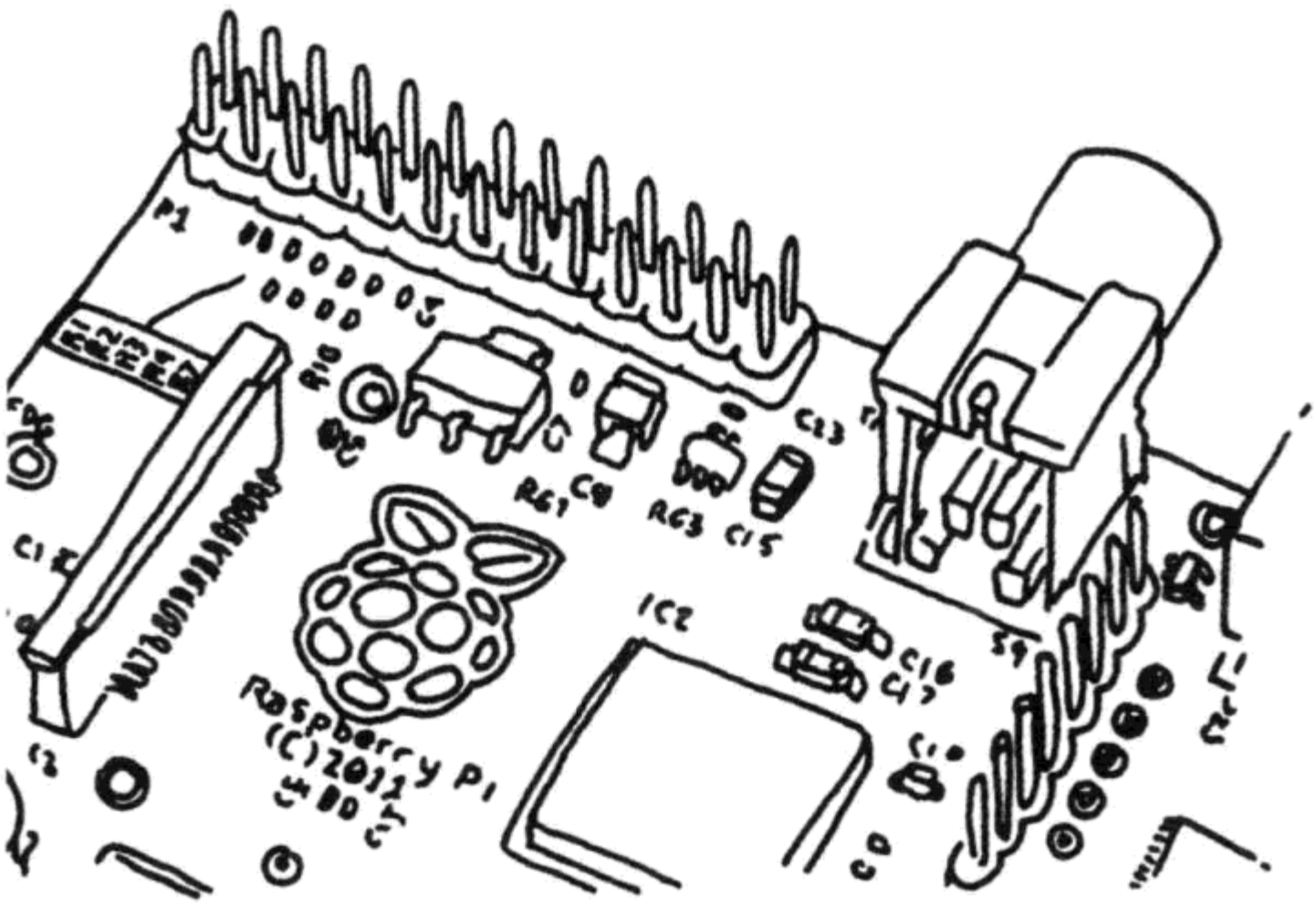
第 3 章

Pi 上的 Python

Python on the Pi



把 Python 作为入门语言非常合适，它的代码非常清晰，安装和设置运行环境也很容易。更重要的是，它有一个庞大的用户群，大家可以在一起分享代码或共同分析、解决问题。

Guido van Rossum 创造了 Python 语言，并在很早的时候就把它设计成一门适合入门的计算机语言。1999 年，van Rossum 提出一项叫作“人人能编程”（Computer Programming for Everybody，*http://www.python.org/doc/essays/cp4e.html*）的宏伟计划，计划用 Python 语言在中小学校开展编程教学。十多年时间过去，Raspberry Pi 的出现让这个计划得以实现。

Python 是一种解释性语言，你所编写的程序或脚本可以直接被执行，而不需要把它们先整体编译成机器码。解释性语言用起来很便捷，并且有一些额外的优点。例如，在 Python 中，你不需要显式地指定某一个变量是数字、列表还是字符串，解释器在执行你的脚本时会自动推断出变量的类型。

Python 的解释器有两种运行模式：既可以当成一个交互式的终端执行单条命令，也可以作为一个命令行工具运行独立的脚本。Raspberry Pi 上也提供了与 Python 绑定在一起的集成开发环境 IDLE（图 3.1）。

Python 的版本之谜

Pi 上预装了两个版本的 Python，所以在 Pi 上能看到两

套 IDLE 开发环境。这看起来有点让人糊涂，但却是一种常见的做法。原因在于，目前 Python 3 是最新的 Python 版本，但 Python 2 到 Python 3 之间发生了很多变化，并且并不向下兼容。虽然 Python 3 已经出现很久，但仍有很多软件包没有升级到 Python 3。当你查找 Python 的文档时尤其要注意区分不同的版本，避免造成混淆。

没有特别说明的话，本书中的代码既可以在 Python 2.7 上运行，也可以在 Python 3.x 上运行。

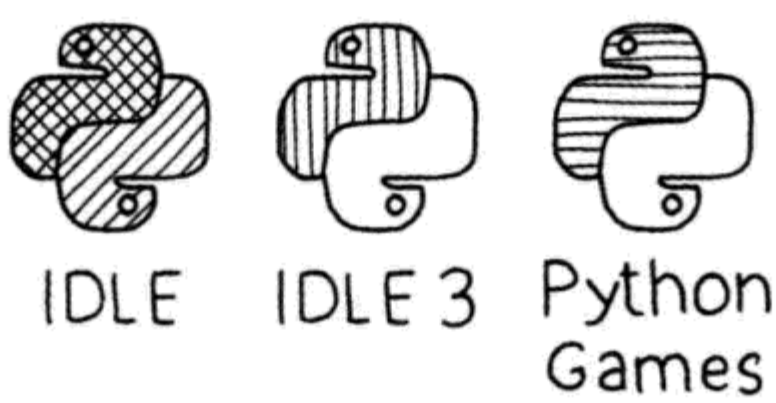


图3.1 Raspbian桌面上的Python图标：左侧是Python 2.0的IDLE，中间是Python 3.0的IDLE，右侧是集合了inventwithpython.com（*http://inventwithpython.com*）上很多游戏的示例程序，这些游戏是通过Pygame开发的

初识 Python

学习 Python 最好的方式就是通过实践，虽然你可以用任何文本编辑器来编写脚本，但我们仍会从使用 IDE（集成开发环境）开始。可以双击桌面上的 IDLE 3 图标打开 IDLE 3 开发环境，也可以点击左下角的桌面菜单，选择编程（Programming）→ IDLE 3。

IDLE 启动需要花费几秒钟时间，当它启动后，你可以看到它的界面，里面包含了一个交互式的终端。行首显示的 3 个大于号

（>>>）是命令提示符，当你看到这个提示符时，就表明解释器正在等待你输入命令。下面，尝试在提示符后输入：

```
>>> print("Saluton Mondo!")
```

按回车键，Python 就会执行你输入的命令，并在窗口中显示运行的结果。注意，print() 命令是在 Python 3.0 中引入的重要改变之一，如果你运行命令时出错了，请检查一下你运行的是不是 3.0 版本的 IDLE。

你可以把命令行当成计算器来计算表达式的值：

```
>>> 3+4+5
12
```

你可以把命令提示符后面输入的每一条命令看成一个程序，只不过每次只运行了这个程序中的一行而已。你可以在命令行中创建变量或导入模块：

```
>>> import math
>>> (1 + math.sqrt(5)) / 2
1.618033988749895
```

import 命令把 Python 数学函数库的功能都导入你的程序供你使用（参考“对象与模块”了解更多具体的信息）。如果要创建变量，可以使用赋值运算符（=）：

```
>>> import math
>>> radius = 20
>>> radius * 2 * math.pi
125.66370614359173
```

如果你想清除所有的变量并恢复到初始状态，选择 Shell → Restart Shell 即可。你可以通过 help 命令获取某个表达式、模块或

其他 Python 功能的描述：

```
help("print")
```

如果要显示主题的列表，可以运行：

```
help("topics")
help("keywords")
help("modules")
```

Python 解释器很适合用于测试表达式或进行简单的操作，但通常更希望把 Python 脚本当作一个程序来运行。要新建一个 Python 程序，选择菜单中的 File → New Window，IDLE 会帮你打开一个脚本编辑窗口（图 3.2）。

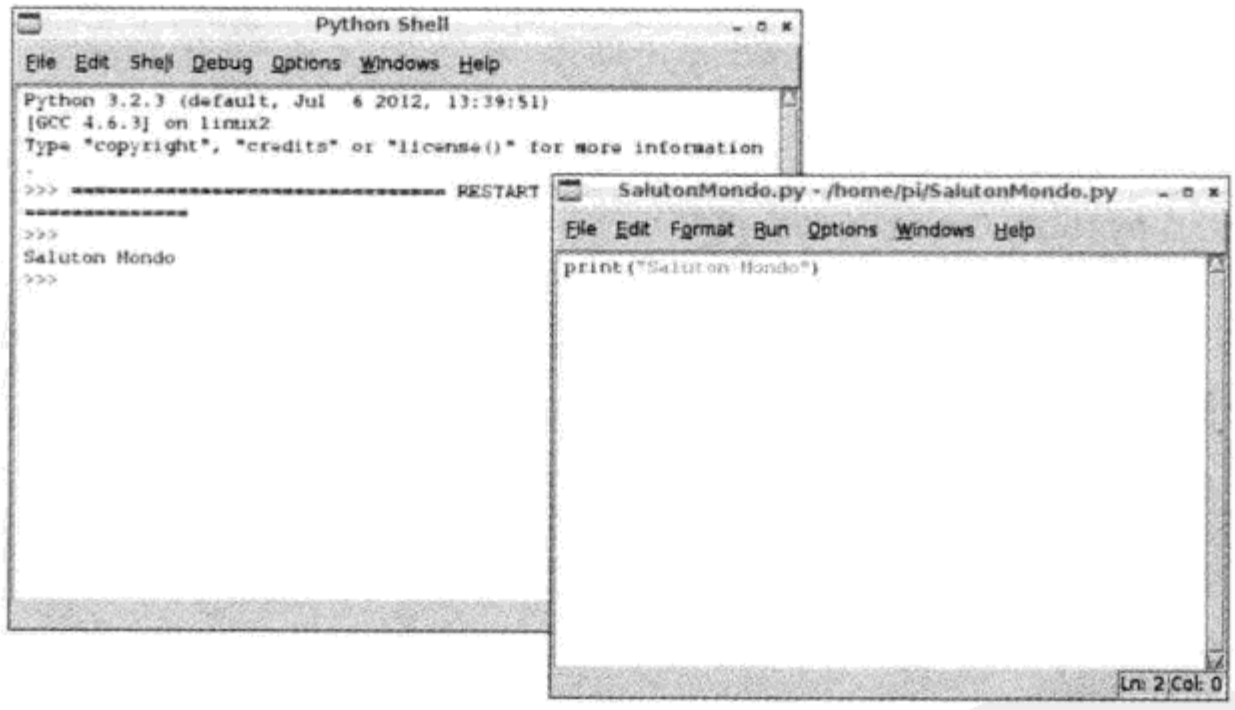


图3.2　IDLE的交互式命令行（左）与脚本编辑窗口（右）

你可以尝试输入一些命令，然后选择 Run → Run Module，系统会弹出一个警告框“Source Must Be Saved OK To Save?”。把这个脚本保存到你的主目录中，命名为 *SalutonMondo.py*，你就可以在终端中执行它了。

如果不想通过 IDLE 环境来运行程序，你也可以打开 LX 终端（LXTerminal），在命令行上输入：

```
python SalutonMondo.py
```

来运行这个程序。

前面介绍的都是最基本的环境使用知识，下面我们可以真正开始学习这门语言了。

命令行与 IDLE

你可能会发现，在 IDLE 中运行示例代码向终端输出内容时速度非常缓慢。要想知道有多慢，你可以同时打开一个 IDLE 窗口和一个 LX 终端（LXTerminal）窗口，在 IDLE 中把下一节中的 *CountEvens.py* 脚本保存为文件并在命令行上执行它：

```
python CountEvens.py
```

在此之前，也可以用 IDLE 中的 Run Module 菜单项运行同一个脚本，你就能马上发现在 Pi 这种资源有限的设备上使用 IDE 所带来的额外性能开销。本书后续的例子都将在命令行中直接运行，不过你仍然可以用 IDLE 作为你的代码编辑器。

进一步学习 Python

如果以前用过 Arduino，你会习惯把程序（在 Arduino 术语中称为 Sketch，但在 Python 中称为脚本）写成 `setup/loop` 结构，`setup` 只在程序启动时运行一次，而 `loop` 函数会一直被循环执行。下面的例子展示了如何在 Python 中实现这样的结构。在 IDLE 3 中

选择 New Windows 创建一个新窗口，输入下面的代码：

```
# Setup
n = 0
# Loop
while True:
    n = n + 1
    # The % is the modulo operator
    if ((n % 2) == 0):
        print(n)
```

执行 Run Module 并给你的脚本起一个名字（如 *EvenIntegers.py*）。当程序运行起来后，你可以看到屏幕上依次打印所有的偶数（按 Control-C 键中断程序，否则它会一直运行下去）。

在上面的代码示例中，每一层的缩进都用了 4 个空格，而不是 Tab 键（不过在 IDLE 中你也可以用 Tab 键缩进，它会自动帮你转换成多个空格）。在 Python 中，缩进是程序语意的一部分。这对于初学者来说，可能是一个学习的障碍；当复制 / 粘贴代码时，也可能会带来一些麻烦。但是，我们仍然认为严格的缩进要求使 Python 成为一种非常易读的语言。参考 Python Style Guidelines（*http://www.python.org/dev/peps/pep-0008/#indentation*）学习如何编写可读性高的代码。

在 Python 程序中，空格非常重要，空格和缩进决定了程序的逻辑结构。在下面的例子中，`loop()` 函数下面同一缩进层次的代码被定义为这个函数的函数体。当代码的缩进层次提高一层（或到达文件结尾）时，就意味着循环的结束。这与 C 语言等以花括号或其他符号作为代码块分隔符的语言很不一样。

通过定义函数，可以把一个代码块组合成一个整体，在脚本中其他地方进行调用。我们可以用函数把上一个例子进行重写，如下（运行时，请把它保存为 *CountEvens.py*）：

```
# Declare global variables
n = 0 ❶

# Setup function
def setup(): ❷
    global n
    n = 100
def loop(): ❸
    global n
    n = n + 1
    if ((n % 2) == 0):
        print(n)

# Main ❹
setup()
while True:
    loop()
```

在这个例子中，输出的值是从 102 开始的所有偶数。解释如下。

❶ 首先，变量 n 被定义成全局变量，所以它可在整个脚本的任意位置使用。

❷ 这里定义了 setup() 函数（但还没有执行它）。

❸ 类似地，这里定义了 loop() 函数。

❹ 在主程序段中，setup() 被调用了一次，然后循环调用 loop()。

每个函数第一行的 global 关键字非常重要，它告诉解释器，这个函数中将要使用全局变量 n，而不是在这个函数中创建一个只能在本函数中使用的新的局部变量（local，只对本函数有效）n。

受篇幅所限，我们的教程不可能写成一份完整的 Python 参考

手册，如果要深入学习这门语言，可以参考 *Think Python*（*http://shop.oreilly.com/product/0636920025696.do*） 和 *Python Pocket Reference*（*http://shop.oreilly.com/product/9780596158095.do*）。 本章的后续内容会向你介绍编写与运行后续章节的示例程序中所需要用到的 Python 基础知识和一些有用的模块。第 4 章会介绍 Pygame 框架，它是在 Pi 上编写多媒体应用程序的好帮手。

对象与模块

要理解本书后续的示例代码，你需要理解有关对象和模块的基本语法。Python 是一种很纯净的语言，只有 34 个保留的关键字（表 3.1）。这些关键字是语言的核心部件，在编写脚本时，通过使用它们来构造程序结构与流程。在 Python 中，除了这些关键字以外的所有东西都可被看成是一个对象。所谓对象，是指数据和与其相关操作的结合体，并拥有一个名字。你可以修改一个对象的数据、获取它的相关信息或通过它操作其他对象。

表3.1 Python只有34个保留关键字

条 件	循 环	内置函数	类、模块与函数	错误处理
if	for	print	class	try
else	in	pass	def	except
elif	while	del	global	finally
not	break		lambda	raise
or	as		nonlocal	assert
and	continue		yield	with
is			import	
True			return	
False			from	
None				

在 Python 中，字符串、列表、函数、模块甚至数字都是对象。一个 Python 的对象可以被理解为是对一组属性和方法的封装，你可以通过点（.）语法来访问这些属性和方法。例如，在交互命令行中输入下面的代码创建一个 String（字符串）对象，然后调用它自身的一个方法把这个字符串转换为首字母大写的形式：

```
>>> myString = "quux"
>>> myString.capitalize()
'Quux'
```

也可以用 List（列表）对象的 reservse() 方法对列表中元素的顺序进行反转：

```
>>> myList = ['a', 'man', 'a', 'plan', 'a', 'canal']
>>> myList.reverse()
>>> print(myList)
['canal', 'a', 'plan', 'a', 'man', 'a']
```

String 和 List 都是 Python 标准库中的内置模块，任何 Python 程序中都可以使用。String 和 List 模块各自定义了一系列的函数分别用于处理字符串和列表，包括我们上面演示过的 capitalize( ) 和 reverse( )。

有一些标准库模块不是内置的，如果要使用它们，必须显式地使用 import 命令。例如，要使用标准库中的 time（时间）模块来进行时间相关的处理，需要：

```
import time
```

你也可以用 import as 在你的程序中加载一个模块并改变它的名字：

```
import time as myTime
```

还可以用 from import 来指定加载某个模块中的部分功能：

```
from time import clock
```

下面是一个简单的程序，使用 Python 脚本调用标准库中的 time 和 datetime 模块，实现每秒钟显示一次当前的时间：

```
from datetime import datetime
from time import sleep

while True:
    now = str(datetime.now())
    print(now)
    sleep(1)
```

sleep 函数可以让程序的运行暂停 1s。在运行这段程序时，你可能会发现每次打印的时间与实际的时间稍稍有些偏移。造成这个问题的原因如下。

- 这段代码没有考虑程序计算当前时间所花费的时间（所以暂停 .9s[①]也许是一个更好的选择）。
- 其他的进程也在分享 CPU 资源，在你的程序执行时，CPU 的计算周期也可能被其他进程占用。这一点非常重要：在 Raspberry Pi 上编写程序时，你并没有工作在一个实时环境中。

在 Pi 上，sleep() 函数的精度在 5ms 以内。

下面，我们修改一下这个示例程序，引入一个文本文件并定期往里面写入一些日志数据。在处理文本文件时，所有的处理对象都是字符串。可以用 str() 函数把数字转换成字符串，也可以用

① 在计算机语言中，常常可以省略掉整数位的 0，所以这里的 .9 就是 0.9。——译者注

int() 函数把字符串转换回整型数。

```
from datetime import datetime
from time import sleep
import random

log = open("log.txt", "w")

for i in range(5):
    now = str(datetime.now())
    # Generate some random data in the range 0-1024
    data = random.randint(0, 1024)

    log.write(now + " " + str(data) + "\n")
    print(".")
    sleep(.9)
log.flush()
log.close()
```

在一个实际的数据日志记录程序中，你需要保证你的 Raspberry Pi 已经按第 2 章所讲述的方法设置了正确的日期时间。

下面是另外一个例子（*ReadFile.py*），它从命令行参数读入一个文件名（用 python3 ReadFile.py filename 的形式在命令行中运行这个程序），然后程序打开这个文件，读出文件中的每一行并显示在屏幕上。在 Python 中，print() 函数与其他语言中的 println() 函数很相似，会在每次输出后自动添加一个换行。用 print() 函数的 end 参数可以改变这个添加换行的行为。

```
# Open and read a file from command line argument
import sys
```

```
if (len(sys.argv) != 2):
    print("Usage: python ReadFile.py filename")
    sys.exit()
scriptname = sys.argv[0]
filename = sys.argv[1]

file = open(filename, "r")
lines = file.readlines()
file.close()

for line in lines:
    print(line,end='' )
```

更多模块

Python 之所以广受欢迎是因为它有大量由用户基于标准库开发的模块。Python 包索引（Python Package Index， PyPI，*http://pypi.python.org/pypi*）是一个完整的包和模块的列表，其中的一部分在 Raspberry Pi 上尤其有用，如表 3.2 所示。后续你会陆续使用到这些模块，尤其是用于操作 Raspberry Pi 通用输入输出接口的 GPIO 模块。

表3.2 一些对Pi用户尤其有用的Python包

模块名	描 述	URL	软件包名
RPi.GPIO	访问 GPIO 接口	*http://code.google.com/p/raspberry-gpio-python/*	*python-rpi.gpio*
Pygame	游戏开发框架	*http://pygame.org*	*python- pygame*
SimpleCV	简易的计算机视觉库	*http://simplecv.org/*	没有打包
Scipy	科学计算	*http://www.scipy.org/*	*python-scipy*
Numpy	Scipy 库的数值基础库	*http://numpy.scipy.org/*	*python-numpy*
Flask	微型 Web 开发框架	*http://flask.pocoo.org/*	*python-flask*
Requests	友好的 HTTP 协议库	*http://docs.python-requests.org*	*python-requests*
PIL	图像处理	*http://www.pythonware.com/products/pil/*	*python-imaging*

续表3.2

模块名	描　述	URL	软件包名
wxPython	图形用户界面（GUI）框架	*http://wxpython.org*	*python-wxgtk2.8*
PySerial	串口通信	*http://pyserial.sourceforge.net/*	*python-serial*
pyUSB	FTDI-USB 接口	*http://bleyer.org/pyusb*	没有打包

要使用这些模块，你需要下载代码、配置包并安装。例如，PySerial 模块可以这样安装：

```
sudo apt-get install python-serial
```

如果一个包的开发者使用了标准的打包方式（通过使用 Python 的 distutils 工具），那么安装这个包只需下载代码、解压缩并执行下面的命令即可：

```
python setup.py install
```

你还可以学习一下 Pip package installer（*http://www.pip-installer.org*），它使从 PyPI 上安装包的过程变得非常容易。

错误调试

你可能会遇到代码运行出错、需要调试的情况，这时 IDLE 的交互模式就很有用，它的 Debug 菜单中提供了一些有用的工具，帮助你了解程序的实际运行情况。你可以在程序运行过程中查看各变量的值或单步执行程序。

语法错误是最容易解决的问题，这类问题常常是输入错误或对语言语法的理解有偏差造成的。语义错误就会难发现得多，这种错

误表现为程序语法正确但运行结果不是我们所想要的。调试器可以在这种情况下大显身手。熟练地掌握各种调试技巧可能需要数年的积累，但以下内容可以帮助你调试在 Pi 上运行的 Python 程序中一些最常见的问题：

- 用 `print()` 打印信息表明程序曾经执行到这个位置。
- 用 `print()` 显示程序运行过程中变量的值。
- 仔细确认代码块缩进是否与你设想的逻辑一致。
- 如果是调试语法错误，要意识到真正出错的地方也许在解释器给你报错的位置的前面。
- 仔细检查所有的局部变量与全局变量。
- 检查开闭括号是否都有正确的匹配。
- 确认表达式中运算符的优先级是否正确，如果不太确定，就使用括号来明确优先级。例如 `3+4*2` 与 `(3+4)*2` 得到的是不同的结果。

进一步学习

有关 Python，还有很多的东西可以学习，以下资源也许会对你有所帮助。

Allen Downey 所著的 *Think Python*

（*http://shop.oreilly.com/product/0636920025696.do*）

这本书简明而又精炼地介绍了有关编程的入门知识，并正好使用了 Python 作为教学语言。

The Python Pocket Reference

（*http://shop.oreilly.com/product/9780596158095.do*）

有时候，完整地读完一本书会比在 Stack Overflow 网站上查阅一堆帖子更有价值。

Stack Overflow

（*http://stackoverflow.com/*）

Stack Overflow 是一个分享群体知识的网站，尤其当你在寻找一个特定的解决方案或分析某个特定的出错提示时，你遇到的问题常常也是别人曾经遇到过的问题。

Learn Python the Hard Way

（*http://learnpythonthehardway.org/*）

一本出色的书及在线资源。至少你应该先读一读它的序言 *The Hard Way Is Easier*。

Jason R. Briggs 所著的 *Python For Kids*

（*http://shop.oreilly.com/product/9781593274078.do*）

又一本介绍编程的入门书，正好用了 Python 作为教学语言，这本书适合年龄较小的读者。