 spam = spam + 5 的意思是：“spam 变量中的新值是，spam 当前的值加上 5”。通过在交互式 shell 中输入如下的语句，让 spam 中的值持续几次增加 5：

```
>>> spam = 15
```

```
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

使用多个变量

在程序中，可以根据需要创建任意多个变量。例如，让我们给名为 eggs 和 bacon 的两个变量分配不同的值，如下所示：

```
>>> bacon = 10
>>> eggs = 15
```

现在，变量 bacon 中是 10，变量 eggs 中是 15。每个变量都有自己的盒子，其中拥有其自己的值，如图 2-6 所示。

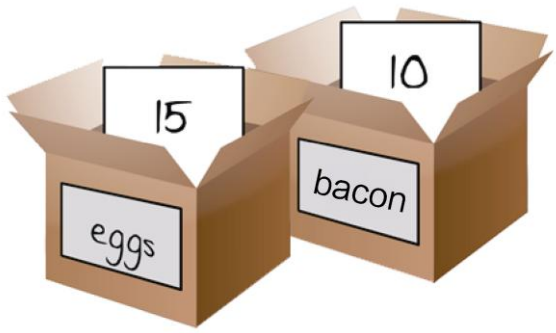


图 2-6 变量“bacon”和变量“eggs”中所存储的值

尝试在交互式 shell 中输入 spam = bacon + eggs，然后查看 spam 中的新值：

```
>>> bacon = 10
>>> eggs = 15
>>> spam = bacon + eggs
>>> spam
    25
```

现在，spam 中的值是 25。当把 bacon 和 eggs 相加时，就是把其值 10 和 15 相加。变量包含的是值而不是表达式。把值 25 赋给变量 spam，而不是把表达式 bacon + eggs 赋给变量。在 spam = bacon + eggs 赋值语句之后，对于 bacon 或者 eggs 的修改不会再影响到 spam。

2.4　本章小结

在本章中，我们介绍了关于编写 Python 指令的基础知识。Python 需要你以严格的方式，准确地告诉它要做什么工作。计算机不具备人的常识，并且只能理解特定的指令。

表达式是用操作符（如+或-）把值（如 2 或 5）组合起来。Python 可以计算表达式，也就是把表达式规约为一个值。可以把值保存在变量中，以便程序可以记住它们，并且随后可以使用它们。

在 Python 中，有许多其他类型的操作符和值。在下一章中，我们将介绍更多的基础概念，并且编写第一个程序。我们还会介绍在表达式中使用文本。Python 不仅能够处理数字，它的功能比计算器强大得多。

第 3 章　编写程序

本章主要内容：

- 执行的流程；
- 字符串；
- 连接字符串；
- 数据类型（诸如字符串或者整数）；
- 使用文件编辑器来编写程序；
- 在 IDLE 中保存并且运行程序；
- print()函数；
- input()函数；
- 注释；
- 区分大小写。

到目前为止，我们已经介绍了足够的数学相关的知识。现在，我们来看看 Python 能够对文本做些什么。在本章中，我们将介绍如何把文本保存到变量中，如何合并文本，以及如何在屏幕上显示文本。

几乎所有的程序都会向用户显示文本，并且用户可以使用键盘向程序输入文本。在本章中，你还会创建自己的第一个程序。这个程序显示了一句问候：“Hello World!”，并且会询问用户的姓名。

3.1　字符串

在 Python 中，把文本值叫做字符串。字符串值可以像整数或者浮点数一样地使用。我们可以把字符串保存到变量中。在代码中，字符串值使用单引号（'）作为起始和结束。尝试在交互式 shell 中输入如下代码：

```
>>> spam = 'hello'
```

单引号告诉 Python，字符串从哪里开始到哪里结束。单引号不是字符串值的文本的一部分。现在，如果在交互式 shell 中输入 spam，就可以看到 spam 变量中的内容。请记住，Python 将变量计算为变量中所存储的值。在这个示例中，spam 存储的值就是字符串'hello'。

```
>>> spam = 'hello'
>>> spam
'hello'
```

字符串可以包含任意的键盘字符，其长度也可以是任意的。一些字符串示例如下所示：

```
'hello'
'Hi there!'
'KITTENS'
'7 apples, 14 oranges, 3 lemons'
'Anything not pertaining to elephants is irrelephant.'
'A long time ago, in a galaxy far, far away...'
'O*&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'
```

3.2　连接字符串

可以使用操作符把字符串值组合起来生成表达式，就像对整数和浮点数所做的那样。可以使用+操作符组合两个字符串。这就是字符串连接。尝试在交互式 shell 中输入'Hello' + 'World!'。

```
>>> 'Hello' + 'World!'
'HelloWorld!'
```

这个表达式的结果是一个字符串值'HelloWorld!'。两个单词之间没有空格，因为两个待连接的字符串中都没有空格，这和下面示例不同：

```
>>> 'Hello ' + 'World!'
'Hello World!'
```

对于字符串和整数来讲，+操作符的作用并不相同，因为字符串和整数是不同的数据类型。所有的值都有一个数据类型。'Hello'的数据类型是字符串。5 的数据类型是整数。数据类型告诉 Python，当计算表达式时，操作符应该做什么。对于字符串，+操作符会把它们连接起来；而对于整数和浮点数，+操作符会把它们相加。

3.3　在 IDLE 的文件编辑器中编写程序

到目前为止，我们已经在 IDLE 的交互式 shell 中输入过指令，并且一次只输入一条指令。然而，当编写程序时，会输入多条指令，然后让它们一起运行。让我们来编写第一个程序！

IDLE 的另外一个部分叫做文件编辑器。单击交互式 shell 窗口顶端的 File 菜单，然后选择 New File。将会出现一个空白窗口供你输入程序代码，如图 3-1 所示。

两个窗口看上去很相似，但是请记住：交互式 shell 窗口有>>>提示符，而文件编辑器窗口则没有。

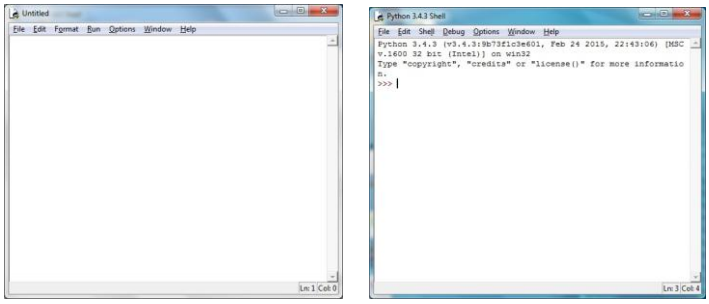


图 3-1 文件编辑器窗口（左边）和交互式 shell 窗口（右边）

3.4 Hello World!

对于程序员来讲，让自己的第一个程序在屏幕上显示“Hello world!”，这是一个传统。现在，你也要创建自己的 Hello World 程序了。

当输入程序时，不要输入代码左边的数字。这些数字是为了方便本书按照行号来引用代码。文件编辑器窗口的右下角会告诉你当前光标的位置。图 3-2 展示了当前光标在第 1 行第 0 列。

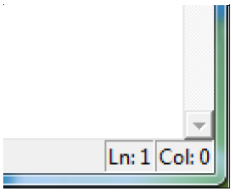


图 3-2 文件编辑器窗口右下角会告诉你当前光标在何处

hello.py

在新的文件编辑窗口中输入如下文本。这就是程序的源代码。其中包含了当程序运行时 Python 所要执行的指令。

> **请注意！**
>
> 本书中的程序只能在 Python 3 中运行，而在 Python 2 中则不无法运行。当 IDLE 窗口启动时，它会在上方显示类似“Python 3.4.2”的信息。如果你已经安装了 Python 2，可以同时安装 Python 3。可以去 https://python.org/download/下载 Python 3。

hello.py

```
1. # This program says hello and asks for my name.
2. print('Hello world!')
3. print('What is your name?')
4. myName = input()
5. print('It is good to meet you, ' + myName)
```

IDLE 程序用不同的颜色来表示不同类型的指令。输入代码之后，窗口看上去如下所示：

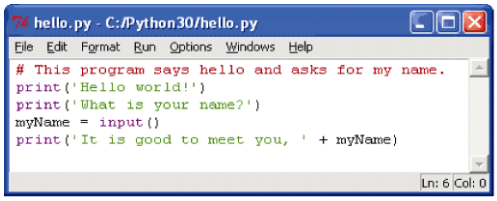


图 3-3　输入代码后，文件编辑器窗口的样子

3.5　保存程序

一旦输入了源代码，就可以点击 File ▶ Save As 来保存它。或者按下快捷键 Ctrl-S 来保存源代码。图 3-4 展示了这将会打开的 Save As 窗口。在 File name 文本框中输入 hello.py，然后点击 Save 按钮。

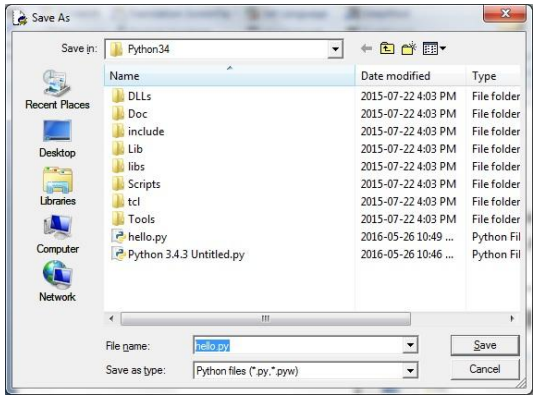


图 3-4　保存程序

当输入程序时，应该经常保存。这样的话，即使计算机崩溃或者突然退出 IDLE，也不会丢失太多的工作成果。

3.6　打开保存过的程序

要加载之前保存过的程序，单击 File ▶ Open。在所出现的窗口中选择文件，并且单击 Open 按钮。你所保存过的 hello.py 程序将会在文件编辑器窗口中打开。

现在，是时候运行程序了。在文件编辑器窗口中点击 File ▶ Run ▶ Run Module 或者按下 F5 键。程序会在交互式 shell 窗口中运行。

当程序要求输入姓名时，请输入你的姓名，如图 3-5 所示。

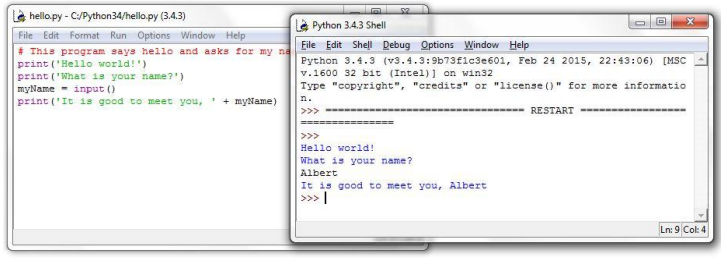


图 3-5　运行 hello.py 之后的交互式 shell 窗口

当输入姓名并且按下回车键时，程序将会用你的名字来打招呼。恭喜！你已经编写了第一个程序，并且现在已经成为一名软件程序员了。1 秒钟后再次按下 F5 键运行该程序，输入另一个名字。

如果有错误，使用 http://invpy.com/diff 上的在线 diff 工具，把你的代码与书中的代码进行比较。复制文件编辑器中的代码并且粘贴到这个 Web 页面中，然后点击 Compare 按钮。这个工具会高亮显示你的代码与本书代码之间的所有区别，如图 3-6 所示。

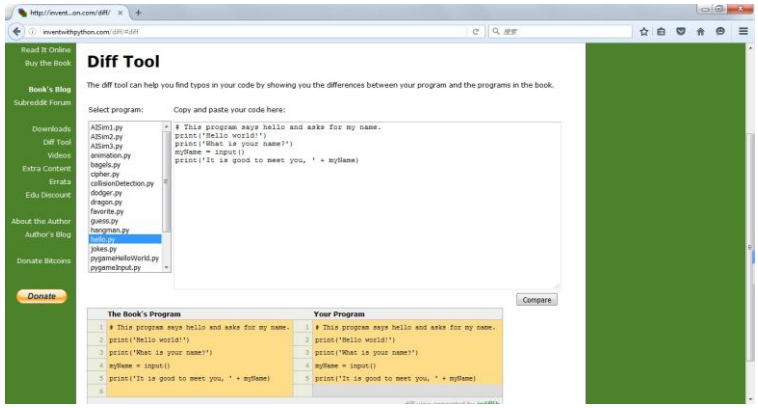


图 3-6　http://invpy.com/diff 上的在线 diff 工具

如果在编写代码时得到一个 NameError 错误，如下所示：

```
Hello world!
What is your name?
Albert
Traceback (most recent call last):
  File "C:/Python26/test1.py", line 4, in <module>
    myName = input()
  File "<string>", line 1, in <module>
NameError: name 'Albert' is not defined
```

这表示你使用的是 Python 2，而不是 Python 3。请从 https://python.org/download 下

载并安装 Python 3 版本。用 Python 3 重新运行该程序。

3.7　“Hello World”程序如何工作

每行代码就是一条可供 Python 解释的指令。这些指令构成了程序。计算机程序的指令就像是菜谱的操作步骤。从程序的顶部，沿着指令的列表向下，顺序地执行每条指令。

在程序中，把 Python 当前所处的指令叫做执行。当程序开始运行时，执行是第一条指令。执行完这条指令之后，执行移到下一条指令。

我们来看一下每行代码是如何工作的。我们从第 1 行开始。

注释

```
1. # This program says hello and asks for my name.
```

这条指令是一条注释。跟在#（叫做井号）后边的任何文本都是一条注释。注释不是供 Python 读取的，而是供程序员阅读的。Python 会忽略掉注释。注释是程序员针对代码做些什么而给出的注解。你可以在注释中写下任何内容。为了能够更容易阅读源代码，本书中的注释使用灰色文本表示。

程序员通常会在代码的开始处放置一个注释，以作为程序的标题。

3.7.1　函数

函数就像是程序中的一个小程序。函数包含许多可执行的指令，当调用函数时，就会执行这些指令。Python 提供了一些已经内建的函数。下面会介绍 print()和 input()这两个函数。函数的最美妙之处在于，你只需要知道函数是做什么的，而无需知道它是如何做的。

函数调用是一条指令，它告诉 Python 运行函数中的代码。例如，你的程序调用了 print()函数，在屏幕上显示一个字符串。print()函数接受在括号中字符串作为输入，并且把该文本显示到屏幕上。

要在屏幕上显示 Hello world!，输入 print 函数名称，后边跟随着开始圆括号，然后跟随着'Hello world!'字符串和结束圆括号。

3.7.2　print()函数

```
2. print('Hello world!')
3. print('What is your name?')
```

第 2 行和第 3 行是对 print()函数的调用。在函数调用中，括号之间的值是参数。第 2 行 print()函数调用的参数是'Hello world!'。第 3 行 print()函数调用的参数是'What is your

name?'。这叫做给 print()函数传递参数。

在本书中，函数名称后边都有一个圆括号。这使得 print()在本书中的含义更加明确，它表示名为 print()的一个函数，而不是名为 print 的一个变量。这就像用引号把数字 42 括起来，告诉 Python 这是一个字符串'42'，而不是数字 42。

3.7.3 input()函数

```
4. myName = input()
```

第 4 行是带有变量（myName）和函数调用（input()）的一条赋值语句。当调用 input()时，程序等待用户输入文本。用户输入的文本字符串成为函数调用所得到的结果值。在表达式中，可以使用一个值的任何位置，也都可以使用函数调用。

函数调用的结果值叫做返回值（实际上，“函数调用返回的值”和“函数调用的结果值”的含义是相同的）。在这个示例中，input()函数的返回值是用户输入他们自己的名称的字符串。如果用户输入“Albert”，input()函数调用的结果就是字符串'Albert'。计算过程如下所示：

```
myName = input()
           ▼
myName = 'Albert'
```

这样就把字符串值'Albert'存储到了 myName 变量中。

3.7.4 在函数调用中使用表达式

```
5. print('It is good to meet you, ' + myName)
```

上面这行代码是另一个 print()函数调用。在 print()括号中，是表达式'It is good to meet you, ' + myName。然而，参数总是单个的值。Python 会先计算这个表达式，然后将其值作为参数传递给函数。如果 MyName 中存储的是'Albert'，计算过程如下所示：

```
print('It is good to meet you, ' + myName)
                          ▼
print('It is good to meet you, ' + 'Albert')
                          ▼
print('It is good to meet you, Albert')
```

程序就是这样根据名称来向用户打招呼的。

3.7.5 终止程序

如果程序执行了最后一行代码，它会终止或者退出。这意味着程序将停止运行。Python

会忘掉在变量中保存的所有的值，也包括存储在 myName 中的字符串。如果再次运行程序，并且输入一个不同的名字，程序会认为这才是你的名字。

```
Hello world!
What is your name?
Carolyn
It is good to meet you, Carolyn
```

请记住，计算机只能做程序要求它做的事情。计算机不会说话，只能严格遵循你给它的指令。计算机不会在乎你录入的是自己的名字、别人的名字还是一些无聊的内容。你可以输入任何想要输入的内容。计算机都会以相同的方式来处理它：

```
Hello world!
What is your name?
poop
It is good to meet you, poop
```

3.8　变量名称

给变量一个具有描述性的名称，会更容易理解它在程序中的用途。假设你正在搬家，并且把一个要搬运的盒子都贴上“Stuff”。这么做根本就不会有任何帮助！

你也可以把这个变量命名为 abrahamLincoln 或 nAmE，而不是命名为 myName。Python 不关心这些。它只是以相同的方式运行程序。

变量名称区分大小写。区分大小写意味着会把字母相同但大小写不同的变量名当做是不同的变量。所以在 Python 中，spam、SPAM、Spam 和 sPAM 是 4 个不同的变量。它们每一个都包含自己的不同的值。在程序中使用字母相同而大小写不同的变量，这不是一个好主意。应该为变量使用具有描述性的名称。

变量名通常是小写的。如果变量名中有多个单词，第一个单词之后的每一个单词的首字母都要大写。这会使得代码更容易阅读。例如，变量名 whatIHadForBreakfastThisMorning 要比 whatihadforbreakfastthismorning 更容易阅读。这是一种惯例：你也可以不这么做，但是在 Python 编程中，这是标准的做法。

简短的变量名称比冗长的变量名称更好：breakfast 或者 foodThisMorning，要比 whatIHadForBreakfastThisMorning 更容易阅读。

本书中的交互式 shell 示例，都使用像 spam、eggs、ham 和 bacon 这样的变量名。这是因为在这些示例中，变量名并不重要。然而，本书中的程序都使用具有描述性的名称。你的程序也应该使用具有描述性的变量名。

3.9 本章小结

一旦学习了字符串和函数，就可以开始编写与用户交互的程序了。这很重要，因为文本是用户和计算机彼此通信的主要方式。使用 input()函数，用户通过键盘输入文本。使用 print()函数，计算机把文本显示到屏幕上。

字符串只是一种具有新的数据类型的值。任何值都有一种数据类型，在 Python 中，有许多的数据类型。+操作符可以连接字符串。

函数可以用做程序的一部分，以执行一些复杂的指令。Python 有许多内建函数，我们会在本书中介绍它们。在表达式中，在可以使用值的任何位置，也都可以使用函数调用。

在程序中，把 Python 当前所处的指令叫做执行。在下一章中，我们将介绍使得执行移动的更多的方式，而不只是直接从上向下执行。一旦我们掌握了这些知识，就为创建游戏做好了准备。

第 4 章　“猜数字”游戏

本章主要内容：

- import 语句；
- 模块；
- while 语句；
- 条件；
- 语句块；
- 比较操作符；
- =和==的区别；
- if 语句；
- 关键字 break；
- str()函数、int()函数和 float()函数；
- random.randint()函数。

在本章中，我们准备编写一个叫做“猜数字”的游戏。计算机想到一个 1 到 20 之间的随机数，让你来猜它是几。计算机会告诉你每次猜的数太大还是太小。如果能够在 6 次之内猜到正确的数字，就赢得游戏。

这是一个进行编程练习的很好的游戏，因为在一个小程序中，用到了随机数字、循环和用户输入。我们已经介绍过如何把值转换成不同的数据类型，以及为什么需要这么做。因为这个程序是一个游戏，所以我们会把用户称为玩家，但是称为“用户”也没有问题。

4.1　“猜数字”游戏的运行示例

如下是玩家运行程序时的样子。玩家输入的文本用粗体表示。

```
Hello! What is your name?
Albert
Well, Albert, I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too high.
Take a guess.
2
Your guess is too low.
Take a guess.
```

```
4
Good job, Albert! You guessed my number in 3 guesses!
```

4.2 “猜数字”游戏的源代码

通过点击 File▶New Window 打开一个新的文件编辑器窗口。在出现的空白窗口中，输入源代码，并且把它保存为 guess.py。然后按下 F5 键来运行程序。当在文件编辑器中输入代码时，要留意一些代码行前面的空格。有些行有 4 个或者 8 个缩进空格。

> **请注意！**
>
> 本书中的程序只能在 Python 3 中运行，而在 Python 2 中则无法运行。当开启 IDLE 窗口时，它会在顶端显示类似于“Python 3.4.2”的信息。如果你已经安装了 Python 2，也可以同时安装 Python 3。请通过 https://python.org/download/下载 Python 3。

如果输入这些代码之后出现错误，请使用在 http://invpy.com/diff/guess 的在线 diff 工具，对比你输入的代码和本书代码。

guess.py

```
1. # This is a guess the number game.
2. import random
3.
4. guessesTaken = 0
5.
6. print('Hello! What is your name?')
7. myName = input()
8.
9. number = random.randint(1, 20)
10. print('Well, ' + myName + ', I am thinking of a number between 1 and 20.')
11.
12. while guessesTaken < 6:
13.     print('Take a guess.') # There are four spaces in front of print.
14.     guess = input()
15.     guess = int(guess)
16.
17.     guessesTaken = guessesTaken + 1
18.
19.     if guess < number:
20.         print('Your guess is too low.') # There are eight spaces in front
of print.
21.
22.     if guess > number:
23.         print('Your guess is too high.')
```

```
24.
25.     if guess == number:
26.         break
27.
28. if guess == number:
29.     guessesTaken = str(guessesTaken)
30.     print('Good job, ' + myName + '! You guessed my number in ' +
guessesTaken + ' guesses!')
31.
32. if guess != number:
33.     number = str(number)
34.     print('Nope. The number I was thinking of was ' + number)
```

4.3　import 语句

```
1.# This is a guess the number game.
2. import random
```

第 1 行是一条注释。记住，Python 会忽略掉#号后边的所有内容。注释只是提醒我们程序要做什么。

第 2 行是一条 import 语句。记住，语句是执行某些动作的指令，而不像表达式那样会计算为一个值。我们已经见过语句了，例如把一个值存储到一个变量中的赋值语句。

Python 包含许多内建的函数，有些函数存在于称为模块的单独的程序中。可以使用一条 import 语句把它们的模块导入到你的程序中，这样就可以使用这些函数了。

第 2 行导入了名为 random 的模块，以便程序可以调用 random.randint()函数。这个函数将产生一个随机数字，供用户进行猜测。

```
4. guessesTaken = 0
```

第 4 行创建了一个名为 guessesTaken 的新变量。我们将把玩家猜过的次数保存到这个变量中。因为此时玩家还没有做过任何猜测，所以这里保存的是整数 0。

```
6. print('Hello! What is your name?')
7. myName = input()
```

第 6 行和第 7 行与我们在第 3 章的 Hello World 程序中见到的代码行一样。程序员经常复用其他程序中的代码，以减少自己的工作量。

第 6 行是对 print()函数的一次调用。请记住，函数就像是程序中的一个小程序。当程序调用一个函数时，它会运行这个小程序。print()函数中的代码把传递给它的字符串参数显示到屏幕上。

第 7 行要求用户输入姓名，并且将输入存储到 myName 变量中（记住，这个字符串可能并不是玩家的真实姓名。它可能只是玩家输入的任意字符串。计算机则只会无条件地执行指令）。

4.4 random.randint()函数

```
9. number = random.randint(1, 20)
```

第 9 行调用了一个名为 randint()的新函数，并且把返回值存储到了变量 number 中。记住，函数调用可以是表达式的一部分，因为函数调用也会求得一个值。

randint()函数是由 random 模块提供的，所以在它前边要加上 random.（别漏掉那个点），这用来告知 Python，randint()是 random 模块中的函数。

randint()函数会返回一个随机的整数，该整数在接收到的两个整数参数之间（也包括这两个整数）。第 9 行把 1 和 20 传入到函数名称后边的圆括号中，两个数之间用逗号隔开。把 randint()返回的随机整数存储到名为 number 的变量中，这就是玩家试图猜测的神秘数字。

稍等片刻，回到交互式 shell，并且输入 import random，以导入 random 模块。然后输入 random.randint(1, 20)，查看这个函数调用的结果。它会返回 1 到 20 之间的一个整数。再输一次这行代码，函数调用会返回一个不同的整数。randint()函数每次返回一个随机的整数，就像每次掷色子都会得到一个随机数字一样：

```
>>> import random
>>> random.randint(1, 20)
12
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
3
>>> random.randint(1, 20)
18
>>> random.randint(1, 20)
7
```

当你想要为游戏增加随机性时，使用 randint()函数。在许多游戏中，都会用到随机性（想想看，那么多的桌上游戏要使用色子）。

也可以通过修改参数，来尝试不同范围的数字。例如，输入 random.randint(1, 4)，只会得到 1 到 4 之间的整数（包含 1 和 4）。或者尝试 random.randint(1000, 2000)，来获取 1000 到 2000 之间的整数。

例如，在交互式 shell 中输入下面的语句。当调用 random.randint()函数时，所得到的结果可能是不同的（毕竟它是随机的）。

```
>>> random.randint(1, 4)
3
>>> random.randint(1000, 2000)
1294
```

对游戏代码稍作修改，就可以使得游戏行为有所不同。尝试把第 9 行和第 10 行的内容：

```
9. number = random.randint(1, 20)
10. print('Well, ' + name + ', I am thinking of a number between 1 and 20.')
```

修改为如下所示：

```
9. number = random.randint(1, 100)
10. print('Well, ' + name + ', I am thinking of a number between 1 and 100.')
```

现在计算机将会考虑 1 到 100 之间的一个整数，而不是 1 到 20 之间的一个整数。第 9 行的改动将会改变随机数字的范围，但是要记住还要修改第 10 行，以便让游戏告诉玩家新的数字范围而不是旧的范围。

欢迎玩家

```
10. print('Well, ' + myName + ', I am thinking of a number between 1 and 20.')
```

第 10 行的 print()函数根据姓名来欢迎玩家，并且告诉他们计算机所考虑的数字范围。

在第 10 行的代码中，看上去好像不是 1 个字符串参数，但是仔细看这一行。加号把 3 个字符串连接成 1 个字符串。最终，只有 1 个字符串作为参数传递给了 print()函数。如果再看一下，会看到引号中的逗号以及各个部分的字符串。

4.5　循环

```
12. while guessesTaken < 6:
```

第 12 行是一条 while 语句，它表示 while 循环的开始。循环可以一遍遍地重复执行代码。然而在学习循环之前，我们需要先学习一些其他的概念，包括语句块、布尔值、比较操作符、条件和 while 语句。

4.6　语句块

可以把许多代码行组织到一个语句块中。语句块中的每一行代码都有相同的、最小数量的缩进。可以通过查看代码行前面的空格数，来判断语句块的起始和结束。这就是

代码行的缩进。

当代码行的缩进增加时（通常是增加 4 个空格），就开始了一个语句块。接下来任何也缩进 4 个空格的代码行，都是这个语句块的一部分。当有一行代码和该语句块开始之前的缩进相同，那么，这个语句块就结束了。这意味着，语句块可以嵌套在其他已有的语句块之中。图 4-1 所示的代码中，我们将一个语句块标记出来并进行编号。

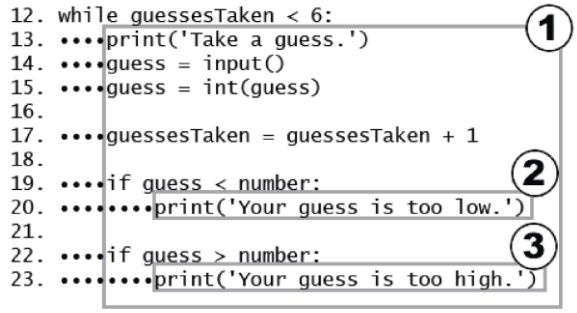


图 4-1 语句块和它们的缩进。语句块中的点表示空格

在图 4-1 中，第 12 行没有缩进，并且它不在任何语句块之中。第 13 行有 4 个空格的缩进。既然这行的缩进大于前一行的缩进，那么就开始了一个新的语句块。这个语句块在图 4-1 中标记为①。这个语句块将继续，直到出现没有空格（这是语句块开始之前最初的缩进）的一行代码。空行可以忽略掉。

第 20 行有 8 个空格的缩进。8 个空格多于 4 个空格，这开始了一个新的语句块。这个语句块在图 4-1 中标记为②。这个语句块位于另一个语句块之中。

第 22 行只有 4 个空格。因为缩进减少了，我们知道这个语句块结束了。第 20 行是该语句块中唯一的一行代码。第 22 行与其他带有 4 个空格的代码行位于同一个语句块之中。

第 23 行的缩进增加到了 8 个空格，所以又开始了一个新的语句块。它在图 4-1 中标记为③。

简单回顾一下，第 12 行不在任何的语句块之中。第 13 行到第 23 行都属于编号为①的语句块。第 20 行属于编号为②的语句块。第 23 行是编号为③的另一个语句块的唯一的一行。

4.7 布尔数据类型

布尔数据类型只有两个值：True 或者 False。这两个值的首字母必须大写，值的剩余部分必须小写。我们将用布尔值（称为 Boolean，或者简单称为 bool）和比较操作符来组成条件（条件会稍后介绍）。

例如，尝试把布尔值存储到变量中：

```
>>> spam = True
>>> eggs = False
```

到目前为止，我们已经介绍过的数据类型有整数、浮点数、字符串和现在的布尔值。Python 中的每一个值都属于某一种数据类型。

4.8　比较操作符

第 12 行有一条 while 语句：

```
12. while guessesTaken < 6:
```

在 while 关键字后边的这个表达式（guessesTaken < 6）中，包含了两个值（变量 guessesTaken 中的值和整数 6），用一个操作符（即小于号 <）来连接它们。<是一个比较操作符。

比较操作符比较两个值，并且会得到一个 True 或者 False 的布尔值。所有比较操作符如表 4-1 所示。

表 4-1　比较操作符

操作符	操作符名称
<	小于
>	大于
<=	小于等于
>=	大于等于
==	等于
!=	不等于

我们已经介绍过数学操作符+、-、*和/。同其他操作符一样，比较操作符把值组合成诸如 guessesTaken < 6 这样的表达式。

4.9　条件

条件是用比较操作符（如<或>）把两个值组合起来的一个表达式，并且条件的结果是一个布尔值。条件只是结果为 True 或 False 的表达式的另一种叫法而已。条件用于 while 语句（以及稍后介绍的一些其他指令）中。

例如，条件 guessesTaken < 6 表示“存储在 guessesTaken 中的值是否小于数字 6？”。

如果是，那么该条件结果为 True。如果不是，该条件结果为 False。

在“猜数字”程序中，在第 4 行，我们把值 0 存储到了 guessesTaken 中。因为 0 小于 6，所以这个条件的结果是布尔值 True。结果看上去如下所示：

```
guessesTaken < 6
             ▼
            0 < 6
             ▼
             True
```

体验布尔值、比较操作符和条件

在交互式 shell 中，输入如下表达式来查看它们的布尔值结果：

```
>>> 0 < 6
True
>>> 6 < 0
False
>>> 50 < 10
False
>>> 10 < 11
True
>>> 10 < 10
False
```

因为数字 0 小于数字 6，所以条件 0 < 6 会返回布尔值 True。但是因为 6 不小于 0，所以条件 6 < 0 的结果是 False。50 并不小于 10，所以 50 < 10 的结果是 False。10 小于 11，所以 10 < 11 的结果是 True。

请注意，10 < 10 的结果为 False，因为数字 10 并不小于数字 10，它们一样大。如果 Alice 和 Bob 一样高，你不能说 Alice 比 Bob 高或者 Alice 比 Bob 矮，这两种说法都不对。

现在尝试在交互式 shell 中输入如下表达式：

```
>>> 10 == 10
True
>>> 10 == 11
False
>>> 11 == 10
False
>>> 10 != 10
False
>>> 10 != 11
True
```

```
>>> 'Hello' == 'Hello'
True
>>> 'Hello' == 'Goodbye'
False
>>> 'Hello' == 'HELLO'
False
>>> 'Goodbye' != 'Hello'
True
```

4.10　=和==的区别

不要把赋值操作符（=）和“等于”比较操作符（==）搞混淆了。等号（=）用于赋值语句，用来把值存储到变量中；而双等号（==）用于表达式，用来判断两个值是否相等。当你想使用其中某一个操作符时，很容易会错误地使用了另一个操作符。

只要记住，“等于”比较操作符（==）有两个字符，就像“不等于”比较操作符（!=）也有两个字符一样。

字符串和整数值彼此之间永远都不会相等。例如，尝试在交互式 shell 中输入如下语句：

```
>>> 42 == 'Hello'
False
>>> 42 != '42'
True
```

4.11　while 循环语句

while 语句表示一个循环的开始。循环可以重复执行相同的代码。当执行到一条 while 语句时，会计算 while 关键字后边的条件。如果该条件结果为 True，执行会移入到后续的语句块中，也就是 while 语句块（在这个程序中，while 语句块从第 13 行开始）。如果条件结果为 False，执行会跳过整个 while 语句块。在猜数字程序中，while 语句块之后的第一行代码是第 28 行。

while 语句在条件后边总是有一个冒号（:）。以冒号结尾的语句，期待下一行是一个新的语句块。

```
12. while guessesTaken < 6:
```

图 4-2 展示了如何通过条件来控制程序的执行流程。如果条件的结果为 True（第 1 次就是这样，因为 guessesTaken 的值是 0），执行会在第 13 行进入 while 语句块，并且一直往下执行。一旦程序到达了 while 语句块的结尾处，并不会执行下一行代码，而是会回到 while

语句的那一行（第 12 行）执行循环并且重新计算条件结果。和之前一样，如果条件为 True，执行会再次进入 while 语句块。通过循环的每一次执行，叫做一次迭代。

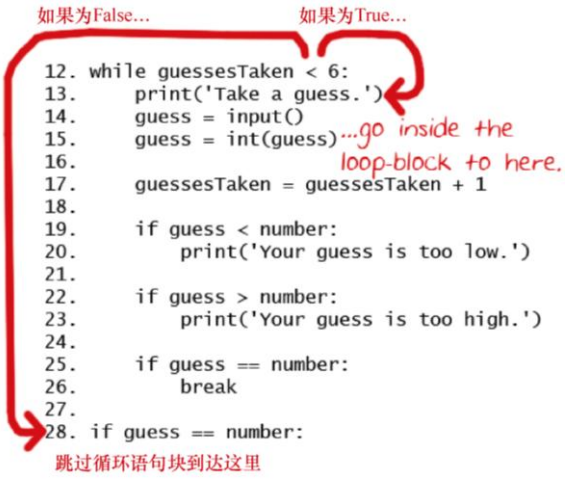


图 4-2　while 循环的条件

循环就是这样工作的。只要条件为 True，程序就会重复执行 while 语句块中的代码，直到条件第一次为 False。可以把 while 语句理解为：“当条件为真时，继续执行后续语句块中的代码”。

玩家的猜测

```
13. print('Take a guess.') # There are four spaces in front of print.
14. guess = input()
```

第 13 行到第 17 行要求玩家去猜这个神秘的数字是几，并且让他们输入这个数字。输入的这个数字会存储到一个名为 guess 的变量中。

4.12　使用 int()函数、float()函数、str()函数和 bool()函数来转换值

```
15. guess = int(guess)
```

第 15 行调用了一个名为 int()的新函数。int()函数接受一个参数，并且返回该参数的整数形式。尝试在交互式 shell 中输入如下语句：

```
>>> int('42')
42
>>> 3 + int('2')
5
```

int('42')调用将返回整数值 42。然而，尽管可以传递一个字符串给 int()函数，但是不能传递任意的字符串。传递'forty-two'给 int()函数将会导致一个错误。传递给 int()函数的字符串必须是由数字构成的：

```
>>> int('forty-two')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
int('forty-two')
ValueError: invalid literal for int() with base 10: 'forty-two'
```

3 + int('2')这一行给出了一个表达式，使用 int()函数的返回值作为该表达式一部分。其结果是整数值 5：

```
3 + int('2')
 ▼
3 + 2
 ▼
  5
```

请记住，input()函数总是返回玩家所输入的文本的一个字符串。如果玩家输入的是 5，input()函数将返回字符串'5'，而不是整数 5。Python 不能使用比较操作符<和>来比较一个字符串和一个整数值：

```
>>> 4 < '5'
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    4 < '5'
TypeError: unorderable types: int() < str()
```

```
14.     guess = input()
15.     guess = int(guess)
```

在第 14 行中，guess 变量最初存储的是玩家输入的字符串值。第 15 行使用 int()返回的整数值覆盖了 guess 中的字符串值。这就使得程序后面的代码能够比较 guess 是否大于、小于或者等于 number 变量中的神秘数字。

最后一点需要注意：调用 int(guess)不会改变 guess 变量中的值。代码 int(guess)是一个表达式，它获取了存储在 guess 变量中的字符串的整数形式。如下的赋值语句才

会修改 guess：

```
guess = int(guess)
```

类似的，float()、str()和 bool()函数分别返回和所传递的参数对应的浮点数、字符串和布尔值等版本。尝试在交互式 shell 中输入如下内容：

```
>>> float('42')
42.0
>>> float(42)
42.0
>>> str(42)
'42'
>>> str(42.0)
'42.0'
>>> str(False)
'False'
>>> bool('')
False
>>> bool('any nonempty string')
True
```

使用 int()、float()、str()和 bool()函数，就可以接受一种数据类型的值，而返回另一种数据类型的值。

变量的递增

```
17. guessesTaken = guessesTaken + 1
```

一旦玩家做了 1 次猜测，猜测的次数就应该增加 1 次。

在循环的第 1 次迭代中，guessesTaken 的值为 0。Python 会获取这个值，并且把它加 1。0+1 的结果为 1，然后把 1 存储为 guessesTaken 的新值。第 17 行的含义是：“guessesTaken 变量应该比它已有的值大 1”。

把变量的整数值或者浮点数值加 1，这叫做变量的递增。把变量的整数值或者浮点数值减 1，这叫做变量的递减。

4.13 if 语句

```
19.     if guess < number:
20.         print('Your guess is too low.') # There are eight spaces in front
of print.
```

第 19 行是一条 if 语句。如果 if 语句的结果为 True，执行将会运行后续的语句块中的代码。如果条件为 False，那么执行将会跳过 if 语句块中的代码。使用 if 语句，可以让程序只运行我们想要让它运行的某些代码。

第 19 行判断玩家的猜测是否小于计算机的神秘数字。如果是的话，那么执行会移入到第 20 行的 if 语句块，并且打印出一条消息告诉玩家这一点。

if 语句和 while 语句的工作几乎也是相同的。和 while 语句块不同的是，在 if 语句块结尾处，执行不会跳转回 if 语句，而是会继续向下执行下一行代码。换句话讲，if 语句不会循环。两种语句的比较如图 4-3 所示。

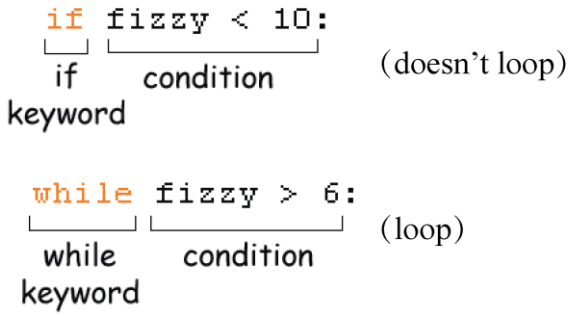


图 4-3　if 语句和 while 语句

```
22.     if guess > number:
23.         print('Your guess is too high.')
```

第 22 行判断玩家猜测的数字是否大于神秘数字。如果这个条件为 True，那么 print()函数调用会告诉玩家，他们的猜测的数字太大了。

4.14　用 break 语句提早离开循环

```
25.     if guess == number:
26.         break
```

第 25 行的 if 语句判断 guess 是否等于神秘数字。如果是相等的，程序运行第 26 行的 break 语句。

break 语句告诉执行要立即跳出 while 语句块，跳到 while 语句块结束之后的第一行。break 语句不会再重新检查 while 循环的条件了。

break 语句只会出现在循环中，如 while 语句块中。

如果玩家猜测的数字不等于神秘数字，执行会到达 while 语句块的底部。这意味着执行

将会返回到循环的顶部，重新判断第 12 行的条件（guessesTaken < 6）。记住，在执行了 guessesTaken = guessesTaken + 1 指令之后，guessesTaken 的新值是 1。因为 1 < 6 为 True，执行将再次进入循环。

如果玩家继续猜测的数还是太小或者太大，guessesTaken 的值会变为 2，然后是 3、4、5 和 6。当 guessesTaken 中存储的值为 6 时，while 语句的条件(guessesTaken < 6)为 False，因为 6 并不小于 6。因为 while 语句的条件为 False，执行会移动到 while 语句块之后的第 1 行，也就是 28 行。

4.14.1 判断玩家是否赢了

```
28. if guess == number:
```

第 28 行没有缩进，这表示 while 语句块已经结束了，这是 while 语句块之后的第 1 行。要么 while 语句的条件为 False(玩家用完了猜测次数),要么执行了第 26 行的 break 语句(玩家猜对了数字)，执行就会离开 while 语句块。

第 28 行判断玩家是否猜对了。如果猜对了，执行进入第 29 行的 if 语句块。

```
29.     guessesTaken = str(guessesTaken)
30.     print('Good job, ' + myName + '! You guessed my number in ' +
guessesTaken + ' guesses!')
```

只有第 28 行的 if 语句中的条件为 True 的时候（也就是玩家猜对了计算机的数字），才会执行第 29 行和第 30 行。

第 29 行调用 str()函数，该函数会返回 guessesTaken 的字符串形式。第 30 行把字符串连接到一起，告诉玩家他们赢得了游戏以及他们猜了多少次。只有字符串值才可以和其他的字符串连接。这就是为什么第 29 行必须把 guessesTaken 修改为字符串形式。否则的话，试图把一个字符串和一个整数连接在一起，将会导致 Python 显示一个错误。

4.14.2 判断玩家是否输了

```
32. if guess != number:
```

第 32 行使用“不等于”比较操作符!=来判断玩家最后猜测的数字是否不等于神秘数字。如果这个条件结果为 True，执行会移入到第 33 行的 if 语句块中。

第 33 行和第 34 行在 if 语句块中，只有第 32 行的条件为 True 的时候，才会执行这两行。

```
33.     number = str(number)
34.     print('Nope. The number I was thinking of was ' + number)
```

在这个语句块中，程序告诉玩家他们没有猜中的神秘数字是什么。这需要把字符串连接起来，但是 number 中存储的是一个整数值。第 33 行会用字符串形式覆盖 number，以便能够在第 34 行把它与字符串'Nope. The number I was thinking of was '连接起来。

此时，执行已经到了代码的结尾，程序结束了。恭喜！你已经编写了自己的第一个真正的游戏！

你可以通过修改玩家猜测的次数来让程序变的更难。要让玩家只能猜 4 次，把第 12 行的代码：

```
12. while guessesTaken < 6:
```

修改为：

```
12. while guessesTaken < 4:
```

在 while 语句块中，随后的代码在每次迭代中会把变量 guessesTaken 递增。通过设置条件 guessesTaken < 4，就可以确保循环中的代码只运行 4 次，而不是 6 次。这会使得游戏变得更难。要让游戏简单一些，把条件设置为 guessesTaken < 8 或 guessesTaken < 10。这会导致循环运行的次数更多一些，并且接受玩家更多次的猜测。

4.15　流程控制语句

在前面各章中，程序执行都是从程序中最顶端的指令开始，直接向下执行，顺序地执行每一条指令。但是有了 while 语句、if 语句、else 语句和 break 语句，我们就可以让程序循环或者根据条件来略过指令。这类语句叫做流程控制语句（flow control statement），因为当执行程序的时候，它们改变了程序执行的“流程”。

4.16　本章小结

如果有人问你：“到底什么是编程？”，你会怎么说？编程就是为程序编写代码的行为，也就是说，创建计算机可以执行的程序。

“但是，到底什么是程序呢？”，当你看到某人使用计算机程序时（例如，玩你的“猜数字”游戏），你所看到的只是屏幕上出现的一些文本。根据程序的指令以及玩家用键盘输入的文本（程序的输入），程序决定了到底在屏幕上显示什么样的文本（程序输出）。程序只是基于用户输入而执行的动作的一个指令集。

“有什么样的指令？”，实际上，只有几个不同种类的指令。

1．表达式是由操作符连接的值。表达式的最终结果是一个单独的值，例如 2 + 2 的结果是 4，或者'Hello' + ' ' + 'World'的结果是'Hello World'。当表达式跟在 if 和 while 关键字后

面的时候，我们也可以把它们称为条件。

2．赋值语句把值存储到变量中，以便在随后的程序中能够使用这个值。

3．if、while 和 break 语句都是流程控制语句，可以导致执行跳过指令、循环地执行指令或者跳出循环。函数调用也可以通过跳转到函数中的指令来改变执行的流程。

4．print()函数和 input()函数。这两个函数分别在屏幕上显示文本和从键盘上接收文本。这叫做 I/O（输入/输出），因为 I/O 负责程序的输入和输出。

就这些了，只有这 4 种指令。当然，关于这 4 种类型的指令，还有许多的细节。在本书中，我们将学习新的数据类型和操作符、新的流程控制语句以及许多其他的 Python 函数。还有不同类型的 I/O，如用鼠标输入、输出声音和图形，而不是只输出文本。

对于使用你的程序的人，他们只需要关心最后一种类型的指令，也就是 I/O。用户通过键盘输入，然后在屏幕上看到一些东西或者通过扬声器听到一些东西。但对于计算机来说，要搞清楚显示什么内容以及播放什么声音，它需要一个程序，程序只是程序员（你）已经编写好的一堆指令而已。

第 5 章　Jokes

本章主要内容：

- 转义字符；
- 使用单引号和双引号的字符串；
- 使用 print()的 end 关键字参数来略过换行。

5.1　用好 print()函数

本书中大部分的游戏都使用简单的文本作为输入和输出。输入是用户使用键盘来键入的。输出是在屏幕上显示的文本。在 Python 中，print()函数在屏幕上显示文本性的输出。但是，我们还需要进一步学习 Python 中字符串和 print()函数是如何工作的。

本章的程序为用户讲述了一些不同的笑话，并且展示了使用高级的字符串和 print()函数的代码。

5.2　Jokes 游戏的运行示例

```
What do you get when you cross a snowman with a vampire?
Frostbite!
What do dentists call an astronaut's cavity?
A black hole!
Knock knock.
Who's there?
Interrupting cow.
Interrupting cow wh-MOO!
```

5.3　Jokes 游戏的源代码

通过点击 File▶New Window 打开一个新的文件编辑器窗口。在出现的空白窗口中，输入源代码，并且把它保存为 jokes.py。然后按下 F5 键来运行程序。

> **请注意！**
>
> 本书中的程序只能运行在 Python 3 上，而在 Python 2 上则不可以运行。当 IDLE 窗口启动时，它会在上方显示类似“Python 3.4.2”的信息。如果你已经安装了 Python2，可以同时安装 Python 3。请通过 https://python.org/download/下载 Python 3。

如果有错误，请使用 http://invpy.com/diff/jokes 上的在线 diff 工具，把你输入的代码与书中的代码进行比较。

jokes.py

```
 1. print('What do you get when you cross a snowman with a vampire?')
 2. input()
 3. print('Frostbite!')
 4. print()
 5. print('What do dentists call a astronaut\'s cavity?')
 6. input()
 7. print('A black hole!')
 8. print()
 9. print('Knock knock.')
10. input()
11. print("Who's there?")
12. input()
13. print('Interrupting cow.')
14. input()
15. print('Interrupting cow wh', end='')
16. print('-MOO!')
```

代码如何工作

```
 1. print('What do you get when you cross a snowman with a vampire?')
 2. input()
 3. print('Frostbite!')
 4. print()
```

第 1 行到第 4 行调用了 print()函数 3 次。因为不想让玩家马上看到 Joke 的逗笑包袱，所以在第 1 次 print()函数之后，立即调用了 input()函数。玩家可以阅读笑话，然后按下回车键，然后再阅读逗笑包袱。

用户也可以输入一个字符串，并且按下回车键，但是这并不会把返回的字符串存储到任何的变量中。

程序将会忘记这个字符串，并且进入下一行代码。

第 4 行的 print()函数的调用并没有字符串参数。这就告诉程序只打印一个空白行。空白行很有用，它可以防止文本挤在一起。

5.4 转义字符

```
 5. print('What do dentists call a astronaut\'s cavity?')
 6. input()
 7. print('A black hole!')
```

```
8. print()
```

在第 5 行中，单引号之前有一个反斜杠，即\'。注意，\是一个反斜杠，/是一个斜杠。反斜杠告诉我们，在它之后的字母是转义字符。转义字符使我们能够打印那些很难输入到源代码中的字符。第 5 行的转义字符是一个单引号。

如果这里没有单引号转义字符，Python 会认为这个引号表示字符串的结束。但是，这个引号需要成为字符串的一部分。转义后的单引号告诉 Python，这个单引号是字符串的一部分，而不是表示字符串值的结束。

一些其他的转义字符

如果真的想要显示一个反斜杠，那该怎么办？如下的指令是无效的：

```
>>> print('They flew away in a green\teal helicopter.')
They flew away in a green    eal helicopter.
```

这是因为它把“teal”中的“t”当成了一个转义字符，因为它在反斜杠之后。这个转义字符模拟键盘上的 Tab 按键。相反，试试下面这行代码：

```
>>> print('They flew away in a green\\teal helicopter.')
They flew away in a green\teal helicopter.
```

表 5-1 是 Python 中的转义字符列表。

表 5-1　转义字符

转义字符	实际打印出的内容
\\	反斜杠（\）
\'	单引号（'）
\"	双引号（"）
\n	换行符
\t	Tab

5.5　引号和双引号

在 Python 中，字符串不一定要放在单引号之间，也可以把它们放在双引号之间。如下两行代码打印的是相同的内容：

```
>>> print('Hello world')
Hello world
```

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

但是，这两种引号不能混用。如果试图像下面这样使用引号，会得到一个错误：

```
>>> print('Hello world")
SyntaxError: EOL while scanning single-quoted string
```

我个人喜欢使用单引号，因为输入它们无需按下 Shift 键。这会使得输入更容易，而且 Python 也不在乎你使用单引号还是双引号。

要在单引号包围的字符串中出现单引号，需要使用转义字符\'，同样，要在双引号包围的字符串中出现双引号，也需要使用转义字符\"。例如，请看如下两行代码：

```
>>> print('I asked to borrow Abe\'s car for a week. He said, "Sure."')
I asked to borrow Abe's car for a week. He said, "Sure."

>>> print("She said, \"I can't believe you let them borrow your car.\"")
She said, "I can't believe you let them borrow your car."
```

在单引号包围的字符串中，对于双引号不需要转义；在双引号包围的字符串中，对于单引号不需要转义；如"astronaut's"。Python 解释器足够聪明，知道如果以一种类型的引号开始一个字符串的话，是不会以另一种类型的引号表示字符串的结束的。

5.6 print()的 end 关键字参数

```
 9. print('Knock knock.')
10. input()
11. print("Who's there?")
12. input()
13. print('Interrupting cow.')
14. input()
15. print('Interrupting cow wh', end='')
16. print('-MOO!')
```

注意到第 15 行的 print()函数的第 2 个参数了吗？通常，print()函数会在所打印的字符串的末尾添加一个换行符。这就是为什么一个空的 print()函数会只打印出一个换行符。但是，print()函数也可以选择使用另外一个参数（参数名是 end）。

传递的空字符串叫做关键字参数。end 参数有一个特定的名称，并且要为这个特定的参数传递一个关键字参数的话，必须在参数前边输入 end=。

为 end 传递一个空字符串，print()函数就不会在字符串的末尾添加一个换行符，而是会添加了一个空字符串。这就是为什么'-MOO!'出现在前一行的末尾，而不是单独出现在新的

一行。在打印了'Interrupting cow wh'字符串之后，并没有换行。

5.7　本章小结

本章介绍了 print()函数的不同使用方式。转义字符用于那些很难或者不太可能使用键盘输入到代码中的字符。转义字符是键入到字符串中的，以反斜杠开始，随后是作为转义字符的单个字母。例如，\n 是一个换行符。要在字符串中包含一个反斜杠，就要使用转义字符\\。

对于传递给 print()函数并且要显示到屏幕上的字符串，print()函数会自动在其末尾添加一个换行符。大多数时候，这是一种非常有用的快捷方式。但是，有时你并不想在末尾使用一个换行符。要改变这一点，只需要为 print()函数传递带有一个空字符串的 end 关键字参数。例如，要把没有换行符的“spam”打印到屏幕上，可以这样调用函数：print('spam', end='')。

Python 提供了众多的灵活方式在屏幕上显示文本。

第 6 章　Dragon Realm

本章主要内容：

- time.sleep()函数；
- 用 def 关键字创建自己的函数；
- 关键字 return；
- 布尔操作符 and、or 和 not；
- 真值表；
- 全局变量作用域和局部变量作用域；
- 形参和实参；
- 流程图。

6.1　函数

我们已经使用了一些函数，包括 print()、input()、random.randint()、str()和 int()。我们曾经调用这些函数去执行其中的代码。在本章中，我们将编写自己的函数以供程序来调用。函数就像是一个程序中的小程序。

函数使得我们可以多次运行相同的代码，而无需多次复制源代码。相反，可以把代码放入到一个函数中，多次调用这个函数。这还有另外一个好处，就是如果函数的代码中有一个错误，只需要在程序中的一个地方进行修改就可以了。

本章将要创建的游戏叫做“Dragon Realm”。游戏中有两个山洞，一个藏有宝藏宝藏，另一个则藏有厄运，玩家选择进入哪个山洞。

6.2　如何玩 Dragon Realm

在这个游戏中，玩家在一片到处是龙的陆地上。龙生活的洞穴里装满了它们收集的大量宝藏。有些龙很友善，愿意与你分享宝藏。而另外一些龙则很饥饿，会吃掉闯入它们洞穴的任何人。玩家站在两个洞前，一个山洞住着友善的龙，另一个山洞住着饥饿的龙。玩家必须从这两个山洞之间选择一个。

单击 File ▶ New Window，打开一个新的文件编辑器。在所出现的空白窗口中输入源代码，并且把它保存为 dragon.py。然后按下 F5 键运行这个程序。

6.3　Dragon Realm 游戏的运行示例

```
You are in a land full of dragons. In front of you,
you see two caves. In one cave, the dragon is friendly
and will share his treasure with you. The other dragon
is greedy and hungry, and will eat you on sight.
Which cave will you go into? (1 or 2)
1
You approach the cave...
It is dark and spooky...
A large dragon jumps out in front of you! He opens his jaws and...
Gobbles you down in one bite!
Do you want to play again? (yes or no)
no
```

6.4　Dragon Realm 的源代码

> **请注意！**
>
> 本书中的程序只能运行在 Python 3 上，而在 Python 2 上则不可以运行。当开启 IDLE 窗口时，它会在顶端显示类似于“Python 3.4.2”的信息。如果已经安装了 Python 2，也可以同时安装 Python3。请通过 https://python.org/download/下载 Python 3。

如果输入这些代码之后出现错误，请使用 http://invpy.com/diff/dragon 上的在线 diff 工具，把你输入的代码与书中的代码进行比较。

dragon.py

```
 1. import random
 2. import time
 3.
 4. def displayIntro():
 5.     print('You are in a land full of dragons. In front of you,')
 6.     print('you see two caves. In one cave, the dragon is friendly')
 7.     print('and will share his treasure with you. The other dragon')
 8.     print('is greedy and hungry, and will eat you on sight.')
 9.     print()
10.
11. def chooseCave():
12.     cave = ''
13.     while cave != '1' and cave != '2':
14.         print('Which cave will you go into? (1 or 2)')
```

```
15.     cave = input()
16.
17.     return cave
18.
19. def checkCave(chosenCave):
20.     print('You approach the cave...')
21.     time.sleep(2)
22.     print('It is dark and spooky...')
23.     time.sleep(2)
24.     print('A large dragon jumps out in front of you! He opens his jaws
and...')
25.     print()
26.     time.sleep(2)
27.
28.     friendlyCave = random.randint(1, 2)
29.
30.     if chosenCave == str(friendlyCave):
31.          print('Gives you his treasure!')
32.     else:
33.          print('Gobbles you down in one bite!')
34.
35. playAgain = 'yes'
36. while playAgain == 'yes' or playAgain == 'y':
37.
38.     displayIntro()
39.
40.     caveNumber = chooseCave()
41.
42.     checkCave(caveNumber)
43.
44.     print('Do you want to play again? (yes or no)')
45.     playAgain = input()
```

代码如何工作

让我们来详细地看一下源代码。

```
1. import random
2. import time
```

这个程序导入了两个模块。random 模块将提供 random.randint()函数，就像它在“猜数字”游戏中所做的那样。我们还将要用到 time 模块所包含的和时间相关的函数，所以第 2 行导入了 time 模块。

6.5　def 语句

```
4. def displayIntro():
5.     print('You are in a land full of dragons. In front of you,')
6.     print('you see two caves. In one cave, the dragon is friendly')
7.     print('and will share his treasure with you. The other dragon')
8.     print('is greedy and hungry, and will eat you on sight.')
9.     print()
```

第 4 行是一条 def 语句。这条 def 语句定义了一个新的函数，可以在随后的程序中调用该函数。当定义这个函数的时候，在它的 def 语句块中指定函数的指令。当调用这个函数的时候，会执行 def 语句块中的代码。

图 6-1 展示了一条 def 语句的各个部分。def 关键字后边紧跟着带圆括号的函数名称，然后是一个冒号（:）。def 语句后边的语句块叫作 def 语句块。

图 6-1　def 语句的各个部分

记住，def 语句并不执行代码，它只是定义了当调用这个函数时所要执行的代码。当执行到一条 def 语句时，会跳转到 def 语句块之后的第一行。

但是，当调用 displayIntro()函数时（如第 38 行），执行会进入到 displayIntro()函数中，即移动到 def 语句块的第一行。

```
38.     displayIntro()
```

然后所有 print()函数调用都会运行，并且会显示“You are in a land full of dragons...”这些介绍游戏的信息。

6.5.1　把函数定义放在哪里

函数的 def 语句和 def 语句块必须放在该函数调用之前。这就像是在使用变量之前必须先为变量赋一个值一样。如果把函数调用放在函数定义之前，就会得到一个错误。例如，来看如下的代码：

```
sayGoodbye()

def sayGoodbye():
print('Goodbye!')
```

如果试图运行它，Python 会给出类似于下面这样的错误信息：

```
Traceback (most recent call last):
  File "C:\Python34\spam.py", line 1, in <module>
sayGoodbye()
NameError: name 'sayGoodbye' is not defined
```

为了修复这个错误，把函数定义放在函数调用之前：

```
def sayGoodbye():
    print('Goodbye!')

sayGoodbye()
```

6.5.2 定义 chooseCave()函数

```
11. def chooseCave():
```

第 11 行定义了另一个名为 chooseCave()的函数。这个函数询问玩家想进入哪个山洞，是 1 还是 2。

```
12.     cave = ''
13.     while cave != '1' and cave != '2':
```

这个函数需要确定玩家输入的是 1 或者 2，而不是任何其他的内容。这里会有一个循环来持续询问玩家，直到他们输入两个有效答案中的一个。这就是所谓的输入验证。

第 12 行创建了一个名为 cave 的新变量，并且把一个空白字符串存储到其中。然后，第 13 行开始了一个 while 循环。循环的条件包含了我们之前所没有见过的一个叫做 and 的新操作符。就像-或*是算术操作符，==或!=是比较操作符，操作符 and 是一个布尔操作符。

6.6 布尔操作符

布尔逻辑负责处理那些 True 或 False 的事件。布尔操作符对值进行比较，并且得到一个布尔值。

我们来看一下这句话：“Cats have whiskers and dogs have tails”。“Cats have whiskers”是真的，“dogs have tails”也是真的，所以整个句子“Cats have whiskers and dogs have tails”是真的。

但是，“Cats have whiskers and dogs have wings”这句话将会为假。尽管“Cats have whiskers”是真的，但是狗并没有翅膀，所以“dogs have wings”为假。在布尔逻辑中，要么为真，要么为假。因为单词“and”，只有两部分都为真，整条语句才会为真。如果有一部

分为假或者两部分都为假，那么整条语句也为假。

6.6.1　and 和 or 操作符

在 Python 中，and 操作符的含义和单词 and 的含义相同。如果 and 关键字两边的布尔值都为 True，那么表达式的结果为 True。如果一个布尔值为 False 或者两个布尔值都为 False，那么表达式的结果为 False。

尝试在交互式 shell 中输入使用 and 操作符的表达式，如下所示：

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
>>> spam = 'Hello'
>>> 10 < 20 and spam == 'Hello'
True
```

or 操作符和 and 操作符类似，只不过两个布尔值中只要有一个为 True，它的结果就为 True。只有两个布尔值都为 False 时，or 操作符的结果才为 False。

尝试在交互式 shell 中输入如下语句：

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
>>> 10 > 20 or 20 > 10
True
```

6.6.2　not 操作符

not 操作符只能作用于一个值，而不能把两个值组合在一起。not 操作符的计算方式是对布尔值取反。表达式 not True 的结果是 False，not False 的结果为 True。

尝试在交互式 shell 中输入如下语句：

```
>>> not True
False
>>> not False
True
>>> not ('black' == 'white')
True
```

6.6.3 真值表

如果你忘记了布尔操作符的工作方式，可以查看如下的真值表：

表 6-1 and 操作符的真值表

A	and	B	结果	整个语句
True	and	True	为	True
True	and	False	为	False
False	and	True	为	False
False	and	False	为	False

表 6-2 or 操作符的真值表

A	or	B	结果	整个语句
True	or	True	为	True
True	or	False	为	True
False	or	True	为	True
False	or	False	为	False

表 6-3 not 操作符的真值表

not A	结果	整个语句
not True	为	False
not False	为	True

6.6.4 布尔操作符的运算

再看一下第 13 行：

```
13.     while cave != '1' and cave != '2':
```

这个条件由布尔操作符 and 连接两个部分构成。只有两个部分都为 True 时，条件才会

是 True。

第 1 次检查 while 语句的条件时，将 cave 设置为空字符串''。空字符串不等于字符串'1'，所以左边的结果为 True。空字符串也不等于字符串'2'，所以右边的结果也为 True。

所以该条件转换为 True and True。因为两个值都是 True，所以该条件最终的结果为 True。所以程序执行进入 while 语句块。

执行看上去如下所示（如果 cave 的值是空字符串）：

```
while cave != '1' and cave != '2':
                   ▼
while  ''   != '1' and cave != '2':
                   ▼
while     True    and cave != '2':
                   ▼
while     True    and '' != '2':
                   ▼
while     True    and True:
                   ▼
while             True:
```

6.6.5　获取玩家的输入

```
13.     while cave != '1' and cave != '2':
14.         print('Which cave will you go into? (1 or 2)')
15.         cave = input()
```

第 14 行询问玩家选择哪一个山洞。第 15 行让玩家输入响应并按下回车键。把玩家的输入存储到变量 cave 中。在执行了这行代码之后，执行循环回到 while 语句的顶部，重新检查条件。

如果玩家输入 1 或 2，那么 cave 将会是'1'或'2'（因为 input()总是返回字符串）。这会导致条件为 False，程序执行将继续并跳过这个 while 循环。例如，如果用户输入'1'，那么结果如下所示：

```
while cave != '1' and cave != '2':
                   ▼
while  '1'  != '1' and cave != '2':
                   ▼
while      False   and cave != '2':
                   ▼
while      False   and '1' != '2':
                   ▼
while      False   and True:
```

```
                    ▼
while               False:
```

但是，如果玩家输入 3 或 4 或 HELLO，这些输入将会是无效的。条件将为 True，并且进入 while 语句块中要求玩家重新输入。程序会持续询问，直到玩家输入 1 或 2。这将保证一旦执行开始，cave 变量总是会包含一个有效的输入。

6.7 返回值

```
17.     return cave
```

这是一条 return 语句，它只会在 def 语句块中出现。还记得 input()函数是如何返回玩家输入的一个字符串值吗？chooseCave()函数也会返回一个值。第 17 行返回变量 cave 中存储的字符串，也就是'1'或'2'。

一旦执行了 return 语句，程序执行会立即跳出 def 语句块（这就像 break 语句会让执行跳出 while 语句块一样）。程序执行将返回到调用该函数的那行代码。这个函数调用自身的结果就是返回值。

向下走，快速地看一下第 40 行：

```
40.     caveNumber = chooseCave()
```

当稍后在第 40 行代码中程序调用 chooseCave()函数时，把返回值存储到 caveNumber 变量中。while 循环保证 chooseCave()函数将只返回'1'或'2'作为其返回值。

所以当第 17 行返回一个字符串时，第 40 行的函数调用的结果就是这个字符串，然后把它存储到变量 caveNumber 中。

6.8 全局作用域和局部作用域

在程序终止后，程序的变量会被忘记。当执行位于一个函数调用中时，所创建的变量也同样位于一个函数调用中：当调用函数时，会创建该变量；当函数返回时，会忘记该变量。记住，函数就像是程序中的一种小程序。

当执行位于一个函数中时，你无法修改函数之外的变量（包括其他函数之中的变量）。这是因为这些变量存在于不同的“作用域”中。所有变量要么存在于全局作用域中，要么存在于一个函数调用的局部作用域中。

在所有函数之外的作用域叫做全局作用域（global scope）。在函数内部（特定函数调用期间）的作用域叫做局部作用域（local scope）。

整个程序中只有一个全局作用域。在全局作用域中创建的变量，能够在函数外部和函数

内部读取，但是只能够在所有函数之外（全局作用域中）修改。在函数调用中创建的变量，只能在函数调用过程中读取和修改。

我们可以在局部作用域中读取全局变量的值，但是无法在局部作用域中修改一个全局变量。在这种情况下，Python 实际所做的事情是创建和全局变量同名的一个局部变量。例如，有一个名为 spam 的全局变量，同时也可以有一个名为 spam 的局部变量。Python 会认为这是两个不同的变量。

来看一个示例，当试图在一个局部作用域中修改一个全局变量的时候，会发生什么情况。注释解释了正在发生的事情：

```
def bacon():
    # We create a local variable named "spam"
    # instead of changing the value of the global
    # variable "spam":
    spam = 99
    # The name "spam" now refers to the local
    # variable only for the rest of this
    # function:
    print(spam)     # 99

spam = 42 # A global variable named "spam":
print(spam) # 42
bacon() # Call the bacon() function:
# The global variable was not changed in bacon():
print(spam) # 42
```

当运行的这段代码的时候，输出如下所示：

```
42
99
42
```

创建变量的位置决定了它所在的作用域。当 Dragon Realm 程序第一次执行如下的代码行时：

```
12.     cave = ''
```

在 chooseCave()函数中创建了变量 cave。这表示在 chooseCave()函数的局部作用域中创建了该变量。当 chooseCave()返回时，会忘记该变量，如果再一次调用 chooseCave()函数，会重新创建该变量。在两次函数调用之间，不会记住一个局部变量的值。

6.9 形参

```
19. def checkCave(chosenCave):
```

程序定义的下一个函数是 checkCave()。注意圆括号之间的文本内容。这是一个形参（parameter），这是一个局部变量，当调用函数时，要将实参（argument）赋值给它。

还记得吧，在调用一些像 str()或者 randint()这样的函数的时候，要在圆括号中间传递一个或者多个实参：

```
>>> str(5)
'5'
>>> random.randint(1, 20)
    14
```

当调用 checkCave()函数时，我们也可以传递一个实参。把这个实参存储在名为 chosenCave 的一个新变量中。把这些变量也叫做形参。

例如，下面是一个简短的程序，它展示了如何定义带有一个形参的函数：

```
def sayHello(name):
    print('Hello, ' + name + '. Your name has ' + str(len(name)) + ' letters.')

sayHello('Alice')
sayHello('Bob')
spam = 'Carol'
sayHello(spam)
```

如果运行这个程序，结果如下所示：

```
Hello, Alice. Your name has 5 letters.
Hello, Bob. Your name has 3 letters.
Hello, Carol. Your name has 5 letters.
```

当调用 sayHello()的时候，把实参赋给了形参 name。形参只是普通的局部变量。就像所有的局部变量一样，当函数调用返回时，将会忘记圆括号中的值。

6.9.1 显示游戏结果

回到游戏的源代码：

```
20.     print('You approach the cave...')
21.     time.sleep(2)
```

time 模块有一个名为 sleep()的函数，它会暂停程序。第 21 行代码传递了整数值 2 作为参数，所以 time.sleep()函数会暂停程序 2 秒钟。

```
22.     print('It is dark and spooky...')
23.     time.sleep(2)
```

这次代码打印了更多的文本，并且又等待了 2 秒钟。这些较短的暂停为程序增加了悬念，而不是立刻显示所有的文本。在第 5 章的 Jokes 程序中，我们调用 input()函数来暂停程序，直到玩家按下回车键。在这里，玩家不必做任何事情，只是等待了几秒钟。

```
24.     print('A large dragon jumps out in front of you! He opens his jaws
and...')
25.     print()
26.     time.sleep(2)
```

接下来会发生什么？以及程序如何决策？这会在下一节中解释。

6.9.2　决定哪个山洞有友善的龙

```
28.     friendlyCave = random.randint(1, 2)
```

第 28 行调用了 random.randint()函数，它将返回 1 或者 2。把这个整数值存储到变量 friendlyCave 中，这是友善的龙所在的山洞。

```
30.     if chosenCave == str(friendlyCave):
31.         print('Gives you his treasure!')
```

第 30 行代码判断在 chosenCave 变量（'1' 或'2'）中存储的玩家所选择的山洞，是否和友善的龙所在的山洞相同。

但是 friendlyCave 中的值是一个整数，因为 random.randint()函数返回的是整数。我们不能用==来比较字符串和整数，因为它们之间永远不会相等。'1'不等于 1，'2'也不等于 2。

因此，将 friendlyCave 传递给了 str()函数，它会返回 friendlyCave 的字符串值。通过这种方式，两个值会具有相同的数据类型，并且能够进行有意义的比较。也可以使用如下代码把 chosenCave 转换成整数值：

```
    if int(chosenCave) == friendlyCave:
```

如果该条件为 True，第 31 行代码告诉玩家他们赢得了宝藏。

```
32.     else:
33.         print('Gobbles you down in one bite!')
```

第 32 行代码是一条 else 语句。else 语句只能跟在 if 语句块之后。如果 if 语句的条件为 False，就会执行 else 语句块。程序表达的意思是：“如果这个条件为真，那么执行 if 语句块，否则执行 else 语句块”。

记住，在 else 关键字后边要写一个冒号（:）。

6.9.3 程序的主要部分从哪里开始

```
35. playAgain = 'yes'
36. while playAgain == 'yes' or playAgain == 'y':
```

第 35 行是第一个不位于 def 语句块中的代码行。这行代码是程序主要部分的开始。前边的 def 语句只是定义了函数。它们并不会运行函数中的代码。

第 35 行和第 36 行代码建立了一个循环，游戏代码的剩余部分都在这个循环中。在游戏结尾，如果玩家还想再玩，可以通过输入来告知游戏。如果他们这么做，执行就进入 while 循环，再次运行整个游戏。如果他们没有这么做，while 语句的条件将为 False，执行会到达程序的末尾并且终止游戏。

第一次执行到这条 while 语句，第 35 行将会把'yes'赋给变量 playAgain。这意味着条件为 True。这就保证了执行至少能够进入循环一次。

6.9.4 在程序中调用函数

```
38.     displayIntro()
```

第 38 行调用了 displayIntro()函数。这不是一个 Python 内建函数，而是我们之前在第 4 行定义的函数。当调用这个函数时，程序执行跳到 displayIntro()函数的第 1 行，也就是整个程序的第 5 行代码。当该函数中所有的代码行都执行完后，执行跳回到第 38 行，继续向下移动。

```
40.     caveNumber = chooseCave()
```

第 40 行也调用了我们定义的一个函数。记住，chooseCave()函数让玩家输入他们想要进入的山洞。当执行第 17 行的 return cave 时，程序执行跳转回到第 40 行，chooseCave()调用的结果就是这个返回值。把这个返回值存储到名为 caveNumber 的一个新变量中。然后程序执行移动到第 42 行。

```
42.     checkCave(caveNumber)
```

第 42 行代码调用了 checkCave()函数，把 caveNumber 中的值作为参数传递给该函数。

这不仅执行会跳转到第 20 行，而且会把 caveNumber 中的值复制给 checkCave()函数中的形参 chosenCave。根据玩家选择进入的山洞，这个函数将要显示'Gives you his treasure!'或者'Gobbles you down in one bite!'。

6.9.5　询问玩家要不要再玩一局

```
44.     print('Do you want to play again? (yes or no)')
45.     playAgain = input()
```

无论玩家是输是赢，游戏都会询问他们要不要再玩一局。把玩家输入的字符串存储到变量 playAgain 中。第 45 行是 while 语句块的最后一行代码，所以玩家跳转回到第 36 行，检查 while 循环的条件 playAgain == 'yes' or playAgain == 'y'。

如果玩家输入的字符串是'yes'或'y'，那么执行将在第 38 行再次进入循环。

如果玩家输入的是'no'或'n'，或者其他类似于'Abraham Lincoln'的文本，那么条件将为 False。程序执行将从 while 语句块之后的那行开始继续。但是，因为 while 语句块之后没有其他的代码行了，所以程序结束了。

要注意一件事情：字符串'YES'不等于字符串'yes'。如果玩家输入的是字符串'YES'，那么 while 语句的条件所得到的结果为 False，程序仍然会终止。本书后边的程序将会向你展示如何避免这个问题。

你刚刚完成了自己的第 2 个游戏！在 Dragon Realm 中，我们使用了在“猜数字”中学过的许多知识，并且学会一些新的技巧。如果你还不理解这个程序中的一些概念，那么重新回顾一下源代码的每一行，并且尝试修改源代码，看看程序会如何改变。

在下一章中，我们不会创建游戏，而是介绍如何使用一种叫做调试器的 IDLE 功能。

6.10　设计游戏

Dragon Realm 是一个简单的游戏。本书中其他的游戏将会更复杂一些。在开始编写代码之前，把想要让游戏或者程序做的每一件事情都写下来，这么做有时候会有所帮助。这个过程叫作“设计程序”。

例如，绘制流程图可能是有所帮助的。流程图是这样的一种图，它展示了程序中发生的各种可能的动作，以及哪一种动作会导致另一种动作发生。图 6-2 是 Dragon Realm 的流程图。

要查看在游戏中发生了什么，把手指放在“开始”框上。然后，从这个框顺着箭头到下一个框。手指就像是程序执行一样。当手指到了“结束”框，程序就结束了。

当到了“查看山洞中是友善的龙还是饥饿的龙”框时，可以去“玩家获胜”框，也可以去“玩家输了”框。这种分支点展示了程序如何做不同的事情。无论哪种方式，这两条路径都会在“询问是否再玩一次”框结束。

6.11 本章小结

在 Dragon Realm 游戏中，我们创建了自己的函数。函数是程序中的小程序。当调用函数时，这个函数中的代码才会运行。通过把代码分解到函数中，我们就可以把代码组织成为更小、更容易理解的部分。

当调用函数时，实参是复制到形参中的值。函数调用本身会把返回值作为结果。

我们还介绍了变量的作用域。在函数中创建的变量存在于局部作用域中，在所有函数之外创建的变量存在于全局作用域中。全局作用域中的代码不能使用局部变量。如果一个局部变量的名称和全局作用域中的一个变量的名称相同，那么 Python 会认为这个局部变量是另一个不同的变量，并且给这个局部变量赋一个新的值也并不会改变全局变量中的值。

变量作用域可能看上去有点复杂，但是对于把函数组织成和程序的其他部分区分开的代码块来说，变量作用域非常有用。因为每个函数都有自己的局部作用域，这就可以确保一个函数中的代码不会导致其他函数中的 bug。

几乎每个程序都使用函数，因为它们非常有用。通过理解函数如何工作，你可以减少许多录入，并且更容易修正错误。

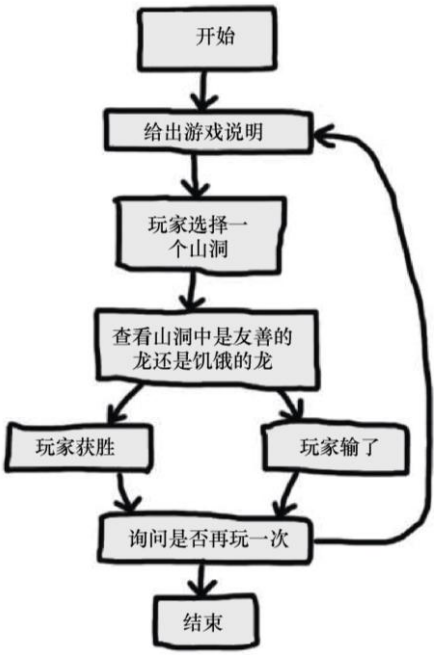


图 6-2 Dragon Realm 游戏的流程图

第 7 章　使用调试器

本章主要内容：

- 3 种错误类型；
- IDLE 的调试器；
- 单步进入、单步跳过和单步退出；
- Go 按钮和 Quit 按钮；
- 断点。

7.1　Bug!

“在两个场合我被问到：‘巴贝奇先生，请你告诉我，如果你给机器输入了错误的数字，那么，是否还能得到正确的答案？’。我并不能正确领会这类想法。”

——查尔斯·巴贝奇，19 世纪可编程计算机概念的先驱

如果我们输入了错误的代码，计算机不可能给出正确的程序。计算机程序总是做你告诉它要做的事情，但是你告诉程序要做的事情可能和你真正想要让程序做的事情有所不同。这些错误就是计算机程序中的 bug。当程序员没有认真考虑这个程序要做什么的时候，bug 就出现了。在程序中，可能会出现 3 种类型的 bug。

- 语法错误：这是一种由错误输入而引发的错误。当 Python 解释器看到一个语法错误时，是由于代码没有以正确的 Python 语言编写而引起的。即使只有一个语法错误，Python 程序也不会运行。
- 运行时错误：程序运行时产生的错误。程序将一直工作，直到它碰到有错误的代码行，然后程序将终止运行（这叫作崩溃）并给出一条错误消息。Python 解释器将会显示一条“traceback”消息，并且显示是哪一行出现了问题。
- 语义错误：这是最难修复的错误。这些 bug 不会让程序崩溃，但是程序不会做程序员想要让它做的事情。例如，如果程序员希望变量 total 是变量 a、b 和 c 的值的总和，但是写成了 total = a * b * c，那么 total 中的值将会是错误的。稍后这可能会让程序崩溃，但是哪里出现了语义错误并不是立即一目了然。

查找程序中的 bug 可能很难，甚至你可能根本不会注意到有 bug！当运行程序时，你可能会发现有时候函数没有像预期的那样被调用，或者可能被调用太多次。可能一个 while 循环的条件编写有误，所以它循环了错误的次数（程序中永远不会退出的循环叫作无限循环，这是一种 bug。要停止这个程序，可以在交互式 shell 中按下 Ctrl-C 来结束程序）。如果不小

心，你的编码中可能就会出现类似这样的错误。

实际上，在交互式 shell 中，通过输入如下代码可以创建一个无限循环。最后，你需要按下两次输入键，让交互式 shell 知道，你已经录入完了一个 while 语句块：

```
>>> while True:
...     print('Press Ctrl-C to stop this infinite loop!!!')
...
```

现在按住 Ctrl 键，并按下 C 键来终止程序。交互式 shell 看上去如下所示：

```
Press Ctrl-C to stop this infinite loop!!!
Press Ctrl-C to stop this infinite loop!!!
Press Ctrl-C to stop this infinite loop!!!
Press Ctrl-C to stop this infinite loop!!!
Press Ctrl-C to stop this infinite loop!!!
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    while True: print('Press Ctrl-C to stop this infinite loop!!!')
KeyboardInterrupt
```

7.2 调试器

可能很难弄清楚代码是如何导致 bug 的。代码行执行的很快，变量中的值也经常改变。调试器就是让我们可以按照 Python 执行代码时的相同顺序来单步执行代码的一个程序。调试器还展示了执行每一步时变量中存储的值。

启动调试器

在 IDLE 中，打开我们第 6 章中创建的 Dragon Realm 游戏。在打开 dragon.py 文件之后，点击 Debug ▶Debugger，打开 Debug Control 窗口（如图 7-1 所示）。

现在，当按下 F5 键运行 Dragon Realm 游戏时，就会触发 IDLE 的调试器。这叫作“在调试器下”运行程序，如图 7-2 所示。在 Debug Control 窗口中，勾选 Source 和 Globals 复选框。

当在调试器下运行 Python 程序时，在执行第一条指令前，程序将会停止。如果点击文件编辑器窗口的标题栏（我们已经在 Debug Control 窗口中勾选了 Source 复选框），第 1 条指令以灰色高亮显示。Debug Control 窗口显示执行到第 1 行，这一行的内容是 import random。

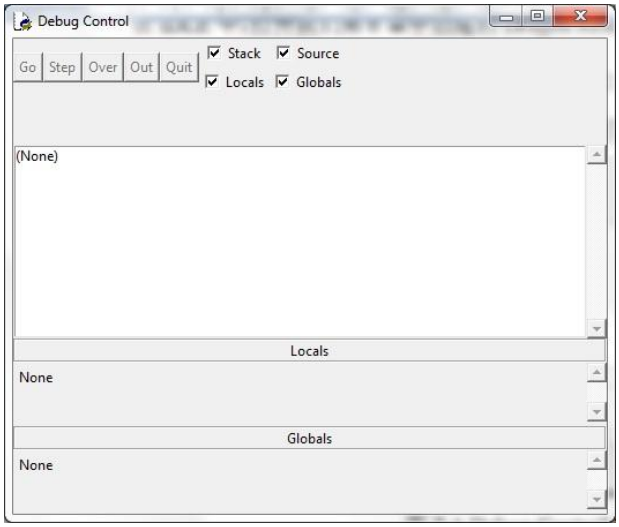


图 7-1　Debug Control 窗口

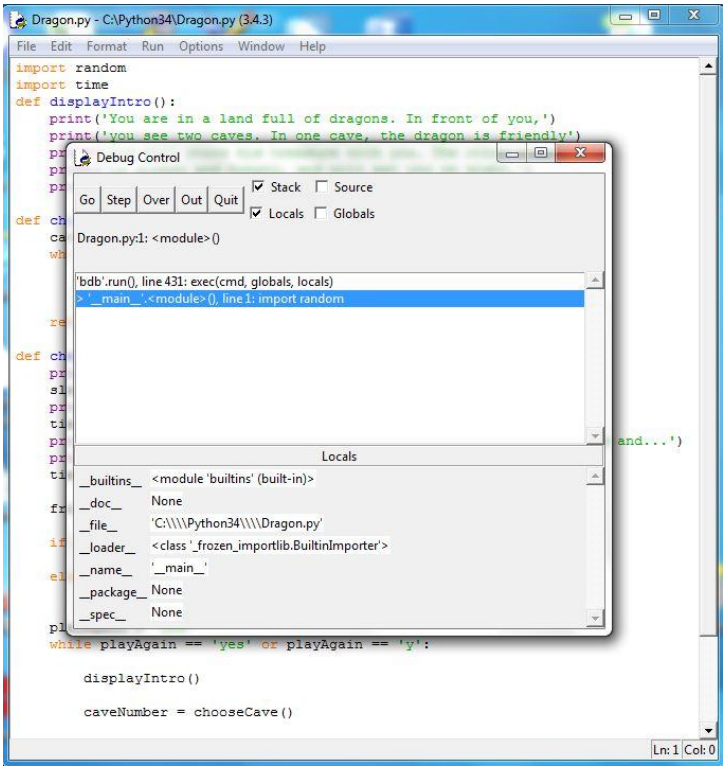


图 7-2　在调试器下运行 Dragon Realm 游戏

7.3 单步执行

调试器允许我们每次执行一条指令。这叫做单步执行。要执行单条指令，点击 Debug 窗口中的 Step 按钮。现在就这么做。Python 会执行 import random 指令，然后在执行下一条指令前停下来。Debug Control 窗口会显示当前执行到了第 2 行，也就是 import time。现在，点击 Quit 按钮来终止这个程序。

在调试器下运行 Dragon Real 游戏时，当按下 Step 按钮，所发生的一切事情就像上面所概述的那样。按下 F5 键再次开始运行 Dragon Realm，然后执行如下指令：

1．点击两次 Step 按钮，来运行两条 import 语句。

2．再次点击 3 次 Step 按钮，执行 3 条 def 语句。

3．再次点击 Step 按钮，定义变量 playAgain。

4．点击 Go 按钮来运行剩余的程序，或者点击 Quit 按钮来终止程序。

当点击 Debug Control 窗口中的 Step 按钮时，该窗口会显示哪行代码将要执行。调试器跳过了第 3 行，因为这是一个空白行。注意，调试器只能向前单步执行，无法后退。

7.3.1 Globals 区域

在 Debug Control 窗口中的 Globals 区域，可以看到所有的全局变量。记住，全局变量是在所有的函数之外创建的变量（也就是，创建于全局作用域中）。

执行这 3 条 def 语句并且定义了函数之后，函数将出现在 Debug Control 窗口的 Globals 区域。

在 Globals 区域中，函数名称后边的文本看上去类似于“<function checkCaveat0x012859B0>”。模块名称后边也有令人混淆的文本，诸如“<module 'random' from 'C:\\Python31\\lib\\random.pyc'>”。我们不需要知道这对程序调试有什么含义。只要看到函数和模块在 Globals 区域中，我们就知道函数是否已经被定义，或者模块是否已被导入。

也可以忽略掉 Globals 区域中的__builtins__、__doc__和__name__等行（这些变量在每个 Python 程序中都会出现）。

当创建了 playAgain 变量，它会出现在 Globals 区域中。变量名后边是字符串'yes'。当程序运行时，调试器允许我们看到程序中所有变量的值。这对修复 bug 很有帮助。

7.3.2 Locals 区域

这里还有一个 Locals 区域，它为我们展示了局部作用域变量以及这些变量的值。当程序在一个函数中执行时，Locals 区域中将只有函数中的变量；当在全局作用域中执行时，这个

区域是空白的。

7.3.3　Go 按钮和 Quit 按钮

如果你懒得反复点击 Step 按钮，只是想要程序正常的运行，那就点击 Debug Control 窗口上部的 Go 按钮。这将告诉程序正常运行而不是单步执行。

要彻底终止程序，只需要点击 Debug Control 窗口顶部的 Quit 按钮。程序将立刻跳出执行。如果必须要从程序的起点重新启动调试，这会很有用。

7.3.4　单步进入、单步跳过和单步退出

用调试器启动 Dragon Realm 程序。持续单步执行，直到调试器到达第 38 行。如图 7-3 所示，这行的内容是 displayIntro()。当再次点击 Step 时，调试器将会进入到这个函数调用中，并且出现在第 5 行，也就是 displayIntro()函数的第 1 行。这种单步执行叫做单步进入（Step into）。它有别于接下来要介绍的单步跳过。

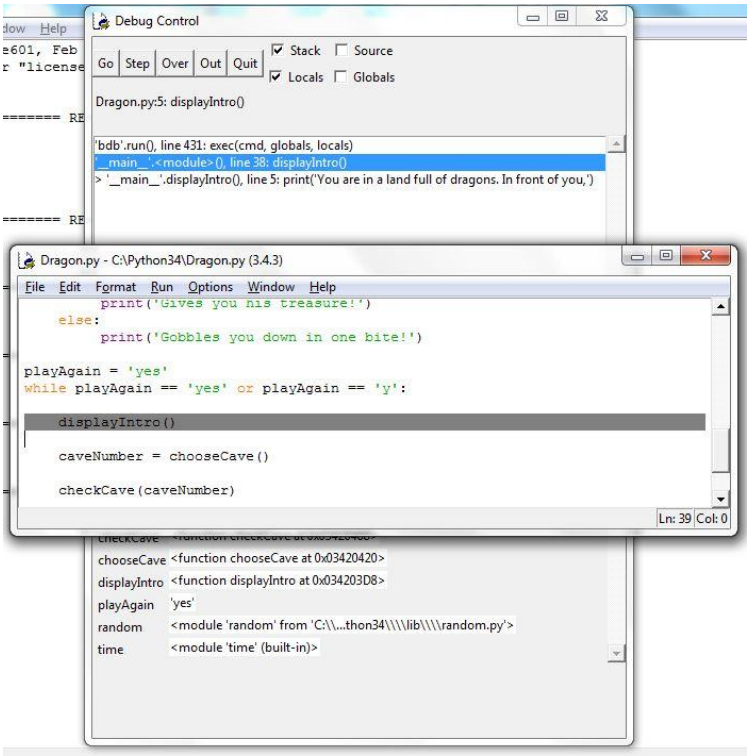


图 7-3　保持单步执行直到到达第 38 行

当执行在第 5 行暂停时，再次点击 Step 将会单步进入到 print()函数中。print()函数是 Python 的一个内建函数，所以使用调试器单步跟踪它没有什么用。Python 自己的函数，诸如 print()、input()、str()或 random.randint()等，都已经很认真地进行过错误检查了。我们可以假设它们不是程序中引起 bug 的部分。

所以，我们不想浪费时间对内建的 print()函数进行单步跟踪。所以，不要点击 Step 单步进入到 print()函数的代码中，而是点击 Over。这会单步跳过 print()函数中的代码。print()中的代码会以正常的速度执行，然后一旦执行从 print()中返回，调试器将会暂停。

单步跳过（Step over）是跳过单步跟踪函数中的代码的一种方便的方式。调试器现在暂停于第 40 行，也就是 caveNumber = chooseCave()。

再点一次 Step 按钮，单步进入到 chooseCave()函数中。持续单步跟踪代码，直到第 15 行调用 input()函数。程序将等待，直到在交互式 shell 中输入一个响应，就像我们通常运行该程序那样。如果现在尝试点击 Step 按钮，什么都不会发生，因为程序在等待键盘的响应。

在交互式 shell 窗口中点击 Back 按钮，然后输入想要进入的山洞。在输入之前，闪烁的光标必须在交互式 shell 中的下方。否则，你输入的文本无法显示。

一旦按下回车键，调试器将再次继续单步执行代码行。点击 Debug Control 窗口的 Out 按钮。这叫做单步退出（Step out），因为这将导致调试器单步跳过尽可能多的行，直到执行从它所在函数中返回。当跳出之后，执行将停在调用函数之后的那一行。

例如，在 displayIntro()函数中的第 6 行点击 Out 按钮，将会单步跳出函数，返回到调用 displayIntro()之后的那一行。单步退出可以省去反复点击 Step 的操作，而直接跳出这个函数。

如果不在函数中，点击 Out 按钮将导致调试器执行程序中所有剩余的代码行。这和点击 Go 按钮的行为是相同的。

这里重新回顾一下每个按钮所做的事情：

- Go：像通常一样执行剩余的代码，或者直到到达一个断点（断点稍后介绍）。
- Step：单步执行一条指令。如果执行的代码行是一个函数调用，调试器将会单步进入到函数中。
- Over：单步执行一条指令。如果执行的代码行是一个函数调用，调试器不会单步进入到这个函数中，而是跳过这个函数调用。
- Out：当点击 Out 按钮时，持续跳过代码行，直到调试器离开当前所在的函数。也就是跳出这个函数。
- Quit：立即终止程序。

7.4 查找 Bug

调试器可以帮助我们找到程序中导致 bug 的原因。例如，下面是一个有 bug 的小程序。

这个程序提出一个随机加法问题让用户解答。在交互式 shell 窗口中，点击 File，然后点击 New Window，会打开一个新的文件编辑器窗口。在窗口中输入这个程序，然后把它保存为 buggy.py。

buggy.py

```
1. import random
2. number1 = random.randint(1, 10)
3. number2 = random.randint(1, 10)
4. print('What is ' + str(number1) + ' + ' + str(number2) + '?')
5. answer = input()
6. if answer == number1 + number2:
7.     print('Correct!')
8. else:
9.     print('Nope! The answer is ' + str(number1 + number2))
```

即使你已经找出了 bug，也请像上面一样输入程序。然后按下 F5 键，尝试运行程序。这是使用两个随机数的一道简单算术题，要求你把它们加起来。当运行这个程序时，看上去可能如下所示：

```
What is 5 + 1?
6
Nope! The answer is 6
```

有一个 bug! 这个程序没有崩溃，但是它也没有正常工作。尽管用户输入了正确的答案，但是程序还是说用户出错了。

在调试器下运行程序将有助于查找导致 bug 的原因。在交互式 shell 窗口的上部，勾选全部 4 个复选框（Stack、Source、Locals 和 Globals）。这会使得 Debug Control 窗口提供最多的信息。然后在文件编辑器窗口按下 F5 键来运行程序。这次它会在编辑器窗口下运行。

```
1. import random
```

调试器从 import random 行开始。这里没有什么特别事情发生，所以只要点击 Step 按钮来执行代码。我们会看到把 random 模块添加到了 Globals 区域中。

```
2. number1 = random.randint(1, 10)
```

再次点击 Step 按钮执行第 2 行。将会出现带有 random.py 文件的一个新的文件编辑器窗口。我们已经单步进入到 random 模块中的 randint()函数中。Python 的内建函数不是导致 bug 的原因，所以点击 Out 按钮跳出 randint()函数，并返回到程序中。然后关闭 random.py 文件的窗口。

```
3. number2 = random.randint(1, 10)
```

接下来，可以点击 Over 按钮来跳过 randint()函数，而不是进入到函数中。第 3 行也是一个 randint()函数调用。点击 Over 按钮跳过这行代码。

```
4. print('What is ' + str(number1) + ' + ' + str(number2) + '?')
```

第 4 行是一个 print()调用，它向玩家展示了随机数。我们甚至可以在打印那些数字之前，就知道程序将要打印的数字内容！只要看一下 Debug Control 窗口的 Globals 区域即可。我们可以看到变量 number1 和 number2，紧接着是存储在变量中的整数值。

变量 number1 的值是 4，变量 number2 的值是 8。当我们点击 Step 按钮的时候，程序将会调用 print()函数以字符串的形式显示这些值。str()函数将连接这些整数的字符串版本。当我们运行调试器时，它看上去如图 7-4 所示（随机数可能会有所不同）。

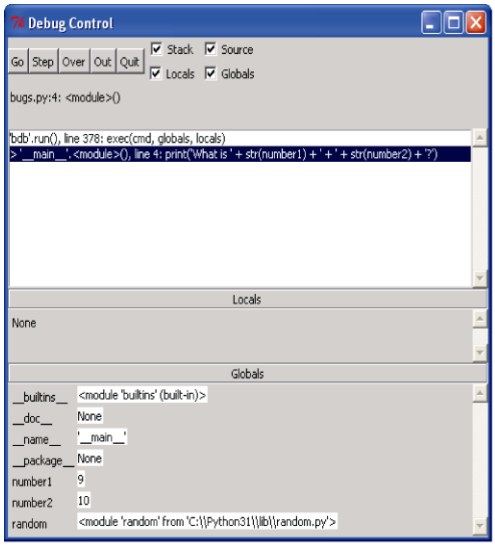


图 7-4 把 number1 设置为 4，把 number2 设置为 8

```
5. answer = input()
```

在第 5 行点击 Step 按钮，将会执行 input()。调试器等待，直到玩家输入一个响应。输入正确的答案（在这个例子中是 12）到交互式 shell 的窗口中。调试器将继续执行并向下到达第 6 行。

```
6. if answer == number1 + number2:
7.     print('Correct!')
```

第 6 行是一条 if 语句。条件是 answer 中的值必须等于 number1 和 number2 之和。如果条件是 True，那么调试器将会移到第 7 行。如果条件是 False，调试器将会移到第 9 行。再点击一次 Step，会知道它移动到哪里。

```
8. else:
9.     print('Nope! The answer is ' + str(number1 + number2))
```

调试器现在在第 9 行！发生了什么？if 语句中的条件肯定只会是 False。再看一下 number1、number2 和 answer。你会注意到 number1 和 number2 是整数，所以它们的和也一定是整数。但是 answer 是一个字符串。

这意味着 answer == number1 + number2 的计算结果是'12' == 12。一个字符串值和一个整数值永远也不会相等，所以条件结果为 False。

这就是程序中的 bug，即代码本来应该是 int(answer)，而现在是 answer。把第 6 行改为 int(answer) == number1 + number2，并且再次运行程序。

```
What is 2 + 3?
5
Correct!
```

这次，程序正确地运行了。再次运行程序，并且故意输入一个错误答案。这将完整地测试该程序。我们已经调试了这个程序！记住，计算机将严格按照你输入的程序来执行，即使你的输入和预期并不相符。

7.5　断点

单步跟踪代码可能还是太慢了。我们经常想要让程序以正常速度运行，直到到达某一特定的行。断点设置于某一行，我们想要一旦执行到这一行，调试器就开始控制。如果你认为第 17 行代码有问题，只要在第 17 行（或者可能是在此之前的几行）设置一个断点即可。

当执行到这一行代码时，调试器将“中断进入调试”。然后我们可以单步跟踪代码行来查看发生了什么。点击 Go 按钮将正常执行程序，直到到达了另一处断点或者程序末尾。

要设置一个断点，在文件编辑器中右键点击代码行，从出现的菜单中选择 Set Breakpoint。文件编辑器将用黄色高亮显示该行。你可以根据需要来设置多个断点。要移除断点，点击该行，在弹出的菜单中选择 Clear Breakpoint。

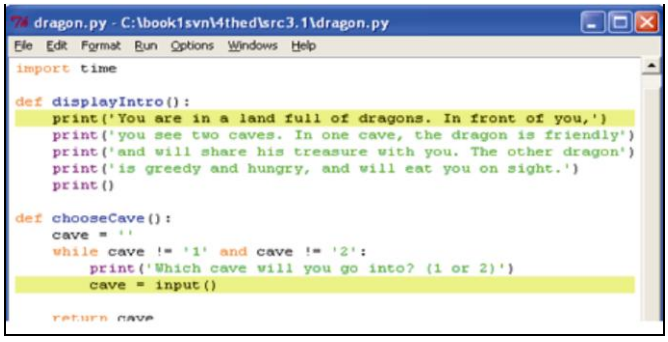


图 7-5　设置了两个断点的文件编辑器

7.6 使用断点的示例

下面调用 random.randint(0, 1)函数来模拟抛硬币的一个程序。函数返回整数 1 表示“正面”，返回整数 0 表示“反面”。变量 flips 将会记录抛了多少次硬币。变量 heads 将会记录得到了多少个正面。

这个程序将会“抛 1000 次硬币”。如果人工来做要花费一个多小时，但是计算机可以在 1 秒钟就完成！在文件编辑器中输入如下代码，并且把它们保存为 coinFlips.py。如果输入这些代码之后出现错误，请使用 http://invpy.com/diff/coinflips 上的在线 diff 工具，把你输入的代码与书中的代码进行比较。

coinFlips.py

```
 1. import random
 2. print('I will flip a coin 1000 times. Guess how many times it will come
up heads. (Press enter to begin)')
 3. input()
 4. flips = 0
 5. heads = 0
 6. while flips < 1000:
 7.     if random.randint(0, 1) == 1:
 8.         heads = heads + 1
 9.     flips = flips + 1
10.
11.     if flips == 900:
12.         print('900 flips and there have been ' + str(heads) + ' heads.')
13.     if flips == 100:
14.         print('At 100 tosses, heads has come up ' + str(heads) + ' times
so far.')
15.     if flips == 500:
16.         print('Half way done, and heads has come up ' + str(heads) +
' times.')
17.
18. print()
19. print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')
20. print('Were you close?')
```

程序运行得很快。等待用户按下回车键所花费的时间比抛硬币的时间还要长。假设你想要查看每一次抛硬币的结果。在交互式 shell 的窗口中，点击 Debug ▶ Debugger，打开 Debug Control 窗口。然后按下 F5 键来运行这个程序。

在调试器中，这个程序从第 1 行开始。在 Debug Control 窗口中，按下 3 次 Step 执行前 3 行（也就是第 1 行、第 2 行和第 3 行）。我们会注意到这个按钮将变得不可用，因为调用

了 input()函数，交互式 shell 窗口要等待用户输入一些内容。点击交互式 shell 窗口，并且按下回车键（一定要点击交互式 shell 窗口下方的文本，否则 IDLE 可能接收不到键盘输入）。

可以多点几次 Step 按钮，但是你会发现遍历整个程序的话，将需要相当长的时间。相反，我们在第 12 行、第 14 行和第 16 行设置了断点。文件编辑器将会高亮显示这些行，如图 7-6 所示。

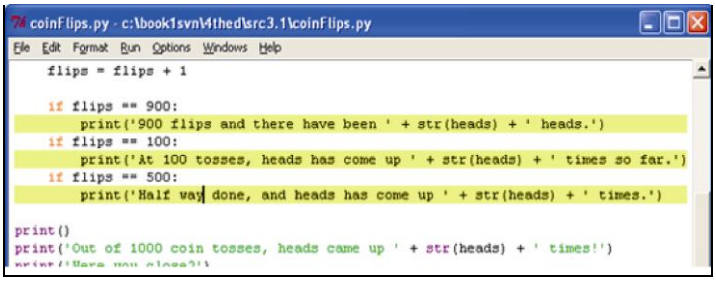


图 7-6　设置了 3 个断点

设置了断点之后，在 Debug Control 窗口点击 Go 按钮。程序按照正常的速度运行，直到到达下一个断点。当 flips 设置为 100 时，第 13 行的 if 语句的条件为 True。这会导致第 14 行（这里设置了一个断点）执行，它告诉调试器停止程序并接管。查看一下 Debug Control 窗口中的 Globals 区域，会看到变量 flips 和 heads 的值。

再次点击 Go 按钮，程序将继续，直到到达第 16 行处的下一个断点。再查看一下变量 flips 和 heads 中的值的变化。

如果再次点击 Go 按钮，执行将继续，直到到达下一个断点，也就是第 12 行。

7.7　本章小结

写出程序只是编程工作的第一部分。第二部分工作是确保编写的代码能够真正地工作。调试器让我们可以单步跟踪代码。我们可以查看哪些代码以何种顺序执行，以及变量中所包含的值。当这一切太慢了时，我们可以设置断点来让调试器在想要停止的代码行上停止。

使用调试器是了解程序正在做什么的一种很好的方式。虽然本书针对所有游戏代码给出了说明，但是调试器可以帮助你自己找出更多答案。

第 8 章　流程图

本章主要内容：

- 如何玩 Hangman；
- ASCII 字符图（ASCII Art）；
- 用流程图来设计一个程序。

在本章中，我们将设计一个 Hangman 游戏。这个游戏比之前的游戏都更复杂，但是也更有趣。因为这个游戏比较高级，所以我们应该先通过创建一个流程图（稍后介绍流程图）来仔细规划。在下一章中，我们将真正地编写 Hangman 的代码。

8.1　如何玩 Hangman

Hangman 是一个双人游戏，通常用纸和铅笔来玩。一个玩家想好一个单词，然后在纸上为单词的每个字母画一个空格。然后，第二个玩家尝试猜测这个单词中可能包含的字母。

如果第二个玩家猜对了，第一个玩家在正确的空格处填写这个字母。如果没有猜对，第一个玩家就画出火柴人的身体的一部分。如果第二个玩家在火柴人画好之前能够猜对单词中的所有字母，他就获胜。但是，如果第二个玩家不能按时完成猜测，他就失败了。

8.2　Hangman 的运行示例

当玩家运行第 9 章中所编写的 Hangman 程序的时候，可能会看到如下的示例。玩家输入的文本用粗体显示。

```
H A N G M A N
  +---+
  |   |
      |
      |
      |
      |
=========
Missed letters:

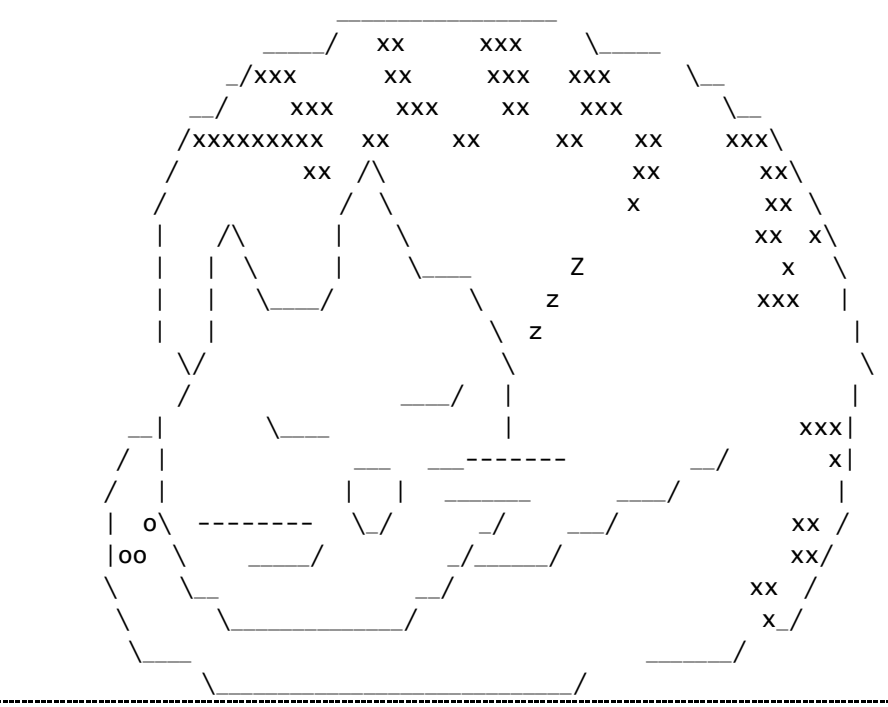
_ _ _
Guess a letter.
a
  +---+
```

```
  |   |
      |
      |
      |
      |
=========
Missed letters:
_ a _
Guess a letter.
o
  +---+
  |   |
  O   |
      |
      |
      |
=========
Missed letters: o
_ a _
Guess a letter.
r
  +---+
  |   |
  O   |
  |   |
      |
      |
=========
Missed letters: or
_ a _
Guess a letter.
t
  +---+
  |   |
  O   |
  |   |
      |
      |
=========
Missed letters: or
_ a t
Guess a letter.
a
You have already guessed that letter. Choose again.
Guess a letter.
```

```
c
Yes! The secret word is "cat"! You have won!
Do you want to play again? (yes or no)
no
```

8.3 ASCII 字符图

Hangman 的图形是打印到了屏幕上的键盘字母。这种类型的图形叫做 ASCII 字符图（ASCII Art），它是绘文字（emojii）的一种早期形式。如下是用 ASCII 字符图绘制的猫。



8.4 用流程图来设计一个程序

这个游戏比我们目前为止见到过的其他游戏都复杂，所以我们要花一点时间来考虑如何把它整合起来。首先，我们将创建一个流程图（就像第 6 章所做的那样），帮助我们将程序要做的事情可视化。本章将重点介绍什么是流程图以及它为什么很有用。下一章将详细介绍 Hangman 游戏的源代码。

流程图（flow chart）是一幅图形，使用箭头连接的方框来展示一系列的步骤。每个方

框表示一个步骤，箭头展示了这些步骤会导致其他的哪一些步骤。把手指放在流程图中“开始”方框上，然后按照箭头指到其他的方框，追踪程序，直到到达“结束”方框。

图 8-1 是 Hangman 的一个完整的流程图。我们只能从一个方框顺着箭头的方向移动到另一个方框，而不能往回走，除非有第 2 个箭头往后指，就像“玩家已经猜对了这个字母”方框。

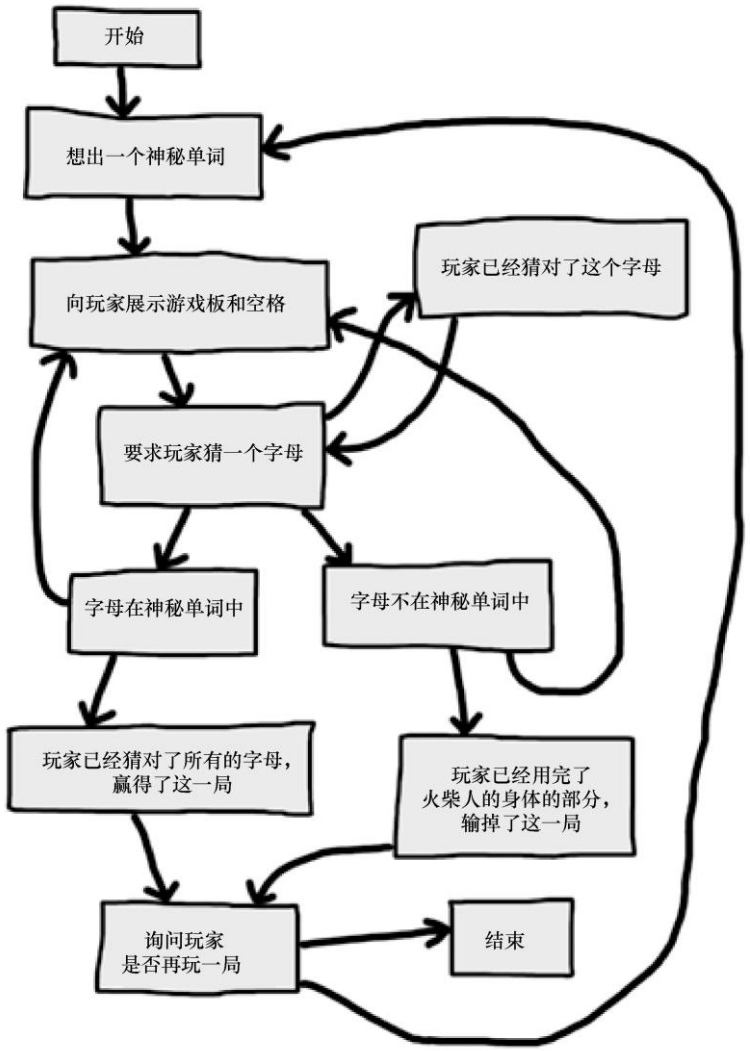


图 8-1　在 Hangman 游戏中所发生事情的完整流程图

当然，不一定非要生成流程图，可以直接开始编写代码。但是通常一旦开始编程，就会

想到必须增加和修改的事项。可能最后还不得不删除一些代码，这将会浪费精力。为了避免这种情况，最好在开始编写代码之前，计划好程序将会如何工作。

8.5 生成流程图

流程图并不总是非要这个样子。只要你能明白所创建的流程图，在开始编写代码时就会有所帮助。流程图最初只有“开始”方框和“结束”方框，如图 8-2 所示。

现在开始考虑当我们玩 Hangman 的时候会发生什么。首先，计算机会想到一个神秘的单词。然后，玩家将猜测字母。为这些事件添加方框，如图 8-3 所示。在每个流程图中，新的方框外侧有一圈虚线包围。

箭头显示了程序应该移动的顺序。也就是，首先程序应该想到一个神秘单词，然后应该要求玩家猜测一个字母。

图 8-2 流程图最初的时候只有“开始”方框和“结束”方框

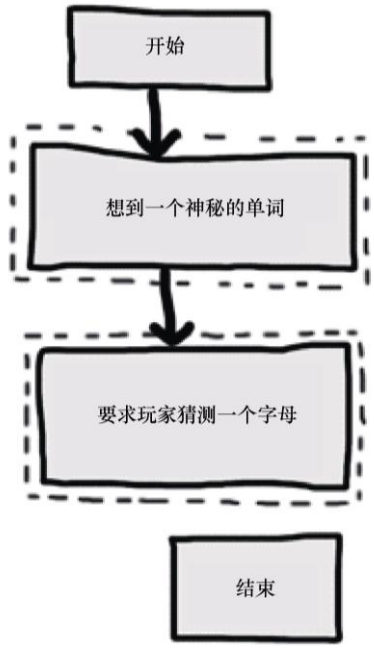


图 8-3 用带说明的方框画出 Hangman 的前两步

但是玩家猜测完一个字母之后游戏并没有结束。它需要判断该字母是否在神秘的单词中。

8.5.1　流程图的分支

这里有两种可能：字母在单词中或者字母不在单词中。我们需要为流程图增加两个新的方框，每种情况一个方框。这会在流程图中创建一个分支，如图 8-4 所示。

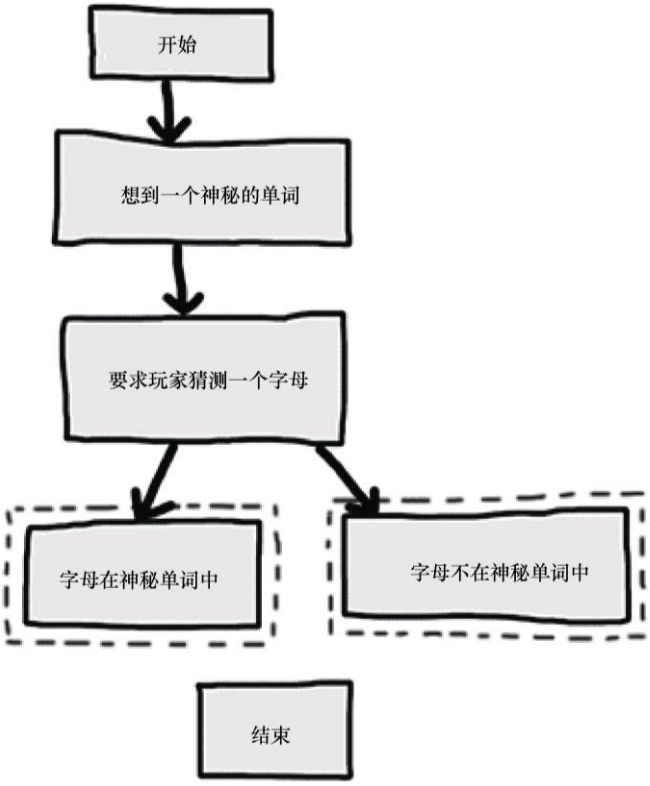


图 8-4　分支有两个箭头分别指向两个方框

如果字母在这个神秘的单词中，判断玩家是否猜对了所有字母并且赢得了游戏。如果字母不在这个神秘的单词中，为火柴人添加另一个身体部位。再次，为这些情况添加方框。

从“字母在神秘单词中”方框到“玩家已用完火柴人的身体部分而失败”方框之间不需要箭头，因为玩家不可能由于猜对了字母而失败，也不可能因为猜错了字母而胜利，所以不需要在二者之间画箭头。流程图现在看上去如图 8-5 所示。

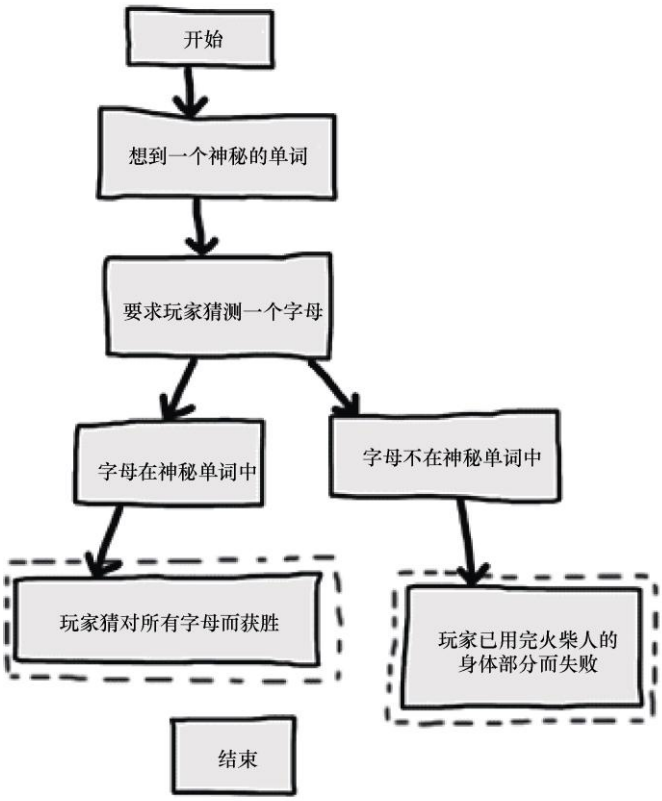


图 8-5 分支之后，在各自的路径上继续步骤

8.5.2 结束或者重新开始游戏

一旦玩家获胜或者失败，就会询问玩家是否用一个新的神秘单词再玩一局。如果玩家不想再玩了，程序将结束。如果程序还没有结束，它会想一个新的神秘单词，如图 8-6 所示。

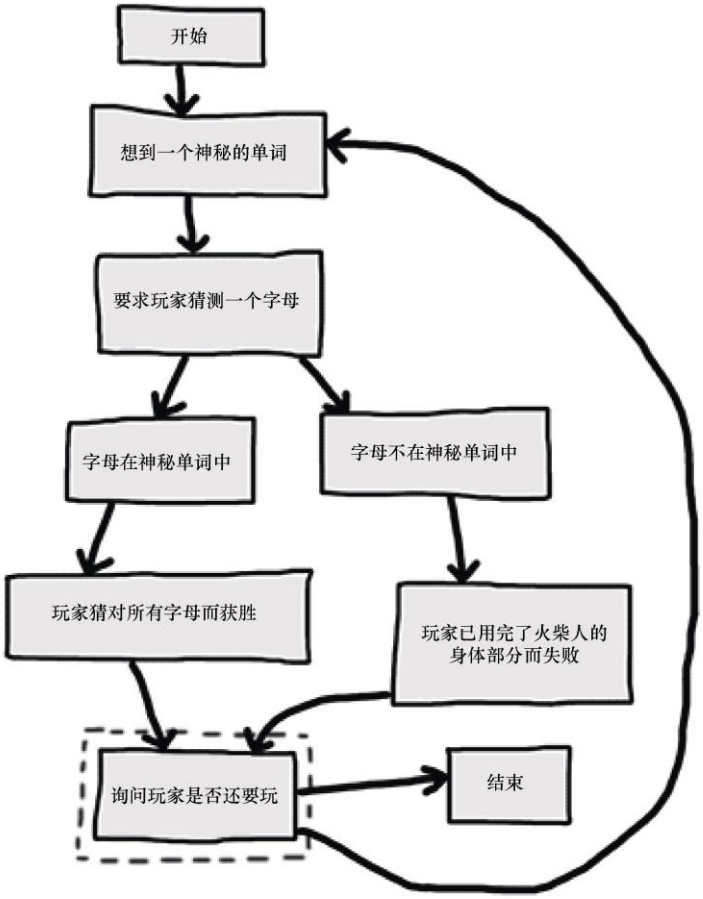


图 8-6　询问玩家是否再玩的时候，流程图出现分支

8.5.3　再猜一次

玩家不能只是猜一个字母。他们必须一直猜测字母，直到获胜或者失败。我们将画两个新的箭头，如图 8-7 所示。

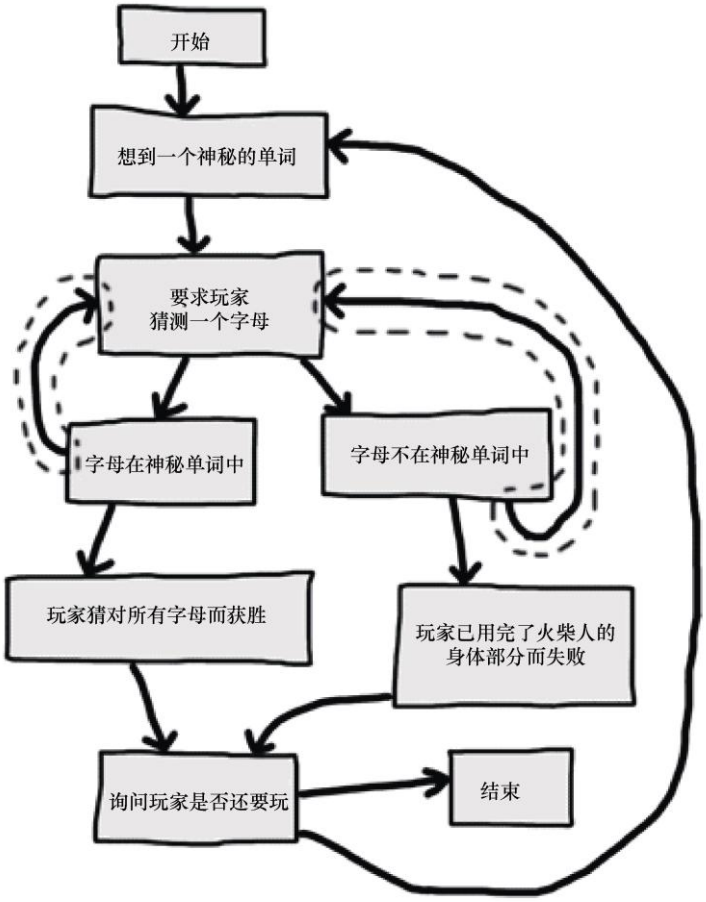


图 8-7 新的箭头（虚线框起来的）展示了玩家可以再次猜测

如果玩家再次猜测相同的字母该怎么办？这种情况，玩家既不会获胜也不会失败，而是会让玩家再猜一个不同的字母。新的方框如图 8-8 所示。

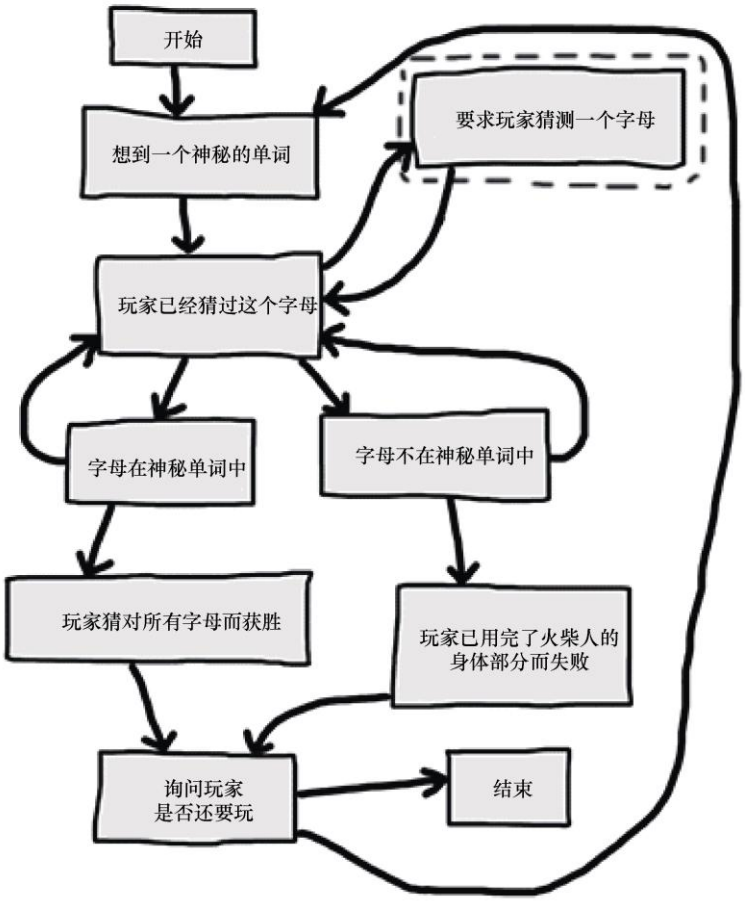


图 8-8　增加一个步骤，以免玩家已经猜过这个字母

8.5.4　为玩家提供反馈

玩家需要知道他们在游戏中正在做什么。游戏应该向他们展示火柴人游戏板和神秘单词（还没有猜到的字母用空格表示）。这些画面让玩家能够看到他们离游戏的胜利或失败有多接近。

每次玩家猜测一个字母，都会更新一下这个信息。在流程图中的“想到一个神秘的单词”方框和“要求玩家猜测一个字母”方框之间增加一个“向玩家显示游戏板和空格”方框，如图 8-9 所示。

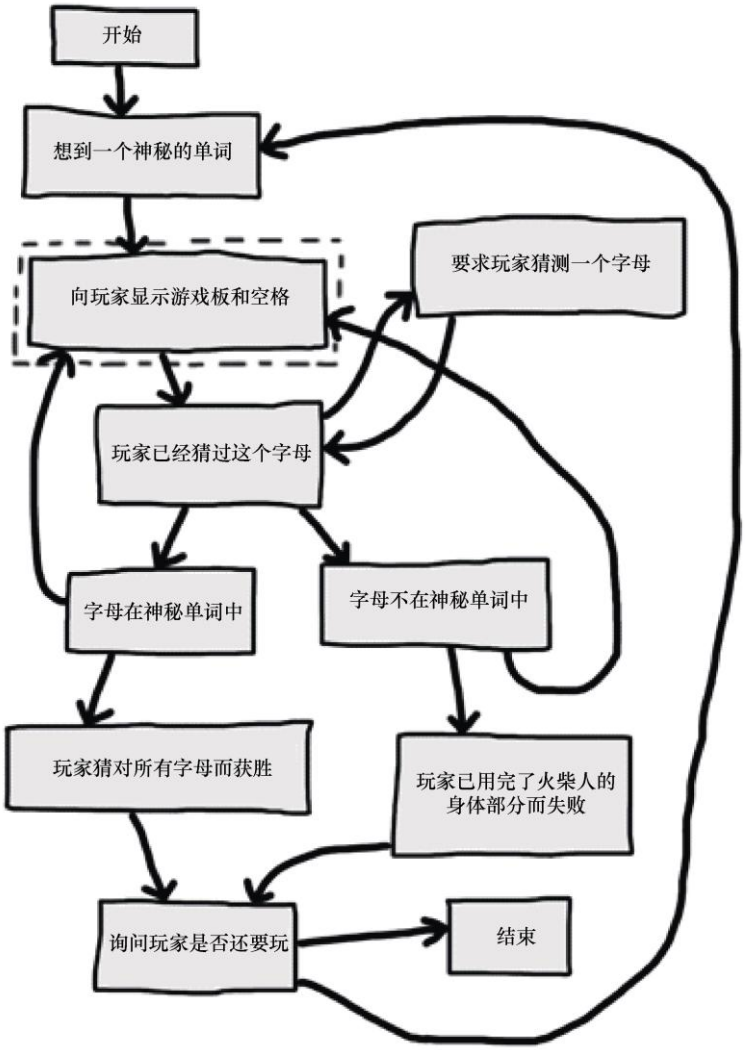


图 8-9 增加“向玩家显示游戏板和空格”为玩家提供反馈

看上去不错！这个流程图完全规划了 Hangman 中可能发生的每一件事及其顺序。当我们设计自己的游戏的时候，流程图可以帮助我们记住需要编写的每一件事。

8.6 本章小结

先为程序绘制一个流程图，这看上去好像是很大的工作量。毕竟，人们想要玩游戏，而

不是想要看流程图！但是，在编写代码之前，先思考程序是如何工作的，这更易于做出修改和发现问题。

如果你先一头扎入代码编写中，可能会发现需要修改已经编写好的代码。每次或多或少地修改代码，都可能会导致新的 bug。在构建之前，最好搞清楚你想要构建什么。

第 9 章　Hangman

本章主要内容：

- 多行字符串；
- 方法；
- 列表；
- 列表方法 append()和 reverse()；
- 字符串方法 lower()、upper()、split()、startswith()和 endswith()；
- 操作符 in 和 not in；
- range()函数和 list()函数；
- del 语句；
- for 循环；
- elif 语句。

本章的游戏介绍了许多新的概念，但是不用担心，我们将首先在交互式 shell 中体验这些编程概念。我们将介绍方法，方法是有返回值的函数。我们还将介绍一种叫做 for 循环的新的循环类型，以及一种叫做列表（list）的新的数据类型。一旦能够理解这些概念，编写 Hangman 程序就会容易很多。

9.1　Hangman 的源代码

本章的游戏要比之前的游戏的代码都长一些，不过很多代码是负责绘制火柴人图形的 ASCII 字符图。在文件编辑器中输入如下内容，并且把它保存为 hangman.py。

hangman.py

```
 1. import random
 2. HANGMANPICS = ['''
 3.
 4.   +---+
 5.   |   |
 6.       |
 7.       |
 8.       |
 9.       |
10. =========''', '''
11.
12.   +---+
```

```
  |   |
  O   |
      |
      |
      |
=========''', '''

  +---+
  |   |
  O   |
  |   |
      |
      |
=========''', '''

  +---+
  |   |
  O   |
 /|   |
      |
      |
=========''', '''

  +---+
  |   |
  O   |
 /|\  |
      |
      |
=========''', '''

  +---+
  |   |
  O   |
 /|\  |
 /    |
      |
=========''', '''

  +---+
  |   |
  O   |
 /|\  |
 / \  |
      |
```

```
 58. =========''']
 59. words = 'ant baboon badger bat bear beaver camel cat clam cobra cougar coyote
crow deer dog donkey duck eagle ferret fox frog goat goose hawk lion lizard llama
mole monkey moose mouse mule newt otter owl panda parrot pigeon python rabbit
ram rat raven rhino salmon seal shark sheep skunk sloth snake spider stork swan
tiger toad trout turkey turtle weasel whale wolf wombat zebra'.split()
 60.
 61. def getRandomWord(wordList):
 62.     # This function returns a random string from the passed list of
strings.
 63.     wordIndex = random.randint(0, len(wordList) - 1)
 64.     return wordList[wordIndex]
 65.
 66. def displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord):
 67.     print(HANGMANPICS[len(missedLetters)])
 68.     print()
 69.
 70.     print('Missed letters:', end=' ')
 71.     for letter in missedLetters:
 72.         print(letter, end=' ')
 73.     print()
 74.
 75.     blanks = '_' * len(secretWord)
 76.
 77.     for i in range(len(secretWord)): # replace blanks with correctly
guessed letters
 78.         if secretWord[i] in correctLetters:
 79.             blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
 80.
 81.     for letter in blanks: # show the secret word with spaces in between
each letter
 82.         print(letter, end=' ')
 83.     print()
 84.
 85. def getGuess(alreadyGuessed):
 86.     # Returns the letter the player entered. This function makes sure the
player entered a single letter, and not something else.
 87.     while True:
 88.         print('Guess a letter.')
 89.         guess = input()
 90.         guess = guess.lower()
 91.         if len(guess) != 1:
 92.             print('Please enter a single letter.')
```

```
 93.         elif guess in alreadyGuessed:
 94.             print('You have already guessed that letter. Choose again.')
 95.         elif guess not in 'abcdefghijklmnopqrstuvwxyz':
 96.             print('Please enter a LETTER.')
 97.         else:
 98.             return guess
 99.
100. def playAgain():
101.     # This function returns True if the player wants to play again,
otherwise it returns False.
102.     print('Do you want to play again? (yes or no)')
103.     return input().lower().startswith('y')
104.
105.
106. print('H A N G M A N')
107. missedLetters = ''
108. correctLetters = ''
109. secretWord = getRandomWord(words)
110. gameIsDone = False
111.
112. while True:
113.     displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
114.
115.     # Let the player type in a letter.
116.     guess = getGuess(missedLetters + correctLetters)
117.
118.     if guess in secretWord:
119.         correctLetters = correctLetters + guess
120.
121.         # Check if the player has won
122.         foundAllLetters = True
123.         for i in range(len(secretWord)):
124.             if secretWord[i] not in correctLetters:
125.                 foundAllLetters = False
126.                 break
127.         if foundAllLetters:
128.             print('Yes! The secret word is "' + secretWord + '"! You have
won!')
129.             gameIsDone = True
130.     else:
```

```
131.        missedLetters = missedLetters + guess
132.
133.        # Check if player has guessed too many times and lost
134.        if len(missedLetters) == len(HANGMANPICS) - 1:
135.            displayBoard(HANGMANPICS, missedLetters, correctLetters,
secretWord)
136.            print('You have run out of guesses!\nAfter ' +
str(len (missedLetters))+ ' missed guesses and ' + str(len(correctLetters)) + '
correct guesses, the word was "' + secretWord + '"')
137.            gameIsDone = True
138.
139.    # Ask the player if they want to play again (but only if the game is
done).
140.    if gameIsDone:
141.        if playAgain():
142.            missedLetters = ''
143.            correctLetters = ''
144.            gameIsDone = False
145.            secretWord = getRandomWord(words)
146.        else:
147.            break
```

代码如何工作

```
 1. import random
```

Hangman 程序随机地从神秘单词列表中选择了一个神秘单词。random 模块将会提供这一功能，所以第 1 行代码导入了该模块。

```
 2. HANGMANPICS = ['''
 3.
 4.   +---+
 5.   |   |
 6.       |
 7.       |
 8.       |
 9.       |
10. =========''', '''

...the rest of the code is too big to show here...
```

在源代码中，这条赋值语句包含了从第 2 行到第 58 行。为了帮助你更好地理解这段代码的含义，我们先介绍多行字符串。

9.2　多行字符串

到目前为止，我们见到的字符串都在一行中，并且在开始处和结束处都有一个引号。然而，如果在字符串开始处和结束处使用 3 个单引号，那么该字符串可以跨越多行：

```
>>> fizz = '''Dear Alice,
I will return to Carol's house at the end of the month. I will see you then.
Your friend,
Bob'''
>>> print(fizz)
Dear Alice,
I will return to Carol's house at the end of the month. I will see you then.
Your friend,
Bob
```

这就是多行字符串（multi-line strings）。在多行字符串中，换行字符也作为字符串的一部分包含在其中。我们无需使用\n 转义字符，也不需要转义引号（只要不会在字符串中将 3 个单引号一起使用）。这使得代码更容易读取大量的文本。

9.3　常量

变量 HANGMANPICS 的名称是全部大写的。这是表示常量的编程惯例。常量（constant）是在第一次赋值之后其值就不再变化的变量。尽管可以像修改任意其他变量一样来修改 HANGMANPICS 中的值，但是，其变量名全部大写的形式还是提醒我们不要这样做。既然 HANGMANPICS 变量从不需要修改，那么就把它标记为一个常量。

同所有惯例一样，我们也不一定必须要遵循它。但是遵循这个惯例，会让其他程序员更容易阅读你的代码。他们将会知道，HANGMANPICS 的值总是在第 2 行中所赋予的值。

9.4　列表

列表（list）值之中可以包含许多其他值。尝试在交互式 shell 中输入如下代码：

```
>>> spam = ['Life', 'The Universe', 'Everything', 42]
>>> spam
['Life', 'The Universe', 'Everything', 42]
```

spam 列表包含了 4 个值。当在代码中输入列表值的时候，以开始方括号（[）开始并且以结束方括号（]）结束。这就像字符串开始和结束都用引号字符一样。

逗号隔开列表中的各个值。这些值也叫做元素（item）。

9.4.1 索引

尝试在交互式 shell 中输入 animals = ['aardvark', 'anteater', 'antelope', 'albert']，从而把一个列表保存到变量 animals 中。方括号也用来访问列表中的一个元素。尝试在交互式 shell 中输入 animals[0]、animals[1]、animals[2]和 animals[3]，以查看它们的结果：

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[0]
'aardvark'
>>> animals[1]
'anteater'
>>> animals[2]
'antelope'
>>> animals[3]
'albert'
```

方括号之间的数字就是索引（index）。在 Python 中，列表中第 1 个元素的索引是 0，第 2 个元素的索引是 1，第 3 个元素的索引是 2，以此类推。因为索引是从 0 开始，而不是从 1 开始，所以我们说 Python 列表是基于 0 索引的（zero-indexed）。

列表适用于将多个值存储到一个变量中，而又不必为每个值使用一个变量。如果不使用列表的话，代码看上去可能如下所示：

```
>>> animals1 = 'aardvark'
>>> animals2 = 'anteater'
>>> animals3 = 'antelope'
>>> animals4 = 'albert'
```

如果有成百上千个字符串，代码可能很难管理。而一个列表可以很容易地保存任意多个值。使用方括号，就可以像处理任何其他值一样来处理列表中的元素。尝试在交互式 shell 中输入 animals[0] + animals[2]：

```
>>> animals[0] + animals[2]
'aardvarkantelope'
```

结果如下所示：

```
animals[0] + animals[2]
           ▼
'aardvark' + animals[2]
           ▼
```

```
'aardvark' + 'antelope'
           ▼
  'aardvarkantelope'
```

9.4.2　IndexError

如果试图访问的索引太大，就会得到一个 IndexError，这会让程序崩溃。尝试在交互式 shell 中输入如下代码：

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[9999]
Traceback (most recent call last):
File "", line 1, in
animals[9999]
IndexError: list index out of range
```

9.4.3　用指定索引来修改列表元素的值

也可以使用方括号来修改列表的元素值。尝试在交互式 shell 中输入如下代码：

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> animals[1] = 'ANTEATER'
>>> animals
['aardvark', 'ANTEATER', 'antelope', 'albert']
```

新的字符串'ANTEATER'覆盖了 animals 列表中的第 2 个元素。因此，表达式中的 animals[1]将得到列表的第 2 个元素，但是也可以把 animals[1]放在赋值语句的左边，从而为列表的第 2 个元素赋值。

9.4.4　列表连接

我们可以使用+操作符把一个列表连接到另一个列表中，就像连接字符串一样。用+操作符连接列表叫做列表连接（list concatenation）。尝试在交互式 shell 中输入如下代码：

```
>>> [1, 2, 3, 4] + ['apples', 'oranges'] + ['Alice', 'Bob']
[1, 2, 3, 4, 'apples', 'oranges', 'Alice', 'Bob']
```

['apples'] + ['oranges']的结果是['apples', 'oranges']。但是['apples'] + 'oranges'会导致一个错误。我们无法把一个列表值和一个字符串值相加，而只能把两个列表值相加。如果想要把非列表值添加到一个列表中，使用 append()方法（稍后介绍）。

9.4.5 in 操作符

in 操作符可以告诉我们，一个值是否在一个列表中。使用 in 操作符的表达式会返回一个布尔值：如果该值在列表中，返回值是 True；如果该值不在列表中，返回值是 False。尝试在交互式 shell 中输入如下代码：

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> 'antelope' in animals
True
```

字符串'antelope'是 animals 列表中的一个值，所以表达式'antelope' in animals 返回 True。这个字符串所在的索引是 2。

但是，如果输入的表达式是'ant' in animals，由于字符串'ant'不存在于列表中，该表达式将返回 False。

```
>>> animals = ['aardvark', 'anteater', 'antelope', 'albert']
>>> 'antelope' in animals
True
>>> 'ant' in animals
False
```

in 操作符也可以用于处理字符串。它判断一个字符串是否存在于另一个字符串中。尝试在交互式 shell 中输入如下代码：

```
>>> 'hello' in 'Alice said hello to Bob.'
True
```

9.4.6 使用 del 语句删除列表中的元素

del 语句将删除列表中特定索引的元素。尝试在交互式 shell 中输入如下语句：

```
>>> spam = [2, 4, 6, 8, 10]
>>> del spam[1]
>>> spam
[2, 6, 8, 10]
```

注意，当我们删除索引为 1 的元素时，原来索引为 2 的元素将成为索引 1 位置的新值。原来索引为 3 的元素移动到索引 2 的位置，成为索引 2 的新值。被删除的元素之后的每个元素都向前移动 1 个索引位置。

我们可以反复输入 del spam[1]，持续地从列表中删除元素：

```
>>> spam = [2, 4, 6, 8, 10]
>>> del spam[1]
>>> spam
[2, 6, 8, 10]
>>> del spam[1]
>>> spam
[2, 8, 10]
>>> del spam[1]
>>> spam
[2, 10]
```

del 语句是一条语句，而不是一个函数或者一个操作符。它后面没有圆括号，也不能得到返回值。

9.4.7　列表的列表

列表可以包含其他值，这也包括其他的列表。假设我们有一个食品列表（groceries）、一个家务活列表（chores）和一个最喜欢的派的列表（favoritePies）。我们可以把 3 个列表全部放入到另一个列表之中。尝试在交互式 shell 中输入如下代码：

```
>>> groceries = ['eggs', 'milk', 'soup', 'apples', 'bread']
>>> chores = ['clean', 'mow the lawn', 'go grocery shopping']
>>> favoritePies = ['apple', 'frumbleberry']
>>> listOfLists = [groceries, chores, favoritePies]
>>> listOfLists
[['eggs', 'milk', 'soup', 'apples', 'bread'], ['clean', 'mow the lawn', 'go
grocery shopping'], ['apple', 'frumbleberry']]
```

要获取列表中的列表的元素，像下面这样使用两组方括号：listOfLists[1][2]，它会得到字符串'go grocery shopping'。

这是因为 listOfLists[1][2]等于['clean', 'mow the lawn', 'go grocery shopping'][2]。最终的结果是'go grocery shopping'：

```
listOfLists[1][2]
          ▼
[['eggs', 'milk', 'soup', 'apples', 'bread'], ['clean', 'mow the lawn', 'go
grocery shopping'], ['apple', 'frumbleberry']][1][2]
          ▼
['clean', 'mow the lawn', 'go grocery shopping'][2]
          ▼
'go grocery shopping'
```

图 9-1 是包含在列表中的列表的另一个示例，还带有一些索引来指定元素。箭头指向内部列表自身的索引。我们还把图形横着放，以便于更容易阅读。

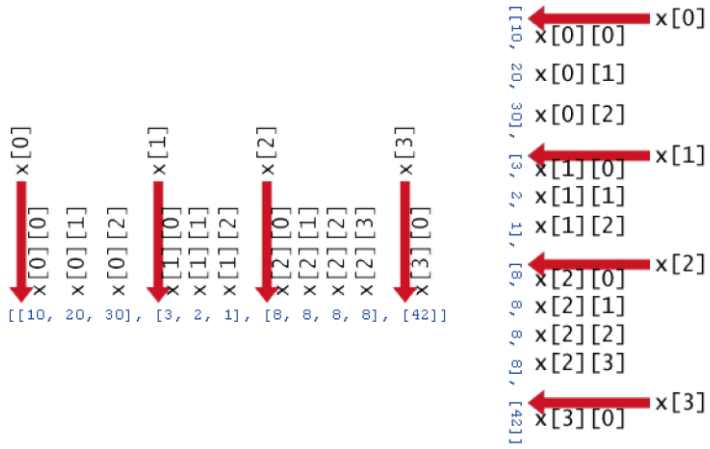


图 9-1　列表中的列表的索引

9.5　方法

方法（method）是附加到一个值之上的函数。例如，所有字符串值都有一个 lower()方法，它会返回字符串值的全部小写的副本。我们可以像'Hello'.lower()这样来调用该方法，它会返回'hello'。我们无法调用 lower()本身，也不能传递字符串参数给 lower()（即，不能像 lower('Hello')这样）。我们必须使用一个句点将该方法调用附加到一个指定的字符串值上。下一节将进一步介绍字符串方法。

9.6　字符串方法 lower()和 upper()

尝试在交互式 shell 中输入'Hello world!'.lower()，以查看该方法的一个示例：

```
>>> 'Hello world!'.lower()
'hello world!'
```

字符串还有一个 upper()方法，它会返回所有字符均为大写的一个字符串。尝试在交互式 shell 中输入'Hello world!'.upper()：

```
>>> 'Hello world!'.upper()
'HELLO WORLD! '
```

因为 upper()方法返回一个字符串，所以我们也可以在该字符串上调用一个方法。尝试在交互式 shell 中输入'Hello world!'.upper().lower()：

```
>>> 'Hello world!'.upper().lower()
'hello world!'
```

'Hello world!'.upper()会得到字符串'HELLO WORLD!'，然后调用这个字符串的 lower()方法。这会返回字符串'hello world!'，这才是求得的最终的值。

```
'Hello world!'.upper().lower()
                       ▼
       'HELLO WORLD!'.lower()
                       ▼
                  'hello world!'
```

顺序很重要。'Hello world!'.lower().upper()和'Hello world!'.upper().lower()并不相同：

```
>>> 'Hello world!'.lower().upper()
'HELLO WORLD!'
```

后者的计算过程看上去如下所示：

```
'Hello world!'.lower().upper()
                       ▼
         'hello world!'.upper()
                       ▼
                 'HELLO WORLD!'
```

如果变量中存储了一个字符串，我们就可以调用这个变量的字符串方法。看看如下的示例：

```
>>> spam = 'Hello world!'
>>> spam.upper()
'HELLO WORLD!'
```

这并没有改变 spam 中的值。spam 变量仍然将包含'Hello world!'。请注意，整数类型和浮点数类型都没有任何方法。

9.7　列表方法 reverse()和 append()

列表数据类型也有方法。reverse()方法会把列表中的元素的顺序反转。尝试输入 spam = [1, 2, 3, 4, 5, 6, 'meow', 'woof']，然后 spam.reverse()会反转这个列表。再次尝试输入 spam 来查看该变量中的内容。

```
>>> spam = [1, 2, 3, 4, 5, 6, 'meow', 'woof']
>>> spam.reverse()
>>> spam
['woof', 'meow', 6, 5, 4, 3, 2, 1]
```

我们最常用到的列表方法是 append()。这个方法会把作为参数传递给它的值添加到列表的末尾。尝试在交互式 shell 中输入如下语句：

```
>>> eggs = []
>>> eggs.append('hovercraft')
>>> eggs
['hovercraft']
>>> eggs.append('eels')
>>> eggs
['hovercraft', 'eels']
>>> eggs.append(42)
>>> eggs
['hovercraft', 'eels', 42]
```

这些方法确实修改了调用它们的列表。它们并没有返回一个新的列表。我们说这些方法就地（in-place）修改了这个列表。

9.8 列表方法 split()

第59行是很长的一行代码，但它实际上只是一条简单的赋值语句。这一行也使用了 split()方法，就像 lower()和 upper()方法一样，它也是字符串数据类型的一个方法。

```
59. words = 'ant baboon badger bat bear beaver camel cat clam cobra cougar
coyote crow deer dog donkey duck eagle ferret fox frog goat goose hawk lion
lizard llama mole monkey moose mouse mule newt otter owl panda parrot pigeon
python rabbit ram rat raven rhino salmon seal shark sheep skunk sloth snake
spider stork swan tiger toad trout turkey turtle weasel whale wolf wombat
zebra'.split()
```

这条赋值语句只有一个很长的字符串，所有的单词都用空格隔开。在字符串末尾是一个 split()方法调用。split()方法得到一个列表，字符串中的每一个单词都是列表中的一个元素。字符串中任何出现一个空格的地方，都会进行“分隔”。

使用 split()可以使得代码的输入更容易。如果从头开始创建列表的话，要输入['ant', 'baboon', 'badger'等等，每个单词之间都要用引号和逗号隔开。

例如，尝试在交互式 shell 中输入如下语句：

```
>>> sentence = input()
My very energetic mother just served us nachos.
>>> sentence.split()
['My', 'very', 'energetic', 'mother', 'just', 'served', 'us', 'nachos.']
```

结果是包含 9 个字符串的一个列表，原始字符串中的每个单词都是列表中的一个字符串。列表中的任何元素都不包含空格。

也可以在第 59 行中添加自己的单词，或者删除任何不想让其在游戏中出现的单词。只要确保用空格分隔开这些单词即可。

9.8.1　代码如何工作

第 61 行定义了 getRandomWord()方法。该方法将会接受一个列表参数 wordList。这个函数将返回 wordList 列表中的一个神秘单词。

```
61. def getRandomWord(wordList):
62.     # This function returns a random string from the passed list of strings.
63.     wordIndex = random.randint(0, len(wordList) - 1)
64.     return wordList[wordIndex]
```

第 63 行把这个列表的一个随机索引存储到了 wordIndex 变量中。通过使用两个参数来调用 randint()方法而做到这一点。第 1 个参数是 0（列表的第 1 个可能的索引），第 2 个参数是表达式 len(wordList) - 1 的值（wordList 中最后一个可能的索引）。

列表索引从 0 开始，而不是 1 开始。如果一个列表包含 3 个元素，第 1 个元素的索引是 0，第 2 个元素的索引是 1，第 3 个元素的索引是 2。这个列表的长度是 3，但是索引 3 将会在最后一个元素之后。这就是为什么第 63 行要用列表的长度减去 1。无论 wordList 的长度是多少，第 63 行代码都能工作。现在我们可以根据喜好添加或删除 wordList 中的字符串了。

我们会把变量 wordIndex 设置为列表的一个随机索引，而这个列表是作为 wordlist 参数传递进来的。第 64 行将会返回 wordList 中整数索引为 wordIndex 的元素。

我们假设['apple', 'orange', grape']是传递给 getRandomWord()的参数，并且 randint(0, 2)会返回整数 2。这意味着第 64 行将会返回 wordList[2]，然后会得到返回值'grape'。这就是 getRandomWord()返回 wordList 列表中的一个随机字符串的方法。

所以输入到 getRandomWord()中的是一个字符串列表，输出的返回值是从该列表中随机选取的一个字符串。对于 Hangman 游戏来说，这用来选择一个让玩家猜测的神秘单词。

9.8.2　向玩家显示游戏板

接下来，我们需要一个函数以便在屏幕上打印 Hangman 游戏板。这个函数还会显示玩

家已经猜对（或猜错）了多少个字母。

```
66. def displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord):
67.     print(HANGMANPICS[len(missedLetters)])
68.     print()
```

这段代码定义了一个名为 displayBoard()的新函数。该函数有 4 个参数：

- HANGMANPICS——多行字符串的一个列表，它将会以 ASCII 字符图来显示游戏板（全局变量 HANGMANPICS 将会传递给这个参数）。
- missedLetters——玩家已经猜过并且不在神秘单词中的字母所组成的字符串。
- correctLetters——玩家已经猜过并且在神秘单词中的字母所组成的字符串。
- secretWord——玩家试图猜测的神秘单词。

第 1 个 print()函数调用将会显示游戏板。HANGMANPICS 是构成每种可能的游戏板的字符串的一个列表。HANGMANPICS[0]显示了一个空的绞刑架，HANGMANPICS[1]显示一个头（当玩家猜错了一个字母），HANGMANPICS[2]显示一个头和身体（当玩家猜错了两个字母），依次类推，直到 HANGMANPICS[6]，它会显示一个完整的火柴人。

missedLetters 中的字母数目将会反映出玩家做了多少次错误的猜测。调用 len(missedLetters)以得到这个数字。所以，如果 missedLetters 是'aetr'，那么 len('aetr')将会返回 4。打印 HANGMANPICS[4]，将会显示猜错了 4 次所对应的 Hangman 游戏板。这是第 67 行 HANGMANPICS[len(missedLetters)]的结果。

```
70.     print('Missed letters:', end=' ')
71.     for letter in missedLetters:
72.         print(letter, end=' ')
73.     print()
```

第 70 行打印了字符串'Missed letters:'，在末尾使用的是一个空格字符而不是换行符。记住，关键字参数 end=' '只使用了一个等号（=），而不是两个等号（==）。

第 71 行是一个新的循环类型，叫做 for 循环。for 循环经常会使用 rang()函数。下一节会介绍它们。

9.9 range()函数和 list()函数

当用 1 个参数调用 range()的时候，会返回从 0 到该参数（但不包括该参数）的一个整数范围。可以用 list()函数把这个范围转换为我们更为熟悉的列表数据类型。尝试在交互式 shell 中输入 list(range(10))：

```
>>> list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```

list()函数和 str()或 int()函数类似。它接受传递的值，并且返回一个列表。使用 range()函数可以很容易地生成一个巨大的列表。尝试在交互式 shell 中输入 list(range(10000))：

```
>>> list(range(10000))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
       ...skipped for brevity...
...9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
```

这个列表很大，以至于无法全部显示在屏幕上。但是，我们可以把这个列表存储到一个变量中：

```
>>> spam = list(range(10000))
```

如果为 range()函数传递两个整数参数，返回的范围就在第 1 个整数参数到第 2 个整数参数（但不包括第 2 个参数）之间。

```
>>> list(range(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

range()函数经常用于 for 循环中，for 循环很像我们之前已经见过的 while 循环。

9.10　for 循环

for 循环对于遍历一个列表中的值很有帮助。这有别于 while 循环，只要指定条件为 True，while 循环就会一直循环下去。for 语句以关键字 for 开始，后边跟着一个新的变量名称，然后是关键字 in，然后是一个可迭代的值，最后用一个冒号结尾。

可迭代的值可以是一个列表的值、范围的值或者字符串数据类型的值。还有一些数据类型也可以当作可迭代的值，我们稍后会介绍。

程序执行每次通过循环的时候，for 语句中的新变量都会赋值为列表中下一个元素的值。

```
>>> for i in range(5):
...     print('i is set to ' + str(i))
...
i is set to 0
i is set to 1
i is set to 2
i is set to 3
```

```
i is set to 4
```

for 语句中的 range(5)返回的范围等同于 list [0, 1, 2, 3, 4]。第 1 次执行 for 语句块中的代码的时候，会将变量 i 设置为 0。在下一次迭代的时候，会把 i 设置为 1，以此类推。

for 语句把 range()函数返回的范围自动转换到一个列表中，所以在这个 for 语句中，不需要使用 list(range(5))，而是直接使用 range(5)。

列表和字符串都是可迭代的数据类型。我们可以在 for 语句中使用它们。尝试在交互式 shell 中输入如下语句：

```
>>> for thing in ['cats', 'pasta', 'programming', 'spam']:
...     print('I really like ' + thing)
...
I really like cats
I really like pasta
I really like programming
I really like spam

>>> for i in 'Hello':
...     print(i)
...
H
e
l
l
o
```

while 循环相当于 for 循环

for 循环类似于 while 循环，但是当只需要对一个列表中的元素进行迭代时，使用 for 循环可以输入更少的代码。下面的 while 循环通过添加了一些额外代码，实现了和前面 for 循环相同的功能：

```
>>> iterableVal = ['cats', 'pasta', 'programming', 'spam']
>>> index = 0
>>> while (index < len(iterableVal)):
...     thing = iterableVal[index]
...     print('I really like ' + thing)
...     index = index + 1
...
I really like cats
I really like pasta
I really like programming
I really like spam
```

但是使用 for 语句可以自动实现这些额外的代码，并且由于输入的代码更少，编程也更加容易。

displayBoard()函数的剩余部分显示了猜错的字母，并且创建了神秘单词的一个字符串，将所有还没有猜对的字母用空格来表示。

```
70.    print('Missed letters:', end=' ')
71.    for letter in missedLetters:
72.        print(letter, end=' ')
73.    print()
```

第 71 行的 for 循环将会遍历 missedLetters 字符串中的每个字符，并且把它们打印到屏幕上。记住，end=' '将会用一个空格字符替换字符串后边的换行字符。

例如，如果 missedLetters 是'ajtw'，这个 for 循环将会显示 a j t w。

9.11　分片

列表分片（list slicing）使用一个列表中的元素的一个子集，创建了另一个新的列表。在代码中，在列表后边的方括号中，使用冒号指定两个索引（开始索引和结束索引）。例如，尝试在交互式 shell 中输入如下语句：

```
>>> spam = ['apples', 'bananas', 'carrots', 'dates']
>>> spam[1:3]
['bananas', 'carrots']
```

表达式 spam[1:3]使用 spam 中索引从 1 到 3（但是不包括 3）的元素来组成一个列表。

如果没有开始索引，Python 会自动认为我们想要把索引 0 作为开始索引：

```
>>> spam = ['apples', 'bananas', 'carrots', 'dates']
>>> spam[:2]
['apples', 'bananas']
```

如果没有结束索引，Python 会自动认为我们想要列表的剩余部分：

```
>>> spam = ['apples', 'bananas', 'carrots', 'dates']
>>> spam[2:]
['carrots', 'dates']
```

分片是从列表中获取元素子集的一种简单的方法。可以使用与列表分片相同的方式来切片字符串。字符串中的每个字符就像是列表中的一个元素。尝试在交互式 shell 中输入如下语句：

```
>>> myName = 'Zophie the Fat Cat'
>>> myName[4:12]
'ie the F'
>>> myName[:10]
'Zophie the'
>>> myName[7:]
'the Fat Cat'
```

Hangman 中下一部分的代码用到了分片。

9.11.1 用空格表示神秘单词

现在我们想要编写打印出神秘单词的代码，但是对于没有猜对的字母，使用空格来表示。我们可以使用_字符（下划线）来表示。首先创建一个字符串，神秘单词中的每个字母都用下划线表示。然后用 correctLetters 中的字母来替换这些空格。

所以，如果神秘单词是'otter'，那么空白的字符串应该是'_ _ _ _ _'（5 个连续的_字符）。如果 correctLetters 是字符串'rt'，我们将会把字符串修改为'_tt_r'。从第 75 行到 79 行的代码就是做这些事情的。

```
 75.     blanks = '_' * len(secretWord)
```

第 75 行使用字符串复制创建了一个全是下划线的变量 blanks。记住，*操作符也可以用于字符串和整数，所以表达式'_' * 5 的结果是'_ _ _ _ _'。这会确保当 secretWord 有字母时，blanks 会具有相同数量的下划线。

```
 77.     for i in range(len(secretWord)): # replace blanks with correctly
guessed letters
 78.         if secretWord[i] in correctLetters:
 79.             blanks = blanks[:i] + secretWord[i] + blanks[i+1:]
```

第 77 行有一个 for 循环，用来遍历 secretWord 中的每个字母，并且如果该字母存在于 correctLetters 中，就用这个真实的字母来替换下划线。

例如，假设 secretWord 的值是'otter'，correctLetters 中的值是'tr'。我们想向玩家展示的字符串是'_tt_r'。来看看如何创建这个字符串。

第 77 行的 len(secretWord)调用会返回 5。range(len(secretWord))调用将会变成 range(5)，它会使得 for 循环分别在 0、1、2、3 和 4 迭代。

因为 i 的值将是[0, 1, 2, 3, 4]中的每个值，所以 for 循环中的代码等同于：

```
if secretWord[0] in correctLetters:
    blanks = blanks[:0] + secretWord[0] + blanks[1:]
```

```
if secretWord[1] in correctLetters:
    blanks = blanks[:1] + secretWord[1] + blanks[2:]

if secretWord[2] in correctLetters:
    blanks = blanks[:2] + secretWord[2] + blanks[3:]

if secretWord[3] in correctLetters:
    blanks = blanks[:3] + secretWord[3] + blanks[4:]

if secretWord[4] in correctLetters:
    blanks = blanks[:4] + secretWord[4] + blanks[5:]
```

如果还搞不清像secretWord[0]或blanks[3:]这样的值，那么看一下图9-2。它展示了secretWord和 blanks 变量的值，以及字符串中每个字母的索引。

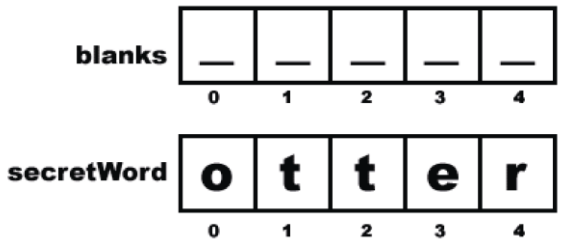


图 9-2　blanks 的索引和 secretWord 字符串

如果用图 9-2 中显示的值代替列表分片和列表索引，循环代码将等同于：

```
if 'o' in 'tr': # False
    blanks = '' + 'o' + '___' # This line is skipped.

if 't' in 'tr': # True
    blanks = '_' + 't' + '__' # This line is executed.

if 't' in 'tr': # True
    blanks = '_t' + 't' + '_' # This line is executed.

if 'e' in 'tr': # False
    blanks = '_tt' + 'e' + '_' # This line is skipped.

if 'r' in 'tr': # True
    blanks = '_tt_' + 'r' + '' # This line is executed.

# blanks now has the value '_tt_r'
```

当 secretWord 是'otter'并且 correctLetters 是'tr'时，上述示例代码所做的事情都相同。接下来的几行代码打印了 blanks 中的新值，每个字母之间都有空格隔开。

```
 81.    for letter in blanks: # show the secret word with spaces in between
each letter
 82.        print(letter, end=' ')
 83.    print()
```

9.11.2 获取玩家的猜测

我们将会调用 getGuess()函数，以便玩家可以输入一个猜测字母。这个函数将玩家猜测的字母作为一个字符串返回。另外，在返回之前，getGuess()会确保玩家输入了一个有效的字母。

```
 85. def getGuess(alreadyGuessed):
 86.     # Returns the letter the player entered. This function makes sure
the player entered a single letter, and not something else.
```

玩家猜过的字母所组成的字符串，作为 alreadyGuessed 参数传递给 getGuess()函数。然后，getGuess()函数要求玩家猜测一个字母。猜测的这个字母会作为 getGuess()的返回值。

```
 87.    while True:
 88.        print('Guess a letter.')
 89.        guess = input()
 90.        guess = guess.lower()
```

第 87 行的 while 循环会一直询问玩家，直到他们输入：

1．一个字母；

2．一个之前没有猜过的字母。

while 循环的条件简化成一个布尔值 True。这表示只有一种方式能够让执行跳出循环，就是执行一条 break 语句（跳出循环）或者 return 语句（return 不仅是跳出循环，还跳出了整个函数）。

循环中的代码要求玩家输入一个字母，并且会把这个字母保存在变量 guess 中。如果玩家输入一个大写字母，它将会在第 90 行用一个小写字母覆盖。

9.12 elif ("Else If")语句

Hangman 程序的下一部分用到了 elif 语句。我们可以把 elif("else if")语句看成是“如果这个条件为真，那么这样做；否则，如果下一个条件为真，就那样做；或者，如果所有条件

都不为真，那么最后这么来做”。

看看如下的代码：

```
if catName == 'Fuzzball':
    print('Your cat is fuzzy.')
elif catName == 'Spots':
        print('Your cat is spotted.')
else:
    print('Your cat is not fuzzy or spotted.')
```

如果变量 catName 等于字符串'Fuzzball'，那么 if 语句的条件为 True，并且 if 语句块告诉用户'Your cat is fuzzy.'。然而，如果条件为 False，那么 Python 尝试 elif (“else if”)语句作为接下来的条件。如果 catName 是'Spots'，那么会把字符串'Your cat is spotted.'打印到屏幕上。如果都为 False，那么代码会告诉用户'Your cat is not fuzzy or spotted.'。

只要你愿意，可以有任意多个 elif 语句：

```
if catName == 'Fuzzball':
    print('Your cat is fuzzy.')
elif catName == 'Spots':
    print('Your cat is spotted.')
elif catName == 'Chubs':
    print('Your cat is chubby.')
elif catName == 'Puff':
    print('Your cat is puffy.')
else:
    print('Your cat is neither fuzzy nor spotted nor chubby nor puffy.')
```

当其中一个 elif 条件为 True，它的代码就会执行，然后执行就会跳到该 else 语句块之后的第一行代码。所以在 if-elif-else 语句中，有且只有一个语句块会执行。如果一个 else 语句块都不需要，也可以去掉 else 语句块，只留下 if-elif 语句。

9.12.1　确保玩家输入正确的猜测

```
91.         if len(guess) != 1:
92.             print('Please enter a single letter.')
93.         elif guess in alreadyGuessed:
94.             print('You have already guessed that letter. Choose again.')
95.         elif guess not in 'abcdefghijklmnopqrstuvwxyz':
96.             print('Please enter a LETTER.')
97.         else:
98.             return guess
```

变量 guess 包含了玩家猜测的字母。程序需要确保玩家输入了有效的猜测：一个且只有一个小写字母。如果玩家没有这样做，执行会循环回来，再次要求他们输入一个字母。

第 91 行条件判断 guess 长度是否为 1。第 93 行条件判断 guess 是否已经存在于 alreadyGuessed 变量中。第 95 行条件判断 guess 是不是一个小写字母。

如果所有这些条件都是 False（即用户输入了单个的字母，它不是已经猜测过的字母，而且还是一个小写的字母），那么执行 else 语句块，第 98 行 getGuess()返回 guess 中的值。

记住，if-elif-else 语句中只有一个语句块会执行。

9.12.2 家是否想再玩一局

```
100. def playAgain():
101.     # This function returns True if the player wants to play again,
otherwise it returns False.
102.     print('Do you want to play again? (yes or no)')
103.     return input().lower().startswith('y')
```

playAgain()函数只有一个 print()函数调用和一条 return 语句。return 语句有一个看上去很复杂的表达式，但是可以分解来看。如果用户输入 YES，下面是 Python 一步步地执行这个表达式的过程。

```
input().lower().startswith('y')
           ▼
  'YES'.lower().startswith('y')
           ▼
         'yes'.startswith('y')
           ▼
          True
```

playAgain()函数的目的是让玩家输入 yes 或 no，以告知程序是否想要再玩一局 Hangman。玩家应该可以输入 YES、yes、Y 或者以“Y”开头的任何内容，以表示“yes”。如果玩家输入 YES，那么 input()的返回值是字符串'YES'。'YES'.lower()返回该字符串的一个小写字母的版本。所以'YES'.lower()的返回值是'yes'。

但是，这里还有第 2 个方法调用，startswith('y')。如果相关的字符串以圆括号之间的参数为起始字母，这个函数就返回 True；如果不是，则返回 False。'yes'.startswith('y')的返回值是 True。

现在我们已经得到这个表达式的结果！它所做的事情是让玩家输入一个回答，将这个回答都变成小写字母，判断回答是否以字母'y'开始。如果是，就返回 True；如果不是，就返回 False。

另一方面还要注意一下，还有一个字符串方法 endswith(someString)，如果字符串以

someString 中的内容结尾，该方法会返回 True，否则该方法会返回 False。endswith()就像是和 startswith()相反的函数。

9.12.3　回顾 Hangman 中的函数

如下是我们为这个游戏创建的所有函数。我们再来回顾一下：

- getRandomWord(wordList)将接受传递给它的一个字符串列表作为参数，并且从中返回一个字符串。该函数选中一个单词供玩家猜测。
- displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)将会显示游戏板的当前状态，包括目前为止玩家已经猜到的神秘单词部分，以及玩家猜错的字母。这个函数需要接受 4 个参数才能正常工作。HANGMANPICS 是一个字符串列表，保存了每种可能的 Hangman 游戏板的 ASCII 字符图。correctLetters 和 missedLetters 分别是由玩家已经猜测过的在神秘单词中的字母和不在神秘单词中的字母所构成的字符串。secretWord 是玩家试图猜测的神秘单词。这个函数没有返回值。
- getGuess(alreadyGuessed)接受由玩家已经猜过的字母所组成的一个字符串作为参数，并且继续要求玩家输入一个不在 alreadyGuessed 中的字母。这个函数返回玩家已经猜测过的、有效的字母所构成的字符串。
- playAgain()询问玩家是否想要再玩一局 Hangman 游戏。如果玩家想要再玩一局，该函数返回 True；否则，返回 False。

在这些函数之后，从第 106 行开始了程序的主要部分。之前的各行代码只是函数定义和 HANGMANPICS 的大量的赋值语句。

9.12.4　创建变量

```
106. print('H A N G M A N')
107. missedLetters = ''
108. correctLetters = ''
109. secretWord = getRandomWord(words)
110. gameIsDone = False
```

当游戏运行时，会先执行第 106 行的第一个 print()函数调用。它会显示游戏的名称。接下来把 missedLetters 和 correctLetters 设置为空字符串，因为玩家还没有猜错或猜对任何字母。

getRandomWord(words)函数调用会从 words 列表中得到一个随机选取的单词。

第 110 行把 gameIsDone 设置为 False。当想要表示游戏结束时，代码会将 gameIsDone 设置为 True，并且将询问玩家是否还想再玩一局。

9.12.5 为玩家显示游戏板

```
112. while True:
113.     displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
```

这个 while 循环的条件总是 True，这意味着它会永远循环，直到遇到一条 break 语句（会这在第 147 行出现）。

第 113 行调用了 displayBoard()函数，它接受 Hangman ASCII 字符图的列表和在第 107 行、108 行、109 行所设置的 3 个变量作为参数。根据玩家猜对了多少个字母和猜错了多少个字母，这个函数为玩家显示了相应的 Hangman 游戏板。

9.12.6 让玩家输入他们的猜测

```
115.     # Let the player type in a letter.
116.     guess = getGuess(missedLetters + correctLetters)
```

getGuess()函数需要 missedLetters 和 correctLetters 组合中的所有字母，所以第 116 行把这两个变量中的字符串连接在一起，并且把结果作为参数传递给该函数。getGuess()需要这个参数，因为这个函数需要检查玩家输入的字母是否已经猜测过了。

9.12.7 判断字母是否在这个神秘单词中

```
118.     if guess in secretWord:
119.         correctLetters = correctLetters + guess
```

如果 guess 字符串存在于 secretWord 中，那么把 guess 连接到 correctLetters 字符串的末尾。连接后的字符串将会是 correctLetters 的新值。

9.12.8 判断玩家是否获胜

```
121.         # Check if the player has won
122.         foundAllLetters = True
123.         for i in range(len(secretWord)):
124.             if secretWord[i] not in correctLetters:
125.                 foundAllLetters = False
126.                 break
```

程序如何知道玩家是否已经猜中了神秘单词中的每个字母？correctLetters 中有玩家正确猜测的每个字母，secretWord 是神秘单词本身。但是我们不能只是判断 correctLetters == secretWord，因为要考虑这种情况：如果 secretWord 是字符串'otter'，correctLetters 是字符

串'orte'，那么即便玩家已经猜出了神秘单词中的每个字母，但 correctLetters == secretWord 还是会为 False。

确定玩家获胜的唯一一种方式是遍历 secretWord 中的每个字母，并且判断它是否已经存在于 correctLetters 中。只有 secretWord 中的每个字母都存在于 correctLetters 中，玩家才会获胜。

如果发现 secretWord 中的一个字母不在 correctLetters 中，我们就知道玩家没有猜对所有字母。在循环开始前，第 122 行代码把新的变量 foundAllLetters 设置为 Ture。该循环最初假设神秘单词中的所有字母都已经被找到了。但是在循环代码的第 125 行，第一次发现 secretWord 中的一个字母不在 correctLetters 中，就会把 foundAllLetters 改为 False。

```
127.        if foundAllLetters:
128.            print('Yes! The secret word is "' + secretWord + '"! You have
won!')
129.            gameIsDone = True
```

如果神秘单词中的所有字母都被找到了，就会告知玩家，他们获胜了，并且会把 gameIsDone 设置为 True。

9.12.9　当玩家猜错时

```
130.    else:
131.        missedLetters = missedLetters + guess
```

else 语句块由此开始。记住，如果条件为 False，这个语句块的代码将会执行。但是，是哪一个条件呢？要找到答案，把你的手指指向 else 关键字的开始处，并且一直向上移动，如图 9-3 所示。我们会看到 else 关键字的缩进与第 118 行的 if 关键字的缩进是一样的。

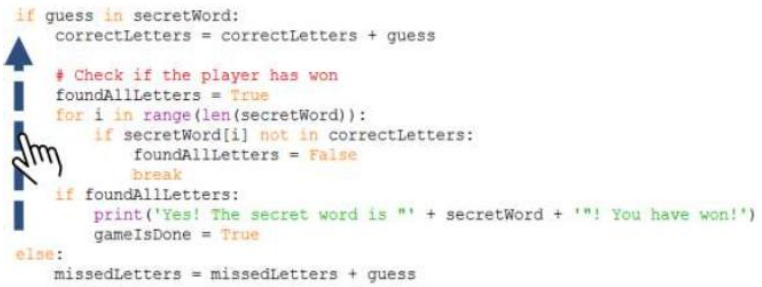


图 9-3　else 语句与具有相同缩进的 if 语句匹配

所以，如果第 118 行的条件（guess in secretWord）是 False，那么执行会进入到这个 else 语句块中。

第 131 行代码会把猜错的字母连接到 missedLetters 字符串。这就像第 119 行代码对于玩家猜对的字母所做的事情一样。

```
133.        # Check if player has guessed too many times and lost
134.        if len(missedLetters) == len(HANGMANPICS) - 1:
135.            displayBoard(HANGMANPICS, missedLetters, correctLetters,
secretWord)
136.            print('You have run out of guesses!\nAfter ' +
str(len(missedLetters)) + ' missed guesses and ' + str(len(correctLetters)) +
' correct guesses, the word was "' + secretWord + '"')
137.            gameIsDone = True
```

玩家每次猜错，代码都会把猜错的字母连接到 missedLetters 中的字符串。所以 missedLetters 的长度（代码为 len(missedLetters)）也就是猜错的次数。

HANGMANPICS 列表有 7 个 ASCII 字符图。所以当 len(missedLetters)等于 6 的时候，我们就知道玩家已经失败了，因为 Hangman 图片已经完成了。记住，HANGMANPICS[0]是列表中的第一项，HANGMANPICS[6]是最后一项。

所以，当 missedLetters 字符串的长度等于 len(HANGMANPICS) - 1（也就是 6）时，玩家用尽了猜测次数。第 136 行打印出了神秘单词，第 137 行把 gameIsDone 变量设置为 True。

```
139.    # Ask the player if they want to play again (but only if the game
is done).
140.    if gameIsDone:
141.        if playAgain():
142.            missedLetters = ''
143.            correctLetters = ''
144.            gameIsDone = False
145.            secretWord = getRandomWord(words)
```

如果玩家猜测了字母之后获胜或者失败，游戏应该会询问玩家是否再玩一局。playAgain()函数从玩家处获取一个 yes 或者 no，因此第 141 行调用了这个函数。

如果玩家想要再玩一局，必须重新将 missedLetters 和 correctLetters 中的值设置为空字符串，将 gameIsDone 设置为 False，并且把一个新的神秘单词存储到 secretWord 中。当执行循环回到 112 行 while 循环的开始处时，游戏板将开始一次全新的游戏。

```
146.        else:
147.            break
```

当询问玩家是否想要再玩一局时，如果玩家没有输入以“y”开头的单词，那么 141 行的条件将为 False，并且会执行 else 语句块。break 语句会导致执行跳到循环之后的第一条指令。但是，因为循环之后没有其他的指令了，所以程序终止了。

9.13　本章小结

本章是篇幅很长的一章，我们介绍了很多新的概念。但是，Hangman 是目前为止我们编写的最高级的游戏。随着游戏越来越复杂，针对程序中会发生的事情，先在纸上草拟一个流程图，这是一个好主意。

列表是包含了其他值的值。方法是特定于一个数据类型的函数。列表有 append()和 reverse()方法。字符串有 lower()、upper()、split()、startswith()和 endswith()方法。我们将在本书的剩余部分中学习更多的数据类型和方法。

for 循环是遍历列表中的元素的一种循环，它和 while 循环不同：只要条件为 True，while 循环就会一直进行下去。elif 语句允许我们在 if-else 语句中间添加添加一条“or else if”子句。del 语句可以删除变量或者列表中的元素。

第 10 章　Hangman 扩展

本章主要内容：

- 字典类型；
- 键-值对；
- 字典方法 keys()和 values()；
- 多变量赋值。

Hangman 程序比 Dragon Realm 程序大很多，也更复杂一些。用一张流程图或者一张小的草图来记录想要做的每一件事，的确会很有帮助。现在，我们已经创建了一个基础的 Hangman 游戏，再来看看为它扩展新功能的一些方法。

当你玩了几次 Hangman 之后，可能会认为对许多单词来讲，6 次猜测机会是远远不够的。通过为 HANGMANPICS 列表添加多行字符串，可以很容易地为玩家提供更多的猜测机会。

把 hangman.py 程序保存为 hangman2.py，然后添加如下的指令：

```
58. ==========''', '''
59.   +----+
60.   |    |
61.  [O    |
62.  /|\   |
63.  / \   |
64.        |
65. ==========''', '''
66.   +----+
67.   |    |
68.  [O]   |
69.  /|\   |
70.  / \   |
71.        |
72. ==========''']
```

HANGMANPICS 列表中有两个新的多行字符串，一个字符串画了火柴人的左耳朵，另一个字符串画了火柴人的两只耳朵。因为第 134 行代码 len(missedLetters)==len(HANGMANPICS) - 1 将会告诉玩家已经输了，所以这里是唯一一处必须要做的改动。程序剩余部分所做的修改只是想让新的 HANGMANPICS 列表工作的更好而已。

我们还可以通过改变第 59 行中的单词来修改 words 列表。可以不再使用动物，而是使用颜色：

```
59. words = 'red orange yellow green blue indigo violet white black
brown'.split()
```

或者使用形状：

```
59. words = 'square triangle rectangle circle ellipse rhombus trapezoid
chevron pentagon hexagon septagon octagon'.split()
```

或者使用水果：

```
59. words = 'apple orange lemon lime pear watermelon grape grapefruit cherry
banana cantaloupe mango strawberry tomato'.split()
```

10.1　字典

我们可以修改代码来让游戏产生一些变化，以便 Hangman 游戏可以使用一组词汇，诸如动物、颜色、形状或水果。程序可以告诉玩家神秘单词来自于哪种类型的词汇（动物、颜色、形状或水果）。

要做出这些修改，需要使用一种叫做字典（dictionary）的新的数据类型。字典是像列表一样的一个值的集合。但是访问字典中的元素使用的不是整数索引，我们可以使用任意数据类型的索引来访问字典。这些索引叫做字典的键（key）。

字典使用花括号（{和}），而不是方括号（[和]）。尝试在交互式 shell 中输入如下代码：

```
>>> spam = {'hello':'Hello there, how are you?', 4:'bacon', 'eggs':9999 }
```

在花括号之间的值是键-值对（key-value pair）。冒号的左边是键，冒号的右边是该键的值。我们可以使用键来访问值，就像用索引访问列表中的元素一样。尝试在交互式 shell 中输入如下代码：

```
>>> spam = {'hello':'Hello there, how are you?', 4:'bacon', 'eggs':9999}
>>> spam['hello']
'Hello there, how are you?'
>>> spam[4]
'bacon'
>>> spam[eggs]
9999
```

可以看到，放在方括号中的不是整数，而是一个字符串。这将会得到该键所对应的值。

10.1.1 用 len()函数获取字典的大小

可以使用 len()函数来得到字典中的键-值对的数目。尝试在交互式 shell 中输入如下代码：

```
>>> stuff = {'hello':'Hello there, how are you?', 4:'bacon', 'spam':9999}
>>> len(stuff)
3
```

10.1.2 字典和列表的区别

字典可以拥有任意数据类型的键，而不仅仅是字符串类型的键。但是记住，因为 0 和'0'是不同的值，所以它们是不同的键。尝试在交互式 shell 中输入如下代码：

```
>>> spam = {'0':'a string', 0:'an integer'}
>>> spam[0]
'an integer'
>>> spam['0']
'a string'
```

也可以使用 for 循环来遍历字典中的键。尝试在交互式 shell 中输入如下代码：

```
>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}
>>> for k in favorites:
...     print(k)
fruit
number
animal
>>> for k in favorites:
...     print(favorites[k])
apples
42
cats
```

字典与列表不同，因为字典中的值是无序的。listStuff 列表中的第 1 个元素是 listStuff[0]。但是字典中没有“第 1 个”元素，因为字典没有任何的顺序可言。尝试在交互式 shell 中输入如下代码：

```
>>> favorites1 = {'fruit':'apples', 'number':42, 'animal':'cats'}
>>> favorites2 = {'animal':'cats',  'number':42, 'fruit':'apples'}
>>> favorites1 == favorites2
True
```

表达式 favorites1 == favorites2 的结果是 True，因为字典是无序的，所以只要它们拥有相同的键-值对，就会把它们看做是相等的。然而列表是有序的，所以值相同但是顺序不同的两个列表，彼此是不相等的。尝试在交互式 shell 中输入：

```
>>> listFavs1 = ['apples', 'cats', 42]
>>> listFavs2 = ['cats', 42, 'apples']
>>> listFavs1 == listFavs2
False
```

字典有两个很有用的方法，keys()和 values()。这两个方法将分别返回类型为 dict_keys 和 dict_values 的值。和 range 对象类似，通过使用 list()函数，可以返回这些数据类型的值。尝试在交互式 shell 中输入如下代码：

```
>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}
>>> list(favorites.keys())
['fruit', 'number', 'animal']
>>> list(favorites.values())
['apples', 42, 'cats']
```

10.1.3　为 Hangman 设置词汇

让我们修改 Hangman 游戏中的代码，以提供不同的神秘词汇。首先，将赋给变量 words 的值替换为一个字典，该字典的键是字符串，而字典的值是字符串列表。字符串方法 split()将会返回由每一个单词的字符串所组成的一个列表。

```
59. words = {'Colors':'red orange yellow green blue indigo violet white black
brown'.split(),
60. 'Shapes':'square triangle rectangle circle ellipse rhombus trapezoid
chevron pentagon hexagon septagon octagon'.split(),
61. 'Fruits':'apple orange lemon lime pear watermelon grape grapefruit cherry
banana cantaloupe mango strawberry tomato'.split(),
62. 'Animals':'bat bear beaver cat cougar crab deer dog donkey duck eagle fish
frog goat leech lion lizard monkey moose mouse otter owl panda python rabbit
rat shark sheep skunk squid tiger turkey turtle weasel whale wolf wombat
zebra'.split()}
```

第 59 行到第 62 行的代码跨越了多行，但是它们仍然只是一条赋值语句。直到第 62 行的结束花括号}为止，这条语句才结束。

10.2 random.choice()函数

random 模块中的 choice()函数接受一个列表作为参数，并且从中返回一个随机值。这类似于之前的 getRandomWord()函数所做的工作。在新的 getRandomWord()函数中，我们将使用 random.choice()。

要看看 random.choice()如何工作，尝试在交互式 shell 中输入如下代码：

```
>>> import random
>>> random.choice(['cat', 'dog', 'mouse'])
'mouse'
>>> random.choice(['cat', 'dog', 'mouse'])
'cat'
>>> random.choice([2, 22, 222, 223])
2
>>> random.choice([2, 22, 222, 223])
222
```

修改 getRandomWord()函数，将其参数从一个字符串列表改为字符串列表的一个字典。这个函数最初如下所示：

```
61. def getRandomWord(wordList):
62     .# This function returns a random string from the passed list of
strings.
63.     wordIndex = random.randint(0, len(wordList) - 1)
64.     return wordList[wordIndex]
```

将这个函数中的代码修改为如下所示：

```
64. def getRandomWord(wordDict):
65.     # This function returns a random string from the passed dictionary of
lists of strings, and the key also.
66.     # First, randomly select a key from the dictionary:
67.     wordKey = random.choice(list(wordDict.keys()))
68.
69.     # Second, randomly select a word from the key's list in the dictionary:
70.     wordIndex = random.randint(0, len(wordDict[wordKey]) - 1)
71.
72.     return [wordDict[wordKey][wordIndex], wordKey]
```

把参数名字从 wordList 改为 wordDict，使其更有意义。现在该函数首先调用 random.choice()，从 wordDict 字典中选取一个随机的键，而不再是从一个字符串列表中选

择一个随机单词。

并且，函数返回包含两个元素的一个列表，而不再返回字符串 wordList[wordIndex]。返回的列表的第 1 个元素是 wordDict[wordKey][wordIndex]，第 2 个元素是 wordKey。

列表的字典的计算过程

第 72 行的表达式 wordDict[wordKey][wordIndex]看上去可能有点复杂，但它只是一个表达式，我们可以像任何其他表达式一样，对其一次只计算一步。首先，假设 wordKey 有一个值'Fruits'（在第 65 行选取的），并且 wordIndex 的值是 5（在第 68 行选取的）。wordDict[wordKey][wordIndex]的计算过程如下所示：

```
wordDict[wordKey][wordIndex]
           ▼
wordDict['Fruits'][wordIndex]
           ▼
['apple', 'orange', 'lemon', 'lime', 'pear', 'watermelon', 'grape',
'grapefruit', 'cherry', 'banana', 'cantaloupe', 'mango', 'strawberry',
'tomato'][wordIndex]
           ▼
['apple', 'orange', 'lemon', 'lime', 'pear', 'watermelon', 'grape',
'grapefruit', 'cherry', 'banana', 'cantaloupe', 'mango', 'strawberry',
'tomato'][5]
           ▼
     'watermelon'
```

在上述示例中，这个函数返回的列表中的元素将会是字符串'watermelon'（记住，索引从 0 开始，所以[5]指的是列表中的第 6 个元素，而不是第 5 个元素）。

因为 getRandomWord()函数现在返回的是带有两个元素的一个列表，而不是一个字符串，所以会给 secretWord 赋值一个列表，而不是一个字符串。我们可以使用多变量赋值，把这两个元素分别赋值给两个变量。接下来会介绍多变量赋值。

10.3　多变量赋值

多变量赋值（multiple assignment）是给多个变量赋值的一种快捷方式，变量放在赋值语句的左边，变量之间用逗号隔开。尝试在交互式 shell 中输入如下语句：

```
>>> a, b, c = ['apples', 'cats', 42]
>>> a
'apples'
>>> b
```

```
'cats'
>>> c
42
```

上述示例等同于如下的赋值语句：

```
>>> a = ['apples', 'cats', 42][0]
>>> b = ['apples', 'cats', 42][1]
>>> c = ['apples', 'cats', 42][2]
```

在赋值操作符（=）左边放置的变量，必须和赋值操作符右边的列表中的元素具有相同的数目。Python 将自动把列表中第 1 个元素的值赋给第 1 个变量，将第 2 个元素的值赋给第 2 个变量，以此类推。但是，如果变量的数目和元素的数目不相等，Python 解释器将会报错。

```
>>> a, b, c, d = ['apples', 'cats', 42, 10, 'hello']
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    a, b, c, d = ['apples', 'cats', 42, 10, 'hello']
ValueError: too many values to unpack

>>> a, b, c, d = ['apples', 'cats']
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    a, b, c = ['apples', 'cats']
ValueError: need more than 2 values to unpack
```

修改 Hangman 中第 109 行和第 145 行的代码，将 getRandomWord()的返回值用于多变量赋值：

```
108. correctLetters = ''
109. secretWord, secretKey = getRandomWord(words)
110. gameIsDone = False
...
144. gameIsDone = False
145. secretWord, secretKey = getRandomWord(words)
146. else:
```

向玩家显示单词的分类

我们要做的最后的修改是告诉玩家他们将要猜测的单词是哪一类的。这样，当玩家玩游戏时，他们就知道神秘单词是动物、颜色、形状还是水果。把这行代码添加到第 112 行之后。最初的代码如下所示：

```
112. while True:
113.     displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
```

添加了这行代码之后，如下所示：

```
112. while True:
113.     print('The secret word is in the set: ' + secretKey)
114.     displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)
```

现在，我们已经完成了对 Hangman 程序的修改。神秘单词不再只是一个字符串列表，而是从众多不同类型的字符串列表中选取出来的。程序还会告诉玩家，神秘单词属于哪一类词汇。试着玩玩这个新的版本。我们可以很容易地修改第 59 行的 words 字典，以包含更多的词汇集。

10.4　本章小结

我们已经完成了 Hangman 游戏。即使已经完成了一个游戏的编写，但是在学习了更多 Python 编程知识之后，你总是可以添加更多的功能。

字典与列表类似，不过它们可以使用任意类型的值作为索引，而不只是以整数为索引。字典的索引叫做键。

多变量赋值是将列表中的值赋给多个变量的一种快捷方式。

与本书前边介绍的程序相比，Hangman 相当高级。截止到现在，我们了解了编写程序的大部分概念：变量、循环、函数以及诸如列表和字典这样的 Python 数据类型。尽管本书后面的程序仍然会有一些挑战，但是你已经完成了“攀爬”过程中最“陡峭”的部分。

第 11 章　Tic Tac Toe

本章主要内容：

- 人工智能；
- 列表引用；
- 短路运算；
- None 值。

本章通过一个简单的人工智能来介绍 Tic Tac Toe。人工智能（artificial intelligence，AI）是一个计算机程序，它可以对玩家的落子智能地作出响应。这个游戏并没有引入任何复杂的新的概念。玩 Tic Tac Toe 的人工智能实际上只是寥寥数行代码而已。

两个人可以用纸和铅笔玩 Tic Tac Toe。一个玩家画 X，另一个玩家画 O。玩家交替画下 X 或 O。如果一位玩家在游戏板中的一行、一列或一个对角线上画下 3 个标记，就获胜。当游戏板画完，仍没有玩家获胜，游戏就以平局结束。

本章没有引入新的编程概念，而是使用我们已经学过的编程知识，来创建一位智能的 Tic Tac Toe 玩家。我们先看一下游戏运行的简单示例。玩家通过输入他们想要去的格子的编号，来表示他们的落子。这些编号与键盘上的按键位置相同，如图 11-2 所示。

11.1　Tic Tac Toe 的运行示例

```
Welcome to Tic Tac Toe!
Do you want to be X or O?
X
The computer will go first.
   |   |
 O |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
   |   |
   |   |
What is your next move? (1-9)
3
   |   |
```

```
 O |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
 O |   | X
   |   |
What is your next move? (1-9)
4
   |   |
 O |   | O
   |   |
-----------
   |   |
 X |   |
   |   |
-----------
   |   |
 O |   | X
   |   |
What is your next move? (1-9)
5
   |   |
 O | O | O
   |   |
-----------
   |   |
 X | X |
   |   |
-----------
   |   |
 O |   | X
   |   |
The computer has beaten you! You lose.
Do you want to play again? (yes or no)
no
```

11.2　Tic Tac Toe 的源代码

在一个新的文件编辑器窗口中，输入如下源代码，并且把它保存为 tictactoe.py。然后，按下 F5 键运行这个游戏。

tictactoe.py

```
 1.# Tic Tac Toe
 2.
 3. import random
 4.
 5. def drawBoard(board):
 6.     # This function prints out the board that it was passed.
 7.
 8.     # "board" is a list of 10 strings representing the board (ignore index
0)
 9.     print('   |   |')
10.     print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
11.     print('   |   |')
12.     print('-----------')
13.     print('   |   |')
14.     print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
15.     print('   |   |')
16.     print('-----------')
17.     print('   |   |')
18.     print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
19.     print('   |   |')
20.
21. def inputPlayerLetter():
22.     # Lets the player type which letter they want to be.
23.     # Returns a list with the player's letter as the first item, and the
computer's letter as the second.
24.     letter = ''
25.     while not (letter == 'X' or letter == 'O'):
26.         print('Do you want to be X or O?')
27.         letter = input().upper()
28.
29.     # the first element in the list is the player's letter, the second is
the computer's letter.
30.     if letter == 'X':
31.         return ['X', 'O']
32.     else:
33.         return ['O', 'X']
34.
35. def whoGoesFirst():
36.     # Randomly choose the player who goes first.
37.     if random.randint(0, 1) == 0:
38.         return 'computer'
39.     else:
40.         return 'player'
41.
42. def playAgain():
43.     # This function returns True if the player wants to play again,
otherwise it returns False.
```

```
 44.     print('Do you want to play again? (yes or no)')
 45.     return input().lower().startswith('y')
 46.
 47. def makeMove(board, letter, move):
 48.     board[move] = letter
 49.
 50. def isWinner(bo, le):
 51.     # Given a board and a player's letter, this function returns True if
that player has won.
 52.     # We use bo instead of board and le instead of letter so we don't
have to type as much.
 53.     return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the
top
 54.     (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
 55.     (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
 56.     (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
 57.     (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
 58.     (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
 59.     (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
 60.     (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal
 61.
 62. def getBoardCopy(board):
 63.     # Make a duplicate of the board list and return it the duplicate.
 64.     dupeBoard = []
 65.
 66.     for i in board:
 67.         dupeBoard.append(i)
 68.
 69.     return dupeBoard
 70.
 71. def isSpaceFree(board, move):
 72.     # Return true if the passed move is free on the passed board.
 73.     return board[move] == ' '
 74.
 75. def getPlayerMove(board):
 76.     # Let the player type in their move.
 77.     move = ' '
 78.     while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board,
int(move)):
 79.         print('What is your next move? (1-9)')
 80.         move = input()
 81.     return int(move)
 82.
 83. def chooseRandomMoveFromList(board, movesList):
 84.     # Returns a valid move from the passed list on the passed board.
 85.     # Returns None if there is no valid move.
 86.     possibleMoves = []
```

```
 87.     for i in movesList:
 88.         if isSpaceFree(board, i):
 89.             possibleMoves.append(i)
 90.
 91.     if len(possibleMoves) != 0:
 92.         return random.choice(possibleMoves)
 93.     else:
 94.         return None
 95.
 96. def getComputerMove(board, computerLetter):
 97.     # Given a board and the computer's letter, determine where to move
and return that move.
 98.     if computerLetter == 'X':
 99.         playerLetter = 'O'
100.     else:
101.         playerLetter = 'X'
102.
103.     # Here is our algorithm for our Tic Tac Toe AI:
104.     # First, check if we can win in the next move
105.     for i in range(1, 10):
106.         copy = getBoardCopy(board)
107.         if isSpaceFree(copy, i):
108.             makeMove(copy, computerLetter, i)
109.             if isWinner(copy, computerLetter):
110.                 return i
111.
112.     # Check if the player could win on their next move, and block them.
113.     for i in range(1, 10):
114.         copy = getBoardCopy(board)
115.         if isSpaceFree(copy, i):
116.             makeMove(copy, playerLetter, i)
117.             if isWinner(copy, playerLetter):
118.                 return i
119.
120.     # Try to take one of the corners, if they are free.
121.     move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
122.     if move != None:
123.         return move
124.
125.     # Try to take the center, if it is free.
126.     if isSpaceFree(board, 5):
127.         return 5
128.
129.     # Move on one of the sides.
130.     return chooseRandomMoveFromList(board, [2, 4, 6, 8])
131.
132. def isBoardFull(board):
133.     # Return True if every space on the board has been taken. Otherwise
```

```
return False.
    for i in range(1, 10):
        if isSpaceFree(board, i):
            return False
    return True


print('Welcome to Tic Tac Toe!')

while True:
    # Reset the board
    theBoard = [' '] * 10
    playerLetter, computerLetter = inputPlayerLetter()
    turn = whoGoesFirst()
    print('The ' + turn + ' will go first.')
    gameIsPlaying = True

    while gameIsPlaying:
        if turn == 'player':
            # Player's turn.
            drawBoard(theBoard)
            move = getPlayerMove(theBoard)
            makeMove(theBoard, playerLetter, move)

            if isWinner(theBoard, playerLetter):
                drawBoard(theBoard)
                print('Hooray! You have won the game!')
                gameIsPlaying = False
            else:
                if isBoardFull(theBoard):
                    drawBoard(theBoard)
                    print('The game is a tie!')
                    break
                else:
                    turn = 'computer'

        else:
            # Computer's turn.
            move = getComputerMove(theBoard, computerLetter)
            makeMove(theBoard, computerLetter, move)

            if isWinner(theBoard, computerLetter):
                drawBoard(theBoard)
                print('The computer has beaten you! You lose.')
                gameIsPlaying = False
            else:
```

```
179.                if isBoardFull(theBoard):
180.                    drawBoard(theBoard)
181.                    print('The game is a tie!')
182.                    break
183.                else:
184.                    turn = 'player'
185.
186.    if not playAgain():
187.        break
```

11.3 设计程序

Tic Tac Toe 的流程图如图 11-1 所示。在 Tic Tac Toe 程序中，玩家可以选择他们想要 X 还是 O。谁先走是随机选择的。然后，玩家和计算机轮流下子。

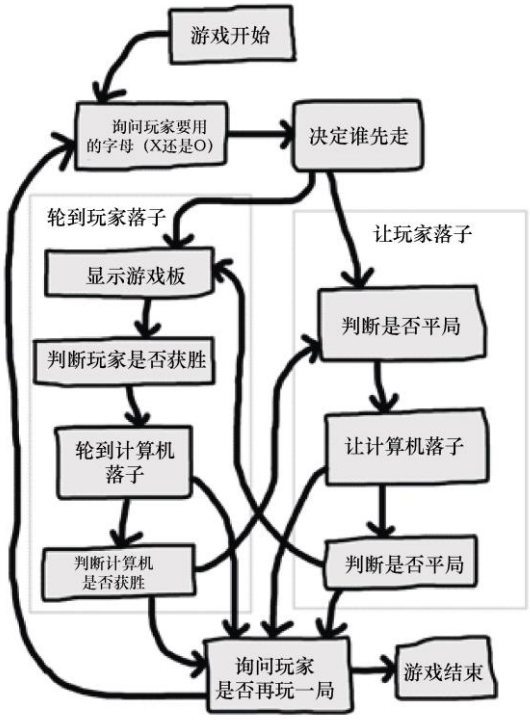


图 11-1 Tic Tac Toe 的流程图

流程图左边的方框是轮到玩家落子时发生的动作。右边的方框是轮到计算机落子时发生的动作。在玩家或计算机落子后，程序判断他们是否获胜或平局，然后游戏交换下棋的顺序。

在游戏结束之后，程序会询问玩家是否想要再玩一局。

用数据表示游戏板

首先，我们必须解决如何用变量中的数据来表示游戏板。在纸上，Tic Tac Toe 游戏板绘制成两条水平线和两条垂直线，在 9 个格子中要么是 X，要么是 O，要么是空的格子。

在这个程序中，用一个字符串列表来表示 Tic Tac Toe 游戏板。每个字符串将表示游戏板上 9 个格子中的一个。为了更便于记忆，列表中的索引就是对应的格子的编号，它们与键盘上数字键的排列一致，如图 11-2 所示。

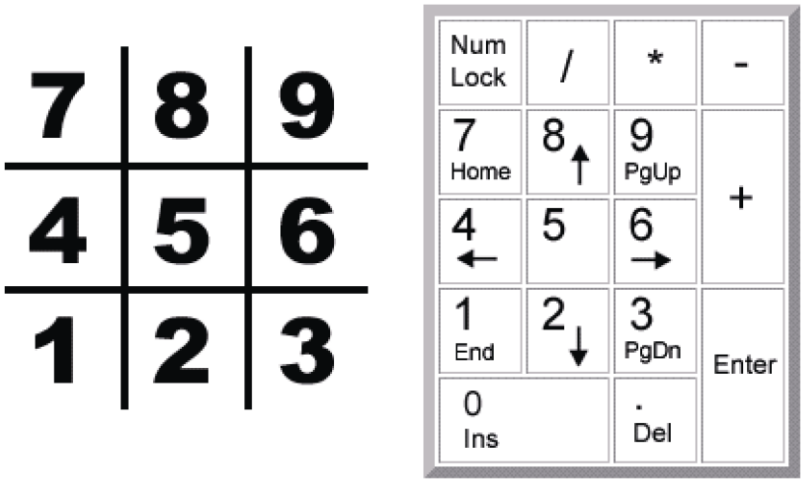


图 11-2　游戏板的编号类似于键盘的数字键

字符串'X'对应 X 玩家，'O'对应 O 玩家，一个空格字符' '对应一个空的格子。

所以，如果把包含 10 个字符串的列表保存到名为 board 的变量中，那么 board[7]将是游戏板上的左上格。board[5]将是中间格，board[4]将是其左边格，以此类推。程序将忽略列表中索引为 0 的字符串。玩家将输入 1 到 9 之间的一个数字，来告知游戏，他们想要在哪个格子上落子。

11.4　游戏 AI

这个游戏的 AI 需要能够查看游戏板，并且决定它们将要落子的格子的类型。为了尽量讲得清楚一些，我们将把 Tic Tac Toe 游戏板上的格子分为 3 种：角（corner）、边（side）和中心（center）。每个格子的样子如图 11-3 所示。

玩 Tic Tac Toe 时，AI 的智慧将遵循一个简单的算法。算法（algorithm）是计算一个结

果的一系列有限的指令。一个简单的程序可以使用几种不同的算法。算法可以用流程图来表示。Tic Tac Toe 的 AI 的算法将计算最佳的落子位置，如图 11-4 所示。

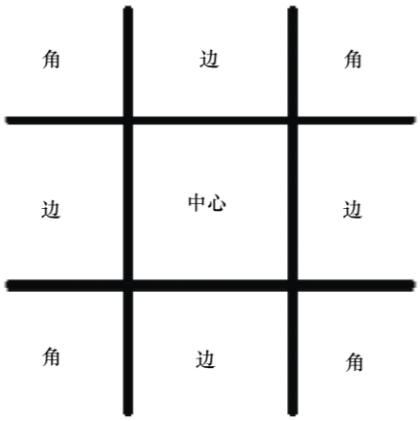


图 11-3 边、角和中心的位置

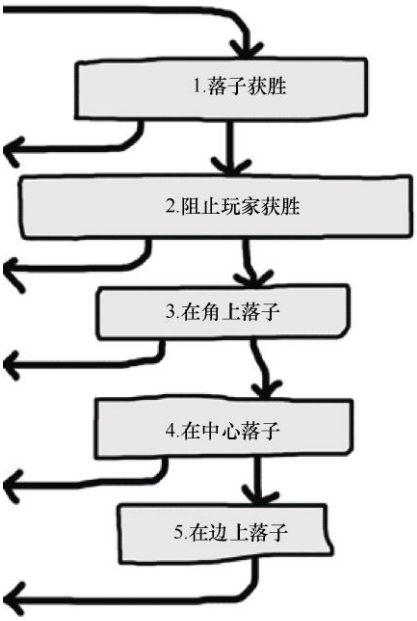


图 11-4 “让计算机落子”的算法的 5 个步骤，离开的箭头指向了方框“判断计算机是否获胜”

该 AI 的算法将遵循如下步骤：

1．首先，判断是否有能够让计算机获胜的落子位置。如果是，在那里落子；否则，执行步骤 2 。

2．判断这是否有能够让玩家失败的落子位置。如果是，在那里落子，以便堵住玩家；否则，执行步骤 3 。

3．判断是否还有角（格子 1 、 3 、 7 或者 9 ）为空。如果有，在此处落子；如果没有角为空，那么执行步骤 4 。

4．判断是否中心（格子 5）为空。如果有，在此处落子；否则，执行步骤 5 。

5．在任意一个边（格子 2 、 4 、 6 或 8 ）落子。没有其他的步骤了，因为如果执行到第 5 步，边是仅剩的空地了。

这些全部发生在图 11-1 所示的流程图中的“让计算机落子”的方框中。我们可以将这些信息添加到图 11-4 的流程图的方框中。

该算法在 getComputerMove()函数以及调用 getComputerMove()的其他函数中实现。

11.4.1　程序开始

```
 1.# Tic Tac Toe
 2.
 3. import random
```

前几行代码是一条注释，并且导入了 random 模块，以便可以调用 randint()函数。

11.4.2　在屏幕上打印游戏板

```
 5. def drawBoard(board):
 6.     # This function prints out the board that it was passed.
 7.
 8.     # "board" is a list of 10 strings representing the board (ignore index
0)
 9.     print('   |   |')
10.     print(' ' + board[7] + ' | ' + board[8] + ' | ' + board[9])
11.     print('   |   |')
12.     print('-----------')
13.     print('   |   |')
14.     print(' ' + board[4] + ' | ' + board[5] + ' | ' + board[6])
15.     print('   |   |')
16.     print('-----------')
17.     print('   |   |')
18.     print(' ' + board[1] + ' | ' + board[2] + ' | ' + board[3])
19.     print('   |   |')
```

drawBoard()函数将打印 board 参数所表示的游戏板。记住，board 是包含了 10 个字符串的一个列表，索引为 1 的字符串表示 Tic Tac Toe 游戏板上的格子 1，以此类推。我们把索引为 0 的字符串忽略了。这个游戏的许多函数，都是通过传入包含 10 个字符串的一个列表作为游戏板而工作的。

一定要保证字符串之间保留正确的空格，否则，当把游戏板打印到屏幕上时，看上去很滑稽。如下是调用 drawBoard()的一些示例（带有 board 参数），以及函数打印出的效果：

```
>>> drawBoard([' ', ' ', ' ', ' ', 'X', 'O', ' ', 'X', ' ', 'O']
   |   |
 X |   | O
   |   |
-----------
   |   |
 X | O |
   |   |
```

```
-----------
   |   |
   |   |
   |   |
>>> [' ', 'O', 'O', ' ', ' ', 'X', ' ', ' ', ' ', ' ']
   |   |
   |   |
   |   |
-----------
   |   |
   | X |
   |   |
-----------
   |   |
 O | O |
   |   |
>>> [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
   |   |
   |   |
   |   |
-----------
   |   |
   |   |
   |   |
-----------
   |   |
   |   |
   |   |
```

11.4.3 让玩家来选择 X 或 O

```
21. def inputPlayerLetter():
22.     # Lets the player type which letter they want to be.
23.     # Returns a list with the player's letter as the first item, and the
computer's letter as the second.
24.     letter = ''
25.     while not (letter == 'X' or letter == 'O'):
26.         print('Do you want to be X or O?')
27.         letter = input().upper()
```

inputPlayerLetter()函数询问玩家想要使用 X 还是 O。它会一直询问玩家，直到玩家输入 X 或 O。第 27 行通过调用字符串方法 upper()，自动把 input()所返回的字符串转变为大写字母。

这个 while 循环的条件包含了圆括号，这意味着先计算圆括号中的表达式。如果把变量 letter 设置为'X'，表达式的计算过程如下所示：

```
not (letter == 'X' or letter == 'O')
      ▼
not ('X' == 'X'    or 'X' == 'O')
      ▼
not (   True       or    False)
      ▼
not (True)
      ▼
not True
      ▼
   False
```

如果 letter 的值是'X'或'O'，那么循环的条件是 False，程序跳过 while 语句块往下执行。

```
29.     # the first element in the list is the player's letter, the second
is the computer's letter.
 30.    if letter == 'X':
 31.        return ['X', 'O']
 32.    else:
 33.        return ['O', 'X']
```

这个函数返回带有两个元素的一个列表。第 1 个元素（索引为 0 的字符串）是玩家的字母，第 2 个元素（索引为 1 的字符串）是计算机的字母。if-else 语句负责选择相应的列表以返回。

11.4.4　决定谁先走

```
 35. def whoGoesFirst():
 36.    # Randomly choose the player who goes first.
 37.    if random.randint(0, 1) == 0:
 38.        return 'computer'
 39.    else:
 40.        return 'player'
```

whoGoesFirst()函数执行了类似抛硬币的一个虚拟动作，以决定是计算机先走还是玩家走。所谓的抛硬币动作，就是调用 random.randint(0, 1)函数。如果这个函数调用返回 0，whoGoesFirst()函数将返回字符串'computer'。否则，该函数返回字符串'player'。调用这个函数的代码将根据这个返回值来获知谁将在游戏中先走。

11.4.5 询问玩家是否再玩一局

```
 42. def playAgain():
 43.     # This function returns True if the player wants to play again,
otherwise it returns False.
 44.     print('Do you want to play again? (yes or no)')
 45.     return input().lower().startswith('y')
```

playAgain()函数询问玩家是否想要再玩一局游戏。如果玩家输入'yes'、'YES'、'y'或者以字母 Y 开头的任何内容，该函数将返回 True。如果玩家输入任何其他的响应，该函数将返回 False。这个函数与 Hangman 游戏中的该函数完全相同。

11.4.6 在游戏板上做标记

```
 47. def makeMove(board, letter, move):
 48.     board[move] = letter
```

makeMove()函数很简单，只有 1 行代码。其参数是包含了 10 个字符串的一个列表 board，一个玩家的字母（'X'或'O'）letter，以及表示玩家想要在游戏板上落子的一个位置 move（这是 1 到 9 之间的一个整数）。

但是等一下。这行代码看上去似乎把 board 列表中的一个元素修改为 letter 中的值。但是因为这行代码在一个函数中，所以当函数返回时，会把参数 board 忘掉。对 board 所做的修改也会被忘掉吗？

事实上，并非如此。这是因为当我们把列表作为参数传递给函数时，它很特殊。我们实际上传递的是对这个列表的引用，而不是列表本身。我们来了解一下列表和列表引用之间的区别。

11.5 引用

尝试在交互式 shell 中输入如下代码：

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

按照我们目前所介绍的，这些结果很合理。我们把 42 赋给变量 spam，然后把 spam 中

的值赋给变量 cheese。当后边我们用 100 覆盖 spam 中的值时，这并不会影响到 cheese 中的值。这是因为 spam 和 cheese 是不同的变量，它们存储了不同的值。

但是列表不是这样工作的。当用等号（=）给变量赋一个列表时，实际上是为这个变量分配了一个列表引用。引用（reference）是指向一些数据的值。通过下面的代码，这会更容易理解一些。在交互式 shell 中输入：

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = spam
>>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

这看上去有点奇怪。代码只是修改了 cheese 列表，但是看上去 cheese 列表和 spam 列表都改变了。这是因为变量 spam 并没有包含列表值本身，而只是包含了对列表的一个引用，如图 11-5 所示。列表本身并没有被包含在任何变量中，而是独立于变量存在的。

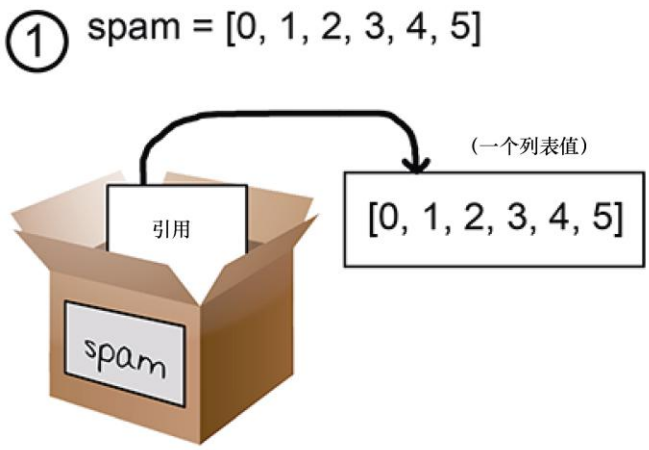


图 11-5　变量没有存储列表，而是存储了对列表的引用

注意，cheese = spam 表达式把 spam 中的列表引用复制到了 cheese 中，而并没有复制这个列表值本身。现在，spam 和 cheese 都存储了指向相同列表值的引用。但只有一个列表。这个列表并没有被复制，复制的是对这个列表的引用。图 11-6 展示了这种复制。

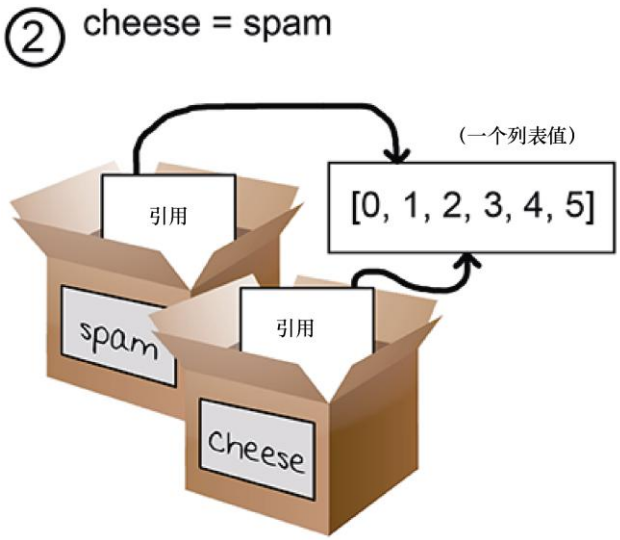


图 11-6　两个变量存储了对同一个列表的两个引用

所以代码行 cheese[1] = 'Hello!'修改了 spam 所指向的同一个列表。这就是为什么看上去 spam 拥有和 cheese 相同的列表值。它们所拥有的列表引用都指向了同一个列表，如图 11-7 所示。

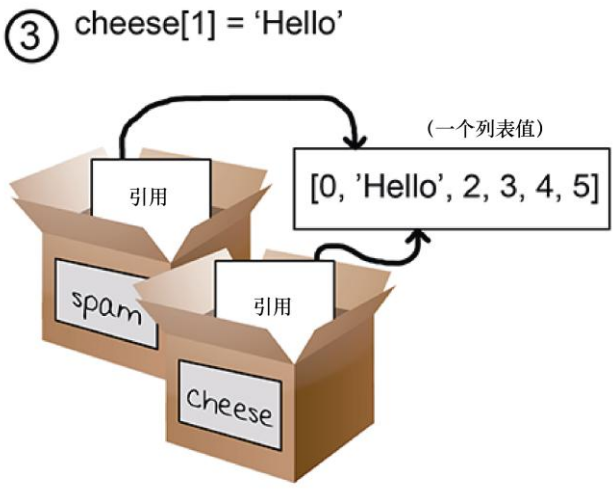


图 11-7　对这个列表的修改，也改变了引用这个列表的所有变量

如果你想要让 spam 和 cheese 存储两个不同的列表，就必须要创建两个不同的列表，而不是复制一个引用，如下所示。

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = [0, 1, 2, 3, 4, 5]
```

在上面的示例中，spam 和 cheese 存储了两个不同的列表（即便这两个列表的内容是完全一致的）。现在，如果修改其中一个列表，不会影响到另一个列表，因为 spam 和 cheese 引用了两个不同的列表：

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = [0, 1, 2, 3, 4, 5]
>>> cheese[1] = 'Hello!'
>>> spam
[0, 1, 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

图 11-8 展示了这两个引用如何指向两个不同的列表的情况。

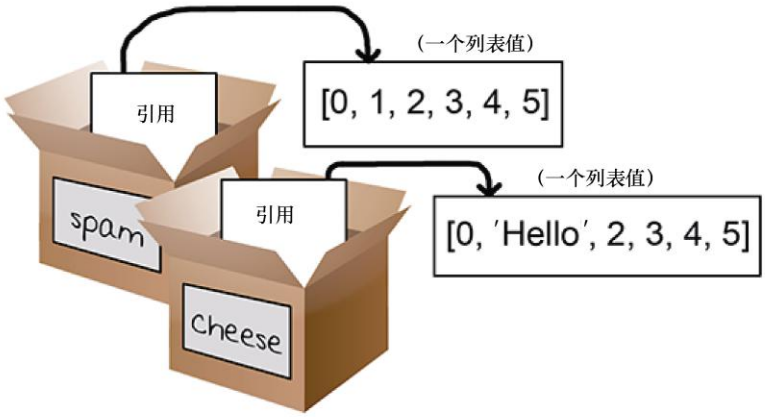


图 11-8　两个变量分别存储了对两个不同的列表的引用

字典也以相同的方式工作。变量没有存储字典，它们存储的是对字典的引用。

11.5.1　在 makeMove()中使用列表引用

让我们回来看看 makeMove()函数：

```
47. def makeMove(board, letter, move):
48.     board[move] = letter
```

当把一个列表值传递给 board 参数时，函数的局部变量实际是这个列表引用的一个副本，而不是列表的副本。但是这个引用的副本仍然指向最初的引用所指向的同一个列表，所以在这个函数中对于 board 的任何修改，也将作用于最初的列表之上。即使 board 是一个局部变量，makeMove()函数仍然修改了最初的列表。

参数 letter 和 move 是传递的字符串值和整数值的副本。因为它们是值的副本，所以即使在函数中修改了 letter 或 move，当我们再次调用 makeMove()时，最初的变量也不会被修改。

11.5.2 判断玩家是否获胜

```
 50. def isWinner(bo, le):
 51.     # Given a board and a player's letter, this function returns True
if that player has won.
52.      # We use bo instead of board and le instead of letter so we don't
have to type as much.
 53.    return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across
the top
 54.    (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
 55.    (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
 56.    (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
 57.    (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
 58.    (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
 59.    (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
 60.    (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal
```

在 isWinner()函数中，第 53 行到第 60 行实际上是一条很长的 return 语句。bo 和 le 是参数 board 和 letter 的缩写。这些更简短的名字意味着我们在函数中输入的内容会更少。记住，Python 并不关心我们如何对变量命名。

在 Tic Tac Toe 中，有 8 种可能的获胜方式。可以是最上面一行、中间一行和最下面一行连成一条线。也可以是最左边的列、中间的列和最右边的列连成为一条线。或者可以是两条对角线中的一条连成一条线。

注意，每行的条件都会判断 3 个格子是否等于所提供的字母（用操作符 and 组合起来），并且使用操作符 or 把 8 种不同的获胜方式组合起来。这意味着，8 种方式中必须有 1 种为真时，才能确定使用 le 中的字母的玩家获胜。

我们假设 le 是'O'并且 bo 是[' ', 'O', 'O', 'O', ' ', 'X', ' ', 'X', ' ', ' ']，游戏板如下所示：

```
   |   |
 X |   |
   |   |
-----------
   |   |
   | X |
   |   |
-----------
   |   |
 O | O | O
   |   |
```

第 53 行 return 关键字之后的表达式计算如下：

```
53.     return ((bo[7] == le and bo[8] == le and bo[9] == le) or # across the
top
54.     (bo[4] == le and bo[5] == le and bo[6] == le) or # across the middle
55.     (bo[1] == le and bo[2] == le and bo[3] == le) or # across the bottom
56.     (bo[7] == le and bo[4] == le and bo[1] == le) or # down the left side
57.     (bo[8] == le and bo[5] == le and bo[2] == le) or # down the middle
58.     (bo[9] == le and bo[6] == le and bo[3] == le) or # down the right side
59.     (bo[7] == le and bo[5] == le and bo[3] == le) or # diagonal
60.     (bo[9] == le and bo[5] == le and bo[1] == le)) # diagonal
```

首先，Python 会用变量 bo 和 le 中的值来替换这两个变量：

```
    return (('X' == 'O' and ' ' == 'O' and ' ' == 'O') or
    (' ' == 'O' and 'X' == 'O' and ' ' == 'O') or
    ('O' == 'O' and 'O' == 'O' and 'O' == 'O') or
    ('X' == 'O' and ' ' == 'O' and 'O' == 'O') or
    (' ' == 'O' and 'X' == 'O' and 'O' == 'O') or
    (' ' == 'O' and ' ' == 'O' and 'O' == 'O') or
    ('X' == 'O' and 'X' == 'O' and 'O' == 'O') or
    (' ' == 'O' and 'X' == 'O' and 'O' == 'O'))
```

接下来，Python 将会计算所有圆括号中的==比较，以得到一个布尔值：

```
    return ((False and False and False) or
    (False and False and False) or
    (True and True and True) or
    (False and False and True) or
    (False and False and True) or
    (False and False and True) or
    (False and False and True) or
    (False and False and True))
```

然后，Python 解释器将会计算所有圆括号中的表达式：

```
return ((False) or
(False) or
(True) or
(False) or
(False) or
(False) or
(False) or
(False))
```

由于现在圆括号中只有一个值，所以我们可以去掉这些圆括号：

```
return (False or
False or
True or
False or
False or
False or
False or
False)
```

现在计算由 or 操作符连接的整个这个表达式：

```
return (True)
```

再次去掉圆括号，我们只剩下一个值：

```
return True
```

所以给定了 bo 和 le 的值，表达式将会得到 True。程序就是这样来判断一个玩家是否赢得了游戏的。

11.5.3 复制游戏板的数据

```
62. def getBoardCopy(board):
63.     # Make a duplicate of the board list and return it the duplicate.
64.     dupeBoard = []
65.
66.     for i in board:
67.         dupeBoard.append(i)
68.
69.     return dupeBoard
```

这里的 getBoardCopy()函数让我们可以很容易地创建给定的包含了 10 个字符串的列表的一个副本，而这个列表用于表示 Tic Tac Toe 的游戏板。有时候，我们想要让 AI 算法只是对游戏板的一个临时副本做出临时性的改动，而不会修改最初的游戏板。在这种情况下，调用这个函数来创建游戏板列表的一个副本。第 64 行使用空的列表方括号[]来创建这个新的列表。

但是在第 64 行，在 dupeBoard 中存储的列表只是一个空列表。for 循环将会遍历参数 board，把最初的游戏板中的字符串值的副本添加到这个游戏板的副本中。最终，当循环结束之后，返回 dupeBoard。getBoardCopy()函数创建了最初的游戏板的一个副本，并且以 dupeBoard 的形式返回对这个新的游戏板的引用，而不是返回 board 中对最初的游戏板的引用。

11.5.4　判断游戏板上的格子是否为空

```
 71. def isSpaceFree(board, move):
 72.     # Return true if the passed move is free on the passed board.
 73.     return board[move] == ' '
```

这是一个简单的函数，它接受一个 Tic Tac Toe 游戏板和一个可能的落子作为参数，将是否可以落子作为结果返回。记住，board 列表上空的格子用一个单个的空格字符串来表示。如果格子的索引所对应的元素不等于这个空格字符串，那么表示格子已经被占用了。

11.5.5　让玩家输入他们的落子

```
 75. def getPlayerMove(board):
 76.     # Let the player type in their move.
 77.     move = ' '
 78.     while move not in '1 2 3 4 5 6 7 8 9'.split() or not isSpaceFree(board,
int(move)):
 79.         print('What is your next move? (1-9)')
 80.         move = input()
 81.     return int(move)
```

getPlayerMove()函数要求玩家输入他们想要落子的格子的编号。在玩家输入一个 1 到 9 之间的整数之前，循环确保不会向下执行。给定 Tic Tac Toe 游戏板作为 board 参数，该函数也会判断玩家所输入的格子是否已经被占用了。

while 循环中的两行代码直接要求玩家输入 1 到 9 之间的数字。如果操作符 or 左边或右边的表达式中有一个为 True，那么第 78 行的条件就为 True。

左边的表达式使用相应的字符串创建来一个列表（split()方法）并且判断 move 是否在该列表中，从而判断玩家的落子是否等于'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'或'9'。

'1 2 3 4 5 6 7 8 9'.split()的计算结果是['1', '2', '3', '4', '5', '6', '7', '8', '9']，但是前一种形式更

容易输入。

右边的表达式判断玩家输入的落子位置在游戏板上是否为空的格子。通过调用isSpaceFree()函数来判断这一点。记住，如果传递的落子位置在棋盘上是空的，isSpaceFree()将返回 True。注意，isSpaceFree()期待为 move 接受一个整数，所以 int()函数返回一个整数形式的 move。

在两边都添加了 not 操作符，以便当任意一个需求不满足的时候，条件为 True。这将导致这个循环一遍遍地询问玩家，直到他们输入一个正确的落子位置。

最后，第 81 行返回玩家输入的任意落子位置的整数形式。记住，input()返回的是字符串，所以调用 int()函数来返回该字符串的整数形式。

11.6 短路求值

你可能已经注意到，在 getPlayerMove()函数中有一个可能存在的问题。如果玩家输入'Z'或其他的非整数字符串，会怎么样？or 左边的表达式 move not in '1 2 3 4 5 6 7 8 9'.split()将按照预期返回 False，然后 Python 将计算操作符 or 右边的表达式。

但是调用 int('Z')会出现一个错误。因为 int()函数只能接受数字字符的字符串，如'9'或'0'，而无法接受像'Z'这样的字符串，所以 Python 会给出这个错误。

尝试在交互式 shell 中输入如下内容，查看这种类型的错误的一个示例：

```
>>> int('42')
42
>>> int('Z')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    int('Z')
ValueError: invalid literal for int() with base 10: 'Z'
```

但是，当你玩 Tic Tac Toe 的时候并且尝试输入'Z'作为落子位置时，却没有发生这种错误。这是因为 while 循环的条件短路了。

短路（short-circuiting）在这里表示由于关键字 or 左边部分（move not in '1 2 3 4 5 6 7 8 9'.split()）为 True，Python 解释器就知道整个表达式的计算结果为 True。关键字 or 右边的表达式是 True 还是 False 已经无关紧要了，因为 or 操作符只需要有一边的值为 True 即可。

可以这样来考虑：表达式 True or False 的结果为 True，而表达式 True or True 的结果也为 True。如果 or 左边的值为 True，那就不需要关心右边的值是什么了：

False and <<<anything>>>总是计算为 False
True or <<<anything>>>总是计算为 True

所以，Python 停止查看表达式的剩余部分，甚至都不再计算 not isSpaceFree(board, int(move))部分。这意味着只要 move not in '1 2 3 4 5 6 7 8 9'.split()为 True，根本就不会调用 int()函数和 isSpaceFree()函数。

这样程序就会很正常地工作，因为如果左边是 True，那么 move 就不是 number 形式的一个字符串，就会导致 int()出现一个错误。只有当 move 不是单个的数字字符串时，表达式 move not in '1 2 3 4 5 6 7 8 9'.split()才会计算为 False。在这种情况下，调用 int()函数也不会引发错误。

11.6.1　短路运算的一个示例

如下是一个小程序，它会给出短路运算的一个很好的示例。尝试在交互式 shell 中输入如下代码：

```
>>> def ReturnsTrue():
      print('ReturnsTrue() was called.')
      return True

>>> def ReturnsFalse():
      print('ReturnsFalse() was called.')
      return False

>>> ReturnsTrue()
ReturnsTrue() was called.
True
>>> ReturnsFalse()
ReturnsFalse() was called.
False
```

当调用 ReturnsTrue()时，它会打印出'ReturnsTrue() was called.'，然后也会显示 ReturnsTrue()的返回值。ReturnsFalse()的工作方式也一样。

现在尝试在交互式 shell 中输入如下代码：

```
>>> ReturnsFalse() or ReturnsTrue()
ReturnsFalse() was called.
ReturnsTrue() was called.
True
>>> ReturnsTrue() or ReturnsFalse()
ReturnsTrue() was called.
True
```

第 1 部分很合理：表达式 ReturnsFalse() or ReturnsTrue()调用了这两个函数，所以我们

看到了两条打印消息。

但是，第 2 个表达式只显示了'ReturnsTrue() was called.'，而没有显示'ReturnsFalse() was called.'。这是因为 Python 根本没有调用 ReturnsFalse()函数。由于操作符 or 的左边是 True，ReturnsFalse()函数返回什么都无关紧要，Python 也不会调用这个函数。计算被“短路”了。

这同样适用于操作符 and。尝试在交互式 shell 中输入如下代码：

```
>>> ReturnsTrue() and ReturnsTrue()
ReturnsTrue() was called.
ReturnsTrue() was called.
True
>>> ReturnsFalse() and ReturnsFalse()
ReturnsFalse() was called.
False
```

如果 and 操作符的左边是 False，那么整个表达式是 False。and 操作符右边是 True 还是 False 都无关紧要，所以 Python 不会计算右边的函数。False and True 和 False and False 的运算结果都是 False，所以 Python“短路”了该计算。

11.6.2 从落子列表中选择一个落子

```
83. def chooseRandomMoveFromList(board, movesList):
84.     # Returns a valid move from the passed list on the passed board.
85.     # Returns None if there is no valid move.
86.     possibleMoves = []
87.     for i in movesList:
88.         if isSpaceFree(board, i):
89.             possibleMoves.append(i)
```

chooseRandomMoveFromList()函数对于程序后边的 AI 代码很有用。board 参数是一个字符串列表，它表示了 Tic Tac Toe 游戏板。第 2 个参数 movesList 是可供选择的格子的整数列表。例如，如果 movesList 是[1, 3, 7, 9]，那么表示 chooseRandomMoveFromList()应该返回表示角格子的一个整数。

然而，chooseRandomMoveFromList()将先判断格子是否可以落子。possibleMoves 列表刚开始的时候是一个空白列表。for 循环会遍历 movesList。那些导致 isSpaceFree()函数返回 True 的落子位置，就会通过 append()方法添加到 possibleMoves 中。

```
91.     if len(possibleMoves) != 0:
92.         return random.choice(possibleMoves)
93.     else:
94.         return None
```

此时，possibleMoves 列表拥有了 movesList 中仍然为空的所有格子。如果这个列表不为空，那么在游戏板上至少有一个可以落子的位置。

但是这个列表可能是空的。例如，如果 movesList 是[1, 3, 7, 9]，但是用 board 参数表示的游戏板上所有的角格子都已经被占领了，那么 possibleMoves 列表将是[]。在这种情况下，len(possibleMoves)的结果为 0，chooseRandomMoveFromList()函数返回的值是 None。下一节将介绍 None 值。

11.7　None 值

None 值是表示没有值的一个值。None 是数据类型 NoneType 的唯一的值。当需要表示“不存在”或“以上都不是”的一个值时，使用 None 值会很有帮助。

例如，假设有一个变量叫做 quizAnswer，它存储了用户对于一些真假问答题的回答。这个变量可以为用户存储答案 True 或 False。如果用户跳过并且没有回答某一个问题，就可以把 quizAnswer 设置为 None。使用 None 更好一些，否则的话，看上去好像是用户回答了问题，而实际上他并没有回答。

当到达 chooseRandomMoveFromList()函数的末尾（并且没有 return 语句）的时候，函数以 None 作为一个返回值。None 值写为不带引号的形式，只是一个大写的“N”和小写的“one”。

另外要注意的是，None 在交互式 shell 中并不会像其他值那样显示：

```
>>> 2 + 2
4
>>> 'This is a string value.'
'This is a string value.'
>>> None
```

看上去没有返回任何东西的函数，其实际上返回的是 None 值。例如，print()返回 None：

```
>>> spam = print('Hello world!')
Hello world!
>>> spam == None
True
```

11.7.1　创建计算机的人工智能

```
 96. def getComputerMove(board, computerLetter):
 97.     # Given a board and the computer's letter, determine where to move
and return that move.
 98.     if computerLetter == 'X':
 99.         playerLetter = 'O'
```

```
100.    else:
101.        playerLetter = 'X'
```

getComputerMove()函数包含了这个 AI 的代码。第 1 个参数是表示一个 Tic Tac Toe 游戏板的 board 参数。第 2 个参数是计算机所使用的字母'X'或'O'，存储于变量 computerLetter 中。前几行直接把另一个字母赋值给名为 playerLetter 的变量。无论计算机是 X 还是 O，都可以使用相同的代码。

这个函数将返回一个 1 到 9 之间的整数，以表示计算机的落子。

记住，Tic Tac Toe AI 算法的工作方式如下：

- 第 1 步，判断这是否是能够让计算机获胜的落子位置。如果是的，落子；否则，执行步骤 2 。
- 第 2 步，判断这是否是能够让计算机失败的落子位置。如果是的，在此处落子，以便堵住玩家；否则，执行步骤 3。
- 第 3 步，判断是否还有角的格子（1、3、7 或者 9）为空。如果有，在此处落子；如果没有角的格子为空，那么执行步骤 4 。
- 第 4 步，判断中心的格子（5）是否为空。如果是，在此处落子；否则，执行步骤 5。
- 第 5 步，在任意一个边的格子（2、4、6 或 8）落子。没有更多的步骤了，因为如果执行到第 5 步，边的格子是仅剩的空的格子了。

11.7.2 计算机判断自己能否落子即获胜

```
103.    # Here is our algorithm for our Tic Tac Toe AI:
104.    # First, check if we can win in the next move
105.    for i in range(1, 10):
106.        copy = getBoardCopy(board)
107.        if isSpaceFree(copy, i):
108.            makeMove(copy, computerLetter, i)
109.            if isWinner(copy, computerLetter):
110.                return i
```

最重要的是，如果在下一步落子中计算机可以获胜，计算机应该立即落下这制胜的一子。从第 105 行开始的 for 循环，遍历 1 到 9 之间的各种可能的落子。循环中的代码将会模拟计算机落子所发生的情况。

循环中的第 1 行（106 行）会创建 board 列表的一个副本。正是通过这样，在循环中模拟落子，而不会修改存储在 board 变量中真正的 Tic Tac Toe 游戏板。getBoardCopy()会返回相同的另外一个游戏板列表值。

第 107 行判断这个格子是否为空，如果是，在游戏板的副本上模拟落子。如果这个落子导致计算机获胜，这个函数返回表示落子位置的整数。

如果没有能够获胜的落子格子，循环将结束，并且程序继续执行第 113 行代码。

11.7.3　计算机判断玩家是否可以落子即获胜

```
112.    # Check if the player could win on their next move, and block them.
113.    for i in range(1, 10):
114.        copy = getBoardCopy(board)
115.        if isSpaceFree(copy, i):
116.            makeMove(copy, playerLetter, i)
117.            if isWinner(copy, playerLetter):
118.                return i
```

接下来，这段代码将模拟人类玩家在每个格子上的落子。代码类似于前面的循环，只不过是把玩家的字母放在游戏板的副本上。如果 isWinner()函数显示玩家可以通过这一步落子而获胜，那么计算机将返回相同的落子位置来阻止发生这种情况。

如果人类玩家不能在一步落子中获胜，for 循环将结束，并继续执行第 121 行代码。

11.7.4　依次判断角、中心和边

```
120.    # Try to take one of the corners, if they are free.
121.    move = chooseRandomMoveFromList(board, [1, 3, 7, 9])
122.    if move != None:
123.        return move
```

以[1, 3, 7, 9]为参数调用 chooseRandomMoveFromList()，以确保它返回一个角格子的整数：1、3、7 或 9。如果所有角格子都被占领了，chooseRandomMoveFromList()函数将返回 None，并且程序执行移动到第 126 行代码。

```
125.    # Try to take the center, if it is free.
126.    if isSpaceFree(board, 5):
127.        return 5
```

如果没有可用的角，而中心格子为空，第 127 行代码在中心格子落子。如果中心格子不为空，执行将移动到第 130 行。

```
129.    # Move on one of the sides.
130.    return chooseRandomMoveFromList(board, [2, 4, 6, 8])
```

这段代码也会调用 chooseRandomMoveFromList()，只是传递的是边格子的列表([2, 4, 6, 8])。这个函数不会返回 None，因为边格子是仅剩的有可能为空的格子了。getComputerMove()函数和 AI 算法到此结束。

11.7.5 判断游戏板是否满了

```
132. def isBoardFull(board):
133.     # Return True if every space on the board has been taken. Otherwise
return False.
134.     for i in range(1, 10):
135.         if isSpaceFree(board, i):
136.             return False
137.     return True
```

最后一个函数是 isBoardFull()。该函数接受包含 10 个字符串的一个列表作为游戏板参数，如果列表中的每个索引（除了索引 0，它被忽略掉了）都有'X'或'O'，那么该函数返回 True。如果在 board 中至少有一个格子设置为空格字符串' '，那么该函数返回 False。

这个 fox 循环将检查 board 列表中的索引 1 到 9。只要发现游戏板上有一个空的格子（也就是当 isSpaceFree(board, i)返回 True 时），isBoardFull()函数就返回 False。

如果执行通过了循环中的每次迭代，那么就没有空的格子了。然后，第 137 行将执行 return True。

11.7.6 游戏的开始

```
140.print('Welcome to Tic Tac Toe!')
```

第 140 行是函数之外的第一行，所以当运行这个程序时，它是要执行的第一行代码。它向玩家打了个招呼。

```
142. while True:
143.     # Reset the board
144.     theBoard = [' '] * 10
```

第 142 行的 while 循环的条件是 True，循环会一直继续，直到执行遇到一条 break 语句。第 133 行在一个名为 theBoard 的变量中设置了主 Tic Tac Toe 游戏板。这是包含 10 个字符串的一个列表，每个字符串都是一个单独的空格' '。

第 144 行并没有输入这个完整的列表，而是使用了列表复制。输入[' '] * 10 要比输入[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']更简短。

11.7.7 决定玩家的符号和谁现走

```
145.     playerLetter, computerLetter = inputPlayerLetter()
```

inputPlayerLetter()函数让玩家输入他们想要的 X 或 O。这个函数返回包含两个字符串的列表，['X', 'O']或['O', 'X']。多变量赋值的技术将把返回列表中的第 1 个元素赋值给 playerLetter，把第 2 个元素赋值给 computerLetter。

```
146.    turn = whoGoesFirst()
147.    print('The ' + turn + ' will go first.')
148.    gameIsPlaying = True
```

whoGoesFirst()函数随机地决定谁将先走，并且返回字符串'player'或'computer'，并且第 147 行告诉玩家谁将先走。变量 gameIsPlayer 将记录游戏是仍在进行，还是谁赢了或打成平局。

11.7.8　运行玩家的轮次

```
150.    while gameIsPlaying:
```

只要 gameIsPlaying 被设置为 True，第 150 行循环将在玩家轮次和计算机轮次的代码之间来回切换。

```
151.        if turn == 'player':
152.            # Player's turn.
153.            drawBoard(theBoard)
154.            move = getPlayerMove(theBoard)
155.            makeMove(theBoard, playerLetter, move)
```

最初通过在第 146 行调用 whoGoesFirst()函数来设置变量 turn，把它设置为'player'或'computer'。如果变量 turn 等于'computer'，那么第 151 行的条件为 False，执行会跳转到第 169 行。

第 153 行调用 drawBoard()函数，并且传递变量 theBoard，从而把 Tic Tac Toe 游戏板打印到屏幕上。然后，getPlayerMove()函数让玩家输入他们的落子（并且还会确认是有效的落子）。makeMove()函数把玩家的 X 或 O 添加到变量 theBoard 中。

```
157.            if isWinner(theBoard, playerLetter):
158.                drawBoard(theBoard)
159.                print('Hooray! You have won the game!')
160.                gameIsPlaying = False
```

现在，玩家已经落子了，计算机要判断他们是否通过这步落子赢得了游戏。如果 isWinner()函数返回 True，if 语句块的代码将显示获胜的游戏板，并且打印一条消息告诉玩家他们获胜了。

还会把 gameIsPlaying 变量设置为 False，以便不再继续执行计算机的轮次。

```
161.           else:
162.               if isBoardFull(theBoard):
163.                   drawBoard(theBoard)
164.                   print('The game is a tie!')
165.                   break
```

如果玩家没有通过其最近的落子而获胜，可能他们的落子会填满了整个游戏板，打成平局。在这个 else 语句块中，如果没有更多的空的格子可以落子，isBoardFull()函数返回 True。在这种情况下，第 162 行开始的 if 语句块显示了平局的游戏板，并且告诉玩家出现了平局。

```
166.               else:
167.                   turn = 'computer'
```

如果玩家还没有获胜或者打成平局，那么第 167 行将 turn 变量设置为'computer'，以便在下一次迭代中运行计算机轮次的代码。

11.7.9 运行计算机的轮次

如果第 151 行条件中 turn 变量不是'player'，那么肯定是计算机的轮次。else 语句块中的代码和玩家轮次的代码类似。

```
169.       else:
170.           # Computer's turn.
171.           move = getComputerMove(theBoard, computerLetter)
172.           makeMove(theBoard, computerLetter, move)
174.           if isWinner(theBoard, computerLetter):
175.               drawBoard(theBoard)
176.               print('The computer has beaten you! You lose.')
177.               gameIsPlaying = False
178.           else:
179.               if isBoardFull(theBoard):
180.                   drawBoard(theBoard)
181.                   print('The game is a tie!')
182.                   break
183.               else:
184.                   turn = 'player'
```

第 170 行到 184 行代码几乎与玩家轮次的 152 行到 167 行代码相同。唯一的区别是，这部分代码使用了计算机的字母并且调用了 getComputerMove()。

如果游戏没有输赢或平局，第 184 行把 turn 设置为玩家的轮次。在 while 循环中，已经

没有其他的代码了，所以执行将跳回到 150 行的 while 语句。

```
186.    if not playAgain():
187.        break
```

从第 150 行开始的 while 语句块的后面，紧跟着的是第 186 行和 187 行。当游戏已经结束时，把 gameIsPlaying 设置为 False，此时游戏会询问玩家是否想再玩一局。

如果 playAgain()返回 False，那么 if 语句的条件是 True（因为 not 操作符会将布尔值取反），会执行 break 语句。这会跳出从第 142 行开始的 while 循环。但是由于在该 while 语句块之后已没有代码行了，所以程序终止了。

11.8　本章小结

创建一个能够玩游戏的程序，要仔细考虑 AI 所面临的所有可能发生的情况，以及它应该如何应对这些情况。Tic Tac Toe 游戏的 AI 很简单，因为相较于象棋或跳棋而言，Tic Tac Toe 没有那么多可能的落子位置。

我们的 AI 会判断是否有任何可能的落子位置能让自己获胜。否则的话，它会检查是否必须要阻止玩家的落子。然后 AI 直接选择任何可供落子的角格子、中心格子和边格子。这是计算机所遵循的一个简单的算法。

实现 AI 的关键是创建游戏板数据的副本，并且在副本上模拟落子。通过这种方式，AI 代码可以判断一个落子是否导致输或赢。然后 AI 可以在真实的游戏板上落子。这种类型的模拟，对于预测是一个好的落子还是一个糟糕的落子非常有效。

第 12 章　Bagels

本章主要内容：

- 复合赋值操作符+=、-=、*=、/=；
- random.shuffle()函数；
- 列表方法 sort()和 join()；
- 字符串插值（也叫做字符串格式化）；
- 转换说明符%s；
- 嵌套循环。

在本章中，我们将介绍 Python 的一些新的方法和函数。我们将介绍复合赋值操作符和字符串插值。如果之前你不能够做什么事情的话，这些方法和函数也并不能带来什么改变，但是，它们是让编程变得更为简单的快捷方法。

Bagels 是可以和朋友一起玩的一个推理游戏。你的朋友想到一个随机的、没有重复的 3 位数字，你尝试去猜测它是什么。每次猜测之后，朋友就会给出 3 种类型的线索：

- Bagels——你猜测的 3 个数都不在神秘数字中；
- Pico——你猜测的是神秘数字中的一个数，但是位置不对；
- Fermi——你猜测的是正确位置上的一个正确的数。

每次猜测之后，都可以得到多条线索。如果神秘数字是 456，而猜测是 546，那么线索就是“pico pico fermi”。6 提供的线索是“fermi”，5 和 4 提供的线索是“pico pico”。

12.1　Bagels 的运行示例

Bagels 的运行示例如下。

```
I am thinking of a 3-digit number. Try to guess what it is.
Here are some clues:
When I say:    That means:
  Pico         One digit is correct but in the wrong position.
  Fermi        One digit is correct and in the right position.
  Bagels       No digit is correct.
I have thought up a number. You have 10 guesses to get it.
Guess #1:
123
Fermi
Guess #2:
```

```
453
Pico
Guess #3:
425
Fermi
Guess #4:
326
Bagels
Guess #5:
489
Bagels
Guess #6:
075
Fermi Fermi
Guess #7:
015
Fermi Pico
Guess #8:
175
You got it!
Do you want to play again? (yes or no)
no
```

12.2　Bagels 的源代码

如果输入下面的代码后出现错误，请使用 http://invpy.com/diff/bagels 上的在线 diff 工具，把你的代码与书中的代码进行比较。

bagels.py

```
 1. import random
 2. def getSecretNum(numDigits):
 3.     # Returns a string that is numDigits long, made up of unique random
digits.
 4.     numbers = list(range(10))
 5.     random.shuffle(numbers)
 6.     secretNum = ''
 7.     for i in range(numDigits):
 8.         secretNum += str(numbers[i])
 9.     return secretNum
10.
11. def getClues(guess, secretNum):
12.     # Returns a string with the pico, fermi, bagels clues to the user.
```

```
13.     if guess == secretNum:
14.         return 'You got it!'
15.
16.     clue = []
17.
18.     for i in range(len(guess)):
19.         if guess[i] == secretNum[i]:
20.             clue.append('Fermi')
21.         elif guess[i] in secretNum:
22.             clue.append('Pico')
23.     if len(clue) == 0:
24.         return 'Bagels'
25.
26.     clue.sort()
27.     return ' '.join(clue)
28.
29. def isOnlyDigits(num):
30.     # Returns True if num is a string made up only of digits. Otherwise
returns False.
31.     if num == '':
32.         return False
33.
34.     for i in num:
35.         if i not in '0 1 2 3 4 5 6 7 8 9'.split():
36.             return False
37.
38.     return True
39.
40. def playAgain():
41.      # This function returns True if the player wants to play again,
otherwise it returns False.
42.     print('Do you want to play again? (yes or no)')
43.     return input().lower().startswith('y')
44.
45. NUMDIGITS = 3
46. MAXGUESS = 10
47.
48. print('I am thinking of a %s-digit number. Try to guess what it is.' %
(NUMDIGITS))
49. print('Here are some clues:')
50. print('When I say:    That means:')
51. print('  Pico         One digit is correct but in the wrong position.')
52. print('  Fermi        One digit is correct and in the right position.')
53. print('  Bagels       No digit is correct.')
```

```
54.
55. while True:
56.     secretNum = getSecretNum(NUMDIGITS)
57.     print('I have thought up a number. You have %s guesses to get it.' %
(MAXGUESS))
58.
59.     numGuesses = 1
60.     while numGuesses <= MAXGUESS:
61.         guess = ''
62.         while len(guess) != NUMDIGITS or not isOnlyDigits(guess):
63.             print('Guess #%s: ' % (numGuesses))
64.             guess = input()
65.
66.         clue = getClues(guess, secretNum)
67.         print(clue)
68.         numGuesses += 1
69.
70.         if guess == secretNum:
71.             break
72.         if numGuesses > MAXGUESS:
73.                 print('You ran out of guesses. The answer was %s.' %
(secretNum))
74.
75.     if not playAgain():
76.         break
```

12.2.1　设计程序

程序的流程图如图 12-1 所示，它描述了在这个游戏中发生了什么以及发生的顺序。

12.2.2　代码如何工作

```
1. import random
2. def getSecretNum(numDigits):
3.     # Returns a string that is numDigits long, made up of unique random digits.
```

在程序开始处，导入了 random 模块。然后定义了一个名为 getSecretNum()的函数。这个函数创建了一个神秘数字，其中的每个数都是唯一的。参数 numDigits 允许函数创建任意位数的神秘数字，而不是只能创建 3 位数的神秘数字。例如，我们可以给 numDigits 传递 4 或 6，来创建 4 位或 6 位的一个神秘数字。

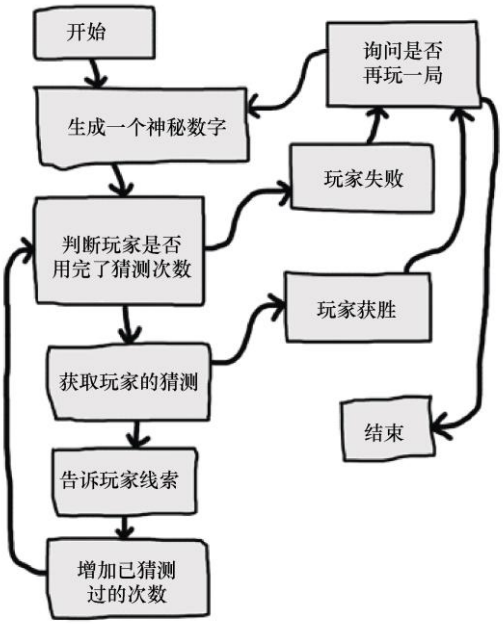


图 12-1 Bagels 游戏的流程图

12.2.3 打乱一组唯一数的顺序

```
4.    numbers = list(range(10))
5.    random.shuffle(numbers)
```

第 4 行的 list(range(10))总是得到[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]。输入 list(range(10))只是会更简单一些。变量 numbers 是包含了所有 10 个数字的一个列表。

12.3 random.shuffle()函数

random.shuffle()函数随机修改列表元素的顺序。这个函数并不返回一个值，而是把传递给它的列表“就地”做了改动。这有点类似于第 11 章 Tic Tac Toe 游戏中 makeMove()函数，makeMove()函数把传递给它的列表就地做了修改，而不是返回修改过的一个新列表。这就是为什么我们没有编写诸如 numbers = random.shuffle(numbers)这样的代码。

通过在交互式 shell 中输入如下的代码，来尝试体验 random.shuffle()函数：

```
>>> import random
>>> spam = list(range(10))
>>> print(spam)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> random.shuffle(spam)
>>> print(spam)
[3, 0, 5, 9, 6, 8, 2, 4, 1, 7]

>>> random.shuffle(spam)
>>> print(spam)
[1, 2, 5, 9, 4, 7, 0, 3, 6, 8]

>>> random.shuffle(spam)
>>> print(spam)
[9, 8, 3, 5, 4, 7, 1, 2, 0, 6]
```

我们希望 Bagels 游戏中的神秘数字的每一位数均不相同。如果神秘数字中没有像'244'或'333'这样重复的数，Bagels 游戏会更有意思。shuffle()函数可以帮助我们做到这一点。

从打乱次序的数中获取神秘数字

```
 6.    secretNum = ''
 7.    for i in range(numDigits):
 8.        secretNum += str(numbers[i])
 9.    return secretNum
```

这个神秘数字将是打乱次序的整数列表的前 numDigits 个数字组成的一个字符串。例如，如果这个打乱次序的列表中的数是[9, 8, 3, 5, 4, 7, 1, 2, 0, 6]，而 numDigits 是 3，那么 getSecretNum()函数返回的字符串将会是'983'。

为了做到这一点，secretNum 变量一开始是一个空白字符串。第 7 行的 for 循环迭代了 numDigits 次。在循环的每次迭代中，都会从打乱顺序的列表中获取索引为 i 的整数，将其转换成一个字符串，并连接到变量 secretNum 的末尾。

例如，如果 numbers 引用列表[9, 8, 3, 5, 4, 7, 1, 2, 0, 6]，那么在第 1 次迭代中，将会把 numbers[0]（也就是 9）传递给 str()函数，该函数返回'9'，会将其连接到变量 secretNum 的末尾。在第 2 次迭代中，对 numbers[1]（也就是 8）做了同样的处理；在第 3 次迭代中，对 numbers[2]（也就是 3）做了同样的处理。最终返回的 secretNum 值是'983'。

注意，这个函数中的 secretNum 包含了一个字符串，而不是一个整数。这看上去可能有点奇怪，但是记住，我们不能把整数连接到一起。表达式 9 + 8 + 3 的结果是 20，而我们想要的是'9' + '8' + '3'，其结果是'983'。

12.4 复合赋值操作符

第 8 行的+=操作符是一个复合赋值操作符（augmented assignment operators）。通常，如果想要把一个值增加或者连接到一个变量中，应该使用如下的代码：

```
spam = 42
spam = spam + 10

eggs = 'Hello '
eggs = eggs + 'world!'
```

复合赋值操作符是一种快捷方式，它使得我们不必再重复地输入了变量名称。下面代码做了和上面代码相同的事情：

```
spam = 42
spam += 10      # Like spam = spam + 10

eggs = 'Hello '
eggs += 'world!' # Like eggs = eggs + 'world!'
```

还有一些其他的复合赋值操作符。尝试在交互式 shell 中输入如下代码：

```
>>> spam = 42
>>> spam -= 2
>>> spam
40
>>> spam *= 3
>>> spam
120
>>> spam /= 10
>>> spam
12.0
```

计算要给出的线索

```
11. def getClues(guess, secretNum):
12.     # Returns a string with the pico, fermi, bagels clues to the user.
13.     if guess == secretNum:
14.         return 'You got it!'
```

getClues()函数将根据参数 guess 和 secretNum，返回线索 fermi、pico 和 bagels 组成的一个字符串。最显而易见和最简单的步骤是判断猜测是否与神秘数字相等。当玩家的猜测

和神秘数字相等的时候，第 14 行会返回'You got it!'。

```
16.    clue = []
17.
18.    for i in range(len(guess)):
19.        if guess[i] == secretNum[i]:
20.            clue.append('Fermi')
21.        elif guess[i] in secretNum:
22.            clue.append('Pico')
```

如果猜测和神秘数字不相等，代码必须知道要给玩家什么线索。clue 中的列表最初为空，根据需要来加入'Fermi'和'Pico'字符串。

通过循环遍历 guess 和 secretNum 中可能的索引来做到这一点。因为两个变量中的字符串具有相同的长度，所以第 18 行既可以使用 len(guess)，也可以使用 len(secretNum)，其效果是一样的。由于 i 的值从 0 变为 1，再变为 2，依次类推，所以第 19 行会判断 guess 的第 1 个字母、第 2 个字母、第 3 个字母以及之后的字母是否与 secretNum 中的相同索引中的数字相等。如果相等，第 20 行将把字符串'Fermi'加入到 clue 中。

如果不相等，第 21 行将判断 guess 中第 i 个位置的数是否存在于 secretNum 中。如果存在，我们知道这个数在神秘单词中，但是位置不正确。随后第 22 行会把'Pico'添加到 clue 中。

```
23.    if len(clue) == 0:
24.        return 'Bagels'
```

如果循环之后 clue 列表是空的，那么我们就知道 guess 中根本没有正确的数。这种情况下，第 24 行返回字符串'Bagels'作为唯一的线索。

12.5　列表方法 sort()

```
26.    clue.sort()
```

列表有一个叫做 sort()的方法，它按照字母顺序或数字顺序重新排列列表中的元素。尝试在交互式 shell 中输入如下代码：

```
>>> spam = ['cat', 'dog', 'bat', 'anteater']
>>> spam.sort()
>>> spam
['anteater', 'bat', 'cat', 'dog']

>>> spam = [9, 8, 3, 5, 4, 7, 1, 2, 0, 6]
```

```
>>> spam.sort()
>>> spam
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

sort()方法没有返回一个排序的列表，而是对这个列表进行了所谓的“就地”排序。这就和 reverse()方法的工作方式一样。

我们不想要使用这样的一行代码：return spam.sort()，因为那将会返回一个 None 值（而这正是 sort()返回的值）。相反，我们想要的是单独的一行 spam.sort()，然后是代码行 return spam。

我们想要对 clue 列表进行排序的原因是，去除掉线索中和顺序相关的额外信息。如果 clue 是['Pico', 'Fermi', 'Pico']，那这将会告诉玩家猜测的中间数是在正确的位置上。由于另外两个线索都是 Pico，玩家就会知道必须要把神秘数字中的第 1 个数和第 3 个数进行交换。

如果线索总是按照字母先后顺序来排序，那么玩家就无法确认 Fermi 线索指的是哪个数。这是我们想要游戏实现的效果。

12.6 字符串方法 join()

```
27.     return ' '.join(clue)
```

字符串方法 join()将字符串的列表连接起来，作为一个单个的字符串返回。调用该方法的字符串（在第 27 行，这是一个空格字符串' '）会出现在列表中每个字符串之间（即作为分隔符）。例如，在交互式 shell 中输入如下代码：

```
>>> ' '.join(['My', 'name', 'is', 'Zophie'])
'My name is Zophie'
>>> ', '.join(['Life', 'the Universe', 'and Everything'])
'Life, the Universe, and Everything'
```

所以，第 27 行返回的字符串，是把 clue 中的各个字符串组合到一起，每个字符串之间会有一个空格。join()就像是和 split()相反的字符串方法。split()方法通过分割字符串而返回一个列表，而 join()方法返回组合列表而得到的一个字符串。

12.6.1 检查字符串中是否只包含数字

```
29. def isOnlyDigits(num):
30.     # Returns True if num is a string made up only of digits. Otherwise
returns False.
31.     if num == '':
32.         return False
```

isOnlyDigits()函数帮助判断玩家输入的是否是一个有效的猜测。第 31 行代码判断 num

是否为空字符串，如果是，返回 False。

```
34.     for i in num:
35.         if i not in '0 1 2 3 4 5 6 7 8 9'.split():
36.             return False
37.
38.     return True
```

for 循环遍历字符串 num 中的每个字符。在每次迭代中，i 的值将是一个单个的字符。在 for 语句块中，代码判断 i 是否不存在于'0 1 2 3 4 5 6 7 8 9'.split()返回的列表中（split()的返回值是['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']，只是 split()更易于输入一些）。如果 i 不存在于列表之中，我们就知道 num 中包含了非数字字符。在这种情况下，第 36 行返回 False。

如果程序跳过 for 循环往下执行，我们就知道 num 中的每个字符都是一个数字。在这种情况下，第 38 行返回 True。

12.6.2　了解玩家是否想再玩一局

```
40. def playAgain():
41.     # This function returns True if the player wants to play again,
otherwise it returns False.
42.     print('Do you want to play again? (yes or no)')
43.     return input().lower().startswith('y')
```

playAgain()函数与第 9 章和第 11 章中的该函数相同。第 43 行的这个长长的表达式的结果到底是 True 还是 False，取决于玩家给出的回答。

12.6.3　游戏的开始

```
45. NUMDIGITS = 3
46. MAXGUESS = 10
47.
48. print('I am thinking of a %s-digit number. Try to guess what it is.' %
(NUMDIGITS))
49. print('Here are some clues:')
50. print('When I say:    That means:')
51. print('  Pico         One digit is correct but in the wrong position.')
52. print('  Fermi        One digit is correct and in the right position.')
53. print('  Bagels       No digit is correct.')
```

在定义了所有的函数之后，这里是程序真正开始的地方。在我们的程序中，对于要猜测的位数使用的不是整数 3，而是常量 NUMDIGITS。允许玩家猜测的最多次数同样使用的是

常量 MAXGUESS，而不是整数 10。这样会更易于修改允许猜测的次数和神秘数字的位数。只要修改第 45 行和 46 行代码，而无需再做其他的修改，程序剩余的部分即可工作。

print()函数调用将告诉玩家游戏的规则，以及线索 Pico、Fermi 和 Bagels 所表达的含义。第 48 行的 print()调用在末尾加入了% (NUMDIGITS)，并且在字符串中加入了%s。这种技术叫做字符串插值。

12.7 字符串插值

字符串插值是编码的一种快捷方式。通常，如果想要在一个字符串中使用一个变量中的字符串值的话，必须使用连接操作符+：

```
>>> name = 'Alice'
>>> event = 'party'
>>> where = 'the pool'
>>> day = 'Saturday'
>>> time = '6:00pm'

>>> print('Hello, ' + name + '. Will you go to the ' + event + ' at ' + where
+ ' this ' + day + ' at ' + time + '?')
Hello, Alice. Will you go to the party at the pool this Saturday at 6:00pm?
```

正如你所看到的，很难把连接多个字符串的代码输入到一行中。相反，可以使用字符串插值（stringinterpolation），它允许放入像%s 这样的占位符。这样的占位符叫做转换说明符（conversion specifiers）。然后，在末尾放置所有的变量名称。每个%s 会被代码行末尾的一个变量所替换。例如，下面代码所做的事情和之前的代码相同：

```
>>> name = 'Alice'
>>> event = 'party'
>>> where = 'the pool'
>>> day = 'Saturday'
>>> time = '6:00pm'

>>> print('Hello, %s. Will you go to the %s at %s this %s at %s?' % (name,
event, where, day, time))
Hello, Alice. Will you go to the party at the pool this Saturday at 6:00pm?
```

字符串插值可以使得代码的输入更为简单。第 1 个变量名用于第 1 个%s，第 2 个变量名用于第 2 个%s，以此类推。我们的变量的数目必须和%s 转换说明符的数目相同。

使用字符串插值而不是字符串连接的另一个好处是，插值对于任意的数据类型都有效，而不仅仅是对字符串有效。所有值都会自动转换为字符串数据类型。如果连接一个整数和一个字符串，会得到如下的错误：

```
>>> spam = 42
>>> print('Spam == ' + spam)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

字符串连接只能把两个字符串组合起来，而 spam 是一个整数。我们必须要记住要使用 str(spam)，而不是 spam。但是字符串插值会为我们做这种字符串的转换。尝试在交互式 shell 中输入：

```
>>> spam = 42
>>> print('Spam is %s' % (spam))
Spam is 42
```

字符串插值也叫做字符串格式化（string formatting）。

12.7.1　创建神秘数字

```
55. while True:
56.     secretNum = getSecretNum(NUMDIGITS)
57.     print('I have thought up a number. You have %s guesses to get it.' %
(MAXGUESS))
58.
59.     numGuesses = 1
60.     while numGuesses <= MAXGUESS:
```

第 55 行是一个无限 while 循环，条件是 True，所以它会一直循环直到执行一条 break 语句。在这个无限循环中，我们通过 getSecretNum()函数得到一个神秘的数字，把 NUMDIGITS 作为参数传递给该函数，以告知想要一个多少位的神秘数字。把这个神秘数字赋值给 secretNum。记住，secretNum 中的值是一个字符串，而不是整数。

第 57 行使用字符串插值而不是字符串连接，以告诉玩家总共能够猜测了多少次。第 59 行把变量 numGuesses 设置为 1，表示这是第 1 次猜测。然后第 60 行开始是一个新的 while 循环，只要 numGuesses 小于或等于 MAXGUESS，这个循环就一直进行下去。

12.7.2　获取玩家的猜测

```
61.         guess = ''
62.         while len(guess) != NUMDIGITS or not isOnlyDigits(guess):
63.             print('Guess #%s: ' % (numGuesses))
64.             guess = input()
```

guess 变量会保存 input()函数返回的玩家的猜测。这部分代码会一直循环并要求玩家做出猜测，直到玩家输入一个有效的猜测。有效的猜测只包含数字，并且和神秘数字拥有相同的位数。这就是从第 62 行开始的 while 循环所做的事情。

第 61 行把 guess 变量设置为空字符串，以便第一次判断 while 循环条件时，条件为 False，从而确保执行可以进入到循环中。

12.7.3 根据玩家的猜测给出线索

```
66.         clue = getClues(guess, secretNum)
67.         print(clue)
68.         numGuesses += 1
```

当执行跳过从第 62 行开始的 while 循环之后，guess 中包含了一个有效的猜测。把变量 guess 和 secretNum 传递给 getClues()函数。该函数返回包含线索的一个字符串，第 67 行把该字符串显示给玩家。第 68 行使用加法复合赋值操作符把 numGuesses 加 1。

12.7.4 判断玩家的输赢

注意，第 60 行的第 2 个 while 循环位于另一个 while 循环之中，后者是从第 55 行开始。这种位于循环之中的循环叫做嵌套循环（nested loop）。任何 break 语句或 continue 语句将只能跳出最内层的循环，而不是跳出外围循环。

```
70.         if guess == secretNum:
71.             break
72.         if numGuesses > MAXGUESS:
73.             print('You ran out of guesses. The answer was %s.' % (secretNum))
```

如果 guess 和 secretNum 的值相同，玩家就正确地猜测到了神秘数字，第 71 行跳出从第 60 行开始的 while 循环。

如果 guess 和 secretNum 的值不相同，将继续执行第 72 行，它会判断玩家是否用完了猜测次数。如果用完了，程序会告诉玩家，他们失败了。

此时，执行跳转回到第 60 行的 while 循环，这里会让玩家再做一次猜测。如果玩家用完了猜测次数（或使用第 71 行的 break 语句跳出了循环），那么执行将跳过这个循环到第 75 行。

12.7.5 询问玩家是否再玩一局

```
75.     if not playAgain():
76.         break
```

第 75 行通过调用 playAgain()函数来询问玩家是否想再玩一局。如果 playAgain()返回 False，会跳出从第 55 行开始的 while 循环。因为这个循环之后就没有其它代码，程序结束了。

如果 playAgain()返回 True，那么将不会执行 break 语句，执行将跳转回到第 55 行。程序将创建一个新的神秘数字，以便玩家可以开始玩新的一局游戏。

12.8　本章小结

就编程来讲，Bagels 是一个简单的游戏，但是想在游戏中获胜却很难。不过，如果你一直玩，最终将发现有更好的方法来猜测和使用游戏所提供的线索。这就好像越坚持编程，也会越来越善于编程。

本章介绍了一些新的函数和方法（(random.shuffle()、sort()和 join())，以及两个方便的快捷方式。当想要修改一个变量的时候，复合赋值操作符减少了输入工作量，例如 spam = spam + 1 可以简写为 spam += 1。字符串插值通过在字符串中使用%s（称为转换说明符）而不是使用多个字符串连接操作，从而让代码更容易阅读。

下一章不是直接和编程相关的，但是对于本书后边各章中想要创建的游戏来说，下一章是必不可少的。我们将介绍笛卡尔坐标和负数的数学概念。这些概念将用于 Sonar、Reversi 和 Dodger 游戏，但是笛卡尔坐标和负数也可以用于很多其他游戏中。如果你已经了解了这些概念，那么对下一章做一个简短的阅读，以复习一下相关知识。

第 13 章　笛卡尔坐标

本章主要内容：

- 笛卡尔坐标系；
- X 轴和 Y 轴；
- 加法可交换性；
- 绝对值和 abs()函数。

本章不会介绍新的游戏，而是介绍了本书后面将用到的一些简单的数学概念。在 2D 游戏中，屏幕上的图形可以向左或向右移动，也可以向上或向下移动。这两个方向构成了两个维度（即 2D 或二维空间）。对象在二维计算机屏幕上来回移动，这种游戏需要一种方法将屏幕上的位置转换成程序能够处理的整数。

这就是笛卡尔坐标系的用武之地。坐标是表示屏幕上的一个特定的点的数字。这些数字可以作为整数存储到程序的变量中。

13.1　网格和笛卡尔坐标

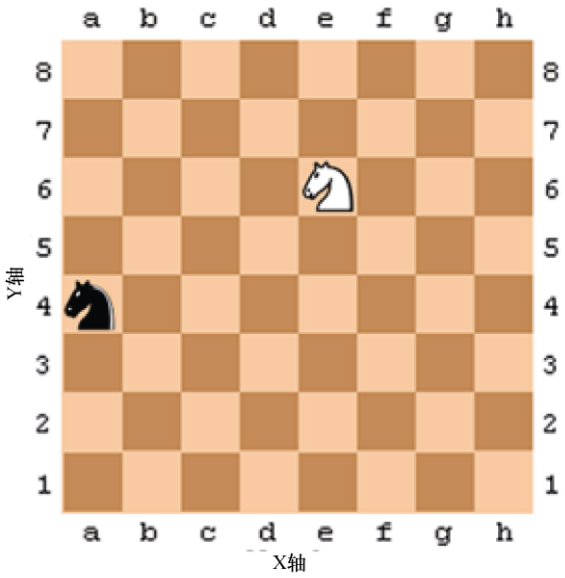


图 13-1　一个简单的国际象棋棋盘，黑方的马在 a4，白方的马在 e6

在棋盘上指定一个特定位置的通用方式是使用字母和数字来标记每行和每列。图 13-1 是标记了每行和每列的一个国际象棋棋盘。

在棋盘上，一个格子的坐标是一个行和一个列的组合。在国际象棋中，马这个棋子看起来像是一个马头。在图 13-1 中，白方的马位于 e6 点，黑方的马位于 a4 点。

这个带有标记的棋盘就是一个笛卡尔坐标系。通过使用一个行坐标和一个列坐标，我们可以给棋盘上的每一个格子一个唯一的坐标。如果你已经了解 math 类中的笛卡尔坐标系，可能会知道行和列都是使用数字来表示的。棋盘看上去如图 13-2 所示。

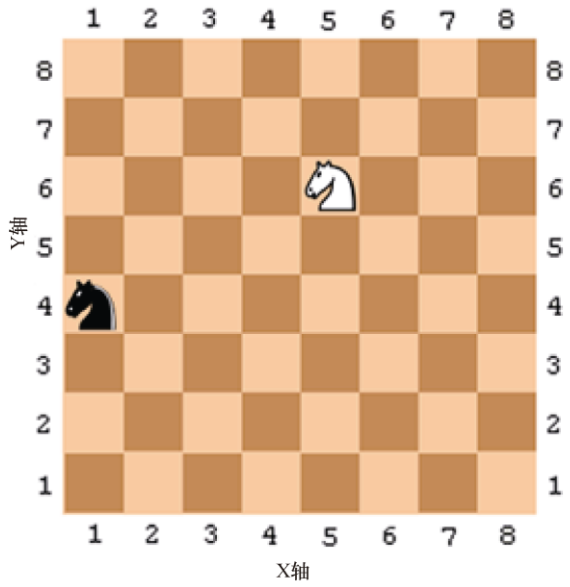


图 13-2　相同的棋盘，只是行和列都是数字坐标

在一列的左边和右边的数字是 X 轴（X-axis）的部分。在一行的上边和下边的数字是 Y 轴（Y-axis）的部分。坐标总是先说 X 轴，后说 Y 轴。在图 13-2 中，白方的马位于坐标（5，6），而不是坐标（6，5）。黑方的马位于坐标（1，4），不要误认为是（4，1）。

注意，黑方的马要移动到白方的马的位置的话，必须向上移动两个格子并向右移动 4 个格子。但是，我们无须查看棋盘来完成这次移动。如果我们知道白方的马位于坐标（5，6），黑方的马位于坐标（1，4），那么可以使用减法来完成这次移动。

用白方的马的 X 坐标减去黑方的马的 X 坐标：5-1=4。黑方的马必须沿 X 轴移动 4 个格子。

用白方的马的 Y 坐标减去黑方的马的 Y 坐标：6-4=2。黑方的马必须沿 Y 轴移动 2 个格子。

通过用坐标数字做一些数学运算，我们就可以计算出两个坐标轴之间的距离。

13.2 负数

笛卡尔坐标系用到了负数。负数（negative number）是小于零的数字。数字前的负号表示它是一个负数。-1 比 0 小。-2 比-1 小。如果你认为常规的数字（叫做正数，positive number）是从 1 开始递增，那么就可以把负数当作从-1 开始递减。0 本身既不是正数也不是负数。在图 13-3 中，我们可以看到正数在数轴上向右递增，负数在数轴上向左递减。

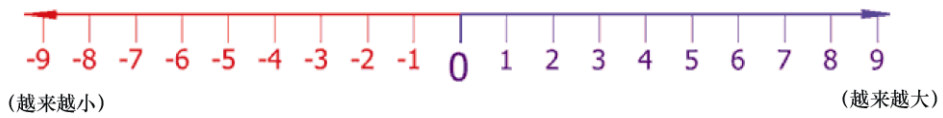


图 13-3 数轴

数轴对于查看负数的加法和减法的结果很有用。表达式 4 + 3 可以看作白色马从位置 4 开始，向右移动 3 个格子（加法表示递增，也就是向右移动）。

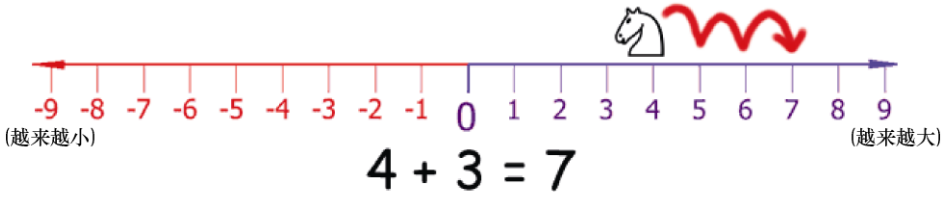


图 13-4 白方的马向右移动增加了坐标

可以看到白色马最终的位置是 7，如图 13-4 所示。这是合理的，因为 4+3 是 7。

白方的马向左移动的话用减法。减法表示递减，向左移动。4 - 6 表示白方的马开始位置是 4，向左移动 6 个格子，如图 13-5 所示。

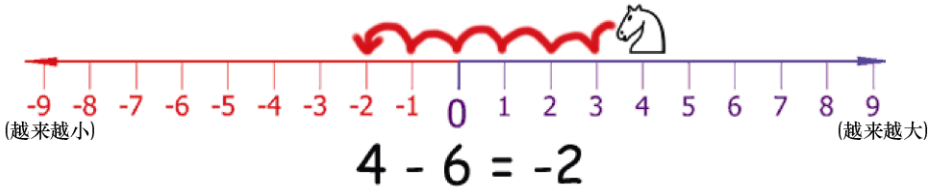


图 13-5 白方的马向左移动减少了坐标

白方的马最终的位置是-2。这意味着 4-6 等于-2。

如果加上或减去一个负数，白方的马将向相反方向移动。如果加上一个负数，马将向左

移动。如果减去一个负数，马将向右移动。表达式-6 -（-4）将等于-2。马起始位置是-6，向右移动 4 个格子，如图 13-6 所示。注意，-6 -（-4）和-6 + 4 的结果相同。

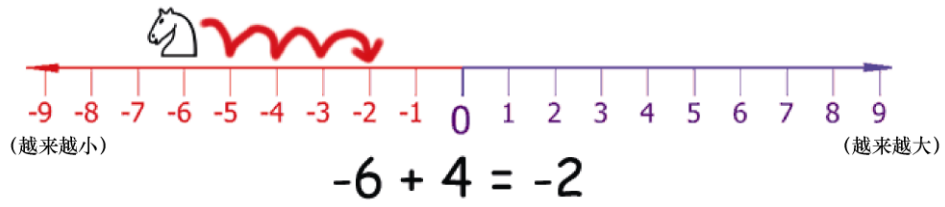


图 13-6　即使白方的马最初在一个负坐标，向右移动仍然是增加坐标

我们可以把 X 轴当作是一个数轴。添加另一条上下走向的数轴作为 Y 轴。如果把这两条数轴放在一起，就得到如图 13-7 所示的一个笛卡尔坐标系了。

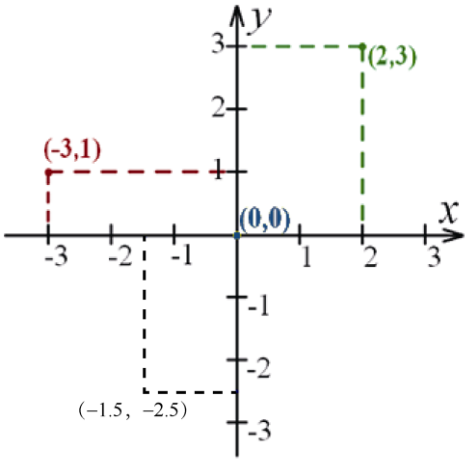


图 13-7　把两个数轴放在一起创建一个笛卡尔坐标系

加上一个正数（或减去一个负数），马就会向数轴上方移动；减去一个正数（或加上一个负数），马将向下移动。

坐标（0, 0）叫做原点（origin）。

13.3　数学技巧

当有一个数轴在你面前，减去和加上负数就会很简单。没有数轴也可以很简单。下面有 3 种技巧帮助我们自己来加上和减去负数。

技巧 1：“负号吃掉它左边的加号”

当我们看到负号左边有一个加号时，可以用这个负号来替代掉这两个符号（也就是完全忽略掉加号）。把它想象成是这个负号“吃掉”了它左边的加号。结果仍然是相同的，因为加上一个负数与减掉一个正数是一样的。4 + -2 和 4 - 2 都等于 2，如图 13-8 所示。。

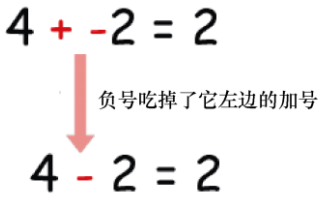


图 13-8 技巧 1：一个正数和一个负数相加

技巧 2：“两个减号合并为一个加号”

当看到两个减号中间没有数字时，可以把它们合并成为一个加号，如图 13-9 所示。结果仍然是相同的，因为减去一个负值和加上一个正值是一样的。

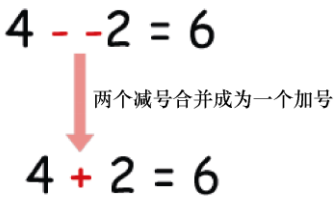


图 13-9 技巧 2：一个正数减去一个负数

技巧 3：加法的可交换性

我们总是可以交换加法中的数字。这就是加法的可交换性（commutative property）。这意味着，像 6 + 4 和 4 + 6 这样的交换，并不会改变结果。

如果我们来数图 13-10 中的小方框的数目，可以看到加法不在乎是否交换了加数。

假设我们正在将一个负数和一个正数相加，就像-6 + 8。因为我们在做加法，所以可以交换数字的顺序而不会改变结果。-6 + 8 和 8 + -6 是相等的。

那么当看到 8 + -6 时，负号可以吃掉它左边的加号，问题就变为 8 - 6 = 2。但是，这意味

着-6 + 8 也是 2！我们重新排列问题，得到了相同的结果，在没有使用计算器或计算机的情况下，这使得问题更容易计算，如图 13-11 所示。

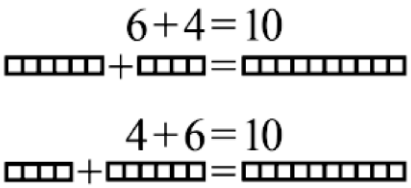


图 13-10　技巧 3：加法的可交换性

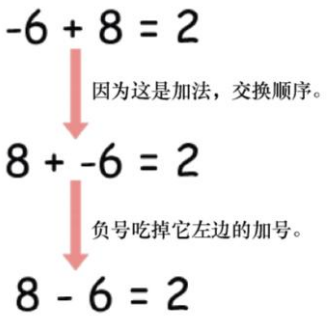


图 13-11　使用数学技巧

13.4　绝对值和 abs()函数

一个数字的绝对值（absolute value）就是数字前没有负号。因此，正数的绝对值没有变化，但是负数的绝对值变成了正数。例如，-4 的绝对值是 4。-7 的绝对值是 7。5（这是一个正数）的绝对值就是 5。

我们可以将两个对象位置相减并且取差的绝对值，来计算它们之间的距离。假设白方的马的位置是 4，黑方的马的位置是-2。它们之间的距离是 6，因为 4 -（-2）是 6，6 的绝对值是 6。

无论数字的顺序是什么，这种计算方法都有效。-2 - 4（也就是负 2 减去 4）是-6，-6 的绝对值也是 6。

Python 的 abs()函数返回整数的绝对值。尝试在交互式 shell 中输入：

```
>>> abs(-5)
5
>>> abs(42)
42
>>> abs(-10.5)
10.5
```

13.5　计算机屏幕的坐标系

通常计算机屏幕使用的坐标系的原点为(0, 0)位于屏幕的左上角，坐标值向下和向右递增，如图 13-12 所示。没有负坐标。大部分的计算机图形都使用这种坐标系，在本书的游戏

中，也使用这种坐标系。

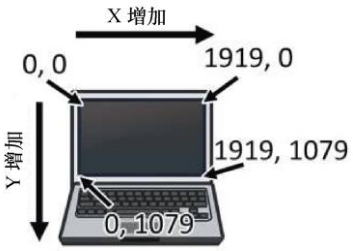


图 13-12　计算机屏幕的坐标系

13.6　本章小结

大多数编程不需要理解很多数学知识。在本章之前，我们一直使用简单的加法和乘法。

要描述二维区域中的一个特定位置，需要使用笛卡尔坐标系。坐标有两个数：X 坐标和 Y 坐标。X 轴是左右延伸，Y 轴是上下延伸。在计算机屏幕上，原点是左上角，坐标向右方和下方递增。

在本章中，我们介绍了 3 种数学技巧，使得正数和负数的加法变得更简单。第 1 种技巧是负号吃掉它左边的加号。第 2 种技巧是两个相邻的减号合并成一个加号。第 3 种技巧是交换相加的数字的顺序。

在本书剩余的部分，我们将在游戏中使用本章中介绍的概念，因为我们的游戏都拥有二维区域。所有的图形化游戏都需要理解笛卡尔坐标是如何工作的。

第 14 章　Sonar Treasure Hunt

本章的主要内容：

- 数据结构；
- 列表方法 remove()；
- 字符串方法 isdigit()；
- sys.exit()函数。

在本章的游戏中，我们第一次使用在第 13 章介绍过的笛卡尔坐标。这个游戏也有数据结构（数据结构是描述诸如包含列表的列表这样的复杂变量的一种有趣的方法）。当我们编写的游戏变得越来越复杂时，就需要在数据结构中组织数据。

在本章的游戏中，玩家把声纳设备放置到海洋中的不同位置，以找到沉没的藏宝箱的位置。声纳（sonar）是轮船用于定位海底物体的一种技术。（本章的游戏中的）声纳设备将告诉玩家距离最近的藏宝箱有多远，但是不会告诉玩家藏宝箱在哪个方向。不过通过放下多个声纳设备，玩家就可以知道藏宝箱在哪儿了。

要收集 3 个藏宝箱，而玩家只有 16 个声纳设备可用于找到它们。假设你无法看到图 14-1 中的藏宝箱。因为每个声纳设备只能发现有多远的距离，而不能获知方向，所以藏宝箱可能会在声纳设备四周方形环区的任何地方（如图 14-1 所示）。

但是，多个声纳设备一起工作，就可以把藏宝箱缩小到一个确切的位置，方形区域在这个位置环彼此相交，如图 14-2 所示（通常这些闭合环区是圆形的，但是本章游戏中将使用方形的闭合环区，以便让程序更简单一些）。

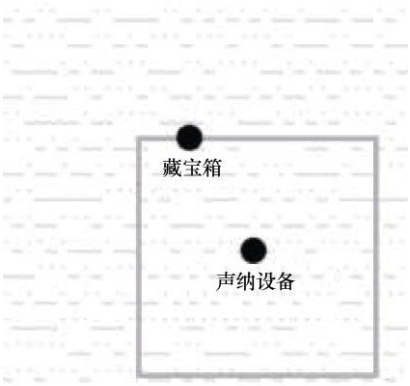


图 14-1　声纳设备四周的方形环区可能包含（隐藏的）藏宝箱

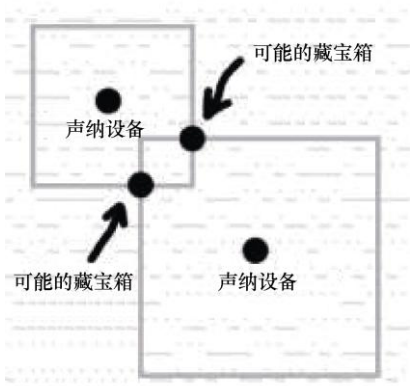


图 14-2　结合多个方形区域可以显示出藏宝箱可能在哪里

14.1 Sonar Treasure Hunt 的运行示例

```
S O N A R !
Would you like to view the instructions? (yes/no)
no
           1       2       3       4       5
  012345678901234567890123456789012345678901234567890123456789
 0 `~~~`~~~`~`~~`~~~~~~`~``~~~~~`~`~~~`~``~``~~````~`~
```~`~~~~``  0
 1~`~~~```~~~~`~`~~`~``~`~~```~`~`~~`~`~~~~~~`~`````~
`~~`~~~~` 1
 2 `~``~``~~~`~``~`~`~``~`````~~~~~~~~~~`~`~~`~``~~~~~~
```~~`~``` 2
 3 ``~`~~``~`~``~`~`~`~~`~`~~`~`~``~~~`~``~````~
``````~~~~``~`` 3
 4 ``~~`~~~``~``~~````~`~`~`~``~~~``~~~```~`~~`~~~`~`~`~~
`~~~~``` 4
 5~~```~~~`~`~~``~`~``~```~`~~`~~~~~`~~``~`~`~~~`~~
`~`~`~`~~~` 5
 6 ``~~`````~~~~`~`~~~```~~~~`~~`~~`~~```~~`~~~`~~~
``~`~~~``~~~~6
 7 `~`````````~```~``~``~~`~~~~`~~``~``~~~```~`~~`~``~
``~~```~~7
 8 `~````~```~`~~`~~~`~~``~~~``~`~``~~~``~`~`````~`~~
```~`~~~~` 8
 9~```~~`~`~``~``~~``~``~```~`~``~~~~`~`~`~~~`~`~`~
`~~~``~~``` 9
10 ```~`~```~``~``~`~~`~``~````~``~~~`~~`~~``~~~~`~~~`~
`~~````~10
11 ```~```~~~`~```~~`~~~`~`````~`~~`~`~~`~~`~`~~`~~~
````~````~` 11
12~~~`~`~~~``~~~~~~~`~~~``~`~`~~`~`~~`~```~~~```~~
`~~`~``~``~`~12
13 `~~````~~``~```~~~`~```~`~~~~~~~~~~`~~``~~~~~`````~
`~`~``~~~~13
14 `~~`~`~````~```~`~`~```~~`~~~~`~```~``~``~``~~~````~~
``````~14
  012345678901234567890123456789012345678901234567890123456789
           1       2       3       4       5
You have 16 sonar devices left. 3 treasure chests remaining.
Where do you want to drop the next sonar device? (0-59 0-14) (or type quit)
10 10
           1       2       3       4       5
```

```
   0123456789012345678901234567890123456789012345678901234567890123456789
 0 `~~~~`~~~~`~`~~~`~~~~~~~`~``~~~~~`~`~~~~`~``~``~~````~`~
```~`~~~~~`` 0
 1~`~~~~```~~~~~`~`~~`~``~`~~~```~`~`~~~`~`~~~~~~~`~`````~
`~~`~~~~~` 1
 2 `~``~``~~~~`~``~`~`~``~`````~~~~~~~~~~~~`~`~~`~``~~~~~~
```~~`~``` 2
 3 ``~`~~``~`~``~`~`~`~~`~`~~`~`~``~~~~`~``~````~
``````~~~~~``~`` 3
 4 ``~~~`~~~``~``~~````~`~`~`~``~~~~``~~~```~`~~`~~`~`~`~~
`~~~~~``` 4
 5~~```~~~~`~`~~``~`~``~```~`~~`~~~~~~`~~``~`~`~~~~`~~
`~`~`~`~~~~` 5
 6 ``~~`````~~~~~`~`~~~~```~~~~~`~~`~~`~~```~~`~~~~`~~~
``~`~~~~``~~~6
 7 `~`````````~```~``~``~~`~~~~`~~``~``~~~~```~`~~`~``~
``~~```~~7
 8 `~````~```~`~~`~~~~`~~~``~~~~``~`~``~~~~``~`~`````~`~~
```~`~~~~` 8
 9~```~~`~`~``~``~~~``~``~```~`~``~~~~~`~`~`~~~`~`~`~
`~~~``~~~``` 9
10 ```~`~````~5`~``~`~~`~``~````~``~~~`~~`~~``~~~~`~~~`~
`~~````~10
11 ```~```~~~`~```~~`~~~`~`````~`~~`~`~~`~~`~`~~`~~~~
````~````~` 11
12~~~`~`~~~~``~~~~~~~`~~~``~`~`~~`~`~~`~```~~~```~~
`~~`~``~~``~`~10
13 `~~````~~``~```~~~`~```~`~~~~~~~~~~~`~~``~~~~~`````~
`~`~``~~~~~13
14 `~~`~`~````~```~`~`~```~~`~~~~`~```~``~``~``~~~````~~
``````~14
   0123456789012345678901234567890123456789012345678901234567890123456789
             1         2         3         4         5
Treasure detected at a distance of 5 from the sonar device.
You have 15 sonar devices left. 3 treasure chests remaining.
Where do you want to drop the next sonar device? (0-59 0-14) (or type quit)
15 6
             1         2         3         4         5
   0123456789012345678901234567890123456789012345678901234567890123456789
 0 `~~~~`~~~~`~`~~~`~~~~~~~`~``~~~~~`~`~~~~`~``~``~~````~`~
```~`~~~~~`` 0
 1~`~~~~```~~~~~`~`~~`~``~`~~~```~`~`~~~`~`~~~~~~~`~`````~
`~~`~~~~~` 1
 2 `~``~``~~~~`~``~`~`~``~`````~~~~~~~~~~~~`~`~~`~``~~~~~~
```~~`~``` 2
```

```
 3 ``~`~~``~`~``~`~`~`~~`~`~~`~`~``~~~`~``~````~
``````~~~~``~`` 3
 4 ``~~`~~~``~``~~````~`~`~`~``~~~``~~```~`~~`~~`~`~`~~
`~~~~``` 4
 5~~```~~~`~`~~``~`~``~```~`~~`~~~~~`~~``~`~`~~~`~~
`~`~`~`~~~` 5
 6 ``~~`````~~~~~`~4~~~```~~~~`~~`~~`~~```~~`~~~`~~~
``~`~~~``~~~6
 7 `~``````````~```~``~``~~`~~~~`~~``~``~~~```~`~~`~``~
``~~```~~7
 8 `~````~```~`~~`~~~`~~``~~~``~`~``~~~``~`~`````~`~~
```~`~~~~~` 8
 9~```~~`~`~``~``~~``~``~```~`~``~~~~`~`~`~~~`~`~`~
`~~~``~~``` 9
10 ```~`~```~5`~``~`~~`~``~```~``~~~`~~`~~``~~~~`~~~`~
`~~````~10
11 ```~```~~~`~```~~`~~~`~`````~`~~`~`~~`~~`~`~~`~~~
````~````~` 11
12~~~`~`~~~``~~~~~~~`~~~``~`~`~~`~`~~`~```~~~```~~
`~~`~``~``~`~12
13 `~~````~~``~```~~~`~```~`~~~~~~~~~~`~~``~~~~~`````~
`~`~``~~~~13
14 `~~`~`~````~```~`~`~```~~`~~~~`~```~``~``~``~~~````~~
``````~14
   012345678901234567890123456789012345678901234567890123456789
             1         2         3         4         5
Treasure detected at a distance of 4 from the sonar device.
You have 14 sonar devices left. 3 treasure chests remaining.
Where do you want to drop the next sonar device? (0-59 0-14) (or type quit)
15 10
             1         2         3         4         5
   012345678901234567890123456789012345678901234567890123456789
 0 `~~~`~~~`~`~~`~~~~~`~``~~~~`~`~~~`~``~``~~````~`~
```~`~~~~~`` 0
 1~`~~~```~~~~`~`~~`~``~`~~```~`~`~~`~`~~~~~~`~`````~
`~~`~~~~` 1
 2 `~``~``~~~`~``~`~`~``~````~~~~~~~~~~`~`~~`~``~~~~~
```~~`~``` 2
 3 ``~`~~``~`~``~`~`~`~~`~`~~`~`~``~~~`~``~````~
``````~~~~``~`` 3
 4 ``~~`~~~``~``~~````~`~`~`~``~~~``~~```~`~~`~~`~`~`~~
`~~~~``` 4
 5~~```~~~`~`~~``~`~``~```~`~~`~~~~~`~~``~`~`~~~`~~
`~`~`~`~~~` 5
 6 ``~~`````~~~~~`~0~~~```~~~~`~~`~~`~~```~~`~~~`~~~
``~`~~~``~~~6
```
```

```
 7 `~`````````~```~``~``~~`~~~~~`~~``~``~~~```~`~~`~``~
``~~```~~7
 8 `~````~```~`~~`~~~`~~``~~~``~`~``~~~``~`~`````~`~~
```~`~~~~~` 8
 9~```~~`~`~``~~``~~``~```~`~``~~~~`~`~`~~~`~`~`~
`~~~``~~``` 9
10 ```~`~```~0`~``0`~~`~``~````~``~~~`~~`~~``~~~~`~~~`~
`~~````~10
11 ```~```~~~`~```~~`~~~`~`````~`~~`~`~~`~~`~`~~`~~~
````~````~` 11
12~~~`~`~~~``~~~~~~~`~~~``~`~`~~`~`~~`~```~~~```~~
`~~`~``~``~`~10
13 `~~````~~``~```~~~`~```~`~~~~~~~~~~`~~``~~~~~`````~
`~`~``~~~~~13
14 `~~`~`~````~```~`~`~```~~`~~~~`~```~``~``~``~~~````~~
``````~14
  0123456789012345678901234567890123456789012345678901234567890123456789
           1         2         3         4         5
You have found a sunken treasure chest!
You have 13 sonar devices left. 2 treasure chests remaining.
Where do you want to drop the next sonar device? (0-59 0-14) (or type quit)

...skipped over for brevity...

           1         2         3         4         5
  0123456789012345678901234567890123456789012345678901234567890123456789
 0 `~~~`~~~`~`~~`~~~~~`~``~~~~`~`~~~`~``~``~~````~`~
```~`~~~~``  0
 1~`~~~```~~~~`~`~~`~``~`~~```~0~`~~`~`~~~~~~`~`````~
`~~`~~~~` 1
 2 `~``~``~~~`~``~`~`~``~`````~~0~~~0~~`~`~~`~``~~~~~
```~~`~``  2
 3 ``~3~~``8`~``~`~`~`~~`~`~~`~`~``~~~~`~`0~````~
``````~~~~``~``  3
 4 ``~~`~~~``~``~~````~`~`~`~0`~~0``~~```~`~~`~~`~`~`~~
`~~~~``` 4
 5~~```~~~`~`~~``~`~``~```~`~~`~~~~~~`~~``~`~`~~~`~~
`~`~`~`~~~` 5
 6 ``~~`````~~~~~`~0~~~```~~~~`~~`~~`~~```~~`~~~`~~~
``0`~~~``~~~6
 7 `~`````````~```~``~``~~`~~~~`~~``~``~~~```~`~~`~```~
``~~```~~7
 8 `~````~```~`~~`~~~`~~``~~~``~`~``~~~``~`0```0`~`~~```~
`~~~~` 8
```

```
 9~```~~`~`~`~``~``~~~``~~``~```~0~``~~~~~`~`~`~~~~`~`~`~`~
`~~~``~~~``` 9
10 ```~`~```~0`~``0`~~`~``~````~``~~~`~~`~~``~~~~`~~~`~
`~~````~10
11 ```~```~~~`~```~~`~~~`~`````~`~~`~`~~`~~`~`~~`~~~
````~````~` 11
12~~~`~`~~~``~~~~~~~`~~~``~`~`~~`~`~~`~```~~~```~~
`~~`~``~``~`~10
13 `~~````~~``~```~~~`~```~`~~~~~~~~~~`~~``~~~~~`````~
`~`~``~~~~13
14 `~~`~`~````~```~`~`~```~~`~~~~`~```~``~``~``~~~````~~
``````~14
  0123456789012345678901234567890123456789012345678901234567890123456789
            1         2         3         4         5
Treasure detected at a distance of 4 from the sonar device.
We've run out of sonar devices! Now we have to turn the ship around and head
for home with treasure chests still out there! Game over.
   The remaining chests were here:
   0, 4
Do you want to play again? (yes or no)
no
```

## 14.2 Sonar Treasure Hunt 的源代码

下面是这个游戏的源代码。把它们输入到一个新的文件中，然后把文件保存为 sonar.py，然后按下 F5 键运行这个程序。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/sonar 上的在线 diff 工具，把你的代码与书中的代码进行比较。

sonar.py

```
 1.# Sonar
 2.
 3.import random
 4.import sys
 5.
 6.defdrawBoard(board):
 7.     # Draw the board data structure.
 8.
 9.     hline = '    ' # initial space for the numbers down the left side
of the board
10.    for i in range(1, 6):
11.        hline += (' ' * 9) + str(i)
12.
13.    # print the numbers across the top
```

```
 14.     print(hline)
 15.     print('   ' + ('0123456789' * 6))
 16.     print()
 17.
 18.     # print each of the 15 rows
 19.     for i in range(15):
 20.         # single-digit numbers need to be padded with an extra space
 21.         if i< 10:
 22.            extraSpace = ' '
 23.         else:
 24.            extraSpace = ''
 25.        print('%s%s %s %s' % (extraSpace, i, getRow(board, i), i))
 26.
 27.     # print the numbers across the bottom
 28.     print()
 29.     print('   ' + ('0123456789' * 6))
 30.     print(hline)
 31.
 32.
 33.def getRow(board, row):
 34.     # Return a string from the board data structure at a certain row.
 35.     boardRow = ''
 36.     for i in range(60):
 37.         boardRow += board[i][row]
 38.     returnboardRow
 39.
 40.def getNewBoard():
 41.     # Create a new 60x15 board data structure.
 42.     board = []
 43.     for x in range(60): # the main list is a list of 60 lists
 44.         board.append([])
 45.         for y in range(15): # each list in the main list has 15 single-
character strings
 46.             # use different characters for the ocean to make it more readable.
 47.             ifrandom.randint(0, 1) == 0:
 48.                   board[x].append('～')
 49.             else:
 50.                   board[x].append('`')
 51.     return board
 52.
 53.def getRandomChests(numChests):
 54.     # Create a list of chest data structures (two-item lists of x, y int
coordinates)
 55.     chests = []
 56.     for i in range(numChests):
```

```
 57.         chests.append([random.randint(0, 59), random.randint(0, 14)])
 58.     return chests
 59.
 60.def isValidMove(x, y):
 61.     # Return True if the coordinates are on the board, otherwise False.
 62.     return x >= 0 and x <= 59 and y >= 0 and y <= 14
 63.
 64.defmakeMove(board, chests, x, y):
 65.     # Change the board data structure with a sonar device character.
Remove treasure chests
 66.     # from the chests list as they are found. Return False if this is an
invalid move.
 67.     # Otherwise, return the string of the result of this move.
 68.     if not isValidMove(x, y):
 69.         return False
 70.
 71.     smallestDistance = 100 # any chest will be closer than 100.
 72.     for cx, cy in chests:
 73.         if abs(cx - x) > abs(cy - y):
 74.             distance = abs(cx - x)
 75.         else:
 76.             distance = abs(cy - y)
 77.
 78.         if distance <smallestDistance: # we want the closest treasure
chest.
 79.             smallestDistance = distance
 80.
 81.     if smallestDistance == 0:
 82.         # xy is directly on a treasure chest!
 83.         chests.remove([x, y])
 84.         return 'You have found a sunken treasure chest!'
 85.     else:
 86.         if smallestDistance< 10:
 87.             board[x][y] = str(smallestDistance)
 88.             return 'Treasure detected at a distance of %s from the sonar
device.' % (smallestDistance)
 89.         else:
 90.             board[x][y] = 'O'
 91.             return 'Sonar did not detect anything. All treasure chests
out of range.'
 92.
 93.
 94. def enterPlayerMove():
 95.     # Let the player type in their move. Return a two-item list of int
xy coordinates.
```

```
    print('Where do you want to drop the next sonar device? (0-59 0-14)
(or type quit)')
    while True:
        move = input()
        if move.lower() == 'quit':
            print('Thanks for playing!')
            sys.exit()

        move = move.split()
        if len(move) == 2 and move[0].isdigit() and move[1].isdigit() and
isValidMove(int(move[0]), int(move[1])):
            return [int(move[0]), int(move[1])]
        print('Enter a number from 0 to 59, a space, then a number from
0 to 14.')


def playAgain():
    # This function returns True if the player wants to play again,
otherwise it returns False.
    print('Do you want to play again? (yes or no)')
    return input().lower().startswith('y')


def showInstructions():
    print('''Instructions:
You are the captain of the Simon, a treasure-hunting ship. Your current
mission
is to find the three sunken treasure chests that are lurking in the part
of the
ocean you are in and collect them.

To play, enter the coordinates of the point in the ocean you wish to drop
a
sonar device. The sonar can find out how far away the closest chest is to
it.
For example, the d below marks where the device was dropped, and the 2's
represent distances of 2 away from the device. The 4's represent
distances of 4 away from the device.

    444444444
    4       4
    4 22222 4
    4 2   2 4
    4 2 d 2 4
```

```
    4 2   2 4
    4 22222 4
    4       4
    444444444
Press enter to continue...''')
    input()

    print('''For example, here is a treasure chest (the c) located a
distance of 2 away
from the sonar device (the d):

    22222
    c   2
    2 d 2
    2   2
    22222

The point where the device was dropped will be marked with a 2.

The treasure chests don't move around. Sonar devices can detect treasure
chests up to a distance of 9. If all chests are out of range, the point
will be marked with O

If a device is directly dropped on a treasure chest, you have discovered
the location of the chest, and it will be collected. The sonar device will
 remain there.

When you collect a chest, all sonar devices will update to locate the next
 closest sunken treasure chest.
Press enter to continue...''')
    input()
    print()


print('S O N A R !')
print()
print('Would you like to view the instructions? (yes/no)')
if input().lower().startswith('y'):
    showInstructions()

while True:
    # game setup
    sonarDevices = 16
```

```
174.     theBoard = getNewBoard()
175.     theChests = getRandomChests(3)
176.     drawBoard(theBoard)
177.     previousMoves = []
178.
179.     whilesonarDevices> 0:
180.         # Start of a turn:
181.
182.         # show sonar device/chest status
183.         ifsonarDevices> 1: extraSsonar = 's'
184.         else: extraSsonar = ''
185.         if len(theChests) > 1: extraSchest = 's'
186.         else: extraSchest = ''
187.          print('You have %s sonar device%s left. %s treasure chest%s
remaining.' % (sonarDevices, extraSsonar, len(theChests), extraSchest))
188.
189.         x, y = enterPlayerMove()
190.         previousMoves.append([x, y]) # we must track all moves so that
sonar devices can be updated.
191.
192.         moveResult = makeMove(theBoard, theChests, x, y)
193.         if moveResult == False:
194.             continue
195.         else:
196.             if moveResult == 'You have found a sunken treasure chest!':
197.                 # update all the sonar devices currently on the map.
198.                 for x, y in previousMoves:
199.                     makeMove(theBoard, theChests, x, y)
200.             drawBoard(theBoard)
201.             print(moveResult)
202.
203.         if len(theChests) == 0:
204.             print('You have found all the sunken treasure chests!
Congratulations and good game!')
205.             break
206.
207.         sonarDevices -= 1
208.
209.     ifsonarDevices == 0:
210.         print('We\'ve run out of sonar devices! Now we have to turn the
ship around and head')
211.         print('for home with treasure chests still out there! Game over.')
212.         print('    The remaining chests were here:')
213.         for x, y in theChests:
214.            print('    %s, %s' % (x, y))
```

```
215.
216.    if not playAgain():
217.        sys.exit()
```

## 14.3 设计程序

在试图理解这些源代码之前，先玩几次这个游戏，以便了解它是怎么玩的。Sonar 游戏使用列表的列表和其他诸如此类的、复杂的变量，也就是所谓的数据结构（data structure）。数据结构是存储表示某些事物的值的排列组合。例如，在第 11 章中，Tic Tac Toe 游戏板的数据结构是一个字符串列表。用字符串来表示 X、O 或空的格子，用列表中的字符串的索引来表示游戏板上的格子。在 Sonar 游戏中，针对藏宝箱与声纳设备的位置也使用类似的数据结构。

### 14.3.1 代码如何工作

```
 1. # Sonar
 2.
 3. import random
 4. import sys
```

第 3 行和第 4 行导入了 random 和 sys 模块。sys 模块包含了 exit()函数，它会导致程序立刻结束。在这个程序中，稍后会用到这个函数。

### 14.3.2 绘制游戏板

```
 6. def drawBoard(board):
```

Sonar 游戏的游戏板是周围由 X 轴和 Y 轴坐标包围着的一个 ASCII 字符图“海洋”。使用键盘上 1 键旁边的重音符（`）和波浪号（～）来表示海洋的波浪。如下所示：

```
    1         2         3         4         5
  0123456789012345678901234567890123456789012345678901234567890123456789
 0~~~`~``~~~~``~~~~~``~`~`~`~`~~`~~~`~~~`~~`~`````~~`~``~~`~~```~`~` 0
 1 `~`~````~~``~`~```~```~```~`~~~``~~~`~~~`````~`~``~~~~``~~`~~~` 1
 2 ```~~~~~`~`~~```~~~~``~````~~`~`~~~`~`~`~```~~`~``~~~`~`~~~~~~`~2
 3~~~~`~~~~``~```~``~~`~`~~`~`~~``~````~`~````~```~`~`~`~~`~~~~~3
 4~```~~~~~~`~~````~~~~```~~~`~`~`~````~`~~`~`~~``~~~`~
```

```
``~`~``~~4
 5 `~```~`~`~~`~~~```~~``~``````~~``~`~`~~~~`~~
``~~~~~~`~```~~` 5
 6 ``~~`~~`~``~`````~````~~``~`~~~~`~~```~~~``~`~`~~
``~~~```~~~6
 7 ``~``~~~~~~~```~`~```~~~``~`~``~`~~~~~~~``````~~~`~~
`~~`~~`~~7
 8~~`~`~~```~``~~``~~~~``~~`~`~~`~`~```~```~~~
```~~~~~~`~`~~~~` 8
 9 ```~``~`~~~`~~```~``~``~~~```~````~```~`~~`~~~~~`~
``~~~~~```  9
10 `~~~~```~`~````~`~`~~``~`~~~~`~``~``~```~~``````~`~
``~`````  10
11~~`~`~~`~``~`~~~`````````````~~`````~`~~``~`~~~~
`~~~`~~`~11
12~~`~~~~```~~~`````~~``~`~`~~`````~`~~``~``````~~
``~~~`~~`~12
13 `~``````~~``~`~~~```~~~~```~~`~`~~~`~``````~~`~```~
``~`~~~~13
14~~~``~```~`````~~`~`~``~~`~``~`~~`~`~``~`~``~~``~`~``~
```~~~14
  0123456789012345678901234567890123456789012345678901234567890123456789
            1         2         3         4         5
```

drawBoard()函数的绘制有 4 个步骤：

- 第 1 步，创建一个字符串变量，旁边用 1、2、3、4 和 5 标记，数字中间用较宽的空格隔开（用来表示 X 轴上的坐标 10、20、30、40 和 50）。
- 第 2 步，使用这个字符串在屏幕顶部显示 X 轴坐标。
- 第 3 步，打印出表示海洋的每一行，并且在屏幕的两边都打印出 Y 轴坐标。
- 第 4 步，在底部再次打印 X 轴。在所有四个边上都给出坐标，这样会更容易查看在哪里放置了声纳设备。

### 14.3.3　在顶部绘制 X 轴

```
 7.     # Draw the board data structure.
 8.
 9.     hline = '    ' # initial space for the numbers down the left side of
the board
10.     for i in range(1, 6):
11.         hline += (' ' * 9) + str(i)
```

再看一下图 14-3 所示的游戏板的顶部。它使用加号（+），而不是空格，所以我们可以很容易地数出空格的数目：

```
+++++++++++++1+++++++++2+++++++++3 # first line
+++0123456789012345678901234567890123456789 # second line

+0 ~~~`~`~`~~~`~`~~~~`~`~`~`~`~`~~`~~~~`~`~`~` 0 # third line
```

图 14-3 用来打印游戏板顶部的空格

第 1 行中表示 10 个坐标刻度的所有数字之间，都有 9 个空格，在 1 的前面有 13 个空格。第 9 行到第 11 行代码创建了表示这一行内容的字符串，并且把它保存到名为 hline 的变量中。

```
13.     # print the numbers across the top
14.     print(hline)
15.     print('   ' + ('0123456789' * 6))
16.     print()
```

要把这些数字打印到游戏板的顶部，首先打印出 hline 变量的内容。然后在下一行中，打印 3 个空格（使得该行能够正确地对齐），然后打印字符串'0123456789012345678901234567890123456789012345678901234567890123456789'。但是可以使用快捷方式('0123456789' * 6)来生成字符串，这会得到相同的结果。

## 14.3.4 绘制表示海洋的行

```
18.     # print each of the 15 rows
19.     for i in range(15):
20.         # single-digit numbers need to be padded with an extra space
21.         if i< 10:
22.             extraSpace = ' '
23.         else:
24.             extraSpace = ''
25.         print('%s%s %s %s' % (extraSpace, i, getRow(board, i), i))
```

第 19 行到 25 行打印了表示海洋波浪的每一行，包括向下一直增加的、表示 Y 轴坐标的数字。for 循环打印 0 到 14 行，以及游戏板的两边的行号。

这里有一个小问题。当打印时，一位数（像 0、1 和 2 等）的数字只占一个位置，而两位数的数字（像 10、11 和 12 等）要占用两个位置。如果坐标轴的大小不同，行就无法对齐。将会如下所示：

```
8~~`~`~~```~``~~``~~~~``~~`~`~~~`~`~```~```~~~```~~~~~~~
`~`~~~~` 8
9 ```~``~`~~~~`~~```~``~``~~~~```~````~```~`~~`~~~~~~`~
``~~~~~``` 9
10 `~~~~~```~`~````~`~`~~``~`~~~~`~``~``~```~~``````~`~
``~````` 10
11~~`~`~~`~``~`~~~````````````````~~`````~`~~``~`~~~~
`~~~`~~`~11
```

解决方法很简单。在所有一位数前加一个空格。第 21 行到 24 行的代码把变量 extraSpace 设置为 1 个空格或 1 个空字符串。总是会打印出变量 extraSpace，但是只有当行号是一位数时，它才是 1 个空格字符。否则，它只是 1 个空字符串。使用这种方法，当打印所有行时，它们就会对齐。

getRow()函数接受行数，并且返回一个字符串表示该行的海洋波浪。该函数的两个参数分别是，存储在 board 变量中的游戏板数据结构和行数。下面我们来看看这个函数。

### 14.3.5　在底部绘制 X 轴坐标

```
27.     # print the numbers across the bottom
28.     print()
29.     print('   ' + ('0123456789' * 6))
30.     print(hline)
```

第 27 行到 30 行和第 13 行到 16 行类似。它们在屏幕底部打印 X 轴坐标。

### 14.3.6　获取海洋中一行的状态

```
33. def getRow(board, row):
34.     # Return a string from the board data structure at a certain row.
35.     boardRow = ''
36.     for i in range(60):
37.         boardRow += board[i][row]
38.     returnboardRow
```

参数 board 是表示整个海洋波浪的一个数据结构，getRow()函数为单个的一行创建了一个字符串。

首先将变量 boardRow 设置为空字符串。把 Y 轴坐标作为 row 参数来传递。这个字符串是通过把 board[0][row]、board[1][row]、board[2][row]……board[59][row]连接起来而构成的。这是因为，该行包含从索引 0 到索引 59 的 60 个字符。

第 36 行的 for 循环遍历了整数 0 到 59。在每次迭代中，会把游戏板数据结构中的下一个字符复制到 boardRow 的末尾。等到循环完成，boardRow 就拥有整个一行的 ASCII 字符图波浪，并将其返回。

### 14.3.7 创建一个新的游戏板

```
40. def getNewBoard():
41.     # Create a new 60x15 board data structure.
42.     board = []
43.     for x in range(60): # the main list is a list of 60 lists
44.         board.append([])
```

在每一次开始新的游戏的时候，都需要一个新的 board 数据结构。board 数据结构是包含字符串列表的一个列表。第 1 个列表表示 X 坐标。由于游戏板拥有 60 个字符的宽度，所以第 1 个列表需要包含 60 个列表。创建一个 for 循环，它将给 board 附加 60 个空的列表。

```
45.             for y in range(15): # each list in the main list has 15
single-character strings
46.             # use different characters for the ocean to make it more readable.
47.             if random.randint(0, 1) == 0:
48.                 board[x].append('~')
49.             else:
50.                 board[x].append('`')
```

但是 board 远不止是 60 个空的列表的一个列表。60 个列表中的每一个列表，都表示这个游戏板上的一个 X 坐标。在游戏板上有 15 个行，所以 60 个列表中的每一个列表中，都必须有 15 个字符。第 45 行是另一个 for 循环，它添加了 15 个单个字符串以表示海洋。

“海洋”是一串随机选取'~'和'`'组合成的字符串。如果 random.randint()返回的值是 0，添加字符串'~'；否则，添加字符串'`'。这使得海洋看上去有一种随机的、波涛汹涌的感觉。

记住，board 变量是包含 60 个列表的一个列表，60 个列表中的每个列表拥有 15 个字符串。这意味着，要获取坐标（26, 12）的字符串，需要访问 board[26][12]，而不是 board[12][26]。先是 X 坐标，然后是 Y 坐标。

```
51.     return board
```

最后，这个函数返回 board 变量中的值。

### 14.3.8　创建随机的藏宝箱

```
 53. def getRandomChests(numChests):
 54.     # Create a list of chest data structures (two-item lists of x, y int
coordinates)
 55.     chests = []
 56.     for i in range(numChests):
 57.         chests.append([random.randint(0, 59), random.randint(0, 14)])
 58.     return chests
```

游戏也随机决定将藏宝箱藏在哪儿。用包含两个整数的列表的一个列表来表示藏宝箱。这两个整数将是一个箱子的 X 坐标和 Y 坐标。

例如，如果箱子的数据结构是[[2, 2], [2, 4], [10, 0]]，那这就表示有 3 个藏宝箱，一个箱子在 (2,2)，另一个箱子在(2,4)，第三个箱子在(10,0)。

numChests 参数告诉函数要生成多少个藏宝箱。第 56 行的 for 循环将会迭代 numChests 次，在每次迭代中，第 57 行添加包含两个随机整数的一个列表。X 坐标可以是 0 到 59 之间的任意数，Y 坐标可以是 0 到 14 之间的任意数。作为参数传递给 append 方法的表达式[random.randint(0, 59), random.randint(0, 14)]，将会得到类似于[2, 2]、[2, 4]或[10, 0]这样的一个列表值。把这个列表值添加到 chests 中。

### 14.3.9　判断一次移动是否有效

```
 60. def isValidMove(x, y):
 61.     # Return True if the coordinates are on the board, otherwise False.
 62.     return x >= 0 and x <= 59 and y >= 0 and y <= 14
```

当玩家输入想要放置声纳设备的 X 坐标和 Y 坐标时，他们输入的可能不是有效的值。X 坐标必须在 0 到 59 之间，Y 坐标必须在 0 到 14 之间。

isValidMove()函数使用一个简单的表达式，并用 and 操作符来确保该条件的每个部分都为 True。如果有一个部分是 False，那么整个表达式的结果都为 False。该函数返回最终的布尔值。

### 14.3.10　在游戏板上进行一次移动

```
 64.def makeMove(board, chests, x, y):
 65.     # Change the board data structure with a sonar device character.
Remove treasure chests
 66.     # from the chests list as they are found. Return False if this is an
invalid move.
```

```
67.     # Otherwise, return the string of the result of this move.
68.     if not isValidMove(x, y):
69.         return False
```

在 Sonar 游戏中，通过更新游戏板，针对投下的每个声纳设备显示一个数字，表示它距离最近的藏宝箱有多远。所以，当玩家通过给程序一个 X 坐标和 Y 坐标来进行一次移动的时候，游戏板会根据藏宝箱的位置做出修改。

makeMove()函数接受 4 个参数：游戏板数据结构、藏宝箱数据结构、X 坐标和 Y 坐标。如果传递的 X 坐标和 Y 坐标不在游戏板中，第 69 行将返回 False。如果 isValidMove()函数返回 False，那么 makeMove()函数也将返回 False。

否则，makeMove()将返回一个字符串，描述如何响应移动：

- 如果该坐标直接位于藏宝箱之上，makeMove()返回'You have found a sunken treasure chest!'。
- 如果该坐标距离藏宝箱的距离在 9 以内，makeMove()返回'Treasure detected at a distance of %s from the sonar device.'（其中%s 会用整数的距离值来替换）。
- 否则，makeMove()将返回'Sonar did not detect anything. All treasure chests out of range.'。

```
71.     smallestDistance = 100 # any chest will be closer than 100.
72.     for cx, cy in chests:
73.         if abs(cx - x) > abs(cy - y):
74.             distance = abs(cx - x)
75.         else:
76.             distance = abs(cy - y)
77.
78.         if distance <smallestDistance: # we want the closest treasure chest.
79.             smallestDistance = distance
```

给定了玩家想要投下声纳设备的坐标，以及包含藏宝箱的 XY 坐标的一个列表，我们需要一种算法来找到最近的藏宝箱。

## 14.4 找到最近的藏宝箱的算法

x 参数和 y 参数是整数（假设是 3 和 2），它们一起表示玩家在游戏板上猜测的位置。变量 chests 将拥有诸如[[5, 0], [0, 2], [4, 2]]这样的值。这些值表示 3 个藏宝箱的位置。我们可以把它表示为图 14-4 所示。对于位于(3,2)的声纳设备，其周围不同距离所构成的方形环区如图 14-5 所示。

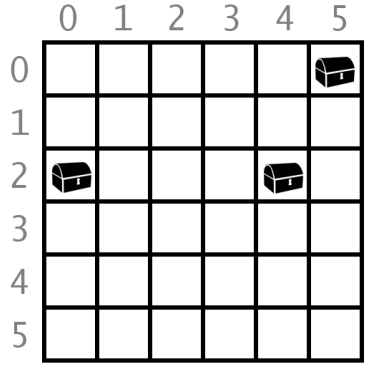



图 14-4　[[5, 0], [0, 2], [4, 2]]所表示的藏宝箱

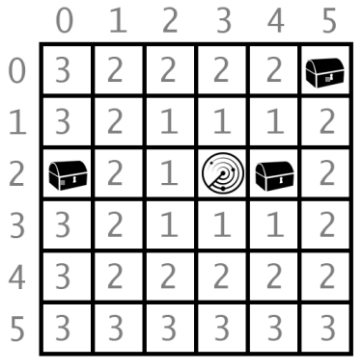



图 14-5　游戏板上标记出了到位置(3,2)的距离

但是，如何把它转换为游戏中的代码呢？我们需要一种方法把方形环区的距离表示为一个表达式。注意，到一个 XY 坐标的距离总是两个 X 坐标的差的绝对值和两个 Y 坐标的差的绝对值这两个值中更大的那个。

这意味着，我们应该用声纳设备的 X 坐标减去藏宝箱的 X 坐标，然后取这个值的绝对值。以同样方式来处理声纳设备的 Y 坐标和藏宝箱的 Y 坐标。两个值中较大的值就是声纳设备和藏宝箱之间的距离。

例如，假设声纳的 X 坐标和 Y 坐标是 3 和 2，如图 14-4 所示。第 1 个藏宝箱（也就是列表[[5, 0], [0, 2], [4, 2]]中的第 1 项）的 X 坐标和 Y 坐标是 5 和 0。

1．对于 X 坐标，3 - 5 的结果是-2，-2 的绝对值是 2。

2．对于 Y 坐标，2 - 1 的结果是 1，1 的绝对值是 1。

3．比较两个绝对值 2 和 1，较大的值是 2，所以 2 应该是声纳设备和位于坐标(5,0)的藏宝箱之间的距离。

我们可以看一下图 14-4 中的游戏板，看看这个算法是如何工作的，因为位于(5,0)的藏宝箱在声纳设备的第 2 个方形环区中。让我们快速比较一下另外两个藏宝箱，也判断一下距离的计算是否正确。

我们来求位于（3,2）的声纳设备到位于（0, 2）的藏宝箱之间的距离：

1．abs(3 - 0)的结果是 3。

2．abs(2 - 2)的结果是 0。

3．3 比 0 大，所以从位于（3, 2）的声纳设备到位于（0, 2）的藏宝箱的距离是 3。

我们来求位于（3,2）的声纳设备到位于（4, 2）的藏宝箱之间的距离：

1．abs(3 - 4)的结果是 1。

2．abs(2 - 2)的结果是 0。

3．1 比 0 大，所以之间的距离是 1。

看一下图 14-4，我们可以知道所有这 3 个距离都是正确的。这个算法是有效的。从声纳

设备到 3 个沉没的藏宝箱之间的距离分别是 2、3 和 1。在每一次猜测中，我们想要知道从声纳设备到最近的 3 个藏宝箱之间的距离。为了做到这一点，使用一个叫做 smallestDistance 的变量。我们再来看一下代码：

```
 71.     smallestDistance = 100 # any chest will be closer than 100.
 72.     for cx, cy in chests:
 73.         if abs(cx - x) > abs(cy - y):
 74.             distance = abs(cx - x)
 75.         else:
 76.             distance = abs(cy - y)
 77.
 78.     if distance <smallestDistance: # we want the closest treasure
chest.
 79.             smallestDistance = distance
```

第 72 行，在一个 for 循环中，使用了多变量赋值的技巧。例如，赋值语句 spam, eggs = [5, 10]将把 5 赋值给 spam，把 10 赋值给 eggs。

因为 chests 是一个列表，列表中的每个元素本身又是包含两个整数的列表，所以第 1 个整数赋值给 cx，第 2 个整数赋值给 cy。所以，如果 chests 的值是[[5, 0], [0, 2], [4, 2]]，那么在循环的第一次迭代中，cx 的值是 5，cy 的值是 0。

第 73 行判断哪个距离较大：是 X 坐标的差的绝对值，还是 Y 坐标的差的绝对值。abs(cx - x) > abs(cy - y)似乎是一种更简短的表示方式，难道不是吗？第 73 行到 76 行，把较大的值赋给了变量 distance。

所以，在 for 循环中的每次迭代中，distance 变量都会存储从这个声纳设备到藏宝箱之间的距离。但是，我们想要得到从这个声纳设备到所有藏宝箱之间最短的距离。这就是变量 smallestDistance 的用武之地。当 distance 变量小于 smallestDistance 时，distance 中的值会成为 smallestDistance 的新值。

在循环开始时，给 smallestDistance 一个不可能达到的、较高的值 100，以便至少可以找到一个合适的藏宝箱距离能够放入到 smallestDistance 中。当完成了 for 循环，我们就知道 smallestDistance 中存储了游戏中的声纳设备到所有藏宝箱之间最短的距离。

## 14.5 列表方法 remove()

列表方法 remove()将移除列表中与传入的参数相匹配的第一个值。例如，尝试在交互式 shell 中输入如下代码：

```
>>> x = [42, 5, 10, 42, 15, 42]
>>>x.remove(10)
>>>x
```

```
[42, 5, 42, 15, 42]
```

从 x 列表中移除了 10。remove()方法移除了列表中与传入的参数相匹配的第 1 个值，并且只移除第 1 个值。例如，在交互式 shell 中输入：

```
>>> x = [42, 5, 10, 42, 15, 42]
>>>x.remove(42)
>>>x
[5, 10, 42, 15, 42]
```

注意，只有第 1 个 42 从列表中移除了，而第 2 个 42 和第 3 个 42 仍然还在列表中。如果试图移除列表中不存在的一个值，remove()方法将产生一个错误：

```
>>> x = [5, 42]
>>>x.remove(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

```
 81.     if smallestDistance == 0:
 82.         # xy is directly on a treasure chest!
 83.         chests.remove([x, y])
 84.         return 'You have found a sunken treasure chest!'
```

唯一让 smallestDistance 等于 0 的情况，就是当声纳设备的 XY 坐标和藏宝箱的 XY 坐标相同的时候。这意味着玩家猜对了藏宝箱的位置。使用列表方法 remove()，从 chests 数据结构中删除包含了该藏宝箱的坐标的两个整数的列表。然后，该函数将返回'You have found a sunken treasure chest!'。

```
 85.     else:
 86.          if smallestDistance< 10:
 87.              board[x][y] = str(smallestDistance)
 88.              return 'Treasure detected at a distance of %s from the sonar
device.' % (smallestDistance)
 89.         else:
 90.               board[x][y] = '0'
 91.               return 'Sonar did not detect anything. All treasure chests out
of range.'
```

如果 smallestDistance 不等于 0，执行从第 86 行开始的 else 语句块，这意味着玩家没有猜到藏宝箱的确切位置。如果声纳设备的距离小于 10，第 87 行用 smallestDistance 的字符串形式来标记游戏板上的位置。否则，把游戏板位置标记为'0'。

### 14.5.1 获取玩家的移动

```
 94.def enterPlayerMove():
 95.     # Let the player type in their move. Return a two-item list of int
xy coordinates.
 96.     print('Where do you want to drop the next sonar device? (0-59 0-14)
(or type quit)')
 97.     while True:
 98.         move = input()
 99.         if move.lower() == 'quit':
100.             print('Thanks for playing!')
101.             sys.exit()
```

enterPlayerMove()函数收集玩家下一步移动的 XY 坐标。while 循环将一直询问玩家下一步移动，直到他们输入一个有效的移动。玩家也可以输入'quit'来退出游戏。在这种情况下，第 101 行将调用 sys.exit()函数来立刻终止程序。

```
103.        move = move.split()
104.        if len(move) == 2 and move[0].isdigit() and move[1].isdigit() and
isValidMove(int(move[0]), int(move[1])):
105.            return [int(move[0]), int(move[1])]
106.         print('Enter a number from 0 to 59, a space, then a number from
0 to 14.')
```

如果玩家没有输入'quit'，那么代码必须确保这是一次有效的移动，也就是说，移动是用逗号隔开的两个整数。第 103 行调用 move 的 split()方法，并将其作为 move 的新值。

如果玩家输入像'1 2 3'这样的值，那么 split()返回的列表将是['1', '2', '3']。在这种情况下，表达式 len(move) == 2 将为 False，并且整个表达式的结果立刻成为 False。因为短路求值方式（第 11 章介绍过），Python 不会再检查表达式的剩余部分。

如果列表的长度是 2，那么两个值将是 move[0]和 move[1]。要检查这两个值是否为数字（像'2'或'17'这样），可以使用类似第 12 章中的 isOnlyDigits()这样的一个函数。但是，Python 已经有另一个函数来做这件事情。

如果这个字符串只包含数字，字符串方法 isdigit()返回 True；否则，返回 False。尝试在交互式 shell 中输入：

```
>>> '42'.isdigit()
True
>>> 'forty'.isdigit()
False
>>> ''.isdigit()
```

```
False
>>> 'hello'.isdigit()
False
>>> x = '10'
>>>x.isdigit()
True
```

move[0].isdigit()和 move[1].isdigit()必须都为 True，整个条件才会为 True。第 104 行的条件的最后部分，调用了 isValidMove()函数来判断 XY 坐标是否在游戏板上。

如果整个条件为 True，第 105 行返回了包含 XY 坐标的两个整数的列表。否则，执行循环，要求玩家再次输入坐标。

### 14.5.2　询问玩家是否再玩一局

```
109. def playAgain():
110.     # This function returns True if the player wants to play again,
otherwise it returns False.
111.     print('Do you want to play again? (yes or no)')
112.     return input().lower().startswith('y')
```

playAgain()函数与前几章中的 playAgain()函数类似。

### 14.5.3　为玩家打印出游戏说明

```
115.def showInstructions():
116.     print('''Instructions:
117. You are the captain of the Simon, a treasure-hunting ship. Your current
mission
118.is to find the three sunken treasure chests that are lurking in the part
of the
119. ocean you are in and collect them.
120.
121. To play, enter the coordinates of the point in the ocean you wish to drop
a
122.sonar device. The sonar can find out how far away the closest chest is
to it.
123. For example, the d below marks where the device was dropped, and the
2's
124.represent distances of 2 away from the device. The 4's represent
125.distances of 4 away from the device.
126.
127.    444444444
128.    4       4
```

```
129.     4 22222 4
130.     4 2   2 4
131.     4 2 d 2 4
132.     4 2   2 4
133.     4 22222 4
134.     4       4
135.     444444444
136. Press enter to continue...''')
137.     input()
```

showInstructions()函数调用了几次 print()函数，以打印出多行字符串。在打印下一个字符串之前，input()函数给玩家一次按下 Enter 键的机会。这是因为 IDLE 窗口一次只可以显示这么多的文本。

```
139.     print('''For example, here is a treasure chest (the c) located a
distance of 2 away
140. from the sonar device (the d):
141.
142.     22222
143.     c   2
144.     2 d 2
145.     2   2
146.     22222
147.
148. The point where the device was dropped will be marked with a 2.
149.
150. The treasure chests don't move around. Sonar devices can detect treasure
151. chests up to a distance of 9. If all chests are out of range, the point
152. will be marked with 0
153.
154. If a device is directly dropped on a treasure chest, you have discovered
155. the location of the chest, and it will be collected. The sonar device will
156. remain there.
157.
158. When you collect a chest, all sonar devices will update to locate the next
159. closest sunken treasure chest.
160. Press enter to continue...''')
161.     input()
162.     print()
```

当玩家按下 Enter 键，该函数返回。

### 14.5.4　游戏开始

```
165. print('S O N A R !')
166. print()
167. print('Would you like to view the instructions? (yes/no)')
168. if input().lower().startswith('y'):
169.       showInstructions()
```

表达式 input().lower().startswith('y')会询问玩家是否想要查看游戏说明，如果玩家输入以'y'或'Y'开头的字符串，结果为 True。如果是这样，就会调用 showInstructions()函数；否则，游戏开始。

```
171. while True:
172.       # game setup
173.       sonarDevices = 16
174.       theBoard = getNewBoard()
175.       theChests = getRandomChests(3)
176.       drawBoard(theBoard)
177.       previousMoves = []
```

第 171 的 while 循环是程序的主循环。从第 173 行到 177 行，创建了几个变量，如表 14-1 所示。

表 14-1　主游戏循环中用到的变量

| 变量 | 说明 |
| --- | --- |
| sonarDevices | 玩家还剩下的声纳设备（轮次）的数目 |
| theBoard | 这个游戏使用的游戏板数据结构 |
| theChests | 藏宝箱数据结构的列表。getRandomChests()函数将会返回一个列表，表示游戏板上随机放置的 3 个藏宝箱 |
| previousMoves | 玩家在这个游戏中所做的所有 XY 移动的列表 |

### 14.5.5　为玩家显示游戏的状态

```
179.     while sonarDevices> 0:
180.         # Start of a turn:
181.
182.         # show sonar device/chest status
183.         if sonarDevices> 1: extraSsonar = 's'
184.         else: extraSsonar = ''
```

```
185.          if len(theChests) > 1: extraSchest = 's'
186.          else: extraSchest = ''
187.          print('You have %s sonar device%s left. %s treasure chest%s
remaining.' % (sonarDevices, extraSsonar, len(theChests), extraSchest))
```

只要玩家还剩下声纳设备，第 179 行的 while 循环就会执行。第 187 行打印出一条消息，告诉玩家还剩下多少个声纳设备和藏宝箱。但是，这里有一个小问题。

如果剩下两个或多个声纳设备，我们想要打印'2 sonar devices'。但是，如果只剩下 1 个声纳设备，我们想要打印'1 sonar device'。如果有多个声纳设备，我们只需要复数形式的“devices”即可。这与'2 treasure chests'和'1 treasure chest'的情况相同。

从第 183 行到 186 行，代码放在了 if 和 else 语句的冒号之后。这在 Python 中是有效的。我们可以使用语句的同一行后面的剩余部分，而不在语句后面使用代码块，从而使得代码更为简洁。

如果还有多个声纳设备或藏宝箱，把名为 extraSsonar 和 extraSchest 的两个变量设置为's'（space）。否则，把这两个变量设置为空字符串。我们将会在第 187 行用到这些变量。

## 14.5.6 获取玩家的移动

```
189.          x, y = enterPlayerMove()
190.          previousMoves.append([x, y]) # we must track all moves so that
sonar devices can be updated.
191.          moveResult = makeMove(theBoard, theChests, x, y)
192.          if moveResult == False:
193.              continue
```

第 189 行使用了多变量赋值，因为 enterPlayerMove()函数返回了包含两个元素的列表。把返回列表中的第 1 个元素赋值给 x 变量。第 2 个元素赋值给 y 变量。

然后会把它们添加到 previousMoves 列表的末尾。这意味着 previousMoves 是记录玩家在游戏中所做出的每一步移动的 XY 坐标的一个列表。随后的 198 行中，将要使用这个列表。

变量 x、y、theBoard 和 theChests 都传递给 makeMove()函数。这个函数对游戏板做出必要的修改，从而在游戏板上放置声纳设备。

如果 makeMove()返回值 False，那么作为参数传递给它的 x 值和 y 值就有问题。continue 语句将把执行送回到第 179 行 while 循环的开始处，要求玩家再次输入 XY 坐标。

## 14.5.7 找到一个沉没的藏宝箱

```
195.          else:
196.              if moveResult == 'You have found a sunken treasure chest!':
197.                  # update all the sonar devices currently on the map.
```

```
198.                 for x, y in previousMoves:
199.                     makeMove(theBoard, theChests, x, y)
200.             drawBoard(theBoard)
201.             print(moveResult)
```

如果 makeMove()没有返回值 False，它将返回一个字符串，表示该移动的结果。如果字符串是'You have found a sunken treasure chest!'，那么游戏板上所有的声纳设备都会更新，以便检测游戏板上的下一个最近的藏宝箱。所有声纳设备的 XY 坐标都在 previousMoves 中。通过第 198 行对 previousMoves 的遍历，我们可以把所有这些 XY 坐标都传递给 makeMove()函数，以便重新绘制游戏板上的值。

因为程序没有打印任何新的东西，所以玩家不会意识到之前所有的移动都重置了。它只会显示更新后的游戏板本身。

## 14.5.8　判断玩家是否赢了

```
203.         if len(theChests) == 0:
204.             print('You have found all the sunken treasure chests!
Congratulations and good game!')
205.             break
```

记住，makeMove()函数修改了我们发送给它的 theChests 列表。因为 theChests 是列表，所以在函数中对它的任何修改，都会在执行从函数返回后持久性地存在。当发现了藏宝箱，makeMove()从 theChests 中移除了元素，所以最终所有藏宝箱都将被移除（如果玩家一直能找对的话）。记住，藏宝箱意味着在 theChests 列表中表示 XY 坐标的两个元素的列表。

当找到游戏板上的所有藏宝箱并将其从 theChests 中移除时，theChests 列表的长度将为 0。当出现这种情况，要祝贺玩家，然后执行 break 语句跳出这个 while 循环。随后，执行将移到第 209 行，也就是 while 语句块之后的第一行。

## 14.5.9　判断玩家是否输了

```
207.         sonarDevices -= 1
```

第 207 行代码是从第 179 行开始的 while 循环的最后一行。因为玩家已经用了一个声纳设备，所以变量 sonarDevices 会减 1。如果玩家一直没找到藏宝箱，最后 sonarDevices 将减少为 0。在这行代码之后，执行会跳转回第 179 行，以便可以重新判断 while 语句的条件（也就是 sonarDevices>0）。

如果 sonarDevices 是 0，那么条件将为 False，将继续执行 while 语句块之外的第 209 行代码。但在此之前，条件仍然为 True，玩家可以继续进行猜测。

```
209.     if sonarDevices == 0:
210.         print('We\'ve run out of sonar devices! Now we have to turn the
ship around and head')
211.         print('for home with treasure chests still out there! Game over.')
212.         print('    The remaining chests were here:')
213.         for x, y in theChests:
214.             print('    %s, %s' % (x, y))
```

第 209 行是 while 循环之外的第 1 行。当执行到这里，游戏就结束了。如果 sonarDevices 是 0，我们就知道玩家用尽了声纳设备也没能找到所有的藏宝箱，因此，他输掉了游戏。

第 210 行和 212 行会告诉玩家他们输了。第 213 行的 for 循环将会遍历 theChests 中剩余的藏宝箱，并且将它们的位置显示给玩家，以便玩家可以知道这些藏宝箱到底潜伏在哪里。

### 14.5.10 sys.exit()函数

```
216.      if not playAgain():
217.        sys.exit()
```

无论输赢，都会再调用 playAgain()，让玩家输入是想要再玩一局还是退出游戏。如果不再玩了，playAgain()返回 False。第 216 行的 not 操作符会把它变为 True，以使得这个 if 语句的条件为 True，并且执行 sys.exit()函数。这会导致程序终止。

否则，执行将跳转回从 171 行开始的 while 循环，新的游戏开始了。

## 14.6 本章小结

还记得 Tic Tac Toe 游戏中如何为游戏板上 1 到 9 的格子编号吗？对于小于 10 个格子的游戏板，这种坐标系统可能是适用的。但是，Sonar 游戏的游戏板有 900 个格子！我们在第 13 章中学过的笛卡尔坐标系使得所有这些格子成为可管理的，特别是当游戏需要找到游戏板上的两个点之间的距离时。

在使用笛卡尔坐标系的游戏中，可以将位置存储到包含列表的一个列表中，从而用列表的第 1 个索引表示 X 坐标，第 2 个索引表示 Y 坐标。这使得可以像 board[x][y]这样来访问一个坐标。

这些数据结构（例如，用来表示海洋和财宝的位置的数据结构），使得用复杂概念表示数据成为可能，并且游戏程序变成主要就是修改这些数据结构。

在下一章中，我们将用 ASCII 数字，把字母表示为数字（就像我们前边用过的“ASCII 字符图”）。通过用数字表示文本，我们可以对它们执行数学运算，这样就可以对秘密的消息进行加密和解密。

# 第 15 章　Caesar Cipher

**本章主要内容：**

- 密码学和密码；
- 加密和解密；
- 密文、明文、密钥和符号；
- 凯撒密码；
- ASCII 编码；
- chr()函数和 ord()函数；
- 字符串方法 isalpha()；
- 字符串方法 isupper()和 islower()；
- 密码分析学；
- 暴力破解（Brute-Force）技术。

本章的程序并不是一个真正的游戏，而是一个有趣的程序。这个程序将把常规的英语转换成一个秘密的代码。它也可以把秘密代码再转换为常规的英语。只有知道秘密代码的人，才能够理解我们的秘密消息。

因为这个程序能够把文本转换成秘密消息，所以我们将介绍处理字符串的一些新的函数和方法。我们还将介绍程序如何对文本字符串进行数学运算，就好像能够对数字所做的那样。

## 15.1　密码学

编写秘密代码的科学叫做密码学（cryptography）。几千年来，密码学用来制作只有发送者和接收者能够读懂的秘密消息，即使某人俘获了信使并且读取了编码后的消息，也无法读懂这些消息。我们把秘密代码的系统叫做密码（cipher）。本章中的程序用到的密码叫做凯撒密码（caesar cipher）。

在密码学中，我们把想要加密的消息叫做明文（plain text）。明文可以像下面这样：

```
Hello there! The keys to the house are hidden under the flower pot.
```

把明文转换成加密后的消息叫做对明文加密（encrypting）。明文加密后变成密文（cipher text）。密文看上去就像是随机的字母，只查看密文，我们无法理解最初的明文是什么。把之前的示例加密成密文，如下所示：

```
Yvccf kyviv! Kyv bvpj kf kyv yfljv riv yzuuve leuvi kyv wcfnvi gfk.
```

但是如果知道用来加密消息的密码，就可以把这些密文解密（decrypt）成明文（解密是加密的相反的操作）。

许多密码也使用密钥。密钥（key）是神秘的值，使得我们可以解密那些用特定的密码加密的密文。可以把密码想象成为一个门锁。只能用一把特定的钥匙来打开它。

如果对编写密码学程序感兴趣的话，可以阅读我的另一本书籍，《Python 密码学编程》。

## 15.2 凯撒密码

凯撒密码的密钥是 1 到 26 之间的一个数字。除非知道这个键（也就是知道用于加密消息的数字），否则无法对这个保密的代码进行解密。

凯撒密码是人类最早发明的密码之一。在本章中，我们将获取消息中的每个字母（在密码学中，这些字母叫做符号，因为它们可以是字母、数字或任何其他符号），并用一个“移位后的”字母来代替它，以实现对这条消息的加密。如果把字母 A 移动 1 格，就会得到字母 B。如果把 A 移动两格，就会得到字母 C。图 15-1 是把一些字母移动 3 格后的一张图。

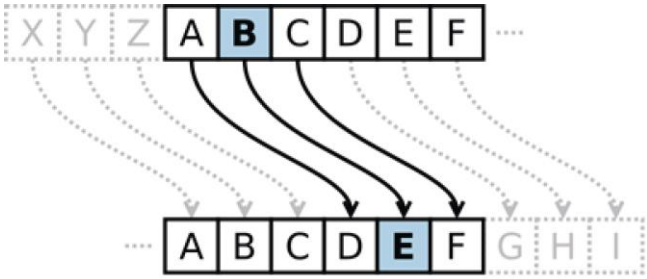



图 15-1 字母移动 3 格。这里，B 变成了 E

要得到每个移位后的字母，需要为字母表中的字母绘制一排格子。然后在它下面再绘制一排格子，但是，第 2 排的格子是从特定数目的格子（这个数字就是密钥）之后开始的。字母到达末尾之后，再折返到格子的开始处。字母移动 3 个格子的例子如下所示。

移位的格子的数目，就是凯撒密码中的密钥。上面的示例显示了密钥为 3 时的字母转换。

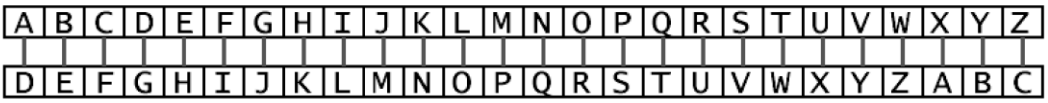



图 15-2　字母表整体移位 3 格

如果以 3 作为一个密钥来加密明文“Howdy”，则：

- 字母“H”变成了“K”；
- 字母“o”变成了“r”；
- 字母“w”变成了“z”；
- 字母“d”变成了“g”；
- 字母“y”变成了“b”。

秘文“Howdy”用 3 作为密钥进行加密，变成了“Krzgb”。

对于任意非字母的字符，保持其不变就行了。要用 3 作为密钥来解密“Krzgb”，我们从下边格子返回到上面的格子：

- 字母“K”变成了“H”；
- 字母“r”变成了“o”；
- 字母“z”变成了“w”；
- 字母“g”变成了“d”；
- 字母“b”变成了“y”。

## 15.3　ASCII 码以及用数字表示字母

我们如何用代码来实现这种字母移位呢？可以把每个字母表示成一个数字，把这个数字叫做顺序编码（ordinal），然后增加或减少这个数字来形成一个新的顺序编码（和一个新的字母），从而实现这一点。ASCII（American Standard Code for Information Interchange，美国标准信息交换代码）是一种代码，它将每个字符和 32 到 126 之间的一个数字进行关联。

大写字母“A”到“Z”是 ASCII 数字 65 到 90。小写字母“a”到“z”是 ASCII 数字 97 到 122。数字“0”到“9”的 ACSII 数字是 48 到 57。表 15-1 展示了所有的 ASCII 字符和编码。

现代计算机使用 UTF-8，而不是 ASCII。但是 UTF-8 向后兼容 ASCII，所以 ASCII 字符的 UTF-8 编码和 ASCII 的编码是一样的。

表 15-1　ASCII 表

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | (space) | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | | |

因此，如果想要把“A”移动 3 格，需要像下面这样做：

- 把“A”转换成编码（65）；
- 65 加 3，得到 68；
- 把编码 68 转换回字母（“D”）。

## 15.4　函数 chr()和 ord()

函数 chr()（“character”的简写）接受一个整数形式的顺序编码，并且返回单个字符的字符串。ord()函数接受单个字符的字符串，并且返回整数形式的顺序编码。尝试在交互式 shell 中输入如下代码：

```
>>> chr(65)
'A'
>>> ord('A')
65
>>> chr(65+8)
'I'
>>> chr(52)
'4'
>>> chr(ord('F'))
```

```
'F'
>>> ord(chr(68))
68
```

在第 3 行中，chr(65+8)的结果是 chr(73)。如果查看 ASCII 表，可以看到 73 是大写字母“I”的编码。

在第 5 行中，chr(ord('F'))的结果是 chr(70)，它等于'F'。函数 ord()和 chr()的操作是彼此相反的。

## 15.5　凯撒密码的运行示例

下面是凯撒密码的一个运行示例，它加密一条消息：

```
Do you wish to encrypt or decrypt a message?
encrypt
Enter your message:
The sky above the port was the color of television, tuned to a dead channel.
Enter the key number (1-26)
13
Your translated text is:
Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.
```

现在运行该程序，并且解密刚才加密过的文本。

```
Do you wish to encrypt or decrypt a message?
decrypt
Enter your message:
Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.
Enter the key number (1-26)
13
Your translated text is:
The sky above the port was the color of television, tuned to a dead channel.
```

如果不能用正确的密钥来解密，那么加密的文本将成为垃圾数据：

```
Do you wish to encrypt or decrypt a message?
decrypt
Enter your message:
Gur fxl nobir gur cbeg jnf gur pbybe bs gryrivfvba, gharq gb n qrnq punaary.
Enter the key number (1-26)
15
Your translated text is:
Rfc qiw yzmtc rfc nmpr uyq rfc amjmp md rcjctgqgml, rslcb rm y bcyb afyllcj.
```

## 15.6 Caesar Cipher 的源代码

如下是凯撒密码程序的源代码。把它们输入到一个文件中，然后把文件保存为 cipher.py。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/cipher 上的在线 diff 工具，把你的代码与书中的代码进行比较。

caesar.py

```
 1.# Caesar Cipher
 2.
 3. MAX_KEY_SIZE = 26
 4.
 5. def getMode():
 6.     while True:
 7.         print('Do you wish to encrypt or decrypt a message?')
 8.         mode = input().lower()
 9.         if mode in 'encrypt e decrypt d'.split():
10.             return mode
11.         else:
12.             print('Enter either "encrypt" or "e" or "decrypt" or "d".')
13.
14. def getMessage():
15.     print('Enter your message:')
16.     return input()
17.
18. def getKey():
19.     key = 0
20.     while True:
21.         print('Enter the key number (1-%s)' % (MAX_KEY_SIZE))
22.         key = int(input())
23.         if (key >= 1 and key <= MAX_KEY_SIZE):
24.             return key
25.
26. def getTranslatedMessage(mode, message, key):
27.     if mode[0] == 'd':
28.         key = -key
29.     translated = ''
30.
31.     for symbol in message:
32.         if symbol.isalpha():
33.             num = ord(symbol)
34.             num += key
35.
```

```
36.            if symbol.isupper():
37.                if num > ord('Z'):
38.                    num -= 26
39.                elif num < ord('A'):
40.                    num += 26
41.            elif symbol.islower():
42.                if num > ord('z'):
43.                    num -= 26
44.                elif num < ord('a'):
45.                    num += 26
46.
47.            translated += chr(num)
48.        else:
49.            translated += symbol
50.    return translated
51.
52. mode = getMode()
53. message = getMessage()
54. key = getKey()
55.
56. print('Your translated text is:')
57. print(getTranslatedMessage(mode, message, key))
```

## 15.7　代码如何工作

加密和解密过程是彼此反向的过程，即便如此，它们仍然会有很多相同的代码。我们来看一下每行代码是如何工作的。

```
1. # Caesar Cipher
2.
3. MAX_KEY_SIZE = 26
```

第 1 行只是一条注释。MAX_KEY_SIZE 是一个常量，把整数 26 存储于其中。在这个程序中，MAX_KEY_SIZE 提醒我们，在密码中使用的密钥应该在 1 到 26 之间。

### 15.7.1　决定加密还是解密

```
5. def getMode():
6.    while True:
7.        print('Do you wish to encrypt or decrypt a message?')
8.        mode = input().lower()
```

```
 9.        if mode in 'encrypt e decrypt d'.split():
10.            return mode
11.        else:
12.            print('Enter either "encrypt" or "e" or "decrypt" or "d".')
```

getMode()函数让用户输入是想要程序进入加密模式还是解密模式。把 input()函数和 lower()函数返回的值存储到变量 mode 中。if 语句的条件负责判断存储在 mode 中的字符串是否存在于通过'encrypt e decrypt '.split()返回的列表中。

这个列表是['encrypt', 'e', 'decrypt', 'd']，但是输入'encrypt e decrypt d'.split()会更简单，这样就不用输入那些引号和逗号。使用对于你来说最简单的方式，这两种方式的结果都是相同的列表值。

只要 mode 等于'encrypt'、'e'、'decrypt'或'd'，这个函数就返回 mode 中的字符串。因此，getMode()将返回字符串'e'或字符串'd'（但用户可以输入'encrypt'、'e'、'decrypt'或'd'中的任何一个）。

### 15.7.2 从玩家处得到消息

```
14. def getMessage():
15.     print('Enter your message:')
16.     return input()
```

getMessage()函数直接从用户处得到要加密的消息和要解密的消息，并将其返回。

### 15.7.3 从玩家处得到密钥

```
18. def getKey():
19.     key = 0
20.     while True:
21.         print('Enter the key number (1-%s)' % (MAX_KEY_SIZE))
22.         key = int(input())
23.         if (key >= 1 and key <= MAX_KEY_SIZE):
24.             return key
```

getKey()函数让玩家输入用于加密消息和解密消息的密钥。while 循环确保函数一直循环，直到用户输入一个有效的密钥。

有效的密钥是 1 到 26 之间的一个整数（记住，MAX_KEY_SIZE 的值只会是 26，因为它是常量）。然后将返回这个密钥。第 22 行将用户输入的内容转换成整数赋值给 key，所以 getKey()返回了一个整数。

### 15.7.4　用给定的密钥来加密和解密消息

```
26. def getTranslatedMessage(mode, message, key):
27.     if mode[0] == 'd':
28.         key = -key
29.     translated = ''
```

getTranslatedMessage()函数进行加密和解密。它有 3 个参数：

- mode 设置函数是加密模式还是解密模式；
- message 是待加密的明文（或待解密的密文）；
- key 是这个密码中用到的密钥。

第 27 行判断变量 mode 中的第 1 个字母是否是字符串'd'。如果是，那么程序以解密模式工作。解密模式和加密模式的唯一区别是，在解密模式中，把密钥设置为其自身的负数。如果 key 是整数 22，那么在解密模式中，把它设置为-22。我们稍后会解释这么做的原因。

变量 translated 是得到的结果字符串：也就是密文（如果是加密的话）或明文（如果是解密的话）。它一开始是空字符串，并且把加密后的字符或解密后的字符连接到字符串的末尾。

## 15.8　字符串方法 isalpha()

如果字符串是从 A 到 Z 的一个大写或小写字母，则字符串方法 isalpha()将返回 True。如果字符串包含了非字母字符，那么 isalpha()方法将返回 False。尝试在交互式 shell 中输入如下代码：

```
>>> 'Hello'.isalpha()
True
>>> 'Forty two'.isalpha()
False
>>> 'Fortytwo'.isalpha()
True
>>> '42'.isalpha()
False
>>> ''.isalpha()
False
```

正如你所看到的，'Forty two'.isalpha()将返回 False，因为'Forty two'中间有一个空格，它是一个非字母的字符。'Fortytwo'.isalpha()返回 True，因为它没有空格。

'42'.isalpha()方法返回 False，因为'4'和'2'都是非字母字符。如果字符串只有字母字符并且不为空，则 isalpha()方法只会返回 True。

在程序接下来的几行代码中，会用到 isalpha()方法。

```
31.    for symbol in message:
32.        if symbol.isalpha():
33.            num = ord(symbol)
34.            num += key
```

第 31 行的 for 循环遍历了 message 字符串中的每一个字母（在密码学中，它们叫做符号）。在循环的每一次迭代中，symbol 将拥有 message 中的一个字母值。

因为只有字母才能加密和解密，所以有了第 32 行。数字、标点符号和其他的所有内容都保持其原来的形式不变。把存储在变量 symbol 中的字母的整数编码保存在变量 num 中。然后，第 34 行代码把 num 中的值“移动”key 那么多个位。

## 15.9 字符串方法 isupper()和 islower()

（第 36 行和第 41 行的）字符串方法 isupper()和 islower()的工作方式，与 isdigit()和 isalpha()方法的工作方式类似。

如果调用该方法的字符串至少包含一个大写字母并且没有小写字母，isupper()方法将返回 True。如果调用该方法的字符串至少包含一个小写字母并且没有大写字母，islower()方法将返回 True。否则，这两个方法都返回 False。

尝试在交互式 shell 中输入如下代码：

```
>>> 'HELLO'.isupper()
True
>>> 'hello'.isupper()
False
>>> 'hello'.islower()
True
>>> 'Hello'.islower()
False
>>> 'LOOK OUT BEHIND YOU!'.isupper()
True
>>> '42'.isupper()
False
>>> '42'.islower()
False
>>> ''.isupper()
False
>>> ''.islower()
False
```

### 15.9.1　加密或解密每个字母

```
36.           if symbol.isupper():
37.               if num > ord('Z'):
38.                   num -= 26
39.               elif num < ord('A'):
40.                   num += 26
```

第 36 行判断符号是否是一个大写字母。如果是，要注意两种特殊情况。如果 symbol 是'Z'，并且密钥是 4，那该怎么办？如果出现这种情况，这里的 num 的值将是字符'^'（'^'的编码是 94）。但是，^根本不是一个字母。我们要把这个密文“折返”到字母表的开始处。

判断 num 的值是否大于“Z”的编码。如果大于，那么用 num 减去 26（因为一共有 26 个字母）。这样做之后，num 的值是 68。68 是'D'的编码，这就对了。

```
41.           elif symbol.islower():
42.               if num > ord('z'):
43.                   num -= 26
44.               elif num < ord('a'):
45.                   num += 26
```

如果符号是一个小写字母，程序运行的代码类似于第 36 行到 40 行。唯一的区别是，它使用了 ord('z')和 ord('a')，而不是 ord('Z')和 ord('A')。

在解密模式中，密钥将是负数。特殊的情况是 num -= 26 小于“a”的 ASCII 值。在这种情况下，num 加上 26 以“折返”到字母表的末尾。

```
47.           translated += chr(num)
48.       else:
49.           translated += symbol
```

第 47 行把加密后（解密后）的字符连接到 translated 字符串的末尾。

如果符号不是一个大写字母或小写字母，那么第 48 行把原来的符号连接到 translated 字符串上。因此，空格、数字、标点符号和其他字符不会加密或解密。

```
50.    return translated
```

在 getTranslatedMessage()函数中的最后一行，返回了字符串 translated。

### 15.9.2　程序开始

```
52. mode = getMode()
53. message = getMessage()
54. key = getKey()
```

```
55. print('Your translated text is:')
56. print(getTranslatedMessage(mode, message, key))
```

游戏开始处调用了之前定义的 3 个函数，以便从用户那里获取模式、消息和密钥。然后把这 3 个值传送给 getTranslatedMessage()函数，该函数的返回值（转换后的字符串）会打印给用户。

## 15.10 暴力破解

这就是整个凯撒密码程序。这个密码可能会骗过一些不知道密码学的人，但对于了解密码分析学的人来说，这无法为一条消息进行保密。密码学是创建编码的科学，密码分析学（cryptanalysis）是破解编码的科学。

```
Do you wish to encrypt or decrypt a message?
encrypt
Enter your message:
Doubts may not be pleasant, but certainty is absurd.
Enter the key number (1-26)
8
Your translated text is:
Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.
```

密码学的关键是，如果别人得到了加密消息，也无法从中得到最初未加密的消息。假设我们是编码破译员，有一条加密的文本如下所示：

```
Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.
```

暴力破解（brute force）是对每一种可能的秘钥进行尝试，直到找到正确的密钥的一种技术。因为只有 26 种可能的密钥，对一个密码分析师来讲，编写一个黑客程序，然后用每种可能的密钥进行破解，这是一件很容易的事情。然后，他们可以找到把密文解密成普通英语的密钥。我们来对这个程序进行暴力破解。

### 添加暴力破解模式

首先，修改第 7 行、第 9 行和第 12 行代码（这些代码行在 getMode()函数中），使其如下所示（修改的地方用粗体表示）：

```
 5. def getMode():
 6.     while True:
 7.         print('Do you wish to encrypt or decrypt or brute force a message?')
 8.         mode = input().lower()
```

```
 9.        if mode in 'encrypt e decrypt d brute b'.split():
10.            return mode[0]
11.        else:
12.            print('Enter either "encrypt" or "e" or "decrypt" or "d" or
"brute" or "b".')
```

这行代码将让用户把暴力破解作为一种可选模式。修改程序的主体部分，并增加如下的代码：

```
52. mode = getMode()
53. message = getMessage()
54. if mode[0] != 'b':
55.     key = getKey()
56.
57. print('Your translated text is:')
58. if mode[0] != 'b':
59.     print(getTranslatedMessage(mode, message, key))
60. else:
61.      for key in range(1, MAX_KEY_SIZE + 1):
62.            print(key, getTranslatedMessage('decrypt', message, key))
```

这里的修改是，如果用户没有进入暴力破解模式，就要求用户输入一个密钥。调用原来的 getTranslatedMessage()函数，并将转换后的字符串打印出来。

然而，如果用户在暴力破解模式中，那么 getTranslatedMessage()循环从 1 迭代到 MAX_KEY_SIZE（26）。记住，当 range()函数返回一个整数列表时，这个列表以第 2 个参数为上限但是并不包括它，这就是为什么要使用+1。这个程序将打印每一种可能的转换消息（包括在转换中用到的密钥数字）。下面是运行这个修改过程序的一个示例：

```
Do you wish to encrypt or decrypt or brute force a message?
brute
Enter your message:
Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.
Your translated text is:
1 Kvbiaz thf uva il wslhzhua, iba jlyahpuaf pz hizbyk.
2 Juahzy sge tuz hk vrkgygtz, haz ikxzgotze oy ghyaxj.
3 Itzgyx rfd sty gj uqjfxfsy, gzy hjwyfnsyd nx fgxzwi.
4 Hsyfxw qec rsx fi tpiewerx, fyx givxemrxc mw efwyvh.
5 Grxewv pdb qrw eh sohdvdqw, exw fhuwdlqwb lv devxug.
6 Fqwdvu oca pqv dg rngcucpv, dwv egtvckpva ku cduwtf.
7 Epvcut nbz opu cf qmfbtbou, cvu dfsubjouz jt bctvse.
8 Doubts may not be pleasant, but certainty is absurd.
9 Cntasr lzx mns ad okdzrzms, ats bdqszhmsx hr zartqc.
10 Bmszrq kyw lmr zc njcyqylr, zsr acpryglrw gq yzqspb.
```

```
11 Alryqp jxv klq yb mibxpxkq, yrq zboqxfkqv fp xyproa.
12 Zkqxpo iwu jkp xa lhawowjp, xqp yanpwejpu eo wxoqnz.
13 Yjpwon hvt ijo wz kgzvnvio, wpo xzmovdiot dn vwnpmy.
14 Xiovnm gus hin vy jfyumuhn, von wylnuchns cm uvmolx.
15 Whnuml ftr ghm ux iextltgm, unm vxkmtbgmr bl tulnkw.
16 Vgmtlk esq fgl tw hdwsksfl, tml uwjlsaflq ak stkmjv.
17 Uflskj drp efk sv gcvrjrek, slk tvikrzekp zj rsjliu.
18 Tekrji cqo dej ru fbuqiqdj, rkj suhjqydjo yi qrikht.
19 Sdjqih bpn cdi qt eatphpci, qji rtgipxcin xh pqhjgs.
20 Rciphg aom bch ps dzsogobh, pih qsfhowbhm wg opgifr.
21 Qbhogf znl abg or cyrnfnag, ohg pregnvagl vf nofheq.
22 Pagnfe ymk zaf nq bxqmemzf, ngf oqdfmuzfk ue mnegdp.
23 Ozfmed xlj yze mp awpldlye, mfe npceltyej td lmdfco.
24 Nyeldc wki xyd lo zvokckxd, led mobdksxdi sc klcebn.
25 Mxdkcb vjh wxc kn yunjbjwc, kdc lnacjrwch rb jkbdam.
26 Lwcjba uig vwb jm xtmiaivb, jcb kmzbiqvbg qa ijaczl.
```

当遍历了每行之后，就会看到第 8 条消息不是垃圾信息，而是普通的英语！密码分析师可以推断出，这个加密文本的原始密钥肯定是 8。这种暴力破解对于凯撒时代和罗马帝国时期来说很困难，但是今天我们有了计算机，可以在很短的时间内很快地遍历几百万个甚至几十亿个密钥。

## 15.11 本章小结

计算机很擅长做数学运算。当我们创建一个系统把一些信息转换成一些数字时（就像我们把文本转换成编码或把格子转换成坐标系），计算机程序可以快速高效地处理这些数字。

尽管我们的凯撒密码程序可以加密消息，使得人们只是使用笔和纸做计算的话无法破解它，然而，对知道如何用计算机处理信息的人来说，这是无法保密的（我们的暴力破解模式证明了这一点）。

要搞清楚如何编写程序，很大一部分工作是搞清楚如何将想要操作的信息表示为 Python 所能够理解的值。

下一章将要介绍 Reversi（也叫做 Othello）。这个游戏的 AI 要比第 10 章中的 Tic Tac Toe 的 AI 更高级。实际上，这个 AI 相当厉害，大多数的时候，你是无法战胜它的。

# 第 16 章　Reversi

**本章主要内容：**

- bool()函数；
- 如何玩 Reversi 游戏。

在本章中，我们将创建一款名为 Reversi（也叫做 Othello）的游戏。Reversi 是一款在网格上玩的游戏板游戏，所以我们将使用带有 XY 坐标的一个笛卡尔坐标系。这是一款双人游戏。我们的这个游戏版本将有一个计算机 AI，它要比我们在 Tic Tac Toe 中创建的 AI 更高级。实际上，这个 AI 如此厉害，以至于它几乎在每一次游戏中都能够击败你（我知道任何时候和它玩我都会输！）。

Reversi 有一个 8×8 的游戏板，一方的棋子是黑色，另一方的棋子是白色（我们的游戏使用 O 和 X 代替这两种颜色）。开始的游戏板看上去如图 16-1 所示。黑方玩家和白方玩家轮流下一个自己颜色的棋子。在新的棋子和同一颜色的另一个棋子之间，如果有任何对手的棋子，都将其反转。游戏的目标是让你的棋子尽可能地多。例如，如图 16-2 所示，白方玩家在格子（5，6）处落下了一个新棋子。

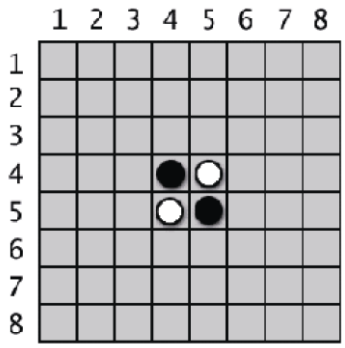



图 16-1　Reversi 游戏板上最初有两个白子和两个黑子

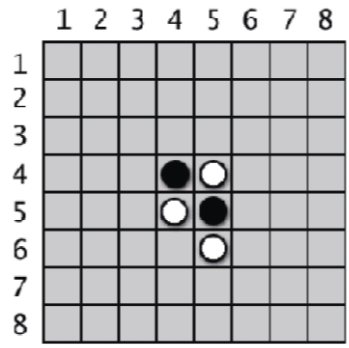



图 16-2　白方下了一个新子

（5，5）处的黑子处于新的白子和（5，4）处已有的白子之间。这个黑子被反转为一个新的白子，游戏板看上去如图 16-3 所示。黑方接下来做了一个类似的移动，在（4，6）处放了一个黑子。这导致游戏板如图 16-4 所示。

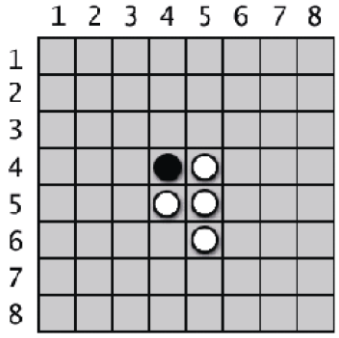



图 16-3 白方的移动反转了一个黑方棋子

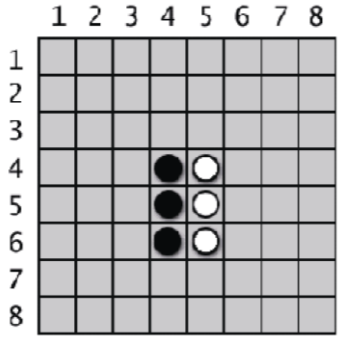



图 16-4 黑方下了一个新棋子，它反转了一个白方棋子

在各个方向上的棋子，只要它在玩家的新落棋子和已有的棋子之间，都会被反转。在图 16-5 中，白方玩家在（3，6）处下了一个棋子，反转了两个方向的黑子（用连线来表示）。结果如图 16-6 所示。

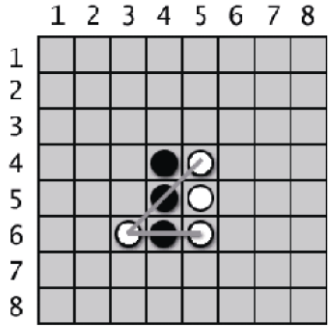



图 16-5 白方的第 2 次移动在（3，6）处，它将反转两个黑方的棋子

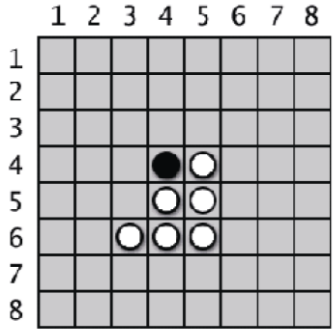



图 16-6 白方的第 2 次移动之后的游戏板

每个玩家可以在一两步之内快速反转棋盘上的很多棋子。玩家每一回合必须至少要翻转一颗对手的棋子。当任意一位玩家不能落子或游戏板填满了的时候，游戏就结束了。棋子多的玩家最终获胜。

我们为这个游戏创建的 AI 将直接寻找所有可以落子的角落。如果没有角落可以落子，那么计算机将选择能够反转最多棋子的位置落子。

## 16.1 Reversi 的运行示例

```
Welcome to Reversi!
Do you want to be X or O?
```

```
x
The player will go first.
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
3 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
4 |   |   |   | X | O |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
5 |   |   |   | O | X |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
You have 2 points. The computer has 2 points.
Enter your move, or type quit to end the game, or hints to turn off/on hints.
53
    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
1 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
```

```
  |   |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
3 |   |   |   |   | X |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
4 |   |   |   | X | X |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
5 |   |   |   | O | X |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
6 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
You have 4 points. The computer has 1 points.
Press Enter to see the computer's move.

...skipped for brevity...

    1   2   3   4   5   6   7   8
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
1 | O | O | O | O | O | O | O | O |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
2 | O | O | O | O | O | O | O | O |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
3 | O | O | O | O | O | O | O | O |
```

```
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |
  4 | O | O | X | O | O | O | O | O |
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |
  5 | O | O | O | X | O | X | O | X |
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |
  6 | O | X | O | X | X | O | O |   |
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |
  7 | O | X | X | O | O | O | O | O |
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
    |   |   |   |   |   |   |   |   |
  8 | O | X | X | O |   |   | X |   |
    |   |   |   |   |   |   |   |   |
    +---+---+---+---+---+---+---+---+
You have 12 points. The computer has 48 points.
Enter your move, or type quit to end the game, or hints to turn off/on hints.
86
X scored 15 points. O scored 46 points.
You lost. The computer beat you by 31 points.
Do you want to play again? (yes or no)
no
```

正如你所见到的，AI 非常厉害，以 46 比 15 击败了我。为了帮助玩家胜出，我们编写的游戏中提供了提示功能。当轮到玩家移动时，如果他们输入了'hints'，就可以在开启提示模式和关闭提示模式之间进行切换。当开启提示模式时，玩家所有可能的落子都将以'.'字符的形式显示在游戏板上，如下所示：

```
     1   2   3   4   5   6   7   8
   +---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |
 1 |   |   |   |   |   |   |   |   |
   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |
 2 |   |   |   | . |   | . |   |   |
   |   |   |   |   |   |   |   |   |
```

```
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
3 |   |   |   | O | O | O |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
4 |   |   | . | O | O | X |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
5 |   |   | . | O | O | O | X |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
6 |   |   |   | . |   | . |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
7 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
  |   |   |   |   |   |   |   |   |
8 |   |   |   |   |   |   |   |   |
  |   |   |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+
```

## 16.2 Reversi 的源代码

和前面的程序相比，Reversi 是一个很大的程序。它的代码超过了 300 行！但是别担心，这些代码行中很多是注释和用来间隔代码以使其更具可读性的空白行。

和其他程序一样，我们将先创建几个函数，以便程序的主体部分能够调用它们来执行与 Reversi 相关的任务。大体上，前 250 行的代码是这些辅助性的函数，剩下的 50 行代码实现了 Reversi 游戏本身。

如果输入这些代码后出现错误，请使用 http://invpy.com/diff/reversi 上的在线 diff 工具，把你的代码与书中的代码进行比较。

reversi.py

```
1.# Reversi
2.
3. import random
4. import sys
5.
```

```
 6. def drawBoard(board):
 7.     # This function prints out the board that it was passed. Returns None.
 8.     HLINE = '  +---+---+---+---+---+---+---+---+'
 9.     VLINE = '  |   |   |   |   |   |   |   |   |'
10.
11.     print('    1   2   3   4   5   6   7   8')
12.     print(HLINE)
13.     for y in range(8):
14.         print(VLINE)
15.         print(y+1, end=' ')
16.         for x in range(8):
17.             print('| %s' % (board[x][y]), end=' ')
18.         print('|')
19.         print(VLINE)
20.         print(HLINE)
21.
22.
23. def resetBoard(board):
24.     # Blanks out the board it is passed, except for the original starting
position.
25.     for x in range(8):
26.         for y in range(8):
27.             board[x][y] = ' '
28.
29.     # Starting pieces:
30.     board[3][3] = 'X'
31.     board[3][4] = 'O'
32.     board[4][3] = 'O'
33.     board[4][4] = 'X'
34.
35.
36. def getNewBoard():
37.     # Creates a brand new, blank board data structure.
38.     board = []
39.     for i in range(8):
40.         board.append([' '] * 8)
41.
42.     return board
43.
44.
45. def isValidMove(board, tile, xstart, ystart):
46.     # Returns False if the player's move on space xstart, ystart is
invalid.
47.     # If it is a valid move, returns a list of spaces that would become
the player's if they made a move here.
```

```
    if board[xstart][ystart] != ' ' or not isOnBoard(xstart, ystart):
        return False

    board[xstart][ystart] = tile # temporarily set the tile on the board.

    if tile == 'X':
        otherTile = 'O'
    else:
        otherTile = 'X'

    tilesToFlip = []
    for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0,
-1], [-1, -1], [-1, 0], [-1, 1]]:
        x, y = xstart, ystart
        x += xdirection # first step in the direction
        y += ydirection # first step in the direction
        if isOnBoard(x, y) and board[x][y] == otherTile:
            # There is a piece belonging to the other player next to our piece.
            x += xdirection
            y += ydirection
            if not isOnBoard(x, y):
                continue
            while board[x][y] == otherTile:
                x += xdirection
                y += ydirection
                if not isOnBoard(x, y): # break out of while loop, then
continue in for loop
                    break
            if not isOnBoard(x, y):
                continue
            if board[x][y] == tile:
                # There are pieces to flip over. Go in the reverse direction
until we reach the original space, noting all the tiles along the way.
                while True:
                    x -= xdirection
                    y -= ydirection
                    if x == xstart and y == ystart:
                        break
                    tilesToFlip.append([x, y])

    board[xstart][ystart] = ' ' # restore the empty space
    if len(tilesToFlip) == 0: # If no tiles were flipped, this is not
a valid move.
```

```
 87.         return False
 88.     return tilesToFlip
 89.
 90.
 91. def isOnBoard(x, y):
 92.     # Returns True if the coordinates are located on the board.
 93.     return x >= 0 and x <= 7 and y >= 0 and y <=7
 94.
 95.
 96. def getBoardWithValidMoves(board, tile):
 97.     # Returns a new board with . marking the valid moves the given player
can make.
 98.     dupeBoard = getBoardCopy(board)
 99.
100.     for x, y in getValidMoves(dupeBoard, tile):
101.         dupeBoard[x][y] = '.'
102.     return dupeBoard
103.
104.
105. def getValidMoves(board, tile):
106.     # Returns a list of [x,y] lists of valid moves for the given player
on the given board.
107.     validMoves = []
108.
109.     for x in range(8):
110.         for y in range(8):
111.             if isValidMove(board, tile, x, y) != False:
112.                 validMoves.append([x, y])
113.     return validMoves
114.
115.
116. def getScoreOfBoard(board):
117.     # Determine the score by counting the tiles. Returns a dictionary
with keys 'X' and 'O'.
118.     xscore = 0
119.     oscore = 0
120.     for x in range(8):
121.         for y in range(8):
122.             if board[x][y] == 'X':
123.                 xscore += 1
124.             if board[x][y] == 'O':
125.                 oscore += 1
126.     return {'X':xscore, 'O':oscore}
127.
```

```
128.
129. def enterPlayerTile():
130.     # Lets the player type which tile they want to be.
131.     # Returns a list with the player's tile as the first item, and the
computer's tile as the second.
132.     tile = ''
133.     while not (tile == 'X' or tile == 'O'):
134.         print('Do you want to be X or O?')
135.         tile = input().upper()
136.
137.     # the first element in the list is the player's tile, the second
is the computer's tile.
138.     if tile == 'X':
139.         return ['X', 'O']
140.     else:
141.         return ['O', 'X']
142.
143.
144. def whoGoesFirst():
145.     # Randomly choose the player who goes first.
146.     if random.randint(0, 1) == 0:
147.         return 'computer'
148.     else:
149.         return 'player'
150.
151.
152. def playAgain():
153.     # This function returns True if the player wants to play again,
otherwise it returns False.
154.     print('Do you want to play again? (yes or no)')
155.     return input().lower().startswith('y')
156.
157.
158. def makeMove(board, tile, xstart, ystart):
159.     # Place the tile on the board at xstart, ystart, and flip any of
the opponent's pieces.
160.     # Returns False if this is an invalid move, True if it is valid.
161.     tilesToFlip = isValidMove(board, tile, xstart, ystart)
162.
163.     if tilesToFlip == False:
164.         return False
165.
166.     board[xstart][ystart] = tile
167.     for x, y in tilesToFlip:
```

```
168.        board[x][y] = tile
169.    return True
170.
171.
172. def getBoardCopy(board):
173.    # Make a duplicate of the board list and return the duplicate.
174.    dupeBoard = getNewBoard()
175.
176.    for x in range(8):
177.        for y in range(8):
178.            dupeBoard[x][y] = board[x][y]
179.
180.    return dupeBoard
181.
182.
183. def isOnCorner(x, y):
184.    # Returns True if the position is in one of the four corners.
185.    return (x == 0 and y == 0) or (x == 7 and y == 0) or (x == 0 and
y == 7) or (x == 7 and y == 7)
186.
187.
188. def getPlayerMove(board, playerTile):
189.    # Let the player type in their move.
190.    # Returns the move as [x, y] (or returns the strings 'hints' or 'quit')
191.    DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
192.    while True:
193.        print('Enter your move, or type quit to end the game, or hints
to turn off/on hints.')
194.        move = input().lower()
195.        if move == 'quit':
196.            return 'quit'
197.        if move == 'hints':
198.            return 'hints'
199.
200.         if len(move) == 2 and move[0] in DIGITS1TO8 and move[1] in
DIGITS1TO8:
201.            x = int(move[0]) - 1
202.            y = int(move[1]) - 1
203.            if isValidMove(board, playerTile, x, y) == False:
204.                continue
205.            else:
206.                break
207.        else:
208.            print('That is not a valid move. Type the x digit (1-8), then
```

```
the y digit (1-8).')
209.            print('For example, 81 will be the top-right corner.')
210.
211.    return [x, y]
212.
213.
214. def getComputerMove(board, computerTile):
215.    # Given a board and the computer's tile, determine where to
216.    # move and return that move as a [x, y] list.
217.    possibleMoves = getValidMoves(board, computerTile)
218.
219.    # randomize the order of the possible moves
220.    random.shuffle(possibleMoves)
221.
222.    # always go for a corner if available.
223.    for x, y in possibleMoves:
224.       if isOnCorner(x, y):
225.          return [x, y]
226.
227.    # Go through all the possible moves and remember the best scoring
move
228.    bestScore = -1
229.    for x, y in possibleMoves:
230.       dupeBoard = getBoardCopy(board)
231.       makeMove(dupeBoard, computerTile, x, y)
232.       score = getScoreOfBoard(dupeBoard)[computerTile]
233.       if score > bestScore:
234.          bestMove = [x, y]
235.          bestScore = score
236.    return bestMove
237.
238.
239. def showPoints(playerTile, computerTile):
240.    # Prints out the current score.
241.    scores = getScoreOfBoard(mainBoard)
242.     print('You have %s points. The computer has %s points.' %
(scores[playerTile], scores[computerTile]))
243.
244.
245.
246. print('Welcome to Reversi!')
247.
248. while True:
249.    # Reset the board and game.
```

```
250.     mainBoard = getNewBoard()
251.     resetBoard(mainBoard)
252.     playerTile, computerTile = enterPlayerTile()
253.     showHints = False
254.     turn = whoGoesFirst()
255.     print('The ' + turn + ' will go first.')
256.
257.     while True:
258.         if turn == 'player':
259.             # Player's turn.
260.             if showHints:
261.                 validMovesBoard = getBoardWithValidMoves(mainBoard,
playerTile)
262.                 drawBoard(validMovesBoard)
263.             else:
264.                 drawBoard(mainBoard)
265.             showPoints(playerTile, computerTile)
266.             move = getPlayerMove(mainBoard, playerTile)
267.             if move == 'quit':
268.                 print('Thanks for playing!')
269.                 sys.exit() # terminate the program
270.             elif move == 'hints':
271.                 showHints = not showHints
272.                 continue
273.             else:
274.                 makeMove(mainBoard, playerTile, move[0], move[1])
275.
276.             if getValidMoves(mainBoard, computerTile) == []:
277.                 break
278.             else:
279.                 turn = 'computer'
280.
281.         else:
282.             # Computer's turn.
283.             drawBoard(mainBoard)
284.             showPoints(playerTile, computerTile)
285.             input('Press Enter to see the computer\'s move.')
286.             x, y = getComputerMove(mainBoard, computerTile)
287.             makeMove(mainBoard, computerTile, x, y)
288.
289.             if getValidMoves(mainBoard, playerTile) == []:
290.                 break
291.             else:
292.                 turn = 'player'
```

```
293.
294.    # Display the final score.
295.    drawBoard(mainBoard)
296.    scores = getScoreOfBoard(mainBoard)
297.    print('X scored %s points. O scored %s points.' % (scores['X'],
scores['O']))
298.    if scores[playerTile] > scores[computerTile]:
299.        print('You beat the computer by %s points! Congratulations!' %
(scores[playerTile] - scores[computerTile]))
300.    elif scores[playerTile] < scores[computerTile]:
301.        print('You lost. The computer beat you by %s points.' %
(scores[computerTile] - scores[playerTile]))
302.    else:
303.        print('The game was a tie!')
304.
305.    if not playAgain():
306.        break
```

# 16.3 代码如何工作

## 16.3.1 游戏板数据结构

在开始阅读代码之前，我们先来搞清楚游戏板的数据结构。这个数据结构是列表的列表，就像第 15 章的 Sonar 游戏中的游戏板一样。创建列表的列表，以便使用 board[x][y]来表示坐落于 X 坐标轴（向左/向右）上的 x 位置和坐落于 Y 坐标轴（向上/向下）上的 y 位置的格子上的字符。

这个字符既可以是一个空格（' '）字符（表示一个空的格子）、一个点（'.'）字符（在提示模式中表示一个可以落子的格子），或者一个'X'字符或'O'字符（表示玩家的棋子）。任何时候，当看到一个名为 board 的参数，它都表示这种数据结构，即列表的列表。

## 16.3.2 导入其他模块

```
1. # Reversi
2. import random
3. import sys
```

第 2 行导入了 random 模块，以便使用其 randint()函数和 choice()函数。第 3 行导入了 sys 模块，以便使用其 exit()函数。

### 16.3.3　在屏幕上绘制游戏板数据结构

```
 6. def drawBoard(board):
 7.     # This function prints out the board that it was passed. Returns None.
 8.     HLINE = '  +---+---+---+---+---+---+---+---+'
 9.     VLINE = '  |   |   |   |   |   |   |   |   |'
10.
11.     print('    1   2   3   4   5   6   7   8')
12.     print(HLINE)
```

drawBoard()函数将根据 board 中的数据结构来打印当前游戏板。注意，游戏板的每个方块如下所示（也可能不是字符串'X'，而是'O'、'.'或' '）：

```
+---+
|   |
| X |
|   |
+---+
```

既然要一遍又一遍地打印水平线条，所以第 8 行把它存储到了一个名为 HLINE 的常量中。这将避免重复性地录入字符串。

在中心棋子的上面和下面都是'|'字符（叫做“管道”字符），每隔 3 个空格字符就有一个该字符。把这些垂直线条的字符存储到一个名为 VLINE 的常量中。

第 11 行是第一次调用 print()函数，它打印了游戏板顶部 X 坐标轴的标签。第 12 行打印了游戏板顶部的水平线条。

```
13.     for y in range(8):
14.         print(VLINE)
15.         print(y+1, end=' ')
16.         for x in range(8):
17.             print('| %s' % (board[x][y]), end=' ')
18.         print('|')
19.         print(VLINE)
20.         print(HLINE)
```

这个 for 循环将循环 8 次，每行一次。第 15 行代码打印了游戏板左边的 Y 坐标轴的标签，用关键字参数 end=' '来打印一个空格，而不是使用一个换行符。这里有另一个循环（它也会循环 8 次，针对每个格子一次）打印每个格子（是'X'、'O'还是' '则取决于 board[x][y]中存储的内容）。

在内部的循环中调用的 print()函数，其末尾也有关键字参数 end=' '，表示打印一个空格字符而不是一个换行符。这将在屏幕上创建诸如'| X | X | X | X | X | X | X | X '的单独一行

（如果 board[x][y]的每个值都是'X'的话）。

在内部循环之后，第 18 行调用 print()函数打印了最后一个'|'字符以及 1 个换行符。

从第 14 行到第 20 行的外部 for 循环的代码打印了游戏板的整个一行，如下所示：

```
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

当从第 13 行开始的 for 循环打印该行 8 次时，它就创建了整个游戏板（当然，游戏板上的一些格子将是'O'或' '，而不是'X'）：

```
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
| X | X | X | X | X | X | X | X |
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

### 16.3.4　重置游戏板

```
 23. def resetBoard(board):
 24.     # Blanks out the board it is passed, except for the original starting
position.
 25.     for x in range(8):
 26.         for y in range(8):
 27.             board[x][y] = ' '
```

第 25 行和第 26 行是嵌套循环，把 board 数据结构都设置为单个的空格字符串。这会设置一个空的 Reversi 游戏板。调用了 resetBoard()函数，作为开始一个新游戏的一部分。

### 16.3.5　设置开始棋子

```
 29.     # Starting pieces:
 30.     board[3][3] = 'X'
 31.     board[3][4] = 'O'
 32.     board[4][3] = 'O'
 33.     board[4][4] = 'X'
```

在游戏开始时，每个玩家都有两个棋子已经放在棋盘的中心了。第 30 行到第 33 行在空游戏板上设置了这些棋子。

resetBoard()函数不一定要返回 board 变量，因为 board 是对一个列表的引用。在函数的局部作用域中所做出的修改，将会改变作为参数传入的最初的列表（参见第 11 章的引用部分的介绍）。

### 16.3.6　创建一个新的游戏板数据结构

```
 36. def getNewBoard():
 37.     # Creates a brand new, blank board data structure.
38.     board = []
 39.     for i in range(8):
 40.         board.append([' '] * 8)
 41.
 42.     return board
```

getNewBoard()函数创建了一个新的游戏板数据结构，并将其返回。第 38 行创建了外部列表，并把对这个列表的引用存储到了 board 中。第 40 行使用列表复制创建了内部列表（[' '] * 8 相当于[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']，但是这种写法减少了输入量）。

第 39 行的 for 循环创建了 8 个内部列表。空格字符表示一个完全空白的游戏板。

board 最终是由 8 个列表构成的一个列表，并且这 8 个列表中的每一个列表自身都包含 8 个字符串。结果就是有 64 个' '字符串。

## 16.3.7 判断一次落子是否有效

```
45. def isValidMove(board, tile, xstart, ystart):
 46.     # Returns False if the player's move on space xstart, ystart is
invalid.
 47.     # If it is a valid move, returns a list of spaces that would become
the player's if they made a move here.
 48.     if board[xstart][ystart] != ' ' or not isOnBoard(xstart, ystart):
 49.         return False
 50.     board[xstart][ystart] = tile # temporarily set the tile on the board.
 51.     if tile == 'X':
 52.         otherTile = 'O'
 53.     else:
 54.         otherTile = 'X'
 55.     tilesToFlip = []
```

给定了一个游戏板数据结构、玩家的棋子以及玩家落子的 XY 坐标，如果 Reversi 游戏规则允许在该坐标落子，isValidMove()函数应该返回 True；否则，它返回 False。

第 48 行判断 XY 坐标是否在游戏板上，或者格子是否不为空。isOnBoard()是在程序的后面部分定义的一个函数，它确保 X 坐标和 Y 坐标都在 0 到 7 之间。

下一步是在游戏板上临时放置玩家的棋子。这个棋子最后会被移除（通过把游戏板格子设置回' '字符串，然后返回）。

玩家的棋子类型（可能是人类玩家也可能是计算机玩家）存储在 tile 中，但是这个函数需要知道对手玩家的棋子类型。如果玩家的棋子是'X'，那么显然对手的棋子是'O'；反之亦然。

最后，如果给定的 XY 坐标最终是一个有效位置，那么 isValidMove()函数返回这一步落子可能导致翻转的所有对手的棋子的一个列表。

```
59.     for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0,
-1], [-1, -1], [-1, 0], [-1, 1]]:
```

这个 for 循环遍历了一个列表的列表，它们表示在游戏板上可以移动的各个方向。游戏板是用 X 坐标和 Y 坐标表示的笛卡尔坐标系。这里有 8 个可以移动的方向：上、下、左、右和 4 个对角线的方向。在第 59 行，列表中的 8 个两元素列表中的每一个，都用来表示一个移动方向。程序通过把两元素列表中的第 1 个值加到 X 坐标上，将两元素列表中的第 2 个值加到 Y 坐标上，从而实现在该方向上的移动。

因为向右移动会增加 X 坐标，我们可以通过为 X 坐标增加 1 而向右“移动”。所以列表

[1, 0]给 X 坐标增加 1，给 Y 坐标增加 0，导致向右“移动”。向左移动则是相反操作：X 坐标减 1（也就是增加-1）。

但是要向对角线方向移动，需要对两个坐标都增加或减少。例如，X 坐标加 1 向右移动，Y 坐标增加-1 向上移动，结果就是向右对角线的右上方向移动。

### 16.3.8　查看 8 个方向中的每一个方向

图 16-7 使我们更容易记住两元素列表中的哪一项表示哪一个方向：

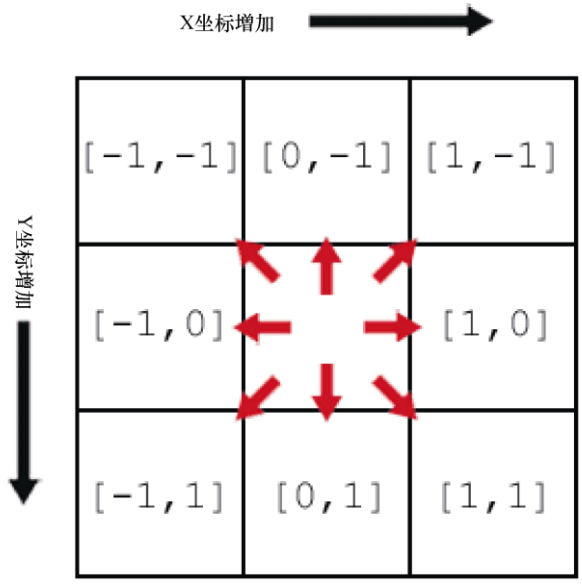



图 16-7　两元素列表中的每一个列表都表示 8 个方向中的 1 个方向

```
59.    for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0,
-1], [-1, -1], [-1, 0], [-1, 1]]:
 60.        x, y = xstart, ystart
 61.        x += xdirection # first step in the direction
 62.        y += ydirection # first step in the direction
```

第 60 行使用多变量赋值，把变量 x 和 y 分别设置成与 xstart 和 ystart 相同的值。修改 x 和 y，以便在 xdirection 和 ydirection 所确定的方向上的“移动”。变量 xstart 和 ystart 保持不变，以便程序可以记住最初是从哪个格子开始的。

```
 63.        if isOnBoard(x, y) and board[x][y] == otherTile:
 64.            # There is a piece belonging to the other player next to our piece.
```

```
 65.            x += xdirection
 66.            y += ydirection
 67.            if not isOnBoard(x, y):
 68.                continue
```

记住，为了让其成为一个有效的移动，在该方向上的移动的第一步必须（1）在游戏板上，（2）必须被对手玩家的棋子所占据。否则，就没有任何可供反转的对手的棋子，而有效移动必须要至少能够反转一个棋子。如果这两个条件不为真，第 63 行的条件就不为 True，执行就会返回到 for 语句以迭代下一个方向。

但是如果第 1 个格子有对方的棋子，那么程序应该在该方向上继续判断，直到遇到一个玩家的棋子。如果超出了游戏板的边界，第 68 行的 continue 语句返回到 for 语句，继续尝试下一个方向。

```
 69.            while board[x][y] == otherTile:
 70.                x += xdirection
 71.                y += ydirection
 72.                if not isOnBoard(x, y): # break out of while loop, then
continue in for loop
 73.                    break
 74.            if not isOnBoard(x, y):
 75.                continue
```

第 69 行的 while 循环会一直循环，以便只要看到对手玩家的棋子，x 和 y 就能够一直在当前方向上前进。在第 72 行，如果检测到 x 和 y 移出了游戏板，第 73 行就跳出这个 while 循环，接下来执行将移动到第 74 行。

我们真正想做的就是跳出这个 while 循环，但是继续在 for 循环中。这就是为什么第 74 行重新判断 not isOnBoard(x, y)，并且运行 continue 语句，它把执行跳转到第 59 行的 for 语句，以遍历下一个方向。记住，break 语句和 continue 语句只是跳出其所在的最内部的循环。

### 16.3.9 发现是否有可以反转的棋子

```
 76.            if board[x][y] == tile:
 77.                # There are pieces to flip over. Go in the reverse direction
until we reach the original space, noting all the tiles along the way.
 78.                while True:
 79.                    x -= xdirection
 80.                    y -= ydirection
 81.                    if x == xstart and y == ystart:
 82.                        break
 83.                    tilesToFlip.append([x, y])
```

当代码到达了最后一个 otherTile 中的棋子时，第 69 行的 while 循环结束了。第 76 行代码判断游戏板上的这个格子是否存储了我们的一个棋子。如果是的话，那么最初传递给 isValidMove()函数的移动就是有效的。

第 78 行的循环，通过对 x 和 y 做减法，将 x 和 y 向最初的 xstart 和 ystart 的位置移动。每个格子都会添加到 tilesToFlip 列表中。

```
 85.     board[xstart][ystart] = ' ' # restore the empty space
 86.     if len(tilesToFlip) == 0: # If no tiles were flipped, this is not
a valid move.
 87.         return False
 88.     return tilesToFlip
```

从第 59 行开始的 for 循环在 8 个方向上把这一切都做了一遍。在该循环完成之后，tilesToFlip 列表将包含了如果玩家移动到 xstart 和 ystart，所有应该被反转的对手的棋子的 XY 坐标。记住，isValidMove()函数只会判断最初的移动是否有效。它实际上不会持久性地修改游戏板的数据结构。

如果在 8 个方向上最终没有反转对手的任何一个棋子，那么 tilesToFlip 将是一个空的列表。这标志着这次移动不是有效的，isValidMove()应该返回 False。

否则，isValidMove()返回 tilesToFlip。

### 16.3.10　判断有效的坐标

```
 91. def isOnBoard(x, y):
 92.     # Returns True if the coordinates are located on the board.
 93.     return x >= 0 and x <= 7 and y >= 0 and y <=7
```

isOnBoard()是 isValidMove()调用的一个函数。调用 isOnBoard()函数只不过是第 93 行布尔表达式的一种快捷方式，如果 x 和 y 都在 0 到 7 之间，这个表达式为 True。这个函数判断 X 和 Y 坐标是否真的在游戏板上。例如，如果 X 坐标为 4，Y 坐标为 9999，就不在游戏板上，因为 Y 坐标最大只能到 7。

### 16.3.11　得到所有有效移动的一个列表

```
 96. def getBoardWithValidMoves(board, tile):
 97.     # Returns a new board with . marking the valid moves the given player
can make.
 98.     dupeBoard = getBoardCopy(board)
 99.
100.     for x, y in getValidMoves(dupeBoard, tile):
```

```
101.         dupeBoard[x][y] = '.'
102.     return dupeBoard
```

getBoardWithValidMoves()返回一个游戏板数据结构，对于所有有效移动的格子都使用'.'字符表示。这里的点是为了在提示模式下显示出标记了所有可能移动的一个游戏板。

这个函数创建了 board 数据结构的一个副本（在第 98 行由 getBoardCopy()函数返回），而不是修改在 board 参数中传递给它的那个数据结构。第 100 行调用了 getValidMoves()函数，以得到玩家可以做出的所有有效移动的 XY 坐标的一个列表。这个 board 副本在那些有效移动的格子中标记了点，并且返回。

```
105. def getValidMoves(board, tile):
106.     # Returns a list of [x,y] lists of valid moves for the given player
on the given board.
107.     validMoves = []
108.
109.     for x in range(8):
110.         for y in range(8):
111.             if isValidMove(board, tile, x, y) != False:
112.                 validMoves.append([x, y])
113.     return validMoves
```

getValidMoves()函数返回了包含两元素列表的一个列表。这些列表保存了 tile 玩家可以在参数 board 中的游戏板数据结构中进行的所有的有效移动的 XY 坐标。

这个函数使用了嵌套循环（第 109 行和第 110 行），通过对该格子调用 isValidMove()函数，并且检查它是返回了 False 还是可能的移动的一个列表（在后一种情况下，它是一个有效的移动），从而检查了每一个 XY 坐标。把每个有效的 XY 坐标都添加到 validMoves 列表中。

## 16.4 bool()函数

bool()函数与 int()函数和 str()函数类似。它将传递给它的值，以布尔类型的值的形式返回。

大部分数据类型都有一个值可以当作布尔类型的 False，而其他值则当成是 True。例如，当整数 0、浮点数 0.0、空字符串、空的列表和空的字典用做 if 语句或循环语句的条件时，都可以看做是 False，而所有其他值都是 True。尝试在交互式 shell 中输入：

```
>>> bool(0)
False
>>> bool(0.0)
False
>>> bool('')
False
```

```
>>> bool([])
False
>>> bool({})
False
>>> bool(1)
True
>>> bool('Hello')
True
>>> bool([1, 2, 3, 4, 5])
True
>>> bool({'spam':'cheese', 'fizz':'buzz'})
True
```

想象一下，将任何语句的条件都放置在对 bool()函数的一次调用之中。这样一来，条件会自动解释为布尔值。这就是为什么第 111 行的条件可以正常工作。对 isValidMove()函数的调用要么返回布尔值 False，要么返回一个非空列表。

如果假设整个条件都放在 bool()函数的一次调用之中，那么第 111 行的条件 False 变成了 bool(False)（当然，结果为 False）。并且，当非空列表的一个条件作为参数传递给 bool()函数的时候，该函数将会返回 True。

## 16.4.1　计算游戏板的得分

```
116. def getScoreOfBoard(board):
117.     # Determine the score by counting the tiles. Returns a dictionary
with keys 'X' and 'O'.
118.     xscore = 0
119.     oscore = 0
120.     for x in range(8):
121.         for y in range(8):
122.             if board[x][y] == 'X':
123.                 xscore += 1
124.             if board[x][y] == 'O':
125.                 oscore += 1
126.     return {'X':xscore, 'O':oscore}
```

getScoreOfBoard()函数使用嵌套 for 循环检查游戏板上的所有 64 个格子（8 行乘以每行的 8 列，一共是 64 个格子），并且看看哪些棋子在上面（如果有棋子的话）。对于每个'X'棋子，第 123 行代码会将 xscore 加 1。对于每个'O'棋子，第 125 行代码会将 oscore 加 1。

## 16.4.2　获取玩家的棋子选择

```
129. def enterPlayerTile():
130.     # Lets the player type which tile they want to be.
```

```
131.    # Returns a list with the player's tile as the first item, and the
computer's tile as the second.
132.    tile = ''
133.    while not (tile == 'X' or tile == 'O'):
134.        print('Do you want to be X or O?')
135.        tile = input().upper()
```

这个函数询问玩家想要什么棋子，是'X'还是'O'。这个 for 循环将一直循环，直到玩家输入'X'或'O'。

```
137.    # the first element in the list is the player's tile, the second
is the computer's tile.
138.    if tile == 'X':
139.        return ['X', 'O']
140.    else:
141.        return ['O', 'X']
```

然后，enterPlayerTile()函数返回两元素的一个列表，其中，玩家的棋子选择是第 1 个元素，计算机的棋子是第 2 个元素。第 252 行调用了 enterPlayerTile()函数，它使用多变量赋值，把返回的这两个元素存入到两个变量中。

### 16.4.3 决定谁先走

```
144. def whoGoesFirst():
145.     # Randomly choose the player who goes first.
146.    if random.randint(0, 1) == 0:
147.        return 'computer'
148.    else:
149.        return 'player'
```

whoGoesFirst()函数随机选择由谁先走，并且返回字符串'computer'或字符串'player'。

### 16.4.4 询问玩家是否要再玩一局

```
152. def playAgain():
153.    # This function returns True if the player wants to play again,
otherwise it returns False.
154.    print('Do you want to play again? (yes or no)')
155.    return input().lower().startswith('y')
```

playAgain()函数与前面游戏中的函数类似。如果玩家输入以'y'开头的字符串，那么该函数返回 True。否则，该函数返回 False。

### 16.4.5　在游戏板上落下一个棋子

```
158. def makeMove(board, tile, xstart, ystart):
159.     # Place the tile on the board at xstart, ystart, and flip any of
the opponent's pieces.
160.     # Returns False if this is an invalid move, True if it is valid.
161.     tilesToFlip = isValidMove(board, tile, xstart, ystart)
```

当想要在游戏板上落下一个棋子时，调用 makeMove()函数，并根据 Reversi 的游戏规则反转其他棋子。这个函数就地修改了所传入的 board 数据结构。对于 board 变量所做的修改（因为它是一个列表引用），将对全局作用域有效。

isValidMove()函数做了大部分的工作，它返回了需要反转的棋子的 XY 坐标（在一个两元素的列表中）的一个列表（记住，如果参数 xstart 和 ystart 指向一个无效的移动，那么 isValidMove()函数将返回布尔值 False）。

```
163.     if tilesToFlip == False:
164.         return False
165.
166.     board[xstart][ystart] = tile
167.     for x, y in tilesToFlip:
168.         board[x][y] = tile
169.     return True
```

在第 163 行和第 164 行中，如果 isValidMove()函数的返回值（现在存储在 tilesToFlip 中）是 False，那么 makeMove()函数也将返回 False。

否则，isValidMove()函数返回在游戏板上落子（tile 中的字符串'X'或'O'）的格子的一个列表。第 166 行设置了玩家已经落子的格子。第 167 行的 for 循环设置了 tilesToFlip 中的所有棋子。

### 16.4.6　复制游戏板数据结构

```
172. def getBoardCopy(board):
173.     # Make a duplicate of the board list and return the duplicate.
174.     dupeBoard = getNewBoard()
175.
176.     for x in range(8):
177.         for y in range(8):
178.             dupeBoard[x][y] = board[x][y]
179.
180.     return dupeBoard
```

getBoardCopy()函数和 getNewBoard()函数不同。getNewBoad()函数将创建一个空的游戏板数据结构，它只有空的格子和 4 个起始的棋子。getBoardCopy()将创建一个空白游戏板数据结构，然后从参数 board 中复制所有的格子。AI 使用这个函数，从而得到一个可以修改的游戏板，而无需修改游戏中真正的游戏板。这项技术也曾在之前的 Tic Tac Toe 游戏中使用。

调用 getNewBoard()函数来获取一个全新的游戏板数据结构。然后，使用两个嵌套的 for 循环，将游戏板的 64 个格子都复制到游戏板数据结构的副本 dupeBoard 中。

### 16.4.7 判断一个格子是否在角落上

```
183. def isOnCorner(x, y):
184.     # Returns True if the position is in one of the four corners.
185.     return (x == 0 and y == 0) or (x == 7 and y == 0) or (x == 0 and
y == 7) or (x == 7 and y == 7)
```

如果坐标是(0,0)、(7,0)、(0,7)和(7,7)，那么这个格子位于游戏板的角落之上，isOnCorner()函数就返回 True。否则，isOnCorner()函数返回 False。

### 16.4.8 获取玩家的移动

```
188. def getPlayerMove(board, playerTile):
189.     # Let the player type in their move.
190.     # Returns the move as [x, y] (or returns the strings 'hints' or 'quit')
191.     DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
```

调用 getPlayerMove()函数，让玩家输入他们下一步移动的坐标（并且检查这个移动是否有效）。玩家也可以输入'hints'来开启提示模式（如果是关闭状态），或者关闭提示模式（如果是开启状态）。玩家也可以输入'quit'来退出游戏。

DIGITS1TO8 常量是列表['1', '2', '3', '4', '5', '6', '7', '8']。因为输入 DIGITS1TO8 要比输入整个列表更为简单，所以我们使用这个常量。不能使用 isdigit()方法，因为那样的话，将允许输入 0 到 9，而这对于 8×8 游戏板上的坐标来说可能是无效的。

```
192.     while True:
193.         print('Enter your move, or type quit to end the game, or hints
to turn off/on hints.')
194.         move = input().lower()
195.         if move == 'quit':
196.             return 'quit'
197.         if move == 'hints':
198.             return 'hints'
```

这个 while 循环将一直循环，直到玩家输入一个有效的移动。第 195 行到第 198 行判断玩家是否想要退出游戏，或者是否想要切换提示模式，并且分别返回字符串'quit'和'hints'。通过对 input()函数返回的字符串调用 lower()方法，这样即使玩家输入'HINTS'或'Quit'，程序仍然能够理解该命令。

调用 getPlayerMove()函数的代码，将负责处理玩家想要退出游戏或者切换提示模式时所要做的事情。

```
200.         if len(move) == 2 and move[0] in DIGITS1TO8 and move[1] in
DIGITS1TO8:
201.             x = int(move[0]) - 1
202.             y = int(move[1]) - 1
203.             if isValidMove(board, playerTile, x, y) == False:
204.                 continue
205.             else:
206.                 break
```

游戏希望玩家输入他们移动的 XY 坐标，并且这两个数字之间没有其他东西。第 200 行先判断玩家输入的字符串的大小是否为 2。然后，它还会判断 move[0]（字符串中的第 1 个字符）和 move[1]（字符串中的第 2 个字符）是否都是 DIGITS1TO8 中已有的字符串。

记住，游戏板数据结构的索引是从 0 到 7，而不是 1 到 8。当在 drawBoard()函数中显示游戏板时，代码打印 1 到 8，因为非程序员都习惯于从 1 开始而不是从 0 开始计数，所以为了把 move[0]和 move[1]中的字符串转换成整数，第 201 行和第 202 行都有减 1 操作。

即使玩家输入了一个正确的移动，代码仍然需要判断：根据 Reversi 的规则，是否允许这个移动。isValidMove()函数可以做到这一点，它会接受游戏板数据结构、玩家的棋子类别以及这个移动的 XY 坐标作为参数。

如果 isValidMove()函数返回 False，那么执行第 204 行的 continue 语句。然后将返回到 while 循环的起始处，并且要求玩家再次输入一个有效的移动。

否则，玩家已经输入了一个有效的移动，执行需要跳出这个 while 循环。

```
207.         else:
208.             print('That is not a valid move. Type the x digit (1-8), then
the y digit (1-8).')
209.             print('For example, 81 will be the top-right corner.')
```

如果第 200 行的 if 语句的条件为 False，那么玩家没有输入一个有效的移动。第 208 行和第 209 行将告诉他们如何正确地输入移动。之后，回到第 192 行的 while 语句，因为第 209 行语句不仅是 else 语句块的最后一行，而且也是 while 语句块的最后一行。

```
211.     return [x, y]
```

最后，getPlayerMove()函数返回包含玩家有效移动的 XY 坐标的一个两元素列表。

## 16.4.9 获取计算机的移动

```
214. def getComputerMove(board, computerTile):
215.     # Given a board and the computer's tile, determine where to
216.     # move and return that move as a [x, y] list.
217.     possibleMoves = getValidMoves(board, computerTile)
```

getComputerMove()函数是实现 AI 算法的地方。通常，在提示模式下，会使用 getValidMoves()函数的结果。提示模式将在游戏板上打印'.'字符，以显示玩家可以进行的所有潜在的移动。

但是，如果使用计算机 AI 的棋子（在 computerTile 中）来调用 getValidMoves()函数，它也将找到计算机可以进行的所有可能的移动。AI 将从这个列表中选择最佳的移动。

```
219.     # randomize the order of the possible moves
220.     random.shuffle(possibleMoves)
```

首先，random.shuffle()函数将随机打乱 possibleMoves 列表中的移动的顺序。稍后，我们会介绍为什么要打乱 possibleMoves 列表的顺序，但是我们先来看看算法。

## 16.4.10 角落移动是最佳移动

```
222.     # always go for a corner if available.
223.     for x, y in possibleMoves:
224.         if isOnCorner(x, y):
225.             return [x, y]
```

首先，第 223 行循环遍历 possibleMoves 中的每次移动。如果它们中的任何一次移动是在角落，将其作为移动目标而返回该格子。在 Reversi 中，角落移动是一个好主意，因为一旦在角落上落下一个棋子，它就再也不能被反转了。因为 possibleMoves 是包含了两元素列表的一个列表，在 for 循环中，使用多变量赋值来设置 x 和 y。

如果 possibleMoves 包含多个角落移动，则总是使用第 1 个角落移动。但是由于在第 220 行中，possibleMoves 是随机排序的，所以列表中到底哪个角落移动会是第一个，将会随机选取。

## 16.4.11 获取最高得分的移动的列表

```
227.     # Go through all the possible moves and remember the best scoring
move
228.     bestScore = -1
```

```
229.    for x, y in possibleMoves:
230.        dupeBoard = getBoardCopy(board)
231.        makeMove(dupeBoard, computerTile, x, y)
232.        score = getScoreOfBoard(dupeBoard)[computerTile]
233.        if score > bestScore:
234.            bestMove = [x, y]
235.            bestScore = score
236.    return bestMove
```

如果没有角落的移动，循环会遍历整个列表，以找出哪个移动得分最高。第 229 行的 for 循环将把 x 和 y 设置为 possibleMoves 中的每一个移动。bestMove 是目前为止所发现的得分最高的移动，把 bestScore 设置为这个最佳移动的得分。

当循环中的代码找到得分比 bestScore 高的一个移动，第 233 行到第 235 行将把这个移动及其得分作为 bestMove 和 bestScore 中的新值存储起来。

## 16.4.12　在游戏板数据结构的副本上模拟所有可能的移动

在模拟一个移动之前，第 230 行通过调用 getBoardCopy()函数来创建游戏板数据结构的一个副本。我们想要一个可供修改的副本，从而不会改变 board 变量中真正的游戏板数据结构。

第 231 行调用 makeMove()函数，传递的参数是游戏板的副本（存储在 dupeBoard 中），而不是真正的游戏板。这模拟了所做出的移动会导致在真实的游戏板上发生什么。makeMove()函数将处理计算机的棋子的放置，并且在游戏板的副本上反转玩家的棋子。

第 232 行调用 getScoreOfBoard()函数，参数是游戏板的副本，它返回一个字典，其键是'X'和'O'，值是分数。

例如，假设 getScoreOfBoard()函数返回字典{'X':22, 'O':8}，并且 computerTile 是'X'。那么 getScoreOfBoard(dupeBoard)[computerTile]将会计算为{'X':22, 'O':8}['X']，然后求得结果为 22。如果 22 比 bestScore 大，把 bestScore 设置为 22，把 bestMove 设置为当前的 x 值和 y 值。

当这个 for 循环结束后，可以确保 bestScore 是一次移动可以得到的最高得分，并且该移动存储于变量 bestMove 中。

第 228 行首先把 bestScore 设置为-1，以便代码把检测到的第 1 次移动设置为第 1 个 bestMove。这就确保当函数返回时，possibleMoves 中有 1 个移动会被设置为 bestMove。

即使代码总是选取这些相关的移动的列表中的第 1 个移动，但是由于第 220 行把列表顺序随机打乱，所以选择也是随机的。这就确保了当有多个最佳移动时，无法预测 AI 采取哪一个移动。

## 16.4.13　在屏幕上打印分数

```
239. def showPoints(playerTile, computerTile):
240.     # Prints out the current score.
```

```
241.    scores = getScoreOfBoard(mainBoard)
242.      print('You have %s points. The computer has %s points.' %
(scores[playerTile], scores[computerTile]))
```

showPoints()函数调用 getScoreOfBoard()函数，然后打印玩家和计算机的分数。记住，getScoreOfBoard()函数所返回的字典的键是'X'和'O'，值是 X 和 O 玩家的分数。

这就是 Reversi 游戏的所有函数。从第 246 行开始的代码将要实现这个真实的游戏并且在需要的时候调用这些函数。

### 16.4.14 游戏开始

```
246. print('Welcome to Reversi!')
247.
248. while True:
249.    # Reset the board and game.
250.    mainBoard = getNewBoard()
251.    resetBoard(mainBoard)
252.    playerTile, computerTile = enterPlayerTile()
253.    showHints = False
254.    turn = whoGoesFirst()
255.    print('The ' + turn + ' will go first.')
```

第 248 行的 while 循环是主游戏循环。当一个新游戏开始时，这个程序将循环返回到第 248 行。首先，通过调用 getNewBoard()函数获取一个新的游戏板数据结构，并且通过调用 resetBoard()函数来设置起始的棋子。mainBoard 是程序的主游戏板数据结构。通过调用 enterPlayerTile()函数，让玩家输入他们想要'X'还是'O'。然后使用多变量赋值把返回值存储到 playerTile 和 computerTile 中。

showHints 是决定提示模式开还是关的一个布尔值。在第 253 行，在一开始将其设置为 False，表示关闭提示模式。

turn 变量是一个字符串，其值是'player'或'computer'。它将记录是谁的轮次。把它设置为 whoGoesFirst()函数的返回值，以随机地选择谁先走。

### 16.4.15 运行玩家的轮次

```
257.    while True:
258.       if turn == 'player':
259.          # Player's turn.
260.          if showHints:
261.                validMovesBoard = getBoardWithValidMoves(mainBoard,
playerTile)
```

```
262.                drawBoard(validMovesBoard)
263.            else:
264.                drawBoard(mainBoard)
265.            showPoints(playerTile, computerTile)
```

从第 257 行开始的 while 循环将一直循环玩家或计算机所进行的每一个轮次。当前游戏结束时，执行将跳出本次循环。

第 258 行有一条 if 语句，判断是否是玩家的轮次，其后面包含了玩家轮次的代码（从第 282 行开始的 else 语句块是计算机轮次的代码）。

首先，在屏幕上显示游戏板。如果开启了提示模式（也就是 showHints 为 True），那么在玩家可以移动的每一个有效的格子中，游戏板数据结构中都有一个'.'字符。

getBoardWithValidMoves()函数就是用来实现这个功能的。它接受一个游戏板数据结构作为参数，并且返回其包含'.'字符的一个副本。第 262 行将这个游戏板参数传递给 drawBoard()函数。

如果提示模式关闭，那么第 264 行把 mainBoard 传递给 drawBoard()函数。

当为玩家打印出游戏板之后，第 265 行调用 showPoints()函数打印出当前的分数。

```
266.            move = getPlayerMove(mainBoard, playerTile)
```

接下来，让玩家输入他们的移动。getPlayerMove()函数负责这项工作，它的返回值是包含玩家的移动的 XY 坐标的一个两元素列表。getPlayerMove()函数已经确保了玩家输入的移动是一个有效的移动。

### 16.4.16　处理退出或提示的命令

```
267.            if move == 'quit':
268.                print('Thanks for playing!')
269.                sys.exit() # terminate the program
270.            elif move == 'hints':
271.                showHints = not showHints
272.                continue
273.            else:
274.                makeMove(mainBoard, playerTile, move[0], move[1])
```

如果玩家针对移动输入字符串'quit'，那么 getPlayerMove()函数将返回字符串'quit'。在这种情况下，第 269 行将调用 sys.exit()来终止程序。

如果玩家针对移动输入字符串'hints'，那么 getPlayerMove()函数将返回字符串'hints'。在这种情况下，我们将切换为提示模式（如果是关闭状态）或关闭模式（如果是开启状态）。

第 271 行的赋值语句 showHints = not showHints 处理这两种情况，因为 not False 的结果是 True，not True 的结果是 False。然后，continue 语句将执行移动到循环的开始处（变

量 turn 还没有修改，所以仍然是玩家的轮次）。

否则，如果玩家没有选择退出或者切换提示模式，第 274 行调用 makeMove()函数在游戏板上进行玩家的移动。

## 16.4.17 进行玩家的移动

```
276.           if getValidMoves(mainBoard, computerTile) == []:
277.               break
278.           else:
279.               turn = 'computer'
```

在进行了玩家的移动之后，第 276 行调用 getValidMoves()函数来判断计算机能否进行任何移动。如果 getValidMoves()函数返回一个空白列表，那么就没有可供计算机进行的有效移动。在这种情况下，第 277 行跳出 while 循环并且结束游戏。

否则，第 279 行把变量 turn 设置为'computer'。随后的执行跳过 else 语句块，并且到达 while 语句块的末尾，所以执行跳转到第 257 行的 while 语句。然而，这一次将是计算机的轮次。

## 16.4.18 运行计算机的轮次

```
281.       else:
282.           # Computer's turn.
283.           drawBoard(mainBoard)
284.           showPoints(playerTile, computerTile)
285.           input('Press Enter to see the computer\'s move.')
286.           x, y = getComputerMove(mainBoard, computerTile)
287.           makeMove(mainBoard, computerTile, x, y)
```

当用 drawBoard()函数打印输出游戏板之后，第 284 行调用 showPoints()函数来打印当前的分数。

第 285 行调用 input()函数来暂停脚本，以便玩家可以查看游戏板。这和本书第 5 章使用 input()函数来暂停程序的做法很相似。我们并没有在调用 input()函数之前调用 print()函数来打印一个字符串，而是通过给 input()函数传递一个要打印的字符串来做相同的事。

玩家看到游戏板并按下回车键之后，第 286 行调用 getComputerMove()函数来获取计算机的下一步移动的 XY 坐标。使用多变量赋值将这些坐标存储到变量 x 和 y 中。

最后，把 x 和 y 以及游戏板数据结构和计算机的棋子传递给 makeMove()函数。这会把计算机的棋子放在 mainBoard 中的游戏板上，以反映出计算机的移动。第 286 行调用 getComputerMove()函数，获取了计算机的移动（并且把它存储到变量 x 和 y 中）。第 287

行调用 makeMove()函数在游戏板上做出移动。

```
289.            if getValidMoves(mainBoard, playerTile) == []:
290.                break
291.            else:
292.                turn = 'player'
```

第 289 行到第 292 行的代码和第 276 行到第 279 行的代码类似。在计算机做出移动之后，第 289 行判断是否存在人类玩家可以进行的任何有效移动。如果 getValidMoves()函数返回一个空的列表，那么表示没有有效移动。这意味着游戏结束了，并且第 290 行跳出了 while 循环。

否则，玩家应该至少有一种可能的移动。变量 turn 是'player'。在第 292 行之后，while 语句块中没有其他的代码了，所以执行返回到第 257 行的 while 语句。

## 16.4.19　在屏幕上绘制所有内容

```
294.    # Display the final score.
295.    drawBoard(mainBoard)
296.    scores = getScoreOfBoard(mainBoard)
297.    print('X scored %s points. O scored %s points.' % (scores['X'],
scores['O']))
298.    if scores[playerTile] > scores[computerTile]:
299.        print('You beat the computer by %s points! Congratulations!' %
(scores[playerTile] - scores[computerTile]))
300.    elif scores[playerTile] < scores[computerTile]:
301.            print('You lost. The computer beat you by %s points.' %
(scores[computerTile] - scores[playerTile]))
302.    else:
303.        print('The game was a tie!')
```

第 294 行是从第 257 行开始的 while 语句块之外的第 1 行代码。当执行从第 290 行或第 277 行跳出 while 循环时，将会执行这里的代码。至此，游戏结束。现在，程序应该打印游戏板和分数，并且决定谁赢得了游戏。

getScoreOfBoard()函数将返回一个字典，其键为'X'和'O'，其值为两个玩家的分数。通过判断玩家的分数是大于、小于或等于计算机的分数，就可以知道玩家是赢了、输了还是平局。

## 16.4.20　询问玩家是否再玩一局

```
305.    if not playAgain():
306.        break
```

调用 playAgain()函数，如果玩家输入想要再玩一局游戏，该函数返回 True。如果 playAgain()函数返回 False，not 操作符使得 if 语句的条件为 True，执行将跳出从第 248 行开始的 while 循环。由于这个 while 语句块之后没有其他代码行了，程序就结束了。

否则，playAgain()函数返回 True（使得这个 if 语句的条件为 False），执行由此循环返回到第 248 行的 while 语句，并且创建了一个新的游戏板。

## 16.4.21 修改 drawBoard()函数

为 Reversi 游戏绘制的游戏板很大。但是，也可以修改 drawBoard()函数的代码来绘制一个小一点儿的游戏板，同时要让游戏的剩余代码保持一致。新的小一点儿的游戏板如下所示：

```
  12345678
 +--------+
1|    O   |
2|   XOX  |
3|    O   |
4| XXXXX  |
5|  .OX   |
6|  OOO   |
7| ..O..  |
8|    O   |
 +--------+
You have 8 points. The computer has 9 points.
Enter your move, or type quit to end the game, or hints to turn off/on hints.
```

如下是从第 6 行开始的新的 drawBoard()函数。你也可以从 http://invpy.com/reversi_mini.py 下载这些代码。

```
 6. def drawBoard(board):
 7.     # This function prints out the board that it was passed. Returns None.
 8.     HLINE = ' +--------+'
 9.     print(' 12345678')
10.     print(HLINE)
11.     for y in range(8):
12.         print('%s|' % (y+1), end='')
13.         for x in range(8):
14.             print(board[x][y], end='')
15.         print('|')
16.     print(HLINE)
```

## 16.5　本章小结

这个游戏的 AI 几乎无法战胜，但这并不是因为计算机很聪明。它遵循的策略很简单：如果可以，就在角落落子；否则执行将会翻转最多的棋子的移动。我们也可以做到这些，但是要算出每一种可以执行的有效移动能够翻转多少个棋子，计算的过程会很慢。而计算机进行这些计算却很简单。所以说，计算机并不比我们聪明，它只是比我们快很多！

这个游戏和 Sonar 类似，因为它也使用一个网格游戏板。它也很像 Tic Tac Toe 游戏，因为这里也有一个 AI，可以规划出最佳的移动。本章只引入了一个新的概念：在一个条件中，空的列表、空的字符串和整数 0 都计算为 False。

除此之外，这个游戏使用的都是已经介绍过的编程概念！你不必知道太多有关编程的知识，就可以创建出有趣的游戏。然而，可以使用 ASCII 字符图来对这个程序进行扩展。绘制的游戏板几乎会占据整个屏幕，并且这个游戏没有任何颜色。

在本书的后边，我们将学习如何创建带有图形和动画的游戏，而不再仅仅只有文本。我们将使用一个叫做 Pygame 的模块，它会为 Python 添加一些新的函数和特性，使得我们可以不再只是使用文本和键盘来输入。

# 第 17 章　Reversi AI 模拟

**本章的主要内容：**

- 模拟；
- 百分数；
- 饼图；
- 整数除法；
- round()函数；
- “计算机对计算机”游戏。

Reversi AI 算法很简单，但是它几乎每一次都会击败我。这是因为计算机可以快速地处理指令，由此检查游戏板上的每种可能并且选择得分最高的移动，对于计算机来说很简单。对于我们人类来说，用这种方式来找出最佳移动，却需要花很多时间。

第 15 章的 Reversi 游戏有两个函数，getPlayerMove()和 getComputerMove()，它们以[x, y]这样的两元素列表的形式返回所选中的移动。这两个函数都有相同的参数，即游戏板数据结构以及它们使用什么类型的棋子。getPlayerMove()函数通过让玩家输入坐标，来决定返回到哪个[x, y]格子的移动。getComputerMove()函数通过运行 Reversi AI 算法，来决定返回到哪个[x, y]格子的移动。

当我们用 getComputerMove()函数来替代 getPlayerMove()函数的时候，会怎么样？这样的话，玩家就不用再输入一步移动，而由计算机来代替他们做决定！也就是说，计算机和计算机下棋！

根据上一章中的 Reversi 游戏，我们将创建 3 个新的游戏：

- 通过对 reversi.py 进行修改，来创建 AISim1.py；
- 通过对 AISim1.py 进行修改，来创建 AISim2.py；
- 通过对 AISim2.py 进行修改，来创建 AISim3.py。

你可以自己输入这些修改，或者从本书的配套站点 http://invpy.com/chap16 下载代码。

## 17.1　让计算机和自己下棋

通过点击 File▶Save As，把旧的 reversi.py 文件另存为 AISim1.py，然后输入文件名 AISim1.py，并且点击 OK 按钮。这将创建 Reversi 源代码的一个副本，作为可以对其进行修改的新的文件，而原有的 Reversi 游戏则保持不变（你可能还想玩它）。把 AISim1.py 中的如下代码：

```
266. move = getPlayerMove(mainBoard, playerTile)
```

修改为（修改部分用粗体表示）：

```
266. move = getComputerMove(mainBoard, playerTile)
```

现在运行程序。注意，游戏仍然会询问你想要 X 还是 O，但是它不会要求你输入任何移动。当替换了 getPlayerMove()函数，就不再需要调用接受玩家输入的任何代码了。在最初计算机的移动之后，仍然按下回车键（因为第 285 行的代码 input('Press Enter to see the computer\'s move.')），但是游戏可以自己玩！

让我们对 AISim1.py 做一些其他的改动。我们为 Reversi 定义的所有函数都可以保持不变。但是替换了程序的整个主体部分（从第 246 行开始），如下面的代码所示。有一些代码保留了，但是更改了大部分代码。而在第 246 行之前所有的代码与第 16 章中的 Reversi 游戏的代码是一致的。你也可以通过从 http://invpy.com/chap16 下载源代码，以避免自己输入代码。

如果输入这些代码后出现错误，请使用 http://invpy.com/diff/AISim1 上的在线 diff 工具，把你的代码与书中的代码进行比较。

AISim1.py

```
246. print('Welcome to Reversi!')
247.
248. while True:
249.     # Reset the board and game.
250.     mainBoard = getNewBoard()
251.     resetBoard(mainBoard)
252.     if whoGoesFirst() == 'player':
253.         turn = 'X'
254.     else:
255.         turn = 'O'
256.     print('The ' + turn + ' will go first.')
257.
258.     while True:
259.         drawBoard(mainBoard)
260.         scores = getScoreOfBoard(mainBoard)
261.           print('X has %s points. O has %s points' % (scores['X'],
scores['O']))
262.         input('Press Enter to continue.')
263.
264.         if turn == 'X':
265.             # X's turn.
266.             otherTile = 'O'
267.             x, y = getComputerMove(mainBoard, 'X')
```

```
268.             makeMove(mainBoard, 'X', x, y)
269.         else:
270.             # O's turn.
271.             otherTile = 'X'
272.             x, y = getComputerMove(mainBoard, 'O')
273.             makeMove(mainBoard, 'O', x, y)
274.
275.         if getValidMoves(mainBoard, otherTile) == []:
276.             break
277.         else:
278.             turn = otherTile
279.
280.     # Display the final score.
281.     drawBoard(mainBoard)
282.     scores = getScoreOfBoard(mainBoard)
283.     print('X scored %s points. O scored %s points.' % (scores['X'],
scores['O']))
284.
285.     if not playAgain():
286.         sys.exit()
```

### 17.1.1 AISim1.py 代码如何工作

AISim1.py 程序与最初的 Reversi 程序一样，只不过把对 getPlayerMove()函数的调用改为对 getComputerMove()函数的调用。此外，还对显示到屏幕上的文本做了一些修改，以便能够更容易跟上游戏的运行。

当运行 AISim1.py 程序时，我们所要做的就是针对每一个轮次按下回车键，直到游戏结束。运行几次游戏，观察一下计算机和自己下棋。因为 X 玩家和 O 玩家都使用了相同的算法，所以谁获得胜利，真的只是靠运气。X 玩家有一半的机会获胜，O 玩家也有一半的机会获胜。

### 17.1.2 让计算机和自己下几盘棋

但是，如果我们创建一个新的算法，会怎么样呢？然后，我们可以设置这个新的 AI 算法与 getComputerMove()函数中实现的算法进行对战，看看哪一方会获胜。让我们对源代码做一些修改。进行如下的修改，以得到 AISim2.py：

1．点击 File▶Save As。

2．把这个文件另存为 *AISim2.py*，以便所做的修改不会影响到 AISim1.py（此时，AISim1.py 和 AISim2.py 将具有相同的代码）。

3．对 AISim2.py 进行修改并保存该文件（AISim2.py 将有新的变化，而 AISim1.py 还是最初的未改变的代码）。

添加如下代码。添加的行用粗体表示，并删除了一些行。当完成了文件修改之后，把它另存为 AISim2.py。

如果觉得容易搞混淆，可以从本书的配套网址 http://invpy.com/chap16 下载 AISim2.py 的源代码。

### 17.1.3　AISim2.py

如果输入这些代码后出现错误，请使用 http://invpy.com/diff/AISim2 上的在线 diff 工具，把你的代码与书中的代码进行比较。

AISim2.py

```
246. print('Welcome to Reversi!')
247.
248. xwins = 0
249. owins = 0
250. ties = 0
251. numGames = int(input('Enter number of games to run: '))
252.
253. for game in range(numGames):
254.     print('Game #%s:' % (game), end=' ')
255.     # Reset the board and game.
256.     mainBoard = getNewBoard()
257.     resetBoard(mainBoard)
258.     if whoGoesFirst() == 'player':
259.         turn = 'X'
260.     else:
261.         turn = 'O'
262.
263.     while True:
264.         if turn == 'X':
265.             # X's turn.
266.             otherTile = 'O'
267.             x, y = getComputerMove(mainBoard, 'X')
268.             makeMove(mainBoard, 'X', x, y)
269.         else:
270.             # O's turn.
271.             otherTile = 'X'
272.             x, y = getComputerMove(mainBoard, 'O')
273.             makeMove(mainBoard, 'O', x, y)
274.
275.         if getValidMoves(mainBoard, otherTile) == []:
276.             break
277.         else:
```

```
278.           turn = otherTile
279.
280.     # Display the final score.
281.     scores = getScoreOfBoard(mainBoard)
282.     print('X scored %s points. O scored %s points.' % (scores['X'],
scores['O']))
283.
284.     if scores['X'] > scores['O']:
285.         xwins += 1
286.     elif scores['X'] < scores['O']:
287.         owins += 1
288.     else:
289.         ties += 1
290.
291. numGames = float(numGames)
292. xpercent = round(((xwins / numGames) * 100), 2)
293. opercent = round(((owins / numGames) * 100), 2)
294. tiepercent = round(((ties / numGames) * 100), 2)
295. print('X wins %s games (%s%%), O wins %s games (%s%%), ties for %s games
(%s%%) of %s games total.' % (xwins, xpercent, owins, opercent, ties,
tiepercent, numGames))
```

### 17.1.4 AISim2.py 代码如何工作

我们在第 248 行到第 250 行添加了变量 xwins、owins 和 ties，用来记录 X 玩家赢的次数、O 玩家赢的次数和平局的次数。第 284 行到第 289 行在每一次游戏的最后，在循环返回开始一个新的游戏之前，增加了这些变量的值。

我们删除了程序中的大部分 print()函数调用，以及对 drawBoard()函数的调用。当运行 AISim2.py 时，它会问你想要运行多少次游戏。现在，我们已经去掉了对 drawBoard()函数的调用，用“for game in range(numGames):”循环代替了“while True:”循环，可以多次运行游戏而无需停下来等待用户输入。下面是运行了计算机对计算机的 Reversi 游戏 10 次的一个示例：

```
Welcome to Reversi!
Enter number of games to run: 10
Game #0: X scored 40 points. O scored 23 points.
Game #1: X scored 24 points. O scored 39 points.
Game #2: X scored 31 points. O scored 30 points.
Game #3: X scored 41 points. O scored 23 points.
Game #4: X scored 30 points. O scored 34 points.
Game #5: X scored 37 points. O scored 27 points.
Game #6: X scored 29 points. O scored 33 points.
```

```
Game #7: X scored 31 points. O scored 33 points.
Game #8: X scored 32 points. O scored 32 points.
Game #9: X scored 41 points. O scored 22 points.
X wins 5 games (50.0%), O wins 4 games (40.0%), ties for 1 games (10.0%) of
10.0 games total.
```

因为该算法包含了随机性，所以当你运行的时候不会得到和上面完全一样的数字。

在屏幕上打印内容比屏幕下滚的速度慢，但是现在我们删除了打印部分的代码，计算机可以在一两秒钟内运行整个 Reversi 游戏。想一想。每次程序都会打印出一行的最终得分，它都将整个游戏运行了一次（大概有 50 到 60 步移动，每步移动都会仔细检查，以得到最高得分）。

## 17.2　百分数

百分数表示总量的一部分，其范围从 0%到 100%。如果有一个 100%的饼，就是一个完整的饼。如果有一个 0%的饼，就是彻底没有饼。50%的饼是半个饼。饼是用来表示百分数的常见图形。实际上，有一种叫做饼图（pie chart）的图形，它可以显示特定部分占总量的多少。图 17-1 是包含了 10%、15%、25%和 50%的一个饼图。注意，10% + 15% + 25% + 50%加起来是 100%，也就是一个完整的饼。

我们可以用除法来计算百分数。要得到百分数，用已有部分除以总数，然后乘以 100。例如，如果 X 玩家赢了 100 局游戏中的 50 局，我们将计算表达式 50/100，结果是 0.5。将其乘以 100 得到一个百分数（也就是 50%）。

注意，如果 X 玩家在 200 局游戏中赢得了 100 局，可以用 100/200 来计算百分数，也将得到 0.5。当将其乘以 100 得到百分数时，会得到 50%。200 局游戏中赢得其中的 100 局，与 100 局游戏中赢得 50 局,都是相同的百分数（也就是相同份额）。

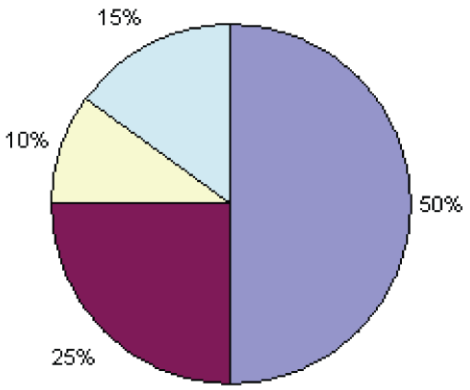



图 17-1　10%、15%、25%和 50%的一个饼图

### 除法的计算结果是浮点数

特别需要注意的是，当使用除法操作符（/）时，表达式将总是得到一个浮点数结果。例如，表达式 10/2 将得到浮点数 5.0，而不是整数 5。

记住这一点特别重要，因为用加法操作符（+）把一个整数和一个浮点数相加，总是会得到一个浮点数。例如，3+4.0 的结果是浮点数 7.0，而不是整数 7。

尝试在交互式 shell 中输入如下代码：

```
>>> spam = 100 / 4
>>> spam
25.0
>>> spam = spam + 20
>>> spam
45.0
```

注意，在上面的示例中，存储在变量 spam 中的值的数据类型总是浮点数。我们把浮点值传递给 int()函数，函数将返回该浮点值的整数形式。但这总是会舍去小数点后的值。例如，表达式 int(4.0)、int(4.2)和 int(4.9)的结果都是 4，而不是 5。

## 17.3 round()函数

round()函数把一个浮点数四舍五入为最近的浮点数。尝试在交互式 shell 中输入如下代码：

```
>>> round(10.0)
10.0
>>> round(10.2)
10.0
>>> round(8.7)
9.0
>>> round(3.4999)
3.0
>>> round(2.5422, 2)
2.54
```

round()函数也有一个可选的参数，我们可以指定想要四舍五入的位数。例如，表达式 round(2.5422, 2)的结果是 2.54，而表达式 round(2.5422, 3)的结果是 2.542。

### 显示统计结果

```
291. numGames = float(numGames)
292. xpercent = round(((xwins / numGames) * 100), 2)
293. opercent = round(((owins / numGames) * 100), 2)
294. tiepercent = round(((ties / numGames) * 100), 2)
295. print('X wins %s games (%s%%), O wins %s games (%s%%), ties for %s games
(%s%%) of %s games total.' % (xwins, xpercent, owins, opercent, ties,
tiepercent, numGames))
```

程序末尾的代码将向用户显示 X 玩家和 O 玩家分别赢了多少局、平了多少局以及这些数字所占的百分比。从统计学来看，玩游戏的次数越多，百分数也就越精确，也越可能找到最佳的 AI 算法。如果只运行 10 局游戏，X 赢了 3 局，那么看上去 X 的算法只有 30%的获胜机会。然而，如果运行了 100 次甚至 1000 次游戏，就会发现 X 的算法赢得了将近 50% （也就是一半）的游戏。

用赢得次数或平局次数除以总的游戏次数，来得到百分数。然后将其结果乘以 100。然而，最终得到的数字可能像 66.66666666666667 这样。所以把这个数字传递给 round()函数，并且使用 2 作为该函数的第 2 个参数，从而得到把精度限制为两位的小数，所以它将返回了 66.67 这样的一个浮点数（这更具可读性）。

我们再做一次试验。再次运行 AISim2.py，但是这次运行 100 局游戏。

## 17.4　AISim2.py 的运行示例

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 42 points. O scored 18 points.
Game #1: X scored 26 points. O scored 37 points.
Game #2: X scored 34 points. O scored 29 points.
Game #3: X scored 40 points. O scored 24 points.
...skipped for brevity...
Game #96: X scored 22 points. O scored 39 points.
Game #97: X scored 38 points. O scored 26 points.
Game #98: X scored 35 points. O scored 28 points.
Game #99: X scored 24 points. O scored 40 points.
X wins 46 games (46.0%), O wins 52 games (52.0%), ties for 2 games (2.0%)
of 100.0 games total.
```

这次运行可能会花费几分钟，具体取决于你的计算机运行的有多快。我们可以看到 100 局游戏的结果大约仍然是 50 比 50，因为 X 玩家和 O 玩家都使用了相同的算法。

## 17.5　比较不同的 AI 算法

让我们为新的算法添加一些新的函数。但是，首先点击 File▶Save As，并且把这个文件另存为 AISim3.py。把这些函数添加到如下的代码清单中，放在 print('Welcome to Reversi!')代码行之前。

### 17.5.1　AISim3.py

如果输入这些代码后出现错误，使用 http://invpy.com/diff/AISim3 上的在线 diff 工具，

把你的代码与书中的代码进行比较。

AISim3.py

```
245. def getRandomMove(board, tile):
246.     # Return a random move.
247.     return random.choice( getValidMoves(board, tile) )
248.
249.
250. def isOnSide(x, y):
251.     return x == 0 or x == 7 or y == 0 or y ==7
252.
253.
254. def getCornerSideBestMove(board, tile):
255.     # Return a corner move, or a side move, or the best move.
256.     possibleMoves = getValidMoves(board, tile)
257.
258.     # randomize the order of the possible moves
259.     random.shuffle(possibleMoves)
260.
261.     # always go for a corner if available.
262.     for x, y in possibleMoves:
263.         if isOnCorner(x, y):
264.             return [x, y]
265.
266.     # if there is no corner, return a side move.
267.     for x, y in possibleMoves:
268.         if isOnSide(x, y):
269.             return [x, y]
270.
271.     return getComputerMove(board, tile)
272.
273.
274. def getSideBestMove(board, tile):
275.     # Return a corner move, or a side move, or the best move.
276.     possibleMoves = getValidMoves(board, tile)
277.
278.     # randomize the order of the possible moves
279.     random.shuffle(possibleMoves)
280.
281.     # return a side move, if available
282.     for x, y in possibleMoves:
283.         if isOnSide(x, y):
284.             return [x, y]
285.
286.     return getComputerMove(board, tile)
```

```
287.
288.
289. def getWorstMove(board, tile):
290.     # Return the move that flips the least number of tiles.
291.     possibleMoves = getValidMoves(board, tile)
292.
293.     # randomize the order of the possible moves
294.     random.shuffle(possibleMoves)
295.
296.     # Go through all the possible moves and remember the best scoring
move
297.     worstScore = 64
298.     for x, y in possibleMoves:
299.         dupeBoard = getBoardCopy(board)
300.         makeMove(dupeBoard, tile, x, y)
301.         score = getScoreOfBoard(dupeBoard)[tile]
302.         if score < worstScore:
303.             worstMove = [x, y]
304.             worstScore = score
305.
306.     return worstMove
307.
308.
309. def getCornerWorstMove(board, tile):
310.     # Return a corner, a space, or the move that flips the least number
of tiles.
311.     possibleMoves = getValidMoves(board, tile)
312.
313.     # randomize the order of the possible moves
314.     random.shuffle(possibleMoves)
315.
316.     # always go for a corner if available.
317.     for x, y in possibleMoves:
318.         if isOnCorner(x, y):
319.             return [x, y]
320.
321.     return getWorstMove(board, tile)
322.
323.
324.
325. print('Welcome to Reversi!')
```

## 17.5.2　AISim3.py 代码如何工作

这些函数中的很多都是彼此相似的，并且有一些函数使用了新的 isOnSide()函数。下面

来看一下所创建的新的算法，参见表 17-1。

表 17-1 Reversi AI 用到的函数

| 函数 | 描述 |
|---|---|
| getRandomMove() | 随机选取一个有效的移动 |
| getCornerSideBestMove() | 如果可能的话，得到一个角落移动。如果没有角落，得到一个在边上的格子。如果没有边上的格子可以用，使用常规的 getComputerMove()算法 |
| getSideBestMove() | 如果可能的话，得到一个在边上的格子。如果没有，那么使用常规的 getComputerMove()算法。这意味着，在选择角落格子之前，先选择边上的格子 |
| getWorstMove() | 得到的格子将导致最少的棋子被翻转 |
| getCornerWorstMove() | 如果可能的话，得到一个角落的格子。如果不可以，使用 getWorstMove()算法 |

## 17.5.3 比较随机算法和常规算法

现在唯一要做的事,就是用一个新的函数来替代程序的主体部分中的一个 getComputerMove()函数调用。然后可以运行游戏几次，看看一种算法能赢另一种算法多少次。首先，我们在第 351 行用 getRandomMove()函数代替 O 玩家的算法中的 getComputerMove()函数。

```
351.          x, y = getRandomMove(mainBoard, 'O')
```

现在，当我们运行 100 次游戏时，它看上去如下所示：

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 25 points. O scored 38 points.
Game #1: X scored 32 points. O scored 32 points.
Game #2: X scored 15 points. O scored 0 points.

...skipped for brevity...

Game #97: X scored 41 points. O scored 23 points.
Game #98: X scored 33 points. O scored 31 points.
Game #99: X scored 45 points. O scored 19 points.
X wins 84 games (84.0%), O wins 15 games (15.0%), ties for 1 games (1.0%)
of 100.0 games total.
```

喔！X 玩家赢得要比 O 玩家多。这意味着 getComputerMove()函数（移动尽可能地在角落上，否则，移动落在能够翻转最多棋子的格子上）中的算法要比 getRandomMove()函

数（移动随机放置在格子上）中的算法赢得更多次的游戏。这是合理的，因为做出明智的选择通常要比做出随机的选择更好。

### 17.5.4　比较用随机算法对弈

如果我们把 X 玩家的算法也改为使用 getRandomMove()函数中的算法，结果会怎样？我们把第 346 行的 getComputerMove()函数改为 getRandomMove()函数，并且再次运行程序。

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 37 points. O scored 24 points.
Game #1: X scored 19 points. O scored 45 points.

...skipped for brevity...

Game #98: X scored 27 points. O scored 37 points.
Game #99: X scored 38 points. O scored 22 points.
X wins 42 games (42.0%), O wins 54 games (54.0%), ties for 4 games (4.0%)
of 100.0 games total.
```

我们看到，当两个玩家都使用随机移动时，他们中的每一方会赢得将近 50%的胜利（在上面的示例中，O 玩家更幸运一些，赢得了超过一半次数的胜利）。

因为放在角落格子上的移动，不会被翻转，所以这样的移动是一个好主意，而在边上的格子中的移动也是一个好主意。在后一种情况下，棋子在棋盘的边上，而不像是其他棋子一样在开阔的空地上。角落的格子仍然要优先于边上的格子，但是在边上的移动（即使有一个移动可以翻转更多的棋子）可能也是一种很好的策略。

### 17.5.5　比较常规算法和角边优先算法

在第 346 行，把 X 的算法改为使用 getComputerMove()函数（最初的算法），在第 351 行，把 O 玩家的算法改为使用 getCornerSideBestMove()函数（该算法先尝试在角落上移动，然后尝试在一个边上的格子上移动，然后从剩下的移动中选取最佳的），让我们运行 100 次程序，看哪个算法更好。尝试修改这个函数的调用，并且再次运行程序。

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 52 points. O scored 12 points.
Game #1: X scored 10 points. O scored 54 points.
```

```
...skipped for brevity...

Game #98: X scored 41 points. O scored 23 points.
Game #99: X scored 46 points. O scored 13 points.
X wins 65 games (65.0%), O wins 31 games (31.0%), ties for 4 games (4.0%)
of 100.0 games total.
```

喔！出乎意料。看上去，选择边上的格子而不选择翻转更多棋子的格子，似乎是一个糟糕的策略。在边上的格子中移动所带来的好处，还没有少反转几个对手的棋子所付出的代价大。能确认这些结果正确吗？让我们再来运行程序，这次我们运行游戏 1000 次。这会花费几分钟来运行（但是要是手工计算的话，需要几个星期！）。尝试修改函数调用，并且再次运行程序。

```
Welcome to Reversi!
Enter number of games to run: 1000
Game #0: X scored 20 points. O scored 44 points.
Game #1: X scored 54 points. O scored 9 points.

...skipped for brevity...

Game #998: X scored 38 points. O scored 23 points.
Game #999: X scored 38 points. O scored 26 points.
X wins 611 games (61.1%), O wins 363 games (36.3%), ties for 26 games (2.6%)
of 1000.0 games total.
```

从 1000 局游戏运行中得到了更准确的统计结果，这与 100 局游戏运行的统计结果是相同的。看上去，选择翻转最多棋子的移动要比在边上的格子的移动要更好一些。

### 17.5.6 比较常规算法和最糟糕算法

现在，在第 346 行，将 X 玩家的算法设置为使用 getComputerMove()函数，在第 351 行，将 O 玩家的算法设置为使用 getWorstMove()函数（该算法选择翻转最少的棋子的移动），并且运行 100 次游戏。尝试修改函数的调用，并再次运行程序。

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 50 points. O scored 14 points.
Game #1: X scored 38 points. O scored 8 points.
...skipped for brevity...
Game #98: X scored 36 points. O scored 16 points.
Game #99: X scored 19 points. O scored 0 points.
X wins 98 games (98.0%), O wins 2 games (2.0%), ties for 0 games (0.0%) of
100.0 games total.
```

喔！getWorstMove()中的算法，也就是总是选择翻转最少的棋子的移动，几乎总是会输给常规算法。这根本就不奇怪（实际上，让人奇怪的是，这种策略居然都能赢得 2%的胜利！）。

### 17.5.7　比较常规算法和 WorstCorner 算法

当我们在第 351 行用 getCornerWorstMove()函数代替的 getWorstMove()函数时，会怎么样呢？算法是相同的，只不过在采取最糟糕的移动之前，先在任何可能的角落做移动。尝试修改函数调用，并再次运行这个程序。

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 36 points. O scored 7 points.
Game #1: X scored 44 points. O scored 19 points.

...skipped for brevity...

Game #98: X scored 47 points. O scored 17 points.
Game #99: X scored 36 points. O scored 18 points.
X wins 94 games (94.0%), O wins 6 games (6.0%), ties for 0 games (0.0%) of
100.0 games total.
```

getCornerWorstMove()仍然输掉了大部分的游戏，但是它好像比 getWorstMove()函数赢得了更多一些的游戏（6%比 2%）。当有角落可供移动时，在角落进行的移动，是否真的会有所不同？

### 17.5.8　比较最糟糕的算法和 WorstCorner 算法

我们可以把 X 玩家的算法设置为 getWorstMove()，把 O 玩家的算法设置为 getCornerWorstMove()，然后运行这个程序。尝试修改函数调用，并再次运行这个函数。

```
Welcome to Reversi!
Enter number of games to run: 100
Game #0: X scored 25 points. O scored 39 points.
Game #1: X scored 26 points. O scored 33 points.

...skipped for brevity...

Game #98: X scored 36 points. O scored 25 points.
Game #99: X scored 29 points. O scored 35 points.
X wins 32 games (32.0%), O wins 67 games (67.0%), ties for 1 games (1.0%)
of 100.0 games total.
```

没错，即使都采用了最糟糕的移动，似乎采用角落移动也会取得更多的胜利。既然已经发现在边上的移动会让我们输得更多，那么，在角落的移动总是一个好主意。

## 17.6 本章小结

本章并不是真正地介绍了一个游戏，而是模拟 Reversi 的各种策略。如果我们认为在 Reversi 游戏中靠边落子是一个好主意的话，我们必须用几个星期甚至是几个月的时间，认真地手工来玩 Reversi 游戏，并且把结果都记录下来。但是，如果知道如何通过编程让计算机来玩 Reversi 游戏，那么我们可以让计算机使用这些策略来玩 Reversi 游戏。如果想到了这一点，你就会意识到，计算机执行几百万行 Python 程序代码只要几秒钟！模拟 Reversi 的体验，对于在真实世界中玩 Reversi 游戏会有所帮助。

实际上，本章会创建一个很好的、科学公平的项目。你的问题可能是，与其他组移动相比，哪一组的移动会获取最多的胜利，并且针对哪一个是最好的策略来提出一个假设。当进行了多次模拟之后，你就可以判断哪个策略可以工作的最好。通过编程，我们就可以模拟任何棋盘游戏，创建一个科学公平的项目！这都是因为我们知道如何命令计算机按部就班、一行一行地去实现它。我们可以说计算机能够理解的语言，并且让它为我们处理大量的数据和数字。

本书中所有的基于文本的游戏到此为止。只使用文本的游戏可能也很好玩，即使它们很简单。但是，大部分现代游戏使用图形、声音和动画，这会使得游戏看上去更有趣。在本书的剩下各章中，我们将学习如何使用一个叫做 Pygame 的 Python 模块，来创建带有图形的游戏。

# 第 18 章　图形和动画

**本章的主要内容：**

- 安装 Pygame；
- Pygame 中的颜色和字体；
- 锯齿图像和抗锯齿图像；
- 属性；
- 数据类型 pygame.font.Font、pygame.Surface、pygame.Rect 和 pygame.PixelArray；
- 构造函数；
- Pygame 的绘制函数；
- Surface 对象的 blit()方法；
- 事件；
- 动画。

到目前为止，我们所有的游戏都只用到了文本。屏幕上显示的文本是输出，玩家通过键盘输入的文本是输入。只使用文本会使得编程更容易学习。但是，在本章中，通过使用 Pygame 模块，我们将创建一些带有图形和声音的、更有趣的高级游戏。

第 18 章、第 19 章和第 20 章会介绍如何使用 Pygame 来生成带有图形、动画、鼠标输入和声音的游戏。在这些章中，我们将编写一些简单程序的源代码，它们不是游戏但却用来展示所学习的 Pygame 概念。第 21 章将使用所有这些概念来创建一个游戏。

## 18.1　安装 Pygame

Pygame 并不是 Python 所附带的。和 Python 一样，Pygame 也是可以免费下载的。在 Web 浏览器中，访问 http://invpy.org/downloadpygame，下载适合你的操作系统和 Python 版本的 Pygame 安装程序。

在下载了安装程序之后，运行安装程序，并且遵照指令直到完成了 Pygame 的安装。要检查 Pygame 是否安装正确，在交互式 shell 中输入如下内容：

```
>>> import pygame
```

如果按下回车键后什么都没有出现，那么我们知道已经成功安装了 Pygame。如果出现了 ImportError：No module named pygame 错误，那么，尝试重新安装 Pygame（并且确保正确地输入了 import pygame）。

Pygame 网站 http://pygame.org，介绍了如何使用 Pygame，还介绍了一些使用 Pygame

创建的游戏。Pygame 网站如图 18-1 所示。

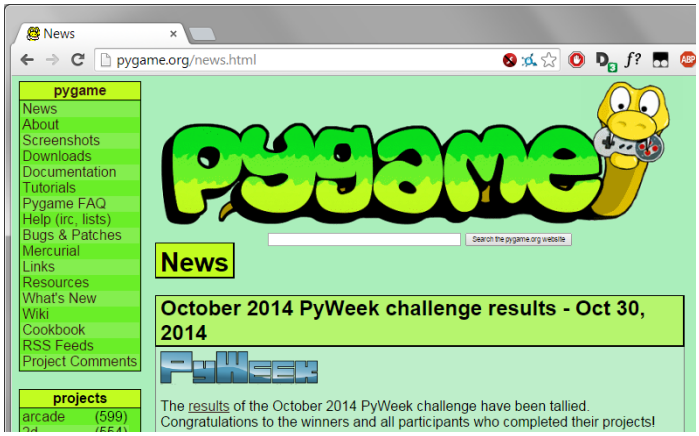



图 18-1 Pygame.org 网站

## 18.2 Pygame 中的 Hello World

第一个 Pygame 程序是一个新的“Hello World!”程序，就像我们在本书开始所创建的程序一样。这次，我们将使用 Pygame 来制作一个在图形窗口中显示的“Hello world!”，而不再只是显示文本。

Pygame 不能在交互式 shell 中工作。因此，我们只能编写 Pygame 程序，并且不能通过交互式 shell 每次给 Pygame 发送一条指令。

Pygame 程序也不使用 input()函数。这里没有文本输入和输出。相反，程序通过把图形和文本绘制到窗口，在一个窗口中显示输出内容。通过调用事件，Pygame 程序接收来自键盘和鼠标的输入。我们将在下一章介绍事件。

## 18.3 Hello World 的源代码

在文件编辑器中输入如下代码，并且把它保存为 pygameHelloWorld.py。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/pygameHelloWorld 上的在线 diff 工具，把你的代码与书中的代码进行比较。

pygameHelloWorld.py

```
1.importpygame, sys
2.frompygame.locals import *
3.
```

```
 4.# set up pygame
 5.pygame.init()
 6.
 7.# set up the window
 8.windowSurface = pygame.display.set_mode((500, 400), 0, 32)
 9.pygame.display.set_caption('Hello world!')
10.
11.# set up the colors
12. BLACK = (0, 0, 0)
13. WHITE = (255, 255, 255)
14. RED = (255, 0, 0)
15. GREEN = (0, 255, 0)
16. BLUE = (0, 0, 255)
17.
18.# set up fonts
19. basicFont = pygame.font.SysFont(None, 48)
20.
21.# set up the text
22.text = basicFont.render('Hello world!', True, WHITE, BLUE)
23.textRect = text.get_rect()
24.textRect.centerx = windowSurface.get_rect().centerx
25.textRect.centery = windowSurface.get_rect().centery
26.
27.# draw the white background onto the surface
28.windowSurface.fill(WHITE)
29.
30.# draw a green polygon onto the surface
31.pygame.draw.polygon(windowSurface, GREEN, ((146, 0), (291, 106), (236,
277), (56, 277), (0, 106)))
32.
33.# draw some blue lines onto the surface
34.pygame.draw.line(windowSurface, BLUE, (60, 60), (120, 60), 4)
35.pygame.draw.line(windowSurface, BLUE, (120, 60), (60, 120))
36.pygame.draw.line(windowSurface, BLUE, (60, 120), (120, 120), 4)
37.
38.# draw a blue circle onto the surface
39.pygame.draw.circle(windowSurface, BLUE, (300, 50), 20, 0)
40.
41.# draw a red ellipse onto the surface
42.pygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
43.
44.# draw the text's background rectangle onto the surface
45.pygame.draw.rect(windowSurface, RED, (textRect.left - 20, textRect.top
- 20, textRect.width + 40, textRect.height + 40))
```

```
46.
47.# get a pixel array of the surface
48.pixArray = pygame.PixelArray(windowSurface)
49.pixArray[480][380] = BLACK
50.delpixArray
51.
52.# draw the text onto the surface
53.windowSurface.blit(text, textRect)
54.
55.# draw the window onto the screen
56.pygame.display.update()
57.
58.# run the game loop
59.while True:
60.    for event in pygame.event.get():
61.        if event.type == QUIT:
62.            pygame.quit()
63.            sys.exit()
```

## 18.4　运行 Hello World 程序

当运行这个程序时，我们应该看到一个新的窗口出现，如图 18-2 所示。

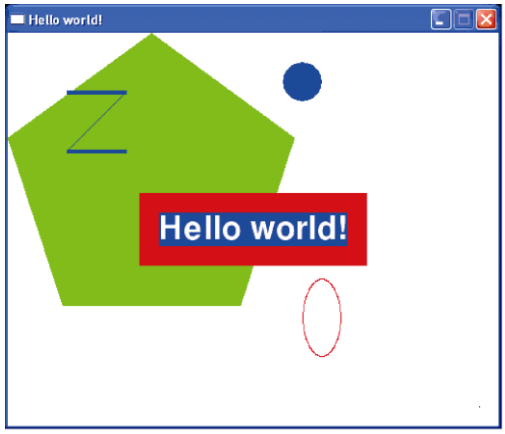



图 18-2　“Hello World”程序

使用窗口而不是控制台的好处就是，文本可以出现在窗口中的任意地方，而不是只能出现在之前已经打印的文本的后面。文本可以是任意的颜色和大小。窗口就像一块黑色的油画布，我们可以在上面绘制任何图形。

### 18.4.1　导入 Pygame 模块

我们先浏览一下这些代码，看看它们做些什么事情。

```
1.importpygame, sys
2.frompygame.locals import *
```

首先，需要导入 pygame 模块，以便可以调用 Pygame 的函数。可以在同一行中导入多个模块，模块名之间用逗号隔开。第 1 行代码导入了 pygame 和 sys 两个模块。

第 2 行代码导入了 pygame.locals 模块。这个模块包含了许多将要在 Pygame 中用到的常量，如 QUIT 或 K_ESCAPE（稍后解释其含义）。然而，使用 from moduleName import *的格式可以导入 pygame.locals 模块，并且不需要在模块的常量前面输入 pygame.locals。

如果在程序中使用的是 from sys import *导入语句，而不是 import sys 导入语句，那么在代码中要使用 exit()调用该函数，而不是使用 sys.exit()。但是，大多数时候，使用完整的函数名称会更好，因为这样可以知道函数是在哪个模块中。

### 18.4.2　pygame.init()函数

```
4.# set up pygame
5.pygame.init()
```

在导入了 pygame 模块后，并且在调用任何其他的 Pygame 函数之前，所有 Pygame 程序都必须先调用 pygame.init()函数。这项工作是 Pygame 必须的初始化步骤。

## 18.5　元组

元组（tuple）值与列表类似，只是它们使用的是圆括号而不是方括号。元组也和字符串相似，不能够修改。例如，尝试在交互式 shell 中输入如下代码：

```
>>> spam = ('Life', 'Universe', 'Everything', 42)
>>>spam[0]
'Life'
>>>spam[3]
42
>>>spam[1:3]
('Universe', 'Everything')
```

### pygame.display.set_mode()函数和 pygame.display.set_caption()函数

```
7.# set up the window
8.windowSurface = pygame.display.set_mode((500, 400), 0, 32)
9.pygame.display.set_caption('Hello world!')
```

第 8 行通过调用 pygame.display 模块中的 set_mode()方法，创建了一个 GUI 窗口（display 模块是 pygame 模块中的一个模块。即使是 pygame 模块，它也有自己的模块）。

像素是计算机屏幕上最小的点。屏幕上的一个像素可以显示成任何颜色。屏幕上所有像素一起工作，就可以显示出我们所看到的各种图像。要创建 500 个像素宽和 400 个像素高的窗口，使用元组(500, 400)作为 pygame.display.set_mode()的第 1 个参数。

set_mode()方法有 3 个参数。第 1 个参数就是包含两个整数的一个元组，这两个整数分别表示窗口的宽度和高度，以像素为单位。第 2 个参数和第 3 个参数是高级选项，超出了本书的范畴。只需要知道为它们分别传递 0 和 32 即可。

set_mode()函数返回了一个 pygame.Surface 对象（为了简单起见，我们称之为 Surface 对象）。对象（object）是对拥有方法的数据类型的值的另外一种叫法。例如，字符串就是 Python 中的对象，因为它们有数据（字符串本身）和方法（诸如 lower()和 split()）。Surface 对象表示这个窗口。

变量存储对对象的引用，就像它们存储了对列表和字典的引用一样。第 11 章介绍过引用。

## 18.6 RGB 颜色

```
11. # set up the colors
12. BLACK = (0, 0, 0)
13. WHITE = (255, 255, 255)
14. RED = (255, 0, 0)
15. GREEN = (0, 255, 0)
16. BLUE = (0, 0, 255)
```

光有 3 种主要的颜色：红色、绿色和蓝色。通过将这 3 种颜色的不同的量组合起来，就可以形成任何其他的颜色（计算机屏幕就是这么做的）。在 Pygame 中，表示颜色的数据结构是 3 个整数的元组。这些叫做 RGB 颜色（RGB Color）值。

元组中的第 1 个值，表示颜色中有多少红色。整数值 0 表示该颜色中没有红色，而 255 表示该颜色中的红色达到最大值。第 2 个值表示绿色，第 3 个值表示蓝色。这 3 个整数值构成了一个 RGB 元组。

例如，元组(0, 0, 0)表示没有红色、绿色和蓝色。最终的颜色是完全的黑色。元组(255, 255, 255)表示红色、绿色和蓝色都达到最大量，最终得到结果就是白色。

元组(255, 0, 0)表示红色达到最大量，而没有绿色和蓝色，因此最终的颜色是红色。类似的，(0, 255, 0)是绿色，(0, 0, 255)是蓝色。

可以组合红色、绿色和蓝色的量来形成其他的颜色。表 18-1 包含了许多常见的颜色和它们的 RGB 值。网页 http://invpy.com/colors 也列出了许多不同颜色的元组值。

表 18-1　颜色及其 RGB 值

| 颜色 | RGB 值 |
|---|---|
| Black | (0, 0, 0) |
| Blue | (0, 0, 255) |
| Gray | (128, 128, 128) |
| Green | (0, 128, 0) |
| Lime | (0, 255, 0) |
| Purple | (128, 0, 128) |
| Red | (255, 0, 0) |
| Teal | (0, 128, 128) |
| White | (255, 255, 255) |
| Yellow | (255, 255, 0) |

## 18.7　字体和 pygame.font.SysFont()函数

```
18. # set up fonts
19. basicFont = pygame.font.SysFont(None, 48)
```

字体是以统一风格绘制的一整套的字母、数字、符号和字符。图 18-3 展示了以不同字体打印出相同语句的效果。

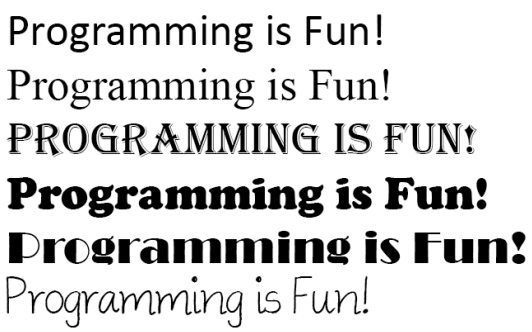



图 18-3　不同字体的示例

在之前的游戏中，我们只告诉 Python 打印文本。用于显示文本的颜色、大小和字体，则完全取决于操作系统。Python 程序不能修改字体。但是，Pygame 可以以计算机上的任何字体来绘制文本。

第 19 行通过调用 pygame.font.SysFont()函数来创建一个 pygame.font.Font 对象（简称为 Font 对象）。第 1 个参数是字体的名称，但是我们将传递 None 值以使用默认的系统字体。第 2 个参数是字体的大小（以点为单位）。

### Font 对象的 render()方法

```
21.# set up the text
22.text = basicFont.render('Hello world!', True, WHITE, BLUE)
23.textRect = text.get_rect()
```

存储在变量 basicFont 中的 Font 对象有一个叫做 render()的方法。这个方法将返回一个 Surface 对象，文本就绘制于其上。render()的第 1 个参数是要绘制的文本的字符串。第 2 个参数是指定是否想要抗锯齿的一个 Boolean 值。

第 22 行传递 True 来使用抗锯齿。抗锯齿使得文本稍微模糊，让它看起来更平滑。图 18-4 展示了放大了像素的一条使用抗锯齿的线条和一条未使用抗锯齿的线条。

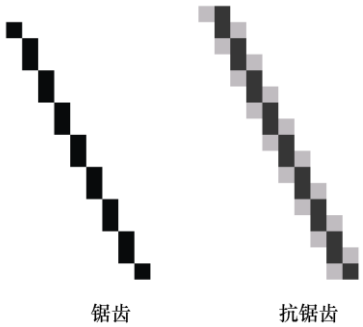



图 18-4 一条锯齿线条和一条抗锯齿线条的放大视图

## 18.8 属性

```
24.textRect.centerx = windowSurface.get_rect().centerx
25.textRect.centery = windowSurface.get_rect().centery
```

pygame.Rect 数据类型（简称为 Rect）表示特定大小和位置的矩形区域。调用函数 pygame.Rect()来创建一个新的 Rect 对象。参数是表示左上角的 XY 坐标的整数，后边跟随着宽度和高度，都是以像素为单位。

带有参数的这个函数看上去是这样的：pygame.Rect(left, top, width, height)。

就像方法是与对象相关的函数一样，属性（attribute）是与对象相关的变量。Rect 数据类型有许多属性，用来描述它所表示的矩形。表 18-2 是名为 myRect 的一个 Rect 对象的属性的列表。

表 18-2　Rect 属性

| pygame.Rect 属性 | 描述 |
|---|---|
| myRect.left | 矩形的左边的 X 坐标的整数值 |
| myRect.right | 矩形的右边的 X 坐标的整数值 |
| myRect.top | 矩形的顶部的 Y 坐标的整数值 |
| myRect.bottom | 矩形的底部的 Y 坐标的整数值 |
| myRect.centerx | 矩形的中央的 X 坐标的整数值 |
| myRect.centery | 矩形的中央的 Y 坐标的整数值 |
| myRect.width | 矩形的宽度的整数值 |
| myRect.height | 矩形的高度的整数值 |
| myRect.size | 两个整数的一个元组：(width, height) |
| myRect.topleft | 两个整数的一个元组：(left, top) |
| myRect.topright | 两个整数的一个元组：(right, top) |
| myRect.bottomleft | 两个整数的一个元组：(left, bottom) |
| myRect.bottomright | 两个整数的一个元组：(right, bottom) |
| myRect.midleft | 两个整数的一个元组：(left, centery) |
| myRect.midright | 两个整数的一个元组：(right, centery) |
| myRect.midtop | 两个整数的一个元组：(centerx, top) |
| myRect.midbottom | 两个整数的一个元组：(centerx, bottom) |

Rect 对象的好处是，如果修改了这些属性中的任意一个，所有其他属性也将自动修改。例如，如果创建了 20 个像素宽和 20 个像素高的一个 Rect 对象，其左上角位于坐标(30, 40)，那么右边的 X 坐标自动设置为 50（因为 20+30=50）。

然而，如果把 left 属性修改为 myRect.left = 100，那么 Pygame 将自动把 right 属性修改为 120（因为 20+100=120）。Rect 对象的每一个其他的属性也会相应地改变。

## pygame.font.Font 对象和 pygame.Surface 对象的 get_rect()方法

注意，（第 23 行存储于 text 变量中的）Font 对象和（第 24 行存储于 windowSurface 变量中的）Surface 对象都有一个叫做 get_rect()的方法。就技术而言，这是两个不同的方法。但是，因为它们都做同样的事情，并且分别返回表示 Font 对象或 Surface 对象的大小和位置的 Rect 对象，所以 Pygame 程序员给予它们相同的名称。

我们导入的模块是 pygame，在 pygame 模块中，有 font 模块和 surface 模块。在这些模块中是 Font 数据类型和 Surface 数据类型。Pygame 程序员使得模块的起始字母是小写的，而数据类型的起始字母是大写的。这样就更容易分辨数据类型和模块了。

# 18.9 构造函数

通过调用名为 pygame.Rect()的函数，来创建一个 pygame.Rect()对象。pygame.Rect()函数拥有和 pygame.Rect 数据类型同样的名称。我们把和某种数据类型具有相同名称并且能够创建这种数据类型的对象或值的函数，称为构造函数（constructor function）。

### Surface 对象的 fill()方法

```
27.# draw the white background onto the surface
28.windowSurface.fill(WHITE)
```

我们想要用白色来填充存储在变量 windowSurface 中的整个 Surface 对象。fill()函数将会使用传递给参数的颜色来填充整个Surface对象(在这个示例中,把WHITE变量设置为(255, 255, 255))。

关于 Pygame，需要知道的重要的一点就是，当调用 fill()方法或其他任何绘制函数时，屏幕上的窗口都不会改变。这些函数将会改变 Surface 对象，但是在调用 pygame.display.update()函数之前，不会把 Surface 对象绘制到屏幕上。

这是因为，在计算机内存中修改 Surface 对象要比在屏幕上修改图像更快。当所有绘制函数完成了对 Surface 对象的绘制之后，再在屏幕上绘制，这样要高效很多。

# 18.10 Pygame 的绘制函数

## 18.10.1 pygame.draw.polygon()函数

```
30.# draw a green polygon onto the surface
31.pygame.draw.polygon(windowSurface, GREEN, ((146, 0), (291, 106), (236,
277), (56, 277), (0, 106)))
```

多边形是由多条直线边组成的形状。圆和椭圆不是多边形。图 18-5 是多边形的一些示例。

pygame.draw.polygon()函数可以绘制你所指定的任意的多边形。参数依次是：

- 要在其上绘制多边形的 Surface 对象；
- 多边形的颜色；
- 由要依次绘制的点的 XY 坐标的元组所构成的一个元组。最后一个元组将自动连接到第一个元组以完成该形状；

图 18-5 多边形的一些示例

- 可选项，表示多边形边线条的宽度的整数值。没有这个选项的话，多边形将会填充。

第 31 行代码在 Surface 对象上绘制了一个绿色的五角星。

## 18.10.2　pygame.draw.line()函数

```
33.# draw some blue lines onto the surface
34.pygame.draw.line(windowSurface, BLUE, (60, 60), (120, 60), 4)
35.pygame.draw.line(windowSurface, BLUE, (120, 60), (60, 120))
36.pygame.draw.line(windowSurface, BLUE, (60, 120), (120, 120), 4)
```

参数依次是：

- 要在其上绘制直线的 Surface 对象；
- 直线的颜色；
- 包含直线的一端的 XY 坐标的两个整数的一个元组；
- 包含直线的另一端的 XY 坐标的两个整数的一个元组；
- 可选项，表示线条的宽度的整数值。

如果把传递给函数的 4 作为宽度，这条线的宽度将是 4 个像素的宽度。如果没有指定这个 width 参数，它将采用默认值 1。第 34 行、第 35 行和第 36 行分别调用了 3 次 pygame.draw.line() 函数在 Surface 对象上绘制一个蓝色的“Z”。

## 18.10.3　pygame.draw.circle()函数

```
38.# draw a blue circle onto the surface
39.pygame.draw.circle(windowSurface, BLUE, (300, 50), 20, 0)
```

参数依次如下：

- 要在其上绘制圆的 Surface 对象；
- 圆的颜色；
- 表示圆心的 XY 坐标的两个整数的一个元组；
- 表示圆的半径（也就是大小）的整数值；
- 可选项，表示线的宽度的一个整数值。宽度值为 0，表示填充圆。

第 39 行在 Surface 对象上绘制了一个蓝色的圆。

## 18.10.4　pygame.draw.ellipse()函数

```
41.# draw a red ellipse onto the surface
42.pygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
```

pygame.draw.ellipse()函数与 pygame.draw.circle()函数类似。参数依次如下：

- 要在其上绘制椭圆的 Surface 对象；
- 椭圆的颜色；
- 传递分别表示椭圆的左上角的 X 和 Y 坐标以及椭圆的宽和高的 4 个整数的一个元组；
- 可选项，表示宽度的整数值。宽度值为 0，表示填充椭圆。

第 42 行在 Surface 对象上绘制了一个红色的椭圆。

### 18.10.5 pygame.draw.rect()函数

```
44.# draw the text's background rectangle onto the surface
45.pygame.draw.rect(windowSurface, RED, (textRect.left - 20, textRect.top
- 20, textRect.width + 40, textRect.height + 40))
```

pygame.draw.rect()函数将要绘制一个矩形。第 3 个参数是包含了表示矩形的左上角的 X 和 Y 坐标，以及矩形的宽和高的 4 个整数的一个元组。也可以给第 3 个参数传递一个 Rect 对象，而不是传递 4 个整数的一个元组。

第 45 行，我们想要绘制的矩形距文本四周有 20 个像素。这就是为什么我们想要绘制的矩形的左上角是 textRect 的左上角的坐标减去 20（记住，使用减法，是因为向左和向上移动的时候，坐标值会减小）。宽和高等于 textRect 的宽和高加上 40（因为向左和向上移动了 20 个像素，所以需要增加宽度和高度来弥补这个空间）。

### 18.10.6 pygame.PixelArray 数据类型

```
47.# get a pixel array of the surface
48.pixArray = pygame.PixelArray(windowSurface)
49.pixArray[480][380] = BLACK
```

第 48 行创建了一个 pygame.PixelArray 对象（简称 PixelArray 对象）。PixelArray 对象是一个颜色元组的列表组成的一个列表，这些颜色元组表示传递给它的 Surface 对象。

第 48 行把 windowSurface 作为参数传递给 pygame.PixelArray()函数调用，所以在第 49 行把 BLACK 分配给 pixArray[480][380]，将会把坐标为(480, 380)的像素改变为一个黑色像素。Pygame 将把这个改变自动更新到 windowSurface 对象。

PixelArray 对象中的第 1 个索引是 X 坐标。第 2 个索引是 Y 坐标。PixelArray 对象使得在一个 PixelArray 对象上将单独像素设置为特定颜色变得很容易。

```
50.delpixArray
```

从一个 Surface 对象创建 PixelArray 对象，将会锁定这个 Surface 对象。锁定意味着该

Surface 对象不能调用 blit()函数（稍后会介绍）。要解锁这个 Surface 对象，必须用 del 操作符删除 PixelArray 对象。如果忘记了删除 PixelArray 对象，就会得到一条错误信息：pygame.error: Surfaces must not be locked during blit。

### 18.10.7　Surface 对象的 blit()方法

```
52.# draw the text onto the surface
53.windowSurface.blit(text, textRect)
```

blit()方法将一个 Surface 对象的内容绘制到另一个 Surface 对象之上。第 53 行将绘制变量 text 中的“Hello world!”Surface 对象，并将其绘制到存储在 windowSurface 变量中的 Surface 对象上。

blit()的第 2 个参数指定了要把变量 text 的 Surface 绘制到 windowSurface 的 Surface 上的什么位置。把第 23 行调用 text.get_rect()函数所获取的 Rect 对象，作为参数传递给了 blit()方法。

### 18.10.8　pygame.display.update()函数

```
55.# draw the window onto the screen
56.pygame.display.update()
```

在 Pygame 中，在调用 pygame.display.update()函数之前，并不会真的将任何内容绘制到屏幕上。这是因为在屏幕上绘图要比在计算机内存中的 Surface 对象上绘图慢。在每次调用了绘图函数之后，并不希望更新到屏幕上，只有在所有绘制函数都调用完后，才想要一次性更新屏幕。

## 18.11　事件和游戏循环

在前边的游戏中，所有程序都会立即打印每一项内容，直到它们遇到一个 input()函数调用。此时，程序会停止，等待用户输入一些内容并按下回车键。但是，Pygame 程序不断地运行叫做游戏循环（game loop）的一个循环。在这个程序中，游戏循环中的所有代码行每秒钟都会执行 100 次左右。

游戏循环是不断地查看新事件、更新窗口的状态并在屏幕上绘制窗口的一个循环。事件（event）是 pygame.event.Event 数据类型的对象，任何时候，当用户按下一个按键、点击或移动鼠标或使得其他一些其他事件发生的时候（这些事件如表 19-1 所示），Pygame 都会创建该对象。

```
58.# run the game loop
59.while True:
```

第 59 行是游戏循环的开始。把 while 语句的条件设置为 True，所以它会永远循环。退出循环的唯一方式是，如果有一个事件导致了程序终止。

### 18.11.1 pygame.event.get()函数

```
60.    for event in pygame.event.get():
61.        ifevent.type == QUIT:
```

调用 pygame.event.get()函数检索自从上次调用 pygame.event.get()后所生成的任何新的 pygame.event.Event 对象（简称为 Event 对象）。这些事件会以 Event 对象的一个列表的形式返回。所有 Event 对象都有一个叫做 type 的属性，它告诉我们事件是什么类型（在本章中，我们只处理 QUIT 类型的事件。下一章将会介绍其他类型的事件）。

第 60 行有一个 for 循环，遍历了 pygame.event.get()所返回的列表中的每一个 Event 对象。如果事件的 type 属性等于常量 QUIT，那么我们就知道用户已经关闭了窗口并且想要终止程序。

当用户点击程序窗口的关闭按钮（通常是×图标）时，Pygame 创建了 QUIT 事件（从 pygame.locals 模块导入）。如果关闭了计算机并尝试终止所有运行的程序，也会产生该事件。无论什么原因产生了 QUIT 事件，都应该终止程序。

### 18.11.2 pygame.quit()函数

```
62.        pygame.quit()
63.        sys.exit()
```

如果已经产生了 QUIT 事件，程序就应该调用 pygame.quit()和 sys.exit()。

这就是 Pygame 中简单的“Hello world!”程序。我们介绍了在之前游戏中不需要涉及的许多新的概念。尽管代码更加复杂，但是 Pygame 程序也比文本游戏更有趣。接下来，我们介绍如何创建带有动画图形的游戏。

## 18.12 动画

在这个程序中，我们有几个不同的积木在窗口的边缘反弹回来。这些积木有不同的颜色和大小，并且只在对角线上移动。为了让积木有动画的效果（也就是让它们看上去像是在移动），我们将在游戏循环的每一次迭代中，让这些积木移动一些像素。这就会使得积木看上去

像是在屏幕上移动，如图 18-6 所示。

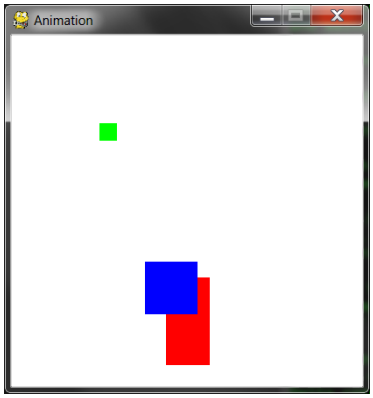



图 18-6　Animation 程序的动态截图

## 18.13　Animation 程序的源代码

在文件编辑器中输入如下代码，并且把它保存为 animation.py。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/animation 上的在线 diff 工具，把你的代码与书中的代码进行比较。

animation.py

```
 1.importpygame, sys, time
 2.frompygame.locals import *
 3.
 4.# set up pygame
 5.pygame.init()
 6.
 7.# set up the window
 8. WINDOWWIDTH = 400
 9. WINDOWHEIGHT = 400
10.windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0,
32)
11.pygame.display.set_caption('Animation')
12.
13.# set up direction variables
14. DOWNLEFT = 1
15. DOWNRIGHT = 3
16. UPLEFT = 7
17. UPRIGHT = 9
18.
```

```
19. MOVESPEED = 4
20.
21.# set up the colors
22. BLACK = (0, 0, 0)
23. RED = (255, 0, 0)
24. GREEN = (0, 255, 0)
25. BLUE = (0, 0, 255)
26.
27.# set up the block data structure
28.b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}
29.b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
30.b3 = {'rect':pygame.Rect(100, 150, 60, 60), 'color':BLUE, 'dir':DOWNLEFT}
31.blocks = [b1, b2, b3]
32.
33.# run the game loop
34.while True:
35.     # check for the QUIT event
36.     for event in pygame.event.get():
37.         ifevent.type == QUIT:
38.             pygame.quit()
39.             sys.exit()
40.
41.     # draw the black background onto the surface
42.     windowSurface.fill(BLACK)
43.
44.     for b in blocks:
45.         # move the block data structure
46.         if b['dir'] == DOWNLEFT:
47.             b['rect'].left -= MOVESPEED
48.             b['rect'].top += MOVESPEED
49.         if b['dir'] == DOWNRIGHT:
50.             b['rect'].left += MOVESPEED
51.             b['rect'].top += MOVESPEED
52.         if b['dir'] == UPLEFT:
53.             b['rect'].left -= MOVESPEED
54.             b['rect'].top -= MOVESPEED
55.         if b['dir'] == UPRIGHT:
56.             b['rect'].left += MOVESPEED
57.             b['rect'].top -= MOVESPEED
58.
59.         # check if the block has move out of the window
60.         if b['rect'].top < 0:
61.             # block has moved past the top
62.             if b['dir'] == UPLEFT:
63.                 b['dir'] = DOWNLEFT
```

```
64.             if b['dir'] == UPRIGHT:
65.                 b['dir'] = DOWNRIGHT
66.         if b['rect'].bottom > WINDOWHEIGHT:
67.             # block has moved past the bottom
68.             if b['dir'] == DOWNLEFT:
69.                 b['dir'] = UPLEFT
70.             if b['dir'] == DOWNRIGHT:
71.                 b['dir'] = UPRIGHT
72.         if b['rect'].left < 0:
73.             # block has moved past the left side
74.             if b['dir'] == DOWNLEFT:
75.                 b['dir'] = DOWNRIGHT
76.             if b['dir'] == UPLEFT:
77.                 b['dir'] = UPRIGHT
78.         if b['rect'].right > WINDOWWIDTH:
79.             # block has moved past the right side
80.             if b['dir'] == DOWNRIGHT:
81.                 b['dir'] = DOWNLEFT
82.             if b['dir'] == UPRIGHT:
83.                 b['dir'] = UPLEFT
84.
85.         # draw the block onto the surface
86.         pygame.draw.rect(windowSurface, b['color'], b['rect'])
87.
88.     # draw the window onto the screen
89.     pygame.display.update()
90.     time.sleep(0.02)
```

## 18.14　Animation 程序如何工作

在这个程序中，我们将 3 种不同颜色的积木四处移动并且从墙上反弹回来。要实现这些功能，需要先考虑想要让这些积木怎样移动。

### 18.14.1　让积木移动和弹回

每个积木将在 4 条对角线方向中的一条上移动。当积木碰到了窗口，它就会从边缘上弹回来，并在一条新的对角线上移动。积木弹回动作如图 18-7 所示。

在积木弹回之后，其新的移动方向取决于两点：在弹回前移动的方向以及它从哪一面墙弹回。每块积木都有 8 种可能的弹回方式：在 4 面墙中每一面之上，都有两种不同的弹回方式。

例如，如果一个积木向右下方移动，那么在窗口的底部弹回，我们认为积木的新的移动方向是右上方。

我们可以用一个 Rect 对象表示积木，用 Rect 对象的位置和大小表示积木的位置和大小，用 3 个整数的一个元组表示积木的颜色，用一个整数表示积木当前在朝 4 条对角线中的哪一条移动。

在游戏循环中的每一次迭代中，调整 Rect 对象中的积木的 X 和 Y 位置。而且，在每次迭代中，在屏幕上绘制所有积木的当前位置。当程序执行游戏循环的迭代时，积木将缓缓地在屏幕上移动，以使其看上去是很平滑地移动且相对于自身弹回的。

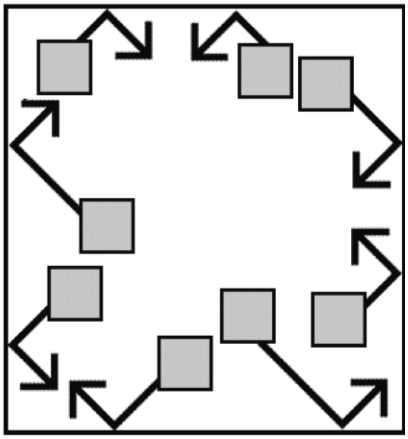


图 18-7 积木弹回的示意图

## 18.14.2 创建并设置 Pygame 和主窗口

```
 1.importpygame, sys, time
 2.frompygame.locals import *
 3.
 4.# set up pygame
 5.pygame.init()
 6.
 7.# set up the window
 8. WINDOWWIDTH = 400
 9. WINDOWHEIGHT = 400
10.windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0,
32)
```

在这个程序中，我们将看到窗口的宽和高的大小不仅仅用来调用 set_mode()函数。使用常量，以便当我们想要修改窗口大小的时候，只需要修改第 8 行和第 9 行即可。既然在程序执行时，窗口的宽和高从不改变，那么使用常量是一个好主意。

```
11.pygame.display.set_caption('Animation')
```

第 11 行通过调用 pygame.display.set_caption()函数，将窗口的标题设置为'Animation'。

## 18.14.3 为方向设置常量

```
13.# set up direction variables
14. DOWNLEFT = 1
15. DOWNRIGHT = 3
```

```
16. UPLEFT = 7
17. UPRIGHT = 9
```

使用键盘的数字小键盘上的键，来帮助记住哪个键属于哪个方向。这与 Tic Tac Toe 游戏类似。1 是左下，3 是右下，7 是左上，9 是右上。然而，这可能很难记住，所以使用常量而不是这些数字。

我们也可以不使用常量，而使用任何想要的值来表示这些方向。例如，可以使用字符串'downleft'表示左下对角线方向。然而，如果输入'downleft'字符串时出现错误（例如输入成'fownleft'），Python 将无法识别到你想要输入的是'downleft'而不是'fownleft'。这个错误将会导致程序行为异常，但是程序又不会崩溃。

但是，如果使用常量，不小心错误地输入了变量名 FOWNLEFT 而没有输入 DOWNLEFT，Python 将注意到没有名为 FOWNLEFT 的变量，就会报出程序错误。这仍然是一个相当糟糕的错误，但是至少我们将立刻发现它并修改它。

```
19. MOVESPEED = 4
```

### 18.14.4　为颜色设置常量

```
21.# set up the colors
22. BLACK = (0, 0, 0)
23. RED = (255, 0, 0)
24. GREEN = (0, 255, 0)
25. BLUE = (0, 0, 255)
```

第 22 行到 25 行代码为颜色设置常量。记住，Pygame 使用 3 个整数值的一个元组，来表示红色、绿色和蓝色数量的 RGB 值。整数值从 0 到 255。

使用常量是为了有更好的可读性。如果用一个名为 GREEN 的变量表示绿色，计算机也无所谓。GREEN 表示绿色，要比(0, 255, 0)表示绿色更容易理解。

### 18.14.5　设置积木数据结构

```
27.# set up the block data structure
28.b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}
```

创建一个字典来表示每个积木的数据结构（第 10 章介绍过字典）。这个字典将有'rect'键（用 Rect 对象作为其值）、'color'键（用包含 3 个整数的一个元组作为其值）和'dir'键（用一个方向常量作为其值）。

变量 b1 将存储这样的一个积木数据结构。这个积木的左上角的 X 坐标是 300、Y 坐标是 80。它有 50 个像素宽，100 个像素高。它的颜色是红色，其方向设置为 UPRIGHT。

```
29.b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
30.b3 = {'rect':pygame.Rect(100, 150, 60, 60), 'color':BLUE, 'dir':DOWNLEFT}
```

第 29 行和 30 行代码为积木创建了两个类似的数据结构，只是大小、位置、颜色和方向不同。

```
31.blocks = [b1, b2, b3]
```

第 31 行把这些数据结构放入到一个列表中，并且把这个列表保存到名为 blocks 的变量中。

blocks 变量存储了一个列表。blocks[0]将是 b1 中的字典数据结构。blocks[0]['color']是 b1 中'color'键的值，所以表达式 blocks[0]['color']的结果是(255, 0, 0)。通过以 blocks 开头的这种方式，可以引用任何积木数据结构的任何值。

## 18.15 运行程序循环

```
33.# run the game loop
34.while True:
35.    # check for the QUIT event
36.    for event in pygame.event.get():
37.        if event.type == QUIT:
38.            pygame.quit()
39.            sys.exit()
```

在游戏循环中，积木将在屏幕上以它们正朝向的方向移动，如果碰到墙就弹回。代码还是把所有积木绘制到 windowSurface 的 Surface 对象上，并且调用 pygame.display.update()。

for 循环查看 pygame.event.get()函数返回的列表中的所有事件，就像在“Hello World!”程序中所做的一样。

```
41.    # draw the black background onto the surface
42.    windowSurface.fill(BLACK)
```

首先，第 42 行用黑色填充了整个 Surface 对象，所以前面在 Surface 对象上绘制的所有内容都擦除了。

### 18.15.1 移动每块积木

```
44.    for b in blocks:
```

接下来，代码必须更新每块积木的位置，所以要遍历 blocks 列表。在这个循环中，我们直接用 b 引用当前的积木，以便于录入。

```
45.         # move the block data structure
46.         if b['dir'] == DOWNLEFT:
47.             b['rect'].left -= MOVESPEED
48.             b['rect'].top += MOVESPEED
49.         if b['dir'] == DOWNRIGHT:
50.             b['rect'].left += MOVESPEED
51.             b['rect'].top += MOVESPEED
52.         if b['dir'] == UPLEFT:
53.             b['rect'].left -= MOVESPEED
54.             b['rect'].top -= MOVESPEED
55.         if b['dir'] == UPRIGHT:
56.             b['rect'].left += MOVESPEED
57.             b['rect'].top -= MOVESPEED
```

为 left 属性和 top 属性设置的新值，取决于积木的方向。如果积木的方向（存储在'dir'键中）是 DOWNLEFT 或 DOWNRIGHT，那么就要增加 top 属性。如果方向是 UPLEFT 和 UPRIGHT，那么就要减少 top 属性。

如果积木的方向是 DOWNRIGHT 和 UPRIGHT，那么就要增加 left 属性。如果积木的方向是 DOWNLEFT 和 UPLEFT，那么就要减少 left 属性。

通过 MOVESPEED 中存储的整数，来修改这些属性的值。MOVESPEED 存储了在游戏循环的每次迭代中积木将要移动的像素数目，第 19 行代码进行了这一设置。

## 18.15.2　判断积木是否弹回

```
59.         # check if the block has move out of the window
60.         if b['rect'].top < 0:
61.             # block has moved past the top
62.             if b['dir'] == UPLEFT:
63.                 b['dir'] = DOWNLEFT
64.             if b['dir'] == UPRIGHT:
65.                 b['dir'] = DOWNRIGHT
```

第 44 行到 57 行代码移动积木，判断积木是否越过了窗口的边缘。如果越过了，就要把积木“弹回”。在代码中，这表示为积木的'dir'键设置一个新的值。积木将在游戏循环的下一次迭代中朝着新的方向移动。这使得积木看上去像是从窗口的边缘弹回来。

第 60 行的 if 语句中，如果积木的 Rect 对象的 top 属性小于 0，积木就越过了窗口的顶部。在这种情况中，根据积木移动的方向来修改方向（UPLEFT 或 UPRIGHT）。

### 18.15.3 修改积木的弹回方向

看一下本章前面介绍的弹回的图形。要越过窗口的顶部，积木必须在 UPLEFT 或 UPRIGHT 方向移动。如果积木在 UPLEFT 方向移动，新的方向（根据弹回图形）将是 DOWNLEFT。如果积木在 UPRIGHT 方向移动，新的方向将是 DOWNRIGHT。

```
66.         if b['rect'].bottom > WINDOWHEIGHT:
67.             # block has moved past the bottom
68.             if b['dir'] == DOWNLEFT:
69.                 b['dir'] = UPLEFT
70.             if b['dir'] == DOWNRIGHT:
71.                 b['dir'] = UPRIGHT
```

如果积木越过窗口的底部，第 66 行到 71 行代码处理这种情况。判断 bottom 属性（而不是 top 属性）是否大于 WINDOWHEIGHT。记住，Y 坐标在窗口顶部为 0，增加到底部的时候为 WINDOWHEIGHT。

剩下的代码根据图 18-7 中的弹回图形，来对方向进行了修改。

```
72.         if b['rect'].left < 0:
73.             # block has moved past the left side
74.             if b['dir'] == DOWNLEFT:
75.                 b['dir'] = DOWNRIGHT
76.             if b['dir'] == UPLEFT:
77.                 b['dir'] = UPRIGHT
78.         if b['rect'].right > WINDOWWIDTH:
79.             # block has moved past the right side
80.             if b['dir'] == DOWNRIGHT:
81.                 b['dir'] = DOWNLEFT
82.             if b['dir'] == UPRIGHT:
83.                 b['dir'] = UPLEFT
```

第 78 行到 83 行与 72 行到 77 行类似，但是这里判断的是积木的左边是否越过了窗口的左边缘。记住，X 坐标从窗口左边缘的 0 开始，增加到窗口右边缘时为 WINDOWHEIGHT。

### 18.15.4 在窗口上绘制积木的新位置

```
85.         # draw the block onto the surface
86.         pygame.draw.rect(windowSurface, b['color'], b['rect'])
```

现在积木已经移动了，所以应该通过调用 pygame.draw.rect()函数，在 windowSurface 的 Surface 对象上绘制它们新的位置。传递 windowSurface，因为要把矩形绘制在该 Surface

对象上。传递 b['color']，因为这是矩形的颜色。传递 b['rect']，因为它是带有绘制的矩形的位置和大小的 Rect 对象。

第 86 行代码是 for 循环的最后一行。如果想要增加新的积木，只需要修改第 31 行的 blocks 列表，其他的代码仍然可以工作。

### 18.15.5　在屏幕上绘制窗口

```
88.     # draw the window onto the screen
89.     pygame.display.update()
90.     time.sleep(0.02)
```

blocks 列表中的每块积木都绘制好之后，调用 pygame.display.update()函数，以便将 windowSurface 的 Surface 对象绘制到屏幕上。

在这行代码之后，执行循环将回到游戏循环的起始处，重新开始这个过程。通过这种方式，积木不断地一点点地移动，从墙上弹回，以新的位置绘制到屏幕上。

这里调用了 time.sleep()函数，因为计算机可以很快地对积木进行移动、弹回和绘制，所以如果程序全速运行的话，积木看上去就会是一团模糊（尝试注释掉 time.sleep(0.02)代码行，并且运行程序，看看这种效果）。

这里的 time.sleep()函数调用会将程序停止 0.02 秒，或 20 毫秒。

### 18.15.6　绘制积木的轨迹

通过在第 42 行（windowSurface.fill(BLACK)行）之前加一个#号，将该行注释掉。

如果不调用 windowSurface.fill(BLACK)函数，在新的位置绘制矩形之前，并没有把整个窗口都涂黑。因为游戏循环的上一次迭代中绘制的矩形并没有涂黑掉，所以显示出了矩形的轨迹。

记住，积木不是真的在移动。在游戏循环的每次迭代中，代码都会重新绘制整个窗口，新的积木每次都有几个像素的位移。

## 18.16　本章小结

本章介绍了创建计算机程序的一种全新的方式。第 17 章的程序会停止下来并且等待玩家输入文本。然而，在我们的动画程序中，程序不用等待玩家的输入，就可以不断地更新物体的数据结构。

在 Hangman 和 Tic Tac Toe 游戏中，有表示游戏板状态的数据结构，把这些数据结构传递给一个 drawBoard()函数，以便将其显示在屏幕上。我们的动画程序也做类似的事情。

blocks 变量存储了表示要绘制到屏幕上的积木的数据结构的一个列表，在游戏循环中，这些积木将会绘制到屏幕上。

但是不调用 input()函数，我们如何得到用户的输入呢？在下一章中，我们将介绍程序如何知道玩家按下了键盘上的按键。我们还会介绍碰撞检测的概念。

# 第 19 章　碰撞检测与鼠标/键盘的输入

**本章的主要内容：**

- 碰撞检测；
- 当遍历一个列表的时候不要修改它；
- Pygame 中的键盘输入；
- Pygame 中的鼠标输入。

碰撞检测负责计算屏幕上的两个物体何时发生彼此接触（也就是发生碰撞）。例如，如果玩家接触到了一个敌人，它们可能会损失生命值。或者，当玩家接触到金币的时候，程序需要能够知道这一点，以便玩家可以自动地把金币捡起来。碰撞检测可以帮助判断游戏角色是站在地面上，还是漂浮在空中。

在我们的游戏中，碰撞检测将判断两个矩形是否彼此重叠。下一个示例程序将介绍这种基本的技术。

在本章的后边，我们将看到 Pygame 程序如何接收用户通过键盘和鼠标的输入。这要比我们在文本程序中调用 input()函数复杂一些。但是在 GUI 程序中，使用键盘要更具有交互性。而使用鼠标在文本游戏中几乎是不可能的。这两个概念将使得游戏更加有趣！

## 19.1　Collision Detection 程序的源代码

许多代码与 Animation 程序类似，所以我们会省略移动和弹跳这部分代码的介绍（请参见第 18 章的 Animation 程序）。一个“大块头”将会在窗口中蹦跳。Rect 对象的一个列表将表示食物方块。

在游戏循环的每一次迭代中，程序都会读取列表中的每个 Rect 对象，并且在窗口上绘制一个绿色的方块。游戏循环中每 40 次迭代，就会增加一个新的 Rect 对象到列表中，以便屏幕中一直有新的食物方块。

用一个字典来表示“大块头”。这个字典有一个叫做'rect'的键（其值是 pygame.Rect 对象）和一个叫做'dir'的键（其值是方向常量之一，就像 Animation 程序中使用的方向常量一样）。

由于“大块头”在窗口中蹦跳，我们将判断它是否和任何食物方块有所碰撞。如果是的，我们需要删除掉食物方块，以便屏幕上不再绘制这个食物方块。看上去，就好像是“大块头”“吃掉了”窗口中的食物方块。

输入如下内容到一个新的文件，并且把它保存为 collisionDetection.py。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/collisionDetection 上的在线 diff 工具，把你的代码与书中的代码进行比较。

collisionDetection.py

```
 1. import pygame, sys, random
 2. from pygame.locals import *
 3.
 4. def doRectsOverlap(rect1, rect2):
 5.     for a, b in [(rect1, rect2), (rect2, rect1)]:
 6.         # Check if a's corners are inside b
 7.         if ((isPointInsideRect(a.left, a.top, b)) or
 8.            (isPointInsideRect(a.left, a.bottom, b)) or
 9.            (isPointInsideRect(a.right, a.top, b)) or
10.            (isPointInsideRect(a.right, a.bottom, b))):
11.            return True
12.
13.     return False
14.
15. def isPointInsideRect(x, y, rect):
16.     if (x > rect.left) and (x < rect.right) and (y > rect.top) and (y
< rect.bottom):
17.         return True
18.     else:
19.         return False
20.
21.
22. # set up pygame
23. pygame.init()
24. mainClock = pygame.time.Clock()
25.
26. # set up the window
27. WINDOWWIDTH = 400
28. WINDOWHEIGHT = 400
29. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT),
0, 32)
30. pygame.display.set_caption('Collision Detection')
31.
32. # set up direction variables
33. DOWNLEFT = 1
34. DOWNRIGHT = 3
35. UPLEFT = 7
36. UPRIGHT = 9
37.
38. MOVESPEED = 4
39.
40. # set up the colors
41. BLACK = (0, 0, 0)
```

```
 42. GREEN = (0, 255, 0)
 43. WHITE = (255, 255, 255)
 44.
 45. # set up the bouncer and food data structures
 46. foodCounter = 0
 47. NEWFOOD = 40
 48. FOODSIZE = 20
 49. bouncer = {'rect':pygame.Rect(300, 100, 50, 50), 'dir':UPLEFT}
 50. foods = []
 51. for i in range(20):
 52.     foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH -
FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
 53.
 54. # run the game loop
 55. while True:
 56.     # check for the QUIT event
 57.     for event in pygame.event.get():
 58.         if event.type == QUIT:
 59.             pygame.quit()
 60.             sys.exit()
 61.
 62.     foodCounter += 1
 63.     if foodCounter >= NEWFOOD:
 64.         # add new food
 65.         foodCounter = 0
 66.         foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH -
FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
 67.
 68.     # draw the black background onto the surface
 69.     windowSurface.fill(BLACK)
 70.
 71.     # move the bouncer data structure
 72.     if bouncer['dir'] == DOWNLEFT:
 73.         bouncer['rect'].left -= MOVESPEED
 74.         bouncer['rect'].top += MOVESPEED
 75.     if bouncer['dir'] == DOWNRIGHT:
 76.         bouncer['rect'].left += MOVESPEED
 77.         bouncer['rect'].top += MOVESPEED
 78.     if bouncer['dir'] == UPLEFT:
 79.         bouncer['rect'].left -= MOVESPEED
 80.         bouncer['rect'].top -= MOVESPEED
 81.     if bouncer['dir'] == UPRIGHT:
 82.         bouncer['rect'].left += MOVESPEED
 83.         bouncer['rect'].top -= MOVESPEED
 84.
```

```
    # check if the bouncer has move out of the window
    if bouncer['rect'].top < 0:
        # bouncer has moved past the top
        if bouncer['dir'] == UPLEFT:
            bouncer['dir'] = DOWNLEFT
        if bouncer['dir'] == UPRIGHT:
            bouncer['dir'] = DOWNRIGHT
    if bouncer['rect'].bottom > WINDOWHEIGHT:
        # bouncer has moved past the bottom
        if bouncer['dir'] == DOWNLEFT:
            bouncer['dir'] = UPLEFT
        if bouncer['dir'] == DOWNRIGHT:
            bouncer['dir'] = UPRIGHT
    if bouncer['rect'].left < 0:
        # bouncer has moved past the left side
        if bouncer['dir'] == DOWNLEFT:
            bouncer['dir'] = DOWNRIGHT
        if bouncer['dir'] == UPLEFT:
            bouncer['dir'] = UPRIGHT
    if bouncer['rect'].right > WINDOWWIDTH:
        # bouncer has moved past the right side
        if bouncer['dir'] == DOWNRIGHT:
            bouncer['dir'] = DOWNLEFT
        if bouncer['dir'] == UPRIGHT:
            bouncer['dir'] = UPLEFT

    # draw the bouncer onto the surface
    pygame.draw.rect(windowSurface, WHITE, bouncer['rect'])

    # check if the bouncer has intersected with any food squares.
    for food in foods[:]:
        if doRectsOverlap(bouncer['rect'], food):
            foods.remove(food)

    # draw the food
    for i in range(len(foods)):
        pygame.draw.rect(windowSurface, GREEN, foods[i])

    # draw the window onto the screen
    pygame.display.update()
    mainClock.tick(40)
```

程序运行情况如图 19-1 所示。“大块头”方块将会在窗口中来回弹跳。当它碰撞到绿色的食物方块时，食物方块将从屏幕上消失。

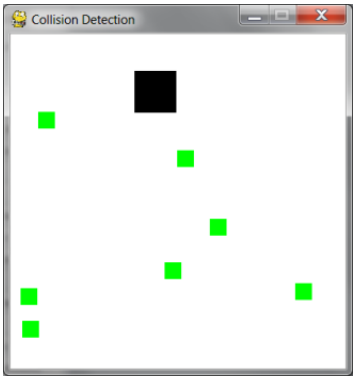



图 19-1　Collision Detection 程序的动态截图

### 导入模块

```
1. import pygame, sys, random
2. from pygame.locals import *
```

Collision Detection 程序导入的内容和第 18 章中 Animation 程序一样，除此之外，它还导入了 random 模块。

## 19.2　Collision Detection 算法

```
4. def doRectsOverlap(rect1, rect2):
```

要进行碰撞检测，需要有一个函数能够判断两个矩形是否彼此碰撞。图 19-2 展示了碰撞的矩形和没有碰撞的矩形。

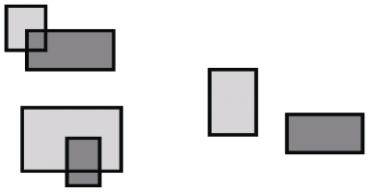


图 19-2　碰撞的矩形（左边）和没有碰撞的矩形（右边）的示例

doRectsOverlap()接受两个 pygame.Rect 对象作为参数。如果两个矩形碰撞，函数返回 True；如果两个矩形没有碰撞，函数返回 False。有一个简单的规则来判断矩形是否碰撞。查看两个矩形的 4 个角中的每个角。如果这 8 个角中至少有一个角在另外一个矩形之中，那么，我们就知道两个矩形碰撞了。我们可以根据这个事实来确定 doRectsOverlap()函数返回 True 还是 False。

```
5.     for a, b in [(rect1, rect2), (rect2, rect1)]:
6.         # Check if a's corners are inside b
7.         if ((isPointInsideRect(a.left, a.top, b)) or
```

```
 8.            (isPointInsideRect(a.left, a.bottom, b)) or
 9.            (isPointInsideRect(a.right, a.top, b)) or
10.            (isPointInsideRect(a.right, a.bottom, b))):
11.            return True
```

第 5 行到第 11 行的代码判断一个矩形的角是否在另一个矩形中。稍后，我们将创建一个名为 isPointInsideRect()的函数，如果点的 XY 坐标在矩形中，该函数返回 True。针对 8 个角中的每个角调用这个函数，如果任何一次调用返回 True，or 操作符就会使得整个条件为 True。

doRectsOverlap()的参数是 rect1 和 rect2。首先判断 rect1 的角是否在 rect2 中，然后判断 rect2 的角是否在 rect1 中。

不需要重复这段代码来判断 rect1 和 rect2 中的所有 4 个角。只需要重复使用第 7 到第 10 行的 a 和 b 就可以了。第 5 行 for 的循环使用了多变量赋值。在第 1 次迭代中，把 rect1 设置为 a，把 rect2 设置为 b。在循环的第 2 次迭代中，反过来把 rect2 设置为 a，把 rect1 设置为 b。

```
13.    return False
```

第 11 行代码不返回 True，那么 8 个角都不在其他的矩形中。在这个示例中，矩形没有发生碰撞，第 13 行代码返回 False。

### 19.2.1 判断一个点是否在一个矩形中

```
15. def isPointInsideRect(x, y, rect):
16.     if (x > rect.left) and (x < rect.right) and (y > rect.top) and (y
< rect.bottom):
17.         return True
```

doRectsOverlap()函数调用了 isPointInside Rect()函数。如果传入的参数 XY 坐标位于作为第 3 个参数而传入的 pygame.Rect 对象之中，isPointInsideRect()函数将返回 True。否则，这个函数返回 False。

图 19-3 是一个矩形和几个点的示例图。这些点和矩形的角的坐标分别标记了出来。

如果以下 4 个条件都为 True，那么点就在矩形中：

- 点的 X 坐标大于矩形的左边的 X 坐标；
- 点的 X 坐标小于矩形的右边的 X 坐标；
- 点的 Y 坐标大于矩形的上边的 Y 坐标；

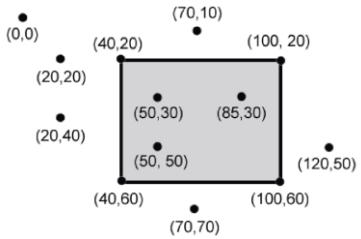



图 19-3 矩形之内和之外的点的坐标示例。点(50, 30)、(85, 30)和(50, 50)都在矩形之内，其他点都在矩形之外

- 点的 Y 坐标小于矩形的下边的 Y 坐标。

如果任意一个条件为 False，那么点在矩形的之外。第 16 行使用 and 操作符，把所有这 4 个条件组合到一条 if 语句之中。

```
18.    else:
19.        return False
```

doRectsOverlap()函数调用 isPointInsideRect()函数，来判断两个 pygame.Rect 对象中的任何角是否在另外一个矩形之中。这两个函数提供了在两个矩形之间进行碰撞检测的功能。

### 19.2.2　pygame.time.Clock 对象和 tick()方法

第 22 行到第 43 行所做的事情和第 18 章中的 Animation 程序所做事情的一样：初始化 Pygame、设置 WINDOWHEIGHT 和 WINDOWWIDTH 并指定颜色和方向常量。

但是，第 24 行是新的代码：

```
24. mainClock = pygame.time.Clock()
```

在之前的 Animation 程序中，一次 time.sleep(0.02)函数调用可以减缓程序的执行速度，使得程序不会运行的太快。time.sleep()的问题是，对于较慢的计算机会暂停的时间太久，而对于较快的计算机暂停时间又不够。

pygame.time.Clock 对象可以对于任何计算机都暂停适当的时间。第 125 行在游戏循环中调用了 mainClock.tick(40)函数。这次调用 Clock 对象的 tick()方法，会等待足够长的时间，以便无论计算机速度有多快，它都可以每秒钟迭代 40 次。这就确保了游戏的运行速度不会超过预期。在游戏循环中只能调用 tick()一次。

### 19.2.3　创建窗口和数据结构

```
45.# set up the bouncer and food data structures
46. foodCounter = 0
47. NEWFOOD = 40
48. FOODSIZE = 20
```

第 46 行到第 48 行代码为屏幕上的食物方块创建了一些变量。foodCounter 的初始值为 0，NEWFOOD 的初始值为 40，FOODSIZE 的初始值为 20。

```
49. bouncer = {'rect':pygame.Rect(300, 100, 50, 50), 'dir':UPLEFT}
```

第 49 行创建一个名为 bouncer. Bouncer 的新的数据结构，它是包含两个键的一个字典。'rect'键拥有表示“大块头”的大小和位置的一个 pygame.Rect 对象。

'dir'键拥有的值表示当前“大块头”移动的一个方向。“大块头”的移动方式和第 18 章的 Animation 程序中的积木的移动方式相同。

```
 50. foods = []
 51. for i in range(20):
 52.     foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE),
random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
```

程序将使用 foods 中的 Rect 对象的一个列表来记录每一个食物方块。第 51 行和第 52 行代码创建在屏幕上随机放置的 20 个食物方块。可以使用 random.randint()函数来得到随机的 XY 坐标。

在第 52 行，我们将调用 pygame.Rect()构造函数来返回一个新的 pygame.Rect 对象。它将表示食物方块的位置和大小。pygame.Rect()函数的前两个参数是左上角的 XY 坐标。我们想要一个随机的坐标，它在 0 和窗口大小减去食物方块大小所得的结果之间。如果使用了从 0 到窗口大小之间的一个随机坐标，那么可能会把食物方块完全推到了窗口之外，如图 19-4 所示。

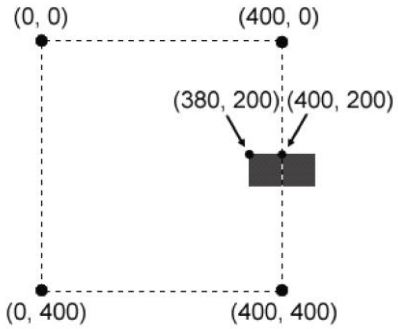



图 19-4　一个宽 20 像素高 20 像素的矩形，如果放置在宽 400 像素高 400 像素的窗口中的左上角，其坐标位置是（400, 200），那么这个矩形会放置在窗口之外。要将其放在窗口内，左上角的位置应该是（380, 200）

pygame.Rect()函数的第 3 个参数是包含食物方块的宽和高的一个元组。宽和高都是常量 FOODSIZE 中的值。

## 19.2.4　在屏幕上绘制“大块头”

第 71 行到第 91 行代码创建了在窗口中移动并且会从窗口边缘弹回的“大块头”。这些代码和第 18 章 Animation 程序中的第 44 行到第 83 行代码类似，这里就不再解释。

```
111.    # draw the bouncer onto the surface
112.    pygame.draw.rect(windowSurface, WHITE, bouncer['rect'])
```

在移动了“大块头”之后，第 112 行代码在新的位置绘制它。windowSurface 作为第一个参数传递，告诉 Python 要将矩形绘制在哪个 Surface 对象之上。变量 WHITE（存储的是(255, 255, 255)）告诉 Python 要绘制一个白色的矩形。存储在 bouncer 字典中的键为'rect'的 Rect 对象，告知要绘制的矩形的位置和大小。

### 19.2.5　食物方块的碰撞

```
114.    # check if the bouncer has intersected with any food squares.
115.    for food in foods[:]:
```

在绘制食物方块之前，判断“大块头”是否已经和任何食物方块有所重叠（碰撞）。如果有，从 foods 列表中删除食物方块。这是因为 Python 不会再绘制“大块头”已经“吃掉”的任何的食物方块。

在 for 循环的每次迭代中，foods（复数）列表中当前的食物方块在变量 food（单数）中。

## 19.3　当遍历一个列表的时候，不要修改该列表

注意，这个 for 循环会有一个细微的差别。如果仔细地查看第 116 行代码，会发现它不是对 foods 进行迭代，实际上是对 foods[:]进行迭代。

还记得切片是如何工作的吗？foods[:2]得到从索引为 2 的元素开始（但不包括该元素）的元素列表的一个副本。foods[3:]得到从索引为 3 的元素到列表末尾的一个列表副本。

foods[:]将得到从开始元素到结束元素的元素列表的一个副本。基本上，foods[:]创建了一个新的列表，其中的元素是 foods 中的所有元素的副本。相对于在 Tic Tac Toe 游戏中 getBoardCopy()函数的做法，这是复制列表的一种更为简便的方法。

当对列表进行迭代时，不能向列表中添加元素或者从列表中删除元素。如果 foods 列表总是变化，那么 Python 可能会无法记录 food 变量的下一个值。想一下，如果要数一个罐子中的糖豆数量时候，同时有人向其中添加糖豆或者从中取出糖豆，要数清罐子中的糖豆会有多难。

但是，如果对列表的一个副本进行迭代，那么向原始列表中添加元素或从中删除元素都不是问题。

### 19.3.1 删除食物方块

```
116.        if doRectsOverlap(bouncer['rect'], food):
117.            foods.remove(food)
```

第 116 行用到了 doRectsOverlap()函数。如果表示“大块头”和当前食物方块的两个矩形重叠，doRectsOverlap()函数将返回 True，并且第 117 行将会从 foods 列表中删除产生重叠的食物方块。

### 19.3.2 在屏幕上绘制食物方块

```
119.    # draw the food
120.    for i in range(len(foods)):
121.        pygame.draw.rect(windowSurface, GREEN, foods[i])
```

第 120 行与第 121 行代码和我们为玩家绘制白色方块的代码类似。第 120 行遍历了 foods 列表中的每个食物方块。第 121 行把食物方块绘制到 windowSurface 的 Surface 对象上。这个程序与第 18 章的弹跳积木的程序类似，只不过这里“大块头”将“吃掉”它碰到的其他方块。

前边几个程序看上去很有趣，但是用户什么都不能控制。在下一个程序中，我们将介绍如何从键盘获取输入。

## 19.4 键盘输入程序的源代码

把如下内容输入到一个新的文件，并且把它保存为 pygameInput.py。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/pygameInput 上的在线 diff 工具，把你的代码与书中的代码进行比较。

pygameInput.py

```
 1. import pygame, sys, random
 2. from pygame.locals import *
 3.
 4. # set up pygame
 5. pygame.init()
 6. mainClock = pygame.time.Clock()
 7.
 8. # set up the window
 9. WINDOWWIDTH = 400
10. WINDOWHEIGHT = 400
```

```
 11. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
 12. pygame.display.set_caption('Input')
 13.
 14. # set up the colors
 15. BLACK = (0, 0, 0)
 16. GREEN = (0, 255, 0)
 17. WHITE = (255, 255, 255)
 18.
 19. # set up the player and food data structure
 20. foodCounter = 0
 21. NEWFOOD = 40
 22. FOODSIZE = 20
 23. player = pygame.Rect(300, 100, 50, 50)
 24. foods = []
 25. for i in range(20):
 26.     foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - FOODSIZE),
random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
 27.
 28. # set up movement variables
 29. moveLeft = False
 30. moveRight = False
 31. moveUp = False
 32. moveDown = False
 33.
 34. MOVESPEED = 6
 35.
 36.
 37. # run the game loop
 38. while True:
 39.     # check for events
 40.     for event in pygame.event.get():
 41.         if event.type == QUIT:
 42.             pygame.quit()
 43.             sys.exit()
 44.         if event.type == KEYDOWN:
 45.             # change the keyboard variables
 46.             if event.key == K_LEFT or event.key == ord('a'):
 47.                 moveRight = False
 48.                 moveLeft = True
 49.             if event.key == K_RIGHT or event.key == ord('d'):
 50.                 moveLeft = False
 51.                 moveRight = True
 52.             if event.key == K_UP or event.key == ord('w'):
```

```
 53.                moveDown = False
 54.                moveUp = True
 55.            if event.key == K_DOWN or event.key == ord('s'):
 56.                moveUp = False
 57.                moveDown = True
 58.        if event.type == KEYUP:
 59.            if event.key == K_ESCAPE:
 60.                pygame.quit()
 61.                sys.exit()
 62.            if event.key == K_LEFT or event.key == ord('a'):
 63.                moveLeft = False
 64.            if event.key == K_RIGHT or event.key == ord('d'):
 65.                moveRight = False
 66.            if event.key == K_UP or event.key == ord('w'):
 67.                moveUp = False
 68.            if event.key == K_DOWN or event.key == ord('s'):
 69.                moveDown = False
 70.            if event.key == ord('x'):
 71.                player.top = random.randint(0, WINDOWHEIGHT - player.height)
 72.                player.left = random.randint(0, WINDOWWIDTH - player.width)
 73.
 74.        if event.type == MOUSEBUTTONUP:
 75.            foods.append(pygame.Rect(event.pos[0], event.pos[1], FOODSIZE,
FOODSIZE))
 76.
 77.    foodCounter += 1
 78.    if foodCounter >= NEWFOOD:
 79.        # add new food
 80.        foodCounter = 0
 81.        foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH -
FOODSIZE), random.randint(0, WINDOWHEIGHT - FOODSIZE), FOODSIZE, FOODSIZE))
 82.
 83.    # draw the black background onto the surface
 84.    windowSurface.fill(BLACK)
 85.
 86.    # move the player
 87.    if moveDown and player.bottom < WINDOWHEIGHT:
 88.        player.top += MOVESPEED
 89.    if moveUp and player.top > 0:
 90.        player.top -= MOVESPEED
 91.    if moveLeft and player.left > 0:
 92.        player.left -= MOVESPEED
 93.    if moveRight and player.right < WINDOWWIDTH:
 94.        player.right += MOVESPEED
```

```
 95.
 96.    # draw the player onto the surface
 97.    pygame.draw.rect(windowSurface, WHITE, player)
 98.
 99.    # check if the player has intersected with any food squares.
100.    for food in foods[:]:
101.        if player.colliderect(food):
102.            foods.remove(food)
103.
104.    # draw the food
105.    for i in range(len(foods)):
106.        pygame.draw.rect(windowSurface, GREEN, foods[i])
107.
108.    # draw the window onto the screen
109.    pygame.display.update()
110.    mainClock.tick(40)
```

这个程序和 Collision Detection 程序几乎相同。但是在这个程序中，只有当用户按下键盘的方向键时，“大块头”才会移动。

也可以在窗口中的任何地方点击鼠标，来创建新的食物对象。另外，按下 ESC 键将退出游戏，按下“X”键将把玩家（“大块头”）转移到屏幕上的一个随机位置。

### 19.4.1　设置窗口和数据结构

从第 29 行开始的代码，创建了记录“大块头”的移动的一些变量。

```
28. # set up movement variables
29. moveLeft = False
30. moveRight = False
31. moveUp = False
32. moveDown = False
```

这 4 个布尔值变量用来记录按下了哪个方向键。例如，当用户按下键盘上向左的方向键时，把 moveLeft 设置为 True。当松开这个键时，把 moveLeft 设置回 False。

第 34 行到第 43 行的代码和前边的 Pygame 程序代码相同。这些代码行处理了游戏循环开始时以及当用户退出程序时要做的事情。因为在第 18 章已经介绍过，所以这里我们将不再解释这部分代码。

### 19.4.2　事件和 KEYDOWN 事件的处理

处理按下键和释放键的事件的代码从第 44 行开始。在程序开始处，把记录“大块头”移

动的布尔变量都设置为 False。

```
44.         if event.type == KEYDOWN:
```

Pygame 有一个叫做 KEYDOWN 的事件类型。这是 Pygame 可以创建的其他事件之一。调用 pygame.event.get()函数，可以返回一个简短的事件列表，如表 19-1 所示。

表 19-1 事件及何时创建这些事件

| 事件类型 | 描述 |
| --- | --- |
| QUIT | 当用户关闭窗口时触发的事件 |
| KEYDOWN | 当用户按下键盘时触发的事件。有一个 key 属性来识别按下的是哪个键。还有一个 mode 属性来表明是否有 Shift、Ctrl、Alt 或其他的键和该键同时按下 |
| KEYUP | 当用户释放一个按键时触发的事件。有一个 key 属性和一个 mod 属性，它们和 KEYDOWN 中的属性类似 |
| MOUSEMOTION | 任何时候，当鼠标移动经过窗口时都会触发该事件。有一个 pos 属性，它返回鼠标在窗口中的坐标的元组(x, y)。rel 属性也返回一个(x, y)元组，但是它给出的是相对于上一次的 MOUSEMOTION 事件的坐标。例如，如果鼠标从(200, 200)向左移动 4 个像素到(196, 200)，那么 rel 就是元组值(-4, 0)。<br>buttons 属性返回包含 3 个整数的一个元组。元组中的第 1 个整数是鼠标左键，第 2 个整数是鼠标中间键（如果有中间键的话），第 3 个整数是鼠标右键。当鼠标移动时，如果没有按下这些键，这些整数值是 0；如果按下这些键，其对应的整数值就是 1 |
| MOUSEBUTTONDOWN | 当在窗口中按下鼠标时触发的事件。这个事件有一个 pos 属性，它是当按下鼠标键时，鼠标所在位置的坐标的(x, y)元组。它也有一个 button 属性，用整数 1 到 5 来分别表示按下了哪个键，如表 19-2 所示 |
| MOUSEBUTTONUP | 当释放鼠标时触发的事件。它具有和 MOUSEBUTTONDOWN 相同的属性 |

表 19-2 button 属性和 mouse 属性

| 键值 | 鼠标键 |
| --- | --- |
| 1 | 左键 |
| 2 | 中间键 |
| 3 | 右键 |
| 4 | 向上滚轮 |
| 5 | 向下滚轮 |

### 19.4.3　设置 4 个键盘变量

```
45.            # change the keyboard variables
46.            if event.key == K_LEFT or event.key == ord('a'):
47.                moveRight = False
48.                moveLeft = True
49.            if event.key == K_RIGHT or event.key == ord('d'):
50.                moveLeft = False
51.                moveRight = True
52.            if event.key == K_UP or event.key == ord('w'):
53.                moveDown = False
54.                moveUp = True
55.            if event.key == K_DOWN or event.key == ord('s'):
56.                moveUp = False
57.                moveDown = True
```

如果事件类型是 KEYDOWN，那么事件对象将有一个 key 属性来识别按下的是哪个键。第 46 行代码把这个属性和 K_LEFT 进行比较，K_LEFT 是表示键盘上向左方向键的 pygame.locals 常量。第 46 行到第 57 行代码对其他的每个方向键（K_LEFT、K_RIGHT、K_UP 和 K_DOWN）做类似的判断。

当按下这些键中的某一个键时，把相应的移动变量设置为 True，并把相反方向的移动变量设置为 False。

例如，当按下左方向键时，程序执行第 47 行和第 48 行代码。在这种情况下，把 moveLeft 设置为 True，把 moveRight 设置为 False（即使 moveRight 可能已经是 False，还是要确保把它设置为 False）。

在第 46 行中，event.key 中的值既可能等于 K_LEFT，也可能等于 ord('a')。把 event.key 中的值设置为键盘上按下的那个键的顺序编码（方向键没有顺序编码，这就是为什么要使用常量 K_LEFT）。可以使用 ord()函数获取任何单个字符的编码，将其与 event.key 进行比较。

如果按键是 K_LEFT 或 ord('a')，通过执行第 47 行和第 48 行代码，就可以让左方向键和 A 键做同样的事。W 键、A 键、S 键和 D 键全部用做修改移动变量的替代键。左手可以使用 WASD 键。右手可以使用方向键。

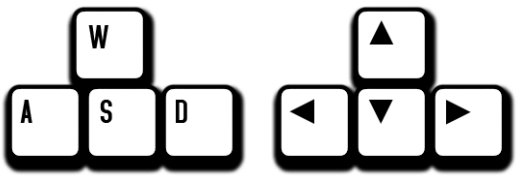



图 19-5　可以编程让 WASD 键和方向键做同样的事

### 19.4.4 处理 KEYUP 事件

```
 58.        if event.type == KEYUP:
```

当用户释放按下的键时，会触发 KEYUP 事件。

```
 59.            if event.key == K_ESCAPE:
 60.                pygame.quit()
 61.                sys.exit()
```

如果用户释放的是 ESC 键，那么程序终止。记住，在 Pygame 中，在调用 sys.exit()函数之前，必须先调用 pygame.quit()函数。

如果释放的是某一个方向键，第 62 行到 69 行代码将把一个移动变量设置为 False。

```
 62.            if event.key == K_LEFT or event.key == ord('a'):
 63.                moveLeft = False
 64.            if event.key == K_RIGHT or event.key == ord('d'):
 65.                moveRight = False
 66.            if event.key == K_UP or event.key == ord('w'):
 67.                moveUp = False
 68.            if event.key == K_DOWN or event.key == ord('s'):
 69.                moveDown = False
```

### 19.4.5 转移玩家

```
 70.            if event.key == ord('x'):
 71.                        player.top = random.randint(0, WINDOWHEIGHT -
player.height)
 72.                        player.left = random.randint(0, WINDOWWIDTH -
player.width)
```

也可以为游戏添加转移玩家的功能。如果用户按下“X”键，那么第 71 行和 72 行代码将把用户的方块的位置设置为窗口中的一个随机位置。通过按下“X”键，将为用户提供在窗口中转移的能力。然而对于将会把用户转移到什么位置，是无法控制的，这完全是随机的。

### 19.4.6 处理 MOUSEBUTTONUP 事件

```
 74.        if event.type == MOUSEBUTTONUP:
 75.            foods.append(pygame.Rect(event.pos[0], event.pos[1],
FOODSIZE, FOODSIZE))
```

处理鼠标输入事件就像处理键盘输入事件一样。当用户释放点击的鼠标时，会触发 MOUSEBUTTONUP 事件。Event 对象的 pos 属性设置为两个整数的一个元组，表示点击鼠标的时候其光标所在的 XY 坐标。

在第 75 行中，把 X 坐标存储到 event.pos[0]中，把 Y 坐标存储到 event.pos[1]中。第 75 行代码创建了一个新的 Rect 对象来表示新的食物，并把它放置在 MOUSEBUTTONUP 事件发生的地方。通过为 foods 列表增加一个新的 Rect 对象，代码将会在屏幕上显示一个新的食物方块。

### 19.4.7　在屏幕上移动玩家

```
 86.    # move the player
 87.    if moveDown and player.bottom < WINDOWHEIGHT:
 88.        player.top += MOVESPEED
 89.    if moveUp and player.top > 0:
 90.        player.top -= MOVESPEED
 91.    if moveLeft and player.left > 0:
 92.        player.left -= MOVESPEED
 93.    if moveRight and player.right < WINDOWWIDTH:
 94.        player.right += MOVESPEED
```

我们已经根据用户的按键，设置了移动变量（moveDown、moveUp、moveLeft 和 moveRight）。现在，通过调整玩家的 XY 坐标，来移动玩家的方块（用存储在 player 中的 pygame.Rect 对象来表示）。

如果把 moveDown 设置为 True（并且玩家方块的底部不低于窗口的底部），那么第 88 行通过给玩家的当前 top 属性增加 MOVESPEED，将玩家的方块向下移动。第 89 行到 94 行针对其他 3 个方向做了相同的事情。

## 19.5　colliderect()方法

```
 99.    # check if the player has intersected with any food squares.
100.    for food in foods[:]:
101.        if player.colliderect(food):
102.            foods.remove(food)
```

在前面的 Collision Detection 程序中，doRectsOverlap()函数判断一个矩形是否和另一个矩形发生碰撞。本书中包含了这个函数，以便于我们可以理解碰撞检测背后的代码是如何工作的。

在这个程序中，我们可以使用 Pygame 的碰撞检测函数。pygame.Rect 对象的 colliderect()

方法接受另一个 pygame.Rect 作为一个参数，并且如果这两个矩形发生碰撞的话，该函数返回 True，如果没有发生碰撞，该函数返回 False。

```
110.    mainClock.tick(40)
```

剩余的代码与 Input 程序和 Collision Detection 程序中的代码类似。

## 19.6 本章小结

本章介绍的碰撞检测的概念，在许多图形游戏中都会用到。两个矩形的碰撞检测很简单：判断一个矩形的 4 个角是否在另一个矩形中。由于经常会做这样的检测，所以 Pygame 为 pygame.Rect 对象提供了自己的碰撞检测方法，这个方法名为 colliderect()。

本书中的前几个程序是基于文本的。程序的输出是打印到屏幕上的文本，程序输入是用户在键盘上的输入。而图形化的程序可以接受键盘和鼠标的输入。

更进一步，当用户按下或释放单独一个键时，这些程序可以对单独的按键做出响应。用户不需要输入完整的响应并按下回车键。这就允许即时反馈并由此产生更具交互性的游戏。

# 第 20 章　声音和图像

**本章的主要内容：**

- 声音文件和图像文件；
- 绘制精灵；
- pygame.image.load()函数；
- pygame.mixer.Sound 数据类型；
- pygame.mixer.music 模块。

在第 18 章和第 19 章中，我们介绍了如何创建具有图形并且可以接收键盘和鼠标输入的 GUI 程序。我们还介绍了如何绘制不同的形状。在本章中，我们将介绍如何在游戏中显示图像和精灵图形、播放声音和音乐。

精灵（sprite）是指用作屏幕上图形的一部分的一个单独二维图像。图 20-1 展示了一些精灵的示例。

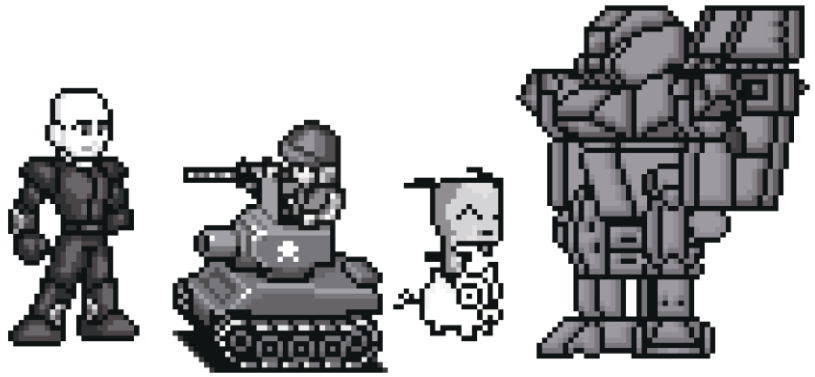


图 20-1　精灵的一些示例

在完整场景中使用精灵的示例如图 20-2 所示。

把精灵图像绘制在背景之上。注意，我们可以水平方向反转精灵图像，以使得精灵面朝其他方向。可以在相同的窗口中多次绘制相同的精灵图像。也可以重新调整精灵大小，使其比初始的精灵图像更大或更小。可以把背景图像看做是一个很大的精灵。

下面的程序将会描述如何使用 Pygame 播放声音和绘制精灵。

图 20-2 完整场景的一个示例，在背景之上绘制了精灵

## 20.1 声音文件和图像文件

精灵是存储在计算机上的图像文件。Pygame 可以使用几种不同的图像格式。可以通过查看文件末尾（最后的点之后）的名称来识别图像文件的格式。把个名称这叫做文件扩展名（file extension）。例如，player.png 是 PNG 格式。Pygame 支持的图像格式包含 BMP、PNG、JPG 和 GIF。

可以从 Web 浏览器下载图像。在大多数 Web 浏览器上，需要用鼠标右键点击 Web 页面上的图像，从弹出的菜单中选择 Save。记住图像文件在硬盘上的存储位置。把这个下载的图像文件复制到与你的 Python 程序的.py 文件相同的文件夹下。也可以使用诸如 MS Paint 或 Tux Paint 这样的程序来创建自己的图像。

Pygame 支持的声音文件格式有 MID、WAV 和 MP3。可以从互联网下载下载声音效果，就像下载图像文件一样。它们必须是这 3 种格式之一。如果计算机有一个麦克风，你也可以录音，并且在游戏中使用自己的 WAV 文件。

## 20.2 精灵和声音程序

这个程序和第 19 章中的键盘输入程序相同。然而，在这个程序中，我们使用精灵，而不是看上去很普通的方块。我们使用一个小人儿的精灵来代替白色的玩家方块，使用樱桃精灵代

替绿色的食物方块。当玩家精灵吃掉一个樱桃精灵时，我们还可以播放背景音乐和声音效果。该程序的运行效果，如图 20-3 所示。

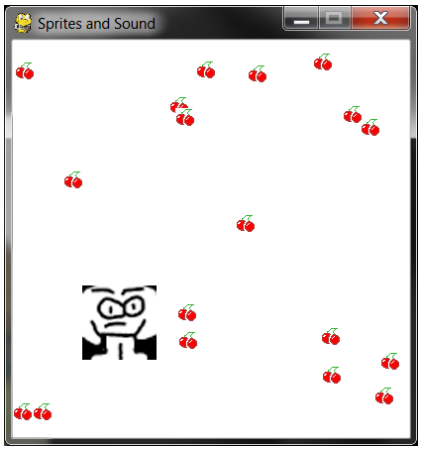



图 20-3　Sprites and Sounds 程序的动态截图

## 20.3　Sprites and Sounds 程序的源代码

如果你知道如何使用诸如 Photoshop 或 MS Paint 这样的图像软件，就可以自己绘制图像。如果你不知道如何使用这些软件，可以从网站下载图形并使用这些图像文件。这也同样适用于音乐文件和声音文件。也可以从网站或数码相机中找到图像文件。可以从本书配套网站 http://invpy.com/downloads 下载图像文件和声音文件。

如果输入这些代码后出现错误，请使用 http://invpy.com/diff/spritesAndSounds 上的在线 diff 工具，把你的代码与书中的代码进行比较。

spritesAndSounds.py

```
 1. import pygame, sys, time, random
 2. from pygame.locals import *
 3.
 4. # set up pygame
 5. pygame.init()
 6. mainClock = pygame.time.Clock()
 7.
 8. # set up the window
 9. WINDOWWIDTH = 400
10. WINDOWHEIGHT = 400
11. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
```

```
 12. pygame.display.set_caption('Sprites and Sound')
 13.
 14. # set up the colors
 15. BLACK = (0, 0, 0)
 16.
 17. # set up the block data structure
 18. player = pygame.Rect(300, 100, 40, 40)
 19. playerImage = pygame.image.load('player.png')
 20. playerStretchedImage = pygame.transform.scale(playerImage, (40, 40))
 21. foodImage = pygame.image.load('cherry.png')
 22. foods = []
 23. for i in range(20):
 24.       foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - 20),
random.randint(0, WINDOWHEIGHT - 20), 20, 20))
 25.
 26. foodCounter = 0
 27. NEWFOOD = 40
 28.
 29. # set up keyboard variables
 30. moveLeft = False
 31. moveRight = False
 32. moveUp = False
 33. moveDown = False
 34.
 35. MOVESPEED = 6
 36.
 37. # set up music
 38. pickUpSound = pygame.mixer.Sound('pickup.wav')
 39. pygame.mixer.music.load('background.mid')
 40. pygame.mixer.music.play(-1, 0.0)
 41. musicPlaying = True
 42.
 43. # run the game loop
 44. while True:
 45.     # check for the QUIT event
 46.     for event in pygame.event.get():
 47.         if event.type == QUIT:
 48.             pygame.quit()
 49.             sys.exit()
 50.         if event.type == KEYDOWN:
 51.             # change the keyboard variables
 52.             if event.key == K_LEFT or event.key == ord('a'):
 53.                 moveRight = False
 54.                 moveLeft = True
 55.             if event.key == K_RIGHT or event.key == ord('d'):
```

```
 56.                moveLeft = False
 57.                moveRight = True
 58.            if event.key == K_UP or event.key == ord('w'):
 59.                moveDown = False
 60.                moveUp = True
 61.            if event.key == K_DOWN or event.key == ord('s'):
 62.                moveUp = False
 63.                moveDown = True
 64.        if event.type == KEYUP:
 65.            if event.key == K_ESCAPE:
 66.                pygame.quit()
 67.                sys.exit()
 68.            if event.key == K_LEFT or event.key == ord('a'):
 69.                moveLeft = False
 70.            if event.key == K_RIGHT or event.key == ord('d'):
 71.                moveRight = False
 72.            if event.key == K_UP or event.key == ord('w'):
 73.                moveUp = False
 74.            if event.key == K_DOWN or event.key == ord('s'):
 75.                moveDown = False
 76.            if event.key == ord('x'):
 77.               player.top = random.randint(0, WINDOWHEIGHT - player.height)
 78.               player.left = random.randint(0, WINDOWWIDTH - player.width)
 79.            if event.key == ord('m'):
 80.                if musicPlaying:
 81.                    pygame.mixer.music.stop()
 82.                else:
 83.                    pygame.mixer.music.play(-1, 0.0)
 84.                musicPlaying = not musicPlaying
 85.
 86.        if event.type == MOUSEBUTTONUP:
 87.            foods.append(pygame.Rect(event.pos[0] - 10, event.pos[1]
- 10, 20, 20))
 88.
 89.    foodCounter += 1
 90.    if foodCounter >= NEWFOOD:
 91.        # add new food
 92.        foodCounter = 0
 93.        foods.append(pygame.Rect(random.randint(0, WINDOWWIDTH - 20),
random.randint(0, WINDOWHEIGHT - 20), 20, 20))
 94.
 95.    # draw the black background onto the surface
 96.    windowSurface.fill(BLACK)
```

```
 97.
 98.    # move the player
 99.    if moveDown and player.bottom < WINDOWHEIGHT:
100.        player.top += MOVESPEED
101.    if moveUp and player.top > 0:
102.        player.top -= MOVESPEED
103.    if moveLeft and player.left > 0:
104.        player.left -= MOVESPEED
105.    if moveRight and player.right < WINDOWWIDTH:
106.        player.right += MOVESPEED
107.
108.
109.    # draw the block onto the surface
110.    windowSurface.blit(playerStretchedImage, player)
111.
112.    # check if the block has intersected with any food squares.
113.    for food in foods[:]:
114.        if player.colliderect(food):
115.            foods.remove(food)
116.            player = pygame.Rect(player.left, player.top, player.width
+ 2, player.height + 2)
117.            playerStretchedImage = pygame.transform.scale(playerImage,
(player.width, player.height))
118.            if musicPlaying:
119.                pickUpSound.play()
120.
121.    # draw the food
122.    for food in foods:
123.        windowSurface.blit(foodImage, food)
124.
125.    # draw the window onto the screen
126.    pygame.display.update()
127.    mainClock.tick(40)
```

## 创建窗口和数据结构

程序中大部分的代码和第 19 章中的 Collision Detection 程序相同。我们只是重点介绍精灵和声音部分。

```
 12. pygame.display.set_caption('Sprites and Sound')
```

首先，我们在第 12 行把描述这个程序的字符串设置为标题栏的名称。给 pygame.display.set_caption()函数传递字符串'Sprites and Sound'。

```
17. # set up the block data structure
18. player = pygame.Rect(300, 100, 40, 40)
19. playerImage = pygame.image.load('player.png')
20. playerStretchedImage = pygame.transform.scale(playerImage, (40, 40))
21. foodImage = pygame.image.load('cherry.png')
```

我们将使用 3 个不同的变量来表示玩家，而不是像第 19 章那样只使用 1 个变量。

第 18 行 player 变量将存储一个 Rect 对象，记录玩家的位置以及玩家的大小。player 变量没有包含玩家的图像，只有玩家的大小和位置。在程序开始处，玩家左上角位于坐标(300, 100)处，一开始玩家有 40 个像素高和 40 个像素宽。

表示玩家的第 2 个变量是第 19 行代码中的 playerImage。pygame.image.load()函数接收了一个字符串参数，这是要加载的图像的文件名。返回值是一个 Surface 对象，把图像文件中的图形绘制到了该对象上。我们把这个 Surface 对象保存到 playerImage 中。

第 3 个变量在下一节中介绍。

## 20.4　pygame.transform.scale()函数

在第 20 行代码中，我们使用了 pygame.transform 模块中一个新的函数。pygame.transform.scale()函数可以缩小和放大一个精灵。第 1 个参数是在其上绘制了图像的 pygame.Surface 对象。第 2 个参数是一个元组，表示第 1 个参数中的图像的新的宽度和高度。pygame.transform. scale()函数返回一个 pygame.Surface 对象，图像以新的大小绘制于其上。我们在 playerImage 变量中保存了初始图像，而在 playerStretchedImage 变量中保存了拉伸后的图像。

第 121 行代码中，我们再次调用 pygame.image.load()函数来创建在其上绘制了樱桃图像的 Surface 对象。要确保在和 spritesAndSounds.py 文件相同的目录下有 player.png 文件和 cherry.png 文件，否则 Pygame 无法找到它们，并将给出错误。

### 20.4.1　创建音乐和声音

```
37. # set up music
38. pickUpSound = pygame.mixer.Sound('pickup.wav')
39. pygame.mixer.music.load('background.mid')
40. pygame.mixer.music.play(-1, 0.0)
41. musicPlaying = True
```

接下来需要加载声音文件。在 Pygame 中有两个声音模块。pygame.mixer 模块可以在游戏中播放简短音效。pygame.mixer.music 模块可以播放背景音乐。

调用 pygame.mixer.Sound()构造函数来创建一个 pygame.mixer.Sound 对象（简称

Sound 对象）。这个对象有一个 play()方法，调用该方法可以播放音效。

第 39 行调用 pygame.mixer.music.load()函数来加载背景音乐。第 40 行调用 pygame.mixer.music.play()函数开始播放背景音乐。第 1 个参数告诉 Pygame 在第一次播放音乐之后还要再播放几次背景音乐。所以，传入参数 5 将导致 Pygame 播放背景音乐 6 次。-1 是一个特殊值，将它作为第 1 个参数传入的话，会循环播放这首背景音乐。

pygame.mixer.music.play()函数的第 2 个参数是开始播放声音文件的位置。传入参数 0.0 将从头开始播放背景音乐。第 2 个参数是 2.5，表示将从音乐开头的 2.5 秒处开始播放背景音乐。

最后，musicPlaying 变量将有一个 Boolean 值，告诉程序是否应该播放背景音乐和音效。给玩家一个选项，让他们自己决定运行程序时是否播放声音，这会很棒。

## 20.4.2 切换和关闭声音

```
79.           if event.key == ord('m'):
80.               if musicPlaying:
81.                   pygame.mixer.music.stop()
82.               else:
83.                   pygame.mixer.music.play(-1, 0.0)
84.               musicPlaying = not musicPlaying
```

M 键将打开和关闭背景音乐。如果把 musicPlaying 设置为 True，那么现在正在播放背景音乐，我们应该通过调用 pygame.mixer.music.stop()来停止音乐。如果把 musicPlaying 设置为 False，那么当前没有播放背景音乐，应该通过调用 pygame.mixer.music.play()开始播放音乐。

最后，无论打开还是关闭音乐，我们都想切换 musicPlaying 中的值。切换一个 Boolean 值，意味着要把它的当前值设置为相反的值。如果 musicPlaying 当前为 True，代码行 musicPlaying = not musicPlaying 把这个变量设为 False；如果 musicPlaying 当前是 False，代码行 musicPlaying = not musicPlaying 把这个变量设为 True。想一下当开关电灯时是如何进行切换的：切换电灯开关，将其改为相反的设置。

## 20.4.3 把玩家绘制到窗口上

```
109.     # draw the block onto the surface
110.     windowSurface.blit(playerStretchedImage, player)
```

记住，playerStretchedImage 中存储的值是一个 Surface 对象。第 110 行代码把玩家的精灵绘制到窗口的 Surface 对象上（存储在 windowSurface 中）。

blit()方法的第 2 个参数是一个 Rect 对象，它指定了把精灵渲染到 Surface 对象上的何处。存储在 player 中的 Rect 对象，记录了玩家在窗口中的位置。

### 20.4.4　判断玩家是否和樱桃有碰撞

```
114.        if player.colliderect(food):
115.            foods.remove(food)
116.            player = pygame.Rect(player.left, player.top, player.width
+ 2, player.height + 2)
117.            playerStretchedImage = pygame.transform.scale(playerImage,
(player.width, player.height))
118.            if musicPlaying:
119.                pickUpSound.play()
```

这部分代码与前边程序中的代码类似。但是，有一些新的代码行。存储在 pickUpSound 变量中的 Sound 对象，调用了 play()方法。但是，只有把 musicPlaying 设置为 True 时（这意味着打开声音），才会这么做。

当玩家吃掉 1 个樱桃时，玩家的高度和宽度都会增加 2 个像素。在第 116 行中，新的 Rect 对象要比旧的 Rect 对象大两个像素，它将成为 player 的新值。

Rect 对象表示玩家的位置和大小，玩家的图片以 Surface 对象的形式存储在 playerStretchedImage 中。通过调用 pygame.transform.scale()来创建一张新的拉伸的图像。确保传递的是 playerImage 中初始的 Surface 对象，而不是 playerStretchedImage 中的对象。

拉伸一张图像经常会让它有点扭曲变形。如果一遍遍地重新拉伸一张已经拉伸过的图像，扭曲变形的会很快。但是，将原始图像拉伸到新的大小，则只会扭曲一次。这就是我们为什么把 playerImage 作为 pygame.transform.scale()函数的第 1 个参数。

### 20.4.5　在窗口中绘制樱桃

```
121.    # draw the food
122.    for food in foods:
123.        windowSurface.blit(foodImage, food)
```

在前边的程序中，我们调用了 pygame.draw.rect()函数为存储在 foods 列表中的每一个 Rect 对象绘制一个绿色的方块。然而，在这个程序中，我们想要绘制樱桃精灵。调用 blit()方法，并且把存储在 foodImage 中的 Surface（这是在其上绘制了樱桃图像的 Surface 对象）传递给函数。

food 变量（通过 for 循环中的每次迭代，将包含 foods 中的每一个 Rect 对象）告诉 blit()方法，在哪里绘制 foodImage。

## 20.5 本章小结

这个游戏添加了图像和声音。图像（叫做精灵）要比前边程序中使用的简单的形状看上去好很多。本章介绍的游戏还有背景音乐的播放，还可以播放音效。

精灵可以缩放（也就是拉伸）为更大或更小的尺寸。这样一来，我们可以以任意的大小显示精灵。在第 21 章所介绍的游戏中，这一功能会派上用场。

现在，我们知道了如何创建一个窗口、显示精灵、绘制基本图元、收集键盘和鼠标输入、播放声音以及实现碰撞检测，这就已经准备好在 Pygame 中创建一个图形游戏了。在下一章中，我们将把所有这些元素放在一起，来创建一个本书中最高级的游戏。

# 第 21 章　Dodger

**本章的主要内容：**

- pygame.FULLSCREEN 标志；
- Pygame 键盘值的常量；
- Rect 的 Move_ip( ) 方法；
- pygame.mouse.set_pos()函数；
- 实现游戏作弊的代码；
- 修改 Dodger 程序。

在第 18 章、第 19 章和第 20 章中，我们介绍了 Pygame 模块，并且展示了如何使用它的各项功能。在本章中，我们将使用这些学过的知识来创建一个叫做 Dodger 的图形游戏。

Dodger 游戏有一个玩家控制的小人（我们叫做玩家的角色），它必须避开从屏幕顶部落下的一大堆的敌人。玩家躲避敌人的时间越久，得到的分数越高。

为了好玩，我们还会为游戏加入一些作弊模式。如果玩家按下“x”键，每一个敌人的速度就会降低到超慢。如果玩家按下“z”键，敌人就会反转方向，沿着屏幕向上移动而不是往下落。

## 21.1　回顾 Pygame 的基本数据类型

我们来回顾一下 Pygame 中一些基本的数据类型：

- pygame.Rect——Rect 对象表示一个矩形空间的位置和大小。位置可以通过 Rect 对象的 topleft（或者 topright、bottomleft 和 bottomright）属性来确定。这些属性是表示 $x$ 坐标和 $y$ 坐标的整数的一个元组。矩形的大小可以通过 width 属性和 height 属性来决定，这些属性表示矩形区域的长和高是多少像素。Rect 对象有一个 colliderect()方法，用来检查矩形是否和其他 Rect 对象有碰撞。
- pygame.Surface——Surface 对象是带颜色的像素的区域。Surface 对象表示一个矩形图像，而 Rect 对象只表示一个矩形的空间和位置。Surface 对象有一个 blit()方法，它将一个 Surface 对象上的图像绘制到另一个 Surface 对象之上。由 pygame.display.set_mode()函数返回的 Surface 对象是特殊的，因为当调用 pygame.display.update()函数时，在该 Surface 对象上绘制的任何物体都会显示在用户的屏幕上。
- pygame.event.Event——当用户提供键盘、鼠标或其他类型的输入时，pygame.event 模块会创建 Event 对象。pygame.event.get()函数返回这些 Event 对象的一个列表。可以通过查看 Event 对象的 type 属性，来查看事件的类型。QUIT、KEYDOWN 和

MOUSEBUTTONUP 是一些事件类型的示例。

- pygame.font.Font——pygame.font 模块拥一个 Font 数据类型，用于表示 Pygame 中的文本的字体。传递给 pygame.font.SysFont()函数的参数是表示字体名称的一个字符串以及表示字体大小的一个整数。然而，通常传递 None 作为字体名称以获取默认系统字体。
- pygame.time.Clock——pygame.time 模块中的 Clock 对象有助于避免程序运行的过快。Clock 对象有一个 tick()方法，它接收的参数表示想要游戏运行速度是每秒钟多少帧（FPS）。FPS 越高，游戏运行越快。

输入如下代码，并且把它保存为 dodger.py。这个游戏还需要一些其他的图像文件和声音文件，你可以从 http://invpy.com/downloads 下载。

## 21.2 Dodger 的源代码

可以从 http://invpy.com/chap20 下载这些代码。如果输入这些代码后出现错误，请使用 http://invpy.com/diff/dodger 上的在线 diff 工具，把你的代码与书中的代码进行比较。

```
 1. import pygame, random, sys
 2. from pygame.locals import *
 3.
 4. WINDOWWIDTH = 600
 5. WINDOWHEIGHT = 600
 6. TEXTCOLOR = (255, 255, 255)
 7. BACKGROUNDCOLOR = (0, 0, 0)
 8. FPS = 40
 9. BADDIEMINSIZE = 10
10. BADDIEMAXSIZE = 40
11. BADDIEMINSPEED = 1
12. BADDIEMAXSPEED = 8
13. ADDNEWBADDIERATE = 6
14. PLAYERMOVERATE = 5
15.
16. def terminate():
17.     pygame.quit()
18.     sys.exit()
19.
20. def waitForPlayerToPressKey():
21.     while True:
22.         for event in pygame.event.get():
23.             if event.type == QUIT:
```

```
 24.                terminate()
 25.            if event.type == KEYDOWN:
 26.                if event.key == K_ESCAPE: # pressing escape quits
 27.                    terminate()
 28.                return
 29.
 30. def playerHasHitBaddie(playerRect, baddies):
 31.     for b in baddies:
 32.         if playerRect.colliderect(b['rect']):
 33.             return True
 34.     return False
 35.
 36. def drawText(text, font, surface, x, y):
 37.     textobj = font.render(text, 1, TEXTCOLOR)
 38.     textrect = textobj.get_rect()
 39.     textrect.topleft = (x, y)
 40.     surface.blit(textobj, textrect)
 41.
 42. # set up pygame, the window, and the mouse cursor
 43. pygame.init()
 44. mainClock = pygame.time.Clock()
 45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
 46. pygame.display.set_caption('Dodger')
 47. pygame.mouse.set_visible(False)
 48.
 49. # set up fonts
 50. font = pygame.font.SysFont(None, 48)
 51.
 52. # set up sounds
 53. gameOverSound = pygame.mixer.Sound('gameover.wav')
 54. pygame.mixer.music.load('background.mid')
 55.
 56. # set up images
 57. playerImage = pygame.image.load('player.png')
 58. playerRect = playerImage.get_rect()
 59. baddieImage = pygame.image.load('baddie.png')
 60.
 61. # show the "Start" screen
 62. drawText('Dodger', font, windowSurface, (WINDOWWIDTH/3), (WINDOWHEIGHT
/3))
 63. drawText('Press a key to start.', font, windowSurface, (WINDOWWIDTH /
3) - 30, (WINDOWHEIGHT / 3) + 50)
 64. pygame.display.update()
 65. waitForPlayerToPressKey()
```

```
66.
67.
68. topScore = 0
69. while True:
70.     # set up the start of the game
71.     baddies = []
72.     score = 0
73.     playerRect.topleft = (WINDOWWIDTH / 2, WINDOWHEIGHT - 50)
74.     moveLeft = moveRight = moveUp = moveDown = False
75.     reverseCheat = slowCheat = False
76.     baddieAddCounter = 0
77.     pygame.mixer.music.play(-1, 0.0)
78.
79.     while True: # the game loop runs while the game part is playing
80.         score += 1 # increase score
81.
82.         for event in pygame.event.get():
83.             if event.type == QUIT:
84.                 terminate()
85.
86.             if event.type == KEYDOWN:
87.                 if event.key == ord('z'):
88.                     reverseCheat = True
89.                 if event.key == ord('x'):
90.                     slowCheat = True
91.                 if event.key == K_LEFT or event.key == ord('a'):
92.                     moveRight = False
93.                     moveLeft = True
94.                 if event.key == K_RIGHT or event.key == ord('d'):
95.                     moveLeft = False
96.                     moveRight = True
97.                 if event.key == K_UP or event.key == ord('w'):
98.                     moveDown = False
99.                     moveUp = True
100.                if event.key == K_DOWN or event.key == ord('s'):
101.                    moveUp = False
102.                    moveDown = True
103.
104.            if event.type == KEYUP:
105.                if event.key == ord('z'):
106.                    reverseCheat = False
107.                    score = 0
108.                if event.key == ord('x'):
109.                    slowCheat = False
110.                    score = 0
```

```
111.                if event.key == K_ESCAPE:
112.                       terminate()
113.
114.                if event.key == K_LEFT or event.key == ord('a'):
115.                   moveLeft = False
116.                if event.key == K_RIGHT or event.key == ord('d'):
117.                   moveRight = False
118.                if event.key == K_UP or event.key == ord('w'):
119.                   moveUp = False
120.                if event.key == K_DOWN or event.key == ord('s'):
121.                   moveDown = False
122.
123.            if event.type == MOUSEMOTION:
124.                 # If the mouse moves, move the player where the cursor is.
125.                 playerRect.move_ip(event.pos[0] - playerRect.centerx,
event.pos[1] - playerRect.centery)
126.
127.        # Add new baddies at the top of the screen, if needed.
128.        if not reverseCheat and not slowCheat:
129.           baddieAddCounter += 1
130.        if baddieAddCounter == ADDNEWBADDIERATE:
131.           baddieAddCounter = 0
132.           baddieSize = random.randint(BADDIEMINSIZE, BADDIEMAXSIZE)
133.           newBaddie = {'rect': pygame.Rect(random.randint(0, WINDOWWIDTH-
baddieSize), 0 - baddieSize, baddieSize, baddieSize),
134.                        'speed': random.randint(BADDIEMINSPEED,
BADDIEMAXSPEED),
135.                          'surface':pygame.transform.scale(baddieImage,
(baddieSize, baddieSize)),
136.                           }
137.
138.           baddies.append(newBaddie)
139.
140.        # Move the player around.
141.        if moveLeft and playerRect.left > 0:
142.           playerRect.move_ip(-1 * PLAYERMOVERATE, 0)
143.        if moveRight and playerRect.right < WINDOWWIDTH:
144.           playerRect.move_ip(PLAYERMOVERATE, 0)
145.        if moveUp and playerRect.top > 0:
146.           playerRect.move_ip(0, -1 * PLAYERMOVERATE)
147.        if moveDown and playerRect.bottom < WINDOWHEIGHT:
148.           playerRect.move_ip(0, PLAYERMOVERATE)
149.
150.        # Move the mouse cursor to match the player.
151.        pygame.mouse.set_pos(playerRect.centerx, playerRect.centery)
```

```
152.
153.        # Move the baddies down.
154.        for b in baddies:
155.            if not reverseCheat and not slowCheat:
156.                b['rect'].move_ip(0, b['speed'])
157.            elif reverseCheat:
158.                b['rect'].move_ip(0, -5)
159.            elif slowCheat:
160.                b['rect'].move_ip(0, 1)
161.
162.        # Delete baddies that have fallen past the bottom.
163.        for b in baddies[:]:
164.            if b['rect'].top > WINDOWHEIGHT:
165.                baddies.remove(b)
166.
167.        # Draw the game world on the window.
168.        windowSurface.fill(BACKGROUNDCOLOR)
169.
170.        # Draw the score and top score.
171.        drawText('Score: %s' % (score), font, windowSurface, 10, 0)
172.        drawText('Top Score: %s' % (topScore), font, windowSurface, 10,
40)
173.
174.        # Draw the player's rectangle
175.        windowSurface.blit(playerImage, playerRect)
176.
177.        # Draw each baddie
178.        for b in baddies:
179.            windowSurface.blit(b['surface'], b['rect'])
180.
181.        pygame.display.update()
182.
183.        # Check if any of the baddies have hit the player.
184.        if playerHasHitBaddie(playerRect, baddies):
185.            if score > topScore:
186.                topScore = score # set new top score
187.            break
188.
189.        mainClock.tick(FPS)
190.
191.    # Stop the game and show the "Game Over" screen.
192.    pygame.mixer.music.stop()
193.    gameOverSound.play()
194.
195.     drawText('GAME OVER', font, windowSurface, (WINDOWWIDTH / 3),
```

```
(WINDOWHEIGHT / 3))
196.     drawText('Press a key to play again.', font, windowSurface,
(WINDOWWIDTH / 3) - 80, (WINDOWHEIGHT / 3) + 50)
197.     pygame.display.update()
198.     waitForPlayerToPressKey()
199.
200.     gameOverSound.stop()
```

当运行这个程序时，游戏如图 21-1 所示。

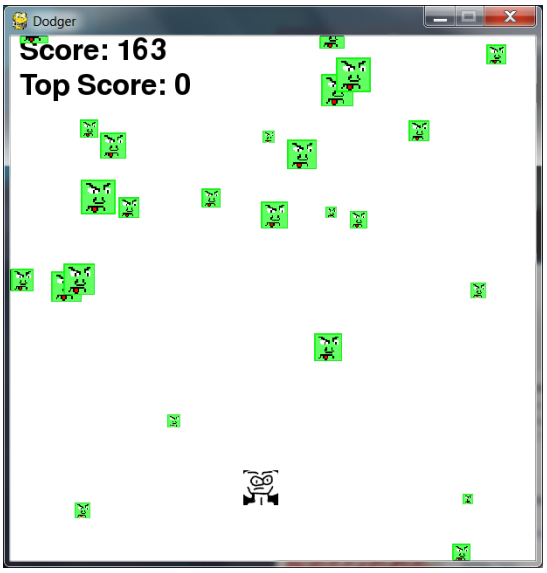



图 21-1　Dodger 程序的屏幕截图

## 21.2.1　导入模块

```
1. import pygame, random, sys
2. from pygame.locals import *
```

Dodger 游戏导入模块，和之前的 Pygame 程序使用的模块相同：pygame、random、sys 和 pygame.locals。pygame.locals 模块包含了几个供 Pygame 使用的常量，如事件类型（QUIT 和 KEYDOWN 等）和键盘按键（K_ESCAPE 和 K_LEFT）等。通过使用 from pygame.locals import * syntax 语句，我们可以在源代码中只输入 QUIT，而不必输入 pygame.locals.QUIT。

### 21.2.2 创建常量

```
 4. WINDOWWIDTH = 600
 5. WINDOWHEIGHT = 600
 6. TEXTCOLOR = (255, 255, 255)
 7. BACKGROUNDCOLOR = (0, 0, 0)
```

第 4 行到第 14 行中的常量，要比直接录入值更具有描述性。例如，语句 windowSurface.fill (BACKGROUNDCOLOR)要比语句 windowSurface.fill((0, 0, 0))更好理解。

通过修改常量可以很容易地修改游戏。通过修改第 4 行的 WINDOWWIDTH，可以自动地修改用到 WINDOWWIDTH 的每一处代码。如果使用值 600 来代替它，那么就必须修改代码中出现 600 的每一个地方。只修改一次常量中的值，则会更简单一些。

```
 8. FPS = 40
```

第 189 行代码调用了 mainClock.tick()方法，这会将游戏的速度减慢以确保可玩性。我们会为 mainClock.tick()函数传递一个整数，以便函数可以知道程序要暂停多久。这个整数值是想要游戏每秒钟运行的帧数。

一“帧”就是在游戏循环的单次迭代中在屏幕上所绘制的图形。可以把 FPS 设置为 40，总是调用 mainClock.tick(FPS)。然后，可以把 FPS 修改为一个更高的值，让游戏运行得更快；或者把 FPS 修改为一个更低的值，让游戏运行得更慢。

```
 9. BADDIEMINSIZE = 10
10. BADDIEMAXSIZE = 40
11. BADDIEMINSPEED = 1
12. BADDIEMAXSPEED = 8
13. ADDNEWBADDIERATE = 6
```

第 9 行到第 13 行设置了更多的常量，用来描述落下的敌人。敌人的宽度和高度均在 BADDIEMINSIZE 和 BADDIEMAXSIZE 之间。在游戏循环的每次迭代中，敌人从屏幕上落下的速率在每秒钟 BADDIEMINSPEED 到 BADDIEMINSPEED 多个像素之间。游戏循环的每经过 ADDNEWBADDIERATE 次迭代之后，将在窗口的顶部增加一个新的敌人。

```
14. PLAYERMOVERATE = 5
```

如果玩家的角色是移动的，在游戏循环的每次迭代中，PLAYERMOVERATE 将保存玩家的角色在窗口中移动的像素数。通过加大这个数字，就可以加快角色移动的速度。

### 21.2.3　定义函数

我们将为这个游戏创建几个函数：

```
16. def terminate():
17.     pygame.quit()
18.     sys.exit()
```

Pygame 需要调用 pygame.quit()和 sys.exit()。把这两个函数都放入到一个名为 terminate()的函数中。现在，只需要调用 terminate()函数，而不再单独调用 pygame.quit()和 sys.exit()函数。

```
20. def waitForPlayerToPressKey():
21.     while True:
22.         for event in pygame.event.get():
```

有时，我们想要暂停游戏，直到玩家按下一个键。创建一个新的名为 waitForPlayerToPressKey()的函数。在这个函数中，有一个无限循环，只有当接收到一个 KEYDOWN 或 QUIT 事件时，才会跳出该循环。在循环开始处，pygame.event.get()返回了要检查的事件对象的一个列表。

```
23.             if event.type == QUIT:
24.                 terminate()
```

当程序等待玩家按键的时候，如果玩家关闭了这个窗口，Pygame 将生成一个 QUIT 事件。在这种情况下，调用第 24 行代码的 terminate()函数。

```
25.             if event.type == KEYDOWN:
26.                 if event.key == K_ESCAPE: # pressing escape quits
27.                     terminate()
28.                 return
```

如果接收到一个 KEYDOWN 事件，那么应该先判断是否按下了 ESC 键。如果玩家按下的是 ESC 键，程序将终止。如果不是这种情况，那么执行将跳过第 27 行的 if 语句块并且直接到达 return 语句，这会退出 waitForPlayerToPressKey()函数。

如果没有生成一个 QUIT 或 KEYDOWN 事件，那么代码将保持循环。由于循环什么都没有做，这将使得游戏看上去像是已经冻结了，直到玩家按下一个键。

```
30. def playerHasHitBaddie(playerRect, baddies):
31.     for b in baddies:
32.         if playerRect.colliderect(b['rect']):
```

```
33.             return True
34.     return False
```

如果玩家的角色和一个敌人碰撞，playerHasHitBaddie()函数将返回 True。baddies 参数是“baddie”字典数据结构的一个列表。其中的每一个字典都有 rect 键，该键的值是表示敌人大小和位置的一个 Rect 对象。

playerRect 也是一个 Rect 对象。Rect 对象有一个名为 colliderect()的方法，如果 Rect 对象与传递给该函数的 Rect 对象发生碰撞，该方法返回 True。

第 31 行的 for 循环遍历了 baddies 中的每一个 baddie 字典。如果任何一个 baddie 与玩家的角色发生碰撞，那么 playerHasHitBaddie()函数将返回 True。如果负责遍历 baddies 列表中所有 baddie 的代码并没有检测到任何碰撞，该函数将返回 False。

```
36. def drawText(text, font, surface, x, y):
37.     textobj = font.render(text, 1, TEXTCOLOR)
38.     textrect = textobj.get_rect()
39.     textrect.topleft = (x, y)
40.     surface.blit(textobj, textrect)
```

在窗口上绘制文本包含了几个步骤。首先，第 37 行代码调用 render()方法创建了一个 Surface 对象，文本以特定的字体渲染其上。

接下来，需要知道 Surface 对象的大小和位置。可以通过 Surface 的 get_rect()方法获取 Rect 对象的这些信息。

在第 38 行代码中，Surface 对象的 get_rect()方法所返回的 Rect 对象，拥有宽度和高度信息的一个副本。第 39 行代码通过设置这个 Rect 对象的 topleft 属性，来改变它的位置。

最后，第 40 行将其上已经绘制了文本的 Surface 对象，绘制到了作为参数传递给 drawText()函数的 Surface 上。在 Pygame 中显示文本，要比直接调用 print()函数多花一些步骤。但是，如果把这些代码放入到一个名为 drawText()的函数中，那么要在屏幕上显示文本，只需要调用 drawText()函数即可。

### 21.2.4 初始化 Pygame 并设置窗口

现在已经完成了常量和函数的编写，下面开始调用创建窗口和时钟的 Pygame 函数。

```
42.# set up pygame, the window, and the mouse cursor
43. pygame.init()
44. mainClock = pygame.time.Clock()
```

第 43 行代码通过调用 pygame.init()函数创建了 Pygame。第 44 行创建一个 pygame.time.Clock()对象，并将其保存在 mainClock 变量中。这个对象将帮助我们防止程序运行的太快。

```
45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
```

第 45 行创建了一个新的 Surface 对象，用于在屏幕上显示窗口。可以通过传递 WINDOWWIDTH 常量和 WINDOWHEIGHT 常量的一个元组，来指定 Surface 对象（以及窗口）的宽和高。注意，pygame.display.set_mode()函数只接收 1 个参数，即 1 个元组。pygame.display.set_mode()函数的参数不是两个整数，而是两个整数组成的 1 个元组。

```
46. pygame.display.set_caption('Dodger')
```

第 46 行把窗口的标题设置为字符串“Dodger”。这个标题将显示于窗口顶端的标题栏上。

```
47. pygame.mouse.set_visible(False)
```

在 Dodger 中，鼠标的光标应该不可见。这是因为我们只想要鼠标能够移动屏幕上的角色，而不想让鼠标的光标妨碍到屏幕上的角色的图像。调用 pygame.mouse.set_visible(False)函数，将告诉 Pygame 让光标不可见。

## 21.3　全屏模式

pygame.display.set_mode()函数的第 2 个参数是可选的。我们可以传递 pygame.FULLSCREEN 常量，使得窗口占据整个屏幕，而不是显示为一个小窗口。看一下对 45 行代码的修改。

```
45. windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT),
pygame.FULLSCREEN)
```

WINDOWWIDTH 和 WINDOWHEIGHT 仍然是窗口的宽和高，但是将把图像拉伸以充满整个屏幕。尝试以全屏模式和非全屏模式来运行程序。

```
49. # set up fonts
50. font = pygame.font.SysFont(None, 48)
```

在第 49 行代码中，通过调用 pygame.font.SysFont()函数创建了一个 Font 对象以供使用。传递参数 None 将使用默认字体。传递 48，使得字体的大小为 48 点。

```
52. # set up sounds
53. gameOverSound = pygame.mixer.Sound('gameover.wav')
54. pygame.mixer.music.load('background.mid')
```

接下来，创建 Sound 对象，并且设置背景音乐。背景音乐将在游戏期间持续播放，但是只有当玩家输掉了游戏，才会播放 Sound 对象。

可以为这款游戏使用任何的.wav 或.mid 文件。本书的配套站点 http://invpy.com/downloads 有一些声音文件可供使用，或者，你可以为游戏使用自己的声音文件，只要文件名称是 gameover.wav 和 background.mid 就行（可以修改第 53 行和 54 行代码中的字符串以匹配文件名称）。

pygame.mixer.Sound()构造函数创建一个新的 Sound 对象，并将这个对象的一个引用保存到 gameOverSound 变量中。在你自己的游戏中，可以创建许多自己喜欢的 Sound 对象，每个 Sound 对象都有不同的声音文件。

pygame.mixer.music.load()函数加载了一个声音文件以播放背景音乐。这个函数没有返回任何对象，并且每次只能加载一个背景音乐文件。

```
 56. # set up images
 57. playerImage = pygame.image.load('player.png')
 58. playerRect = playerImage.get_rect()
 59. baddieImage = pygame.image.load('baddie.png')
```

接下来，我们将加载图像文件，以用于屏幕上的玩家和角色敌人。玩家角色的图像存储在 player.png 中，敌人的角色图像存储在 baddie.png 中。所有敌人角色看上去都是一样的，所以只要为它们准备一个图像文件就可以了。可以从本书的网站 http://invpy.com/downloads 下载这些图像。

### 21.3.1 显示开始界面

当游戏第一次启动时，在屏幕上显示“Dodger”名称。我们还想告诉用户，按下任意键可以开始游戏。在运行程序之后，显示这个屏幕，以便玩家有时间准备开始玩游戏。

```
 61. # show the "Start" screen
 62. drawText('Dodger', font, windowSurface, (WINDOWWIDTH / 3),
(WINDOWHEIGHT / 3))
 63. drawText('Press a key to start.', font, windowSurface, (WINDOWWIDTH /
3) - 30, (WINDOWHEIGHT / 3) + 50)
 64. pygame.display.update()
 65. waitForPlayerToPressKey()
```

第 62 行和第 63 行代码调用了 drawText()函数，并且为其传递了 5 个参数：

（1）想要显示的文本字符串；

（2）想要显示的字符串的字体；

（3）文本渲染于其上的 Surface 对象；

（4）在 Surface 对象上绘制文本的 $x$ 坐标；

（5）在 Surface 对象上绘制文本的 $y$ 坐标。

看上去好像为这个函数传递了很多参数，但是记住，通过调用这个函数避免了每次都要写 5 行代码。这缩短了程序，并且由于要检查的代码更少了，所以更容易找到 bug。

在循环时，waitForPlayerToPressKey()函数将暂停游戏，该函数不断循环，直到产生了一个 KEYDOWN 事件。然后，执行将跳出循环，程序继续运行。

### 21.3.2　主游戏代码的开始部分

```
68. topScore = 0
69. while True:
```

当程序开始运行时，topScore 变量中的值最初为 0。任何时候，当玩家输掉游戏并且得分大于当前的 topScore，就用这个更高的分数来替换 topScore。

从第 69 行开始的无限循环，技术上来讲不是“游戏循环”。当程序运行时，游戏循环处理事件并绘制窗口。而每次玩家开始一个新的游戏时，这个 while 循环将进行迭代。当玩家输了并且游戏重置，程序的执行将跳转回到第 69 行的循环。

```
70.     # set up the start of the game
71.     baddies = []
72.     score = 0
```

在开始时，我们想要把 baddies 设置为一个空的列表。baddies 变量是包含如下键的字典对象的一个列表：

'rect'——描述敌人位置和大小的 Rect 对象。

'speed'——敌人从屏幕落下的速度。这个整数表示游戏循环中每次迭代的像素。

'surface'——缩放后的敌人图像绘制于其上的 Surface 对象。这个 Surface 对象绘制于 pygame.display.set_mode()函数所返回的 Surface 对象之上。

第 72 行代码把玩家的分数重置为 0。

```
73.     playerRect.topleft = (WINDOWWIDTH / 2, WINDOWHEIGHT - 50)
```

玩家的起始位置是屏幕的中间并且距离底部有 50 个像素的距离。第 73 行代码中的元组的第 1 个元素是玩家左边距的 $x$ 坐标，第 2 个元素是上边距的 $y$ 坐标。

```
74.     moveLeft = moveRight = moveUp = moveDown = False
75.     reverseCheat = slowCheat = False
76.     baddieAddCounter = 0
```

把移动变量 moveLeft、moveRight、moveUp 和 moveDown 设置为 False。把变量 reverseCheat 和 slowCheat 也设置为 False。只有当玩家分别按下“z”键和“x”键，开启

作弊模式时，才会把这两个变量设置为 True。

变量 baddieAddCounter 是一个计数器，它告诉程序何时在屏幕顶部增加一个新的敌人。游戏循环每迭代一次，baddieAddCounter 中的值都会加 1。

当 baddieAddCounter 等于 ADDNEWBADDIERATE 时（会在后边的第 130 行代码处做这个判断），把 baddieAddCounter 计数器重置为 0，并且在屏幕的顶部增加一个新的敌人。

```
77.     pygame.mixer.music.play(-1, 0.0)
```

在第 77 行代码中，通过调用 pygame.mixer.music.play()函数，开始播放背景音乐。第 1 个参数是音乐要重复的次数。-1 是个特定的值，它告诉 Pygame 想要音乐不停地重复播放。

第 2 个参数是一个浮点数，表示想要音乐从第几秒开始播放。传递 0.0，表示音乐要从头开始播放。

## 21.4 游戏循环

游戏循环的代码通过修改玩家和敌人的位置、处理 Pygame 生成的事件以及在屏幕上绘制游戏世界，来不断地更新游戏世界的状态。所有这些事情会在 1 秒钟内发生很多次，这使得游戏“实时”地运行。

```
79.     while True: # the game loop runs while the game part is playing
80.         score += 1 # increase score
```

第 79 行代码是主程序循环的开始。第 80 行代码在游戏循环的每次迭代中增加了玩家的分数。玩家保持不输掉游戏的时间越长，他们的分数也越高。当玩家输掉游戏或退出程序时，循环才会退出。

## 21.5 事件处理

有 4 种不同类型的事件要处理：QUIT、KEYDOWN、KEYUP 和 MOUSEMOTION。

```
82.         for event in pygame.event.get():
83.             if event.type == QUIT:
84.                 terminate()
```

第 82 行代码是事件处理代码的开始。它调用 pygame.event.get()函数，该函数返回 Event 对象的一个列表。每个 Event 对象表示自上次对 pygame.event.get()函数调用后产生的一个事件。代码将查看事件对象的 type 属性，以查看事件是什么类型并相应地处理事件。

如果 Event 对象的 type 属性等于 QUIT，则用户关闭了程序。QUIT 常量是从 pygame.locals 模块中导入的。

```
 86.           if event.type == KEYDOWN:
 87.               if event.key == ord('z'):
 88.                   reverseCheat = True
 89.               if event.key == ord('x'):
 90.                   slowCheat = True
```

如果事件的类型是 KEYDOWN，那么玩家按下了一个按键。键盘事件的 Event 对象还有一个 key 属性，它会设置为被按下的那个按键的整数顺序编码。ord()函数将返回传递给它的字母的顺序编码。

例如，第 87 行代码使用 event.key == ord('z')语句来判断是否按下了“z”键。如果条件为 True，将变量 reverseCheat 设置为 Ture，表示激活了反向作弊模式。第 89 行判断是否按下了“x”键以激活慢速作弊模式。

Pygame 的键盘事件总是使用小写字母而不是大写字母的顺序编码。总是使用 event.key == ord('z')，而不是 event.key == ord('Z')。如果不这样做，即便没有按下该键，程序也可能开始起作用。

```
 91.               if event.key == K_LEFT or event.key == ord('a'):
 92.                   moveRight = False
 93.                   moveLeft = True
 94.               if event.key == K_RIGHT or event.key == ord('d'):
 95.                   moveLeft = False
 96.                   moveRight = True
 97.               if event.key == K_UP or event.key == ord('w'):
 98.                   moveDown = False
 99.                   moveUp = True
100.               if event.key == K_DOWN or event.key == ord('s'):
101.                   moveUp = False
102.                   moveDown = True
```

第 91 行到第 102 行代码，判断事件是否是由于玩家按下一个方向键或 WASD 键而产生的。并不是键盘上的每个键都有顺序编码，例如，方向键和 ESC 键就没有顺序编码。pygame.locals 模块提供了用于替代的常量。

第 91 行用代码 event.key == K_LEFT 来判断玩家是否按下了左键。注意，按下一个方向键，不仅要把一个移动变量设置为 True，而且要把其相反方向的移动变量设置为 False。

例如，如果按下左键，那么第 93 行代码会把 moveLeft 设置为 True，但是也会把 moveRight 设置为 False。这就避免了玩家把程序搞晕了，以至于玩家认为角色应该同时朝两个相反方向移动。

表 21-1 列出了与键盘相关的 Event 对象的 key 属性的常用常量。

**表 21-1 键盘按键的常量**

| Pygame 常量 | 按键 | Pygame 常量 | 按键 |
|---|---|---|---|
| K_LEFT | 左键 | K_HOME | Home 键 |
| K_RIGHT | 右键 | K_END | End 键 |
| K_UP | 上键 | K_PAGEUP | 向上翻页键 |
| K_DOWN | 下键 | K_PAGEDOWN | 向下翻页键 |
| K_ESCAPE | Esc 键 | K_F1 | F1 键 |
| K_BACKSPACE | 退格键 | K_F2 | F2 键 |
| K_TAB | Tab 键 | K_F3 | F3 键 |
| K_RETURN | 回车键 | K_F4 | F4 键 |
| K_SPACE | 空格键 | K_F5 | F5 键 |
| K_DELETE | 删除键 | K_F6 | F6 键 |
| K_LSHIFT | 左 Shift 键 | K_F7 | F7 键 |
| K_RSHIFT | 右 Shift 键 | K_F8 | F8 键 |
| K_LCTRL | 左 Ctrl 键 | K_F9 | F9 键 |
| K_RCTRL | 右 Ctrl 键 | K_F10 | F10 键 |
| K_LALT | 左 Alt 键 | K_F11 | F11 键 |
| K_RALT | 右 Alt 键 | K_F12 | F12 键 |

```
104.            if event.type == KEYUP:
105.                if event.key == ord('z'):
106.                    reverseCheat = False
107.                    score = 0
108.                if event.key == ord('x'):
109.                    slowCheat = False
110.                    score = 0
```

当玩家停止按键并且释放该键时，会产生 KEYUP 事件。拥有 KEYUP 类型的 Event 对象也有一个 key 属性，就像 KEYDOWN 事件一样。

第 105 行判断玩家是否释放了“z”键，如释放了该键，将解除反向作弊模式。在这种情况下，第 106 行把 reverseCheat 设置为 False，第 107 行把得分重置为 0。重置分数是为了不鼓励玩家使用这种作弊模式。

第 108 行到 110 行代码对“x”键和慢速作弊模式做了相同的处理。当释放了“x”键，把 slowCheat 设置为 False，把玩家的得分重置为 0。

```
111.                if event.key == K_ESCAPE:
112.                    terminate()
```

在游戏运行中的任何时侯，玩家都可以按下键盘上的 Esc 键来退出游戏。第 14 行代码通过查看 event.key == K_ESCAPE 语句来判断是否释放了 Esc 键。如果是的，第 112 行代码调用 terminate()函数来退出程序。

```
114.                if event.key == K_LEFT or event.key == ord('a'):
115.                    moveLeft = False
116.                if event.key == K_RIGHT or event.key == ord('d'):
117.                    moveRight = False
118.                if event.key == K_UP or event.key == ord('w'):
119.                    moveUp = False
120.                if event.key == K_DOWN or event.key == ord('s'):
121.                    moveDown = False
```

第 114 行到 121 行判断玩家是否停止按住方向键或 WASD 键之一。在这种情况下，代码会将相应的移动变量设置为 False。

例如，如果玩家按住左方向键，那么第 93 行把 moveLeft 设置为 True。当释放该键时，第 114 行代码把条件计算为 True，并且把变量 moveLeft 设置为 Fasle。

## 21.6　move_ip()方法

```
123.            if event.type == MOUSEMOTION:
124.                # If the mouse moves, move the player where the cursor is.
125.                playerRect.move_ip(event.pos[0] - playerRect.centerx,
event.pos[1] - playerRect.centery)
```

现在，我们已经处理了键盘事件，接下来处理可能产生的任何鼠标事件。如果玩家点击鼠标按键，Dodger 游戏不会做任何事情，但是当玩家移动鼠标的时候，游戏会做出响应。这就使得玩家在游戏中可以有两种方法来控制玩家角色：键盘和鼠标。

当鼠标移动时，会产生 MOUSEMOTION 事件。MOUSEMOTION 类型的 Event 对象，也有一个名为 pos 的属性，表示鼠标事件的位置。pos 属性保存了一个元组，是鼠标光标在窗口中移动到的位置的 X 坐标和 Y 坐标。如果事件的类型是 MOUSEMOTION，玩家的角色将移动到鼠标光标的位置。

Rect 对象的 move_ip()方法将会水平地或垂直地将 Rect 对象的位置移动一定的像素数

目。例如，playerRect.move_ip(10, 20)将 Rect 对象向右移动 10 个像素，向下移动 20 个像素。要将 Rect 对象向左或向上移动，传递负值作为参数即可。例如，playerRect.move_ip(-5, -15)将 Rect 对象向左移动 5 个像素，向上移动 15 个像素。

move_ip()方法末尾的“ip”表示“in place（就地）”。这是因为这个方法修改的是 Rect 对象本身，而不是返回一个修改后的新的 Rect 对象。还有一个 move()方法，它没有修改 Rect 对象，而是在新的位置创建一个新的 Rect 对象并返回。

### 21.6.1 增加敌人

```
127.        # Add new baddies at the top of the screen, if needed.
128.        if not reverseCheat and not slowCheat:
129.            baddieAddCounter += 1
```

在游戏循环的每一次迭代中，变量 baddieAddCounter 会加 1。只有在作弊模式未启用才会这么做。记住，只要分别按下“z”键和“x”键，就会把 reverseCheat 和 slowCheat 设置为 True。

当按下这些键时，baddieAddCounter 就不会增加。因此，也不会有新的敌人会出现在屏幕的顶部。

```
130.        if baddieAddCounter == ADDNEWBADDIERATE:
131.            baddieAddCounter = 0
132.            baddieSize = random.randint(BADDIEMINSIZE, BADDIEMAXSIZE)
133.            newBaddie = {'rect': pygame.Rect(random.randint(0,
WINDOWWIDTH-baddieSize), 0 - baddieSize, baddieSize, baddieSize),
134.                         'speed': random.randint(BADDIEMINSPEED,
BADDIEMAXSPEED),
135.                           'surface':pygame.transform.scale(baddieImage,
(baddieSize, baddieSize)),
136.                         }
```

当 baddieAddCounter 等于 ADDNEWBADDIERATE 中的值时，就会在屏幕的顶部增加一个新的敌人。首先，会把 baddieAddCounter 计数器重置为 0。

第 132 行生成一个表示敌人的大小的值，以像素为单位。这个大小将是 BADDIEMINSIZE 和 BADDIEMAXSIZE 之间的一个随机整数，在第 9 行和第 10 行，把这两个常量分别设置为 10 和 40。

第 133 行创建了一个新的 baddie 数据结构。记住，baddie 的数据结构是带有键'rect'、'speed'和'surface'的一个字典。'rect'键保存了一个 Rect 对象的引用，这个 Rect 对象存储了敌人的位置和大小。pygame.Rect()构造函数有 4 个参数：区域的上边缘的 X 坐标、区域的左边缘的 Y 坐标、以像素为单位的宽度和以像素为单位的高度。

敌人需要随机地出现在窗口的顶部，所以传入 random.randint(0, WINDOWWIDTH-baddieSize)作为左边缘的 X 坐标的参数。传入 WINDOWWIDTH-baddieSize 而不是 WINDOWWIDTH，是因为这个值表示敌人的左边缘。如果敌人的左边缘超出了屏幕的右边缘，那么敌人身体的一部分将会在窗口之外并且不可见。

敌人的底边刚好在窗口的顶边之上。窗口的顶边的 Y 坐标是 0。要把敌人的底边放置在这里，把敌人的顶边设置为 0-baddieSize。

敌人的宽和高应该是相同的（图像是正方形的），所以把 baddieSize 作为第 3 个参数和第 4 个参数。

把'speed'键对应的值，设置为敌人在屏幕上向下移动的速度。把它设置为 BADDIEMINSPEED 和 BADDIEMAXSPEED 之间的一个随机整数。

```
138.            baddies.append(newBaddie)
```

第 138 行会把新创建的 baddie 数据结构添加到 boddie 数据结构的列表中。程序将使用这个列表来判断玩家是否和任何一个敌人有碰撞，并且通过这个列表获知在窗口的什么位置绘制敌人。

## 21.6.2　移动玩家角色

```
140.        # Move the player around.
141.        if moveLeft and playerRect.left > 0:
142.            playerRect.move_ip(-1 * PLAYERMOVERATE, 0)
```

当 Pygame 触发了 KEYDOWN 和 KEYUP 事件时，把 4 个移动变量 moveLeft、moveRight、moveUp 和 moveDown 分别设置为 True 和 False。

如果玩家的角色向左移动，并且玩家的角色的左边缘大于 0（0 是窗口的左边缘），那么应该把 playerRect 向左移动。

我们总是把 playerRect 对象移动 PLAYERMOVERATE 多个像素。要得到一个整数的负数形式，将其乘以-1。在第 142 行，由于 5 存储在 PLAYERMOVERATE 中，所以表达式-1 * PLAYERMOVERATE 的结果是-5。

因此，调用 playerRect.move_ip(-1 * PLAYERMOVERATE, 0)函数，会改变 playerRect 的位置，将其从当前位置向左移动 5 个像素。

```
143.        if moveRight and playerRect.right < WINDOWWIDTH:
144.            playerRect.move_ip(PLAYERMOVERATE, 0)
145.        if moveUp and playerRect.top > 0:
146.            playerRect.move_ip(0, -1 * PLAYERMOVERATE)
147.        if moveDown and playerRect.bottom < WINDOWHEIGHT:
148.            playerRect.move_ip(0, PLAYERMOVERATE)
```

第 143 行到 148 行的代码，对其他 3 个方向（右边、上边和下边），做了同样的处理。在 143 行到 148 行的代码中，3 条 if 语句中的每一条，都会判断它们的移动变量是否为 True，以及玩家的 Rect 对象的边缘是否在这个窗口中。然后调用 move_ip()函数来移动这个 Rect 对象。

## 21.7 pygame.mouse.set_pos()函数

```
150.        # Move the mouse cursor to match the player.
151.        pygame.mouse.set_pos(playerRect.centerx, playerRect.centery)
```

第 151 行把鼠标光标移动到和玩家角色相同的位置。pygame.mouse.set_pos()函数把鼠标光标移动到传递给该函数的 X 坐标和 Y 坐标。因此鼠标光标和玩家的角色总是在相同的位置。

具体地讲，光标将在角色的 Rect 对象的正中心，因为把 centerx 属性和 centery 属性作为坐标传递给了 playerRect 函数。尽管第 47 行代码调用 pygame.mouse.set_visible(False)语句使得鼠标光标不可见，但是光标仍然是存在的，并且可以移动它。

```
153.        # Move the baddies down.
154.        for b in baddies:
```

现在遍历 baddies 列表中的每一个 baddie 数据结构，使它们向下移动一点点。

```
155.            if not reverseCheat and not slowCheat:
156.                b['rect'].move_ip(0, b['speed'])
```

如果没有激活任何的作弊模式，那么向下移动敌人位置的像素数目就等于它的速度，该值存储在'speed'键中。

### 21.7.1 实现作弊模式

```
157.            elif reverseCheat:
158.                b['rect'].move_ip(0, -5)
```

如果激活了反向作弊模式，那么敌人应该以 5 个像素的速度向上移动。传入-5 作为第 2 个参数，将把 Rect 对象向上移动 5 个像素。

```
159.            elif slowCheat:
160.                b['rect'].move_ip(0, 1)
```

如果激活了慢速作弊模式，那么敌人应该向下移动，但是移动的速度很慢，在每次游戏

循环迭代中只是向下移动 1 个像素。当激活了慢速作弊模式，敌人的正常速度（存储在 baddie 数据结构的'speed'键中）会被忽略。

### 21.7.2　删除敌人

```
162.        # Delete baddies that have fallen past the bottom.
163.        for b in baddies[:]:
```

任何跌落到窗口底边之下的敌人都应该从 baddies 列表中删除。记住，当遍历一个列表时，不能增加或删除元素以修改列表中的内容。所以，for 循环遍历的不是这个 baddies 列表，而是遍历这个 baddies 列表的一个副本。使用空白分片操作符[:]来创建这个副本。

在遍历 baddies[:]的时候，第 163 行的 for 循环使用变量 b 表示当前元素。

```
164.            if b['rect'].top > WINDOWHEIGHT:
165.                baddies.remove(b)
```

我们来计算一下表达式 b['rect'].top。b 是 baddies[:]列表中当前的 baddie 数据结构。列表中的每一个 baddie 数据结构都是带有一个'rect'键的字典，该键存储了一个 Rect 对象。所以，b['rect']是敌人的 Rect 对象。

最后，top 属性是矩形区域的顶边的 Y 坐标。记住，Y 坐标是向下增加的。所以，b['rect'].top > WINDOWHEIGHT，将判断敌人的顶边是否低于窗口的底边。

如果这个条件为 True，那么第 165 行代码从 baddies 列表中删除 baddie 数据结构。

### 21.7.3　绘制窗口

把所有数据结构都修改完之后，使用 Pygame 的图像函数来绘制游戏世界。由于每秒钟会执行多次游戏循环，在新的位置绘制敌人和玩家，会使得它们的移动看上去更平滑而自然。

```
167.        # Draw the game world on the window.
168.        windowSurface.fill(BACKGROUNDCOLOR)
```

首先，在绘制任何事物之前，第 168 行将整个屏幕清除，擦去之前绘制的所有事物。

记住，windowSurface 中的 Surface 对象是一个特殊的 Surface 对象，因为它是 pygame.display.set_mode()函数返回的一个 Surface 对象。因此，在调用 pygame.display. update()函数之后，在该 Surface 对象上绘制的任何事物都将出现在屏幕上。

### 21.7.4 绘制玩家的得分

```
170.        # Draw the score and top score.
171.        drawText('Score: %s' % (score), font, windowSurface, 10, 0)
172.        drawText('Top Score: %s' % (topScore), font, windowSurface, 10,
40)
```

第 171 行和第 172 行绘制了分数文本，把最高得分绘制到窗口的左上角。表达式'Score: %s' % (score)使用字符串插值，将 score 变量中的值插入到字符串中。

将这个字符串、存储在 font 变量中的 Font 对象、要在其上绘制文本的 Surface 对象、文本要放置的位置的 X 坐标和 Y 坐标都传递给 drawText()函数。drawText()函数将负责调用 render()方法和 blit()方法。

对于最高得分，做了同样的处理。为 Y 坐标传递 40 而不是 0，以便最高得分文本出现在得分文本的下方。

### 21.7.5 绘制玩家角色

```
174.        # Draw the player's rectangle
175.        windowSurface.blit(playerImage, playerRect)
```

玩家的信息保存在两个不同的变量中。playerImage 是一个 Surface 对象，它包含了构成玩家角色的形象的所有彩色像素。playerRect 是存储了玩家角色的大小和位置信息的一个 Rect 对象。

blit()方法将玩家角色的图像（存储在 playerImage 中）绘制到了 windowSurface 上的 playerRect 位置。

```
177.        # Draw each baddie
178.        for b in baddies:
179.            windowSurface.blit(b['surface'], b['rect'])
```

第 178 行的 for 循环在 windowSurface 对象上绘制了每个敌人。baddies 列表中的每一个元素都是一个字典。这个字典的'surface'键和'rect'键，分别对应包含敌人图像的 Surface 对象以及带有位置和大小信息的 Rect 对象。

```
181.        pygame.display.update()
```

现在已经将所有内容都绘制到了 windowSurface，通过调用 pygame.display.update()函数，将这个 Surface 对象绘制到屏幕上。

### 21.7.6　碰撞检测

```
183.        # Check if any of the baddies have hit the player.
184.        if playerHasHitBaddie(playerRect, baddies):
185.            if score > topScore:
186.                topScore = score # set new top score
187.            break
```

第 184 行代码通过调用 playerHasHitBaddie()函数，判断玩家是否与任何的敌人发生了碰撞。如果玩家角色与 baddies 列表中的任何敌人发生碰撞，该函数返回 True。否则，该函数返回 False。

如果玩家角色碰到一个敌人，并且如果当前分数大于最高分，第 185 行和 186 行会更新最高分。然后程序会在第 187 行跳出游戏循环。程序的执行将移动到第 191 行。

```
189.        mainClock.tick(FPS)
```

要避免计算机在游戏循环中运行的太快（这将会导致玩家跟不上游戏的节奏），调用 mainClock.tick()函数暂停一个短暂的时间。这个暂停时间将会足够长，以确保每秒钟迭代游戏循环 40 次（该值存储在 FPS 变量中）。

### 21.7.7　游戏结束屏幕

```
191.    # Stop the game and show the "Game Over" screen.
192.    pygame.mixer.music.stop()
193.    gameOverSound.play()
```

当玩家输掉游戏时，游戏停止播放背景音乐，并且播放“游戏结束”的声音效果。第 193 行调用 pygame.mixer.music 模块中的 stop()函数，来停止背景音乐。第 193 行调用存储在 gameOverSound 中的 Sound 对象的 play()方法。

```
195.    drawText('GAME OVER', font, windowSurface, (WINDOWWIDTH / 3),
(WINDOWHEIGHT / 3))
196.    drawText('Press a key to play again.', font, windowSurface,
(WINDOWWIDTH / 3) - 80, (WINDOWHEIGHT / 3) + 50)
197.    pygame.display.update()
198.    waitForPlayerToPressKey()
```

第 195 行和第 196 行代码调用 drawText()函数，将“game over”文本绘制到 windowSurface 对象。第 197 行调用 pygame.display.update()函数将这个 Surface 对象绘制到屏幕上。在显

示了“game over”文本之后，通过调用 waitForPlayerToPressKey()函数，游戏停止，直到玩家按下键盘。

```
200.    gameOverSound.stop()
```

玩家按键之后，程序执行将会从第 198 行代码调用的 waitForPlayerToPressKey()函数返回。根据玩家按键时间的长短，“game over”声音效果可能仍在播放也可能不继续播放。在开始一个新的游戏前，要停止这个声音效果，因此第 200 行调用了 gameOverSound.stop()函数。

## 21.8 修改 Dodger 游戏

这就是我们的图形化游戏。你可能觉得这个游戏太简单或者太难。但是这个游戏很容易修改，因为我们花了工夫，使用的是常量，而不是直接输入值。现在，要修改游戏的话，我们只需要修改在常量中设置的值。

例如，如果想要游戏整体运行得更慢一点，把第 8 行的 FPS 变量改为更小的一个值，例如 20。这会使得敌人和玩家的角色移动得更慢一点，因为游戏循环一秒只会执行 20 次，而不是 40 次。

如果只是想要降低敌人的速度，而不是玩家的速度，那么把 BADDIEMAXSPEED 修改为一个更小的值，例如 4。这会使得在每次游戏迭代中所有敌人都移动 1 个像素（BADDIEMINSPEED 中的值）到 4 个像素之间，而不是 1 个像素到 8 个像素。

如果想要游戏有更少但是更大的敌人，而不想要众多的、快速的敌人，那么把 ADDNEWBADDIERATE 增加到 12、把 BADDIEMINSIZE 增加到 40 并且把 BADDIEMAXSIZE 增加到 80。现在，每 12 次游戏循环迭代才会增加新的敌人，而不再是每 6 次游戏循环迭代就增加新的敌人，这比以前要减少一半的敌人。但是为了保证游戏的趣味性，敌人现在比之前的敌人更大。

当基本的游戏保持相同时，可以修改任何的常量，以便对游戏的行为产生显著的影响。不断为这些常量尝试新的值，直到找到一套最喜欢的值。

## 21.9 本章小结

和之前基于文本的游戏不同，Dodger 看上去真得像是我们通常玩的各种现代化的计算机游戏。它有图形和音乐，并且使用了鼠标。Pygame 提供了函数和数据类型作为构建模块，作为程序员的你，需要把它们组合起来以创建有趣的交互式游戏。

而所有这些，都是因为你知道如何指挥计算机逐步地来实现它。你可以讲计算机的语言，并让计算机做大量的数学运算和绘图。这是一种很有用的技能，我希望你继续学习关于

Python 编程的更多知识（还有很多知识要学习!）。

有几个网站可以帮助你学习更多的 Python 编程知识：

- http://reddit.com/r/inventwithpython——这个网站有几个用户可以为你提供本书中的素材。
- http://inventwithpython.com——本书的网站，它包含了这些程序的所有源代码以及一些额外信息。这个网站还拥有 Pygame 程序中用到的一些图形文件和声音文件。
- http://inventwithpython.com/pygame——我的第 2 本书，《Python 和 Pygame 游戏开发指南》，介绍了 Pygame 的更多细节。
- http://inventwithpython.com/hacking——我的第 3 本书《Python 密码学编程》，它介绍了更多密码学和密码破解程序。
- http://inventwithpython.com/automate——我的第 4 本书《Python 编程快速上手——让繁琐工作自动化》，它介绍了实际的编程技巧。
- http://python.org/doc/ ——所有 Python 模块和函数的更多 Python 教程和文档。
- http://pygame.org/docs/ ——Pygame 模块和函数的完整文档。
- al@inventwithpython.com ——我的邮箱地址。如果对于本书或者 Python 编程有什么问题，请给我发邮件。

或者可以从网上找到更多 Python 相关的知识。到 http://google.com 网站搜索“Python programming”和“Python tutorials”，可以找到介绍更多 Python 编程知识的网站。

现在，来创建你自己的游戏吧。祝你好运!

# 欢迎来到异步社区！

## 异步社区的来历

异步社区 (www.epubit.com.cn) 是人民邮电出版社旗下 IT 专业图书旗舰社区，于 2015 年 8 月上线运营。

异步社区依托于人民邮电出版社 20 余年的 IT 专业优质出版资源和编辑策划团队，打造传统出版与电子出版和自出版结合、纸质书与电子书结合、传统印刷与 POD 按需印刷结合的出版平台，提供最新技术资讯，为作者和读者打造交流互动的平台。


## 社区里都有什么？

### 购买图书

我们出版的图书涵盖主流 IT 技术，在编程语言、Web 技术、数据科学等领域有众多经典畅销图书。社区现已上线图书 1000 余种，电子书 400 多种，部分新书实现纸书、电子书同步出版。我们还会定期发布新书书讯。

### 下载资源

社区内提供随书附赠的资源，如书中的案例或程序源代码。

另外，社区还提供了大量的免费电子书，只要注册成为社区用户就可以免费下载。

### 与作译者互动

很多图书的作译者已经入驻社区，您可以关注他们，咨询技术问题；可以阅读不断更新的技术文章，听作译者和编辑畅聊好书背后有趣的故事；还可以参与社区的作者访谈栏目，向您关注的作者提出采访题目。

## 灵活优惠的购书

您可以方便地下单购买纸质图书或电子图书，纸质图书直接从人民邮电出版社书库发货，电子书提供多种阅读格式。

对于重磅新书，社区提供预售和新书首发服务，用户可以第一时间买到心仪的新书。

用户帐户中的积分可以用于购书优惠。100 积分 =1 元，购买图书时，在 0 使用积分 里填入可使用的积分数值，即可扣减相应金额。

### 纸电图书组合购买

社区独家提供纸质图书和电子书组合购买方式，价格优惠，一次购买，多种阅读选择。


## 社区里还可以做什么？

### 提交勘误

您可以在图书页面下方提交勘误，每条勘误被确认后可以获得 100 积分。热心勘误的读者还有机会参与书稿的审校和翻译工作。

### 写作

社区提供基于 Markdown 的写作环境，喜欢写作的您可以在此一试身手，在社区里分享您的技术心得和读书体会，更可以体验自出版的乐趣，轻松实现出版的梦想。

如果成为社区认证作译者，还可以享受异步社区提供的作者专享特色服务。

### 会议活动早知道

您可以掌握 IT 圈的技术会议资讯，更有机会免费获赠大会门票。

## 加入异步

扫描任意二维码都能找到我们：

| 异步社区 | 微信服务号 | 微信订阅号 | 官方微博 | QQ 群：368449889 |
|---|---|---|---|---|

社区网址：www.epubit.com.cn

官方微信：异步社区

官方微博：@ 人邮异步社区，@ 人民邮电出版社 - 信息技术分社

投稿 & 咨询：contact@epubit.com.cn

# Python 游戏编程快速上手

本书通过编写一个个小巧、有趣的游戏来教授Python编程。

本书包含了14个游戏程序和示例，贯穿其中，介绍了Python基础知识、数据类型、函数、流程控制、程序调试、流程图、字符串操作、列表和字典、图形和动画、碰撞检测、声音和图像等方方面面的知识。本书可以帮助读者在轻松有趣的过程中，掌握Python游戏编程的基本技能。

本书包含的示例游戏程序包括猜数字游戏、Jokes、Dragon Realm、Hangman、Tic Tac Toe、Bagels、Sonar Treasure Hunt、Reversi、Dodger等。

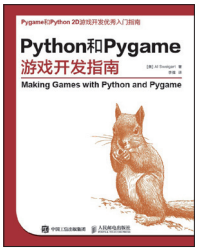
本书的Web站点 http://inventwithpython.com，提供源代码下载等更多资源。

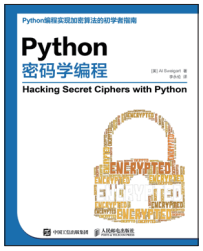
本书适合不同年龄和层次的Python编程初学者阅读。

**作者简介**

**Al Sweigart**是加利福尼亚州旧金山的一名软件开发者。他很喜欢骑自行车、当志愿者、泡咖啡吧以及开发有用的软件。他编写了《Python游戏编程快速上手》《Python和Pygame游戏开发指南》《Python密码学编程》《Python编程快速上手——让繁琐工作自动化》等图书，深受读者欢迎。他生于德克萨斯的休斯顿。他在德克萨斯大学Austin分校读完了计算机科学学位。

**Al Sweigart系列作品将陆续由人民邮电出版社出版，敬请关注！**



ISBN 978-7-115-42903-2


异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

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