

PYTHON HOW TO PROGRAM

Introducing

XML



- CONTROL STRUCTURES
- FUNCTIONS
- LISTS AND TUPLES
- DICTIONARIES
- EXCEPTIONS
- MODULES
- XHTML™/CSS™
- CGI
- CLASSES
- CLASS ATTRIBUTES
- CLASS CUSTOMIZATION
- INHERITANCE
- METHOD OVERRIDING
- GUI/TkINTER
- PYTHON MEGA WIDGETS
- STRING MANIPULATION
- REGULAR EXPRESSIONS
- FILE PROCESSING
- SERIALIZATION
- XML PROCESSING
- DATABASES/DB-API/SQL
- PROCESS MANAGEMENT
- INTERPROCESS COMMUNICATION
- MULTITHREADING
- NETWORKING/SOCKETS
- SECURITY
- RESTRICTED EXECUTION
- DATA STRUCTURES
- MULTIMEDIA
- PyOPENGL
- PYTHON SERVER PAGES

DEITEL
DEITEL
LIPERI
WIEDERMANN



DEITEL 编程金典

PYTHON 编程金典

内容简介

本书由全球著名的程序语言培训专家精心编著，解释了如何将Python用作常规用途，编写多层、客户端/服务器结构、数据库密集型的、基于Internet和Web的应用程序。书中采用作者独创的“活代码”教学方式，层层揭示了Python这一程序设计语言的强大功能，并通过穿插在全书各处的屏幕输出和编程技巧与提示，帮助读者搭建良好的知识结构、养成良好的编程习惯、避免常见的编程错误以及写出高效、可靠的应用程序。

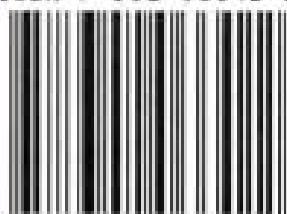
特色主题

- ◆ Python服务器页/CGI
- ◆ 联网/套接字
- ◆ GUI/Tkinter/Python Mega Widgets
- ◆ PyOpenGL/多媒体
- ◆ 数据库/DB-API/SQL
- ◆ 文件处理/序列化
- ◆ 模块/类/类属性
- ◆ 类自定义/方法覆盖
- ◆ 控制结构/函数/继承
- ◆ 字符串处理/正则表达式
- ◆ 列表/元组/字典/数据结构
- ◆ 进程管理/多线程处理
- ◆ 进程间通信
- ◆ 异常/XML处理
- ◆ 安全性/限制执行

读者对象

本书沿袭Deitel公司“How To Program”系列的写作风格，适合对Python感兴趣的初中级程序员阅读和参考。

ISBN 7-302-06642-6



9 787302 066422 >

定价：88.00元



责任编辑：文开棋
封面设计：立日新设计公司
读者信箱：Book@21bj.com
信息网站：<http://www.ePress.cn>
<http://www.34.cn>

DEITEL 编程金典

Python 编程金典

[美] H. M. Deitel, P. J. Deitel 著
J. P. Liperi, B. A. Wiedermann 译
周 靖 译

清华大学出版社
北京

内 容 简 介

本书由全球著名的编程培训专家 H. M. Deitel 博士领头编写，解释了如何将 Python 用做常规用途，编写基于 Internet 和 Web 的、数据密集型的、多层的客户端/服务器系统。书中采用作者独创的“活代码”教学方法，揭示了 Python 这一语言的强大功能。与此同时，本书还提供了大量的示例代码和编程技巧与提示，帮助读者搭建良好的知识结构，养成良好的编程习惯，指导读者如何避免常见的编程错误以及写出高效、可靠的应用程序。

本书适合对 Python 感兴趣的读者阅读和参考。

EISBN: 0-13-092361-3

Python How To Program

H. M. Deitel, P. J. Deitel, J. P. Liperi, B. A. Wiedermann

Copyright © 2002 by Prentice-Hall, Inc.

Original English language edition published by Prentice-Hall, Inc.

All right reserved.

For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macau SAR).

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签，无标签者不得销售。

北京市版权局著作权合同登记号：图字 01-2002-4423 号

图书在版编目 (CIP) 数据

Python 编程金典 / (美) 迪特尔等著；周靖译。—北京：清华大学出版社，2003

(DEITEL 编程金典)

书名原文：Python How To Program

ISBN 7-302-06642-6

I. P... II. ①迪... ②周... III. 软件工具 程序 设计 IV. TP311.56

中国版本图书馆 CIP 数据核字 (2003) 第 036550 号

出 版 者：清华大学出版社(北京清华大学学研大厦,邮编 100084)

<http://www.tup.com.cn>

<http://www.tup.tsinghua.edu.cn>

责 任 编 辑：文升棋

印 刷 者：清华大学印刷厂

发 行 者：新华书店总店北京发行所

开 本：787×1092 1/16 **印 张：**37.25 **字 数：**1270 千字

版 次：2003 年 6 月第 1 版 2003 年 6 月第 1 次印刷

书 号：ISBN 7-302-06642-6/TP · 4970

印 数：0001 ~ 2000

定 价：88.00 元

前言

欢迎进入 Python 编程世界！Python 是一种强大的常规用途程序语言，尤其适合开发基于 Internet 和 Web 的、数据库密集型的、多层的客户机/服务器系统。本书讲解了大量先进的计算技术，它是我们的第二本有关开放源码程序语言的参考书。^①

在我们编写本书时，Python 2.2 刚刚发布。为此，我们进行了艰苦的工作，以便将 Python 2.2 的功能合并到本书中。附录 B 将讲解 Python 2.2 的其他一些特性。

希望读者通过本书获取有用的信息，既能感到适度的挑战，又能从中获得无穷乐趣！本书写作过程令人愉快。Deitel & Associates 的团队长期致力于程序语言教科书和 e-Learning 素材的开发。我们涉足几乎每一种主流程序语言。在编写本书过程中，我们也注意到一些特别的地方。我们的开发者和撰稿人对 Python 给与高度评价。它强大的功能、高度的可靠性和编码的简洁性，给人留下深刻印象。他们喜欢 Python 能让他们运用自如。他们喜欢这种开发源码的软件开发世界，为 Python 而开发的模块正在与日俱增。

无论老师、学生、有经验的专业程序员还是新手，都能通过本书汲取有益的知识。Python 是一种出色的程序语言，也是开发具有工业强度的商业应用程序的优秀语言。对于学生和新手级程序员，通过前几章的学习，可打下良好的基础。我们讨论了许多程序开发模型，包括结构化编程、基于对象的编程、面向对象的编程以及事件驱动的编程等。对于专业开发人员，我们则选用 Python 真正强大的功能来创建实用的、进行了完全实现的系统。这部分的重点在于第 23 章的案例分析，它详细讲解了如何构建一个真正的网上书店。

本书涉及了所有标准主题，包括数据类型、运算符、控制结构、算术运算、字符串、决策、算法开发、函数和随机数/模拟等等。

本书的特色之一是全面讲解了数据结构，书中首先介绍了 Python 的内建结构——列表、元组和字典。之后，还对包括队列、堆栈、链表和二叉树在内的传统数据结构进行了深入讲解。

本书强调了 Internet 和 Web 开发——我们首先介绍了 CGI，并在随后几章用它来构建基于 Web 的应用程序。我们用整章篇幅（第 25 章）介绍了 PSP（Python Server Pages，Python 服务器页），并利用它改编了第 16 章介绍的一个论坛案例。

本书用 3 章篇幅详细介绍了面向对象编程，涉及的主题包括类、封装、对象、属性、方法、构造函数、析构函数、自定义、运算符重载、继承、基类、派生类和多态性等等。

本书透彻讲解了如何用 Tkinter 进行 GUI（Graphical User Interface，图形用户界面）编程，涉及的主题包括事件驱动编程、标签、按钮、复选框、单选钮、鼠标事件处理、键盘事件处理、布局管理器以及一系列高级 GUI 功能，利用它们可创建和处理菜单和滚动组件。

我们讨论了如何利用异常处理使程序更“健壮”。Python 强大的字符串处理功能在此得到了深入讲解。至于正则表达式的主题，虽然它不易于理解，但由于它功能强大，所以我们也进行了详尽的解释。

我们讨论了文件处理、顺序访问文件、随机访问文件（以及 shelve 模块），同时还开发了一个事务处理程序，论述了对象序列化的问题。通过讨论文件处理，为后来的 Python 数据库编程奠定了良好的基础，后者通过 Python 的 DB-API（Database Application Programming Interface，数据库应用程序编程接口）来实现。我们讨论了关系数据库模型，并概述了 SQL（Structured Query Language，结构化查询语言）。

许多人都熟悉 HTML，但很少有人知道万维网协会（W3C）——HTML 技术的创建者——已声称 HTML 已成为“过去”，不会继续开发它。全世界正逐步过渡到 XML（eXtensible Markup Language，可扩展标记语言）。在这期间，Web 开发将采用一种名为 XHTML 的过渡技术。本书许多应用程序都将采用 XHTML。至于 XML 的常规主题，将采用一整章（即第 15 章）的篇幅来介绍它。对如今的 Web 应用

程序开发者来说，这是必须掌握的。然后，我们将另起一章（第 16 章），专门讲解 Python 的 XML 处理技术，并提供一个详细的案例分析，运用 CGI 和 XML 来构建论坛。

计算机应用程序通常能很好地一次做一件事。今天，较高级的应用程序需要同时做许多事，在计算机领域内，我们更喜欢将其称为“并发性”。我们会分别用整章的篇幅来讲解进程管理（第 18 章）和多线程处理（第 19 章）。Python 程序员利用这些技术，可做到以前只有系统程序员在操作系统的级别上才能做到的事情。

我们讨论了联网问题，包括 Web 使用的 HTTP 协议，使用流套接字进行的客户机/服务器联网，使用数据文报进行的无连接的客户机/服务器交互，另外还使用多线程服务器，实现了一个客户机/服务器的 Tic-Tac-Toe（即三连棋）游戏。

我们全面讨论了常见的计算机安全问题，并讲解了 Python 特有的一些安全问题。讨论了如何使用模块 `Bastion` 在一个限制环境中执行恶意代码。另外，还演示了如何用模块 `rotor` 对文本进行加密。

作为本书的一个重点，第 23 章展示了一个详尽的案例分析，它使用前几章和本书附录讨论的许多技术来实现一个电子商务网上书店。我们介绍了 HTTP 会话和会话跟踪技术，并将这个书店构建成一个多层次的客户机/服务器系统，它有能力处理大批量的客户，其中包括标准 Web 浏览器（使用 XHTML）和无线客户（使用 WML 和 XHTML Basic）。

我们用整章（第 24 章）的篇幅讲解多媒体所涉及的主题。使用 3D 图形例子介绍了 PyOpenGL，并介绍了 3D 环境 Alice，它提供的对象可通过 Python 脚本“动”起来。我们通过设计一个 CD 播放器，一个影片播放器和一个太空船游戏，演示了 `pygame`。

认识到服务器端开发的重要性之后，我们将展示 PSP（Python 服务器页），它可取代 CGI。另外，我们将论坛案例从 CGI 技术转换成了 PSP。

本书提供了两个附录，均与 Python 有关。其中，附录 A 介绍了 Python 开发环境，附录 B 介绍了 Python 2.2 的其他特点，其中还讨论了迭代器、生成器和嵌套作用域。

阅读本书的过程中，不管遇到什么问题，都请联系 `deitel@deitel.com`，来信必复。另外，请经常访问我们的网站 `www.deitel.com`，并订阅免费的 The Deitel BUZZ 电子刊物。我们会通过网站及电子刊物介绍最新的 Python 信息以及我们推出的其他产品和服务。

本书特色

本书具有许多特色，包括：

- **代码清洗 (code washing)**。这是我们自己创造的一个术语，是指对程序进行全面格式化，为其精心添加注释，并使其具有一个开放布局。程序代码组合成小的、结构清晰的块，这大大增强了可读性——这正是我们要达到的一个重要目标，尤其是全书总共含有 14 930 行代码！
- **面向对象编程**。面向对象编程是目前广泛采用的一种编程技术，用于开发健壮的、可重用的软件。Python 被设计成一种面向对象语言，本书全面讨论了 Python 的各种面向对象特性。数据完整性是 Python 中尤其要关注的一个问题。所有 Python 类数据默认都是公共的，但几种技术可用于确保数据完整性。我们用 3 章篇幅详细讨论了这些问题以及其他面向对象的主题。其中，第 7 章介绍如何创建类，并讨论了公共、私有和 get/set 方法。第 8 章解释如何使创建的类具有自定义行为，比如运算符重载、字符串表示、列表和字典行为以及用于自定义属性访问的方法等。第 9 章对这些概念进行了进一步扩展，我们讨论了如何创建新类，令其“吸收”现有类的功能。通过这一章的学习，读者会熟悉一些关键概念，比如多态性、抽象类和具体类等，利用它们可更方便地处理一个继承层次结构中的对象。本章最后讨论了 Python 2.2 提供的其他面向对象能力，其中包括“属性”(Properties)。
- **数据库应用程序编程接口**。数据库存储着大量信息，个人和单位需要访问它们以处理各种事务。数据库管理系统 (DBMS) 供单位和个人用来操纵数据库——Python 提供了相应的数据

库应用程序编程接口 (DB-API)，以访问数据库管理系统。第 17 章详细介绍了这些功能，另外还介绍了用于查询 MySQL 数据库的结构化查询语言 (SQL)。

- **XML。** 可扩展标记语言 (XML) 在软件开发领域和电子商务社区获得了蓬勃发展。由于 XML 是一种与平台无关的技术，用于描述数据和创建标记语言，所以 XML 的数据移植性能与 Python 的可移植应用程序和服务较好地集成在一起。第 15 章介绍了 XML，我们讨论了基本的 XML 标记和技术，比如 DTD 和 Schema，它们用于校验 XML 文档内容。第 16 章则解释了如何用文档对象模型 (Document Object Model, DOM) 来处理 XML 文档，以及如何通过可扩展样式表语言转换 (eXtensible Stylesheet Language Transformation, XSLT)，将 XML 文档转换成其他文档类型。本章还介绍了 DOM 的一种替代物，名为 Simple API for XML (简称 SAX)，它充当用于 XML 的一个基于事件的 API。
- **公共网关接口 (CGI) 和 Python 服务器页 (PSP)。** 因特网和万维网已深入人们的日常生活，交互式网站是商业成功的关键。第 6 章和第 25 章介绍了服务器端 Web 技术，开发者利用它们可创建交互式的、基于 Web 的应用程序。第 23 章提供了一个详细的案例分析，它综合运用 MySQL、XML、XHTML、XHTML Basic、层叠样式表 (CSS)、XSLT、CGI 和无线标记语言 (Wireless Markup Language, WML) 来构建一个动态电子商务应用程序。本书演示了一个 XML 论坛的两种实现方式，用户可将自己的文章张贴到在线论坛。第 16 章使用的是 CGI，第 25 章使用的则是 PSP。
- **图形用户界面 (GUI)。** Python 没有内建图形用户界面功能，但有许多模块可供使用，它们提供对现有的 GUI 软件的访问途径。第 10 章和第 11 章讨论了 Tkinter 模块（包括在 Python 标准库中），它允许 Python 程序员访问 Tool Command Language/Tk Kit (Tcl/Tk) 这一流行的 GUI 工具包。利用这些编程工具，开发者可快速、方便地创建图形程序。利用在这几章所学的知识，读者可为本书其余部分的程序开发 GUI。第 11 章还讨论了模块 Pmw，它利用 Tkinter 提供更复杂的 GUI 组件。
- **多媒体。** 多媒体功能可生成具有丰富视听感受的强大应用程序，由此增强用户的体验。可利用几个 Python 模块创建令人印象深刻的多媒体应用程序。第 24 章探讨了 PyOpenGL 和 Alice 的功能，它们可创建 3D 图形，并让它“动”起来。同时还讨论了 pygame，它所包含的模块便于开发者访问强大的多媒体库。第 24 章使用 pygame 创建一个 CD 播放器、一个电脑游戏以及一个影片播放器。
- **多线程处理和进程管理。** 计算机可并发执行大量任务，比如同时打印文档、从网络下载文件和在 Web 上冲浪等等。利用多线程处理技术，应用程序可执行并发性任务。Python 的多线程处理和进程管理功能尤其适合今天高度复杂的、多媒体密集型的、数据库密集型的、基于网络的、基于多处理器的以及分布式的应用程序。第 18 章讨论了并发性和进程间通信；第 19 章详细讨论了多线程处理的问题，其中详细解释了 Python 的 Global Interpreter Lock (即全局解释器锁，负责管理线程执行)。本章还通过几个例子，介绍了常见的线程同步机制。
- **文件处理和序列化。** 大多数应用程序都要在磁盘上读写数据。Python 针对数据存储和获取提供了几项高级功能。第 14 章讨论了用于存储顺序数据的基本文件对象、用于存储随机访问数据的 shelve 对象，以及用于将整个对象序列化到磁盘的 cPickle 模块。

此外，本书还讨论了其他许多主题。要详细了解每章的特点，请参阅 1.6 节。

Python 2.2 的特性

本书出版时，喜闻 Python 2.2 正式版刚刚发布。然而，本书所有示范代码都通过了 Python 2.2b2 (即 Beta 2) 和 Release Candidate 1 的测试。测试平台包括 Windows 和 Linux 操作系统。我们在各章尽可能介

绍 Python 2.2 的特性和功能。¹本书要介绍 Python 2.2 的以下特性。

Floor 除法和 True 除法：Python 2.2 引入新运算符（//）进行 Floor（整数）除法。在此之前，Python 版本中，除法运算符（/）的默认行为是 Floor 除法；在 2.2 以后的版本，默认行为变成 True（浮点）除法。通过定义两个除法运算符，Python 新版本可在同时使用了整数及浮点除法的程序中，避免出现类型混淆的问题。本书 2.6 节讨论了这两种除法的区别，并解释了程序如何更改除法运算符（/）的默认行为，令其执行 True 除法。

嵌套作用域：Python 2.2 引入了嵌套作用域的概念，它意味着嵌套的类、方法和函数现在可访问其封闭作用域中定义的变量。这种行为尤其适合编写 lambda 表达式。第 4 章讨论了 Python 的基本作用域规则，并提供了一系列万维网资源，便于读者更深入地了解嵌套作用域。如程序员在一种功能性编程模型中使用 Python，作用域的嵌套就显得非常重要。附录 B 更详细地讨论了嵌套作用域。由于本书强调的主要是面向对象的编程风格，所以只指出了使用嵌套作用域的一个高级动机，并推荐了可进一步参考的资源，便于读者在需要时了解 Python 中的嵌套作用域和功能性编程。

更多的面向对象功能：Python 2.2 的大多数新特性是为语言添加更多的面向对象功能。第 8 章和第 9 章介绍了其中的一些新特性。第 8 章讨论了如何重载一个由程序员定义的类，以便为运算符（包括新运算符//）定义行为。介绍了一个字典方法，它也是 Python 2.2 版本新增的，便于程序用 if/in 语句检测字典中是否包含一个特定的键。第 9 章讨论开发者们期待已久的新特性——允许从内建类型继承程序员定义的类。本章展示了一个实例，它继承自内建类型 list，目的是实现由程序员定义一个列表，其中只包含惟一性的元素。还讨论了其他面向对象特性，其中包括静态方法，`__slots__`（用于定义类中可能包含的一个对象的属性），方法`__getattribute__`（客户每次访问一个对象的属性时执行），以及属性（允许类定义 get/set 方法，以便在客户访问一个属性时执行）。

迭代器：附录 B 介绍的其他 Python 2.2 特点中，有的未在正文中详细讲解。附录 B 首先全面地探讨了迭代器——用于遍历一系列值的特殊对象。B.2 节提供了两个例子，它们展示了由程序员定义的迭代器类，并演示该类的一个客户如何使用迭代器从一个序列中获取值。第一个例子展示了如何定义一个类，使它的对象支持迭代器；第二个例子展示了一个计算机猜谜游戏，它展示如何利用迭代器处理长度不确定的序列。Python 2.2 采用了新的迭代器机制后，性能比以前的版本有了显著改进。另外，软件设计也因为程序员能分离迭代行为和随机访问行为而得以改进。

生成器：这是一种“可恢复函数”，能记住两次调用期间的状态。生成器也有利于性能和设计的改进。通常，程序中可以写一个生成器，采用一种简单、直观方式定义如何生成一个序列的元素。生成器还有利于执行重复性任务，或要求复杂逻辑和状态信息才能完成的任务。B.2 节围绕上述问题讨论了生成器，并定义了两个版本的生成器来计算斐波拉契序列。第一个版本不确定地生成序列中的下一个值；第二个生成所有序列值，直到包括用户自定义的第 *n* 个值。

万维网访问

本书（以及我们的其他出版物）所有示例代码都可从以下网站下载：

www.deitel.com

www.prenhall.com/deitel

注册过程非常简单。建议下载所有例子，并在阅读时运行相应的程序。修改例子，可马上看到修改效果——这是提升编程水平的有效方式之一。上述网站还解释了如何安装本书用到的各种软件（比如 Apache Web Server）。网站还提供其他 Web 服务器和软件的安装指南（注意，它们是有版权的。学习时可任意使用，但未经 Prentice Hall 和作者的书面许可，不得采取任何方式重新出版它的任何部分）。

¹ 阅读本书前，先从 python.org 下载最新 Python 版本。新版本发布后，我们会对本书代码进行测试，并在 www.deitel.com 进行相应的更新。阅读每一章之前，最好能访问我们的网站查看这些更新。

目 录

第 1 章 绪论	1
1.1 简介	1
1.2 开放源码软件的革命	1
1.3 Python 的历史	2
1.4 Python 模块	3
1.5 Python 和本书的一般注意事项	3
1.6 本书导读	3
1.7 因特网和万维网资源	8
第 2 章 Python 编程概述	9
2.1 简介	9
2.2 第一个 Python 程序：打印一行文本	9
2.3 修改第一个 Python 程序	11
2.4 另一个 Python 程序：整数求和	12
2.5 内存概念	14
2.6 算术运算	15
2.7 字符串格式化	19
2.8 做出决策：相等运算符和关系运算符	21
2.9 缩进	24
2.10 对象思想：对象技术简介	25
第 3 章 控制结构	27
3.1 概述	27
3.2 算法	27
3.3 伪代码	27
3.4 控制结构	28
3.5 if 选择结构	29
3.6 if/else 和 if/elif/else 选择结构	30
3.7 while 重复结构	34
3.8 算法陈述：案例分析 1（由计数器控制的重复）	35
3.9 算法陈述，自上而下求精法：案例分析 2（由哨兵值控制的重复）	37
3.10 算法陈述，自上而下求精法：案例分析 3（嵌套控制结构）	40
3.11 增量赋值符号	43
3.12 由计数器控制的重复的本质	44
3.13 for 重复结构	45
3.14 使用 for 重复结构	47
3.15 break 和 continue 语句	49
3.16 逻辑运算符	50
3.17 结构化编程总结	53
第 4 章 函数	57
4.1 概述	57
4.2 Python 中的程序组件	57

4.3 函数.....	58
4.4 math 模块的函数.....	58
4.5 函数定义.....	60
4.6 随机数生成.....	62
4.7 示例：博彩游戏.....	63
4.8 作用域规则.....	65
4.9 关键字 import 和命名空间.....	68
4.10 递归.....	70
4.11 递归示例：斐波拉契序列.....	72
4.12 递归与重复.....	74
4.13 默认参数.....	74
4.14 关键字参数.....	75
第 5 章 列表、元组和字典.....	77
5.1 概述.....	77
5.2 序列.....	77
5.3 创建序列.....	79
5.4 使用列表和元组.....	80
5.5 字典.....	86
5.6 列表和字典方法.....	88
5.7 引用和引用参数.....	92
5.8 将列表传给函数.....	92
5.9 列表排序和搜索.....	93
5.10 多下标序列.....	95
第 6 章 公共网关接口（CGI）入门.....	99
6.1 概述.....	99
6.2 客户和 Web 服务器交互.....	99
6.3 简单的 CGI 脚本.....	103
6.4 向 CGI 脚本发送输入.....	108
6.5 用 XHTML 表单发送输入并用 cgi 模块获取表单数据.....	110
6.6 用 cgi.FieldStorage 读取输入.....	113
6.7 其他 HTTP 标头.....	114
6.8 示例：交互式门户网站.....	114
6.9 因特网和万维网资源.....	117
第 7 章 基于对象的编程.....	118
7.1 概述.....	118
7.2 用类实现一个 Time 抽象数据类型.....	118
7.3 特殊属性.....	121
7.4 控制属性访问.....	122
7.5 为构造函数使用默认参数.....	128
7.6 析构函数.....	131
7.7 类属性.....	131
7.8 合成：对象引用作为类成员使用.....	133
7.9 数据抽象和信息隐藏.....	135
7.10 软件重用性.....	136

第 8 章	自定义类	138
8.1	概述	138
8.2	自定义字符串表示: <code>__str__</code> 方法	138
8.3	自定义属性访问	140
8.4	运算符重载	142
8.5	运算符重载的限制	143
8.6	重载一元运算符	144
8.7	重载二元运算符	144
8.8	重载内建函数	145
8.9	类型转换	146
8.10	案例分析: Rational 类	146
8.11	重载序列运算	152
8.12	案例分析: SingleList 类	152
8.13	重载映射运算	156
8.14	案例分析: SimpleDictionary 类	156
第 9 章	面向对象编程: 继承	159
9.1	概述	159
9.2	继承: 基类和派生类	160
9.3	创建基类和派生类	161
9.4	在派生类中覆盖基类方法	164
9.5	继承的软件工程学	165
9.6	合成与继承	166
9.7	“使用”和“知道”关系	166
9.8	案例分析: Point, Circle 和 Cylinder	167
9.9	抽象基类和具体类	170
9.10	案例分析: 继承接口和实现	170
9.11	多态性	173
9.12	类和 Python 2.2	174
第 10 章	图形用户界面组件 (一)	188
10.1	概述	188
10.2	Tkinter 简介	189
10.3	简单的 Tkinter 例子: Label 组件	190
10.4	事件处理模型	192
10.5	Entry 组件	192
10.6	Button 组件	195
10.7	Checkbutton 和 Radiobutton 组件	197
10.8	鼠标事件处理	201
10.9	键盘事件处理	205
10.10	布局管理器	207
10.11	洗牌和发牌模拟	213
10.12	因特网和万维网资源	215
第 11 章	图形用户界面组件 (二)	216
11.1	概述	216
11.2	Pmw 简介	216
11.3	ScrolledListBox 组件	216

11.4	ScrolledText 组件.....	218
11.5	MenuBar 组件.....	220
11.6	弹出菜单.....	223
11.7	Canvas 组件.....	225
11.8	Scale 组件	226
11.9	其他 GUI 工具包.....	227
第 12 章	异常处理.....	229
12.1	概述.....	229
12.2	引发异常.....	229
12.3	异常处理.....	230
12.4	示例：DivideByZeroError	232
12.5	Python 的 Exception 层次结构	234
12.6	finally 语句	235
12.7	Exception 对象和跟踪.....	238
12.8	程序自定义异常类.....	240
第 13 章	字符串处理和正则表达式.....	243
13.1	概述.....	243
13.2	字符和字符串基础.....	243
13.3	字符串表示.....	245
13.4	搜索字符串.....	246
13.5	连接和分解字符串.....	247
13.6	正则表达式.....	248
13.7	编译正则表达式和处理正则表达式对象.....	249
13.8	正则表达式的重复和置位字符.....	250
13.9	字符类和特殊序列.....	252
13.10	正则表达式的字符串处理函数.....	254
13.11	分组	255
13.12	因特网和万维网资源.....	256
第 14 章	文件处理和序列化.....	257
14.1	概述.....	257
14.2	数据层次结构.....	257
14.3	文件和流.....	258
14.4	创建顺序访问文件	259
14.5	从顺序访问文件读取数据.....	261
14.6	更新顺序访问文件.....	265
14.7	随机访问文件.....	265
14.8	模拟随机访问文件：shelve 模块	266
14.9	将数据写入 shelve 文件.....	266
14.10	从 shelve 文件获取数据.....	267
14.11	示例：一个事务处理程序.....	268
14.12	对象序列化.....	271
第 15 章	可扩展标记语言（XML）.....	274
15.1	概述.....	274
15.2	XML 文档.....	274
15.3	XML 命名空间.....	277

15.4 文档对象模型 (DOM)	280
15.5 Simple API for XML (SAX)	280
15.6 文档类型定义 (DTD)、架构和验证	281
15.7 XML 词汇表	287
15.8 可扩展样式表语言 (XSL)	292
15.9 因特网和万维网资源	296
第 16 章 Python 的 XML 处理.....	298
16.1 概述	298
16.2 动态生成 XML 内容	298
16.3 XML 处理包	300
16.4 文档对象模型 (DOM)	301
16.5 用 <code>xml.sax</code> 解析 XML	307
16.6 案例分析：用 Python 和 XML 实现论坛	309
16.7 因特网和万维网资源	321
第 17 章 数据库应用程序编程接口 (DB-API)	322
17.1 概述	322
17.2 关系数据库模型	322
17.3 关系数据库简介：Books 数据库	323
17.4 结构化查询语言 (SQL)	327
17.5 Python DB-API 规范	338
17.6 数据库查询示例	338
17.7 查询 Books 数据库	341
17.8 读取、插入和更新数据库	344
17.9 因特网和万维网资源	348
第 18 章 进程管理.....	349
18.1 概述	349
18.2 <code>os.fork</code> 函数	349
18.3 <code>os.system</code> 函数和 <code>os.exec</code> 函数家族	355
18.4 控制进程的输入和输出	358
18.5 进程间通信	361
18.6 信号处理	363
18.7 发送信号	364
第 19 章 多线程处理.....	367
19.1 概述	367
19.2 线程状态：生命期	367
19.3 <code>threading.Thread</code> 示例	369
19.4 线程同步	371
19.5 生产者/消费者关系：无线程同步	372
19.6 生产者/消费者关系：有线程同步	376
19.7 生产者/消费者关系：Queue 模块	380
19.8 生产者/消费者关系：循环缓冲区	383
19.9 信号机	388
19.10 事件	390
第 20 章 联网.....	392
20.1 概述	392

20.2 通过 HTTP 定址 URL	392
20.3 建立简单服务器（使用流套接字）	394
20.4 建立简单客户（使用流套接字）	395
20.5 通过流套接字连接进行客户/服务器交互	396
20.6 通过数据文报进行无连接的客户/服务器交互	399
20.7 使用多线程服务器的客户/服务器 Tic-Tac-Toe 游戏	401
第 21 章 安全性	409
21.1 概述	409
21.2 密码系统古今谈	409
21.3 加密密钥	412
21.4 公钥加密	414
21.5 密码破解	415
21.6 密钥协商协议	416
21.7 密钥管理	416
21.8 数字签名	417
21.9 公钥基础结构	418
21.10 安全协议	420
21.11 身份验证	422
21.12 安全攻击	424
21.13 运行受限 Python 代码	427
21.14 网络安全	430
21.15 隐写术	432
第 22 章 数据结构	434
22.1 概述	434
22.2 自引用类	434
22.3 链表	434
22.4 堆栈	441
22.5 队列	443
22.6 树	444
第 23 章 案例分析：网上书店	449
23.1 概述	449
23.2 HTTP 会话和会话跟踪技术	449
23.3 在网上书店中跟踪会话	450
23.4 网上书店体系结构	453
23.5 配置网上书店	455
23.6 进入网上书店	456
23.7 从数据库获得书籍列表	457
23.8 查看一本书的详细资料	462
23.9 在购物车中添加商品	465
23.10 查看购物车	466
23.11 结账	470
23.12 处理订单	472
23.13 错误处理	473
23.14 处理无线客户端（XHTML Basic 和 WML）	475
23.15 因特网和万维网资源	494

第 24 章 多媒体.....	495
24.1 概述.....	495
24.2 PyOpenGL 简介.....	495
24.3 PyOpenGL 示例.....	495
24.4 Alice 简介.....	501
24.5 狐狸、鸡和种子问题.....	501
24.6 pygame 简介.....	505
24.7 Python CD Player.....	506
24.8 Python Movie Player.....	510
24.9 用 pygame 开发太空船游戏.....	513
24.10 因特网和万维网资源.....	524
第 25 章 Python 服务器页 (PSP)	525
25.1 概述.....	525
25.2 Python Servlet.....	525
25.3 PSP 简介.....	526
25.4 第一个 PSP 示例.....	527
25.5 隐式对象.....	529
25.6 脚本编程.....	529
25.7 标准动作.....	532
25.8 预编译指令.....	540
25.9 案例分析：用 Python 和 XML 实现论坛.....	545
25.10 因特网和万维网资源.....	559
附录 A Python 开发环境.....	560
A.1 概述.....	560
A.2 集成开发环境：IDLE.....	560
A.3 其他集成开发环境.....	564
A.4 因特网和万维网资源.....	566
附录 B Python 2.2 的其他特点.....	567
B.1 概述.....	567
B.2 迭代器.....	567
B.3 生成器.....	574
B.4 嵌套作用域.....	577
B.5 因特网和万维网资源.....	579

第 1 章 绪 论

学习目标

- 了解开放源码软件
- 熟悉 Python 程序语言的历史
- 本书导读

1.1 简介

欢迎进入 Python 的世界！希望通过我们艰苦的努力，带给读者一本内涵丰富、寓教于乐的计算机参考书。为此，我们采用了多种方法，最终写成了一本与众不同的 Python 参考书。例如，我们很早便讲解了 Python 如何与公共网关接口（CGI）配合，以便编写基于 Web 的应用程序。这样一来，便可在本书剩余部分，更好地演示大量动态的、基于 Web 的应用程序。本书介绍了大量重要主题，包括面向对象编程（OOP）、Python 数据库应用程序编程接口（DB-API）、图形、可扩展标记语言（XML）以及安全性。不管您是一名新手还是有经验的程序员，本书提供的信息量、趣味性和挑战性，都会让您称心如意。

本书面向所有层次的读者，从正式的程序员，一直到很少或根本没有编程经验的自学者。同一本书，怎么可能同时适用于高低两个层次的读者呢？其中的关键在于，我们以成熟的“结构化编程”以及“基于对象的编程”技术为准，始终都在强调如何编写“思路清晰”的程序。非程序员出身的人可以学会基本的技能，为将来进行良好编程奠定基础。有经验的程序员将获得对语言的严谨解释，而且书中的内容有助于改进他们的编程风格。为帮助刚入门的程序员，我们采用一种清晰的、平铺直叙的方式，其间穿插大量插图。另外，更重要的是，本书提供了数百个功能完整的 Python 程序。本书所有示例程序都可从我们的网站 (www.deitel.com) 下载。

大多数人或多或少都熟悉计算机令人激动的功能。使用本书，可学会如何指挥计算机，亲自实现那些功能。毕竟，是“软件”（要求计算机采取行动和做出决策的指令）在控制着计算机（通常称为“硬件”）。

今天，几乎每个领域都越来越依赖计算机。在其他成本都在稳定、缓慢提升的同时，计算成本却呈显著下降态势——这完全归功于硬件和软件技术的快速发展。25~30 年前，需要几个大房间才能摆下的大型计算机，以及那些动辄数百万美元的“巨无霸”，现在只需指甲大小的硅芯片即可搞定。成本也降至每片几美元左右。不过，具有讽刺意味的是，“硅”是我们这个地球上不值钱的东西之一。在海边随手抓一把沙子，里面含的绝大多数元素便是“硅”。硅芯片技术的问世，使得计算成本变得异常低廉，也直接促成了如今数亿台计算机广泛应用于各行各业。在商业、工业、政府以及我们的个人生活方面，计算机都能提供强有力的帮助。而且再过几年，这个数字还可以轻松翻一倍。

从现在起，您将开始一段美妙的、令人激动的、充满挑战的以及令人回味无穷的学习之旅。在这个旅程中，如果您碰到什么问题，不妨发信给 deitel@deitel.com，或浏览我们的网站 (www.deitel.com、www.prenhall.com/deitel 和 www.InformIT.com/deitel)。祝您学习顺利。

1.2 开放源码软件的革命

如果程序的源代码免费提供给任何开发人员进行修改，传播，以及用作其他软件的基础，就称为“开放源码软件”或“开源软件”。¹相反，“封闭源码软件”则禁止其他开发者以之为基础开发其他软件。

开放源码并不是一个新概念。早在 20 世纪 60 年代，开源技术就是现代计算工业得以迅速发展的一项主要推动力。特别是美国政府出资兴建了今天 Internet 的前身，并鼓励计算机科学家开发各种各样的

¹ 真正的开源（Open-Source）软件要符合 9 条要求。详情参见 www.opensource.org/docs/definition.html。

技术，以实现在各种计算机平台上的分布式计算。这演变成了今天的一系列重要技术，比如用于实现 Internet 通信的协议。Internet 建好之后，封闭源码技术及软件才逐渐在软件工业中流行起来，开放源码的思想在 20 世纪 80 年代和 90 年代初一度受到抑制。但在这之后，为了反击大多数商业软件的“封闭”本质，并因为封闭源码厂商冷漠的态度，开放源码软件再度流行。如今，Python 已成为快速成长的开源软件社区的一部分，其他还有 Linux 操作系统、Perl 脚本编程语言、Apache Web 服务器以及成百上千的其他软件项目。

计算机行业的一些人将开放源码视为 Free（免费）软件。大多数情况下，这种说法是成立的。不过，在谈到开放源码软件“Free”时，一种更合适的说法是“Freedom”（自由）——任何开发者都可自由修改源码，交流编程思想，参与软件开发过程，以及基于现有的开源软件开发出新的软件程序。大多数开源软件实际都是有版权的，要使用它们，需要获得相应的许可证。开放源码许可协议所规定的条款是各不相同的；有的只有极少限制（比如 Artistic License），另一些可能有许多限制，详细规定软件的修改和使用方式。通常，软件的版权由个人开发者或者一个组织所持有。若要查看许可证/许可协议的一个例子，可访问 www.python.org/2.2/license.html，仔细阅读 Python 的使用许可协议。

通常，可通过 Internet 下载开源产品的源代码。这使开发者能够学习、检查和修改源码，以满足他们自己的需求。由于有整个开发者社区作为后盾，有许多人审查代码，所以相较于封闭源码软件开发，性能和安全问题能更快得到检测和解决。另外，更大的开发者社区可为一个软件提供更多的特性。通常，代码修正数小时内便可推出，开源软件新版本的推出频率也要比封闭源码软件高出许多。在开源软件的使用许可协议中，通常要求开发者发表他们所做的任何改进，这使得开源社区不断地发展壮大，不断地改进现有的产品。例如，Python 开发者喜欢加入 comp.lang.python 新闻组，交流有关 Python 开发的心得体会。Python 开发者还可为自己的修改编写文档，并通过 Python Enhancement Proposals (PEPs) 提交给 Python Software Foundation (Python 软件基金会)。这样一来，Python 开发团体就能对提议的修改进行评估，并在未来的版本中集成好的改进。

许多公司（比如 IBM、Red Hat 和 Sun）都支持开源开发者及项目。有时，这些公司还会取得开源应用程序，并通过商业途径销售它们（这取决于软件的使用许可协议）。为获得赢利，他们还为开源软件提供一系列服务，比如技术支持、为客户定做软件和培训等等。开发者可作为顾问或培训者提供服务，帮助用户实现软件。欲知开源软件的详情，请访问 Open Source Initiative 网站 (www.opensource.org)。

1.3 Python 的历史

Python 起源于 1989 年末。当时，CWI（阿姆斯特丹国家数学和计算机科学研究所）的研究员 Guido van Rossum 需要一种高级脚本编程语言，为其研究小组的 Amoeba 分布式操作系统执行管理任务。为创建新语言，他从高级教学语言 ABC (All Basic Code) 汲取了大量语法，并从系统编程语言 Modula-3 借鉴了错误处理机制。然而，ABC 的一个重大缺点是扩展性不足：语言不是开放式的，不利于改进或扩展。因此，Van Rossum 决定在新语言中合成来自现有语言的许多元素，但要求必须能通过类和编程接口进行扩展。他将这种新语言命名为 Python（原意为“大蟒蛇”）——来源于 BBC 当时正在热播的喜剧片连续剧“Monty Python”。

自 1991 年初公开发行后，Python 开发者和用户社区逐渐壮大，Python 语言逐渐演变成一种成熟的、并获得良好支持的程序语言。Python 被用来开发大量应用程序，从创建网上电子邮件程序到控制水下交通工具，以及配置操作系统和创建动画片等等。2001 年，核心 Python 开发团队移师 Digital Creations 公司，后者是 Zope (用 Python 编写的一个 Web 应用程序服务器) 的创始人。预计 Python 会继续成长与发展，进入一个全新的领域。

1.4 Python 模块

Python 是一种模块化的可扩展语言，它可随时集成新“模块”（Modules）——一种可重用的软件组件。任何 Python 开发人员都能编写新模块来扩充 Python 的功能。Python 源代码、模块和文档资料的主要“集散地”是 Python 网站 (www.python.org)，该网站还计划建立一个专门维护 Python 模块的网站。

1.5 Python 和本书的一般注意事项

Python 经过了良好的设计，无论新手还是有经验的程序员都能快速学习和理解这种语言，并轻松上手。和其前身不同，Python 具有良好的移植和扩展能力。Python 的语法和设计有利于养成良好的编程习惯，并可在不牺牲程序扩展性与维护性的同时，显著缩短开发时间。

Python 相当简单，新手程序员可轻松上手；但它同时具有强大的功能，对专家也有足够的吸引力。本书通过丰富、完整且实际有效的例子和讨论，介绍了大量编程概念。随着学习的深入，读者可通过我们创建的实际应用程序，探索更加复杂的主题。贯穿全书，我们始终在强调良好的编程习惯，并给出大量移植性提示以及解释如何防范常见的编程错误。

Python 是当前移植能力最强的程序语言之一。最初，它是在 UNIX 上实现的。但之后扩展到了其他许多平台，其中包括 Microsoft Windows 和 Apple Mac OS X。Python 程序通常可直接从一种操作系统移植到另一种操作系统，无需任何更改，而且能确保正确执行。

1.6 本书导读

本节简要介绍全书讲解的各个主题。有的章末附有“因特网和万维网资源”小节，提供有关 Python 编程的更多信息。

第1章——绪论：介绍开放源码革命，讲述 Python 程序语言的起源，并概括本书其余各章主要内容。

第2章——Python 编程概述：介绍一个典型的 Python 编程环境，并解释了 Python 程序的基本语法。我们讨论了如何从命令行运行 Python。除解释器之外，Python 还可在一一个交互模式中执行语句，即输入一个语句，立即执行一个。在本章及全书，我们会展示许多交互会话，强调各种平常容易忽视的编程要点。本章讨论了变量，介绍了算术运算、赋值、相等、关系和字符串运算符。我们介绍了决策和算术运算。字符串是一种基本的、功能强大的内建数据类型。我们介绍了一些标准的输出格式化技术，并讨论了“对象”和“变量”的概念。对象是值的容器，而变量是用于引用对象的名称。结束本章的学习后，读者将理解如何编写简单而又完整的 Python 程序。

第3章——控制结构：介绍用于解决问题的“算法”（过程）。解释了高效率使用控制结构的重要性：控制结构可使程序易于理解、调试和维护，而且有助于首次试运行即告成功。本章介绍了选择结构（if, if/else 和 if/elif/else）和重复结构（while 和 for）。我们详尽解释了如何进行重复，并对比了计数器控制的循环和哨兵值控制的循环。同时还解释了“自上而下求精法”为什么有助于生成结构正确的程序，有助于建立一种流行的程序设计辅助手段——伪代码。本章提供的案例分析演示了如何快速方便地将伪代码算法转换成能实际工作的 Python 代码。本章还解释了用于改变控制流程的 break 和 continue 语句。另外，我们展示了怎样用逻辑运算符 and, or 和 not 在程序中做出复杂的决策。本章的几个交互式会话演示了如何创建一个 for 结构，以及如何避免结构化编程中的一些常见编程错误。本章最后对结构化编程进行了总结。利用本章介绍的技术，能在任何程序语言中有效使用控制结构，而非局限于 Python。本章有助于读者培养良好的编程习惯，为进行本书后面更高级的编程任务打好基础。

第4章——函数：讨论了“函数”的设计与构建。在 Python 中，和函数相关的功能包括内建函数、程序员定义的函数和递归等。本章介绍的技术是创建具有正确结构的程序的基础（尤其是系统程序员和

应用程序开发者针对实际环境而开发的较大的程序和软件时)。本章展示了“分而治之”策略,它是解决复杂问题的有效手段,具体做法是将这些问题分解成较简单的交互式组件。我们首先将模块描述成一种容器,它用于容纳一组有用的函数。我们介绍了math模块,并讨论了其中许多同数学有关的函数。许多人喜欢学习随机数和模拟,在掷骰子游戏案例分析(其中充分运用了控制结构)过程中,他们会觉得非常有趣。本章解释了如何解决一个斐波拉契和阶乘问题,具体利用的是一种名为“递归”的技术(即函数调用自身)。本章讨论了作用域规则,并用一个例子来检查局部变量和全局变量。本章还讨论了程序如何采取各种不同的方式导入模块及其元素,import语句会对程序的命名空间造成什么影响。Python的函数可指定默认参数和关键字参数。我们讨论了向函数传递信息的两种方式,并在一个交互式会话中讲解了某些常见的编程错误。

第5章——列表、元组和字典:本章详细介绍了3种高级的Python数据类型,即列表、元组和字典。利用这些数据类型,Python程序员可通过最少的代码行完成非常复杂的任务。字符串、列表和元组全都属于“序列”的范畴,后者是一种数据类型,可通过索引和“切片”进行处理。我们讨论了怎样创建、访问和操纵序列,并提供了一个例子(它根据由值构成的一个序列来创建柱形图)。我们介绍了列表和元组用于Python程序的各种方式。字典是“可映射”类型——键和值成对保存,并相互映射(对应)。我们通过一个用于存储学生成绩的例子解释了怎样创建、初始化和操纵字典。另外还介绍了“方法”,它们是用于执行对象(比如列表和字典)的各种操作的函数,并解释了如何利用方法来访问、排序和搜索数据。这些方法可轻松地完成各项计算任务;换用其他语言,则可能要用大量代码行才可获得同样的效果。我们还介绍了“不可变序列”和“可变序列”。如果将多个可变序列传递给函数,会产生一个重要的、可能出乎预料的“副作用”。我们用一个例子来展示这种副作用的效果。

第6章——公共网关接口(CGI)入门:本章介绍了可使应用程序(CGI程序或脚本)与Web服务器进行交互的一种协议,即“超文本传输协议”(HTTP),它是Web服务器和Web浏览器之间进行数据通信的一种基本组件。我们解释了客户如何通过Internet连接到服务器,以及运行CGI程序的Web服务器如何将响应发送给客户。从Web服务器发送给Web浏览器的最常见的数据就是网页,它是用“可扩展超文本标记语言”(XHTML)格式化的一种文档。在本章,将介绍如何创建简单CGI脚本。另外,还介绍了如何将用户输入从浏览器发送给一个CGI脚本,并用实例展示了如何在Web浏览器中显示一个人的姓名。然后重点讲解怎样利用XHTML表单,将用户输入发送给一个CGI脚本,以及在客户端和服务器上的CGI程序之间传递数据。随后演示如何用cgi模块处理表单数据。本章描述了随CGI使用的各种HTTP标头。最后,我们在一个Web门户案例分析中运用了CGI知识,它允许用户登录到一个虚拟旅游网站,查看特价信息。

第7章——基于对象的编程:本章开始讨论基于对象的编程。这是讲解“数据抽象”的良机——利用的是一开始就设计为面向对象的Python语言。近几年,数据抽象已成为计算机初级课程的重要主题之一。我们讨论了如何通过一个类实现时间抽象数据类型,如何初始化和访问类的成员。和其他语言不同,Python不允许程序员禁止属性访问。本章和第8章及第9章讨论了几种访问控制技术。我们介绍了“私有”属性以及get和set方法,它们控制着对数据的访问。所有对象和类都具有公共属性,本章讨论了它们的名称和值。讨论了默认构造函数后,我们进一步扩展了以前的例子。另外还介绍了用于指出错误的raise语句。类可包含类属性——即只需创建一次,然后可供类的所有实例使用的数据。还讨论了一个有关合成的例子,其中的实例可将其他实例作为数据成员引用。本章最后讨论了软件重用问题。

第8章——自定义类:本章讨论Python用于自定义类行为的几个方法。这些方法扩展了以前各章介绍的访问控制机制。自定义技术最强大的功能或许就是“运算符重载”,它允许程序员告诉Python解释器如何将现有的运算符用于新类型的对象。Python已知如何将这些运算符用于内建类型的对象,比如整数、列表和字符串等。但是,假如创建了一个新的Rational类,那么在Rational对象之间使用的一个加号(+)到底表示什么呢?在本章,程序员将学习如何“重载”加号。这样一来,如果在表达式的两个Rational对象之间使用加号,解释器就会生成对一个“运算符方法”的方法调用,它负责将两个Rational对象“加”到一起。本章讨论了运算符重载的基础知识及其存在的限制,如何重载一元运算符和二元运算符,以及如何在不同类型之间转换。本章还讨论了如何自定义类,使其具有列表或字典行为。

第9章——面向对象编程：继承：本章介绍了面向对象程序语言最基本的能力之一，即“继承”。继承是软件重用的一种形式，通过吸收现有类的功能，再添加合适的新功能，可以快速和简单地开发出新类。本章讨论了基类和派生类的记号法、直接基类、间接基类、基类和派生类中的构造函数和析构函数以及继承的软件工程学问题。本章还比较了各种面向对象关系，比如继承与合成。通过继承，人们发展出了Python最值得骄傲的一种编程技术，即“多态性”。如果许多类都通过继承和同一个公共基类联系，那么每个派生类对象都可视为一个基类实例。这样一来，程序可采用一种泛型方式编写，独立于派生类对象的具体类型。新的对象可由同一个程序处理，使系统更容易扩展。人们经常采取这种形式的编程来实现当今流行的“图形用户界面”(GUI)。本章最后讨论了Python 2.2支持的新的面向对象编程技术。

第10章——图形用户界面组件（一）：本章介绍了Tkinter，该模块为流行的Tcl/Tk图形用户界面工具包提供了一个Python接口。本章在简要概述Tkinter模块之后了一番概述。使用Tkinter，可快速方便地创建图形化程序。本章介绍了几个基本Tkinter组件：Label、Button、Entry、Checkbutton和Radiobutton。我们讨论了事件处理的概念，它是GUI编程的核心；本章还通过几个例子揭示如何在GUI应用程序中处理鼠标及键盘事件。本章最后深入探讨了Pack、Grid和Place这几种Tk布局管理器。结束本章的学习后，读者应能理解大多数Tkinter应用程序。

第11章——图形用户界面组件（二）：本章讨论了附加的GUI编程主题。我们介绍了Pmw模块，它对基本的Tk GUI元件集进行了扩展。本章展示了如何创建菜单、弹出菜单、滚动文本框和窗口。本章的例子演示了如何将文本从一个窗口拷贝到另一个，允许用户选择和显示图片，更改文本字体以及更改窗口背景颜色。最有趣的是一个长度仅39行的程序，它允许用户用鼠标在Canvas组件上绘图。本章最后讨论了适用于Python程序员的其他GUI工具包，其中包括pyGTK、PyOpenGL以及wxWindows。在本书剩余部分，大量例子都要用到第10章和第11章讲解的GUI技术。结束这两章的学习后，读者就可着手为数据库、联网和简单游戏程序编写其中的GUI部分。

第12章——异常处理：利用本章提供的知识，程序员可使自己编写的程序可靠性和容错性更佳、更适合用于面向关键任务和事务的环境。本章首先解释了异常处理技术，然后讨论了应在什么时候进行异常处理，还通过在一个例子中使用try/except/else语句（该例子能非常好地处理“除以零”这一严重逻辑错误），从而介绍了异常处理的基础知识。程序员可用raise语句明确引发异常；我们讨论了该语句的语法，并演示了它的用法。本章解释了如何从异常提取信息，以及以何种方式在何时引发异常。我们解释了finally语句，详细解释了应在什么时候、在什么地方捕捉异常。在Python中，异常的本质是“类”。我们通过探讨异常层次结构，解释如何创建自定义异常，阐述了异常和类的关系。本章最后提供了一个例子，它利用traceback模块来探讨Python异常的本质及内容。

第13章——字符串处理和正则表达式：本章解释了如何处理字符串外观、顺序和内容。字符串是大多数Python输出的基础。本章讨论了count、find和index方法，它们用于在字符串中搜索子字符串。split方法可将字符串分解成一个字符串列表。replace方法可将字符串中的一个子字符串替换成另一个。这些方法提供了基本的文字处理能力，但程序员经常需要进行更强大的、基于模式的文字处理。re正则表达式模块在Python中实现了基于模式的文字处理。正则表达式处理是一个非常复杂的话题，而且存在许多缺陷。我们从基本正则表达式开始，一直讲到较高级的一些主题。我们指出了最常见的编程错误，并用例子解释这些错误是如何发生的，以及如何避免。我们讨论了re模块的常用函数和类，并介绍了常用的正则表达式元字符和序列。我们演示了分组的概念，它允许程序员从正则表达式处理结果中提取信息。Python正则表达式可进行编译以增强正则表达式的处理性能，所以我们讨论了何时适合这样做。

第14章——文件处理和序列化：本章讨论了用于处理顺序访问和随机访问文本文件的技术，并根据位、字节、字段、记录和文件，对数据层次结构进行了概述。然后，我们介绍了Python的简单文件及文件句柄视图。我们用示例程序展示了如何打开和关闭文件，如何将数据顺序存储到一个文件中，以及如何从一个文件中顺序读取文件，由此讨论了顺序访问文件的详情。这些例子使用第13章介绍的字符串格式化技术输出从一个文件中读取的数据。本章还提供了一个比较实用的信用查询程序，可从一个顺序访问文件中获取数据，然后根据获取的数据，对输出进行格式化。本章讨论了如何使用print语句将文本重定向至任何一个文件，其中包括程序用于显示错误消息的“标准错误文件”。讨论随机访问文件时，我们

使用 `shelve` 创建一个随机访问文件，并在该文件中读写数据。最后展示了一个较大的事务处理程序，它综合运用了本章讨论的所有技术。Python 高级数据类型的一个优点是程序可对任意 Python 对象进行序列化（存储到磁盘）。我们在一个例子中使用 `cPickle` 模块将 Python 字典保存到磁盘，以便将来使用。

第 15 章——可扩展标记语言（XML）：XML 是用于创建标记语言的一种语言。HTML 用子格式化要显示的信息；相反，XML 用于对信息进行结构化。它不像 HTML 那样有一套固定标记，它允许文档作者自行创建新标记。本章概要介绍了“解析器”（对 XML 文档及其数据进行处理的程序）。另外还解释了“良构文档”（即符合语法规则的文档）的要求。随后介绍了命名空间（用于区分同名元素）。还介绍了文档类型定义（DTD）文件和 Schema（架构）文件，它们通过指定元素在 XML 文档中的类型、顺序、数量和属性，为 XML 文档提供了一个结构化定义。通过定义一个 XML 文档的结构，DTD 或 Schema 可减少使用该文档的那个应用程序的验证和错误检查工作。本章介绍了一种非常流行与 XML 相关的技术，名为“可扩展样式表语言”（XSL），它的作用是将 XML 文档转换成另一种文档格式，比如 XHTML。本章只是对 XML 的概述。第 16 章要进一步介绍如何在 Python 中处理 XML。

第 16 章——Python 的 XML 处理：本章要讨论如何使用标准和第三方模块，简单但高效地进行 Python XML 处理和操纵。本章概述了处理 XML 文档的几种方式，并讨论了 W3C 的文档对象模型（DOM），它是用于 XML 的一种应用程序编程接口（API），与平台和语言无关。DOM API 提供了一系列标准接口（即方法、对象等等），可用它们处理一个 XML 文档的内容。XML 文档具有层次结构，所以 DOM 将 XML 文档表示成树结构。使用 DOM，程序可动态修改文档的内容、结构和格式。本章还介绍了除 DOM 之外的另一种选择，名为 Simple API for XML（SAX）。DOM 是在内存中构建一个树结构；SAX 则不同，它在文档中遇到起始标记、结束标记、属性等等时，会调用特定的方法。因此，通常将 SAX 称为“基于事件的 API”。Python 的 XML 支持是借助 `xml.dom.ext`（DOM）以及 `xml.sax`（SAX）模块来实现的。本章使用了 4Suite（由 FourThought 公司开发）和 PyXML，这是 Python XML 模块的两个集合。本章最后提供了一个案例分析，它用 XML 实现了一个基于 Web 的论坛。

第 17 章——数据库应用程序编程接口（DB-API）：本章介绍如何用程序查询和处理数据库。大多数实用的商业程序和 Web 应用程序都要基于“数据库管理系统”（DBMS）。为支持 DBMS 应用程序，Python 提供了“数据库应用程序编程接口”（DB-API）。本章使用“结构化查询语言”（SQL）来查询和处理“关系数据库管理系统”（RDBMS）。具体地说来，RDBMS 是一个 MySQL 数据库。为建立与 MySQL 的接口，Python 使用了模块 `MySQLdb`。本章包含 3 个例子：第一个是根据用户指定条件显示作者信息的 CGI 程序；第二个创建了 GUI 程序，允许用户输入 SQL 查询，然后显示查询结果；第 3 个例子是更实用的 CGI 程序，允许用户维护一个联系人列表。可在数据库中添加、删除、更新和查找联系人。

第 18 章——进程管理：本章讨论了“并发性”。大多数程序语言都提供一系列简单控制结构，允许程序员每次执行一个任务，上一个任务结束之后才能执行下一个。这种控制结构不允许采取并发操作。今天，由计算机执行的并发性操作通常以操作系统指令的形式实现，这仅适用于有经验的系统程序员。Python 则不然，它使应用程序的开发者也能使用并发性指令。本章介绍了如何使用 `fork` 命令新建进程；如何用 `exec` 和 `system` 命令执行单独的程序；还演示了如何用 `popen` 命令控制输入和输出。注意，有的命令只适合 Unix 平台。本章讨论了 Python 的跨平台能力，并用实例演示如何根据操作系统来执行特定的任务。我们讨论了进程之间的通信方法，其中包括管道和信号。在本章的信号处理例子中，演示了如何判断用户正在试图中断程序；如何指定发生此类事件时要执行的行动。

第 19 章——多线程处理：本章介绍了“线程”，它是“轻量级进程”。相较于第 18 章用 `fork` 等命令创建的全功能进程，线程通常更有效。本章探讨了基本的线程处理概念，其中包括线程生命期中可能出现的各种状态。我们讨论了如何通过子类化 `threading.Thread` 和覆盖 `run` 方法，将线程包括到一个程序中。本章后半部分用一系列例子阐述了经典的生产者/消费者关系。我们为这个问题开发了几个解决方案，并介绍了线程同步和资源分配概念。本章讨论了线程处理控制指令，比如锁、条件变量、信号机和事件等等。最后一个方案用 `Queue` 模块保护对队列所存储共享数据的访问。示例演示了多线程程序存在的危险，以及如何避免这些危险。我们的解决方案还演示了可重用类的巨大价值。我们重用生产者类和消费者类来访问各种同步和非同步的数据类型。结束本章的学习后，读者就可利用许多 Python 工具编写实用的、

可扩展的专业程序。

第 20 章——联网：本章介绍通过计算机网络进行通信的应用程序。Python 等高级语言的一个重要优势在于，可通过较小的、能实际工作的例子来方便地展示及讨论非常复杂的主题。我们讨论了基本联网概念，并提供两个例子：一个 CGI 程序（在浏览器中显示选择的网页）；一个 GUI 程序（在文本区域中显示页面内容）。还讨论了通过套接字进行的客户/服务器通信。示例程序演示了如何利用无连接和基于连接的协议，通过网络来收发消息。本章最具特色的内容就是以“活代码”方式实现一个协作式的客户机/服务器 Tic-Tac-Toe（三连棋）游戏。两个客户与一个多线程的服务器通信，共同玩一个网络版游戏，服务器负责维护游戏的状态。

第 21 章——安全性：本章讨论了 Web 编程的安全问题。Web 编程可快速创建功能强大的应用程序，但这也使计算机易于受到外界攻击。本章讨论了防御性编程技术，它们利用特定技术及工具，帮助程序员防范安全问题。一种工具是加密，我们举例说明了如何用 `rotor` 模块进行加密和解密，`rotor` 模块实现了一个置换密码系统。另一个工具是 `sha` 模块，它用于哈希处理。第 3 个工具是 Python 的限制访问（`rexec`）模块，它可创建一个受限环境。不受信任的代码将在其中执行，以避免它们损害本地计算机。本章探讨了保证网络安全性的各种技术，比如公钥密码、安全套接字层（SSL）、数字签名、数字证书、数字隐写术和生物测定等等。还讨论了其他类型的网络安全性，比如防火墙和防病毒程序。另外还介绍了一些常见的安全威胁，包括密码破解攻击、病毒、蠕虫和特洛伊木马等。

第 22 章——数据结构：本章讨论 Python 中用于创建和处理标准数据结构的技术。尽管高级数据类型是 Python 内建的，但从概念和编程角度来讨论这些常用数据结构，对读者来说大有益处。本章首先讨论了自引用结构，接着讨论如何创建和维护各种数据结构，包括链表、队列、堆栈和二叉树。我们通过重用链表类来实现队列和堆栈，尽可能减少继承的类的代码，并将讨论的重点集中于代码重用。二叉树类包含了用于进行前序遍历、中序遍历和后序遍历的方法。针对每种类型的数据结构，书中都展示了完整的、可实际工作的程序，并展示了示范输出。

第 23 章——案例分析：网上书店：本章实现了一个网上书店，它综合运用 MySQL、XML 和 XSLT 向不同的客户发送网页。我们首先介绍了 HTTP 会话框架，它可在几个页之间维护客户信息。客户信息在服务器的计算机上进行序列化，供服务器日后使用。然后讨论了 WML，它是无线客户用于在 Web 上传输文档的一种标记语言。尽管我们在演示时用 XHTML、XHTML Basic 和 WML 等客户来访问这个应用程序，但书店系统具有灵活的扩展性，可方便地添加新的客户类型。Python CGI 程序本身不必改变，但程序员可修改网上书店系统，为新客户端创建新的 XML 和 XSLT 文档，从而为新的客户提供服务。网上书店程序确定客户类型，并将恰当的数据发送给客户。本章综合了本书以前许多章的主题，并有力证明了 Python 的一个重要优势——快速而方便地集成多种技术。这些主题包括文件处理、序列化（`cPickle` 模块）、CGI 表单处理（`cgi` 模块）、数据库访问（`MySQLdb` 模块）、XML DOM 和 XSLT 处理（`4Suite` 模块集）。

第 24 章——多媒体：本章展示了 Python 对多媒体应用程序的支持。对于选修入门级程序课程的学生，本章内容相当重要。一些有趣的多媒体应用程序包括：`PyOpenGL`（该模块将 Python 绑定到 OpenGL API，创建彩色的、富有吸引力的交互式图形）、`Alice`（它采取面向对象方式创建和操纵 3D 图形世界）以及 `Pygame`（它是许多 Python 模块的集合，用于创建跨平台多媒体应用程序，比如交互式游戏等等）。`PyOpenGL` 示例创建了旋转对象和三维图形。`Alice` 示例创建了一个图形版本的流行解谜游戏。本例创建的世界中，包括一只狐狸、一只鸡和一粒种子。游戏目标是将它们都移过河，但又不让任何捕食者和被捕食者有机会单独相处。第一个 `Pygame` 例子合并了 `Tkinter` 和 `Pygame` 以创建 GUI 形式的 CD 播放器。第二个例子演示如何播放 MPEG 电影。最后一个 `Pygame` 例子创建一个电脑游戏，玩家驾驶一艘太空船经过小行星带并收集能源。通过这个例子，我们讨论了图形编程的大量要点和技术。换用其他程序语言，这些项目会变得过于复杂或琐碎，以至于无法放到本书进行讲解。然而，Python 高级的抽象能力、简单的语法以及丰富的模块使不可能的任务变成可能。所有这些令人激动的例子都可放到同一章讲解！

第 25 章——Python 服务器页 (PSP)：本章要用熟悉的可扩展超文本标记语言（XHTML）以及 Python 脚本来创建动态 Web 内容。我们分别讨论了客户端和服务端的问题。本章所用的工具包括 Apache 和

Webware for Python，后者是一套用来创建动态 Web 内容的软件。简要介绍了 Python Servlet 之后，本章还提供了一些例子，其中讲解了 PSP 如何处理 Python 独特的缩进样式，解释了 Scriptlet、动作以及预编译指令。

附录 A——Python 开发环境：本附录简要介绍了几种 Python 开发环境，其中包括 IDLE。

附录 B——Python 2.2 的其他特点：本书出版时正值 Python 2.2 发布。书中反映了大量 Python 2.2 特性，但仍有一些无法写入正文。因此，本附录整理了这些附加特性。阅读各章时，注意参考附录 B，看看是否有附加的讨论及相应的示例。

网站资源

Deitel 网站 (www.deitel.com) 提供许多 Python 资源，帮助您在 Windows 或 UNIX/Linux 系统上安装和配置 Python。这些资源包括“Installing Python”，“Installing the Apache Web Server”，“Installing MySQL”，“Installing Database Application Programming Interface (DB-API) modules”，“Installing Webware for Python”以及“Installing Third-Party Modules”。

好了，就是这么多！我们经过艰苦工作，为您创作了这本书。书中含有大量能实际运行的例子、编程提示以及数不清的学习要点，可帮助您全面地掌握 Python。利用学到的知识，您可快速和高效地编写基于 Web 的应用程序。阅读本书的过程中，如果有任何不明白的地方，或者想报告所发现的错误，请致函 deitel@deitel.com。我们会尽快答复，并在 www.deitel.com 公布勘误和其他信息。

Prentice Hall 维护着 www.prenhall.com/deitel，该网站专门介绍我们为 Prentice Hall 开发的参考书、多媒体软件和基于 Web 的培训产品。可在这里找到针对我们的每本书的“配套网站”，并提供了包括 FAQ、下载、勘误、更新和自测题在内的大量资源。Deitel & Associates 公司每周为流行的 InformIT 电子刊物撰写一个专栏，该刊物目前有 80 多万名订阅者。详情请见 www.InformIT.com。

现在，您将开始一个令人激动的、充满乐趣的 Python 学习之旅，祝一路顺风！

1.7 因特网和万维网资源

www.python.org

这是寻找 Python 信息的主要地方。Python 主页提供最新新闻、FAQ 以及到 Python 资源的链接，这些资源包括 Python 软件、教程、用户组和演示等等。

www.zope.com 或 www.zope.org

Zope 是一个可扩展的、开放源码的 Web 应用程序服务器，它是用 Python 编写的。它由 Digital Creations 公司创建，整个 Python 开发团队都在这个公司中。

www.activestate.com

ActiveState 为程序员创建开放源码工具。该公司提供的 Python 产品名为 ActivePython 和 Komodo，这是一个为许多语言（包括 Python、XML、Tcl 和 PHP 在内）提供的、开放源码的 IDE（集成开发环境）。ActiveState 为 Windows 平台提供 Python 工具，并提供了一个名为“Python Cookbook”的 Python 程序集。

homepage.ntlworld.com/tibsnjoan/python.html

提供大量链接，可通过它们访问正在开发和使用 Python 的许多个人和组织。

www.ddj.com/topics/pythonurl/

“Dr. Dobb's Journal”是一份编程出版物，它提供了一系列有用的 Python 链接。

第 2 章 Python 编程概述

学习目标

- 理解一个典型的 Python 程序开发环境
- 用 Python 编写简单的计算机程序
- 使用简单的输入语句和输出语句
- 熟悉基本数据类型
- 会使用算术运算符
- 理解算术运算的优先顺序
- 会编写简单的、用于做出决策的语句

2.1 简介

Python 为计算机程序设计提供了一种规范方式。本章将介绍 Python 编程，并提供几个例子，演示该语言的重要特性。为了理解每个例子，我们打算每次只分析一个语句。展示了基本概念之后，我们将在第 3~5 章探讨“结构化编程”方式。在探讨 Python 的基本主题的同时，还会提前进行面向对象编程的讨论——这是贯穿全书的一种关键性编程方法。为此，我们专门提供了 2.10 节“对象思想”。

2.2 第一个 Python 程序：打印一行文本

先来看一个能打印一行文本的简单程序。图 2.1 展示了这个程序及其屏幕输出。^①

```
1 # Fig. 2.1: fig02_01.py
2 # Printing a line of text in Python.
3
4 print "Welcome to Python!"
```



图 2.1 文本打印程序

该程序演示了 Python 语言的几项重要特性。让我们分别研究每一行代码。本书展示的每个程序都有行号，以便读者参考。但行号并非实际的 Python 程序的一部分。图 2.1 中，第 4 行是这个程序的重点，它在屏幕上显示短语“Welcome to Python!”。

第 1~2 行都以符号#开头，表明每一行剩余的内容都是“注释”。插入注释的目的是“文档化”程序并改善程序的可读性。注释还可帮助其他程序员阅读和理解您的程序。程序运行时，注释不会导致计算机对其采取任何操作——Python 会忽略注释。我们的每个程序都用一条注释开头，它指出图号以及存储了程序的文件名（第 1 行）。在注释中可使用任何文本。本书所有 Python 程序都可从 www.deitel.com 免费下载。

以符号#开头的一条注释称为“单行注释”，注释会在当前行的末尾结束。以符号#开头的注释也可从一行的中间开始，并一直延续到行末，这样的注释通常用于文档化位于那一行开头的 Python 代码。和其他程序语言不同，Python 没有单独的符号用于多行注释，所以多行注释的每一行都必须以符号#开头。注释文本“Printing a line of text in Python”描述了程序的用途（第 2 行）。

良好编程习惯 2.1 在程序中使用丰富的注释。注释有助于其他程序员理解程序，有助于程序调试（即

^① 本书的资源（包括在 Windows 和 Unix/Linux 平台上安装 Python 的每个步骤的指导）都已发布在 www.deitel.com 网站上。

发现和排除程序中的错误), 并列出有用的信息。以后修改或更新代码时, 注释还有助于您理解自己当初编写的程序。

良好编程习惯 2.2 每个程序都应以一条注释开头, 描述该程序的用途。

第 3 行是空行。程序员使用空行和空格字符使程序易于阅读。空行、空格字符和制表符统称为“空白”或“空距”(White Space), 其中, 空格字符和制表符称为“空白字符”。空行会被 Python 忽略。

良好编程习惯 2.3 加一些空行来增强程序的可读性。

Python 的 print 命令(第 4 行)指示计算机显示包含在引号 ("") 之间的字符串。字符串是一系列字符的组合, 它们包含在一对双引号中。整行称为一个“语句”。在某些程序语言(比如 C++, Java 和 C#)中, 语句必须以分号 (;) 结尾。在 Python 中, 一旦当前行结束, 大多数语句也就结束了。第 4 行执行时, 会在屏幕上显示“Welcome to Python!”消息。注意, 对字符串进行定界的双引号不会出现在输出中。

Python 中的输出(显示信息)和输入(接收信息)是用字符“流”来完成的。上述语句执行时, 它将字符流“Welcome to Python!”发送给“标准输出流”。标准输出流是应用程序向用户展示信息的一种渠道, 这种信息通常在屏幕上显示, 但也可打印到打印机, 或者写到文件, 甚至可以被朗读出来, 或者输出到盲文设备, 使视力有缺陷的人也能接收输出。

Python 语句可通过两种方式执行。第一种是在编辑器中输入语句以创建一个程序, 再使用.py 扩展名来保存文件(如图 2.1 所示)。Python 文件名通常以.py 结尾, 但也可使用其他扩展名(例如 Windows 平台上的.pyw)。要用 Python 解释器执行文件中的程序, 请在 DOS 或 Unix shell 命令行中输入:

```
python file.py
```

其中的 file.py 就是 Python 文件的名称。shell 命令行是一种文本“终端”, 用户可在其中输入命令, 让计算机系统做出响应。注意, 为了能调用 Python, 系统路径变量必须正确设置, 以包括 python 可执行文件——其中包含了可以运行的 Python 解释器程序。本书的资源(发布在 www.deitel.com 网站上)提供了相应的指导, 帮您设置正确的系统路径变量。

Python 解释器运行存储在文件中的程序时, 会从文件的第一行开始, 并一直执行到文件结束。图 2.1 的输出框显示了 Python 解释器运行 fig02_01.py 的结果。

执行 Python 语句的另一种方式是“交互模式”。在 shell 命令行输入:

```
python
```

即可在“交互模式”中运行 Python 解释器, 它允许程序员直接向解释器输入语句, 每次执行一个语句。

测试和调试提示 2.1 在交互模式输入一个 Python 语句就会执行一个。调试程序时, 这种模式尤其有用。

测试和调试提示 2.2 对一个文件调用 Python 解释器后, 解释器会在文件中的最后一个语句执行之后退出。然而, 如果使用 -i 选项(例如 python -i file.py)针对文件调用解释器, 会导致编译器在执行了文件中的语句后进入交互模式。这非常适用于调试程序。

图 2.2 展示了 Python 2.2 在 Windows 上以交互模式运行的样子。前 2 行显示了与所用 Python 的版本信息(2.2b2 表示“版本 2.2, Beta 2”)。第 3 行包含“Python 提示符”(>>>)。在 Python 提示行输入一个语句, 并按回车键之后, 解释器会执行该语句。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Welcome to Python!"
Welcome to Python!
>>> ^Z
```

图 2.2 交互模式

图 2.2 的第 3 行在屏幕上显示文本“Welcome to Python!”。再次提醒，用于定界的双引号不会打印出来。将文本打印到屏幕后，解释器等待用户输入下一个语句。要退出交互模式，只需输入文件结束符 Ctrl-Z（在 Microsoft Windows 平台上）并按回车键。图 2.3 列出了不同计算机系统上的“文件结束”符。

计算机系统	组合键
UNIX/Linux 系统	Ctrl-D（要在单独一行上）
DOS/Windows	Ctrl-Z（有时要接着按回车键）
Macintosh	Ctrl-D

图 2.3 各种主流计算机系统上的文件结束组合键

2.3 修改第一个 Python 程序

本节通过两个例子继续介绍 Python 编程，它们对图 2.1 中的例子进行了修改，使用多个语句显示单行文本，以及使用单个语句显示多行文本。

2.3.1 用多个语句显示单行文本

“Welcome to Python!”可用几种方式打印。例如，图 2.4 使用两个 print 语句（第 4~5 行）生成和图 2.1 一样的输出。程序的大多数地方都和图 2.1 相同，所以这里只讨论改动的地方。

第 4 行显示字符串“Welcome”。通常，print 语句显示了它的字符串后，Python 会开始一个新行——后续的输出在 print 语句所打印的字符串之后的一行或多行上显示。然而，第 4 行末尾的逗号（,）告诉 Python 不要另起新行，而是在字符串后添加一个空格。因此，程序显示的下一个字符串（第 5 行）会与字符串“Welcome”出现在同一行。

```
1 # Fig. 2.4: fig02_04.py
2 # Printing a line with multiple statements.
3
4 print "Welcome",
5 print "to Python!"
```

Welcome to Python!

图 2.4 使用几个 print 语句打印单行文本

2.3.2 用单个语句显示多行文本

使用“换行符”，一个语句可显示多行。换行符属于“特殊字符”，用于将屏幕光标定位到下一行的开头。图 2.5 输出 4 行文本，并用换行符决定每个新行何时开始。

```
1 # Fig. 2.5: fig02_05.py
2 # Printing multiple lines with a single statement.
3
4 print "Welcome\nto\n\nPython!"
```

Welcome
to

Python!

图 2.5 用单个 print 语句打印多行

这里只讨论图 2.5 和图 2.1 及图 2.4 所示程序有区别的地方。第 4 行在屏幕上显示 4 行独立的文本。

通常，字符串中的字符会采取与它们在双引号中一模一样的方式显示。但要注意，有两个字符\和n没有在输出中出现（它们在第4行出现了3次）。Python提供了“特殊字符”，可用它们来执行特定的任务，比如退格和回车等等。特殊字符要用反斜杠(\)加以区分，反斜杠字符也称为“转义字符”。如果字符串中存在一个反斜杠，它和紧接其后的字符便构成了“转义序列”。\n就是这样的一个转义序列，它代表换行符。\\每出现一次，就会导致屏幕光标（它控制下一个字符的出现位置）移到下一行的开头。要打印一个空行，只需连续使用两个换行符。图2.6总结了其他常见的转义序列。

转义序列	说明
\n	换行符。将屏幕光标定位至下一行起始位置
\t	水平制表符。将屏幕光标移至下一个制表位
\r	回车符。将屏幕光标定位至当前行起始位置；不转到下一行
\a	响铃符。发出系统响铃声
\\\	反斜杠。用于打印一个反斜杠字符
\"	双引号。用于打印一个双引号字符
\'	单引号。用于打印一个单引号字符

图2.6 转义序列

2.4 另一个Python程序：整数求和

下一个程序要求用键盘输入两个整数（如-22, 7和1024等），程序计算两者之和，并显示结果。程序调用Python函数raw_input和int来获得两个整数。同样地，用print语句显示求和结果。图2.7展示了示例程序及其输出。

```

1 # Fig. 2.7: fig02_07.py
2 # Simple addition program.
3
4 # prompt user for input
5 integer1 = raw_input("Enter first integer:\n") # read string
6 integer1 = int(integer1) # convert string to integer
7
8 integer2 = raw_input("Enter second integer:\n") # read string
9 integer2 = int(integer2) # convert string to integer
10
11 sum = integer1 + integer2 # compute and assign sum
12
13 print "Sum is", sum      # print sum

```

```

Enter first integer:
45
Enter second integer:
72
Sum is 117

```

图2.7 求和程序

第1~2行包含注释，说明程序的编号、文件名和用途。第5行调用Python内建函数raw_input，要求用户输入。内建函数是由Python提供的一部分代码，用于执行特定任务。任务通过调用函数（函数名，后续一对圆括号()）来执行。执行任务后，函数可能返回代表任务结果的值。第4章将深入学习函数，届时会提到其他许多内建函数，并解释如何创建程序员自定义函数。

Python的raw_input函数取得一个参数，即"Enter first integer:\n"，它请求用户输入。所谓参数（或“实参”）是指函数能接受的一个值，函数要利用它来执行任务。在本例，函数raw_input接受的是“提示内容”参数（要求用户输入），并将提示内容显示在屏幕上。用户看到提示消息后，需要输入一个数字，并按回车键。这样即可采用字符串形式将数字发送给函数raw_input。

`raw_input` 的结果是一个字符串，其中包含由用户输入的字符，它会被指派给变量 `integer1`，这是用赋值符号=实现的。在 Python 中，可将变量称为“对象”。对象驻留于计算机内存中，并包含程序使用的信息。平时提到“对象”时，通常暗示着有“属性”（数据）和“行为”（方法）同对象联系在一起。对象的方法利用属性来执行任务。变量名（例如 `integer1`）包括字母、数位和下划线（_），但不可用一个数位开头。Python 是一种要区分大小写的语言，例如 `a1` 和 `A1` 属于不同的变量。对象可以有多个名称，即“标识符”。每个标识符（或变量名）都引用（或指向）内存中的一个对象（或变量）。第 5 行的语句通常读作“变量 `integer1` 被指派给 `raw_input("Enter first integer:\n")` 所返回的值”。然而，通过前面的解释，我们知道这行代码实际的含义是“`integer1` 引用了 `raw_input("Enter first integer:\n")` 返回的值”。

良好编程习惯 2.4 有意义的变量名可改善程序的“自编档能力”；也就是说，只需读一读程序，就能轻松理解它，而不必非要阅读手册或使用过多的注释。

良好编程习惯 2.5 避免标识符以下划线和双下划线开头，因为 Python 解释器可能保留了那些名称，供内部使用。这样可避免您选择的名称与解释器选择的名称混淆。

除了名称和值，每个对象还有一个“类型”。对象类型标识对象中存储的信息种类（比如整数、字符串等等），整数包括负整数 (-14)、零 (0) 和正整数 (6)。在诸如 C++ 和 Java 的语言中，程序员必须先声明对象类型，然后才能在程序中使用它。但是，Python 使用的是“动态类型定义”技术，即在程序执行期间决定一个对象的类型。例如，假如对象 `a` 初始化成 2，对象的类型会被定为 `integer`（整数），因为 2 是一个整数。类似地，如果对象 `b` 初始化成“Python”，对象的类型会成为 `string`（字符串）。函数 `raw_input` 返回类型为 `string` 的值，因此 `integer1` 引用的值属于 `string` 类型。

为了对 `integer1` 返回的值执行整数加法，必须先将字符串值转换成整数值。Python 的 `int` 函数（第 6 行）可将字符串或数字转换成整数值，并返回新值。如没有为变量 `integer1` 获得一个整数值，便无法获得希望的结果——程序会对两个字符串进行合并操作，而不是执行整数加法。图 2.8 对此进行了演示。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = raw_input("Enter an integer: ")
Enter an integer: 2
>>> value2 = raw_input("Enter an integer: ")
Enter an integer: 4
>>> print value1 + value2
24
```

图 2.8 事先不转换为整数便对 `raw_input` 返回的值进行相加，会得到错误结果（正确结果应该是 6）

赋值语句（图 2.7 的第 11 行）计算变量 `integer1` 和 `integer2` 的和，并将结果指派给变量 `sum`，这是用赋值符号=来完成的。这个语句可读作“`sum` 引用 `integer1 + integer2` 的值”。大多数计算都是通过赋值语句来执行的。

加号 (+) 是一个运算符，它是一个执行特定操作的特殊符号。本例中，运算符+执行加法运算，我们将其称为“二元运算符”，因为它需要两个操作数（值）才能执行运算。本例的两个操作数分别是 `integer1` 和 `integer2`。注意在 Python 中，符号=不是运算符而是赋值符。

常见编程错误 2.1 试图访问一个未赋值的变量，会产生运行时错误。

良好编程习惯 2.6 在二元运算符两端添加一个空格。这样可突出运算符，增强程序的可读性。

第 13 行显示字符串“Sum is”和变量 `sum` 的数值。我们想要输出的项目用逗号 (,) 分隔。注意，这个 `print` 语句输出的值具有不同的类型，即一个字符串和一个整数。

还可在输出语句中执行计算。可将第 11 行和第 13 行的语句合并成：

```
print "Sum is", integer1 + integer2
```

这可避免使用变量 sum。只有在认为合并能使程序更清晰的前提下，才可这样做。

2.5 内存概念

integer1, integer2 和 sum 等变量实际对应于 Python 的对象。每个对象都有自己的类型、大小、值以及在计算机内存中的位置。程序不可更改对象的类型或位置。有的对象类型允许程序员更改对象的值。第 5 章将详细地讨论这些类型。

图 2.7 的程序执行以下语句时：

```
integer1 = raw_input("Enter first integer:\n")
```

Python 首先创建一个对象，以容纳用户输入的字符串，然后将它放到一个内存位置。然后，赋值符=将名称 integer1 同新建的对象绑定（关联）起来。假定用户在 raw_input 提示处输入 45，Python 就会将字符串"45"放到与名称 integer1 对应的内存位置，如图 2.9 所示。

执行以下语句时：

```
integer1 = int(integer1)
```

函数 int 会新建一个对象，用它来保存整数值 45。整数对象将从一个新内存位置开始，Python 将名称 integer1 同这个新内存位置绑定（图 2.10）。从此，变量 integer1 不再引用包含了字符串值"45"的内存位置。



图 2.9 显示一个变量值及其绑定名称的内存位置

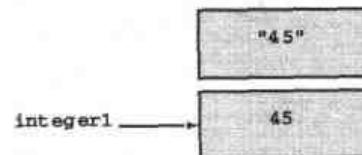


图 2.10 显示变量名称和值的内存位置

返回我们的求和程序，执行以下语句时：

```
integer2 = raw_input("Enter second integer:\n")
integer2 = int(integer2)
```

假定用户输入字符串"72"。程序把这个值转换成整数值 72，并将其放到与 integer2 绑定的内存位置之后，内存的布局如图 2.11 所示。注意，这些对象的位置在内存中不一定是相邻的。

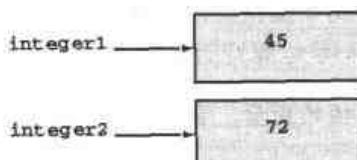


图 2.11 两个变量的值都已输入后的内存位置

程序获得 integer1 和 integer2 的值后，会把这些值加到一起并将结果指派给变量 sum。当以下语句：

```
sum = integer1 + integer2
```

执行加法后，内存布局如图 2.12 所示。注意在 sum 计算前后，integer1 和 integer2 的值是没有变化的。

换言之，从内存位置读取一个值时，该值不会改变。

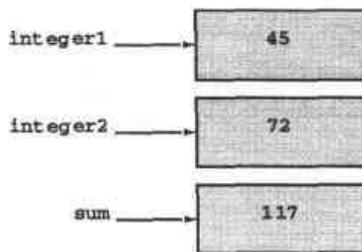


图 2.12 计算后的内存位置

从图 2.13 可以看出，每个 Python 对象都有一个位置、一个类型以及一个值，而且所有这些对象“属性”都是通过对对象的名称来访问的。该程序和图 2.7 中的程序基本一致，只是我们在这一程序的各个地方，增添了一些语句以显示每个对象的内存位置、类型和值。

```

1 # Fig. 2.13: fig02_13.py
2 # Displaying an object's location, type and value.
3
4 # prompt the user for input
5 integer1 = raw_input("Enter first integer:\n") # read a string
6 print "integer1: ", id(integer1), type(integer1), integer1
7 integer1 = int(integer1) # convert the string to an integer
8 print "integer1: ", id(integer1), type(integer1), integer1
9
10 integer2 = raw_input("Enter second integer:\n") # read a string
11 print "integer2: ", id(integer2), type(integer2), integer2
12 integer2 = int(integer2) # convert the string to an integer
13 print "integer2: ", id(integer2), type(integer2), integer2
14
15 sum = integer1 + integer2 # assignment of sum
16 print "sum: ", id(sum), type(sum), sum
  
```

```

Enter first integer:
5
integer1: 7956744 <type 'str'> 5
integer1: 7637688 <type 'int'> 5
Enter second integer:
27
integer2: 7776366 <type 'str'> 27
integer2: 7637352 <type 'int'> 27
sum: 7637436 <type 'int'> 32
  
```

图 2.13 对象的位置、类型和值

完成 `raw_input` 调用后，第 6 行打印 `integer1` 的位置、类型和值。Python 函数 `id` 可返回解释器对变量内存位置的表示。函数 `type` 用于返回变量类型。将 `integer1` 的字符串值转换成整数值后，第 8 行再次打印这些值。注意在执行以下语句后，变量 `integer1` 的类型和位置都发生了变化：

```
integer1 = int(integer1)
```

这一变化证明了程序不可更改变量类型这一事实。相反，这条语句会导致 Python 在新的内存位置创建一个新的整数值，并将 `integer1` 这个名称指派给该位置。`integer1` 以前引用的位置不能继续访问。示例程序剩余部分采用类似的方式，打印变量 `integer2` 和 `sum` 的位置、类型和值。

2.6 算术运算

大多数程序都要执行算术运算。图 2.14 总结了算术运算符。注意，一些特殊符号在标准代数中是没

有的。星号 (*) 代表乘法，百分号 (%) 是取模运算符，不久就会讨论这两个运算符。图 2.14 中的算术运算符是二元运算符，因为它们都要用到两个操作数。例如，表达式 integer1 + integer2 包含了二元运算符+以及两个操作数 integer1 和 integer2。

Python 运算	算术运算符	代数表达式	Python 表达式
加	+	$f + 7$	<code>f + 7</code>
减	-	$p - c$	<code>p - c</code>
乘	*	$b \cdot m$	<code>b * m</code>
求幂	**	x^y	<code>x ** y</code>
除	/	x / y 或 $\frac{x}{y}$ 或 $x \div y$	<code>x / y</code>
取模	%	$r \bmod s$	<code>x // y</code>
			<code>x % s</code>

图 2.14 算术运算符

Python 是一种不断进步的语言，它的一些特性会随着时间的推移而发生变化。从 Python 2.2 起，除法运算符 (/) 的行为由“Floor 除法”变成“True 除法”。Floor 除法有时也称为整数除法，它将分子除以分母，返回不大于结果的最大的一个整数值。例如，如采用 Floor 除法，7 除以 4 结果是 1；17 除以 5 是 3。注意在 Floor 除法中，所有小数部分都被抛弃（即“截尾”），不进行进位处理。相反，True 除法产生精确的浮点结果，其中含有小数点，比如 7.0, 0.0975 和 100.12345。例如，如采用 True 除法，7 除以 4 的结果就是 1.75。

在以前的版本中，Python 只提供一个除法运算符，即运算符/。运算符的行为（即 Floor 还是 True 除法）由操作数的类型决定。如操作数全是整数，就执行 Floor 除法。如一个或两个操作数是浮点数，就执行 True 除法。

语言设计者和许多程序员都不喜欢运算符/的这种随意性，所以决定在 2.2 版采用两个运算符。其中，运算符/执行 True 除法，运算符//则执行 Floor 除法。然而，在用老版本 Python 写的程序中，这样的设计有可能导致错误。因此，设计者采取了一个折衷方案：从 Python 2.2 开始，未来的所有 2.x 版本都会采用两个运算符，但假如程序作者希望使用新的行为，就必须使用以下语句明确表达自己的意愿：

```
from __future__ import division
```

Python 一旦看到这个语句，运算符/就会严格执行 True 除法，运算符//则严格执行 Floor 除法。图 2.15 中的交互会话演示了这两种除法。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3 / 4      # floor division (default behavior)
0
>>> 3.0 / 4.0  # true division (floating-point operands)
0.75
>>> 3 // 4     # floor division (only behavior)
0
>>> 3.0 // 4.0 # floating-point floor division
0.0
>>> from __future__ import division
>>> 3 / 4      # true division (new behavior)
0.75
>>> 3.0 / 4.0  # true division (same as before)
0.75
```

图 2.15 运算符/行为的差异

首先对表达式 $3 / 4$ 进行求值。表达式的结果是值 0，因为运算符/对于两个整数操作数的默认行为是 Floor 除法。表达式 $3.0 / 4.0$ 的求值结果是 0.75。在这里，由于使用的是浮点操作数，所以运算符/会执行 True 除法。表达式 $3 // 4$ 和 $3.0 // 4.0$ 的求值结果分别为 0 和 0.0，因为运算符//总是执行 Floor 除法，不管操作数的对象是什么。然后，在第 13 行，我们用 import 语句更改运算符/的行为。实际上，该语句打开了运算符/的 True 除法能力。现在，表达式 $3 / 4$ 的计算结果是 0.75。注意本书只采用运算符/默认的 2.2 版本的行为，即：针对整数执行 Floor 除法（参见图 2.15 的第 5~6 行），针对浮点数执行 True 除法（参见图 2.15 的第 7~8 行）。

移植性提示 2.1 预计在 Python 3.0 中，运算符/只能执行 True 除法。3.0 版本发布后，程序员需要更新自己的程序，以兼容新行为。欲知详情，访问 python.sourceforge.net/peps/pep-0238.html。

Python 提供了取模运算符%，用于得到整除后的余数。 $x \% y$ 这个表达式会得到 x 被 y 除之后的余数。因此， $7 \% 4$ 等于 3，而 $17 \% 5$ 等于 2。这个运算符经常用于整数操作数，但也可用于其他算术类型。后文还会讨论取模运算符的许多有趣的应用，比如判断一个数字是否另一个数字的倍数（它的一个特例便是判断一个数字是奇数还是偶数）。注意，取模运算符可用于整数和浮点数类型。

Python 中的算术表达式必须采用“直线”形式输入计算机。因此，像“ a 被 b 除”这样的表达式必须写成 a / b ，使所有常量、变量和运算符出现在一条直线中。下述代数表达式：

$$\frac{a}{b}$$

将是编译器或解释器无法接受的——尽管一些特殊用途的软件包允许以这种更自然的记号法来表示复杂的代数表达式。

在 Python 表达式中，圆括号的用法和代数表达式中大致相同。例如要将 a 乘以 $b + c$ 的和，可写成：

$$a * (b + c)$$

Python 根据由下述“运算符优先级规则”规定的精确顺序，在算术表达式中应用不同的运算符。这些规则同在代数中的规则大致相同：

1. 圆括号中的表达式先求值。所以程序员可根据自己的愿望，用圆括号强制按任何顺序求值。圆括号具有“最高优先级”。在嵌套或嵌入圆括号的情况下，由内向外求值。
2. 接着是求幂运算。如表达式包含几个求幂运算，就按从右到左的顺序进行。
3. 接着是乘法、除法和取模运算。如一个表达式中包含了几个乘、除和取模运算，就按从左到右的顺序求值。我们认为乘、除和取模具有相同的优先级。
4. 加、减运算最后进行。如果一个表达式同时包含了几个加法和减法运算，就按从左到右的顺序求值。加、减也具有相同的优先级。

如表达式中包含几对圆括号，这些圆括号并不一定是嵌套的。例如表达式：

$$a * (b + c) + c * (d + e)$$

并不包含嵌套圆括号。但该表达式中的圆括号具有相同的优先级。

当我们说特定的运算符从左到右求值时，事实上说的是运算符顺序关联性。比如以下表达式中：

$$a + b - c$$

加法运算符 (+) 从左到右顺序关联。注意，也有一些运算符是从右到左顺序关联的。

图 2.16 总结了这些运算符优先级规则。介绍了更多的 Python 运算符后，我们还会对该表进行扩充。

运算符	运算	求值顺序
()	圆括号	最先求值。如果圆括号是嵌套的，最内层的表达式先求值。如果同时有几对圆括号具有相同优

运算符	运算	求值顺序
$**$	求幂	先级（即没有嵌套），就按从左到右的顺序求值
$*$ / $//$ $%$	乘、除、取模	其次求值。如同时有几个，就按从右到左的顺序求值
$+/-$	加、减	第三个求值。如同时有几个，就按从左到右的顺序求值。注意，运算符 $//$ 是 Python 2.2 新增的

图 2.16 算术运算符的优先级

现在，让我们基于运算符优先顺序来讨论几个例子。每个例子都列出了代数表达式及其对应的 Python 表达式。下面是求 5 个值的算术平均值的例子：

$$\text{代数: } m = \frac{a+b+c+d+e}{5}$$

Python: `m = (a + b + c + d + e) / 5`

圆括号是必需的，因为除法拥有的优先级比加法高。我们的目的是将 $(a + b + c + d + e)$ 除以 5。遗漏了圆括号，得到的就是 $a + b + c + d + e / 5$ 。它等价于下面这个不正确的代数表达式：

$$a + b + c + d + \frac{e}{5}$$

下面是一个直线形式的方程式例子：

$$\text{代数: } y = mx + b$$

Python: `y = m * x + b`

无需任何圆括号。首先会执行乘法，因为乘法具有比加法更高的优先级。

下例包含了取模 (%)、乘法、除法、加法和减法运算：

$$\text{代数: } z = pr \% q + w / x - y$$

Python: `z = p * r % q + w / x - y`
 ① ② ④ ③ ⑤

圆圈内的数字指明 Python 执行运算的顺序。首先，乘法、取模和除法运算符按从左到右的顺序依次执行（即从左到右顺序关联），这是由于它们的优先级要高于加法和减法。接着执行的是加法和减法运算符——也是按从左到右的顺序。一旦表达式求值结束，Python 就将结果指派给变量 `z`。

为了更好地理解运算符优先级规则，我们来看看一个二次多项式是如何求值的：

`y = a * x ** 2 + b * x + c`
 ⑥ ② ① ④ ③ ⑤

圆圈内的数字指明 Python 执行运算时的顺序。

假定变量 `a`, `b`, `c` 和 `x` 像下面这样初始化：`a = 2`, `b = 3`, `c = 7` 和 `x = 5`，就可用图 2.17 来展示在上述二次多项式中运算符的应用顺序。

以上赋值语句也可用可有可无的圆括号来强调语句的清晰性，如下所示：

`y = (a * (x ** 2)) + (b * x) + c`

良好编程习惯 2.7 和在代数中一样，可在表达式中添加原本不需要的圆括号，使其更清晰。这些括号叫做冗余括号。冗余括号通常用于分组大型表达式中的各个子表达式，使表达式更清晰。将一条长的语句分解成一系列较短的、较简单的语句，有助于使语句更清晰。

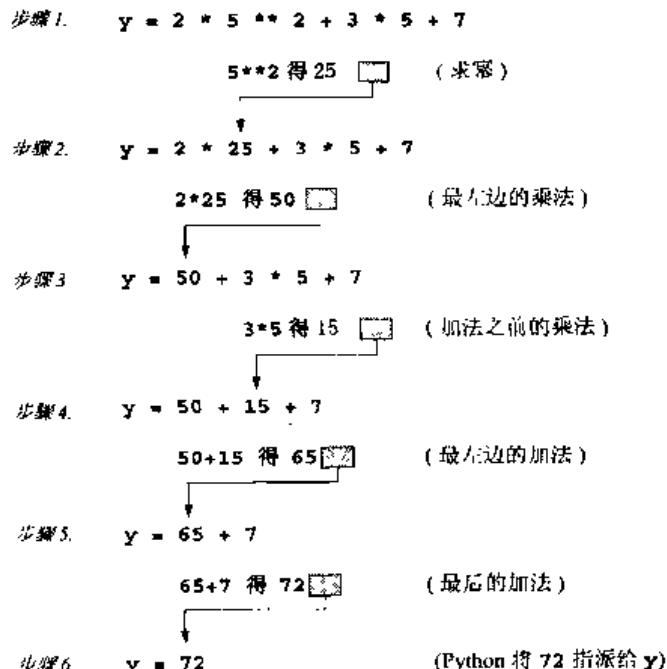


图 2.17 二次多项式求值顺序

2.7 字符串格式化

前面讨论了数值的问题，现在来看看字符串。和其他流行的程序语言不同，Python 将字符串作为内建数据类型提供，这使得 Python 程序可执行非常强大的、基于文本的处理。我们已学习了如何将文本放在双引号内（"）以创建一个字符串。Python 字符串还可采用其他许多方式创建，如图 2.18 所示。

```

1 # Fig. 2.18: fig02_18.py
2 # Creating strings and using quote characters in strings.
3
4 print "This is a string with \"double quotes.\""
5 print 'This is another string with "double quotes."'
6 print 'This is a string with \'single quotes.\'''
7 print "This is another string with 'single quotes.'"
8 print """This string has "double quotes" and 'single quotes'."""
9     You can even do multiple lines."""
10 print '''This string also has "double" and 'single' quotes.'''

```

```

This is a string with "double quotes."
This is another string with "double quotes."
This is a string with 'single quotes.'
This is another string with 'single quotes.'
This string has "double quotes" and 'single quotes'.
    You can even do multiple lines.
This string also has "double" and 'single' quotes.

```

图 2.18 创建 Python 字符串

第 4 行用大家熟悉的双引号字符来创建一个字符串。如果希望在屏幕上打印双引号，必须使用双引号字符的转义序列，即\"，不能只使用一个双引号。

字符串还可用单引号（'）创建，如第 5 行所示。如希望在用单引号创建的字符串中使用双引号字符，就不需要转义字符。类似地，要想在用双引号创建的字符串中使用单引号字符，也不必使用转义字符（第 7 行）。然而，如果想在使用单引号创建的字符串中使用单引号字符（第 6 行），就必须使用转义序列\'。

Python 还支持三引号字符串，如第 8~10 行所示。三引号字符串尤其适合在字符串中输出特殊字符，

比如引号字符。在三引号字符串中，单或双引号字符不必添加转义字符。另外，也可将三引号字符用于大型的文本块，因为三引号字符串可跨过多行。本书中，示例程序要为 Web 输出大的文本块时，就会使用三引号字符串。

Python 字符串支持简单但功能强大的输出格式化。可采用几种方式对输出的字符串进行格式化：

1. 为浮点值取指定的小数位。
2. 使用指数（科学）计数法表示浮点数。
3. 使一列数字的小数点对齐。
4. 使输出右对齐或左对齐。
5. 在一行输出的指定位置插入字符或字符串。
6. 用固定字段宽度和精度显示所有类型的数据。

图 2.19 的程序演示了基本的字符串格式化功能。第 4~7 行演示了如何在字符串中表示整数。第 5 行直接显示变量 `integerValue` 的值，不进行任何格式化。格式化运算符 `%` 可在字符串中插入一个变量值（第 6 行）。运算符左边的值是一个字符串，它包含一个或多个“转换指示符”（值在字符串中的占位符）。每个转换指示符都以百分号（%）开头——不要把它同格式化运算符 `%` 弄混了——并以一个“转换指示符符号”结尾。转换指示符符号 `d` 表明我们希望在当前字符串的指定位置放入一个整数。图 2.20 总结了用于字符串格式化的几个转换指示符符号。

格式化运算符 `%` 的右边，指定了要用什么来替换字符串中的占位符。在第 6 行，我们指定要用值 `integerValue` 来替换字符串中的占位符 `%d`。第 7 行将变量 `integerValue` 的值转换成十六进制形式后插入字符串。

```

1 # Fig. 2.19: fig02_19.py
2 # String formatting.
3
4 integerValue = 4237
5 print "Integer ", integerValue
6 print "Decimal integer %d" % integerValue
7 print "Hexadecimal integer %x\n" % integerValue
8
9 floatValue = 123456.789
10 print "Float", floatValue
11 print "Default float %f" % floatValue
12 print "Default exponential %e\n" % floatValue
13
14 print "Right justify integer (%8d)" % integerValue
15 print "Left justify integer (%-8d)\n" % integerValue
16
17 stringValue = "String formatting"
18 print "Force eight digits in integer %.8d" % integerValue
19 print "Five digits after decimal in float %.5f" % floatValue
20 print "Fifteen and five characters allowed in string:"
21 print "%(.15s) (%.5s)" % (stringValue, stringValue)

```

```

Integer 4237
Decimal integer 4237
Hexadecimal integer 108d

Float 123456.789
Default float 123456.789000
Default exponential 1.234568e+005

Right justify integer ( 4237)
Left justify \integer: (4237  )

Force eight digits in integer 00004237
Five digits after decimal in float 123456.78900
Fifteen and five characters allowed in string:
(String formatt) (Strin)

```

图 2.19 字符串格式化运算符%

转换指示符符号	含义
c	单个字符（即长度为 1 的字符串），或者一个 ASCII 字符的整数表示
s	字符串，或者要转换成字符串的一个值
d	有符号（正负号）的整数
u	无符号十进制整数
o	无符号八进制整数
x	无符号十六进制整数（a 到 f 的数位采取小写形式）
X	无符号十六进制整数（A 到 F 的数位采取小写形式）
f	浮点数
e, E	浮点数（使用科学计数法）
g, G	浮点数（采用最低有效数位）

图 2.20 字符串格式化字符

第 9~12 行演示了如何在字符串中插入浮点值。转换指示符充当一个浮点值的占位符（第 11 行）。在格式化运算符 % 的右边，将变量 floatValue 用作要显示的值。e 转换指示符是一个采用指数计数法的浮点值的占位符。指数计数法等价于数学中的科学计数法，只是前者在计算机中使用。例如，150.4582 可采用科学计数法表示为 1.504582×10^2 ，计算机的指数计数法则将其表示成 1.504582E+002，即 1.504582 乘以 10 的 2 次方 (E+002)。E 是 Exponent (指数) 的简称。

第 14~15 行演示了如何用“字段宽度”来格式化字符串。字段宽度是指一个字段的最小长度，值将在这个字段中打印。如字段宽度大于要打印的值的宽度，数据通常在字段内“右对齐”。要使用字段宽度，必须在百分号和转换指示符之间插入一个整数以指定字段宽度。第 14 行在宽度为 8 的一个字段中右对齐显示变量 integerValue 的值。要想左对齐，需要将字段宽度设为负数（第 15 行）。

第 17~21 行演示了如何用精度来格式化字符串。对于不同数据类型，“精度”有不同的含义。如果用于整数转换指示符，精度指定的是要打印的最小位数。如打印值包含的位数少于指定的精度，就为打印的值加上前置零，直到总位数等于精度。要想使用精度，请在百分号和转换指示符符号之间插入一个小数点 (.)，并后跟一个代表精度的整数。例如，第 18 行在打印变量 integerValue 的值时，采用了 8 位精度。

将精度用于一个浮点转换指示符时，精度就是小数点后出现的位数。第 19 行在打印变量 floatValue 时，采用了 5 位精度。

用于字符串转换指示符时，精度是指从字符串中提取并显示的最大字符数。第 21 行打印两次变量 stringValue 的值，一次精度为 15，另一次为 5。注意，转换指示符包含在圆括号中。若格式化运算符 % 左边的字符串包含多个转换指示符，右边的值就必须是一个用逗号分隔的值序列。该序列包含在圆括号中，而且值的数量必须与含有转换指示符的字符串的数量一致。Python 按从左到右的顺序构造字符串——用圆括号中的值逐个替换格式化字符。

Python 字符串还通过“字符串方法”，支持更强大的字符串格式功能，详情参见第 13 章。

2.8 做出决策：相等运算符和关系运算符

针对 Python 的 if 结构，本节要介绍一个简单版本，它允许程序根据一些条件是否成立，从而做出决策。如条件成立（条件为真），if 结构主体中的语句就会执行。如条件不成立（条件为假），主体语句就不执行。稍后会介绍一个例子。

if 结构中的条件可用如图 2.21 所示的相等运算符和关系运算符构建。关系运算符全都在同一个优先级上，并从左到右顺序关联。所有相等运算符也在同一个优先级上，但其优先级低于关系运算符。相等运算符也从左到右顺序关联。

常见编程错误 2.2 `==`、`!=`、`>=`和`<=`这几个运算符的两个符号之间出现空格，会造成语法错误。

常见编程错误 2.3 `!=`、`<`、`>`和`<=`这几个运算符中，假如两个字符的顺序弄反了（分别写成`!=`、`><`、`=>`和`=<`），会造成语法错误。

常见编程错误 2.4 切不可将相等运算符“`==`”同赋值运算符“`=`”弄混了。其实按正统逻辑，在读的时候，相等运算符才应该读成“...等于...”。赋值运算符则应该读成“...获得...”、“获得...的值”或者“被赋值为...”。有人喜欢把相等运算符读成“等于等于”。在 Python 中，如果在条件语句中使用了错误的赋值符号，会造成语法错误。

标准的代数相等运算符或关系运算符	Python 相等运算符或关系运算符	Python 条件示例	Python 条件的含义
关系运算符			
<code>></code>	<code>></code>	<code>x > y</code>	<code>x 大于 y</code>
<code><</code>	<code><</code>	<code>x < y</code>	<code>x 小于 y</code>
<code>≥</code>	<code>≥</code>	<code>x ≥ y</code>	<code>x 大于或等于 y</code>
<code>≤</code>	<code>≤</code>	<code>x ≤ y</code>	<code>x 小于或等于 y</code>
相等运算符			
<code>=</code>	<code>==</code>	<code>x == y</code>	<code>x 等于 y</code>
<code>≠</code>	<code>!=, <></code>	<code>x != y</code> <code>x <> y</code>	<code>x 不等于 y</code>

图 2.21 相等运算符和关系运算符

下例用 6 个 if 结构来比较两个用户输入的数字。满足任何一个 if 条件，就会执行与 if 对应的 print 语句。用户要输入两个值，程序将其转换成整数，并指派给变量 number1 和 number2。然后，程序比较这两个数字并显示比较结果。图 2.22 展示了示例程序以及一些示范结果。

```

1 # Fig. 2.22: fig02_22.py
2 # Compare integers using if structures, relational operators
3 # and equality operators.
4
5 print "Enter two integers, and I will tell you"
6 print "the relationships they satisfy."
7
8 # read first string and convert to integer
9 number1 = raw_input( "Please enter first integer: " )
10 number1 = int( number1 )
11
12 # read second string and convert to integer
13 number2 = raw_input( "Please enter second integer: " )
14 number2 = int( number2 )
15
16 if number1 == number2:
17     print "%d is equal to %d" % ( number1, number2 )
18
19 if number1 != number2:
20     print "%d is not equal to %d" % ( number1, number2 )
21
22 if number1 < number2:
23     print "%d is less than %d" % ( number1, number2 )
24
25 if number1 > number2:
26     print "%d is greater than %d" % ( number1, number2 )
27
28 if number1 ≤ number2:

```

```

29     print "%d is less than or equal to %d" % ( number1, number2 )
30
31 if number1 >= number2:
32     print "%d is greater than or equal to %d" % ( number1, number2 )

```

```

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 37
Please enter second integer: 42
37 is not equal to 42
37 is less than 42
37 is less than or equal to 42

```

```

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 7
Please enter second integer: 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7

```

```

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 54
Please enter second integer: 17
54 is not equal to 17
54 is greater than 17
54 is greater than or equal to 17

```

图 2.22 用相等运算符和关系运算符判断逻辑关系

程序使用 Python 函数 `raw_input` 和 `int` 输入两个整数（第 8~14 行）。首先获得变量 `number1` 的值，然后获得 `number2` 的值。

第 16~17 行的 `if` 结构比较变量 `number1` 和 `number2` 的值，测试其相等性。如果两个值相等，就显示一行文本，指出数字是相等的（第 17 行）。如果满足一个或多个从第 19 行、第 22 行、第 25 行、第 28 行和第 31 行开始的 `if` 结构的条件，相应的 `print` 语句就会显示一行文本。

每个 `if` 结构都包括单词 `if`、要测试的条件以及一个冒号 (`:`)。`if` 结构还包含一个主体（称为 `suite`）。图 2.22 中的每个 `if` 结构主体都只包含一个语句，而且每个主体都是缩进的。一些语言（如 C++、Java 和 C#）用花括号 {} 标明 `if` 结构主体；Python 则用“缩进”来达到这个目的。下一节会详细讨论 Python 的缩进问题。

常见编程错误 2.5 忘记在 `if` 结构中插入冒号 (`:`) 是语法错误。

常见编程错误 2.6 忘记对 `if` 结构的主体进行缩进（缩排）是语法错误。

良好编程习惯 2.8 事先建立一个约定：设置您喜欢的缩进量，然后始终贯彻这一约定。虽然按 Tab 键可生成缩进，但制表位的长度在不同系统上是不同的。建议将 3 个空格定为一个缩进级别。

Python 对语法的评估取决于空距；因此，如果空距的用法不统一，可能会造成语法错误。例如，将语句分解为多行可能导致语法错误。如语句过长，可使用续行字符将其分为多行。有的 Python 解释器用“...”来注明一个延续的行。图 2.23 中的交互式会话演示了续行字符的用法。

```

Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print 1 +
File "<string>", line 1
    print 1 +
          ^
SyntaxError: invalid syntax
>>> print 1 + \

```

```
... 2
3
>>>
```

图 2.23 续行字符 (\)

良好编程习惯 2.9 可用续行字符\将长语句分为几行。如一个语句必须分解成多行，请选择有意义的断点，比如在 print 语句的逗号之后，或者在一个较长表达式的运算符之后。

图 2.24 总结了本章介绍的运算符的优先级。各运算符根据优先级，从上到下降序排列。注意，除了求幂运算符之外，其他所有运算符都从左到右顺序关联。

运算符	顺序关联性	类型
()	从左到右	圆括号
**	从右到左	求幂
* / // %	从左到右	乘
+ -	从左到右	加
< <= > >=	从左到右	关系
== != <>	从左到右	相等

图 2.24 已讨论过的运算符的优先级和顺序关联性

测试和调试提示 2.3 如果一个表达式里包含许多运算符，请务必参考运算符优先级表，核实表达式中的运算符按自己希望的顺序执行。如表达式过于复杂以至于无法确定顺序，不妨将表达式分割为几个小语句，或干脆用圆括号强行规定顺序——在代数表达式中也可采用一样的做法。注意，某些运算符（比如求幂运算符**)是按右到左顺序关联的，而非从左到右。

2.9 缩进

Python 利用代码文本的缩进来定界（区分）代码的不同区域。其他程序语言则通常使用花括号来定界不同区域。在 Python 中，一个“suite”是指同控制结构的主体对应的一个代码区域。第 3 章将介绍“block”（块或代码块）的概念，它通常就是函数的主体。Python 程序员要自行决定一个 suite 或 block 的缩进量。另外，suite 或 block 中每个语句的缩进量都必须保持一致。Python 通过缩进量的变化来识别新的 suite 或 block。

常见编程错误 2.7 如果一个区域包含的代码行没有统一进行缩进，Python 解释器会认为那些行从属于其他区域，因而造成语法或逻辑错误。

图 2.25 对图 2.22 中的程序进行了修改，演示了不正确缩进的后果。第 21~22 行显示的是 if 结构的错误缩进。尽管程序不会报错，但它确实跳过了一个相等运算符。第 21 行的语句：

```
if number != number:
```

只有在第 16 行的 if number1 == number2 语句执行的前提下才会执行。本例中，第 21 行的 if 语句永远不会执行，因为两个相等的数字肯定不会不相等（2 不可能不等于 2）。所以，图 2.25 中的输出本应指出 1 is not equal to 2（1 不等于 2），但因上述错误无法显示。

```
1 # Fig. 2.25: fig02_25.py
2 # Using if statements, relational operators and equality
3 # operators to show improper indentation.
4
5 print "Enter two integers, and I will tell you"
6 print "the relationships they satisfy."
```

```

7
8 # read first string and convert to integer
9 number1 = raw_input( "Please enter first integer: " )
10 number1 = int( number1 )
11
12 # read second string and convert to integer
13 number2 = raw_input( "Please enter second integer: " )
14 number2 = int( number2 )
15
16 if number1 == number2:
17     print "%d is equal to %d" % ( number1, number2 )
18
19 # improper indentation causes this if statement to execute only
20 # when the above if statement executes
21 if number1 != number2:
22     print "%d is not equal to %d" % ( number1, number2 )
23
24 if number1 < number2:
25     print "%d is less than %d" % ( number1, number2 )
26
27 if number1 > number2:
28     print "%d is greater than %d" % ( number1, number2 )
29
30 if number1 <= number2:
31     print "%d is less than or equal to %d" % ( number1, number2 )
32
33 if number1 >= number2:
34     print "%d is greater than or equal to %d" % ( number1, number2 )

```

```

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 1
Please enter second integer: 2
1 is less than 2
1 is less than or equal to 2

```

图 2.25 用于演示错误缩进的 if 语句

测试和调试提示 2.4 为避免难以察觉的错误，务必在 Python 程序中采用统一和正确的缩进。

2.10 对象思想：对象技术简介

本书前 6 章的重点是结构化编程的“常规”方法论，这是因为即将构建的对象会包含一部分结构化程序组件。本节提前介绍面向对象的概念。通过本节的学习，您能理解面向对象是一种思考这个世界和编写计算机程序的一种自然方式。

首先从一些关键概念和术语开始“面向对象之旅”。观察一下身旁的现实世界，对象无处不在——人、动物、植物、汽车、飞机、建筑物、计算机等等，一切都是对象。人们通过对对象来思考问题。每一个人都有非凡的抽象能力，可将屏幕上显示的一幅图像看成人、飞机、树和山这样的对象，而非看成单独的颜色点。如果愿意，我们可以看到沙滩而不是沙子，看到森林而不是树木，看到房子而不是砖块。

人们习惯将对象划分为两大类——动物和静物。其中，动物在某种程度上是“有生命的”。它们四处运动，并做一些事情。静物，比如毛巾等，则表面上什么事情也不做。它们只是某种静态的物品。然而，所有这些对象都有一些东西是共通的。它们都有自己的一些属性，例如大小、形状、颜色、重量等等；而且都在表现出某种行为，例如球会滚动、弹跳、膨胀和收缩，婴儿会哭、睡、爬、走和眨眼睛，车会加速、刹车和转弯，毛巾会吸水等等。

人们通过研究其属性和观察其行为，从而了解对象。不同的对象可能具有相同的属性，并可表现出类似的行为。例如，可以比较一下婴儿和成人，人和猩猩等等。对于轿车、卡车、小童车和溜冰鞋等来说，它们其实也有许多共通之处。

而向对象编程（OOP）以软件形式对现实世界对象进行建模。它利用了类关系；在这种关系中，特

定类（比如一个汽车类）的所有对象都具有相同的特征。它还利用了继承关系，甚至利用了多重继承关系。在这种关系中，通过吸收现有类的特征建立新的对象类，同时可能添加了自己独有的一些特征。“敞篷车”类的一个对象肯定具有一个更常规的“汽车”类的特征，但敞篷车的车篷还可展开和折叠，这是普通汽车不具备的。

面向对象的编程还为我们提供了一种更加自然和直观的方式来对待编程过程。换言之，编程过程就是对现实对象及其属性 / 行为进行建模的一种过程。OOP 还可为对象之间的通信进行建模。就和人们向同伴发送消息一样（例如军官命令士兵立正），对象之间也通过消息来沟通。

OOP 将数据（属性）和函数（行为）封装到一种名为“对象”的包中；一个对象的数据同函数紧密联系在一起。对象具有“信息隐藏”的特点。也就是说，尽管一个对象可能知道如何通过经良好定义的接口与另一个对象通信，但一个对象通常不允许知道其他对象内部是如何实现的——实现的细节被隐藏在对象内部。这是非常合理的，我们完全能很好地驾驶一辆汽车，而不必了解发动机、传动机构和排气系统的内部工作原理。以后，我们会解释信息隐藏对于保障“良好的软件工程”为何如此重要。

在 C 和其他过程式程序语言中，编程通常是“面向行动”的；但在 Python 中，编程是“面向对象”的（在理想情况下）。在过程式语言中，基本编程单元是“函数”；而在面向对象语言中，基本编程单元是“类”，最终通过它实例化（即“创建”的一种更好听的说法）出对象。Python 类包含了函数（用于实现类的行为）和数据（用于实现类的属性）。

采用过程式编程，程序员可把重点放在写函数上。用于执行一些任务的行动被组合成函数，不同的函数进一步组合，即构成程序。在过程式编程中，数据肯定是重要的，但数据的主要用途是为函数执行的行动提供支持。在一份系统规格说明（描述应用程序所提供的服务的一份文档）中，那些动词帮助过 程式程序员确定并设计一系列协同工作的函数，它们用于实现整个系统。

采用面向对象编程，程序员则把重点放在创建他们自己的用户自定义类型上，即“类”。每个类都包含数据以及一系列函数，函数用于对数据进行处理。一个类的数据组件称为数据成员或属性；一个类的函数组件则称为方法（在其他面向对象编程语言中，也可能称为“数据成员”）。面向对象编程侧重于类，而不是函数。在一份系统规格说明中，那些名词帮助面向对象程序员确定并设计一系列类，以便通过它们创建一系列协同工作的对象，这些对象用于实现整个系统。

类之于对象，犹如蓝图之于房屋。可依据蓝图建造许多房屋，也可依据一个类创建许多对象。一个类可与其他类建立关系。例如，假定一家银行采取了面向对象的设计，那么 BankTeller（出纳员）类需要同 Customer（客户）类建立关系。这种关系称为“关联”。

我们以后会体验到，一旦软件被打包成类，这些类便可重复用于未来的软件系统。一组相关的类通常打包成可重复使用的“组件”或“模块”。房地产经纪人会告诉他们的客户，影响房地产价格的三大因素是“地段，地段，还是地段”。类似地，我们也认为影响软件开发前途的三大因素是“重用，重用，还是重用”。

事实上，利用对象技术，今后大多数软件都可通过合成一系列“标准化、可互换零件”而构建，这些零件称为“组件”。本书将教您如何“创建宝贵的类”，以达到重用、重用、再重用之目的。您创建的每个新类都可能成为一项价值无限的软件资产，您和其他程序员可用它加快未来软件开发的速度，并改进其质量。类的重用，大有潜力可挖。

本章介绍了 Python 的大量重要特性，包括在屏幕上打印数据，从键盘输入数据，执行计算和做出决策等。第 3 章介绍“结构化编程”时，会频繁用到这些技术。届时，您将学习如何指定和更改语句执行顺序——该顺序称为“控制流程”。本章还介绍了面向对象的基本概念和术语。第 7~9 章将继续讨论面向对象编程。

第3章 控制结构

学习目标

- 理解基本的求解方法
- 会利用“自上而下求精法”开发自己的算法
- 会使用 if、if/else 和 if/elif/else 结构选择恰当的行动
- 会用 while 和 for 重复结构在一个程序中重复执行语句
- 理解由计数器控制的重复，以及由哨兵值控制的重复
- 会使用增量赋值符号和逻辑运算符
- 会使用 break 和 continue 程序控制语句

3.1 概述

为解决特定的问题而写一个程序之前，首先有必要透彻理解该问题，并精心计划好每一步操作，最终圆满解决问题。写程序时，理解基本构建单元的类型，并严格遵守已证明行之有效的程序构建原则，同样不可忽视。本章将讨论所有这些问题，向大家展示结构化编程的理论及原则。注意，这里描述的技术适用于大多数高级程序语言。从第 7 章开始学习面向对象编程时，我们会利用本章介绍的控制结构来构建和操纵对象。

3.2 算法

任何计算问题都可通过按指定顺序采取一系列行动得到解决。所谓“算法”，是指解决一个问题的“过程”，它有两方面的含义：

1. 要采取的“行动”
2. 采取这些“行动”的顺序

下例演示了正确指定行动顺序的重要性。

假定要设计一个“起床上班算法”，模拟一名职员早上起床并去上班的过程。他的行动是：① 起床；② 脱下睡衣；③ 洗个淋浴；④ 穿好衣服；⑤ 吃早餐；⑥ 开车上班。只有按以上顺序行动，职员才能获得最高效率。

如果调换一下顺序，变成：① 起床；② 脱下睡衣；③ 穿好衣服；④ 洗个淋浴；⑤ 吃早餐；⑥ 开车上班，可怜的职员只好穿着湿淋淋的衣服去上班了。

在计算机程序中，指定语句执行顺序的操作称为“程序控制”。本章将探讨 Python 的程序控制。

3.3 伪代码

伪代码（Pseudocode）是一种虚的、非正式的语言，目的是帮助程序员设计算法。伪代码包括对“可执行语句”的描述——一旦从伪代码转换成 Python 语句，并开始运行，这些语句就可以执行。利用这里提供的伪代码，可开发出实际、有效的算法，并可方便地转换成 Python 程序。伪代码就像我们的日常语言：既方便，又好用——尽管它并不是真正的计算机程序语言。

伪代码程序无法在计算机上执行。它们唯一的用处是，在使用 Python 等程序语言编写正式代码之前，

可帮助程序员“计划”好整个程序。本章用几个例子演示了如何通过伪代码高效率地开发 Python 程序。

软件工程知识 3.1 程序设计过程中，常用伪代码来“思考”一个程序，再将伪代码程序转换成 Python 程序。

我们展示的伪代码程序纯粹由字符构成，所以程序员可用一个编辑器程序方便地输入这些代码。计算机可根据需要显示伪代码程序的最新拷贝。对精心设计的伪代码程序来讲，它们可轻松转换成相应的 Python 程序。许多时候，都只需用对应的 Python 语句替换伪代码语句。

3.4 控制结构

通常，程序中的语句是按原先书写的顺序执行的。这称为“顺序执行”。然而，许多 Python 语句允许程序员自行控制接着要执行的语句，该语句可能是、也可能不是顺序的“下一条”语句。这称为“转交控制权”，具体由 Python 的“控制结构”来实现。本节要讨论控制结构开发的背景知识，并介绍 Python 用于在程序中转交控制权的特定工具。

20世纪60年代，大量事实证明，假如胡乱转交控制权，会使软件开发变成一件令人非常头疼的事情。于是，人们将此归咎于 goto 语句（在包括 C 和 Basic 在内的几种程序语言中使用）。利用该语句，程序员可将控制权转交给一个程序里的几乎任意位置。因此，结构化编程的概念问世时，就郑重宣布自己是“免 goto”的。

Bohm 和 Jacopini 的研究证实，一个程序完全可以不必使用任何 goto 语句。大势所趋下，程序员们也逐渐改变了自己对 goto 的“偏爱”，变成“较少 goto 编程”。直到 20 世纪 70 年代，程序员们才开始认真看待“结构化编程”。结果令人振奋。采用结构化编程后，程序员们普遍的反映是，一个软件项目的各个方面（无论开发时间、速度还是成本），都得到了有效改善。之所以取得这些成功，是由于结构化程序条理更清晰、更易调试、更易修改，而且一开始就能避免大多数编程错误。

Bohm 和 Jacopini 的研究结果表明，事实上只需 3 种控制结构便可写出所有程序。这 3 种控制结构包括：顺序结构、选择结构以及重复结构。其中，顺序结构是 Python 内建的。除非特别声明，否则计算机都会顺序执行 Python 语句。图 3.1 的流程图演示了一个典型的顺序结构，两个计算将依序执行。“流程图”（Flowchart）是对全部或部分算法的一种图形化表示。

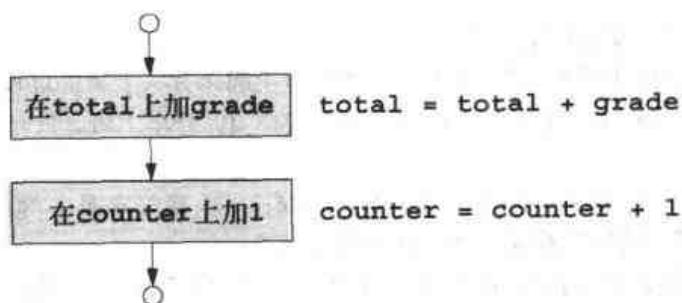


图 3.1 顺序结构流程图

流程图往往会使用一些具有特殊意义的符号，比如矩形、菱形、椭圆以及小圆圈等等。这些符号通过一些箭头线（称为流线，即 Flowline）连接起来，它指明算法采取行动的顺序。类似于伪代码，流程图在开发和表示算法时特别有用。尽管大多数程序员最喜欢的还是伪代码，但流程图明确指出了控制结构的运作机制。读者应仔细比较每种控制结构的伪代码及流程图表示方法。

请观察图 3.1 的顺序结构流程图，它用矩形符号（或称行动符号）指出每种类型的行动，包括计算或输入 / 输出操作等。图中的流线指出行动的执行顺序。首先，将 grade 加到 total 上，再把 1 加到 counter 上。Python 允许在顺序结构中采取数量不限的行动。在可以放入单个行动的任何位置，都可顺序放入几个行动。

要用流程图表示一个完整的算法，应将椭圆符号作为流程图的第一个符号，并在其中写上“开始”；最后一个符号也是一个椭圆符号，并在其中写上“结束”。但如图3.1所示，如果绘制的只是算法的一部分，头尾的椭圆符号都可省略，分别换成一个小圆圈符号（或称接头符号）。

在所有流程图符号中，最重要的或许是菱形符号（或称决策符号），它指出要在那个位置做出一项决定。下一节将更深入地讨论菱形符号。这里展示的伪代码有助于开发出能直接转换成结构化Python程序的算法。

Python提供3种类型的选择结构：if, if/else和if/elif/else。本章要分别讨论它们。假如某个条件成立（条件为true），if选择结构会执行（选择）一项行动；假如不成立（条件为false），则跳过这项行动。如条件成立(true)，if/else选择结构会采取一项行动；条件不成立(false)，则采取另一项行动。if/elif/else选择结构则在多个不同的行动中选择一个，具体取决于几个条件成立还是不成立。

if选择结构是一种单选结构——要么选择、要么忽略一项行动；if/else是一种双选结构——在两项不同的行动中做出选择；if/elif/else则是一种多选结构——在许多不同的行动中选择实际要执行的。

Python提供2种类型的重复结构：while和for。注意，if, else, switch, while, do和for都是Python关键字。这些关键字是该语言为自己保留的，用于实现控制结构这样的Python特性。千万不要将关键字用作标识符（比如变量名）。图3.2总结了所有Python关键字。^①

Python 关键字							
and	continue	else	for	import	not	raise	
assert	def	except	from	in	or	return	
break	del	exec	global	is	pass	try	
class	elif	finally	if	lambda	print	while	

图3.2 Python关键字

常见编程错误3.1 将关键字用作标识符是语法错误。

单入/单出控制结构便于构建程序。只需将一个控制结构的出口同下一个控制结构的入口连接到一起，便可将不同控制结构有机联系到一起。这类似于小孩子搭积木，所以，我们把这种方法称为“控制结构堆叠”（Control-Structure Stacking）。另外还有一种方法叫做“控制结构嵌套”（Control-Structure Nesting），它也能连接控制结构，稍后还会对此详加讨论。

软件工程知识3.2 所有Python程序都可基于前述6类控制结构来创建（顺序、if、if/else、if/elif/else、while和for）。不同控制结构可采用2种方法连接，即控制结构堆叠和嵌套。

3.5 if选择结构

利用选择结构，可在大量候选行动中挑选一个。例如，假定通过一次考试的分数线是60分，那么伪代码应该像下面这样：

```
假如 (If) 学生成绩大于或等于 60
    打印 "Passed"
```

在上述代码中，它判断的是“学生成绩大于或等于60”这个条件为true(真)还是为false(假)。如条件为true，则打印一条“Passed”，再顺序“执行”下一条伪代码语句（记住伪代码并不是真正的程序语言）。如条件为false，则忽略打印语句，并顺序执行下一条伪代码语句。选择结构第2行进行了缩进处理。虽然并不一定要缩进（对伪代码而言），但建议您这么做，因为它强调了结构化程序固有的层次结构。

^① 请访问Python网站(www.python.org)，查看这些关键字的一个列表，并注意避免将它们用作标识符。

将伪代码转换成 Python 代码时，则必须进行缩进。

在 Python 中，上述伪代码 if 语句可写成：

```
if grade >= 60:  
    print "Passed"
```

注意 Python 代码同伪代码是严格对应的。正是因为具有这种关系，伪代码才能成为强有力的一种开发工具。if 结构主体中的语句输出字符串“Passed”。

图 3.3 的流程图向大家揭示了单选 if 结构和菱形（决策）符号。该符号包含一个表达式（比如条件），它要么为 true，要么为 false。从菱形符号必须引出两条流线：一条指出表达式为 true 时的程序执行方向，另一条指出表达式为 false 时的程序执行方向。通过第 2 章的学习，您知道可根据包含关系或相等运算符的条件做出决策。实际上，一项决策可基于任何表达式——如表达式的值为零，就把它当作 false；如表达式的值不是零（非零），就把它当作 true。

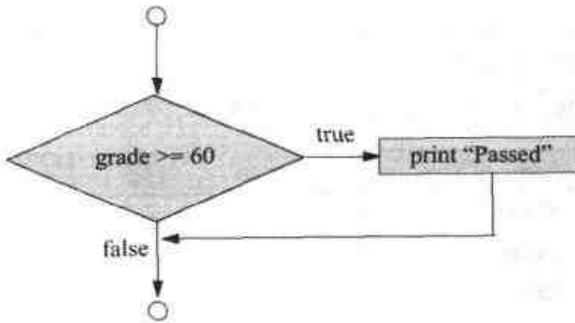


图 3.3 if 单选结构流程图

注意，if 结构是“单入/单出”结构。在其他控制结构流程图中，除小圆圈和流线外，还可包含矩形和菱形——前者指出行动，后者指出决策。这种类型的流程图强调了“行动 / 决策编程模型”。

想象有 6 个柜子，每个都包含了 6 种控制结构中的一种。这些控制结构是空的。在矩形和菱形符号中，均不包含任何内容。然后，程序员的任务是在惟一可选的两种方法中挑选一种（堆叠或嵌套），将这些控制结构合并到一起。然后，依据计划好的算法，用适当方式在其中填充行动和决策。稍后会讲解编写行动及决策内容的各种方式。

3.6 if/else 和 if/elif/else 选择结构

只有条件为 true，if 选择结构才会执行指定的行动；否则会跳过行动。利用 if/else（假如 / 否则）选择结构，程序员可针对条件成立与不成立这两种情况，分别指定一项行动。例如下面的伪代码语句：

```
假如学生成绩大于或等于 60  
    打印 "Passed"  
否则  
    打印 "Failed"
```

上例中，假如学生成绩大于或等于 60 分，则打印“Passed”；如小于 60 分，则打印“Failed”。不管哪种情况，都会在完成打印操作后，顺序执行下一条伪代码语句。注意在伪代码中，else（否则）的主体代码也进行了缩进。控制结构缩进的主体称为一个 suite。记住，在程序中应始终贯彻选择的缩进约定。缩进不正确，Python 就无法正确执行代码；另外，如程序不遵守统一的间距约定，也会导致代码难以阅读。

良好编程习惯 3.1 如果同时有几级缩进，每个 suite 都必须缩进。相同级别的不同 suite 不必具有相同的缩进量，但这是一种良好的编程习惯。

上述伪代码 *if/else* 结构可用 Python 写成：

```
if grade >= 60:
    print "Passed"
else:
    print "Failed"
```

常见编程错误 3.2 没有对从属于 *if* 或 *else* 这两种 suite 的全部语句进行正确缩进 b，会造成语法错误。

图 3.4 的流程图演示了 *if/else* 结构中的控制流程。再次重申，除小圆圈和箭头（流线）之外，流程图里还使用了矩形（标明行动）和菱形（标明决策）。后文将继续强调这种行动 / 决策计算模型。同样地，可想象一个柜子里包含了空的双选结构。程序员的任务是采用堆叠和嵌套方式，将这些选择结构同算法所需的其他控制结构合并到一起，再使用适合算法的行动与决策来填充矩形和菱形。

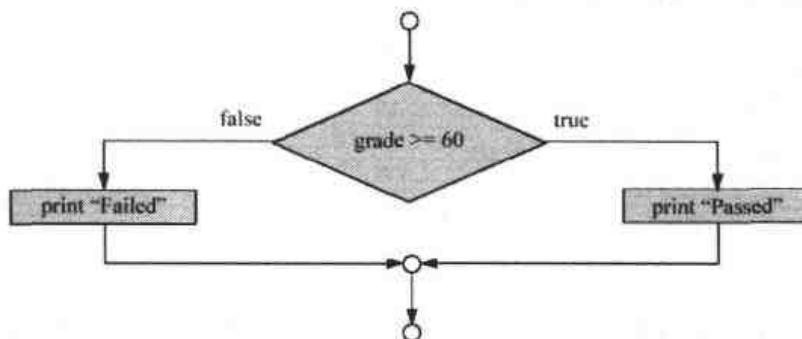


图 3.4 *if/else* 双选结构流程图

嵌套 *if/else* 结构检测多个条件，它的做法是将一个 *if/else* 选择结构放入另一个 *if/else* 选择结构。例如，下列伪代码语句为大于或等于 90 分的成绩打印 A；为 80~90 分之间的成绩打印 B；为 70~79 分之间的成绩打印 C；为 60~69 分之间的成绩打印 D；为其他成绩打印 F。

```

假如学生成绩大于或等于 90
    打印 "A"
否则
    假如学生成绩大于或等于 80
        打印 "B"
    否则
        假如学生成绩大于或等于 70
            打印 "C"
        否则
            假如学生成绩大于或等于 60
                打印 "D"
            否则
                打印 "F"
```

上述伪代码用 Python 写成，便是：

```
if grade >= 90:
    print "A"
else:
    if grade >= 80:
        print "B"
    else:
        if grade >= 70 :
            print "C"
```

```

else:
    if grade >= 60:
        print "D"
    else:
        print "F"

```

如 grade 大于或等于 90，前 4 个条件都为 true，但只有第一次检测后的 print 语句才会执行。之后，外层 if/else 语句的 else 部分会被跳过。

性能提示 3.1 嵌套 if/else 结构比一系列单选 if 结构快，因为只要有一个条件满足，其余测试就会终止。

性能提示 3.2 在嵌套 if/else 结构中，把最可能成立的条件放在该嵌套 if / else 结构的开始处。和把不常见的条件放在开始处相比，采用这种做法后，嵌套 if/else 结构运行得更快。

许多 Python 程序员都喜欢将上述 if 结构写成：

```

if grade >= 90:
    print "A"
elif grade >= 80:
    print "B"
elif grade >= 70:
    print "C"
elif grade >= 60:
    print "D"
else:
    print "F"

```

这样可将双选 if/else 结构替换为多选 if/elif/else 结构。两种形式效果一样。后者之所以流行，是因其避免了代码过分朝右边缩进。这样的缩进会在一行上留下较少的空间，造成不得不将一行分解成多行，从而降低了程序的可读性。

每个 elif 都可包含一项或多项行动。图 3.5 的流程图显示常规的 if/elif/else 多选结构。该流程图表明，执行了 if 或 elif 语句后，控制权会立即退出 if/elif/else 结构。再次重申，除了小圆圈和箭头之外，流程图里包含了矩形和菱形符号。想象程序员可访问一个大柜子，其中包含大量空的 if/elif/else 结构——具体数量由程序员决定，目的是与其他控制结构堆叠和嵌套，以便结构化地实现一个算法的控制流程。然后，用适合算法的行动和决策来填充矩形和菱形。

在 if/elif/else 结构中，else 语句是可选的。然而，大多数程序员都在一系列 elif 语句结束时插入一个 else 语句，以便处理同 elif 语句中指定的任何条件都不匹配的条件。我们将 else 语句所处理的条件称为“默认条件”。如 if/elif 结构指定了一个 else 语句，它就必须是结构中的最后一个语句。

良好编程习惯 3.2 要在 if/elif 结构中提供一个默认条件。无默认条件的 if/elif 结构中，没有被显式检测的条件会被忽略。包括一个默认条件，可强迫程序员处理异常的条件。

软件工程知识 3.3 在程序中，可以放入一个语句的任何地方都可放入一个 suite。

if 选择结构主体（suite）可包含几个语句，所有这些语句都必须缩进。下例中，if/else 结构的 else 部分包括了一个 suite，其中含有两个语句。包含多个语句的 suite 有时也称为“复合语句”。

```

if grade >= 60:
    print "Passed."
else:
    print "Failed."
    print "You must take this course again."

```

在这个例子中，如 grade 小于 60，程序会执行 else 中的两个语句，并打印：

```

Failed.
You must take this course again.

```

注意，else 这个 suite 的两个语句都进行了缩进。如果以下语句没有缩进：

```
print "You must take this course again."
```

无论成绩是否小于 60，它都会执行。这是“逻辑错误”的一个典型例子。

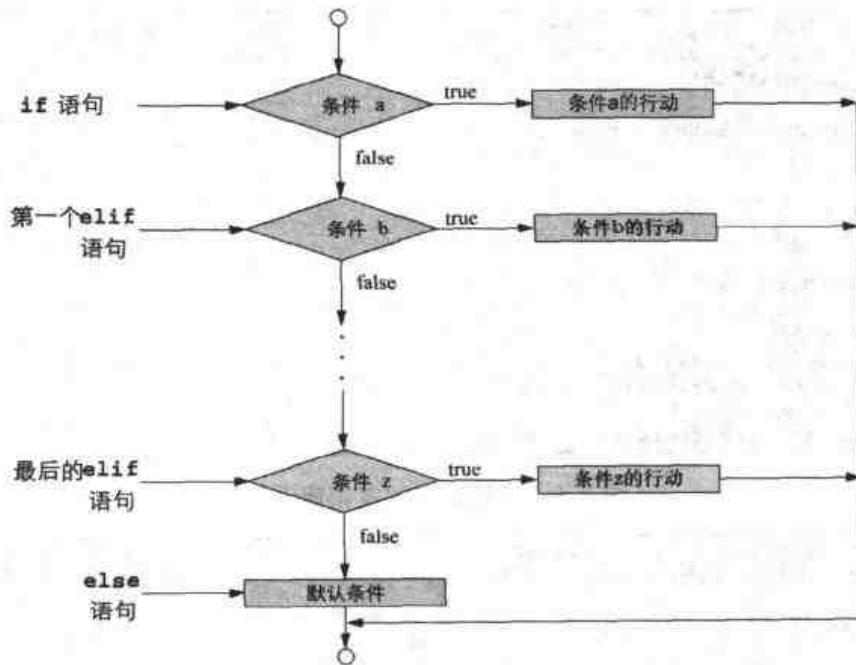


图 3.5 if/elif/else 多选结构

程序中可能出现两种主要错误：语法错误和逻辑错误。语法错误是指违反了程序语言的规则，例如将关键字用作标识符，或忘记在 if 语句后添加冒号（:）。解释器能捕捉语法错误，并显示错误提示消息。

逻辑错误导致程序产生非预期的结果，而且不可由解释器捕捉。“严重逻辑错误”会导致程序失败，并提前终止。对于严重错误，Python 会打印一条错误消息（名为“traceback”），然后退出。非严重逻辑错误则允许程序继续执行，但会产生不正确的结果。

常见编程错误 3.3 忘记对 suite 中的所有语句进行缩进，会导致语法或逻辑错误。

图 3.6 的交互式会话试图对两个用户输入的值执行除法运算，并演示一个语法错误和两个逻辑错误。语法错误出现在下面这一行：

```
print value1 +
```

运算符+需要右操作数，所以解释器会报告语法错误。第一个逻辑错误出现在下面这一行：

```
print value1 + value2
```

目的是打印两个用户输入的整数值的和。但是，字符串没有先转换成整数，所以语句不会产生希望的结果。相反，语句会连接两个字符串——把两个字符串合到一起。注意，解释器没有显示任何消息，因为语句本身是合法的。

第二个逻辑错误出现在下面这一行：

```
print int( value1 ) / int( value2 )
```

由于未检查用户输入的第二个值是否为 0，所以程序可能出现除以零的情况，这属于严重逻辑错误。

常见编程错误 3.4 除以零是严重的逻辑错误。

以前说过，任何可放入单个语句的地方都可放入多个语句。除此之外，还有另一种可能，即什么语句都不放入（或者说“放入空语句”）。要表示空语句，请在正常语句出现的位置放入一个关键字 `pass`，如图 3.7 所示。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = raw_input("Enter a number: ")
Enter a number: 3
>>> value2 = raw_input("Enter a number: ")
Enter a number: 0
>>> print value1 +
  File "<stdin>", line 1
    print value1 +
      ^
SyntaxError: invalid syntax
>>> print value1 + value2
30
>>> print int(value1) / int(value2)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
    ZeroDivisionError: integer division or modulo by zero
```

图 3.6 语法错误和逻辑错误

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> if 1 < 2:
...     pass
...     
```

图 3.7 关键字 `pass`

常见编程错误 3.5 所有控制结构至少要包含一个语句。不包含语句的控制结构会导致语法错误。

3.7 while 重复结构

“重复结构”可指定一项行动。只要某些条件保持 `true`，行动就会不断执行。例如以下伪代码语句：

```
只要 (While) 我的购物清单上还有商品
    购买下一种商品，并把它从清单上划掉
```

它描述了购物时的重复操作。其中，“我的购物清单上还有商品”是一个条件，既可能成立 (`true`)，也可能不成立 (`false`)。如果为 `true`，就采取“购买下一种商品，并把它从清单上划掉”行动。只要条件为 `true`，就重复这一行动。

`while` 重复结构包含的语句构成了 `while` 的主体 (`suite`)。`while` 结构主体可由一个或多个语句构成。最终，条件应变为 `false`（买了最后一件商品，并把它从清单上划掉）。在这个时候，重复会终止，并执行重复结构之后的第一个语句。

常见编程错误 3.6 在 `while` 结构主体中，如果没有提供最终会造成 `while` 条件变成 `false` 的一项行动，会造成“无限循环”这种逻辑错误——重复结构永远不能终止。

常见编程错误 3.7 如将关键字 `while` 拼成 `While`，会导致语法错误（记住 Python 是要区分大小写的）。所有 Python 保留关键字，比如 `while`、`if`、`elif` 和 `else` 等等，都只能采用小写字母。

下面来考虑 `while` 结构的一个例子，以下程序段用于计算 2 的所有乘方值，并判断哪个值首先大于 1000。在此，假定变量 `product` 创建并初始化成 2。下述 `while` 重复结构执行完毕之后，`product` 就会包含我们需要的答案：

```

product = 2
while product <= 1000:
    product = 2 * product

```

进入 while 结构时, product 为 2。product 变量不断地被 2 乘, 被连续指派值 4, 8, 16, 32, 64, 128, 256, 512 以及 1024。一旦 product 变成 1024, while 结构的条件 (product <= 1000) 就成为 false, 从而终止重复——product 的终值为 1024。程序将继续执行 while 之后的下一条语句。

图 3.8 的流程图揭示了与前述 while 结构对应的控制流程。同样地, 注意在流程图中, 除了小圆圈和箭头之外, 还包含一个矩形和一个菱形符号。

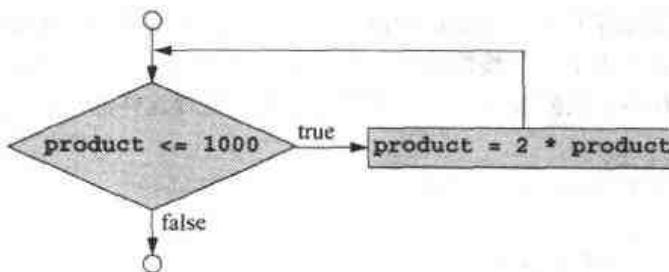


图 3.8 while 重复结构流程图

设想在一个柜子中, 包含了一系列空的 while 结构, 它们可堆叠在其他控制结构上面, 或与它们嵌套, 以实现算法的控制流程。随后, 可用适当的行动和决策来填充空矩形和菱形。该流程图清晰展示了重复过程。从矩形出发的流线会返回决策 (菱形), 除非决策结果为 false, 否则每次都要对决策进行检测。变成 false 后, 会立即退出 while 结构, 并将控制权移交给程序中的下一个语句。

3.8 算法陈述: 案例分析 1 (由计数器控制的重复)

为演示算法的开发, 下面来解决一个求平均成绩问题的几种变化形式。问题陈述如下:

全班 10 名学生参加一个测验。成绩 (0 到 100 之间的一个整数) 已经出来。请计算平均成绩。

平均成绩等于成绩总和除以学生人数。解决这个问题的算法是获得每个成绩, 执行求平均值计算, 再打印结果。

使用伪代码, 我们列出要采取的行动, 并指出它们的顺序。这里使用“由计数器控制的重复”来每次输入一个成绩。该技术要求使用一个名为 counter (计数器) 的变量, 用它控制一系列语句重复执行的次数。一旦计数器超过 10, 便终止重复。在本节, 我们展示了伪代码算法 (图 3.9), 以及相应的程序 (图 3.10)。在下一节, 还要展示如何开发伪代码算法。由计数器控制的重复通常称为“确定重复”(Definite Repetition)——因为在循环执行之前, 已经知道了要重复的次数。

```

将总和 (total) 设成 0
将成绩计数器 (counter) 设成 1

只要成绩计数器小于或等于 10
    输入下一个成绩
    将成绩加到总和中
    成绩计数器加 1

将平均成绩设为总和除以 10
打印平均成绩

```

图 3.9 使用“由计数器控制的重复”解决平均成绩问题的伪代码算法

伪代码算法中提到了一个总成绩 (total) 和一个计数器 (counter)。在图 3.10 的程序中, 变量 total

(第 5 行) 用于累加一系列值; 变量 counter 则负责计数——在这个例子中, 它统计输入了多少个成绩。total 变量通常应初始化成零。

第 5~6 行是赋值语句, 将 total 初始化为 0, 将 gradeCounter 初始化为 1。第 9 行指出只要 gradeCounter 值小于或等于 10, while 结构就应继续。第 10~11 行对应伪代码语句“输入下一个成绩”。函数 raw_input 在屏幕上提示: “Enter grade:”, 并接受用户输入。第 11 行将用户输入的字符串转换成整数。接着, 程序使用新输入的 grade 更新 total——第 12 行将 grade 加到 total 的上一个值上, 并将结果指派给 total。

然后, 程序使变量 gradeCounter 递增, 指出已处理了一个成绩。第 13 行使 gradeCounter 自增 1, 使 while 结构的条件最终能变成 false 并终止循环。while 终止后, 会执行第 16 行, 并将平均值计算结果指派给变量 average。第 17 行显示字符串“Class average is”, 后续一个空格(由 print 插入), 最后是变量 average 的值。注意, 求平均值计算产生了一个整数结果。实际上, 这个例子的成绩总和为 817。它被 10 除后, 结果应是 81.7, 这是一个有小数点的数字。下一节将讨论如何处理这样的浮点数。

```

1 # Fig. 3.10: fig03_10.py
2 # Class average program with counter-controlled repetition.
3
4 # initialization phase
5 total = 0          # sum of grades
6 gradeCounter = 1   # number of grades entered
7
8 # processing phase
9 while gradeCounter <= 10:           # loop 10 times
10    grade = raw_input("Enter grade: ") # get one grade
11    grade = int(grade)   # convert string to an integer
12    total = total + grade
13    gradeCounter = gradeCounter + 1
14
15 # termination phase
16 average = total / 10      # integer division
17 print "Class average is", average

```

```

Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81

```

图 3.10 使用“由计数器控制的重复”解决平均成绩问题

良好编程习惯 3.3 初始化 counter 和 total 变量。

在图 3.10 的程序中, 假如第 16 行使用 gradeCounter 而不是用 10 来执行计算, 程序会输出不正确的 74, 这是由于 while 循环终止后, gradeCounter 包含的值是 11。图 3.11 通过一个交互式会话演示了 while 循环重复 10 次后, gradeCounter 的值是多少。

```

Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> gradeCounter = 1
>>> while gradeCounter <= 10:
...     gradeCounter = gradeCounter + 1
...
>>> print gradeCounter
11

```

图 3.11 由计数器控制的循环终止后使用的计数器值

3.9 算法陈述，自上而下求精法：案例分析 2（由哨兵值控制的重复）

下面要对求平均成绩问题进行“泛化”。请考虑以下问题：

开发一个求平均成绩程序。该程序每次执行时，都能处理任意数量的成绩。

在第一个求平均成绩例子中，成绩的数量（10）事先已经知道。在下例中，则事先不假定要输入多少个成绩。程序必须能处理任意数量的成绩。那么，程序何时应该停止成绩输入呢？什么时候应该计算，什么时候则应打印平均成绩呢？

为解决这个问题，一个办法是使用名为“哨兵”的一个特殊值来指出“输入结束”。也可把这种值称为“信号值”、“假值”或“旗帜值”等。用户要不断输入成绩，直到所有成绩都输入完毕。然后，可输入一个特殊的哨兵值，指明已经输入了最后一个成绩。由哨兵值控制的重复通常称为“不确定重复”(Indefinite Repetition)，因为在循环开始之前，并不知道重复次数。

显然，哨兵值的选择是有讲究的，否则可能与正式输入产生混淆。由于测验成绩通常都是非负的整数，所以-1对该问题来说是能接受的一个哨兵值。因此，执行一遍这个求平均成绩程序，可能得到像“95, 96, 75, 74, 89 和 -1”这样的一连串输入。遇到-1，程序应能马上计算并打印出 95, 96, 75, 74 和 89 这 5 个值的平均值。

常见编程错误 3.8 选择一个合法数据作为哨兵值，会造成逻辑错误。

为生成求平均成绩程序，我们采用一种名为“自上而下求精法”的技术，它有助于开发具有良好结构的程序。首先用伪代码来表示“上部”：

判断测验的平均成绩

在上部，是一条简单语句，它概括了程序的总体功用。因此，最上部的语句实际是对程序的一个完整描述。但令人遗憾的是，顶部（如上所示）无法传达足够细节，无法据此写一个完整 Python 程序。所以，接下来要进行“求精”。首先，将上部的语句分解成一系列更小的任务，然后按它们的执行顺序排列好，这样便获得了“第一次求精”结果：

初始化变量
输入、总计和计数测验成绩
计算并打印平均成绩

这里只使用了顺序结构——上述步骤会顺序执行。

软件工程知识 3.4 每次求精都获得了算法的一个完整规范说明，只是细化程度有所区别。

软件工程知识 3.5 许多程序都可从逻辑上分解成 3 个阶段：初始化阶段（对变量进行初始化）；处理阶段（输入数据值，并相应地调节程序变量）和结束阶段（计算并打印最终结果）。

如果是第一次求精，参照上述“软件工程知识”进行操作便足够了。要进行下一级求精（第二次求精），需要用到一些特定的变量。程序要维持一个不断变化的 total 值，统计处理了多少个成绩的 count 值，包含了每个成绩值的一个变量，以及包含了计算好的平均成绩的一个变量。以下伪代码语句：

初始化变量

可“求精”为：

将总和初始化成零
将计数器初始化成零

以下伪代码语句：

输入、总计和计数测验成绩

则需要用一个重复（即“循环”）结构来实现，它能连续输入每个成绩。由于事先不知道要输入的成绩个数，所以计划用一个“由哨兵值控制的重复”。用户可不断输入合法的成绩值。最后一个合法成绩输入完毕后，在下一次重复时，就输入哨兵值。程序会在每次输入成绩后检测是否为哨兵值，如答案是肯定的，就终止循环。上述伪代码语句的二次求精结果如下：

```

输入第一个成绩（可能是哨兵值）
只要 (While) 用户还没有输入哨兵值
    把这个成绩加到总和里
    成绩计数器自增 1
    输入下一个成绩（可能是哨兵值）

```

以下伪代码语句：

计算并打印平均成绩

可求精为：

```

假如 (If) 计数器不等于 0
    将平均值设为总和除以计数器值
    打印平均值
否则 (else)
    打印“没有输入成绩”

```

注意这儿检测了“除以零”的可能。“除以零”属于严重逻辑错误。如果未能检测到它，程序就会失败（崩溃）。图 3.12 展示了求平均成绩问题伪代码的完整二次求精结果：

```

将总和初始化成零
将计数器初始化成零

输入第一个成绩（可能是哨兵值）
只要 (While) 用户还没有输入哨兵值
    把这个成绩加到总和里
    成绩计数器自增 1
    输入下一个成绩（可能是哨兵值）

假如 (If) 计数器不等于 0
    将平均值设为总和除以计数器值
    打印平均值
否则 (else)
    打印“没有输入成绩”

```

图 3.12 用哨兵值控制的重复来解决平均成绩问题的伪代码算法

良好编程习惯 3.4 执行除法运算时，如除数可能为零，请务必明确检测，并在程序中进行相应的处理（比如打印一条错误信息），不要任由严重错误发生。在第 12 章，我们还会具体讨论如何使程序识别此类错误，并采取相应的行动。这称为“异常处理”。

图 3.9 和图 3.12 的伪代码中包含一些空行，目的是改善伪代码的可读性。这些空行清楚地将这些语句分为不同的执行阶段。

图 3.12 的伪代码算法用于解决更“泛化”的求平均成绩问题。算法是在经两次“求精”后开发出来的。有些情况下，可能还要进一步求精。

软件工程知识 3.6 只要伪代码算法提供了足够细节，利用这些细节可将伪代码轻松转换成 Python 程序，便应停止“自上而下求精”。之后，即可轻松根据伪代码写一个 Python 程序。

图 3.13 展示了这个 Python 程序及其示范执行。尽管输入的是整数成绩，但平均值计算可能生成小数（实数）。整数数据类型不能表示实数，所以程序用浮点数据类型来处理小数，并引入了 float 函数，它强迫求平均值计算生成一个浮点数结果。

```

1 # Fig. 3.13: fig03_13.py
2 # Class average program with sentinel-controlled repetition.
3
4 # initialization phase
5 total = 0          # sum of grades
6 gradeCounter = 0   # number of grades entered
7
8 # processing phase
9 grade = raw_input("Enter grade, -1 to end: ")  # get one grade
10 grade = int(grade)    # convert string to an integer
11
12 while grade != -1:
13     total = total + grade
14     gradeCounter = gradeCounter + 1
15     grade = raw_input("Enter grade, -1 to end: ")
16     grade = int(grade)
17
18 # termination phase
19 if gradeCounter != 0:
20     average = float(total) / gradeCounter
21     print "Class average is", average
22 else:
23     print "No grades were entered"

```

```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 63
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.5

```

图 3.13 用于解决平均成绩问题的、由哨兵值控制的重复

本例证明，控制结构能一个接一个地堆叠起来（按顺序），就像小孩子搭积木那样。while 结构（第 12~16 行）后面紧跟一个 if/else 结构（第 19~23 行）。程序的许多代码与图 3.10 完全一致，所以这里只讲解其中的新特性和新问题。

第 6 行将变量 gradeCounter 初始化成 0，这是由于当前未输入任何成绩。为准确记录输入成绩数量，只有在输入一个成绩值后，变量 gradeCounter 才应增值。

良好编程习惯 3.5 在由哨兵值控制的循环中，当提示输入时，应明确指出哨兵值是哪一个。

现在研究一下由哨兵值控制的重复（图 3.13）和由计数器控制的重复（图 3.10）对程序逻辑造成的差异。在由计数器控制的重复中，每遍历一次 while 结构（遍历次数是事先指定的），都要从用户处读取一个值。在由哨兵值控制的重复中，是在程序抵达 while 结构之前读取一个值（第 9~10 行）。这个值决定了程序控制流程是否应该进入 while 结构主体。假如 while 结构的条件为 false（表明用户已输入哨兵值），程序不再执行 while 循环（因为没有输入实际的成绩）。与此相反，假如条件为 true，程序会执行 while 循环，并处理用户输入的值（即将 grade 加到 total 上）。处理完这个 grade 后，程序请求用户输入下一个 grade。执行 while 循环的最后一行（缩进的那一行）后，即第 16 行，会使用刚才用户输入的新值，再次检测 while 结构条件，判断是否应该再次执行 while 主体。注意，程序是在对 while 结构进行求

值之前请求下一个值。这样一来，就能先判断刚才输入的值是否为哨兵值，再对该值进行处理（即加到 total 上）。如果输入的是哨兵值，while 结构会终止，新输入的值不会加到 total 上。

第 9~10 行和第 15~16 行包含完全相同的代码。在 3.15 节，我们将介绍能避免重复代码的技术。

平均值并非一定是整数值。平均值一般都包含小数部分，比如 7.2 或 -93.5。这些值称为“浮点数”。

`total / gradeCounter` 计算会得到一个整数结果，因为 `total` 和 `counter` 都包含整数值。两个整数相除叫做整数除法，根据 Python 的约定，结果中的任何小数部分都会被删除（即进行“截尾”处理）。注意，小数部分是在结果指派给 `average` 之前被截去的。要通过整数值生成浮点值结果，需要用 `float` 函数将一个或两个值转换成浮点值。记住，函数本质上是执行特定任务的代码片断：在第 20 行，`float` 函数将变量 `sum` 的整数值转换成浮点值。现在计算的是一个被整数 `gradeCounter` 除的浮点值。

在表达式中，如果操作数的数据类型完全一致，Python 解释器知道怎样对其进行求值。为了保证操作数具有相同的类型，解释器会对选择的操作数执行“提升”操作，这也称为“隐式转换”。例如，在同时包含整数和浮点数据的表达式中，整数操作数会被“提升”为浮点数。在我们的例子中，`gradeCounter` 的值会提升成一个浮点数。之后才会执行计算，并将浮点除法运算的结果指派给变量 `average`。

常见编程错误 3.9 将所有浮点数都假定为是精确的，会导致不正确的结果。浮点数在大多数计算机中只是近似值。

尽管浮点数并不精确，但的确很有用。例如，当我们说人体正常体温是 98.6 华氏度时，不需要精确到许多位。用一支温度计查看温度时，虽然读数是 98.6，但实际可能是 98.5999473210643。总之，数字虽然只是 98.6，但足以满足我们的需要。

浮点数只是“近似值”可通过除法运算加以证明。10 除以 3，结果是 3.3333333...。一系列 3 会无限重复下去。然而，计算机只分配了一个固定大小的空间来容纳这样的一个值，所以很明显，浮点数只是近似值。

3.10 算法陈述，自上而下求精法：案例分析 3（嵌套控制结构）

再来研究另一个完整的例子。同样，先用伪代码技术和“自上而下求精法”设计一个算法，再根据它写出相应的 Python 程序。下面是问题陈述：

一所大学提供一门课程，让学生准备参加本州房地产经纪人资格考试。去年完成这门课的几名学生参加了资格考试。自然，这所大学想知道自己的学生在考试时的表现。现在，要求您写一个程序，对考试结果进行总结。您得到 10 名学生的一个列表。在每个姓名旁边，1 表示该学生通过了考试；2 表示没有通过。

程序应该像下面这样分析考试结果：

1. 输入每一个考试结果（1 或 2）。每次请求另一个考试结果时，都在屏幕上显示消息：“Enter result”（输入结果）。
2. 统计两类考试结果的数量（1 的数量和 2 的数量）。
3. 显示考试结果总结，分别指出通过和没有通过考试的学生的人数。
4. 假如有 8 名以上的学生通过考试，便打印一条消息“Raise tuition”（提高学费）。

仔细阅读上述问题陈述，可得出以下几个结论：

1. 程序必须处理 10 个考试结果，所以应该使用一个由计数器控制的循环。
2. 每个考试结果都是一个数字，要么是 1，要么是 2。程序每次读入一个考试结果，都必须检测该数字是 1 还是 2。在我们的算法中，打算检测的是 1。假如数字不是 1，则假定它是 2。
3. 要使用两个计数器：一个统计通过考试的学生数量，另一个统计没有通过的数量。

4. 程序处理完所有结果后，必须判断是否有 8 名以上的学生通过了考试。

现在开始运用“自上而下求精法”，首先是上部的伪代码表示：

分析考试结果，判断是否该提高学费

同样地，它虽然是对程序的完整表示，但还要进一步求精，否则无法将伪代码自然地转换成 Python 程序。第一次求精结果是：

初始化变量

连续输入 10 个考试成绩，统计通过和没有通过考试的学生人数

打印考试结果总结，判断是否该提高学费

显然，以上的信息还不够“细”。虽然获得了程序的完整表示，但仍需进一步求精。先考虑一系列特定的变量。我们要用两个计数器记录通过和未通过的学生人数，要用一个计数器控制循环，还要用一个变量保存用户输入。以下伪代码语句：

初始化变量

可求精为：

把 passes（通过考试）初始化成 0

把 failures（未通过考试）初始化成 0

把 studentCounter（学生计数器）初始化成 1

注意，上面只初始化了用于统计通过、未通过和学生人数的计数器。以下伪代码语句：

连续输入 10 个考试成绩，统计通过和没有通过考试的学生人数

需要用一个循环来连续输入每个考试成绩。由于提前知道需要 10 个考试成绩，所以当然应该使用一个由计数器控制的循环。在循环内部（它嵌套于另一个循环内），用一个双选结构判断每个考试结果到底是通过，还是没有通过。然后，让相应的计数器自增 1。对上述伪代码语句的二次求精结果是：

假如 (While) 学生计数器小于或等于 10

 输入下一个考试结果

 假如 (If) 学生通过考试

 在 passes 上加 1

 否则 (Else)

 在 failures 上加 1

 在学生计数器上加 1

注意用空行将 If/else 控制结构独立出来，以改善程序可读性。以下伪代码语句：

 打印考试结果总结，判断是否该提高学费

可求精为：

 打印通过考试的人数 (passes)

 打印没有通过考试的人数 (failures)

 假如 8 名以上的学生通过

打印 “Raise tuition” (提高学费)

图 3.14 展示了完整的二次求精结果。注意，伪代码也用空行将 while 结构独立出来，以改善程序的可读性。

```

把 passes (通过考试) 初始化成 0
把 failures (未通过考试) 初始化成 0
把 studentCounter (学生计数器) 初始化成 1

假如 (While) 学生计数器小于或等于 10
    输入下一个考试结果

    假如 (If) 学生通过考试
        在 passes 上加 1
    否则 (Else)
        在 failures 上加 1
        在学生计数器上加 1

    打印通过考试的人数 (passes)
    打印没有通过考试的人数 (failures)

假如 8 名以上的学生通过
    打印 “Raise tuition” (提高学费)

```

图 3.14 考试结果问题的伪代码

伪代码已充分进行了求精，可方便地转换成 Python。图 3.15 展示了 Python 程序以及两次示范执行结果。

注意，第 4 行使用相等运算符（==）检测变量 result 的值是否等于 1。千万不要混淆相等运算符和赋值运算符（=），否则容易造成语法或逻辑错误。

常见编程错误 3.10 在条件语句中用符号=来判断相等性是语法错误。

常见编程错误 3.11 用运算符==赋值是逻辑错误。

软件工程知识 3.7 经验表明，用计算机解决问题最有效的办法是为解决方案开发一种算法。一旦开发出正确的算法，通常能根据它方便地生成一个能实际工作的 Python 程序。

软件工程知识 3.8 许多有经验的程序员在写程序之前，从来不用伪代码这样的程序开发工具。他们觉得自己的最终目标是用计算机解决问题，写伪代码会推迟进度。尽管对于简单和熟悉的程序可以这样做，但在从事大型的、复杂的项目时，这样做有可能导致严重错误，反而会推迟进度。

```

1 # Fig. 3.15: fig03_15.py
2 # Analysis of examination results.
3
4 # initialize variables
5 passes = 0          # number of passes
6 failures = 0         # number of failures
7 studentCounter = 1    # student counter
8
9 # process 10 students; counter-controlled loop
10 while studentCounter <= 10:
11     result = raw_input("Enter result (1-pass, 2-fail): ")
12     result = int(result) # one exam result
13
14     if result == 1:
15         passes = passes + 1
16     else:
17         failures = failures + 1
18
19     studentCounter = studentCounter + 1
20

```

```

21 # termination phase
22 print "Passed", passes
23 print "Failed", failures
24
25 if passes > 8:
26     print "Raise tuition"

```

```

Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Passed 9
Failed 1
Raise tuition

```

```

Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Passed 6
Failed 4

```

图 3.15 考试结果问题

3.11 增量赋值符号

Python 提供几个增量赋值符号，用于简写赋值表达式。例如以下语句：

```
c = c + 3
```

使用“增量加法赋值符号”`+=`，可把它简写成：

```
c += 3
```

`+=` 符号将符号右边的表达式的值加到左边的变量值上，再将结果存回左边的变量。任何具有以下形式的语句（其中的“运算符”是指一个二元运算符，比如`+`，`-`，`**`，`*/`或`%`）

变量 = 变量 运算符 表达式

都可简写成

变量 运算符 = 表达式

使用了增量赋值符号的语句称为“增量赋值语句”。图 3.16 总结了增量算术赋值符号。

移植性提示 3.1 Python 2.0 版本开始引入增量赋值符号，在老版本 Python 中使用增量赋值符号是语法错误。

常见编程错误 3.12 在赋值符号左边的变量初始化之前试图使用增量赋值是错误的。

运算符	示范表达式	解释	赋值
假定: c = 3, d = 5, e = 4, f = 2, g = 6, h = 12			
$+=$	$c += 7$	$c = c + 7$	10 指派给 c
$-=$	$d -= 4$	$d = d - 4$	1 指派给 d
$*=$	$e *= 5$	$e = e * 5$	20 指派给 e
$**=$	$f **= 3$	$f = f ** 3$	8 指派给 f
$/=$	$g /= 3$	$g = g / 3$	2 指派给 g
$\%=$	$h \%= 9$	$h = h \% 9$	3 指派给 h

图 3.16 增量算术赋值符号

3.12 由计数器控制的重复的本质

由计数器控制的重复需要:

1. 一个控制变量的名称 (或循环计数器)
2. 控制变量的初始值
3. 每次循环时, 控制变量的自增或自减量
4. 用于检测控制变量终值的条件 (也就是循环是否应该继续)

图 3.17 的程序打印从 0~9 的数字。第 4 行命名控制变量 (counter), 并把初始值设为 0。在第 8 行, while 结构针对每次循环都使 counter 自增 1。while 结构的循环继续条件检测 counter 的值是否小于 10。如果大于或等于 10, 循环终止。

```

1 # Fig. 3.17: fig03_17.py
2 # Counter-controlled repetition.
3
4 counter = 0
5
6 while counter < 10:
7     print counter
8     counter += 1

```

```

0
1
2
3
4
5
6
7
8
9

```

图 3.17 由计数器控制的重复

常见编程错误 3.13 由于浮点值可能是近似值, 所以如果用浮点变量控制循环计数, 可能导致不准确的计数器值, 不能准确检测终止条件。程序应该用整数值控制循环计数。

良好编程习惯 3.6 在每个控制结构前后各留一个空行, 将其同程序的其余部分区分开。

良好编程习惯 3.7 嵌套级别过多, 会使程序难以理解。通常应将嵌套控制在 3 级以内。

良好编程习惯 3.8 在每个控制结构上下留一个空行，并对每个控制结构的主体进行缩进，使程序具有清晰的二维外观，增强可读性。

3.13 for 重复结构

for 重复结构处理由计数器控制的重复。为展示 for 的功能，下面改写图 3.17，结果见图 3.18。

```
1 # Fig. 3.18: fig03_18.py
2 # Counter-controlled repetition with the
3 # for structure and range function.
4
5 for counter in range( 10 ):
6     print counter
```

```
0
1
2
3
4
5
6
7
8
9
```

图 3.18 用 for 结构进行由计数器控制的重复

程序工作原理如下：for 结构开始执行，函数 range 会创建一个值的序列，范围为 0~9（图 3.19）。序列的第一个值指派给变量 counter，for 结构主体（第 6 行）开始执行。在序列中，后续每个值都会指派给变量 counter，并执行 for 结构主体。这个过程会一直重复，直至序列处理完毕。

图 3.19 显示了由函数 range 返回的序列。该序列是一个 Python 列表，其中包含 0~9 的整数。注意，列表中的值用方括号封闭 ([])，并用逗号分隔。列表详情在第 5 章介绍。

注意，函数 range 返回的序列的最后一个值要比传给函数的参数值小 1。如错误地写成：

```
for counter in range( 9 ):
    print counter
```

循环执行 9 次。这是一种常见的逻辑错误，即“相差 1”错误。函数 range 可取得一个、二个或三个参数。向函数传递一个参数（图 3.19），那个参数（叫做 end）要比序列的上界（最高的值）大 1。在这种情况下，range 将返回以下范围的一个序列：

0 到 (end - 1)

如传递两个参数，第一个是 start，代表序列下界（返回序列中最低的值）；第二个参数是 end。在这种情况下，range 将返回以下范围的一个序列：

(start) 到 (end - 1)

如传递 3 个参数，前两个参数分别是 start 和 end，第三个参数是 increment，代表“自增值”。在这种情况下，调用 range 所生成的序列会从 start 到 end 递增，每次都递增或递减固定的值——如果 increment 值为正数，会递增；如果为负数，则会递减。以下 3 个 range 调用会生成相同的序列，如图 3.19 所示。图 3.20 则展示了一个递减序列的例子。

```
range( 10 )
range( 0, 10 )
range( 0, 10, 1 )
```

常见编程错误 3.14 如果忘记 range 函数返回的序列的第一个值是 0（前提是提供了下界），可能导致值

相差 1 的错误。

常见编程错误 3.15 如果忘记 range 函数返回的序列的最后一个值比函数的 end 参数值小 1，可能导致值相差 1 的错误。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

图 3.19 range 函数

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> range(10, 0, -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

图 3.20 第 3 个参数为负数的 range 函数

for 结构中的序列不一定要用 range 函数生成。for 结构的常规格式是：

```
for element in sequence:
    statement(s)
```

其中，sequence（序列）是指一系列项目的集合（第 5 章会详细讲解序列）。循环第一次遍历时，序列中的第一项会指派给变量 element，并执行 statement（语句）。以后每次循环时，都先将序列中的下一项指派给变量 element，再执行 statement。为序列中的每一项都执行了一次循环后，循环会终止。大多数情况下，for 结构可用等价的 while 结构表示，例如：

```
initialization
while loopContinuationTest:
    statement(s)
    increment
```

其中，initialization（初始化）表达式对循环的控制变量进行初始化；loopContinuationTest 是循环继续条件，而 increment 使控制变量增值。

常见编程错误 3.16 创建一个没有主体语句的 for 结构是语法错误。

如果 for 结构的 sequence 部分是空白的（即序列里不包含值），程序不会执行 for 结构的主体，而是从 for 结构之后的语句继续执行。

程序有时会在循环主体中显示控制变量（element），或在计算时用到它。然而，并非一定要这样做。常见的做法是，仅用控制变量来控制重复，for 结构主体中则根本不用它。

良好编程习惯 3.9 避免在 for 循环主体更改控制变量的值，这有可能导致不易发现的逻辑错误。

for 结构的流程图和 while 差不多。图 3.21 展示了以下 for 语句的流程图：

```
for x in y:
    print x
```

流程图显示了初始化和更新过程。程序每次执行了主体语句后，都会进行更新。除小圆圈和箭头，流程图只包含矩形和菱形符号。程序员要在其中填充适合算法的行动和决策。

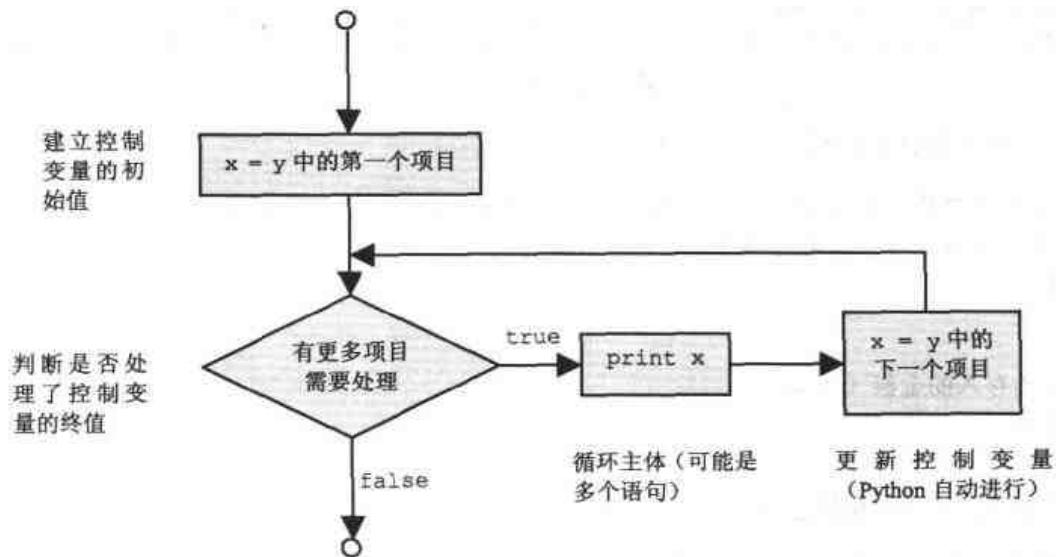


图 3.21 for 重复结构流程图

3.14 使用 for 重复结构

下例展示了在 for 结构中改变控制变量（循环计数器）的各种方法。每每遇到其中的各种情况，我们都会写下相应的 for 头部语句。注意，对 range 的第 3 个参数进行修改，可使控制变量递减。

a) 控制变量从 1 变成 100，每次增加 1。

```
for counter in range( 1, 101 ):
```

b) 控制变量从 100 变成 1，每次增加-1（减小 1）。

```
for counter in range( 100, 0, -1 ):
```

c) 控制变量从 7 变成 77，每次增加 7。

```
for counter in range( 7, 78, 7 ):
```

d) 控制变量从 20 变成 2，每次减小 2。

```
for counter in range( 20, 1, -2 ):
```

e) 控制变量为多个值的一个序列：2, 5, 8, 11, 14, 17, 20。

```
for counter in range( 2, 21, 3 ):
```

f) 控制变量为多个值的一个序列：98, 88, 77, 66, 55, 44, 33, 22, 11, 0。

```
for counter in range( 99, -1, -11 ):
```

下面两个例子演示 for 结构的简单应用。图 3.22 的程序用 for 结构求 2~100 的所有偶数整数的和。

```

1 # Fig. 3.22: fig03_22.py
2 # Summation with for.
3
4 sum = 0
5
6 for number in range( 2, 101, 2 ):
7     sum += number
8
9 print "Sum is", sum

```

Sum is 2550

图 3.22 用 for 求和

下例将用 for 结构计算复利。问题陈述如下：

某人新开一个户头，存入 1000 美元，年利率 5%。假定所有利息收入都重新存入户头，计算并打印在为期 10 年的时间里，每年结束时的存款金额。计算公式为：

$$a = p(1 + r)^n$$

其中：

p 是最开始存入的金额（本金）

r 是年利率

n 是存款年数

a 是在第 n 年结束时的存款金额

下面用循环来解决这个问题，它将计算在 10 年中每年累积的存款金额。图 3.23 展示了具体的程序。for 结构将循环主体执行 10 次，控制变量 year 从 1 递增至 10。在这个例子中，表达式 $(1 + r)^n$ 写成 $(1 + \text{rate})^{** \text{year}}$ 。其中，变量 rate 代表 r ，变量 year 代表 n 。

```

1 # Fig. 3.23: fig03_23.py
2 # Calculating compound interest.
3
4 principal = 1000.0      # starting principal
5 rate = .05               # interest rate
6
7 print "Year %2is" % "Amount on deposit"
8
9 for year in range( 1, 11 ):
10    amount = principal * ( 1.0 + rate ) ** year
11    print "%4d%21.2f" % ( year, amount )

```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

图 3.23 用于计算复利的 for 结构

for 循环之前的输出语句（第 7 行）和 for 循环中的输出语句（第 11 行）共同打印变量 year 和 amount 的值，并采用由%格式化运算符规范所指定的格式。字符%4d 提出 year 列要用一个宽度为 4 的字段打印（也就是说，打印的值至少要占据 4 个字符位置）。如果要输出的值小于 4 个字符位置，就默认在字段中右对齐。如果要输出的值超过 4 个字符位置，就自动扩展字段宽度，以适应整个值的宽度。

字符%21.2f 指出变量 amount 要打印成一个浮点值（由字符 f 指定），其中包含小数点。此列的总字段宽度是 21 个字符位置，而且小数点后有 2 位小数；在总字段宽度里，包括小数点和它右边的 2 个小数位；所以，小数点左边有 18 个位置。

注意，变量 amount, principal 和 rate 是浮点值。这样只是为了简化问题，因为要处理带有小数的美元金额，所以需要一种允许使用小数的类型。令人遗憾的是，这有时会造成问题。不妨举例说明用浮点值表示美元金额会造成什么问题（假定美元金额的小数点右侧显示 2 个数位）：假定在机器中保存的两个美元金额是 14.234（打印成 14.23）和 18.673（打印成 18.67）。两个金额相加，内部求和应得到 32.907，

打印时一般取近似值 32.91。所以打印结果可能是：

```
14.23
+18.67
-----
32.91
```

但是自己加一加，就知道结果应该是 32.90。务必注意这个问题！

良好编程习惯 3.10 用浮点值来执行财务方面的计算时，可一定要谨慎，近似值错误可能会导致不希望的结果。

注意，for 结构主体包含了 $1.0 + \text{rate}$ 这一计算（第 10 行）。实际上，每次循环时，该计算都会产生相同的结果，重复计算纯属浪费。更好的方案是定义一个变量（例如 finalRate），它在 for 结构开始前引用 $1.0 + \text{rate}$ 的值。然后，将第 10 行的 $1.0 + \text{rate}$ 替换成变量 finalRate。

性能提示 3.3 不要在循环内放入值不发生变化的表达式。

3.15 break 和 continue 语句

break 和 continue 语句用于更改控制流程。break 语句在 while 或 for 结构中执行时，会导致立即退出那个结构。程序继续执行结构之后的第一个语句。图 3.24 演示了在一个 for 重复结构中使用 break 语句的情况。if 结构一旦检测到 x 等于 5，就会执行 break。这样会终止 for 语句，并继续执行 print 语句（第 11 行）。循环将输出 4 个数字。

```
1 # Fig. 3.24: fig03_24.py
2 # Using the break statement in a for structure.
3
4 for x in range( 1, 11 ):
5
6     if x == 5:
7         break
8
9     print x,
10
11 print "\nBroke out of loop at x =", x
```

```
1 2 3 4
Broke out of loop at x = 5
```

图 3.24 for 结构中使用的 break 语句

图 3.25 是图 3.13 的一个修改版本，这个版本消除了老版本程序中的重复代码。第 9 行开始一个无限 while 循环。循环条件永远为 true，因为 1 始终代表 true。第 10~11 行提示用户输入成绩，并将输入转换成整数。如成绩是哨兵值 -1，就退出循环（第 14~15 行）。

```
1 # Fig. 3.25: fig03_25.py
2 # Using the break statement to avoid repeating code
3 # in the class-average program.
4
5 # initialization phase
6 total = 0          # sum of grades
7 gradeCounter = 0   # number of grades entered
8
9 while 1:
10    grade = raw_input( "Enter grade, -1 to end: " )
11    grade = int( grade )
12
13    # exit loop if user inputs -1
```

```

14     if grade == -1:
15         break
16
17     total += grade
18     gradeCounter += 1
19
20 # termination phase
21 if gradeCounter != 0:
22     average = float( total ) / gradeCounter
23     print "Class average is", average
24 else:
25     print "No grades were entered"

```

```

Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.5

```

图 3.25 用 break 语句消除重复代码

continue 语句在 while 或 for 结构中执行时，会跳过结构中其余的语句，继续下一次循环。在 while 结构中，循环继续检测会在 continue 语句执行之后立即进行。在 for 结构中，会将序列的下一个值指派给控制变量（前提是序列包含更多的值）。前面说过，通常可用 while 结构来表示 for 结构。但是，如果 while 结构中的自增表达式跟在 continue 语句之后，就不能这么做。在这种情况下，在检测循环继续条件之前，不会执行自增，所以 while 的执行方式有别于 for。图 3.26 在一个 for 结构中使用 continue 语句跳过结构中的输出语句，直接重复下一次循环。

```

1 # Fig. 3.26: fig03_26.py
2 # Using the continue statement in a for/in structure.
3
4 for x in range( 1, 11 ):
5
6     if x == 5:
7         continue
8
9     print x,
10
11 print "\nUsed continue to skip printing the value 5"

```

```

1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5

```

图 3.26 在 for 结构中使用 continue 语句

良好编程习惯 3.11 有的程序员认为 break 和 continue 违反了结构化编程准则。由于可采用后文即将讨论的结构化编程技术达到同样的目的，所以这些程序员不使用 break 和 continue。

3.16 逻辑运算符

到目前为止学习的都是“简单条件”，比如 counter ≤ 10 , total > 1000 以及 number $\neq \text{sentinelValue}$ 等。我们用关系运算符 $>$, $<$, \geq 和 \leq 以及相等运算符 $=$ 和 \neq 表示这些条件。每项决策检测的都只是一个条件。如需检测多个条件才能做出一项决策，就要用单独的语句执行这些检测，或者通过嵌套 if 或 if/else 结构进行。Python 提供了“逻辑运算符”，可合并简单条件而构成复杂条件。逻辑运算符包括 and (逻辑 AND), or (逻辑 OR) 以及 not (逻辑 NOT, 也叫做逻辑非)。下面分别举例说明。

选择特定的执行路径前，假定想确保两个条件都为 true，那就可以像下面这样使用逻辑 and 运算符：

```
if gender == "Female" and age >= 65:
    seniorFemales += 1
```

if 语句实际包含两个简单条件。条件 gender == "Female" 判断一个人是否女性；条件 age >= 65 判断一个人是否为高龄居民。首先求值 and 运算符左边的简单条件，因为==的优先级高于 and。如有必要，接着会求值 and 运算符右边的简单条件，因为>=的优先级高于 and（后面会讲到，只有逻辑 AND 表达式左侧求值结果为 true，才会对右侧求值）。然后，if 语句会对以下复合条件进行判断：

```
gender == "Female" and age >= 65
```

只有两个简单条件都为 true，上述复合条件才会为 true。最后，如果该复合条件确实为 true，那么 seniorFemales 的计数会自增 1。假如两个简单条件有一个或全部为 false，程序会跳过自增运算，执行 if 之后的下一条语句。加上冗余括号之后，上述复合条件显得更易懂：

```
( gender == '"Female" ) and ( age >= 65 )
```

图 3.27 中的表总结了 and 运算符。它针对表达式 1 和表达式 2，列出了 4 种可能的真假组合。这种表称为“真值表”(Truth Table)。

表达式 1	表达式 2	表达式 1 and 表达式 2
假	假	假
假	真	假
真	假	假
真	真	真

图 3.27 and (逻辑 AND) 运算符真值表

任何表达式只要包含关系运算符和相等运算符，Python 都会将该表达式求值为真或假。为 false 的简单条件（例如 age >= 65）会求值为整数值 0；为 true 的简单条件会求值为整数值 1。求值为 0 的 Python 表达式为 false；求值为非 0 的整数值的 Python 表达式为 true，图 3.28 的交互式会话演示了这些概念。

```
Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> if 0:
...     print "0 is true"
... else:
...     print "0 is false"
...
0 is false
>>> if 1:
...     print "non-zero is true"
...
non-zero is true
>>> if -1:
...     print "non-zero is true"
...
non-zero is true
>>> print 2 < 3
1
>>> print 0 and 1
0
>>> print 1 and 3
3
```

图 3.28 真值

交互式会话的第 5~10 行显示值 0 为 false。第 11~18 行显示任何非零的整数值都是 true。第 19 行

的简单条件求值为 true (第 20 行)。第 21 行和第 23 行的复合条件演示了 and 运算符的返回值。如果复合条件求值为 false (第 21 行), and 运算符就返回求值为 false 的第一个值 (第 22 行)。相反, 如复合条件求值为 true (第 23 行), and 运算符就返回条件中的最后一个值 (第 24 行)。

现在来看看 or (逻辑 OR) 运算符。在程序的某个位置采取特定行动之前, 如希望保证两个条件之一或者全部为 true, 就可使用 or 运算符。如下所示:

```
if semesterAverage >= 90 or finalExam >= 90:  
    print "Student grade is A"
```

上述条件也合并了两个简单条件。简单条件 semesterAverage >= 90 判断学生是否因为他在学期中一贯的良好表现而得 “A”; 简单条件 finalExam >= 90 则判断是否由于学生的期末成绩大于或等于 90 分而得 “A”。然后, if 语句考察复合条件:

```
semesterAverage >= 90 or finalExam >= 90
```

只要满足其中任何一个条件, 或两个条件全满足, 学生就可得 “A”。注意, 假如上述两个简单条件都为 false, 就不打印 “Student grade is A” 消息。图 3.29 是 or 运算符的真值表。

表达式 1	表达式 2	表达式 1 or 表达式 2
假	假	假
假	真	真
真	假	真
真	真	真

图 3.29 or (逻辑 OR) 运算符真值表

如果复合条件求值为 true, or 运算符就返回求值为 true 的第一个值。相反, 如果复合条件求值为 false, or 运算符就返回条件中的最后一个值。

and 的优先级高于 or。两个运算符都从左到右顺序关联。包含 and 或 or 运算符的表达式会不断求值, 一直到能确定真假为止。这称为“短路求值”。因此, 对以下表达式的求值:

```
gender == "Female" and age >= 65
```

会在 gender 不等于 "Female" 的前提下立即停止, 因为已能确定整个表达式为 false。但假如 gender 等于 "Female", 就会继续求值 (因为如果条件 age >= 65 为 true, 整个表达式会为 true)。

性能提示 3.4 在使用了 and 运算符的表达式中, 假如不同条件是相互独立的, 就将最有可能为 false 的条件放在最左边。在使用了 or 运算符的表达式中, 要把最有可能为 true 的条件放在最左边。这样做可缩短程序执行时间。

Python 还提供了 not (逻辑非) 运算符, 用于反转一个条件的含义。and 和 or 运算符对两个条件进行合并 (二元运算符), 而逻辑非运算符只需要一个条件作为操作数 (也就是说, not 是一元运算符)。逻辑非运算符要放在希望反转的条件之前。如果希望在原始条件为 false 的前提下选择一个执行路径, 就可考虑使用逻辑非运算符, 如下例所示:

```
if not grade == sentinelValue:  
    print "The next grade is", grade
```

图 3.30 是逻辑非运算符的真值表。许多时候, 程序员可避免使用逻辑非, 具体做法是采用合适的关系运算符或相等运算符, 以不同方式表示条件。例如, 上例可改写成:

```
if grade != sentinelValue:  
    print "The next grade is", grade
```

借助于这种灵活性，程序员通常可采用更自然、更方便的方式来表示一个条件。

表达式	not 表达式
假	真
真	假

图 3.30 not（逻辑非）运算符真值表

图 3.31 总结了前面已介绍过的 Python 运算符的优先级。各运算符根据优先级从上到下降序排列。

运算符	顺序关联性	类型
()	从左到右	圆括号
**	从右到左	求幂
* / // %	从左到右	乘
+ -	从左到右	加
< <= > >=	从左到右	关系
== != <>	从左到右	相等
and	从左到右	逻辑 AND
or	从左到右	逻辑 OR
not	从右到左	逻辑 NOT

图 3.31 运算符的优先级和顺序关联性

3.17 结构化编程总结

建筑师设计建筑物时，必须综合运用历年积累的行业知识与经验。程序员设计程序时，也不例外。不过，这个行业要比建筑业年青，所以积累的知识与经验较少。通过以前的学习，我们知道同非结构化程序相比，结构化编程所生成的程序要容易理解得多，所以更易测试、调试和修改，而且不易出错。

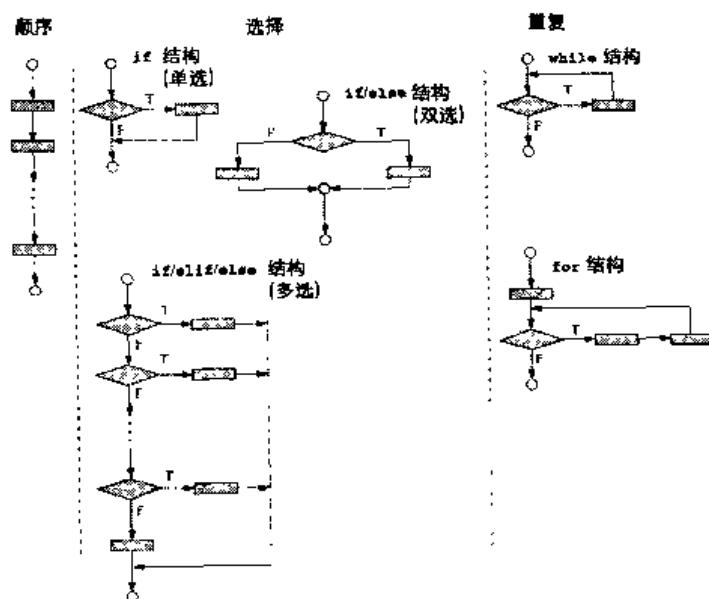


图 3.32 单入/单出的顺序、选择和重复结构

图 3.32 总结了 Python 控制结构。小圆圈表示每种结构的单一入口和出口。如果任意连接各种流程

图符号，可能会开发出非结构化的程序。因此，设计者决定合并流程图符号，构成一个有限的控制结构集，并只允许用两种简单方式来正确合并控制结构，以构建最终的结构化程序。

为简化起见，我们使用了单入 / 单出控制结构。对于每个控制结构，只有一条路可以进入，也只有一条路可以退出。顺序连接不同控制结构，可方便地构建一个结构化程序——把一个控制结构的出口同下一个控制结构的入口连起来即可。也就是说，各控制结构在程序中一个接一个地放在一起——这称为“控制结构的堆叠”。结构化程序的构建规则也允许进行控制结构的嵌套。

图 3.33 总结了正确构建结构化程序的规则。规则假定用流程图中的矩形符号标注一项行动，包括输入和输出等等。规则还假定从最简流程图（图 3.34）开始。

结构化程序的构建规则：

- (1) 从如图 3.34 所示的“最简流程图”开始
- (2) 任何矩形（行动）都可被替换成两个顺序的矩形（行动）
- (3) 任何矩形（行动）都可被替换成任何控制结构（顺序、if、if/else、if/elif/else、while 或 for）
- (4) 规则 (2) 和 (3) 可根据需要应用任意次数，并可按任意顺序应用

图 3.33 构建结构化程序的规则

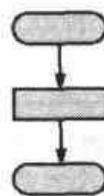


图 3.34 最简流程图

按图 3.33 的规则操作，肯定能获得一个条理清晰的、积木式的结构化流程图。例如，将规则 (2) 重复应用于最简流程图，最终的结构化流程图中会包含一系列顺序排列的矩形，如图 3.35 所示。注意由于规则 (2) 可生成控制结构的堆叠，所以该规则也称为“堆叠规则”。

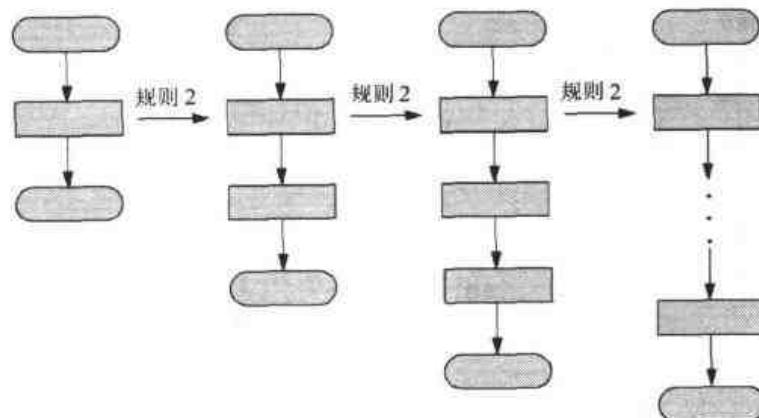


图 3.35 将图 3.33 的规则 (2) 应用于最简流程图

规则 (3) 称为“嵌套规则”。将规则 (3) 重复应用于最简流程图，会生成一个嵌套控制结构。例如在图 3.36 中，首先用一个双选 (if/else) 结构替换最简流程图中的矩形。随后，再次将规则 (3) 应用于双选结构中的两个矩形，用双选结构替换每个矩形。围绕每个双选结构的虚线框表示被替换的矩形。

规则 (4) 可生成更大的、头绪更多的、嵌套更深的结构。应用图 3.33 的规则，可获得任何可能的结构化流程图，所以也能生成任何可能的结构化程序。

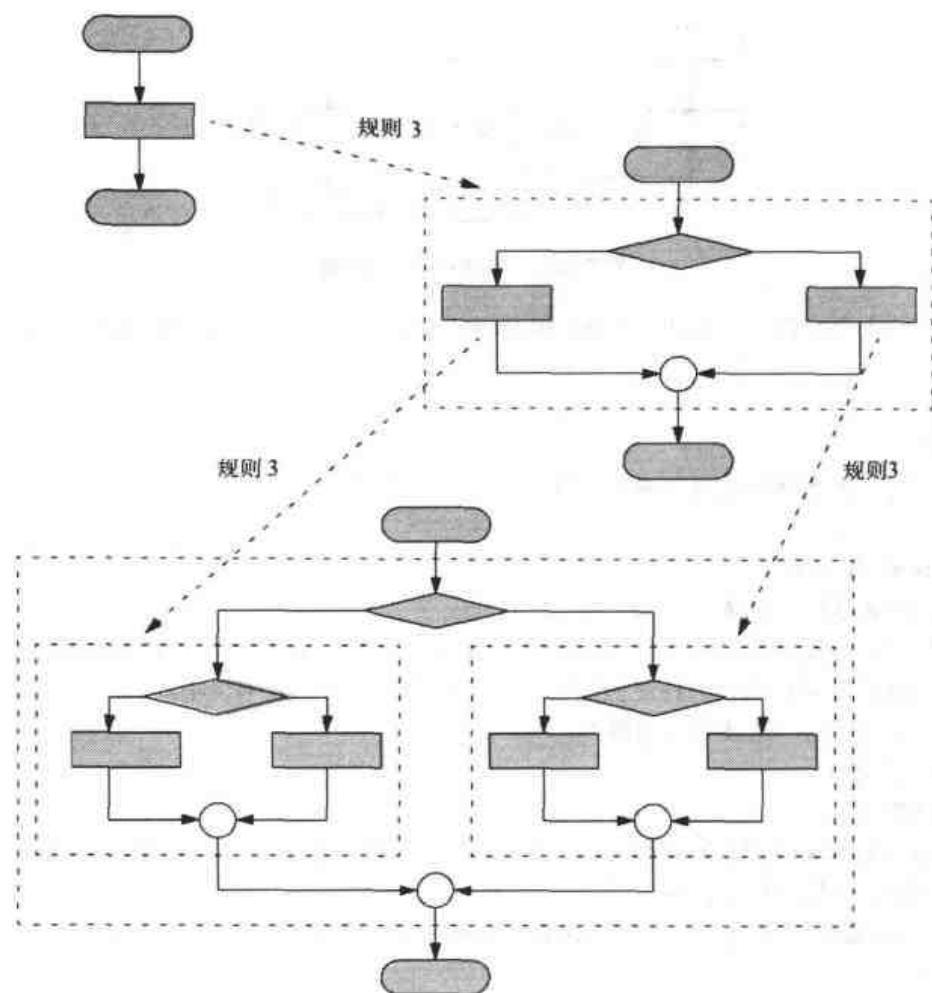


图 3.36 将图 3.35 的规则 (3) 应用于最简流程图

结构化编程的好处在于，整个过程只需使用 6 种简单的单入 / 单出结构，而且只需使用 2 种简单方法来“组装”它们。图 3.37 展示了一系列堆叠的“积木”，应用规则 (2) 生成；以及一系列嵌套的“积木”，应用规则 (3) 生成。该图还显示了积木重叠的情况，它们不可出现在结构化流程图中（因为已取消了 goto 语句）。

遵循图 3.33 的规则，不可能得到一个非结构化流程图（像图 3.38 那样）。如果不能确定流程图是否结构化，可反方向应用图 3.33 的规则，看看是否能将流程图简化成最简形式。如果能还原成最简流程图，就表明原来的流程图是结构化的；否则是非结构化的。

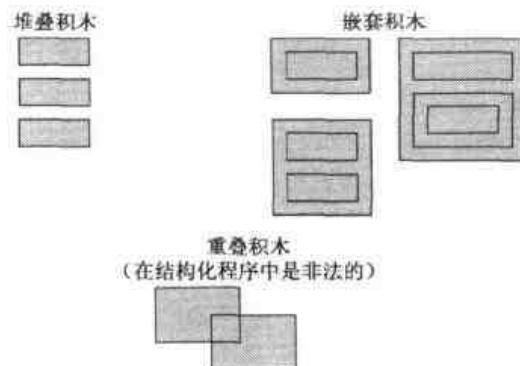


图 3.37 堆叠、嵌套和重叠积木

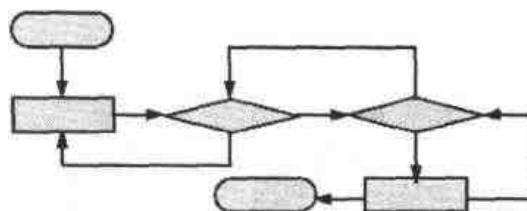


图 3.38 非结构化流程图

结构化编程的主要宗旨就是简化。Bohm 和 Jacopini 告诉我们，只需 3 种形式的控制：

- 顺序
- 选择
- 重复

顺序控制最简单。选择控制可采取以下 3 种方式之一实现：

- if 结构（单选）
- if/else 结构（双选）
- if/elif/else 结构（多选）

其实，很容易便可证明简单的 if 结构足以实现任何形式的选择。也就是说，用 if/else 和 if/elif/else 结构能做到的事情，通过合并 if 结构一样能做到（尽管可能条理不清，效率也不高）。

可用以下 3 种方式之一来实现重复控制：

- while 结构
- for 结构

同样很容易证明，仅用 while 结构便足以实现任何形式的重复。也就是说，用 for 结构能做到的事情，用 while 结构也能做到（尽管可能有点难度）。

综上所述，Python 程序需要的任何形式的控制都可通过下面 3 种形式来表示：

- 顺序
- if 结构（用于选择）
- while 结构（用于重复）

另外，这些控制结构只需选用两种方式（即堆叠和嵌套）来组合。总而言之，结构化编程大大简化了编程工作。

本章讨论了如何利用包含行动和决策的控制结构来构建程序。下一章要介绍另一种程序结构单元，即“函数”。您将学习如何通过合并各个函数（每个函数又可包含多种控制结构）来构建大型程序。此外，还要讨论函数是如何增强软件重用性的。在第 7 章，我们将介绍 Python 的另一种程序构建单元，即“类”。此后，还将用类来创建对象，开始进入面向对象编程（OOP）技术的学习。

第4章 函数

学习目标

- 理解如何利用函数模块化地构建程序
- 会新建函数
- 理解函数间的信息传递机制
- 会利用随机数生成来实现模拟
- 理解标识符的可见性如何被限定在程序的特定区域
- 理解如何编写和使用递归函数（即能够调用自身的函数）
- 理解默认和关键字参数

4.1 概述

解决实际问题的大多数计算机程序都比前几章介绍的大。经验表明，开发和维护一个大程序时，最好是基于较小的部分或“组件”构建。同原始程序相比，每个组件都更易管理。这称为“分而治之”。本章将介绍用于简化大型程序设计、实现、运行和维护的许多 Python 语言特性。

4.2 Python 中的程序组件

Python 的程序组件包括函数、类、模块和包。程序员通常将自己定义的函数和类与现成 Python 模块提供的函数和类进行合并，从而写出新程序。“模块”（module）是包含函数和类定义的文件。许多模块可组合成一个集合，称为“包”（package）。本章重点在于函数，并适当介绍了模块和包；类的详情在第 7 章讲解。

程序员可定义函数以执行特定任务，这些任务可在程序的多个位置进行。这称为“程序员自定义函数”。定义函数的实际语句只需编写一次，但可在程序的多个位置调用。因此，函数是在 Python 中实现“软件重用”的一种基本单元，因其实现了程序代码的重用。

Python 模块提供了用于执行常见任务的函数，比如数学计算、字符串处理、字符处理、Web 编程、图形编程和其他许多操作。函数简化了程序员的工作，因为他们不必再写新函数来执行常见任务。模块集合（即“标准库”）被作为核心 Python 语言的一部分提供。这些模块位于 Python 安装目录的库目录下。在 Unix/Linux 上，是 /usr/lib/python2.2 或 /usr/local/lib/python2.2；在 Windows 上，则是 \Python\lib 或 \Python22\lib。

“模块”组合了相关定义，“包”则组合了相关模块。包作为一个整体，提供了帮助程序员完成常规任务的工具（比如图形或音频编程）。包中每个模块都定义了类、函数或数据，用于执行特定的、相关的任务（比如创建颜色、处理.wav 文件等）。本书介绍了许多现成的 Python 包，如何创建一个可靠的包，则属于软件工程的范畴，不在本书进行讨论。

良好编程习惯 4.1 尽快熟悉核心 Python 模块提供的函数和类集合。

软件工程知识 4.1 避免重复别人的劳动。尽量使用标准库模块函数，不要写新函数。这可加快程序开发进度，并增强可靠性，因为您使用的是经过良好设计和测试的代码。

移植性提示 4.1 使用核心 Python 模块中的函数，通常可使程序更易移植。

性能提示 4.1 不要试图改写现成的模块函数使其更高效。这些函数已非常完美了。

函数需要被调用才能完成其目标任务。在函数调用中，需要指定函数名称，并提供函数执行任务所需的信息（比如参数）。这类似于公司的层次管理系统。老板（即“调用函数”或“调用者”）要求一名员工（即“被调用函数”）完成一项任务，并在完成后返回（汇报）结果。老板不关心员工具体如何完成任务。员工完全能调用其他员工，而老板根本注意不到。后面会解释如何通过“隐藏”实现细节来促进良好的软件工程。图 3.1 展示了老板函数（boss）函数如何通过一个层次化结构，与员工函数（worker1, worker2 和 worker3）通信。boss 在调用 worker1 时，不必知道 worker1 同 worker4 和 worker5 的关系。注意函数的实际关系可能有别于这张图显示的层次结构。

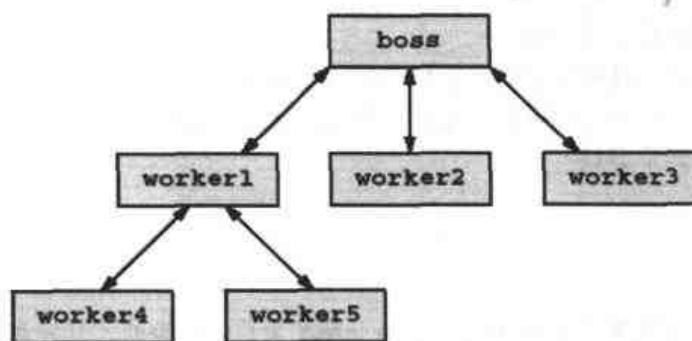


图 4.1 层次化的 boss/worker 函数关系

4.3 函数

程序员利用函数来模块化一个程序。函数定义中创建的所有变量都是“局部变量”——只存在于声明它们的函数中。许多函数都有一个参数列表，用于进行函数间的通信，这些参数也是局部变量。

有几方面的动机要求对一个程序进行“函数化”。首先，“分而治之”技术使程序开发更易管理。其次是“软件重用性”——以现有函数为基础来创建新程序。软件重用性是面向对象编程的一个主要优势（详情参见第 7 章、第 8 章和第 9 章）。采用良好的函数命名和定义，程序可通过负责特定任务的标准化函数来创建，而不必为每个任务都编写自定义的代码。最后则是在程序中避免重复代码。把代码包装成一个函数，可通过调用函数在不同地方执行这些代码，而不必在用到它们的每个地方都重写一遍。

软件工程知识 4.2 每个函数都应该只限执行单一的、良好定义的任务，函数名应清楚地描述那个任务。这样有利于提升软件的重用性。

软件工程知识 4.3 如实在想不出能准确表达函数作用的名称，就表明函数可能执行了太多分散的任务。通常，最好将这种函数分解成多个更小的函数。

4.4 math 模块的函数

模块包含了函数定义和执行相关任务的其他元素（例如类定义）。利用 math 模块的函数，可执行特定的数学计算。我们用 math 模块的各个函数来介绍函数和模块的概念。书中各处还将讨论核心 Python 模块中的其他许多函数。

调用一个函数时，通常要写下它的名称，后续一个左圆括号、要传给函数的参数（或者用逗号分隔的参数值列表）以及一个右圆括号。要使用模块定义的一个函数，程序必须导入模块，这是用关键字 import 来实现的。模块导入后，程序即可调用模块中的函数，这需要使用模块名、一个小圆点（.）和正式的函数调用，即 `moduleName.functionName()`。图 4.2 的交互式会话演示了如何用 math 模块打印 900 的平方根。

```

Python 2.2b2 (#26, Nov 16 2001, 11:44:11) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> print math.sqrt( 900 )
30.0
>>> print math.sqrt( -900 )
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ValueError: math domain error

```

图 4.2 math 模块的 sqrt 函数

执行下面这一行时：

```
print math.sqrt( 900 )
```

math 模块的 sqrt 函数会计算括号中包含的数字（900）的平方根。900 是 math.sqrt 函数的参数。函数会返回浮点值 30.0，以便在屏幕上显示。执行下面这一行时

```
print math.sqrt( -900 )
```

函数调用会生成一个错误，即“异常”，因为 sqrt 函数不能处理负的参数值。解释器将在屏幕上显示同这个错误有关的信息。异常和异常处理的问题将在第 12 章讨论。

常见编程错误 4.1 使用 math 模块的函数时，忘记导入 math 模块属于运行时错误。只有导入模块，才能使用其中的函数和变量。

常见编程错误 4.2 用 import 语句导入模块后，如果忘记为它的函数附加模块名前缀，就会造成运行时错误。

函数的参数可为值、变量或表达式。假定 $c1 = 13.0$, $d = 3.0$, 而 $f = 4.0$, 那么下述语句：

```
print math.sqrt( c1 + d * f )
```

会计算并打印 $13.0 + 3.0 * 4.0 = 25.0$ 的平方根，结果为 5.0。图 4.3 总结了其他一些 math 模块函数，注意有的结果已经取整。

函数	说明	示例
<code>acos(x)</code>	求 x 的反余弦（结果是弧度）	<code>acos(1.0)</code> 等于 0.0
<code>asin(x)</code>	求 x 的反正弦（结果是弧度）	<code>asin(0.0)</code> 等于 0.0
<code>atan(x)</code>	求 x 的反正切（结果是弧度）	<code>atan(0.0)</code> 等于 0.0
<code>ceil(x)</code>	为 x 取整，结果是不小于 x 的最小整数	<code>ceil(9.2)</code> 等于 10.0 <code>ceil(-9.8)</code> 等于 -9.0
<code>cos(x)</code>	求 x 的余弦（ x 是弧度）	<code>cos(0.0)</code> 等于 1.0
<code>exp(x)</code>	求幂函数 e^x	<code>exp(1.0)</code> 等于 2.71828 <code>exp(2.0)</code> 等于 7.38906
<code>fabs(x)</code>	求 x 的绝对值	<code>fabs(5.1)</code> 等于 5.1 <code>fabs(-5.1)</code> 等于 5.1
<code>floor(x)</code>	为 x 取整，结果是不大于 x 的最大整数	<code>floor(9.2)</code> 等于 9.0 <code>floor(-9.8)</code> 等于 -10.0
<code>fmod(x, y)</code>	求 x/y 的余数，结果是浮点数	<code>fmod(9.8, 4.0)</code> 等于 1.8
<code>hypot(x, y)</code>	求直角三角形的斜边长度，直边长度为 x 和 y : $\sqrt{x^2 + y^2}$	<code>hypot(3.0, 4.0)</code> 等于 5.0