普通高等教育“十三五”规划教材

Python 3 程序设计基础

刘德山　付彬彬　黄　和　主　编
孙治军　张云龙　刘　钢　副主编

科学出版社
北　京

内 容 简 介

本书以易懂的语言、翔实的示例、全新的内容来诠释 Python 这门“简单”“优雅”“明确”“易学”的计算机语言。全书共 12 章，第 1～8 章是 Python 语言基础，覆盖了全国计算机等级考试二级 Python 语言程序设计的主要内容；第 9～11 章是 Python 语言的应用，包括图形用户界面、面向对象程序设计、数据库编程等内容；第 12 章重点介绍了 Python 第三方库的应用。本书内容以应用为核心展开，力求以知识的最小集来实现最大范围的应用。

本书难度适中，主要面向普通本科院校非计算机专业的学生，适合作为初学者学习 Python 程序设计的教材，也可作为全国计算机等级考试二级 Python 程序设计的参考用书。

图书在版编目(CIP)数据

Python 3 程序设计基础 / 刘德山，付彬彬，黄和主编. —北京：科学出版社，2018.11

普通高等教育“十三五”规划教材

ISBN 978-7-03-059071-8

Ⅰ. ①P… Ⅱ. ①刘… ②付… ③黄… Ⅲ. ①软件工具－程序设计－高等学校－教材 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 232899 号

责任编辑：宋 丽 王 惠 / 责任校对：马英菊

责任印制：吕春珉 / 封面设计：东方人华平面设计部

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

印刷

科学出版社发行 各地新华书店经销

*

2018 年 11 月第 一 版 开本：787×1092 1/16

2018 年 11 月第一次印刷 印张：19

字数：448 000

定价：49.00 元

（如有印装质量问题，我社负责调换〈 〉）

销售部电话 010-62136230 编辑部电话 010-62135397-2052

前　言

近年来，Python 已经成为计算机语言中的热词。

人工智能已经上升为国家战略。Python 丰富的 AI 库、机器学习库、自然语言和文本处理库，使其成为人工智能时代最好的语言。Python 还应用于数据分析、组件集成、图像处理、科学计算等众多领域。

为适应跨界创新的需求，不同层次、不同专业的读者，迫切需要一种可以更多地专注于要解决的问题，而不必更多地考虑细节的计算机语言，让计算机语言回归服务的功能，Python 是最佳选择。

Python 以“简单”“优雅”“明确”“易学”等特点成为学习编程的入门语言。十几万的第三方库形成了 Python 的“计算生态”，推动了 Python 的发展。

Python 在业界广泛使用，如 YouTube、BT、豆瓣、知乎、Google、Yahoo!、Facebook、百度、腾讯、美团等都在使用 Python。

面对 Python 诸多的应用和需求，以及适合于所有专业学生学习的特点，2018 年，教育部将 Python 纳入全国计算机等级考试范围，相信未来几年 Python 将得到更好地普及与发展。

作者曾主讲 C、Java、Python 系列课程，从教学实践中精选了大量的示例，从“实用、易用、有效”的角度组织编写了本书，让读者能以最短的时间，了解和学习这门简单、优雅的语言。本书内容以应用为核心展开，难度适中，力求以知识的最小集来实现最大范围的应用。本书追求语言易懂、示例翔实、结构明晰的风格。

本书主要有以下特色：

1．重点突出。在保证内容科学、完整的前提下，由浅入深地安排章节次序。考虑到 Python 语言的应用特色，更多强调应用思维。例如，在组合数据类型的应用、文件操作、第三方库等章节，更着重体现应用特性。将 Python 面向对象程序设计部分安排到第 11 章，教师可以根据需要选讲。

2．案例资源丰富。全书设计了 237 个示例，覆盖 Python 的重要知识点；加强实践环节，精心设计和编写课后习题；教材编写与教学资源建设同步，提供教学课件、程序源码。

3. 综合考虑全国计算机等级考试和 Python 应用需求。书中的知识点基本覆盖了等级考试需要的核心内容，又对部分使用频率低的内容做了删减。

本书结构如下：

第 1～8 章是 Python 语言基础，覆盖了全国计算机等级考试二级 Python 语言程序设计的主要内容，教学设计约 32 学时。

第 9～11 章是 Python 语言的应用，包括图形用户界面、面向对象程序设计、数据库编程等内容。

第 12 章详细介绍了 Python 第三方库的应用。读者应掌握 Python 第三方库文档的查阅方法，了解 Python 库编程的特点，领会 Python 编程的“计算生态”特征。

本书建议教学的组织形式是示例—分析—练习—总结。从应用的角度介绍语言，通过示例来说明编程的方法和过程。建议授课 32～48 学时，第 9～12 章根据需要选讲。

本书适合作为 Python 程序设计课程的教材，以及全国计算机等级考试辅导教材。

本书由刘德山、付彬彬、黄和任主编，孙治军、张云龙、刘钢任副主编，李舒、王婷婷参与编写。书中难免存在疏漏和不足之处，恳请读者指正。

48 学时不长，稍纵即逝；48 学时不短，您可以学习和发挥 Python 的荣光。人生苦短，一起学习 Python。

编　者

2018 年 8 月

目　录

第 1 章
Python 概述

Python 是一种面向对象的、解释型的高级计算机语言，可应用于 Web 开发、科学计算、游戏开发、图形用户界面设计等方面。那么，什么是计算机语言？解释型语言有什么特点？Python 语言有什么特点？本章将介绍这些内容，并介绍 Python 程序的开发环境和程序的执行过程。

1.1 计算机语言

1.1.1 计算机语言概述

让计算机按照用户的目的完成相应的操作，需要使用计算机语言。计算机语言即程序设计语言，是用于描述计算机所执行的操作的语言。从计算机产生到现在，作为软件开发工具的程序设计语言经历了机器语言、汇编语言、高级语言等几个阶段。

1. 机器语言

机器语言是采用计算机指令格式并以二进制编码表达各种操作的语言。计算机能够直接理解和执行机器语言程序。

机器语言的特点是能够被计算机直接识别，执行速度快，占用存储空间小，但难读、难记，编程难度大，调试、修改烦琐，而且不同型号的计算机具有不同的机器指令系统。

2. 汇编语言

汇编语言是一种符号语言，它用助记符来表达指令功能。

汇编语言程序较机器语言程序易读、易写，并保持了机器语言执行速度快、占用存储空间小的优点。但汇编语言的语句功能比较简单，程序的编写仍然比较复杂，而且程序难以移植，因为汇编语言和机器语言都是面向机器的语言，都是为特定的计算机系统而设计的。汇编语言程序不能被计算机直接识别和执行，需要由一种起翻译作用的程序

（汇编程序），将其翻译成机器语言程序（目标程序），计算机才能执行，这一翻译过程称为“汇编”。

机器语言和汇编语言统称为低级语言。

3. 高级语言

高级语言是面向问题的语言，比较接近于人类的自然语言。高级语言是与计算机结构无关的程序设计语言，具有更强的表达能力，可以方便地表示数据的运算和程序控制结构，能更有效地描述各种算法，使用户容易掌握。

Python 语言是一种高级语言，例如，计算并显示 5+11 运算结果的 Python 语言程序如下：

```
>>> print(5+11)
16    #运算结果
```

用高级语言编写的程序（源程序）不能被计算机直接识别和执行，需要经过翻译程序翻译成机器语言程序（目标程序）才能执行，高级语言的翻译程序有编译程序和解释程序两种。下面分别介绍编译程序和解释程序。

1.1.2 编译与解释

用不同的高级语言编写的计算机程序，其执行方式是不同的。这里所说的执行方式是指计算机执行一个程序的过程。按照计算机程序的执行方式，将高级语言分成两类：静态语言和脚本语言。静态语言采用编译执行的方式，脚本语言采用解释执行的方式。无论哪种方式，用户执行程序的方法是一致的，如通过鼠标双击执行一个程序。

1. 编译

编译是将源程序代码转换成目标代码的过程。源代码是计算机高级语言代码，而目标代码则是机器语言代码。执行编译的计算机程序称为编译器（compiler）。

2. 解释

解释是将源代码逐条转换成目标代码，同时逐条运行目标代码的过程。执行解释的计算机程序称为解释器（interpreter）。

解释和编译的区别在于，编译是一次性地翻译，程序被编译后，运行时不再需要源代码。解释则是在每次程序运行时都需要解释器和源代码。这两者的区别类似于外语资料的翻译和实时的同声传译。

编译的过程只进行一次，所以编译过程的速度并不是关键，关键是生成的目标代码的执行速度。因此，编译器一般集成尽可能多的优化技术，使生成的目标代码有更高的执行效率。而解释器考虑到执行速度，不会集成太多的优化技术。

1.2 Python 简介

1.2.1　Python 的历史

Python 的作者 Guido van Rossum 是荷兰人。Guido 理想中的计算机语言，应能够方便地调用计算机的各项功能，如打印、绘图、语音等，而且程序可以轻松地编辑与运行，适合所有人学习和使用。1989 年，Guido 开始编写这种理想的计算机语言的脚本解释程序，并命名为 Python。Python 语言的目标是成为功能全面、易学易用、可拓展的语言。

Python 的第一个公开版本于 1991 年发布。它是用 C 语言实现的，能够调用 C 语言的库文件，具有类、函数、异常处理等功能，包含表和词典等核心数据类型及以模块为基础的拓展系统。

之后，在 Python 的发展过程中，存在 Python 2.x 和 Python 3.x 两个不同系列的版本，这两个版本之间不兼容。为了满足用户的需要，目前是 Python 2.x 和 Python 3.x 两个版本并存。Python 2.x 和 Python 3.x 两个不同的版本并存的原因是，Python 3.0 发布时不支持 Python 2.0 版本，导致很多用户无法正常升级使用新版本，所以后来又发布了一个 Python 2.7 的过渡版本，而且 Python 2.7 将被支持到 2020 年。Python 3.x 从 2008 年开始发布，目前的最高版本是 Python 3.6，本书的程序使用 Python 3.6 版本的编译器。

1.2.2　Python 的特点

Python 语言是目前最流行和发展最迅速的计算机语言，它具有如下特点：

（1）简单，所以易学

Python 以简单易学的特点成为学习编程的入门语言。一个良好的 Python 程序像一篇英文文档，非常接近于人的自然语言，用户在应用 Python 过程中，可以更多地专注于要解决的问题，而不必过多地考虑计算机语言的细节，回归语言的服务功能。

（2）开源，拥有众多的开发群体

用户可以查看 Python 源代码，研究其代码细节或进行二次开发。用户不需要为使用 Python 进行软件开发支付费用，不涉及版权问题。因为开源，越来越多的优秀程序员加入 Python 开发中，Python 功能愈加丰富和完善。

（3）Python 是解释型语言

Python 程序可以直接从源代码运行。在计算机内部，Python 解释器把源代码转换成称为字节码的中间形式，然后把它翻译成计算机使用的机器语言并运行。基于解释型的特点，用户可以将一些代码行在交互方式下直接测试执行，使得 Python 的学习更加容易。

（4）良好的跨平台性和可移植性

Python 的开源本质，使得它可以被移植到多个平台。如果用户的 Python 程序使用了依赖于系统的特性，Python 程序可能需要修改与平台相关的代码。Python 的应用平台包括 Linux、Windows、Mac OS、Solaris、OS/2、FreeBSD、Amiga、Android、iOS 等。

（5）面向对象

Python 既支持面向过程的编程，也支持面向对象的编程。在面向过程的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在面向对象的语言中，程序是由数据和功能组合而成的对象构建起来的。与其他主要的语言如 C++和 Java 相比，Python 以一种非常强大又简单的方式实现面向对象编程，为大型程序的开发提供了方便。

（6）可扩展性和丰富的第三方库

Python 程序可以使用 C 或 C++编写的程序，从而使某段关键代码运行得更快或者实现某些算法不公开；也可以把 Python 程序嵌入 C 或 C++程序，提高 C 或 C++程序的脚本支持能力，具有良好的可扩展性。

Python 还有功能强大的开发库。Python 标准库可以处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、HTML、WAV 文件、密码系统、GUI（图形用户界面）和其他与系统有关的操作。Python 的这些功能都是可用且完备的。除了标准库，还有大量高质量的第三方库，如 wxPython、Twisted 和 Python 图像库等。

1.2.3 Python 的应用

Python 应用领域覆盖 Web 开发、科学运算、系统运维、GUI 编程、数据库编程等诸多方面。

（1）Web 开发

Python 包含标准的 Internet 模块，可用于实现网络通信及应用。Python 的第三方框架包括 Django、Web2py、Zope 等，可以让程序员方便地开发 Web 应用程序。典型的 Web 应用如 Google 爬虫、Google 广告、世界上最大的视频网站 YouTube、豆瓣、知乎等都使用 Python 开发。

（2）科学运算

Python 广泛应用于人工智能与深度学习领域，典型的第三方库包括 NumPy、SciPy、Matplotlib 等。随着众多程序库的开发，Python 越来越适合于做科学计算、绘制高质量的 2D 和 3D 图像。例如，美国航空航天局（NASA）大量使用 Python 进行数据分析和运算。

（3）云计算

Python 是云计算领域应用最广泛的语言，典型应用如开源的云计算管理平台 OpenStack。

（4）系统运维

Python 是运维人员必备语言。Python 标准库包含多个调用操作系统功能的库。通过第三方软件包 pywin32，Python 能够访问 Windows API。使用 IronPython，Python 能够直接调用.NET Framework。一般来说，Python 编写的系统管理脚本在可读性、性能、代码重用度、扩展性几方面都优于普通的 Shell 脚本。

（5）GUI 编程

Python 可以非常简单、快捷地实现 GUI（图形用户界面）程序。Python 内置了 tkinter 的标准面向对象接口 Tk GUI API，可以非常方便地开发图形界面应用程序。还可以使用其他一些扩展包，如 wxPython、PyQt、Dabo 等，也可在 Python 中创建 GUI 应用。

1.3 Python 的开发环境

1.3.1 Python 下载和安装

Python 是一个轻量级的软件，可以在 Python 官网下载，网址如下：

https://www.python.org/downloads/

Python 开发包下载页面如图 1-1 所示，本书使用的是适合 Windows 操作系统的 Python 3.6.5 版，也可以下载适合 Linux、iOS、Android 等操作系统的 Python 开发包，或选择其他 Python 版本。

图 1-1　Python 官网下载页面

双击下载的 Python 安装程序 python-3.6.5.exe，将启动安装向导，按提示操作即可。在图 1-2 的安装页面中，选中 Add Python 3.6 to PATH 复选框，将 Python 的可执行文件路径添加到 Windows 操作系统的环境变量 path 中，方便在将来的开发中启动各种 Python 工具。

图 1-2　安装程序界面

安装成功后的界面如图 1-3 所示，并会在 Windows 操作系统的“开始”菜单中显示图 1-4 所示的 Python 工具。这些工具的含义如下：

- IDLE (Python 3.6 32-bit)，用于启动 Python 软件包自带的集成开发环境 IDLE。
- Python 3.6 (32-bit)，用于以命令行方式启动 Python 的解释器。
- Python 3.6 Manuals (32-bit)，用于打开 Python 的帮助文档。
- Python 3.6 Module Docs (32-bit)，用于以内置服务器的方式打开 Python 的模块帮助文档。

在学习 Python 的过程中，通常使用 Python 自带的集成开发环境 IDLE。

图 1-3　Python 安装成功界面

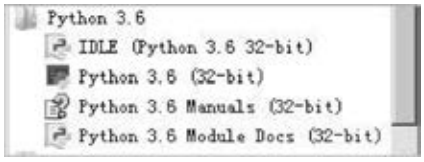

图 1-4 “开始”菜单中的 Python 命令

Python 安装时，在 Windows 10 操作系统下，默认的安装路径是 C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32。如果用户想要自己定义 Python 解释器的安装路径，可以在图 1-2 中选择“Customize installation”选项，并可以选择需要安装的部件。

1.3.2　Python 的 IDLE 开发环境

Python 是一种脚本语言，开发 Python 程序首先要在文本编辑工具中书写 Python 程序，然后由 Python 解释器执行。用户选择的编辑工具可以是记事本、notepad+、EditPlus 等。Python 开发包自带的编辑器 IDLE 是一个集成开发环境（integrated development environment，IDE），启动文件是 idle.bat，它位于安装目录的 Lib\idlelib 文件夹下。用户可以在“开始”菜单的“所有程序”组中选择 Python 3.6 分组下面的 IDLE (Python 3.6 32-bit)菜单项，打开 IDLE 窗口，如图 1-5 所示。

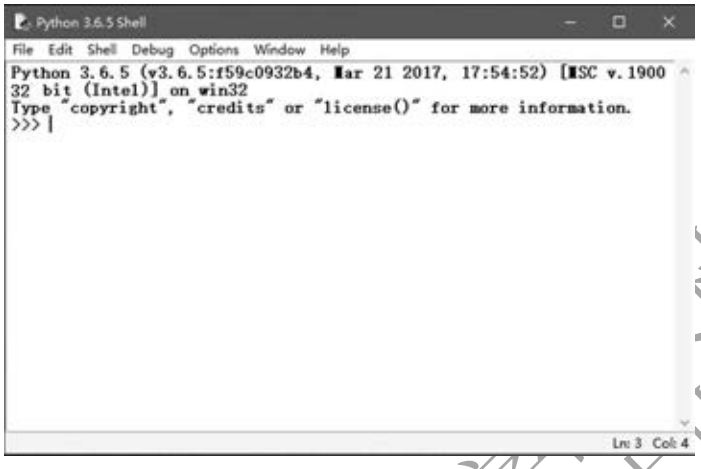

图 1-5　IDLE 窗口

在 IDLE 环境下，编写和运行 Python 程序（也称为 Python 脚本）的主要操作如下：

（1）新建 Python 程序

在 IDLE 菜单中依次选择[File]/[New File]命令，或按 Ctrl+N 组合键，就可以新建 Python 的脚本程序，窗口的标题显示程序名称，初始的文件名为 Untitled，表示 Python 程序还没有保存。

（2）保存 Python 程序

在 IDLE 菜单中依次选择 [File]/[Save File]命令，或按 Ctrl+S 组合键，即可保存

Python 程序。如果是第一次保存，则会弹出保存文件对话框，要求用户输入保存的文件名。

（3）打开 Python 程序

在 IDLE 菜单中依次选择[File]/[Open File]命令，或按 Ctrl+O 组合键，会弹出打开文件对话框，要求用户选择要打开的 Python 文件。

（4）运行 Python 程序

在 IDLE 菜单中依次选择[Run]/[Run Module]命令，或按 F5 键，可以在 IDLE 中运行当前的 Python 程序。

如果程序中存在语法错误，运行时会弹出对话框，显示“invalid syntax”，一个浅红色方块会定位在错误处。

（5）语法高亮

IDLE 支持 Python 的语法高亮，即 IDLE 能够以彩色标示出 Python 语言的关键字，提醒开发人员该词的特殊作用。例如，注释以红色显示，关键字以紫色显示，字符串以绿色显示。

（6）常用快捷键

IDLE 除了支持撤销、全选、复制、粘贴、剪切等常规快捷键外，还提供了其他多个快捷键。使用 IDLE 的快捷键能显著提高代码编辑速度和开发效率。IDLE 的常用快捷键如表 1-1 所示。

表 1-1　IDLE 的常用快捷键

快捷键	功能说明
Ctrl + [	缩进代码
Ctrl +]	取消缩进代码
Alt+3	注释代码行
Alt+4	取消注释代码行
Alt+/	单词自动完成，只要文中出现过某单词，就可以帮助编程者自动补齐。多按几次可以循环选择
Alt+P	浏览历史命令（上一条）
Alt+N	浏览历史命令（下一条）
F1	打开 Python 帮助文档
F5	运行程序
Ctrl+F6	重启 Shell，之前定义的对象和导入的模块全部失效

1.3.3　PyCharm 集成开发环境

IDLE 是 Python 开发包自带的集成开发环境，功能相对简单。PyCharm 则是由 JetBrains 公司开发的一款专业级的 Python IDE。PyCharm 具有典型 IDE 的多种功能，如程序调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试、版本控制等功能。

1. PyCharm 的下载和安装

访问 PyCharm 官方网址 http://www.jetbrains.com/pycharm/download/，进入 PyCharm 的下载页面，如图 1-6 所示。

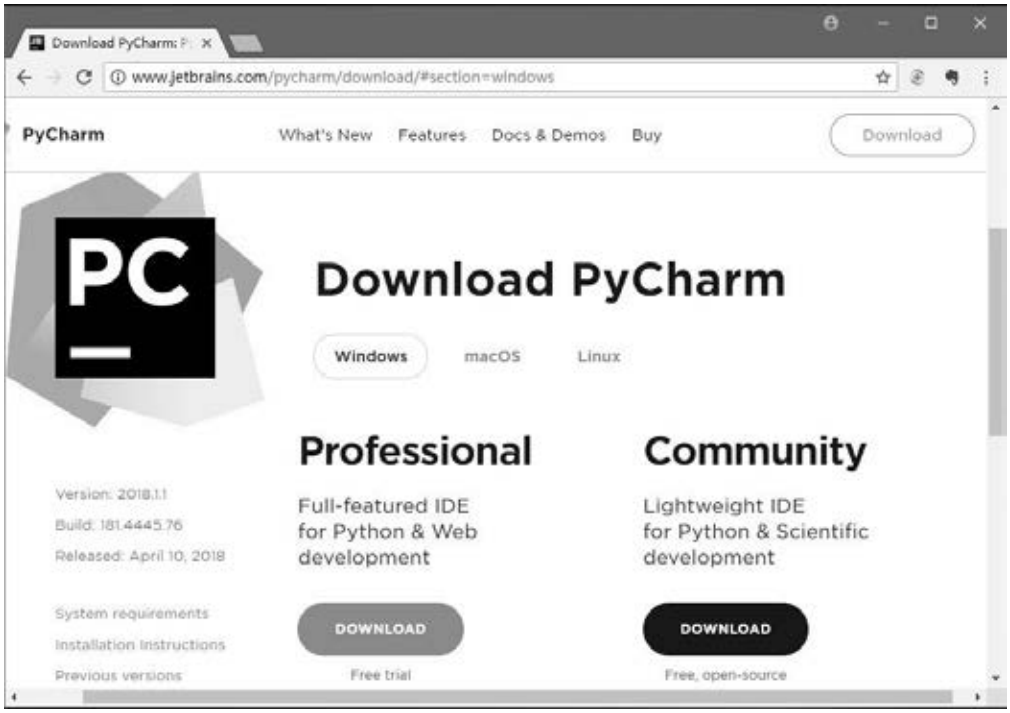

图 1-6　PyCharm 的下载页面

用户可以根据自己的操作系统平台下载不同版本的 PyCharm，并且每个平台可以选择下载 Professional 和 Community 两个版本。PyCharm Professional 是需要付费的版本，它提供 Python IDE 的所有功能，支持 Web 开发，支持 Django、Flask、Google App 引擎、Pyramid 和 web2py 等框架，同时支持远程开发、Python 分析器、数据库和 SQL 语句等。PyCharm Community 版本是轻量级的 Python IDE，是免费和开源的版本，但只支持 Python 开发，适合初学者学习使用。如果要开发 Python 的应用项目，则需要 Professional 版提供丰富的功能。

安装 Pycharm 的过程十分简单，只要按照安装向导的提示逐步安装即可。图 1-7 是安装过程中选择安装路径的界面。安装完成后的界面如图 1-8 所示。

2. 建立 Python 项目和文件

第一次启动 PyCharm 时，会显示若干初始化的提示信息，选择使用默认值即可。之后，进入创建项目的界面。如果不是第一次启动 PyCharm 并且以前创建过项目，项目会出现在窗口中，如图 1-9 所示。界面中的 3 个选项的含义分别是“创建新项目”“打开已经存在的项目”“从版本控制中检测项目”。

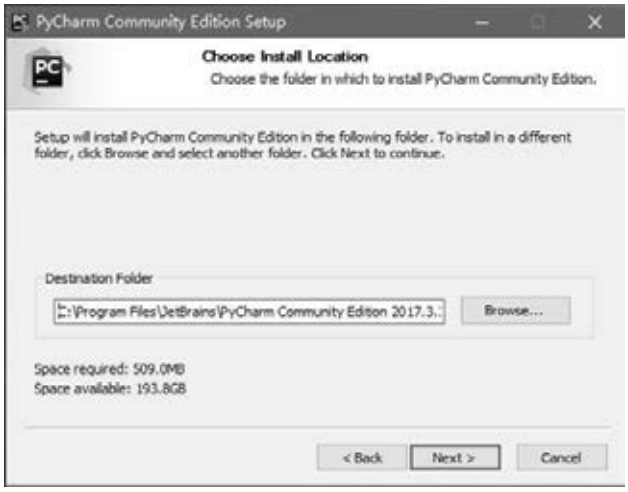

图 1-7　选择 PyCharm 的安装路径

图 1-8　安装成功界面

图 1-9　创建 Python 项目界面

（1）创建项目

当选择 Create New Project 选项创建项目后，弹出选择项目存放路径界面，如图 1-10 所示。

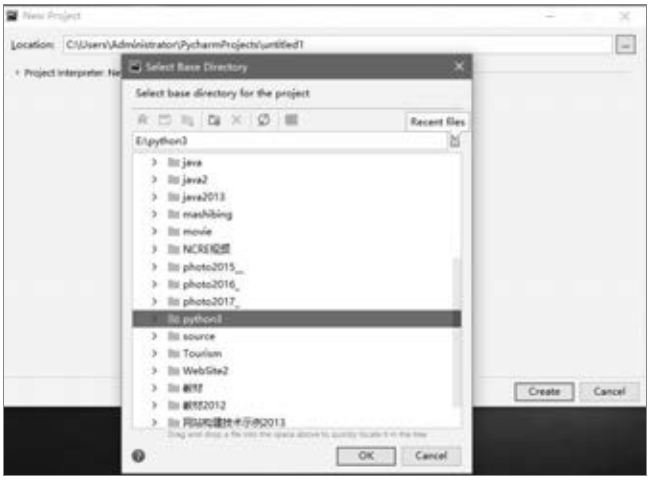

图 1-10　选择项目存放路径

（2）新建文件

项目创建完成后，如果要在项目中创建 Python 文件，则选中项目名称并右击，在弹出的快捷菜单中选择[New]/[Python File]命令，如图 1-11 所示。

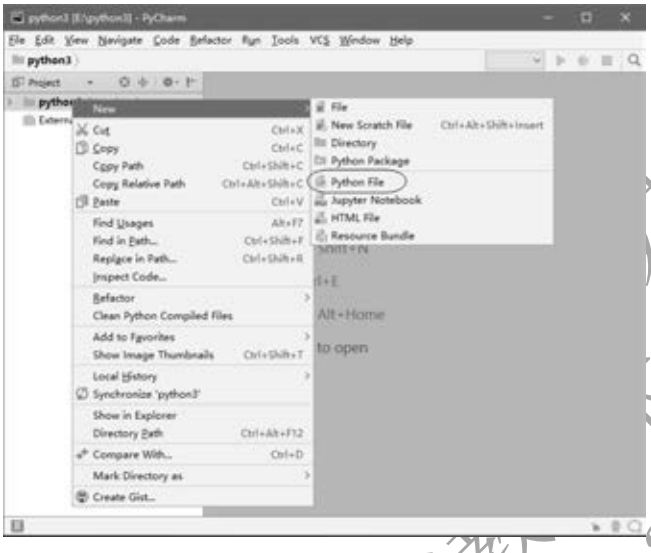

图 1-11　在项目中建立 Python 文件

（3）保存和运行文件

在程序编辑窗口中输入代码后，可以保存文件，并选择 Run 菜单中的命令运行程序。图 1-12 中书写了一个完整的程序，使用 Run 菜单下的命令可以调试和运行程序。

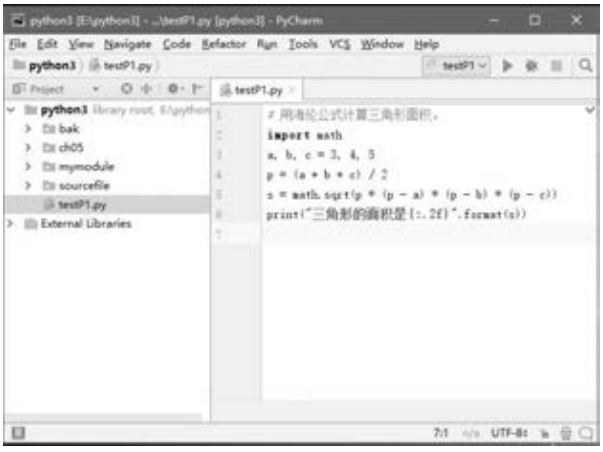

图 1-12　编辑程序文件

1.4 Python 程序执行过程

1.4.1　Python 程序执行原理

Python 是一种脚本语言，编辑完成的源程序也叫源代码，可以直接运行。从计算机的角度看，Python 程序的运行过程包含两个步骤：解释器将源代码翻译成字节码，然后由虚拟机解释执行，如图 1-13 所示。

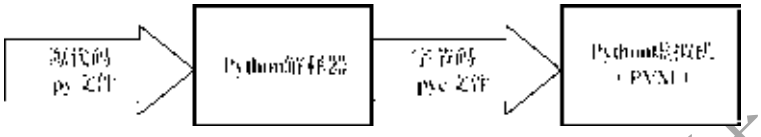

图 1-13　Python 程序的执行原理

Python 代码源文件的扩展名通常为 .py。在执行时，首先由 Python 解释器将.py 文件中的源代码翻译成中间代码，中间代码是一个扩展名为 .pyc 的文件，再由 Python 虚拟机（Python virtual machine，PVM）逐条将字节码翻译成机器指令执行。

需要说明的是，.pyc 文件保存在 Python 安装目录的__pycache__文件夹下，如果 Python 无法在用户的计算机上写入字节码，字节码文件将只在内存中生成，并在程序结束运行时自动丢弃。而主文件（直接执行的文件）因为只需要装载一次，并没有保存 .pyc 文件。当 Python 源文件用于导入时，将生成 .pyc 文件，并且在__pycache__文件夹下可以看到该文件。

.pyc 文件可以重复使用，并可以提高执行效率。

1.4.2　建立和运行 Python 程序

前面提到的 Python 文件、Python 程序、Python 程序文件是同义的，都是指 Python

的程序。程序是由若干行代码组成的，通常用于完成一定的功能。

运行 Python 程序有两种方式：交互方式和文件方式。交互方式指 Python 解释器即时响应用户输入的程序代码，并运行，如果有输出，则显示结果。文件方式即编程方式，用户将 Python 代码写在程序文件中，然后启动 Python 解释器批量执行文件中的代码。交互式一般用于调试少量代码，文件式则是最常用的编程方式。多数计算机的编程语言只有文件执行方式，Python 的交互模式主要实现了代码的易学、易理解。下面以 Windows 操作系统中求一组数值中的最大值和最小值的程序为例来说明两种方式的启动和执行方法。

1. Python 交互执行方式

在 Windows“开始”菜单中执行[开始]/[Python 3.6]/[Python 3.6 (32-bit)]命令，启动 Python 交互式运行环境，逐行输入代码，实现求一组数据中最大值和最小值的程序，每输入一条语句并换行后，直接交互执行，如图 1-14 所示。

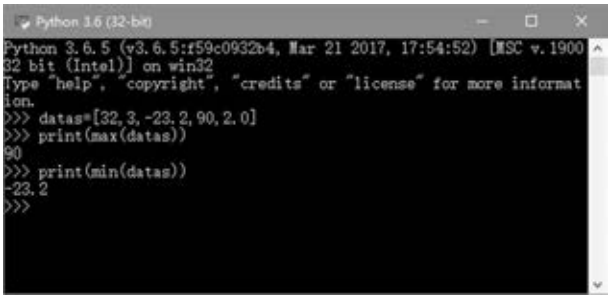

图 1-14　在 Python 交互模式下执行代码

每行代码均以 Enter 键结束，之后立即执行。如果是打印语句则显示输出结果。

在“>>>”提示符后输入 exit()或者 quit()，可以退出 Python 运行环境。

2. IDLE 交互执行方式

前面已经介绍过，IDLE 是 Python 内置的集成开发环境，在 Windows“开始”菜单中执行[开始]/[Python 3.6]/[IDLE(Python 3.6 32-bit)]命令，启动 IDLE 交互方式，输入代码，实现求一组数据中最大值和最小值的程序，每输入一条语句后，即直接交互执行，如图 1-15 所示。

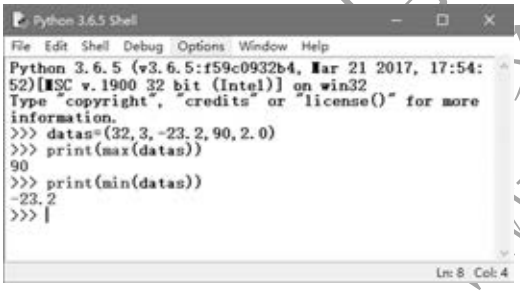

图 1-15　在 IDLE 交互模式下执行代码

比较 Python 交互方式和 IDLE 交互方式，可以看出，代码执行的过程非常相似。但 IDLE 交互式提供了更多快捷的操作方式，比 Python 交互式使用起来更加方便。另外，上面两个例子中的数据分别用方括号[]和圆括号()标记，这是两种不同的组合数据类型，将在第 4 章介绍；max()和 min()是 Python 的内置函数，将在第 3 章介绍。

3. IDLE 程序文件执行方式

在 Windows“开始”菜单中执行[开始]/[Python 3.6]/[IDLE(Python 3.6 32-bit)]命令，启动 IDLE，打开 IDLE 窗口，执行[File]/[New File]命令，或按 Crtl+N 组合键，打开一个程序编辑窗口，在其中输入程序代码，如图 1-16 所示。

图 1-16　在 IDLE 下编写并运行程序

这个程序编辑窗口不是交互窗口，而是 IDLE 的集成开发环境，该环境具备 Python 语法高亮辅助的编辑器，可以进行代码编辑。在其中输入 Python 程序代码后，将程序保存为以.py 为扩展名的文件，如 program1.py，按 F5 键，或在菜单中选择[Run]/[Run Module]命令，将在 IDLE 环境中显示运行结果。如果程序出现错误，将给出错误提示，用户修改程序后，可以继续调试运行。

上面 3 种代码运行方式中，IDLE 的交互方式适用于初学者学习语句或函数的功能，每执行一行代码即可以看到运行结果，简单直观，但程序代码无法保存。IDLE 的程序文件方式适合书写多行代码，方便用户编程，在实际开发中使用得比较多。

除此之外，在 Windows 操作系统中，双击 Python 源文件也可以执行程序，这种方式在实际应用中较少使用。

1.5 Python 程序方法与应用

1.5.1　程序设计方法

程序是完成一定功能的语句的集合，用于解决特定的计算问题。按照软件开发的思

想，程序设计可以分为分析、设计、实现、测试运行等阶段。结构化程序设计是一种典型的程序设计方法，是程序设计的基础思想，它把一个复杂程序逐级分解成若干个相互独立的程序，然后设计与实现每个程序。

程序在具体实现上遵循一定的模式，典型的程序设计模式是 IPO 模式，即程序包括输入（input）、处理（process）、输出（output）3 部分。输入是程序设计的起点，包括文件输入、网络输入、交互输入、参数输入等。输出是程序展示运算成果的方式，包括文件输出、网络输出、控制台输出、图表输出等。处理部分则是编程的核心，包括数据处理与赋值，更重要的是算法。例如，给定两点的坐标，求两点的距离，需要一个公式，这个公式就是一个算法；再如，求三角形面积的海伦公式也是一个算法；更多的算法需要用户去设计，例如，从一组数据中查找某一数据的位置，需要根据数据的特点，由用户设计算法。

除了 IPO 模式外，程序中通过添加足够的注释加强程序的可读性，通过调试进一步完善程序，都是程序设计中不可缺少的环节。

从上述内容可以看出，使用计算机编程解决计算问题主要包括下列步骤：分析问题、设计算法、编写程序、调试运行等。其中，与程序设计语言和具体语法有关的步骤是编写程序和调试运行。在解决计算问题过程中，编写程序只是一个环节。在此之前，分析问题、设计算法都是重要步骤，经过这些步骤，一个计算问题已经能够在设计方案中被解决了，这个过程可以看作思维的创造过程；编写程序和调试测试则是解决方案的计算机实现，属于技术实现过程。

1.5.2　程序设计示例

前面已经介绍了程序文件的建立和执行过程，下面给出 9 个简单的 Python 程序，方便读者了解 Python 的基本知识点，这些程序涉及 IDLE 下交互执行程序、程序的分支与循环结构、函数等内容。

读者通过查阅文档了解这些程序，会显著提高后面章节的学习效率；也可以忽略一些程序的具体语法含义，在后面章节中逐步学习。

例 1-1　根据圆的半径计算圆的面积和周长。

```
#program0101.py
#计算圆形面积和周长
r=3.2
area=3.14*r*r
perimeter=2*3.14*r
print("圆形的面积:{:.2f},周长:{:.2f}".format(area,perimeter))
```

本例知识点在第 2 章。

程序的第 1 行和第 2 行是注释，注明程序的名字和功能，注释语句不运行，可以书写任何描述或代码。

程序的第 3 行是赋值语句，将值 3.2 赋给半径 r。

程序的第 4 行和第 5 行，用 3.14*r*r 和 2*3.14*r 公式计算圆面积和周长。该行是程序的核心，是程序的算法实现。

第 6 行是打印语句，程序的运行结果是“圆形的面积:32.15,周长:20.10”。

按程序设计的 IPO 模式，该段代码没有明显的输入，而是采用赋值输入的形式，处理或算法是求圆面积和周长的公式，输出是一个打印语句。具体的语法细节请读者查阅文档。

例 1-2 IDLE 交互方式下，根据圆的半径计算圆的面积和周长。

```
>>> r=3.2
>>> area=3.14*r*r
>>> print("圆形的面积:{:.2f}".format(area))
圆形的面积:32.15
>>> perimeter=2*3.14*r
>>> print("圆形的半径:{:.2f},周长:{:.2f}".format(r,perimeter))
圆形的半径:3.20,周长:20.10
```

可以看出，每行输入结束后，代码立即执行。

例 1-3 输入三角形 3 条边长，用海伦公式计算三角形面积。

```
1  #program0103.py
2  #输入三角形 3 条边长，用海伦公式计算三角形面积 s
3  import math
4  a=eval(input("请输入 a 边长："))
5  b=eval(input("请输入 b 边长："))
6  c=eval(input("请输入 c 边长："))
7  p=(a+b+c)/2
8  s=math.sqrt(p*(p-a)*(p-b)*(p-c))
9  print("三角形的面积是{:.2f}".format(s))
```

本例主要知识点在第 3 章和第 5 章。

第 3 行导入 math 模块。导入 math 模块后，可以使用第 8 行的 math.sqrt()方法。

第 4 行、第 5 行、第 6 行使用 input()函数接收用户的键盘输入，并使用 eval()函数将输入转换为数值类型，从而可以参与数学运算。

第 7 行是计算赋值语句，计算 3 条边的和除以 2，赋给变量 p。

第 8 行是程序的算法，是程序的核心代码，用海伦公式计算三角形面积，并赋给变量 s。

第 9 行是打印语句，输出程序的运行结果。

例 1-3 如果输入的数据不是数值，例如，输入 a11、ab 等形式，运行时将产生错误。为避免这种情况发生，例 1-4 加以改进，进行了异常处理。如果无法读懂该程序，可以在学习异常处理后继续读该程序。

例 1-4 输入三角形 3 边长，用海伦公式计算三角形面积，对输入数据进行了异常处理。

```
#program0104.py
'''
输入三角形 3 条边长，用海伦公式计算三角形面积 s
对 3 边数据进行了异常处理
'''
import math
try:
   a=eval(input("请输入 a 边长："))
   b=eval(input("请输入 b 边长："))
   c=eval(input("请输入 c 边长："))
   p=(a+b+c)/2
   s=math.sqrt(p*(p-a)*(p-b)*(p-c))
   print("三角形的面积是{:.2f}".format(s))
except NameError:
   print("请输入正数数值")
```

本例主要知识点在第 8 章异常处理部分。程序运行结果如下，输入数据错误时，给出提示“请输入正数数值”。

```
>>>
请输入 a 边长：8
请输入 b 边长：e4
请输入正数数值
>>>
```

当输入 3 条边的边长不符合构成三角形的条件时，需要做出处理。

例 1-5　用海伦公式计算三角形面积，判断构成三角形的条件。

```
#program0105.py
'''
输入三角形 3 条边长，用海伦公式计算三角形面积 s
在对 3 边数据进行了异常处理基础上，判断 3 边符合构成三角形条件
'''
import math
try:
   a=eval(input("请输入 a 边长："))
   b=eval(input("请输入 b 边长："))
   c=eval(input("请输入 c 边长："))
except NameError:
   print("请输入正数数值")
if a<0 or b<0 or c<0:
   print("输入数据不可以为负数")
elif a+b<=c or a+c<=b or b+c<=a:
   print("不符合两边之和大于第三边原则")
else:
   p=(a+b+c)/2
```

```
19      s=math.sqrt(p*(p-a)*(p-b)*(p-c))
20      print("三角形的面积是{:.2f}".format(s))
```

本例知识点主要在第 3 章和第 8 章。

上面的代码用分支语句判断 3 条边构成三角形的条件，某次运行结果如下，分支语句将在第 3 章学习。

```
>>>
请输入 a 边长：1
请输入 b 边长：2
请输入 c 边长：3
不符合两边之和大于第三边原则
>>>
```

例 1-6　给出用列表保存的一组数据，求数据的平均值。

```
1   #program0106.py
2   lst=[89,5,-34,23.1]
3   total=sum(lst)
4   number=len(lst)
5   print("列表 lst 的平均值是",total/number)
```

例 1-7　给出用列表保存的一组成绩数据，统计不及格的人数和最高分。

```
 1   #program0107.py
 2   lst=[89,45,-34,23.1,98,33]
 3   #notpass 为不及格人数，maxscore 为最高分
 4   notpass=maxscore=0
 5   for item in lst:
 6       if maxscore<item:
 7           maxscore=item
 8       if item<60:
 9           notpass+=1
10   print("最高分是{}，不及格人数是{}".format(maxscore,notpass))
```

本例知识点主要在第 3 章和第 4 章，遍历列表实现数据统计。

例 1-8　用函数式统计列表中的不及格的人数和最高分。

```
#program0108.py
lst=[89,45,-34,23.1,98,33]
maxscore=max(lst)                    #最高分
lst2=filter(lambda x:x<60,lst)       #不及格数据的序列
notpass=len(list(lst2))              #不及格人数
print("最高分是{}，不及格人数是{}".format(maxscore,notpass))
```

例 1-9　文本文件中保存一组用逗号分隔的成绩数据，统计不及格的人数和最高分。文本文件名为 data.txt，内容是“89,45,-34,23.1,98,33,56,98”。

```
#program0109.py
file=open("file.txt",'r')
s1=file.read()
file.close()
lst=s1.split(',')
lst2=[]
for item in lst:
    lst2.append(eval(item))
#print(lst2)
#notpass 为不及格人数，maxscore 为最高分
notpass=maxscore=0
for item in lst2:
    if maxscore<item:
        maxscore=item
    if item<60:
        notpass+=1
print("最高分是{}，不及格人数是{}".format(maxscore,notpass))
```

本例主要知识点在第 7 章。

代码第 2～4 行读取文件内容；第 5～8 行拆分字符串和解析字符串内容（转换为数字）；之后，完成数据统计功能。

小　结

本章介绍了计算机语言的概念，机器语言、汇编语言、高级语言的区别。按照计算机程序的执行方式，将高级语言分为静态语言和脚本语言两类。静态语言采用编译执行的方式，脚本语言采用解释执行的方式。

目前是 Python 2.x 和 Python 3.x 两个版本并存，这两个版本之间不兼容。Python 应用领域覆盖 Web 开发、科学计算、系统运维、GUI 编程、数据库编程等诸多方面。

Python 安装包可以在 Python 的官网下载。Python 解释器内置集成开发工具 IDLE。PyCharm 是由 JetBrains 公司开发的一款专业级的 Python IDE，具有程序调试、语法高亮、智能提示等功能。

Python 代码源文件的扩展名通常为.py。在执行时，首先由 Python 解释器将.py 文件中的源代码翻译成中间代码，再由 Python 虚拟机将字节码逐条翻译成机器指令执行。

运行 Python 程序有交互方式和文件方式两种方式。交互方式指 Python 解释器即时响应用户输入的每行程序代码。文件方式即编程方式，用户将 Python 程序写在程序文件中，然后启动 Python 解释器批量执行文件中的代码。

典型的程序设计模式是 IPO 模式，即程序包括输入（input）、处理（process）、输出（output）3 部分，并介绍了一些程序的典型示例。

习　题

1. 选择题

（1）下面不属于 Python 特性的是（　　）。

A．简单易学　　B．开源的、免费的

C．属于低级语言　　D．高可移植性

（2）下列计算机语言中，不属于解释型语言的是（　　）。

A．Python　　B．JavaScript

C．C++　　D．HTML

（3）Python 脚本文件的扩展名是（　　）。

A．.pyc　　B．.py　　C．.pt　　D．.pyw

（4）下面（　　）方面的应用不适合使用 Python 开发。

A．科学计算　　B．系统运维　　C．网站设计　　D．数据库编程

（5）下列关于 Python 版本的说法中，正确的是（　　）。

A．目前存在 Python 3.x 兼容 Python 2.x 版本的程序

B．Python 2.x 版本需要升级到 Python 3.x 版本才能使用

C．目前的 Python 2.x 版本已经被淘汰

D．Python 2.x 和 Python 3.x 是两个不兼容的版本

2. 简答题

（1）简述 Python 程序的执行过程。

（2）列举出 IDLE 编程环境 5 个以上快捷键的功能。

（3）简述程序的编译方式和解释方式的区别。

（4）简述程序设计的 IPO 模式的特点。

3. 编程题

（1）参考例 1-3，输入三角形的底边长和高，计算并输出三角形的面积。

（2）参考例 1-6，在列表中给出若干字符串，计算并输出最长的字符串。

（3）查阅 Python 的帮助文档，查找其中的 Numeric Types 类型，试使用其中的函数计算一组数中的最大值和最小值。

第 2 章
Python 基础知识

用计算机语言书写的程序称为源程序，也叫源代码。书写程序要注意语句的格式、语法约束、保留字等内容。本章介绍如何书写 Python 语言程序，Python 的数据类型约束、变量和运算符等内容。

2.1 程序的书写规范

在 Python 的代码编辑环境中，程序的书写规范主要体现在语句的格式、代码块与缩进、注释等方面。

2.1.1 Python 的语句

Python 通常是一行书写一条语句，如果一行内写多条语句，要求使用分号分隔。建议每行只写一条语句，并且语句结束时不写分号。

如果一条语句过长，可能需要换行书写，这时可以在语句的外部加上一对圆括号来实现，也可以使用“\”（反斜线）来实现分行书写功能。

与写在圆括号()中的语句类似，写在[]、{}内的跨行语句被视为一行语句，不再需要使用反斜线换行。

例 2-1　Python 语句的分行书写。

```
str1=("当一条语句过长时，可能需要进行换行处理，这时可以\
      在语句的外侧加上一对圆括号来实现。也可以使用反斜线\
      来分行书写。")                              #第 1 种写法，用\续行

str2=("当一条语句过长时，可能需要进行换行处理，这时可以"
      "在语句的外侧加上一对圆括号来实现。""也可以使用反"
      "斜线来分行书写。")                         #第 2 种写法

months=['january','february','march',"april",'may','june','july','august',
        'september','october','november','','december']  #写在[]内的代码
```

上面的代码中，用单引号和双引号括起来的都是字符串，语句前后的空格是在 IDLE 环境中换行自动产生的，也可以删除。

2.1.2 代码块与缩进

代码块也称为复合语句，由多行代码组成，这些代码能完成相对复杂的功能。Python 中的代码块使用缩进来表示，缩进是指代码行前部预留若干空格。其他一些计算机语言，如 C 语言、Java 语言，使用花括号{ }表示代码块。

Python 语句行缩进的空格数在程序编辑环境中是可调整的，但要求同一个代码块的语句必须包含相同的缩进空格数。

看下面的表示程序分支执行的示例代码。

例 2-2　Python 语句的缩进和代码块。

```
#分支语句中代码块的缩进
score=54
pass=60
if score>pass:
    gpoint=1+(score-pass)/10
    print("学分绩点为",gpoint)
    print("通过考试")
else:
    print("学分绩点为 0")
    print("未通过考试")
```

上面的代码中，if 语句后缩进的 3 行构成一个代码块，else 语句后缩进的 2 行也构成一个代码块。如果同一代码块中各语句前的空格数不一致，运行时将报告出错信息。

关于代码的缩进，需要注意：

- 虽然 Python 代码行缩进可以调整，但建议读者使用 4 个空格宽度的行首缩进。
- 不同的文本编辑器中制表符（Tab 键）表示的空白宽度不一致，如果读者编写的代码可能要跨平台使用，建议不使用制表符。

2.1.3 注释

注释用于说明程序或语句的功能。Python 的注释分为单行注释和多行注释两种。单行注释以“#”开头，可以是独立的 1 行，也可以附在语句的后部。多行注释可以使用 3 个引号（英文的单引号或双引号）作为开始和结束符号。

例 2-3　Python 的注释。

```
'''
使用 math 库中的 pi 常数，计算圆的面积和体积。
math 库是 Python 的内置数学函数库，需要导入
后使用
上面是多行注释
```

```
'''
#程序。用分支判断半径 r 的值     ——单行注释
import math
r=-2
if r>0:
    area=math.pi*r*r   #附在语句后的单行注释
    print(area)
else:
    print("半径为负，请修改程序")
```

多行注释通常用来说明程序的功能、作者、完成时间、输入/输出等，单行注释一般用来解释代码行的功能。

2.2 标识符和关键字

标识符和关键字是计算机语言的基本语法元素，是编写程序的基础，不同计算机语言的标识符和关键字略有区别。

2.2.1 标识符

计算机中的数据，如变量、方法、对象等都需要有名称，以方便程序使用。这些用户定义的、由程序使用的符号就是**标识符**。用户可以根据程序设计的需要来定义标识符，规则如下：

- Python 的标识符可以由字母、数字和下划线“_”组成，且不能以数字开头。
- 标识符区分大小写，没有长度限制。
- 标识符不能使用计算机语言中预留有特殊作用的关键字。
- 标识符的命名尽量符合见名知意的原则，从而提高代码的可读性。例如，程序中的用户名使用 username 来表示，学生对象使用 student 来表示。

下面是 Python 中合法的标识符：

```
myVar    _Variable    姓名
```

下面是 Python 中非法标识符：

```
2Var    vari#able    finally    stu@lnnu    my name
```

2.2.2 关键字

Python 语言保留某些单词用做特殊用途，这些单词被称为关键字，也叫保留字。用户定义的标识符（变量名、方法名等）不能与关键字相同，否则编译时就会出现异常。Python 常用的关键字如表 2-1 所示。

表 2-1　Python 常用的关键字

and	as	assert	break	class	continue
def	del	elif	else	except	False
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
None	pass	raise	return	True	try
while	with	yield			

在 Python 中，需要注意 True、False、None 的写法，与其他计算机语言是不同的。如果用户需要查看关键字的信息，可以使用 help()命令进入帮助系统查看。

例 2-4　使用 Python 的帮助功能，显示提示信息。

```
>>> help()                #进入 Python 的帮助系统
help> keywords            #查看关键字列表
help> break               #查看 break 关键字说明
help> quit                #退出帮助系统
```

2.3 Python 的数值类型与变量

计算机程序设计的目的是存储和处理数据，将数据分为合理的类型既可以方便数据处理，也可以提高处理效率，节省存储空间。数据类型指明了数据的状态和行为。Python 的数据类型包括数值类型、字符串类型、列表类型、元组类型等。程序员在程序中使用变量来临时保存数据，变量用标识符来命名。

2.3.1　数值类型

数值类型（number）是 Python 的基本数据类型，包含整型、浮点型、复数类型和布尔类型等 4 种。

1. 整数类型（int）

整数类型简称整型，它与数学中整数的概念一致。整型的表示方式有 4 种，分别是十进制、二进制（以“0B”或“0b”开头）、八进制（以“0o”或“0O”开头）和十六进制（以“0x”或“0X”开头）。

Python 的整型理论上的取值范围是[−∞,∞]，实际取值范围受限于运行 Python 程序的计算机内存大小。下面是一些整数类型的数据。

```
100    21    0O234    0o67    0B1011    0b1101    0x1FF    0X1DF
```

Python 有多种数据类型，并且有些数据类型的表现形式相同或相近，可使用 Python 的内置函数 type()来测试各种数据类型。

例 2-5　使用 type()函数测试数据类型。

```
>>> x=0O234
>>> y=0B1011
>>> z=0X1DF
>>> print(x,y,z)
156 11 479
>>> type(x),type(y),type(z)
(<class 'int'>,<class 'int'>,<class 'int'>)
```

上述代码中定义了 3 个变量，变量的内容将在下一小节介绍。第 1 行代码中，变量 x 的值是一个八进制的整数；第 2 行代码中，变量 y 的值是一个二进制的整数；第 3 行代码中，变量 z 的值是一个十六进制的整数，它们都属于 int 类型。运行结果是 x、y、z 这 3 个变量的十进制值，并显示了它们的数据类型。

2. 浮点型（float）

浮点型用于表示数学中的实数，是带有小数的数据类型。例如，3.14、10.0 都属于浮点型。浮点型可以用十进制或科学记数法表示。下面是用科学记数法表示的浮点型数据。

```
3.22e3    0.24E6    1.5E-3
```

E 或 e 表示基数是 10，后面的整数表示指数，指数的正负使用“+”或者“-”表示，其中，“+”可以省略。需要注意的是，Python 的浮点型数据占 8 字节，能表示的数的范围是-1.8^{308}～1.8^{308}。

3. 复数类型（complex）

复数类型用于表示数学中的复数，例如，5+3j、-3.4-6.8j 都是复数类型。多数计算机语言没有复数类型，Python 中的复数类型具有以下特点：

- 复数由实数部分 real 和虚数部分 imag 构成，表示为 real+imagj 或 real+imagJ。
- 实数部分 real 和虚数部分 imag 都是浮点型。

需要说明的是，一个复数必须有表示虚部的实数和 j，如 1j、-1j 都是复数，而 0.0 不是复数；并且表示虚部的实数部分即使是 1 也不可以省略。复数的示例代码如下，从运行结果可以看出，复数的实部和虚部都是浮点数。

例 2-6　复数类型测试。

```
>>> f1=3.3+2j
>>> print(f1)
(3.3+2j)
>>> type(f1)
<class 'complex'>
>>> f1.real
3.3
```

```
>>> f1.imag
2.0
```

4. 布尔类型（bool）

布尔类型可以看做一种特殊的整型，布尔型数据只有两个取值：True 和 False。如果将布尔值进行数值运算，True 会被当作整型 1，False 会被当作整型 0。每一个 Python 对象都自动具有布尔值（True 或 False），进而可用于布尔测试（如用在 if 结构或 while 结构中）。

以下对象的布尔值都是 False：None、False、整型 0、浮点型 0.0、复数 0.0+0.0j、空字符串""、空列表[]、空元组()、空字典{}，这些数据的值可以用 Python 的内置函数 bool() 测试。

例 2-7　布尔类型测试。

```
>>> x1=0
>>> type(x1),bool(x1)
(<class 'int'>,False)
>>> x2=0.0
>>> type(x2),bool(x2)
(<class 'float'>,False)
>>> x3=0.0+0.0j
>>> type(x3),bool(x3)
(<class 'complex'>,False)
>>> x4=""
>>> type(x4),bool(x4)
(<class 'str'>,False)
>>> x5=[]                    #列表类型
>>> type(x5),bool(x5)
(<class 'list'>,False)
>>> x6={}                    #字典类型
>>> type(x6),bool(x6)
(<class 'dict'>,False)
```

2.3.2　变量和常量

1. 变量

变量是计算机内存的存储位置的表示，也叫内存变量，用于在程序中临时保存数据。变量用标识符来命名，变量名区分大小写。Python 定义变量的格式如下：

```
varName=value
```

其中，varName 是变量名，value 是变量的值，这个过程称作为变量赋值，“=”称为赋值运算符，即将“=”后面的值传递给前面的变量名。

关于变量，使用时需要注意下面的问题：

● 计算机语言中的赋值是一个重要的概念，例如，x=8，含义是将 8 赋予变量 x；而 x=x+1 赋值运算的含义是将 x 加 1 之后的值再送给 x，x 的值是 9，这与数学中的等于（=）含义是不同的。

● Python 中的变量具有类型的概念，变量的类型由所赋的值来决定。在 Python 中，只要定义了一个变量，并且该变量存储了数据，那么变量的数据类型就确定了，系统会自动识别变量的数据类型。例如，若 x=8，则 x 是整型数据；若 x="Hello"，则 x 是一个字符串类型。变量也可以是列表、元组或对象等类型。

如果希望查看变量的类型，可以使用函数 type(varName)来实现。

2. 常量

与变量对应，计算机语言中还有常量的概念，常量就是在程序运行期间值不发生改变的量。实质上，常量是内存中用于保存固定值的单元。常量也有各种数据类型。例如，"Python"、3.14、100、True 都是常量，其类型定义与 Python 的数据类型是一致的。

2.4 Python 的字符串类型

字符串（str）是一种表示文本的数据类型，字符串的表示、解析、处理是 Python 的重要内容，也是 Python 编程的基础之一。

2.4.1 字符串的表示

1. 字符串的定义

Python 中的字符串被定义为一个字符集合，它被引号所包含，引号可以是单引号、双引号或者三引号（3 个连续的单引号或者双引号）。

单引号和双引号包含的是单行字符串，两者的作用相同。三引号可以包含多行字符串。下面代码分别表示出 3 种类型的字符串。

```
'college'  '12'  'true' 'st"ude"nt'        #单引号包围的字符串
"student"  "id"  "116000"  "st'ud'ent"     #双引号包围的字符串
'''                                        #三引号包围的字符串
单引号和双引号包含的是单行字符串，
"两者的作用相同"。
三引号包括的是多行字符串.
'''
```

需要注意的是，3 个引号能包含多行字符串，这种字符串常常出现在函数声明的下一行，用来注释函数的功能。这个注释被认为是函数的一个默认属性，可以通过“函数名.__doc__”的形式进行访问。关于函数的内容将在第 5 章介绍。

2. 转义字符

转义字符用于表示一些在某些场合不能直接输入的特殊字符。例如下面的代码：

```
'type('abc') '
```

在由单引号包围的字符串中再次使用了单引号，代码运行时将报错。再如，代码中需要输入退格符、换行符、换页符等不可见字符，就要使用转义字符。转义符由反斜杠（\）引导，与后面相邻的字符组成了新的含义。典型的如，\n 表示换行，\\表示输入反斜杠，\t 表示制表符。常用的转义符如表 2-2 所示。

表 2-2　常用的转义符

转义符	含义描述	转义符	含义描述
\（在行尾时）	Python 的续行符	\n	换行
\\	反斜杠符号	\t	横向制表符
\'	单引号	\r	回车
\"	双引号	\f	换页
\a	响铃	\ooo	八进制数表示的 ASCII 码对应字符
\b	退格（Backspace）	\xhh	十六进制数表示的 ASCII 码对应字符
\0	空	\other	其他字符以普通格式输出

例 2-8　字符串的转义符。

```
>>> x='\000\101\102'
>>> y='\000\x61\x63'
>>> x,y
('\x00AB','\x00ac')
>>> print(x,y)                #运行结果字符前有空格
  AB  ac
>>> print("Python\n 语言\t 程序\tDesign")
Python
语言    程序    Design
```

2.4.2　字符串输出的格式化

程序运行输出的结果很多时候以字符串形式呈现，为了实现输出的灵活性和可编程性，需要控制字符串的输出格式，即字符串类型的格式化。Python 支持两种字符串格式化方法：一种是使用格式化操作符“%”，另一种是采用专门的 str.format()方法。Python 的后续版本中不再改进使用%操作符的格式化方法，将主要使用 format()方法实现字符串格式化。

1. 用%操作符格式化字符串

Python 的%操作符可用于格式化字符串，控制字符串的呈现格式。格式化字符串时，

Python 使用一个字符串作为模板。模板中有格式符，这些格式符为显示值预留位置，并说明显示数值应该呈现的格式。

Python 用一个元组（tuple）将多个值传递给模板，元组中的每个值对应一个格式符，元组是用圆括号包围并用逗号分隔的若干数值，将在第 6 章介绍。

使用%操作符格式化字符串的模板格式如下：

```
%[(name)][flags][width].[precision]typecode
```

参数的含义如下：

- (name)：可选参数，当需要格式化的值为字典类型时，用于选择指定字典的 key。
- flags：可选参数，可供选择的值如下：
  +，表示右对齐，在正数前添加正号，在负数前添加负号；
  -，表示左对齐，在正数前无符号，在负数前添加负号；
  空格，表示右对齐，在正数前添加空格，在负数前添加负号；
  0，表示右对齐，正数前无符号，在负数前添加负号，并用 0 填充空白处。
- width：可选参数，格式字符串的占用宽度。
- precision：可选参数，数值型数据保留的小数位数。
- typecode：必选参数，格式控制符，常用符号如表 2-3 所示。

表 2-3　字符串格式化控制符

符号	描述
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%f	格式化浮点数，可指定小数点后的精度
%e	用科学记数法格式化浮点数

例 2-9　用%操作符格式化字符串。

```
#显示十进制数，将浮点数转换为十进制数
>>> "%d  %d"%(12,12.3)
'12  12'
#设置十进制数显示宽度
>>> "%6d  %6d"%(12,12.3)
'    12     12'
#设置十进制数宽度和对齐方式
>>> "%-6d"%(12)
'12    '
#以浮点数方式显示
>>> "%f"%(100)
'100.000000'
#以浮点数方式显示，并设置宽度和小数位数
>>> "%6.2f"%(100)
```

```
'100.00'
#以科学记数法表示
>>> "%e"%(100)
'1.000000e+02'
#显示字符串和整数，分别设置宽度
>>> "%10s is %-3d years old"%("Rose",18)
'     Rose is 18  years old'
```

2. format()方法

从 Python 2.6 开始，增加了一种格式化字符串的 str.format()方法，该方法更便于对字符串进行格式化处理。

（1）模板字符串与 format()方法中参数的对应关系

str 被称为模板字符串，其中包括多个由“{}”表示的占位符，这些占位符接收 format()方法中的参数。str 模板字符串与 format()方法中的参数的对应关系有以下 3 种情况：

1）位置参数匹配。在模板字符串中，如果占位符{}为空（没有表示顺序的序号），则按照参数出现的先后次序匹配。如果在占位符{}中指定参数的序号，则按照序号对应参数替换。

2）使用键值对的关键字参数匹配。format()方法中的参数用键值对形式表示时，在模板字符串中用“键”来表示。

3）使用序列的索引作为参数匹配。如果 format()方法的参数是列表或元组，可以用其索引（序号）来匹配。

例 2-10　模板字符串与 format()方法中参数的关系。

```
#位置参数
>>> "{} is {} years old".format("Rose",18)
'Rose is 18 years old'
>>> "{0} is {1} years old".format("Rose",18)
'Rose is 18 years old'
>>> "Hi,{0}!{0} is {1} years old".format("Rose",18)
'Hi,Rose!Rose is 18 years old'

#关键字参数
>>>"{name} was born in {year},He is {age} years old".format(name="Rose",
   age=18,year=2000)
'Rose was born in 200,He is 18 years old'

#下标参数
>>> student=["Rose",18]
>>> school=("Dalian","LNNU")
>>> "{1[0]} was born in {0[0]},She is {1[1]} years old".format(school,
student)
'Rose was born in Dalian,She is 18 years old'
```

（2）模板字符串 str 的格式控制

下面详细说明模板字符串 str 的格式控制，其语法格式如下：

```
[[fill]align][sign][width][,][.precision][type]
```

参数的含义如下：

- fill：可选参数，空白处填充的字符。
- align：可选参数，用于控制对齐方式，配合 width 参数使用，align 参数的取值如下：
  <，内容左对齐；
  >，内容右对齐（默认）；
  ^，内容居中对齐。
- sign：可选参数，数字前的符号。
  +，在正数数值前添加正号，在负数数值前添加负号；
  −，正号不变，在负数数值前添加负号；
  空格，在正数数值前添加空格，在负数数值前添加负号。
- width：可选参数，格式化后的字符串所占宽度。
- 逗号（,）：可选参数，为数字添加千分位分隔符。
- precision：可选参数，指定小数位的精度。
- type：可选参数，格式化类型。
  整数常用的格式化类型包括以下几种：
  b，将十进制整数自动转换成二进制表示然后格式化；
  c，将十进制整数自动转换为其对应的 Unicode 字符；
  d，十进制整数；
  o，将十进制整数自动转换成八进制表示，然后格式化；
  x，将十进制整数自动转换成十六进制表示，然后格式化；
  X，将十进制整数自动转换成十六进制表示，然后格式化。
  浮点型常用的格式化类型包括以下几种：
  e，转换为科学记数法（小写 e）表示，然后格式化；
  E，转换为科学记数法（大写 E）表示，然后格式化；
  f，转换为浮点型（默认小数点后保留 6 位）表示，然后格式化；
  F，转换为浮点型（默认小数点后保留 6 位）表示，然后格式化；
  %，输出浮点数的百分比形式。

例 2-11　使用 str.format()方法格式化字符串。

```
>>> print('{:*>8}'.format('3.14'))              #宽度 8 位，右对齐
****3.14
>>> print('{:*<8}'.format('3.14'))              #宽度 8 位，左对齐
3.14****
>>> print('{0:^8},{0:*^8}'.format('3.14'))      #宽度 8 位，居中对齐
  3.14  ,**3.14**                                #科学记数法表示
>>> print('{0:e},{0:.2e}'.format(3.14159))
3.141590e+00,3.14e+00
```

2.4.3 字符串的操作符

字符串由若干字符组成，为实现字符串的连接、子串的选择、子串的包含判断等，Python 提供了系列字符串的操作符，如表 2-4 所示。其中，a、b 是两个字符串，a="Hello"，b="Python"。

表 2-4 字符串的操作符

操作符	描述	示例
+	连接字符串	a+b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[i]	切片操作。通过索引获取字符串中字符，i 是字符的索引	a[1]输出结果：e
[:]	切片操作。截取字符串中的一部分	a[1:4]输出结果：ell
in	如果字符串中包含给定的字符，则返回 True	'H' in a 输出结果：True
not in	如果字符串中不包含给定的字符，则返回 True	'M' not in a 输出结果：True
r/R	原始字符串，用来替代转义符表示的特殊字符，在原字符串的第一个引号前加上字母 r（R），与普通字符串操作相同	print(r'\n')等价于 print('\\n')，输出\: \n
b	返回二进制字符串，在原字符串的第一个引号前加上字母 b，可用于写二进制文件，如 b"123"	
%	格式化字符串操作符	

例 2-12 字符串操作符的应用。

```
>>> str1="Hi,Python!"
>>> str1*2              #str1 重复显示 2 次，str1 未发生改变
'Hi,Python!Hi,Python!'
>>> id(str1)            #测试 str1 在内存中的标识
54364264
>>> str1+="Hi,Java!"
>>> id(str1)            #str1 连接字符串后，id 发生改变
54338768
>>> str1
'Hi,Python!Hi,Java!'
#字符串切片操作
>>> str1[3:9]
'Python'
>>> str1[-5:-1]         #从后向前切片，最后一个字符索引是-1
'Java'
>>> str1[:-6]           #从索引为-6 的字符至字符串开头
'Hi,Python!Hi'
>>> "java" in str1
False
>>> "Java" in str1
True
```

2.4.4　内置的字符串处理函数

前面学习的 type()函数用于测试变量类型，id()函数用于测试变量的 id 值，format()函数用于格式化字符串，都是 Python 的内置函数。Python 提供了很多用于处理字符串的内置函数，部分函数如表 2-5 所示，之后通过具体示例加以说明。

表 2-5　Python 内置的常用字符串处理函数

函数名	功能描述
大小写转换函数	
lower()	将字符串中的大写字母转换为小写
upper()	将字符串中的小写字母转换为大写
capitalize()	将字符串的第一个字母转换为大写
swapcase()	英文字母大小写互换
查找替换函数	
find(str[,start[,end]])	检测 str 是否包含在字符串中，如果指定范围 start 和 end，则检查是否包含在指定范围内。如果包含则返回 str 的索引值，否则返回−1
index(str[,start[,end]])	同 find()方法。当 str 不在字母串中报告异常
rfind(str[,start[,end]])	类似于 find()函数，从右侧开始查找
rindex(str[,start[,end]])	类似于 index()函数，从右侧开始查找
replace(old,new [,count])	将字符串中的 old 替换成 new，如果指定 count，则替换不超过 count 次
字符判断函数	
isalnum()	如果字符串至少包含一个字符，并且所有字符都是字母或数字，则返回 True，否则返回 False
isalpha()	如果字符串至少包含一个字符，并且所有字符都是字母，则返回 True，否则返回 False
isdigit()	如果字符串只包含数字，则返回 True，否则返回 False
islower()	如果字符串中至少包含一个区分大小写的字母，并且所有这些（区分大小写的）字母都小写，则返回 True，否则返回 False
isnumeric()	如果字符串中只包含数字字符，则返回 True，否则返回 False
isspace()	如果字符串中只包含空白，则返回 True，否则返回 False
isupper()	如果字符串中至少包含一个区分大小写的字母，并且所有这些（区分大小写的）字母都大写，则返回 True，否则返回 False
isdecimal()	如果字符串只包含十进制字符，则返回 True，否则返回 False
字符串头尾判断函数	
startswith(str[,start[,end]])	检查字符串是否是以 str 开头，如果是返回 True，否则返回 False。如果指定 start 和 end 值，则在指定范围内检查
endswith(str[,start[,end]])	检查字符串是否以 str 结束，如果是返回 True，否则返回 False。如果指定 start 和 end 值，则在指定范围内检查
计算函数	
len(str)	返回字符串长度
max(str)	返回字符串中最大的字符
min(str)	返回字符串中最小的字符
count(str,[,start[,end]])	返回 str 在字符串中出现的次数，如果指定 start 或者 end 值，则返回指定范围内 str 出现的次数

续表

函数名	功能描述
对齐函数	
center(width,fillchar)	返回一个指定宽度 width 且居中的字符串，fillchar 为填充的字符，默认为空格
ljust(width[,fillchar])	返回一个左对齐的字符串，并使用 fillchar 填充至长度 width，fillchar 默认为空格
rjust(width,[,fillchar])	返回一个右对齐的字符串，并使用 fillchar 填充至长度 width，fillchar 默认为空格
字符串拆分合并函数	
split(sep,num)	以 sep 为分隔符分隔字符串，如果 num 有指定值，则仅截取 num 个子字符串
join(seq)	以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
删除字符串中空格函数	
lstrip()	删除字符串左边的空格
rstrip()	删除字符串末尾的空格
strip([chars])	在字符串上执行 lstrip()和 rstrip()函数

1. 大小写转换函数

例 2-13　大小写转换函数的应用。

```
>>> str1="hi,Python"
>>> str1.lower()
'hi,python'
>>> str1.upper()
'HI,PYTHON'
>>> str1.capitalize()
'Hi,python'
>>> str1.swapcase()
'HI,pYTHON'
```

2. 查找替换函数

例 2-14　查找替换函数的应用。

```
>>> str1="hi,Python!hi,Java!"
>>> str1.find("hi")
0
>>> str1.rfind("hi")
10
>>> str1.index("a")
14
>>> str1.rindex("a")
16
>>> str1.replace("hi","Hello")
'Hello,Python!Hello,Java!'
```

3. 字符判断函数

例 2-15　字符判断函数的应用。

```
>>> "aabbcc$123".isalnum()      #因为存在$，返回 False
False
>>> "hello9".isalpha()          #因为存在 9，返回 False
False
>>> "123".isdigit()
True
>>> "123".isnumeric()           #识别全角数字
True
>>> "12 二".isnumeric()         #识别汉字数字
True
>>> "12 二".isdigit()           #不识别汉字数字
False
>>> "ABc".isupper()
False
```

4. 字符串头尾判断函数

例 2-16　字符判断函数的应用。

```
>>> str1="hi,Python!hi,Java!"
>>> str1.startswith("hi")
True
>>> str1.endswith("Java!")
True
>>> str1.startswith("hi",3)     #从 str1 的第 3 个字符开始判断，不以“hi”开头
False
>>> str1.endswith("hi",3,12)    #从 str1 的第 3 至第 11 个字符判断，以“hi”结尾
True
```

5. 计算函数

例 2-17　计算函数的应用。

```
>>> str1="hi,Python!hi,Java!"
>>> len(str1)
18
>>> max(str1),min(str1)
('y','!')
>>> str1.count("hi")
2
```

6. 字符串拆分与合并

例 2-18　字符串拆分与合并函数的应用。

```
>>> str1="hi,Python,hi,Java!"
>>> str1.split()       #默认使用空格作分隔符，str1 中无空格，列表中只有一个元素
['hi,Python,hi,Java!']
>>> str1.split(",")    #使用逗号作分隔符，3 个逗号，分隔 3 次
['hi','Python','hi','Java!']
>>> str1.split(",",2) #使用逗号作分隔符，限制分隔 2 次
['hi','Python','hi,Java!']
>>> lst=['hi','Python!','hi','Java!']
>>> s=""
>>> s.join(lst)        #将列表连接为字符串
'hiPython!hiJava!'
```

2.5 Python 的运算符

运算符是用于表示不同运算类型的符号，可以将运算符分为算术运算符、比较运算符、逻辑运算符、赋值运算符等。Python 的变量由运算符连接构成表达式。

2.5.1 算术运算符

算术运算完成数学中的加、减、乘、除四则运算。算术运算符包括+（加）、-（减）、*（乘）、/（除）、%（求余）、**（求幂）、//（整除）。其中，幂运算返回 a 的 b 次幂，例如，12**3 计算的是 12 的 3 次方；整除运算返回商的整数部分，例如，24//10 的结果是 2。

例 2-19　算术运算符的应用。

```
>>> x1=17
>>> result1=x1+x2      #21
>>> result2=x1-x2      #4
>>> result3=x1*x2      #68
>>> result4=x1/x2      #4.25
>>> result5=x1%x2      #1
>>> result6=x1**x2     #835221
>>> result7=x1//x2     #4
```

由算术运算符将数值类型的变量连接起来构成算术表达式，它的计算结果是一个数值。不同类型的数据进行运算时，这些数据的类型应当是兼容的，并遵循运算符的优先级规则。关于数据类型转换请参考 2.6 节的内容。

2.5.2　比较运算符

比较运算是指两个数据之间的比较运算。比较运算符有 6 个，即 >（大于）、<（小于）、>=（大于等于）、<=（小于等于）、==（等于）和 !=（不等于）。

比较运算符多用于数值型数据的比较，有时也用于字符串数据的比较，比较结果返回 True 或 False。用比较运算符连接的表达式称为关系表达式，一般在程序分支结构中使用。

例 2-20　比较运算符的应用。

```
>>> x='student'
>>> y="teacher"
>>> x>y
False
>>> len(x)==len(y)
True
>>> x!=y
True
>>> x+y==y+x
False
```

2.5.3　逻辑运算符

逻辑运算符包括 and、or、not 等 3 个，分别表示逻辑与、逻辑或、逻辑非，运算结果是布尔值 True 或 False。其功能描述如表 2-6 所示，其中 x=12，y=0。

表 2-6　逻辑运算符

运算符	表达式	描述	示例
and	x and y	x、y 有一个为 False，逻辑表达式的值为 False	x and y，值为 False
or	x or y	x、y 有一个为 True，逻辑表达式的值为 True	x or y，值为 True
not	not x	x 值为 True，逻辑表达式的值为 False，x 值为 False，逻辑表达式的值为 True	not x，值为 False not y，值为 True

2.5.4　赋值运算符

赋值运算符用于计算表达式的值并赋给变量。在 Python 中，赋值运算有以下 3 种情况：为单一变量赋值，为多个变量赋一个值，为多个变量赋多个值。赋值运算是将赋值号右边的值传送给赋值号左边的变量，赋值表达式的运算方向是从右到左。例如，x=x+1 是一个合法的赋值运算，先计算 x+1 的值，再传送给赋值号左边的 x，这和数学中的等式是完全不同的含义。

例 2-21　赋值运算符的应用。

```
>>> x=5                 #为一个变量赋值，x 值为 5
>>> x=x+1               #赋值运算，x 值最后为 6
>>> x=y=z=5             #为多个变量赋值，x,y,z 值均为 5
>>> x,y,z=3,4,5         #为多个变量赋多个值，x 值为 3，y 值为 4，z 值为 5
```

赋值运算符可以和算术运算符组合成复合赋值运算符，如+=、-=、*=等。它是一种缩写形式，在修改变量时显得更为简单。表 2-7 列举了 Python 中的复合赋值运算符，其中 x=5，y=3。

表 2-7　复合赋值运算符

运算符	功能描述	示例
+=	加法赋值运算符	x+=y 相当于 x=x+y，x 计算后的结果为 8
-=	减法赋值运算符	x-=y 相当于 x=x-y，x 计算后的结果为 2
*=	乘法赋值运算符	x*=y 相当于 x=x*y，x 计算后的结果为 15
/=	除法赋值运算符	x/=y 相当于 x=x/y，x 计算后的结果为 1.6666667
%=	取余赋值运算符	x%=y 相当于 x=x%y，x 计算后的结果为 2
**=	幂赋值运算符	x**=y 相当于 x=x**y，x 计算后的结果为 125
//=	整除赋值运算符	x//=y 相当于 x=x//y，x 计算后的结果为 1

2.5.5　位运算符

位运算符用于对整数中的位进行测试、置位或移位处理，可以对数据进行按位操作。Python 的位运算符有 6 个，即～（按位取反）、&（按位与）、|（按位或）、^（按位异或）、>>（按位右移）、<<（按位左移）。位运算符的运算规则如表 2-8 所示。其中，op1、op2 指的是参与运算的整型变量。

表 2-8　位运算符的运算规则

运算符	用法	描述
～	～op1	按位取反
&	op1&op2	按位与
|	op1|op2	按位或
^	op1^op2	按位异或
>>	op1>> op2	右移 op2 位
<<	op1<< op2	左移 op2 位

例 2-22　位运算符的应用。

```
>>> op1=6
>>> op2=2
>>> ～op1              #等价于二进制～00000110=11111001，输出-7
```

```
-7
>>> op1|op2          #等价于二进制 0110|0010=0110，输出 6
6
>>> op1&op2          #等价于二进制 0110&0010=0010，输出 2
2
>>> op1>>op2         #0110 左移 2 位为 11000，输出 24
1
>>> op1<<op2         #0110 无符号右移 2 位为 0001，输出 1
24
```

需要说明的是，进行位运算后得到的二进制值是补码的形式，如果首位是 1，表示这是个负数，需要按照按位取反、末位加 1 的规则计算输出值。

2.6 运算符的优先级

表达式是变量和运算符按一定的语法形式组成的符号序列。表达式中的运算符是存在优先级的，优先级是指在同一表达式中多个运算符被执行的次序，在计算表达式值时，应按运算符的优先级别由高到低的次序执行。如果一个运算对象两侧的运算符优先级相同，则按规定的结合方向处理，称为运算符的结合性。在 Python 中，!（非）、+（正）、-（负）及赋值运算符的结合方向是“先右后左”，其余运算符的结合方向都是“先左后右”。

运算符的优先级如表 2-9 所示。在表达式中，可以使用括号()显式地标明运算次序，括号中的表达式首先被计算。

表 2-9　运算符的优先级

优先次序	运算符	优先次序	运算符
1	**（指数）	8	|
2	～（按位取反）　+（正数）　-（负数）	9	<　>　<=　>=
3	*　/　%　//	10	== !=
4	+　-	11	= += -= *= /= %= //=
5	>>（右移）　<<（左移）	12	not
6	&	13	and　or
7	^		

例 2-23　运算符优先级的应用。

```
>>> x=10
>>> y=20
>>> m=3.0
>>> n=8.2
>>> b=x+y>x-y*-1 and m<n%3
```

```
>>> b
False
>>> b1=((x+y)>(x-y*(-1))) and m<(n%3)
>>> b1
False
>>> b2=((x+y)>(x-y*(-1)))and(m<(n%3))
>>> b2
False
```

可以看出，b1、b2 表达式的可读性比 b 表达式的可读性明显增强。

小　结

本章内容包括程序的书写规范、标识符与关键字、数据类型与变量等内容，还介绍了数值型数据和字符型数据，介绍了 Python 的运算符和运算符的优先级等内容。

程序的书写，包括代码缩进、注释、语句续行、标识符与关键字等内容，是 Python 程序最基础的内容。

重点介绍了 Python 的数值类型数据和字符类型数据，尤其是字符串类型的运算符和内置函数。Python 的运算符包括算术运算符、比较运算符、逻辑运算符、赋值运算符等，这些运算符在表达式中存在优先级的问题。

Python 不要求在使用变量之前声明其数据类型，但数据类型决定了数据的存储和操作方式。熟练掌握各种数据类型的操作，可以提高编程效率。

习　题

1. 选择题

（1）下列选项中，不是 Python 关键字的是（　　）。

A. pass　　B. from　　C. yield　　D. static

（2）下列选项中，可以作为 Python 标识符的是（　　）。

A. getpath()　　B. throw　　C. my#var　　D. _My _price

（3）下列选项中，使用 bool()函数测试，值不是 False 的是（　　）。

A. 0　　B. []　　C. {}　　D. −1

（4）假设 x,y,z 的值都是 0，下列表达式中非法的是（　　）。

A. x=y=z=2　　B. x,y=y,x　　C. x=(y==z+1)　　D. x=(y=z+1)

（5）下列关于字符串的定义，错误的是（　　）。

A. '"hipython"'　　B. 'hipython'　　C. "hipython"　　D. [hipython]

2. 简答题

（1）简述 Python 标识符的命名规则。

（2）整数的二进制、八进制、十六进制都用什么格式描述？将十进制数转换为二进制、八进制、十六进制的函数是什么？

（3）Python 的数值类型数据有几种？举例说明。

3. 编程题

（1）编写程序，给出一个英文句子，统计单词个数。

（2）编写程序，给出三角形的三边长，输出三角形的面积。

（3）编写程序，给出一个字符串，将其中的字符“E”用空格替换后输出。

第 3 章
Python 程序的流程控制

程序是由若干语句组成的，目的是实现一定的计算或处理功能。程序中的语句可以是单一的一条语句，也可以是一个语句块（复合语句）。编写程序要解决特定的问题，这些问题通过多种形式输入，程序运行并处理，形成结果并输出，所以，输入、处理、输出是程序的基本结构。在程序内部，有逻辑判断与流程控制的问题。Python 的流程控制包括顺序、分支和循环 3 种结构。程序也支持函数调用。本章主要介绍 Python 程序的流程控制相关知识。

3.1 输入/输出语句

计算机程序都用来解决特定计算问题，每个程序都有统一的运算模式：输入数据、处理数据和输出数据。这种朴素运算的模式构成了基本的程序编写方法：IPO。

输入是一个程序的开始。程序要处理的数据有多种来源，例如，从控制台交互式输入数据、使用图形用户界面输入、从文件或网络读取数据输入。也可以由其他程序的运行结果得到输入数据等。输出是程序展示运算结果的方式。程序的输出方式包括控制台输出、图形输出、文件或网络输出等。

下面主要介绍使用控制台的输入/输出，其他输入/输出方式在相关章节中陆续学习。

3.1.1 输入语句

Python 的内置函数 input()用于取得用户输入数据，其语法格式如下：

```
varname=input("promptMessage")
```

其中，varname 是 input()函数返回的字符串数据；promptMessage 是提示信息，该参数可以省略。程序执行到 input()函数时，暂停执行，等待用户输入，用户输入的数据均作为输入内容。需要注意的是，如果要得到整数或小数，可以使用 eval()函数得到表达式的值，也可以用 int()或 float()函数转换。eval()函数将字符串对象转化为有效的表达式，再

参与求值运算返回计算结果。

例 3-1　使用 input()函数输入数据。

```
>>> name=input("请输入姓名：")
请输入姓名：Rose
#score1 为数值，需要参与数学计算，使用 eval()函数
>>> score1=eval(input("请输入科目 1 成绩："))
请输入科目 1 成绩：89
>>> score2=eval(input("请输入科目 2 成绩："))
请输入科目 2 成绩：60
>>> print("您的总成绩是：",(score1+score2))
您的总成绩是：149
```

3.1.2　输出语句

Python 3 使用 print()函数完成基本输出操作。print()函数的语法格式如下：

print([obj1,…][,sep=' '][,end='\n'][,file=sys.stdout])

print()函数所有参数均可省略，如果没有参数，print()函数输出一个空行。根据 print()函数给出的参数，在实际应用中分为以下几种情况：

- 包含多个参数时，参数之间默认用逗号分隔。
- 指定输出分隔符，可使用 sep 参数指定特定符号作为输出对象的分隔符号。
- 指定输出结尾符号，默认以回车换行符号作为输出结尾符号，可以用 end 参数指定输出结尾符号。
- 输出到文件，默认输出到标准输出设备（显示器），可使用 file 参数指定输出到特定文件。

例 3-2　print()函数的使用。

```
>>> x,y,z=100,200,300
>>> print(x,y,z)                    #print()函数中的多个参数用逗号分隔
100 200 300
>>> print(x,y,z,sep="##")           #设置 print()函数输出分隔符为##
100##200##300
>>> print(x);print(y);print(z)      #3 个 print()语句，默认分行显示
100
200
300
#print()设置 end 参数，用空格分隔，不换行
>>> print(x,end="  ");print(y,end="  ");print(z)
100  200  300
```

3.2 程序设计流程

计算机程序设计包括面向过程和面向对象两种方法。面向对象程序设计在细节实现上，也需要面向过程的内容。结构化程序设计是公认的面向过程的编程方法，按照自顶向下、逐步求精和模块化的原则进行程序的分析与设计。为提高程序设计的质量和效率、增强程序的可读性，可以使用程序流程图、PAD 图、N-S 图等作为辅助设计工具。

3.2.1 程序流程图

流程图是一种传统的、应用广泛的程序设计表示工具，也称为程序框图。程序流程图表达直观、清晰，易于学习掌握，独立于任何一种程序设计语言。

构成程序流程图的基本图例如图 3-1 所示。

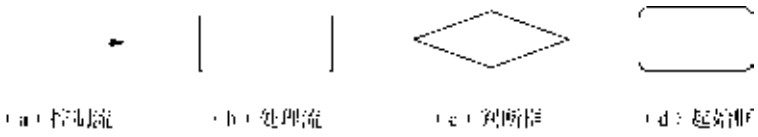

图 3-1 程序流程图基本元素

3.2.2 结构化程序设计基本流程

结构化程序设计大致包括 3 种基本流程：顺序结构、分支结构和循环结构。这 3 种结构的框架如图 3-2 所示。

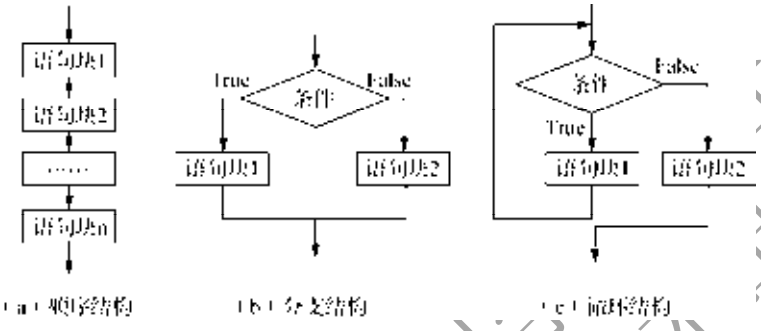

图 3-2 结构化程序设计的 3 种框架

顺序结构是 3 种结构中最简单的一种，即语句按照书写的顺序依次执行；分支结构又称为选择结构，它根据计算所得的表达式的值来判断应执行哪一个流程的分支；循环结构则是在一定条件下反复执行一段语句的流程结构。这 3 种结构构成了程序局部的基本框架。

不论是面向对象的计算机语言，还是面向过程的计算机语言，在局部的语句块内部，仍然需要使用流程控制语句来编写程序，完成相应的逻辑功能。Python 语言提供了实现分支结构的条件分支语句和实现循环结构的循环语句。

3.3 分 支 结 构

Python 使用 if 语句来实现分支结构。根据条件数量，如果是一个条件，形成简单分支结构；如果是两个条件，形成选择分支结构；如果是多个分支，形成多重分支结构。分支语句中还可以包含分支结构，形成分支的嵌套结构。

1. 简单分支结构：if 语句

if 语句的语法格式如下：

```
if <boolCondition>:
    <statements>
```

其中，boolCondition 是一个逻辑表达式，用来判断程序的流程走向。在程序的实际执行过程中，如果逻辑表达式的取值为 True，则执行 if 分支的语句块 statements，否则，绕过 if 分支直接执行 if 语句块后面的其他语句。

2. 选择分支结构：if-else 语句

Python 使用 if-else 语句实现选择分支结构，其语法格式如下：

```
if <boolCondition>:
    <statements1>
else:
    <statements2>
```

在程序执行过程中，如果 boolCondition 的取值为 True，则执行 if 分支的 statements1 语句块，否则执行 else 分支的 statements2 语句块。

例 3-3　分支语句示例。分段函数计算，根据 x 的值，输出 y 的值。

```
1  #program0303.py
2  import math
3  x=-37
4  if x<0:
5      y=math.fabs(x)
6  else:
7      y=math.sqrt(x)
8  print("计算的结果是：",y)
```

其中，import math 语句用于导入 math 模块，然后调用其中的 fabs()函数和 sqrt()函数。

3. 多分支结构：if-elif-else 语句

多分支 if 结构是选择分支结构的扩展，程序根据条件判断执行相应的分支，但只执

行第一个条件为 True 的语句块，即执行一个分支后，其他分支不再执行。如果所有条件均为 False，则执行 else 后面的语句块，else 分支是可选的。其语法格式如下：

```
if  <boolCondition1>:
    <statements1>
elif  <boolCondition2>:
    <statements2>
…
else:
    <statementsn>
```

例 3-4　多分支程序示例。根据月份计算该月的天数（未考虑闰年的情况）。

```
#program0304.py
month=eval(input("请输入您选择的月份："))
days=0;
if (month==1 or month==3 or month==5 or month==7 or month==8\
   or month==10 or month==12):
   days=31
elif (month==4 or month==6 or month==9 or month==11):
   days=30
else:
    days=28
print("{}月份的天数为{}".format(month,days))
```

4. 分支的嵌套

分支的嵌套是指分支中还存在分支的情况，即 if 语句中还包含着 if 语句。有问题描述如下：

有一个计算购书款的程序，如果有会员卡，购书 5 本以上，书款按 7.5 折结算，5 本以下按 8.5 折结算；如果没有会员卡，购书 5 本以上，书款按 8.5 折结算，5 本以下按 9.5 折结算。

例 3-5　利用分支的嵌套计算购书款。

```
#program0305.py
flag=1            #flag=1 表示有会员卡
books=8           #购书数量
price=234         #单价
actualpay=0
if flag==1:
   if books>=5:
      actualpay=price*0.75*books
   else:
      actualpay=price*0.85*books
else:
```

```
    if books>=5:
        actualpay=price*0.85*books
    else:
        actualpay=price*0.95*books
print("您的实际付款金额是：",actualpay)
```

本例中，读者可以尝试交互式输入不同的 flag、books、price 等变量值，然后运行程序，查看各分支的运行情况，完成程序的调试和运行。

3.4 循 环 结 构

循环结构是在一定条件下，反复执行某段程序的控制结构，反复执行的程序块称为循环体。循环结构是程序中非常重要的一种结构，它是由循环语句来实现的。Python 中的循环包括 for 循环和 while 循环两种。

3.4.1 遍历循环：for

for 循环是 Python 语言中使用较广泛的一种循环，它是一种遍历循环，主要用于遍历一个序列，如一个列表或一个字典。

1. for 循环结构

for 循环的流程结构见图 3-2（c）。其语法格式如下：

```
for <var> in <seq>:
    <statements>
```

其中，var 是一个变量，seq 是一个序列。for 循环的执行次数是由遍历结构中元素的个数决定的，可以理解为 for 循环从序列中逐一提取元素，放在循环变量中，对于序列中的每个元素执行一次语句块。序列可以是字符串、列表、文件或 range()函数等。经常使用的遍历方式如下：

（1）有限次遍历

```
for i in range(n):          #n 为遍历次数
    <statements>
```

（2）遍历文件

```
for line in myfile:         #myfile 为文件的引用
    <statements>
```

（3）遍历字符串

```
for ch in mystring:         #mystring 为字符串的引用
    <statements>
```

（4）遍历列表

```
for item in mylist:          #mylist 为列表的引用
    <statements>
```

2. range()函数

range()函数是 Python 的内置函数，用于创建一个整数列表，一般用在 for 循环中。range()函数的语法格式如下：

```
range(start,stop[,step])
```

函数的参数说明如下：

- start：计数从 start 开始，默认从 0 开始。例如，range(5)等价于 range(0,5)。
- end：计数到 end 结束，但不包括 end。例如，range(0,5)是[0,1,2,3,4]，不包括 5。
- step：步长，默认为 1。例如，range(0,5)等价于 range(0,5,1)。

例 3-6　range()函数的应用。

```
>>>range(10)                  #从 0 开始到 10
[0,1,2,3,4,5,6,7,8,9]
>>> range(1,11)               #从 1 开始到 11
[1,2,3,4,5,6,7,8,9,10]
>>> range(0,30,5)             #步长为 5
[0,5,10,15,20,25]
>>> range(0,10,3)             #步长为 3
[0,3,6,9]
>>> range(0,-10,-1)           #负数
[0,-1,-2,-3,-4,-5,-6,-7,-8,-9]
```

3. for 循环示例

例 3-7　for 循环示例。计算 1～100 能被 3 整除的数之和。

```
1  #program0307.py
2  s=0
3  for i in range(100):
4      if i%3==0:
5          s+=i
6          print(i)
7  print(s)
```

例 3-8　for 循环示例。计算 1!+2!+…+n!。

```
1  #program0308.py
2  '''计算 1!+2!+…+5!'''
3  def factorial(n):          #计算阶乘的函数
4      t=1
```

```
    for i in range(1,n+1):
        t=t * i
    return t
#计算阶乘和
k=6
sum1=0
for i in range(1,6):
    sum1+=factorial(i)
print("1!+2!+…+5!=",sum1)
```

3.4.2　条件循环：while

程序有时需要根据初始条件进行循环判断，当条件不满足时，循环结束。这种循环结构可以用 while 语句实现，其语法格式如下：

```
while <boolCondition>:
    <statements>
```

其中，boolCondition 为逻辑表达式，statements 语句块是循环体。

while 语句的执行过程是先判断逻辑表达式的值，若为 True，则执行循环体，循环体执行完毕后再转向逻辑表达式并做计算与判断；当计算出逻辑表达式的值为 False 时，跳过循环体执行循环体外的语句。

例 3-9　while 循环示例。将一个列表中的元素做头尾置换，即列表中第 1 个元素和倒数第 1 个元素交换，第 2 个元素和倒数第 2 个元素交换，依次进行，最后打印输出列表。

```
#program0309.py
lst=[1,3,7,-23,34,0,23,2,9,7,79]
head=0
tail=len(lst)-1
while head<len(lst)/2:
    lst[head],lst[tail]=lst[tail],lst[head]    #头尾互换
    head+=1                                    #头指针后移
    tail-=1                                    #尾指针前移
for item in lst:
    print(item,end="  ")
```

语句 lst[head],lst[tail]=lst[tail],lst[head]也可以用下列语句来替换：

```
temp=lst[head]
lst[head]=lst[tail]
lst[tail]=temp
```

由于这个程序的操作数据是一个列表，也可以用 for 循环来遍历。下面代码中使用了表达式 int(len(lst)/2)，是因为 range()函数的参数必须是整数，所以使用 int()函数进行了转换。

```
lst=[1,3,7,-23,34,0,23,2,9,7]
head=0
tail=len(lst)- 1
for head in range(0,int(len(lst)/2)):
   lst[head],lst[tail]=lst[tail],lst[head]
   head+=1
   tail-=1
for item in lst:
   print(item,end="  ")
```

3.4.3 循环的嵌套

无论是 for 循环还是 while 循环，其中都可以再包括循环，从而构成循环的嵌套。例 3-8 通过函数 factorial(n)计算阶乘，然后再计算阶乘之和，也可以使用二重循环来计算阶乘之和。

例 3-10　使用嵌套的 for 循环计算 1!+2!+…+n!。

```
1  #program0310.py
2  k=eval(input("请输入计算阶乘的数值:"))
3  sum1=0
4  for i in range(1,k):
5     t=1
6     for j in range(1,i+1):
7        t*=j
8     sum1+=t
9  print(sum1)
```

for 循环和 while 循环有时可以相互替代，下面用嵌套的 while 循环计算阶乘之和。

例 3-11　使用嵌套的 while 循环计算 1!+2!+…+n!。

```
1  #program0311.py
2  k=eval(input("请输入计算阶乘的数值："))
3  sum1=0
4  i=1
5  while i<=k:
6     t=j=1
7     while j<=i:
8        t*=j
9        j+=1
10    sum1+=t
11    i+=1
12 print(sum1)
```

3.5 流程控制的其他语句

3.5.1 跳转语句

跳转语句用来实现程序执行过程中流程的转移，主要包括 break 语句和 continue 语句。

1. break 语句

break 语句的作用是从循环体内部跳出，即结束循环。break 语句有时也称为断路语句，意为中断循环，不再执行循环体。

例 3-12　break 语句示例。求 99 的最大真约数。

```
#program0312.py
a=eval(input("请输入的数值："))
i=a//2                    #等价于 i=int(a/2)
while i>0:
    if a%i==0: break
    i-=1
print(a,"的最大真约数为：",i)
```

一个数的最大真约数不会大于这个数的 1/2，所以，从输入数据的 1/2 开始测试，如果能整除，则这个数就是最大真约数，程序中断；否则，减 1 后继续测试，直至程序执行结束。

2. continue 语句

continue 语句必须用于循环结构中，它的作用是终止本轮循环，跳过本轮剩余的语句，直接进入下一轮循环。continue 语句有时也称为短路语句，是指只对本次循环短路，并不终止整个循环。

例 3-13　continue 语句示例。求输入数值中正数之和，负数忽略。

```
#program0313.py
s=0
for i in range(6):
    x=eval(input("请输入数值数据："))
    if x<0:continue
    s+=x
print("正数之和是：",s)
```

3.5.2 pass 语句

pass 语句是空语句，主要是为了保持程序结构的完整性设计的。pass 语句一般用作

占位语句，该语句不影响其后面语句的执行。下面是使用 pass 语句的一个示例。

例 3-14　pass 语句示例。打印列表中的奇数。

```
#program0314.py
for i in [1,4,7,8,9]:
    if i%2==0:
        pass
        print("pass 语句处将来可以添加偶数处理的代码")
        continue
    print("奇数",i)
```

程序运行结果如下：

```
>>>
奇数 1
pass 语句处将来可以添加偶数处理的代码
奇数 7
pass 语句处将来可以添加偶数处理的代码
奇数 9
```

如果程序省略了 pass 语句，运行结果没有任何变化。但使用 pass 语句，可以作为将来添加偶数处理代码的占位符，提高了程序的可读性。

3.5.3　循环结构中的 else 语句

在其他各种计算机语言中，else 语句主要用在分支结构中；而在 Python 中，for 循环、while 循环、异常处理结构中也可以使用 else 语句。在循环中使用时，else 语句在循环正常结束后被执行，也就是说，如果有 break 语句，也会跳过 else 语句块。

例 3-15　循环结构中使用 else 语句。

```
str1="Hi,Python"
for ch in str1:
   print(ch,end="")
else:
   print("字符串遍历结束")
```

程序运行结果如下：

```
>>>
Hi,Python 字符串遍历结束
```

else 语句用在二重循环中，有时有更好的作用。下面代码的功能是计算 50 以内的质数，内层循环用于判断一个数是否为质数，如果循环正常结束，表明该数为质数，向列表中添加这个元素；否则在外层循环继续判断下一个数。

例 3-16　二重循环中的 else 语句。

```
#program0316.py
num=[];
i=2
for i in range(2,100):
   j=2
   for j in range(2,i):
      if(i%j==0):
         break
   else:
      num.append(i)
print(num)
```

程序运行结果如下：

```
>>>
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
```

3.6　流程控制语句的应用

1. 用蒙特卡罗方法计算圆周率

蒙特卡罗方法使用随机数和概率来解决问题，这个方法在计算数学、计算物理学和化学等方面有广泛的应用。

为了使用蒙特卡罗方法计算圆周率π，画一个圆的外接正方形，如图 3-3 所示。假设圆的半径是 1，那么圆的面积是π，外接正方形的面积是 4。任意产生正方形内的一个点，该点落在圆内的概率是圆面积/正方形面积，即π/4。

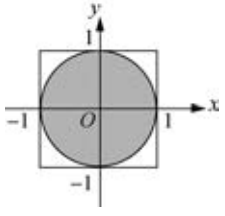

图 3-3　圆及外接正方形

编写程序，在正方形内随机产生 10000 个点，落在圆内点的数量用 n 表示。因此，n 的值约是 10000*π/4。可以估算π的值约为 4*n/10000。判断点(x,y)落在圆的公式是 *x**x+y*y*<=1。产生随机数使用 random 模块中的 random()函数。

例 3-17　用蒙特卡罗方法计算圆周率。

```
#program0317.py
import random
NUMBER=100000
n=0
for i in range(NUMBER):
    x=random.random()*2-1
    y=random.random()*2-1
```

```
 8      if((x*x+y*y)<=1):
 9          n+=1
10   pi=4.0*n/NUMBER
11   print("使用蒙特卡罗方法计算圆周率的值是：",pi)
```

程序运行结果如下：

```
>>>
使用蒙特卡罗方法计算圆周率的值是：3.14084
```

2. 使用循环控制输出格式

例 3-18　用“*”输出金字塔形状。

在程序的第 i 行，每行打印(2i−1)个“*”，在之前输出 n−i 个空格，n 是用户指定打印的行数。

```
n=eval(input("请输入打印的行数："))
for i in range(1,n):
   print(' '*(n-i)+'*'*(2*i-1))
```

运行结果如图 3-4 所示。

图 3-4　例 3-18 程序运行结果

例 3-19　输出数字金字塔。

程序的外循环控制输出行数。内循环分为两个，第一个 while 循环从变量 x 开始递减，在同行内输出，直至输出 1 为止；第二个 while 循环从初始值 2 开始递增，在行内输出，直到输出变量 x。两部分都输出后，执行 print()换行，输出下一行内容。

```
1   #program0319.py
2   n=eval(input("请输入打印的行数："))
3   for x in range(1,n+1):
4      print(' '*(10-x),end="")
5      n=x
6      while n>=1:
7         print(n,sep="",end="")
8         n-=1
```

```
 9     n=2
10     while n<=x:
11         print(n,sep="",end="")
12         n+=1
13     print()
```

运行结果如图 3-5 所示。

图 3-5　例 3-19 程序运行结果

小　结

本章内容包括 Python 中的输入/输出语句和流程控制语句。

input()函数用于输入数据，print()函数用于输出数据，注意 print()函数参数的使用。

结构化程序设计使用程序流程图、PAD 图、N-S 图等作为设计工具，结构化程序设计包括顺序结构、分支结构和循环结构 3 种流程。

Python 使用 if 语句来实现分支结构，使用 for 语句和 while 语句实现循环结构。分支结构和循环结构都可以嵌套。

跳转语句包括 break 语句和 continue 语句。break 语句的作用是从循环体内部跳出，continue 语句必须用于循环结构中，它的作用是跳过当前循环，进入下一轮循环。pass 语句的含义是空语句，主要是为了保持程序结构的完整性设计的。

本章内容是编写程序的基础，读者需要通过不断地书写和阅读程序来加强训练。

习　题

1. 选择题

（1）下列选项中，不属于 Python 循环结构的是（　　）。

A．for 循环　　B．while 循环

C．do while 循环　　D．嵌套的 while 循环

（2）以下代码段的运行结果是（　　）。

```
a=17
b=6;
result=a%b if(a%b>4) else a/b
print(result)
```

A．0　　B．1　　C．2　　D．5

（3）以下代码段的运行结果是（　　）。

```
i=3;
j=0;
k=3.2;
if(i<k):
    if(i==j):
        print(i)
    else:
        print(j)
else:
    print(k)
```

A．3　　B．0　　C．3.2　　D．以上都不对

（4）关于下面代码的陈述，正确的是（　　）。

```
x=0
while x<10:
    x+=1
    print(x)
    if x>3:
        break
```

A．代码编译异常　　B．输出：0 1 2

C．输出：1 2 3　　D．输出：1 2 3 4

（5）以下代码段的运行结果是（　　）。

```
for i in range(4):
    if i==3:
        break
    print(i)
print(i)
```

A．0123　　B．0122　　C．123　　D．234

2. 简答题

（1）叙述 pass 语句的作用。

（2）跳转语句 break 和 continue 的区别是什么？

（3）简述 for 循环和 while 循环的执行过程。

3. 编程题

（1）给定某一字符串 s，对其中的每一个字符 c 进行大小写转换：如果 c 是大写字母，则将它转换成小写字母；如果 c 是小写字母，则将它转换成大写字母；如果 c 不是字母，则不转换。

（2）输入一个整数，将各位数字反转后输出。

（3）计算 $1^2-2^2+3^2-4^2+\cdots+97^2-98^2+99^2$。

（4）一个数如果恰好等于它的因子之和，则称此数为“完数”。例如，6 的因子为 1,2,3，而 6=1+2+3，因此 6 就是“完数”。编程找出 100 内的所有完数。

（5）输入两个正整数 m 和 n，求其最大公约数和最小公倍数。

第 4 章
Python 的组合数据类型

除整型、浮点型等基本数据类型外，Python 还提供了列表、元组、字典、集合等组合数据类型。组合数据类型能将不同类型的数据组织在一起，实现更复杂的数据表示或数据处理功能。

根据数据之间的关系，组合数据类型可以分为 3 类：序列类型、映射类型和集合类型。序列类型包括列表、元组和字符串 3 种。映射类型用键值对表示数据，典型的映射类型是字典。集合类型数据中的元素是无序的，且不允许有相同的元素存在。

4.1 序 列 类 型

序列类型的元素之间存在先后关系，可以通过索引来访问。当需要访问序列中的某个元素时，只要找出其索引即可。由于元素之间存在顺序的位置关系，序列中不同的位置可存在数值相同的元素。

序列类型支持成员关系操作符（in）、分片运算符（[]），序列中的元素还可以是序列类型。

Python 中典型的序列类型包括字符串、列表（list）和元组。字符串可以看作单一字符的有序组合，属于序列类型。同时，由于字符串类型十分常用且单一字符串只表达一个含义，也被看作基本数据类型。无论哪种具体数据类型，只要它是序列类型，都可以使用相同的索引体系，即正向递增序号和反向递减序号，通过索引非常容易地查找序列中的元素。序列类型元素的索引如图 4-1 所示。

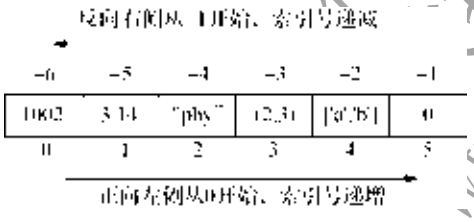

图 4-1　序列的正向索引和反向索引

序列的常用操作符和方法如表 4-1 所示。其中，s 和 t 是序列，x 是引用序列元素的变量，i,j,k 是序列的索引，这些操作符和方法是学习列表和元组的基础。

表 4-1　序列类型的常用操作符

操作符或方法	功能描述
x in s	如果 x 是 s 的元素，则返回 True，否则返回 False
x not in s	如果 x 不是 s 的元素，则返回 True，否则返回 False
s+t	返回 s 和 t 的连接
s*n	将序列 s 复制 n 次
s[i]	索引，返回序列的第 i 个元素
s[i:j]	分片，返回包含序列 s 第 i～j 个元素的子序列（不包含第 j 个元素）
s[i:j:k]	返回包含序列 s 第 i～j 个元素以 k 为步长的子序列
len(s)	返回序列 s 的元素个数（长度）
min(s)	返回序列 s 中的最小元素
max(s)	返回序列 s 中的最大元素
s.index(x[,i[,j]])	序列 s 中从 i 开始到 j 位置中第一次出现元素 x 的位置
s.count(x)	序列 s 中出现 x 的总次数

4.2 列　　表

列表是 Python 中最常用的序列类型，列表中的元素（也称数据项）不需要具有相同的类型。创建列表时，只要将以逗号分隔的元素使用方括号括起来即可。列表是可变的，可向列表中任意增加或删除元素，可以对列表进行遍历、排序、反转等操作。

4.2.1　列表的基本操作

列表作为一种序列类型，使用序列的常用操作符可以完成列表的切片、检索、计数等基本操作。

例 4-1　列表的基本操作。

```
#创建列表
>>> lst1=[]                                    #创建空列表
>>> lst2=["python",12,2.71828,[0,0],12]        #创建由不同类型元素组成的列表
>>> lst3=[21,10,55,100,2]
#访问列表元素
>>> "python" in lst2
True
>>> lst2[3]                                    #通过索引访问列表中的元素
[0,0]
>>> lst2[1:4]                                  #通过切片访问列表中的元素
[12,2.71828,[0,0]]
>>> lst2[-4:-1]                                #通过切片访问列表中的元素
```

```
[12,2.71828,[0,0]]
>>> len(lst2)                          #计算列表的长度
5
>>> lst2.index(12)                     #检索列表中的元素
1
>>> lst2.count(12)                     #计算列表中出现元素的次数
2
>>> max(lst3)                          #计算列表中的最大值
100
```

4.2.2 列表的方法

除了使用序列操作符操作列表，列表还有特有的方法，如表 4-2 所示，主要功能是完成列表元素的增、删、改、查，其中 ls、lst 为两个列表，x 是列表中的元素，i 和 j 是列表的索引。

表 4-2 序列类型的常用操作符和方法

操作符	功能描述
ls[i]=x	将列表 ls 的第 i 项元素替换为 x
ls[i:j]=lst	用列表 lst 替换列表 ls 中第 i～j 项元素（不含第 j 项）
ls[i:j:k]=lst	用列表 lst 替换列表 ls 中第 i～j 项以 k 为步长的元素（不含第 j 项）
del ls[i:j]	删除列表 ls 第 i～j 项元素
del ls[i:j:k]	删除列表 ls 第 i～j 项以 k 为步长的元素
ls+=lst 或 ls.extend(lst)	将列表 lst 元素追加到列表 ls 中
ls*=n	更新列表 ls，其元素重复 n 次
ls.append(x)	在列表 ls 最后增加一个元素 x
ls.clear()	清除列表 ls
ls.copy()	复制生成一个包括 ls 中所有元素的新列表
ls.insert(i,x)	在列表 ls 的第 i 个位置增加元素 x
ls.pop(i)	返回列表 ls 中的第 i 项元素并删除该元素
ls.remove(x)	删除列表中出现的第一个 x 元素
ls.reverse(x)	反转列表 ls 中的元素
ls.sort()	排序列表 ls 中的元素

例 4-2 列表的常用方法。

```
#初始化 3 个列表
>>> lst2=["python",12,2.71828,[0,0],12]
>>> lst3=[21,10,55,100,2]
>>> lst=['aaa','bbb']
#替换列表元素
>>> lst2[2]=3.14
>>> lst2
```

```
['python',12,3.14,[0,0],12]
>>> lst2[0:3]=lst
>>> lst2
['aaa','bbb',[0,0],12]
#追加（合并）列表
>>> lst2+=lst3
>>> lst2
['aaa','bbb',[0,0],12,21,10,55,100,2]
>>> del lst2[:3]            #删除 0，1，2 等 3 个列表元素
>>> lst2
[12,21,10,55,100,2]
>>> lst2.append(99)         #追加列表元素
>>> lst2
[12,21,10,55,100,2,99]
>>> lst4=lst2.copy()        #复制列表
>>> lst4
[12,21,10,55,100,2,99]
>>> lst4.clear()            #清除列表
>>> lst4
[]
>>> lst2.pop(6)             #删除列表指定位置上的元素，并返回删除元素值
99
>>> lst2
[12,21,10,55,100,2]
>>> id(lst2)
55026272
>>> lst2.reverse()          #翻转列表
>>> lst2
[2,100,55,10,21,12]
>>> id(lst2)
55026272
>>> lst2.sort()             #排序列表
>>> lst2
[2,10,12,21,55,100]
>>> id(lst2)
55026272
```

4.2.3　遍历列表

遍历列表可以逐个处理列表中的元素，通常使用 for 循环和 while 循环来实现。例 4-3 使用 for 循环遍历了列表中的所有元素，显示时以逗号分隔。

例 4-3　用 for 循环遍历列表。

```
#program0403.py
lst=['primary school','secondary school','high school','college']
```

```
for item in lst:
    print(item,end=",")
```

用 while 循环遍历列表时，需要先获取列表的长度，将获得的长度作为循环条件。例 4-4 先构造一个初始值为 2、步长为 3、终值为 21 的列表，即[2,5,8,11,14,17,20]，然后在 while 循环中遍历，将计算得到的新值添加到空列表 result 中。

例 4-4 用 while 循环遍历列表。

```
#program0404.py
lst=list(range(2,21,3))
i=0
result=[]
while i<len(lst):
    result.append(lst[i]*lst[i])
    i+=1
print(result)
```

4.3 元　　组

元组是包含 0 个或多个元素的不可变序列类型。元组生成后是固定的，其中的任何元素都不能替换或删除。元组与列表的区别在于元组的元素不能修改。创建元组时，只要将元组的元素用圆括号包围，并使用逗号隔开即可。

4.3.1 元组的基本操作

使用表 4-1 中的序列类型的常用操作符，可以完成元组的基本操作。

例 4-5 元组的基本操作。

```
#创建元组
tup1=('physics','chemistry',1997,2000)    #元组中包括不同类型数据
tup2=(1,2,3,4,5)
tup3="a","b","c","d"                      #声明元组的括号可以省略
tup4=(50,)                                #元组只有一个元素时，逗号不可省略
tup5=((1,2,3),(4,5),(6,7),9)
>>> type(tup3),type(tup4)                 #变量类型测试
(<class 'tuple'>,<class 'tuple'>)
>>> 1997 in tup1
True
>>> tup2+tup3                             #元组连接
(1,2,3,4,5,'a','b','c','d')
>>> tup1[1]                               #使用索引访问元组中元素
'chemistry'
>>> len(tup1)
```

```
4
>>> max(tup3)
'd'
>>> tup1.index(2000)                          #检索元组中元素位置
3
>>> help(tuple)                               #显示元组的属性和方法
>>> tup3.index(2000)                          #检索的元素不存在，运行报告异常
Traceback (most recent call last):
  File "<pyshell#130>",line 1,in <module>
    tup3.index(2000)
ValueError: tuple.index(x): x not in tuple
```

4.3.2　元组与列表的转换

元组与列表是非常类似的，但元组的元素值不能被修改，如果想要修改其元素值，可以将元组转换为列表，修改后，再转换为元组。列表和元组相互转换的函数是：将列表转换为元组的 tuple(lst)和将元组转换为列表的 list(tup)，其中的参数是被转换对象。

例 4-6　元组与列表相互转换。

```
>>> tup1=(123,'xyz','zara','abc')
>>> lst1=list(tup1)
>>> lst1.append(999)
>>> tup1=tuple(lst1)
>>> tup1
(123,'xyz','zara','abc',999)
```

4.4　字　　典

字典是 Python 中内置的映射类型。映射是通过键值查找一组数据值信息的过程，由 key-value 键值对组成，通过 key 可以找到其映射的值 value。

字典可以看作由元素对构成的列表，其中一个元素是键，另一个元素是值。在搜索字典时，首先查找键，当查找到键后就可以直接获取该键对应的值，这是一种高效实用的查找方法。这种数据结构之所以被命名为字典，是因为它的存储和检索过程与真正的字典类似。键类似于字典中的单词，根据字典的组织方式（如按字母顺序排列）找到单词（键）非常容易，找到键就能找到相关的值（定义）；但反向的搜索，即使用值来搜索键难以实现。

字典中的值并没有特定的顺序，但是都存储在一个特定的键（key）里，键可以是数字、字符串及元组等。此外，字典中的元素（键值对）是无序的。当添加键值对时，Python 会自动修改字典的排列顺序，提高搜索效率，而且这种排列方式对用户是隐藏的。

4.4.1 字典的基本操作

字典的基本操作包括创建字典、添加字典元素、查找字典、修改字典等。

1. 创建字典

字典是由标记“{}”定义，字典中每个元素包含键和值两部分，键和值用冒号分开，元素之间用逗号分隔。下面给出创建字典的代码，dict()是用于创建字典的函数。

例 4-7 创建字典。

```
>>> dict1={}
>>> dict2={"id":101,"name":"Rose","address":"Changjiangroad","pcode":"116022"}
>>> dict3=dict(id=101,name="Rose",address="changjiangroad",pcode="116022")
>>> dict4=dict([('id',101),('name','Rose'),('address','changjiangroad'),
                ('pcode','116022')])
>>> dict2     #显示字典内容
{'id':101,'name':'Rose','address':'Changjiangroad','pcode':'116022'}
```

这段代码中：

第 1 行用于创建一个空的字典，该字典不包含任何元素，可以向字典中添加元素。

第 2 行是典型的创建字典的方法，是用“{}”括起来的键值对。

第 3 行使用 dict()函数，使用关键字参数创建字典。

第 4 行使用 dict()函数，通过键值对序列创建字典。

2. 字典检索

可以使用 in 运算符测试一个指定的键值是否存在于字典中，表达式的格式是 key in dicts，其中 dicts 是字典名，key 是键名。如果需要通过键来查找值，可以使用表达式 dicts['key']，将返回 key 所对应的值。

例 4-8 字典检索。

```
#使用 in 运算符检索
>>> dicts={"id":101,"name":"Rose","address":"Changjiangroad","pcode":"116022"}
>>> "id" in dicts
True
>>> "address" in dicts
True
>>> "Rose" in dicts
False
#使用关键字检索
>>> dicts["id"]
101
>>> dicts["pcode"]
'116022'
>>> t1=dicts["id"],dicts["pcode"]
```

```
>>> t1,type(t1)
((101,'116022'),<class 'tuple'>)
```

3. 添加与修改字典元素

字典的大小是动态的，不需要事先指定其容量大小，可以随时向字典中添加新的键值对，或者修改键所关联的值。添加字典元素和修改字典元素的方法相同，都是使用 dicts[key]=value 的形式，如果字典中存在该键值对，则修改字典元素的值，否则实现的是字典元素的添加功能。

例 4-9　字典元素修改与添加。

```
>>> dict1={"id":101,"name":"Rose","address":"Changjiangroad"}
#修改元素值
>>> dict1["address"]="Huangheroad"
>>> dict1
{'id':101,'name':'Rose','address':'Huangheroad'}
#添加加字典元素
>>> dict1["email"]="python@learning.com"
>>> dict1
{'id':101,'name':'Rose','address':'Huangheroad','email':'python@learning.
com'}
```

在这段代码中，字典 dict1 已经存在"address"键值对，所以语句 dict1["address"]= "Huangheroad"仅仅是修改元素值；字典 dict1 没有"email"键值对，所以语句 dict1["email"]= "python@learning.com"是向字典中添加了一个元素（键值对）。

4.4.2　字典的常用方法

Python 内置了一些字典的操作方法，如表 4-3 所示。其中，dicts 为字典名，key 为键，value 为值。

表 4-3　字典类型的常用方法或操作

方法或操作	功能描述
dicts.keys()	返回所有的键信息
dicts.values()	返回所有的值信息
dicts.items()	返回所有的键值对
dicts.get(key,default)	键存在则返回相应值，否则返回默认值
dicts. pop(key,default)	键存在则返回相应值，同时删除键值对，否则返回默认值
dicts. popitem()	随机从字典中取出一个键值对，以元组(key,value)形式返回
dicts.clear()	删除所有的键值对
del dicts[key]	删除字典中某一个键值对
key in dicts	如果键在字典中则返回 True，否则返回 False
dicts.copy()	复制字典
dicts.update(dicts2)	更新字典，参数 dicts2 为更新的字典

下面通过示例介绍字典的常用方法。

1. keys()、values()和 items()方法

这 3 个方法分别返回字典的键的视图、值的视图和键值对的视图。视图对象与列表不同，它不支持索引，但可以迭代访问，通过遍历视图可以获得字典的信息。

例 4-10　字典操作的 keys()、values()和 items()方法。

```
>>> dicts={"id":101,"name":"Rose","address":"Changjiangroad","pcode":"116022"}
#获得键的视图
>>> key1=dicts.keys():
>>> type(key1)
<class 'dict_keys'>
>>> key1=dicts.keys()
>>> for k in key1:
      print(k,end=",")
id,name,address,pcode,
#获得值的视图
>>> values1=dicts.values()
>>> type(values1)
<class 'dict_values'>
>>> for v in values1:
      print(v,end=",")
101,Rose,Changjiangroad,116022,
#获得键值对的视图
>>> items=dicts.items()
>>> type(items)
<class 'dict_items'>
>>> for item in items:
      print(item,end=",")
('id',101),('name','Rose'),('address','Changjiangroad'),('pcode','116022'),
```

2. get()、pop()和 popitem()方法

get()方法返回键对应的值。如果 key 不存在，则返回空值。default 参数可以指定键不存在时的返回值。

pop()方法从字典中删除键，并返回对应的值。如果 key 不存在，则返回 default 值，如果未指定 default 参数，代码运行时会产生异常。

popitem()方法从字典删除并返回键值对。字典为空会产生 keyerror 异常。

例 4-11　字典操作的 get()、pop()与 popitem()方法。

```
>>> dicts={"id":101,"name":"Rose","address":"Changjiangroad"}
#get()方法
>>> dicts.get("address")
'Changjiangroad'
```

```
>>> dicts.get("pcode")
>>> dicts.get("pcode","116000")    #字典中不存在 pcode，返回默认值
'116000'
>>> dicts
{'id':101,'name':'Rose','address':'Changjiangroad'}
#pop()方法
>>> dicts.pop('name')
'Rose'
>>> dicts
{'id':101,'address':'Changjiangroad'}
>>> dicts.pop("email","u1@u2")     #字典中不存在 email，返回默认值
'u1@u2'
>>> dicts
{'id':101,'address':'Changjiangroad'}
>>> dicts={"id":101,"name":"Rose","address":"Changjiangroad"}
#popitem()方法，逐一删除键值对
>>> dicts.popitem()
('address','Changjiangroad')
>>> dicts.popitem()
('name','Rose')
>>> dicts.popitem()
('id',101)
>>> dicts
{}
```

3. copy()和 update()方法

copy()方法返回一个字典的副本，但新产生的字典与原字典的 id 是不同的，当修改一个字典对象时，对另一个字典对象没有影响。

update()方法可以使用一个字典更新另一个字典，如果两个字典存在相同的键，键值对会进行覆盖。

例 4-12　字典操作的 copy()与 update()方法。

```
>>> dict1={"id":101,"name":"Rose","address":"Changjiangroad"}
#copy()方法
>>> dict2=dict1.copy()
>>> id(dict1),id(dict2)
(62627152,68030112)
>>> dict1 is dict2
False
>>> dict2["id"]=102
>>> dict2
{'id':102,'name':'Rose','address':'Changjiangroad'}
>>> dict1
{'id':101,'name':'Rose','address':'Changjiangroad'}
```

```
#update()方法
>>> dict3={"name":"John","email":"u1@u2"}
>>> dict1.update(dict3)
>>> dict1
{'id':101,'name':'John','address':'Changjiangroad','email':'u1@u2'}
```

4.5 集　　合

集合（set）是 0 个或多个元素的无序组合，但集合本身是可变的，可以很容易地向集合中添加元素或移除集合中的元素。集合中的元素只能整数、浮点数、字符串等基本数据类型，而且这些元素是无序的，没有索引位置的概念。

集合中的任何元素都没有重复，这是集合的一个非常重要的特点。集合与字典有一定程度的相似性，但集合只是一组 key 的集合，这些 key 不可以重复，集合中没有 value。

4.5.1 集合的常用操作

1. 创建集合

可以使用 set()函数来创建一个集合。其与列表、元组、字典等数据结构不同，创建集合没有快捷方式，必须使用 set()函数。set()函数最多有一个参数。如果没有参数，会创建空集合；如果有一个参数，那么参数必须是可迭代的类型，如字符串或列表，可迭代对象的元素将生成集合的成员。

例 4-13　创建集合。

```
>>> aset=set("python")          #字符串作为参数创建集合
>>> bset=set([1,2,3,5,2])       #列表作为参数创建集合
>>> cset=set()                  #创建空集合
>>> aset,bset,cset
({'o','p','t','y','h','n'},{1,2,3,5},set())
```

从运行结果可以看出，集合的初始顺序和显示顺序是不一致的，表明集合中的元素是无序的。

2. 集合的操作

Python 提供了众多的操作集合的方法，用于向集合中添加元素、删除元素或复制集合等，常用的方法如表 4-4 所示。其中，S、T 为集合，x 为集合中的元素。

表 4-4　集合类型的方法

方法	功能描述
S.add(x)	添加元素。如果元素 x 不在集合 S 中，将 x 添加到 S
S.clear()	清除元素。移除 S 中的所有元素
S.copy()	复制集合。返回集合 S 的一个副本