



华章科技

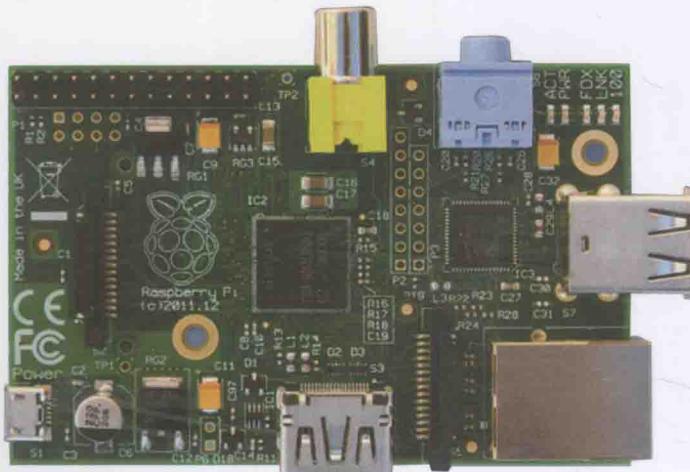
树莓派基金会资深软件开发工程师亲笔撰写，系统阐释在树莓派上使用Python开发游戏、多媒体等的实用工具、方法和最佳实践

深入剖析Python常见开发问题，包含大量实践案例，可操作性强，能为用户使用树莓派高效编写Python程序提供有效指导



数字匠人

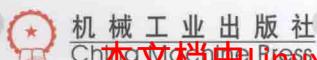
WILEY



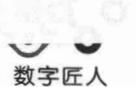
Learning Python
with Raspberry Pi

树莓派 Python编程指南

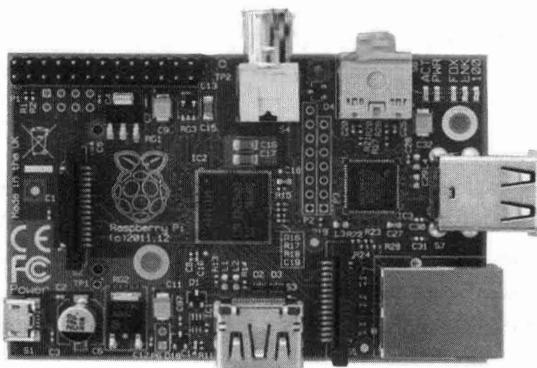
[美] Alex Bradbury Ben Everard 著 王文峰 译



本文档由Linux公社 www.linuxidc.com 整理



数字匠人



Learning Python
with Raspberry Pi

树莓派 Python编程指南

[美] Alex Bradbury Ben Everard 著 王文峰 译



机械工业出版社
China Machine Press

本文档由Linux公社 www.linuxidc.com 整理

图书在版编目 (CIP) 数据

树莓派 Python 编程指南 / (美) 布拉德伯里 (Bradbury, A.) 等著; 王文峰译. —北京: 机械工业出版社, 2015.1
(数字匠人)

书名原文: Learning Python with Raspberry Pi

ISBN 978-7-111-48986-3

I. 树… II. ①布… ②王… III. 软件工具 – 程序设计 – 指南 IV. TP311.56-62

中国版本图书馆 CIP 数据核字 (2014) 第 303346 号

本书版权登记号: 图字: 01-2014-3575

Copyright © 2014 Alex Bradbury and Ben Everard

All Rights Reserved. This translation published under license. Authorized translation from the English language edition, entitled Learning Python with Raspberry Pi, ISBN 978-1-118-71705-9, by Alex Bradbury and Ben Everard, Published by John Wiley & Sons. No part of this book may be reproduced in any form without the written permission of the original copyrights holder.

本书中文简体字版由约翰 - 威利父子公司授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封底贴有 Wiley 防伪标签，无标签者不得销售。

树莓派 Python 编程指南

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 秦 健

责任校对: 殷 虹

印 刷: 北京市荣盛彩色印刷有限公司

版 次: 2015 年 1 月第 1 版第 1 次印刷

开 本: 186mm × 240mm 1/16

印 张: 14

书 号: ISBN 978-7-111-48986-3

定 价: 59.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有 • 侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

Preface 前 言

计算机已经不再只是用来工作的方盒子。任何拥有一个可编程处理中心的东西都可以称作计算机。游戏终端、智能手机、全球卫星定位系统（GPS）单元、平板电脑以及其他许多令人难以置信的设备都说明了这一点。它们都是计算机，并且它们已经接管了这个世界。我们用它们来工作、通信，以及娱乐。事实上，很难找到有哪个领域还从未使用过计算机。

销售人员喜欢用智能来描述嵌入式计算机设备（智能手机、智能电视、智能手表等），事实上它们却并不智能。处理单元只是块执行指令的硅片。智能手机的“智能”并非来自计算机芯片，而是来自对它们编程的人。

计算机是迄今为止人类发明的最强大的工具，但是由于很少有人知道如何开发它的潜能，目前我们只是使用了它的一小部分功能。在充满计算机的世界中，最重要的就是那些可以发挥计算机全部性能的编程人员。编程，是一项基本技能，并且在未来会变得更重要。

什么是编程

我们已经指出，计算机并不智能。它只是个按照清单一步步执行指令的单元。这个指令清单就是程序。编程，就是接受任务，将其分解成多个步骤，然后把它们用计算机可以理解的语言写下来。

树莓派可以理解多种语言，在本书中，你将学习到 Python3——一种非常强大易学的语言。

本书适用于拥有树莓派并希望学些计算机编程的读者。学习本书不要求读者具有编程经验或者其他类似的技术。即便你只爱看漫画和科幻小说，都没关系，只要你具备前两个基本条件，这本书就是为你准备的。

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`



读完本书，你将会对 Python3 有很深的理解，并且熟悉许多非常有用的模块（Python 附加模块）。通过 Python3 和这些模块，你将能控制树莓派的各个功能。你可以通过控制基本输入输出口（GPIO）使它和外界交互，或者用它连通互联网。拥有一个摄像头，你可以用树莓派拍照片，也可以制作游戏，操纵三维世界。简而言之，这是一本介绍如何发掘你的树莓派的全部潜力的图书。

什么是树莓派

树莓派是个用来学习编程的优秀设备。第一，它很便宜。价格差不多只有低端计算机的十分之一，作为你的主计算机的一个附件它非常便宜。由于程序员往往需要调试开发设备，而调试就有可能破坏某些东西，这就使树莓派显得很有用。通常来说，调试并不会破坏到设备本身，但是可能需要重装系统，这就有可能损失数据并且在几个小时内你都无法使用计算机。如果你有个树莓派，用它来编程，这些就不是问题。如果你的计算机需要和别人共享，使用树莓派就显得更重要。

第二，树莓派是个裸设备。它没有藏在盒子里或者在一个完整的系统中。这意味着你可以自己决定将其做成一个什么系统。你可以将其放到盒子里，也可以就让它裸着运行。你也可以使用 GPIO，这在其他机器上是无法做到的。大多数计算机的用途都已经预先设计好了（如用来网上冲浪或者玩游戏的平板电脑，用来看电影或玩游戏的游戏终端，用来工作或玩游戏的笔记本电脑等）。而只要一点技术手段树莓派就可以做任何事情。

第三，树莓派使用 Linux——一个类似于 Windows 或 Mac OS X 的操作系统。它提供了视窗系统和一个用来操控树莓派的基于文本的命令行接口。如果你之前没有使用过 Linux，会发现它和你使用过的系统有点不同。对于崭露头角的程序员来说，最重要的区别就是 Linux 比其他系统更灵活。正如树莓派的物理设计崇尚体验一样，这个操作系统也是如此。

如何阅读本书

前 3 章介绍了如何在树莓派上使用 Python。读完这 3 章后，你将对 Python 编程有个很好的认识。本书剩余部分将分章来介绍不同的应用，如游戏和多媒体。这些章节涉及 Python 的不同领域。因此前一章没有读完不会影响你对后一章的理解（有些时候，我们可能会引用前面的某些概念，但我们在引用时标注清楚）。

这意味着你可以在读本书第二部分时自己决定阅读顺序。例如，如果你对多媒体很感兴趣，可以直接跳到这一章，之后再去读其他章节。

学习编程必须多动手实践。也就是说，仅仅坐下来读完本书是不够的。你必须去实践这些学到的东西。贯穿本书，我们设计了很多练习让你来实践所学到的知识。有时通过特定的练习来培养你的技能，有时你需要给我们介绍过的程序添加特性。编程的一个重要部分就是确定程序要完成什么的创造力。因此你不需要完全听从我们的建议。事实上，我们鼓励你把我们的建议和代码作为一个起点——一个开启你的数字艺术征程的起点。

致 谢 *Acknowledgements*

感谢大家帮助我完成此书。在 Wiley 公司，Kezia Ednsley 和 Craig Smith 从本书开始写作起就帮忙审稿。感谢 Erin Zeltner，是他让本书的文字看起来更优美，也让排版更合理。

还有许多人需要感谢。没有编程环境就不会有这本关于编程的书。基于树莓派的 Python 包含了数以千计的程序员的工作，其中有很多人还是免费贡献。这些人都需要感谢，受限于篇幅，我们仅列出三位：Guido van Rossum、Linux Torvalds 和 Richard Stallman。

当然，软件需要运行在硬件之上，我们也要感谢 Eben Upton 和树莓派基金会。

本书中的任何错误都由作者独自承担。

Contents 目 录

前言	2
致谢	3
第1章 起航(启动和运行)	1
1.1 组装好你的树莓派	1
1.2 可能遇到的问题	2
1.3 树莓派快速指南	3
1.3.1 使用 LXDE (轻量级 X11 桌面环境)	3
1.3.2 使用终端	4
1.3.3 通过 Raspi-Config 改变配置	6
1.3.4 安装软件	6
1.4 Python3	6
1.4.1 Python 解释器	7
1.4.2 运行 Python 程序	7
1.5 小结	8
第2章 Python 简介	9
2.1 使用 Turtles 绘画	9
2.1.1 使用循环	12
2.1.2 条件处理: if、elif 和 else	14
2.1.3 使用函数和方法组织代码	15
2.2 一个 Python 游戏: 猫和老鼠	16
2.2.1 理解变量	19
2.2.2 定义函数	19
2.2.3 在游戏中循环	19
2.3 小结	20
第3章 Python 基础	22
3.1 变量、值和类型	22
3.1.1 值和类型	23
3.1.2 数字排序	24
3.1.3 使用 Strings 保存文字	25
3.1.4 布尔值: 真或假	25
3.1.5 数据类型转换	26
3.1.6 知识测试	26
3.2 在结构体中存储值	27
3.2.1 字典和集合中的非序列 元素	30
3.2.2 知识测试	31
3.3 控制程序流程	32
3.3.1 用循环遍历数据	32
3.3.2 深入理解循环嵌套	33
3.3.3 使用 if 语句控制程序分支	34

3.3.4 捕获异常	35	第 6 章 使用 OpenGL 创建图形	92
3.4 使用函数复用代码	36	6.1 获取模块	93
3.5 组合装配	38	6.2 创建旋转立方体	93
3.6 使用类来构建对象	40	6.2.1 向量和矩阵	95
3.7 使用模块获得附加特性	45	6.2.2 组合包装	98
3.8 小结	46	6.2.3 增加光照	101
3.9 习题答案	47	6.3 让屏幕起舞	107
第 4 章 图形编程	48	6.3.1 建立 3D 模型	109
4.1 图形用户界面 (GUI) 编程	48	6.3.2 计算声音强度	110
4.2 添加控制	50	6.4 继续完善	115
4.3 创建 Web 浏览器	52	6.5 添加纹理	115
4.4 添加窗口菜单	60	6.6 小结	116
4.5 小结	62	第 7 章 Python 与网络	117
4.6 习题答案	62	7.1 理解主机、端口和套接字	117
第 5 章 搭建游戏	65	7.1.1 使用 IP 地址定位计算机	117
5.1 构建游戏	66	7.1.2 搭建会话服务器	118
5.2 初始化 PyGame	69	7.1.3 “推”向世界	121
5.3 为角色创建世界	73	7.1.4 使用 JSON 做天气预报	123
5.3.1 检测冲突	74	7.2 知识测验	125
5.3.2 左右移动	76	7.3 走向网站	125
5.3.3 达到目标	78	7.3.1 让网站动起来 (动态网站)	127
5.3.4 制造挑战	79	7.3.2 使用模板	128
5.4 在游戏中加入自己的风格	83	7.3.3 使用表格回传数据	129
5.5 添加音乐	83	7.4 安全	131
5.6 添加布景	84	7.5 小结	134
5.7 让游戏更上一层楼	87	7.6 习题答案	134
5.8 逼真的游戏物理	87	第 8 章 我的世界	137
5.9 小结	91	8.1 畅游我的世界	138

8.1.1 控制我的世界	138	10.1.3 正则表达式	170
8.1.2 用 Python 创建我的世界	139	10.2 知识测验	173
8.1.3 深入探索	142	10.3 脚本中的网络	174
8.2 制作贪吃蛇游戏	142	10.4 组合包装	175
8.2.1 移动贪吃蛇	146	10.5 在 Python 中操作文件	180
8.2.2 增长贪吃蛇	146	10.6 小结	182
8.2.3 添加苹果	146		
8.3 深入探索	147		
8.4 小结	148		
第 9 章 多媒体	149	第 11 章 硬件接口	183
9.1 使用 PyAudio 让计算机发声	149	11.1 硬件设置选择	183
9.1.1 录音	151	11.1.1 母转公接头	183
9.1.2 向树莓派讲话	151	11.1.2 无焊面包板	184
9.1.3 向程序提问	152	11.1.3 成品板和万能板	185
9.1.4 组合包装	153	11.1.4 PCB 加工	185
9.1.5 深入探索	155	11.2 辅助工具	185
9.2 制作电影	155	11.2.1 剪线 / 剥线器	185
9.2.1 使用 USB 网络摄像头	155	11.2.2 万用表	185
9.2.2 使用 OpenCV 添加计算机 图像特性	158	11.2.3 电烙铁	185
9.2.3 深入探索	160	11.3 本章所需的硬件	186
9.2.4 使用树莓派摄像头模块	160	11.3.1 第一个电路	186
9.2.5 创建直播视频	162	11.3.2 保护树莓派	189
9.2.6 深入探索	165	11.3.3 电源限制	190
9.3 小结	165	11.3.4 获得输入	191
第 10 章 脚本	166	11.4 使用 I2C、SPI 和串口扩展 GPIO	192
10.1 从 Linux 命令行开始	166	11.4.1 SPI 通信协议	193
10.1.1 使用 subprocess 模块	168	11.4.2 I2C 通信协议	196
10.1.2 命令行标签	169	11.4.3 串口通信协议	196
		11.5 深入研究	196
		11.5.1 Arduino	197
		11.5.2 PiFace	197
		11.5.3 Gertboard	197

11.5.4 Wireless Inventor's Kit	198	12.2 通过测试发现故障	203
11.6 尝试一些流行工程	198	12.2.1 使用单元测试检查代码	
11.6.1 机器人	198	片段	204
11.6.2 家庭自动化	198	12.2.2 获得更多断言	207
11.6.3 防盗报警器	199	12.2.3 使用测试集进行回归	
11.6.4 数字艺术	199	测试	209
11.7 小结	199	12.2.4 测试整个程序包	210
第 12 章 测试与调试	200	12.2.5 保证软件可用性	210
12.1 通过打印变量调查故障	200	12.3 究竟需要多少测试	211
		12.4 小结	211

起航（启动和运行）

欢迎阅读本书。本书从三维图像、游戏编程到控制电子学，再到推文，会让你学到如何解放这个小计算机的全部能量。你将会看到掩藏在表象之下的内部世界，并学会如何创建程序以发挥这台小计算机的全部特性。

1.1 组装好你的树莓派

为跟进本书，你需要一些设备：

- 树莓派

- USB 键盘

- USB 鼠标

- SD 卡

- 显示器

- 电源

还有一些有用的可选设备：

- 有源 USB 集线器（强烈推荐）

- 摄像头模块

- USB 网络摄像头

- USB 无线网络（WiFi）适配器

本书中的所有内容都可以在 A 版树莓派上完成。从编程角度而言，B 版树莓派的强大之处在于多了个网络接口。这个接口在你需要安装软件时可以方便地接入互联网。

树莓派兼容任何 USB 键盘、鼠标和大多数的 SD 卡。仅有少数 SD 卡可能会存在问题。如果不确定，可以通过树莓派网上商城购买 (<http://raspberrypi.org> 上可以找到商店链接)。

树莓派拥有一个 HDMI (高清多媒体) 视频输出端口，但是大多数显示器只有 VGA 或者 DVI 输入。如果可能，请选用具有 DVI 或者 HDMI 输入的显示器。一个 HDMI 转 DVI 接头只要几十块钱，并且其不会降低图像质量。市场上也有 HDMI 转 VGA 转接器，但是其价格昂贵且性能不稳定。所以只有在没有其他选择时才会使用这种转接头方案。

品牌厂商出品的 USB 电源都可以工作，而一些杂牌廉价 USB 电源可能会出现问题。如果可能，建议不要在电源方面太过节俭。当然，你也可以通过 USB 电缆连接普通计算机给树莓派供电。

有源 USB 集线器可以减少本章后边提到的电源相关的问题。并不是所有的 USB 集线器都是有源的，所以要保证你的 USB 集线器可以通过市电供电。

我们将在第 9 章中谈论如何选择摄像头。这里唯一要指出的是，如果你选择 USB 网络摄像头，请保证它兼容树莓派。部分支持树莓派的网络摄像头可以参考 http://elinux.org/RPi_USB_Webcams。

通过将树莓派用网线连接到路由器或者使用 USB 无线适配器连接到无线网络，你可以将树莓派接入互联网，并安装本书中需要用到的软件。

1.2 可能遇到的问题

树莓派第一个最常见的问题是电源相关的问题。有的 USB 电源无法提供足够的电流，尤其当树莓派连上外围设备或者你超频它时（详见第 5 章），使用这些电源可能会导致更多问题。电源相关的问题通常会导致计算机崩溃。因此，当发现树莓派运行不稳定时，首先要检查电源。为了避免这类问题，可以给树莓派单独使用一个电源，外围设备（如键盘、鼠标等）通过有源 USB 集线器连接到树莓派。

第二个最常见的问题是 SD 卡问题。这类问题可能是电源供电不足或者 SD 卡本身的问题。因此，我们有必要做好备份以保证数据安全。我们可以使用谷歌云硬盘服务来备份数据（在树莓派上运行有点慢），或者简单地在优盘中保存个副本。SD 卡相关问题通常表现在树莓派开机时显示出错信息。多数情况下，重装树莓派可以解决问题。如果重装无法解决，就需要更换新的 SD 卡。

如果遇到了其他问题，我们就需要深入挖掘了，通常，我们需要查看内核缓冲区和系统日志文件。如果遇到的是硬件相关问题，如 USB 设备无法工作，最好去看看内核缓冲区。打开终端模拟器 LXTerminal 并输入：

```
dmesg
```

这条命令将输出 Linux 内核的所有信息。它按照时间先后顺序显示，最后显示的是最近发生的事件。有任何问题都可以显示出来。

下面的命令可以打印出系统日志（通常称为 syslog）：

```
cat /var/log/syslog
```

同样，最新的消息仍然放在最后面。这两个信息可能有时会令人难以捉摸。如果读了之后仍然无法解决问题，最好去树莓派论坛上看看 www.raspberrypi.org/phpBB3/。这个论坛上有很多热心人，他们会帮助我们找到正确的方向。

树莓派上安装 Raspbian 的最简便方法是使用 NOOBS，可以从 www.raspberrypi.org/downloads 下载。网站上还有一份快速入门指南，它为树莓派的启动和运行提供了完整的帮助。

1.3 树莓派快速指南

本书主要面向编程方向，而不是介绍如何使用树莓派。因此我们不会过多地介绍树莓派。但你会从中了解到树莓派是如何工作的。

有不少操作系统可以运行在树莓派上，本书的命令都是基于 Raspbian 的（一个树莓派默认的操作系统），也是初学者最好的选择[⊖]。如果有 Linux 的使用经验，可以选择 Arch 或者 Fedora。选择了其他系统，就需要将本书的 apt-get 命令转换为相应系统的命令以完成软件包管理。

在树莓派上安装 Raspbian 的最简便方法是使用 NOOBS，可以从 www.raspberrypi.org/downloads 下载。网站上还有一份快速入门指南，它为树莓派的启动和运行提供了完整的帮助。Raspbian 提供两种不同的交互方式——终端命令和图形系统（LXDE）。

1.3.1 使用 LXDE（轻量级 X11 桌面环境）

轻量级 X11 桌面环境是 Raspbian 的标准窗口系统。其基本组件和大多数的 Windows 8 之前的 Windows 窗口相似。屏幕左下角有一个按钮，可以打开应用程序菜单。当前运行的程序显示在该按钮右侧的长条上（参见图 1-1）。

[⊖] 基于 Debian，由 Mike Thompson 和 Peter Green 针对树莓派硬件对 Debian 进行了专门优化和移植。
——译者注

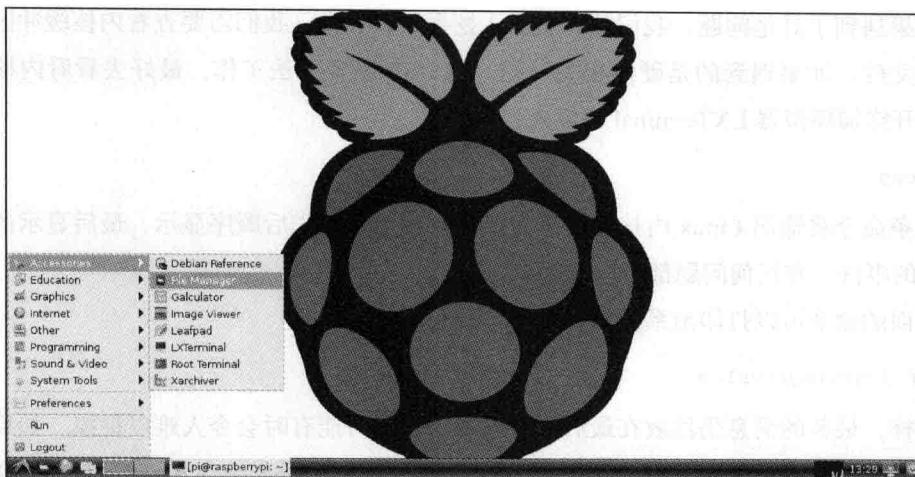


图 1-1 程序菜单打开的 LXDE 桌面

启动树莓派之后，如果你看到的是黑屏白字，询问是否登录，这表示还没有设置自动启动 LXDE。不用担心，只需使用用户名 pi，密码 raspberry 登录后输入如下命令就可以启动 LXDE。

```
startx
```

通过配置 raspi-config 可以设置成启动时自动运行 LXDE（参见下一节）。

1.3.2 使用终端

对于多数应用 LXDE 都很方便，但有时候我们也需要使用命令行。它是一个通过终端操作的非常强大的接口。在 LXDE 环境下，需要打开 LXTerminal 应用来使用它。

打开 LXTerminal，将会看到下面这行字：

```
pi@raspberrypi ~ $
```

它表示目前正使用用户名 pi 登录到名叫 raspberrypi 的计算机上，当前目录是 ~。

Linux 的所有目录都起始于 / 或者 root，它是目录树的基础，每个目录都位于根目录 (root) 的某个子目录下。cd (更改目录) 命令可以在不同的目录间切换。下面的命令展示了如何切换到根目录：

```
cd /
```

执行这条命令后，命令提示符将变为：

```
pi@raspberrypi ~ $
```

ls命令可以列出这个目录下的内容。有一个称为home的子目录，系统中的每个用户都有自己的home目录。执行下面的命令可以让我们进入home目录并查看其中的内容：

```
cd home
```

```
ls
```

此时，home目录中只有一个目录：pi。并且命令提示符也提示我们当前正处于/home目录。

移动到我们仅有的一个子目录中：

```
cd pi
```

现在，命令提示符重新变回：

```
pi@raspberrypi~$
```

这是因为字符～是当前用户home目录的简写。在终端中输入～时，系统会将其转换为/home/pi。

关于命令行还有许多值得学习的地方。要完整地介绍命令行的各个方面，需要再写一本和本书一样厚的书。然而，开始使用命令行时并不需要完全了解它。本书中在使用LXTerminal时，都会完整地列出要使用的命令行。

 提示 如果你希望学习到更多树莓派相关的知识或Linux通用知识，命令行是一个非常棒的开端。不论是在线资料还是印刷图书，都可以找到大量命令行相关的信息。Linux命令行这本书可以免费在线浏览，这是一本非常棒的启蒙读物。网址<http://linuxcommand.org/tcl.php>。

这里我们有两个建议。首先，不要害怕命令行。开始使用时可能会有点头疼，但学习如何使用它的唯一方法就是多使用它。其次，几乎所有的命令都有内置帮助，通过选项`--help`可以查看这些帮助。例如，如果需要查看ls命令的更多用法，可以输入：

```
ls --help
```

该命令会输出：

```
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by
default). Sort entries alphabetically if none of -cftuvSUX nor
--sort is specified.
```

后面还会列出该命令可以使用的各种选项。

1.3.3 通过 Raspi-Config 改变配置

Raspbian 提供了一个帮助设置树莓派硬件的工具，称为 raspi-config。打开 LXTerminal 并输入如下命令就可以使用了：

```
sudo raspi-config
```

在这里，你会发现启动时自动启动 LXDE、超频树莓派等各种选项。超频树莓派会使本书中的一些例子运行得更好，尤其是在安装新软件时。

1.3.4 安装软件

在终端中使用 apt-get 可以给树莓派安装新软件。安装之前，最好把当前的所有软件升级到最新版本。下面是升级命令：

```
sudo apt-get update  
sudo apt-get upgrade
```

然后就可以使用 apt-get 安装任何你需要的软件了。例如，如果想使用 iceweasel（火狐浏览器的 Debian 再发布版），可以打开 LXTerminal 并输入：

```
sudo apt-get install iceweasel
```

如果更喜欢使用图形界面来安装程序，可以安装 synaptic[⊖]：

```
sudo apt-get install synaptic
```

只需开启它就可以安装程序：

```
sudo synaptic
```

然后就可以直接搜索需要安装的东西。



注意 安装软件需要在命令行前加上 sudo。这是为了告诉计算机，我们需要做些系统级别的变化，请给予程序足够的权限以完成这些操作。

1.4 Python3

本书中，你将会学到如何使用 Python3 这门编程语言。在 Raspbian 中使用这门语言有

[⊖] Debian 及其衍生版本的包管理工具 apt 的图形化前端。——译者注

多种方式。

1.4.1 Python 解释器

有两种方式可以使用 Python，分别是 shell 交互和文本程序。shell 交互可以执行用户输入的每条指令，对于调试和实验非常有利。文本程序就是保存在文本文件中的 Python 代码，它可以一次性全部运行。很容易区分这两种运行方式。如果处于 shell 交互模式，每行都会以三个大于号开始：

```
>>>
```

本书大多数时候都使用程序文件。偶尔使用 shell（尤其是在早期时候）时我们会明确指出。为了明确区分代码运行环境，我们对运行在 shell 下的代码前都加上三个大于号。

1.4.2 运行 Python 程序

写 Python 程序有两种不同的方法。第一种方法是创建一个包含 Python 代码的文本文档，然后运行它。第二种方法是使用集成开发环境（IDE），如 IDLE3。以上方法的运行方式一样，结果也一样。可以根据个人爱好自由选择。

如果希望把程序写入文本文档，就需要一个文本编辑器如 Leafpad。文字处理软件如 LibreOffice 的 Writer 是不能用来写 Python 程序的，因为 Python 无法识别其使用的一些格式。下面给出一个例子，打开 Leafpad 并创建一个新文件，然后输入下面的文字：

```
print("Hello World!")
```

创建和保存文件时，请使用 .py 扩展名，如 testfile.py。要运行它，可以打开 LXTerminal 并移动到文件保存的子目录，执行 python <文件名>。使用 cd 命令可以移动到不同的目录。比如文件保存在当前 home 目录下的 programming 文件夹下，在 LXTerminal 中只执行下面命令就可以运行 python 程序了：

```
cd programming
python3 testfile.py
```

如果一切顺利，你将会在屏幕上看到下面一行字：

```
Hello World!
```

第二种方法更简单，IDE 中集成了文字编辑器和 Python 解析器。例如，打开 IDLE3（注意要带数字 3），然后点击 File → New Window。在新窗口中输入以下代码：

```
print("Hello IDLE")
```

继续点击 Run → Run Module。IDLE3 会提示是否保存该模块，选择文件名保存后，它将会返回到 Python 解释器并输出下列内容：

```
Hello IDLE
```

无论选择哪种方式来学习都可以，接下来，不妨选一个自己喜欢的方式开始吧。

1.5 小结

读完本章，你应该理解以下内容：

- 你需要一些额外的硬件以更好地利用树莓派。
- 供电不足是最常见的问题。
- 遇到问题时，最好从 dmesg 和 syslog 开始查找原因。
- Raspbian 使用 LXDE 桌面环境。
- 终端是和底层操作系统最有效的沟通方式。
- raspi-config 可以用来配置树莓派。
- apt-get 可以用来安装新软件。
- Python 可以通过 shell 交互或者文本程序运行。

Python 简介

本章通过一些代码示例来逐步介绍 Python。你不必了解这些代码的所有细节，本章只是带你领略下编程的感觉。你将会学到如何在屏幕上绘画，甚至如何编写简单的游戏。同时，你将会学到一些编程的基本概念。如果无法完全理解本章中的程序，不用担心，我们会在后面章节中做详细介绍。

2.1 使用 Turtles 绘画

是时候开始编程了！我们强烈建议你将代码一行行输入 IDLE3，这样可以帮助你理解每行都做了什么事情。言归正传，打开 IDLE3，点击 File → New Window，然后输入：

```
import turtle  
window = turtle.Screen()  
babbage = turtle.Turtle()  
babbage.left(90)  
babbage.forward(100)  
babbage.right(90)  
babbage.circle(10)  
window.exitonclick()
```

然后点击 Run → Run Module 或者按 F5 键来运行程序。此时会跳出一个对话框让你输入文件名。起什么名字都可以，但起一个容易理解的名字可以帮助我们记住它（这里我们使用 chapter2-example1.py）。

文件中每一行都是一个 Python 指令。Python 一行行检查它们，然后按照先后顺序来执行。图 2-1 显示了计算机执行完所有步骤后的结果：在屏幕上画出一条直线，直线上面连接一个圈。看起来像棒棒糖，实际上，这是一朵花的第一部分。如果你的屏幕没有显示出这个结果，请检查下文件的每一行是否正确，然后再试一次。

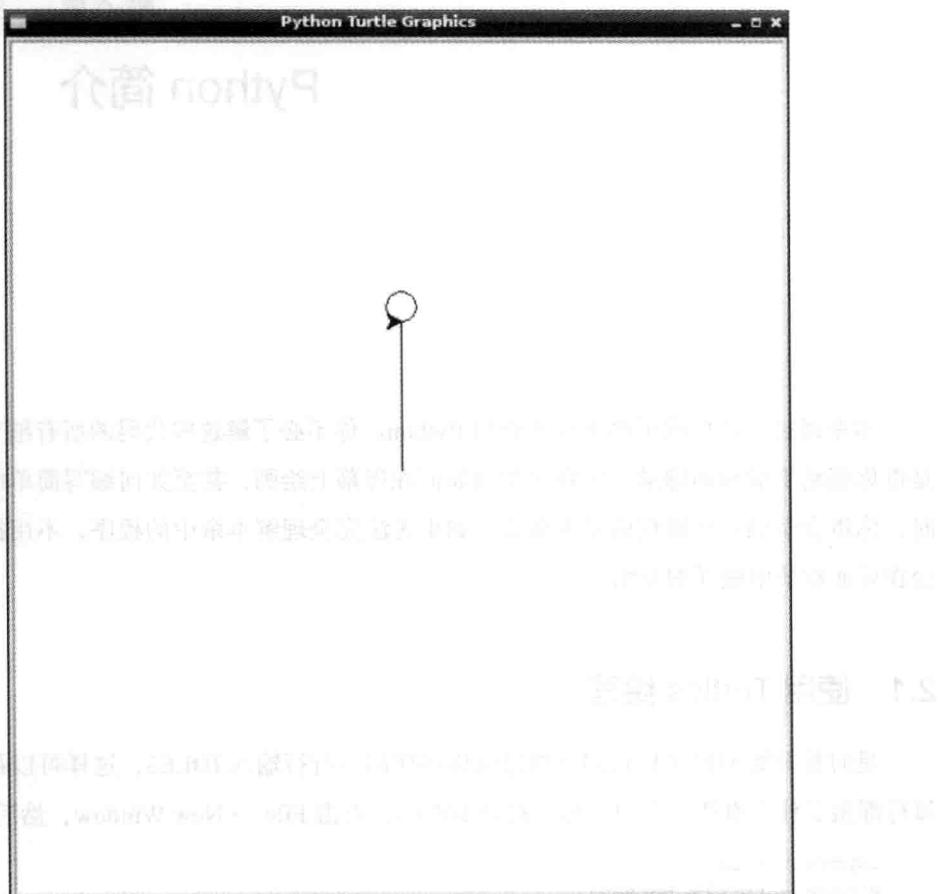


图 2-1 使用 Python turtle 绘出的第一张图

接下来我们仔细看下 Python 是如何根据代码来工作的。

```
import turtle
```

在 Python 程序的开头部分，通常会有一些 import 行。它们为程序导入一些附加特性，就像其他软件中的附件或者插件一样。这些特性被分成不同的模块。后面章节中将介绍更多的 import 命令。本例中我们仅导入 turtle 模块以完成绘图工作。

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`



接下来的代码：

```
window = turtle.Screen()
...
window.exitonclick()
```

创建了一个新窗口用于绘图，并定义了一个动作，单击窗口就可以将其关闭。

下面这行代码使用了我们在第一行中导入的 turtle 模块创建的一个名字叫 babbage 的 turtle 对象（Charles Babbage，计算机先驱，创造了计算机的概念）。

```
babbage = turtle.Turtle()
```

Babbage 有很多个方法可供我们使用。例如下面这行：

```
babbage.left(90)
```

使用了 left() 方法，使 babbage 向左转一定角度。位于 left() 方法后面括号里的参数可以控制该方法运行时的角度。本例中，输入参数为 90，因此 babbage 向左转 90° 。下一行使用了 forward()、right() 和 circle() 方法。

```
babbage.forward(100)
babbage.right(90)
babbage.circle(10)
```

第一个方法将 turtle 向前移动 100 个像素，第二个将其右转 90° ，最后一个画了一个半径为 10 像素的圆。

现在开始添加花瓣。按下面例子编辑代码（变化部分已经用黑体标出）：

```
import turtle
#create window and turtle
window = turtle.Screen()
babbage = turtle.Turtle()
#draw stem and centre
babbage.left(90)
babbage.forward(100)
babbage.right(90)
babbage.circle(10)
#draw first petal
babbage.left(15)
babbage.forward(50)
babbage.left(157)
babbage.forward(50)
#tidy up window
window.exitonclick()
```

运行后会看见这朵花已经有了第一片花瓣。注意我们添加了一些带 # 符号的行，计算机将忽略以 # 开始的行。因此我们可以用这种方法添加注释（或者其他任何阅读代码的

人)。它增加了程序的可读性。当我们几天、几周甚至几年后再来看这段代码，有了注释就可以很容易地理解它做了什么。

2.1.1 使用循环

画花瓣的部分很容易理解(我们使用三角函数来计算出两个左转之间的夹角，但不用担心，我们不会陷入数学计算中)。

现在可以添加一段新代码来画第二片花瓣(总共要画 24 片花瓣)。和画第一片花瓣完全相同，因此只要复制粘贴就可以得到下列代码：

```
#draw second petal
babbage.left(15)
babbage.forward(50)
babbage.left(157)
babbage.forward(50)
```

然后，重复 22 次画出剩余花瓣。好吧，够了，根据编程的一般经验法则，不能重复出现相同的代码。假设你要改变花瓣尺寸，而且需要改 48 个地方(每片花瓣两次)，如果忘记任何一个地方，将会得到一个不靠谱的图片。因此，你可以使用循环，用一小段代码告诉计算机反复执行特定部分的代码。

你可以用下面代码来替代从 # 画第一片花瓣 (#draw first petal) 开始的所有代码：

```
#draw all petals
for i in range(1,24):
    babbage.left(15)
    babbage.forward(50)
    babbage.left(157)
    babbage.forward(50)
window.exitonclick()
```

我们将会在第 3 章中介绍注释后面的第一行代码。现在，只需要理解它表示重复运行后面那段代码 24 次之后，计算机将执行这块代码后的下一个命令。

Python 中循环(或者之后接触的其他类型条件)的代码段通常使用相同的格式。第一行以冒号结尾，之后的每一行都使用相同的缩进。当 tab 或者缩进结束时，Python 就认为该代码段结束了。如果你之前使用过其他编程语言，将会发现 Python 的做法和其他语言有所不同。

运行代码后，你会发现 babbage 转了一圈将所有花瓣都画完了。我们得到了图 2-2 所示的完整花朵。

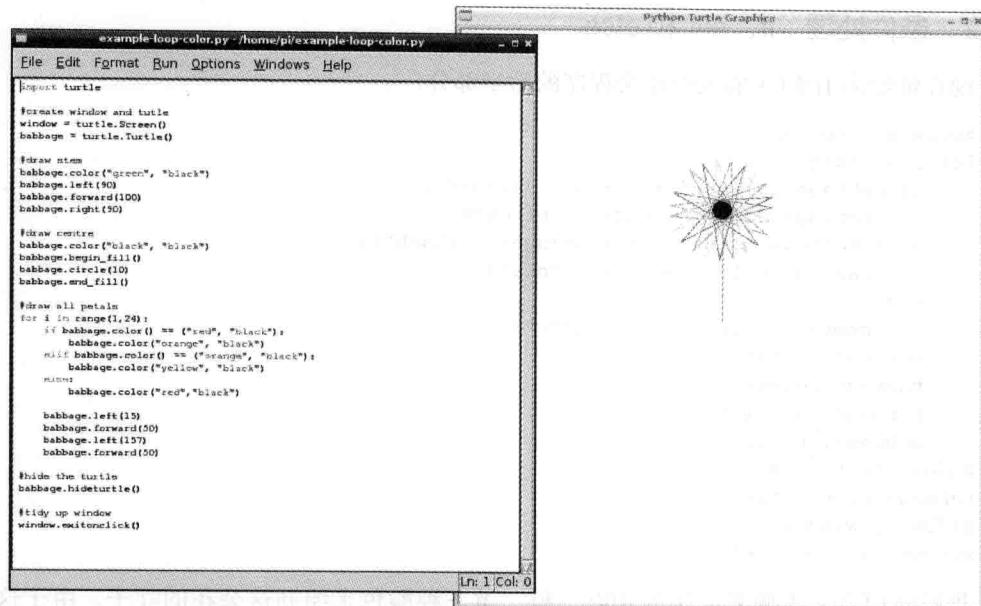


图 2-2 使用循环轻易地画出一朵花

仅用 13 行代码，干得不错。当然，不是所有的花都是黑色的，在图片中添加点颜色会更好看点。turtle 模块提供了一些方法让我们可以选择画线的颜色。下面对前面的代码做了些修改（更改部分以黑体显示）。

```

import turtle
#Create window and turtle
window = turtle.Screen()
babbage = turtle.Turtle()
#draw stem
babbage.color("green", "black")
babbage.left(90)
babbage.forward(100)
babbage.right(90)
#draw centre
babbage.color("black", "black")
babbage.begin_fill()
babbage.circle(10)
babbage.end_fill()

```

上述代码中，我们使用了 `color(colour1, colour2)` 方法（英国人应该看得出来该方法的名字是美式写法）。这里 `colour1` 是画笔色，`colour2` 是填充色。画完花朵中心后，我们告诉计算机用 `begin_fill()` 方法填充中心。然后，我们调用 `end_fill()` 方法，以防它填充所有的花瓣。

2.1.2 条件处理：if、elif 和 else

现在继续向 IDLE3 输入画花朵程序的后半部分：

```
#draw all petals
for i in range(1,24):
    if babbage.color() == ("red", "black"):
        babbage.color("orange", "black")
    elif babbage.color() == ("orange", "black"):
        babbage.color("yellow", "black")
    else:
        babbage.color("red", "black"))
    babbage.left(15)
    babbage.forward(50)
    babbage.left(157)
    babbage.forward(50)
#hide the turtle
babbage.hideturtle()
#tidy up window
window.exitonclick()
```

我们使用点艺术眼光，决定用红、橙、黄三种颜色来绘制这朵花的叶子。由于这是本黑白印刷的书，只有在树莓派上运行程序之后，才能看到彩色图像。当然也可以通过本书网站上的 flower.png 来查看结果。为了转换花瓣色彩，我们使用了 if ... elif ... else 语句。它告诉 Python 根据不同的数据做不同的事情。基本结构如下所示：

```
if <condition> :
    code
```

这里的 <condition> 表示条件，可以为真或假。本例中，我们使用如下条件：

```
babbage.color() == ("red", "black")
```

babbage.color()（注意该方法不带任何参数）告诉应用程序我们当前使用的颜色。不同于我们之前遇到的方法，它反过来给我们输出了一些信息。它的返回值是一对颜色——第一个是画笔色，第二个是填充色（从画完花朵中心之后，我们就没有改变过这两个颜色，因此程序的后继部分仍然使用它们）。双等于号（==）表示“相等”。使用双等于号是因为等号已经被定义为“赋值”，我们在创建 window 和 turtle 对象时已经使用过等号。

如果条件为真（本例中，如果 turtle 的颜色为（“red”，“black”）），Python 将执行 if 之后的代码。如果条件为假，Python 将转而执行 elif（elif 是 else if 的简写）。这里的判断条件和 if 一样。

如果 elif 处的条件为假，Python 将转到 else 处执行。到此为止（也就是说，if 和 elif

处的判断条件都为假), Python 将执行 else 之后的代码。else 处没有判断条件。图 2-3 描述了上述逻辑流程。

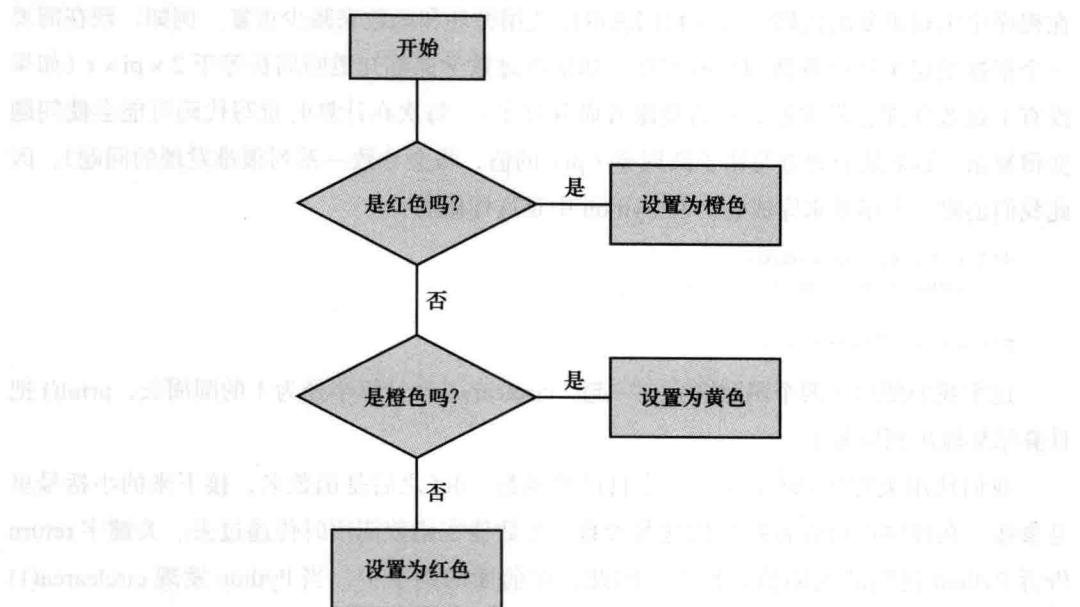


图 2-3 判断花瓣颜色的流程图

这些 if 语句能够在画完一个花瓣后改变画笔颜色。我们在程序后面新加了下面这行：

```
babbage.hideturtle()
```

它会隐藏 turtle (鼠标), 以保证鼠标不会遮挡我们的图片。至此, 我们的第一个 Python 程序圆满完成。

2.1.3 使用函数和方法组织代码

在开始第二个 Python 程序之前, 我们先停下来讨论下“方法”。从前面的代码可以看出, 方法能够有效地控制程序。在前面的例子中, 我们使用它们来移动 turtle, 改变颜色或者创建窗体。每次都调用这些方法来完成某些事情。例如, 通过 babbage.forward(50) 来调用 babbage 的 forward(50) 方法, 通过 window.onclick() 来调用 window 的 onclick() 方法。每次调用这些方法, 都会运行保存在 Python 模块中的相应代码。Python 还有个相似的特性——函数。函数和方法的工作方式有点类似, 但是函数不需要 import 任何模块。例如, 在 Python 解释器中输入:

```
>>> print("Hello World")
```

这行代码将运行 `print` 函数，将参数输出到屏幕上。是否还记得我们前面提到的不要在程序中出现重复的代码？当时我们说可以使用循环和函数来减少重复。例如，现在需要一个根据给定半径计算圆周长的程序。如果听过数学课都知道圆周长等于 $2 \times \pi \times r$ （如果没有上过数学课也没关系，只需要跟着做就好了）。每次在计算时重写代码可能会使问题变得复杂（如果某个地方写错了圆周率（ π ）的值，将会导致一系列很难发现的问题）。因此我们创建一个函数来完成它。在 Python 中是这样做的：

```
def circleara(radius):
    return radius * 3.14 * 2

print(circleara(1))
```

这里我们使用了两个函数嵌套在一起。`circleara(1)` 计算半径为 1 的圆周长，`print()` 把计算结果输出到屏幕上。

我们使用关键字 `def` 定义了一个自己的函数。`def` 之后是函数名，接下来的小括号里是参数。在程序中可以直接使用这些参数，参数值在函数调用时传递过去。关键字 `return` 告诉 Python 我们的返回值是什么。因此，在前面的例子中，当 Python 发现 `circleara(1)` 时，将会把 1 作为参数 `radius` 的值传入函数，然后执行 `def circleara(radius)`。接着，它返回计算结果 (6.28) 给函数 `print`。以后你将会看到，方法也可以像函数一样，一个方法通过嵌套将信息传递给另外一个方法。这是一种非常有用的方法，可以在程序的不同部分间传递数据流。

2.2 一个 Python 游戏：猫和老鼠

现在，让我们开始第二个 Python 程序。这次你将写出一个猫和老鼠的游戏。游戏者使用方向键来控制老鼠，使其保持在猫的前方（由计算机控制猫）。保持时间越长，得分越高。



注意 本书中的大段程序大多数都可以从网站 www.wiley.com/go/python-raspberrypi.com 下载。为避免潜在的输入错误，你可以下载这些程序，复制粘贴到 IDE 或者代码编辑器中。下面例子中的程序名字是 `Chapter2-catandmouse.py`。

打开一个 IDLE3 新窗口并输入下面的代码：



```
import turtle
import time

boxsize = 200
caught = False
score = 0

#functions that are called on keypresses
def up():
    mouse.forward(10)
    checkbound()

def left():
    mouse.left(45)

def right():
    mouse.right(45)

def back():
    mouse.backward(10)
    checkbound()

def quitTurtles():
    window.bye()

#stop the mouse from leaving the square set by box size
def checkbound():
    global boxsize
    if mouse.xcor() > boxsize:
        mouse.goto(boxsize, mouse.ycor())
    if mouse.xcor() < -boxsize:
        mouse.goto(-boxsize, mouse.ycor())
    if mouse.ycor() > boxsize:
        mouse.goto(mouse.xcor(), boxsize)
    if mouse.ycor() < -boxsize:
        mouse.goto(mouse.xcor(), -boxsize)

#set up screen
window = turtle.Screen()
mouse = turtle.Turtle()
cat = turtle.Turtle()
mouse.penup()
mouse.penup()
mouse.goto(100,100)

#add key listeners
window.onkeypress(up, "Up")
window.onkeypress(left, "Left")
window.onkeypress(right, "Right")
window.onkeypress(back, "Down")
window.onkeypress(quitTurtles, "Escape")
```

```

difficulty = window.numinput("Difficulty",
    "Enter a difficulty from easy (1), for hard (5)",
    minval=1, maxval=5)

window.listen()
#main loop
#note how it changes with difficulty
while not caught:
    cat.setheading(cat.towards(mouse))
    cat.forward(8+difficulty)
    score = score + 1
    if cat.distance(mouse) < 5:
        caught = True
    time.sleep(0.2-(0.01*difficulty))
window.textinput("GAME OVER", "Well done. You scored:\n" +
    "+ str(score*difficulty))")
window.bye()

```

代码很多，在仔细读之前可以先试玩几次感受一下。这样也可以检验你的输入是否完全正确。如果程序出错，先检查下输入，然后再试一下。图 2-4 给出了程序运行界面。

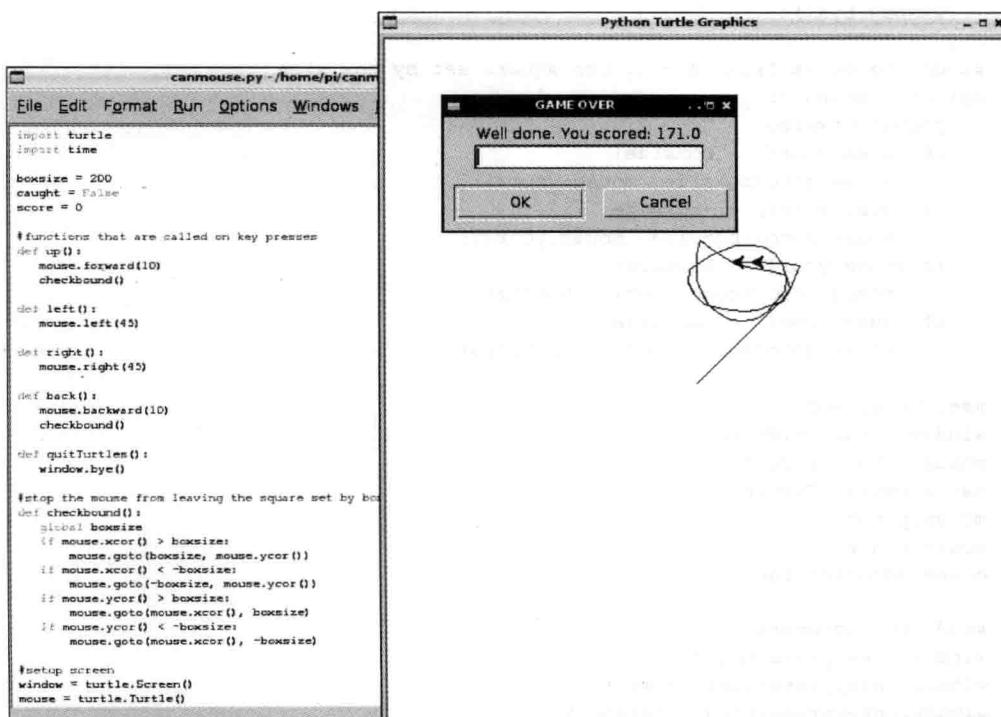


图 2-4 一个简单的猫和老鼠游戏

2.2.1 理解变量

程序前两行导入 turtle 和 time 模块，接下来的三行：

```
boxsize = 200
caught = False
score = 0
```

使用了我们之前接触过但没有深入介绍的概念：变量（variable）。我们可以在变量中存入值以便后面使用。例如，第一行中在变量 boxsize 中存入 200。写完这行，你就可以在程序中使用 boxsize，Python 将使用正确的值来替代它。这个结构之所以称为变量是因为它可以变化。在我们的这个程序中，boxsize 保持不变，但 caught 和 score 的值会发生变化。每次需要一个新值时，可以使用单个等号对其重新赋值。这和我们在第一个程序中对 window 和 babbage 赋值的做法一样，它们分别保存了 screen 和 turtle。在第 3 章中我们将介绍变量，以及可以在变量中存储什么值。

2.2.2 定义函数

程序中接下来的代码定义了一些函数。注意在函数 checkbounds() 中的这一行：

```
global boxsize
```

这行是必需的，因为函数无法存取定义在函数外面的变量。这一行告诉 Python 我们将在本函数中使用变量 boxsize，而该变量是在函数外部定义的。

可能最让人困惑的就是这一节：

```
#add key listeners
window.onkeypress(up, "Up")
window.onkeypress(left, "Left")
window.onkeypress(right, "Right")
window.onkeypress(back, "Down")
window.onkeypress(quitTurtles, "Escape")
```

这段代码告诉窗体当不同的键按下时应该做什么。例如，第一行表示当“向上”键（键盘方向键的向上按键）按下时，运行函数 up（我们已经定义好了）。

2.2.3 在游戏中循环

接下来就到了游戏的主循环中：

```
while not caught:
    cat.setheading(cat.towards(mouse))
    cat.forward(8+difficulty)
    score = score + 1
```

```

if cat.distance(mouse) < 5:
    caught = True
time.sleep(0.2-(0.01*difficulty))

```

代码中使用了不同的循环方法。while 循环如下：

```

while condition:
    loop code

```

它表示如果条件 (condition) 为真，将执行后面的循环代码 (loop code)。在程序的变量列表中，可以看到变量 caught 被赋为 False：

```
caught = False
```

因此，这里使用的条件为 not caught (因为 not False 为真，程序开始时该条件为真)，程序一直运行到 caught 被赋值为真，因为 not True 表示假。这样描述会有点复杂。但如果按字面意思就很容易理解：前面的 not 表示将真和假反一下。

time.sleep() 告诉 Python 停止在给定的时间，单位为秒。本例中我们随着难度等级（由用户输入一个值存在变量中）增加而减少程序等待时间。你会发现，随着难度增加，猫的运动速度也在增加。



此时也许你很想知道如何记住各个模块中对应的所有方法。比如，怎么才能知道应该使用 forward(10) 而不是 forwards(10) 或者 move_forwards(10)，或者怎么才能知道哪里可以找到向前移动的方法呢？使用 Python 不需要有超人的记忆力，你只需要知道在哪里可以查看帮助就好了。如果使用 IDLE3，按下 F1 Python 文档就会显示在 Web 浏览器里。这里有很多有用信息，非常值得浏览一下。例如需要查看 turtle 模块的信息，只需要在快速查找窗口中输入 turtle，然后选择第一项，你将会看到，我们已经使用过它的方法了。

2.3 小结

我们的 Python 快速导览到此为止。希望这些程序能够帮助你理解它。不用担心无法百分之百地理解全部内容，我们会在第 3 章比较详细地介绍 Python 的其他部分。无论如何，还是希望你能理解下列几个方面：

- Python 程序由一系列命令组成并从上到下执行。
- 可以通过循环和 if 语句控制程序的执行顺序。
- 不必事必躬亲，通过导入模块，使用模块中的方法可以完成许多工作。



- 函数可以帮助重用代码，也可以使程序变得易于理解和维护。
- 变量可以存储信息以便后面使用。

在 Python 中画一朵花或者写个游戏非常容易。

请记住，编程时，完成一件事情可以有多种方法。如果选对了路（或者至少不走错路），工作会变得更轻松。然而，有时并不能轻易判断出对错。因此，我们以 Python 对此的建议来结束本章。在 Python 解释器中，输入：

```
>>> import this
```

this 是一个特殊的模块。导入之后可以输出很多有用的 Python 建议。现在，你对 Python 已经有一些了解，让我们更进一步去探索 Python 代码吧。

Python 基础

在前面章节中，我们直接深入 Python 程序中，希望能够给你一个概念——什么是 Python。但你还不清楚它具体是如何工作的。本章中，我们将回答这些问题，并详细介绍如何在 Python 中创建自己的程序。在后继的章节中，我们将分别介绍 Python 的不同特性，帮助你为树莓派编写不同类型的程序。

3.1 变量、值和类型

第 2 章中，我们看到变量可以将数据存储下来供我们在别的地方使用。它们是程序员手里的一个强大工具。现在让我们来看看它们究竟是什么。如果你之前有过其他语言的编程经验，在这里会发现 Python 和其他语言有点不同。

在 Python 解释器中输入语句：

```
>>> score = 0
```

它告诉 Python 你想使用一个名字为 `score`，值为 0 的变量。在此之后，Python 只要看到 `score`，就会用值 0 来替换 `score`。为了验证这一点，继续输入：

```
>>> print(score)
```

请记住，Python 是顺序执行我们的命令的，在使用 `score` 之前必须先给它赋值。否则，Python 将会报错。

如果想改变 score 的值，只需要给它赋一个新值，如：

```
>>> score = 1
```

现在 Python 再遇到 score 时就会用 1 来替换它（你可以再次执行 print(score) 来验证一下）。你也可以在更新它的值时使用它：

```
>>> score = score + 1
```

变量几乎可以使用任何名字但必须以字母或下划线开始，并且不能使用 Python 关键字（如 if、for 等）。Python 的命名习惯是使用小写字母，用下划线将单词分开，如：

```
high_score = 1000
```

在前面的例子中，所有的值都是数字，然而，值不仅可以是数字，也可以文字，如：

```
player_name = "Ben"
```

我们甚至可以把同一个变量轮换赋值成数字和文字，例如：

```
>>> our_variable = 1000
>>> print(our_variable)
>>> our_variable = "Some Text"
>>> print(our_variable)
```

然而，变量的当前值只能是一种类型。

3.1.1 值和类型

看到数字 3 时，你只是看到一个 3，而不关心它究竟是一个文字，还是数字。3 就是 3。Python 却不一样。每个数据都有特定的类型，这样 Python 才知道该如何处理它们。通过函数 type() 可以看到 Python 数据的类型。在 Python 解释器中输入：

```
>>> type(3)
<class 'int'>
>>> type("3")
<class 'str'>
```

Python 告诉我们，第一个是 int（整数 integer 的简写），第二个是 str（字符 string 的简写）。这是因为 Python 认为整数 3 和字符 3 是不同的。执行下面这两行代码可以明显看出它们之间的区别：

```
>>> 3+3
6
>>> "3" + "3"
33
```

第一行将两个数字加一起，而第二行却是将两个字符合并在一起。由此可见，区分值的类型非常重要，如果出错，将会得到非常有意思的结果。为了探索更多的类型，可以输入：

```
>>>type(3.0)
<class 'float'>

>>>type('3>2')
<class 'bool'>
```

第一行输出 float(一个浮点数表示一个实数，小数点位置不固定)。第二行输出 bool(布尔类型，只有两个值：True 和 False)。

3.1.2 数字排序

数据的具体类型决定了 Python 可以执行哪些操作。这里我们从数值开始（包括 int 和 float 类型，但不包括 string，虽然它也包含数字）。对于数值，可以有两种操作类型：比较和数值操作。比较，需要两个操作数，返回值为 bool 型。如表 3-1 所示。

表 3-1 数值类型的比较操作

操作符	含义	例子
<	小于	$3 < 2 \rightarrow \text{False}$
>	大于	$3 > 2 \rightarrow \text{True}$
$=$	等于	$3 = 3 \rightarrow \text{True}$
\leq	小于等于	$3 \leq 3 \rightarrow \text{True}$
\geq	大于等于	$3 \geq 4 \rightarrow \text{False}$
\neq	不等于	$3 \neq 4 \rightarrow \text{True}$

数值操作返回一个数值类型，如表 3-2 所示。

表 3-2 数值操作

操作符	含义	例子
+	加	$2 + 2 \rightarrow 4$
-	减	$3 - 2 \rightarrow 1$
*	乘	$2 * 3 \rightarrow 6$
/	除	$10 / 2 \rightarrow 5$
%	求余	$5 \% 2 \rightarrow 1$
**	乘方	$4 ** 2 \rightarrow 16$
int()	转换为 int 型	$\text{int}(3.2) \rightarrow 3$
float()	转换为 float 型	$\text{float}(2) \rightarrow 2.0$

你可以在 Python 解释器中输入任何一个操作符来验证一下。例如：

```
>>> 3!=3
False
```

在程序中使用数值运算，通常都将其返回值赋值给某个变量。例如：

```
>>> number_1 = 10
>>> number_2 = number_1**2
```

3.1.3 使用 Strings 保存文字

string 类型可以用来保存任何文字。创建字符串只需要将数据用单引号或者双引号括起来就可以了。在 Python 中，不论哪种引号都可以。我们首选双引号，因为它可以处理带'号（单引号或撇号）的字符串。但这绝不是普遍适用的。有些程序员喜欢使用单引号，因为其输入起来更方便。

这个数据类型不同于其他类型，因为许多时候，string 不只是单个数据而是一组字母。它的名字也反映了这个特点——字符“串”，一串字符。

和数值类型一样，Python 也为我们提供了一些操作方法。表 3-3 给出了一些常用的操作。

表 3-3 字符串操作

操作符	含义	例子
string[x]	获取第 x 个字符（从 0 开始数）	"abcde"[1] → "b"
string[x:y]	获取所有从 x 到 y 的字符	"abcde"[1:3] → "bc"
string[:y]string[:y]	获取从字符串开始到第 y 个的字符	"abcde"[:3] → "abc"
string[x:]	获取从第 x 个开始到字符串结束的字符	"abcde"[3:] → "de"
len(string)	返回字符串长度	len("abcde") → 5
string+string	合并两个字符串	"abc" + "def" → "abcdef"

3.1.4 布尔值：真或假

最后我们来看 bool 类型。它非常简单，只有两种可能取值：True 和 False。注意在 Python 中，这两个值的首字母要大写，并且不需要任何引号。同时，这个值通常不存在变量中（虽然我们在第 2 章中见到过存在变量中），它通常用于条件语句如 if 的判断条件中，我们将在本章后面部分详细介绍。其主要操作符是与 (and)、或 (or) 和非 (not)。

非，就是简单地转换下取值：

```
>>> not True
False
```

```
>>> not False
True
```

与，需要两个操作数，如果两个数都为真，则返回真，否则，返回假：

```
>>> True and False
False
```

```
>>> False and False
False
```

```
>>> True and True
True
```

或，也需要两个操作数，如果两个数中任何一个为真，则返回真：

```
>>> True or False
True
```

```
>>> True or True
True
```

```
>>> False or False
False
```

3.1.5 数据类型转换

使用函数 `int()`、`float()` 和 `str()` 可以转换数据类型。它们分别将其他数据类型转换为整数、浮点数和字符串。然而它们却不能随意转换。如果将浮点数转为整数，Python 将舍去所有小数部分。当字符串中只有一个字符时，才能转换成数字。但是，其他类型几乎都可以转换成字符串。下面给出了一些例子。

```
>>> print(int(3.9))
3
>>> print(str(True))
True

>>> print(float("Three point two"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'Three point two'
```

3.1.6 知识测试

下面是一些 Python 语句。看一下你是否能弄懂它们的意思。想出来什么意思之后，将它们输入 Python 解释器检验自己的判断。提示：有一些语句可能会导致错误。

```

■ int("two")
■ print(str(3+3) + "3")
■ type(3=3)
■ "4" == 4
■ "Python"[4]
■ (3 > 2) or (2 > 3)
■ not "True"
■ 2345[2]
■ str((not True) and (not False))
■ 10 % 3

```

练习 1

下面程序中哪些是变量，它们的值是什么，类型是什么？答案在本章结尾。如果不确定，可以在代码中的某些地方加上 `print` 语句，并输入 Python 解释器中看看结果。当程序的运行结果和我们期望的不一样时，这个方法很有用，可以用来确定某些代码究竟做了什么事。

```

prompt_text = "Enter a number: "
user_in = input(prompt_text)
user_num = int(user_in)

for i in range(1,10):
    print(i, " times ", user_num, " is ", i*user_num)

even = (user_num % 2) == 0

if even:
    print(user_num, " is even")
else:
    print(user_num, " is odd")

```

3.2 在结构体中存储值

除了简单数据类型，Python 还允许我们将数据用不同方式组合起来创建结构体。最简单的结构体是 `sequences`（线性结构）。它将信息一个接一个地存储起来。它分为有两类：`lists`（列表）和 `tuples`（元组）。多数情况下，它们是相似的。

来看接下来的例子：

```
>>> list_1 = [1, 2, 3, 4]
>>> tuple_1 = (1, 2, 3, 4)
>>> list_1[1]
2
>>> tuple_1[1]
2
```

我们用方括号将数字括起来构成列表，用圆括号将数字括起来构成元组。到现在为止，它们两个工作起来都是一致的。在结构体名后面跟方括号，方括号中填下标就可以访问单个元素。注意下标从 0 开始，因此 `list_1[0]` 和 `tuple_1[0]` 可以访问线性结构中的第一个元素。

当你去更新元素时就会发现列表和元组间的差别：

```
>>> list_1[1]=99
>>> list_1
[1, 99, 3, 4]
>>> tuple_1[1]=99
Traceback (most recent call last):
File "<pyshell#35>", line 1, in <module>
    t1[1]=99
TypeError: 'tuple' object does not support item assignment
```

可见，你可以更新列表中的单个元素，却不能更新元组中的单个元素。然而，你可以一次性覆盖元组中的所有元素。这时，可以告诉 Python 将变量 `tuple_1` 赋一个新值以取代旧值。

```
>>> tuple_1
(1, 2, 3, 4)
>>> tuple_1=(1,99,2,4)
>>> tuple_1
(1, 99, 2, 4)
```

上一节中我们提到的 strings 就是一串字符，它的操作符可以用于列表和元组。下面继续以 `list_1` 和 `tuple_1` 为例：

```
>>> len(list_1)
4
>>> tuple_1[:3]
[1, 99, 2]
```

参考表 3-3，复习下我们可以在这里使用的字符串操作。

列表和元组中的元素可以是任意数据类型，包括列表和元组自身。如果愿意，你可以创建列表的列表的列表的列表。当然，如果真的这样做了，你的代码将会变得异常难懂。

但列表的列表，通常会比较有用。你可以把它当做一个二维表：

```
>>> list_2 =
  [["a", "b", "c"], ["d", "e", "f"], ["g", "h", "i"], ["j", "k", "l"]]
```

你可以通过主列表和子列表索引来获取元素：

```
>>> list_2[2][0]
'g'
```

```
>>> list_2[0]
['a', 'b', 'c']
```

```
>>> list_2[0][1:]
['b', 'c']
```

从表 3-4 中可以看出来为什么它可以当做二维表。

表 3-4 二维列表示例

list_2[0][0] = "a"	list_2[1][0] = "d"	list_2[2][0] = "g"	list_2[3][0] = "j"
list_2[0][1] = "b"	list_2[1][1] = "e"	list_2[2][1] = "h"	list_2[3][1] = "k"
list_2[0][2] = "c"	list_2[1][2] = "f"	list_2[2][2] = "i"	list_2[3][2] = "l"

有许多方法可以用来操纵列表。表 3-5 给出了常用的方法。

表 3-5 列表操作方法（以 list_3 = [3, 2, 1] 为例）

操作符	含义	例子
list.append(item)	添加元素到列表尾部	list_3.append(0) → [3, 2, 1, 0]
list.extend(list_2)	合并 list_2 到列表尾部	list_3.extend([-1]) → [3, 2, 1, 0, -1]
list.pop(x)	返回并删除第 x 个元素	See below
list.insert(x, item)	插入元素到第 x 个位置	list_3.insert(99, 1) → [3, 99, 2, 1]
list.sort()	排序列表	list_3.sort() → [1, 2, 3]
list.index(item)	返回列表中第一次出现该元素的位置	List_3.index(2) → 1
list.count(item)	计算列表中该元素出现的次数	list_3.count(2) → 1
list.remove(item)	删除列表中第一次出现的该元素	list_3.remove(2) → [3, 1]

这里的多数例子都和我们之前遇见的不一样，因为它们（除了 index() 和 sort()）改变了 list_3 的值而不是返回一些值。例如，在 Python 解释器中运行第一个例子，当然，你需要额外的一行来显示 list_3 的值。

```
>>> list_3 = [3, 2, 1]
>>> list_3.append(0)
>>> list_3
[3, 2, 1, 0]
```

然而，`index()` 和 `count()` 仅仅返回一个值：

```
>>> list_3.index(2)
1
```

`pop(x)` 有点特殊，因为它一次做了两件事。首先，它返回列表中第 x 个位置的元素值，同时它还从列表中删除了该元素。试着运行下面的例子，感受下它是如何工作的：

```
>>> list_3 = [1, 2, 3]
>>> out = list_3.pop(1)
>>> out
2
>>> list_3
[1, 3]
```

上面说过，元组除了不能被修改，它和列表非常类似。所有用于列表操作，只要不改变元素值，都可以用于元组：

```
>>> tuple_2 = (1, 2, 3)
>>> tuple_2.index(2)
1
>>> tuple_2.sort()
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <module>
    tuple_2.sort()
AttributeError: 'tuple' object has no attribute 'sort'
```

3.2.1 字典和集合中的非序列元素

你可以认为列表和元组是元素的集合，每个元素都对应了其中的一个下标。例如，在列表 [“a”, “b”, “c”, “d”] 中，a 的下标是 0，b 的下标是 1，以此类推。然而，当你想用非数字的下标时该怎么做呢？例如，你想要创建一个数据结构，把朋友的昵称和真实名字关联起来，如：

```
>>> real_name["Benny"]
'Benjamin Everard'
```

在 Python 中，可以使用通过花括号来定义的字典（dictionary）。你可以使用下列语句创建字典 `real_name`：

```
>>> real_name = {"Benny" : "Benjamin Everard",
                 "Alex" : "Alexander Bradbury"}
```

字典中的元素称为键值对（key/value pair），其中第一部分（本例中的昵称）是键

(key), 第二部分 (全名) 是值 (value)。只需要给定一个新 key 及其对应的值就可以在字典中新加元素：

```
>>> real_name["Liz"] = "Elizabeth Upton"
>> real_name
{'Alex': 'Alexander Bradbury', 'Benny': 'Benjamin Everard',
 'Liz': 'Elizabeth Upton'}
```

你可能想知道为什么需要下标或者键。实际上，这些并不是必需的。Python 中的集合 (set) 允许你把一堆数据放在一起而不用指定下标或序号。例如：

```
>>> herbs = {'thyme', 'dill', 'corriander'}
>>> spices = {'cumin', 'chilli', 'corriander'}
>>> "thyme" in herbs
True
```

可以看出，Python 使用 in 操作来测试给定的值是否在集合中。还有些其他操作可以应用到集合中。参见表 3-6。

表 3-6 集合的操作（例子中使用了先前定义的集合）

操作符	含义	例子
set_1 & set_2	返回两个集合共有的元素	herbs & spices→{'corriander'}
set_1 set_2	合并两个集合中的元素	herbs spices→{'dill', 'thyme', 'chilli', 'corriander', 'cumin'}
set_1 - set_2	set_1 中存在 set_2 中不存在的元素	herbs - spices→{'dill', 'thyme'}
set_1 ^ set_2	set_1 或 set_2 中存在的元素，不包括两个集合共有的元素	herbs ^ spices→{'dill', 'cumin', 'thyme', 'chilli'}

3.2.2 知识测试

下面的 Python 语句是什么意思？试想一下，然后将它们输入 Python 解释器检验下自己的判断。提示：有一些语句会导致错误。

- ["a", "b", "c"].index("c")
- (3, 2, 1).pop(2)
- {1, 3, 5} & {2, 4, 6}
- {1, 2, 3} & {1}
- 3 in {1, 2, 3} ^ {3, 4, 5}
- "abcde".remove("c")
- 3 not in (1, 2, 3)

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`



3.3 控制程序流程

`while` 循环是一种最简单的循环。只要结果是布尔类型的任何语句都可以做它的判断条件，它将会持续循环到条件为假。如果条件始终为真，它将一直循环下去。例如：

```
>>> while True:  
    print("Ben is awesome")
```

但愿你还记得第 2 章的循环体。在条件后面要加上冒号，接下来的一行要有缩进，所有缩进部分都属于循环体。要在 Python 解释器中运行这段代码，输入 `print` 语句之后要按回车键（Enter）。用退格键（backspace）删掉自动产生的 tab（缩进），然后再按回车键。这是告诉 Python 循环体结束了，你需要执行这段代码。

这段代码会陷入死循环，按 `Ctrl+C` 组合键可以终止它。

无论判断条件选择的多么复杂都可以，只要它的返回值是真或假就可以了。但为了最终能够跳出循环，通常需要一个或多个变量以方便在循环内部改变判断条件。比如说，下面这个猜数字游戏中的主循环。

```
import random  
secret = int(random.uniform(0,10))  
print("I'm thinking of a number between zero and ten."  
     , "Can you guess what it is?")  
guess = 11  
  
while guess != secret:  
    guess = int(input("Take a guess: "))  
  
print("Well done!")
```

注意，如果输入非数字的字符，这个程序将报错。后面我们会介绍如何简单地修复这个问题，但让我们先来看一下另一种循环。

3.3.1 用循环遍历数据

`for` 循环可以用来遍历数据，它在每次循环中对一个数据进行处理。通常我们在 `range(x, y)` 的帮助下，遍历从 `x` 到 `y` 的每一个数据^②。例如，你可以得出一个 12 倍表：

```
>>> for i in range(1,13):  
    print(i, " times twelve is ", i*12)
```

`range()` 还有一个参数，可以设定两个连续数字之间的间隔。例如我们把 `range(1,13)` 改成 `range(2,14,2)`，它将会只计算从 2 到 14 的所有偶数^③。我们还可以使用其他线性数据类

② 包含 `x`，不包含 `y`。——译者注

③ 不包括 14。——译者注

型（包括字符串）和集合来控制 for 循环。下列例子都是合法的：

```
>>> for i in [1,2,3,4,5,6,7,8,9,10,11,12]: print(i, " times 12 is "
      ", i * 12)
>>> for i in (1,2,3,4,5,6,7,8,9,10,11,12):
      print(i, " times 12 is ", i*12)
>>> for i in {1,2,3,4}:
      print(i, " times 12 is ", i*12)
>>> for i in "123456789":
      print(i, " times 12 is ", int(i)*12)
```

最后一行是一个很好的例子，它展示了使用错误的数据类型会有什么结果。集合和其他类型有点不一样，因为它没有顺序，只是一堆数据的集合。下面两个例子的执行结果是一样的。

```
>>> for i in {1,2,3,4}:
      print(i, " times 12 is ", i*12)

>>> for i in {4,3,2,1}:
      print(i, " times 12 is ", i*12)
```

字典有点特别，因为它保存的不是元素，而是键值对。for 循环中可以使用它，但必须做些特殊处理：

```
>>> real_name = {"Benny": "Benjamin Everard",
                  "Alex": "Alexander Bradbury"}
>>> for key,value in real_name.items():
      print("The real name of " + key + " is " + value)
```

使用集合，你就不能控制循环中访问数据的顺序。实际上，如果你期望用 for 循环顺序访问数据，则需要使用线性数据结构，如列表或元组，而不是使用像集合或字典这样的无序数据结构。

3.3.2 深入理解循环嵌套

编程时经常需要同时遍历多种数据，比如下面这个用来找出 1 到 30 之间所有素数的程序。

```
for i in range(1,30):
    is_prime = True
    for k in range(2,i):
        if (i%k) == 0:
            print(i, " is divisible by ", k)
            is_prime = False
    if is_prime:
        print(i, " is prime ")
```

请注意缩进级别，第一个循环体的缩进为一，第二个循环体的缩进为二。这一点非常重要，只有这样，Python 才能理解哪个代码属于哪个循环体，以及每个循环体在何处结束。

使用嵌套时要注意，它可能会使程序变慢。上面的例子运行得很快，但如果你试图计算 3000 以内的素数（只需要在第一行的 30 后面加两个零），程序运行就会花非常长的时间。不只是因为外层循环要多循环上千次，每次走进内层循环也需要多执行很多次。你会发现整个程序运行起来很慢（如果你在做这个实验，记得按 Ctrl+C 组合键可以停止它）。幸运的是，我们还可以改进它。试一下这个程序：

```
for i in range(1,3000,2):
    is_prime = True
    for k in range(2,i):
        if (i%k) == 0:
            print(i, " is divisible by ", k)
            is_prime = False
            break
    if is_prime:
        print(i, " is prime ")
```

首先，使用 `range(1, 3000, 2)` 跳过所有的偶数。我们直接省去一半时间。其次，你可能注意到了，在 `if` 里面我们多加了一个 `break`。这会让我们节约更多时间。我们是在计算素数，因此不用关心每个数的所有约数。一旦发现这个数字是非素数^①，就可以直接使用 `break` 跳出循环，继续执行下面一行 (`if is_prime:`)。这两点优化可以使程序运行得更快。

3.3.3 使用 if 语句控制程序分支

不仅可以使用循环来不断执行某段代码，还可以使用分支来控制 Python 程序流，使其根据不同条件，执行不同的代码。分支由 `if` 语句实现。我们已经见过很多次 `if` 语句，现在来复习一下。`if` 语句，像 `while` 循环一样，只需要一个布尔类型的条件。它后面还可以有附加语句如 `elif` (`else-if`) 和 `else` 语句。例如：

```
num = int(input("enter a number: "))
if num%2 == 0:
    print("Your number is divisible by 2")
elif num%3 == 0:
    print("your number is divisible by 3")
elif num%5 == 0:
    print("your number is divisible by 5")
else:
    print("your number isn't divisible by 2,3 or 5")
```

^① 除了 1 和它本身还有其他约数。——译者注

一个 if 语句最多只执行一段代码，只要 Python 发现条件为真，就执行该段代码并结束整个 if 语句。因此如果在上面的程序中输入 10，它将只返回该数字可以被 2 整除，而不会试下该数字能否被 5 整除（如果你期望它试一下，就需要写一个新的 if 语句而不是用 elif）。如果没有一个条件为真，则执行 else 后面的代码段。在其他例子中可以看到，if 语句可以不带 elif 或 else。如果没有 else 语句，同时判断条件也不成立，Python 就会跳过 if 语句，不执行其中的任何代码。

3.3.4 捕获异常

如果你喜欢尝试，在前面的例子中试图输入过非数字的字符，就会发现程序会报错。这是因为 Python 不能把任意字符转换成数字，它不知道该怎么做了。此时，Python 就会报出错误。目前为止，我们只是放任它这么做，然后，就会导致程序崩溃。

然而，如果你知道某段代码可能会报出异常，可以告诉 Python，这样程序就知道遇到问题时该如何处理。可以参考下面代码：

```
try:  
    code where there might be a problem  
except type_of_error:  
    code to run if there's an error
```

你可以告诉 Python 要处理的错误类型，因为 Python 在异常时会输出错误类型。如：

```
>>> num = int(input("Enter a number: "))  
Enter a number: dasd  
Traceback (most recent call last):  
  File "<pyshell#176>", line 1, in <module>  
    num = int(input("Enter a number: "))  
ValueError: invalid literal for int() with base 10: 'dasd'
```

可以看到，这是个 ValueError（值错误）。因此前一个例子可以改成：

```
is_number = False  
num = 0  
  
while not is_number:  
    is_number = True  
    try:  
        num = int(input("enter a number: "))  
    except ValueError:  
        print("I said a number!")  
        is_number = False  
  
if num%2 == 0:  
    print("Your number is divisible by 2")  
elif num%3 == 0:  
    print("your number is divisible by 3")
```

```

elif num%5 == 0:
    print("your number is divisible by 5")
else:
    print("your number isn't divisible by 2,3 or 5")

```

练习 2

试着使用 try 语句来捕获由于用户输入错误导致的异常，修复本章开始的猜数字游戏。

3.4 使用函数复用代码

我们已经使用过一些函数，如 print() 和 input()。这些都是 Python 内置函数。你还可以自己定义函数。第 2 章中你已经定义过自己的函数，这里我们来复习一下。看下这个：

```

>>> def square(num):
    return num**2
>>> square(4)
16

```

这段代码定义了一个名叫 square（平方）的函数，然后使用这个函数计算了 4 的平方。使用它的时候需要在名字后面跟一个包含在小括号里的数字——我们称为参数。函数运行时，参数名（本例中是 num）将会被赋值为你给定的数字。return 语句可以出现在程序中的某个地方，用来给主程序返回数据。如果有多个 return 语句，Python 将在第一次遇到 return 时返回。

你也可以创建包含多个参数的函数。例如，下面的程序就有一个函数需要两个参数并返回其中较大的一个。

```

def biggest(a,b):
    if a>b:
        return a
    else:
        return b

print("The biggest of 2 and 3 is ", biggest(2,3))
print("The biggest of 10 and 5 is ", biggest(10,5))

```

至此为止，它们都工作地很好。然而如果需要在函数中改变了变量的值呢？看看下面的程序：

```

def add_one(num):
    num = num + 1
    return num

number_1 = 1
number_2 = add_one(number_1)

```

```
print("number_1: ", number_1)
print("number_2: ", number_2)
```

运行这段程序之前，先想一下运行结果。可以肯定的是，`number_2` 将会是 2，但 `number_1` 呢？它应该是 1 吗？因为我们在主程序中给它赋值为 1。还是 2 呢？因为我们把它传递给函数 `add_one()` 并在里面改变了它的值。

运行后，你会发现 `number_1` 是 1。这是因为这一行：

```
num = num + 1
```

你告诉 Python 不想让 `num` 继续保持原来的值（和 `number_1` 一样），而是给它一个加 1 后的新值。

然而，可变数据类型如列表、集合和对象就不一样了。如果它们中的一种数据类型传入函数并改变它，那么作为参数传入的原始值也会跟着变化。比如：

```
def add_item(list_1):
    list_1.append(1)
    return list_1

list_2 = [2,3,4]
list_3 = add_item(list_2)
print("list_2: ", list_2)
print("list_3: ", list_3)
```

运行之后将得到：

```
list_2:  [2, 3, 4, 1]
list_3:  [2, 3, 4, 1]
```

通常情况下，这样做没有什么问题，但有时你并不想让作为参数传入的原始值发生变化。这种情况下，你需要使用 `copy.deepcopy()`，不过首先你得导入 `copy` 模块。如果你把上个例子改为：

```
import copy
def add_item(list_1):
    list_1.append(1)
    return list_1

list_2 = [2,3,4]
list_3 = add_item(copy.deepcopy(list_2))
print("list_2: ", list_2)
print("list_3: ", list_3)
```

你将得到：

```
list_2:  [2, 3, 4]
list_3:  [2, 3, 4, 1]
```

可选参数

有时你需要创建这样一个函数，它有时候需要参数，有时候不需要。比如，你可能需要创建函数 `increment()`，该函数接受两个参数并将它们相加。当只给一个参数时，该函数就把这个参数加上 1。这时候你就可以使用默认值（本例中为 1）作为参数[⊖]：

```
def increment(num1=1, num2):
    return num2 + num1
```

这里唯一需要注意的就是可以省略的参数必须放在其他参数之后。

3.5 组合装配

本章中我们已经介绍过很多东西，但还没有介绍如何把这些东西组装起来，创建一个你想要的程序。本节中，我们将介绍一个简单的学生成绩数据库程序，其中使用的大多数知识都是在前面讲过的。

该程序包含一些默认数据并允许你修改它。我们将使用合适的数据类型和数据结构来存储不同的信息，使用不同的函数来操纵这些数据。我们甚至还加了个简单的菜单以方便用户管理这些数据。

 **注意** 本书中大多数较长代码的例子都可以从网站 www.wiley.com/go/python-raspberrypi 下载。为避免输入错误，你可以下载，复制并粘贴到 IDE 或代码编辑器中。

代码如下（你也可以从本书网站上找到它，名字是 `chapter3-student-1.py`）

```
students = [[{"Name": "Ben", "Maths": 67, "English": 78, "Science": 72},
            {"Name": "Mark", "Maths": 56, "Art": 64, "History": 39,
             "Geography": 55},
            {"Name": "Paul", "English": 66, "History": 88}]]
```

```
grades = ((0, "FAIL"), (50, "D"), (60, "C"), (70, "B"),
          (80, "A"), (101, "CHEAT!"))
```

```
def print_report_card(report_student=None):
    for student in students:
        if (student[0] == report_student) or
           (report_student == None):
            print("Report card for student ", student[0])
            for subject, mark in student[1].items():
                print(subject, mark)
```

[⊖] 原文如此，实际上 `num1 = 1` 应该放在后面，因为默认参数需要放在参数表后面。——译者注

```

        for grade in grades:
            if mark < grade[0]:
                print(subject, " : ", prev_grade)
                break
            prev_grade = grade[1]

def add_student(student_name):
    global students
    for student in students:
        if student[0] == student_name:
            return "Student already in database"
    students.append([student_name, {}])
    return "Student added successfully"

def add_mark(student_name, subject, mark):
    global students
    for student in students:
        if student[0] == student_name:
            if subject in student[1].keys():
                print(student_name, " already has a mark for ", subject)
                user_input = input("Overwrite Y/N? ")
                if user_input == "Y" or "y":
                    student[1][subject] = mark
                    return "Student's mark updated"
                else:
                    return "Student's mark not updated"
            else:
                student[1][subject] = mark
                return "Student's mark added"
    return "Student not found"

while True:
    print("Welcome to the Raspberry Pi student database")
    print("What can I help you with?")
    print("Enter 1 to view all report cards")
    print("Enter 2 to view the report card for a student")
    print("Enter 3 to add a student")
    print("Enter 4 to add a mark to a student")
    print("Enter 5 to exit")

    try:
        user_choice = int(input("Choice: "))
    except ValueError:
        print("That's not a number I recognise")
        user_choice = 0

    if user_choice == 1:
        print_report_card()
    elif user_choice == 2:
        enter_student = input("Which student? ")

```

```

print_report_card(enter_student)
elif user_choice == 3:
    enter_student = input("Student name? ")
    print(add_student(enter_student))
elif user_choice == 4:
    enter_student = input("Student name? ")
    enter_subject = input("Subject? ")
    num_error = True
    while num_error:
        num_error = False
        try:
            enter_mark = int(input("Mark? "))
        except ValueError:
            print("I don't recognise that as a number")
            num_error = True
    print(add_mark(enter_student, enter_subject, enter_mark))
elif user_choice == 5:
    break
else:
    print("Unknown choice")
input("Press enter to continue")
print("Goodbye and thank you for using the Raspberry Pi"
      , "Student database")

```

目前，该程序的学生数据结构是字典的列表的列表^②。使用字典的字典或者列表的列表的列表都可以完成相同的效果。试着分别使用这两种数据结构修改程序，感觉下哪种方式更自然一点。

练习 3

该程序中的数据可以用元组来存取么？为什么呢？答案详见本章末尾。

3.6 使用类来构建对象

类允许你将数据和函数组合起来构成一个对象。实际上，我们在前面章节中已经使用过它。回忆下这两行：

```
window = turtle.Screen()
babbage = turtle.Turtle()
```

当时我们只是快速跳过去，现在让我们来看看它是如何工作的。`turtle.Turtle()` 返回一个由 `turtle` 模块中 `Turtle` 类创建的对象。同样的，`turtle.Screen()` 返回一个由 `turtle` 模块中

^② 各科成绩用字典保存，每个学生和自己的成绩组成一个列表，最后所有学生共同构成最外面的列表。
——译者注

Screen 类创建的对象。简而言之，类是用来构建对象的蓝图。对象可以存储数据，并且提供可以让你操作数据的方法。而方法其实也就是类中的函数。

你已经见识过对象是多么有用了。在第 2 章的例子中，你不用关心 turtle 的数据是如何存放的，因为它们已经包含在对象中了。你只需要将 turtle 对象保存在一个叫 babbage 的对象里，当调用某个方法时，该方法就知道如何存取它需要的各种东西。这样可以使程序整洁易用。例如下面的例子：

```
babbage.forward(100)
```

这行代码将 turtle 向前移动并将结果画在屏幕上。在屏幕上画这条线，它知道用什么颜色的画笔，turtle 的起始位置在哪里以及其他各种它所需要的信息，因为它们都已经存储在对象中了。

让我们用一个简单的例子看看对象里都有什么东西：

```
class Person():
    def __init__(self, age, name):
        self.age = age
        self.name = name

    def birthday(self):
        self.age = self.age + 1

ben = Person(31, "Ben")
paul = Person(42, "Paul")
print(ben.name, ben.age)
print(paul.name, paul.age)
```

这里有几点需要注意，Python 中，变量、函数和方法的名字通常用小写字母，类是个例外。因此 Person 类由大写字母 P 开头。如果不这样做 Python 也不会报错，但遵守惯例可以方便人们阅读对方的代码。可以看到，方法的定义方式和函数一样，区别只是参数总是以 self 开始，这表示本地变量。在这个例子中，本地变量包括 self.age 和 self.name。它们会在类的每一个实例中都创建一份。本例中，我们用 People 类创建了两个对象（即类的实例），每个对象都有各自的一份 self.age 和 self.name 拷贝。我们可以在对象外面读写它们（就像我们在 print 方法中使用的那样）。这被称为 Person 类的属性。

这里还有两个方法。`__init__` 是每个类都有的特殊方法。该方法在创建或“初始化”类的实例时会被调用。因此，`ben = Person(31, "Ben")` 会创建 Person 类的一个对象，并使用参数 (31, "Ben") 调用 `__init__` 方法。通常可以用来设置属性。第二个方法 `birthday()` 展示了如何使用类方法而不用在类的外面关心数据的保存问题。给 Person 对象一个 `birthday()` 方法，拿来用就可以了，例如：

```
ben.birthday()
```

它将把 age 加一。

有时，我们不希望从头开始创建类，而是根据已经存在的类来建一个新的类。例如，如果你想创建一个类来保存 parents（父母）的相关信息，它们也存在年龄（age），名字（name），和生日（birthday），如果再为 Parent 写一遍这些代码就会显得浪费。Python 允许我们从其他类中继承。下面给出一个例子：

```
class Person():
    def __init__(self, age, name):
        self.age = age
        self.name = name

    def birthday(self):
        self.age = self.age + 1

class Parent(Person):
    def __init__(self, age, name):
        Person.__init__(self, age, name)
        self.children = []

    def add_child(self, child):
        self.children.append(child)

    def print_children(self):
        print("The children of ", self.name, " are:")
        for child in self.children:
            print(child.name)

john = Parent(60, "John")
ben = Person(32, "Ben")
print(john.name, john.age)
john.add_child(ben)
john.print_children()
```

Person 是 Parent 的超类，Parent 是 Person 的子类。把类名放入要定义的类名后面的括号里，它就变成这个要定义的类的超类。你可以调用超类的 `__init__` 方法，会自动获得超类的属性和方法的访问权限而不用重写代码。

类的最大优势就是它可以方便重用代码。像我们在前几章中看到的那样，它可以方便地操纵 turtle 而不用关心它做了些什么，是怎么做的。因为 turtle 类封装了这些信息，你只要知道方法名字，就可以毫无障碍地使用它们了。贯穿本书，你将看到如何使用类提供的方法方便地构建复杂游戏，而不用担心这些方法的技术细节是如何实现的。

下面的代码使用类重写了学生数据库程序，感受下类是如何工作的（文件名 chapter3-

```

student-2.py)  :/home/muhammad/Downloads/Python/Ch3

student_data= [{"Ben", {"Maths": 67, "English": 78,
                      "Science": 72}}, {"Mark", {"Maths": 56, "Art": 64, "History": 39,
                      "Geography": 55}}, {"Paul", {"English": 66, "History": 88}}]

grades = ((0, "FAIL"), (50, "D"), (60, "C"), (70, "B"), (80, "A"),
          (101, "CHEAT!"))

class Student():
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def print_report_card(self):
        print("Report card for student ", self.name)
        for subject, mark in self.marks.items():
            for i in grades:
                if mark < i[0]:
                    print(subject, " : ", prev_grade)
                    break
            prev_grade = i[1]

    def add_mark(self, subject, mark):
        if subject in self.marks.keys():
            print(student_name, " already has a mark for ",
                  subject)
            user_input = input("Overwrite Y/N? ")
            if user_input == "Y" or "y":
                self.marks[subject] = mark
                return "Student's mark updated"
            else:
                return "Student's mark not updated"
        else:
            self.marks[subject] = mark
            return "Student's mark added"

class Students():
    def __init__(self, all_students):
        self.students = []
        for student, mark in all_students:
            self.add_student(student, mark)

    def add_student(self, student_name, marks = {}):
        if self.exists(student_name):
            return "Student already in database"
        else:
            self.students.append(Student(student_name, marks))
            return "Student added"

```

```

def print_report_cards(self, student_name = None):
    for student in self.students:
        if student_name == None or student.name == student_name:
            student.print_report_card()

def exists(self, student_name):
    for student in self.students:
        if student_name == student.name:
            return True
    return False

def add_mark(self, student_name, subject, mark):
    for student in self.students:
        if student_name == student.name:
            return student.add_mark(subject, mark)
    return "Student not found"

students = Students(student_data)
print(students.students)
while True:
    print("Welcome to the Raspberry Pi student database")
    print("What can I help you with?")
    print("Enter 1 to view all report cards")
    print("Enter 2 to view the report card for a student")
    print("Enter 3 to add a student")
    print("Enter 4 to add a mark to a student")
    print("Enter 5 to exit")
    user_choice = int(input("Choice: "))
    try:
        except ValueError:
            print("That's not a number I recognise")
            user_choice = 0

        if user_choice == 1:
            students.print_report_cards()
        elif user_choice == 2:
            enter_student = input("Which student? ")
            students.print_report_cards(enter_student)
        elif user_choice == 3:
            enter_student = input("Student name? ")
            print(students.add_student(enter_student))
        elif user_choice == 4:
            enter_student = input("Student name? ")
            enter_subject = input("Subject? ")
            num_error = True
            while num_error:
                num_error = False
                try:
                    enter_mark = int(input("Mark? "))
                except ValueError:

```

```

        print("I don't recognise that as a number")
        num_error = True
    print(students.add_mark(enter_student, enter_subject,
                           enter_mark))
    elif user_choice == 5:
        break
    else:
        print("Unknown choice")
    input("Press enter to continue")

print("Goodbye and thank you for using the Raspberry"
      ", \"Pi Student database\"")

```

3.7 使用模块获得附加特性

到目前为止，你已经见过 import 很多次了，但我们并没有解释它们究竟做了什么。事实上，非常简单，import 只是将 Python 代码从另外一个文件中转移到当前程序中。如果你创建一个名为 module_example.py 的文件，并写入下面这行：

```
print("Hello World")
```

将其保存在你的 home 目录（对于默认用户就是 /home/pi）。现在，在 IDLE3 的 Python 解释器中输入：

```
>>> import module_example
Hello World
```

当然，这是个毫无意义的模块。通常模块中包括可以使用的函数或对象。把 module_example 改成：

```
def hello():
    print ("Hello World")
```

你必须重启 IDLE 才能重新使用新代码。重启之后运行：

```
>>> import module_example
>>> module_example.hello()
```

第一行将 module_example 的所有函数和类导入你的工程中，在函数或类名前加上模块名作为前缀就可以使用它们了。有时，你只需要模块中的一部分，那么可以只导入其中的一部分：

```
>>> from module_example import hello
```

现在，只要输入下面这行就可以运行该函数：

```
>>> hello()
Hello, world!
```

注意，这里不需要加模块名前缀。因为我们已经将其导入当前名字空间。这样做的时候需要确保它和自己定义的其他函数或类没有冲突。为了方便使用，你也可以将某个模块的所有东西都导入当前名字空间：

```
>>> from module_example import *
```

这样做需要非常小心，以避免名字空间冲突。

使用模块而不是将所有东西放入一个文件有很多优点。它使得代码可以在不同工程间复用（关于代码复用，还记得我们是怎么说的么？）。它还可以避免将大工程保存在单个文件中带来的不便。你可以将不同的模块分给不同的组，方便团队一起工作。

后继章节中，我们将围绕几个特别的模块，为你的树莓派工程增加许多足够酷的特性。

3.8 小结

我们并没有完整覆盖 Python 的所有特性。这样做需要写一本更厚的书，同样，在后继章节中，我们也没有足够的篇幅去介绍各个方面。然而，我们介绍的这些已经足够让你开始起步，同时也希望这些东西足够让你理解大多数 Python 程序。如果你卡在某个地方了，请翻到本章，或者查阅 Python 文档（它才是真的包含了所有特性，就是不太容易理解）。

学完本章，你应该了解下列内容：

- 变量可以保存数据。
- 数据具有类型，如 int、float 或 bool。
- 列表、元组、字典或集合可以将数据聚集起来以方便存取。
- while 循环会重复运行直到条件为假 (False)。
- for 循环可以操作一组数据中的每个元素。
- 函数帮助你重用代码，可以避免重写它们。
- 类可以封装数据和方法使之更容易使用。
- Python 有上百个模块，在程序中导入它们可以增加附加特性。
- 你可以创建自己的模块。这样可以帮助把程序分割成多个方便管理的文件。

3.9 习题答案

练习 1

变量包括：

- `prompt_text` holds "Enter a number: " which is a string.
- `user_in` holds whatever the user types and is a string.
- `user_num` holds the number version of whatever the user types, converted into an int.
- `i` holds the numbers 1 to 9 as int.
- `even` is a bool that holds True or False (depending on whether the user's number is even).

练习 2

```
import random
```

```
secret = int(random.uniform(0,10))
print("I'm thinking of a number between zero and ten.")
print("Can you guess what it is?")
guess = 11

while guess != secret:
    try:
        guess = int(input("Take a guess: "))
    except ValueError:
        print("A number! Guess a number!")
print("Well done!")
```

练习 3

从技术上来说是可行的。但只是无谓地增加了代码复杂性。因为你无法改变它的值，必须每次创建新的元组而不能够简单地增加或修改当前值。

*Chapter 4***第 4 章****图形编程**

第 3 章中，我们介绍了很多关于如何加工、处理数据的方法，毕竟操纵数据是计算机的基本工作。你已经见过如何构建一个简单的字符驱动菜单来帮助控制程序。然而，这种风格在 20 世纪 80 年代已经不流行了。现在，虽然它在某些应用中还有使用，但是大多数人都喜欢用鼠标（或者触摸屏）。

你可能见过这三种不同类型的图形工具箱 Tk、GTK 和 Qt。Tk 是古典样式的库，现在还有应用，但缺乏现代特性。GTK 是一个很流行的工具箱，其中 LXDE（Raspbian 默认桌面[⊖]）就使用了 GTK。Qt（有时读作 cute）由诺基亚原创，当时主要是为了它们那个时运不佳的智能手机。后来被诺基亚卖给 Digia 公司，由其继续开发。GTK 和 Qt 都可以免费使用，说实话，在这两个中间没什么好选择的。本章选择使用 Qt，是因为它有较好的可移植性和较好的技术支持。

开始使用 `pyside` 模块之前需要先安装它。在 LXTerminal 中，输入：

```
sudo apt-get install python3-pyside
```

整个过程可能需要一点时间，可以先去喝杯茶。

4.1 图形用户界面（GUI）编程

贯穿本书，你将发现如果使用正确的模块来完成那些繁重工作，在 Python 里可以非

[⊖] 在本书翻译时，有消息称由于后向兼容性，Raspbian 将放弃 GTK 转投 Qt。——译者注

常轻易完成很多事情。图形用户界面（GUI）并不难。这里有各种各样有用的小工具，你需要做的就是选择几个需要的，并将其添加到自己的程序中。

你也可以使用继承。第3章中，我们介绍了类，你可以创建一个继承了超类所有特性的新类。这里，你将看到如何快速使用该方法基于原先的类创建一个新类。

让我们直接看下例子。第2章中，你见过turtle模块，也知道如何设置它使其等待按键按下。这比简单的文字输入要好，但也好不了太多。因此这里的第一个例子中，你将看到如何创建一个简单的GUI来控制turtle。让我们从下面的代码开始（同样的，可以手工输入或从网站上找到文件chapter4-turtle-start.py）：

```
import turtle
import sys
from PySide.QtCore import *
from PySide.QtGui import *

class TurtleControl(QWidget):
    def __init__(self, turtle):
        super(TurtleControl, self).__init__()
        self.turtle = turtle

        self.left_btn = QPushButton("Left", self)
        self.right_btn = QPushButton("Right", self)
        self.move_btn = QPushButton("Move", self)
        self.distance_spin = QSpinBox()

        self.controlsLayout = QGridLayout()
        self.controlsLayout.addWidget(self.left_btn, 0, 0)
        self.controlsLayout.addWidget(self.right_btn, 0, 1)
        self.controlsLayout.addWidget(self.distance_spin, 1, 0)
        self.controlsLayout.addWidget(self.move_btn, 1, 1)
        self.setLayout(self.controlsLayout)

        self.distance_spin.setRange(0, 100)
        self.distance_spin.setSingleStep(5)
        self.distance_spin.setValue(20)

#set up turtle
window = turtle.Screen()
babbage = turtle.Turtle()

# Create a Qt application
app = QApplication(sys.argv)
control_window = TurtleControl(babbage)
control_window.show()

# Enter Qt application main loop
app.exec_()
sys.exit()
```

现在就可以运行它，但上面的按钮还没有动作（等会儿再来添加）。首先让我们来看看这里都做了些什么。主要代码集中在 `TurtleControl` 类中，它继承自 `QWidget`（大多数 Qt 类以字母 Q 开头）。从 `QWidget` 继承后，你得到了所需的全部基本功能。现在需要做的就是将其从一个通用组件变得满足适合本程序需要。只需要告诉它，你需要哪些组件就可以了。

这里有三个按钮和一个整数旋转框（`spinbox`，允许输入数字，并可以增大、减小该数字——看看运行后的程序就知道 `spinbox` 如何工作了）。

除了用户看到的这些组件，你还要添加一个布局（`layout`）。Qt 有几种不同的布局类型（后面将会遇到另外一种），本程序使用 `QGridLayout`。像本例这样的简单控制面板程序非常适合使用网格布局（grid layout）。它的基本原理是将窗口分割成网格，由你来告诉 Qt 希望将组件放入哪个网格中。如果用户改变了窗口大小，Qt 将动态调整网格充满整个空间，并保持原有组件在网格中的位置不变。

想要在窗口中显示任何部件，都需要将它们加入布局（`layout`）。例如：

```
self.controlsLayout.addWidget(self.right_btn, 0, 1)
```

其中 0 和 1 分别表示从窗口左上角开始的水平和垂直坐标（即将平面直角坐标系上下颠倒一下）。这里，按钮位于窗口最上方一行，从左边数第二列。

当所有条目都添加到布局之后，你要用下面这行代码告诉窗口可以使用该布局。

```
self.setLayout(self.controlsLayout)
```

你还可以通过修改窗口组件的一些设置而改变它们的行为。本例中，可以使用以下方法调整整数旋转框（`spinbox`）：

```
self.distance_spin.setRange(0, 100)
self.distance_spin.setSingleStep(5)
self.distance_spin.setValue(20)
```

分别设置了最大最小值，以及每次单击改变的步长和初始值。

希望你能认出了刚开始的 `turtle` 代码。最后五行用来创建控制窗口并运行它。

4.2 添加控制

这些代码只是创建了一个漂亮的控制窗口，实际上并没有做任何事情。下一步要告诉

Qt 你要如何控制它。通过为事件链接一个动作就可以做到这一点。这里的事件指的是点击按键，动作指的是当事件发生时你希望运行的方法。

为了做到这点，请把下面代码添加到 `TurtleControl` 类的 `__init__` 方法末尾：

```
self.move_btn.clicked.connect(self.move_turtle)
self.right_btn.clicked.connect(self.turn_turtle_right)
self.left_btn.clicked.connect(self.turn_turtle_left)

def turn_turtle_left(self):
    self.turtle.left(45)

def turn_turtle_right(self):
    self.turtle.right(45)

def move_turtle(self):
    self.turtle.forward(self.distance_spin.value())
```



你可能会注意到在每次链接方法时，不像平时调用方法一样，作为参数的方法后面没有跟小括号。就像这样：

```
self.move_btn.clicked.connect(self.move_turtle)
```

而不是这样：

```
self.move_btn.clicked.connect(self.move_turtle())
```

这是因为在方法后面加上括号，就是告诉 Python 运行该方法并运行结果作为参数。就像：

```
def move_turtle(self):
    self.turtle.forward(self.distance_spin.value())
```

然而，如果你不在方法后面加括号，就是告诉 Python 将该方法本身作为参数。这正是你在这里要做的事情，如此，Qt 才知道在事件发生时去运行什么。

在本书网站上可以找到完整版的代码，文件名是 `chapter4-turtle.py`。图 4-1 给出了程序运行时的截图。

基本上 Pyside 和 Qt 就这么多内容，不会再出现更复杂的问题。但其含有大量的组件，我们不可能将它们全部演示一遍。在下一个例子中，为使你熟悉该工具箱，我们会将其尽可能多地展示出来，这样以后需要时就可以使用其他组件了。

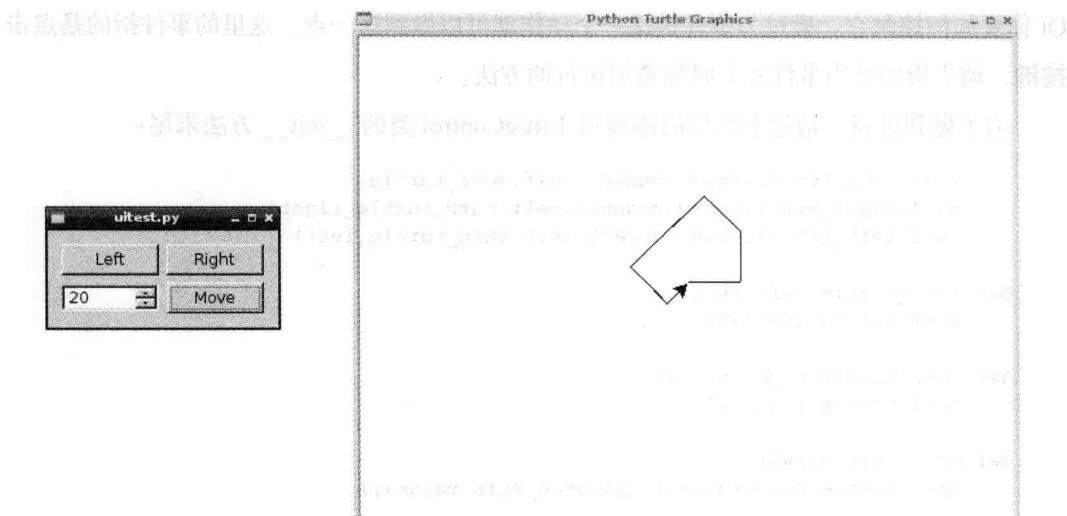


图 4-1 Turtle 的鼠标加强版接口

知识测试

练习 1

扩展 turtle 控制程序，当移动 turtle 时可以改变它的颜色。这里给出了一些提示，如果将设置 turtle 的代码改为：

```
#set up turtle
window = turtle.Screen()
babbage = turtle.Turtle()
window.colormode(255)
```

你就可以把 turtle 的颜色设为从 1 至 255 的红色、绿色、蓝色。如：

```
turtle.color(100,0,0)
```

另一个有用的东西是 QLabels。它可以给窗体增加小段文字描述。可以这样做：

```
self.red_label = QLabel("Red", self)
```

可以用它给整数旋转框加标签（或者其他任何标签）。

4.3 创建 Web 浏览器

前面的例子中可以看出我们可以简单地将组件组合起来创建一个界面。在下面这个例

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`



子中，我们将使用 Qt 组件创造一个自己的浏览器（其实也是由一系列组件组合出来的）。你将会发现，其实不用真正编程实现，只需要将不同的 Qt 部分组合起来就可以了。

首先你需要为浏览器创建一个窗体。上个例子中，你自己创建一个组件，Qt 为它生成了一个窗体。对于简单的工具这样子就足够了。但这次要建立一个更强大的应用，就需要单独创建一个窗体并在它上面添加所有需要的东西。使用 QMainWindow，可以添加很多东西，比如说菜单。当然，在添加菜单前需要先添加些其他东西。

对于任何浏览器来说，最重要的就是显示网页。关于网页中包含了什么东西，第 7 章将会详细介绍。对于本章来说，你只需要知道，QWebView 可以包办这一切就好了。

上个例子中，你使用了网格布局。对于大多数控制面板，它都可以很好地工作。在这个应用中，你将会使用到盒子布局。区别于网格，它有两种不同的布局，QVBoxLayout（垂直布局）和 QHBoxLayout（水平布局），分别是水平盒子和垂直盒子。在盒子布局中添加条目时，它将会水平或垂直地挨着排列在一起。将这些布局以某种合适的方式相互嵌套在一起可以创建出复杂的布局。当然，这需要较长时间练习才能做到。一旦你熟悉之后，就会发现它相当强大。

我们的 Web 浏览器将具有典型的 Web 浏览器布局。即顶部有个控制条，用户浏览的网页将占据剩下的大部分窗体。为创建这种界面，你需要两种布局，一个控制条用的 QHBoxLayout，和一个用来装前面的一个控制条和 QWebView 组件的 QVBoxLayout。当改变窗体尺寸时，Qt 将自动调整布局，各个组件将能很好地利用空间。

希望随着创造浏览器的过程，你会更清楚这些东西，就让我们开始吧！下面的代码创建了窗体，并添加上了适当的组件（参见本书网站中文件 chapter4-Web-browser-begin.py）。

```
import sys
from PySide.QtCore import *
from PySide.QtGui import *
from PySide.QtWebKit import *

class Browser(QWidget):

    def __init__(self):
        super(Browser, self).__init__()

        self.webview = QWebView(self)
        self.webview.load("http://www.google.com")
        self.setGeometry(0, 0, 800, 600)

        self.menu_bar = QHBoxLayout()
        self.main_layout = QVBoxLayout()
        self.main_layout.addLayout(self.menu_bar)
```

```

self.main_layout.addWidget(self.webview)
self.setLayout(self.main_layout)

class BrowserWindow(QMainWindow):
    def __init__(self):
        super(BrowserWindow, self).__init__()
        self.widget = Browser()
        self.setCentralWidget(self.widget)

# Create a Qt application
app = QApplication(sys.argv)
window = BrowserWindow()
window.show()

# Enter Qt application main loop
app.exec_()
sys.exit()

```

这就是一个简单的 Web 浏览器所需要的全部代码。你可以运行一下，它将会打开 Google 的主页，你可以从这里开始浏览（见图 4-2）。你应该已经很熟悉这些代码。新的内容包括 QMainWindow（后面将会介绍如何控制它），QWebView（如你所见，这是个可以轻松添加网络浏览器的方法）和盒子布局。



图 4-2 网络浏览器基础

盒子布局已经全部建立起来，你需要做的就是给浏览器的 self.menu_bar 添加条目，它们将出现在屏幕顶端。

Web 浏览器的最基本控制部分就是后退和前进按钮。在这个任务中，我们可以像在上个例子中一样使用 QPushButton。将下面的黑体部分加入你的代码中的 Browser 类中：

```
class Browser(QWidget):
```

 ~~def __init__(self):~~
 ~~super(Browser, self).__init__()~~
 ~~self.webview = QWebView(self)~~
 ~~self.webview.load("http://www.google.com")~~
 ~~self.setGeometry(0, 0, 800, 600)~~
 ~~self.back_btn = QPushButton("<", self)~~
 ~~self.back_btn.clicked.connect(self.webview.back)~~
 ~~self.back_btn.setMaximumSize(20,20)~~
 ~~self.forward_btn = QPushButton(">", self)~~
 ~~self.forward_btn.clicked.connect(self.webview.forward)~~
 ~~self.forward_btn.setMaximumSize(20,20)~~
 ~~self.menu_bar = QHBoxLayout()~~
 ~~self.menu_bar.addWidget(self.back_btn)~~
 ~~self.menu_bar.addWidget(self.forward_btn)~~
 ~~self.main_layout = QVBoxLayout()~~
 ~~self.main_layout.addLayout(self.menu_bar)~~
 ~~self.main_layout.addWidget(self.webview)~~
 ~~self.setLayout(self.main_layout)~~

现在运行一下，你的浏览器将拥有历史功能，可以前进和后退。当然，QWebView 完成了所有繁重的工作，这里只需要将点击按钮和 QWebView 的前进和后退方法连接起来。



注意 前面提到过，但这里仍值得再次强调——编程时，可以从已有模块中获得的功能，不需要重新再实现一遍。花上一点时间学习一个模块可以在后面节省你很多时间。

和前面使用的网格分布不同，盒子分布中，Qt 为计算绘制组件的尺寸提供了更多自由。有时这很不错，但其他时候，你需要给它更多指导。在 Web 浏览器中，你希望按钮占用尽可能少的空间，把空余屏幕都让给网页。为了做到这一点，我们在添加的组件上调用 setMaximumSize() 方法。比如按钮，用来保证它们不会大于 20×20 。

拥有一个文本输入框是 Web 浏览器的另一个特性，用户可以输入想要访问的站点地址。几种不同的 Qt 组件都可以用作文本输入，最常见的就是 QTextEdit。它可以用来显示或编辑文本。实际上，除了处理文字，它还可以处理图像、表格、标题等其他类似的东西。

QPlainTextEdit 是另一个类似于 **QTextEdit** 的常见组件，只不过它只能用作处理简单文本，不能处理复杂文本。它们都有十分强大的选项，在你的编程生涯中有很大机会能用到。由于它们被设计来处理多行文本，用作地址栏有点浪费。对于单行的简单文本（如 URL 栏），**QLineEdit** 是最好的选择。

你还需要一个“转到”按钮，用来告诉浏览器去加载页面。按下列代码更新 **Browser** 类（更新部分以黑体显示）就可以做到这些。

```
class Browser(QWidget):
    def __init__(self):
        super(Browser, self).__init__()

        self.webview = QWebView(self)
        self.webview.load("http://www.google.com")
        self.setGeometry(0, 0, 800, 600)

        self.back_btn = QPushButton("<", self)
        self.back_btn.clicked.connect(self.webview.back)
        self.back_btn.setMaximumSize(20,20)

        self.forward_btn = QPushButton(">", self)
        self.forward_btn.clicked.connect(self.webview.forward)
        self.forward_btn.setMaximumSize(20,20)
        self.url_entry = QLineEdit(self)
        self.url_entry.setMinimumSize(200,20)
        self.url_entry.setMaximumSize(300,20)

        self.go_btn = QPushButton("Go", self)
        self.go_btn.clicked.connect(self.go_btn_clicked)
        self.go_btn.setMaximumSize(20,20)

        self.menu_bar = QHBoxLayout()
        self.menu_bar.addWidget(self.back_btn)
        self.menu_bar.addWidget(self.forward_btn)
        self.menu_bar.addWidget(self.url_entry)
        self.menu_bar.addWidget(self.go_btn)
        self.menu_bar.addStretch()
        self.main_layout = QVBoxLayout()
        self.main_layout.addLayout(self.menu_bar)
        self.main_layout.addWidget(self.webview)

        self.setLayout(self.main_layout)

    def go_btn_clicked(self):
        self.webview.load(self.url_entry.text())
```

这里有些新东西。一个是在 URL 栏中调用了 **setMinimumSize()** 方法。像 **setMaximumSize**

一样，它向 Qt 传递一些你希望怎么显示窗体的额外信息。另一个帮助 Qt 恰当地排列窗体的新方法是调用 `addStretch()`。它添加了一个虚假组件，仅仅用来填充剩余空间。本例中，它占据了菜单栏右边的所有空间，因此 Qt 把所有的控制组件放到左边。

现在可以运行下试试看。唯一需要注意的是你需要在 Web 地址前输入 `http://`。（从技术角度看是因为 URL 或全球资源定位器需要指定网络协议。例如，`http://yoursite.com/document` 和 `ftp://yoursite.com/document` 就指向不同地方。大多数现代浏览器都允许你省略网络协议并假定你使用的是 http。然而，当一个组件请求 URL 地址时，它仍然需要协议前缀。）

本例中，我们为浏览器添加了一个叫 `go_btn_clicked()` 的新方法。与将方法和事件链接起来相比，这样做会更强大。`self.url_entry.text()` 返回用户输入的文本，它允许你使用这个文本作为参数调用 `Webview` 的 `load` 方法。

此时，你已经拥有一个真正意义上的浏览器。虽然没有主流浏览器，如火狐、谷歌浏览器或 Midori 功能强大，但它并不缺少任何基本要素。我们下一个要添加的功能是书签选择器。选择它一半是因为这是个有用的特性，一半是因为它让我们有理由来炫耀另一个有用的 Qt 组件——组合框（`QComboBox`）。

组合框这个名字比起其他你已经十分熟悉的组件来显得有点古怪。这是一个带有下拉箭头的框，用户可以在下拉列表中选择。如果还不清楚，很快你就能看到了。

本书稍后会介绍几种在对象之间保存信息的方法。但简单起见，我们不让用户改变或添加书签。毕竟，本章只是针对用户界面，我们将紧紧围绕这个主题。

在 `Browser` 类中添加下面例子中的黑体部分（非黑体部分是为了便于你知道添加到哪里）：

```
self.go_btn = QPushButton("Go", self)
self.go_btn.clicked.connect(self.go_btn_clicked)
self.go_btn.setMaximumSize(20,20)

self.favourites = QComboBox(self)
self.favourites.addItems(["http://www.google.com",
                         "http://www.raspberrypi.org",
                         "http://docs.python.org/3/"])
self.favourites.activated.connect(self.favourite_selected)
self.favourites.setMinimumSize(200,20)
self.favourites.setMaximumSize(300,20)

self.menu_bar = QHBoxLayout()
self.menu_bar.addWidget(self.back_btn)
self.menu_bar.addWidget(self.forward_btn)
self.menu_bar.addWidget(self.url_entry)
```

```

    self.menu_bar.addWidget(self.go_btn)
    self.menu_bar.addStretch()
    self.menu_bar.addWidget(self.favourites)
    self.main_layout = QVBoxLayout()
    self.main_layout.setLayout(self.menu_bar)
    self.main_layout.addWidget(self.webview)

    self.setLayout(self.main_layout)

    def go_btn_clicked(self):
        self.webview.load(self.url_entry.text())

    def favourite_selected(self):
        self.webview.load(self.favourites.currentText())

```

十分简单，运行下代码，你就可以看到 QComboBox 执行结果。

像 URL 一样，我们也调用了 self.Webview.load。但这次使用的参数是当前从组合框中选择的文本。

还有两个控制项需要添加到菜单栏中，让我们一次把它们都做出来。第一个是搜索条，输入搜索内容，然后按下按钮就可以运行谷歌搜索。第二个是缩放条，允许用户放大、缩小页面。

使用下面黑体部分更新 Browser 类：

```

self.favourites = QComboBox(self)
self.favourites.addItem("http://www.google.com",
"http://www.raspberrypi.org",
"http://docs.python.org/3/")
self.favourites.activated.connect(self.favourite_selected)
self.favourites.setMinimumSize(200,20)
self.favourites.setMaximumSize(300,20)

self.search_box = QLineEdit(self)
self.search_box.setMinimumSize(200,20)
self.search_box.setMaximumSize(300,20)

self.search_btn = QPushButton("Search", self)
self.search_btn.clicked.connect(self.search_btn_clicked)
self.search_btn.setMaximumSize(50,20)

self.zoom_slider = QSlider(Qt.Orientation(1),self)
self.zoom_slider.setRange(2, 50)
self.zoom_slider.setValue(10)
self.zoom_slider.valueChanged.connect(self.zoom_changed)

self.zoom_label = QLabel("Zoom:")

self.webview.loadStarted.connect(self.page_loading)

```

```

self.menu_bar = QHBoxLayout()
self.menu_bar.addWidget(self.back_btn)
self.menu_bar.addWidget(self.forward_btn)
self.menu_bar.addWidget(self.url_entry)
self.menu_bar.addWidget(self.go_btn)
self.menu_bar.addStretch()
self.menu_bar.addWidget(self.favourites)
self.menu_bar.addStretch()
self.menu_bar.addWidget(self.search_box)
self.menu_bar.addWidget(self.search_btn)
self.menu_bar.addWidget(self.zoom_label)
self.menu_bar.addWidget(self.zoom_slider)
self.main_layout = QVBoxLayout()
self.main_layout.addLayout(self.menu_bar)
self.main_layout.addWidget(self.webview)

self.setLayout(self.main_layout)

def go_btn_clicked(self):
    self.webview.load(self.url_entry.text())

def favourite_selected(self):
    self.webview.load(self.favourites.currentText())

def zoom_changed(self):
    self.webview.setZoomFactor(self.zoom_slider.value()/10)

def search_btn_clicked(self):
    self.webview.load("https://www.google.com/search?q="
                     + self.search_box.text())

def page_loading(self):
    self.url_entry.setText(self.webview.url().toString())

```

虽然有两个控制项，却要用四个组件来完成。搜索框有一个 QLineEdit 和一个 QPushButton。类似于之前添加的 URL 控制项，它们协同工作。只不过它会在你输入的字符前面加上 https://www.google.com/search?q=。例如，如果搜索树莓派，它将打开 URL http://www.google.com/search?q=Raspberries，也就是让谷歌去搜索树莓派。这里用了 https:// 而不是 http://。字母 s 代表安全 (secure)，如果使用 https://，浏览器到网站的所有数据都会被加密。然而，并不是所有网站都支持 https。只要服务器支持，QWebView 可以使用两者中的任何一个。

缩放按钮是 QSlider，这是另外一个你可能熟悉的控制形式。它需要设置的地方会多一点，下面就是如何设置的例子：

```

self.zoom_slider.setRange(2, 50)
self.zoom_slider.setValue(10)

```

第一行设置了滑块的最大和最小值，第二行设置了初始值。

把 valueChanged 动作连接到 zoom_changed() 方法。这里又是连接到一个 QWebView 的方法，把复杂的工作交给它。zoom_changed() 唯一做的就是将滑块的值除以 10 以保证缩放更容易控制。

如果仔细观察，你会发现我们不止添加了两个额外的控制部分。还有这些行：

```
self.webview.loadStarted.connect(self.page_loading)
.
.
def page_loading(self):
    self.url_entry.setText(self.webview.url().toString())
```

它们保证 URL 框能够实时显示当前网页的地址。

4.4 添加窗口菜单

主浏览器布局已经完成，但在应用程序完工前还有很多需要添加。还记得开始时我们说过通过扩展 QMainWindow 可以添加菜单吗？现在我们就开始做。

窗体已经有一个菜单，你需要做的就是添加些东西进去。菜单项和组件类似，只不过我们通过 QAction 创建。

把 BrowserWindow 类改成下面的样子可以给文件菜单添加一项来关闭窗体：

```
class BrowserWindow(QMainWindow):
    def __init__(self):
        super(BrowserWindow, self).__init__()
        self.widget = Browser()
        self.setCentralWidget(self.widget)

        self.exitAction = QAction(QIcon('exit.png'), '&Exit', self)
        self.exitAction.setShortcut('Ctrl+Q')
        self.exitAction.setStatusTip('Exit application')
        self.exitAction.triggered.connect(self.close)

        self.menu = self.menuBar()
        self.fileMenu = self.menu.addMenu('&File')
        self.fileMenu.addAction(self.exitAction)
```

还有一个用来打开本地文件的菜单项要添加。不同于 Web 浏览器中的所有事务，它要打开一个新窗口。在新窗口中，用户可以浏览他们的文件，并选择一个想要打开的。此时，你可能觉得需要很多工作来创建一个新窗口，添加整个布局，并把所有需要的组件联合起来。然而，这里又可以让 Qt 来为我们完成繁重的工作。Qt 有一系列称为对话框的组件。它们就是用来完成通用功能的一些简单窗口，这可以大大减轻我们的工作。按下列方

式更新 BrowserWindow 类，就可以给 Web 浏览器加入打开文件对话框。

```

self.exitAction = QAction(QIcon('exit.png'), '&Exit', self)
self.exitAction.setShortcut('Ctrl+Q')
self.exitAction.setStatusTip('Exit application')
self.exitAction.triggered.connect(self.close)

self.openFile = QAction(QIcon('open.png'), 'Open', self)
self.openFile.setShortcut('Ctrl+O')
self.openFile.setStatusTip('Open new File')
self.openFile.triggered.connect(self.showDialog)

self.menu = self.menuBar()
self.fileMenu = self.menu.addMenu('&File')
self.fileMenu.addAction(self.openFile)
self.fileMenu.addAction(self.exitAction)

def showDialog(self):
    fname, _ = QFileDialog.getOpenFileName(self, 'Open file',
                                           '/home')
    self.widget.webview.load("file:/// " + fname)

```

本例中，我们不需要创建新的对象，只要调用 QFileDialog 的 getOpenFileName() 方法。这将会打开一个标题为“打开 /home 目录下的文件”（Open File in the directory /home）的新窗口（参见图 4-3）。一旦用户选择了想要的文件，它将会返回两个东西：文件名和过滤器。因为 WebView 使用 file:///（注意这里是三个斜线）打开本地文件，因此使用时需要在文

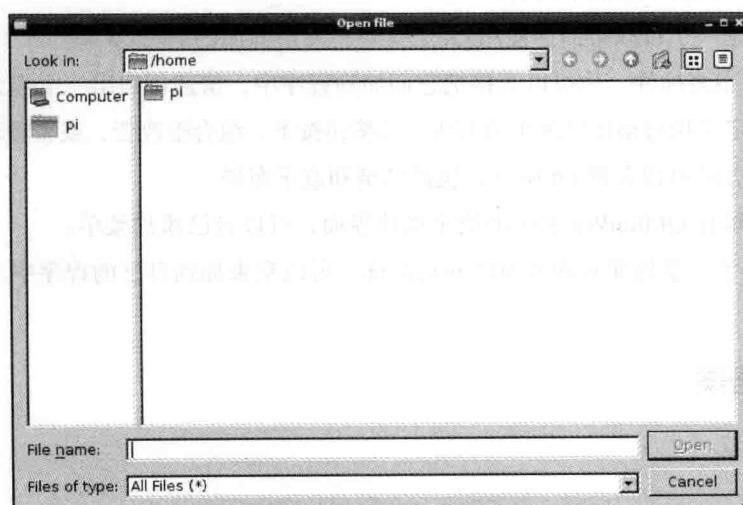


图 4-3 使用 QFileDialog 打开本地文件

WebView 使用 file:///（注意这里是三个斜线）打开本地文件，因此使用时需要在文

件名前加上这个前缀。

至此，Web 浏览器就结束了。如果你没有跟上去，可以从网站上找到完整代码，文件名是 chapter-Web-browser-complete.py。和所有应用程序一样，我们还可以给它加上很多东西。但我们所做的足以用来介绍 pyside 模块和 Qt 工具箱。这是个庞大的工具箱，因此出于需要，我们只能介绍些基础。一切都记录在 <http://srinkikom.github.io/pyside-docs/>。

知识测试

练习 2

你已经看到对话框可以在程序中快速添加功能。在这个练习中，要使用之前的 turtle 程序，在其中添加一个打开 QColorDialog 的按钮，用来设置 turtle 颜色。

这里给出一些提示。QColorDialog.getColor() 会返回类型为 QColor 的值。使用 variable_name.getColor().getRgb()[:3] 可以从 QColor 的变量中获得 RGB 的值。在末尾需要使用 [:3]，因为它的返回值是一个包含 4 个值的元组（最后一个透明度，这里用不到）。

4.5 小结

学完本章，应该了解以下内容：

- Pyside 是一个可以帮助你使用 Qt 工具包写图形界面的 Python 库。
- Qt 含有很多插件，你可以直接把它们加到程序中，快速创建出一个强大的界面。
- 你可以定义执行动作时调用的方法，如按钮按下，组合框改变，或者滚动条移动等。
- 有多种方法可以布置 Qt 窗口，包括网格和盒子布局。
- 使用继承自 QMainWindow 的类来构建界面，可以自己添加菜单。
- Qt 包含了一系列拥有现成窗口的对话框，可以拿来加到自己的程序中。

4.6 习题答案

练习 1

```
import turtle
import sys
from PySide.QtCore import *
from PySide.QtGui import *
```

```

class TurtleControl(QWidget):
    def __init__(self, turtle):
        super(TurtleControl, self).__init__()
        self.turtle = turtle
        self.left_btn = QPushButton("Left", self)
        self.right_btn = QPushButton("Right", self)
        self.move_btn = QPushButton("Move", self)
        self.distance_spin = QSpinBox()
        self.red_spin = QSpinBox()
        self.green_spin = QSpinBox()
        self.blue_spin = QSpinBox()
        self.red_label = QLabel("Red", self)
        self.green_label = QLabel("Green", self)
        self.blue_label = QLabel("Blue", self)
        self.colour_btn = QPushButton("Colour", self)
        self.controlsLayout = QGridLayout()
        self.controlsLayout.addWidget(self.left_btn, 0, 0)
        self.controlsLayout.addWidget(self.right_btn, 0, 1)
        self.controlsLayout.addWidget(self.distance_spin, 1, 0)
        self.controlsLayout.addWidget(self.move_btn, 1, 1)
        self.controlsLayout.addWidget(self.red_spin, 2, 1)
        self.controlsLayout.addWidget(self.green_spin, 3, 1)
        self.controlsLayout.addWidget(self.blue_spin, 4, 1)
        self.controlsLayout.addWidget(self.red_label, 2, 0)
        self.controlsLayout.addWidget(self.green_label, 3, 0)
        self.controlsLayout.addWidget(self.blue_label, 4, 0)
        self.controlsLayout.addWidget(self.colour_btn, 5, 0)
        self.setLayout(self.controlsLayout)

        self.distance_spin.setRange(0, 100)
        self.distance_spin.setSingleStep(5)
        self.distance_spin.setValue(20)

        for spinner in [self.red_spin, self.green_spin,
                        self.blue_spin]:
            spinner.setRange(0, 255)
            spinner.setSingleStep(5)
            spinner.setValue(150)

        self.move_btn.clicked.connect(self.move_turtle)
        self.right_btn.clicked.connect(self.turn_turtle_right)
        self.left_btn.clicked.connect(self.turn_turtle_left)
        self.colour_btn.clicked.connect(self.colour_turtle)

    def turn_turtle_left(self):
        self.turtle.left(45)

    def turn_turtle_right(self):
        self.turtle.right(45)

```

```
def move_turtle(self):
    self.turtle.forward(self.distance_spin.value())

def colour_turtle(self):
    self.turtle.color(self.red_spin.value(),
                      self.green_spin.value(),
                      self.blue_spin.value())

#set up turtle
window = turtle.Screen()
babbage = turtle.Turtle()
window.colormode(255)

# Create a Qt application
app = QApplication(sys.argv)
control_window = TurtleControl(babbage)
control_window.show()

# Enter Qt application main loop
app.exec_()
sys.exit()
```

练习 2

需要将下列函数连接到点击按钮动作上。

```
def colour_turtle(self):
    self.colour = QColorDialog.getColor()
    self.turtle.color(self.colour.getRgb()[:3])
```

搭建游戏

在第 4 章中，你使用 GUI 工具箱构建了图形化软件。它使得在程序中添加按钮、文本框和各种小工具变得十分容易。但是，有另外一类图形化程序并不使用这些东西：游戏。我们仍然需要在屏幕上画一些东西，但不是复选框和菜单，我们需要火球、英雄、恶魔坑和各种梦幻图形。显然，PySide 不能胜任这个任务，但另一个模块 PyGame 可以帮我们完成这些工作。

Raspbian 内置了 PyGame，但只能用于 Python2。由于我们使用 Python3，我们需要安装一下。本例中，你需要从头编译这个模块，这是个学习如何编译的好机会。首先，你需要安装 PyGame 需要的所有包，所以，打开 LXTerminal，输入 apt-get（包管理器）来安装附属程序：

```
sudo apt-get update  
sudo apt-get install libsdl-dev libsdl-image1.2-dev \  
  libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev \  
  libportmidi-dev libavformat-dev libswscale-dev \  
  mercurial python3-dev
```

你可能会注意到，这些代码大部分都以 -dev 结尾。这些是开发文件。编译使用这些库的软件时要用到它们。

下一步就是获取 PyGame 的最新版本。由于我们使用 Python3，我们需要获取做新版本，因此我们直接从开发平台中下载：

```
hg clone https://bitbucket.org/pygame/pygame  
cd pygame
```

这两行命令将会把 PyGame 的当前版本下载到一个名叫 pygame 的目录中。完成之后，你就可以用下列命令给 Python3 编译和安装该模块：

```
python3 setup.py build
sudo python3 setup.py install
```

如果一切顺利，就可以在 Python3 中使用 PyGame 了。在 IDLE 中打开 Python Shell，输入下面命令可以验证它是否工作：

```
>>> import pygame
```

这里如果出现任何错误，就意味着哪里出错了，在继续阅读本章之前，你需要回过头来重复前面步骤。如果一切正常，可以尝试下列命令观察 PyGame 是如何工作的：

```
>>> pygame.init()
>>> window = pygame.display.set_mode((500, 500))
>>> screen = pygame.display.get_surface()
>>> rect = pygame.Rect(100, 99, 98, 97)
>>> pygame.draw.rect(screen, (100, 100, 100), rect, 0)
>>> pygame.display.update()
```

你会看到这些命令打开了一个新窗口并绘制了一个灰色矩形。第一行创建并运行 GyGame。第二行打开一个新窗口。参数 -(500, 500)- 是一个包含新窗口宽度和高度的元组。注意这里为什么要有两层圆括号？外层括号用来传递参数，里层括号表示元组。第三行获得一个可以在上面绘画的平面，并将其存储在变量 screen 中。

你将会用到两个主要的 PyGame 类：Rect 和 Sprite。第二个我们稍后再看。第一个，Rect，对于 PyGame 的工作方式来说非常重要。第四行就创建了这样一个矩形，它的左上角坐标位于 100,99，宽 98 像素，高 97 像素。这里你需要知道，PyGame 的坐标是从屏幕的左上角开始，相对于正常图形坐标，它是上下颠倒的。

矩形并不总是会显示到屏幕上（后面你将会看到它还有其他用途），在下一行代码中，我们将该矩形绘制在屏幕上。参数 (100, 100, 100) 表示红色、绿色和蓝色值（每一个取值范围都在 0 到 255 之间），0 表示线宽（0 表示填充，1 或者大于 1 表示线的具体宽度）。当输入这些代码时，如果你注意观察，会发现矩形并没有出现在屏幕上。这是因为要使该变化生效，你需要刷新屏幕，正如我们在最后一行所做的那样。

你可以尝试在屏幕上画出更多矩形，体会一下不同的参数是如何影响形状的。

5.1 构建游戏

这些都是 PyGame 最基础的部分。现在我们要构建一个简单的平台游戏。在这个系统，

你要控制一个角色跑动并跳跃障碍直到完成目标。为了增加难度，我们从屏幕上方掉落下火球，游戏角色需要躲开它们。角色如果从平台上掉下来，就会被末日之火终结。

开始编程时，我们通常不是首先开始写代码。而是根据自己对程序如何运行的理解制订一个计划。这有点像程序的骨架，我们稍后会使之更充实和具体。我们首先会设计出所有的类和方法，但不去具体实现它们。接下来我们开始编程，不断丰富这些设计，直到编程结束。对于这个游戏，我们给出下面的设计（你可以把它打出来，也可以简单地从网站上下载文件 chapter5-pygame-skell.py，然后再阅读本章过程中给它添加内容）：

```
import pygame
import sys
from pygame.locals import *
from random import randint

class Player(pygame.sprite.Sprite):
    '''The class that holds the main player, and controls
    how they jump. nb. The player doesn't move left or right,
    the world moves around them'''

    def __init__(self, start_x, start_y, width, height):
        pass

    def move_y(self):
        '''this calculates the y-axis movement for the player
        in the current speed'''
        pass

    def jump(self, speed):
        '''This sets the player to jump, but it only can if
        its feet are on the floor'''
        pass

class World():
    '''This will hold the platforms and the goal.
    nb. In this game, the world moves left and right rather
    than the player'''

    def __init__(self, level, block_size, colour_platform,
                 colour_goals):
        pass

    def move(self, dist):
        '''move the world dist pixels right (a negative dist
        means left)'''
        pass

    def collided_get_y(self, player_rect):
        '''get the y value of the platform the player is
        currently on'''
```

```

    currently on''' pass

def at_goal(self, player_rect):
    '''return True if the player is currently in contact
    with the goal. False otherwise''' pass

def update(self, screen):
    '''draw all the rectangles onto the screen''' pass

class Doom():
    '''this class holds all the things that can kill the player'''

    def __init__(self, fireball_num, pit_depth, colour):
        pass

    def move(self, dist):
        '''move everything right dist pixels (negative dist
        means left)''' pass

    def update(self, screen):
        '''move fireballs down, and draw everything on
        the screen''' pass

    def collided(self, player_rect):
        '''check if the player is currently in contact with
        any of the doom.
        nb. shrink the rectangle for the fireballs to
        make it fairer''' pass

class Fireball(pygame.sprite.Sprite):
    '''this class holds the fireballs that fall from the sky'''

    def __init__(self):
        pass

    def reset(self):
        '''re-generate the fireball a random distance along
        the screen and give them a random speed''' pass

    def move_x(self, dist):
        '''move the fireballs dist pixels to the right
        (negative dist means left)''' pass

    def move_y(self):

```

```

'''move the fireball the appropriate distance down
the screen
nb. fireballs don't accellerate with gravity, but
have a random speed. if the fireball has reached the
bottom of the screen, regenerate it'''
pass

def update(self, screen, colour):
    '''draw the fireball onto the screen'''
    pass

#options
#initialise pygame.mixer
#initialise pygame
#load level
#initialise variables
finished = False
#setup the background
while not finished:
    pass
    #blank screen
    #check events
    #check which keys are held
    #move the player with gravity
    #render the frame
    #update the display
    #check if the player is dead
    #check if the player has completed the level
    #set the speed

```

这里真正的代码不多，但已经足以我们了解这是怎么回事。这是一个真正 Python 程序，你可以运行它。它除了不停地循环运行不会做任何事情，按 Ctrl+C 组合键可以停止它。但是这给了你一个开工的基础。随着我们添加代码，我们可以确保它一直可以运行，这样我们就可以不断查看它是如何工作的，以确保它能正确地运行。注意每个方法里的 pass 语句，该语句不做任何事情，但如果一个方法里没有任何代码，Python 将会报错，因此我们需要这一行以确保代码能够运行。

5.2 初始化 PyGame

现在我们添加一些东西来启动它。请在文件中对应的注释后面插入下列代码。

```
#options
screen_x = 600
screen_y = 400
game_name = "Awesome Raspberry Pi Platformer"
```

```

#initialise pygame
pygame.init()
window = pygame.display.set_mode((screen_x, screen_y))
pygame.display.set_caption(game_name)
screen = pygame.display.get_surface()
#initialise variables
clock = pygame.time.Clock()

#check events
for event in pygame.event.get():
    if event.type == QUIT:
        finished = True

#set the speed
clock.tick(20)

```

你应该能认出的 initialise pygame 后面的代码段是我们之前见到的。唯一的区别就是我们把屏幕尺寸换成两个变量。这样方便以后更改它们而不用记住哪些代码具体做了什么。所有关键选项都集中存储在全局变量中以允许你控制程序运行方式。

#check events 后面的部分用来等待用户按下关闭按钮关闭窗口后，退出循环。clock 的两行使用了 PyGame 的 timer 来设置合适的循环速度。由于每轮循环都对应游戏的一帧，我们需要确保它不会运行的太快（否则，在用户有机会操作之前所有的动作都结束了）。你只需要像在第 2 章的 turtle 游戏中那样，告诉 Python 休眠一个合适的时间即可，但有个小问题是我不知道剩下的循环体需要多长运行时间。如果你在一个较慢的计算机上运行，游戏的运行速度会和在较快的机器上不一样。clock.tick(fps) 可以解决这个问题。它会计算循环体运行所需时间，通过暂停一个合适的时间来确保每秒钟固定运行 fps 次循环。在上面代码中，最后一行会计算需要等待多久以确保每秒钟固定执行 20 次循环。

现在让我们从 Player 类开始构建每个类。请将下列代码添加到 Player 类中：

```

def __init__(self, start_x, start_y, width, height):
    pygame.sprite.Sprite.__init__(self)
    self.image = pygame.transform.scale(
        pygame.image.load(player_image), (width, height))
    self.rect = self.image.get_rect()
    self.rect.x = start_x
    self.rect.y = start_y
    self.speed_y = 0
    self.base = pygame.Rect(start_x, start_y + height, width, 2)

```

Player 的关键点就是它继承自 pygame.sprite.Sprite。你可以使用它来绘制图像，但首先你需要设置两个关键变量：self.image 和 self.rect。一旦设置好它们，继承自 pygame.sprite.Sprite 的那部分会把图像画出来。显然，self.image 是你希望传递给 sprite 的图像，self.rect

是 PyGame 要绘制图像的矩形区域。设置完成后，你可以像操作其他矩形区域一样移动，操纵它，PyGame 会相应地移动图像。后面你将看到，改变矩形区域的 x、y 属性可以移动它。

倒数第二个局部变量（speed_y）用来追踪用户控制角色上下跳跃的速度。最后一个局部变量（base）是一个表示角色双脚的非常短小的矩形。你将会使用它来检测角色是否站在平台上。

现在角色已经就绪，可以显示到屏幕上，但是首先你还需要一些代码。将下列代码添加到合适的区域：

```
#options
player_spawn_x = 50
player_spawn_y = 200
player_image = "lidia.png"

#initialise variables
player = Player(player_spawn_x, player_spawn_y, 20, 30)
player_plain = pygame.sprite.RenderPlain(player)

#render the frame
player_plain.draw(screen)

#update the display
pygame.display.update()
```

为了绘制精灵，你需要先给它提供一幅图像。如果你有艺术细胞，可以自己创作一个图像。这个网站提供了大量可以用于制作游戏的图像 <http://opengameart.org>。它们都是可以免费下载并在游戏中使用的。但其中有些是需要许可的，即如果你在游戏中使用了它们并将游戏分发给其他人，你需要向他们提供 Python 代码。如果愿意，他们可以修改代码，或者使用你的代码构建其他游戏。这个就是大家所说的开源（参见注释）。<http://opengameart.org> 网站上的每个文件都有链接说明它们所遵守的许可。值得注意的是除非你要将游戏分发给其他人，否则你不用担心这些东西。并不是所有的文件都可以用于 PyGame。我们推荐使用后缀为 .png 的图像。经过搜索，我们使用 <http://opengameart.org/sites/default/files/styles/medium/public/lidia.png> 作为我们游戏的主角，你也可以自己选择其他图像（当然你需要相应得更新 player_image 变量）。目前的代码只会从它运行的目录中查找 player_image 图像文件。

开 源

开源是指人们相互共享他们编写的程序。不仅限于可执行程序，还包括程序源代码。其他人可以以自认为合适的方法随意修改代码。例如，树莓派上的 Linux 操作系统

就是开源的。因此所有的工具如 Midori 网络浏览器，甚至 Python 自己也遵循同样的标准。大体上说，有两类不同的开源形式——一个是自己做的任何修改都需要共享，另一个却不需要。知识共享（creative commons）和开源类似，只是换成了艺术品，如图像、声音和著作。在你的代码里引入开源软件片段或者知识共享艺术品时，了解你需要承担的责任是很重要的。通常这些都会有链接详细介绍它们遵守的许可。对于知识共享，很容易阅读和理解。如果你需要遵照许可共享你的代码（或者你希望开源自己的工作），最简单的方法就是将其放在开源网站上，如 github.com。网站将免费为你共享它。

为了绘出精灵，你还需要将其加入 RenderPlain。它将会创建一个可以绘制到屏幕上的对象。这里，角色对象有它自己的 RenderPlain 对象，叫做 player_plain。

现在你可以运行代码。它将把精灵显示到坐标 player_spawn_x,player_spawn_y。从技术上说，它将会每秒重绘 20 次，但由于它一直在同一个地方，你无法感觉到。如图 5-1 所示。

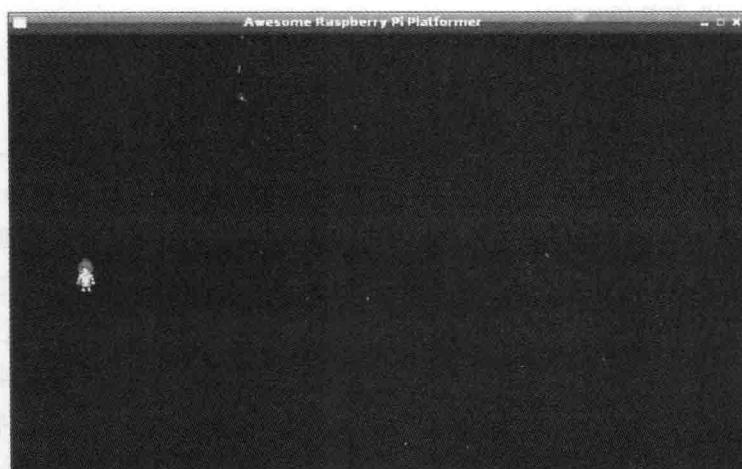


图 5-1 平台游戏简单开头

绘制出角色后，下一步就是让她动起来。这里只是一个简单的测试代码以保证动画可以正常工作。稍后你将会更新代码来使你的角色跳跃起来。将下面方法添加到 Player 类中：

```
def move_y(self):
    self.rect.y = self.rect.y + 1
```

它将把角色缓慢下移。为了完成其他效果，你需要在游戏循环中的合适位置添加下列代码：

```
#blank screen  
screen.fill((0, 0, 0))  
  
#move the player with gravity  
player.move_y()
```

现在运行代码，你将会看到角色向窗口下方移动，直到消失到底端。在构建一个供角色遨游的世界之前，我们只能做这些事情了。

5.3 为角色创建世界

我们希望可以用尽可能简单的方法来扩展这个游戏，让它看起来更酷。因此我们希望能简单地设计出新的关卡。为了达到要求，我们将每个关卡定义成一个字符串列表。每个字符串对应于屏幕上的一行，字符串中的每个字符代表该行上的一个块。短线 - 表示这里有个平台，字母 G 表示这里是目标（即角色为了完成关卡需要抵达的目标），其他表示空白。将下列代码加到 `#options` 之后来创建一个基本关卡：

如果你的输入正确，每行的宽度都应该一样。这是个用来测试的非常简单的关卡。最后两行分别设置了平台和目标的颜色。正如我们在本章开头看到的矩形区域一样，这里使用 RGB 值表示颜色。

现在添加下列代码到 World 类的 `init` 方法中，用来加载关卡：

```
def __init__(self, level, block_size,
            colour_platform,
            colour_goals):
    self.platforms = []
    self.goals = []
    self.posn_y = 0
    self.colour = colour_platform
    self.colour_goals = colour_goals
```

```

self.block_size = block_size

for line in level:
    self.posn_x = 0
    for block in line:
        if block == "-":
            self.platforms.append(pygame.Rect(
                self.posn_x, self.posn_y,
                block_size, block_size))
        if block == "G":
            self.goals.append(pygame.Rect(
                self.posn_x, self.posn_y,
                block_size, block_size))
        self.posn_x = self.posn_x + block_size
    self.posn_y = self.posn_y + block_size

```

代码非常简单，先循环每一行，再循环行内的每一个字符，根据发现的字符构建相应的矩形。在使用这些方块之前，你需要先为 World 类添加 update 方法，该方法可以将方块绘制到屏幕上。

```

def update(self, screen):
    '''draw all the rectangles onto the screen'''

    for block in self.platforms:
        pygame.draw.rect(screen, self.colour, block, 0)
    for block in self.goals:
        pygame.draw.rect(screen, self.colour_goals, block, 0)

```

现在你需要添加下列代码来创建对象并刷新它。和以前一样，只要找到注释，把它们放到正确的地方就可以了。

```
#initialise variables
world = World(level, 30, platform_colour, goal_colour)
```

```
#render the frame
```

```
world.update(screen)
```

运行程序，你会发现没有什么可以玩。游戏的世界已经绘出，但角色还是越过平台，不断向下缓慢移动，直到在屏幕上消失。

5.3.1 检测冲突

幸运的是，PyGame 的 Rect 的方法 collidrect() 能够非常简单地让两个游戏元素相互影响。它简单地不可思议，使用格式如下：

```
rect1.collidrect(rect2)
```

这里 rect1 和 rect2 是两个矩形。如果这两个矩形发生重叠，该语句将返回 True，否则

返回 False。你可以用它来检测游戏角色是否和游戏世界重叠，这样角色就不会穿过平台继续下落。从 World 类开始，添加如下代码：

```
def collided_get_y(self, player_rect):
    '''get the y value of the platform the player is
    currently on'''
    return_y = -1
    for block in self.platforms:
        if block.colliderect(player_rect):
            return_y = block.y - block.height + 1
    return return_y
```

这段代码不仅检查游戏角色是否接触到了游戏世界，还可以返回游戏角色碰到游戏世界中的矩形的上边的纵坐标，返回值 -1 表示没有两者没有接触。下一步我们来更新 Player 类，使其能在合适的时候运动或者静止。

```
def move_y(self):
    '''this calculates the y-axis movement for the player
    in the current speed'''
    collided_y = world.collided_get_y(self.base)
    if self.speed_y <= 0 or collided_y < 0:
        self.rect.y = self.rect.y + self.speed_y
        self.speed_y = self.speed_y + gravity
    if collided_y > 0 and self.speed_y > 0:
        self.rect.y = collided_y
    self.base.y = self.rect.y+self.rect.height
```

为了让游戏角色下落的自然点，你还需要添加一个东西：

```
#options
gravity = 1
```

现在运行程序，如图 5-2 所示，游戏角色会向下降落到平台上。

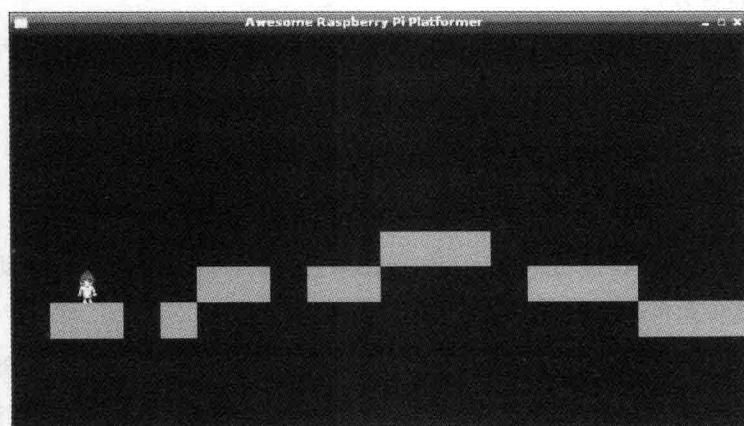


图 5-2 英雄现在可以站在我们为之创造的世界上

让我们来看下 move_y() 是如何使这一切变成现实的。代码处理了两种情况，每种情况都有自己的 if 块。第一种情况是游戏角色可以根据其当前速度和受到的重力自由地上下移动，即它没有碰到游戏世界的任何一个部分（也就是 collided_get_y 返回 -1）。我们也希望游戏角色可以从一个平台跳到更高的平台，因此如果游戏角色当前是向上移动的（也就是 if self.speed_y <= 0），我们像角色没有碰到游戏世界的情况一样处理它。如果发生这两种情况中的任意一种，我们就运行下列代码：

```
self.rect.y = self.rect.y + self.speed_y
self.speed_y = self.speed_y + gravity
```

这段代码根据当前速度将对象移动一个距离，然后根据重力更新当前速度（重力是一个加速度，它模仿了真实世界的物理现象）。

第二种情况（由第二个 if 语句处理）指的是角色开始接触到平台并且不再移动。在这种情况下，程序会调整角色坐标以使之刚好接触到平台。调整坐标是因为如果角色每帧移动多余一个像素，就有可能在接触平台后移动了若干像素才被检测到。随后你会发现当角色从高处下落时，有时候会出现角色向下越过平台一点然后又移动到平台边缘的情况，这也是模仿了人类从高处跳下时恢复站立的过程。

5.3.2 左右移动

现在游戏看上去像点样子了。有角色，有世界，但是角色还不能畅游世界。下一步要给它添加动作。有两种类型的动作。首先我们要让角色在接触到平台后能够跳跃，其次角色还要能够左右移动。

先为 Player 类添加 jump() 方法。因为你已经能够让角色落下，跳跃也一样简单。你要做的就是确保她在地面上，然后设置它向上移动并让其自行落下。

```
def jump(self, speed):
    if world.collided_get_y(self.base) > 0:
        self.speed_y = speed
```

现在开始让角色左右移动。实际上如果让角色保持不动，只是左右移动世界会很简单。这样可以确保角色不会因为移动而从屏幕上消失。

现在你要做的是遍历世界中的所有矩形（包括平台和目标），并按照给定的偏移量移动它们。我们可以通过更新矩形的 x 和 y 坐标来完成这点。Rect 类中有两种方法可供使用：move(x-distance, y-distance) 和 move_ip(x-distance, y-distance)。move() 返回一个移动了给

定距离的相同的新矩形，move_ip() 只是移动当前矩形的坐标（ip 表示就地（in place））。因为不需要每次移动时创建新矩形，我们选择 move_ip()。代码如下：

```
def move(self, dist):
    for block in self.platforms + self.goals:
        block.move_ip(dist, 0)
```

现在剩下的事情就是在主循环中添加代码，当按键按下时能够运行合适的方法。PyGame 为此提供了两种方法。第一种方法是监听按键事件，然后检测哪个键按下并根据检测结果执行相应动作。当你只需要关心按键动作时可以选择这种方法。第二种方法是使用 pygame.key.get_pressed() 返回一个按键事件队列。^①如果键按下队列中相应键的值为真，否则为假。如果你希望游戏玩家可以按住向下键不动使角色持续移动，显然是第二种方法更好。我们希望用户能够持续按下方向键，因此请在游戏循环中的适当位置加入下列代码：

```
#check which keys are held
key_state = pygame.key.get_pressed()
if key_state[K_LEFT]:
    world.move(2)
elif key_state[K_RIGHT]:
    world.move(-2)
if key_state[K_SPACE]:
    player.jump(jump_speed)
```

注意，K_LEFT、K_RIGHT 和 K_SPACE 都是我们从 pygame.locals 中导入的常量，此外还有从 K_a 到 K_z 的字母键。

下面添加一个选项表示跳跃速度（负数是因为 PyGame 的坐标系统是从左上角开始的）：

```
#options
jump_speed = -10
```

现在运行代码，你会将会看到一个所谓的平台游戏基础（见图 5-3）。你可以移动角色，跳跃缺口。然而这里还缺少两个重要的部分。首先，现在没有办法完成关卡，其次，没有任何东西可以让你停下来。

^① 这个队列是一个事件的列表——当事件发生时就会被加到队列中，如果被读过了，它们就会被删除掉。
——译者注

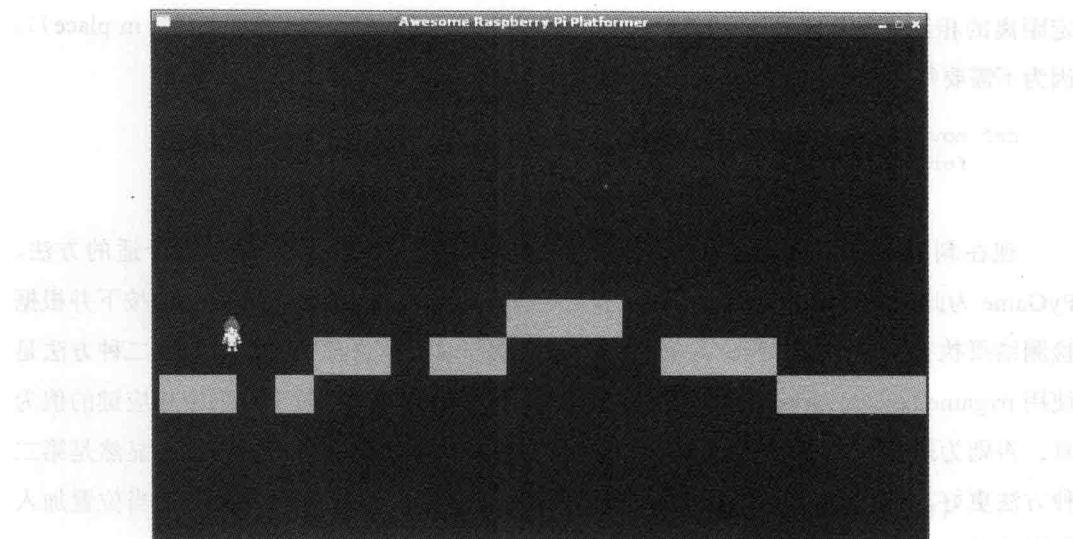


图 5-3 在树莓派上运行的一个不超过两百行的由 Python 写的平台游戏基础

5.3.3 达到目标

让我们先来完善第一个缺陷。一半是因为它比较简单，一半是因为完成之后你就有一个可供测试的游戏了。由于代码已经可以按照我们的意愿创建、显示并移动目标，我们需要做的就是检测角色是否到达目标。我们分两步来做。首先在 `World` 类中添加下列方法：

```
def at_goal(self, player_rect):
    for block in self.goals:
        if block.colliderect(player_rect):
            return True
    return False
```

该方法和我们之前创建的 `collide_get_y()` 方法一样，只是它的返回值是 `True` 和 `False`。之后，你需要在游戏循环中运行该方法，因此，加入下列代码：

```
#check if the player has completed the level
if world.at_goal(player.rect):
    print("Winner!")
    finished = True
```

现在运行程序，你将可以跑动、跳跃，直到抵达目标并完成关卡。你也可也能从平台上掉下来，消失在深渊里。但这还不够有挑战性。

5.3.4 制造挑战

为了给游戏增加剧情，我们加入一个 Doom 类，其中保存了可以杀死角色的道具。在本游戏中，玩家需要面临两个挑战。首先，在屏幕下方会有毁灭之火，如果角色落到上面就会被消灭。其次，从游戏玩法角度看也是更重要的，会有火球从天空落下。角色必须避开它们，因为这些火球挡在通往目标的路上。

首先，燃起毁灭之火。我们把它们绘制在屏幕底端的矩形区域。在 Doom 类中添加下列代码：

```
def __init__(self, fireball_num, pit_depth, colour):
    self.base = pygame.Rect(0, screen_y-pit_depth,
                           screen_x, pit_depth)
    self.colour = colour

def collided(self, player_rect):
    return self.base.colliderect(player_rect)

def update(self, screen):
    '''move fireballs down, and draw everything on the screen'''
    pygame.draw.rect(screen, self.colour, self.base, 0)
```

在选项、变量和游戏循环的适当位置添加下列代码：

```
#options
doom_colour = (255, 0, 0)
#initialise variables
doom = Doom(0, 10, doom_colour)
#render the frame
doom.update(screen)
#check if the player is dead
if doom.collided(player.rect):
    print("You Lose!")
    finished = True
```

这是一段很好的自注释代码。注意，你不需要移动毁灭之火的矩形区域，它将一直停留在屏幕下方。调用 update() 方法是为了移动火球。现在让我们看一下结果。

这里我们添加完整的 Fireball 类：

```
class Fireball(pygame.sprite.Sprite):
    '''this class holds the fireballs that fall from the sky'''
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.transform.scale(
            pygame.image.load(fireball_image),
            (fireball_size, fireball_size))
        self.rect = self.image.get_rect()
        self.reset()
```

```

def reset(self):
    self.y = 0
    self.speed_y = randint(fireball_low_speed,
                           fireball_high_speed)
    self.x = randint(0, screen_x)
    self.rect.topleft = self.x, self.y

def move_x(self, dist):
    self.rect.move_ip(dist, 0)
    if self.rect.x < -50 or self.rect.x > screen_x:
        self.reset()

def move_y(self):
    self.rect.move_ip(0, self.speed_y)
    if self.rect.y > screen_y:
        self.reset()

```

正如你所看到的，该类和 Player 类一样继承了 Sprite 类。move_x() 方法的工作方式和 World 类中相应的函数也一样。所不同的是这里只移动一个火球，因为每个火球都是一个独立对象。

为了保持挑战性，火球需要不断地从空中落下。有很多方法可以做到这一点，这里我们选择创建固定数量的火球，当火球在屏幕上消失后就立即复位它。reset() 方法将火球放到屏幕上方的随机位置并赋予火球一个随机的速度。

randint(a, b) 返回一个在 a 到 b 之间（包含 a 和 b）的随机整数。变量 screen_x 用来确保火球在屏幕上。两个全局变量（fireball_low_speed 和 fireball_high_speed）用于指定火球的最小速度与最大速度。它们的值表示每帧移动的像素数。这里你不需要冲突检测，因为这些都放在 Doom 类中做了。

现在来更新 Doom 类：

```

class Doom():
    '''this class holds all the things that can kill the player'''
    def __init__(self, fireball_num, pit_depth, colour):
        self.base = pygame.Rect(0, screen_y-pit_depth,
                               screen_x, pit_depth)
        self.colour = colour
        self.fireballs = []
        for i in range(0, fireball_num):
            self.fireballs.append(Fireball())
        self.fireball_plain = pygame.sprite.RenderPlain(
            self.fireballs)

    def move(self, dist):
        for fireball in self.fireballs:
            fireball.move_x(dist)

```

```

    def update(self, screen):
        for fireball in self.fireballs:
            fireball.move_y()
        self.fireball_plain.draw(screen)
        pygame.draw.rect(screen, self.colour, self.base, 0)

    def collided(self, player_rect):
        for fireball in self.fireballs:
            if fireball.rect.colliderect(player_rect):
                hit_box = fireball.rect.inflate(
                    -int(fireball_size/2),
                    -int(fireball_size/2))
                if hit_box.colliderect(player_rect):
                    return True
        return self.base.colliderect(player_rect)

```

如你所见，这里创建了一个火球列表，并将它们添加到 `fireball_plain` 中。和 `player_plain` 一样，这也是一个 `RenderPlain`，允许你把火球绘制到屏幕上。注意这里使用全局变量表示火球的大小和数目。改变它们可以急速地影响游戏性，并且在许多方面它都是改变难度的关键变量。

`collided()` 方法和我们之前见到的有点不同。它比较角色和火球体积一半的矩形是否有接触。这是因为无论角色还是火球都不是一个规则矩形，在两个对象还有距离的情况下这两个矩形可能会接触到。这会极大地挫伤游戏玩家的积极性。我们使用的并不是最好的方法，但却是朝着降低角色阵亡的方向努力。换句话说，角色有可能擦到火球边缘而侥幸逃脱，但如果碰撞检测返回 `True`，就意味着真的发生了碰撞。



实际上使用 PyGame 提供的 `pygame.sprite.collide_mask(sprite1, sprite2)` 可以精确地检测到精灵碰撞。但这需要消耗更多的计算资源，并且对于我们这个任务会显得过度杀伤精灵。

添加这两个类之后，游戏只需要添加下列代码就可以工作了：

```

#options
fireball_size = 30
fireball_number = 10
fireball_low_speed = 3
fireball_high_speed = 7
fireball_image = "flame.png"

```

在 `Doom` 的初始化中加入火球：

```
doom = Doom(fireball_number, 10, doom_colour)
```

你还需要在 `keypress` 小节加入几行代码使得火球会随着背景移动（只需要添加黑体部分）：

```
#check which keys are held
key_state = pygame.key.get_pressed()
if key_state[K_LEFT] :
    world.move(2)
    doom.move(2)
elif key_state[K_RIGHT] :
    world.move(-2)
    doom.move(-2)
```

我们仍然使用从 <http://opengameart.org> 中下载的精灵图像。这里我们使用了：<http://opengameart.org/sites/default/files/flame.png>。为了使其能够工作，文件需要下载并保存到你运行程序的那个目录下。或者，你也可以使用精灵的绝对路径。例如，如果你把所有东西都存在 `/home/pi/my_game/` 目录下，你可以将下面这行

```
fireball_image = "flame.png"
```

改为

```
fireball_image = "/home/pi/my_game/flame.png"
```

这样无论你在哪个目录下运行游戏，它都可以工作。现在保存程序并运行，游戏看起来应该和图 5-4 一样。

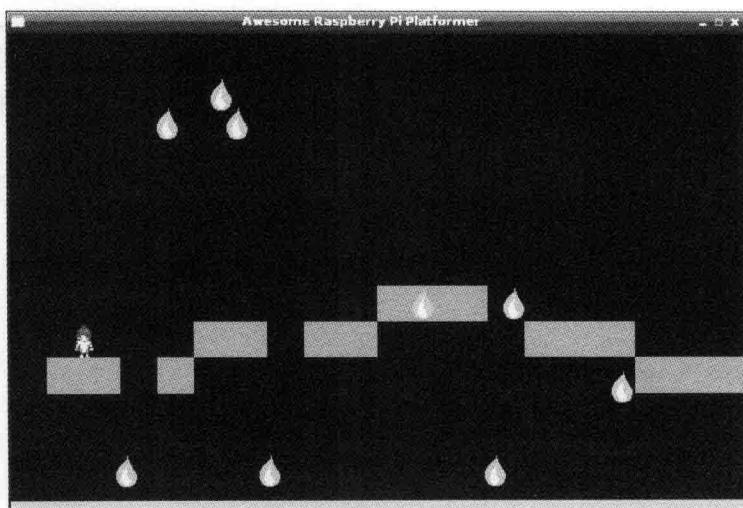


图 5-4 一个边缘略显粗糙但可以工作的平台游戏

5.4 在游戏中加入自己的风格

我们已经揭开游戏的运行机制。玩家需要穿越世界、躲避火球，并最终到达目标。虽然还有需要完善的地方，但基础已经做好。现在是时候添加些自己的风格了。毕竟，本章不是为了教你如何复制代码来创建游戏，而是为了让你能够构建自己的游戏。现在你对于如何定制游戏了解的已经够多了。从选项（option）部分改起是个不错的主意。

根据你的显示器，你可能想改变窗口大小。如果你觉得这个太简单，可以添加更多的火球，或者让火球变大。也许，你希望能跳得更高，跑得更快。所有的这些都很简单。实际上，从前面的章节中你应该学到如何在游戏开始时添加一个游戏菜单。在 `#initialise pygame` 之前，你可以做一个简单的基于文本的菜单，用来选择关卡难度。更难一点的，你可以……事实上，你只需要自己来做。如果你更有雄心，可以做一个图形化菜单替代文本菜单。

5.5 添加音乐

但愿你已经做完了自己的定制版游戏，如果没有完成也不用担心，等本章剩余部分结束后你可以继续进行。

现在需要添加点耀眼的东西。这部分不会影响游戏的运行机制，却可以增加娱乐性。首先添加声音效果，其次是添加背景。

开始添加音乐前，我们需要初始化合成器。它会将声音设备安装好以准备播放。这些都由下列代码完成：

```
#initialise pygame.mixer
pygame.mixer.pre_init(44100, -16, 8, 2048)
pygame.mixer.init()
```

虽然开始的时候我们只播放跳跃音效，但是这段允许我们同时播放 8 种声音。同样，我们访问 <http://opengameart.org>。这次文件来自 <http://opengameart.org/sites/default/files/qubodup-cfork-ccby3-jump.ogg>。你需要下载它或者其他音乐文件。我们将该文件添加到选项中：

```
#options
jump_sound = "qubodup-cfork-ccby3-jump.ogg"
```



注意 MP3 文件有时候也可以工作，但比较难对付。有可能导致 PyGame 游戏崩溃。因此可能还是尽量选择 OGG 文件。

下面我们需要修改 Player 类使其能够在适当时候播放音乐。在 `__init__()` 方法的末尾加入下列代码：

```
self.sound = pygame.mixer.Sound(jump_sound)
```

你还需要修改 `jump()` 方法：

```
def jump(self, speed):
    if world.collided_get_y(self.base) > 0:
        self.speed_y = speed
        self.sound.play()
```

这就是你要给游戏添加的音效。你会发现可以很容易地给游戏添加更多音效。例如当抵达目标时播放一段音乐，或者角色挂掉时候播放一段音乐。你也可以播放背景音乐。然而，要注意你买的这些音乐都是有版权的。在自己的游戏中使用它们没有问题。但是如果要发布自己的游戏，就可能会带来麻烦。可以看看这个网站 <http://freemusicarchive.org> 和 <http://opengameart.org> 一样，该网站上有大量文件可供下载，你也可以将它们放到你的游戏中。其中许多文件都是得到许可的，如果你要发布自己的游戏，需要同时发布自己的源代码。

5.6 添加布景

我们为游戏添加的第二个细节是背景。当然，你需要一个背景图片。我们已经从网站 <http://opengameart.org/sites/default/files/background.zip> 中选取了文件 `background.png`。这个是很好的乡村景色，你也可以换成一些灰暗的图片来改变游戏氛围。为了加载文件你需要在选项中添加下列代码：

```
#options
background_image = "background.png"
```

使用图片前，你需要一个继承自 `Sprite` 类的新类来显示它（像 `Player` 和 `Fireball` 一样）。由于不需要操作矩形区域或者检测碰撞，你可以简单地为其加载一个图像。详见下面代码：

```
#set up the background
background = pygame.transform.scale(pygame.image.load(
    background_image), (screen_x, screen_y)).convert()
bg_1_x = -100
bg_2_x = screen_x - 100
```

第一行加载图片并将其拉伸到整个屏幕，然后运行 `counert()` 方法。该方法将图片从 PNG 转换成 PyGame 的外观。这会让屏幕显示更快，尤其是对于这个尺寸的图像很重要。

这里你也可能使用了其他图像，但你会失去图像的透明部分，使其变成完全的矩形。第二行和第三行创建两个变量来存储图像的 x 坐标。这里之所以有两个值是因为我们要绘制这个图像两次，用来创建一个连续循环的背景，这样玩家就不会移动到图像末尾了。

为了移动背景，需要在游戏循环中的适当位置添加下列代码：

```
#check which keys are held
key_state = pygame.key.get_pressed()
if key_state[K_LEFT]:
    world.move(2)
    doom.move(2)
bg_1_x = bg_1_x + 1
bg_2_x = bg_2_x + 1
if bg_1_x < screen_x:
    bg_1_x = -screen_x
if bg_2_x < screen_x:
    bg_2_x = -screen_x
elif key_state[K_RIGHT]:
    world.move(-2)
    doom.move(-2)
bg_1_x = bg_1_x - 1
bg_2_x = bg_2_x - 1
if bg_1_x > -screen_x:
    bg_1_x = screen_x
if bg_2_x > -screen_x:
    bg_2_x = screen_x
```

这里还有一些故事。首先，你是否注意到背景比 world 或者 doom 移动地慢？这个称为视差滚动（parallax scrolling）。它创建了多层背景使其以不同的速度移动，形成立体的运动效果。就像你坐在运动的车上看窗外，离你近的物体比离你远的物体看起来移动的速度快。这不是 3D 图像，而是帮助创造一种层次错落的错觉。如果你想做得更好一点，可以添加更多的背景层次。比如，可以画一些树，让它们比平台移动地慢一点，然后添加一些山，使其比树移动地慢点，最后添加个太阳，让它移动地最慢。和声音一样，你可以随便添加自己喜欢的东西进去。

第二个值得注意的事情是代码中用于移动背景的 if 语句。当图像的一端移动到屏幕边缘时，程序将其另一边重新移到屏幕另一侧。这样就创造了在两个背景图像中不断循环的无限滚动背景。剩下的事情就是将图像绘制到屏幕上：

```
#render the frame
screen.blit(background, (bg_1_x, 0))
screen.blit(background, (bg_2_x, 0))
```

这两行必须紧跟着 #render the frame，否则它们会遮住其他要显示的部分。图 5-5 给出了游戏的最终结果。

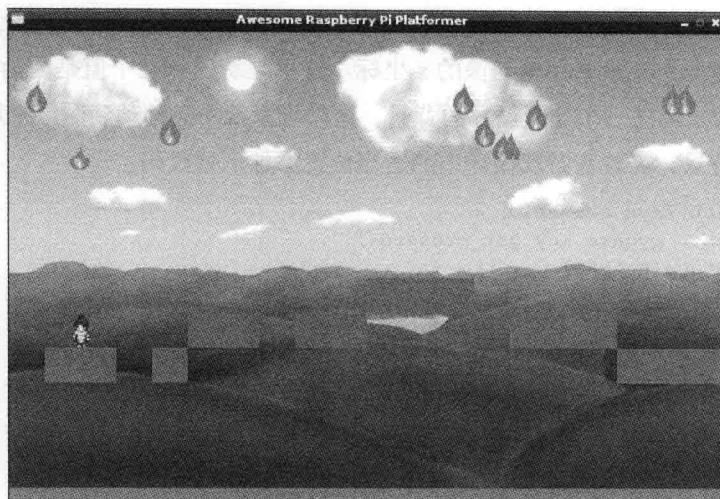


图 5-5 添加了所有元素的游戏

添加点睛之笔

游戏的实质部分已经全部完成。然而，我们还可以做一些事情让人们能够更方便地使用他。之前，我们用到了关卡（level），它被硬编码到程序中。如果我们可以将其放在文本文件中，允许用户选择加载哪个文件来选择关卡，然后程序加载相应文件，这样会比以前更好一点。由于关卡是文本形式，我们的工作就更简单了。

运行 `python3 chapter5-platformer.py`（或者你自己命名的文件）将运行程序内置的默认关卡。而运行 `python3 chapter5-platformer.py mylevel` 将运行加载了指定文件 `mylevel` 的游戏。为了实现这一点，我们需要使用 `sys.argv`。它属于 `sys` 模块，它是一个包含了所有参数的列表，并能将其传递给 Python。`sys.argv[0]` 是我们运行的脚本名字，因此 `sys.argv[1]` 就是后面跟的文件名（如果存在）。我们需要做的就是把下面代码加到程序中：

```
#load level
if len(sys.argv) > 1:
    with open(sys.argv[1]) as f:
        level = f.readlines()
```

读取文件和我们之前读取数组一样。即 - 表示平台，G 表示目标，多行一起构成不同的游戏关卡。



注意 如果你没有跟着一步步做下去，可以从网站上下载完整游戏：`chapter5-platformer.py`。

我们强烈建议你随着本章一起完成该游戏，它可以帮助你更好地理解这些内容。

5.7 让游戏更上一层楼

我们可以继续为游戏添加更多内容。然而，我们的指导将到此为止。不是因为游戏完成了，而是由于你现在应该学习到足够的知识，可以独立完成它了。我们不会命令你要把游戏做成什么样子——它是你的游戏，想加什么都可以。这里我们给出一些改进的想法：

- 如果你还没有尝试过修改过游戏选项，试着修改一下。
- 为游戏创建新关卡会让你感觉到这是你自己的游戏。
- 游戏中使用的图片仅作为推荐，自己上网找一下，尝试替换一下它们。
- 为当前的矩形对象添加精灵，如平台和末日之火。
- 将游戏关卡融入游戏世界。每个世界都有一个不同的主题，为每个主题选择一个不同的图片、音乐。
- 为角色添加道具。可以是统计成绩的金币，或者增强道具使得角色可以跑得更快或者跳得更高。
- 添加更多可以消灭玩家的怪物。如某些持续向右移动的东西，使得玩家必须保持运动以避开它们，或者某些冲向玩家的怪物，玩家必须踩上面才能消灭它们，或者从下面射上去的东西。
- 每一关卡结束时统计成绩。成绩可以是完成时间，玩家收集到的目标（金币），或者其他一些东西。
- 让精灵动起来。<http://opengameart.org> 上有许多动作的连续图像，通过不断地在一組图像中切换，可以使精灵动起来。
- 如果你一直按下方向键，精灵会加速运动而不是保持恒定的速度，也可以增加一个运行键，运行玩家加速游戏。

这仅仅是一些参考想法。我们并不想也无法列出你可以在游戏中添加的全部内容。因此开动脑筋想想看你还可以在游戏中实现什么内容。如果看起来不错，你可以上传到树莓派商店让其他人试一下。最后请不要忘了你在游戏中使用的任何图片和声音的许可。

5.8 逼真的游戏物理

用来创建简单游戏 PyGame 是一个很好的模块。正如你所见到的，使用它可以很简单地在屏幕上绘制对象并移动它们。但有时你需要更多能量。在前面的例子中，重力影响了角色的下落，但其他的物理现象并没有考虑进去。如果需要对象之间以更真实的方式交

互，如相互弹跳，你需要使用物理库。

PyMunk 就是这样一种库，它可以帮助你创建更真实的游戏。使用它，你可以创建一个空间并加入对象。PyMunk 将会解决它们之间的交互。

可以从 <http://code.google.com/p/pymunk/downloads/list> 上下载 PyMunk（你需要源程序）。如下所示，下载完成后，解压缩后进入新目录（使用 LXTerminal 而不是 Python 运行下列命令）：

```
unzip pymunk-4.0.0.zip
cd pymunk-4.0.0
```

不幸的是，build 文件中有一些小错误使得其不能在树莓派上正确构建。为了安装它，你需要使用文本编辑器（如 LeafPad）打开 setup.py，找到下面这行：

```
elif arch == 32 and platform.system() == 'Linux':
    compiler_preargs += ['-m32', '-O3']
```

请确保这行包含的是 arch==32 而不是 64。在第二行中删除 '-m32' 使其变成这样：

```
compiler_preargs += ['-O3']
```

保存文件，现在你可以按照下列方法安装 PyMunk：

```
python3 setup.py build_chipmunk
python3 setup.py build
python3 setup.py install
```

同样，这些命令需要在 LXTerminal 中的 PyMunk 目录下执行。该过程可能需要一段时间，完成之后，你可以打开 Python 并输入下列命令来检测它是否工作：

```
>>> import pymunk
```

希望你没有遇到任何错误

下面的例子可以在网站上的 chapter5-pymunk.py 中找到。

```
import pygame, pymunk
from pygame.locals import *
from pygame.color import *
from pymunk import Vec2d
import math, sys, random

def to_pygame(position):
    return int(position.x), int(-position.y+screen_y)

def line_to_pygame(line):
    body = line.body
    point_1 = body.position + line.a.rotated(body.angle)
    point_2 = body.position + line.b.rotated(body.angle)
    return to_pygame(point_1), to_pygame(point_2)
```

```

#####options#####
screen_x = 600
screen_y = 400
num_balls = 10
pygame.init()
screen = pygame.display.set_mode((screen_x, screen_y))
clock = pygame.time.Clock()
running = True

space = pymunk.Space()
space.gravity = (0.0, -200.0)

#create the base segment
base = pymunk.Segment(pymunk.Body(), (0, 50), (screen_x, 0), 0)
base.elasticity = 0.90
space.add(base)

#create the spinner
spinner_points = [(0, 0), (100, -50), (-100, -50)]
spinner_body = pymunk.Body(100000, 100000)
spinner_body.position = 300, 200
spinner_shape = pymunk.Poly(spinner_body, spinner_points)
spinner_shape.elasticity = 0.5
spinner_joint_body = pymunk.Body()
spinner_joint_body.position = spinner_body.position
joint = pymunk.PinJoint(spinner_body, spinner_joint_body, (0, 0),
                        (0, 0))
space.add(joint, spinner_body, spinner_shape)

#create the balls
balls = []
for i in range(1, num_balls):
    ball_x = int(screen_x/2)
    radius = random.randint(7, 20)
    inertia = pymunk.moment_for_circle(radius, 0, radius, (0, 0))
    body = pymunk.Body(radius, inertia)
    body.position = ball_x, screen_y
    shape = pymunk.Circle(body, radius, (0, 0))
    shape.elasticity = 0.99
    space.add(body, shape)
    balls.append(shape)

while running:
    for event in pygame.event.get():
        if event.type == QUIT:
            running = False
    screen.fill((0, 0, 0))

#draw the ball
    for ball in balls:
        pygame.draw.circle(screen, (100, 100, 100), to_pygame(ball.
body.position), int(ball.radius), 0)

```

```

#draw the spinner
points = spinner_shape.get_vertices()
points.append(points[0])
pygame_points = []
for point in points:
    x,y = to_pygame(point)
    pygame_points.append((x, y))
color = THECOLORS["red"]
pygame.draw.lines(screen, color, False, pygame_points)

#draw the line
pygame.draw.lines(screen, THECOLORS["lightgray"], False,
                  line_to_pygame(base))

space.step(1.0/50.0)
pygame.display.flip()
clock.tick(50)

```

如你所见，这段代码使用 PyGame 绘制图形，使用 PyMunk 解决它们如何运动。

PyMunk 运行需要一个 space，下列两行代码演示了如何建立 space：

```

space = pymunk.Space()
space.gravity = (0.0, -200.0)

```

它们从 Space 类创建了一个新对象，并设置了它的重力。我们可以继续创建对象，并把它们加入 space。本例使用了圆环、线段和一组点构成的多边形。（注意，在树莓派上，PyMunk 有一个 bug，你只能在 space 中添加一个单节段。）关节接头可以将多个形状连接成一个整体，并将其一点固定在某个位置。由于重力原因，它可以沿中心点旋转却不会落下。

在主循环中，我们调用了 space.step(1.0/50.0)，告诉 PyMunk 以五十分之一秒的步进移动 space 中的所有对象。

这里有一个小困扰，PyGame 和 PyMunk 使用了不同的坐标系统。之前我们说过，PyGame 的 (0, 0) 点在左上角，而 PyMunk 的却在左下角。这意味着为了在正确的位置绘制对象，你需要重新计算坐标。这就是 `_pygame(position)` 要做的事情。

和定义不同对象的位置一样，在 PyMunk 中，你可以给它们定义不同的物理属性，如弹性和惯性。通过调整这些属性，你可以控制对象在你的世界中如何交互。

运行代码，你将看到 PyMunk 解决了如何移动的大部分难题，如处理碰撞和计算球如何从地板上弹开等其他问题。结果如图 5-6 所示。然而这里有个代价问题——它需要的计算量远远超过我们之前的游戏。有时树莓派可以处理简单的物理模拟，你需要注意节约使用，否则它们会运行地很慢。使用 raspi-config 工具，可以超频树莓派，可以帮助模拟程序更快运行。

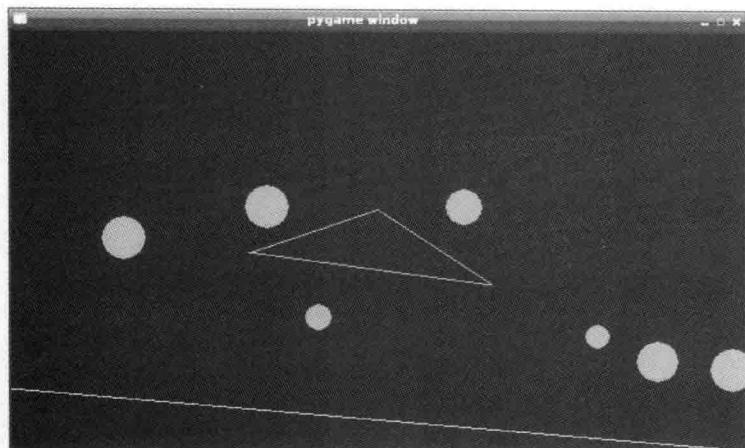


图 5-6 PyMunk 物理模拟引擎完成了繁重的真实世界交互模拟



提示 树莓派默认运行频率是 700MHz。也就意味着它可以每秒钟处理 7000,000,000 条指令。对于需要大量计算的任务，你可以使用 `raspi-config` 工具超频 CPU。它 can 达到 1000MHz（或者 1GHz）。然而，并不是所有的树莓派超频后都可以正常工作，有些可能会不稳定。如果你的树莓派启动时候停住了，试着减少超频。

我们仅仅接触了些基础，希望它们足够为你启蒙。在你之前下载的 PyMunk ZIP 文件中还有一些例子。并不是所有例子都可以在 Python3 下面运行，但它们可以给你一个尝试的机会。这里还有些过时的但却很不错的文档 <http://pymunk.googlecode.com/svn/tags/pymunk-2.0.0/docs/api/index.html>。

5.9 小结

读完本章，你需要了解以下内容：

- PyGame 是一个帮助你在 PyGhon 中创建游戏的模块。
- 继承自 Pygame.sprite.Sprite 的类可以在屏幕上绘制图像。
- 精灵被绘制在类的 `self.rect` 的矩形内。
- 你可以使用这些矩形来检测对象间的冲突。
- 视差滚动可用于使 2D 图形形成立体运动效果。
- PyGame 也可以处理音频。
- 为了更真实地模拟物理世界，可以使用物理引擎如 PyMunk，但它会使运行速度变慢。

使用 OpenGL 创建图形

没有人否认 3D 图形看起来很酷。它们能创造出比 2D 图形更多的深度感，可以让程序员创建出更丰富的世界。然而，随之而来的是费用。首先，相比 2D 图形，它需要更多的计算资源，其次，它的编程显然更复杂。

普通 PC 上你通常会发现显卡（有时也会称为图形处理单元，或 GPU）。它提供了额外的计算资源使得计算机可以显示复杂 3D 场景。基本上，它就是添加了许多可以快速处理浮点运算的处理器。快速扫一眼树莓派你就会发现由于它和 PC 布局不同，没有空间可以添加 GPU。它的主板包括了处理器、内存、扩展槽，一切东西都集成到一个片上系统（SoC）中。这就是树莓派中间的最大的那块芯片。如果你从侧面仔细看，会发现它由两层组成，上面一层是 RAM，下面一层用于处理。

底层不只是 CPU（中央处理器单元）。事实上，CPU 只是它的一小部分。它还包含一个远比 CPU 强大的 GPU。当你运行普通程序时，GPU 处于空闲状态，CPU 处理所有工作。但处理 3D 图形时，CPU 本身不能够处理它，因此你需要将工作分一部分给 GPU。在本章中，我们将看到如何利用 GPU 来使用 OpenGL（GL 代表图形库，Graphics Library）创建 3D 场景。

在开始前我们需要诚实地告诉你：这是本书中最复杂的一章。这是不争的事实。使用 OpenGL 需要一些数学知识、一个新的编程语言和一些新概念。我们将放慢脚步，解释清楚我们遇见的所有事情。然而，如果你只是为了学习在 3D 世界中简单地画个魔方，可以直接跳到第 8 章。

6.1 获取模块

你需要两个模块：PyGame 和 RPiGL。如果你还没有安装 PyGame，请参阅第 4 章中的相应命令。RPiGL 可以从下面地址获取 <https://github.com/stephanh42/rpigl>。使用下载 Zip 按钮下载压缩包，然后按照下列命令（在 LXTerminal 中）解压缩，并安装它。

```
unzip rpigl-master.zip
cd rpigl-master
python3 setup.py build
sudo python3 setup.py install
```

如果一切顺利，你就可以运行 demo 程序了，尝试下列命令：

```
cd demos
python3 bumpedspere.py
```

你将会看到一个凹凸不平的球体在旋转（是的，这个文件名上有个拼写错误）。如果看到出错信息，在继续本章内容之前，你需要修正它们。

6.2 创建旋转立方体

旋转立方体是一个基本图形，所有的新 3D 程序员都需要尝试下它。它简单易懂，也涵盖了所有的基础，并且也不需要大量的数据来建立 3D 模型。



注意 再次提醒本书的网站是 www.wiley.com/go/python-raspberrypi。为了避免潜在的拼写错误，你可以下载、复制和粘贴代码到你的 IDE 或者代码编辑器中。

完整程序位于网站上的 chapter6-spinning-cube.py 中。由于篇幅太长，我们来分段阅读它。

首先你需要建立数据：

```
vertices = [(0.0,0.0,0.0,0.0), (0.5,0.0,0.0,0.0), (0.5,0.5,0.0,0.0),
            (0.0, 0.5,0.0,0.0), 0.0,0.0,-0.5), (0.5,0.0,-0.5),
            (0.5,0.5,-0.5), (0.0, 0.5,-0.5)]
indices_face_1 = (0, 1, 2, 0, 3)
```

我们要绘制的立方体的所有顶点保存在 vertices 列表中。实际上你绘制的不止一个旋转立方体。还包括旋转立方体的六个表面中的四个、静态立方体的边和一些点。做完这些，你将学到在屏幕上绘画的其他方法。

元组 `indices_face_1` 中保存了一组点，你将使用它们在屏幕上绘制特定的形状（其中的数值是列表 `vertices` 中的下标）。这些下标表示了立方体的一个表面，在实际代码中还有更多表面。我们随后会使用这些表面从同一个顶点池中绘制多个形状。

下一步是建立 OpenGL 环境（注意，如果你在跟着下载的文件往下看，这里我们不再按顺序进行）。

```
self.vertex_shader = glesutils.VertexShader(vertex_gls1)
self.fragment_shader = glesutils.FragmentShader(
    fragment_gls1)

self.program1 = glesutils.Program(self.vertex_shader,
                                  self.fragment_shader)
self.program1.use()

glClearDepthf(1.0)
glDepthFunc(GL_LESS)
 glEnable(GL_DEPTH_TEST)

glClearColor(0.5, 0.5, 0.5, 1)
```

之前，我们说过 GPU 是一个额外的处理器，需要使用 3D 图形时你可以用它来做数学计算。要使用它，我们需要对其编程，编程就需要源代码。这些程序由顶点着色和片段着色两部分构成。代码保存在变量 `vertex_gls1` 和 `fragment_gls1` 中，这叫做程序中内嵌程序（稍微我们会详细介绍）。为了使用它们，你需要把它们转化为着色器对象，然后将着色器对象和一个可运行程序绑定在一起。在你的 Python 程序中可以有超过一个的 OpenGL 程序，你可以使用 `use()` 方法来切换它们。这里只有一个程序，因此在开始的地方只调用了一次 `use()`。

最后的四行用来设置 OpenGL。首先，前三行指定清除缓冲区的值为 1。最后一行清除颜色缓冲区，将屏幕设置成中等灰度。OpenGL 颜色的四个值分别是红、蓝、绿和 Alpha（透明度）。每个值的范围都是 0 到 1。

下一个任务是将数据加载到 GPU 中。

```
self.vertices_buffer = array_spec.create_buffer(
    vertex_attrib=vertices)

self.elements_face_1 = glesutils.ElementBuffer(
    indices_face_1)
```

主程序运行在 CPU 上，3D 建模由 GPU 完成。在树莓派中这两者距离很近，但是数据在两个单元之间传输也需要一些时间。由于这一点，在开始运行之前最好将尽可能多的信息加载到 GPU 中。这就是缓冲区的作用。这里我们使用两类缓冲区来存储顶点和

下标信息。程序开始运行之后，只需传送你要使用的缓冲区的标记。在这个特殊程序中，每个 buffer 中只存了少量信息，使用和不适用缓冲区的差别并不明显。然而如果你需要加载复杂的 3D 模型，每个缓冲区都需要存储大量信息，这样就可以明显看出传输时间的区别。

6.2.1 向量和矩阵

在我们创建的 3D 世界中，所有的点都被定义成坐标 (x, y, z)。x 表示水平位置，y 表示垂直位置，z 表示深度。用数学名词描述这组数字，就是向量。物体中的每一个点称为顶点。^①每一个顶点都有一个向量来描述它的位置。你需要操作这些点才能在 3D 世界中移动物体。例如，你想放大物体，需要给每一个顶点一个比例因子。或者你想旋转物体，需要相应地移动每个顶点坐标。

这些都可以由向量矩阵代数完成。简单地说，就是对于你想做的每一个变换，都可以创建一个矩阵（纵横排列的二维数据），然后将向量乘以该矩阵得到一个新的向量。

看到乘这个词不用疑惑，这不是普通乘法。其数学过程有点复杂，由于 OpenGL 已经把这些都做好了，你不用再为此烦恼。在这个阶段你需要知道的是，要移动一个物体，需要创建一个矩阵，并将顶点坐标乘以合适的矩阵。如果你想超越本章范围，有很多在线或者印刷资料可供参考。这对于学习更多相关知识非常有用。

下面这段代码创建了两个矩阵，给我们演示了这一过程。transforms.compose() 用来将多个矩阵合并成一个。

```
self.outer_matrix = transforms.compose(
    transforms.rotation_degrees(20, "z"),
    transforms.rotation_degrees(20, "y"),
    transforms.rotation_degrees(20, "x"),
    transforms.scaling(1.2))

self.points_matrix = transforms.compose(
    transforms.stretching(0.1, 1, 1.5),
    transforms.translation(-0.5, -0.5, -0.5))
```

transforms.rotation_degrees()、transforms.scaling() 和 transforms.stretching() 都会返回一个矩阵，分别用于旋转、缩放和拉伸操作时拿来与向量相乘。

在继续绘制屏幕之前，我们回过头来看一下我们加载到 GPU 的顶点和片段着色器的代码：

^① 原文如此，可能是作者笔误，这里应该是物体中两条边的交叉点。——译者注

```

array_spec = glesutils.ArraySpec("vertex_attrib:3f")

vertex_gsls = array_spec.gsls() + """
uniform mat4 transform_matrix;
void main(void) {
    gl_Position = transform_matrix * vec4(vertex_attrib, 1.0);
    gl_PointSize = 2.0;
}
"""

fragment_gsls = """
uniform vec4 color;
void main(void) {
    gl_FragColor = color;
}
"""

```

它创建了两个包含代码的字符串：vertex_gsls 和 fragment_gsls。你可能注意到，这并不是 Python 代码。对 GPU 编程需要使用一种称为 GLSL (Graphics Library Shader Language, 图形库着色语言) 的特殊语言。有点类似于 C 语言（一种可以为主 CPU 编写程序的语言）。GLSL 和 Python 的主要区别在于：

- 每个语句都以分号结尾。
- 缩进级别无关紧要，代码段包围在花括号中。
- 变量有自己的类型，且只能存储这种类型的数据。

关键字和函数也不一样。由于这些都是全新的概念，我们一行一行来解释。

```
array_spec = glesutils.ArraySpec("vertex_attrib:3f")
```

创建了一个新的 ArraySpec 对象（在别处需要用到）。参数表示你将给它传递一个 vertex_attrib 的属性，该属性是一个三维浮点向量（3f）。

```
vertex_gsls = array_spec.gsls() + """
```

创建了一个变量并赋一个字符串给它。array_spec.gsls() 返回一段代码，用于创建 array_spec(vertex_attrib) 中的属性。三个引号告诉 Python，这是一个多行的字符串。

```
uniform mat4 transform_matrix;
```

创建了一个名为 transform_matrix 的 uniform 变量。该变量可以存储 4×4 矩阵。uniform 关键字表示可以从 Python 代码中给它赋值。

```
void main(void) {
```

创建 main 函数，每次程序运行时都会运行该函数。第一个 void 表示它不返回任何值，第二个表示它不需要任何输入参数。注意花括号，它表示代码段的开始。

```
gl_Position = transform_matrix * vec4(vertex_attrib, 1.0);
```

这里的缩进并不是必须的（Python 中却是必须的），这里加入缩进只是为了方便阅读程序。它将表示顶点位置的向量和表示转换的矩阵相乘。注意它们都是 4 维矩阵。现在先不用关心最后一个数值，先把它设为 1.0。

将顶点绘制在屏幕上时，每个顶点 shader 都需要设置 `gl_Position` 作为自己的变量。设置好之后，OpenGL 会处理好剩下的工作。

```
gl_PointSize = 2.0;
}
```

`gl_PointSize` 只是简单地设置你绘制在屏幕上的点的大小（我们将涉及点、线和一点三角形）。花括号用来结束 `main` 函数。三个双引号表示字符串结束。

顶点着色器在场景里的每个顶点将调用一次，片段着色器在场景里的每个像素（顺序）将调用一次。因此片段着色器的运行次数远比顶点着色器多，片段着色器都很简单。本例中，它只有 4 行：

```
fragment_glsL = """
uniform vec4 color;
void main(void) {
    gl_FragColor = color;
}
```

像顶点着色器通常设置 `gl_Position` 一样，片段着色器通常设置 `gl_FragColor`。它是一个四维向量表示该位置的颜色。

这些都就绪之后，剩下的事情就是把这些物体放入 3D 世界：

```
#Draw outer lines
self.program1.uniform.transform_matrix.value =
    self.outer_matrix
self.program1.uniform.color.value = (1, 1, 1, 1)
self.vertices_buffer.draw(elements=self.elements_outer,
    mode=GL_LINE_STRIP)

#Draw points
self.program1.uniform.transform_matrix.value =
    self.points_matrix
self.program1.uniform.color.value = (0, 0, 0, 1)
self.vertices_buffer.draw(elements=self.elements_points,
    mode=GL_POINTS)

#Draw spinning cube
rotation_matrix =
    transforms.compose(
        transforms.rotation_degrees(self.angle, "z"),
```

```

        transforms.rotation_degrees(self.angle, "y"),
        transforms.rotation_degrees(self.angle, "x")))
self.program1.uniform.transform_matrix.value =
    rotation_matrix
self.program1.uniform.color.value = (1, 0, 0, 1)
self.vertices_buffer.draw(elements=self.elements_face_1,
    mode=GL_TRIANGLE_STRIP)

```

这里有三个不同的 draw 函数集，但它们的格式都一样。首先它们使用属性 self.program1.uniform.transform_matrix.value 为顶点着色器中的 transform_matrix 赋值。然后使用相似的一行代码为片段着色器中的 color 赋值。最后调用 self.vertices_buffer.draw() 为将像素绘制到 3D 世界中。该函数需要两个参数。第一个参数是 elements，即包含了顶角下标的元素缓存。第二个参数是 mode，用来告诉 OpenGL 这些顶点的含义。这里的三个分别是 GL_POINTS、GL_LINE_STRIP 和 GL_TRIANGLE_STRIP。

GL_POINTS 的含义很明显，它将每个顶点绘制成为一个点。GL_LINE_STRIP 表示画一条连续的线，每个顶点都是线上的一个点。还有一种 GL_LINES 模式，表示在每一对顶点间画一条独立的线。

GL_TRIANGLE_STRIP 用来绘制一个连续的三角形。本例中，我们使用 5 个顶点来定义是个正方形。实际上，我们使用了 4 个顶点，只是其中一个使用了两次。使用三角形带，前三个点可以组成一个三角形，然后是第三、四、五个点，然后是第五、六、七个点，然后是第七、八、九个点等[⊖]。前一个三角形的后两个点构成了当前三角形的前两个点。使用这种方法，你可以拼出任意表面。

还有其他两种方法，GL_TRIANGLES 是每三个顶点绘制一个三角形，GL_TRIANGLE_FAN 是每个顶点都共享一个单独的顶点，各个三角形形成一个扇形序列，看起来和花瓣一样。

6.2.2 组合包装

完整代码如下所示（提示，可以从网站上下载 chapter6-spinning-cube.py）：

```

import pygame
from rpigl import glesutils, transforms
from rpigl.gles2 import *

vertices = [(0.0,0.0,0.0), (0.5,0.0,0.0), (0.5,0.5,0.0),
            (0.0,0.0,0.0), (0.5,0.0,0.0), (0.0,0.5,0.0)]

```

[⊖] 即新的三角形可重用最后两个索引，这里可能是作者笔误，应该是第二、三、四、三、四、五、四、五、六个点。——译者注

```

        (0.0, 0.5, 0.0),
        (0.0, 0.0, -0.5), (0.5, 0.0, -0.5), (0.5, 0.5, -0.5),
        (0.0, 0.5, -0.5)]
    indices_outer = (0, 1, 2, 3, 0, 4, 5, 6, 1, 5, 2, 6, 7, 3, 7, 4)
    indices_points = (0, 1, 2, 3)

array_spec = glesutils.ArraySpec("vertex_attrib:3f")
array_spec.add_attribute("position", 0, 3, GL_FLOAT)
array_spec.add_attribute("color", 1, 3, GL_UNSIGNED_BYTE)
array_spec.add_attribute("light_dir", 2, 3, GL_FLOAT)

vertex_gls1 = array_spec.gls1() + """
uniform mat4 transform_matrix;
void main(void) {
    gl_Position = transform_matrix * vec4(vertex_attrib, 1.0);
    gl_PointSize = 2.0;
}
"""

fragment_gls1 = """
uniform vec4 color;
void main(void) {
    gl_FragColor = color;
}
"""

class MyWindow(glesutils.GameWindow):
    def __init__(self):
        self.angle = 10
        self.vertex_shader = glesutils.VertexShader(vertex_gls1)
        self.fragment_shader =
            glesutils.FragmentShader(fragment_gls1)
        self.program1 = glesutils.Program(self.vertex_shader,
                                         self.fragment_shader)
        self.program1.use()
        glClearDepthf(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glClearColor(0.5, 0.5, 0.5, 1)
        self.program1.uniform.light_dir.value = ((0, 1, -1))

```

```

self.vertices_buffer =
    array_spec.create_buffer(vertex_attrib=vertices)
self.elements_face_1 =
    glesutils.ElementBuffer(indices_face_1)
self.elements_face_2 =
    glesutils.ElementBuffer(indices_face_2)
self.elements_face_3 =
    glesutils.ElementBuffer(indices_face_3)
self.elements_face_4 =
    glesutils.ElementBuffer(indices_face_4)

self.elements_outer =
    glesutils.ElementBuffer(indices_outer)
self.elements_points =
    glesutils.ElementBuffer(indices_points)

self.outer_matrix =
    transforms.compose(
        transforms.rotation_degrees(20, "z"),
        transforms.rotation_degrees(20, "y"),
        transforms.rotation_degrees(20, "x"),
        transforms.scaling(1.2))

self.points_matrix =
    transforms.compose(
        transforms.stretching(0.1, 1, 1.5),
        transforms.translation(-0.5, -0.5, -0.5))

def on_frame(self, time):
    self.angle = self.angle + time*0.02
    self.redraw()

def draw(self):
    #Draw outer lines
    self.program1.uniform.transform_matrix.value =
        self.outer_matrix
    self.program1.uniform.color.value = (1, 1, 1, 1)
    self.vertices_buffer.draw(elements=self.elements_outer,
        mode=GL_LINE_STRIP)

    #Draw points at inner points
    self.program1.uniform.transform_matrix.value =
        self.points_matrix
    self.program1.uniform.color.value = (0, 0, 0, 1)
    self.vertices_buffer.draw(elements=self.elements_points,
        mode=GL_POINTS)

#Draw spinning cube
rotation_matrix = transforms.compose
    (transforms.rotation_degrees(self.angle, "z"),
     transforms.rotation_degrees(self.angle, "y"),
     transforms.rotation_degrees(self.angle, "x"))

```

```

        self.program1.uniform.transform_matrix.value =
                    rotation_matrix
        self.program1.uniform.color.value = (1, 0, 0, 1)
        self.vertices_buffer.draw(elements=self.elements_face_1,
                                  mode=GL_TRIANGLE_STRIP)
        self.program1.uniform.color.value = (0, 1, 0, 1)
        self.vertices_buffer.draw(elements=self.elements_face_2,
                                  mode=GL_TRIANGLE_STRIP)
        self.program1.uniform.color.value = (0, 0, 1, 1)
        self.vertices_buffer.draw(elements=self.elements_face_3,
                                  mode=GL_TRIANGLE_STRIP)
        self.program1.uniform.color.value = (0, 1, 1, 1)
        self.vertices_buffer.draw(elements=self.elements_face_4,
                                  mode=GL_TRIANGLE_STRIP)

MyWindow(200, 200, pygame.RESIZABLE).run()

```

运行结果应该是一个如图 6-1 所示的 3D 渲染图：

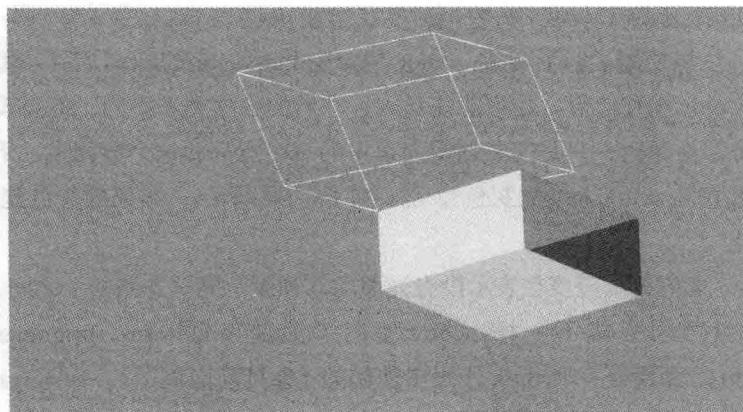


图 6-1 大多数 3D 图形使用的基本技术演示

6.2.3 增加光照

运行前面的例子，你会看到一个立方体的四个面围绕一些线和点在旋转。你可能注意到这里缺少些什么：光。不论这些表面朝向哪里，它们全部都是一个亮度，这几乎不可能出现在现实生活中。相反，根据朝向，总是有一个或更多的光源从不同角度照向物体。

有些版本的 OpenGL 可以自动处理这部分。然而树莓派（和大多数移动设备）上的 OpenGL ES 版本做不到，因此你必须自己来计算光线。

下面的例子将创建一个点光源照射下的旋转立方体。随着立方体的移动，你将会看到

它的不同部分在变亮或者变暗。在计算某个点的亮度时需要使用两个数据——立方体上的点到光源的距离和光线的照射角度。

计算点到光源的距离

我们先看第一点。立方体的每个面都是由屏幕上的上百个像素显示出来。片段着色器计算出这些像素的值。为了计算像素的颜色，片段着色器需要知道每个像素的亮度。这些都由下面的代码来完成：

```
fragment_gls = """
uniform vec4 color;

varying float brightness;

void main(void) {
    gl_FragColor = brightness*color;
}
"""


```

可以看到，这里使用了两个变量：color 和 brightness。brightness 是浮点型，可以用来替换给 gl_FragColor 设置的值。你可能注意到，它们的数据类型不同，但是没有问题。当给向量乘以一个浮点数时，只是简单地将向量中的每个点都和这个数相乘一下。比如，你有一个颜色向量 (0.8, 0.8, 0.8) (亮灰色)，然后你把它乘以 0.5，结果的颜色就是 (0.4, 0.4, 0.4) (中等灰色)。

brightness，取值范围应该在表示没有光照的 0 和表示完全光照的 1 之间。

这两个变量声明方式不一样。color 使用了关键字 uniform，brightness 被设置成 varying。uniform 的变量是在 Python 代码中使用的（参见前面例子）。而 varying 变量是用来给顶点着色器使用的，见下面的代码：

```
vertex_gls = array_spec.gls() + """

...
varying float brightness;

void main(void) {
    ...
    float distance = length(vec4(light_position, 1.0) - gl_Position);
    brightness = 1.0/(distance * distance);
}
"""


```

这里没有计算角度和位置的代码（后面我们再介绍），但这段代码给出了如何根据距离来计算亮度明暗。length() 函数返回一个向量的长度（根据勾股定理计算出来）。本例中，

它用来计算顶点和光源的距离。

当物体远离光源时，它就会变暗。然而，这种变化并不是线性的。如果你把物体移动到原来距离的两倍远，它并不会变得只有以前的一半亮，而是会只剩下四分之一的亮度。物体的亮度和它到光源的距离的平方成反比，这体现在最后一行代码中。

这段代码只计算了顶点的亮度。为了让程序能更好地运行，它需要知道表面上每个点的亮度。幸运的是，OpenGL为我们做好了这部分。当你使用 `varying` 关键字来创建变量时，它会插补一些值传入片段着色器。也就意味着它会根据每个点到三个顶点的距离改变该点的亮度，以渲染出一个平滑协调的亮度。

计算反射角

亮度的另一个因素是表面和光源的角度。为了计算这个角度。你需要一个法线。所谓法线就是垂直于平面的向量。

向量可以用于描述位置，也可以用来描述直线。例如，向量 $(1, 1, 1)$ 可以表示位于这个坐标的点，也可以表示 3D 世界中的一条从点 $(0, 0, 0)$ 到 $(1, 1, 1)$ 的直线。顶点的向量表示顶点的位置，法线的向量表示直线。

法线是一个表示物体朝向的向量。以立方体的第一面为例，顶点为 $(0.0, 0.0, 0.0), (0.5, 0., 0.0), (0.5, 0.5, 0.0), (0.0, 0.5, 0.0)$ 。所有顶点都在 z 轴的一个平面上，因此它的法线是 $(0, 0, 1.0)$ 。如果你从点 $(0, 0, 0)$ 到 $(0, 0, 1.0)$ 画一条线，它将和该平面成 90 度角。法线的长度通常都是 1。

为了计算亮度，你需要计算法线和光源的角度。如果它们在同一个直线上，物体表面应该是全亮。如果成 90 度或者更大角度，该平面不会被照亮。然而，和距离类似，这两者之间也不是线性关系，而是成正弦关系。正弦函数正好和我们的情况不符，因此你需要计算余弦。

为此，你需要使用三角函数工具箱。这里有个函数叫 `dot product`（点积），需要两个向量，返回值如下所示：

```
dot(A, B) = length(A) * length(B) * cosine(A, B)
```

计算点积远比计算余弦快，因此我们可以利用这个数学捷径。如果每个变量的长度都为 1，事情就会变得很简单，这时点积就等于余弦。

法线的长度永远为 1，因此你不用担心它。GLSL 中有个 `normalize()` 函数，可以得到任意向量的归一化向量——和该向量方向相同，长度为 1。因此你可以计算这两个向量的余弦。

```
cosine = dot(surface_normal, normalize(gl_Position -
                                         light_position))
```

然而，大多数时候并不止有一个光源。还有很多从墙壁和其他物体的反光。将所有这些都计算进来显然会很复杂。简单的做法就是增加一个环境光量。也就是简单地将从各个角度照向物体的光加在一起。可以这样添加环境光：

```
brightness = max(cosine, ambient_light)
```

下面代码将所有这些合并在一起构成顶点着色器：

```
vertex_gsls = array_spec.gsls() + """
uniform mat4 transform_matrix;
uniform vec3 light_position;
uniform float ambient_light;
uniform vec3 face_normal;

varying float brightness;
void main(void) {
    gl_Position = transform_matrix * vec4(vertex_attrib, 1.0);
    vec4 spun_face_normal = normalize(transform_matrix *
                                        vec4(face_normal, 1.0));
    float distance = length(vec4(light_position, 1.0) - gl_Position);
    vec4 light_direction = normalize(vec4(light_position, 1.0) -
                                      gl_Position);
    float light_amount_angle = max(dot(spun_face_normal,
                                         light_direction), ambient_light);

    float light_distance_drop = 1.0/(distance * distance);
    brightness = light_amount_angle * light_distance_drop;
}
```

这里点积计算并不是直接使用原始表面法线，而是使用经过变换的立方体的表面法线。本例中，从技术上说 `transform_matrix` 并不缩放或者拉伸法线，因此不需要规范化。然而为了使代码能够用在其他程序中使用，我们加上了规范化。

剩下的代码和上一个例子基本相同。这里没有点或线，只有立方体的 6 个面。这段代码在网站上的 `chapter6-lighting.py`。

```
import pygame
from rpigl import glesutils, transforms
from rpigl.gles2 import *

vertices = [(0.0,0.0,0.0,0.0), (0.5,0.0,0.0,0.0), x/4984720?c=pledges
            (0.5,0.5,0.0), (0.0, 0.5,0.0),
            (0.0,0.0,-0.5), (0.5,0.0,-0.5),
            (0.5,0.5,-0.5), (0.0, 0.5,-0.5)]
```

```

faces = [{"vertex_index":(0, 1, 2, 0, 3), "normal":(0,0,1),
          "colour":(1, 0, 0, 1)},
          {"vertex_index":(4, 5, 6, 4, 7), "normal":(0,0,-1),
          "colour":(0, 1, 0, 1)},
          {"vertex_index":(1, 5, 6, 1, 2), "normal":(1,0,0),
          "colour":(0, 0, 1, 1)},
          {"vertex_index":(0, 4, 7, 0 ,3), "normal":(-1,0,0),
          "colour":(1, 0, 1, 1)},
          {"vertex_index":(3, 2, 6, 3, 7), "normal":(0,1,0),
          "colour":(1, 1, 0, 1)},
          {"vertex_index":(0, 1, 5, 0, 4), "normal":(0,-1,0),
          "colour":(0, 1, 1, 1)}]
array_spec = glesutils.ArraySpec("vertex_attrib:3f")

vertex_gsls = array_spec.gsls() + """
uniform mat4 transform_matrix;
uniform vec3 light_position;
uniform float ambient_light;
uniform vec3 face_normal;

varying float brightness;

void main(void) {
    gl_Position = transform_matrix * vec4(vertex_attrib, 1.0);
    vec4 spun_face_normal = normalize(transform_matrix *
                                         vec4(face_normal, 1.0));
    float distance = length(vec4(light_position, 1.0) - gl_Position);
    vec4 light_direction = normalize(vec4(light_position, 1.0) -
                                      gl_Position);
    float light_amount_angle = max(dot(spun_face_normal,
                                         light_direction), ambient_light);
    float light_distance_drop = 1.0/(distance * distance);
    brightness = light_amount_angle * light_distance_drop;
    gl_PointSize = 2.0;
}
"""

fragment_gsls = """
uniform vec4 color;

varying float brightness;

void main(void) {
    gl_FragColor = brightness*color;
}
"""

class MyWindow(glesutils.GameWindow):

    def init(self):

```

```

        self.angle = 10
        self.framerate = 20
        self.vertex_shader = glesutils.VertexShader(vertex_gls1)
        self.fragment_shader =
            glesutils.FragmentShader(fragment_gls1)
        self.program1 = glesutils.Program(self.vertex_shader,
                                         self.fragment_shader)
        self.program1.use()

        glClearDepthf(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glFrontFace(GL_CW)

        glClearColor(0.5, 0.5, 0.5, 1)

        self.program1.uniform.light_dir.value = ((0, 1, -1))

        self.vertices_buffer =
            array_spec.create_buffer(vertex_attrib=vertices)
        for face in faces:
            face["element_buffer"] =
                glesutils.ElementBuffer(face["vertex_index"])

        self.outer_matrix = transforms.compose(
            transforms.rotation_degrees(20, "z"),
            transforms.rotation_degrees(20, "y"),
            transforms.rotation_degrees(20, "x"),
            transforms.scaling(1.2))

        self.points_matrix = transforms.compose(
            transforms.stretching(0.1, 1, 1.5),
            transforms.translation(-0.5, -0.5, -0.5))

    def on_frame(self, time):
        self.angle = self.angle + time*0.02
        self.redraw()

    def draw(self):
        self.program1.uniform.light_position.value = (0,0,-1)
        self.program1.uniform.ambient_light.value = 0.3

        rotation_matrix = transforms.compose(
            transforms.rotation_degrees(self.angle, "z"),
            transforms.rotation_degrees(self.angle, "y"),
            transforms.rotation_degrees(self.angle, "x"))

        self.program1.uniform.transform_matrix.value =
            rotation_matrix

```

```
for face in faces:  
    self.program1.uniform.color.value = face["colour"]  
    self.program1.uniform.face_normal.value =  
        face["normal"]  
  
    self.vertices_buffer.draw(elements=face["element_buffer"],  
        mode=GL_TRIANGLE_STRIP)  
  
MyWindow(200, 200, pygame.RESIZABLE).run()
```

结果如图 6-2 所示。

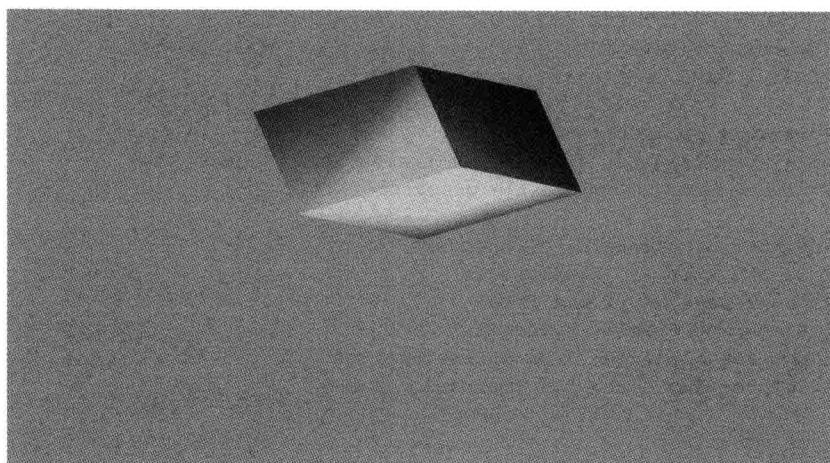


图 6-2 和第一个程序不同，光线增加了现实感

6.3 让屏幕起舞

至此，你见识过如何在屏幕上绘制立方体并照亮它们，但仅限于此。在下一个项目中，你将看到如何创建一个随音乐起舞的 3D 模型。大多数计算机音乐播放器都有相似的功能，为了听众增加点视觉效果而在屏幕上显示出声音效果。

因为本章主要介绍 3D 图形而不是音频处理，为了简单起见我们只处理 WAV 文件。也就是说我们可以使用 Python 的 wave 模块从文件中读取声音数据。如果你的音乐是 MP3 格式，你需要先把它们转换成 WAV 格式，可以在 LXTerminal 下通过 mpg123 命令行工具完成转换。使用之前需要先安装：

```
sudo apt-get install mpg123
```

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`



然后就可以用下面命令完成转换：

```
mpg123 -w output-filename.wav input-filename.mp3
```

如果没有合适的音乐，可以从这个网站上下载一些免费音乐：<http://freemusicarchive.org/>。

第一步是播放音乐。和第 5 章中播放音乐的方法一样，也是使用 PyGame 合成器。它将音乐通过树莓派的音频通道播放出来，并不会给你提供用来控制 3D 图形的声音数据。为此，我们将使用第二个模块 wave。

```
print("opening file")
sound_file = wave.open("test.wav", 'rb')

print("getting parameters")
(channels, sample_size, frame_rate, frames, compression_type,
 compression_name) = sound_file.getparams()

print("Number of channels: ", channels)
print("Sample size: ", sample_size)
print("Frame rate: ", frame_rate)
print("Number of Frames: ", frames)
print("Compression type: ", compression_type)
print("Compression name: ", compression_name)

print("readframes")
data = sound_file.readframes(channels*sample_size*frames)
print(len(data))
```

这里创建了变量 data，声音以字节形式存储在里面。声音编码是两个字节一组。为了读取一组数据，需要使用 int 类中的 from_bytes() 方法将两个字节组合在一起。

```
sound_data = int.from_bytes(data[i:i+1],
                             byteorder='little', signed=True)
```

声音以帧的形式存储，每帧就是上面提到的一组两个字节，其取值范围从 -32 768 到 32 767。如果文件是立体声的，每个时间点有两帧数据。通常每秒有 44 100 帧（该数值存储在变量 frame_rate 中）。

你可以用很多种方式展示这些数据。唯一的需求就是当声音播放时，输出需要以娱乐化的方式展现出来。我们将使用两种方法。当音量变化时从屏幕底下绘出一组变化的 3D 条带，和一组闪光点（如果你喜欢诗意，可以使用星形）用来点缀音乐。为了增加一点花式，我们将整个可视化部分沿纵轴旋转起来，并把 3D 条的颜色设置成从底端的蓝色渐变到顶端的红色。

6.3.1 建立3D模型

有多种方式可以给数据建模，而我们将使用单个立方体顶点数据代表图形上的每个方块并通过变换矩阵来操作它。星形就是简单地使用一组随机出现的点。我们先来处理块状物体。

每个块都有相同是一组顶点和索引。它们之间的区别就在于大小（伸缩矩阵）和位置（变换矩阵）。每个块还有另外两个矩阵：一个用于旋转（每帧改变一次），另一个用于缩放（启动时设置）。

你需要将所有这些矩阵合并在一起。可以使用Python的transforms.compose()方法。然而，这样每帧都需要处理大量数据。我们提到过GPU远比CPU强大，因此让GPU来处理所有的矩阵操作会更有效。

渲染程序如下：

```
vertex_gls = array_spec.gls() + """
uniform mat4 position_matrix;
uniform mat4 eye_matrix;
uniform mat4 scaling_matrix;
uniform mat4 sound_matrix;
uniform float point_size;

varying float red;

void main(void) {
    gl_Position = eye_matrix * position_matrix * scaling_matrix *
                  sound_matrix * vec4(vertex_attrib, 1.0);

    red = (gl_Position[1]+0.9)/2.0;

    gl_PointSize = point_size;
}
"""

fragment_gls = """
uniform vec4 color;
varying float red;

void main(void) {
    gl_FragColor = vec4(red, 0.0, (1.0-red)/5.0, 1.0);
}
"""
```

这些代码和我们之前见到的非常相似。矩阵乘法不可交换并没有意义^⑤。这里以某种方式展示了矩阵乘法和顺序是有关系的。顺序并不总是影响结果，但有些时候会。本

^⑤ 满足乘法交换律的方阵称为可交换矩阵，即矩阵A、B满足 $A \cdot B = B \cdot A$ 。——译者注

例中，确保 eye_matrix（用来旋转整个场景）在 position_matrix（用来将单个长条放在正确位置）之前非常重要。如果交换了两者顺序，就会变成单个长条围绕这某个点旋转而不是整个场景在旋转。这并不是什么大问题，我们只是让它看起来感觉更好一点。你希望矩阵以什么顺序变形，就可以相应地改变矩阵的乘法顺序（尝试一下，如果失败就再试试）。

你可能注意到这里有一个 varying 类型的变量 red。它作为 gl_Position 向量的纵轴（y 轴或者 1）并将其转换为颜色值。

你可能还注意到，点的尺寸由一个 uniform 变量设置，我们稍后再来处理它。

6.3.2 计算声音强度

每显示一帧，程序将计算当前的声音强度并将其中一个长条设置成相应的音量强度。我们将循环所有的长条以使它们作为一个整体显示出音量强度在最近的 20 帧中的变化。

为此，你要能够计算出音量的强度。使用 on_frame() 方法可以做到这一点。

```
scale_factor = 0

if frame_position + 1000 < len(data):
    for i in range(1, 500):
        scale_factor1 = scale_factor +
            int.from_bytes(data[frame_position+2*i:
                                frame_position+(2*i)+1],
                           byteorder='little', signed=True)**2
    scale_factor1 = scale_factor1 / 500

    self.sound_matrix[self.counter%20] =
        transforms.stretching(1.0, scale_factor1, 1.0)

    self.program1.uniform.point_size.value =
        float(scale_factor1/4)

    self.counter = self.counter+1
```

从声音的某个位置（用音轨开始播放起到当前的时间间隔来表示）起，取其后 500 帧计算它们的平方和均值可以得到当前位置的音量强度。使用音量强度的平方有两个好处。首先他可以确保每个值都是正数，因为一个负数的平方还是正数；其次音乐每增长一点，它可以使长条的变化更快，在屏幕上的效果更好。这里只是为了使效果看起来更好一点，并不需要科学地计算出正确图形，因此，使用这种方式计算是可以接受的。

使用 500 帧相加的原因很简单，因为相加的帧越多，结果越精确，但是计算会越慢。我们使用 500 是权衡了性能（与显示帧率有关）和音乐模型的精确性。如果你想使之更完善，可以增加一些前面章节描述的功能。

注意我们还设置了星形的尺寸。我们用来缩小比例因子的值（方块是 500，星形是 4），是通过视觉效果——看起来是否好看而不是通过计算来选择的。

视觉效果的完整代码如下所示（参见网站上的 chapter6-music.py）。

```
import pygame
from rpigl import glesutils, transforms
from rpigl.gles2 import *
import random
import wave
import time

vertices = [(0.0,0.0,0.0), (0.5,0.0,0.0),
            (0.5,0.5,0.0), (0.0, 0.5,0.0),
            (0.0,0.0,-0.5), (0.5,0.0,-0.5),
            (0.5,0.5,-0.5), (0.0, 0.5,-0.5)]

indices_outer = (0, 1, 2, 3, 0, 4, 5, 1, 5, 6, 2, 6, 7, 3, 7, 4)
vertices_points = []
indices_points = []

for i in range(0,100):
    vertices_points.append(((20 * random.random())-10,
                           (20 * random.random()), (20 * random.random())-10))
    indices_points.append(i)

array_spec = glesutils.ArraySpec("vertex_attrib:3f")

vertex_gls1 = array_spec.gls1() + """
uniform mat4 position_matrix;
uniform mat4 eye_matrix;
uniform mat4 scaling_matrix;
uniform mat4 sound_matrix;
uniform float point_size;

varying float red;

void main(void) {
    gl_Position = eye_matrix * position_matrix * scaling_matrix *
    sound_matrix * vec4(vertex_attrib, 1.0);
    red = (gl_Position[1]+0.9)/2.0;
    gl_PointSize = point_size;
}
"""

fragment_gls1 = """


```

```

uniform vec4 color;
varying float red;

void main(void) {
    gl_FragColor = vec4(red, 0.0, (1.0-red)/5.0, 1.0);
}

"""

class MyWindow(glesutils.GameWindow):

    def init(self):
        self.angle_x = 5
        self.angle_y = 10
        self.angle_z = 5
        self.counter = 0
        self.vertex_shader = glesutils.VertexShader(vertex_gsl)
        self.fragment_shader = glesutils.FragmentShader(
            fragment_gsl)

        self.program1 = glesutils.Program(self.vertex_shader,
                                          self.fragment_shader)
        self.program1.use()

        glClearDepthf(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)

        glClearColor(0.0, 0.0, 0.0, 1)

        self.program1.uniform.light_dir.value = ((0, 1, -1))

        self.vertices_buffer = array_spec.create_buffer(
            vertex_attrib=vertices)
        self.points_buffer = array_spec.create_buffer(
            vertex_attrib=vertices_points)

        self.elements_outer = glesutils.ElementBuffer(
            indices_outer)
        self.elements_points = glesutils.ElementBuffer(
            indices_points)

        self.blank_matrix = transforms.translation(0.0, 0.0, 0.0)

        self.position_matrix = []
        for i in range(0,20):

            self.position_matrix.append(transforms.translation((i/10)-0.95,
                                                               0.0, 0.0))
        self.sound_matrix = []
        for i in range(0,20):

```

```

        self.sound_matrix.append(transforms.translation(
            0.0, 0.0, 0.0))

    self.program1.uniform.scaling_matrix.value =
        transforms.scaling(0.1)
    self.program1.uniform.eye_matrix.value =
        transforms.compose(
            transforms.rotation_degrees(self.angle_z, "z"),
            transforms.rotation_degrees(self.angle_x, "y"),
            transforms.rotation_degrees(self.angle_x, "x"),
            transforms.translation(0.0, -0.9, 0.0))
    self.counter = 0

def on_frame(self, ftime):
    global start_time
    global data

    self.angle_y = self.angle_y + .5

    self.program1.uniform.eye_matrix.value =
        transforms.compose(
            transforms.rotation_degrees(self.angle_z, "z"),
            transforms.rotation_degrees(self.angle_y, "y"),
            transforms.rotation_degrees(self.angle_x, "x"),
            transforms.translation(0.0, -0.9, 0.0))

    frame_position = int((pygame.time.get_ticks() -
                          start_time) * 44.1 * 4)

    scale_factor = 0

    if frame_position + 1000 < len(data):
        for i in range(1,500):
            scale_factor1 = scale_factor +
                int.from_bytes(data[frame_position+2*i:
                                    frame_position+(2*i)+1],
                           byteorder='little', signed=True)**2
        scale_factor1 = scale_factor1 / 500

        self.sound_matrix[self.counter%20] =
            transforms.stretching(1.0,scale_factor1,1.0)

        self.program1.uniform.point_size.value =
            float(scale_factor1/4)

        self.counter = self.counter+1

    self.redraw()

```

```

def draw(self):
    self.program1.uniform.color.value = (1, 1, 1, 1)
    for i in range(0,20):
        self.program1.uniform.position_matrix.value =
                                         self.position_matrix[i]
        self.program1.uniform.sound_matrix.value =
                                         self.sound_matrix[i]
        self.verteces_buffer.draw(elements=self.elements_outer,
                                   mode=GL_LINE_STRIP)
        self.counter = self.counter +1

    self.program1.uniform.position_matrix.value =
                                         self.blank_matrix
    self.program1.uniform.sound_matrix.value =
                                         self.blank_matrix
    self.points_buffer.draw(mode=GL_POINTS)

print("starting pygame mixer")
pygame.mixer.pre_init(44100, -16, 2, 2048)
pygame.mixer.init()

music = pygame.mixer.Sound("tell.wav")

print("opening file")
sound_file = wave.open("test.wav",'rb')

print("getting parameters")
(channels, sample_size, frame_rate, frames, compression_type,
 compression_name) = sound_file.getparams()

print("Number of channels: ", channels)
print("Sample size: ", sample_size)
print("Frame rate: ", frame_rate)
print("Number of Frames: ", frames)
print("Compression type: ", compression_type)
print("Compression name: ", compression_name)

print("readframes")
data = sound_file.readframes(4*frames)
print(len(data))

print("starting audio")
music.play()
start_time = pygame.time.get_ticks()

print("start time: ", time.clock())

MyWindow(200, 200, pygame.RESIZABLE).run()

```

运行结果见图 6-3，有点令人惊喜的 80 年代感觉。

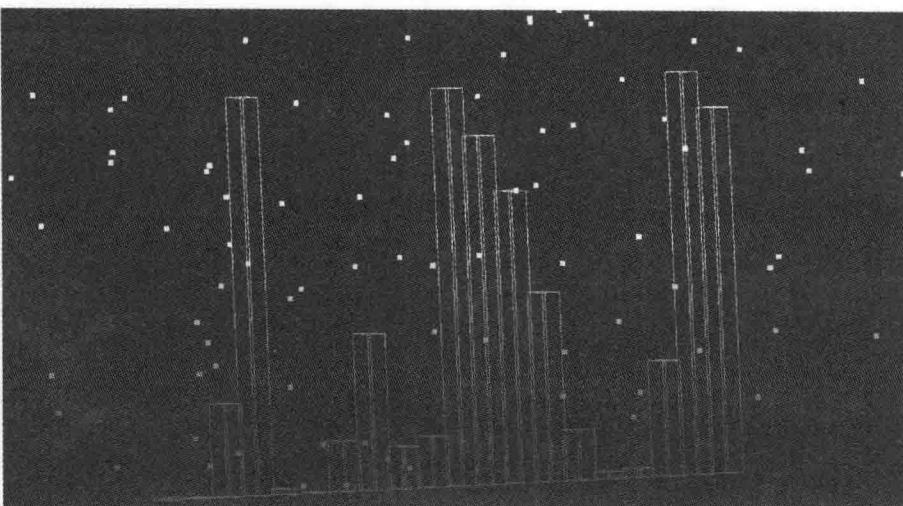


图 6-3 运行时看起来会更好。不要相信我们，自己运行程序看结果

6.4 继续完善

考虑下这些想法，继续完善该项目：

- 将星形设定为固定大小，但改变它们的亮度。可以用另一个 OpenGL 程序来完成，尽管这不是必须的。
- 还可以增加光照，为了更真实，需要用三角形带代替直线。
- 实验下改变方条和星形数量，可以让用户能够轻松配置它们的数量。
- 用图形显示频率图形而不是一直显示音量。为此，你需要使用 SciPi 模块计算音乐的傅里叶变换。注意这项任务需要一定数学知识，很有挑战性，有兴趣的读者可以试试。
- 提供一个更好的接口用来选择播放文件。根据个人喜好，可以选择命令行、文本菜单或者图形界面。这三种例子可以在本书别的部分找到。
- 改变视觉效果。让星星从上落下或者打旋都可以。还可以在旋转的时候缩小或放大它。颜色渲染也不用一直保持不变。要有创造性：3D 世界在等着你发挥想象力来塑造。

6.5 添加纹理

开始时我们说过这是本书中最复杂的一章。虽然它比较难，但也只是介绍了 OpenGL。

还有非常大一部分我们没有覆盖到。如果你觉得有趣并乐于继续深入探索，其中最有意思的一个部分就是纹理。它可以为你的 3D 模型添加很多细节。RPiGL 中提供了几个例子，可以从这些例子入手。在你之前下载的 Python 模块中的 `demos` 子文件夹中可以找到它们。

第一个是 `icoso.py`，它将世界地图设计成一个旋转的球体。图像取自 `world_cube_net.png` 并使用 `glesutils.Texture.from_surface()` 方法加载图像，然后使用 `bind()` 将其应用到着色器中。

第二个是 `bumpedspere.py`，它工作地有点不一样。还记得你是如何使用法线来计算反射光线总量的么？使用这个文件可以创造一个带纹理的平滑表面。如果你改变法线，使其不再完全垂直于表面，这将会改变此处的光线，仿佛表面上有轻微的隆起。这其实是视觉误差，因为世界上不存在这样的表面，但效果却是平面上出现很多小隆起，并不需要使用巨大的三角形网就可以实现。



提示 关于 OpenGL 网上或者书中有很多有用的信息。但你需要记住并不是所有的 OpenGL 工作方式都一样。树莓派的 OpenGL 版本是 ES 2.0，因此你需要确认所做的工作使用的都是这个版本。

6.6 小结

读完本章，你应该更好地理解以下内容：

- OpenGL 为创建和操纵 3D 图形提供了一种强大的方法。
- 然而，与强大相伴的是复杂性。
- 坐标以向量方式存放，你可以通过将向量和变换矩阵相乘来操作它们。
- 如果有多个变换，你要将这些乘法串在一起。然而，相乘的顺序很重要。
- OpenGL 以一个独立的程序运行，它由两个着色器组成——顶点着色和像素着色。
- 这两个着色器不是用 Python 写的，而是由 GL 着色语言或者 GLSL，有点类似于 C 语言。
- 在 GLSL 中，每行都以分号结尾，代码段由花括号包围起来。
- 在 GLSL 中，`uniform` 变量由程序设置并且在整个 `draw()` 函数调用期间保持不变。
`varying` 向量由顶点着色器设置，然后 OpenGL 会对其进行插值。
- OpenGL ES 不包含任何光线，因此你需要根据与光源的距离和法线与光源方向的余弦来自己计算。
- 纹理可以给物体增加细节。

计算机网络与世界 Internet 交织在一起，我们称之为“互联网”。

第 7 章

Chapter 7

Python 与网络

树莓派如何连接到 Internet 上？它如何发送电子邮件？它如何从 Internet 上抓取数据？

当今世界比以前更紧密地连接在一起，几乎你在计算机上做的每件事都有一定形式的联网部分。树莓派也是一样。只要你使用 Model B 树莓派或 USB 无线适配器，就可以非常简单地把树莓派连上互联网。你可以使用 Midori 浏览器浏览网页，还可以使用电子邮件客户端。这些都是用来消费内容的——从网页上获取信息或者使用其他人提供的服务。然而树莓派的强大之处在于创造。写几行 Python 代码，你就可以从网页上抓取信息或者使用树莓派为全世界提供内容和服务。本章将告诉你如何做到这一点。

7.1 理解主机、端口和套接字

为了和其他计算机通信，你需要知道该把数据送到哪里。可能你仅仅将信息传递给同一个房间内的另一台计算机，也可能你需要将信息发送给远在千里之外的另一方。无论如何，你都需要指定一个地址。网际协议（IP）地址通常被用来定位计算机。有两种 IP 地址，版本 4 和版本 6。在本书写作之时，正在大规模使用版本 4（IPv4），因此你读到的都是关于 IPv4 的。IPv6 地址工作方式和 IPv4 基本类似，因此当大规模使用 IPv6 时，你可以轻松地换到 IPv6 上去。

7.1.1 使用 IP 地址定位计算机

IPv4 地址一般的书写法为 4 个用小数点分开的十进制数。想要查看树莓派的 IP 地

址（不止一个地址），只需打开终端（不是 Python 程序，而是 LXTerminal 程序），然后运行 ifconfig。它将会输出如下内容：

```
eth0 Link encap:Ethernet HWaddr b8:27:eb:f3:d5:23
      inet addr:192.168.0.13 Bcast:192.168.0.255
        Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:87523 errors:0 dropped:0 overruns:0 frame:0
      TX packets:59811 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:979997131 (93.4 MiB) TX bytes:12573160 (11.9 MiB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
      UP LOOPBACK RUNNING MTU:16436 Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

可以看到，这里有两个网络接口：eth0 和 lo。eth0 表示有线网络连接，lo 表示回环接口，回环到本主机。本例中，以太网连接使用 192.168.0.13 的 IP 地址，而回环口使用 127.0.0.1（通常都是这个地址）。

除了使用 IP 地址来定位计算机外，我们还可以使用域名（例如 www.google.com，或 localhost）。域名比 IP 地址方便记忆。使用域名时，计算机会首先连接到域名服务器并询问该域名对应的 IP 地址，然后访问域名服务器返回的 IP 地址。localhost 有点特殊，由于它就是 127.0.0.1，代表了本地计算机。

如果把 IP 地址（或者域名）比作现实世界中的楼栋地址，对于同一栋楼来说，你还需要具体到每个人的地址。在计算机网络中，我们使用端口（port）来做到这一点。如果你有一个对外提供服务的软件，它将在特定的端口上监听。客户端就可以连接到该端口上获取信息。有些端口被指定用来提供特定服务。如网络服务通常监听在端口 80 上，SSH 连接使用端口 22。

Python 中的套接字（socket）对象可以用来连接目标主机的目标端口，或者监听在目标端口上等待客户端连接。连接建立之后，就可以使用套接字来收发数据。听起来有点复杂，实际做起来非常简单。理解这个过程最简单的方法就是举例实践。

7.1.2 搭建会话服务器

和本书中大多数程序不同，本程序分为两部分，并且需要同时运行：一个客户端和一个服务器。服务器运行之后处于等待状态，等待连接进来。客户端用来建立连接。下面

服务器端的代码：

```
import socket

comms_socket = socket.socket()
comms_socket.bind(('localhost', 50000))
comms_socket.listen(10)
connection, address = comms_socket.accept()

while True:
    print(connection.recv(4096).decode("UTF-8"))
    send_data = input("Reply: ")
    connection.send(bytes(send_data, "UTF-8"))
```

看得出来，大部分的功能都来自于套接字模块。对于服务器套接字，你首先要把它绑定到主机的某个端口上。（临时使用时，可以随意选择 50 000 以上的端口号。实际上，你可以使用任意当前没被使用的端口号，但最好不要使用 100 以下的端口号，除非你确定没有人使用它。）listen() 使其进入等待状态，等待连接。当有连接进来时，程序跳转到下面一行：

```
connection, address = comms_socket.accept()
```

它会创建一个新套接字（存储在变量 connection 中）。该套接字和客户端连接在一起。现在你就可以通过这个新连接使用 send() 和 recv() 收发数据。数据以字节流而不是字符串的形式存在，因此你需要在字节和 UTF-8（通用字符集转换格式，8 位）之间相互转换。屏幕上显示的信息都是 UTF-8 编码的。[⊖]

下面是客户端代码：

```
import socket

comms_socket = socket.socket()
comms_socket.connect(('localhost', 50000))
while True:
    send_data = input("message: ")
    comms_socket.send(bytes(send_data, "UTF-8"))
    print(comms_socket.recv(4096).decode("UTF-8"))
```

这次，我们不再给套接字绑定主机和端口号，而是直接连接到某个主机和端口。同时也不再创建新的套接字而是使用原有的套接字来收发数据。

除此之外，两个程序非常相似。要发起一个对话，需要两个 Python 会话，最简单的方法是打开两个 LXTerminal 窗口。在第一个窗口中输入 python3 server.py（如果服务器端的代码存储在 server.py 中），在第二个窗口中输入 python3 client.py。

[⊖] 有很多种字符格式，有兴趣的读者可以在互联网上搜索相应信息。——译者注

你将看到信息在两个程序之间反复地传送。当然，这显得不是那么像“网络”。它们实际上运行在同一个主机上。这里只是演示了同一主机上两个程序之间的网络通信，但通常所说的网络指的是在两个主机间传递数据。

这两个程序具备所有在主机间通信的基本条件，但还需要一个菜单让用户可以输入他们想要访问的地址。如果一个程序可以同时处理客户端和服务器会更方便。下面这个程序就是基于这两点的改进版本（参见网站上的 chapter7-chat.py）：

```
import socket

def server():
    global port
    host = "localhost"
    comms_socket = socket.socket()
    comms_socket.bind((host, port))
    print("Waiting for a chat at ", host, " on port ", port)

    comms_socket.listen(10)
    send_data = ""

    while True:
        connection, address = comms_socket.accept()
        print("opening chat with ", address)
        while send_data != "EXIT":
            print(connection.recv(4096).decode("UTF-8"))
            send_data = input("Reply: ")
            connection.send(bytes(send_data, "UTF-8"))
            send_data = ""
        connection.close()

def client():
    global port
    host = input("Enter the host you want to communicate" +
                " with (leave blank for localhost) ")
    if host == "":
        host = "localhost"

    comms_socket = socket.socket()

    print("Starting a chat with ", host, " on port ", port)
    comms_socket.connect((host, port))
    while True:
        send_data = input("message: ")
        comms_socket.send(bytes(send_data, "UTF-8"))
        print(comms_socket.recv(4096).decode("UTF-8"))

    port = int(input("Enter the port you want to communicate on" +
```

```

        " (0 for default))")

if port == 0:
    port = 50000
while True:
    print("Your options are:")
    print("1 - wait for a chat")
    print("2 - initiate a chat")
    print("3 - exit")

    option = int(input("option :"))
    if option == 1:
        server()
    elif option == 2:
        client()
    elif option == 3:
        break
    else:
        print("I don't recognise that option")

```

你应该认识代码中的网络部分，剩下的部分也应该很熟悉了。为了在两个计算机之间通信，需要协商好端口号，设置好监听端的端口号，然后在另一端连接该端口。

该方法还有点小问题。第一点，两个会话端需要交替发送消息，其次你只能在本地网络中和别人通信。实际上，第二点还和你的本地网络是如何搭建的有关。还记得我们说过IP地址有点类似于楼栋地址么？多数情况下都是如此，局域网（LAN）类似于城镇。不同的城镇中两个不同的建筑物可以使用同一个地址“商业街1号”，同样，在不同的局域网中的两台不同主机也可以有相同的IP地址192.168.1.2。有三个不同的IP地址段保留给本地网络使用：

- 10.0.0.0–10.255.255.255
- 72.16.0.0–172.31.255.255
- 192.168.0.0–192.168.255.255

所有这些IPv4地址都只能在本地网络中使用（即你只能和本地网络中的计算机通信），这些IP之外的地址都是公共地址（即所有因特网中的计算机都可以访问）。

有时从外部因特网也可以连接到本地IP地址。这取决于你的因特网提供商或者路由器（参见端口转发port-forwarding设置）。受限于本书篇幅和方向，这里不再展开描述相关内容。

7.1.3 “推”向世界

除了与你的ISP和路由器斗争外，还有许多其他方法可以避开这两个问题。目前为止，最简单的办法就是使用其他服务来帮你处理消息。推特就是这样的一种服务。它可以在单个计算机和全世界之间传递文本消息。

每个推特用户可以一次发送最长 140 个字符的“推特”。如果这些推特中提到其他推友（名字以 @ 开头），该推特将会出现在它们的连接页面上。如果没有推特账号，为完成本章内容，你需要注册一个。即使已经注册过账号，还是推荐你新注册一个以防将不必要的测试信息推送给所有关注你的人。注册账号是免费的，也很简单。只要登录 www.twitter.com 并根据提示一步步做就可以了。注册完成之后，你也可以浏览到网站的更多内容。

通常人们在网站上使用推特发送消息，但这并不是唯一的方法。除了使用网页接口，推特还为从应用程序收发消息提供了应用程序接口（API）。

推特模块为访问推特 API 提供了简便方法。Raspbian 中默认不包含该模块，因此你需要从 <https://github.com/sizohsix/twitter/tree/master> 中下载（使用右下角的下载 ZIP 包按钮下载）。完成之后，打开 LxTerminal 窗口，解压缩并安装模块：

```
unzip twitter-master.zip
cd twitter-master
python3 setup.py build
sudo python3 setup.py install
```

现在已经基本就绪，和推特用户名一样，还需要向推特注册你的应用程序，为你的应用程序取得合适的证书。

登录到推特后，访问 <http://dev.twitter.com>，然后在右上角的用户菜单中选择我的应用程序。在新的页面中点击创建应用程序。在表单中填入应用程序的详细信息。

- 名字在推特中必须是独一无二的，因此不能使用 test-app 这个名字。起个有创意的名字或者在名字后面添加几个特殊字符来保证没有人用过就可以了。
- 名字长度不能超过 10 个字符。
- 网站一栏并不一定指的是真实的网站，只是看起来像而已。我们输入 <http://none.example.com>。
- 回调 URL 可以不填。

除此之外，你还需要接受网站的规则并输入验证码。

应用程序页面有一系列选项卡。我们需要改变默认设置中的一项。选中设置（Setting）选项卡，将应用程序类型由只读改为读写。这样你就可以像读信息一样发布状态了。做完之后，按下更新按钮。这就是推特应用程序的设置。

现在你需要创建访问令牌，转到详情（Details）选项卡，单击创建我的访问令牌。完成之后，推特需要一段更新时间，你可能需要刷新页面，来获取你的访问令牌。一旦出现令牌，所有设置都已经完成。现在需要从网页上记下 4 个信息：访问令牌、访问令牌机密、消费密钥和消费机密。这些都是随机字符串，你需要把它们复制粘贴到下列代码中（参见

网站文件 chapter7-twitter.py):

```
import twitter

def print_tweets(tweets):
    for tweet in tweets:
        print('text: ', tweet['text'])
        print('from: ', tweet['user']['screen_name'])

twitter_user = twitter.Twitter(
    auth=twitter.OAuth("ACCESS-TOKEN", "ACCESS-TOKEN-SECRET",
                        "CONSUMER-KEY", "CONSUMER-SECRET"))

status = twitter_user.statuses

home = status.home_timeline()
print("home")
print_tweets(home)

mentions = status.mentions_timeline()
print("mentions")
print_tweets(mentions)

search_string = input("Enter text to search for,  " +
                      "or press enter to skip: ")
if search_string != "":
    search = twitter_user.search.tweets(q=search_string)
    print("search")
    print_tweets(search['statuses'])

tweet = input("Enter tweet, or press enter to exit: ")

if tweet != "":
    twitter_user.statuses.update(status=tweet)
```

这个迷你推特客户端一定做不到用户友好。事实上，很难想象有人会用它而不是网页来发推特。然而，有很多不同的方式可以与推特交互，你可能希望把它们添加到你自己的应用程序中。改变程序满足你的需求，做起来应该会比较容易。

7.1.4 使用 JSON 做天气预报

上一个例子中，推特模块提供了所有的基本功能。但并不是每次都能找到这样的模块。有时你需要自己写代码完成和网络服务的交互。幸运的是，有一个发送数据往返的标准格式，它可以让你的程序能够轻松地完成与网络服务的交互。这个标准就是基于 JavaScript 语言的轻量级的数据交换格式 JavaScript Object Notation，通常称为 JSON。开始的时候，它是设计出来与 JavaScript（一种主流的网页编程语言）一起工作的，然而它在

Python 上工作得也很好。

OpenWeatherMap.org 是一个免费提供天气预报数据的网站，你可以把它加到软件中。刚好可以使用 JSON。将你的网页浏览器指向 <http://api.openweathermap.org/data/2.5/forecast/daily?cnt=7&units=meteric&mode=json&q=London> 来感受下 JSON 文档是什么样子。它将返回伦敦未来 7 天的天气预报数据。很难看懂，但是你应该注意到它看起来很像包含更多字典列表（其中还有别的东西）的 Python 字典。使用 `url-lib.request` 模块，Python 可以从因特网上获取这些信息，参见下列代码：

```
import urllib.request

url = http://api.openweathermap.org/data/2.5/forecast/" +
      "daily?cnt=7&units=meteric&mode=json&q=London"
req = urllib.request.Request(url)
print(urllib.request.urlopen(req).read())
```

这段代码将从 OpenWeatherMap.org 中抓取信息并将其显示在屏幕上。然而，数据都是字符串形式。虽然它们看起来像字典和列表，你却不能简单地按照字典和列表来存取它们。你需要构建函数读取整个字符串并将其分割开，但非常幸运，你用不着这么做。`json` 模块可以加载它并返回一个含有该字符串每个部分的字典。例如：

```
import urllib.request, json

url = http://api.openweathermap.org/data/2.5/forecast/
      "daily?cnt=7&units=meteric&mode=json&q=London"
req = urllib.request.Request(url)
forecast_string = urllib.request.urlopen(req).read()
forecast_dict = json.loads(forecast_string.decode("UTF-8"))

print(forecast_dict)
```

现在可以从 `forecast_dict` 数据结构中得到任何想要的信息。在下面的例子中，我们编写了一个简单的天气预报程序，它能够打印出指定城市未来 7 天的天气预报（参见网站上文件 `chapter7-weather.py`）：

```
import urllib.request,json

city = input("Enter City: ")

def getForecast(city) :
    url = http://api.openweathermap.org/data/2.5/forecast/ +
          "daily?cnt=7&units=meteric&mode=json&q="
    url = url + city
    req = urllib.request.Request(url)
    response=urllib.request.urlopen(req)
    return json.loads(response.read().decode("UTF-8"))
```

```

forecast = getForecast(city)

print("Forecast for ", city, forecast['city']['country'])

day_num=1
for day in forecast['list']:
    print("Day : ", day_num)
    print(day['weather'][0]['description'])
    print("Cloud Cover : ", day['clouds'])
    print("Temp Min : ", round(day['temp']['min']-273.15, 1),
          "degrees C")
    print("Temp Max : ", round(day['temp']['max']-273.15, 1),
          "degrees C")
    print("Humidity : ", day['humidity'], "%")
    print("Wind Speed : ", day['speed'], "m/s")
    print()
    day_num = day_num+1

```

注意温度的单位是开尔文，将其减去 273.15 就可以将开尔文转换为摄氏。本例仅使用了 API 返回的部分数据。查看一下天气预报数据结构，看看还有其他哪些有用的信息可以输出。

使用这些基本方法，你可以与任何支持 JSON 的 web API 交互。这个地址列出了常用的服务 www.programmableweb.com/apis/directory/1?sort=mashups，从中可以找到方法，让你的应用程序可以从网上获取任意想要的信息。

7.2 知识测验

至此，你已经见过如何向 API 查询并从中获取信息。是时候检验一下你是否完全理解它的工作过程了。试一下下面的练习，遇到任何不确定的问题都可以翻回去参考前面的例子。

练习 1

使用这个 URL 可以得到一个城市（例如伦敦）当前的天气 <http://api.openweathermap.org/data/2.5/weather?q=London>。使用这个 URL 创建一个应用程序将某个地点的当前天气推特出去。参考答案见本章结尾处。

7.3 走向网站

现在你已经见过如何在两个计算机之间传递数据，如何通过在线 API 上发送和接收数据。所有这些都遗漏了网站，毕竟最流行的访问在线信息的方式就是通过网站。你将在本节中学到如何将树莓派作为主机对外提供网页。

网站有两大部分：HTTP 和 HTML。前者是超文本传输协议（Hypertext Transfer Protocol，网络浏览器和网站之间通信时使用的方法），后者是超文本标记语言（Hypertext Markup Language，用来编写网页的语言）。超文本是一个前卫的名字，任何含有链接的文本都是超文本。有很多模块可以处理 HTTP，但你需要先学习一点 HTML 知识。

现代的 HTML 是一种复杂的语言，可以用来创建功能强大的应用程序，其中可以包括包含各种动作和交互操作。然而基本的语言却非常简单。每个网页都是一个独立的 HTML 文件，每个 HTML 文件都分为两部分，文件头和文件体。文件头包含了页面所需的各种信息，文件体包含了需要在屏幕上显示的内容。HTML 使用标签来描述文档的不同部分。几乎所有的标签都成对出现，包括一个起始标签（如 `<h1>` 表示主标题）和一个结束标签（如 `</h1>`）。标签都写在尖括号内，结束标签以斜线开头。下面的例子中使用了常见的基本标签（参见网站文件 chapter7-htmleg1.html）：

```
<!DOCTYPE html>
<head>
An example HTML file</title>
</head>
<body>
<h1>An h1 heading</h1>
<h2>An h2 heading</h2>
<p>
A paragraph of text with a <a href="http://www.raspberrypi.org";link
    <to the Raspberry Pi websites</a>
</p>
<p>
Another paragraph
</p>
</body>
```

除此之外 HTML 还有更多内容，但鉴于本书是一本关于 Python 而不是 HTML 的书，我们不再深入更多细节。了解这些差不多可以满足本章需求。如果你有兴趣了解更多，有很多不错的资源可以学习，这个网站是个不错的开始 <http://w3schools.org>。

保存好文件后可以立即在计算机上打开，它将会在浏览器中打开。这对于检验你的页面是否正确很有用，但这样还不能把它共享给全世界。为了做到这一点，要用到前面提到的 HTTP。在 Python 中，可以非常简单迅速地建立一个 HTTP 服务器。例如：

```
import http.server, os

os.chdir("/home/pi")
httpd = http.server.HTTPServer(('127.0.0.1', 8000),
    http.server.SimpleHTTPRequestHandler)
httpd.serve_forever()
```

它将会在你的树莓派上以 home 目录为根，运行一个使用 8000 端口的网络服务器。

如果你之前关注过端口号，应该还记得 80 是网站服务器的默认端口。我们这里使用 8000 是以防你还运行有其他的网络服务器，如果你喜欢，你也可以把它改成 80。

程序运行之后，可以在树莓派的网络浏览器中输入 `http://localhost:8000/`。`localhost` 通常指向当前运行的主机（它对应的 IP 地址是 127.0.0.1）。:8000 表示使用端口 8000，注意这里不是默认端口 80。你将会在浏览器中看到一个十分普通的文件列表。

HTTP 处理目录和文件的方式与电脑很相似。都是从根目录开始。本例中，根目录是 `/home/pi`。在网络浏览器中你可以指定一个目录（名字以 / 结尾）或者文件名。如果你进入一个目录，网络浏览器首先检查那里是否存在一个名叫 `index.html` 的文件。如果存在，将会显示文件而不是显示目录内容。把你的 HTML 文件改名成 `index.html`，然后用网络浏览器再次访问 `http://localhost:8000`。这次你将会看到文件内容。

7.3.1 让网站动起来（动态网站）

`http.server` 模块可以快速建立一个服务器用于分享信息，但不能很好地应对可变信息。虽然它提供了多种添加交互体验的方法，但还有更好选择。`tornado` 模块被设计来处理动态内容而不是存储在文件中的内容，非常适合那些需要额外的多功能性的项目。

与前一个基于操作系统文件结构的例子不同，Tornado 创建了自己的虚拟结构，并且你需要创建能够输出合适的 HTML 的类而不是直接使用 HTML 文件。首先你需要打开 LXTerminal 并输入下面一行命令来安装它：

```
sudo apt-get install python3-tornado
```

看看下面这个例子。它创建了一个“Hello World！”的网站。

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("<!DOCTYPE html><head><title>" +
                  "Hello world</title></head>" +
                  "<body>Hello World</body>")

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
    ])
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

该例子分为两部分。在第一部分中，我们定义了用于创建网页的类，第二部分（从

`if __name__ == "__main__":` 开始) 定义并启动网站服务。这部分主要包括:

```
application = tornado.web.Application([
    (r"/", MainHandler),
],)
```

这句代码创建了一个 Tornado 网络应用并设置了它的选项。主要的选项是一个元组列表，定义了哪个类处理哪个页面。本例中，只有一个页面，根 (root)，或者 /，由 MainHandler 类来处理。这里还可以添加其他选项，我们在后面的例子中再用。最后两行表示在端口 8888 (这样你就可以同时运行另一个在 8000 上监听的服务 http.server) 上监听，并开始运行。现在运行程序，在网络浏览器中指向 `http://localhost:8888` 检查下它的工作状态。

所有用于处理网页的类都必须继承自 `tornado.Web.RequestHandler`，该类提供了所有的基本功能。你只需要添加一个 `get` 方法来调用 `self.write()` (或者后面将会见到的 `self.render()`)，超类将完成剩下的所有事情。

只定义类而不为它创建任何对象看起来有点奇怪。这是因为 Tornado 使用这个类为你创建了对象。

你可以这样使用 Tornado 来处理静态内容，但不是非常好。所有的 HTML 都包含在 Python 代码中，对于初学者来说有点混乱。Tornado 真正强大的地方体现在你开始创建可以变化的动态网页的时候。下一个例子将给用户带来一个特殊的问候。你可以使用上个例子的文件，但需要根据下列代码创建一个新类：

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self, name):
        self.write("<!DOCTYPE html><head>
                    <title>Hello world</title></head>" +
                    "<body>Hello " + name + "</body>")
```

然后修改后半部分代码以使用这个类 (变化部分使用黑体标出):

```
if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r"/hello/(.*)", HelloHandler),
    ],)
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

修改好后保存并运行代码。在网络浏览器中访问 `http://localhost:8888/hello/Ben`，或者 `http://localhost:8888/hello/Alex`，或者使用你自己的名字。

7.3.2 使用模板

能更改 HTML 代码显然可以让你创造出比以前更强大的网站。然而，这样还是将

HTML 包含在 Python 中，还不够令人满意。解决方案就是使用模板。在 HTML 文档中需要有一点 Python 代码，Tornado 需要使用它来构建网页。下面是 HelloHandler 的模板：

```
<!DOCTYPE html>
<head>
<title>Hello</title>
</head>
<body>
Hello {{ name }}
</body>
```

可以看到，需要打印出来的 Python 变量包围在双层花括号中。将其存为 hello_template.html 文件并保存在你的 home 目录下 (/home/pi)，然后更新 HelloHandler 类：

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self, name_in):
        self.name=name_in
        self.render("/home/ben/hello-template.html",
                   name=self.name)
```

重新运行程序。将会得到相同的结果，但看得出来，这段代码更整洁，也更容易维护。

7.3.3 使用表格回传数据

你可能已经注意到，有些网站需要通过网络地址获取你的输入信息。一般使用 page ID（如你见过的天气 API）来实现。

HTTP 有一个给服务器端回传信息的方法：POST 请求。到目前为止，我们只使用了 Tornado 的 GET 请求（即方法名字）。POST 允许你使用 HTML 表单回传复杂信息。使用下列代码创建一个新的 HTML 文件：

```
<!DOCTYPE html>
<head>
<title>User Information</title>
</head>
<body>
<h1>User Information</h1>
<form action="/hello/" method="post">
Enter your name: <input type="text" name="name">
<input type="submit" value="Sign in">
</form>
</body>
```

将其命名为 user-info.html 保存在你的 home 目录中。然后按照下列代码改变 HelloHandler 类（变化部分以黑体显示）：

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self):
```

```

    self.render("/home/ben/user-info.html")
def post(self):
    self.render("hello-template.html", name = self.get_
        argument("name"))

```

最后一段代码也需要根据新变化作出相应修改：

```

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r"/hello/", HelloHandler),
    ])
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

现在运行代码，再用网络浏览器访问 `http://localhost:8888/hello/`。和前面的结果基本一致，所不同的是现在可以在文本框中输入你的名字。

这是因为当你第一次访问 `http://localhost:8888/hello/` 时，浏览器发送了一个 GET 请求，因此 Tornado 渲染了 `user-template.html`。当你单击 Sign In 按钮时，因为表单中包含 `/hello/` 动作和 POST 方法，浏览器将发送一个 POST 请求（如果动作中不包含服务器名字，浏览器将把请求发送给同一个服务器）。此时 Tornaod 通过 `hello-template.html.self.get_argument()` 捕捉用户输入数据。`name` 就是你在 `user-info.html` 的文本框中输入的名字。

和变量一样，你可以将一些 Python 代码段放到模板中。这些代码包围在 `{%` 和 `%}` 中。最常见的代码是：

```

{% set my_var = 0 %}
{% for x in y %} ... {% end %}

```

可以看出，它和正常 Python 有点不同。在声明变量之前需要加上 `set`，也不再使用缩进（在 HTML 中缩进没有意义），`{% end %}` 表示代码段结尾。

举个例子，`hello-template.html` 可以改成：

```

<!DOCTYPE html>
<head>
<title>Hello</title>
</head>
<body>
{% set char_number = 0 %}
{% for letter in name %}
{{ letter }}<br>
{% set char_number = char_number + 1 %}
{% end %}
{{ name }} has {{char_number}} characters
</body>

```

这虽然不是段很有用的代码，但是它能完成的任务比前面的模板多一点。它将换行显

示名字中的每个字母（
 是 HTML 的换行标签），并且还计算出名字中的字符数。

练习 2

创建一个新的网络应用，用户可以输入城市名，然后得到一个显示该城市未来 7 天天气预报的网页（数据可以从 openweathermap.org 获取）。参考答案见本章结尾处。

7.4 安全

使用 Tornado 开发出越来越强大的网络应用的同时，安全问题迟早会变得重要起来。即使你的程序仅在局域网中运行，还是会有些不希望未授权的人看到的东西。本章的最后一个例子是提供树莓派相关信息的网络应用。

该网络应用在登录系统中使用 cookie。cookie 就是网络应用在浏览器中存储的一些信息。可以使用它来做很多事情，这里我们使用它来追踪登录会话。用户成功登录后，我们根据用户名为他们建立一个安全 cookie。cookie 会在退出登录时清除掉。当用户需要查看网络应用时，程序会检查 cookie，如果没有 cookie，将会重定向到登录页面。程序代码打包存放在网站上的 chapter7-tornado.zip 中。

```
import tornado.ioloop
import tornado.web

users = {"ben": "mypassword"}

class SysStatusHandler(tornado.web.RequestHandler):
    def get(self):
        if not self.get_secure_cookie("user"):
            self.redirect("/login")
            return
        if self.get_argument("type") == "processes":
            com = [["pstree"], ["top", "-bn1"]]
        elif self.get_argument("type") == "system":
            com = [["uname", "-a"], ["uptime"]]
        elif self.get_argument("type") == "syslog":
            com = [["tail", "-n100", "/var/log/syslog"]]
        elif self.get_argument("type") == "storage":
            com = [["df", "-h"], ["free"]]
        elif self.get_argument("type") == "network":
            com = [["ifconfig"]]
        else:
            com = [[["df", "-h"], ["free"], ["uname", "-a"], ["who"],
                    ["uptime"], ["tail", "/var/log/syslog"], ["pstree"],
                    ["top", "-bn1"]]]
        self.render("sysstatus-template.html", commands=com)
```

```

class LoginHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("login-template.html")

    def post(self):
        print(self.get_argument("name"))
        print(self.get_argument("password"))
        if self.get_argument("name") in users.keys() and users[self.get_argument("name")] == self.get_argument("password"):
            self.set_secure_cookie("user", self.get_argument("name"))
            self.redirect("/sysstatus?type=system")
        else:
            self.render("login-fail.html")
class LogoutHandler(tornado.web.RequestHandler):
    def get(self):
        self.clear_cookie("user")
        self.render("logout-template.html")

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/login", LoginHandler),
        (r"/sysstatus", SysStatusHandler),
        (r"/logout", LogoutHandler),
    ], cookie_secret="put your own random text here")
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

可以看到，只用一个称为 type 的页面参数，handler 就可以完成绝大部分的工作（SysStatusHandler）。type 的值可以是 processes、system、syslog、storage、network 或者所有这些的总和。然后它发送一个模板中相应的各种 Linux 系统命令的列表。

HTTP GET 请求中，参数在 URL 的问号之后传入，如 <http://localhost:8888/sysstatus?type=network> 用 network 给参数 type 赋值并传递给 sysstatus 页面。

完成繁重工作的模板是 sysstatus-template.html：

```

<!DOCTYPE html>
<head><title>Raspberry Pi System Status Checker</title></head>
<body><h1>Raspberry Pi System Status Checker</h1>
<a href="sysstatus?type=processes">Processes</a>
<a href="sysstatus?type=system">System</a>
<a href="sysstatus?type=syslog">System Log</a>
<a href="sysstatus?type=storage">Storage</a>
<a href="sysstatus?type=network">Network</a>
<a href="sysstatus?type=all">Display all</a>
<a href="logout">Logout</a>

{% import subprocess %}
{% for command in commands %}

```

```

<h2> Command: {{command[0]}} </h2>
<h2> Output: </h2>
<pre>
{% for line in subprocess.check_output(command).splitlines() %}
{{line}}
{% end %}
</pre>
{% end %}

</body>

```

subprocess 是一个 Python 模块，使用它可以在底层操作系统上运行命令。在树莓派中，操作系统是 Linux。它的输入是一个列表。列表中第一个条目是可执行命令，所有后面的条目都作为命令的参数。因此，列表 ["df",' -h'] 将运行命令 df-h，该命令返回系统的当前磁盘，它的挂载信息和空余磁盘空间。你可以在 LXTerminal 中试一下。实际上，这里提到的所有命令你都可以在 LXTerminal 中试试。

<pre> 标签用来定义预格式化的文本。被包围在 <pre> 标签中的文本通常会保留空格和换行符而不是当做一行处理。

其他的模板都很简单。这个是 login-fail.html：

```

<!DOCTYPE html>
<html>
<head>
<title>Login Fail</title>
</head>
<body>
Your login has failed. Please click <a href="login">here</a> to try again.
</body>

```

这个是 login-template.html：

```

<!DOCTYPE html>
<head>
<title>Raspberry Pi System Status Login Form</title>
</head>
<body>
<p>Please enter your login information</p>
<form action="/login" method="post">
Username: <input type="text" name="name">
<br>
Password: <input type="password" name="password">
<br>
<input type="submit" value="Sign in">
</form>
</body>

```

最后，这个是 `logout-template.html`。

```
<!DOCTYPE html>
<html>
<head>
<title>Login Fail</title>
</head>
<body>
You have logged out. Please click <a href="login">here</a>
if you wish to log in again.
</body>
```

我们这里创建的登录方法提供了一定的安全性，但还远不够安全。如果你创建个处理敏感信息的网络应用，或者允许用户通过控制树莓派但又不信任所有的用户，你需要适当地研究下网络安全。这个话题太过庞大，我们在这里无法展开讨论，但你可以从开放的网站应用程序的项目安全（Open Web Application Security Project，参见 www.owasp.org）开始学习。

7.5 小结

读完本章，你应该能够更好地理解以下内容：

- 使用主机地址和端口号可以在两个计算机之间相互传递数据。
- 主机地址可以是 IP 地址，也可以是域名。
- 你需要使用套接字来连接到其他计算机并通过套接字来发送数据。
- 作为接口程序，API 可以访问网络服务器及其数据。
- 推特 API 提供了一个模块，使用它可以轻松获取和操作 twitter.com 上的信息。
- API 使用一种统一的数据格式 JSON，可以方便地在应用程序之间解析和共享数据。
- 网页由 HTML 编写，由 HTTP 对外提供服务。
- `http.server` 可以用来创建非常简单的网络服务器。
- 可以使用 Tornado 创建更复杂的网络应用。
- Tornado 模板使得在 HTML 中处理复杂、变化的信息变得更容易。
- 服务器可以使用 HTML 表单和 POST 请求接收信息。
- cookie 允许你在客户端浏览器存储一些信息，如用户名。

7.6 习题答案

练习 1

这是个将某个地方的天气预报推特出去的示例程序。

```

import twitter, urllib.request, json

twitter_user = twitter.Twitter(
    auth=twitter.OAuth("1824182228-
1HOzvEpNA31LTiUCHkLSiqqW5Pbe7BvJKvKT2H6",
    "AwYIcpfRFUt6F4hMoGfqMGINDiUrW49R1mjVqY6Bts",
    "xStMTHb0HQZaEFLi2bsWA",
    "YgFhiCpv1qLLe5Is5dRAZWNz1T84KnyZCMKfXIwN8"))

city = input("Which city? ")
url = "http://api.openweathermap.org/data/2.5/weather?q="
url = url+city
req = urllib.request.Request(url)
forecast_string = urllib.request.urlopen(req).read()
forecast_dict = json.loads(forecast_string.decode("UTF-8"))

tweet_text = city + " weather is ' +
forecast_dict['weather'][0]['description']

twitter_user.statuses.update(status=tweet_text)

```

练习 2

有很多方法可以做到，我们使用三个文件。第一个是一个名为 city-info.html 的模板，包含以下内容：

```

<!DOCTYPE html>
<head>
<title>Weather</title>
</head>
<body>
<h1>Which City?</h1>
<form action="/weather/" method="post">
Enter a city: <input type="text" name="city">
<input type="submit" value="Submit">
</form>
</body>

```

第二个名为 weather-template 的模板包括如下内容：

```

<!DOCTYPE html>
<head>
<title>Weather</title>
</head>
<body>
{%
  set day_num = 0
}
{%
  for day in forecast
}
<h2>Day : {{str(day_num)}} </h2>
<h3>{{day['weather'][0]['description']}} </h3>
Cloud Cover : {{str(day['clouds'])}} <br>
Temp Min : {{str(round(day['temp']['min']-273.15, 1))}}

```

```

degrees C<br>
Temp Max : {{ str(round(day['temp']['max']-273.15, 1)) }}}
degrees C<br>
Humidity : {{str(day['humidity'])}} %<br>
Wind Speed : {{str(day['speed'])}}m/s<br>
{ % set day_num = day_num + 1 %}
{ % end %}
</body>

```

第三个 Python 文件包括：

```

import tornado.ioloop
import tornado.web
import subprocess
import urllib.request, json

class WeatherHandler(tornado.web.RequestHandler):
    def get(self):
        self.render("/home/ben/city-info.html")

    def post(self):
        url = "http://api.openweathermap.org/data/2.5/forecast/"
        daily?cnt=7&units=metric&mode=json&q="
        + self.get_argument("city")
        req = urllib.request.Request(url)
        response = urllib.request.urlopen(req)
        self.forecast = json.loads(response.read().decode("UTF-8"))
        self.render("weather-template.html",
                   forec     ast = self.forecast['list'])

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/weather/", WeatherHandler),
    ])

    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

我的世界

给不了解的人介绍一下，《我的世界》(mine craft) 是一个生存游戏，玩家可以在周围世界里采矿获取物资建造建筑物。游戏的世界由方块(或方格)组成，每个方块都可以被摧毁，摧毁后会露出隐藏的财富。图 8-1 给出了一个典型的我的世界的界面。

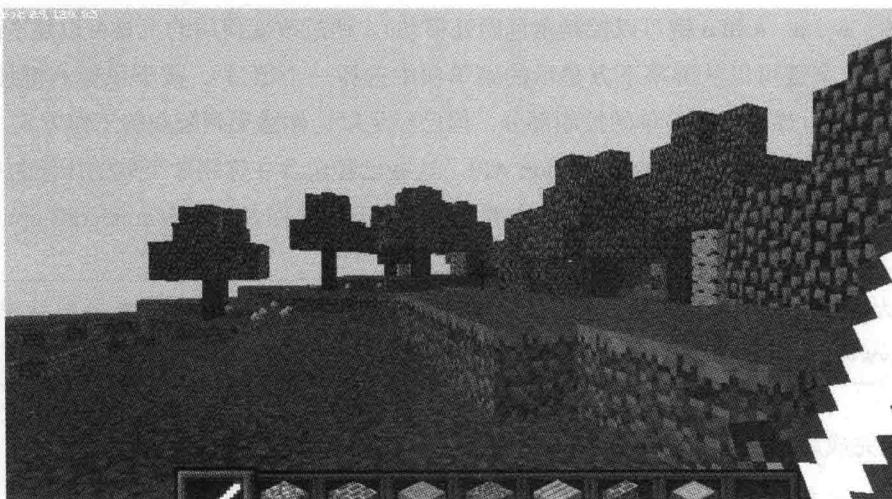


图 8-1 3D 方块成为这个游戏的标志

这是一个相当粗糙的描述，只是介绍个概念让你大致明白它是做什么的。它的完整版本可以运行在大多数计算机上，针对树莓派它也有个特殊版本。树莓派版本的特殊之处在

于你可以用 Python 控制游戏。你不仅可以操作角色本身，还可以操作你身边的整个世界。

8.1 畅游我的世界

这些介绍已经足够了。有些事物必须要亲自看一下才能理解，我的世界就是其中的一个。首先，你需要从网上下载软件 <http://pi.minecraft.net>。

要下载的最后一个文件是 `minecraft-pi-0.1.1.tar.gz`。使用下面的命令行在终端中解压缩文件：

```
tar zxvf minecraft-pi-0.1.1.tar.gz
```

现在可以使用下面的命令打开游戏：

```
mcpi/minecraft-pi
```



注意 在玩我的世界时，它会接管鼠标和键盘的控制。如果你想切换回其他非我的世界窗口，只需按 `Alt+Tab` 组合键，游戏就会释放对鼠标和键盘的控制。

8.1.1 控制我的世界

首先要做的是开始新游戏，并尝试在世界中移动（使用鼠标改变朝向，鼠标左键可以选择物体，`w`、`a`、`s` 和 `d` 键可以控制角色四处移动）。通过改变使用的工具可以建造建筑物（按下某个数字键可以从屏幕下方显示的菜单项中选择一个项目），按下鼠标右键使用它。这些只是我的世界中非常简单的控制部分，但已经足以让你感受到游戏的工作方式。

为了控制世界，你需要使用 Python API。这些已经包含在你刚才下载的压缩包中，但这个版本不支持 Python3。因此你需要从本书网站中下载文件 `chapter8-minecraft-api.tar.gz`。



注意 大段代码的例子和其他 API 以及本书中提到的下载都是指从本书网站中下载：www.wiley.com/go/python-raspberrypi。

下载完成后，使用下面的命令行解压缩：

```
tar zxvf chapter8-minecraft.tar.gz
```

使用 `cd` 命令切换当前目录进入新文件夹。所有本章要创建的程序都应该放在这个文件夹中，这样它们才能使用 Python API：

```
cd mcpi
```

你要写的程序不会运行一个新的我的世界，而是连接到当前在树莓派上运行的我的世界的一个进程上。

认识到这点之后，打开一个新的 Python 解释器。请记住必须转到文件夹 chapter8-minecraft 下面。最简单的办法就是打开 LXTerminal 会话，转到 chapter8-minecraft 文件夹下面（使用前面的 cd 命令），然后执行 python3.

下列代码将连接到当前运行的我的世界的一个进程：

```
>>> import minecraft
>>> mc = minecraft.Minecraft.create()
```

现在可以操作一个称为 mc 的我的世界对象。这个对象有一个可以移动的 player 对象，试试这个命令：

```
>>> mc.player.setPos(10,10,10)
```

你会发现角色移动了。角色移动到坐标 (10,10,10) 处，但由于世界是随机生成的，这个点可能在空中，这种情况下，角色将掉落到地板上。或者，他也可能出现在水底或地下。改变坐标将其移动到地面上的某点。坐标中第一个和最后一个数字是角色在地面上的位置 (x 和 z)，中间的值 (y) 表示垂直高度。

8.1.2 用 Python 创建我的世界

选好角色位置之后，你可以使用 setBlock (x,y,z,type) 方法改变角色周围的世界。x、y、和 z 表示角色周围的世界坐标，和角色坐标的格式一样。type 表示方块的 typeID。每个不同类型的方块都有一个 typeID，你需要使用它们来操纵世界。

例如，当角色在 (10,15,22) 这个位置时，你可以用下列命令在他边上建造一个蘑菇：

```
>>> mc.setBlock(11,15,22, 40)
```

表 8-1 给出了一个有用的方块类型列表

表 8-1 一些有用的类型 ID

| ID | 方块类型 | ID | 方块类型 |
|----|----------|-----|---------|
| 0 | 空气 | 1 | 石头 |
| 2 | 草地 | 3 | 泥土 |
| 4 | 鹅卵石 | 5 | 厚木板 |
| 6 | 树苗 | 8 | (流动的) 水 |
| 10 | (流动的) 岩浆 | 18 | 树叶 |
| 22 | 天青石 | 40 | 蘑菇 |
| 41 | 金块 | 246 | 发光的黑曜石 |

构建世界

至此，你已经能够在自己的“我的世界”中工作了。然而，如果希望创建一个可重用的程序，你需要所有东西的起始位置。为了做到这一点，你需要按自己的想法创建世界。下列脚本将世界中心的一个 200×200 方形区域设置成光滑的草地（类型 2），草地上方是空气（类型 0）。

```
import minecraft
mc = minecraft.Minecraft.create()
mc.setBlocks(-100, -1, -100, 100, 0, 100, 2)
mc.setBlocks(-100, 1, -100, 100, 100, 100, 0)
```

代码中使用了 `setBlocks(x1,y1,z1,x2,y2,z2,type)` 方法，它将 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) 之间的所有方块设置成相应的类型。

绘制画面

现在你已经知道如何设置方块，让我们开始用它来做一些有意义的事情。在完整版的我的世界中，你可以用它来建筑一个超级堡垒作为角色的避难所。然而，树莓派上运行的并不是完整版本。你可以使用这个版本来创作些美术作品，如使用方块画个图。下列代码就是如此（参见网站上的 `chapter8-art.py`）：

```
import minecraft
import sys

mc = minecraft.Minecraft.create()

height = 10
start_x = 0
start_y = 0
canvas = "horizontal"

r = 246
g = 18
b = 22

picture = [[["R", "R", "R"],
            ["G", "G", "G"],
            ["B", "B", "B"]]

if len(sys.argv) > 1:
    with open(sys.argv[1]) as f:
        picture = f.readlines()

pic_x = start_x
for line in picture:
```

```

pic_y = start_y
for block in line:

    if canvas == "horizontal":
        x = pic_y
        y = height
        z = pic_x
    else:
        x = height
        y = pic_y
        z = pic_x
    if block == "R":
        mc.setBlock(x,y,z,r)
    if block == "G":
        mc.setBlock(x,y,z,g)
    if block == "B":
        mc.setBlock(x,y,z,b)

    pic_y = pic_y + 1
    pic_x = pic_x + 1

```

这和第4章中的加载关卡工作方式类似。代码使用 `sys.argv` 来检查用户是否在命令行中指定了文件名。如果指定了，它将使用该文件来绘制图片，否则，使用默认值，画出三条线。

有两个 `for` 循环来遍历每一行的每一个字符。循环中如果遇到 R、G 或 B，就相应地画出一个红色、绿色或者蓝色方块。

然而，你在世界中没有完全的控制权，只能创建一些预定的方块类型，还有并不是所有的方块都可以漂浮在空中。本例中使用型号 246 的方块（发光的黑曜石）表示红色，型号 18（树叶）表示绿色，型号 22（天青石）表示蓝色。当然，你也可以将其换成任何你想要的方块创造出自己的风格。

代码中的坐标有点复杂，因为它有两个模式——垂直和水平——表示了不同绘图方向。正因为如此，他需要处理两组坐标：关于图片的 2D 坐标和关于我的时间的 3D 坐标。`if` 语句根据指定的方向在这两者之间作出选择。

注意，程序将忽略非 R、G 或 B 的所有字符，因此，你可以在图片中包含这些字符，程序将使用空格表示它们。例如，下列文字将创造出一个笑脸图标（见图 8-2）：

```

-----
-G---G-
-G---G-
---B---
-R---R-
--RRR--

```

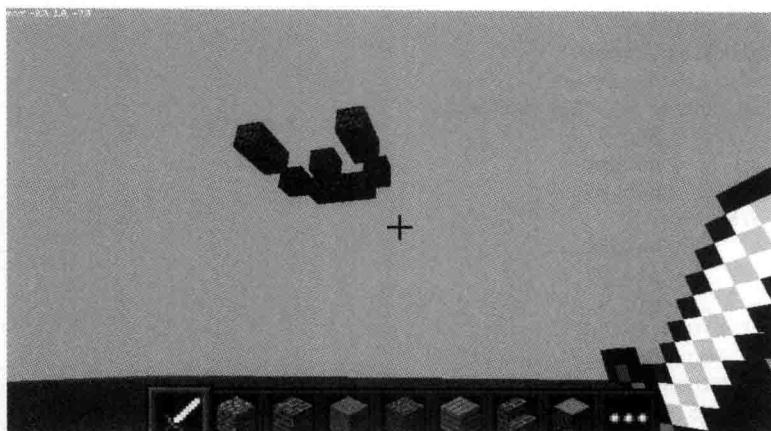


图 8-2 笑脸只是一个开始，你可以使用这个方法在我的世界中画出任意你想要的图形

8.1.3 深入探索

还有很多方法来扩展这个世界。最简单的方法就是将颜色托盘从 3 个扩展到和可供选择的角色和方块类型一样多。稍微复杂的方法是创建一个 3D 图形。这需要你适当地改变下文件结构。我们推荐使用多个 2D 图形组在一起，使用一个特殊行将它们分开。例如，下面这个两层厚的笑脸：

```
-----
-G---G-
-G-B-G-
-----
-B-----
-R---R-
--RRR--
*
-----
-----
```

*

```
-G---G-
-----
```

还可以使用相同的文件结构构建动画，每个组的方块代表一帧。

8.2 制作贪吃蛇游戏

3D 图形的一个主要应用就是制作游戏。我的世界本身就是游戏，你也可以使用它作为游戏引擎构建更多的游戏。在本节中，你将会看到如何制作经典游戏——贪吃蛇。

如果你不熟悉这个游戏，这里简单介绍一下。在游戏中，你控制蛇四处移动吃掉苹果。每吃掉一个苹果，蛇增长一节。如果它碰到自己或者围墙，就挂了。游戏的目标就是尽可能让蛇变得更长。在 iPhone 出现之前，这是个在手机上非常流行的游戏。



注意 本书中出现的大段程序都可以在本书网站上下载 www.wiley.com/go/pythonraspi。

为了避免不必要的输入错误，你可以从网站上下载代码，复制和粘贴到你的 IDE 或代码编辑器中。下面例子的代码参见 chapter8-snake.py

游戏代码如下所示（参见网站上的文件 chapter8-snake.py）。

```
import minecraft
import pygame
import sys
import time
import random

from pygame.locals import *

mc = minecraft.Minecraft.create()
pygame.init()

#blank world
print("Resetting world")
mc.setBlocks(-100,-1,-100, 100, 0, 100, 2)
mc.setBlocks(-100,1,-100, 100, 100, 100, 0)
mc.player.setPos(0,2,0)

#create control window
display = pygame.display.set_mode((200,200))

#draw maze
print("drawing maze")
height = 10

start_x = 0
start_z = 0

difficulty = 0.5
apple_freq = 2

picture = [[ "R", "R", "R", "R", "R", "R", "R", "R", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"],  

           [ "R", "-", "-", "-", "-", "-", "-", "-", "R"]]
```

```

["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "-", "-", "-", "-", "-", "-", "-", "R"],
["R", "R", "R", "R", "R", "R", "R", "R"]
}

if len(sys.argv) > 1:
    with open(sys.argv[1]) as f:
        picture = f.readlines()

posn_z = start_z
for line in picture:
    posn_x = start_x
    for block in line:
        x = height
        y = posn_x
        z = posn_z

        if block == "R":
            mc.setBlock(x,y,z,246)

        posn_x = posn_x + 1
    posn_z = posn_z + 1

snake = [(int((posn_z - start_z)/2),int((posn_x - start_x)/2))]
movement = [-1,0]

finished = False

grow_in = []
apple_in = apple_freq

while not finished:
    for event in pygame.event.get():
        if event.type == QUIT:
            finished = True

        key_state = pygame.key.get_pressed()
        if key_state[K_LEFT]:
            movement = [-1,0]
        if key_state[K_RIGHT]:
            movement = [1,0]
        if key_state[K_UP]:
            movement = [0,1]
        if key_state[K_DOWN]:
            movement = [0,-1]

```

```

movement = [0, -1]

next_position = (height, snake[0][1] + movement[1],
                 snake[0][0] + movement[0])
next_block = mc.getBlock(next_position[0],
                        next_position[1], next_position[2])

if next_block == 246 or next_block == 22:
    finished = True

if next_block == 18:
    grow_in.append(len(snake))
    apple_in = apple_freq

snake.insert(0, (next_position[2], next_position[1]))
for block in snake:
    mc.setBlock(height, block[1], block[0], 22)

if 0 not in grow_in:
    remove_block = snake.pop()
    mc.setBlock(height, remove_block[1], remove_block[0], 0)

grow_in2 = []
for number in grow_in:
    if number > -1:
        grow_in2.append(number - 1)
grow_in = grow_in2

if apple_in == 0:
    apple_y = random.randint(1, int(posn_x - start_x) - 1)
    apple_x = random.randint(1, int(posn_z - start_z) - 1)
    while mc.getBlock(height, apple_y, apple_x) != 0:
        apple_x = random.randint(1, int(posn_z - start_z) - 1)
        apple_y = random.randint(1, int(posn_x - start_x) - 1)
    mc.setBlock(height, apple_y, apple_x, 18)

apple_in = apple_in - 1

time.sleep(difficulty)

mc.postToChat("Game Over")

```

这里有两个之前没有用过的 Minecraft 方法：getBlock() 和 PostToChat()。这两个方法都很好地自注释了本身。第一个返回指定位置的方块类型，第二个在玩家屏幕上显示字符。

你应该能认出第一部分的代码，因为它和之前的那个画图代码几乎一模一样。因为这里只需要一个颜色来表示限制贪吃蛇移动的迷宫，我们做了一些简化（我们选择了红色，你可以替换成任意一种你喜欢的颜色）。

代码的第二部分（包括 while 循环和它上面的 5 行）包含了游戏的运行机制。贪吃蛇

存储在一个元组的列表中，每个元组包含了一节贪吃蛇的 2D 坐标。这些一起组成了 3D 坐标并显示到屏幕上。

8.2.1 移动贪吃蛇

移动贪吃蛇分为两部分。首先，在开始位置增加一个新的方块。贪吃蛇的最前方的方块坐标加上移动变量就是新方块的位置。其次，使用 `pop()` 将贪吃蛇的最后一个方块从列表中移除。`pop()` 方法将方块从列表中去除并将该方块作为返回值返回。这个坐标方块将被设置为 0（空气）。

关键的控制部分和第四章中的一样。使用了 PyGame。唯一的细微差别就是为了能让它工作，就必须有一个 PyGame 窗口，并且为了捕获按键动作，键盘焦点必须在这个窗口中。程序运行时，你会看到有个 200×200 的空窗口，并且要想控制贪吃蛇就必须单击该窗口。

8.2.2 增长贪吃蛇

可能代码中最复杂的部分就是当贪吃蛇吞下苹果时如何增长它。这是因为贪吃蛇吃下苹果之后并不是立即变长的。而是当它的整个身体都通过苹果之后才增长。

为了控制这个过程，这里使用了一个叫做 `grow_in` 的列表。每当贪吃蛇吃下一颗苹果，他的长度就会添加到这个列表中。主循环每运行一次，将列表中的每个数都减一，负数丢弃。这就意味着，当整个贪吃蛇通过苹果之后，列表中只剩下 0。列表里有非零数字时，贪吃蛇的末端部分被移除。实现部分参见下面一段代码：

```
if 0 not in grow_in:
    remove_block = snake.pop()
    mc.setBlock(height, remove_block[1], remove_block[0], 0)

grow_in2 = []
for number in grow_in:
    if number > -1:
        grow_in2.append(number - 1)
grow_in = grow_in2
```

8.2.3 添加苹果

剩下要做的就是添加苹果。贪吃蛇吞下前一个苹果之后经过 `apple_freq` 次循环，新苹果会立即出现。苹果出现的位置是随机的，参见下面的代码：

```
if apple_in == 0:
    apple_y = random.randint(1,int(posn_x - start_x)-1)
    apple_x = random.randint(1,int(posn_z - start_z)-1)
```

```

while mc.getBlock(height, apple_y, apple_x) != 0:
    apple_x = random.randint(1,int(posn_z - start_z)-1)
    apple_y = random.randint(1,int(posn_x - start_x)-1)
    mc.setBlock(height, apple_y, apple_x, 18)

```

这段代码先产生一个随机位置，然后检查下该位置是否已经被占用（如果被占用，产生一个新的随机位置）。apple_in 变量用来统计上个苹果被吃掉之后的循环次数。

从图 8-3 的游戏截图中可以看到游戏的效果，也可以自己运行一下看看。

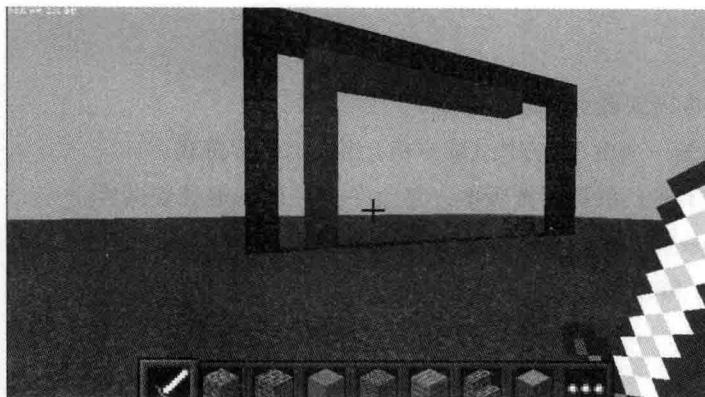


图 8-3 通过改变地图可以改变游戏的难以程序。只要注意不要在围墙上留缝隙以防贪吃蛇逃走，否则将无法在我的世界中继续控制贪吃蛇

8.3 深入探索

和第 4 章的游戏相比，这个游戏还没有结束，我们将剩下的工作留给你。有很多进一步完善的方法，包括：

- 增加更多的图形。例如，贪吃蛇挂掉时候添加一个爆炸效果，或者在开始的时候增加一个倒计时。还可以将贪吃蛇图形变得更引人注目点，比如随着长度的增加添加一些其他方块。
- 计分。可以是贪吃蛇的长度。
- 改变难度系数（在循环中使用 time.sleep()）。这个可以和分数联系起来。贪吃蛇越长，游戏速度可以越快。
- 惊喜。例如，一旦达到一定长度，贪吃蛇会改变颜色。
- 奖励。短暂地在地图上出现，如果贪吃蛇吃到了，玩家会得到加分。
- 更多关卡。可以设计不同尺寸的地图，或者在地图内部添加一些额外的墙增加游戏难度。

不用把自己限定在完善贪吃蛇上。可以使用这些方法创作新游戏。例如，相同的游戏机制可以用在电子世界争霸赛（Tron）中（改编自同名电影。使用 apt-get xtron 安装一下试试看看）。

甚至还可以把它像第 4 章中的一样改造成平台，或者其他任何你想要的。唯一束缚你的是你的想象力。

8.4 小结

读完本章，你需要能更好地理解以下内容：

- 我的世界是一个由 3D 方块（或方格）组成的生存游戏。
- 我的世界有专门的树莓派版本，它允许使用 Python 控制世界。
- 使用 player.setPos (x,y,z) 可以移动角色。
- setBlock() 和 SetBlocks() 可以用来建设世界。

多 媒 体

我们在前 8 章中，大量介绍了处理数据并把它从树莓派中输出的方法。然而到现在为止还没有介绍过多少向树莓派输入数据的方法。前面提到过的一些按下按钮，以及与鼠标相关的操作指的就是这部分。在本章中，我们将会扭转这种情况。给你的树莓派加上两个附加传感器——话筒和摄像头，让它以一种更自然的方式与外界交互。我们将会分别介绍它们，并告诉你在 Python 程序中如何使用它们。

9.1 使用 PyAudio 让计算机发声

一旦将声音导入计算机，你就可以做各种各样的事情，但这些都要从同一件事情开始——输入声音。要想快速、简单地将声音输入计算机，PyAudio 模块是所有工具中最棒的一个。录好声音之后，你就可以任意操作它，但是如果要在其他程序中使用它，就需要将其转换成其他软件可以识别的格式。WAV 是一种简单无损格式，非常适合简单的音频数据（如果记录了大量数据，WAV 格式的文件会非常巨大）。

另外，你还需要一个 USB 话筒用来收集声音。小巧的树莓派只有声音输出接口。

和平时一样，首先要做的事情就是使用下列命令安装模块：

```
sudo apt-get install libportaudio0 libportaudio2 libportaudiocpp0  
libportaudio19-dev python3-setuptools python3-pip  
sudo pip-3.2 install pyaudio
```

现在让我们来录音！代码如下（参见网站文件 chapter9-record.py）：

```

import pyaudio
import wave

def record_sound(seconds, chunk_size, sample_rate, filename,
                 channels, format_type):
    p = pyaudio.PyAudio()

    stream = p.open(format=format_type,
                    channels=channels,
                    rate=sample_rate,
                    input=True,
                    frames_per_buffer=chunk_size)

    print("Speak now")

    frames = []

    for i in range(0, int(sample_rate / chunk_size * seconds)):
        data = stream.read(chunk_size)
        frames.append(data)
        if i%int(sample_rate/chunk_size) == 0:
            print(seconds - round(i/(sample_rate/chunk_size)),
                  "seconds remaining")

    print("Finished")

    stream.stop_stream()
    stream.close()
    p.terminate()

    wf = wave.open("filename", 'wb')
    wf.setnchannels(channels)
    wf.setsampwidth(p.get_sample_size(format_type))
    wf.setframerate(sample_rate)
    wf.writeframes(b''.join(frames))
    wf.close()

    chunk_size = 1024
    sample_rate = 44100
    seconds = 15
    filename = "output.wav"
    channels = 1
    format_type = pyaudio.paInt16
    record_sound(seconds, chunk_size, sample_rate, filename, channels,
                 format_type)

```

看得出来，有很多属性会影响录制，你可以通过修改它们来改变录制过程。它们中有些是用来控制数据如何采集和保存的（例如，`chunk_size` 和 `sample_rate`），其他的是一些高级属性（如录音的时长，使用的音轨数量）。目前，我们就固定使用这些默认值，除非你很有把握再调整它们。有些属性受硬件限制，试图改变它们会引起 PyAudio 异常。

9.1.1 录音

`record_sound()` 函数完成了所有繁重的工作（把它作为内联代码而不是函数也可以正常工作，这里我们将其定义成函数是为了方便在其他程序中使用）。第一行和第二行创建了一个新的 PyAudio 对象，然后使用它来打开流。我们就是从这个流中获取的声音数据。

录音的主要工作由下面的 for 循环完成：

```
for i in range(0, int(sample_rate / chunk_size * seconds)):
    data = stream.read(chunk_size)
    frames.append(data)
```

这段代码把数据保存在给定大小的 `chunk` 中。`chunk` 的数据类型是 `byte`，但你不用关心这些。

现在已经得到了以字节串形式保存的声音数据，还需要再做些处理。最简单的操作就是使用 `wave` 模块将其保存为 WAV 文件，这也是本程序后半部分所做的工作。

运行程序后会看到一个错误提示。不用担心，这只是 PyAudio 的工作方式，用来告诉你程序运行一切正常。（当 PyAudio 在设备上检测到音频设备时会抛出这个提示。只要不属于任何形式的 Python 异常，一切都会正常运行）。

之后，会在程序运行的目录下得到一个名为 `output.wav` 的文件。这是一个 15 秒长的录音，可以使用任何音频软件回放。

9.1.2 向树莓派讲话

你使用自己录下的声音做任何操作，其中最酷的，不需要任何音乐天分的操作就是语音控制。要做到这一点，你需要找到一种能够从声音中获取谈话内容的方法，这绝对是个有难度的工作。

不幸的是，任何一个模块都无法轻松地完成这项工作。然而，正如你在第 7 章中见到的那样，Python 模块并不是在程序中实现附加功能的唯一方法。网络服务是另外一个很好的特性来源，并且可以轻松地加入程序中。

这里使用了 Google 提供的一个非常出色的语音转文字的网络应用。不过它还存在一些限制。首先，每段语音长度不能超过 15 秒。其次，音频片段必须以采样率 16 000 的 FLAC 格式上传。

对于语音控制系统来说第一个限制不会引起什么问题（听写程序除外）。然而第二个限制有点麻烦，因为 Python 还没有标准 FLAC 模块。虽然有一些选项，但没有一个能够很好地满足我们的需求。这里我们可以转过来寻找应该一个附加功能的来源（将在下

一章中详细介绍)——Linux 命令行。你可以在 Python 代码中运行并控制普通的 Linux 程序。

实现语音转换成文字(前提是使用 FLAC 格式, 格式转换稍后介绍)的代码如下所示:

```
def google_speech_recognition(filename):
    global url

    audio = open(filename, 'rb').read()
    http_header={'Content-Type': 'audio/x-flac; rate=16000'}
    request = urllib.request.Request(url, data=audio,
                                      headers=http_header)

    response = urllib.request.urlopen(request)
    out = response.read()
    json_data = json.loads(out.decode("utf-8"))

    return json_data
```

读过第 7 章, 就会非常熟悉这段代码, 因此我们不再详细介绍。这里唯一的新事物就是调用 `urllib.request.Request()` 时的附加参数。参数 `data`, 非常明显, 是用来传送数据的, 本例中就是音频文件。参数 `headers` 用来发送 HTTP 头。`headers` 告诉网络服务谁在请求数据, 以及它们在发送什么数据。在这个例子中, 我们告诉服务器, 在向他发送采样率为 16 000 的 FLAC 格式的音频。它不会直接返回文字, 而是返回一个 JSON 编码的数据, 其中包括文字和文本正确性的证据(如果语音中存在不同的口音和部分俚语, 并不能保证结果完全正确)。如果想知道更多返回内容, 可以向下面一样添加一行打印:

```
print(json_data)
```

在第 7 章中我们详细介绍了该结构体。

处理文字比处理语音要简单得多, 因此现在可以将这些内容整合到软件中。不过这个例子还没有完成, 我们还要进一步扩展它——向它提问, 如果幸运, 它可以回答你。

9.1.3 向程序提问

给程序添加回答问题的特性很有挑战性, 并且编程实现也是一个艰巨的任务。幸运的是我们可以从其他领域得到一点帮助。Wolfram Alpha 是一个试图完成这项工作的网络服务。可以使用网络浏览器访问 www.wolframalpha.com 来体验一下。也许你已经猜到了, 它也提供了给应用程序使用的 API。

同样的, 如果读过第 7 章, 就会很熟悉下面的代码:

```
def wolfram_alpha(speech):
```

```

url_section = urllib.parse.urlencode(dict(
    input=speech,
    appid=wolfram_alpha_app_id,
))

url = http://api.wolframalpha.com/v2/query?' + url_section
response = urllib.request.urlopen(url)

tree = ElementTree.parse(response)
root = tree.getroot()

for node in root.findall('.//pod'):
    print(node.attrib['title'])
    for text_node in node.findall('.//plaintext'):
        if text_node.text:
            print(text_node.text)
    print("*****")

```

你可能不理解后半部分。Wolfram Alpha 的返回结果是 XML 文件。代码中使用 `xml` 模块中的 `ElementTree` 解析文件并取出相应信息。由于和本章内容不相关，我们在此不作详细介绍。如果以后你需要了解相关信息，可以在 `xml` 模块的文档中找到更多信息。



注意 运行这段程序之前，需要先获得一个 Wolfram Alpha 应用 ID。为此，你需要到 <https://developer.wolframalpha.com/portal/signup.html> 注册账号。获得账号之后将其复制到程序中的适当位置。

9.1.4 组合包装

现在可以将这三个功能（录音、转换成文字、回答提问）合并到一起创建一个语音识别程序（我们称为 Pyri，类似于一个基于移动平台的知名应用）。下面给出了代码（参见网站上的 `chapter9-pyri.py`）。前面讨论过的函数这里不再重复，但是你需要在程序中包含它们。



注意 本书中出现的大段程序都可以在本书网站上下载 www.wiley.com/go/pythonraspi。为了避免不必要的输入错误，你可以从网站上下载代码，复制和粘贴到你的 IDE 或代码编辑器中。下面例子的代码参见 `chapter9-pyri.py`。

```

import urllib.request
import json
import pyaudio

```

```

import wave
from xml.etree import ElementTree
import subprocess
import time

def record_sound(seconds, chunk_size, sample_rate, filename,
                 channels, format_type):
    #Code give above

def google_speech_recognition(filename):
    #Code given above
def wolfram_alpha(speech):
    #Code give above

chunk_size = 512
sample_rate = 44100
seconds = 5
filename = "output.wav"
channels = 1
format_type = pyaudio.paInt16

url = "https://www.google.com/speech-api/v1/recognize?"
      + "client=chromium&lang=en-US"

wolfram_alpha_app_id = "PUT-YOUR-APPID-HERE"

record_sound(seconds, chunk_size, sample_rate,
             filename, channels, format_type)

subprocess.call(["sox", "output.wav", "-r16k", "-t",
                 "wav", "output-16k.wav"])
subprocess.call(["flac", "-5", "-f",
                 "output-16k.wav"], stdout=subprocess.PIPE)
time.sleep(2)

try:
    google_data = google_speech_recognition("output-16k.flac")
except urllib.error.HTTPError:
    print("voice recognition failure")
else:
    print(google_data)

    if google_data['status'] == 0:
        wolfram_alpha(google_data['hypotheses'][0]['utterance'])
    else:
        print("Voice recognition failed. Try again")

```

由于最后一个 try 语句包含一个 else 子句，你可能不太熟悉。它表示只要 try 语句中的代码不抛出异常，就会执行 else 这个代码段。本例中就意味着只要 Google 请求不抛

出 `HTTPError` 异常。

运行程序前，需要先确认已经安装好下面这两个模块：

```
sudo apt-get install sox flac
```

然后，有两个系统调用可以将文件转化成合适的格式：

```
subprocess.call(["sox", "output.wav", "-r16k", "-t",
                 "wav", "output-16k.wav"])
subprocess.call(["flac", "-5", "-f",
                 "output-16k.wav"], stdout=subprocess.PIPE)
```

`sox` 也被认为是“音频界的瑞士军刀”。它有很多处理音频文件的特性。本例中，它被用来将文件采样率由 44 100 转换为 16 000。由 `-r16k` 这个参数控制。第二行使用 `flac` 程序将 `WAV` 文件转换位 `FLAC`。

我们将在下一章详细介绍 `subprocess.call()`，现在你只需要知道它可以让你在 Python 中运行 Linux 命令。

9.1.5 深入探索

如果希望把这个例子继续做下去，可以把它作为一个特性添加你在到第 4 章创建的网络浏览器中。不再只是输出文字，而是用来控制浏览器访问哪个页面。

你也可以做一个简单的声音驱动菜单，这样就可以不用在程序中使用鼠标和键盘。例如，可以显示一组带数字的选项，用户只需要说出自己选择的数字即可选中（有点类似于电话交换台）。

9.2 制作电影

现在你应该知道如何在树莓派上处理音频了，因此我们将进到第二个阶段——视频。

树莓派上可以支持两种摄像头类型——USB 网络摄像头和树莓派基金会的摄像头模块。由于它们的应用不同，这里我们将分别介绍。

9.2.1 使用 USB 网络摄像头

两者之中，Python 对 USB 网络摄像头更好点。在所有的计算机中它的工作方式也都一样，因此我们首先介绍它。

说来奇怪，提取图像最简单的方法是在第 5 章中使用的 PyGame 模块。之前，你使用它构建游戏，但是模块包含有图像处理功能（游戏中经常要用到），其中包括从网络摄像头

中提取图像。

如果跟着第 5 章练习过，就应该安装过 PyGame。如果没有装过，现在先要按照下列方法安装模块：

```
sudo apt-get update
sudo apt-get install libsdl-dev libsdl-image1.2-dev \
    libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-dev \
    libportmidi-dev libavformat-dev libswscale-dev \
    mercurial python3-dev
```

然后获取最新版本的 Pygame：

```
hg clone <span style="font-family:monospace"><span
    style="font-family:monospace">https://bitbucket.org/pygame/
    pygame</span></span>
cd pygame
```

这两行将最新版本的 PyGame 下载到 pygame 目录中并将当前目录转到该目录。然后，就可以使用下面命令为 Python3 安装该模块：

```
python3 setup.py build
sudo python3 setup.py install
```

最简单的例子就是从摄像头中获取图像流并将其显示在屏幕上。实际上，它可以把你的树莓派变成一面镜子。下面给出了例子（代码参见网站文件 chapter9-mirror.py）。

```
import pygame
import pygame.camera

pygame.init()
pygame.camera.init()

display = pygame.display.set_mode((640,480),0)
cam = pygame.camera.Camera("/dev/video0", (640,480))
cam.start()

while True:
    image = cam.get_image()
    display.blit(image, (0,0))
    pygame.display.flip()
```

不仅要初始化 PyGame，你还需要初始化模块的摄像头部分。如果你只连接了一个摄像头，它在系统里就是 /dev/video0。如果连接了不止一个摄像头，就需要自己修改这个值。

然后使用模块，就是简单地调用 camera 对象的 get_image() 方法。它会返回一幅图像，你可以像处理 PyGame 图像一样随意使用该图像。本例中，我们只是将其画到主显示器上，然后调用 flip() 更新显示（在第 5 章中介绍过）。

得到图像后，不必立即将其直接显示到屏幕上。可以对它做一些操作。例如，可以使用一些图像效果，然后将其保存到文件中而不是显示出来。

看看下面的代码（参见网站文件 chapter9-mirror-lines.py）：

```
import pygame
import pygame.camera
pygame.init()
pygame.camera.init()
from pygame.locals import *

size = (320, 240)

display = pygame.display.set_mode(size, 0)

cam = pygame.camera.Camera("/dev/video0", size)
cam.start()

inverse = pygame.Surface(size, pygame.SRCALPHA)
for i in range (1,50):
    image = cam.get_image()
    lines = pygame.transform.laplacian(image)
    inverse.fill((255,255,255,255))
    inverse.blit(lines, (0,0), None, BLEND_RGB_SUB)
    display.blit(inverse, (0,0))
    pygame.display.flip()

pygame.image.save(inverse, "/home/pi/test.png")
```

这段代码和前面例子的基本结构相同，只是多了一些操作。首先，创建了一个新图像 lines。它由原始图像经过拉普拉斯（Laplacian）变换而成，即一种数学处理，可以获得图像中的边缘。返回图像将是一幅由黑色背景和表示图像边缘的白线构成的图像。

接着程序将图像转换成白色背景黑色线条。这不仅需要将图像颜色转换一下，为了做到这一点，你需要创建一幅白色图像，然后从中减去图像（即使用特殊标记 BLEND_RGB_SUB 调用 blit()）。

你可能已经注意到，这段代码没有 while 循环，而是使用了 for 循环。该循环将执行 50 次然后退出。这将会生成一个几秒钟的视频（如果想要精确时长，可以在程序中加入第 4 章中使用的定时器函数）。循环结束后，将会执行最后一行，将当前转换过的边缘图像保存为 PNG 文件。这让程序看起来像一个定时摄像机。

本程序和上个例子的最后一个区别就是这次使用的图像比上次小。图像尺寸在变量 size 中设置。这是因为图像变换非常消耗计算资源，将图像宽高分别缩小一半可以将计算量减少到原来的四分之一（计算量和区域面积成比例）。从而使得视频看起来更连续。试

着改变尺寸，对比下性能变化。

9.2.2 使用 OpenCV 添加计算机图像特性

PyGame 可以很好地为计算机获取图像，但用来处理图像却并不合适。另一个叫 OpenCV 的模块，是专门为处理图像设计的。不幸的是在写作本书时它还不支持 Python3，这种情况在未来也许会改变。由于没有这个模块你就无法实现一些非常有意思的特性，我们将使用 Python2 来介绍这部分内容。但愿支持 Python3 的版本快点发布，这样你就可以在新版本的 Python 中使用它。

首先，从命令行中安装它：

```
sudo apt-get install python-opencv libopencv-core-dev
```

第一个例子和使用 PyGame 的第一个例子相似：一个简单的摄像头查看器。它将让你感受到 OpenCV 库是如何工作的：

```
import cv2

capture = cv2.VideoCapture(0)

while True:
    return_val, frame = capture.read()
    cv2.imshow('Face', frame)

    key = cv2.waitKey(5)
    if key == 113:
        break
```

和使用 PyGame 大致相似。打开摄像头，然后读取一帧并显示出来。最后三行监听用户按下 q 键的动作（q 键对应的键是 113），如果按下就退出。

OpenCV 是一个异常强大的用来处理图像的模块。一个最有趣的特性是目标识别。你的程序可以辨识图像的特定部分，例如眼睛或嘴。

这部分由保存在 XML 文件中的 Haar cascades 完成。它由训练图像集生成，计算机使用训练图像集找出相似的目标。你可以使用运行在 Linux 命令行下的 opencv_traincascade 程序训练自己的 Haar cascades。然而，你需要大量的图像（为了获得较好结果一般需要上百张），包括含有和不含有你希望识别物体的图像。关于如何做到，这里有个不错的介绍 http://docs.opencv.org/doc/user_guide/ug_traincascade.html。

我们不会去创建 cascades，OpenCV 在安装时给我们提供了一些例子。下面的例子就是用 OpenCV bundles 来识别眼睛和嘴。代码参见网站文件 chapter9-objectdect.py。

```

import cv2
factor_down = 0.33
factor_up = 3

capture = cv2.VideoCapture(0)
eye_data = cv2.CascadeClassifier('/usr/share/opencv/haarcascades/' +
                                + 'haarcascade_eye.xml')
smile_data = cv2.CascadeClassifier('/usr/share/opencv/' +
                                   haarcascades/' +
                                   'haarcascade_mcs_mouth.xml')

while True:

    return_val, frame = capture.read()
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    small_gray_frame = cv2.resize(gray_frame, (0,0),
                                  fx=factor_down, fy=factor_down)

    eyes = eye_data.detectMultiScale(small_gray_frame, 1.3, 5)
    for (eye_x, eye_y, eye_width, eye_height) in eyes:
        cv2.rectangle(frame, (eye_x*factor_up, eye_y*factor_up),
                      (eye_x*factor_up + eye_width*factor_up,
                       eye_y*factor_up+eye_height*factor_up),
                      (255,0,0),2)

    smiles = smile_data.detectMultiScale(small_gray_frame, 1.3, 5)
    for (smile_x, smile_y, smile_width, smile_height) in smiles:
        cv2.rectangle(frame, (smile_x*factor_up,
                              smile_y*factor_up),
                      (smile_x*factor_up + smile_width*factor_up,
                       smile_y*factor_up+smile_height*factor_up),
                      (0,255,0),2)
    cv2.imshow('Face', frame)

    key = cv2.waitKey(5)
    if key == 113:
        break

```

可以看出，加载 Haar cascades 非常简单，只需要使用一个参数来调用 `cv2.CascadeClassifier()`。它通过使用 `detectMultiScale()` 方法创建一个对象，可以使用该对象返回满足 cascade 条件的矩形区域列表。不过这里还有一些其他事情。

在 `while` 循环中，和上一个 OpenCV 例子一样，程序首先获取一帧图像，然后创建出两幅新图像——`gray_frame` 和 `small_gray_frame`。不要惊讶，它们分别是原始图像的黑白图像和一个缩小版的黑白图像。这样做是由于目标识别非常消耗计算资源，虽然树莓派可以处理完整尺寸的图像，但是运行速度会非常慢。图像越小，目标识别的精度越低。

我们发现将图像缩小到原始图像的三分之一，是个不错的折衷。通过调整 `factor_down` 和 `factor_up`（它们必须互为倒数）的值，找出适合你自己的参数。



提示 如果想得到更好的性能，可以在 LXTerminal 中使用 `raspi-config` 超频你的树莓派。

然后 `detectMultiScale()` 返回匹配目标的所有矩形区域坐标，两个 `for` 循环将这些结果全部绘制到图像上。

9.2.3 深入探索

如果你准备花时间创建训练图像集，那么可以将这个简单的例子运用到多个程序中。例如，可以使用它来识别用户而不再需要用户登录，或者也可以在数控小车上添加 OpenCV 使其能够识别简单标志。

即使没有建立自己的 Haar cascades，你也可以为树莓派添加视觉特性。例如，使用手势 Haar cascades，注意观察矩形区域，你将看到矩形区域随着手的挥动而移动。

9.2.4 使用树莓派摄像头模块

USB 网络摄像头并不是在树莓派工程中获取图像和视频的唯一选择。事实上，它可能也不是最流行方法。树莓派基金会自己发布了可以插在主板上的摄像头模块。这里列出了摄像头模块相比 USB 网络摄像头的优点和缺点。

优点：

- 比大多数摄像头有更高的静态图像分辨率（5M 像素）。
- 可以很好地支持 Linux 命令行。
- 有竞争力的价格（19 英镑，约 31 美元[⊖]）。
- 提供有红外（夜视）版本。

缺点：

- 对于非树莓派专用程序和 Python 模块支持不佳。
- 每个板子只支持一个摄像头。
- 线缆很短（15cm），虽然可延长，但是很难。
- 一般需要根据项目需求选择合适的摄像头。
- 摄像头模块和 OpenCV。

[⊖] 约 200 人民币。——译者注

前一节提到的 OpenCV，原生不支持树莓派基金会的摄像头模块。但是，你可以按照下列网站上的说明修改 OpenCV，使其能够支持摄像头模块：<http://thinkrpi.wordpress.com/2013/05/22/opencv-and-camera-board-csi/>。

写作本书时，还没有支持树莓派摄像头模块的 Python 模块。只有一个支持的项目：<http://github.com/ashtons/picom>。但是仍然不够稳定。此时，最好的方法就是在 Python 中使用命令行工具访问摄像头。

第一件要做的事就是确保摄像头连接到树莓派上并已经启用。打开 LXTerminal 会话并输入：

```
raspistill -o test.png
```

该命令将输出几秒摄像头的预览图像。然后将结果保存到 test.png 中。如果发现任何问题，请翻阅位于 www.raspberrypi.org 的树莓派摄像头设置指南，在继续进行之前先要修复这些问题。

这是一个最基础的使用命令行获取图像的方法。你也可以在 Python 中做同样的事情：

```
>>> import subprocess
>>> subprocess.call(["raspistill", "-o", "test.png"])
```

如果需要，你可以在 Python 中使用该图像，就像使用其他图像一样。例如，可以使用第 4 章中使用的 PyGame 提供的加载图像特性。raspistill 提供了多种选项来控制图像的拍摄方法。在命令行中输入 raspistill 会列出其所有选项。选项范围从简单，例如图像大小到复杂如测光方式或自动白平衡（AWB）。

举个例子，下列代码用来拍摄一个 200×200 像素大小、高对比度、使用素描图像效果的图像。此外，它不在屏幕上预览并有 1 毫秒的延时。

```
raspistill -w 200 -h 200 -co 90 -n -t 1 -ifx sketch -o test.png
```

结果有点像卡通图像。非常适合拿来做缩时摄影卡通视频，这也是我们在下一个例子中要做的（参见网站上文件 chapter9-timelapse.py）。

```
import subprocess
import time

frames = 10

for i in range(1,frames):
    filename = "test" + format(i, "03d") + ".jpg"
    subprocess.call(["raspistill", "-n", "-t", '1', "-w", "200",
                    "-h", "200", '-co', "90", "-ifx", "sketch",
                    "-e", "jpg", "-e", "jpg", "-o", filename])
```

```

print("taking photo", i)
print("Encoding. . .")
subprocess.call(["avconv", "-r", "5", "-i", "test%03d.jpg", "-r",
                 "24", "-s", "200x200",
                 "-vync", "cfr", "out.mpg"])

```

这段代码中 raspistill 使用了选项 -e jpg，将图像保存为 JPEG 而不是 PNG 格式。为以后转成视频提供方便。获取完所有图片之后，使用 avconv 将其转换成视频。这里的关键选项是第一个 -r。后面的数字表示创建一秒钟视频使用的图像总量。本例中，我们使用 5，但你可以改成任何自己想要的数字（第二个 -r 表示最终生成的视频每秒钟的帧数）。-i test%03d.jpg 表示选取五个名字像 test001.jpg 和 test002.jpg 的文件。这个和创建文件的代码相一致：

```
filename = "test" + format(i, "03d") + ".jpg"
```

这两个部分必须一致，这样视频编码器才能够选用刚才创建的图片。你可以将 test 换成任何字母，但必须在两个地方同时更改。如果你希望使用大于 999 的图片，则需要添加更多的 0。例如你希望使用 99 999 张图片，就需要将这两行中的 3 改为 5（变量 frames 保存了总的图片拍摄数目，实际上拍摄的数量远小于实际这个数，我们只是举个例子）。

-s 200×200 表示视频的尺寸是 200×200 像素。它不必和图像分辨率一致。你可以将 out.mpg 改成任何名字，它表示最后生成的视频文件名。

运行代码，将会创建出一个视频。你可以使用任何标准视频播放器如 VLC 来回放结果。

9.2.5 创建直播视频

如果你希望一段预先设定好时长的视频，上面的方法是个不错的选择，但如果希望获得一段不间断地的视频时应该怎么做呢？例如，如果你希望创建一段直播视频使其能够在通过互联网收看时该怎么做呢？在互联网上有个传说，第一个网络摄像头就安装在剑桥大学（树莓派的故乡），人们通过它监控咖啡壶，这样它们就可以在适当的时间到茶水间冲咖啡。下面的例子重现了网络摄像头的案例。

为了通过因特网发送数据，你需要某种形式的服务器。有一些服务器可以做到，但是从熟悉的角度看，我们将继续使用第 7 章中见过的 Tornado 服务器。

下列代码用来创建直播视频。运行之前，你需要首先使用命令 mkdir/home/pi/images

创建目录 /home/pi/images。

```

import tornado.ioloop
import tornado.web
import subprocess
import time

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        subprocess.call(["raspistill", "-w", "200", "-h", "200",
                        "-e", "jpg", "-n", "-t", "1", "-o",
                        "/home/pi/images/live.jpg"])
        time.sleep(2)
        self.write('<!DOCTYPE html><head>' +
                  '<META HTTP-EQUIV="refresh"' +
                  ' CONTENT="5"></head><body>' +
                  '</body>')
class ImageHandler(tornado.web.StaticFileHandler):
    def set_extra_headers(self, path):
        self.set_header('Cache-Control',
                        'no-store, no-cache, must-revalidate,' +
                        ' max-age=0')

application = tornado.web.Application([
    (r"/", MainHandler),
    (r"/images/(.*)", ImageHandler, {"path": "/home/pi/images"})])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

如果不熟悉这段，就翻到第 7 章去看下 Tornado 这一节。不过这里还是有一些新东西。

其实，这里所做的一切都是在等待用户请求含有实时视频流（来自根目录，你可以根据自己需要来改动）的页面，然后开始拍摄新图像。打开网页后，网页内包含有拍摄的图像。为了获得实时视频流而不是实时图片，网页中包含有标签 <META HTTP-EQUIV="refresh" CONTENT="5">，用来告诉网络浏览器每 5 秒刷新一次页面。视频质量不高，但足以让你看清壶中咖啡数量的变化。

类 ImageHandler 继承自 tornado.web.StaticFileHandler。这样我们就可以修改缓存的默认设置。默认情况下，Tornado 为了提高效率不会释放之前保存的图像。当网络浏览器请求一张 Tornado 已经发送过的图像，它会告诉浏览器使用之前发送的图像就可以了。由于浏览器会一直请求图像 live.jpg，这样它就永远只能得到第一张图片。添加 set_extra_headers，就是告诉 Tornado 不使用缓存，每次都重新读取文件。

现在可以工作了，但还不够理想。例如每次有人请求页面时它都会重新获取一张图片，如果很多人不断请求页面，会造成拥塞，因为摄像头每次只能拍摄一张照片。如果只

是监视咖啡壶，这样做没有问题。但如果用作保安摄像头，就会遇到麻烦。

为了避开这个问题，我们需要将拍照部分分割成独立的一部分，而另一部分用来提供服务。下列代码会连续地拍照片，这些照片可以用来合成视频，同时，使用最新的图像来更新 live.jpg。参见网站文件 chapter9-live-take-pics.py：

```
import subprocess
import time
file_number = 0
dir = "/home/pi/images/"

while True:
    file_name = dir + format(file_number, "05d") + ".jpg"
    file_number = file_number + 1
    subprocess.call(["raspistill", "-w", "400", "-h", "400", "-e",
                    "jpg", "-n", "-t", "1", "-o", file_name])
    subprocess.call(["cp", "-f", file_name, dir + "live.jpg"])
    time.sleep(2)
```

下面一段会代码自动运行上面的脚本，并和上个例子一样显示最新一幅图像。参见网站文件 chapter9-live-image-server.py。

```
import tornado.ioloop
import tornado.web
import subprocess

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write('<!DOCTYPE html><head>' +
                  '<META HTTP-EQUIV="refresh"' +
                  ' CONTENT="5"></head><body>' +
                  '</body>')

class ImageHandler(tornado.web.StaticFileHandler):
    def set_extra_headers(self, path):
        self.set_header('Cache-Control',
                       'no-store, no-cache, must-revalidate, max-age=0')

application = tornado.web.Application([
    (r"/", MainHandler),
    (r"/images/(.*)", ImageHandler, {"path": "/home/pi/images"})])

if __name__ == "__main__":
    subprocess.Popen(["python3", "chapter 9-live-take-pics.py"])
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

这里使用 `subprocess.Popen()` 来创建一个新进程，使两个程序同时运行（将在下一章中详细介绍）。

9.2.6 深入探索

不过这里还有一些问题。这段代码将会不停运行，直到图像把 SD 卡占满为止。此时，树莓派将会崩溃。如果你希望进一步探索，可以试着做一下。例如，每隔一定时间（可以是每天，或者每 20 000 张图像），将图片压缩成视频，然后删除图像。另外，也可以将视频传到网上。

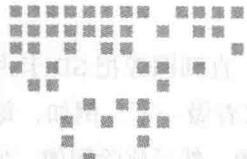
为了更安全，可以将视频传到异地的中央服务器上。当树莓派丢失或损坏后，仍然能够得到这些视频。

如果你是在公共网络服务器上做开发，可以实现一些安全方面的特性，只允许授权用户在线查看视频。详情请参阅第 7 章相应小节。

9.3 小结

读完本章，你应该更好地理解以下内容：

- PyAudio 是一个不错的模块，可以为 Python 程序添加声音。
- wave 模块可以输出 WAVE 格式的文件。
- 并不是所有声音处理任务都有相应的 Python 工具，但没有问题，可以使用 Linux 命令行工具如 sox 和 flac 来弥补这个缺陷。
- 可以使用网络服务如 Google 的语言识别来增加更多的音频特性。
- 为树莓派添加摄像头有两个选择：USB 网络摄像头和树莓派基金会的摄像头模块。
- USB 网络摄像头能很好地兼容标准 Python 工具如 PyGame 和 OpenCV。
- 摄像头模块也在 OpenCV 中工作，但需要使用自己的命令行工具来捕获图像和视频。
- 使用 OpenCV 可以添加机器视觉特性如目标识别。
- 网络服务器如 Tornado 可以用来在互联网上提供图像流服务。



第 10 章

脚本

为了保证计算机正确合理地运行，需要执行很多不同的任务。例如，为防止数据丢失，你需要定期备份数据。还有些任务可以根据你的需求来完成某件事情。你可能需要给音乐、图片排序。本章主要关注如何利用 Python 自动化这些常规工作，尽可能地减少操作计算机，让生活变得更简单。

这需要很多和底层操作系统的交互，因此首先要学习一点 Linux 知识。

10.1 从 Linux 命令行开始

现在你可能已经熟练掌握了 Raspbian 的常规操作。Raspbian 作为操作系统有几个明显特征。首先，它使用了 Linux 内核。内核开机后一直运行，为系统提供了底层支持，用来管理硬件和内存，控制程序如何运行。其次，它为使用 Raspbian 提供了命令和大量工具，这些都是基于文本模式的。在本章中将详细介绍这部分内容。Raspbian 的另一个特性就是提供了桌面环境，之前我们接触过，但在本章中我们很少使用它。默认的 GUI 是 LXDE，基于 LXDE 有很多图形界面程序。

如果经常使用 Windows，你首先面临的最大变化可能就是文件系统。树莓派上找不到 C 盘（或者类似的其他盘符）。所有东西都是基于一个从 /（即根，root）开始的层次结构。开始学习 Python 脚本时，先要东西放在文件系统的什么位置。打开 LXTerminal（树莓派中提供命令行工具的窗口）并输入以下内容：

```
cd /
ls
```

第一个命令将目录更改到 / (根目录), 第二个命令列出当前目录下的所有内容。运行后可以得到类似的结果 (见图 10-1):

```
bin dev home lost+found mnt proc run selinux sys usr
boot etc lib media opt root sbin srv tmp var
```

这些是根目录下的目录 (如果根目录下有文件, 也会在此列出, 但现在这里没有任何文件)。如果使用其他 Linux 系统, 你会在根目录 / 下发现相同的子目录。这里的大多数目录你都不需要使用。Raspbian 会帮你管理好它们, 并保持更新。需要使用的有: 用户 home 目录 /home, 可移动设备如 USB 记忆棒的目录 /media, 和保存系统基本的应用程序集合 /etc。

认识到这个文件系统和保存在硬盘上的文件系统不同非常重要。它和 Windows 的做法不同。例如, /home/pi 存在 SD 卡上。然而如果将名字为 MyStick 的 USB 记忆棒插入树莓派, 你将会在 /media/MyStick 中找到它。

/sys 和 /proc 在任何磁盘上都不存在。它们是虚拟文件系统, 看起来和正常的目录和文件类似, 但只是系统用来显示系统的一种方式。例如, 在 LXTerminal 中输入:

```
cat /proc/cpuinfo
```

cat 是用来输出一个或多个文件内容的命令。/proc/cpuinfo 中包含了 CPU 的技术细节。你将得到类似输出:

```
pi@raspberrypi ~ $ cat /proc/cpuinfo
Processor: ARMv6-compatible processor rev 7 (v6l)
BogoMIPS: 697.95
Features: swp half thumb fastmult vfp edsp java tls
CPU implementer: 0x41
CPU architecture: 7
CPU variant: 0x0
CPU part: 0xb76
CPU revision: 17
Hardware: BCM2708
```

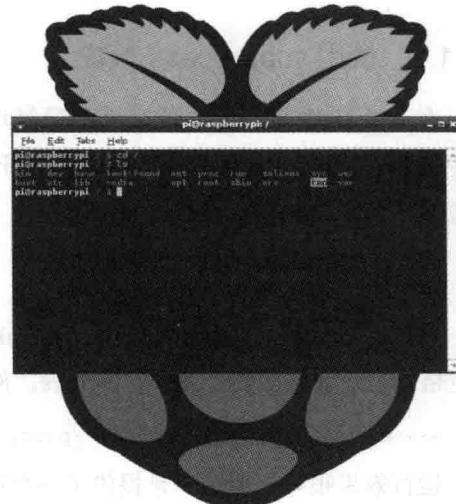


图 10-1 通过 LXTerminal 可以和强大的 Linux 命令行换件交互

```
Revision: 000d
Serial: 000000074f3d523
```

LXTerminal 使用了 Bash 环境。这是个强大的环境，并提供自己的编程语言。如果希望学习更多关于 Raspbian 以及通用 Linux 的内容，可以从 Bash 开始，可用的资源有很多（例如 www.linuxcommand.org）。然而这是一本关于 Python 的书，因为我们将不再深入下去，有兴趣的读者可以自学。

10.1.1 使用 subprocess 模块

在 Python 中与底层系统交互的最简单的方法就是使用 subprocess 模块。我们在第 7 章中使用过它，你应该还有点印象。打开 Python 解释器（如果还在使用 LXTerminal，可以直接输入 `python3`），并输入以下内容：

```
>>> import subprocess
>>> subprocess.call("ls")
```

可以看出，你可以通过 `subprocess.call()` 在底层系统中运行任何命令。如果命令中含有空格，则需要将其划分成字符串列表。例如，命令 `cat/proc/cpuinfo` 需要变成：

```
>>> subprocess.call(["cat", "/proc/cpuinfo"])
```

运行效果很好，但这只是提供了一个更繁琐的执行命令的方式。毕竟，你可以简单地通过 LXTerminal 来执行相同的命令。正如我们在开始时说的那样，本章的目标是自动运行普通任务，为了做到这一点，你需要分析命令输出，然后才能操纵它们。

例如，`cat/proc/cpuinfo` 返回很多信息，大多数你可能都不需要关注。下面的程序剔除了除计算机的处理器类型之外的所有信息。

```
import subprocess

p = subprocess.Popen(["cat", "/proc/cpuinfo"],
                    stdout=subprocess.PIPE)

text = p.stdout.read().decode()

for line in text.splitlines():
    if line[:9] == "Processor":
        print(line)
```

这里使用了 `subprocess.Popen()` 而不是 `subprocess.call()`。它创建了一个新对象，我们可以使用该对象来获取想要的信息，从而实现更多的控制功能。

运行在 Linux 机器上的命令，通常有两个输出：`stdout` 和 `stderr`。所有正常输出都输出到 `stdout`（或者标准输出（standard out））。例如当运行 Python 程序时，所有的 `print()` 语句

都将输出到 stdout。系统将错误信息传送到 stderr（或者标准错误（standard error））中。如果命令在 LXTerminal 中执行，这 stdout 和 stderr 都会显示到屏幕上，但如果使用脚本来运行程序，将这两者分离开会更好。比如，如果运行很多命令，可以将所有出错信息放在一个地方。这样有错误发生时可以立即看到而不用再去检查所有的输出信息。

参数 stdout=subprocess.PIPE 告诉 Python 将 stdout 输出到我们创建的对象中而不是到屏幕上。由于你没有告诉它将 stderr 输出到哪里，默认情况下，它将输出到屏幕。因此，如果在上述程序中使用下面这行：

```
p = subprocess.Popen(["cat", "/proc/cpuinfo"],  
                    stdout=subprocess.PIPE)
```

即使没有任何的 print() 语句中打印出错信息，它仍然会被打印到屏幕上，

如果需要，你也可以捕获 stderr。例如，在下面的程序中，将把用户输入的文件名对应的文件内容。

```
import subprocess  
  
f_name = input("Enter a filename: ")  
  
p = subprocess.Popen(["cat", f_name], stdout=subprocess.PIPE,  
                    stderr=subprocess.PIPE)  
  
text = p.stdout.read().decode()  
error = p.stderr.read().decode()  
  
print(text)  
  
if len(error) > 0:  
    print("*****ERROR*****")  
    print(error)
```

10.1.2 命令行标签

Linux 命令通常带有选项。这些选项跟在主命令之后，为命令提供附加信息。例如，在 LXTerminal 中，执行下列命令：

```
ls  
ls -a  
ls --all
```

ls 列出了当前目录下的内容。使用选项 -a，可以列出目录中的所有东西（通常 ls 会忽略以 ‘.’ 开头的文件和目录）。--all 和 -a 一样。许多命令的同一个选项都有两个版本——以 - 开头的短格式和以 -- 开头的长格式。使用标签 -h 或者 -help 运行 ls，可以获得更多帮助信息。

很快你就会看到，选项后面还可以带值。

如果你在开发脚本，应该尽最大可能遵守这些约定。`optparse` 模块可以帮助我们。上一个例子的文件名可以改成从选项输入而不是提示用户输入。看看下面的例子：

```

import subprocess
from optparse import OptionParser

parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="The file to display")

options, arguments = parser.parse_args()

if options.filename:
    p = subprocess.Popen(["cat", options.filename],
                         stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE)

    text = p.stdout.read().decode()
    error = p.stderr.read().decode()
else:
    test = ""
    error = "Filename not given"

if len(error) > 0:
    print("*****ERROR*****")
    print(error)
else:
    print(text)

```

可以看到，这里创建了 `OptionParser` 对象。本例中只有一个选项，你可以多调用几次 `parser.add_option()` 来增加更多选项。其中前两个参数表示选项的短格式和长格式。`dest="filename"` 表示该选项的值保存在 `options` 返回的 `filename` 属性。

注意 `parser` 将自动添加 `-h` 和 `--help` 选项，在 `add_option()` 调用中使用 `help=` 参数建立帮助文本。

如果你将上述代码保存为 `print-file.py`，可以在 LXTerminal 中 `cd` 到文件所在目录，输入 `python3 print-file.py -f /proc/cpuinfo` 来运行它，也可以通过 `python3 print-file.py --help` 查看帮助。

10.1.3 正则表达式

不过这里只是将 `cat` 命令封装到 Python 中和直接执行该命令的效果相同，并没有增加任何东西。

现在我们来添加一个特性，根据用户指定来显示文件的某几行。

Python（和许多其他编程语言）有一个称为正则表达式的特性。它的名字有点奇怪（通

常缩写为 regex)，可以用来匹配指定的文本。

正则表达式通过特殊字符来工作。最常见的特殊字符是 *。表示匹配前面的字符零至多次（任意数量）。例如，do*g 将匹配 dg、dog、doog 等。字符 + 表示匹配前面的字符一至多次。例如，do+g 将匹配到 dog、doog、dooog 等，但是没有 dg。而 do?g 将只匹配 dg 或者 dog。

句号将用来匹配除换行符外的任意字符，因此，.* 将匹配任意行，而 .+ 将匹配所有非空行。

还可以将字符集合在一起，例如 d[io]g 将匹配 dig 和 dog，d[io]*g 将匹配 dg、dog、dig、doog、dioioioig 等其他类似的字符串。

稍后你将看到正则表达式的更多特性，现在让我们开始学习如何使用它们。下面的代码来自网站文件 chapter10-regex.py。

```
import subprocess
from optparse import OptionParser
import re

parser = OptionParser()

parser.add_option("-f", "-file", dest="filename",
                  help="The file to display")

parser.add_option("-r", "-regex", dest="regex",
                  help="The regular expression to search for")
options, arguments = parser.parse_args()
if options.filename:
    p = subprocess.Popen(["cat", options.filename],
                        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    text = p.stdout.read().decode()
    error = p.stderr.read().decode()
else:
    test = ""
    error = "Filename not given"

if len(error) > 0:
    print("*****ERROR*****")
    print(error)
else:
    for line in text.splitlines():
        if not options.regex or (options.regex
                                 and re.search(options.regex, line)):
            print(line)
```

代码使用了 re 模块提供的正则表达式。使用正则表达式有两种主要方式：re.match() 和 re.search()。第一个方法只检测 RE 是不是在 string 的开始位置匹配，第二种方法则会扫

描整个字符串查找匹配。上述程序使用了后者，因为我们希望能够尽可能简单地匹配到指定的那几行。

if 语句中的条件有点复杂：

```
if not options.regex or (options.regex
    and re.search(options.regex, line)):
```

它用来处理用户没有输入 -r 或者 -regex 选项的情况。如果没有 regex 选项或者有该选项并且匹配到相应的行，就输出该行。

运行这段代码，将得到与第一个例子相同的结果：

```
python3 chapter.py -f /proc/cpuinfo -r Processor
```

Linux 系统将注册的各种事件保存在 log 文件中，对于诊断问题十分有用。但 log 文件会变得很大，使用起来不方便。它们都存放在 /var/log 文件夹中。syslog 保存了系统的各种功能的信息。例如，如果你遇到了 USB 驱动的问题，运行下列命令将列出树莓派中注册新的 USB 外围设备的所有事件（无论是 hub 还是设备）：

```
python3 chapter10-regex.py -f /var/log/syslog -r"USB.*found"
```

这里不需要将正则表达式写在括号中，但有时 Linux shell 在传值给 Python 前会处理一些特殊字符，使用双引号将阻止这些操作，因此在命令行中使用正则表达式时使用双引号是个好习惯。

回到正则表达式的特性：

- [^abc] 匹配任何除 a、b 和 c 之外的任意字符，因此 d[^o]g 匹配 dig、d~~x~~g 和 dag 但不包括 dog。
- [a-c] 匹配从 a 到 c 的字符，因此 d[a-j]g 匹配 dag、dbg 和 dig，但不包括 dkg 或 dog。
- {a,b} 匹配前面的字符 a 至 b 次。因此 do{2,4}g 匹配 doog、do~~oo~~og 和 doooog，但不包括 dog 或 dooooog。

此外，还有一些由斜线开头，有特殊含义的特殊字母。表 10-1 列出了这些特殊字母。

表 10-1 正则表达式中的转义字符

| 代码 | 说明 |
|----|-------------------|
| \n | 换行符 |
| \t | 制表符，Tab |
| \d | 任意数字 |
| \D | 任意非数字的字符 |
| \s | 任意的空白符 |
| \w | 字母或数字或下划线或汉字 |
| \W | 不是字母、数字、下划线、汉字的字符 |

显然，在匹配\字符时会遇到问题。为此，可以使用转义符\来匹配特殊字符（例如.，*或者+）。因此，\\w 匹配字符 w，而 \\w 匹配任意字母数字字符组。

由于斜线需要使用转义符，在 Python 中遇到字符串时会产生新的问题。例如：

```
>>> print("\\")

\
>>> print("\\\\")
```

 **注意** 这个例子中没有问题，因为你已经将字符串传递给 Python。但如果你在 Python 中创建用于匹配正则表达式的字符串，则需要记住在正则表达式中使用的每个斜线字符前多加一个斜线（用作转义符）。

10.2 知识测验

我们用很短的篇幅介绍了大量关于正则表达式的内容。为了确保你能够完全理解，请看下面的练习。

将下列字符保存到文件中：

```
aaa
a10
10
Hello
Helloo
Helloo
```

下列正则表达式分别能够匹配出哪几行？试着回答一下，然后运行上一个例子查看结果（你需要使用 Leafpad 创建文件或者从网站上下载 chapter10-regex-test）。例如，运行下列命令可以得到第一个答案：

```
python3 chapter10-regex.py -f chapter10-regex-test -r"."
```

记得在程序中使用 re.search() 而不是 re.match()

- .
- \d
- \D\d
- l{3,4}
- e*
- e+
- [Ha]

```
■ \d{2,3}
■ 1?
```

10.3 脚本中的网络

在第 7 章中已经介绍过网络，此处不再赘述。这里我们将介绍一个在脚本中使用网络的方法。最常见的事情就是在两个计算机之间复制文件。有个叫 Fabric 的模块非常好用，但在本书写作之时，它还不支持 Python3。情况可能会改变，但也不会很快。

由于缺乏可用模块，你需要构建 Python 代码来获取和复制所有东西。这也确实可行，但需要花费太长时间。还有一个 Linux 命令行程序 scp (secure copy, 安全复制) 可以做到这点，并且你也已经见过如何运行命令行程序。

尽管这是一个很小的程序，在正常运行 scp 时，它需要询问你的密码。这在 Python 脚本中会产生问题，因为你不能简单地让它回答问题。然而，scp 允许你使用证书来登录。这样当你以合法用户身份登录到一个机器上之后，不需要再使用密码就可以直接登录。

scp 可以用于在不同的 Linux 计算机（如树莓派）之间复制信息，因此可以用于两个树莓派，或者树莓派和 Linux 服务器。其他（非 Linux）服务器也可能支持 scp，但需要让你的系统管理员安装这个服务。

首先要做的是登录到你需要传送信息的机器上，并运行在命令行中，如 LXTerminal，运行下列命令。如果你只有一台 Linux 机器，你可以在单个机器上做这个实验，因此在本机上运行该命令。如果你是远程登录，可以通过 ssh 来操作。

```
ssh-keygen -t rsa
```

它将会在你的 home 目录下的 .ssh 文件夹中创建两个文件，id_rsa 和 id_rsa.pub。其中包含了公钥和私钥。id_rsa.pub 是公钥，需要将其复制到你要登录的机器上去。可以使用 USB 记忆棒，云存储（如 Dropbox 或 Google Drive），甚至是 email，但由于我们在介绍 scp，你也可以使用它。scp 的命令格式如下：

```
scp location1 location2
```

即将文件从 location1 复制到 location2。如果其中一个位置是正常文件路径，如 /home/pi/.ssh/id_rsa.pub，scp 将把它当做本地路径。如果路径是这个格式：user@machine:/home/pi，scp 将使用 user 登录远程主机。machine 可以是 IP 地址或主机名。对于树莓派，将使用 IP 地址。如果你希望使用用户名 pi 将 id_rsa.pub 复制到 IP 地址为 192.168.0.10 的主机，则命令如下：

```
scp /home/pi/.ssh/id_rsa.pub pi@192.168.0.10:/home/pi
```

如果你是在单个主机上操作，则不需要移动文件。但还是试一下 scp，可以使用主机 localhost。因此复制命令如下：

```
scp /home/pi/.ssh/id_rsa.pub pi@localhost:/home/pi
```

其次你需要登录刚才复制文件的远程主机，使用下面的命令将文件内容复制到 authorized_keys 中：

```
cat /home/pi/id_rsa.pub >> /home/pi/.ssh/authorized_keys
```

看起来有点烦琐，但现在不使用密码就可以将文件从本机复制到其他机器上了（相反方向不可以）。输入下面的命令试一下：

```
touch test
scp test user@machine:/home/pi/
```

其中 user, machine 和 /home/pi 可以根据自己的环境做适当变化。第一行只是简单地创建一个名为 test 的文件。如果一切顺利，就不再需要输入密码了。

所有这些设置完成之后，你就可以使用 subprocess.call() 在两个机器之间复制文件。例如：

```
subprocess.call(["scp", "file1.py", "pi@localhost:/home/pi"])
```

10.4 组合包装

在本章开始的时候，我们说过 Python 可以使计算机常规操作变得更简单。这里已经演示了很多不错的 Python 工具，但还没有满足这个目标。现在我们就开始。本节中，我们将创建一个 Python 程序，可以帮助你备份常用文件，以防数据丢失。SD 卡损坏后，你也可以恢复自己的数据。

我们所使用的工具大部分都在本章中介绍过。另外还需要一个新的 os 模块。它提供了访问操作系统的功能。下面这段代码（后面有解释）可以从网站上下载 chapter10-backup.py：

```
import os
import tarfile
from optparse import OptionParser
from time import localtime
import datetime
import subprocess
import re
```

```

parser = OptionParser()

parser.add_option("-f", "--file", dest="filename",
                  help="filename to write backup to (if no option
                        is give, backup will be used)", metavar="FILE")
parser.add_option("-p", "--path", dest="path",
                  help="path to backup (if no option is give,
                        will be used)")
parser.add_option("-v", "--verbose", action="store_true",
                  dest="verbose", default=False,
                  help="print status messages to stdout")
parser.add_option("-i", "--images", action="store_true",
                  dest="images", default=False,
                  help="backup image files")
parser.add_option("-c", "--code", action="store_true", dest="code",
                  default=False,
                  help="backup code files")
parser.add_option("-d", "--documents", action="store_true",
                  dest="documents", default=False,
                  help="backup document files")
parser.add_option("-a", "--all", action="store_true", dest="all",
                  default=False,
                  help="backup all filetypes (this overrides
                        c, d & i)")
parser.add_option("-m", "--mtime", dest="mtime", default=False,
                  help="backup files modified less than this many
                        days ago")
parser.add_option("-r", "--regex", dest="regex",
                  help="only back up filenames that match this
                        regex")
parser.add_option("-s", "--server", dest="server", default=False,
                  help="copy backup file to this remote point
                        (should be an scp location)")

options, arguments = parser.parse_args()

if options.filename:
    backup_file_name = options.filename + '.tar.gz'
else:
    backup_file_name = "backup.tar.gz"

backup_tar = tarfile.open(backup_file_name, "w:gz")

file_types = {"code": [".py"],
              "image": [".jpeg", ".jpg", ".png", ".gif"],
              "document": [".doc", "docx", ".odt", ".rtf"]}
backup_types = []
all_types = False

if options.images:
    backup_types.extend(file_types["image"])

```

```

if options.code:
    backup_types.extend(file_types["code"])
if options.documents:
    backup_types.extend(file_types["document"])

if len(backup_types) == 0 or options.all:
    all_types = True
if options.mtime:
    try:
        mtime_option = int(options.mtime)
    except ValueError:
        print("mtime option is not a valid integer.",
              "Ignoring option")
        mtime_option = -1
else:
    mtime_option = -1

if options.path:
    if os.path.isdir(options.path):
        directory = options.path
    else:
        print("Directory not found. Using ~")
        directory = os.getenv("HOME")
else:
    directory = os.getenv("HOME")

for root, dirs, files in os.walk(directory):
    for file_name in files:
        if not options.regex or re.match(options.regex, file_name):
            name, extension = os.path.splitext(os.path.join(root,
                                                               file_name))
            if (extension in backup_types) or all_types:
                modified_days = (datetime.datetime.now() -
                                  datetime.datetime.fromtimestamp(
                                      os.path.getmtime(
                                          os.path.join(root,
                                                      file_name))).days
                if mtime_option < 0 or modified_days <
                   mtime_option:
                    if options.verbose:
                        print("Adding ",
                              os.path.join(root, file_name),
                              "last modified", modified_days,
                              "days ago")
                    backup_tar.add(os.path.join(root, file_name))
if options.server:
    subprocess.call(["scp", backup_file_name, options.server])

```

通过各种 parser.add_option() 可以传递不同的信息，你可以根据自己需要改变这些代码。它的基本功能就是将文件复制到 tar.gz 文件中，即 Linux 系统中流行的一种压缩文件。

改文件随后会自动复制到位于另一个服务器上的安全位置。如果源文件遭到破坏，就可以通过备份恢复它。

运行 `python3 chapter10-backup.py-help`，将会看到如下输出，即帮助信息：

```
Usage: chapter10-backup.py [options]

Options:
  -h, --help            show this help message and exit
  -f FILE, --file=FILE  filename to write backup to (if no option
                        is given, backup will be used)
  -p PATH, --path=PATH  path to backup (if no option is given, ~
                        will be used)
  -v, --verbose         print status messages to stdout
  -i, --images          backup image files
  -c, --code            backup code files
  -d, --documents       backup document files
  -a, --all             backup all filetypes (this overrides c, d &
                        i)
  -m MTIME, --mtime=MTIME
                        backup files modified less than this many
                        days ago
  -r REGEX, --regex=REGEX
                        only back up filenames that match this
                        regex
  -s SERVER, --server=SERVER
                        copy backup file to this remote point
                        (should be an scp location)
```

基本使用方法如下：

```
python3 chapter10-backup.py -f backup.tar.gz -p /home/pi -s \
pi@192.168.0.10:/home/pi/backups/
```

这些参数表示程序将搜索 `/home/pi` 及其子文件夹中的文件。程序中使用了 `os.walk()` 函数来完成此项操作。它将返回一个目录和文件的集合，可以使用 `for` 循环来处理，即下面这两行代码：

```
for root, dirs, files in os.walk(directory):
    for file_name in files:
```

第一行用来遍历目录，三个返回元素分别表示每次遍历的路径名 (`root`)、目录列表 (`dirs`) 和文件列表 (`files`)。由于本程序只关心文件，这里只有一个内层循环来遍历文件。`os.walk()` 会自动遍历所有子文件夹，不需要亲自动手。

还有些选项可以用来限定文件的选择范围。最基本的一个是根据文件类型，`-i`、`-c` 和 `-d` 分别用来限定图像，代码和文档。`-a` 覆盖了这三个选项，并选择所有类型文件（默认值）。这些选择并不是完美的，它只是根据文件类型字典来工作：

```
file_types = {"code": [".py"],  
             "image": [".jpeg", ".jpg", ".png", ".gif"],  
             "document": [".doc", ".docx", ".odt", ".rtf"]}
```

如果选择了相应的文件类型，则会备份其中列出的所有扩展名的文件。下面这行代码用来得到文件的扩展名：

```
name, extension = os.path.splitext(root, file_name)
```

函数 `os.path.splitext()`（注意这里是 `split-ext` 而不是 `split-text`）将文件名分为两部分，分开返回两个值。第一个是主文件名（即 `name` 变量），剩下的一个是扩展名。（存储在 `extension` 变量中）。剩下要做的就是检查扩展名是否包含在 `backup_types` 中。这里我们将所有指定的文件类型都集中到了 `backup_types` 里。指定文件类型之前，先确保该集合包含了所有需要备份的文件类型。

`-r` 或 `--regex` 可以用来指定正则表达式，只有匹配的文件名才会备份。下面这行给出了代码：

```
if not options.regex or re.match(options.regex, file_name):
```

注意这里使用了 `re.match()` 而不是 `re.search()`。这意味着整个文件名必须匹配该正则表达式。例如，正则表达式 `".*\.\py"` 将会匹配所有扩展名为 `.py` 的文件（和使用 `-c` 选项效果相同）。

最后一个选项是 `-m` 或 `--mtime`，即修改时间（modified time）的简写。换句话说，它将备份所有最近修改时间小于指定天数的文件。代码实现比较复杂：

```
modified_days = (datetime.datetime.now() -  
                  datetime.datetime.fromtimestamp(  
                      os.path.getmtime(  
                          os.path.join(root, file_name)))).days  
if mtime_option < 0 or modified_days < mtime_option:
```

其中主要工作都是由 `os.path.getmtime()` 来完成的。这里需要使用绝对路径（即 `/home/pi/filename` 而不只是 `filename`），它将返回文件的最后修改时间。

`os.path.join()` 需要两个参数：路径和文件名，将其合成为前面函数所需的参数（不能简单地将两个字符串合并起来，因为路径有时以 `/` 结尾，有时不是）。

`os.path.getmtime()` 返回的时间并不是标准时间，而是自 1970 年 1 月 1 日以来经过的秒数（这是 Unix 系统标准，在 Linux 系统中也是如此）。因此为了计算出文件的最后修改时间到现在过了几天，我们首先需要使用 `datetime.datetime.fromtimestamp()` 将其转换为 Python 的 `datetime`，然后再减去当前时间。

完成这些之后，剩下的就是比较计算结果是否小于选项中指定的天数。将 (`mtime_option < 0`) 作为 if 中的起始子句是因为如果程序在处理用户输入时出错或者用户没有使用 `mtime` 选项，默认指定 `mtime_option` 是 -1。

这些都是限定文件选择范围的选项。确定之后，剩下的事情就是将其加入 tar 文件。这些由 `tarfile` 模块完成，它提供了一个简单的处理归档文件的方法。你只需要像下面的代码一样，在开始的时候打开文件就够了：

```
backup_tar = tarfile.open(backup_file_name, "w:gz")
```

第二个参数 (`w:gz`) 表示以写方式打开文件，并且指定这是个 `gzipped` (即，压缩) 文件。下面一行代码用来将指定的文件添加到归档中：

```
backup_tar.add(os.path.join(root,file_name))
```

如果指定了相应选项，你应该认识使用 `scp` 复制文件到远程服务器的代码。

这段程序只是展示了一个让计算机保持有序的脚本。它可以用来完成烦琐的备份工作，然而，你还是得记得去运行它来创建备份。这里 Python 帮不了你，但是有个叫做 `crontab` 的 Linux 特性可以在指定时间运行程序，换句话说就是类似使用者的时程表。该程序所有选项都在命令行中设置，你需要做的就是决定需要运行什么，并设置 `crontab` 在指定时间开始运行。

`crontab` 有两个主要选项：-l 列出当前设置的需要运行的程序，-e 打开文本编辑器来什么时间运行什么程序。每个任务占一行，由 5 个用空格隔开的数字或星号开始，命令跟在后面。这 5 个数字分别代表程序需要运行的分钟、小时，一个月中的第几天、月份和一个星期中的第几天，星号表示任意。比如下面的例子（由于排版原因，将其看做一行）：

```
0 0 1 * * python3 /home/pi/chapter10-backup.py -s  
pi@192.168.0.10:/home/pi/backups
```

它将会在每个月第一天的午夜备份你的 `home` 目录。下面的例子将会在每天中午运行：

```
12 0 * * * python3 /home/pi/chapter10-backup.py -s  
pi@192.168.0.10:/home/pi/backups
```

10.5 在 Python 中操作文件

在本章中，已经看到过很多操作文件的方法。然而你还没有真正在 Python 中打开任何文件来读写数据（除了 tar 归档，这是个特殊案例）。这一小节，你将会看到如何将信息存入文本文件，然后再从中读出。

实际上非常简单。你需要做的就是调用 `open()` 函数。例如，下面代码将打开 `myfile.txt` 文件，并把它的每一行都打印出来：

```
file_a = open("myfile.txt", encoding="utf-8")

for line in file_a:
    print(line.strip())

file_a.close()
```

`open()` 函数创建了 `file` 对象。它可以接受多个参数。最基本的就是文件名，编码格式也是一个很重要的参数，用来告诉 Python 文件的格式。大多数文本文件都使用 utf-8 格式，创建自己的文件时也需要使用该格式。



注意 注意不能使用 `file` 作为变量名，以为它在 Python 已经被用作其他用途。

打开文件之后，可以使用 `for` 循环来遍历它。这里唯一有点不常见的是我们在 `line` 后面使用了 `.strip()`。这是因为文件中的每一行都包含有换行符，被打印出来时 `print` 函数又自己添加了新的换行符，因此没有这个函数，打印出来的每两行之间都会多出一个空行。

每打开一个文件，都需要关闭它。这里给出一个自动关闭文件的例子：

```
with open("myfile.txt", encoding="utf-8") as file_a:
    for line in file_a:
        print(line.strip())
```

这两段代码所做的工作都一样。`with` 语句在代码段结束之后会自动关闭文件。如果你容易忘记关闭文件，这将会非常有用。

写文件也很简单。只需要在 `open()` 的参数中加入 `mode="w"` 后就可以写文件了。看看下面的代码：

```
with open("myfile.txt", mode="w", encoding="utf-8") as file_a:
    for letter in "abcde":
        file_a.write(letter + "\n")

with open("myfile.txt", encoding="utf-8") as file_a:
    for line in file_a:
        print(line.strip())
```

它将重写 `myfile.txt`。如果希望在文件末尾添加内容，可以使用 `mode="a"`（附加，`append`）。它将保持文件原有内容不变，将新内容添加到文件末尾。

10.6 小结

读完本章，你应该更好地理解以下内容：

- 树莓派上运行的是一个 Linux 版本的系统。
- Linux 的文件系统和 Windows 不同，是基于根目录，`/` 的文件系统。
- Linux 具有完整的文本模式。
- 使用 `subprocess` 模块可以在文本模式下运行命令。
- Linux 命令可以输出到 `stdout` 和 `stderr`。
- 写 Python 脚本时，如果可能，从命令行的标记中获取全部输入非常有用。
- 正则表达式可以用来匹配文本模式。
- `scp` 可以用来在不同计算机间复制文件。
- `os` 模块提供了很多函数来帮助你和操作系统间交互（远超过我们这里的覆盖范围，查看 Python 文档可以获得更多信息）。
- `open()` 函数可以为读写打开文件。

硬件接口

树莓派 GPIO 口连接到 Arduino 时，可以这样：

和大多数计算机不同，树莓派具有一组通用输入输出口（GPIO），通过它们可以与外界交互。SD 卡边上突出的一组金属针就是 GPIO 口。你可以将它们当做可编程开关控制其他事务，或者用它们来从外界接收信息。简单来说，你可以使用它们对树莓派进行各种各样的扩展。它们的应用范围很广，数字艺术家使用它们来创建交互式显示，机器人建造者使用它们来提升自己的作品。想象一下，使用树莓派的 GPIO 和一些组件几乎可以做任何事情。

11.1 硬件设置选择

在开始构建电路之前，你首先需要知道如何连接树莓派 GPIO 口。由于你不能直接将电线连接到 GPIO 口中（实际上你可以直接将它们焊在一起，但我们并不推荐这么做），下一节中我们将介绍连接它们的一些方法。

11.1.1 母转公接头

这应该是最简单的选择。这种接头一端可以连接到 GPIO 口上，另一端可以连到面包板上。这是访问 GPIO 口的最简便方法。本书给出了该方法的图片。可以只连接一部分管脚，如果一次访问大量管脚则有可能造成混乱。参见图 11-1。

Pi Cobbler (树莓派 GPIO 扩展板)

Pi Cobbler (GPIO 扩展板) 可以简单地将 GPIO 管脚连接到面包板上 (参见图 11-2)。和使用转接线相比, 它并没有提供任何新特性, 只是看起来更整洁, 也不容易混淆管脚。

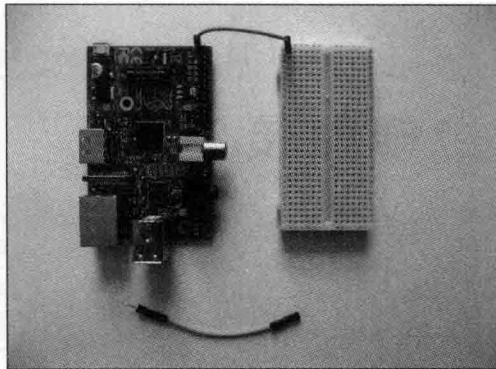


图 11-1 转接头的母头可以连接到 GPIO 管脚，公头可以插入面包板

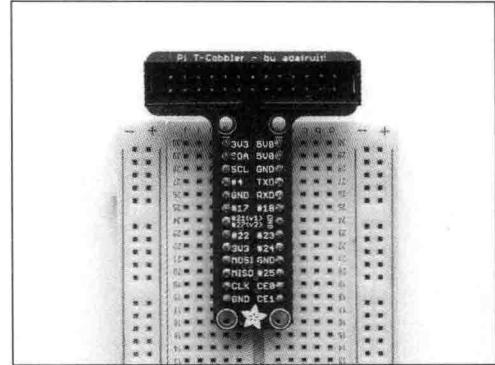


图 11-2 Pi Cobbler 将 GPIO 端口全部连接到面包板上，避免了连线缠绕和混淆
(Adafruit Industries)

11.1.2 无煌面包板

无论你选择哪种方式，都需要使用面包板。它提供了一种简单的方式，可以快速地各个组件的电路连接在一起，完成之后可以轻松拆除。面包板有不同的尺寸，但基本排列都类似。典型的面包板的长边有两条平行接口，可以用来提供正负电源。它们中间是两排插孔，中间留有间隙。一般将与长边垂直的每 5 个孔板用一条金属条连接。板子中央的一条凹槽，是针对需要集成电路、芯片试验而设计的（见图 11-3）。图中，我们用了一个小型无焊面包板，它没有正负电源槽。你可以使用任何尺寸的面包板。如果对电子感兴趣，很有必要购买一个全尺寸的面包板，在大部分项目中都会用到它。

你可以将原件直接插入孔中，使用公对公连接线或者小段单芯线缆将各个元件连接起来。

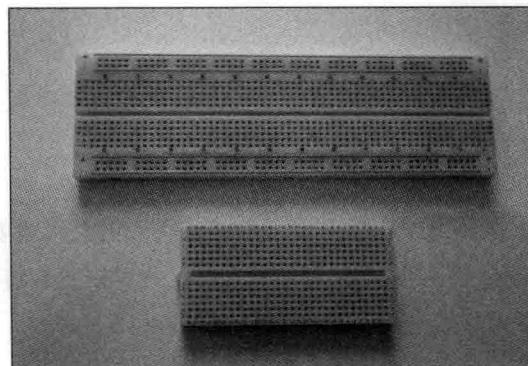


图 11-3 使用无焊面包板可以轻松搭建电路原型，拆除之后还可以使用相同的原件构建新的电路

11.1.3 成品板和万能板

在无焊面包板上完成电路原型后，你可能希望将它们做成长成品，将所有元件永久固定在一起，远比无焊面包板耐用。这有点超出了本章范围，但如果希望进一步做得更完善，很快你就会发现自己需要使用成品板。

另一个选择是使用万能板。它上面有很多洞，你可以在上面焊接。各个洞之间没有连接，可以根据需要将其焊接在一起。

11.1.4 PCB 加工

最高等级的选择是制作自己的 PCB（印刷电路板，Printed Circuit Boards）。包括商业印刷在内，有许多方法可以做到这一点。这是整个设计全部完成之后的最终步骤。如果你希望完成这一步，Fritzing 里有些很好的资源（参见 <http://fritzing.org>）。它们提供软件帮助你设计板子，并提供打印服务。

11.2 辅助工具

不需要任何工具就可以搭建简单的电路（只要有了面包板和公对公跳线），但有了这些工具可以更轻松地搭建电路。

11.2.1 剪线 / 剥线器

从字面上很容易理解它们的用途。如果计划使用单芯电线来连接电路，就需要这个工具（通常这两个功能会集成到一个工具里）。然而如果已经给面包板上用了跳线，就不需要这个工具了。

11.2.2 万用表

万用表用来测量多种参数，包括电压、电流和电阻。它可以帮助定位问题电路。没有万用表，在遇到电路问题时，很难判断连线是否能够导通，或者原件是否坏掉。它还可以简单地判断电阻大小（本章稍后介绍）。

11.2.3 电烙铁

电烙铁可以将两个元件焊接在一起或将元件和电路板焊接在一起。在使用成品电路板或购买了需要焊接到树莓派上的附件时需要使用电烙铁。本章不会讲到焊接，如果需要这

么做，这个网站提供了一个很好的名为“轻松焊接”的指南：http://mightyohm.com/files/soldercomic/FullSolderComic_EN.pdf。

图 11-4 展示了这些工具。



图 11-4 搭建硬件所需工具集。不过对于本章讲到的工程，它们不是必需品。

11.3 本章所需的硬件

我们尽最大努力保证本章所需硬件尽可能地简单、易得，并且尽可能便宜。为了完成本章的工程，至少需要下列硬件：

- 发光二极管（LED）
- 电阻： 220Ω 、 $1.1k\Omega$ 和 $6.2k\Omega$
- 无焊面包板（任意尺寸）
- 面包板跳线（或者单芯电线和剪线钳）
- 一种连接树莓派和面包板的方式——母转公跳线或者 Pi Cobbler 均可
- MCP3008 芯片
- 按键开关
- 光敏电阻（LDR）

在后面的小节中将会用到这些硬件。

11.3.1 第一个电路

在深入电路细节之前，让我们先创建一个简单的电路。你需要一个是面包板，一个 LED，一个 220Ω 电阻，并将一些 GPIO 口连接到面包板上（使用母转公跳线或 Pi Cobbler）。

有了这个电路，就可以使用 Python 控制 LED（一种光源）的开关。可以将 LED 看做是小灯泡，但需要注意两点：首先，它的效率很高，正常使用时不会太热，其次，它是单向导通的。也就是说，它的两个引脚一个是正极，一个是负极，需要分别连接到电源正负极上。发光二极管管体内部金属极较小的是正极，大的片状的是负极[⊖]。

电阻在电路中起到限流作用，避免电流过大。通常情况下，在电路中至少需要一个最小为 220Ω 的电阻，否则就可能烧毁树莓派或者其他元件。其中的细节我们稍后再介绍。

在这个电路中，不一定要使用 220Ω 的电阻， 220Ω 至 470Ω 中间的任意阻值都可以（试一下大阻值的电阻，LED 将会变暗）。

根据电阻上的色环可以直接读出电阻值。通常有四到五个色环（ 220Ω 的通常有四个，分别是红、红、黑，然后是银色或金色）。同样的，稍后我们将解释它的含义。

如图 11-5 所示，将电阻和 LED 连接到面包板上。注意需要将 LED 的正极（内部金属极小的是正极）和电阻连在一起。

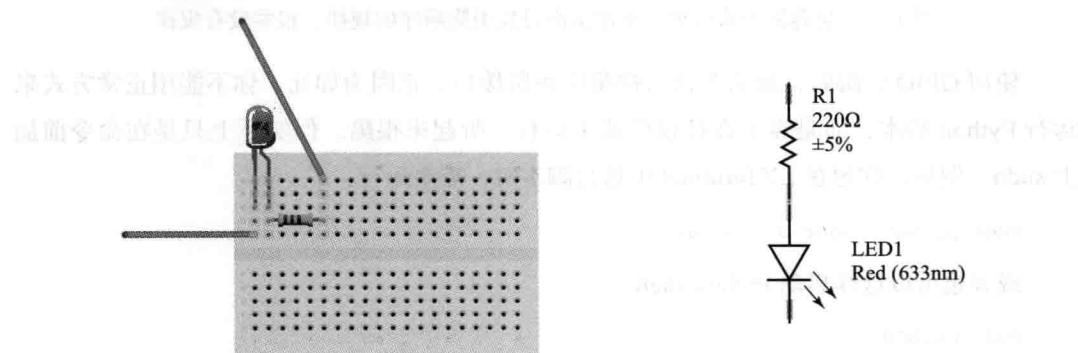


图 11-5 电路的两种图示。左边是物理连接图，右边是原理图

为了确保电路能工作，将电阻上的另一根线和树莓派的 3.3V 引脚相连（参见图 11-6），LED 的另一个引脚和接地引脚相连。如果连接正常，将会点亮 LED。

这次只是将树莓派用做电源。为了能够在 Python 中控制电路，先要安装 RPi.GPIO 模块。首先使用下列命令（在 LXTerminal 中，而不是 Python 中执行）安装了 pip（辅助访问模块的工具）：

```
sudo apt-get install python3-pip
```

然后使用这个命令获得需要的库（还是在 LXTerminal 中）：

```
sudo pip-3.2 install RPi.GPIO
```

[⊖] 适用于直插式 LED，也可以使用万用表判断正负极。——译者注

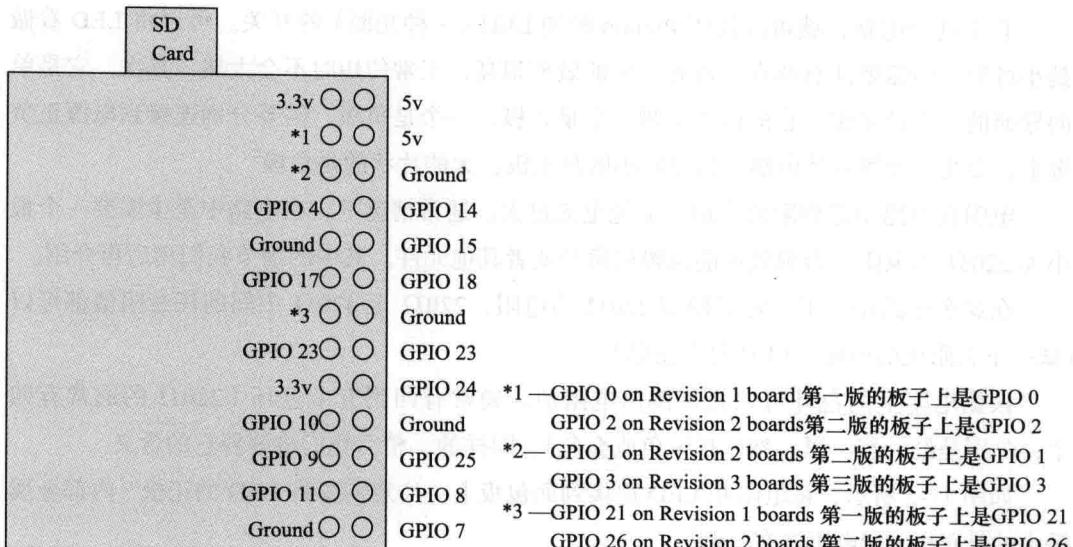


图 11-6 树莓派引脚排列。不用试图寻找引脚顺序的规律，根本没有规律

使用 GPIO 引脚时，就需要访问树莓派底层接口。正因为如此，你不能用正常方式来运行 Python 脚本，而是需要在特权模式下运行。听起来很酷，但实际上只是在命令前加上 sudo。例如，你想在 LXTerminal 中运行脚本时，需要运行：

```
sudo python3 your-script.py
```

或者也可以这样启动 Python shell：

```
sudo python3
```

或者你可以在特权模式下在 LXTerminal 中运行如下代码启动 IDLE3：

```
sudo idle3
```

安装完 RPi.GPIO 之后，只需要将电路连接到一个 GPIO 管脚。断开和 3.3v 引脚相连的线，并将其连接到第 22 角上。完成之后，打开一个 Python 会话（不要忘记 sudo！），并输入下列命令：

```
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>> GPIO.setup(22, GPIO.OUT)
>>> GPIO.output(22, True)
>>> GPIO.output(22, False)
>>> GPIO.output(22, True)
>>> GPIO.output(22, False)
```

可以看到，将引脚置为 True 将点亮 LED，置为 False 将熄灭 LED。电路结构参见图 11-7。

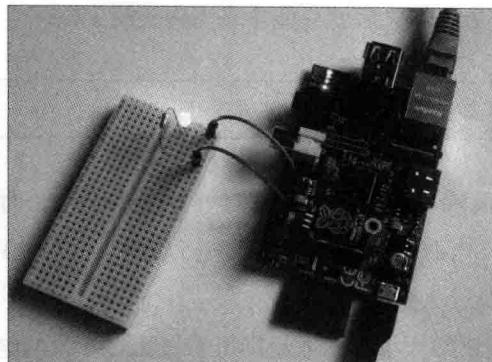


图 11-7 在小型无焊面包板上连接完成的电路

关于电路

基本上，一个电路包含三种要素——电源、负载和导电回路。电路通常都包含有这三个部分。如果只是将负载连接到电源上，而没有构成回路，负载将不会工作。如果直接将电源和地连接起来构成回路，将会发生短路，即产生强大的电流，并损坏电源（参见 11.3.2 节）。

电路结构有简有繁，从简单的电路（例如我们之前搭建的）到复杂系统（如计算机），都遵守同样的原理。

在本章中，限于篇幅我们只讨论如何使用树莓派的输入输出口，如果对电路有兴趣，可以参阅其他资料。

如果希望继续深入了解电路，Penguin Tutor 网站中提供了一个很好的教程，可以帮助你进一步理解它：www.penguintutor.com/electronics/。

11.3.2 保护树莓派

通常，很少出现由于编程导致的硬件物理损坏。编程可能损坏文件（这种情况同样也很少见），但通常情况下，无论将事情搞得多糟，重装系统一般都可以恢复。然而，一旦在 GPIO 口上连接了其他硬件，就有可能直接给 CPU 供电，也就可能损坏它。有两种电学特性可能会引起问题，电压和电流。

关于电压，规则简单也很重要。永远不能将 3.3V 电压连接到 GPIO 口上。这一点非常重要，也是我们要特别注意的地方。有很多电子器件都设计成使用 5V 电压，因为其他的处理器使用这个电压。然而，一但将其直接连接到树莓派上，将给板子带来无可挽回的

损失。结果就是变砖，因为损坏之后就只能拿来当砖头用了。



注意 永远不能在 GPIO 口上连接大于 3.3V 的电压。

注意到，树莓派有 5v 端口。它只是用来对外界提供电源，而不能用于自身。如果需要在 GPIO 口连接使用 5v 电源的设备，就需要一个逻辑电平转换器。大概几英镑就可以买到，用来将 5v 信号转换成 3.3v，反之亦然。

电压可以看做电力能量的大小，而电流则可看做流过导线的电力总量。欧姆定律将这两个量联系起来，即：

$$\text{电压} = \text{电流} \times \text{电阻}$$

或者，换一种形式：

$$\text{电流} = \text{电压} / \text{电阻}$$

电流的单位是安培 (A)，电压的单位是伏特 (V)，电阻的单位是欧姆 (Ω)。在前面的电路中，电压是 3.3V，电阻是 220Ω ，因此：

$$\text{电流} = 3.3/220=0.015\text{A} \quad \text{或者} \quad 15\text{mA}$$

树莓派的任何一个引脚最大只能接受 16mA 的电流，加到所有 GPIO 引脚的电流综总和不能超过 50mA。这意味着同时只能点亮三个 LED (或者使用更大的电阻限制通过每个 LED 的电流)，也就意味着除非和大于或等于 220 欧姆的电阻相连，否则不能直接将 GPIO 引脚接入电路。如果无法确定电阻大小，从安全角度考虑，推荐使用较大的电阻。

从技术方面看，上述说法并非完全正确，因为 LED 和其他原件有点不同。LED 两端会产生一个小的电压降。然而，在理解 LED 两端电压衰落的原理之前，还是先遵守上面的规则好了。

如果需要使用更大的电流，或者只是想保护树莓派免受瞬间大电流冲击，可以使用包含输入输出端口缓冲的扩展板，如 PiFace，或者使用集成缓冲电路 (集成电路或者芯片) 来保护 GPIO 接口。

11.3.3 电源限制

树莓派能够提供的总电流受限于它能接受的总电流。尤其是当树莓派上连接了很多像光电鼠标和 USB 存储器的其他设备时，有些供电口会尽可能地提供更多的电源。

如果在使用 GPIO 口时发现树莓派变得不稳定或者开始自动关机，电源不足可能是主要原因。

为了解决问题，可以升级电源适配器以获得更大电流，或者减少树莓派的输出总电流，如在 raspiconfig 中减少超频或者移除不必要的外围设备。

11.3.4 获得输入

在上一个例子中，我们使用 GPIO 口来开关 LED，用到的是 GPIO 的输出功能。现在，我们来看下输入功能。继续使用全面的电路，不用拆除，但是需要添加一个按键。

按键开关是个很简单的原件。打开时（即没有按下）它的两个引脚是断开的，因此没有电流可以通过。按下时，它将会接通两个引脚，和电线一样开始导电。

和之前看到的电路一样，我们需要一个电阻来防止电路中出现过大的电流。为了更安全，这里我们只用了 1.1k 欧姆的电阻。

然而，如果你只是将电源连接到电阻上，再连接按键，然后是 GPIO，当开关按下时，仍然不会构成电路。如果没有电路，GPIO 口也就不能感应到开或关（如果愿意，也称作 True 或 False）。当开关打开时我们需要一个电路将 GPIO 口拉低到地电平（即 False）。这就是著名的下拉电阻，因为在没有任何电路连接到 GPIO 口上时它将 GPIO 口拉低到 0V。它需要远大于其他电阻以保证开关闭合时有足够的电势输入 GPIO 口。我们使用了 6.2k 欧姆的电阻，当然 10k 欧姆也可以。

图 11-8 给出了电路连接图。注意 GPIO 和地之间的电阻相对较大。如图所示，按键应该位于 3.3V 电源和 GPIO4 之间。

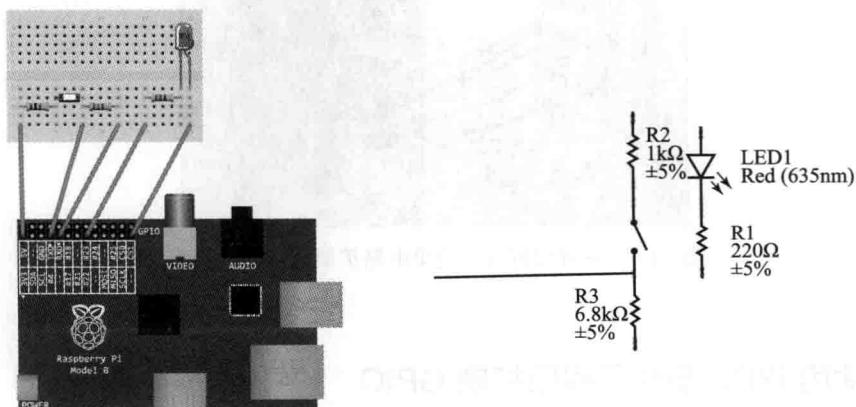


图 11-8 电路图，可以帮助你对电路的理解

设置好之后，下面的代码将创建一个简单的反应游戏。它将等待一个随机时间，然后点亮 LED。LED 点亮之后你需要迅速按下按键，然后它将给出你的反应时间。参见网站文件 chapter11-reaction.py。

图 11-9 给出了游戏运行的效果。

```
import RPi.GPIO as GPIO
import time
import random
from datetime import datetime

GPIO.setmode(GPIO.BCM)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(4, GPIO.IN)

GPIO.output(22, False)
random.seed()

while True:
    time.sleep(random.random()*10)
    start = datetime.now()
    GPIO.output(22, True)
    while not GPIO.input(4):
        pass
    print("Your reaction time: ",
          (datetime.now() - start).total_seconds())
    print("Get ready to try again.")
    GPIO.output(22, False)
```

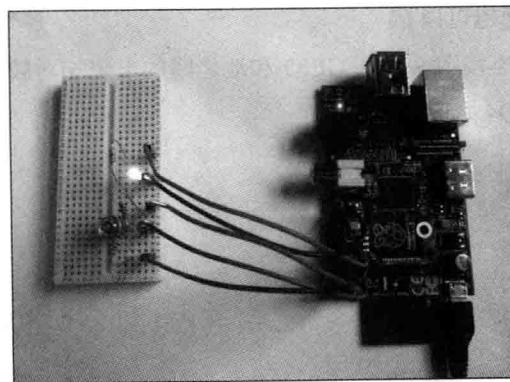


图 11-9 你可以将这个简单电路扩展成一个大工程

11.4 使用 I2C、SPI 和串口扩展 GPIO

两个计算机需要建立连接是，可以使用以太网。这是个系列标准，定义了很多事情，

如使用的物理线缆、计算机的网络寻址方式。只要计算机兼容以太网，它们就可以相互通信。

除此之外，还有一些为小尺寸硬件，如不同的芯片，制定的通信协议。在本章中我们只要介绍三种主要的协议——SPI、I2C 和串口。

11.4.1 SPI 通信协议

SPI 或者串行外设接口（Serial Peripheral Interface）使用 4 根线在两个或多个设备间来提供双向通信——一个主设备（通常是树莓派）和一个或多个从设备（典型代表就是芯片）。四根线包括用来保持设备同步的时钟线，主设备输出从设备输入线（MOSI），主设备输入从设备输出线（MISO），和从设备选择线[⊖]。

下面我们开始将树莓派和从设备的相应引脚连接起来。

SPI 可以提供很多的扩展选则，例如模拟输出。

树莓派有很多输入和输出口，但这些都是数字口。也就是说，它们只能读出开或者关。有些时候已经够用了，如按键和控制 LED，但有时你需要量化地读写具体数值。例如，你可能需要从传感器，如光线或者温度传感器中读入数据。这些数据处于一个范围内，而不是开或关的数值。

这些（位于某个范围内的）值被称为模拟量（相对于数字量）。为了读取它们，你需要一个模拟数字转换器（ADC）。本章中，我们使用 MCP3008，它提供了 8 个模拟通道，和树莓派之间使用 SPI 通信。

图 11-10 给出了 MCP3008 的引脚说明，图 11-11 给出了电路连接示意图。

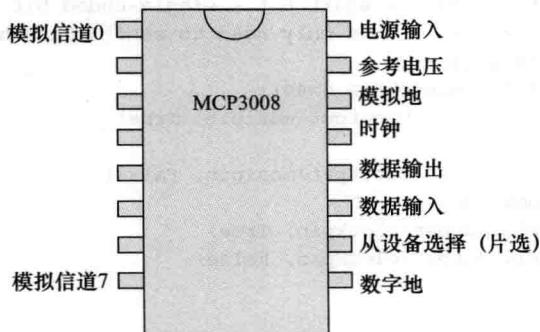


图 11-10 自上至下，芯片的左边 8 个接口分别表示 0~7 的模拟输入，右边 8 个接口用于芯片控制。在塑料封装的上面有一个半圆形缺口，表示芯片上端，在对照图片时需要对准该缺口

[⊖] SS，也称作片选线 CS。——译者注

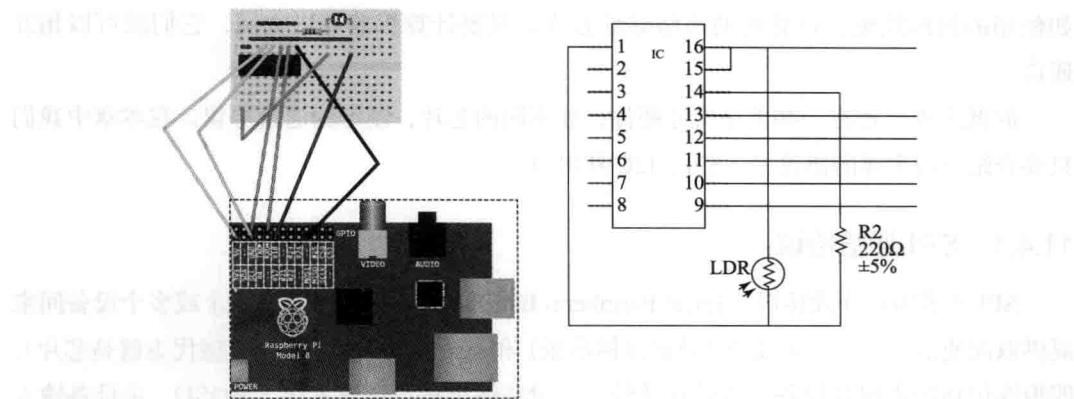


图 11-11 电路连线。改变 MCP3008 连接的 GPIO 口时需要在程序中修改相应的值

代码如下所示，参见网站文件 chapter11-spiadc.py。图 11-12 给出了连接好的电路。

```

import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

# read SPI data from MCP3008 chip, 8 possible adc's (0 thru 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
    if ((adcnum > 7) or (adcnum < 0)):
        return -1
    GPIO.output(cspin, True)

    GPIO.output(clockpin, False) # start clock low
    GPIO.output(cspin, False) # bring CS low

    commandout = adcnum
    commandout |= 0x18 # start bit + single-ended bit
    commandout <= 3 # we only need to send 5 bits here
    for i in range(5):
        if (commandout & 0x80):
            GPIO.output(mosipin, True)
        else:
            GPIO.output(mosipin, False)
        commandout <= 1
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)

    adcout = 0
    # read in one empty bit, one null bit and 10 ADC bits
    for i in range(12):
        GPIO.output(clockpin, True)
        GPIO.output(clockpin, False)
        adcout <= 1
        if (GPIO.input(misopin)):
            adcout |= 0x1

```

```

    GPIO.output(cspin, True)

    adcout >>= 1           # first bit is 'null' so drop it
    return adcout

SPICLK = 4
SPIMISO = 17
SPIMOSI = 18
SPICS = 23

# set up the SPI interface pins
GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN)
GPIO.setup(SPICLK, GPIO.OUT)
GPIO.setSetup(SPICS, GPIO.OUT)

ldr_adc = 0;
last_read = 0
tolerance = 5

while True:
    # we'll assume that the light didn't change
    input_changed = False
    # read the analog pin
    ldr_value = readadc(ldr_adc, SPICLK,
                        SPIMOSI, SPIMISO, SPICS)
    ldr_movement = abs(ldr_value - last_read)

    if ( ldr_movement > tolerance ):
        input_changed = True

    if ( input_changed ):
        print('Light = ', int(ldr_value))
        last_read = ldr_value

    # hang out and do nothing for a half second
    time.sleep(0.5)

```

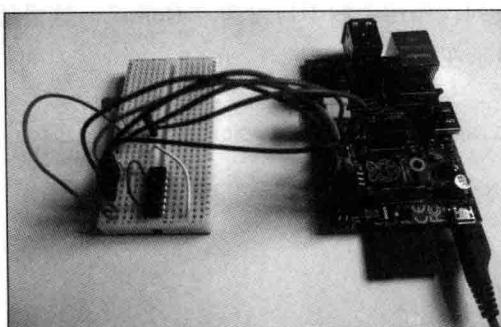


图 11-12 使用 MCP3008 将可变电阻 (LDR) 的模拟信号转换成数字信号输出到树莓派

不用太关心 `readadc()` 函数是如何工作的。它能操作单个字节并将其通过 MOSI 和 MISO 线传送和接收。如果希望增加几路输入，可以使用这个函数来读取 ADC 的其他端口。RPi.GPIO 库计划支持 SPI，但本书写作时，这部分还没有实现。可以通过下列网站查询项目进度 <http://code.google.com/p/raspberry-gpio-python/>。

主循环使用 `readadc()` 函数来获得与 ADC 相连的 LDR 值，并（如果数值变化超过门限）将其输出到屏幕上。

11.4.2 I2C 通信协议

内置集成电路（I2C 或者 I²C，发音为“I 方 C”）是个比 SPI 还要强大的协议。它仍然使用 4 根线，但其中一个是电源线，另一根是底线，因此只有两根数据线。I2C 总线上最多可以连接 127 个设备，它具有寻址能力，而不是像 SPI 那样使用简单的从设备选择（片选）信号。

Quick2wire 制作了一系列的 I2C 板子用来扩展树莓派的功能。虽然它们可以简化处理过程，但是要使用 I2C 并不需要这些板子。它们还开发出了 Python 模块用于 I2C 通信，即使不适用它们的板子也可以使用该模块。模块可以从下面网站上找到 <https://github.com/quick2wire/quick2wire-python-api>。

和 SPI 类似，RPi.GPIO 有计划支持 I2C。请查看前面给出的网站获得最新信息。

11.4.3 串口通信协议

前面两种协议通常用于在设备间传输二进制数据。串口通信，则着手于另外一方面，通常用于传输文本信息（当然这两者之间也有例外）。

`pyserial` 模块用于支持串口通信，可以通过下列命令获得该模块：

```
sudo pip install pyserial
```

`pyserial` 模块安装完成后，就可以创建串口连接，然后使用 `write()` 和 `read()` 方法来发送和接收数据。例如，下列代码用于向串口发送“Hello”。这里使用了 Ciseco PiLite，将会把接收到的字符滚动显示在屏幕上。

```
>>> import serial
>>> pilite = serial.Serial("/dev/ttyAMA0", baudrate=9600)
>>> pilite.open()
>>> pilite.write("Hello", "utf-8")
```

11.5 深入研究

如果希望继续深入研究这个例子，尝试一下随后给出的想法。

11.5.1 Arduino

如果业余喜欢电子制作，最有用的工具套件可能就是 Arduino。它拥有比树莓派略大的微控制器板。其中含有更多的 GPIO 口（具体数目和模型有关），并且还有模拟输入，有些情况下还有模拟输出。

当然，Arduino 最有用的功能还是可以插入主板的扩展板（称为 Shield）。使用它们可以用少量的连线快速搭建出强大的硬件。

树莓派的 GPIO 和 Arduino 之间可以使用 I2C 或者 SPI 连接起来。不过大多数 Arduino 模型使用 5v 电压，连接时需要使用逻辑电平转换器。它们也可以通过 USB 使用串行连接通信。

有人认为连接到树莓派上之后，Arduino 的功能很难发挥出来。事实上，几乎所有在 Arduino 上完成的工作都可以在树莓派上完成。然而，大量已存的 Arduino 硬件和代码使得它们成为对树莓派非常有用的伙伴。

然而这些代码使用的是 C++ 而不是 Python，这可能是对本书读者最大的不利条件。该语言的格式和 Python 不同，但其基本理论是一致的。虽然我们在这里没有打算介绍它，但如果深入阅读过相关书籍，理解基本的 C++ 应该不成问题。

11.5.2 PiFace

PiFace 是一个可以直接插入树莓派 GPIO 口的扩展板，对于扩展 GPIO 非常有用。不但可以缓冲输入输出（保护树莓派并提供 500mA 电流），还提供一组中继用于驱动更耗电的元件比如马达。

它的尺寸和树莓派相同，可以直接叠在树莓派上方。用做简单的机器人是个很好的选择。个人购买该元件并不便宜，大约 25 英镑（折合 41 美元），这和它的易用性和用于支持的工程有关。图 11-13 展示了 A 型树莓派连接上 PiFace 之后的效果。

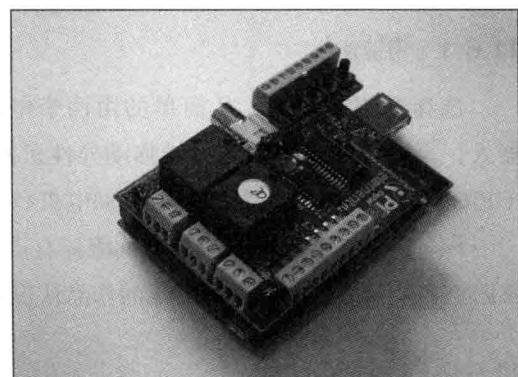


图 11-13 使用 PiFace 的输出缓冲可以控制更多东西而不用担心树莓派的小电流限制

11.5.3 Gertboard

Gertboard 是扩展树莓派 GPIO 口的巨大。它在一个板子上塞满了所有你想要的东西。

它拥有类似 ATMega 芯片的微控制器可以执行板外程序，马达控制器、数模转换器、模数转换器、按键开和一打 LED。当然，这些都需要付出成本，不只是价格（约合 60 英镑 /98 美金），还包括尺寸。由于尺寸太大，很多项目并不适合使用该扩展模块（例如机器人）。

对于学习电子和控制来说，这块板子很合适。一旦有了好主意，可以立即搭建电路实践自己的工程。

这块板子由树莓派基金会硬件最高领导人 Gert Van Loo 设计，完全值得信任，因为设计它的是真正从内到外（正确地）了解树莓派的人。

11.5.4 Wireless Inventor's Kit

到现在为止，我们所做的一切都需要线缆来连接各个组件。对于大多数项目来说都没问题。然而，有些项目需要更自由点。Ciseco 集成了 Wireless Inventor's Kit，其中包含了使用树莓派做简单无线通信需要的所有东西。

它使用的无线系统工作在比 WiFi 网络低的频率，可以用作传感器和树莓派间的无线传输，或者使用树莓派无线控制电路。

11.6 尝试一些流行工程

使用树莓派和现成的组件几乎可以做成任何东西，为了帮助你起步，接下来的小节给出了一些想法。

11.6.1 机器人

选择范围非常大，从简单的由两个电机驱动的轮式设备，到复杂的拟人行走的机器人，再到像蛇或者蜘蛛一样移动异性机器人。开始制作机器人先要学习如何控制电机（GPIO 口提供的电能不足以直接驱动电机）。PiFace 有一对继电器，可以从那里开始。

你可能还需要一些伺服器。伺服器有点类似于电机，但可以按角度转动。还可以在树莓派上安装很多传感器来帮助你的作品认识周围的世界。

11.6.2 家庭自动化

想象一下可以使用智能手机控制暖气和灯光的世界会是什么样子。这不再是个不可实现梦想——使用树莓派、一些电路和 Python 就可以完成。第 7 章展示了如何通过互联网控制 Python 程序，再结合本章学到的知识，你就可以梦想变为现实。

11.6.3 防盗报警器

这个也许没有家庭自动化那么吸引人，但它能保证你的安全。使用树莓派摄像头和一些像被动红外（PIR）移动传感器之类的东西，你可以建造自己的 Fort Knox[⊖]。

11.6.4 数字艺术

艺术不再是在脏暗的房间里无生气地绘画，也不再局限于雕塑、诗歌或者舞蹈。它可以是任何你需要的形式。其中一个新兴形式就是数字艺术。例如，添加变幻的光线，或者压力感应平板电脑，当人们触摸它时会产生反馈。发散一下思维，然后使用树莓派为你的想象植入生命。

11.7 小结

读完本章，你应该更好地理解以下内容：

- 树莓派角落上突出的接口是通用输入输出口（GPIO），在 Python 中可以对其编程。
- RPi.GPIO 库为开关 GPIO 口或者从中读取数值提供了一个简单的接口。
- 使用 GPIO 时需要格外小心，因为输入电压或电流过大会不可逆地损坏树莓派。
- 可以使用 SPI、I2C 和串口添加设备扩展树莓派。
- MCP3008 芯片就是一个例子，它提供了八路模数转换通道，可以使用 SPI 读取数据。
- 树莓派几乎可以控制任意复杂的电路，唯一的限制就是你的想象力。

[⊖] 美国北部军用地。——译者注

测试与调试是软件开发过程中必不可少的环节。通过本章的内容，你将学会如何编写健壮的代码，以及如何有效地进行单元测试、集成测试和系统测试。

在本章中，你将学习到如何编写健壮的代码，以及如何有效地进行单元测试、集成测试和系统测试。通过本章的内容，你将学会如何编写健壮的代码，以及如何有效地进行单元测试、集成测试和系统测试。

Chapter 12

第 12 章

测试与调试

很多时候代码运行的结果和预期的不一样。检查输出，走读代码，但有时还是找不出异常输出的原因。这是编程中遇到的最让人恼火的事情。

通过本章的内容，你将学会如何编写健壮的代码，以及如何有效地进行单元测试、集成测试和系统测试。

12.1 通过打印变量调查故障

很多时候代码运行的结果和预期的不一样。检查输出，走读代码，但有时还是找不出异常输出的原因。这是编程中遇到的最让人恼火的事情。

有很多方法可以找出究竟发生了什么问题，但最简单的方法就是使用 `print()` 语句。使用该语句将每个变量值打印出来，通常可以找出底层发生的问题。

下面一段代码建立了一个简单的菜单系统。运行时没有报错，但无论输入什么，它都提示“Unknown choice”。(参见网站文件 `chapter12-debug.py`。)

```
choices = {1:"Start", 2:"Edit", 3:"Quit"}  
for key, value in choices.items():  
  
    print("Press ", key, " to ", value)  
  
  
    user_input = input("Enter choice: ")  
  
  
    if user_input in choices.values():  
  
        print("You chose", choices[user_input])
```

```

else:
    print("Unknown choice")

```

你可能已经看出问题在哪里，但如果还没有发现问题，怎样才能最好地找到它呢？现在的问题是当 user_input 有效时，if 语句没有正确地判断出来，因此我们添加一些 print() 语句来看看究竟发生了什么：

```

choices = {1:"Start", 2:"Edit", 3:"Quit"}

for key, value in choices.items():
    print("Press ", key, " to ", value)

user_input = input("Enter choice: ")

print("user_input: ", user_input)
print("choices: ", choices)
print("choices.values(): ", choices.values())

if user_input in choices.values():
    print("You chose", choices[user_input])
else:
    print("Unknown choice")

```

问题马上就出现了：choices.values() 应该换成 choices.keys()。代码用错了 choices 字典中的字段。修改代码然后运行。问题解决之后，程序应该能够正常运行。

然而，它还是不工作。一定还有其他错误。再看一下 print 语句的输出：

```

user_input: 1
choices: {1: 'Start', 2: 'Edit', 3: 'Quit'}
choices.values(): dict_values(['Start', 'Edit', 'Quit'])

```

你能看出为什么失败了吗？除了变量的值，第二个关键问题是值的数据类型，因此我们来扩展 print 语句，在其中加入更多细节：

```

print("user_input: ", user_input)
print("choices: ", choices)
print("choices.values(): ", choices.values())
print("type(user_input): ", type(user_input))
for key in choices.keys():
    print("type(key): ", type(key), "key: ", key)

```

如果你运行这些代码，应该会输出：

```
Press 1 to Start
```

```
Press 2 to Edit
```

```
Press 3 to Quit
```

```
Enter choice: 1
```

```
user_input: 1
```

```
choices: {1: 'Start', 2: 'Edit', 3: 'Quit'}
```

```
choices.values(): dict_values(['Start', 'Edit', 'Quit'])
```

```
type(user_input): <class 'str'>
```

```
type(key): <class 'int'> key: 1
```

```
type(key): <class 'int'> key: 2
```

```
type(key): <class 'int'> key: 3
```

```
Unknown choice
```

现在可以看出问题在于 `user_input` 是一个字符串，但是 `choices` 中的 `keys` 是整数。最简单的解决办法就是改变 `choices`，将其中的 `keys` 变成字符串类型：

```
choices = {"1": "Start", "2": "Edit", "3": "Quit"}
```

现在程序的基本逻辑已经工作正常；然而，它会输出一些用户不希望看到的文字。显然，删除 `print()` 语句可以解决问题，但是将来可能还会用到它们。可以将它们留下来，使用一个标记来开关它们，例如：

```
debug = True
```

```
if debug:
    print("user_input: ", user_input)
    print("choices: ", choices)
    print("choices.values(): ", choices.values())
    print("type(user_input): ", type(user_input))
    for key in choices.keys():
        print("type(key): ", type(key), "key: ", key)
```

```
choices = {"1": "Start", "2": "Edit", "3": "Quit"}
```

```

for key, value in choices.items():
    print("Press ", key, " to ", value)

user_input = input("Enter choice: ")
if debug:
    print("DEBUG user_input: ", user_input)
    print("DEBUG choices: ", choices)
    print("DEBUG choices.values(): ", choices.values())
    print("DEBUG type(user_input): ", type(user_input))

    for key in choices.keys():
        print("DEBUG type(key): ", type(key), "key: ", key)

if user_input in choices.keys():
    print("You chose", choices[user_input])
else:
    print("Unknown choice")

```

12.2 通过测试发现故障

调试是一个修正程序存在的问题的过程。这是个很有挑战性的工作，而发现第一处问题尤为困难。听起来有点不可思议，但是事实就是这样。随着程序不断变大，它的应用场景也开始增加，应用场景增加后，遇到问题的地方也会增加。

想象一下，例如，一个字处理程序拥有样式、页面布局、文件格式、布局管理等功能，每个部分都可能存在潜在的问题。因此对于开发者来说，保证每个功能都能够正常运行非常重要。问题也可能隐藏在各模块交界处。例如，可能只有在某个排列下使用了某个字体才有问题。

12.2.1 使用单元测试检查代码片段

单元测试是最基本的测试程序，是指对代码中小片段进行检查和验证。通常用来验证单个方法以保证功能正常。

本质上来说，所有的单元测试都是使用一组特定的输入来运行一段代码，然后检查输出是否正确。

例如，一个将输入的字符串转换成大写字母并返回的函数，可以这样实现并测试：

```
def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.isupper():
            output_string = output_string + character
        else:
            output_string = output_string + chr(ord(character) - 32)
    return output_string

print(capitalise("helloWorld"))
```

程序能够正常工作，是因为 Python 使用和数字相同的存储方式来存放 UTF-8 的字符，大写字母在 ASCII 表中比小写字母靠前 32 个位置。

在这个例子中，我们使用一个简单的测试用例，将结果打印到屏幕上，接受人工检查。我们也可以像下面的例子一样使用 unittest 模块让 Python 替我们检查结果。

```
import unittest

def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.isupper():
            output_string = output_string + character
```

```

    else:
        output_string = output_string + chr(ord(character)-32)

    return output_string

class Tests(unittest.TestCase):
    def test_1(self):
        self.assertEqual("HELLOWORLD", capitalise("helloWorld"))

if __name__ == '__main__':
    unittest.main()

```

这段代码和前面的代码差不多。运行之后，它将会检查字符串“helloWorld”是否变成“HELLOWORLD”。在这个水平上看，它和将结果输出到屏幕上没什么区别。

运行 `unittest.main()` 时，Python 将运行子类 `unittest.TestCase` 中的所有以 `test_` 开头的方法。在这个例子中，它将运行 `test_1`。使用单元测试的优势在于可以将一组测试组合起来运行，能够很快检查出程序是否正常工作。

你可以再添加一个测试用例来检查字符串“hello world”是否变成大写的“HELLO WORLD”：

```

def test_2(self):
    self.assertEqual("HELLO WORLD", capitalise("Hello World"))

```

运行之后会得到下列输出：

```

FAIL: test_2 (__main__.Tests)
-----
Traceback (most recent call last):
  File "capitalise.py", line 17, in test_2
    self.assertEqual("HELLO WORLD", capitalise("Hello World"))
AssertionError: 'HELLO WORLD' != 'HELLO\x00WORLD'
- HELLO WORLD
?
```

```
+ HELLOWORLD
```

```
?      ^ [Err] 42: syntax error at or before "Hello World"
```

看起来测试失败了。可以看到对空格的处理出错了。回头看下代码，我们发现问题出在我们从字符串内所有的字符包括非小写字符的 UTF-8 值中减了 32。因为空格不是小写字母，本不应该处理，但仍然被减了 32。

设计测试用例时应当尝试使用大范围的有效输入。例如：

```
class Tests(unittest.TestCase):
    def test_1(self):
        self.assertEqual("HELLOWORLD", capitalise("hello World"))

    def test_2(self):
        self.assertEqual("HELLO WORLD", capitalise("Hello World"))

    def test_3(self):
        self.assertEqual('!"£$%^&*()_+-=', capitalise('!"£$%^&*()_+-='))

    def test_4(self):
        self.assertEqual("1234567890", capitalise("1234567890"))

    def test_5(self):
        self.assertEqual("HELLO WORLD", capitalise("HELLO WORLD"))

    def test_6(self):
        self.assertEqual("~#~'@;:,.<>/?", capitalise("~#~'@;:,.<>/?"))
```

运行之后会发现更多错误。问题就出在程序的处理逻辑中。程序中不处理大写字符，而处理了所有非大写字符。然而，实际上你期望的是只处理小写字母，不处理其他字符。

如果按照下列代码修改 `capitalise` 函数，它就可以正常工作了：

```

def capitalise(input_string):
    output_string = ""
    for character in input_string:
        if character.islower():
            output_string = output_string + chr(ord(character) - 32)
        else:
            output_string = output_string + character
    return output_string

```

再次运行程序会发现所有的测试都通过了。

默认情况下，只有当测试失败时 unittest 才会给出详细结果。否则，它只会返回测试通过。大多数情况下它都能够满足你的需求，但它还提供了一个选项用来输出更多信息。有两种方式可以设置这个选项。如果在命令行中运行脚本，可以添加 -v 选项来获得更多输出。例如，你已经将程序保存为 capitalise.py，可以用下面的命令运行它以获得更多信息：

```
python3 capitalise.py -v
```

同样，你也可以在程序中添加该选项。例如可以将下列代码：

```

if __name__ == '__main__':
    unittest.main()

```

改为：

```

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

在继续下面内容之前，我们需要指出，这里的 capitalise() 函数只是用来举例的。如果需要将字符转换成大写，应该使用 string 类中的 upper() 函数。例如：

```
>>> 'hello world'.upper()
```

12.2.2 获得更多断言

所有这些测试都调用了 self.assertEqual()。这行语句告诉单元测试模块期望的测试结果是什么。也就是这两个值相等的时候测试才能通过，如果不相同，则测试失败。它在很

多测试用例中都能用到，但你可能还希望检查一些其他东西。除此之外，还有很多不同的 assert 方法可以在测试中。

下面这些用于检查各种结构体是否相同：

- `assertSequencesEqual(sequence1, sequence2)`
- `assertListEqual(list1, list2)`
- `assertTupleEqual(tuple1, tuple2)`
- `assertSetEqual(set1, set2)`
- `assertDictEqual(dict1, dict2)`

在这些结构体中，可能你还希望检查某个值是否属于该结构体而不仅仅是比较两个结构体是否相同。下列方法用于检查某个值是否属于结构体：

- `assertIn(value, structure)`
- `assertNotIn(value, structure)`
- 字符串是一个特殊的结构体，它有自己的方法：
- `assertMultiLineEqual(string1, string2)`

使用下列方法可以检查除相等之外的其他逻辑：

- `assertNotEqual(value1, value2)`
- `assertGreater(value1, value2)`
- `assertGreaterEqual(value1, value2)`
- `assertLess(value1, value2)`
- `assertLessEqual(value1, value2)`

还有一些方法允许适当的误差：

- `assertAlmostEqual(value1, value2)`
- `assertNotAlmostEqual(value1, value2)`

这两个语句用于检测两个值相差是否超过 0.000001。它们在测试浮点数程序时很有用，因为它们可以接受适当的舍入误差。

下面的语句可以用来测试任何可以缩减成 True 或 False 东西：

- `assertTrue(value)`
- `assertFalse(value)`

无论你想检查什么东西，每个 `test_` 函数都应该调用一个 assert 方法，以判断测试用例是否成功。

因为你在许多方面使用单元测试。有一种称为测试驱动开发的开发方式，它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。然而，大多数程序员都在开发完成之后编写测试用例来检测代码是否能够正常工作。

12.2.3 使用测试集进行回归测试

编写程序通常都不是一蹴而就的。并不是坐下来，写一段代码然后就可以停下来去做别的事情了。通常都是实现了某些特性之后交给客户使用，然后在下一个版本中修改错误并添加新的特性。

因为在添加新特性时可能破坏原来的东西，因此不仅需要测试新添加的功能，还需要保证旧功能可以工作。测试旧代码称为回归测试。拥有一个合适的测试集可以让回归测试变得很轻松。

为了保证在原来可以工作的代码中不引入错误，修改完成之后需要重新进行测试。然而，随着程序越变越大，所需的测试也会越来越多。最终，你会发现不可能一次运行所有的测试。这时可以将测试分成不同的测试集。一次只需运行几个相关的测试即可。

继续使用上一个例子，你可以将程序改成：

```
if __name__ == '__main__':
    letters_suite = unittest.TestSuite()
    symbols_suite = unittest.TestSuite()

    letters_suite.addTest(Tests("test_1"))

    symbols_suite.addTest(Tests("test_2"))
    symbols_suite.addTest(Tests("test_3"))

    symbols_suite.addTest(Tests("test_4"))

    symbols_suite.addTest(Tests("test_5"))

    symbols_suite.addTest(Tests("test_6"))

    all_suite = unittest.TestSuite()
    all_suite.addTest(letters_suite)
    all_suite.addTest(symbols_suite)

    unittest.TextTestRunner(verbosity=2).run(all_suite)
```

这段代码和前面的代码所做的工作一样，即运行所有的测试用例。然而它将测试用例分成不同的测试集。`letters_suite` 用来运行检查字符的是测试，`symbols_suite` 用来运行有所检查符号的测试，`all_suite` 运行这两个测试集。你可以使用最后一行来运行这三个测试集。

使用上述代码，你会发现可以轻易地建立一个简单的测试菜单，以保证所有功能都能正常运行。

12.2.4 测试整个程序包

将单元测试自动化了之后可以快速方便地检查所有功能是否正常运行。然而，它并不能够覆盖所有内容。即使看起来所有东西都能正常工作，当你将它们整合起来之后还是能发现问题。

在商业软件开发中，单元测试结束之后，代码将被传递给质量保证组以确保各部分能够正常工作。这个小组将从软件使用角度给出一系列测试用例。这里例子用于检查程序的各个方面并使用一组输入来测试程序，确保它能正常工作。这项工作又是通过测试人员手工模拟用户操作进行测试，又是使用专用的测试软件模拟鼠标和键盘输入。

当然，现在你还可能有一个质量保证小组来帮助你保证软件质量，但从专业方法来看，你可以借鉴一些内容。你需要一个系统化的方法。在开始测试前，先列出程序的所有功能，包括测试输入和期望输出。然后对照列表一项项执行，以确保程序工作正常。

代码每改动一次就执行一遍测试有点过分，但你需要周期性执行它，或者至少在每次的大版本发布时测试一遍。

12.2.5 保证软件可用性

程序完成之后，你对所有内容都十分了解。你知道如何与之交互，如果获得最好的结果，以及如何使用各种选项。然而，你的用户并不知道这些内容。你的软件需要帮助它们理解这些内容，并为用户提供足够的信息，让用户知道如何使用它。毕竟，如果用户不知道如何使用它们，你的特性有多强大都没用。

用户测试就是指用户的使用性测试，用于检测软件实现是否符合自己预期的要求。理想情况下，你可以将用户集中到一个房间里，在用户面前摆上软件，并请他们执行某些任务，然后观察输出结果。然而，这在现实世界中是不可能的。有时你能够说服朋友或者相关人员来帮助你，但开发的程序越多，你会发现能找到的志愿者越少。唯一的解决办法就是聆听使用软件的人的声音，确保能够收到反馈。

12.3 究竟需要多少测试

关于软件缺陷，有条谚语：“没有证据并不是表示没有问题”。基本上无论你做了多少测试，都不能够证明程序中没有缺陷。事实上，几乎不可能写出没有缺陷的程序。测试的目的并不是保证软件没有缺陷，而是用来保证软件足够好。“足够好”在不同的项目中有不同的含义。越重要的软件，越需要测试，但所有的软件都至少应该有一点测试。实现新功能时并不需要多花哨，重要的是要保证足够的测试而不是留下一堆问题。因此，花时间写单元测试并保证每个部分正常工作是值得的。毕竟当程序出错时，损坏的是你的数据。

12.4 小结

读完本章，你应该更好地理解以下内容：

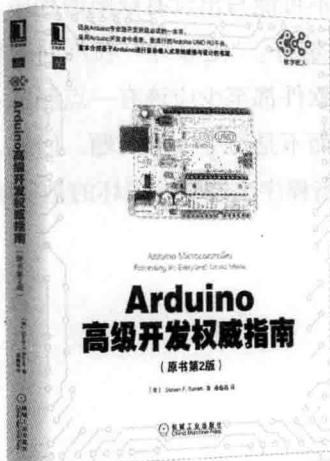
- 调试是在代码中发现并解决问题的过程。
- 正确地使用 `print()` 语句能够帮助你找出问题所在。
- 使用某种方法开关 `print()` 语句有时会很有用，在发现更多问题的时候可以重用这些语句。
- 测试是在代码中找出问题的过程。
- 单元测试是最基本的测试形式，使用 `unittest` 模块可以将其自动化。
- 测试用例可以分成不同的测试集，可以方便你测试程序的某一方面。
- 开发新特性时可能会引入新的问题，因此改过代码之后最好做一下回归测试。
- 单元测试并不能发现所有的问题，因此需要进行系统级别的测试。
- 可用性问题也是缺陷，因此你应该倾听用户反馈，以确保将程序使用方法正确地传达给它们。

至此，本书即将结束。但愿你现在已经有足够的知识和自信来创建自己的程序。如果感觉自己还没有完全掌握 Python 的特性也不用担心，很少有人能全部掌握。如果遇到不懂的问题，可以随时翻阅本书或者 Python 文档 <http://docs.python.org/3/>。

但愿你已经认识到编程并不复杂，如果将一个问题分成多个小步骤，编程就会变得很简单。最主要的是记住编程一个愉快的过程！找到自己感兴趣的领域并实践一下。尽管树莓派尺寸不大，却很少有它做不到的事情。

推荐阅读

沉默的大多数 陈昊鹏



Arduino高级开发权威指南 (原书第2版)

作者: Steven F. Barrett ISBN: 978-7-111-45246-1 定价: 59.00元

例说XBee
无线模块开发



例说XBee无线模块开发

作者: Jonathan A. Titus ISBN: 978-7-111-45681-0 定价: 59.00元



Arduino与LabVIEW开发实战

作者: 沈金鑫 ISBN: 978-7-111-45839-5 定价: 59.00元

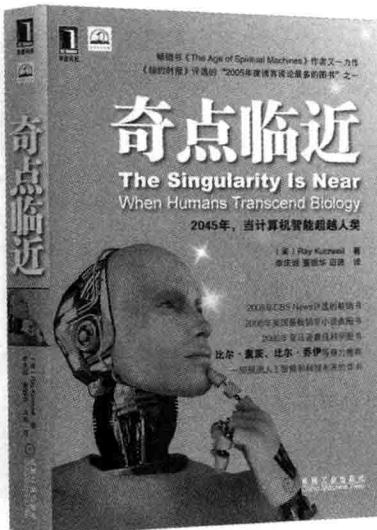


Arduino
开发实战指南
STM32篇

Arduino开发实战指南：STM32篇

作者: 姚汉 ISBN: 978-7-111-44582-1 定价: 59.00元

推荐阅读



奇点临近

畅销书《The Age of Spiritual Machines》作者又一力作
《纽约时报》评选的“2005年度博客谈论最多的图书”之一

2005年CBS News评选的畅销书

2005年美国最畅销非小说类图书

2005年亚马逊最佳科学图书

比尔·盖茨、比尔·乔伊等鼎力推荐

一部预测人工智能和科技未来的奇书

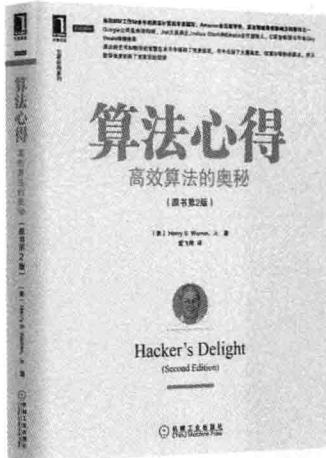
“阅读本书，你将惊叹于人类发展进程中下一个意义深远的飞跃，它从根本上改变了人类的生活、工作以及感知世界的方式。库兹韦尔的奇点是一个壮举，以不可思议的想象力和雄辩论述了即将发生的颠覆性事件，它将像电和计算机一样从根本上改变我们的观念。”

——迪安·卡门，物理学家

“本书对科技发展持乐观的态度，值得阅读并引人深思。对于那些像我这样对“承诺与风险的平衡”这一问题的看法与库兹韦尔不同的人来说，本书进一步明确了需要通过对话的方式来解决由于科技加速发展而引发的诸多问题。”

——比尔·乔伊，SUN公司创始人，前首席科学家

推荐阅读



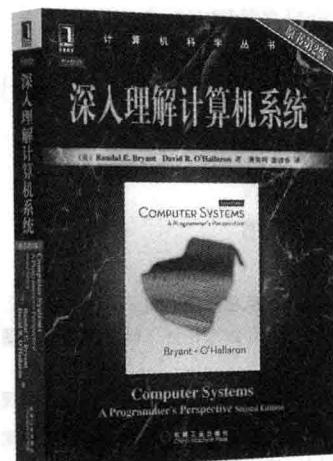
算法心得：高效算法的奥秘（原书第2版）

作者：Henry S. Warren ISBN：978-7-111-45356-7 定价：89.00元



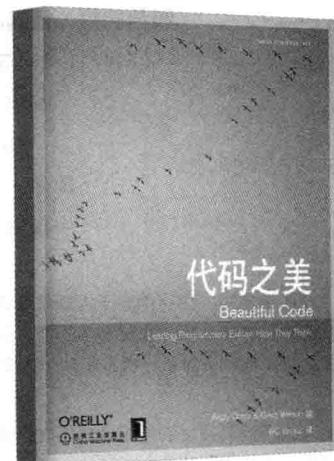
算法导论（原书第3版）

作者：Thomas H. Cormen 等 ISBN：978-7-111-40701-0 定价：128.00元



深入理解计算机系统（原书第2版）

作者：Randal E. Bryant 等 ISBN：978-7-111-32133-0 定价：99.00元



代码之美

作者：Grey Wilson ISBN：978-7-111-25133-0 定价：99.00元

本书由树莓派基金会资深软件开发工程师亲笔撰写，是学习在树莓派上编程的必备手册。即使你没有任何编程经验，也可以畅游树莓派的世界。本书覆盖了初学编程者和第一次做Python开发所需的基础知识，书中首先对Python编程做了基本介绍，并给出了通用的Python代码，然后逐步介绍了：

- 配置并开始Python编程
- 使用变量、循环和函数
- 学习3D图形编程
- 使用PyGame
- 编程“我的世界”游戏
- 编写Python脚本
- 理解传感器和GPIO

本书深入浅出地介绍每条命令，并辅以生动的例子和源代码，可以帮助你学习Python编程所需的各种知识和技能，是开启Python编程之旅的必备指南。

读者可以登录华章网站（www.hzbook.com）或访问<http://www.wiley.com/go/pythonraspi>下载本书源代码。

WILEY

www.wiley.com

投稿热线：(010) 88379604

客服热线：(010) 88378991 88361066

购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com

网上购书：www.china-pub.com

数字阅读：www.zhidireader.com

本文档由Linux公社 www.linuxidc.com 整理



上架指导：计算机\程序设计

ISBN 978-7-111-48986-3



9 787111 489863 >

定价：50.00元

PDF电子书说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

欢迎点击这里的链接进入精彩的[Linux公社](#) 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

[Linux公社](#)是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址: www.linuxidc.com 旗下网站:
www.linuxidc.net

包括: [Ubuntu 专题](#) [Fedora 专题](#) [Android 专题](#) [Oracle 专题](#)
[Hadoop 专题](#) [RedHat 专题](#) [SUSE 专题](#) 红旗 [Linux 专题](#) [CentOS 专题](#)



Linux 公社微信公众号: `linuxidc_com`

