普通高等教育系列教材

数据结构与算法
(Python 版)

周元哲　编著

机械工业出版社

机 械 工 业 出 版 社

本书讲述了 Python 语言与数据结构。主要内容包括数据结构与算法、Python 开发环境、Python 数据类型、Python 三大结构、函数、线性表、树和二叉树、图、查找、排序、异常处理与调试等。

本书内容精炼、由浅入深，注重学习的连续性和渐进性，适合作为高等院校相关专业教材或教学参考书，也可作为计算机技术人员的应用参考书，还可作为全国计算机等级考试、软件技术资格与水平考试的培训资料。

本书配有电子课件和随书源代码，需要的教师可登录 www.cmpedu.com 免费注册，审核通过后下载，或联系编辑索取（教师服务微信：15910938545；电话：010-88379739）。

图书在版编目(CIP)数据

数据结构与算法：Python 版/周元哲编著．—北京：机械工业出版社，2020.9
普通高等教育系列教材
ISBN 978-7-111-66363-8

Ⅰ.①数… Ⅱ.①周… Ⅲ.①数据结构-高等学校-教材 ②算法分析-高等学校-教材 Ⅳ.①TP311.12

中国版本图书馆 CIP 数据核字(2020)第 158692 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)
策划编辑：郝建伟 责任编辑：郝建伟 李培培 车 忱
责任校对：张艳霞 责任印制：常天培

北京虎彩文化传播有限公司印刷

2020 年 9 月第 1 版·第 1 次印刷
184mm×260mm·17.25 印张·426 千字
0001-1500 册
标准书号：ISBN 978-7-111-66363-8
定价：59.00 元

电话服务
客服电话：010-88361066
010-88379833
010-68326294

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
金 书 网：www.golden-book.com
机工教育服务网：www.cmpedu.com

封底无防伪标均为盗版

机械工业出版社

前　言

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言，带有各种库，在大数据、数据分析、科学计算等方面功能卓越。本书讲述了 Python 与数据结构，主要内容包括数据结构与算法、Python 开发环境、Python 数据类型、Python 三大结构、函数、线性表、树和二叉树、图、查找、排序、异常处理与调试等。学习本书内容后，建议完成数据结构课程设计。附录给出了软件考试与软件竞赛、图论相关模块、更多数据类型和习题答案。

本书具有如下特点：①代码详解。传统的数据结构教材“重理论轻代码”，往往只是给出伪代码，而本书的代码都用 Python 实现。②图文并茂。本书利用 Python 语言的特性，如使用 Python 的 deque 讲解栈、networkX 讲解图论，使得数据结构算法可视化，从而便于学生更快地掌握数据结构的思想，提高学生的编程应用开发能力。③突出实用性。本书每章都有用 Python 实现该章内容的案例。

西安邮电大学郝羽、李晓戈、孟伟君、高巍然和孔韦韦等阅读了部分手稿。作为西安邮电大学 ACM 教练，本书与众多同行交流，ACM 亚洲区第一训练委员会主任吴永辉、桂林电子科技大学王子民、华东交通大学周娟、北京化工大学刘勇、中国石油大学（华东）张学辉、太原理工大学林福平、中南民族大学刘卫平，以及机械工业出版社郝建伟等对本教材的写作大纲、写作风格等提出了很多宝贵意见，西安邮电大学 ACM 集训队杨晨磊、张天泰、黄文丰、黄昊、江永文等调试了部分代码。衷心感谢各位的支持和帮助。

本书在写作过程中参阅了大量中英文的专著、教材、论文、报告及网上的原创文章，由于篇幅所限，未能一一列出，在此，一并表示敬意和衷心的感谢。

本书内容精炼、文字简洁、结构合理、实训题目经典实用、综合性强，特别适合作为高等院校相关专业教材或教学参考书，也可供计算机技术人员参考。本书采用 Python 3 版本，所有程序都在 Anaconda 中进行调试和运行。由于作者水平有限，时间紧迫，本书难免存在疏漏之处，恳请广大读者批评指正。

本书编者的电子信箱是 zhouyuanzhe@ 163. com。

编　者

目　录

前言

第 1 章　数据结构与算法 …… *1*

1.1　程序 …… *1*

1.2　数据结构 …… *2*

1.2.1　数据结构的核心地位 …… *2*

1.2.2　数据结构的组成 …… *2*

1.3　算法 …… *3*

1.3.1　算法的 5 个属性 …… *4*

1.3.2　算法的 3 个层次 …… *4*

1.4　算法复杂度 …… *5*

1.4.1　空间复杂度 …… *5*

1.4.2　时间复杂度 …… *6*

1.4.3　提高算法效率的方法 …… *6*

1.5　算法表示方式 …… *7*

1.5.1　流程图 …… *8*

1.5.2　N-S 图 …… *8*

1.5.3　伪语言 …… *9*

1.6　习题 …… *9*

第 2 章　Python 开发环境 …… *10*

2.1　Python 简介 …… *10*

2.1.1　Python 的特点 …… *10*

2.1.2　Python 的应用场合 …… *11*

2.2　Python 解释器 …… *12*

2.2.1　Ubuntu 下安装 Python …… *12*

2.2.2　Windows 下安装 Python …… *13*

2.3　Python 编辑器 …… *14*

2.3.1　IDLE …… *14*

2.3.2　PyCharm …… *15*

2.3.3　Anaconda …… *17*

2.3.4　Jupyter Notebook …… *22*

2.4　代码书写规则 …… *23*

2.4.1　缩进 …… *23*

2.4.2　逻辑行与物理行 …… *23*

2.4.3　注释 …… *24*

2.4.4　编码风格 …… *25*

2.5　习题 …… *25*

第 3 章　Python 数据类型 …… *26*

3.1　变量 …… *26*

3.1.1　变量命名 …… *26*

3.1.2　变量引用 …… *27*

3.2　运算符 …… *27*

3.2.1　算术运算符 …… *27*

3.2.2　关系运算符 …… *28*

3.2.3　赋值运算符 …… *29*

3.2.4　逻辑运算符 …… *29*

3.2.5　位运算符 …… *30*

3.2.6　成员运算符 …… *31*

3.2.7　身份运算符 …… *31*

3.3　表达式 …… *31*

3.3.1　表达式的概念 …… *31*

3.3.2　运算符的优先级 …… *32*

3.4　数据类型 …… *32*

3.4.1　数据类型的概念 …… *32*

3.4.2　数据类型的分类 …… *33*

3.5　数值 …… *33*

3.5.1　数值的概念 …… *33*

3.5.2　数值的操作 …… *33*

3.6　列表 …… *34*

3.6.1　列表的概念 …… *34*

3.6.2　列表的操作 …… *34*

3.7　元组 …… *39*

3.7.1　元组的概念 …… *39*

3.7.2　元组的操作 …… *39*

3.8　字符串 …… *40*

3.8.1　字符串的概念 …… *40*

3.8.2　字符串的操作 …… *41*

3.9　字典 …… *42*

3.9.1　字典的概念 …… *42*

机械工业出版社

3.9.2 字典的操作 …… 43
3.10 集合 …… 46
3.10.1 集合的概念 …… 46
3.10.2 集合的操作 …… 46
3.10.3 集合运算 …… 48
3.11 组合数据总结 …… 49
3.11.1 相互关系 …… 49
3.11.2 数据类型转换 …… 49
3.12 实例 …… 50
3.12.1 发扑克牌 …… 50
3.12.2 统计相同单词出现的次数 …… 51
3.12.3 计算两个日期间隔天数 …… 51
3.13 习题 …… 52
第 4 章 Python 三大结构 …… 53
4.1 3 种基本结构 …… 53
4.2 顺序结构 …… 53
4.2.1 输入、处理和输出 …… 54
4.2.2 顺序程序设计举例 …… 57
4.3 选择结构 …… 57
4.3.1 单分支 …… 57
4.3.2 双分支 …… 58
4.3.3 多分支 …… 58
4.3.4 分支嵌套 …… 60
4.4 循环概述 …… 61
4.4.1 循环结构 …… 61
4.4.2 循环分类 …… 62
4.5 while 语句 …… 62
4.5.1 基本形式 …… 62
4.5.2 else 语句 …… 63
4.5.3 无限循环 …… 63
4.6 for 语句 …… 64
4.6.1 应用序列类型 …… 64
4.6.2 内置函数 range( ) …… 65
4.7 循环嵌套 …… 65
4.7.1 循环嵌套的概念 …… 65
4.7.2 循环嵌套实现 …… 66
4.8 辅助语句 …… 67
4.8.1 break 语句 …… 67
4.8.2 continue 语句 …… 68
4.8.3 pass 语句 …… 68
4.9 迭代器 …… 69
4.9.1 iter( )方法 …… 69
4.9.2 next( )方法 …… 69
4.10 实例 …… 69
4.10.1 猴子吃桃问题 …… 69
4.10.2 买地铁车票 …… 70
4.10.3 打印金字塔 …… 70
4.10.4 冰雹数列 …… 71
4.10.5 输出特定三角形 …… 71
4.11 习题 …… 72
第 5 章 函数 …… 73
5.1 函数声明与调用 …… 73
5.1.1 函数声明 …… 73
5.1.2 函数调用 …… 73
5.1.3 函数返回值 …… 75
5.2 参数传递 …… 76
5.2.1 实参与形参 …… 76
5.2.2 传对象引用 …… 76
5.3 参数分类 …… 77
5.3.1 必备参数 …… 77
5.3.2 默认参数 …… 78
5.3.3 关键参数 …… 78
5.3.4 不定长参数 …… 78
5.4 两类特殊函数 …… 79
5.4.1 lambda 函数 …… 79
5.4.2 递归函数 …… 80
5.5 变量作用域 …… 82
5.5.1 局部变量 …… 82
5.5.2 全局变量 …… 82
5.6 实例 …… 83
5.6.1 筛选法求素数 …… 83
5.6.2 可逆素数 …… 83
5.6.3 递归求 x^n …… 84
5.6.4 孪生素数 …… 84
5.6.5 汉诺塔 …… 85
5.6.6 完全数 …… 86
5.6.7 逆置 …… 87
5.6.8 气温上升最长天数 …… 87

机械工业出版社

5.6.9 兔子上楼梯 …… 88
5.7 习题 …… 89
第 6 章 线性表 …… 90
6.1 线性表的相关概念 …… 90
6.2 线性表的存储 …… 90
6.2.1 线性存储 …… 90
6.2.2 链式存储 …… 90
6.3 单链表操作 …… 91
6.3.1 单链表的概述 …… 91
6.3.2 单链表的操作实现 …… 91
6.4 栈 …… 93
6.4.1 栈的相关概念 …… 93
6.4.2 栈的操作 …… 94
6.5 队列 …… 95
6.5.1 队列的相关概念 …… 95
6.5.2 队列的操作 …… 96
6.6 字符串 …… 97
6.6.1 字符串的相关概念 …… 97
6.6.2 字符串的操作 …… 97
6.7 实例 …… 98
6.7.1 斐波那契数列 …… 98
6.7.2 判断回文数 …… 99
6.7.3 模式匹配 …… 100
6.7.4 字符串统计 …… 103
6.7.5 Anagrams 问题 …… 104
6.7.6 年龄问题 …… 104
6.7.7 恺撒密码 …… 105
6.8 习题 …… 106
第 7 章 树和二叉树 …… 108
7.1 树和二叉树的概述 …… 108
7.1.1 树和二叉树的相关概念 …… 108
7.1.2 二叉树的性质 …… 109
7.2 二叉树存储 …… 110
7.2.1 顺序存储 …… 110
7.2.2 链式存储 …… 111
7.3 二叉树遍历 …… 111
7.3.1 先序遍历 …… 112
7.3.2 中序遍历 …… 112
7.3.3 后序遍历 …… 112
7.3.4 层序遍历 …… 113
7.4 由遍历序列创建二叉树 …… 113
7.4.1 由先序、中序推出后序遍历 … 113
7.4.2 由中序、后序推出先序遍历 … 114
7.4.3 由先序、后序推出中序遍历 … 114
7.5 二叉树的创建 …… 114
7.6 哈夫曼树 …… 115
7.6.1 哈夫曼编码 …… 115
7.6.2 哈夫曼算法 …… 115
7.7 树和二叉树的关系 …… 119
7.7.1 树的存储 …… 119
7.7.2 树与二叉树转换 …… 120
7.8 实例 …… 121
7.8.1 打印二叉树深度 …… 121
7.8.2 打印二叉树左右视图 …… 122
7.8.3 二叉树左右交换 …… 124
7.8.4 括号组合 …… 125
7.8.5 对称二叉树 …… 126
7.9 习题 …… 127
第 8 章 图 …… 129
8.1 图的概述 …… 129
8.1.1 图的相关概念 …… 129
8.1.2 NetworkX 库 …… 129
8.2 图的存储 …… 130
8.2.1 邻接矩阵 …… 130
8.2.2 邻接表 …… 134
8.3 图的遍历 …… 136
8.3.1 深度优先遍历 …… 136
8.3.2 广度优先遍历 …… 138
8.4 最小生成树 …… 139
8.4.1 克鲁斯卡尔（Kruskal）算法 … 139
8.4.2 普里姆（Prim）算法 …… 142
8.5 最短路径 …… 144
8.5.1 迪杰斯特拉（Dijkstra）算法 … 144
8.5.2 弗洛伊德（Floyd）算法 …… 148
8.6 实例 …… 151
8.6.1 旅游路线 …… 151
8.6.2 单词搜索 …… 152
8.7 习题 …… 153

机械工业出版社

第 9 章　查找 …… *154*
9.1　查找算法 …… *154*
9.2　基于线性表查找 …… *154*
9.2.1　顺序查找 …… *154*
9.2.2　二分查找 …… *156*
9.2.3　分块查找 …… *157*
9.3　二叉排序树 …… *159*
9.3.1　二叉排序树的特性 …… *159*
9.3.2　二叉排序树的操作 …… *161*
9.4　平衡二叉树 …… *164*
9.4.1　平衡因子 …… *165*
9.4.2　构建平衡二叉树 …… *166*
9.5　哈希表 …… *168*
9.6　哈希算法 …… *168*
9.6.1　哈希函数 …… *168*
9.6.2　Python 内置方法 …… *169*
9.7　解决冲突的方法 …… *169*
9.7.1　开放定址法 …… *170*
9.7.2　链地址法 …… *172*
9.8　Python 自身查找算法 …… *172*
9.9　实例 …… *173*
9.9.1　查找最大值或最小值 …… *173*
9.9.2　二分查找法递归实现 …… *174*
9.9.3　查找出现次数最多的整数 …… *174*
9.10　习题 …… *175*
第 10 章　排序 …… *176*
10.1　排序概述 …… *176*
10.2　插入排序 …… *177*
10.2.1　直接插入排序 …… *177*
10.2.2　折半插入排序 …… *178*
10.2.3　希尔排序 …… *179*
10.3　交换排序 …… *181*
10.3.1　冒泡排序 …… *181*
10.3.2　快速排序 …… *182*
10.4　选择排序 …… *184*
10.4.1　简单选择排序 …… *184*
10.4.2　堆排序 …… *186*
10.5　归并排序 …… *191*
10.6　排序总结 …… *192*
10.6.1　时间性能 …… *192*
10.6.2　空间性能 …… *192*
10.6.3　稳定性能 …… *193*
10.6.4　排序算法的选择准则 …… *193*
10.7　Python 自身排序算法 …… *193*
10.7.1　sorted( ) …… *194*
10.7.2　list. sort( ) …… *194*
10.7.3　list. reverse( ) …… *194*
10.8　实例 …… *194*
10.8.1　有序序列插入元素 …… *194*
10.8.2　求解第二大整数 …… *195*
10.8.3　输出最小的 k 个数 …… *196*
10.9　习题 …… *197*
第 11 章　异常处理与调试 …… *198*
11.1　错误类型 …… *198*
11.1.1　语法错误 …… *198*
11.1.2　运行时错误 …… *198*
11.1.3　逻辑错误 …… *198*
11.2　捕获和处理异常 …… *199*
11.2.1　try…except…else 语句 …… *199*
11.2.2　try…finally 语句 …… *200*
11.2.3　raise 语句 …… *200*
11.2.4　自定义异常 …… *201*
11.3　3 种调试手段 …… *201*
11.4　Python 调试工具 …… *202*
11.4.1　IDLE …… *202*
11.4.2　IPDB …… *203*
11.4.3　Spyder …… *204*
11.4.4　PDB …… *205*
11.4.5　PyCharm …… *206*
11.5　习题 …… *209*
附录 …… *210*
附录 A　软件考试和软件竞赛 …… *210*
A.1　全国计算机等级考试二级 Python 语言程序设计考试（2018 年版） …… *210*
A.1.1　基本要求 …… *210*
A.1.2　考试内容 …… *210*
A.1.3　考试方式 …… *211*

机械工业出版社

A.2 ACM 国际大学生程序设计竞赛 …… 212
A.2.1 在线判题系统 …… 212
A.2.2 ACM 训练环境 …… 212
A.2.3 ACM 的算法知识点 …… 215
A.3 CSP 认证 …… 219
A.3.1 CSP 认证简介 …… 219
A.3.2 认证形式 …… 220
A.3.3 涉及知识点 …… 220
A.4 牛客网 …… 220
A.5 力扣 …… 221
附录 B 图论相关模块 …… 222
B.1 NumPy …… 222
B.1.1 NumPy 简介 …… 222
B.1.2 创建数组 …… 222
B.1.3 查看数组 …… 224
B.1.4 索引和切片 …… 224
B.1.5 矩阵运算 …… 225
B.1.6 5 个 NumPy 函数 …… 226
B.2 Matplotlib …… 229
B.2.1 Matplotlib 简介 …… 229
B.2.2 5 种图形 …… 229
B.3 NetworkX …… 232
B.3.1 图 …… 232
B.3.2 节点 …… 233
B.3.3 边 …… 234
B.3.4 相关属性 …… 236
B.4 在线图结构绘制工具 …… 238
B.4.1 Graph Editor …… 238
B.4.2 Graphviz …… 238
附录 C 更多数据类型 …… 239
C.1 collections 模块 …… 239
C.1.1 namedtuple …… 239
C.1.2 deque …… 239
C.1.3 Counter …… 242
C.1.4 OrderedDict …… 242
C.1.5 ChainMap …… 243
C.2 heapq 模块 …… 243
C.3 array 模块 …… 245
附录 D 参考答案 …… 248
参考文献 …… 267

机械工业出版社

第 1 章　数据结构与算法

本章首先介绍了什么是程序，其次介绍了数据结构和算法的相关知识（如算法的 3 个层次、5 个属性等），然后重点介绍了算法的空间复杂度和时间复杂度，最后介绍了算法的几种表示方式。

1.1　程序

程序是为实现特定目标或解决特定问题而用计算机语言编写的命令序列的集合。程序设计过程如图 1.1 所示，详细步骤如下所述。

（1）分析问题

对于所需解决的问题及最后应达到的要求要进行认真的分析，确保在任务一开始就对它有详细而确切的了解。

（2）设计数据结构与算法

分析问题，构造模型。在得到一个基本的物理模型后，用数学语言描述它，如列出解题的数学公式或联立方程式，即建立数学模型。找出解决问题的关键之处，即找出解决问题的方法和具体步骤，设计数据结构与算法。

（3）绘制流程图

将算法用流程框图或者伪代码等形式表示出来，使得编程思路清楚，减少程序编写错误。

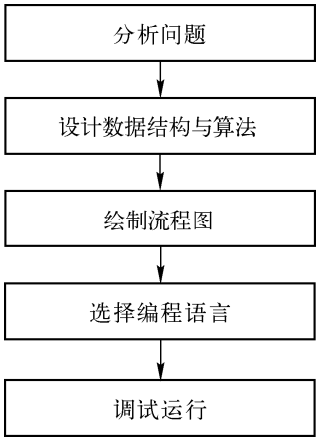

图 1.1　程序设计过程

（4）选择编程语言

将框图或者伪代码等转换为符合特定计算机程序设计语言的语法并编程，对源程序进行编辑、编译和链接。

（5）调试运行

调试程序，发现和排除程序故障，得到必要的运算结果。

著名的瑞士计算机科学家沃思（N. Wirth）教授曾提出：

$$\text{程序} = \text{数据结构} + \text{算法} \tag{1-1}$$

高效的程序需要在数据结构的基础上设计和选择算法。其中，数据结构是算法需要处理问题的载体，解决了“如何描述数据”的问题。算法解决了“如何操作数据”的问题。算法是灵魂，没有算法，编程就是无米之炊。编程语言是工具，没有编程语言，就无法实现算法。以程序设计为手段，将数据结构和算法紧密结合。

机械工业出版社

1.2 数据结构

1.2.1 数据结构的核心地位

数据结构（data structure）在计算机学科中具有重要的地位，是操作系统、人工智能、计算机组成原理、程序设计、软件工程、数据库，以及编译原理等课程的重要基础。数据结构在计算机学科中的地位如图 1.2 所示。

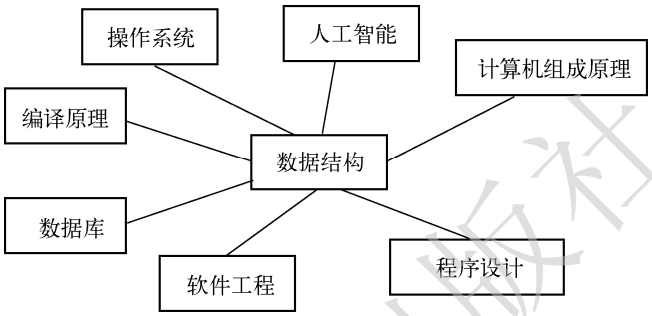

图 1.2　数据结构在计算机学科中的地位

1.2.2 数据结构的组成

计算机处理问题分为数值性问题和非数值性问题。随着计算机应用领域的扩大和软、硬件的发展，非数值性问题显得越来越重要。据统计，当今 90%以上的计算机时间用来处理非数值性问题，这类问题涉及的数据结构更为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决此类问题的关键不再是分析计算方法，而是要设计出合适的数据结构。

数据结构研究相关的各种信息如何表示、组织、存储与加工处理，研究数据的逻辑结构和数据的物理结构，以及它们之间的相互关系。数据结构通常由 3 个部分组成，即数据的逻辑结构、数据的物理结构和数据的运算结构。

（1）逻辑结构

数据的逻辑结构包括集合、线性结构、树形结构和图形结构，如下所述。

- 集合：数据结构中的元素之间除了“同属一个集合”的相互关系外，别无其他关系。
- 线性结构：数据结构中的元素存在一对一的相互关系。
- 树形结构：数据结构中的元素存在一对多的相互关系。
- 图形结构：数据结构中的元素存在多对多的相互关系。

（2）物理结构

数据的物理结构是数据结构在计算机中的表示（又称映像），它包括数据元素的机内表示和关系的机内表示。常用两种存储结构：顺序存储结构和链式存储结构。同一种逻辑结构可以有多种不同的物理存储方式。

- 顺序存储结构依据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。
- 链式存储结构借助指示元素存储位置的指针来表示数据元素之间的逻辑关系。

（3）运算结构

数据的运算结构是指在数据逻辑结构上的操作算法，如检索、插入、删除、更新和排序等。

1.3 算法

下面用一道例题来说明不同的算法效率不同。

【例 1-1】找假币：假设 n(n≥2)枚硬币中有一枚为假币，假币比真币轻，怎样才能找出假币？

方法 1：一个个比较硬币，直到找出假币为止。假设 n=10，首先比较硬币 1 和硬币 2，会出现两种情况。

- 如果重量不一样，较轻者即为假币。
- 如果重量一样，则选取两枚中任意一枚与其他的硬币比较。

如上依次比较硬币 3、4、5……，直到找出假币。在最差的情况下，比较 9 次才能找出假币。即从 n 枚硬币中找出假币，需要比较 n-1 次，比较过程如图 1.3 所示。

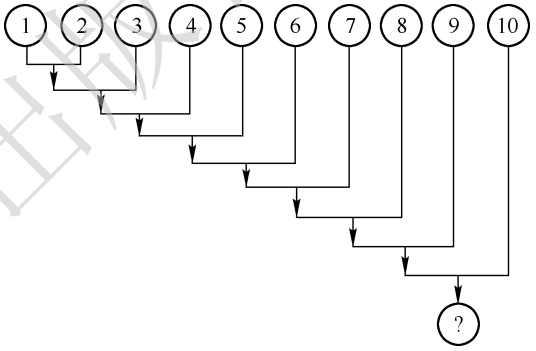

图 1.3　方法 1 示意图

方法 2：方法 1 中，若两枚硬币重量一样，说明都是真币，无须再进行比较。因此，将 n 枚硬币每两枚分为一组进行比较，会出现两种情况。

- 如果重量不一样，较轻者即为假币。
- 如果重量一样，就继续比较下一组的两枚硬币。

如上依次进行比较，直到找出假币。在最差的情况下，比较 5 次就可找出假币。即从 n 枚硬币中找出假币，需要比较 n/2 次，比较过程如图 1.4 所示。

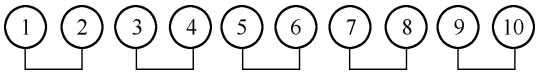

图 1.4　方法 2 示意图

方法 3：方法 2 中，既然所有真币重量一样，将 n 枚硬币分为两组进行比较，有假币的一组必然轻些；再将较轻的这一组等分为两组进行比较，以此类推，直到找出假币。从 n 枚硬币中找出假币，需要比较 $\log_2 n$ 次。10 枚硬币比较过程如图 1.5 所示。

可以看到，方法 3 比较的次数最少，不同的方法（算法）效率差距很大。

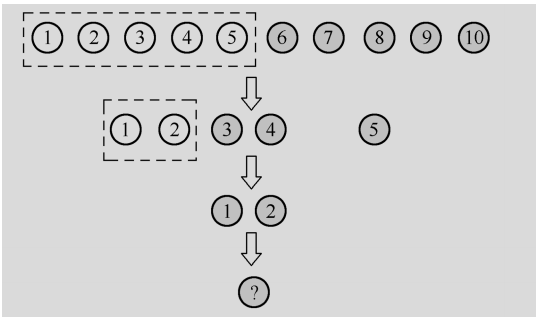

图 1.5　方法 3 示意图

机械工业出版社

1.3.1 算法的5个属性

算法是解决一个问题而采取的方法和步骤，是对解题方案的准确而完整的描述，是一系列解决问题的清晰指令。通过一定规范的输入、处理，在有限时间内获得输出的整个过程，算法与具体的程序语言无关，具备以下5个特性。

- 确定性。算法的每个步骤都应确切无误，没有二义性。
- 可行性。算法的每个步骤都必须满足计算机语言能够有效执行、可以实现的要求，并可得到确定的结果。
- 有穷性。算法包含的步骤必须是有限的，并可以在一个合理的时间限度内执行完毕，不能无休止地执行下去。例如，计算圆周率，只能精确到小数点后某一位。
- 输入性。由于算法的操作对象是数据，因此应在执行操作前提供数据，执行算法时可以有多个输入，例如，求两个整数 m 和 n 的最大公约数，则需要输入 m 和 n 的值；当然也可以没有输入，例如，求4！等。
- 输出性。一般提供1个或多个输出。

【例1-2】从键盘输入三角形的3条边，求三角形的面积。

【解析】其算法步骤如下所述。

1）从键盘任意输入3个整数，用 a、b、c 存储。

2）判断 a、b、c 是否符合三角形的定义（两边之和大于第三边）。

3）如果符合三角形定义，先求出周长的一半 s=(a+b+c)/2，再调用海伦公式，area=$\sqrt{s(s-a)(s-b)(s-c)}$，求出三角形面积 area。

4）输出 area。

下面，用算法的5个特性来分析【例1-2】。

- 确定性。【例1-2】共有4个步骤，每一个步骤都有确定的含义，没有二义性。
- 可行性。【例1-2】的每个步骤都可以用高级程序设计语言，如 Python 语言或 C 语言等实现。
- 有穷性。【例1-2】只有4个步骤，是有限的。
- 输入性。【例1-2】有3个输入，a、b、c 分别代表三角形的3条边。
- 输出性。【例1-2】有一个输出，area 代表三角形的面积。

1.3.2 算法的3个层次

算法是程序设计的核心内容。算法的学习大致分为3个层次，如表1.1所示。

表1.1 算法的3个层次

层　次	内　容
第一层	基本算法，如排序、查找和递归法等
第二层	涉及算法的时间复杂度和空间复杂度，如分治法、贪心算法和动态规划法等
第三层	涉及智能优化算法的学习，如遗传算法、蚁群算法和聚类算法等

机械工业出版社

第一层是“算法基础教学阶段”，涉及基本的算法和程序设计方法，如查找、排序和递归程序设计等。典型的课程是《数据结构》。

第二层是“算法提高教学阶段”，涉及重要的算法设计方法，如分治法、动态规划法、贪心法和回溯法等，理解算法的时间和空间复杂度，以及复杂度分析等重要概念。典型的课程是《算法设计与分析》。

第三层是“算法高级教学阶段”，涉及工程应用中和数据智能处理相关的一些重要算法和模型，如最优化方法（如梯度下降法）、遗传算法和神经网络算法等。典型的课程是《工程最优化方法》《模式识别》《人工智能》等。

按照算法的3个层次进行递进式学习，一般会经历阅读与分析程序、模仿编程、掌握常见程序模块、简单编程及复杂编程等过程，学习步骤如下所述。

1）学习一门语言，如C、C++、Python和Java。

2）熟悉基本的算法，如查找、排序等。

3）掌握数据结构，特别是树和图。

4）在各种刷题网站上进行实践练习，具体网址如下。

https://leetcode-cn.com/problemset/algorithms/

https://www.luogu.com.cn/training/mainpage

https://vjudge.net/contest

http://codeforces.com/contests

http://acm.hdu.edu.cn/

http://bestcoder.hdu.edu.cn/

1.4 算法复杂度

一个算法的优劣主要从算法的执行时间和所需要占用的存储空间两个方面来衡量，即用空间复杂度和时间复杂度来衡量程序的效率。

1.4.1 空间复杂度

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度，记作S(n)=O(f(n))。

一个算法在计算机存储器上所占用的存储空间，包括算法的输入、输出数据所占用的存储空间、算法本身所占用的存储空间和算法在运行过程中临时占用的存储空间。

- 算法的输入、输出数据所占用的存储空间由要解决的问题决定，是通过参数表由调用函数传递而来的，它不随算法的不同而改变。
- 算法本身所占用的存储空间与算法编写的长短成正比，要压缩这方面的存储空间，就必须编写较短的算法。
- 算法在运行过程中临时占用的存储空间随算法的不同而异。

一个算法所占用的存储空间要从各方面综合考虑。递归算法一般都比较简短，算法本身所占用的存储空间较小，但运行时需要一个附加堆栈，会占用较多的临时存储空间。非递归算法一般较长，因此存储算法本身的空间较大，但运行时需要的存储空间较小。

机械工业出版社

1.4.2 时间复杂度

计算机科学中，算法的时间复杂度是一个函数，通常用 O 符号表述，用于定量描述该算法的运行时间。算法中模块 n 基本操作的重复执行次数计为函数 f(n)，算法的时间复杂度为 T(n)= O(f(n))。

一个算法运行的总时间取决于以下两个方面。

- 每条语句执行一次所需的时间。
- 每条语句的执行次数。

每条语句的执行时间为该语句的执行次数乘以该语句执行一次所需时间。

常见的时间复杂度有：常数阶 O(1)、对数阶 O(logn)、线性阶 O(n)、线性对数阶 O(nlogn)、平方阶 O(n^2)、立方阶 O(n^3)等，如表 1.2 所示。

表 1.2　大 O 表示法

f(n)	函　数　名
1	常数函数
logn	对数函数
n	线性函数
nlogn	线性对数函数
n^2	二次函数
n^3	三次函数
2^n	指数函数

1.4.3 提高算法效率的方法

下面介绍提高算法效率的几种方法。

1. 降低程序复杂度

程序复杂度主要是指模块内部程序的复杂度，往往采用 McCabe 度量法，用于计算程序模块中环路的个数。实践表明，当程序内分支数或循环个数增加时，环形复杂度也随之增加，模块的环形复杂度以 V(G)≤10 为宜，也就是说，V(G)= 10 是环形复杂度的上限。

环形复杂度 V(G)主要有如下 3 种方法。

- 将环形复杂度定义为控制流图中的区域数。
- V(G)= E−N+2，E 是控制流图中边的数量，N 是控制流图中节点的数量。
- V(G)= P+1，P 是控制流图 G 中判定节点的数量。

【例 1−3】计算图 1.6 的环形复杂度。

【解析】

1）V(G)= 6。

分析：图中的区域数为 6。

2）V(G)= E−N+2 = 16−12+2 = 6。

分析：其中 E 为控制流图中的边数，N 为节点数。

3）V(G)= P+1 = 5+1 = 6。

分析：其中 P 为谓词节点的个数。在控制流图中，节点 2、3、5、6、9 是谓词节点。

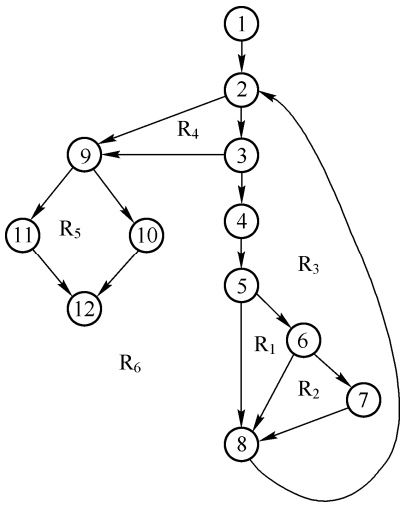

图 1.6　控制流图

2. 选用高效率算法

对算法的空间复杂度和时间复杂度进行优化，从而选择高效的算法。

【例 1−4】鸡兔同笼问题：鸡兔共有 30 只，脚共有 90 只，问鸡、兔各有多少只？

【解析】设鸡为 x 只，兔为 y 只，根据题目要求，列

机械工业出版社

出方程组为：

$$\begin{aligned} x+y&=30 \\ 2x+4y&=90 \end{aligned} \tag{1-2}$$

采用“试凑法”解决方程组的求解问题，将 x 和 y 变量的每一个值都带入方程中进行尝试。

方法 1：利用二重循环来实现。

```
for x in range(0,31):
    for y in range(0,31):
        if (x + y == 30 and 2 * x + 4 * y == 90):
            print("Chicken is " ,x)
            print("rabbit is " , y)
```

【程序运行结果】

```
Chicken is  15
rabbit is  15
```

时间复杂度：

$T(n)=O(n*n)=O(n^2)$

方法 2：利用一重循环来实现。

```
for x in range(0,31):
    y=30-x
    if 2 * x + 4 * y == 90:
      print("Chicken is " ,x)
      print ("rabbit is " , y)
```

时间复杂度：

$T(n)=O(n)$

方法 3：假设鸡兔共有 a 只，脚共有 b 只，a 为 30，b 为 90。那么方程组为：

$$\begin{cases} x+y=a \\ 2x+4y=b \end{cases} \Rightarrow \begin{aligned} X&=(4a-b)/2 \\ Y&=(b-2a)/2 \end{aligned} \tag{1-3}$$

【代码】

```
a = 30;b = 90
x= (4 * a - b) //2
y= (b - 2 * a) //2
print("Chicken is " ,x)
print("rabbit is " , y)
```

时间复杂度：

$T(n)=O(1)$

1.5 算法表示方式

程序设计采用自然语言描述容易产生二义性，即歧义。例如，英文单词“doctor”的汉

机械工业出版社

语含意是“博士”或“医生”，需要根据“doctor”所处的语境决定其含义。为了让算法表示的含义更为准确，往往采用流程图、N-S 图和伪语言等。

1.5.1 流程图

流程图是描述算法最常用的一种方法，通过几何框图、流向线和文字说明等流程图符号表示算法。流程图具有以下优点。

- 采用简单规范的符号，画法简单。
- 结构清晰，逻辑性强。
- 便于描述，容易理解。

流程图如图 1.7 所示，主要采用以下符号进行问题的描述。

- 起止框用于流程的开始和结束。
- 输入框向程序输入数据，输出框用于程序向外输出信息。
- 箭头用来控制流向。
- 执行框又称为方框，用于表示一个处理步骤。
- 判别框又称菱形框，用于表示一个逻辑条件。

【例 1-5】流程图举例。

【题意】求最大公约数，流程图如图 1.8 所示。

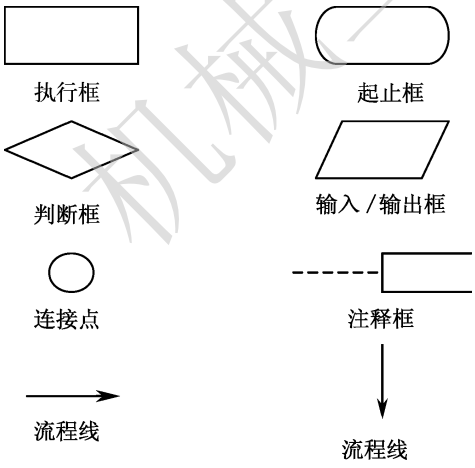

图 1.7 流程图基本符号

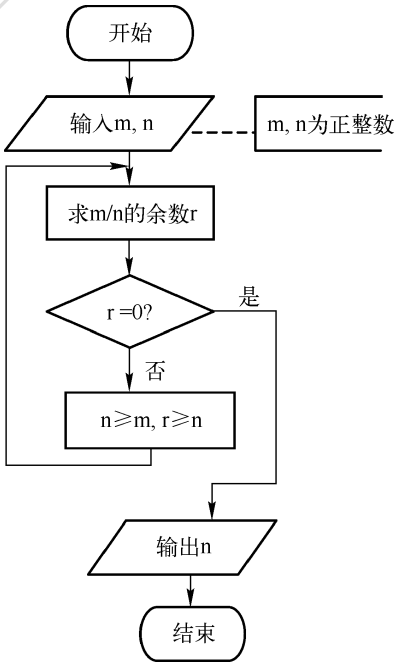

图 1.8 求最大公约数的算法

1.5.2 N-S 图

1973 年美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图形式，称为 N-S 图。N-S 图删除了带箭头的流程线，避免了流程无规律随意转移。N-S 图如图 1.9 所示。

- 顺序结构：语句 1、语句 2 和语句 3 这 3 个框组成一个顺序结构。
- 选择结构：当“条件”成立时执行“选择体 1”操作，“条件”不成立则执行“选择

机械工业出版社

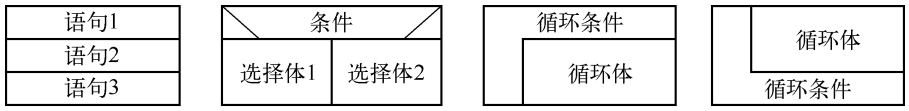

图 1.9　N-S 结构流程图基本元素框

体 2”操作结构。

- 循环结构：循环结构分为当型循环结构和直到型循环结构两种。
  - 当型循环结构。先判断后执行，当“循环条件”成立时反复执行“循环体”操作，直到“循环条件”不成立为止。
  - 直到型循环结构。先执行后判断，当“循环条件”不成立时反复执行“循环体”操作，直到“循环条件”成立为止。

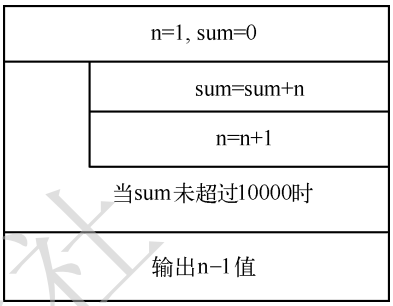

图 1.10　N-S 结构流程图

【例 1-6】 N-S 图举例。

【题意】求整数 1~n 之和不超过 10000 时 n 的最大值，N-S 图如图 1.10 所示。

1.5.3　伪语言

伪语言（Pseudocode）也称伪代码，介于自然语言和计算机语言之间，并不是真正存在的编程语言。伪代码综合使用多种编程语言中的语法、保留字，甚至会用到自然语言，不采用图形符号，因此书写方便、格式紧凑，便于向计算机编程语言（如 Pascal、C 和 Java 等）过渡。

【例 1-7】 伪代码举例。

【题意】输入 3 个数，打印输出其中最大的数。伪代码如下所示。

```
Begin(算法开始)
输入 A,B,C
IF A>B 则 A→Max
否则 B→Max
IF C>Max 则 C→Max
Print Max
End (算法结束)
```

1.6　习题

1. 程序是什么？
2. 什么是算法？算法的 5 个属性是什么？
3. 如何理解算法的空间复杂度和时间复杂度？

河海大学出版社

第 2 章　Python 开发环境

本章首先介绍了 Python 的相关知识，包括 Python 语言的特点、应用场合等。然后介绍了 Python 解释器和 Python 编辑器的安装和配置。最后介绍了代码书写的相关规则。

2.1　Python 简介

2.1.1　Python 的特点

Python 是一种简单易学、功能强大的编程语言，具有高效率的高级数据结构，可方便且有效地实现面向对象编程。

1. 简单易学

Python 语法简洁清晰、结构简单，易于快速上手。由于 Python 不过多计较程序语言在形式上的细节和规则，从而便于编程者专注程序本身的逻辑和算法。

2. 免费开源

Python 是自由/开源软件（Free/Libre and Open source software，FLOSS）之一，人们可以自由地发布这个软件的副本、阅读它的源代码、对它做改动，以及将它用于新的自由软件中。

3. 便于移植

计算机并不能直接接收和执行用高级语言编写的源程序，源程序在输入计算机时，通过“翻译程序”翻译成机器语言形式的目标程序，计算机才能识别和执行。这种“翻译”通常有两种方式：一种是编译执行；另一种是解释执行。编译执行是指源代码先由编译器编译成可执行的机器码，然后再执行；解释执行是指源代码被解释器直接读取执行。编译执行和解释执行各有优缺点，编译执行可一次性将高级语言源程序编译成二进制的可执行指令，通常执行效率高；而解释执行是由该语言（如 HTML）的运行环境（如浏览器）读取一条该语言的源程序，然后转变成二进制指令交给计算机执行，通常可以灵活地跨平台。C、C++等采用编译执行方式，Python 与 Java 语言类似，采用解释执行方式，源代码不需要编译成二进制代码，而是通过解释器把源代码转换成称为字节码的中间形式，由虚拟机负责在不同的计算机上运行，因此，Python 程序便于移植，可在众多平台运行。

4. 面向对象

Python 是完全面向对象的语言。函数、模块、数字和字符串都是对象，并且完全支持继承、重载、派生和多重继承。Python 语言编写程序无须考虑硬件和内存等底层细节。

机械工业出版社

5. 具有丰富的库

Python 称为胶水语言，能够轻松地与其他语言（特别是 C 或 C++）连接在一起，其具有丰富的 API 和标准库，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统和 GUI 等，可以完成各种工作。

2.1.2 Python 的应用场合

Python 功能强大，主要应用于以下场合。

（1）GUI 软件开发

Python 具有 wxPython、PyQt 等工具，使得 Python 可以快速开发出 GUI，并且不做任何改变就可以运行在 Windows、Linux 和 macOS 等平台。

（2）网络应用开发

Python 提供了标准 Internet 模块，可以广泛应用到各种网络任务中，包括服务端和客户端，另外，网站编程第三方工具 HTMLGen、mod_python、Django、TurboGears 和 Zop 可以帮助 Python 快速构建功能完善和高质量的网站。

（3）游戏开发

Pygame 是建立在 SDL（Simple DirectMedia Layer）基础上的软件包，提供了简单的方式控制媒体信息（如图像、声音等），专为电子游戏设计使用。Pygame 下载网址为 www. pygame. org，如图 2. 1 所示。

图 2. 1　Pygame 下载网址

（4）科学计算

Python 具有科学计算的三剑客：NumPy、SciPy 和 Matplotlib。其中，NumPy 负责数值计算、矩阵操作等；SciPy 负责常见的数学算法，如插值、拟合等；Matplotlib 负责数据可视化。

（5）Web 与移动设备应用开发

web2py 是一种免费开源的 Web 开发框架，帮助开发者设计、实施和测试 MVC（模型（Model）、视图（View）、控制器（Controller））模型。web2py 下载网址为 www. web2py. com，如图 2. 2 所示。

图 2. 2　web2py 下载网址

（6）数据库开发

Python 支持所有主流数据库，如 Oracle、Sybase、MySQL、PostgreSQL 和 Informix 等，并通过标准的数据库 API 接口将关系数据库映射到 Python 类，实现面向对象数据库系统。

（7）系统编程

Python 支持对系统级别的编程，利用 API 等函数接口，可以对系统服务进行管理。Python 程序可以搜索文件和目录树，可以运行其他程序，可以用进程或线程进行并行处理等。

2. 2　Python 解释器

2. 2. 1　Ubuntu 下安装 Python

Ubuntu（乌班图）是一个以桌面应用为主的 Linux 操作系统，基于 Debian 发行版和 GNOME 桌面环境，与 Debian 不同，它每 6 个月会发布一个新版本。Ubuntu 的目标在于为用户提供最新的、同时又相当稳定的自由软件构建的操作系统。

在 Ubuntu 下安装 Python 3 版本，步骤如下所述。

1）下载安装包。

```
wget https://www. python. org/ftp/python/3. 6. 0/Python-3. 6. 0a1. tar. xz
```

2）将压缩包进行解压。

```
tar xvf  Python-3. 6. 0a1. tar. xz
```

3）创建安装文件的路径。

```
mkdir /usr/local/python3
```

4）编译安装。

```
./configure --prefix=/usr/local/python3
make
make install
```

5）测试是否安装成功。

输入 python 3 进行测试，按〈Ctrl+D〉组合键退出。

2.2.2 Windows 下安装 Python

Windows 下安装 Python 的步骤如下。

1）下载 Python 3.6.0 安装包进行安装。下载网址为 http://www.python.org，如图 2.3 所示。

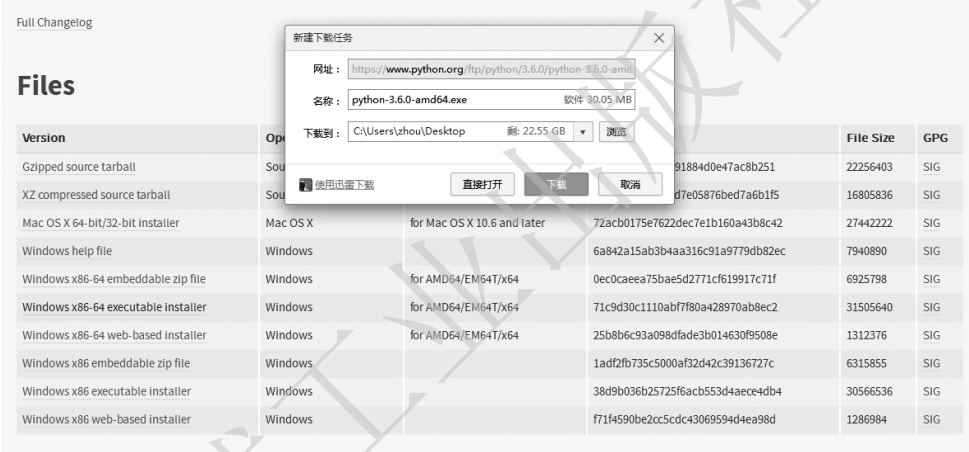

图 2.3　下载 Python 3.6.0

2）在 Windows 环境变量中添加 Python，将 Python 的安装目录添加到 Windows 下的 PATH 变量中，如图 2.4 所示。

图 2.4　设置环境变量

3）测试 Python 是否安装成功。

在 Windows 下使用 cmd 打开命令行，输入 Python 命令，如图 2.5 所示表示安装成功。

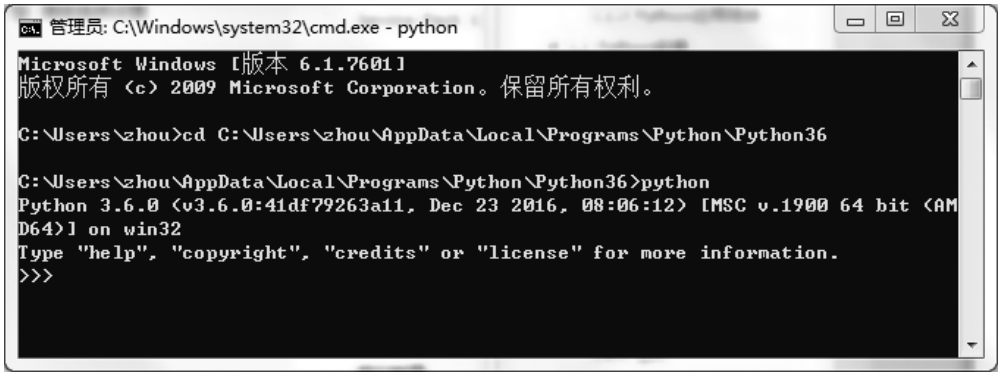

图 2.5　测试 Python 是否安装成功

2.3　Python 编辑器

Python 编辑器众多，有 Python 自带的 IDLE 编辑器、Notepad + +、Eclipse + PyDev、UliPad，以及 Vim 和 Emacs 等。Linux 下的 Eclipse+PyDev 和 Windows 下的 PyCharm 功能较为强大。

2.3.1　IDLE

IDLE 包括能够利用颜色突出显示语法的编辑器、调试工具、Python Shell 以及完整的 Python 3 在线文档集。Python 的 IDLE 具有命令行和图形用户界面两种方式。采用命令行交互式执行 Python 语句，方便快捷，但必须逐条输入语句，不能重复执行，适合测试少量的 Python 代码，不适合复杂的程序设计。

IDLE 的命令行交互式模式如图 2.6 所示。

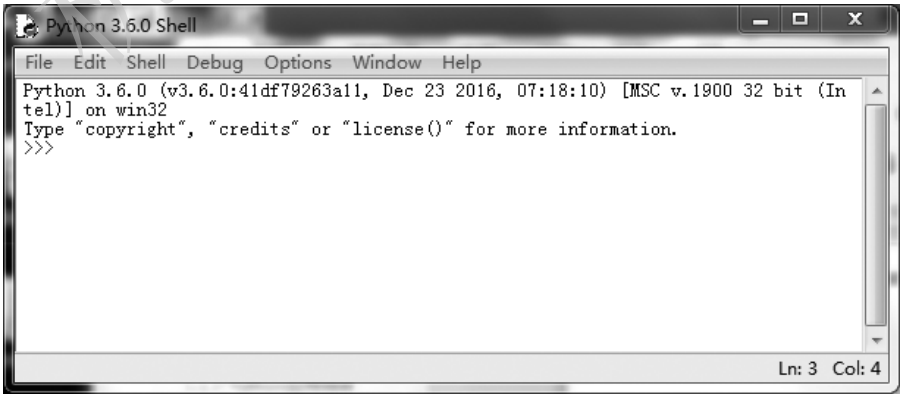

图 2.6　IDLE 的命令行交互式模式

IDLE 图形用户界面模式如图 2.7 所示。

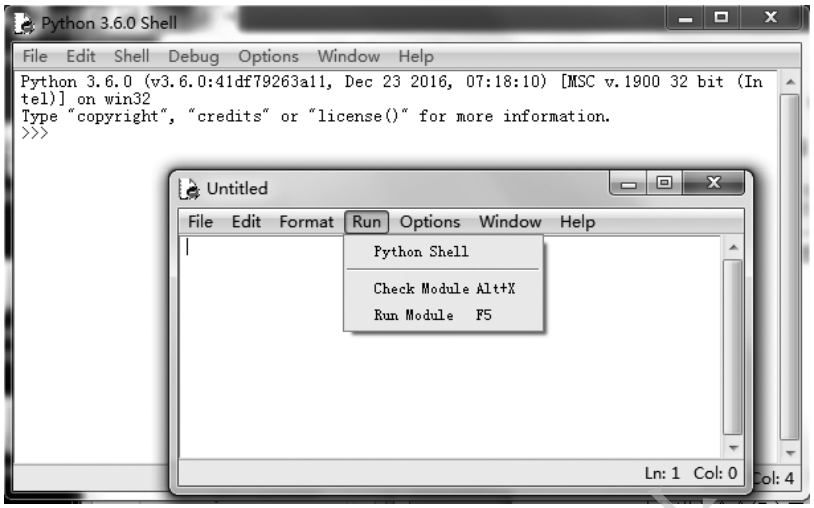

图 2.7　IDLE 的图形用户界面模式

2.3.2　PyCharm

PyCharm 具有一整套可以帮助用户在使用 Python 语言开发时提高效率的工具，如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试和版本控制。此外，PyCharm 还提供了一些高级功能，用于支持 Django 框架下的专业 Web 开发。下载 PyCharm 并双击安装，如图 2.8 所示。

图 2.8　安装 PyCharm 步骤 1

单击 Next 按钮，在出现的界面中单击 Install 按钮进入安装过程，如图 2.9 所示。

安装结束，单击 OK 按钮运行 PyCharm，如图 2.10 所示。

单击 Create New Project，在弹出的对话框中输入项目名、路径，并选择 Python 解释器。如图 2.11 所示。

机械工业出版社

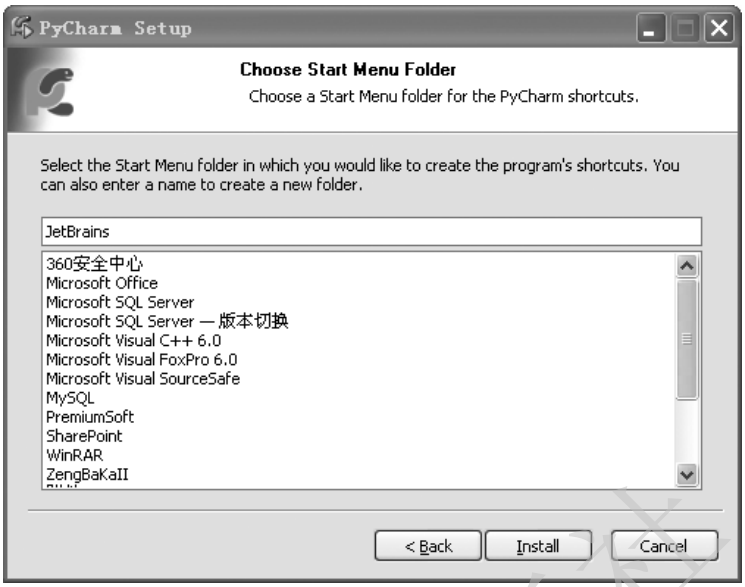

图 2.9　安装 PyCharm 步骤 2

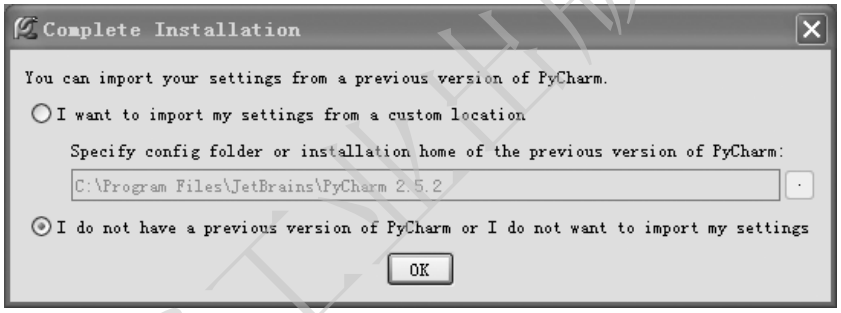

图 2.10　运行 PyCharm

图 2.11　选择 Python 解释器

启动 PyCharm，创建 Python 文件，如图 2.12 所示。

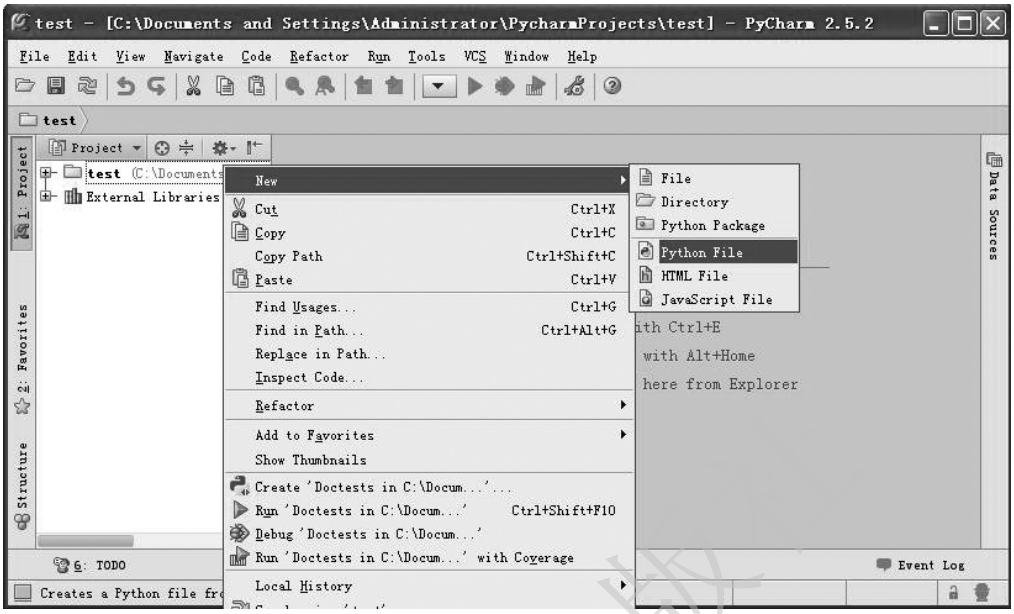

图 2.12　PyCharm 创建 Python 文件

2.3.3　Anaconda

Anaconda 是一个开源的 Python 发行版本，其包含了 Conda、Numpy 等 180 多个科学包及其依赖项，涉及数据可视化、机器学习和深度学习等多方面，本书重点介绍 Anaconda，所有程序均在 Anaconda 下调试与运行。Anaconda 的特点如下。

- 提供包管理。可以使用 conda 和 pip 命令安装、更新 、卸载第三方工具包，简单方便，不需要考虑版本等问题。
- 集成了数据科学相关的工具包。Anaconda 集成了如 NumPy、SciPy、Pandas 等数据分析的各类第三方包。
- 提供虚拟环境管理。在 Conda 中可以建立多个虚拟环境，可以为不同的 Python 版本项目建立不同的运行环境，从而解决了 Python 多版本并存的问题。

Anaconda 的安装步骤如下所述。

1）Anaconda 的官网下载地址为 https://www.anaconda.com/download/，如图 2.13 所示。

2）选择 Python 3.6 version，并根据自己的操作系统是 32 位还是 64 位选择对应的版本下载，如图 2.14 所示。

3）在弹出的对话框中选择下载位置，下载 Anaconda3-5.1.0-Windows-x86_64.exe，大约 537 MB。单击“下载”按钮进行下载，如图 2.15 所示。

注意：如果是 Windows 10 系统，安装 Anaconda 软件时，需要右击安装软件，选择以管理员的身份运行。

4）下载完毕，双击软件进行安装，选择安装路径，如 C:\anaconda3，根据提示单击“下一步”按钮即可完成安装，如图 2.16 所示。

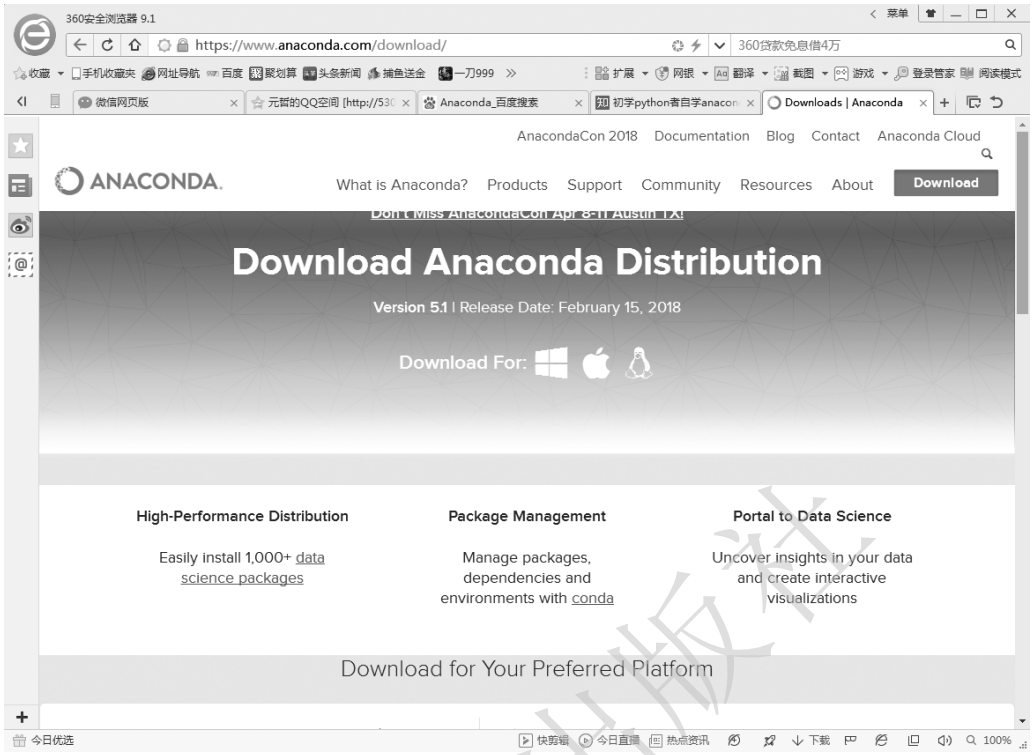

图 2.13　Anaconda 的网站

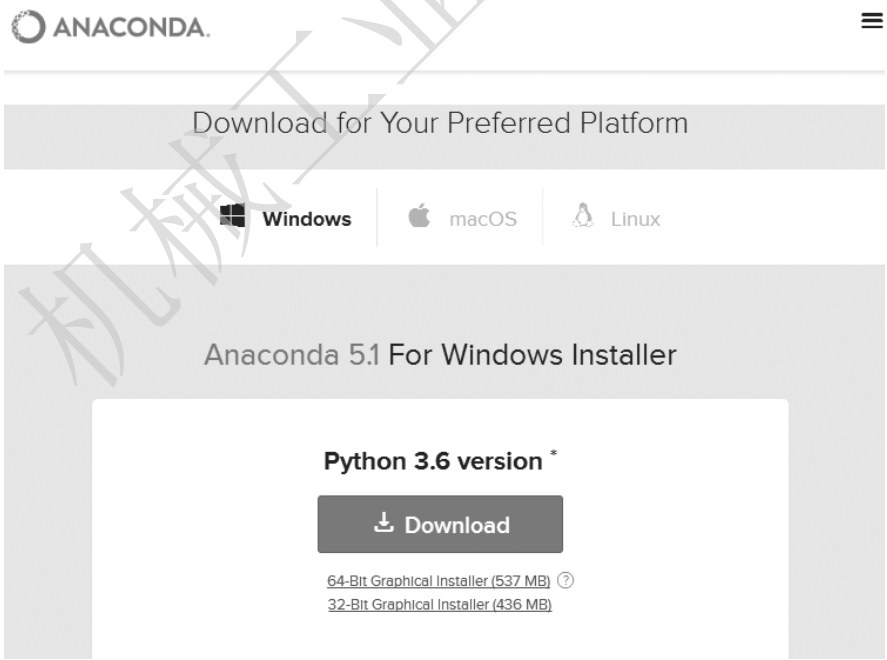

图 2.14　选择 Python 3.6

Anaconda 包含如下应用，如图 2.17 所示。

- Anaconda Navigator：用于管理工具包和环境的图形用户界面，其涉及的众多管理命令也可以在 Navigator 中手工实现。
- Anaconda Prompt：Python 的交互式运行环境。

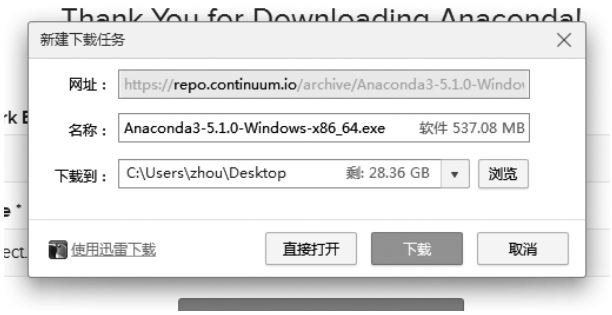

图 2.15　下载 Anaconda 文件

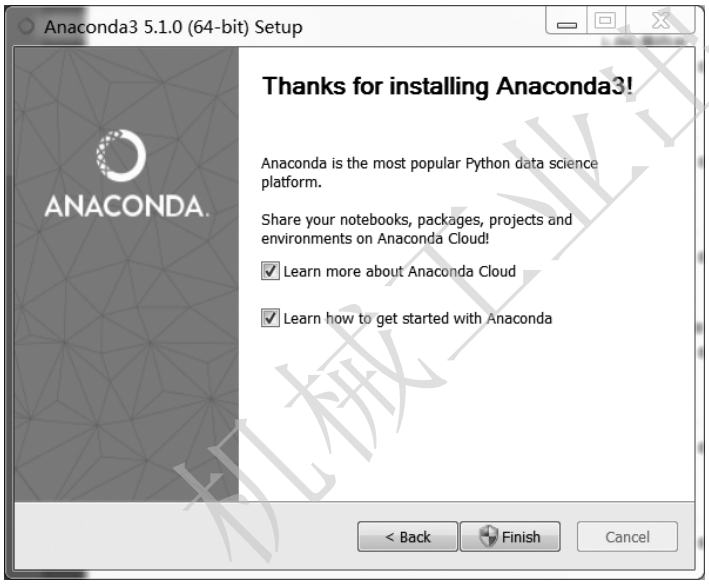

图 2.16　程序运行结果

图 2.17　Anaconda 包含应用

- Jupyter Notebook ：基于 Web 的交互式计算环境，可以编辑易于人们阅读的文档，用于展示数据分析的过程。
- Spyder ：一个使用 Python 语言、跨平台的科学运算开发环境。相对于 PyDev、PyCharm、PTVS 等 Python 编辑器，Spyder 对内存的需求小很多。

下面，对 Anaconda 的环境变量进行配置。打开 Anaconda Prompt，出现类似于 cmd 的窗口，在其中输入 conda --version 命令，运行效果如图 2.18 所示。

```
(base) C:\Users\Administrator>conda --version
conda 4.4.10
```

图 2.18　Anaconda 版本

在 Anaconda Prompt 中输入命令 conda create -n env_name package_names。

其中，env_name 是设置环境的名称，package_names 是安装在环境中的包名称。

```
conda create --name test_py3 python=3.6     #创建基于 Python 3.6 的名为 test_py3 的环境
```

运行效果如图 2.19 所示。

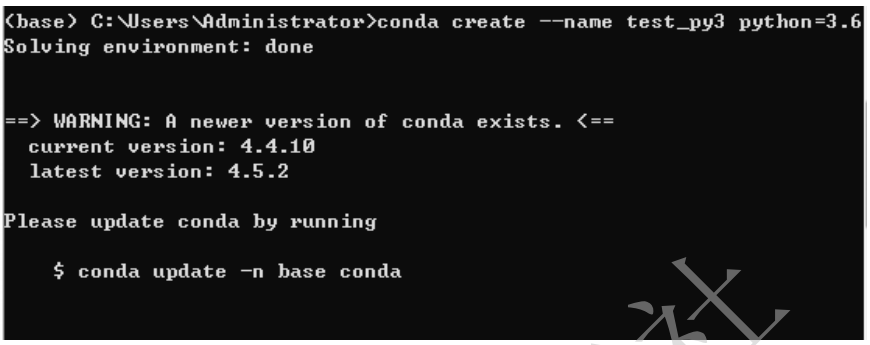

图 2.19　创建基于 Python 3.6 的名为 test_ py3 的环境

在 Anaconda Prompt 中使用 conda list 查看环境中默认安装的包，如图 2.20 所示。

图 2.20　查看环境中默认的安装包

在 Anaconda 下，Python 的编辑和执行有交互式编程、脚本式编程和 Spyder 3 种运行方式。

（1）交互式编程

交互式编程是指在编辑完一行代码后，按〈Enter〉键会立即执行并显示运行结果。在 test_py3 环境中输入 python 命令并按〈Enter〉键后，会出现 >>>，进入交互式编程模式，如图 2.21 所示。

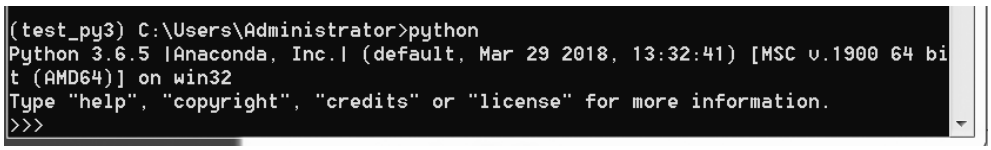

图 2.21　进入交互式编程模式

在>>>后面输入 Python 语言的各种命令。例如，输入 print('Hello world!')命令，如图 2.22 所示。

图 2.22　输入 print(' Hello world! ')命令

（2）脚本式编程

Python 和其他脚本语言（如 R 和 Perl 等）一样，可以直接在命令行里运行脚本程序。首先，在 D:\目录下创建 Hello.py 文件，内容如图 2.23 所示。

然后，进入 test_py3 环境后，输入 python d:\Hello.py 命令，运行结果如图 2.24 所示。

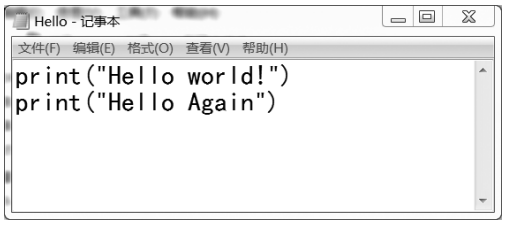

图 2.23　Hello.py 文件内容

图 2.24　运行 d:\Hello.py 文件

（3）Spyder

Spyder 是 Python 的集成开发环境，如图 2.25 所示。

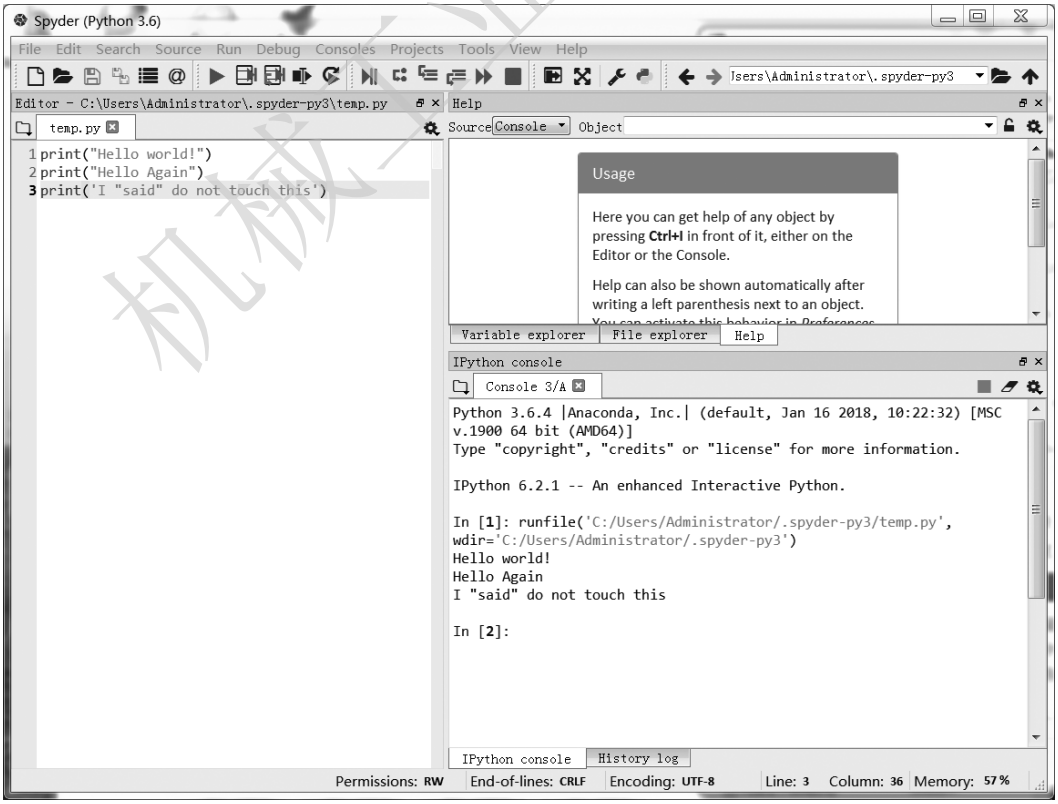

图 2.25　Spyder 编辑器

机械工业出版社

2.3.4 Jupyter Notebook

Jupyter Notebook 是 Python 的在线编辑器，在编辑的过程中，运行结果显示在代码的下方，方便查看。Jupyter Notebook 可以将代码、图像、注释、公式和可视化的结果等信息保存到文件。

在 Anaconda 中打开 Jupyter Notebook，如图 2.26 所示。

图 2.26　Jupyter Notebook 主界面

Jupyter Notebook 有编辑模式和命令模式两种模式。编辑模式用于修改单个单元格，命令模式用于操作整个笔记本，具体如下所述。

（1）编辑模式（Edit Mode）

编辑模式如图 2.27 所示，右上角有一个铅笔图标，单元左侧边框线呈现绿色，按〈Esc〉键或按〈Ctrl+Enter〉组合键运行单元格可切换回命令模式。

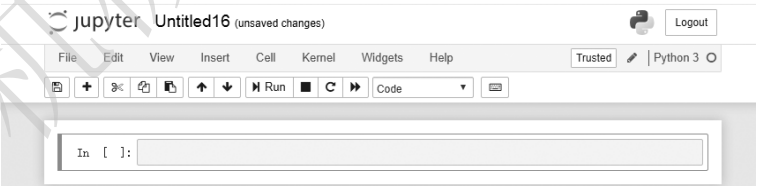

图 2.27　编辑模式

（2）命令模式（Command Mode）

命令模式如图 2.28 所示，铅笔图标消失，单元左侧边框线呈现蓝色，按〈Enter〉键或双击单元格可变为编辑状态。

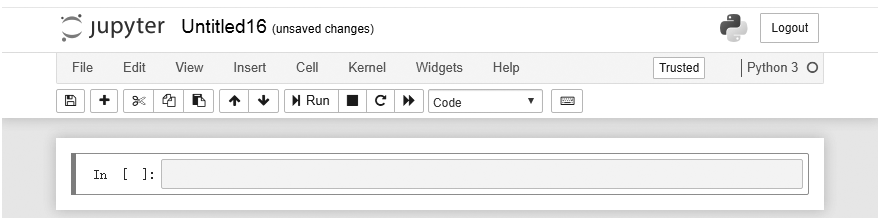

图 2.28　命令模式

机械工业出版社

编辑模式和命令模式两种模式的切换如表 2.1 所示。

表 2.1　切换笔记本模式的可选操作

模　　式	按　　键	鼠 标 操 作
编辑模式	Enter 键	在单元格内单击
命令模式	Esc 键	在单元格外单击

编辑模式下，可以使用非常标准的编辑命令来修改单元格的内容。命令模式的操作如表 2.2 所示。

表 2.2　命令模式的操作

按　　键	功　　能	按　　键	功　　能
H	显示快捷键列表	Shift+V	把单元格粘贴到当前单元格的上面
S	保存笔记本文件	D，两次	删除当前单元格
A	在当前行的上面插入一个单元格	Z	取消一次删除操作
B	在当前行的下面插入一个单元格	L	切换显示/不显示行号
X	剪切一个单元格	Y	把当前单元格切换到 IPython 模式
C	复制一个单元格	M	把当前单元格切换到 Markdown 模式
V	把单元格粘贴到当前单元格的下面	1、2、…、6	设置当前单元格为相应标题大小

2.4　代码书写规则

2.4.1　缩进

程序设计强调“清晰第一，效率第二”，应注意程序代码书写的格式。如果所有程序代码语句都从最左一列开始，则很难清楚程序语句之间的关系。因此针对判断、循环等语句可按一定的规则进行缩进，使得代码具有层次性，可读性也大为改善。

程序设计语言对于缩进有不同的要求，C 语言中的缩进对于代码的编写来说是“有了更好”，而不是“没有不行”，仅作为书写代码风格；Python 语言则将缩进作为语法要求，通过使用代码块的缩进来体现语句的逻辑关系，行首的空白称为缩进，缩进结束就表示一个代码块结束了。C 语言与 Python 语言缩进对比如图 2.29 所示。

```
if (x > y) {
    x = 1;
    y = 2;
}
```

```
if x > y:
    x = 1
    y = 2
```

图 2.29　C 语言与 Python 语言缩进对比

2.4.2　逻辑行与物理行

物理行是书写程序代码的表现形式。逻辑行是程序设计语言解释的代码形式中的单个语句。程序设计语言一方面希望物理行与逻辑行一一对应，每行只有一个语句，便于代码理

解；另一方面希望书写灵活。以 Python 语言为例，其书写规则如下所述。

1）Python 中每个语句以换行结束。

2）一个物理行中若使用多于一个逻辑行，即多条语句书写在一行，则使用分号（;），举例如下。

```
principal = 1000; rate = 0.05;numyears = 5;
```

3）当语句太长时，也可以一条语句跨多行书写，即多个物理行写一个逻辑行，使用反斜线（\）作为续行符。

【例 2-1】 反斜线（\）举例。

```
Print \
i
```

与如下写法效果相同。

```
print i
```

但是，当语句中包含[ ]、{ } 或 ( )就不需要使用多行连接符。举例如下。

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

2.4.3 注释

注释可以帮助读者思考每个过程、每个函数及每条语句的含义，便于编程员的相互讨论，有利于程序的维护和调试。一般情况下，源程序中有效注释量占总代码的 20%以上。

程序的注释分为序言性注释和功能性注释。

- 序言性注释：位于每个模块开始处，作为序言性的注解，简要描述模块的功能、主要算法、接口特点、重要数据及开发简史。
- 功能性注释：插在程序中间，与一段程序代码有关的注解，是针对一些必要的变量、核心的代码进行解释，主要解释这段代码的功能。

以 Python 为例，注释有如下一些约定。

- 注释可以添加在代码中的任意位置，但不能添加在字符串中。
- 若要将注释追加到某语句，可在该语句前后插入一个#号，后面添加注释。
- 注释还可以位于单独的行中，一般位于所要注释的代码上一行。

1. 单行注释（行注释）

Python 中#表示单行注释。单行注释可以作为单独的一行放在被注释代码行之上，也可以放在语句或表达式之后。

```
#这是单行注释
```

2. 多行注释（块注释）

当注释内容过多，一行无法显示时，可以使用多行注释。Python 中使用 3 个单引号或 3 个双引号表示多行注释。

机械工业出版社

```
'''
这是使用 3 个单引号的多行注释
'''
```

2.4.4 编码风格

良好的编码风格有助于编写出可靠、易于维护的程序，在很大程度上决定着程序的质量。下面列出了常用的编码风格。

- 复杂的表达式使用“括号”优先级理，避免二义性。
- 单个函数的代码量最好不要超过 100 行。
- 尽量使用标准库函数和公共函数。
- 不要随意定义全局变量，尽量使用局部变量。
- 保持注释与代码完全一致，修改了代码不要忘记修改注释。
- 循环、分支层次最好不要超过 5 层。
- 在编程序前，尽可能化简表达式。
- 仔细检查算法中嵌套的循环，尽可能将某些语句或表达式移到循环外面。
- 尽量避免使用多维数组。
- 避免混淆数据类型。
- 尽量采用算术表达式和布尔表达式。
- 保持控制流的局部性和直线性。控制流的局部性是为了提高程序的清晰度和易修改性，防止错误的扩散。控制流的直线性主要体现在两个方面：一是对多入口和多出口的控制结构要作适当的处理；二是结构化程序的主要特点是单入口和单出口，保持控制流的直线性使之清晰易懂。其中，高级语言为提前退出循环提供了专用语句。

2.5 习题

1. 简述 Python 的功能和特点。
2. 简述 Python 在 Linux 和 Windows 下的安装步骤。
3. Python 的开发环境有哪些？
4. Python 代码书写规则有哪些？

机械工业出版社

第 3 章　Python 数据类型

本章首先介绍了变量的命名和引用，以及各类运算符，如算术运算符、关系运算符、逻辑运算符、身份运算符等。然后介绍了 Python 的数据类型，特别是序列类型、字典和集合，其中，序列类型包括字符串、列表、元组等具有顺序编号特征的数据类型。最后介绍了数据类型转换的相关知识。

3.1　变量

变量的值可以变化，Python 的变量不需要声明，通过赋值即可创建变量。

3.1.1　变量命名

变量的命名必须遵循以下规则。

- 变量名可以由字母、数字和下画线组成。
- 变量名的第一个字符必须是字母或者下画线“_”，但不能以数字开头。
- 尽量不要使用容易混淆的单个字符作为标识符，如数字 0 和字母 o，数字 1 和字母 l 等。
- 变量名不能和关键字同名。

在 Anaconda Prompt 中输入 import keyword 查看 Python 的关键字，如图 3.1 所示。

图 3.1　Python 的关键字

- 变量名区分大小写，myname 和 myName 不是同一个变量。
- 以双下画线开头的标识符是有特殊意义的，是 Python 采用特殊方法的专用标识，如 __init__( )代表类的构造函数。

例如，a123、XYZ、变量名和 sinx 等符合变量的命名规则。

Python 中，单独的下画线（_）用于表示上一次运算的结果。

机械工业出版社

例如：

```
>>>20
20
>>>_ * 10
200
```

下面的变量命名不符合变量命名规则，导致语法错误，如图 3.2 所示。

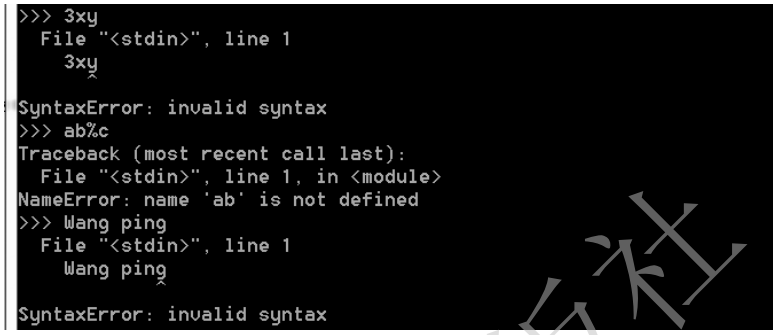

图 3.2　不符合变量命名规则导致语法错误

3.1.2　变量引用

Python 中的变量通过赋值得到值。

【例 3-1】 变量引用举例。

```
>>> number =5
>>> number
5
>>> number =7
>>> print( number )
7
```

3.2　运算符

变量之间的运算可以通过运算符实现，运算符包括算术运算符、关系运算符、赋值运算符、逻辑运算符、位运算符、成员运算符和身份运算符等。

3.2.1　算术运算符

算术运算符如表 3.1 所示。

表 3.1　算术运算符

运　算　符	含　　义	运　算　符	含　　义
+	加法	//	取整除
-	减法	**	幂运算
*	乘法	%	取模
/	除法		

机械工业出版社

运算符的使用和运算数的数据类型有很大关系，加法运行效果如图 3.3 所示。

【例 3-2】算术运算符举例。

下面给出除法（/）、整除（//）和求余数（%）的运算效果如图 3.4 所示。

```
>>> print(10+3)
13
>>> print('a'+'b')
ab
>>> print(a+b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

图 3.3　加法运算效果

```
>>> print(10/3)
3.3333333333333335
>>> print(10//3)
3
>>> print(10%3)
1
```

图 3.4　除法（/）、整除（//）和求余数（%）的运算效果

3.2.2　关系运算符

关系运算符又称比较运算符，是双目运算符，用于比较两个操作数的大小，结果是布尔值，即 True（真）或 False（假）。操作数可以是数值型或字符型。表 3.2 列出了 Python 中的关系运算符。

表 3.2　关系运算符

运　算　符	描　　述	运　算　符	描　　述
==	等于	<	小于
>	大于	<=	小于或等于
>=	大于或等于	!=	不等于

关系运算符在进行比较时，需注意以下规则。

- 两个操作数是数字，按大小进行比较。需要注意的是，Python 中的“==”表示等于，“!=”表示不等于，如图 3.5 所示。

```
>>> print(3<5)
True
>>> print(3=5)
  File "<stdin>", line 1
SyntaxError: keyword can't be an expression
>>> print(3==5)
False
>>> print(3>5)
False
>>> print(3!=5)
True
>>> print(3<>5)
  File "<stdin>", line 1
    print(3<>5)
          ^
SyntaxError: invalid syntax
>>>
```

图 3.5　操作数为数字的运行效果

- 两个操作数是字符型，按字符的 ASCII 码值从左到右逐一进行比较，即首先比较两个字符串中的第 1 个字符，ASCII 码值大的字符串为大，如果第一个字符相同，则比较第二个字符，以此类推，直到出现不同的字符为止，如图 3.6 所示。

```
>>> print("abc"=="abcd")
False
>>> print("abcd"=="abcd")
True
>>> print("abc">"abd")
False
>>> print("abc"<"abd")
True
>>> print("23"<"3")
True
>>> print("abc"!="abc")
False
>>> print("abc"!="ABC")
True
```

图 3.6　操作数为字符串的运行效果

3.2.3　赋值运算符

赋值运算符如表 3.3 所示。

表 3.3　复合赋值运算符

运　算　符	描　　述	运　算　符	描　　述
=	简单赋值运算符	/=	除法赋值运算符
+=	加法赋值运算符	%=	取模赋值运算符
-=	减法赋值运算符	**=	幂赋值运算符
*=	乘法赋值运算符	//=	取整除赋值运算符

【例 3-3】赋值运算符举例。

赋值运算符举例如图 3.7 所示。

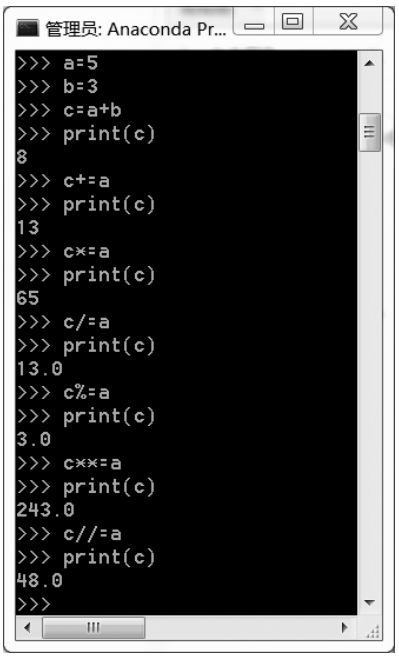

图 3.7　赋值运算符举例

3.2.4　逻辑运算符

逻辑运算符如表 3.4 所示。除 not 是单目运算符外，其余都是双目运算符，逻辑运算的结果是布尔值 True 或 False。

表 3.4　逻辑运算符

运 算 符	含　　义	描　　述
not	取反	当操作数为假时，结果为真；当操作数为真时，结果为假
and	与	当两个操作数均为真时，结果才为真；否则为假
or	或	当两个操作数至少有一个为真时，结果为真；否则为假

【例 3-4】逻辑运算符举例。

逻辑运算符举例如图 3.8 所示。

机械工业出版社

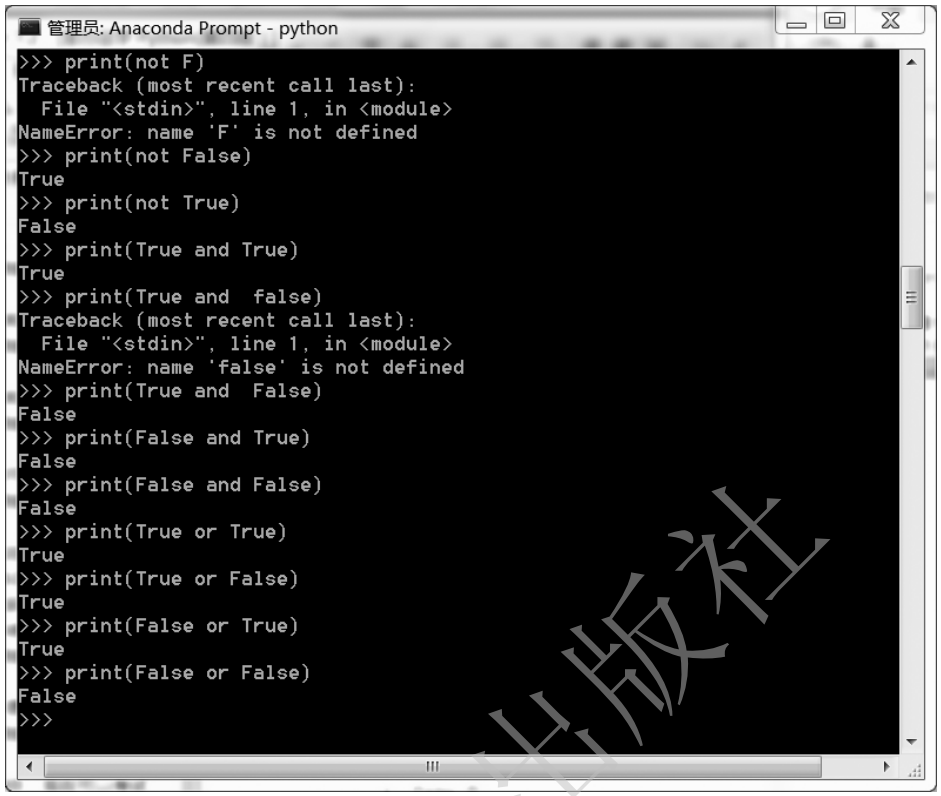

图 3.8　逻辑运算符举例

注意：False 不能简写成 F 或 false 等。

3.2.5　位运算符

位运算就把数字转换为二进制数字来运算。Python 中的位运算符有：左移（<<）、右移（>>）、按位与（&）、按位或（|）、按位异或（^）和按位翻转（~）。位运算符如表 3.5 所示。

表 3.5　位运算符

运　算　符	名　　称	描　　述
<<	左移	把一个数的二进制数字左移一定数目
>>	右移	把一个数的二进制数字右移一定数目
&	按位与	数的按位与
|	按位或	数的按位或
^	按位异或	数的按位异或
~	按位翻转	x 的按位翻转是-(x+1)

【例 3-5】位运算符举例。

位运算符举例如图 3.9 所示。

```
>>> 2<<2
8
>>> 11>>1
5
>>> 5&3
1
>>> 5|3
7
>>> 5^3
6
>>> ~5
-6
>>>
```

图 3.9　位运算符举例

机械工业出版社

3.2.6 成员运算符

成员运算符主要用于字符串、列表或元组等数据类型。如表 3.6 所示。

表 3.6 成员运算符

运 算 符	描 述
in	在指定的序列中找到值返回 True，否则返回 False
not in	在指定的序列中没有找到值返回 True，否则返回 False

【例 3-6】成员运算符举例。

```
>>> 'a' not in 'bcd'
  True
>>> 3 in [1,2,3,4]
  True
```

3.2.7 身份运算符

身份运算符又名同一运算符，用于比较两个对象的存储单元，如表 3.7 所示。

表 3.7 身份运算符

运 算 符	描 述
is	判断两个标识符是不是引用自一个对象
is not	判断两个标识符是不是引用自不同对象

【例 3-7】身份运算符举例。

```
>>> x=y=2.5
>>> z=2.5
>>> x is y
True
>>> x is z
False
>>> x is not z
True
```

3.3 表达式

3.3.1 表达式的概念

表达式通常由运算符（操作符）和参与运算的数（操作数）两部分组成。例如，2+3 就是一个表达式，+是运算符，2 和 3 是操作数。

数学表达式转换为 Python 表达式，如表 3.8 所示。

机械工业出版社

表 3.8　数学表达式转换为 Python 的表达式

数学表达式	Python 表达式
$\frac{abcd}{efg}$	a * b * c * d/e/f/g 或　　a * b * c * d/(e * f * g)
$\sin45°+\frac{e^{10}+\ln10}{\sqrt{x}}$	math. sin(45 * 3. 14/180) +(math. exp(10) +math. log(10))/math. sqrt(x)
$[(3x+y)-z]^{1/2}/(xy)^4$	math. sqrt((3 * x+y) -z)/(x * y) * * 4

数学表达式转化为 Python 表达式应注意如下区别。

- 乘号不能省略。例如，x 乘以 y 写成 Python 表达式为 x * y
- 括号必须成对出现，均使用圆括号，出现多个圆括号时，从内向外逐层配对。
- 运算符不能相邻。例如，a+ -b 是错误的。
- 添加必要的函数。例如，数学表达式 $\sqrt{25}$转换成 Python 表达式为 math. sqrt(25)等。

3.3.2　运算符的优先级

表达式计算根据运算符的优先次序逐一进行计算，Python 运算符的优先级如表 3.9 所示。

表 3.9　Python 运算符的优先级

优先级	运　算　符	描　　述
高 ↑ 低	* *	幂运算
	~、+、-	按位取反、正号、负号
	*、/、%、//	乘、除、取模和取整除
	+、-	加法、减法
	>>、<<	右移、左移
	&	按位与
	^、|	按位异或、按位或
	<=、<、>、>=	比较运算符
	= =、!=	等于、不等于运算符
	=、%=、/=、//=、-=、+=、* =、* * =	赋值运算符
	is、is not	身份运算符
	in、not in	成员运算符
	not、or、and	逻辑运算符

3.4　数据类型

3.4.1　数据类型的概念

计算机能处理数值、文本、图形、音频、视频和网页等各种数据，这些数据通过变量保

机械工业出版社

存起来。由于数据不同，所以需要不同的数据类型。例如，人的年龄为 25，用整数来表示；成绩 78.5，用浮点数来表示；人的姓名如“比尔·盖茨”，用字符串来表示等。

3.4.2 数据类型的分类

Python 3 根据数据描述信息的含义不同，分为 6 种数据类型，如下所述。

- Number（数值）。
- List（列表）。
- Tuple（元组）。
- String（字符串）。
- Dictionary（字典）。
- Set（集合）。

3.5 数值

3.5.1 数值的概念

Python 中的数值有 4 种类型：整数、布尔、浮点数和复数。

- 整数（int）。例如，1、1024 和-982。
- 布尔（bool）。例如，True、False。
- 浮点数（float）。例如，1.23、3.14 和-9.01 等。之所以称为浮点数，是因为按照科学计数法表示，浮点数的小数点位置可变。例如，52.3E4 就是科学计数法，其中，E 表示 10 的幂，52.3E4 表示 52.3×10^4。52.3E4 和 5.23E5 表示同一数字，但是它们小数点的位置不同。
- 复数（complex）。例如，1 + 2j 和 1.1 + 2.2j。

3.5.2 数值的操作

【例 3-8】数值举例。

```
>>> a, b, c, d = 20, 5.5, True, 4+3j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

数学函数如表 3.10，通过 import math 命令使用数学函数。

表 3.10 数学函数

函　数	含　义	举　例
abs(x)	数字的绝对值	math.abs(-10)返回 10
ceil(x)	数字的上入整数	math.ceil(4.1)返回 5
exp(x)	e 的 x 次幂（ex）	math.exp(1) 返回 2.718281828459045
fabs(x)	数字的绝对值	math.fabs(-10)返回 10.0
floor(x)	数字的下舍整数	math.floor(4.9)返回 4

机械工业出版社

（续）

函　数	含　义	举　例
log(x)	x 的对数	math. log(100,10)返回 2. 0
log10(x)	以 10 为基数的 x 的对数	math. log10(100)返回 2. 0
max(x1, x2,…)	给定参数的最大值，参数可以为序列	math. max(2,3)返回 3
min(x1, x2,…)	给定参数的最小值，参数可以为序列	math. min(2,3)返回 2
pow(x, y)	x^y	math. pow(2,3) 返回 8
round(x[,n])	浮点数 x 的四舍五入值，n 代表舍入到小数点后的位数	math. round(2. 4)返回 2
sqrt(x)	数字 x 的平方根	math. sqrt(4)返回 2

3.6　列表

3.6.1　列表的概念

列表（List）是 Python 中使用最频繁的数据类型。列表中的每一个数据称为元素，元素用逗号分隔并放在一对中括号“[]”中，列表可以认为是下标从零开始的数组。列表可以包含混合类型的数据，即在一个列表中的数据类型可以各不相同。

列表举例如下。

```
[10, 20, 30, 40]#所有元素都是整型数据的列表
[' frog', 'cat', 'dog']#所有元素都是字符串类型的列表
['apple', 2.0, 5, [10, 20],True] #列表中包含字符串类型、浮点类型、整型、列表类型、布尔类型
```

Python 创建列表时，解释器在内存中生成一个类似数组的数据结构，数据项自下而上存储，如图 3. 10 所示。

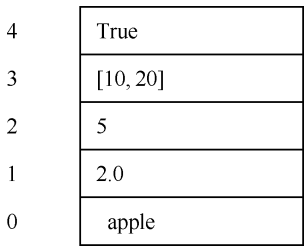

图 3. 10　列表存储方式

3.6.2　列表的操作

下面介绍列表操作。

（1）创建列表

使用“=”将一个列表赋值给变量。

```
>>> a_list = ['a', 'b', 'c']
```

（2）读取元素

1）读取某个元素：用列表名加元素序号读取某个元素。

序列中的每个元素被分配一个序号——即元素的位置，也称为索引。从左至右依次是 0,1,…,n，从右向左计数来存取元素称为负数索引，依次是-1，-2,…,-n。li[-n]==li[len(list)-n]。

【例 3-9】列表索引举例。

```
>>> l1=[1,1.3,"a"]
>>> l1[0]
1
>>> l1[-1]
'a'
```

注意：Python 从 0 开始计数。

2）读取若干元素。

序列切片（Slice）是指使用序列序号截取其中的任何部分从而得到新的序列。切片操作符是在[ ]内提供一对可选数字，用冒号分割。冒号前的数表示切片的开始位置，冒号后的数字表示切片的截止（但不包含）位置。

注意：记住数是可选的，而冒号是必须的；开始位置包含在切片中，而结束位置不包含在切片中。

【例 3-10】列表切片举例。

```
>>> l1=[1,1.3,"a"]
>>> l1[1:2]      #取出位置从 1 开始到位置为 2 的字符,但不包含偏移为 2 的元素
[1.3]
>>> l1[:2]       #不指定第一个数,切片从第一个元素,直到但不包含偏移为 2 的元素
[1, 1.3]
>>> l1[1:]       #不指定第二个数,从偏移为 1 直到末尾之间的元素
[1.3, 'a']
>>> l1[:]        #数字都不指定,则返回整个列表。
[1,1.3,'a']
```

（3）修改元素

只需直接给元素赋值。

```
>>> a_list = ['a', 'b', 'c']
>>>a_list[0] = 123
>>>print a_list
[123, 'b', 'c']
```

（4）添加元素

列表添加元素有“+”、append( )、extend( )和 insert( )方法。

方法 1：使用“+”将一个新列表附加在原列表的尾部。

```
>>> a_list = [1]
>>> a_list = a_list + ['a', 2.0]
>>> a_list
[1, 'a', 2.0]
```

方法 2：使用 append( )方法向列表尾部添加一个新元素。

```
>>> a_list = [1, 'a', 2.0]
>>> a_list.append(True)
>>> a_list
[1, 'a', 2.0, True]
```

机械工业出版社

方法 3：使用 extend( )方法将一个列表添加在原列表的尾部。

```
>>>a_list=[1, 'a', 2.0, True]
>>> a_list.extend(['x', 4])
>>> a_list
[1, 'a', 2.0, True, 'x', 4]
```

方法 4：使用 insert( )方法将一个元素插入到列表的任意位置。

```
>>>a_list=[1, 'a', 2.0, True, 'x', 4]
>>> a_list.insert(0, 'x')
>>> a_list
['x', 1, 'a', 2.0, True, 'x', 4]
```

【例 3-11】 比较“+”和 append( ) 两种方法。

```
import time
result = []
start = time.time()
for i in range(10000):
    result = result + [i]
print("+操作执行",len(result), '次,用时', time.time()-start)
result = []
start = time.time()
for i in range(10000):
    result.append(i)
print("append 操作执行",len(result), '次,用时', time.time()-start)
```

【程序运行结果】

```
+操作执行 10000 次,用时 0.2020115852355957
append 操作执行 10000 次,用时 0.0009999275207519531
```

从程序结果可知，append( )用时较少，明显快于“+”。

【例 3-12】 比较 insert( )和 append( )两种方法。

```
import time
def Insert():
    a = []
    for i in range(10000):
        a.insert(0, i)
def Append():
    a = []
    for i in range(10000):
        a.append(i)
start = time.time()
for i in range(10):
    Insert()
print('Insert:', time.time()-start)
start = time.time()
for i in range(10):
    Append()
print('Append:', time.time()-start)
```

机械工业出版社

程序运行结果如下所示。

```
Insert：0.578000068665
Append：0.0309998989105
```

从程序结果可知，还是 append( )用时较少，明显快于 insert( )。

（5）删除元素

列表删除元素有 del、remove( )和 pop（参数）方法。

方法 1：使用 del 语句删除某个特定位置的元素。

```
>>>a_list=['x', 1, 'a', 2.0, True, 'x', 4]
>>> del a_list[1]
>>> a_list
['x', 'a', 2.0, True, 'x', 4]
```

方法 2：使用 remove( )方法删除某个特定值的元素。

```
>>>a_list = ['x', 'a', 2.0, True, 'x', 4]
>>> a_list.remove('x')
>>> a_list
['a', 2.0, True, 'x', 4]
>>> a_list.remove('x')
>>> a_list
['a', 2.0, True, 4]
>>> a_list.remove('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

【例 3-13】 比较两组代码。

```
>>> x = [1,2,1,2,1,2,1,2,1]
>>> for i in x:
    if i == 1:
      x.remove(i)
>>> x
[2, 2, 2, 2]
```

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in x:
        if i == 1:
            x.remove(i)
>>> x
[2, 2, 1]
```

【解析】 同样的代码，仅仅是所处理的数据发生了一点变化，却导致结果不同。两组数据的区别在于数据中是否有连续的“1”。由于列表的自动内存管理功能，在插入或删除列表元素时，Python 会自动对列表内存进行扩充或收缩并移动列表元素以保证所有元素之间没有空位置，因此，每当插入或删除一个元素后，该元素位置后面所有元素的索引都会改变。

remove( )和 del 都用于删除元素，修改后的代码如下所示。

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in x[::]:     #切片
        if i == 1:
    x.remove(i)
```

```
>>> x = [1,2,1,2,1,1,1]
>>> for i in range(len(x)-1,-1,-1):
        if x[i]==1:
            del x[i]
```

机械工业出版社

方法 3：使用 pop（参数）方法弹出指定位置的元素，默认参数时弹出最后一个元素。

```
>>> a_list=['a', 2.0, True, 4]
>>> a_list.pop( )       #默认参数时弹出最后一个元素
4
>>> a_list
['a', 2.0, True]
>>> a_list.pop(1)
2.0
>>> a_list
['a', True]
>>> a_list.pop(1)
True
>>> a_list
['a']
>>> a_list.pop( )
'a'
>>> a_list
[ ]
>>> a_list.pop( )
Traceback (most recent call last):
  File "< stdin >", line 1, in <module>
IndexError: pop from empty list
```

（6）获取列表中指定元素的下标

```
>>>a=[72, 56, 76, 84, 76,80, 88]
>>>print(a.index(56))   #输出元素 56 的下标
1
>>>b= list(enumerate(a))                    # enumerate 将 list 的元素元组化
>>>print(b)
[(0, 72), (1, 56), (2, 76), (3, 84), (4, 76), (5, 80), (6, 88)]
>>>print("输出元素 76 的下标")
>>>print([i for i,x in b if x==76])      #利用循环方法获取相应的匹配结果
[2, 4]
```

列表方法如表 3.11 所示。

表 3.11　列表方法

函　　数	描　　述
alist.append(obj)	列表末尾增加元素 obj
alist.count(obj)	统计元素 obj 出现次数
alist.extend(sequence)	用 sequence 扩展列表
alist.index(obj)	返回列表中元素 obj 的索引
alist.insert(index,obj)	在 index 索引之前添加元素 obj
alist.pop(index)	删除索引的元素
alist.remove(obj)	删除指定元素

3.7 元组

3.7.1 元组的概念

元组（Tuple）和列表类似，相当于只读列表，其元素不可以修改。元组适合于只需进行遍历操作的运算，对数据进行“写保护”，其操作速度比列表快。

元组与列表相比，有如下不同。

- 元组的所有元素放在一对圆括号“()”中。
- 不能向元组增加元素，元组没有 append()、insert()或 extend()方法。
- 不能从元组删除元素，元组没有 remove()或 pop()方法。
- 元组没有 index()方法，但是，可以使用 in 操作符。
- 元组可以在字典中被用作“键”，而列表不能被用作“键”。

3.7.2 元组的操作

下面介绍元组操作。

（1）创建元组

使用赋值运算符“=”将一个元组赋值给变量，即可创建元组对象。

```
>>>tup1 = ('a', 'b', 1997, 2000)
>>>tup2 = (1, 2, 3, 4, 5, 6, 7 )
```

当创建只包含一个元素的元组时，需要注意它的特殊性。此时，只把元素放在圆括号中是不行的，这是因为圆括号既可以表示元组，又可以表示数学公式中的小括号，从而会产生歧义。因此，Python 规定当创建只包含一个元素的元组时，需在元素的后面加一个逗号“,”。

```
>>> x=(1)
>>> x
1
>>> y=(1,)
>>> y
(1,)
>>> z=(1,2)
>>> z
(1, 2)
```

（2）访问元组

可以使用下标索引来访问元组中的值。

```
>>>tup1 = ('a', 'b', 1997, 2000)
>>>tup2 = (1, 2, 3, 4, 5, 6, 7 )
>>>print("tup1[0]:", tup1[0] )
tup1[0]:  a
```

机械工业出版社

```
>>>print("tup2[1:5]:", tup2[1:5])
tup2[1:5]:(2, 3, 4, 5)
```

（3）元组连接

元组可以进行连接操作。

```
>>>tup1 = (12, 34.56)
>>>tup2 = ('abc', 'xyz')
# tup1[0] = 100                    #元组元素不可以修改
>>>tup3 = tup1 + tup2;             # 创建一个新的元组
>>>print(tup3)
(12, 34.56, 'abc', 'xyz')
```

（4）删除元组

元组中的元素值是不允许删除的，但可以使用 del 语句删除整个元组。

```
>>>tup = ('physics', 'chemistry', 1997, 2000)
>>>del tup[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' objext doesn't support item  deletion
>>>del tup
>>>print(tup)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tup' is not defined
```

3.8 字符串

3.8.1 字符串的概念

字符串在 Python 中是以单引号、双引号或三引号括起来的符号来表示，如'Hello World'、"Python is groovy"和'''What is footnote 5? '''等。请注意，''或""本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有 a、b 和 c 这 3 个字符。用单引号或双引号括起来没有任何区别，但一个字符串用什么引号开头，就必须用什么引号结尾。

单引号与双引号只能创建单行字符串。

```
>>>'Hello'
'Hello'
>>>'Let's go'
'Let's go'
>>>s="'Python' Program"
>>>s
"'Python' Program"
```

机械工业出版社

为了创建多行字符串或者为了使字符串的数据中出现双引号，可以使用三引号。

```
>>> s="'
… We say "Hello" to Python
… '''
>>>s
'\nWe say "Hello" to Python'
```

3.8.2 字符串的操作

字符串（String）、列表和元组都是序列。字符串的方法如表 3.12 所示。

表 3.12 字符串方法

函　数	描　述
s.index（sub ,[start, end]）	定位子串 sub 在 s 中第一次出现的位置
s.find(sub ,[start ,end]})	与 index 函数一样，但如果在 s 中找不到 sub 会返回-1
s.replace(old, new [,count])	将 s 中所有 old 子串替换为 new 子串，count 指定可被替换多少个
s.count(sub[,start,end])	统计 s 中有多少个 sub 子串
s.split( )	拆分字符串，通过指定分隔符对字符串进行切片，并返回分隔后的字符串列表(list)，默认分隔符是空格' '
s.join( )	join( )方法是 split( )方法的逆方法，用来把字符串连接起来
s.lower( )	返回将大写字母变成小写字母的字符串
s.upper( )	返回将小写字母变成大写字母的字符串
sep.join(sequence)	把 sequence 的元素用连接符 sep 连接起来

下面介绍字符串的操作。

（1）index 举例

```
>>> s="Python"
>>> s.index('P')
0
>>> s.index('h',1,4)
3
>>> s.index('y',3,4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s.index('h',3,4)
3
```

（2）find 举例

```
>>> s="Python"
>>> s.find('s')
-1
>>> s.find('t',1)
2
```

机械工业出版社

（3）replace 举例

```
>>> s="Python"
>>> s.replace('h','i')
'Pytion'
```

（4）count 举例

```
>>> s="Python"
>>> s.count('n')
1
```

（5）split 举例

```
>>> s="Python"
>>> s.split()
['Python']
>>> s="hello Python i like it"
>>> s.split()
['hello', 'Python', 'i', 'like', 'it']
>>>s = 'name:zhou,age:20|name:python,age:30|name:wang,age:55'
>>>print(s.split('|'))
['name:zhou,age:20',' name:python,age:30',' name:wang,age:55']
>>>x,y= s.split('|',1)
>>>print(x)
name:zhou,age:20
>>>print(y)
name:python,age:30|name:wang,age:55
```

（6）join 举例

```
>>> li=['apple','peach','banana','pear']
>>> sep=','
>>> s=sep.join(li)                    #连接列表元素
>>> s
'apple,peach,banana,pear'
>>>s5=("Hello","World")
>>>sep=""
>>> sep.join(s5)                      #连接元组元素
'HelloWorld'
```

3.9 字典

3.9.1 字典的概念

【例 3-14】根据同学的名字查找对应的成绩。

【解析】采用列表实现，则需要 names 和 scores 两个列表，并且列表中元素的次序一一对应，如 zhou 对应 95，Bob 对应 75 等，如下所示。

机械工业出版社

```
names = ['zhou', 'Bob', 'Tracy']
scores = [95, 75, 85]
```

通过名字查找对应成绩，先在 names 中遍历查找的名字，再从 scores 中遍历对应的成绩，查找时间会随着列表的长度的增加而增加。为了解决这个问题，Python 提供了字典。

字典（Dict）在其他程序设计语言中称为映射（Map），通过键值对（Key-Value）存储数据，键和值之间用冒号间隔，元素项之间用逗号间隔，整体用一对大括号“{}”括起来。字典语法结构如下所示。

```
dict_name={key:value,key:value}
```

字典有如下特性。

- 字典的值可以是任意数据类型，包括字符串、整数、对象，甚至字典等。
- 键值对没有顺序。
- 键必须是唯一的，不允许同一个键重复出现，如果同一个键被赋值两次，后一个值会覆盖前面的值。举例如下。

```
>>>dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Zhou'}
dict['Name']:  Zhou
```

- 键是不可变的，只能使用整数、字符串或元组充当，不能使用列表。

```
>>>dict={['Name']:'Zhou','Age':7}
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    dict={['Name']:'Zhou','Age':7}
TypeError: unhashable type: 'list'
```

【解析】因为 dict 根据 Key 来计算 Value 的存储位置，在 Python 中，字符串、整数等都是不可变的，而 list 是可变的，因此，list 不能作为 Key。

字典与列表相比，有以下几个特点。

- 字典用空间来换取时间，其查找和插入的速度极快。
- 字典需要占用大量的内存，内存浪费较多。
- 字典是无序的对象集合，字典中的元素是通过键来存取的，而不是通过偏移存取。

采用字典实现上面的例子，则只需创建“名字”-“成绩”的键值对，便可直接通过名字查找成绩。字典实现代码如下。

```
>>> d = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>> d['zhou']
95
```

3.9.2 字典的操作

下面介绍字典元素的访问、删除、修改和增加等相关操作。

机械工业出版社

（1）字典元素的访问

1）keys()方法返回一个包含所有键的列表。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.keys()
['Bob', 'Tracy', 'zhou']
```

2）has_key()方法检查字典中是否存在某一键。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.has_key('zhou')
True
```

3）values()方法返回一个包含所有值的列表。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.values()
[75, 85, 95]
```

4）get()方法根据键返回值，如果不存在则返回 None。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.get('Bob')
75
```

5）items()方法返回一个由（key,value）组成的元组。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.items()
[('Bob', 75), ('Tracy', 85), ('zhou', 95)]
```

6）in 运算用于判断某键是否在字典中，对于 value 值不适用。

```
>>> tel1 = {'gree':5127, 'pang':6008}
>>> 'gree'  in tel1
True
```

7）copy()方法复制字典。

```
>>> stu1 = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>> stu2 = stu1.copy()
>>> print(stu2)
{'zhou': 95, 'Bob': 75, 'Tracy': 85}
```

（2）字典元素的删除

1）del()方法允许使用键从字典中删除元素。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>> deldict['zhou']
>>> print(dict)
{'Bob': 75, 'Tracy': 85}
```

2）clear()方法清除字典中的所有元素。

机械工业出版社

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.clear()
>>>dict
{}
```

3）pop()方法删除一个关键字并返回它的值。

```
>>>dict = {'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.pop('zhou')
95
>>> print(dict)
{'Bob': 75, 'Tracy': 85}
```

（3）字典元素的修改

update()方法类似于合并，它把一个字典的键和值合并到另一个字典以覆盖相同键的值。

```
>>> tel={'gree': 4127, 'mark': 4127, 'jack': 4098}
>>> tel1 = {'gree':5127, 'pang':6008}
>>> tel.update(tel1)
>>> tel
{'gree': 5127, 'pang': 6008, 'jack': 4098, 'mark': 4127}
```

（4）字典元素的增加

```
>>> stu = {'1': 95, '2': 75, '3': 85}
>>>stu['4']=99
>>> print(stu)
{'1': 95, '2': 75, '3': 85,'4':99}
```

字典方法如表 3.13 所示。

表 3.13　字典方法

函　数	描　述
aDic.clear()	删除字典所有元素
aDic.copy()	返回字典副本
aDic.get(key)	返回字典的 key
aDic.has_key(key)	检查字典中是否有给定的键
aDic.items()	返回可遍历的（键，值）元组数组的列表
aDic.keys()	返回字典键的列表
aDic.pop(key)	删除并返回给定键的值
aDic.values()	返回字典值的列表

机械工业出版社

3.10 集合

3.10.1 集合的概念

集合（Set）是一个无序、不重复元素集，基本功能包括关系测试和消除重复元素。集合有如下一些方法，如表 3.14 所示。

表 3.14 集合的方法

函　数	描　述
set. add（x）	将数据项 x 添加到集合 s 中
set. pop( )	随机移除一个元素
set. remove（x）	从集合 s 中删除数据项 x
set. clear( )	清除集合 s 中的所有数据项
set. copy( )	复制 s 中的数据项
set. count(sub[,start,end])	统计 s 里有多少个 sub 子串
set. split( )	默认分隔符是空格 ' '，如果没有分隔符，就把整个字符串作为列表的一个元素
set. join( )	join( )方法是 split( )方法的逆方法，用来把字符串连接起来
set. lower( )	返回将大写字母变成小写字母的字符串
set. upper( )	返回将小写字母变成大写字母的字符串

3.10.2 集合的操作

下面介绍集合的相关操作。

（1）创建集合

```
>>> s=set([1,2,3])
>>>s
{1, 2, 3}
```

重复的元素在 set 中被自动过滤，如下所示。

```
>>> s=set([1,3,2,2,2,3])
>>>s
{1, 2, 3}
```

（2）访问集合

集合本身无序，无法进行索引和切片操作，只能使用 in、not in 或者循环遍历来访问或判断集合元素。

```
>>>a_set = set(['python',2018])
>>>a_set
{2018, 'python'}
>>> 2018 ina_set
```

机械工业出版社

```
True
>>>for i in a_set:
    print(i,end='')

2018python
```

（3）删除集合

使用 del 语句删除集合。举例如下。

```
>>>a_set = set(['python',2018])
>>>del a_set
>>>a_set
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    a_set
NameError: name 'a_set' is not defined
```

（4）向集合中添加元素

使用 add 语句添加元素。举例如下。

```
>>>a_set = set(['python',2018])
>>>a_set.add(29.5)
>>>a_set
{2018, 'python', 29.5}
```

（5）从集合中删除元素

从集合中删除元素有 remove()、pop()、clear()等方法。

1）remove()方法。

```
>>>a_set = set(['python',2018])
>>>a_set.remove(2018)
>>>a_set
{'python'}
```

2）pop()方法。

```
>>>a_set = set(['python',2018])
>>>a_set.pop()
2018
>>>a_set
{'python'}
```

3）clear()方法。

```
>>>a_set = set(['python',2018])
>>>a_set.clear()
>>>a_set
set()
```

机械工业出版社

3.10.3 集合运算

Python 提供方法实现交、并、差集合运算。

1）差集：“ - ”用于求出两个集合的差集。

```
>>> a=set([1,2,3])
>>> b=set([2,3,4])
>>> a-b
{1}
```

2）并集：“|”用于求出两个集合的并集。

```
>>> a|b
{1, 2, 3, 4}
```

3）交集：“&”用于求出两个集合的交集。

```
>>> a&b
{2, 3}
```

4）对称差集：“^”用于求出两个集合中不同时存在的元素。

```
>>> a^b
{1, 4}
```

【例 3-15】每一个列表中只要有一个元素出现两次，那么该列表即被判定为包含重复元素。编写函数判定列表中是否包含重复元素，如果包含重复元素，返回 True，否则返回 False。然后使用该函数对 n 行字符串进行处理。最后分别统计包含重复元素与不包含重复元素的行数。

输入格式如下。

输入 n，代表接下来要输入 n 行字符串。

然后输入 n 行字符串，字符串之间的元素以空格相分隔。

输出格式如下。

True=包含重复元素的行数，False=不包含重复元素的行数。

输入样例如下。

```
5
1 2 3 4 5
1 3 2 5 4
1 2 3 6 1
1 2 3 2 1
1 1 1 1 1
```

输出样例如下。

```
True=3, False=2
```

【代码】

机械工业出版社

```
n = int(input())
true = false = 0
for i in range(n):
    a = input()
    a = list(a.split())
    if len(list(a)) == len(set(a)):          #利用集合中元素不能重复的特性
        false +=1
    else:
        true +=1
print('True=%d, False=%d'%(true,false))
```

3.11 组合数据总结

3.11.1 相互关系

列表（list）、元组（tuple）、字符串（string）、字典（dictionary）、集合（sets）之间的相互关系如图 3.11 所示。

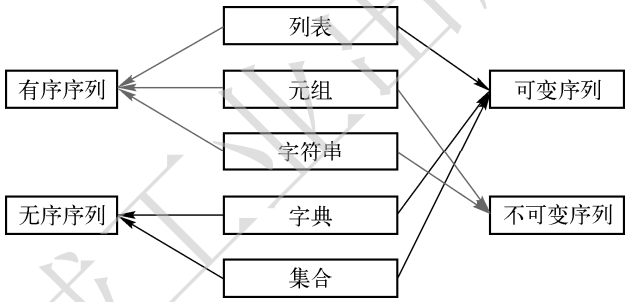

图 3.11 组合数据类型之间相互关系

3.11.2 数据类型转换

以下几个内置函数用于组合数据类型之间的转换，如表 3.15 所示。

表 3.15 数据类型转换

函　数	描　述	举　例
eval(x)	将字符串 x 求值返回结果	>>>eval ("12") 12
tuple(s)	将序列 s 转换为一个元组	>>>tuple([1,2,3]) (1,2,3)
list(s)	将序列 s 转换为一个列表	>>> list((1,2,3)) [1,2,3]
set(s)	转换为可变集合	>>>set([1,4,2,4,3,5]) {1,2,3,4,5} >>> set({1:'a',2:'b',3:'c'}) {1,2,3}
dict(d)	创建一个字典（key,value）	>>>dict([('a', 1), ('b', 2), ('c', 3)]) {'a':1, 'b':2, 'c':3}

机械工业出版社

3.12 实例

3.12.1 发扑克牌

【例 3-16】发扑克牌，输出扑克牌花色和数值的随机组合。

【解析】扑克牌有 54 张牌，其中 52 张是正牌，2 张是副牌（大王和小王）。52 张正牌又根据花色分为黑桃、红桃、梅花和方块 4 组，每组花色的牌包括 1~10（1 通常表示为 A）及 J、Q、K 各 13 张牌。

【代码】

```
import random
SUITS=['Club','Diamond','Heart','Spade']
RANKS=['A','2','3','4','5','6','7','8','9','10','J','Q ','K']
deck=[ ]
for rank in RANKS:
    for suit in SUITS:
        card = rank + 'of' + suit
        deck += [card]
n=len(deck)
for i in range(n):
    r=random. randrange(i,n)
    deck[r],deck[i]=deck[i],deck[r]
for s in deck:print(s)                          #输出扑克牌的随机组合
```

【程序运行结果】

```
3ofHeart
AofHeart
3ofClub
4ofClub
JofClub
3ofDiamond
4ofSpade
2ofClub
KofSpade
7ofHeart
6ofSpade
9ofSpade
JofSpade
9ofHeart
AofSpade
6ofDiamond
AofDiamond
QofSpade
……
```

机械工业出版社

3.12.2 统计相同单词出现的次数

【例 3-17】输入一串字符，统计其中相同单词出现的次数，单词之间用空格分隔开。

【解析】采用字典来实现，将单词作为键，将单词的次数作为值。

【代码】

```
string=input("input string:")
string_list=string.split()
word_dict={}
for word in string_list:
    if word in word_dict:
        word_dict[word] += 1
    else:
        word_dict[word] = 1
print(word_dict)
```

【程序运行结果】

```
input string:I am a boy I am a student
{'I': 2, 'am': 2, 'a': 2, 'boy': 1, 'student': 1}
```

3.12.3 计算两个日期间隔天数

【例 3-18】输入两个同年的日期，计算它们相隔的天数。其中，默认第二个输入日期比第一个输入日期晚。例如，输入 2018-3-1 与 2018-5-25，输出间隔天数 86。

【解析】输入的字符串转换为整数需要用到 split()函数及 map()函数。"2018-3-1".split("-")的含义是将字符串按“-”切分，返回["2018","3","1"]列表。map(int,list)的含义是将 int 函数依次应用于 list 中的每一个元素。闰年是指能被 4 整除但不能被 100 整除或能被 400 整除的年份。

【代码】

```
y1,m1,d1=map(int,input('开始年月日,输入形式如 2018-2-3:').split('-'))
y2,m2,d2=map(int,input('结束年月日,输入形式如 2018-2-3:').split('-'))
a=[31,28,31,30,31,30,31,31,30,31,30,31]
d=0
if m1>m2:
    m1,m2=m2,m1
    d1,d2=d2,d1
for i in range(m1,m2):
    d+=a[i]
d=d-d1+d2
if (m1<=2) and (y1%400==0)or(y1%4==0 and not( y1%100==0)):
    d+=1
print('间隔天数:',d+1)
```

机械工业出版社

【程序运行结果】

```
开始年月日：2018-3-1
结束年月日：2018-5-25
间隔天数： 86
```

3.13 习题

1. 在列表中输入多个数据作为圆的半径，求出相应的圆的面积。

2. 输入一段英文文章，求其长度，并求出包含多少个单词。

3. 随意输入 10 个学生的姓名和成绩构成的字典，按照成绩高低排序。

4. 任意输入一串字符，输出其中不同的字符及各自的个数。例如，输入“abcdefgabc”，输出为 a→2,b→2,c→2,d→1,e→1,f→1,g→1。

5. 设计一个字典，用户输入内容作为键，查找输出字典中对应的值，如果用户输入的键不存在，则输出“该键不存在!”。

6. 已知列表 a_list=[11,22,33,44,55,66,77,88,99]，将所有大于 60 的值保存至字典的第 1 个 key 的值中，将所有小于 60 的值保存至字典的第 2 个 key 的值中，即｛k1：大于 60 的所有值，k2：小于 60 的所有值｝。

7. 给定一个字符串和一个列表，返回该字符串在该列表里面第二次出现的位置的下标，若没有出现第二次则返回-1。

机械工业出版社