

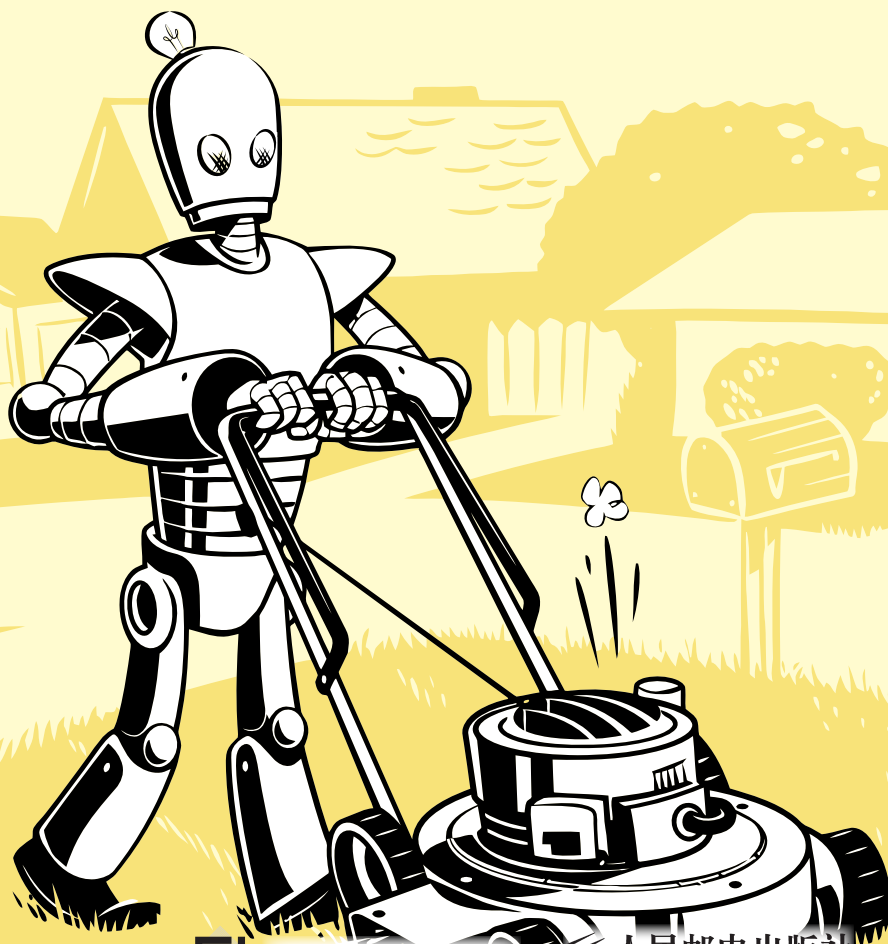
需要整本电子书，联系我QQ：2667271557  
资深Python程序员力作 带你快速掌握Python高效编程



# Python编程快速上手 ——让繁琐工作自动化

AUTOMATE THE BORING STUFF WITH PYTHON

[美] Al Sweigart 著 王海鹏 译



中国工信出版集团 人民邮电出版社  
POSTS & TELECOM PRESS  
需要整本电子书，联系我QQ：2667271557

需要整本电子书，联系我QQ：2667271557



# Python编程快速上手 ——让繁琐工作自动化

AUTOMATE THE BORING STUFF WITH PYTHON

[美] Al Sweigart 著 王海鹏 译

人民邮电出版社

需要整本电子书，联系我QQ：2667271557

# 需要整本电子书，联系我QQ：2667271557

## 图书在版编目（C I P）数据

Python编程快速上手：让繁琐工作自动化 / (美)  
斯维加特 (Al Sweigart) 著；王海鹏译. — 北京：人  
民邮电出版社，2016.7  
ISBN 978-7-115-42269-9

I. ①P… II. ①斯… ②王… III. ①软件工具—程序  
设计 IV. ①TP311.56

中国版本图书馆CIP数据核字(2016)第110206号

## 版 权 声 明

Simplified Chinese-language edition copyright © 2016 by Posts and Telecom Press.

Copyright © 2015 by Al Sweigart. Title of English-language original: Automate The Boring Stuff with  
Python ISBN-13: 978-1-59327-599-0, published by No Starch Press.

All rights reserved.

本书中文简体字版由美国 No Starch 出版社授权人民邮电出版社出版。未经出版者书面许可，对本书  
任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

- 
- ◆ 著 [美] Al Sweigart  
译 王海鹏  
责任编辑 陈冀康  
责任印制 焦志炜
- ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号  
邮编 100164 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京鑫正大印刷有限公司印刷
- ◆ 开本：800×1000 1/16  
印张：26.25  
字数：590 千字 2016 年 7 月第 1 版  
印数：1—3 000 册 2016 年 7 月北京第 1 次印刷
- 著作权合同登记号 图字：01-2015-2962 号
- 

定价：69.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316  
反盗版热线：(010)81055315

# 需要整本电子书，联系我QQ：2667271557

## 内容提要

如今，人们面临的大多数任务都可以通过编写计算机软件来完成。Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言。通过 Python 编程，我们能够解决现实生活中的很多任务。

本书是一本面向实践的 Python 编程实用指南。本书的目的，不仅是介绍 Python 语言的基础知识，而且还通过项目实践教会读者如何应用这些知识和技能。本书的第一部分介绍了基本的 Python 编程概念，第二部分介绍了一些不同的任务，通过编写 Python 程序，可以让计算机自动完成它们。第二部分的每一章都有一些项目程序，供读者学习。每章的末尾还提供了一些习题和深入的实践项目，帮助读者巩固所学的知识。附录部分提供了所有习题的解答。

本书适合任何想要通过 Python 学习编程的读者，尤其适合缺乏编程基础的初学者。通过阅读本书，读者将能利用最强大的编程语言和工具，并且将体会到 Python 编程的快乐。

需要整本电子书，联系我QQ：2667271557

## 作者简介

Al Sweigart 是一名软件开发者和技术图书作者，居住在旧金山。Python 是他最喜欢的编程语言，他开发了几个开源模块。他的其他著作都在他的网站 <http://www.inventwithpython.com/> 上。

## 技术评审者简介

Ari Lacenski 是 Android 应用程序和 Python 软件开发者。她住在旧金山，她写了一些关于 Android 编程的文章，放在 <http://gradlewhy.ghost.io/> 上，并与 Women Who Code 合作提供指导。她还是一个民谣吉他手。

需要整本电子书，联系我QQ：2667271557

## 致谢

没有很多人的帮助，我不可能写出这样一本书。我想感谢 Bill Pollock，我的编辑 Laurel Chun、Leslie Shen、Greg Poulos 和 Jennifer Griffith-Delgado，以及 No Starch Press 的其他工作人员，感谢他们非常宝贵的帮助。感谢我的技术评审 Ari Lacenski，她提供了极好的建议、编辑和支持。

非常感谢 Guido van Rossum，以及 Python 软件基金会的每个人，感谢他们了不起的工作。Python 社区是我在业界看到的最佳社区。

最后，我要感谢我的家人和朋友，以及在 Shotwell 的伙伴，他们不介意我在写这本书时忙碌的生活。干杯！

## 译者序 会编程的人不一样

这是机器代替人的时代，也是人控制机器的时代。这是程序员的时代，也是非程序员学编程的时代。这是算法的时代，也是编程语言的时代。翻译本书期间，深度学习的人工智能程序 AlphaGo 以 4:1 击败了李世石九段。

每一个不会编程的年轻人都应该认真考虑：是不是应该开始学习编程？

学习一门新的语言，总是让人感到畏缩。这让我想起大学时英语老师教的学习方法：听说领先，读写跟上。确实，学语言效果最好的方法就是“用”。本书就遵循了这样的宗旨。本书是面对编程初学者的书，假定读者没有任何编程知识。在简单介绍 Python 编程语言的基本知识后，就开始用一个接一个的例子，教我们如何用 Python 来完成一些日常工作，利用计算机这个强大的工具，节省工作时间，提高工作效率，避免手工操作容易带来的错误。

真正的程序员，用编程来解决自己和别人的问题。俄罗斯有一个程序员编写了一个程序，会给老婆发加班短信，会在宿醉不醒时给自己请假，会自动根据邮件恢复客户的数据库，还可以一键远程煮咖啡。加拿大一名零编程基础的农场主，在学习了一门编程课后，开发了一个程序，自动控制拖拉机，配合联合收割机收割谷物。

若是已经掌握了其他编程语言，想学习 Python，本书也是不错的参考。每一种编程语言，都会提供一种独特的视角，让你对编程有新的认识。我非常喜欢 Python 没有花括号和分号，程序很“清爽”，符合奥卡姆剃刀原理：如无必要，勿增实体。本书并没有深入介绍面向对象和函数式编程范式，如果想了解 Python 这方面的内容，请参考其他书籍。

在本书的翻译过程，我自己也在项目中使用 Python 编程，从中得到许多启发。因此，郑重向大家推荐。翻译中的错误，请不吝指出。

王海鹏  
2016 年春于上海

# 前言



“你在 2 个小时里完成的事，我们 3 个人要做两天。” 21 世纪早期，我的大学室友在一个电子产品零售商店工作。商店偶尔会收到一份电子表格，其中包含竞争对手的数千种产品的价格。由 3 个员工组成的团队，会将这个电子表格打印在一叠厚厚的纸上，然后 3 个人分一下。针对每个产品价格，他们会查看自己商店的价格，并记录竞争对手价格较低的所有产品。这通常会花几天的时间。

“如果你有打印件的原始文件，我会写一个程序来做这件事。” 我的室友告诉他们，当时他看到他们坐在地板上，周围都是散落堆叠的纸张。

几个小时后，他写了一个简短的程序，从文件读取竞争对手的价格，在商店的数据库中找到该产品，并记录竞争对手是否更便宜。他当时还是编程新手，花了许多时间在一本编程书籍中查看文档。实际上程序只花了几秒钟运行。我的室友和他的同事们那天享受了超长的午餐。

这就是计算机编程的威力。计算机就像瑞士军刀，可以用来完成数不清的任务。许多人花上数小时点击鼠标和敲打键盘，执行重复的任务，却没有意识到，如果他们给机器正确的指令，机器就能在几秒钟内完成他们的工作。

## 本书的读者对象

软件是我们今天使用的许多工具的核心：几乎每个人都使用社交网络来进行交流，许多人的手机中都有连接因特网的计算机，大多数办公室工作都涉及操作计算机来完成工作。因此，对编程人才的需求暴涨。无数的图书、交互式网络教程和开发者新兵训练营，承诺将有雄心壮志的初学者变成软件工程师，获得 6 位数的薪水。



本书不是针对这些人的。它是针对所有其他的人。

就它本身来说，这本书不会让你变成一个职业软件开发者，就像几节吉他课程不会让你变成一名摇滚巨星。但如果你是办公室职员、管理者、学术研究者，或使用计算机来工作或娱乐的任何人，你将学到编程的基本知识，这样就能将下面这样一些简单的任务自动化：

- 移动并重命名几千个文件，将它们分类，放入文件夹；
- 填写在线表单，不需要打字；
- 在网站更新时，从网站下载文件或复制文本；
- 让计算机向客户发出短信通知；
- 更新或格式化 Excel 电子表格；
- 检查电子邮件并发出预先写好的回复。

对人来说，这些任务简单，但很花时间。它们通常很琐碎、很特殊，没有现成的软件可以完成。有一点编程知识，就可以让计算机为你完成这些任务。

## 编码规范

本书没有设计成参考手册，它是初学者指南。编码风格有时候违反最佳实践（例如，有些程序使用全局变量），但这是一种折中，让代码更简单，以便学习。本书的目的是让人们编写用完即抛弃的代码，所以没有太多时间来关注风格和优雅。复杂的编程概念（如面向对象编程、列表推导和生成器），在本书中也没有介绍，因为它们增加了复杂性。编程老手可能会指出，本书中的代码可以修改得更有效率，但本书主要考虑的是用最少的工作量得到能工作的程序。

## 什么是编程

在电视剧和电影中，常常看到程序员在闪光的屏幕上迅速地输入密码般的一串 1 和 0，但现代编程没有这么神秘。编程只是输入指令让计算机来执行。这些指令可能运算一些数字，修改文本，在文件中查找信息，或通过因特网与其他计算机通信。

所有程序都使用基本指令作为构件块。下面是一些常用的指令，用自然语言的形式来表示：

“做这个，然后做那个。”

“如果这个条件为真，执行这个动作，否则，执行那个动作。”

“按照指定次数执行这个动作。”

“一直做这个，直到条件为真。”

也可以组合这些构件块，实现更复杂的决定。例如，这里有一些编程指令，称为源代码，是用 Python 编程语言编写的一个简单程序。从头开始，Python 软件执

行每行代码（有些代码只有在特定条件为真时执行，否则 Python 会执行另外一些代码），直到到达底部。

---

```
❶ passwordFile = open('SecretPasswordFile.txt')
❷ secretPassword = passwordFile.read()
❸ print('Enter your password.')
  typedPassword = input()
❹ if typedPassword == secretPassword:
❺     print('Access granted')
❻     if typedPassword == '12345':
❼         print('That password is one that an idiot puts on their luggage.')
else:
❽     print('Access denied')
```

---

你可能对编程一无所知，但读了上面的代码，也许就能够合理地猜测它做的事。首先，打开了文件 `SecretPasswordFile.txt`❶，读取了其中的密码❷。然后，提示用户（通过键盘）输入一个密码❸。比较这两个密码❹，如果它们一样，程序就在屏幕上打印 `Access granted`❺。接下来，程序检查密码是否为 `12345`❻，提示说这可能并不是最好的密码❼。如果密码不一样，程序就在屏幕上打印 `Access denied`❽。

## 什么是 Python

Python 指的是 Python 编程语言（包括语法规则，用于编写被认为是有效的 Python 代码），以及 Python 解释器软件，它读取源代码（用 python 语言编写），并执行其中的指令。Python 解释器可以从 <http://python.org/> 免费下载，有针对 Linux、OS X 和 Windows 的版本。

Python 的名字来自于英国超现实主义喜剧团体，而不是来自于蛇。Python 程序员被亲切地称为 Pythonistas。Monty Python 和与蛇相关的引用常常出现在 Python 的指南和文档中。

## 程序员不需要知道太多数学

我听到的关于学习编程的最常见的顾虑，就是人们认为这需要很多数学知识。其实，大多数编程需要的数学知识不超过基本算数。实际上，善于编程与善于解决数独问题没有太大差别。

要解决数独问题，数字 1 到 9 必须填入 9×9 的棋盘上每一行、每一列，以及每个 3×3 的内部方块。通过推导和起始数字的逻辑，你会找到一个答案。例如，在图 1 的数独问题中，既然 5 出现在了左上角，它就不能出现在顶行、最左列，或左上角 3×3 方块中的其他位置。每次解决一行、一列或一个方块，将为剩下的部分提供更多的数字线索。

仅仅因为数独使用了数字，并不意味着必须精通数学才能求出答案。编程也是这样。就像解决数独问题一样，编程需要将一个问题分解为单个的、详细的步骤。类似地，在调试程序时（即寻找和修复错误），你会耐心地观察程序在做什么，找

出缺陷的原因。像所有技能一样，编程越多，你就掌握得越好。

5	3			7					
6			1	9	5				
	9	8					6		
8				6					3
4			8		3				1
7				2					6
	6					2	8		
			4	1	9				5
				8			7	9	

5	3	4	6	7	8	9	1	2	
6	7	2	1	9	5	3	4	8	
1	9	8	3	4	2	5	6	7	
8	5	9	7	6	1	4	2	3	
4	2	6	8	5	3	7	9	1	
7	1	3	9	2	4	8	5	6	
9	6	1	5	3	7	2	8	4	
2	8	7	4	1	9	6	3	5	
3	4	5	2	8	6	1	7	9	

图1 一个新的数独问题（左边）及其答案（右边）。尽管使用了数字，数独并不需要太多数学知识

## 编程是创造性活动

编程是一项创造性任务，有点类似于用乐高积木构建一个城堡。你从基本的想法开始，希望城堡看起来像怎样，并盘点可用的积木。然后开始构建。在你完成构建程序后，可以让代码变得更美观，就像对你的城堡那样。

编程与其他创造性活动的不同之处在于，在编程时，你需要的所有原材料都在计算机中，你不需要购买额外的画布、颜料、胶片、纱线、乐高积木或电子器件。在程序写好后，很容易将它在线共享给整个世界。尽管在编程时你会犯错，这项活动仍然很有乐趣。

## 本书简介

本书的第一部分介绍了基本 Python 编程概念，第二部分介绍了一些不同的任务，你可以让计算机自动完成它们。第二部分的每一章都有一些项目程序，供你学习。下面简单介绍一下每章的内容。

### 第一部分：Python 编程基础

“第1章：Python 基础”介绍了表达式、Python 指令的最基本类型，以及如何使用 Python 交互式环境来尝试运行代码。

“第2章：控制流”解释了如何让程序决定执行哪些指令，以便代码能够智能地响应不同的情况。

“第3章：函数”介绍了如何定义自己的函数，以便将代码组织成可管理的部分。

“第4章：列表”介绍了列表数据类型，解释了如何组织数据。

“第5章：字典和结构化数据”介绍了字典数据类型，展示了更强大的数据组织方法。

“第 6 章：字符串操作”介绍了处理文本数据（在 Python 中称为字符串）。

## 第二部分：自动化任务

“第 7 章：模式匹配与正则表达式”介绍了 Python 如何用正则表达式处理字符串，以及查找文本模式。

“第 8 章：读写文件”解释了程序如何读取文本文件的内容，并将信息保存到硬盘的文件中。

“第 9 章：组织文件”展示了 Python 如何用比手工操作快得多的速度，复制、移动、重命名和删除大量的文件，也解释了压缩和解压缩文件。

“第 10 章：调试”展示了如何使用 Python 的缺陷查找和缺陷修复工具。

“第 11 章：从 Web 抓取信息”展示了如何编程来自动下载网页，解析它们，获取信息。这称为从 Web 抓取信息。

“第 12 章：处理 Excel 电子表格”介绍了编程处理 Excel 电子表格，这样你就不必去阅读它们。如果你必须分析成百上千的文档，这是很有帮助的。

“第 13 章：处理 PDF 和 Word 文档”介绍了编程读取 Word 和 PDF 文档。

“第 14 章：处理 CSV 文件和 JSON 数据”解释了如何编程操作 CSV 和 JSON 文件。

“第 15 章：保持时间、计划任务和启动程序”解释了 Python 程序如何处理时间和日期，如何安排计算机在特定时间执行任务。这一章也展示了 Python 程序如何启动非 Python 程序。

“第 16 章：发送电子邮件和短信”解释了如何编程来发送电子邮件和短信。

“第 17 章：操作图像”解释了如何编程来操作 JPG 或 PNG 这样的图像。

“第 18 章：用 GUI 自动化控制键盘和鼠标”解释了如何编程控制鼠标和键盘，自动化鼠标点击和击键。

## 下载和安装 Python

可以从 <http://python.org/downloads/> 免费下载针对 Windows、OS X 和 Ubuntu 的 Python 版本。如果你从该网站的下载页面下载了最新的版本，本书中的所有程序应该都能工作。

### 注意

请确保下载 Python 3 的版本（诸如 3.4.0）。本书中的程序将运行在 Python 3 上，有一部分程序在 Python 2 上也许不能正常运行。

你需要在下载页面上找到针对 64 位或 32 位计算机以及特定操作系统的 Python 安装程序，所以先要弄清楚你需要哪个安装程序。如果你的计算机是 2007 年或以后购买的，很有可能是 64 位的系统。否则，可能是 32 位的系统，但下面是确认

的方法：

- 在 Windows 上。选择 Start▶ControlPanel▶System。检查系统类型是 64 位或 32 位。
- 在 OS X 上，进入 Apple 菜单，选择 About This Mac▶MoreInfo▶SystemReport▶Hardware，然后查看 Processor Name 字段。如果是 Intel Core Solo 或 Intel Core Duo，机器是 32 位的。如果是其他（包括 Intel Core 2 Duo），机器是 64 位的。
- 在 Ubuntu Linux 上，打开终端窗口，运行命令 `uname -m`。结果是 i686 表示是 32 位，x86\_64 表示是 64 位。

在 Windows 上，下载 Python 安装程序（文件扩展名是.msi），并双击它。按照安装程序显示在屏幕上的指令来安装 Python，步骤如下。

1. 选择 Install for All Users，然后点击 Next。
2. 通过点击 Next 安装到 C:\Python34 文件夹。
3. 再次点击 Next，跳过定制 Python 的部分。

在 OS X 上，下载适合你的 OS X 版本的.dmg 文件，并双击它。按照安装程序显示在屏幕上的指令来安装 Python，步骤如下。

1. 当 DMG 包在一个新窗口中打开时，双击 Python.mpkg 文件。你可能必须输入管理员口令。

2. 点击 Continue，跳过欢迎部分，并点击 Agree，接受许可证。
3. 选择 HD Macintosh（或者你的硬盘的名字），并点击 Install。

如果使用的是 Ubuntu，可以从终端窗口安装 Python，步骤如下。

1. 打开终端窗口。
2. 输入 `sudo apt-get install python3`。
3. 输入 `sudo apt-get install idle3`。
4. 输入 `sudo apt-get install python3-pip`。

## 启动 IDLE

Python 解释器是运行 Python 程序的软件，而交互式开发环境（IDLE）是输入程序的地方，就像一个字处理软件。现在让我们启动 IDLE。

- 在 Windows7 或更新的版本上，点击屏幕左下角的开始图标，在搜索框中输入 IDLE，并选择 IDLE（Python GUI）。
- Windows XP 上，点击开始按钮，然后选择 Programs▶Python 3.4▶IDLE（Python GUI）。
- 在 OS X 上，打开 Finder 窗口，点击 Applications，点击 Python 3.4，然后点击 IDLE 的图标。
- 在 Ubuntu 上，选择 Applications▶Accessories▶Terminal，然后输入 `idle3`（也许你也

可以点击屏幕顶部的 Applications，选择 Programming，然后点击 IDLE 3）。

## 交互式环境

无论你使用什么操作系统，初次出现的 IDLE 窗口应该基本上是空的，除了类似下面这样的文本：

---

```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64
bit (AMD64)] on win32Type "copyright", "credits" or "license()" for more
information.
>>>
```

---

这个窗口称为交互式环境。这是让你向计算机输入指令的程序，很像 OS X 上的终端窗口，或 Windows 上的命令行提示符。Python 的交互式环境让你输入指令，供 Python 解释器软件来执行。计算机读入你输入的指令，并立即执行它们。

例如，在交互式环境的>>>提示符后输入以下指令：

---

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

---

在输入该行并按下回车键后，交互式环境将显示以下内容作为响应：

---

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

---

## 如何寻求帮助

独自解决编程问题可能比你想象的要容易。如果你不相信，就让我们故意产生一个错误：在交互式环境中输入'42' + 3。现在你不需要知道这条指令是什么意思，但结果看起来应该像这样：

---

```
>>> '42' + 3
❶ Traceback (most recent call last):
  File "<pyshe11#0>", line 1, in <module>
    '42' + 3
❷ TypeError: Can't convert 'int' object to str implicitly
>>>
```

---

这里出现了错误信息❷，因为 Python 不理解你的指令。错误信息的 traceback 部分❶显示了 Python 遇到困难的具体指令和行号。如果你不知道怎样处理特定的错误信息，就在线查找那条错误信息。在你喜欢的搜索引擎上输入“TypeError: Can't convert 'int' object to str implicitly”（包括引号），你就会看到许多的链接，解释这条错误信息的含义，以及什么原因导致这条错误，如图 2 所示。

你常常会发现，别人也遇到了同样的问题，而其他乐于助人的人已经回答了这个问题。没有人知道编程的所有方面，所以所有软件开发者的工作，都是每天在寻找技术问题的答案。

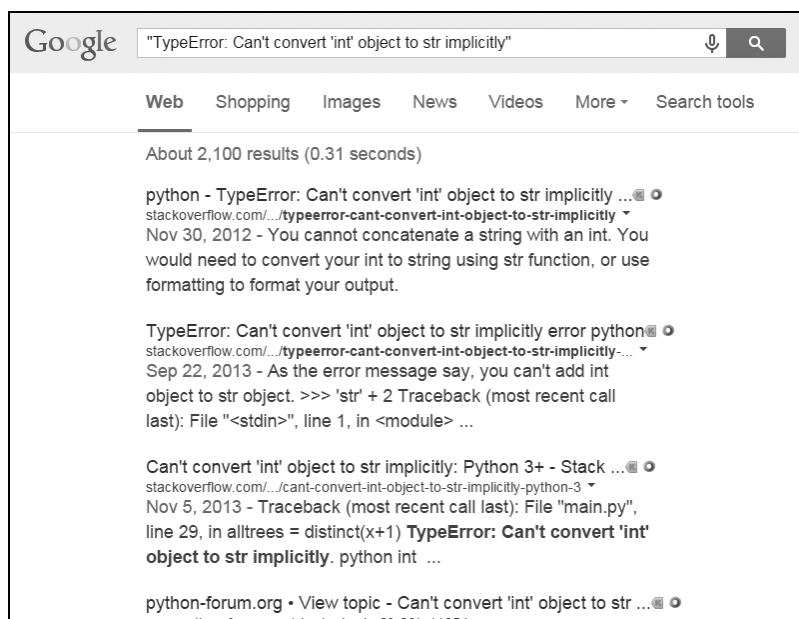


图2 错误信息的 Google 搜索结果可能非常有用

## 聪明地提出编程问题

如果不能在线查找到答案，请尝试在 Stack Overlow (<http://stackoverflow.com/>) 或 “learnprogramming” subreddit (<http://reddit.com/r/learnprogramming/>) 这样的论坛上提问。但要记住，用聪明的方式提出编程问题，这有助于别人来帮助你。确保阅读这些网站的 FAQ（常见问题），了解正确的提问方式。

在提出编程问题时，要记住以下几点。

- 说明你打算做什么，而不只是你做了什么。这让帮助你的人知道你是否走错了路。
- 明确指出发生错误的地方。它是在程序每次启动时发生，还是在你做了某些动作之后？
- 将完整的错误信息和你的代码复制粘贴到 <http://pastebin.com/> 或 <http://gist.github.com/>。

这些网站让你很容易在网上与他人共享大量的代码，而不会丢失任何文本格式。然后你可以将贴出的代码的 URL 放在电子邮件或论坛帖子中。例如，这里是我贴出的一些代码片段：<http://pastebin.com/SzP2DbFx/> 和 <https://gist.github.com/asweigart/6912168/>。

- 解释你为了解决这个问题已经尝试了哪些方法。这会告诉别人你已经做了一些工作来弄清楚状况。

- 列出你使用的 Python 版本（Python 2 解释器和 Python3 解释器之间有一些重要的区别）。而且，要说明你使用的操作系统和版本。
- 如果错误在你更改了代码之后出现，准确说明你改了些什么。
- 说明你是否在每次运行该程序时都能重现该错误，或者它只是在特定的操作执行之后才出现。如果是这样，解释是哪些操作。

也要遵守良好的在线礼节。例如，不要全用大写提问，或者对试图帮助你的人提出无理的要求。

## 小结

对于大多数人，他们的计算机只是设备，而不是工具。但通过学习如何编程，你就能利用现代社会中最强大的工具，并且你会一直感到快乐。编程不是脑外科手术，业余人士是完全可以尝试或犯错的。

我喜欢帮助人们探索 Python。我在自己的博客上编写编程指南（<http://inventwithpython.com/blog/>），你可以发邮件向我提问（[al@inventwithpython.com](mailto:al@inventwithpython.com)）。

本书将从零编程知识开始，但你的问题可能超出本书的范围。记住如何有效地提问，知道如何寻找答案，这对你的编程之旅是无价的工具。

让我们开始吧！



# 目 录

## 第一部分 Python 编程基础

第 1 章 Python 基础 .....	3	2.2 比较操作符 .....	19
1.1 在交互式环境中输入表达式 .....	3	2.3 布尔操作符 .....	20
1.2 整型、浮点型和字符串数据类型 .....	6	2.3.1 二元布尔操作符 .....	20
1.3 字符串连接和复制 .....	6	2.3.2 not 操作符 .....	21
1.4 在变量中保存值 .....	7	2.4 混合布尔和比较操作符 .....	21
1.4.1 赋值语句 .....	7	2.5 控制流的元素 .....	22
1.4.2 变量名 .....	9	2.5.1 条件 .....	22
1.5 第一个程序 .....	9	2.5.2 代码块 .....	22
1.6 程序剖析 .....	11	2.6 程序执行 .....	23
1.6.1 注释 .....	11	2.7 控制流语句 .....	23
1.6.2 print()函数 .....	11	2.7.1 if 语句 .....	23
1.6.3 input()函数 .....	11	2.7.2 else 语句 .....	24
1.6.4 打印用户的名字 .....	12	2.7.3 elif 语句 .....	25
1.6.5 len()函数 .....	12	2.7.4 while 循环语句 .....	30
1.6.6 str()、int()和 float()函数 .....	13	2.7.5 恼人的循环 .....	31
1.7 小结 .....	15	2.7.6 break 语句 .....	33
1.8 习题 .....	15	2.7.7 continue 语句 .....	34
第 2 章 控制流 .....	17	2.7.8 for 循环和 range()函数 .....	37
2.1 布尔值 .....	18	2.7.9 等价的 while 循环 .....	39

2.7.10	range()的开始、停止和步长参数	39	4.1.5	用下标改变列表中的值	62
2.8	导入模块	40	4.1.6	列表连接和列表复制	62
	from import 语句	41	4.1.7	用 del 语句从列表中删除值	63
2.9	用 sys.exit()提前结束程序	41	4.2	使用列表	63
2.10	小结	41	4.2.1	列表用于循环	64
2.11	习题	41	4.2.2	in 和 not in 操作符	65
第 3 章	函数	43	4.2.3	多重赋值技巧	66
3.1	def 语句和参数	44	4.3	增强的赋值操作	66
3.2	返回值和 return 语句	45	4.4	方法	67
3.3	None 值	46	4.4.1	用 index()方法在列表中查找值	67
3.4	关键字参数和 print()	47	4.4.2	用 append()和 insert()方法在列表中添加值	68
3.5	局部和全局作用域	48	4.4.3	用 remove()方法从列表中删除值	69
3.5.1	局部变量不能在全局作用域内使用	48	4.4.4	用 sort()方法将列表中的值排序	69
3.5.2	局部作用域不能使用其他局部作用域内的变量	49	4.5	例子程序：神奇 8 球和列表	70
3.5.3	全局变量可以在局部作用域中读取	49	4.6	类似列表的类型：字符串和元组	71
3.5.4	名称相同的局部变量和全局变量	50	4.6.1	可变和不可变数据类型	72
3.6	global 语句	50	4.6.2	元组数据类型	73
3.7	异常处理	52	4.6.3	用 list()和 tuple()函数来转换类型	74
3.8	一个小程序：猜数字	54	4.7	引用	75
3.9	小结	55	4.7.1	传递引用	76
3.10	习题	56	4.7.2	copy 模块的 copy()和 deepcopy()函数	77
3.11	实践项目	56	4.8	小结	78
3.11.1	Collatz 序列	56	4.9	习题	78
3.11.2	输入验证	57	4.10	实践项目	79
第 4 章	列表	59	4.10.1	逗号代码	79
4.1	列表数据类型	59	4.10.2	字符图网格	79
4.1.1	用下标取得列表中的单个值	60	第 5 章	字典和结构化数据	81
4.1.2	负数下标	61	5.1	字典数据类型	81
4.1.3	利用切片取得子列表	61			
4.1.4	用 len()取得列表的长度	62			

5.1.1	字典与列表	82	6.2.1	字符串方法 upper()、lower()、isupper()和 islower()	99
5.1.2	keys()、values()和 items()方法	83	6.2.2	isX 字符串方法	100
5.1.3	检查字典中是否存在键或值	84	6.2.3	字符串方法 startswith()和endswith()	102
5.1.4	get()方法	84	6.2.4	字符串方法 join()和split()	102
5.1.5	setdefault()方法	85	6.2.5	用 rjust()、ljust()和 center()方法对齐文本	103
5.2	漂亮打印	86	6.2.6	用 strip()、rstrip()和 lstrip()删除空白字符	104
5.3	使用数据结构对真实世界建模	87	6.2.7	用 pyperclip 模块拷贝粘贴字符串	105
5.3.1	井字棋盘	88	6.3	项目：口令保管箱	106
5.3.2	嵌套的字典和列表	91	第 1 步：程序设计和数据结构	106	
5.4	小结	92	第 2 步：处理命令行参数	106	
5.5	习题	93	第 3 步：复制正确的口令	107	
5.6	实践项目	93	6.4	项目：在 Wiki 标记中添加无序列表	108
5.6.1	好玩游戏的物品清单	93	第 1 步：从剪贴板中复制和粘贴	108	
5.6.2	列表到字典的函数，针对好玩游戏物品清单	94	第 2 步：分离文本中的行，并添加星号	109	
第 6 章	字符串操作	95	第 3 步：连接修改过的行	109	
6.1	处理字符串	95	6.5	小结	110
6.1.1	字符串字面量	95	6.6	习题	110
6.1.2	双引号	96	6.7	实践项目	111
6.1.3	转义字符	96	表格打印	111	
6.1.4	原始字符串	96			
6.1.5	用三重引号的多行字符串	97			
6.1.6	多行注释	97			
6.1.7	字符串下标和切片	98			
6.1.8	字符串的 in 和 not in 操作符	98			
6.2	有用的字符串方法	99			

## 第二部分 自动化任务

第 7 章	模式匹配与正则表达式	115	7.2	用正则表达式查找文本模式	117
7.1	不用正则表达式来查找文本模式	116	7.2.1	创建正则表达式对象	118
			7.2.2	匹配 Regex 对象	118
			7.2.3	正则表达式匹配复习	119

7.3 用正则表达式匹配更多模式·····119	7.18.1 强口令检测·····136
7.3.1 利用括号分组·····119	7.18.2 strip()的正则表达式 版本·····136
7.3.2 用管道匹配多个分组·····120	
7.3.3 用问号实现可选匹配·····121	第8章 读写文件·····137
7.3.4 用星号匹配零次或多次·····121	8.1 文件与文件路径·····137
7.3.5 用加号匹配一次或多次·····122	8.1.1 Windows 上的倒斜杠以及 OS X 和 Linux 上的 正斜杠·····138
7.3.6 用花括号匹配特定次数·····122	8.1.2 当前工作目录·····139
7.4 贪心和非贪心匹配·····123	8.1.3 绝对路径与相对路径·····139
7.5 findall()方法·····124	8.1.4 用 os.makedirs()创建新 文件夹·····140
7.6 字符分类·····124	8.1.5 os.path 模块·····140
7.7 建立自己的字符分类·····125	8.1.6 处理绝对路径和相对 路径·····141
7.8 插入字符和美元字符·····126	8.1.7 查看文件大小和文件夹 内容·····142
7.9 通配字符·····126	8.1.8 检查路径有效性·····143
7.9.1 用点-星匹配所有字符·····127	8.2 文件读写过程·····144
7.9.2 用句点字符匹配换行·····127	8.2.1 用 open()函数打开文件·····145
7.10 正则表达式符号复习·····128	8.2.2 读取文件内容·····145
7.11 不区分大小写的匹配·····128	8.2.3 写入文件·····146
7.12 用 sub()方法替换字符串·····129	8.3 用 shelve 模块保存变量·····147
7.13 管理复杂的正则表达式·····129	8.4 用 pprint.pformat()函数保存 变量·····148
7.14 组合使用 re.IGNORECASE、 re.DOTALL 和 re.VERBOSE·····130	8.5 项目：生成随机的测验试卷 文件·····149
7.15 项目：电话号码和 E-mail 地址 提取程序·····130	第1步：将测验数据保存在一个 字典中·····149
第1步：为电话号码创建一个正则 表达式·····131	第2步：创建测验文件，并打乱 问题的次序·····150
第2步：为 E-mail 地址创建一个 正则表达式·····132	第3步：创建答案选项·····151
第3步：在剪贴板文本中找到所有 匹配·····132	第4步：将内容写入测验试卷和 答案文件·····151
第4步：所有匹配连接成一个字符 串，复制到剪贴板·····133	8.6 项目：多重剪贴板·····153
第5步：运行程序·····133	
第6步：类似程序的构想·····134	
7.16 小结·····134	
7.17 习题·····134	
7.18 实践项目·····136	

第 1 步：注释和 shelf 设置 .....	153	第 1 步：弄清楚 ZIP 文件的名称 .....	168
第 2 步：用一个关键字保存剪贴板内容 .....	154	第 2 步：创建新 ZIP 文件 .....	169
第 3 步：列出关键字和加载关键字的内容 .....	154	第 3 步：遍历目录树并添加到 ZIP 文件 .....	170
8.7 小结 .....	155	第 4 步：类似程序的想法 .....	170
8.8 习题 .....	155	9.6 小结 .....	171
8.9 实践项目 .....	156	9.7 习题 .....	171
8.9.1 扩展多重剪贴板 .....	156	9.8 实践项目 .....	171
8.9.2 疯狂填词 .....	156	9.8.1 选择性拷贝 .....	171
8.9.3 正则表达式查找 .....	156	9.8.2 删除不需要的文件 .....	172
第 9 章 组织文件 .....	157	9.8.3 消除缺失的编号 .....	172
9.1 shutil 模块 .....	158	第 10 章 调试 .....	173
9.1.1 复制文件和文件夹 .....	158	10.1 抛出异常 .....	174
9.1.2 文件和文件夹的移动与改名 .....	158	10.2 取得反向跟踪的字符串 .....	175
9.1.3 永久删除文件和文件夹 .....	160	10.3 断言 .....	176
9.1.4 用 send2trash 模块安全地删除 .....	160	10.3.1 在交通灯模拟中使用断言 .....	177
9.2 遍历目录树 .....	161	10.3.2 禁用断言 .....	178
9.3 用 zipfile 模块压缩文件 .....	162	10.4 日志 .....	178
9.3.1 读取 ZIP 文件 .....	163	10.4.1 使用日志模块 .....	178
9.3.2 从 ZIP 文件中解压缩 .....	164	10.4.2 不要用 print() 调试 .....	180
9.3.3 创建和添加到 ZIP 文件 .....	164	10.4.3 日志级别 .....	180
9.4 项目：将带有美国风格日期的文件改名为欧洲风格日期 .....	165	10.4.4 禁用日志 .....	181
第 1 步：为美国风格的日期创建一个正则表达式 .....	165	10.4.5 将日志记录到文件 .....	182
第 2 步：识别文件名中的日期部分 .....	166	10.5 IDLE 的调试器 .....	182
第 3 步：构成新文件名，并对文件改名 .....	167	10.5.1 Go .....	183
第 4 步：类似程序的想法 .....	168	10.5.2 Step .....	183
9.5 项目：将一个文件夹备份到一个 ZIP 文件 .....	168	10.5.3 Over .....	183
		10.5.4 Out .....	183
		10.5.5 Quit .....	183
		10.5.6 调试一个数字相加的程序 .....	184
		10.5.7 断点 .....	185
		10.6 小结 .....	187

10.7 习题	187	查找页面	203
10.8 实践项目	188	第2步：找到所有的结果	203
第11章 从Web抓取信息	189	第3步：针对每个结果打开Web浏览器	204
11.1 项目：利用webbrowser模块的mapIt.py	190	第4步：类似程序的想法	205
第1步：弄清楚URL	190	11.7 项目：下载所有 XKCD 漫画	205
第2步：处理命令行参数	191	第1步：设计程序	206
第3步：处理剪贴板内容，加载浏览器	191	第2步：下载网页	207
第4步：类似程序的想法	192	第3步：寻找和下载漫画图像	207
11.2 用requests模块从Web下载文件	192	第4步：保存图像，找到前一张漫画	208
11.2.1 用requests.get()函数下载一个网页	193	第5步：类似程序的想法	209
11.2.2 检查错误	193	11.8 用selenium模块控制浏览器	210
11.3 将下载的文件保存到硬盘	194	11.8.1 启动selenium控制的浏览器	210
11.4 HTML	195	11.8.2 在页面中寻找元素	211
11.4.1 学习HTML的资源	195	11.8.3 点击页面	212
11.4.2 快速复习	195	11.8.4 填写并提交表单	212
11.4.3 查看网页的HTML源代码	196	11.8.5 发送特殊键	213
11.4.4 打开浏览器的开发者工具	197	11.8.6 点击浏览器按钮	213
11.4.5 使用开发者工具来寻找HTML元素	198	11.8.7 关于selenium的更多信息	214
11.5 用BeautifulSoup模块解析HTML	199	11.9 小结	214
11.5.1 从HTML创建一个BeautifulSoup对象	200	11.10 习题	214
11.5.2 用select()方法寻找元素	200	11.11 实践项目	215
11.5.3 通过元素的属性获取数据	202	11.11.1 命令行邮件程序	215
11.6 项目：“I’m Feeling Lucky” Google 查找	202	11.11.2 图像网站下载	215
第1步：获取命令行参数，并请求		11.11.3 2048	215
		11.11.4 链接验证	215
		第12章 处理Excel电子表格	217
		12.1 Excel 文档	217
		12.2 安装openpyxl模块	218
		12.3 读取Excel文档	218
		12.3.1 用openpyxl模块打开Excel	

文档.....	219	12.13.2 空行插入程序.....	241
12.3.2 从工作簿中取得工作表.....	219	12.13.3 电子表格单元格翻转	
12.3.3 从表中取得单元格.....	220	程序.....	242
12.3.4 列字母和数字之间的		12.13.4 文本文件到电子表格.....	242
转换.....	221	12.13.5 电子表格到文本文件.....	242
12.3.5 从表中取得行和列.....	222		
12.3.6 工作簿、工作表、		第 13 章 处理 PDF 和 Word 文档.....	243
单元格.....	223	13.1 PDF 文档.....	243
12.4 项目：从电子表格中读取		13.1.1 从 PDF 提取文本.....	244
数据.....	223	13.1.2 解密 PDF.....	245
第 1 步：读取电子表格数据.....	224	13.1.3 创建 PDF.....	246
第 2 步：填充数据结构.....	225	13.1.4 拷贝页面.....	246
第 3 步：将结果写入文件.....	226	13.1.5 旋转页面.....	247
第 4 步：类似程序的思想.....	227	13.1.6 叠加页面.....	248
12.5 写入 Excel 文档.....	227	13.1.7 加密 PDF.....	249
12.5.1 创建并保存 Excel 文档.....	227	13.2 项目：从多个 PDF 中合并	
12.5.2 创建和删除工作表.....	228	选择的页面.....	250
12.5.3 将值写入单元格.....	229	第 1 步：找到所有 PDF 文件.....	250
12.6 项目：更新一个电子表格.....	229	第 2 步：打开每个 PDF 文件.....	251
第 1 步：利用更新信息建立数据		第 3 步：添加每一页.....	252
结构.....	230	第 4 步：保存结果.....	252
第 2 步：检查所有行，更新不正确的		第 5 步：类似程序的想法.....	253
价格.....	231	13.3 Word 文档.....	253
第 3 步：类似程序的思想.....	231	13.3.1 读取 Word 文档.....	254
12.7 设置单元格的字体风格.....	232	13.3.2 从.docx 文件中取得完整的	
12.8 Font 对象.....	232	文本.....	254
12.9 公式.....	234	13.3.3 设置 Paragraph 和 Run 对象	
12.10 调整行和列.....	235	的样式.....	255
12.10.1 设置行高和列宽.....	235	13.3.4 创建带有非默认样式的	
12.10.2 合并和拆分单元格.....	236	Word 文档.....	257
12.10.3 冻结窗格.....	237	13.3.5 Run 属性.....	257
12.10.4 图表.....	238	13.3.6 写入 Word 文档.....	258
12.11 小结.....	240	13.3.7 添加标题.....	260
12.12 习题.....	240	13.3.8 添加换行符和换页符.....	261
12.13 实践项目.....	241	13.3.9 添加图像.....	261
12.13.1 乘法表.....	241	13.4 小结.....	262

13.5 习题	262	第 4 步：类似程序的想法	277
13.6 实践项目	263	14.6 小结	277
13.6.1 PDF 偏执狂	263	14.7 习题	277
13.6.2 定制邀请函，保存为 Word 文档	263	14.8 实践项目	277
13.6.3 暴力 PDF 口令破解程序	264	第 15 章 保持时间、计划任务和启动程序	279
第 14 章 处理 CSV 文件和 JSON 数据	265	15.1 time 模块	279
14.1 csv 模块	265	15.1.1 time.time()函数	279
14.1.1 Reader 对象	266	15.1.2 time.sleep()函数	280
14.1.2 在 for 循环中，从 Reader 对象读取数据	267	15.2 数字四舍五入	281
14.1.3 Writer 对象	268	15.3 项目：超级秒表	282
14.1.4 delimiter 和 lineterminator 关键字参数	269	第 1 步：设置程序来记录时间	282
14.2 项目：从 CSV 文件中删除表头	269	第 2 步：记录并打印单圈时间	283
第 1 步：循环遍历每个 CSV 文件	270	第 3 步：类似程序的想法	283
第 2 步：读入 CSV 文件	270	15.4 datetime 模块	284
第 3 步：写入 CSV 文件，没有第一行	271	15.4.1 timedelta 数据类型	285
第 4 步：类似程序的想法	272	15.4.2 暂停直至特定日期	286
14.3 JSON 和 API	272	15.4.3 将 datetime 对象转换为字符串	287
14.4 json 模块	273	15.4.4 将字符串转换成 datetime 对象	288
14.4.1 用 loads()函数读取 JSON	273	15.5 回顾 Python 的时间函数	288
14.4.2 用 dumps 函数写出 JSON	273	15.6 多线程	289
14.5 项目：取得当前的天气数据	274	15.6.1 向线程的目标函数传递参数	290
第 1 步：从命令行参数获取位置	274	15.6.2 并发问题	291
第 2 步：下载 JSON 数据	275	15.7 项目：多线程 XKCD 下载程序	291
第 3 步：加载 JSON 数据并打印天气	275	第 1 步：修改程序以使用函数	292
		第 2 步：创建并启动线程	293
		第 3 步：等待所有线程结束	293
		15.8 从 Python 启动其他程序	294
		15.8.1 向 Popen()传递命令行参数	295
		15.8.2 Task Scheduler、launchd 和	



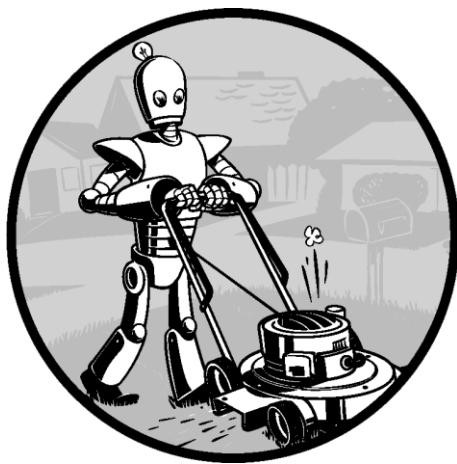
cron.....	296	邮件地址.....	313
15.8.3 用 Python 打开网站.....	296	16.4.9 从原始消息中获取正文.....	314
15.8.4 运行其他 Python 脚本.....	296	16.4.10 删除电子邮件.....	315
15.8.5 用默认的应用程序打开文件.....	297	16.4.11 从 IMAP 服务器断开.....	315
15.9 项目：简单的倒计时程序.....	298	16.5 项目：向会员发送会费提醒电子邮件.....	316
第 1 步：倒计时.....	298	第 1 步：打开 Excel 文件.....	316
第 2 步：播放声音文件.....	298	第 2 步：查找所有未付成员.....	317
第 3 步：类似程序的想法.....	299	第 3 步：发送定制的电子邮件提醒.....	318
15.10 小结.....	299	16.6 用 Twilio 发送短信.....	319
15.11 习题.....	300	16.6.1 注册 Twilio 账号.....	319
15.12 实践项目.....	300	16.6.2 发送短信.....	320
15.12.1 美化的秒表.....	300	16.7 项目：“只给我发短信”模块.....	321
15.12.2 计划的 Web 漫画下载.....	301	16.8 小结.....	322
第 16 章 发送电子邮件和短信.....	303	16.9 习题.....	323
16.1 SMTP.....	303	16.10 实践项目.....	323
16.2 发送电子邮件.....	304	16.10.1 随机分配家务活的电子邮件程序.....	323
16.2.1 连接到 SMTP 服务器.....	304	16.10.2 伞提醒程序.....	324
16.2.2 发送 SMTP 的“Hello”消息.....	305	16.10.3 自动退订.....	324
16.2.3 开始 TLS 加密.....	306	16.10.4 通过电子邮件控制你的电脑.....	324
16.2.4 登录到 SMTP 服务器.....	306	第 17 章 操作图像.....	327
16.2.5 发送电子邮件.....	306	17.1 计算机图像基础.....	327
16.2.6 从 SMTP 服务器断开.....	307	17.1.1 颜色和 RGBA 值.....	328
16.3 IMAP.....	307	17.1.2 坐标和 Box 元组.....	329
16.4 用 IMAP 获取和删除电子邮件.....	307	17.2 用 Pillow 操作图像.....	330
16.4.1 连接到 IMAP 服务器.....	308	17.2.1 处理 Image 数据类型.....	331
16.4.2 登录到 IMAP 服务器.....	309	17.2.2 裁剪图片.....	332
16.4.3 搜索电子邮件.....	309	17.2.3 复制和粘贴图像到其他图像.....	333
16.4.4 选择文件夹.....	309	17.2.4 调整图像大小.....	335
16.4.5 执行搜索.....	310	17.2.5 旋转和翻转图像.....	336
16.4.6 大小限制.....	312		
16.4.7 取邮件并标记为已读.....	312		
16.4.8 从原始消息中获取电子			

17.2.6 更改单个像素 .....	338	第 3 步：获取并打印鼠标坐标 .....	356
17.3 项目：添加徽标 .....	339	18.5 控制鼠标交互 .....	357
第 1 步：打开徽标图像 .....	340	18.5.1 点击鼠标 .....	357
第 2 步：遍历所有文件并打开 图像 .....	341	18.5.2 拖动鼠标 .....	357
第 3 步：调整图像的大小 .....	341	18.5.3 滚动鼠标 .....	359
第 4 步：添加徽标，并保存 更改 .....	342	18.6 处理屏幕 .....	360
第 5 步：类似程序的想法 .....	343	18.6.1 获取屏幕快照 .....	360
17.4 在图像上绘画 .....	344	18.6.2 分析屏幕快照 .....	360
17.4.1 绘制形状 .....	344	18.7 项目：扩展 mouseNow 程序 .....	361
17.4.2 绘制文本 .....	346	18.8 图像识别 .....	362
17.5 小结 .....	347	18.9 控制键盘 .....	363
17.6 习题 .....	348	18.9.1 通过键盘发送一个 字符串 .....	363
17.7 实践项目 .....	348	18.9.2 键名 .....	364
17.7.1 扩展和修正本章项目的 程序 .....	348	18.9.3 按下和释放键盘 .....	365
17.7.2 在硬盘上识别照片 文件夹 .....	349	18.9.4 热键组合 .....	365
17.7.3 定制的座位卡 .....	350	18.10 复习 PyAutoGUI 的函数 .....	366
第 18 章 用 GUI 自动化控制键盘和 鼠标 .....	351	18.11 项目：自动填表程序 .....	367
18.1 安装 pyautogui 模块 .....	351	第 1 步：弄清楚步骤 .....	368
18.2 走对路 .....	352	第 2 步：建立坐标 .....	368
18.2.1 通过注销关闭所有程序 .....	352	第 3 步：开始键入数据 .....	370
18.2.2 暂停和自动防故障装置 .....	352	第 4 步：处理选择列表和单选 按钮 .....	371
18.3 控制鼠标移动 .....	353	第 5 步：提交表单并等待 .....	372
18.3.1 移动鼠标 .....	354	18.12 小结 .....	372
18.3.2 获取鼠标位置 .....	354	18.13 习题 .....	373
18.4 项目：“现在鼠标在 哪里？” .....	355	18.14 实践项目 .....	373
第 1 步：导入模块 .....	355	18.14.1 看起来很忙 .....	373
第 2 步：编写退出代码和无限 循环 .....	355	18.14.2 即时通信机器人 .....	373
		18.14.3 玩游戏机器人指南 .....	374
		附录 A 安装第三方模块 .....	375
		附录 B 运行程序 .....	377
		附录 C 习题答案 .....	381

需要整本电子书，联系我QQ：2667271557

# 第一部分

Python 编程基础



需要整本电子书，联系我QQ：2667271557

# 第1章

## Python 基础



Python 编程语言有许多语法结构、标准库函数和交互式开发环境功能。好在，你可以忽略大多数内容。你只需要学习部分内容，就能编写一些方便的小程序。

但在动手之前，你必须学习一些基本编程概念。就像魔法师培训，你可能认为这些概念既深奥又啰嗦，但有了一些知识和实践，你就能像魔法师一样指挥你的计算机，完成难以置信的事情。

本章有几个例子，我们鼓励你在交互式环境中输入它们。交互式环境让你每次执行一条 Python 指令，并立即显示结果。使用交互式环境对于了解基本 Python 指令的行为是很好的，所以你在阅读时要试一下。做过的事比仅仅读过的内容，更令人印象深刻。

### 1.1 在交互式环境中输入表达式

启动 IDLE 就运行了交互式环境，这是和 Python 一起安装的。在 Windows 上，打开“开始”菜单，选择“All Programs ▶ Python 3.3”，然后选择“IDLE (Python GUI)”。在 OS X 上，选择“Applications ▶ MacPython 3.3 ▶ IDLE”。在 Ubuntu 上，打开新的终端窗口并输入 idle3。

一个窗口会出现，包含>>>提示符，这就是交互式环境。在提示符后输入 2 + 2，让 Python 做一些简单的算术。

```
>>> 2 + 2
4
```

IDLE 窗口现在应该显示下面这样的文本：

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>>
```

在 Python 中，2 + 2 称为“表达式”，它是语言中最基本的编程结构。表达式包含“值”（例如 2）和“操作符”（例如+），并且总是可以求值（也就是归约）为单个值。这意味着在 Python 代码中，所有使用表达式的地方，也可以使用一个值。

在前面的例子中，2 + 2 被求值为单个值 4。没有操作符的单个值也被认为是一个表达式，尽管它求值的结果就是它自己，像下面这样：

```
>>> 2
2
```

**错误没关系！**

如果程序包含计算机不能理解的代码，就会崩溃，这将导致 Python 显示错误信息。错误信息并不会破坏你的计算机，所以不要害怕犯错误。“崩溃”只是意味着程序意外地停止执行。

如果你希望对一条错误信息了解更多，可以在网上查找这条信息的准确文本，找到关于这个错误的更多内容。也可以查看 <http://nostarch.com/automatestuff/>，这里有常见的 Python 错误信息和含义的列表。

Python 表达式中也可以使用大量其他操作符。例如，表 1-1 列出了 Python 的所有数学操作符。

表 1-1 数学操作符，优先级从高到低

操作符	操作	例子	求值为
**	指数	2 ** 3	8
%	取模/取余数	22 % 8	6
//	整除/商数取整	22 // 8	2
/	除法	22 / 8	2.75
*	乘法	3 * 5	15
-	减法	5 - 2	3
+	加法	2 + 2	4

数学操作符的操作顺序（也称为“优先级”）与数学中类似。\*\*操作符首先求值，接下来是\*、/、//和%操作符，从左到右。+和-操作符最后求值，也是从左到右。

如果需要，可以用括号来改变通常的优先级。在交互式环境中输入下列表达式：

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

在每个例子中，作为程序员，你必须输入表达式，但 Python 完成较难的工作，将它求值为单个值。Python 将继续求值表达式的各个部分，直到它成为单个值，如图 1-1 所示。

```
(5 - 1) * ((7 + 1) / (3 - 1))
  ↓
4 * ((7 + 1) / (3 - 1))
  ↓
4 * ( 8 ) / (3 - 1)
  ↓
4 * ( 8 ) / ( 2 )
  ↓
4 * 4.0
  ↓
16.0
```

图 1-1 表达式求值将它归约为单个值

将操作符和值放在一起构成表达式的这些规则，是 Python 编程语言的基本部分，就像帮助我们沟通的语法规则一样。下面是例子：

This is a grammatically correct English sentence.

This grammatically is sentence not English correct a.

第二行很难解释，因为它不符合英语的规则。类似地，如果你输入错误的 Python 指令，Python 也不能理解，就会显示出错误信息，像下面这样：

```
>>> 5 +
File "<stdin>", line 1
  5 +
    ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
File "<stdin>", line 1
  42 + 5 + * 2
```

`SyntaxError: invalid syntax`

你总是可以在交互式环境中输入一条指令，检查它是否能工作。不要担心会弄坏计算机：最坏的情况就是 Python 显示出错信息。专业的软件开发者在编写代码时，常常会遇到错误信息。

1.2 整型、浮点型和字符串数据类型

记住，表达式是值和操作符的组合，它们可以通过求值成为单个值。“数据类型”是一类值，每个值都只属于一种数据类型。表 1-2 列出了 Python 中最常见的数据类型。例如，值-2 和 30 属于“整型”值。整型（或 int）数据类型表明值是整数。带有小数点的数，如 3.14，称为“浮点型”（或 float）。请注意，尽管 42 是一个整型，但 42.0 是一个浮点型。

表 1-2 常见数据类型

数据类型	例子
整型	-2, -1, 0, 1, 2, 3, 4, 5
浮点型	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25
字符串	'a', 'aa', 'aaa', 'Hello!', '11 cats'

Python 程序也可以有文本值，称为“字符串”，或 str（发音为“stirs”）。总是用单引号（'）包围住字符串（例如'Hello'或'Goodbye cruel world!'），这样 Python 就知道字符串的开始和结束。甚至可以有没有字符的字符串，称为“空字符串”。第 4 章更详细地解释了字符串。

如果你看到错误信息 SyntaxError: EOL while scanning string literal，可能是忘记了字符串末尾的单引号，如下面的例子所示：

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

1.3 字符串连接和复制

根据操作符之后的值的数据类型，操作符的含义可能会改变。例如，在操作两个整型或浮点型值时，+是相加操作符。但是，在用于两个字符串时，它将字符串连接起来，成为“字符串连接”操作符。在交互式环境中输入以下内容：

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

该表达式求值为一个新字符串，包含了两个字符串的文本。但是，如果你对一个字符串和一个整型值使用加操作符，Python 就不知道如何处理，它将显示一条错

误信息。

---

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
```

---

错误信息 Can't convert 'int' object to str implicitly 表示 Python 认为，你试图将一个整数连接到字符串'Alice'。代码必须显式地将整数转换为字符串，因为 Python 不能自动完成转换。（1.6 节“程序剖析”在讨论函数时，将解释数据类型转换。）

在用于两个整型或浮点型值时，\*操作符表示乘法。但\*操作符用于一个字符串值和一个整型值时，它变成了“字符串复制”操作符。在交互式环境中输入一个字符串乘一个数字，看看效果。

---

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

---

该表达式求值为一个字符串，它将原来的字符串重复若干次，次数就是整型的值。字符串复制是一个有用的技巧，但不像字符串连接那样常用。

\*操作符只能用于两个数字（作为乘法），或一个字符串和一个整型（作为字符串复制操作符）。否则，Python 将显示错误信息。

---

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

---

Python 不理解这些表达式是有道理的：你不能把两个单词相乘，也很难将一个任意字符串复制小数次。

## 1.4 在变量中保存值

“变量”就像计算机内存中的一个盒子，其中可以存放一个值。如果你的程序稍后将用到一个已求值的表达式的结果，就可以将它保存在一个变量中。

### 1.4.1 赋值语句

用“赋值语句”将值保存在变量中。赋值语句包含一个变量名、一个等号（称为赋值操作符），以及要存储的值。如果输入赋值语句 `spam = 42`，那么名为 `spam` 的变量将保存一个整型值 42。

可以将变量看成一个带标签的盒子，值放在其中，如图 1-2 所示。



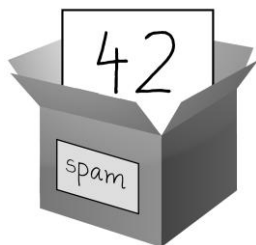


图 1-2 `spam = 42` 就像是告诉程序“变量 `spam` 现在有整数 42 放在里面”

例如，在交互式环境中输入以下内容：

```
❶ >>> spam = 40
>>> spam
40
>>> eggs = 2
❷ >>> spam + eggs
42
>>> spam + eggs + spam
82
❸ >>> spam = spam + 2
>>> spam
42
```

第一次存入一个值，变量就被“初始化”（或创建）❶。此后，可以在表达式中使用它，以及其他变量和值❷。如果变量被赋了一个新值，老值就被忘记了❸。这就是为什么在例子结束时，`spam` 求值为 42，而不是 40。这称为“覆写”该变量。在交互式环境中输入以下代码，尝试覆写一个字符串：

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

就像图 1-3 中的盒子，这个例子中的 `spam` 变量保存了 'Hello'，直到你用 'Goodbye' 替代它。

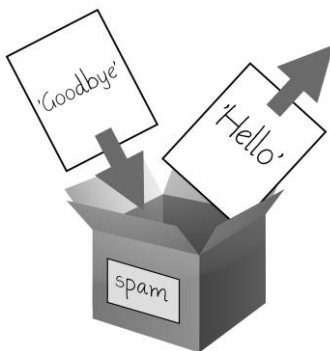


图 1-3 如果一个新值赋给变量，老值就被遗忘了

### 1.4.2 变量名

表 1-3 中有一些合法变量名的例子。你可以给变量取任何名字，只要它遵守以下 3 条规则：

1. 只能是一个词。
2. 只能包含字母、数字和下划线。
3. 不能以数字开头。

表 1-3 有效和无效的变量名

有效的变量名	无效的变量名
balance	current-balance（不允许中划线）
currentBalance	current balanc（不允许空格）
current_balance	4account（不允许数字开头）
_spam	42（不允许数字开头）
SPAM	total_\$um（不允许\$这样的特殊字符）
account4	'hello'（不允许'这样的特殊字符）

变量名是区分大小写的。这意味着，spam、 SPAM、Spam 和 sPaM 是 4 个不同的变量。变量用小写字母开头是 Python 的惯例。

本书的变量名使用了驼峰形式，没有用下划线。也就是说，变量名用 lookLikeThis，而不是 looking\_like\_this。一些有经验的程序员可能会指出，官方的 Python 代码风格 PEP 8，即应该使用下划线。我喜欢驼峰式，这没有错，并认为 PEP 8 本身“愚蠢的一致性”是头脑狭隘人士的心魔”：

“一致地满足风格指南是重要的。但最重要的是，知道何时要不一致，因为有时候风格指南就是不适用。如果有怀疑，请相信自己的最佳判断。”

好的变量名描述了它包含的数据。设想你搬到一间新屋子，搬家纸箱上标的都是“东西”。你永远找不到任何东西！本书的例子和许多 Python 的文档，使用 spam、eggs 和 bacon 等变量名作为一般名称（受到 Monty Python 的“Spam”短剧的影响），但在你的程序中，具有描述性的名字有助于提高代码可读性。

## 1.5 第一个程序

虽然交互式环境对于一次运行一条 Python 指令很好，但要编写完整的 Python 程序，就需要在文件编辑器中输入指令。“文件编辑器”类似于 Notepad 或 TextMate 这样的文本编辑器，它有一些针对输入源代码的特殊功能。要在 IDLE 中打开文件编辑器，请选择 File ▶ New Window。

出现的窗口中应该包含一个光标，等待你输入，但它与交互式环境不同。在交

交互式环境中，按下回车，就会执行 Python 指令。文件编辑器允许输入许多指令，保存为文件，并运行该程序。下面是区别这两者的方法：

- 交互式环境窗口总是有>>>提示符。
- 文件编辑器窗口没有>>>提示符。

现在是创建第一个程序的时候了！在文件编辑器窗口打开后，输入以下内容：

---

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

❷ print('Hello world!')
   print('What is your name?') # ask for their name
❸ myName = input()
❹ print('It is good to meet you, ' + myName)
❺ print('The length of your name is:')
   print(len(myName))
❻ print('What is your age?') # ask for their age
   myAge = input()
   print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

---

在输入完源代码后保存它，这样就不必在每次启动 IDLE 时重新输入。从文件编辑器窗口顶部的菜单，选择 File ▶ Save As。在“Save As”窗口中，在输入框输入 hello.py，然后点击“Save”。

在输入程序时，应该过一段时间就保存你的程序。这样，如果计算机崩溃，或者不小心退出了 IDLE，也不会丢失代码。作为快捷键，可以在 Windows 和 Linux 上按 Ctrl-S，在 OS X 上按⌘-S，来保存文件。

在保存文件后，让我们来运行程序。选择 Run ▶ Run Module，或按下 F5 键。程序将在交互式环境窗口中运行，该窗口是首次启动 IDLE 时出现的。记住，必须在文件编辑器窗口中按 F5，而不是在交互式环境窗口中。在程序要求输入时，输入你的名字。在交互式环境中，程序输出应该看起来像这样：

---

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello world!
What is your name?
A1
It is good to meet you, A1
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

---

如果没有更多代码行要执行，Python 程序就会“中止”。也就是说，它停止运行。（也可以说 Python 程序“退出”了。）

可以通过点击窗口上部的 X，关闭文件编辑器。要重新加载一个保存了的程序，就在菜单中选择 File ▶ Open。现在请这样做，在出现的窗口中选择 hello.py，并点

击“Open”按钮。前面保存的程序 `hello.py` 应该在文件编辑器窗口中打开。

## 1.6 程序剖析

新程序在文件编辑器中打开后，让我们快速看一看它用到的 Python 指令，逐一查看每行代码。

### 1.6.1 注释

下面这行称为“注释”。

---

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

---

Python 会忽略注释，你可以用它们来写程序注解，或提醒自己代码试图完成的事。这一行中，`#`标志之后的所有文本都是注释。

有时候，程序员在测试代码时，会在一行代码前面加上`#`，临时删除它。这称为“注释掉”代码。在你想搞清楚为什么程序不工作时，这样做可能有用。稍后，如果你准备还原这一行代码，可以去掉`#`。

Python 也会忽略注释之后的空行。在程序中，想加入空行时就可以加入。这会让你的代码更容易阅读，就像书中的段落一样。

### 1.6.2 `print()`函数

`print()`函数将括号内的字符串显示在屏幕上。

---

```
❷ print('Hello world!')  
print('What is your name?') # ask for their name
```

---

代码行 `print('Hello world!')`表示“打印出字符串'Hello world!'的文本”。Python 执行到这行时，你告诉 Python 调用 `print()`函数，并将字符串“传递”给函数。传递给函数的值称为“参数”。请注意，引号没有打印在屏幕上。它们只是表示字符串的起止，不是字符串的一部分。

也可以用这个函数在屏幕上打印出空行，只要调用 `print()`就可以了，括号内没有任何东西。

在写函数名时，末尾的左右括号表明它是一个函数的名字。这就是为什么在本书中你会看到 `print()`，而不是 `print`。第 2 章更详细地探讨了函数。

### 1.6.3 `input()`函数

函数等待用户在键盘上输入一些文本，并按下回车键。

---

```
❸ myName = input()
```

---

这个函数求值为一个字符串，即用户输入的文本。前面的代码行将这个字符串

赋给变量 `myName`。

你可以认为 `input()` 函数调用是一个表达式，它求值为用户输入的任何字符串。如果用户输入 'AI'，那么该表达式就求值为 `myName = 'AI'`。

## 1.6.4 打印用户的名字

接下来的 `print()` 调用，在括号间包含表达式 `'It is good to meet you, ' + myName`。

---

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

---

要记住，表达式总是可以求值为一个值。如果 'AI' 是上一行代码保存在 `myName` 中的值，那么这个表达式就求值为 `'It is good to meet you, AI'`。这个字符串传给 `print()`，它将输出到屏幕上。

## 1.6.5 `len()` 函数

你可以向 `len()` 函数传递一个字符串（或包含字符串的变量），然后该函数求值为一个整型值，即字符串中字符的个数。

---

```
❷ print('The length of your name is:')  
print(len(myName))
```

---

在交互式环境中输入以下内容试一试：

---

```
>>> len('hello')  
5  
>>> len('My very energetic monster just scarfed nachos.')  
46  
>>> len('')  
0
```

---

就像这些例子，`len(myName)` 求值为一个整数。然后它被传递给 `print()`，在屏幕上显示。请注意，`print()` 允许传入一个整型值或字符串。但如果在交互式环境中输入以下内容，就会报错：

---

```
>>> print('I am ' + 29 + ' years old.')  
Traceback (most recent call last):  
  File "<pyshell#6>", line 1, in <module>  
    print('I am ' + 29 + ' years old.')  
TypeError: Can't convert 'int' object to str implicitly
```

---

导致错误的原因不是 `print()` 函数，而是你试图传递给 `print()` 的表达式。如果在交互式环境中单独输入这个表达式，也会得到同样的错误。

---

```
>>> 'I am ' + 29 + ' years old.'  
Traceback (most recent call last):  
  File "<pyshell#7>", line 1, in <module>  
    'I am ' + 29 + ' years old.'  
TypeError: Can't convert 'int' object to str implicitly
```

---

报错是因为，只能用 `+` 操作符加两个整数，或连接两个字符串。不能让一个整

数和一个字符串相加，因为这不符合 Python 的语法。可以使用字符串版本的整数，修复这个错误。这在下一节中解释。

## 1.6.6 str()、int()和 float()函数

如果想要连接一个整数（如 29）和一个字符串，再传递给 print()，就需要获得值'29'。它是 29 的字符串形式。str()函数可以传入一个整型值，并求值为它的字符串形式，像下面这样：

---

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

---

因为 str(29)求值为'29'，所以表达式'I am ' + str(29) + ' years old.'求值为'I am ' + '29' + ' years old.'，它又求值为'I am 29 years old.'。这就是传递给 print()函数的值。

str()、int()和 float()函数将分别求值为传入值的字符串、整数和浮点数形式。请尝试用这些函数在交互式环境中转换一些值，看看会发生什么。

---

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

---

前面的例子调用了 str()、int()和 float()函数，向它们传入其他数据类型的值，得到了字符串、整型或浮点型的值。

如果想要将一个整数或浮点数与一个字符串连接，str()函数就很方便。如果你有一些字符串值，希望将它们用于数学运算，int()函数也很有用。例如，input()函数总是返回一个字符串，即使用户输入的是一个数字。在交互式环境中输入 spam = input()，在它等待文本时输入 101。

---

```
>>> spam = input()
101
>>> spam
'101'
```

---

保存在 spam 中的值不是整数 101，而是字符串'101'。如果想要用 spam 中的值进行数学运算，那就用 int()函数取得 spam 的整数形式，然后将这个新值存在 spam 中。

```
>>> spam = int(spam)
>>> spam
101
```

现在你应该能将 `spam` 变量作为整数，而不是字符串使用。

```
>>> spam * 10 / 5
202.0
```

请注意，如果你将一个不能求值为整数的值传递给 `int()`，Python 将显示出错信息。

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

如果需要对浮点数进行取整运算，也可以用 `int()` 函数。

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

在你的程序中，最后 3 行使用了函数 `int()` 和 `str()`，取得适当数据类型的值。

```
⑥ print('What is your age?') # ask for their age
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

`myAge` 变量包含了 `input()` 函数返回的值。因为 `input()` 函数总是返回一个字符串（即使用户输入的是数字），所以你可以使用 `int(myAge)` 返回字符串的整型值。这个整型值随后在表达式 `int(myAge) + 1` 中与 1 相加。

相加的结果传递给 `str()` 函数：`str(int(myAge) + 1)`。然后，返回的字符串与字符串 `'You will be '` 和 `' in a year.'` 连接，求值为一个更长的字符串。这个更长的字符串最终传递给 `print()`，在屏幕上显示。

假定用户输入字符串 `'4'`，保存在 `myAge` 中。字符串 `'4'` 被转换为一个整型，所以你可以对它加 1。结果是 5。`str()` 函数将这个结果转化为字符串，这样你就可以将它与第二个字符串 `'in a year.'` 连接，创建最终的消息。这些求值步骤如图 1-4 所示。

## 文本和数字相等判断

虽然数字的字符串值被认为与整型值和浮点型值完全不同，但整型值可以与浮点值相等。

```
>>> 42 == '42'
False
>>> 42 == 42.0
```

```
True
>>> 42.0 == 0042.000
True
```

Python 进行这种区分，因为字符串是文本，而整型值和浮点型都是数字。

```
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
print('You will be ' + str(int('4') + 1) + ' in a year.')
print('You will be ' + str(4 + 1) + ' in a year.')
print('You will be ' + str(5) + ' in a year.')
print('You will be ' + '5' + ' in a year.')
print('You will be 5' + ' in a year.')
print('You will be 5 in a year.')
```

图 1-4 如果 4 保存在 myAge 中，求值的步骤

## 1.7 小结

你可以用一个计算器来计算表达式，或在文本处理器中输入字符串连接。甚至可以通过复制粘贴文本，很容易地实现字符串复制。但是表达式以及组成它们的值（操作符、变量和函数调用），才是构成程序的基本构建块。一旦你知道如何处理这些元素，就能够用 Python 操作大量的数据。

最好是记住本章中介绍的不同类型的操作符（+、-、\*、/、//、%和\*\*是数学操作符，+和\*是字符串操作符），以及 3 种数据类型（整型、浮点型和字符串）。

我们还介绍了几个不同的函数。print()和 input()函数处理简单的文本输出（到屏幕）和输入（通过键盘）。len()函数接受一个字符串，并求值为该字符串中字符的数目。

在下一章中，你将学习如何告诉 Python 根据它拥有的值，明智地决定什么代码要运行，什么代码要跳过，什么代码要重复。这称为“控制流”，它让你编写程序来做出明智的决定。

## 1.8 习题

1. 下面哪些是操作符，哪些是值？

```
*
'hello'
-88.8
-
/
```



+  
5

---

2. 下面哪个是变量，哪个是字符串？

---

spam  
'spam'

---

3. 说出 3 种数据类型。
  4. 表达式由什么构成？所有表达式都做什么事？
  5. 本章介绍了赋值语句，如 `spam = 10`。表达式和语句有什么区别？
  6. 下列语句运行后，变量 `bacon` 的值是什么？
- 

```
bacon = 20  
bacon + 1
```

---

7. 下面两个表达式求值的结果是什么？

---

```
'spam' + 'spamspam'  
'spam' * 3
```

---

8. 为什么 `eggs` 是有效的变量名，而 `100` 是无效的？
  9. 哪 3 个函数能分别取得一个值的整型、浮点型或字符串版本？
  10. 为什么这个表达式会导致错误？如何修复？
- 

```
'I have eaten ' + 99 + ' burritos.'
```

---

附加题：在线查找 `len()` 函数的 Python 文档。它在一个标题为 “Built-in Functions” 的网页上。扫一眼 Python 的其他函数的列表，查看 `round()` 函数的功能，在交互式环境中使用它。

# 第2章

## 控 制 流



你已经知道了单条指令的基本知识。程序就是一系列指令。但编程真正的力量不仅在于运行（或“执行”）一条接一条的指令，就像周末的任务清单那样。根据表达式求值的结果，程序可以决定跳过指令，重复指令，或从几条指令中选择一条运行。实际上，你几乎永远不希望程序从第一行代码开始，简单地执行每行代码，直到最后一行。“控制流语句”可以决定在什么条件下执行哪些 Python 语句。

这些控制流语句直接对应于流程图中的符号，所以在本章中，我将提供示例代码的流程图。图 2-1 展示了一张流程图，内容是如果下雨怎么办。按照箭头构成的路径，从开始到结束。

在流程图中，通常有不只一种方法从开始走到结束。计算机程序中的代码行也是这样。流程图用菱形表示这些分支节点，其他步骤用矩形表示。开始和结束步骤用带圆角的矩形表示。

但在学习流程控制语句之前，首先要学习如何表示这些 yes 和 no 选项。同时你也需要理解，如何将这些分支节点写成 Python 代码。要做到这一点，让我们先看看布尔值、比较操作符和布尔操作符。

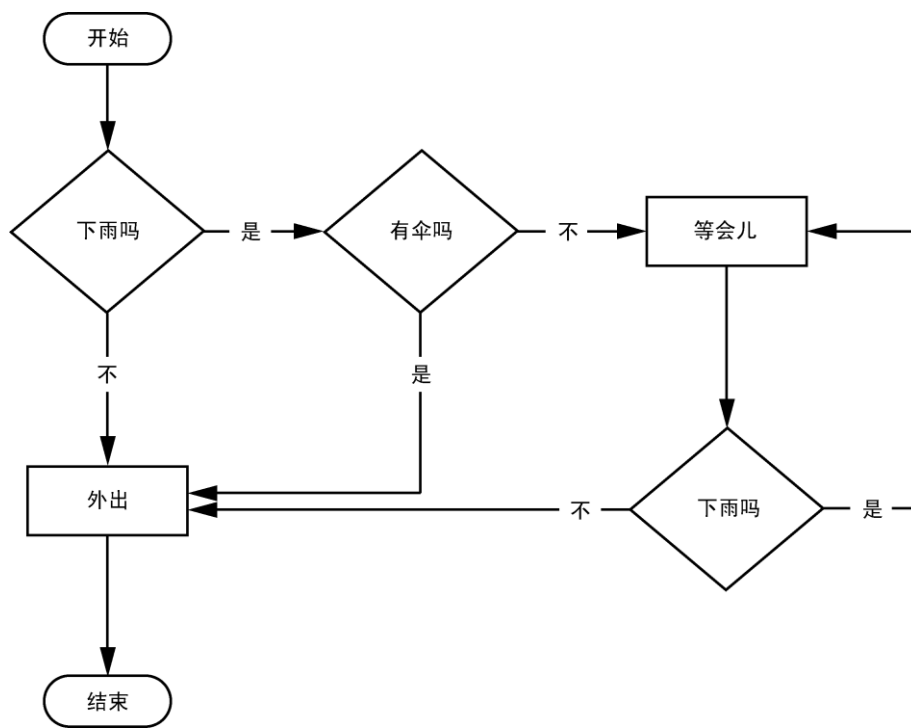


图 2-1 一张流程图，告诉你如果下雨要做什么

## 2.1 布尔值

虽然整型、浮点型和字符串数据类型有无数种可能的值，但“布尔”数据类型只有两种值：True 和 False。Boolean（布尔）的首字母大写，因为这个数据类型是根据数学家 George Boole 命名的。在作为 Python 代码输入时，布尔值 True 和 False 不像字符串，两边没有引号，它们总是以大写字母 T 或 F 开头，后面的字母小写。在交互式环境中输入下面内容，其中有些指令是故意弄错的，它们将导致出错信息。

```
❶ >>> spam = True
>>> spam
True
❷ >>> true
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    true
NameError: name 'true' is not defined
❸ >>> True = 2 + 2
SyntaxError: assignment to keyword
```

像其他值一样，布尔值也用在表达式中，并且可以保存在变量中❶。如果大小写不正确❷，或者试图使用 True 和 False 作为变量名❸，Python 就会给出错误信息。

## 2.2 比较操作符

“比较操作符”比较两个值，求值为一个布尔值。表 2-1 列出了比较操作符。

表 2-1 比较操作符

操作符	含义
==	等于
!=	不等于
<	小于
>	大于
<=	小于等于
>=	大于等于

这些操作符根据给它们提供的值，求值为 `True` 或 `False`。现在让我们尝试一些操作符，从 `==` 和 `!=` 开始。

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

如果两边的值一样，`==`（等于）求值为 `True`。如果两边的值不同，`!=`（不等于）求值为 `True`。`==` 和 `!=` 操作符实际上可以用于所有数据类型的值。

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
❶ >>> 42 == '42'
False
```

请注意，整型或浮点型的值永远不会与字符串相等。表达式 `42 == '42'❶` 求值为 `False` 是因为，Python 认为整数 `42` 与字符串 `'42'` 不同。

另一方面，`<`、`>`、`<=` 和 `>=` 操作符仅用于整型和浮点型值。

```
>>> 42 < 100
True
>>> 42 > 100
False
```

```
>>> 42 < 42
False
>>> eggCount = 42
② >>> eggCount <= 42
True
>>> myAge = 29
③ >>> myAge >= 10
True
```

### 操作符的区别

你可能已经注意到，`==`操作符（等于）有两个等号，而`=`操作符（赋值）只有一个等号。这两个操作符很容易混淆。只要记住：

- `==`操作符（等于）问两个值是否彼此相同。
- `=`操作符（赋值）将右边的值放到左边的变量中。

为了记住谁是谁，请注意`==`操作符（等于）包含两个字符，就像`!=`操作符（不等于）包含两个字符一样。

你会经常用比较操作符比较一个变量和另外某个值。就像在例子 `eggCount <= 42`①和 `myAge >= 10`②中一样（毕竟，除了在代码中输入`'dog' != 'cat'`以外，你本来也可以直接输入 `True`）。稍后，在学习控制流语句时，你会看到更多的例子。

## 2.3 布尔操作符

3 个布尔操作符（`and`、`or` 和 `not`）用于比较布尔值。像比较操作符一样，它们将这些表达式求值为一个布尔值。让我们仔细看看这些操作符，从 `and` 操作符开始。

### 2.3.1 二元布尔操作符

`and` 和 `or` 操作符总是接受两个布尔值（或表达式），所以它们被认为是“二元”操作符。如果两个布尔值都为 `True`，`and` 操作符就将表达式求值为 `True`，否则求值为 `False`。在交互式环境中输入某个使用 `and` 的表达式，看看效果。

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

“真值表”显示了布尔操作符的所有可能结果。表 2-2 是操作符 `and` 的真值表。

表 2-2 `and` 操作符的真值表

表达式	求值为
True and True	True
True and False	False
False and True	False
False and False	False

另一方面，只要有一个布尔值为真，or 操作符就将表达式求值为 True。如果都是 False，所求值为 False。

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

可以在 or 操作符的真值表中看到每一种可能的结果，如表 2-3 所示。

表 2-3 or 操作符的真值表

表达式	求值为
True or True	True
True or False	True
False or True	True
False or False	False

### 2.3.2 not 操作符

和 and 和 or 不同，not 操作符只作用于一个布尔值（或表达式）。not 操作符求值为相反的布尔值。

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

就像在说话和写作中使用双重否定，你可以嵌套 not 操作符❶，虽然在真正的程序中并不经常这样做。表 2-4 展示了 not 的真值表。

表 2-4 not 操作符的真值表

表达式	求值为
not True	False
not False	True

## 2.4 混合布尔和比较操作符

既然比较操作符求值为布尔值，就可以和布尔操作符一起，在表达式中使用。

回忆一下，and、or 和 not 操作符称为布尔操作符是因为，它们总是操作于布尔值。虽然像  $4 < 5$  这样的表达式不是布尔值，但可以求值为布尔值。在交互式环境中，尝试输入一些使用比较操作符的布尔表达式。

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
```

```
>>> (1 == 2) or (2 == 2)
True
```

计算机将先求值左边的表达式，然后再求值右边的表达式。知道两个布尔值后，它又将整个表达式再求值为一个布尔值。你可以认为计算机求值 $(4 < 5)$ 和 $(5 < 6)$ 的过程，如图 2-2 所示。

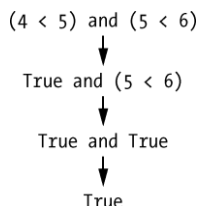


图 2-2  $(4 < 5)$ 和  $(5 < 6)$  求值为 True 的过程

也可以在一个表达式中使用多个布尔操作符，与比较操作符一起使用。

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

和算术操作符一样，布尔操作符也有操作顺序。在所有算术和比较操作符求值后，Python 先求值 not 操作符，然后是 and 操作符，然后是 or 操作符。

## 2.5 控制流的元素

控制流语句的开始部分通常是“条件”，接下来是一个代码块，称为“子句”。在开始学习具体的 Python 控制流语句之前，我将介绍条件和代码块。

### 2.5.1 条件

你前面看到的布尔表达式可以看成是条件，它和表达式是一回事。“条件”只是在控制流语句的上下文中更具体的名称。条件总是求值为一个布尔值，True 或 False。控制流语句根据条件是 True 还是 False，来决定做什么。几乎所有的控制流语句都使用条件。

### 2.5.2 代码块

一些代码行可以作为一组，放在“代码块”中。可以根据代码行的缩进，知道代码块的开始和结束。代码块有 3 条规则。

1. 缩进增加时，代码块开始。
2. 代码块可以包含其他代码块。
3. 缩进减少为零，或减少为外面包围代码块的缩进，代码块就结束了。

看一些有缩进的代码，更容易理解代码块。所以让我们在一小段游戏程序中，

寻找代码块，如下所示：

---

```
if name == 'Mary':  
❶    print('Hello Mary')  
if password == 'swordfish':  
❷    print('Access granted.')  
else:  
❸    print('Wrong password.')
```

---

第一个代码块❶开始于代码行 `print('Hello Mary')`，并且包含后面所有的行。在这个代码块中有另一个代码块❷，它只有一行代码：`print('Access Granted.')`。第三个代码块❸也只有一行：`print('Wrong password.')`。

## 2.6 程序执行

在第 1 章的 `hello.py` 程序中，Python 开始执行程序顶部的指令，然后一条接一条往下执行。“程序执行”（或简称“执行”）这一术语是指当前被执行的指令。如果将源代码打印在纸上，在它执行时用手指指着每一行代码，你可以认为手指就是程序执行。

但是，并非所有的程序都是从上至下简单地执行。如果用手指追踪一个带有控制流语句的程序，可能会发现手指会根据条件跳过源代码，有可能跳过整个子句。

## 2.7 控制流语句

现在，让我们来看最重要的控制流部分：语句本身。语句代表了在图 2-1 的流程图中看到的菱形，它们是程序将做出的实际决定。

### 2.7.1 if 语句

最常见的控制流语句是 `if` 语句。`if` 语句的子句（也就是紧跟 `if` 语句的语句块），将在语句的条件为 `True` 时执行。如果条件为 `False`，子句将跳过。

在英文中，`if` 语句念起来可能是：“如果条件为真，执行子句中的代码。”在 Python 中，`if` 语句包含以下部分：

- `if` 关键字；
- 条件（即求值为 `True` 或 `False` 的表达式）；
- 冒号；
- 在下一行开始，缩进的代码块（称为 `if` 子句）。

例如，假定有一些代码，检查某人的名字是否为 `Alice`（假设此前曾为 `name` 赋值）。



---

```
if name == 'Alice':  
    print('Hi, Alice.')
```

---

所有控制流语句都以冒号结尾，后面跟着一个新的代码块（子句）。语句的 `if` 子句是代码块，包含 `print('Hi, Alice.')`。图 2-3 展示了这段代码的流程图。

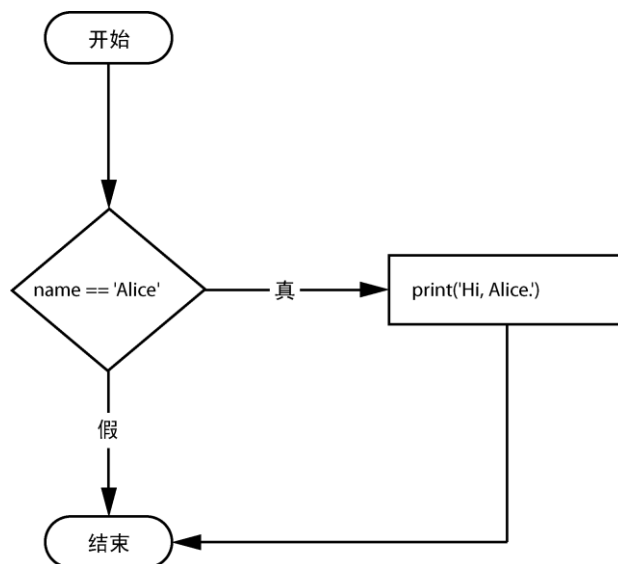


图 2-3 if 语句的流程图

## 2.7.2 else 语句

`if` 子句后面有时候也可以跟着 `else` 语句。只有 `if` 语句的条件为 `False` 时，`else` 子句才会执行。在英语中，`else` 语句读起来可能是：“如果条件为真，执行这段代码。否则，执行那段代码”。`else` 语句不包含条件，在代码中，`else` 语句中包含下面部分：

- `else` 关键字；
- 冒号；
- 在下一行开始，缩进的代码块（称为 `else` 子句）。

回到 Alice 的例子，我们来看看使用 `else` 语句的一些代码，在名字不是 Alice 时，提供不一样的问候。

---

```
if name == 'Alice':  
    print('Hi, Alice.')  
else:  
    print('Hello, stranger.')
```

---

图 2-4 展示了这段代码的流程图。

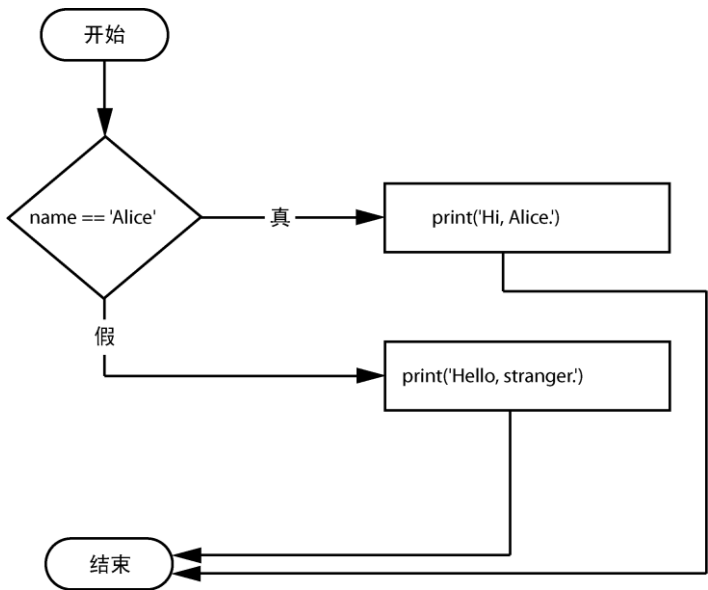


图 2-4 else 语句的流程图

2.7.3 elif 语句

虽然只有 if 或 else 子句会被执行，但有时候可能你希望，“许多”可能的子句中有一个被执行。elif 语句是“否则如果”，总是跟在 if 或另一条 elif 语句后面。它提供了另一个条件，仅在前面的条件为 False 时才检查该条件。在代码中，elif 语句总是包含以下部分：

- elif 关键字；
- 条件（即求值为 True 或 False 的表达式）；
- 冒号；
- 在下一行开始，缩进的代码块（称为 elif 子句）。

让我们在名字检查程序中添加 elif，看看这个语句的效果。

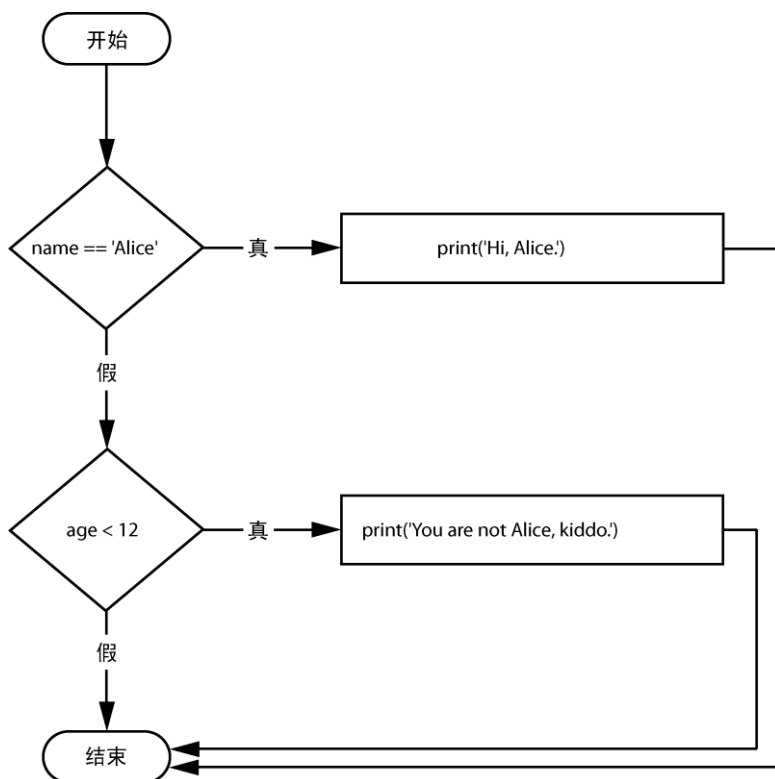
```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

这一次，检查此人的年龄。如果比 12 岁小，就告诉他一些不同的东西。可以在图 2-5 中看到这段代码的流程图。

如果 age < 12 为 True 并且 name == 'Alice'为 False，elif 子句就会执行。但是，如果两个条件都为 False，那么两个子句都会跳过。“不能”保证至少有一个子句会被执行。如果有一系列的 elif 语句，仅有一条或零条子句会被执行。一旦一个语句的条件为 True，剩下的 elif 子句会自动跳过。例如，打开一个新的文件编辑器窗口，

输入以下代码，保存为 vampire.py。

```
if name == 'Alice':  
    print('Hi, Alice.')  
elif age < 12:  
    print('You are not Alice, kiddo.')  
elif age > 2000:  
    print('Unlike you, Alice is not an undead, immortal vampire.')  
elif age > 100:  
    print('You are not Alice, grannie.')
```



这里，我添加了另外两条 elif 语句，让名字检查程序根据 age 的不同答案而发出问候。图 2-6 展示了这段代码的流程图。

但是，elif 语句的次序确实重要。让我们重新排序，引入一个缺陷。回忆一下，一旦找到一个 True 条件，剩余的子句就会自动跳过。所以如果交换 vampire.py 中的一些子句，就会遇到问题。像下面这样改变代码，将它保存为 vampire2.py。

```
if name == 'Alice':  
    print('Hi, Alice.')  
elif age < 12:  
    print('You are not Alice, kiddo.')
```