第 11 章

Python 编程

树莓派名字的前一半来自一个悠久的传统，用水果名称来命名新的计算机系统（从典型的微型计算机，如橡果、杏、柑橘等，到公认的现代品牌，其中包括苹果和黑莓），而树莓派的另一半名字则来自Python编程语言。

11.1 Python介绍

Python具有灵活强大的功能，最初是由20世纪80年代末在全国数学与计算机科学研究所的Guido van Rossum开发的，作为ABC语言的一种继承。自推出以来，由于其清晰的表达语法，致力于代码可读性，Python已经相当普及。

Python是一种高层次的语言。这意味着Python代码是用在很大程度上是可读的英语写的，所谓的派意味着快速学习和易于遵循。这与低层次的语言，如汇编，形成了鲜明的对比，汇编语言更接近计算机“思维”，但让一个没有经验的人这样做几乎是不可能的。对于想要学习编程的人来说，高级别和自然清晰的语法使Python成为一个有价值的工具。它也是树莓派基金会对于那些不满足于Scratch（在第10章中所描述）、寻求进步的人的推荐语言。

Python在一个开放源码许可证下发布，并免费提供给Linux、OS X和Windows等计算机系统。跨平台支持的意思是用派开发的Python软件可以运行在几乎任何其他操作系统上，除了程序使用了特定的硬件，如GPIO端口。要了解如何用Python来解决这个端口问题，请参阅第12章。

11.2 例1：Hello World

正如在第10章所说，学习新的编程语言最简单的方法是创建一个项目输出“Hello World!”到屏幕上。从无到有，你只需拖放预先写好的代码，但在Python中，你必须完全由手工编写这个程序。

一个Python项目本质上就是一个包含了计算机执行指令的文本文件。这个文件可以使用任何文本编辑器创建。例如，如果你喜欢在控制台上工作，或在终端

窗口中，你可以使用 nano；如果你喜欢一个图形用户界面（GUI），你可以使用 Leafpad。另一种方法是使用集成开发环境（IDE），如 IDLE，它提供了 Python 特定的功能，它不仅仅是一个标准的文本编辑器，而且包括语法检查、调试设备，并能够运行你的程序，而无需离开编辑器。本章会教你如何使用 IDLE 创建 Python 文件，当然，IDE 的选择还是取决于你。本章还包括直接从终端上运行你创建的文件的教程，你可以使用任何文本编辑器或结合其他 IDE。

接着我们开始 Hello World 项目，从 Debian 发行版的桌面环境的编程菜单中打开 IDLE。如果你不使用 IDLE，请跳过本段其余部分，在你最喜爱的文本编辑器，创建一个空白文档。默认情况下，IDLE 打开了 Python Shell 模式（如图 11-1 所示），所以你的任何输入都将在初始窗口中立即执行。要打开一个新的可执行 Python 项目，单击菜单上的文件菜单，选择新窗口来打开一个空白文件。

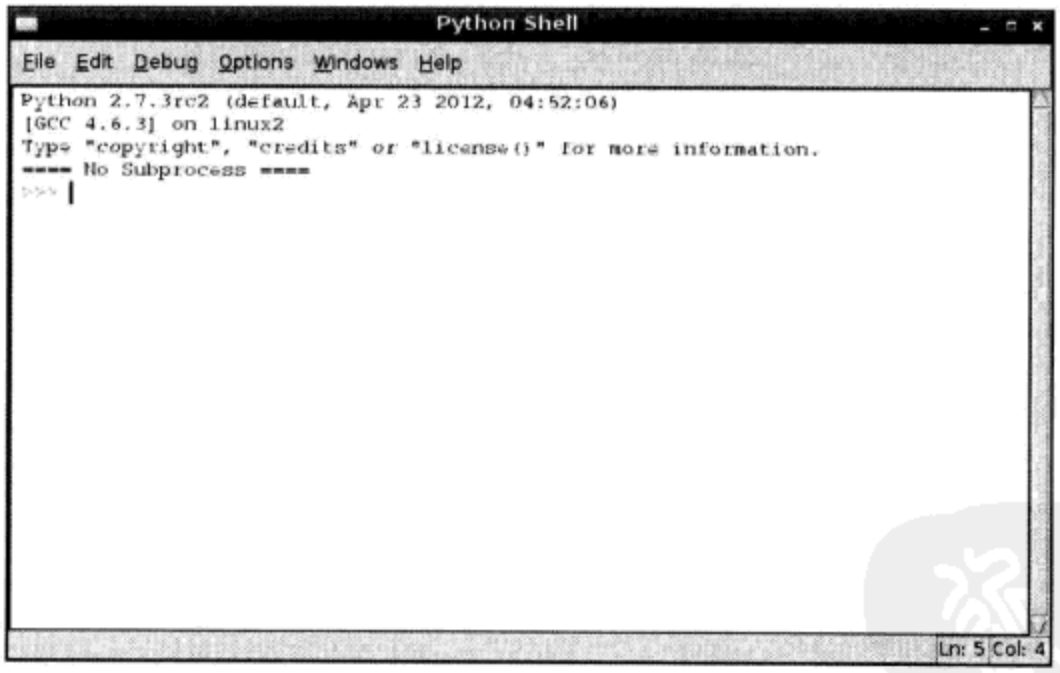

图 11-1 IDLE Python Shell 窗口

小提示 在程序菜单中选择IDLE3会载入Python 3.0版本。它包含了一些IDLE所用 Python 版本所不包含的特性，但是这些特性在本章中都没涉及到。你可以使用任一个版本，这些例子仍然可以正常运行。

用一行被称为 shebang 的代码开始所有的 Python 程序是很好的做法，它的名字来源于#和!字符。这一行告诉操作系统在哪里寻找 Python 文件。虽然这对于运

行在IDLE或显式地在终端调用Python不是必要的，但在通过调用该程序的文件名运行该程序时是必要的。

为确保程序运行时不依赖于Python可执行文件的安装目录，你的程序的第一行应如下所示：

```
#!/usr/bin/env python
```

这一行告诉操作系统查找$ PATH环境变量（这是Linux存储可以执行的程序的位置变量）来寻找Python的位置，它应该在任何派发行的Linux发行版上都可以运行。$ PATH变量包含了一系列的可执行文件存储路径，用于当你在控制台或终端窗口输入它们的名称是查找对应的程序。

为了实现打印消息这一目标，你应该使用Python的`print`命令。正如它的名字所示，此命令打印文本到一个输出设备，默认情况下打印到控制台或正在执行的程序的终端窗口。其用法很简单，任何跟在`print`后边并放在引号之间的文本都将被打印到标准输出设备。在新项目中输入以下命令：

```
print "Hello, World!"
```

最后的程序应该是这样：

```
#!/usr/bin/env python
print "Hello, World!"
```

如果你使用IDLE而不是纯文本编辑器创建示例程序，你会发现文字是彩色的（如图11-2所示，在印刷版中不同深浅的灰色代表不同的颜色）。这是一个被称为语法高亮的功能，是一个集成开发环境（IDE）和更先进的文本编辑工具的特性。为了使程序更易于理解、一目了然，语法高亮根据文字的功能改变颜色。这也使得它很容易发现所谓的语法错误，如忘记在`print`最后加引号，或忘了注释掉无关程序。对于这个简单的例子，语法高亮是没有必要的，但在较大的程序，它是一个用于查找错误的非常宝贵的工具。

在运行程序之前，使用“文件”菜单将它保存为`helloworld.py`。如果你使用IDLE，该文件将自动被给予`.py`扩展名。如果你使用文本编辑器，当保存它时一定要输入`.py`的文件名（不是`.txt`）。该扩展名表示该文件包含Python代码，

虽然 Python 同样能够运行不同的文件扩展名的程序文件。

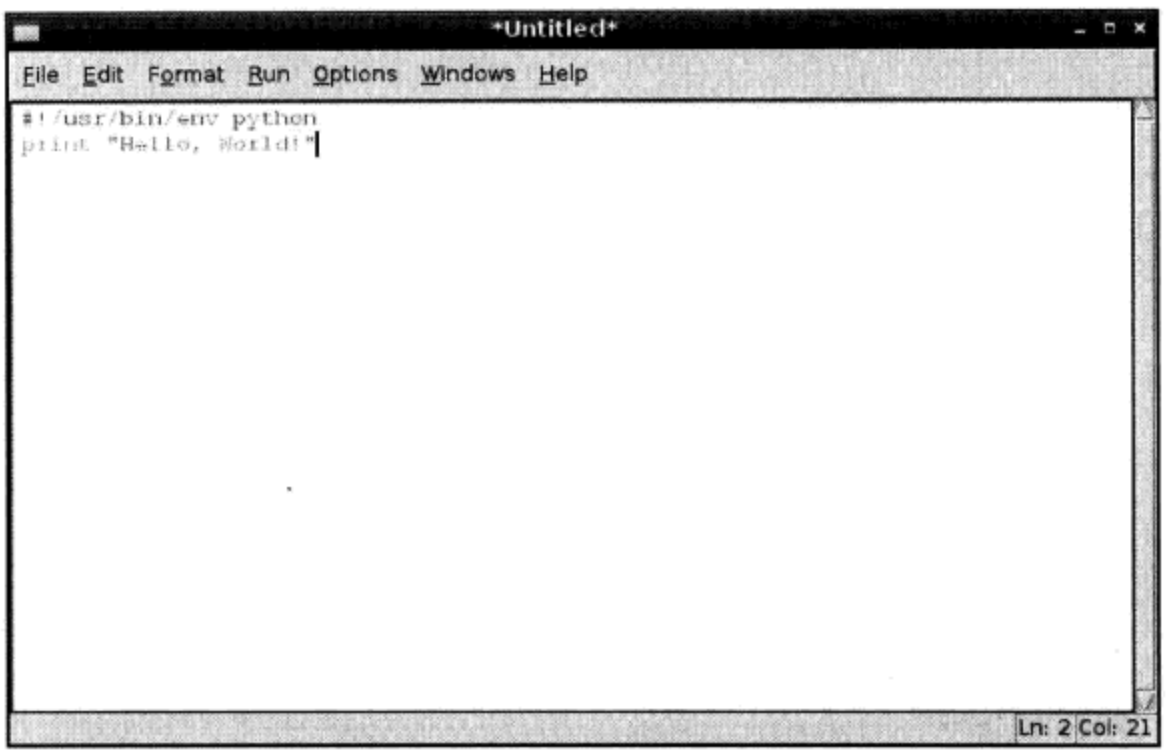

图 11-2 IDLE 的语法高亮

运行该文件的方法取决于你是否使用 IDLE 或文本编辑器。在 IDLE 中，只需从运行菜单选择“运行”模块，或者按键盘上的 F5 键，这将切换 IDLE 到 Python shell 窗口，并运行程序。然后，你应该看到屏幕上显示的蓝色 `Hello World!`（如图 11-3 所示）。如果没有，检查你的语法，特别是检查你打印行的引号开始和结束语句。

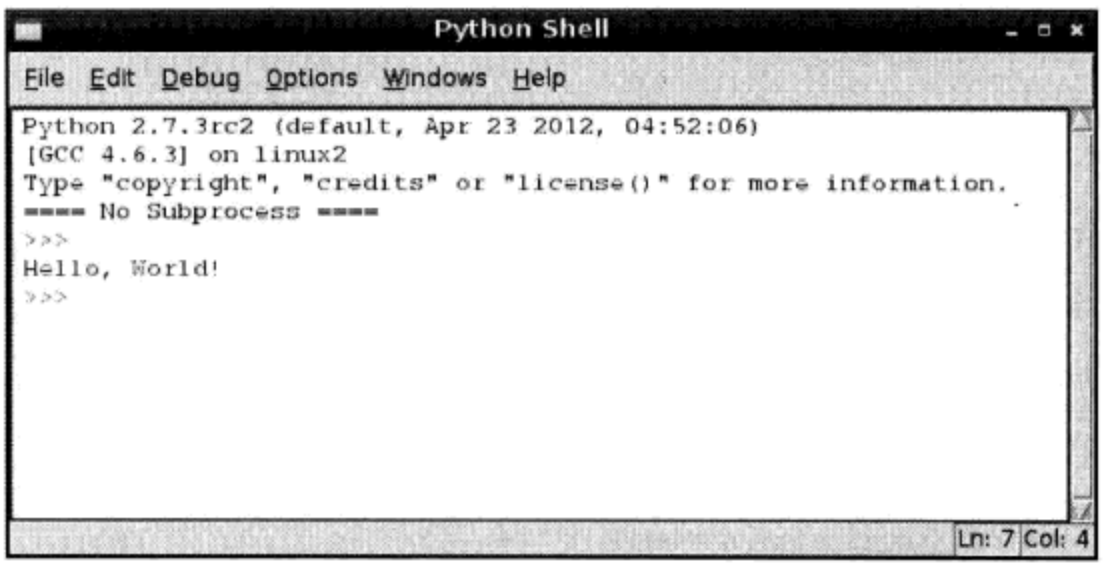

图 11-3 在 IDLE 中运行 helloworld.py

如果你在一个文本编辑器中创建了 `helloworld.py` 程序，你需要从桌面上

的“附件”菜单中打开一个终端窗口。如果你将文件保存在你的 home 目录以外的任何地方，你还必须使用 cd 命令改变该目录（请参阅本书的第 2 章）。一旦你在正确的目录，你可以输入以下命令运行你的程序：

```
python helloworld.py
```

这告诉操作系统运行 Python，然后加载 helloworld.py 文件。不同于 IDLE 的 Python shell，当它到达文件末尾时 Python 会退出并返回到终端。 但是结果是一样的，Hello, World!被打印到标准输出（如图 11-4 所示）。

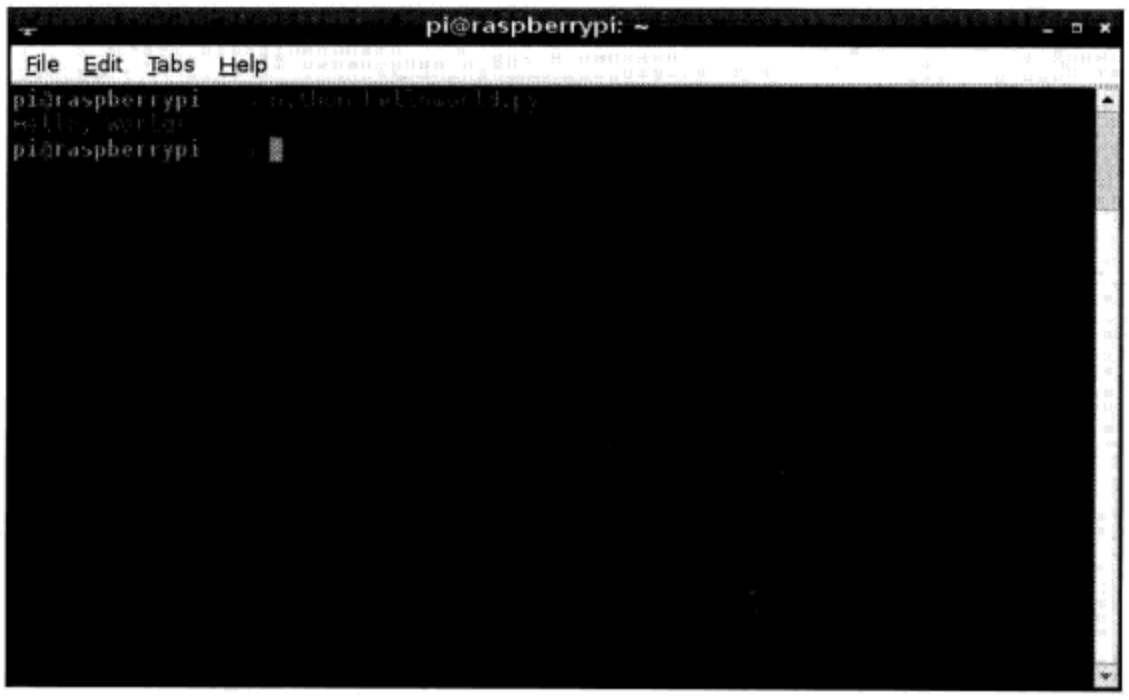

图 11-4 在终端中运行 helloworld.py 程序

让 Python 程序可执行

通常，运行 Python 程序的唯一方法是告诉 Python 软件打开该文件。由于文件的顶部有 shebang，可以直接执行该文件而无需先调用 Python。这是一个有用的方法，可以使自己开发的工具在终端执行。一旦把它的位置加入到系统的$ PATH 环境变量中，该 Python 程序可以直接通过它的名字调用。

首先，你需要告诉 Linux，Python 文件的属性应该被标记为可执行文件，表示该文件是一个程序。为了保护系统防止从互联网上下载恶意软件，系统不会自动设置这个属性，只有手动标记为可执行的文件才可以运行。为了使 helloworld.py

文件可执行，使用 chmod 命令（在第 2 章中详细介绍了），通过输入以下内容：

```
chmod +x helloworld.py
```

现在，通过输入以下内容尝试直接运行程序：

```
./helloworld.py
```

尽管你没有调用 Python 程序，helloworld.py 程序运行结果和输入 python helloworld.py 的结果仍然一样。该程序只能通过它的完整路径（/home/pi/helloworld.py）调用，或者在其所在路径下用./作为路径。为了让这个文件像其他终端命令一样调用，它需要被拷贝到/usr/local/bin，使用如下命令：

```
sudo cp helloworld.py /usr/local/bin/
```

前缀 sudo 是必需的，处于安全考虑，非授权用户是不能写内容到/usr/local/bin 目录的。/usr/local/bin 已经在$PATH 变量里了，helloworld.py 也在/usr/local/bin 里，现在可以在任意路径下通过名字调用该程序。试着切换到另一个不同的目录，通过下边命令运行程序：

```
helloworld.py
```

为了让你自己开发的程序看起来更像 Linux 命令，你可以重命名文件，去掉.py 后缀。要重命名文件，只需在一行中输入下边的命令：

```
sudo mv /usr/local/bin/helloworld.py↩
  /usr/local/bin/helloworld
```

一旦重命名了，该程序可以简单地在终端或控制台输入 helloworld 调用。

11.3 例 2：注释、输入、变量和循环

虽然 Hello World 程序是一个有用的、简单的语言入门程序，但是并不能让人为之兴奋。根据其性质，它只包含了其基础，却无法引入一些开发有用、有趣的程序所必需的概念。下面的例子，使用一些基本工具来创建 Python 的互动程序。

如例 1，在 IDLE 或文本编辑器中打开一个新的空白文档，然后首先输入下边

的 shebang：

```
#!/usr/bin/env python
```

正如前面所讨论的，这一行不是绝对必要的，除非要将它作为可执行程序，但也没有坏处，这是一个很好的开发习惯。

接下来，添加一个注释，方便以后打开该文件时参考。请注意，要把注释放在单独一行，文中➲表示上下两行为同一行：

```
# Example 2: A Python program from the➲
  Raspberry Pi User Guide
```

在 Python 中，任何#号后边的内容（除了 shebang 命令行）都被视为注释。当遇到注释，Python 会忽略它，并跳到下一行。注释你的代码是很好的习惯，虽然你现在可能知道一个特定部分的代码，当你 6 个月后再次打开该文件时，理解可能就变得模糊了。注释还有助于使代码更易于维护，如果你决定要与其他人分享你的代码，注释能帮助他们明白每个部分是做什么的。对于简单的程序，不是一定要添加注释，但是正如每个程序添加 shebang 一样，写注释是一个很好的习惯。注释可以和代码在同一行，或在代码行前，或在行结束，Python 会运行未注释的代码，忽略所有注释的内容。

接下来，用下面的代码询问用户的名字：

```
userName = raw_input("What is your name? ")
```

这一小行实际上包含了很多东西。首先，“userName =”告诉 Python 创建一个变量（一个存储信息的位置），名称为 userName。等于号告诉 Python 这个变量应该被赋值为右边部分。然而，本例中右边并不仅仅是一段信息，而是一个命令 raw_input。这是一个接收键盘输入的字符串（文本）工具，同时有信息显示在默认输出上让用户明白要输入什么内容。这让程序变得很简单，不需要专门打印一个询问语句告诉用户输入的内容，第二行我们需要 print 命令来输出内容。记住在询问内容的最后加一个空格，否则用户输入的内容和我们的询问内容就连在一起了。

警告 当让用户输入文本时，请使用 raw_input。这会提供一定的安全性，而单一的 input 命令却不能。如果你只用了 input，用户可能在你的程序中注入他的代码，使程序崩溃或者以另外的方式运行。

现在，用户的名字安全地存在 userName 变量里了，程序可以变得更聪明。用下边一行代码欢迎用户：

```
print "Welcome to the program,", userName
```

这一行演示了例 1 中介绍的 print 命令的第二个用法：打印变量的内容。该 print 命令被分为两部分，第一部分打印引号中的所有内容，逗号告诉 print 后边还有内容要打印在同一行上。我们只需简单地用 userName 就可以让 Python 知道我们要打印它的内容，这样打印的信息就由用户的名字决定了。

小提示 一种简单整洁的格式化输出的方法是在 print 命令的最后用.format 指令，如果你使用.format 命令，print 的格式应该如下：

```
print "Welcome, {0}, to this program.".format(userName)
```

接下来的例子中，我们要实现一个简单易用的计算器。不同于例 1，它会一直运行，直到用户主动结束它。我们用一个循环来实现这样的效果，就像 Scratch 一样。用下边两行开始一个循环：

```
goAgain = 1
while goAgain == 1:
```

为什么是==?

之前，你用一个等号为变量赋值。然而 while 循环却使用两个等号。使用两个等号表示对前后两个变量进行比较。而一个等号表示为左边的变量赋值。

除了双等号，还有其他的比较运算符，只有当变量满足条件才为真。>表示大于，<表示小于，>=表示大于等于，<=表示小于等于，!=表示不等于。

使用比较运算符，你可以根据布尔逻辑控制程序流，关于布尔逻辑的详细内容，请参阅第 10 章。

第一行创建了一个称为 goAgain 的变量，其值为 1。第二行开始循环，告诉 Python 当 goAgain 等于 1 时，应该继续循环执行后边的代码。在写后边几行代码时，应当在每行开始处用 4 个空格作为缩进。这几个空格告诉 Python 哪几行属

于循环，哪几行不属于循环。如果你使用IDLE，空格会被自动添加；如果你使用文本编辑器，记得手动添加空格。

对于计算器，最简单的形式是接收两个输入，然后对其进行运算。为了让其工作，首先用下边的代码接收用户输入的两个数字：

```
firstNumber = int(raw_input("Type the first number: "))
secondNumber = int(raw_input("Type the second number: "))
```

这几行不仅仅用了raw_input来请求用户输入，还用到了int指令。int指令告诉Python把用户输入视为短整数而不是字符串。这显然对计算器是很重要的，因为计算器不能计算字符串。

有了这两个存有数字的变量后，程序就能进行运算了。用下边的代码来对两个数字进行加、减和乘运算，然后输出结果：

```
print firstNumber, "added to", secondNumber, "equals",↩
firstNumber + secondNumber
print firstNumber, "minus", secondNumber, "equals",↩
firstNumber - secondNumber
print firstNumber, "multiplied by", secondNumber, "equals",↩
firstNumber * secondNumber
```

请注意，加法和减法运算时使用的是一般的加号和减号，乘法使用*符号。还要注意的是，在引号与引号直接没有格式化的空格。这是因为当Python打印的整数和字符串连接在一起时会自动添加空格。最后，请注意，没有除法运算符（例如/符号）。这是因为该计算器程序使用整数操作数，不允许带小数位或分数。

尽管计算器的一部分已经完成，但它会一直运行下去，因为目前还没有告诉Python什么时候退出循环。为了给用户提供退出程序的方式，添加下面一行：

```
goAgain = int(raw_input("Type 1 to enter more numbers,↩
or any other number to quit: "))
```

这让用户能够改变goAgain的值，而goAgain控制着循环。如果用户输入1，goAgain变量仍然等于1，循环继续。如果用户输入其他的数字，比较运算将返回假（goAgain不等于1），循环结束。

完整的程序如下所示，注意➲意思是上下两行需要写在一行。

```
#!/usr/bin/env python
# Example 2: A Python program from the➲
  Raspberry Pi User Guide
userName = raw_input("What is your name? ")
print "Welcome to the program,", userName
goAgain = 1
while goAgain == 1:
    firstNumber = int(raw_input("Type the first number: "))
    secondNumber = int(raw_input("Type the second number: "))
    print firstNumber, "added to", secondNumber, "equals", ➲
    firstNumber + secondNumber
    print firstNumber, "minus", secondNumber, "equals",➲
    firstNumber - secondNumber
    print firstNumber, "multiplied by", secondNumber, "equals",➲
    firstNumber * secondNumber
    goAgain = int(raw_input("Type 1 to enter more numbers, or ➲
    any other number to quit: "))
```

把程序存为 calculator.py，在 IDLE 中的 Run menu 中选择 Run Module 或者在终端中用 python calculator.py 命令来运行该程序。提示输入姓名时输入你的名字，然后输入你想进行计算的两个数字（如图 11-5 所示），当你感觉无聊的时候，输入其他不是 1 的数字来退出程序。

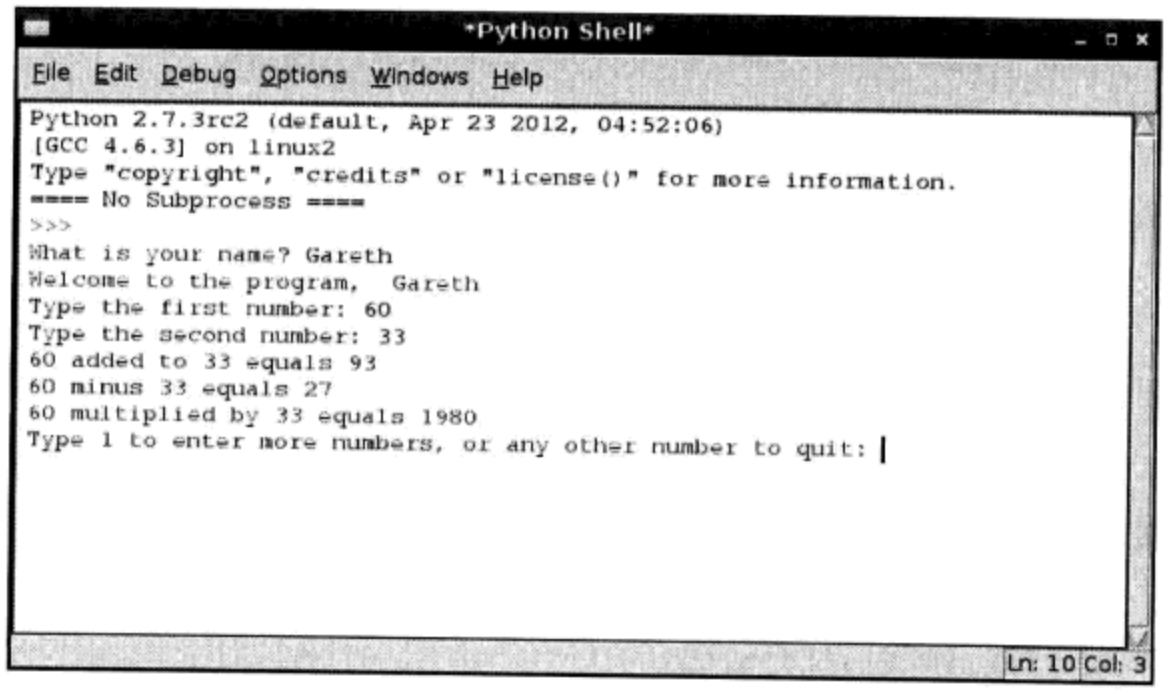

图 11-5 在 IDLE 中运行 calculator.py

更多的小程序和 Python 的概念，参阅 Python Simple Programs 官方网站：http://wiki.python.org/moin/SimplePrograms。

11.4 例3：用 pygame 开发游戏

为了说明 Python 的强大，此示例基于经典的贪吃蛇游戏或猫和老鼠游戏，创建了一个功能全面的街机游戏。要做到这一点，它使用一个 Python 的外部库 pygame。

pygame 最初由 Pete Shinners 开发，它是一个 Python 模块集，能够为 Python 添加新的功能，这些功能使开发者很容易用 Python 写一个游戏。pygame 模块提供了现代游戏所需要的功能，包括声音、图形和网络支持。虽然不使用 pygame 也可以写一个游戏，但如果你能充分利用 pygame 库中已经写好的代码，开发要容易得多。

在开发 pygame 程序之前，你需要安装 pygame 库。如果你使用的是我们推荐的 Debian 发行版，只要在控制台或终端中输入以下内容：

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

对于其他发行版，可以通过 pygame 的官方网站 http://www.pygame.org/download.shtml 下载源文件。安装指导也可以在相应页面找到。

打开 pygame 项目和打开其他 Python 项目的方法一样。在 IDLE 或文本编辑器中新建一个空文档，在顶部添加如下的 shebang：

```
#!/usr/bin/env python
```

然后你需要告诉 Python 该程序用到了 pygame 模块。为了实现此目的，我们用一个 import 指令，该指令告诉 Python 载入外部模块（其他 Python 文件），同时让外部模块在该程序中可用。输入下边两行来在新项目中引入必要的模块：

```
import pygame, sys, time, random
from pygame.locals import *
```

第一行引入 pygame 的主要模块、sys 模块、time 模块和 random 模块，

它们都会在本程序中用到。通常情况下，一个模块必须通过如下的格式使用：模块的名字、模块内的指令。而第二行告诉 Python 载入 pygame.locals 的所有指令使它们成为原生指令。这样，你使用这些指令时就不需要很多代码。其他的模块名（如 pygame.clock，它与 pygame.locals 独立）必须使用全名调用。

输入下边两行来启用 pygame，这样 pygame 在该程序中就可用了：

```
pygame.init()
fpsClock = pygame.time.Clock()
```

第一行告诉 pygame 初始化，第二行创建一个名为 fpsClock 的变量，该变量用来控制游戏的速度。然后，用下面两行代码新建一个 pygame 显示层（游戏元素画布）。

```
playSurface = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Raspberry Snake')
```

接下来，你应该定义一些颜色。虽然这一步并不是必需的，但它会减少你的代码量。如果你想把一个对象设置为红色，你只需要使用 redColour 变量而不用调用 pygame.Color 指令，也不需要记住红绿蓝 3 种颜色值。下面的代码定义了程序中的颜色：

```
redColour = pygame.Color(255, 0, 0)
blackColour = pygame.Color(0, 0, 0)
whiteColour = pygame.Color(255, 255, 255)
greyColour = pygame.Color(150, 150, 150)
```

下面几行代码初始化了一些程序中用到的变量。这是很重要的一步，因为如果游戏开始时这些变量为空，Python 将无法正常运行。别担心看不懂这些变量，先输入下面的代码：

```
snakePosition = [100,100]
snakeSegments = [[100,100],[80,100],[60,100]]
raspberryPosition = [300,300]
raspberrySpawned = 1
direction = 'right'
changeDirection = direction
```

可以看到 3 个变量 snakePosition、snakeSegments 和 raspberry

Position被设置为用逗号分隔的列表。这会让Python创建列表变量（一个变量中存有多个值）。之后，你会明白如何访问每个列表中的变量。

然后你需要定义一个新的函数（Python代码片段，在后边的程序中可以被调用）。函数可以提高代码复用率，也使程序易读。如果程序中很多地方用到了同样的一些指令，用def来创建一个函数，这样就可以只定义它们一次，而且如果程序需要修改，只需要修改一个地方即可。用下边几行代码来定义函数gameOver：

```
def gameOver():
        gameOverFont = pygame.font.Font↩
         ('freesansbold.ttf', 72)
        gameOverSurf = gameOverFont.render↩
         ('Game Over', True, greyColour)
        gameOverRect = gameOverSurf.get_rect()
        gameOverRect.midtop = (320, 10)
        playSurface.blit(gameOverSurf, gameOverRect)
        pygame.display.flip()
        time.sleep(5)
        pygame.quit()
        sys.exit()
```

类似于循环，函数中的代码应该缩进。def后边每行代码开头都应该有4个空格的缩进。如果你使用IDLE，这些空格会被自动添加，但是如果你用文本编辑器，你需要手动添加空格。在函数最后一行sys.exit()之后就不需要缩进了。

gameOver函数用了一些pygame命令来完成一个简单的任务：用大号字体将Game Over打印在屏幕上，停留5秒钟，然后退出pygame和Python程序。在游戏开始之前就定义了结束函数，这看起来有点奇怪，但是所有的函数都应该在被调用前定义。Python是不会自己执行gameOver函数的，直到我们调用该函数。

程序的开头部分已经完成，接下来进入主要部分。该程序运行在一个无限循环（一个永不退出的while循环）中，直到蛇撞到了墙或者自己的尾巴才会导致游戏结束。用下边的代码开始主循环：

```
while True:
```

没有其他的比较条件，Python会检测True是否为真。因为True一定为真，循环会一直进行，直到你调用gameOver函数告诉Python退出该循环。

输入下面的代码，注意代码缩进等级：

```
for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit()
        sys.exit()
    elif event.type == KEYDOWN:
```

第一行紧接着 while 循环的开始处，应该缩进 4 个空格，作为循环体部分，for 指令用来检测例如按键等 pygame 事件。这样，后边的代码应该进一步缩进 4 个空格，总共 8 个空格，用 if 指令来判断用户是否按下了某键，由于它也是 for 指令循环体的部分，所以下一行 pygame.quit() 应该再进一步缩进，为 12 个空格。这种处理缩进的方式告诉 Python 循环的开始和结束的位置，这是很重要的。如果使用了错误的缩进，程序将不能正确地运行。这就是使用类似 IDLE 这样的开发环境的原因，它会自动添加缩进，比使用文本编辑器要方便多了。

一个 if 循环告诉 Python 检测某条件是否为真。第一个检测 if event.type == QUIT 告诉 Python 如果 pygame 发出了 QUIT 信息（当用户按下 Esc 键），执行下边缩进的代码。之后的两行类似 gameOver 函数，通知 pygame 和 Python 程序结束并退出。

elif 开头的行用来扩展 if 语句。它是 else if 的缩写，当前边的 if 指令为假，elif 后边的条件将被判断。在本例中，elif 指令用来检测 pygame 是否发出 KEYDOWN 信号，该信号在用户按下键盘时返回。类似 if 命令，当 elif 后条件为真，其后相应的缩进代码将被执行。输入下边的代码，当用户按下某键，elif 指令执行：

```
        if event.key == K_RIGHT or event.key == ord('d'):
            changeDirection = 'right'
        if event.key == K_LEFT or event.key == ord('a'):
            changeDirection = 'left'
        if event.key == K_UP or event.key == ord('w'):
            changeDirection = 'up'
        if event.key == K_DOWN or event.key == ord('s'):
            changeDirection = 'down'
        if event.key == K_ESCAPE:
            pygame.event.post(pygame.event.Event(QUIT))
```

这些指令修改变量changeDirection的值，该变量用于控制蛇的运动方向。在if后边用or命令添加更多的比较条件。在本例中，提供了两种控制蛇的方法。用鼠标或者键盘的W、D、A和S键，来让蛇向上、右、下和左移动。程序开始时，蛇会按照changeDirection预设的值向右移动，直到用户按下键盘改变其方向。

如果你回过头去看程序开始的初始化部分，你会发现有一个叫direction的变量。这个变量协同changeDirection检测用户发出的命令是否有效。蛇不应该立即向后运动（如果发生该情况，蛇会死亡同时游戏结束）。为了防止这样的情况发生，将用户发出的请求（存在changeDirection里）和目前的方向（存在direction里）进行比较，如果方向相反，忽略该命令，蛇会继续按原方向运动。用下面几行代码来进行比较：

```
 if changeDirection == 'right' and not
direction == 'left':
    direction = changeDirection
 if changeDirection == 'left' and not
 direction == 'right':
    direction = changeDirection
 if changeDirection == 'up' and not
direction == 'down':
    direction = changeDirection
 if changeDirection == 'down' and not
 direction == 'up':
    direction = changeDirection
```

这样就保证了用户输入的合法性，蛇（屏幕上显示为一系列块）就能够按照用户的输入移动。每次转弯时，蛇会向该方向移动一小节。每个小节为20像素，你可以告诉pygame在任何方向移动一小节。输入下面的代码：

```
if direction == 'right':
    snakePosition[0] += 20
if direction == 'left':
    snakePosition[0] -= 20
if direction == 'up':
    snakePosition[1] -= 20
if direction == 'down':
    snakePosition[1] += 20
```

这里的+=和-=操作符用来改变变量的值。+=将变量的值设置为原值与新值的

和，-=将变量设置为原值与新值的差。在本例中，snakePosition[0] += 20 其实是 snakePosition[0] = snakePosition[0] + 20 的简写。SnakePosition 后边方括号中的数字表示列表中的一个项目：第一个数表示 X 轴对应的值，第二个数字代表 Y 轴对应的值。Python 从 0 开始计数，所以 X 轴用 snakePosition[0] 控制，Y 轴用 snakePostion[1]控制。如果列表更长，其他项目通过增加方括号中的值来访问，如[2]、[3]等。

snakePosition 为两个值的长度，程序开始处另一个列表变量 snakeSegments 却不是这样。该列表存储蛇身体的位置（头部后边）。随着蛇吃掉树莓导致长度增加，列表会增加长度同时提高游戏难度。随着游戏进行，避免蛇头撞到身体的难度变大。如果蛇头撞到身体，蛇会死亡同时游戏结束。用下边的代码使蛇身体增长：

```
snakeSegments.insert(0,list(snakePosition))
```

这里用 insert 指令向 snakeSegments 列表（存有蛇当前的位置）中添加新项目。每当 Python 运行到这行，它会将蛇的身体增加一节，同时将这节放在蛇的头部。在玩家看来蛇在增长。当然，你只希望当蛇吃到树莓时才增长，否则蛇会一直变长。输入下面几行：

```
if snakePosition[0] == raspberryPosition[0]↩
and snakePosition[1] == raspberryPosition[1]:
    raspberrySpawned = 0
else:
    snakeSegments.pop()
```

第一条指令检查蛇头部的 X 和 Y 坐标是否等于树莓(玩家的目标点)的坐标。如果等于，该树莓就会被蛇吃掉，同时 raspberrySpawned 变量置为 0。else 指令告诉 Python 如果树莓没有被吃掉要做的事，将 snakeSegments 列表中最早的项目 pop 出来。

pop 指令简单易用。它返回列表中最早加入的项目并从列表中删除，使列表缩短一项。在 snakeSegments 列表里，它使 Python 删掉距离头部最远的一部分。在玩家看来，蛇整体在移动而不会增长。实际上，它在一端增加小节，在另一端删除小节。由于有 else 语句，pop 指令只有在没吃到树莓时执行。如果吃到了树莓，列表中最后一项不会被删掉，所以蛇会增加一小节。

现在，蛇就可以通过吃树莓来让自己变长了。但是游戏中只有一个树莓的话有些无聊，所以如果蛇吃了一个树莓，则用下面的代码增加一个新的树莓到游戏界面中：

```
if raspberrySpawned == 0:
    x = random.randrange(1,32)
    y = random.randrange(1,24)
    raspberryPosition = [int(x*20),int(y*20)]
raspberrySpawned = 1
```

这部分代码通过判断变量 raspberrySpawned 是否为 0 来判断树莓是否被吃掉了，如果被吃掉，使用程序开始引入的 random 模块获取一个随机的位置。然后将这个位置和蛇的每个小节的长度（20 像素宽，20 像素高）相乘来确定它在游戏界面中的位置。随机地放置树莓是很重要的，防止用户预先知道下一个树莓出现的位置。最后，将 raspberrySpawned 变量置 1，以此保证每个时刻界面上只有一个树莓。

现在你有了让蛇移动和生长的必需代码，包括树莓的被吃和新建操作（游戏中称为树莓重生）。但是我们还没有在界面上画东西。输入下面的代码：

```
playSurface.fill(blackColour)
for position in snakeSegments:
    pygame.draw.rect(playSurface,whiteColour,Rect↩
    (position[0], position[1], 20, 20))↩
    pygame.draw.rect(playSurface,redColour,Rect↩
    (raspberryPosition[0], raspberryPosition[1], 20, 20))
pygame.display.flip()
```

这些代码让 pygame 填充背景色为黑色，蛇的头部和身体为白色，树莓为红色。最后一行的 pygame.display.flip()，让 pygame 更新界面（如果没有这条指令，用户将看不到任何东西。每当你在界面上画完对象时，记得使用 pygame.display.flip()来让用户看到更新）。

现在，还没有涉及蛇死亡的代码。如果游戏中角色永远死不了，玩家很快会感觉无聊，所以用下边的代码来设置一些让蛇死亡的场景：

```
if snakePosition[0] > 620 or snakePosition[0] < 0:
    gameOver()
```

```
if snakePosition[1] > 460 or snakePosition[1] < 0:
    gameOver()
```

第一个 if 语句检查蛇是否已经走出了界面的上下边界，而第二个 if 语句检查蛇是否已经走出了左右边界。这两种情况都是蛇的末日，触发前边定义的 gameOver 函数，打印游戏结束信息并退出游戏。如果蛇头撞到了自己的身体的任何部分，也会让蛇死亡，所以输入下面几行代码：

```
for snakeBody in snakeSegments[1:]:
    if snakePosition[0] == snakeBody[0] and
    snakePosition[1] == snakeBody[1]:
        gameOver()
```

这里的 for 语句遍历蛇的每一小节的位置(从列表的第二项开始到最后一项)，同时和当前蛇头的位置比较。这里我们用 snakeSegments[1:]来保证从列表第二项开始遍历。列表第一项为头部的位置，如果从第一项开始比较，那么游戏一开始蛇就死亡了。

最后，只需要设置 fpsClock 变量的值即可控制游戏速度。如果没有这个变量(在程序开始处创建)，游戏会变得太快而无法正常玩。输入下面的代码完成程序：

```
fpsClock.tick(20)
```

如果你觉得游戏太简单或者太慢，你可以增大那个参数；如果你觉得游戏太快或者太难，你可以减小那个参数。把程序存为 raspberrysnake.py，使用 IDLE 的 Run Module 选项或者在终端中输入 python raspberrysnake.py 来运行程序。游戏会在载入程序后立即开始（如图 11-6 所示），所以你要准备好立即开始游戏。

Raspberry Snake 的完整源代码在附录 A 中，可以访问 http://www.wiley.com/go/raspberrypiuserguide 查看“Python Recipes”和树莓派的用户手册。直接从网上下载源码可以节省你的时间，不过自己输入代码也是一个很好的学习方法，可以确保你明白每个部分的功能。除了 Raspberry Snake 所用到的功能，pygame 还提供了很多其他该程序未涉及到的功能，如声音、元素控制、鼠标操作等美化用户界面和提高交互性的功能。最好的学习 pygame 的地方是其官方网站 http://www.pygame.org/wiki/tutorials，你可以下载学习资料和实例程序来进一步掌握其用法。

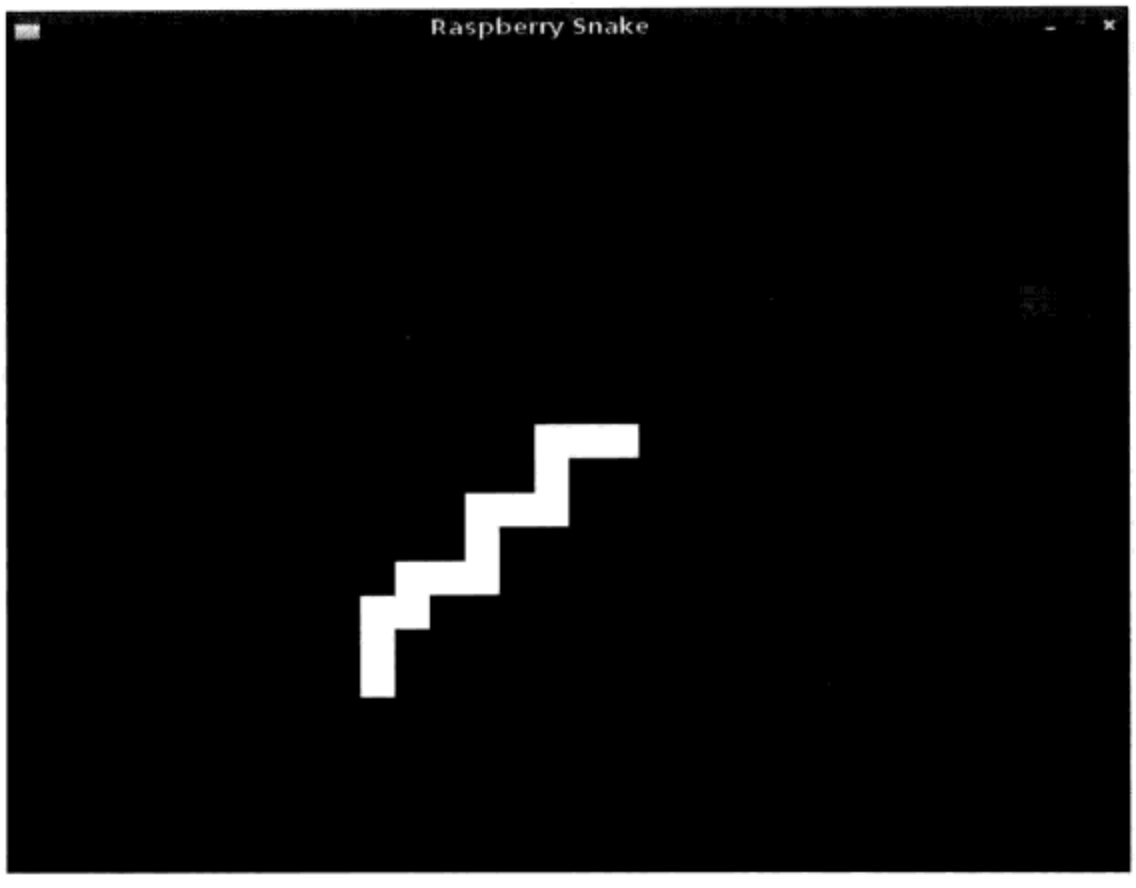

图 11-6 在树莓派上运行 Paspberry Snake 程序

11.5 Python 和网络

目前，你已经学会了如何用 Python 开发独立的程序，但是 Python 还可以通过计算机的网络连接外部世界并与之交互。下面的例子中（由 Tom Hudson 开发）演示了一个监测用户连接到 Internet Relay Chat（IRC）频道的程序。

和之前一样，用 IDLE 或者文本编辑器新建一个项目，输入下面的 shebang 和注释：

```
#!/usr/bin/env python
# IRC Channel Checker, written for the↩
Raspberry Pi User Guide by Tom Hudson
```

然后引入该程序需要的模块 sys、socket 和 time：

```
import sys, socket, time
```

在前边的贪吃蛇程序中，我们已经使用了 sys 和 time 模块，但是还没有涉

及 socket 模块。Socket 模块提供了网络套接字打开、关闭、读取数据和写入数据的功能，为 Python 提供了初步的网络功能。通过该模块，我们可以连接到 IRC 服务器。

该程序用到一些常量。常量和变量一样有自己的值，但是与变量不同的是它的值不能改变。为了区分常量和变量，比较好的方法是用大写字母定义常量，这样就可以很容易判断代码是变量还是常量。输入下面的代码：

```
RPL_NAMREPLY = '353'
RPL_ENDOFNAMES = '366'
```

这是 IRC 的状态码，当服务器完成某操作时返回。它们用于确定从 IRC 服务器返回名字列表的时间。接下来，用下面的代码设置一些连接服务器需要的变量：

```
irc = {
    'host' : 'chat.freenode.net',
    'port' : 6667,
    'channel' : '#raspiuserguide',
    'namesinterval' : 5
}
```

第一行告诉 Python 创建一个字典类型的变量。字典变量允许存储多个变量值。每个单一的变量值在后边的程序中都会用到。当然你也可以不用字典变量，不过那样会让程序变得不易读。字典变量以花括号作为开始和结束标记。

host 变量被设置为 IRC 服务器的全称域名（FQDN）。在本例中为 chat.freenode.net，如果你想连接其他的服务器，只需更改这个名字即可。Port 变量告诉程序 IRC 运行的网络端口，一般来说是 6667。channel 变量告诉 Python 加入哪一个频道来监控用户。namesinterval 控制用户列表刷新的时间，以秒为单位。

用另一个字典变量存储用户信息：

```
user = {
    'nick' : 'botnick',
    'username' : 'botuser',
    'hostname' : 'localhost',
    'servername' : 'localhost',
```

```
        'realname' : 'Raspberry Pi Names Bot'
    }
```

类似irc变量，所有的变量被存在一个字典变量中，这样很容易确定哪个变量是属于哪个部分的。nick变量为IRC的昵称。如果你想同时发起多个连接到IRC服务器，不要使用你的常用昵称，在名字后边加bot来表示连接服务器的是程序而不是实际的用户。username也按同样的方式处理，在username中加入程序的属主信息。hostname和servername变量可以设置为localhost，或者设为你的IP地址。

Socket模块需要用户创建一个socket对象。该对象建立程序所需的网络连接。用下边的代码创建socket对象：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

然后，你需要告诉程序连接程序开始定义的IRC服务器。输入下面的代码：

```
print 'Connecting to %(host)s:%(port)s...' % irc
try:
    s.connect((irc['host'], irc['port']))
except socket.error:
    print 'Error connecting to IRC server↩
    %(host)s:%(port)s' % irc
    sys.exit(1)
```

try和except命令用于处理错误。如果系统连接服务器失败，比如树莓派未联网或者服务器维护关闭，程序会打印一行错误信息并正常退出。s.connect这行告诉socket模块尝试连接IRC服务器，用到了irc字典变量中的host和port变量。

如果程序没有发生异常并退出，说明成功连接到了IRC服务器。在你获取名字列表之前，服务器需要鉴定你的身份，使用send函数来向socket模块发送数据，输入下面的代码：

```
s.send('NICK %(nick)s\r\n' % user)
s.send('USER %(username)s %(hostname)s↩
  %(servername)s :%(realname)s\r\n' % user)
s.send('JOIN %(channel)s\r\n' % irc)
s.send('NAMES %(channel)s\r\n' % irc)
```

Send 函数用法基本上和 print 函数一样，除了不会打印东西到标准输出（一般为终端窗口或控制台），它将输出发送到网络连接。在本例中，程序发送一些字符串到 IRC 服务器，告诉服务器注册自己的昵称和其他详细信息（存在 user 字典变量里）。然后，程序发送加入频道的命令，最后发送获取该频道所有用户名字的命令。虽然该程序是为 IRC 定制的，程序的基本原则可以用在其他网络服务上，修改此程序，此程序就可以通过 FTP 服务获取文件列表或者通过 POP3 服务获取未读邮件列表。

从 socket 获取数据相对来说要复杂些。首先，你需要创建一个空字符串变量作为接收缓冲区来接收服务器发送的数据。用下面的代码初始化该缓冲区：

```
read_buffer = ''
```

注意后边是两个单引号，不是双引号。

然后创建一个空列表用来存储用户的姓名，键入下面的代码：

```
names = []
```

这个列表类型和之前贪吃蛇游戏中用来存储蛇位置的列表类型相同。和一般的变量不同，它可以存储多个值，本例中为 IRC 频道中用户的姓名。

下一步是创建一个无限循环，用来连续地向服务器发出请求用户列表的命令，同时打印在屏幕上。用下面的代码开始循环：

```
while True:
    read_buffer += s.recv(1024)
```

循环中第一行（紧接着 while True:）告诉 socket 模块从 IRC 服务器接收 1024 字节（1KB）数据放在 read_buffer 变量里。由于使用了+=而不是=操作符，接收到的数据将会被追加到缓冲区后边。1024 字节可以是其他任意大小的值。

下一步是将缓冲区中的数据分为独立的文本行，用下面的代码实现：

```
lines = read_buffer.split('\r\n')
read_buffer = lines.pop();
```

第一行代码将接收缓冲区中所有完整的行赋给 lines 变量，这里使用 split 函数来搜索行结束标识（\r\n）。这些标识只出现在行结束处，所以当缓冲区用这样的方式分隔时，你就知道 lines 变量一定是服务器返回的完整行。第二行的 pop 指令保证从 read_buffer 中去掉的都是完整的行。由于服务器的返回数据按照 1KB 的片段接收，很可能在某个时刻缓冲区只收到一行的一部分。当出现这样的情况时，使该不完整片段留在缓冲区里，等待在下一个循环中接收剩余部分。

此时，lines 变量包含了服务器响应的姓名列表（为完整的行）。输入下面的代码处理这些行并提取加入该频道的用户姓名：

```
for line in lines:
     response = line.rstrip().split(' ', 3)
     response_code = response[1]
    if response_code == RPL_NAMREPLY:
         names_list = response[3].split(':')[1]
         names += names_list.split(' ')
```

程序遍历 lines 变量，提取 IRC 响应的状态码。虽然有很多不同的响应状态码，该程序只涉及两种状态码，即程序开始处定义的 353（表示后边为名字列表）和 366（表示列表结束）。这里的 if 语句寻找第一个响应码，然后使用 split 函数获取所有姓名并添加到 names 列表中。

现在，names 列表包含了所有服务器返回的姓名列表。然而这也许并不是所有的姓名，直到收到 366 状态码，表示列表发送完成。这就是最后一行 names += names_list.split(' ') 将新收到的姓名追加到后边而不是清空并替换 names 变量的原因，每当该部分代码执行时，程序只可能收到所有成员列表的一部分。为了告诉 Python 收到完整列表后做什么，输入下面几行代码：

```
if response_code == RPL_ENDOFNAMES:
    # Display the names
    print '\r\nUsers in %(channel)s:' % irc
    for name in names:
        print name
    names = []
```

该代码告诉 Python 当收到 366 响应时，应该在清空 names 列表前将完整的姓名列表打印到标准输出。最后一行 names = []很重要。如果没有这行，每次循环都会追加姓名到列表最后，这会和前边收到的列表重复。

最后，用下面几行代码完成程序：

```
time.sleep(irc['namesinterval'])
s.send('NAMES %(channel)s\r\n' % irc)
```

代码告诉 Python 等待 namesinterval 秒后发送另一个获取用户列表的请求并开始新的循环。要将 namesinterval 小心地设置为一个合理的值，如果 IRC 服务器在短时间内收到太多的请求，它可能会强制断开连接以防止洪范攻击。

将程序存为 ircuserlist.py，然后通过 IDLE 的 Run Module 选项或者通过终端输入 python ircuserlist.py 运行程序。当程序第一次运行时，也许会花一些时间来连接服务器。一旦连接成功，姓名列表（如图 11-7 所示）会快速刷新。要退出程序，按 CTRL+C 组合键。

图 11-7 用 Python 列出 IRC 频道用户列表

程序的完整源代码见附录 A，“Python Recipes”和树莓派的用户手册可以访问 http://www.wiley.com/go/raspberrypiuserguide 查看。直接从网上下载源码可以节省你的时间，不过自己输入代码也是一个很好的学习方法，可以确保你明白每个

部分的功能。

11.6 进一步阅读

这一章介绍了 Python 的基本功能。但还不深入，要全面掌握这门语言需要一本很厚的书。不过，也有足够的资源来学习更多有关 Python 编程的方法：

- Python Beginner's Guide 官方网站：http://wiki.python.org/moin/BeginnersGuide。
- 一个免费的运行在浏览器中的交互学习资料：http://www.learnpython.org。
- Zed A. Shaw 写的《Learn Python The Hard Way》（Shavian Publishing，2012）提供了 Python 的最佳实践，它适合初学者。你可以购买本书，或者在线免费阅读：http://learnpythonthehardway.org。
- 《Dive Into Python 3》虽然有些过时，但是它很好地介绍了 Python 编程的基本方法，你可以免费下载本书：http://www.diveintopython.net。
- 如果你喜欢和其他人一起学习，Python 学习小组（有时称为 PIGgies）在此：http://wiki.python.org/moin/LocalUserGroups。
- 对于学习 pygame，Al Sweigart 缩写的《Making Games with Python》对实际的例子做了详细的介绍。你可以购买此书或者免费下载：http://inventwithpython.com。