Python科学计算基础

罗奇鸣 编著

2022年4月4日

本书是中国科学技术大学“Python科学计算基础”课程的教材，目前正在编写和修改过程中。

目录

第一章　概述

1.1　利用计算机解决问题

计算机是能够按照程序自动对数据进行计算的电子设备。计算机的优势是运算速度快。以下举例说明。

1. 1946年诞生的世界上第一台通用计算机ENIAC每秒能进行5000次加法运算(据测算人最快的运算速度仅为每秒5次加法运算)和400次乘法运算。人工计算一条弹道需要20多分钟时间，ENIAC仅需30秒!

2. 2018年投入使用的派-曙光是首台应用中国国产卫星数据，运行我国自主研发的数值天气预报系统(GRAPES)的高性能计算机系统。该系统峰值运算速度达到每秒8189.5万亿次，内存总容量达到690432GB。近年来，我国台风路径预报24小时误差稳定在70公里左右，各时效预报全面超过美国和日本，达国际领先水平。同样，降水、雷电、雾-霾、沙尘等预报预测准确率也整体得到提升。

为了利用计算机解决问题，必须使用某种程序设计语言把解决问题的详细过程编写为程序，即一组计算机能识别和执行的指令。

1.2　程序设计语言的分类

程序设计语言可分为三类：机器语言、汇编语言和高级语言。早期的计算机只能理解机器语言。机器语言用0和1组成的二进制码串表示CPU指令和数据。之后出现的汇编语言用易于理解和记忆的符号(表示指令、数据、寄存器、地址等概念)来代替二进制码，克服了机器语言难以理解的缺点。机器语言和汇编语言的共同缺点是依赖于硬件处理器，用它们编写的程序无法移植到不同的处理器上。1956年投入使用的的Fortran语言是第一种高级语言。高级语言采用接近自然语言和数学公式的方式表达解决问题的过程，不再依赖于硬件处理器，实现了可移植性。

近几十年来，随着计算机软硬件技术的飞速发展，高级语言不断涌现，数量达到几百种。高级语言按照编程范式(programming paradigm)可划分为以下几个类别:

1. 命令式(imperative): 使用命令的序列修改状态，例如C、C++、Java、Python。

2. 声明式(declarative): 指明求解的结果，而不说明求解的过程，例如SQL。

3. 过程式(procedural): 可进行过程调用的命令式，例如C、C++、Java、Python。

4. 函数式(functional): 不修改状态的函数互相调用，例如Lisp、 ML、 Haskell、 Scala。

5. 逻辑式(logic): 基于已知事实和规则推断结果，例如Prolog。

6. 面向对象(object-oriented): 有内部状态和公开接口的对象互相发送消息，例如Simula 67、C++、Java、Python。

这些编程范式并非互斥，一种语言可同时支持多种范式。

1.3 过程式编程范式

过程式是一种重要的编程范式，它的特点是基于输入和输出定义解决问题的过程，通过自顶向下的功能分解将过程划分为若干个基本模块，形成一个树状结构。程序由多个模块构成，每一模块内部均是由顺序、选择和循环三种基本结构组成。

过程式的长处是可以有效地将一个较复杂的问题逐步分解成多个易于解决的子问题，最终解决原问题。过程式的是缺点是需求发生变化时不易维护，也不易实现代码复用，因此不适于开发大型软件。

以下通过一个实例说明过程式编程范式：根据年和月输出日历。程序的运行结果见图1.1。 采用

```
Enter full year (e.g., 2001): 2019
Enter month in number (1 ... 12): 2

 February 2019
Sun Mon Tue Wed Thu Fri Sat
                        1   2
  3   4   5   6   7   8   9
 10  11  12  13  14  15  16
 17  18  19  20  21  22  23
 24  25  26  27  28
```

图 1.1: 根据年和月输出日历

过程式编程范式进行问题分解的步骤如下：

1. 读取用户输入的年和月

2. 输出日历的标题

3. 输出日历的主体

   (a) 怎样确定指定的某年某月有多少天？

i. 如果是2月，怎样确定指定年是否是闰年？

(b) 怎样确定这个月的第一天是星期几？已知有公式可以计算指定的某年y的一月一日是星期几。公式为：

$$n = (y + \lfloor (y-1)/4 \rfloor - \lfloor (y-1)/100 \rfloor + \lfloor (y-1)/400 \rfloor) \% 7 \tag{1.1}$$

其中$n = 0, 1, ..., 6$依次表示周日、周一...周六，$\lfloor x \rfloor$表示不大于实数x的最大整数。再计算从一月一日到这个月的第一天所经历的总天数即可解决这个问题。

i. 怎样确定任意指定的某年某月有多少天？已由3(a)解决。

1.4 面向对象编程范式

面向对象编程范式的特点如下：

1. 将客观事物直接映射到软件系统的对象。对象是将数据及处理数据的过程封装在一起得到的整体，用以表示客观事物的状态和行为。

2. 对同一类型对象抽象出其共同的属性和操作，形成类。类是创建对象的模板，对象是类的实例。

3. 每个类作为一个独立单元进行开发、测试和维护。如果需要修改类的实现细节，只要不改变类的接口就不会影响使用该类的外部代码，易于维护。

4. 通过继承可以重复利用已有的类的代码，并根据需要扩展。

5. 程序由多个类构成。程序在运行时由各个类创建一些对象，对象之间通过明确定义的接口进行交互，完成软件系统的功能

面向对象编程范式的优点是易于开发和维护大型软件，缺点是与过程式程序设计方法相比在性能上相对较弱。

1.5 学习Python语言的理由

TIOBE指数由荷兰TIOBE公司自2001年开始每月定期发布，用于评估程序设计语言的流行度。近几年Python语言的流行度快速攀升，目前已跃居榜首(图1.2)。

Google公司的决策是"Python where we can, C++ where we must." 即仅在性能要求高和需要精细管理内存的场合使用C++，而在其他场合都使用Python。原因是用Python语言开发软件效率更高，且易于维护和复用。

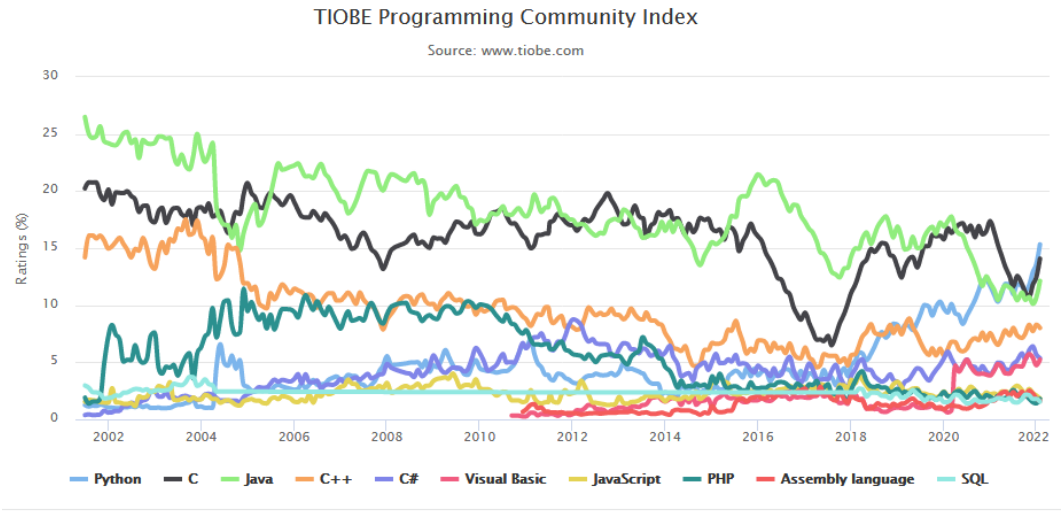

图 1.2: TIOBE指数2022年2月

1.6 Python语言的发展历史

Python由荷兰程序员Guido van Rossum于1989年基于ABC教学语言设计和开发，其命名是源于BBC的喜剧节目“Monty Python’s Flying Circus”。Guido发现像他这样熟练掌握C语言的人，在用C写功能时也不得不耗费大量的时间。Bourne Shell作为UNIX系统的解释器已经长期存在，它可以像胶水一样将UNIX 下的许多功能连接在一起。许多C语言下上百行的程序，在shell下只用几行就可以完成。然而，shell的本质是调用命令，并不是一种通用语言。所以 Guido 希望有一种语言可以兼具 C 和 shell 的优点。Guido总结的设计目标是

1. 一门简单直观的语言并与主要竞争者一样强大；

2. 代码像纯英语那样容易理解；

3. 适用于短期开发的日常任务；

4. 开源，以便任何人都可以为它做贡献；

5. 代码像纯英语那样容易理解。

Google公司的决策是”Python where we can, C++ where we must.” 即仅在性能要求高和需要紧密控制内存的场合使用需要C++，而在其他场合都使用Python。原因是用Python语言开发软件效率更高，且易于维护。 Python软件基金会 (Python Software Foundation, https://www.python.org/psf/)是Python的版权持有者，致力于推动Python开源技术和发布Python的新版本。2008年12月，Python3.0发布，这是一次重大的升级，与Python2.x不兼容。 2019年10月，Python3.8发布。2020年10月，Python3.9发布。

1.7 Python语言的特点

Python是一种简单易学、动态类型、功能强大、面向对象、解释执行、易于扩展的开源通用型语言。Python的主要特点如下。

1. 语法简单清晰，代码精炼。
2. 使用变量之前无需声明其类型，编写程序更方便。变量的类型由运行时系统推断。
3. 标准库提供了数据结构、系统管理、网络通信、文本处理、数据库接口、图形系统、XML处理等丰富的功能。
4. Python社区提供了大量的第三方模块，使用方式与标准库类似。它们的功能覆盖科学计算、人工智能、机器学习、Web开发、数据库接口、图形系统多个领域。
5. 面向对象, 支持继承、重载和泛型，适于大规模软件开发。
6. 解释器提供了一个交互式的开发环境，程序无需编译和链接即可执行。
7. 如果需要一段关键代码运行得更快或者不希望公开某些算法，可以把这部分程序用C或C++编写，然后在Python程序中使用它们。
8. 与C等编译执行的语言相比，Python程序的性能较弱。

1.8 Python科学计算环境

数值计算(也称为科学计算)已成为当今科学研究的三种基本手段之一。它是计算数学、计算机科学和其他工程学科相结合的产物，并随着计算机的普及和各门类科学技术的迅速发展日益受到人们的重视。数值计算的特点是处理连续数学的量(实数量)，问题中还可能涉及微分、积分和非线性。被求解的问题一般没有解析解、或理论上无法通过有限步计算求解。数值计算的目标是寻找迅速完成的(迭代)算法，评估结果的准确度。好数值算法的特点是计算效率高、计算复杂度低和可靠性好。

科学计算使用的程序设计语言主要包括Fortran、C、C++、MATLAB和Python。前三种语言称为低层语言，后两种称为高层语言。用低层语言开发的程序比用高层语言开发的程序执行效率更高，但开发耗时更长。由于人力成本不断上升而硬件成本不断下降，当前趋势是用高层语言开发程序，并通过接口访问用低层语言开发的软件包。 MATLAB是一款价格不菲的商用软件，Python完全免费开源。

Python语言及其众多的扩展库(例如NumPy、SciPy和matplotlib)所构成的开发环境十分适合开发科学计算应用程序。 NumPy提供了N维数组类型以及数组常用运算的高效实现。SciPy基于NumPy实现提供了大量的数值计算算法，包括线性代数、插值、微分、积分、优化、回归、傅里叶变换、常微分方程求解、信号和图像处理等。matplotlib提供了丰富的绘图和数据可视化功能。

1.9 安装和使用Python开发环境

Anaconda是一个开源的Python发行版本，包含Python解释器、包管理器conda和多个科学计算包（numpy、scipy等），可运行在 Windows、Linux和Mac OS系统上。下载和安装。运行Python代码的方式可以有两种,分布适用于简短和较长的程序：

1. 在右下角的 Ipython窗口中输入一条或多条语句，然后回车；
2. 在左边的编辑窗口中输入程序，点击Run菜单的Run菜单项执行。

Ipython可以作为一个计算器使用，例如1.1。其中In[1]表示用户输入的第一条语句，Out[1]表示用户输入的第一条语句的执行结果。

Listing 1.1: **使用Ipython**

```
In[1]: (2+3*7-3)/(13-7)
Out[1]: 3.3333333333333335
```

1.10 选课须知

1.10.1 选课要求和课程安排

本课程面向满足以下条件的学生：

- 已经学习过微积分和线性代数；
- 对于程序设计具有浓厚兴趣；
- 自备笔记本电脑。

所有教学资料(课件和源程序等)将发送至选课同学的电子信箱。每次课程包括理论讲解和课堂编程练习。需要自备笔记本电脑并确保电量充足。

1.10.2 学习方法

1946年著名学习专家爱德加·戴尔发现不同学习方式的学习效果按从高到低呈金字塔分布。总体上，被动学习(听讲、阅读、视听、演示)的效果低于主动学习(讨论、实践、教他人)，其中听老师讲课是所有学习方式中效果最差的，最好的是“教他人”或“马上应用”。

学习软件开发的最有效方法是编写和调试程序，在实践中掌握知识和能力。

编程中遇到自己解决不了的问题怎么办？

- 查阅书籍和技术文档；

- 上网搜索：百度(www.baidu.com), 微软Bing国际版(https://cn.bing.com/?ensearch=1)，stackoverflow，...；
- 向老师和同学求助。

1.10.3 考核方式

考核方式是每位同学设计和实现一个Python科学计算程序并在课堂讲解和演示。采用这种考核方式的目的是：

- 学以致用，利用本学期所学设计和实现科学计算程序，助力自己的专业学习和科研工作。
- 训练计算思维能力。
- 训练创新能力和自主探索能力。

期末考核由三部分组成：程序、文档和课堂讲解，详细规则如下。

1. 程序的主要功能是用Python语言实现一个科学计算程序。程序至少包含50行代码。程序可以是自己独立设计和编写的，也可以对于书籍或互联网上已有程序做一些改进。程序必须能在上课使用的Anaconda环境中运行(版本2021.05)。完成程序以后需要使用已知正确结果的测试数据对程序进行测试。
2. 文档的格式是HTML，至少包含1000汉字。建议使用PythonLinks.html提供的HTML Composer编写文档(下载链接文件并安装以后，在浏览器的窗口菜单下选择Composer，即可打开HTML编辑器)。文档的主要内容包括科学技术原理，设计方案，创新性描述，运行方法和参数设置，参考文献，学习心得和收获。其中科学技术原理部分必须有属于科技论文或科技书籍类别的参考文献。文档的内容应以文字为主，可以包含若干图片，但所有图片文件的长度总和不超过800K字节。如果超过，可适当缩小图片的分辨率或删除非必要图片(例如程序运行可以生成的)。图片文件必须采用JPG或PNG压缩格式。对图片的引用要使用相对路径，例如src="images/formula.png"。
3. 课堂讲解的主要内容是基于文档介绍程序和演示程序的运行结果，时间尽量控制在10分钟以内。课堂讲解的时间可以在本学期的最后四次课(5月19日，5月26日，6月2日和6月9日)中选择。为了确保每位同学有充分的时间进行讲解，每次课最多安排15位同学，按预约次序额满为止。相对于预约时间提前至少一天将打包文件通过电子邮件发送给教师。为防止被淹没在垃圾邮件中，电子邮件的标题必须采用以下格式：Python+学号+姓名。打包文件采用ZIP格式进行压缩，长度不超过1000K字节。打包文件解压缩后的结构应符合以下要求：文件夹名称为学号；文件夹内包含的文件后缀可以为py,pyx,html,css,png,jpg,csv和txt(存储输入数据的文件)；文档HTML文件所引用的所有图片文件位于images子文件夹中。

评分依据有以下几个方面。

1. 创新性：体现在程序功能、设计、实现等方面
2. 技术含量：自己独立完成的代码的数量和质量

3. 程序的易用性：展示的运行结果是否直观易理解？如果程序需要输入大量数据，则应从文件读取数据，而不是让用户在界面上进行很多操作。

4. 学习心得和收获：学习本课程和完成大作业的过程中有哪些学习心得和收获、经验和教训？对教学有何意见建议？

5. 对于以上规则的遵守情况。

6. 预约时间在前三次课的同学在评分时有加分。

以下任何一种情形将导致不及格成绩：

- 未按要求提交程序和文档，或提交的程序无法运行；
- 未在课堂讲解程序；
- 完全抄袭已有代码(举报者将获得加分)。

其他未遵守以上规则的行为将导致扣分。例如：

- 在预约时间当天才提交打包文件；
- 打包文件包含一些非必要文件；
- 文档长度不合要求或内容不完整；
- 打包文件的长度超过1000K字节；
- 每次实验课(即周四下午的第8节课)上课时会随机抽取全班人数的25%进行考勤。为防止冒名顶替，请点到名的同学携带自己的一卡通到讲台登记。考勤完成以后不再接受登记。本学期累计缺席两次则成绩降低一个等级(如A-降为B+)，缺席三次及以上则成绩降低两个等级(如A-降为B)。因疫情原因未能到校的学生不列入实验课考勤范围。

第二章　内建数据类型和运算

程序中的数据都属于某一类型，类型规定了数据的存储形式和可以进行的运算。例如整数类型可以进行四则运算和位运算，而浮点类型可以进行四则运算但不能进行位运算。

和C、Java等静态类型语言不同，Python是一种动态类型语言，变量的类型在使用前无需声明，而是在程序运行的过程中根据变量存储的数据自动推断。Python赋值语句的一般语法形式是“变量=表达式”，其中的等号是赋值运算符，而非数学中的相等运算符。例如赋值语句x=1执行完成以后，变量x的类型即为整数类型int，因为x存储的数据1是整数。
动态类型的益处是程序语法更简单，付出的代价是程序的运行性能不如静态类型语言程序。原因包括无法进行编译时的优化和类型推断占用了程序运行的时间等。

本章介绍了Python语言的常用内建(built-in)数据类型和运算。Python语言的官方文档(https://docs.python.org/)详细说明了所有内建数据类型以及每种类型可进行的运算。

2.1　变量和类型

变量是计算机内存中存储数据的标识符，根据变量名称可以获取内存中存储的数据。变量名只能是字母、数字或下划线的任意组合。变量名的第一个字符不能是数字。以下Python关键字不能声明为变量名：

Listing 2.1: **Python关键字**

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

2.2　数值类型

数值类型包括int(整数)、float(浮点数)和complex(复数)。

2.2.1 int类型

int类型表示任意精度的整数，可以进行的运算包括相反(-)、加(+)、减(-)、乘(*)、除(/)、整数商(//)、取余(%)、乘方(**)和各种位运算。两个整数进行的位运算包括按位取或(|)、按位取与(&)和按位取异或(^)。单个整数进行的位运算包括按位取反(~)、左移(<<)和右移(>>)。内建函数bin可以显示一个整数的二进制形式。

2.2演示了int类型的运算，其中x和y通过赋值运算符(=)分别被赋予了整数值，因此都是int类型的变量。

Listing 2.2: **int类型的运算**

```
In[1]: (32+4)/(23-13)
Out[1]: 3.6
In[2]: (32+4)//(23-13)
Out[2]: 3
In[3]: (32+4)%(23-13)
Out[3]: 6
In[4]: x=379516400906811930638014896080
In[5]: x**2
Out[5]: 144032698557259999607886110560755362973171476419973199366400
In[6]: y=120557357903313594474425387 67
In[7]: 991*y**2
Out[7]: 144032698557259999607886110560755362973171476419973199366399
In[8]: Out[7]-Out[5]
Out[8]: -1
In[9]: x**2-991*y**2-1
Out[9]: 0
In[10]: bin(367), bin(1981)
Out[10]: ('0b101101111', '0b11110111101')
In[11]: bin(367 | 1981), bin(367 & 1981), bin(367 ^ 1981)
Out[11]: ('0b11111111111', '0b100101101', '0b11011010010')
In[12]: bin(~1981), bin(1981 << 3), bin(1981 >> 3)
Out[12]: ('-0b11110111110', '0b11110111101000', '0b11110111')
```

2.2.2 float类型

float类型根据IEEE754标准定义了十进制实数在计算机中如何表示为二进制浮点数。 64位二进制浮点数表示为$(-1)^s(1+f)2^{e-1023}$。s占1位表示正数和负数的符号。f占52位，$1+f$为小数部分。e占11位，$e-1023$为指数部分。$e=2047, f=0, s=\pm 1$表示正无穷和负无穷。$e=2047, f\neq 0$表示非数值(如0/0)。四舍五入的相对误差为$\frac{1}{2}2^{1-53}=2^{53}\approx 1.11\times 10^{-16}$，因此64位二进制浮点数对应的十进制实数的有效数字的位数为15。

sys模块的float_info属性描述了float类型的取值范围(max，min)和有效数字位数(dig)等信息。绝对值超过max的数值表示为inf(正无穷)或-inf(负无穷)。在使用sys模块之前需要用import语句将其导入。

Listing 2.3: **float类型**

```
In[1]: import sys
In[2]: sys.float_info
Out[2]: sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
   max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
   min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16,
    radix=2, rounds=1)
```

float类型可以进行相反(-)、加(+)、减(-)、乘(*)、除(/)、整数商(//)、取余(%)和乘方(**)等运算。由于需要进行进制转换和表示位数的限制，实数在计算机中的表示和计算可能存在误差。例如十进制的0.1表示为二进制无限循环小数 $0.000\overline{1100}$，在截断后导致误差。

在需要高精度计算结果的场合，进行浮点数计算时必须对误差的产生和积累进行严密的分析和控制，否则可能产生严重后果。1991年2月25日海湾战争期间，爱国者导弹防御系统运行100个小时以后积累了0.3422秒的误差。这个错误导致来袭导弹未被拦截，造成28名美军士兵死亡[1]。

2.4演示了float类型的运算，其中max通过赋值运算符(=)被赋予了浮点数值，因此是float类型的变量。”In[4]”这一行包含了多条语句，语句之间用分号(;)分隔，最后一条语句是用来输出max的值。float类型的数值的表示形式有两种：十进制和科学计数法。科学计数法用e或E表示指数部分，例如1.2345678909876543e38表示$1.2345678909876543\times 10^{38}$。

Listing 2.4: **float类型的运算**

```
In[1]: 10000*(1.03)**5
Out[1]: 11592.740743
In[2]: (327.6-78.65)/(2.3+0.13)**6
Out[2]: 1.2091341548676164
In[3]: 4.5-4.4
```

[1] https://www-users.cse.umn.edu/ arnold/disasters/patriot.html

```
Out[3]: 0.09999999999999964       #  TRUE VALUE SHOULD BE 0.1
In[4]: import sys; max = sys.float_info.max; max
Out[4]: 1.7976931348623157e+308
In[5]: max*1.001
Out[5]: inf                       #  OVERFLOW LEADS TO INFINITY
In[6]: sys.float_info.min*0.0000000000000001
Out[6]: 0                         #  UNDERFLOW LEADS TO 0
In[6]: 1.234567890987654321e38
Out[6]: 1.2345678909876543e+38    #  NUMBER OF SIGNIFICANT DIGITS <= 15
In[7]: import numpy as np
In[8]: (2**(2046-1023))*((1 + sum(0.5**np.arange(1, 53))))
Out[8]: 1.7976931348623157e+308
In[9]: (2**(1-1023))*(1+0)
Out[9]: 2.2250738585072014e-308
```

2.2.3 complex类型

complex类型表示实部和虚部为float类型的复数，可以进行相反(-)、加(+)、减(-)、乘(*)、除(/)、和乘方(**)等运算。

2.5演示了complex类型的运算，其中x和y通过赋值运算符(=)分别被赋予了复数值，因此是complex类型的变量。

Listing 2.5: **complex类型的运算**

```
In[1]: x = 3 - 5j;
In[2]: y = -(6 - 21j);
In[3]: (x+y)/(x - y**2)*(x**3 + y - 3j)
Out[3]: (-2.7021404738144748-6.422968879823101j)
In[4]: x.real
Out[4]: 3.0
In[5]: x.imag
Out[5]: -5.0
In[6]: x.conjugate()
Out[6]: (3+5j)
```

2.2.4 数值类型的内建函数

对于一个整数或实数，abs函数获取其绝对值。对于一个复数，abs函数获取其模。int和float函数可以进行这两种类型的相互转换。 complex函数从两个float类型的数值生成一个complex类型的数值。pow计算乘方，等同于**运算符。

2.6演示了这些函数的使用。

Listing 2.6: 数值类型的内建函数

```
In[1]: x = -15.6;
In[2]: y = int(x); y
Out[2]: -15
In[3]: type(y)
Out[3]: int
In[4]: x=float(y); x
Out[4]: -15.0
In[5]: type(x)
Out[5]: float
In[6]: z = complex(abs(x),(2 - y)); z
Out[6]: (15+17j)
In[7]: abs(z)
Out[7]: 22.671568097509265
In[8]: pow(z, 2+9j)
Out[8]: (-0.014836867717222271-0.24910017274317728j)
```

2.2.5 math模块和cmath模块

math模块定义了圆周率math.pi、自然常数math.e和大量以实数作为自变量和因变量的数学函数。cmath模块定义了大量以复数作为自变量和因变量的数学函数。

2.7演示了求解一元二次方程$ax^2 + bx + c = 0$ $(a \neq 0)$的根。当判别式$\Delta = b^2 - 4ac$为负时两个根为复数，使用math模块的求平方根函数(sqrt)会报错(ValueError)。此时需要使用对复数计算的cmath模块的sqrt函数。

Listing 2.7: 求解一元二次方程

```
In[1]: import math; a=2; b = 6; c = 1;
In[2]: r1 = (-b + math.sqrt(b**2 - 4*a*c))/(2*a); r1
Out[2]: -0.17712434446770464
```

```
In[3]: r2 = (-b - math.sqrt(b**2 - 4*a*c))/(2*a); r2
Out[3]: -2.8228756555322954
In[4]: a=2; b = 6; c = 8;
In[5]: r1 = (-b + math.sqrt(b**2 - 4*a*c))/(2*a); r1
Out[5]: ValueError: math domain error
In[6]: import cmath;
In[7]: r1 = (-b + cmath.sqrt(b**2 - 4*a*c))/(2*a); r1
Out[7]: (-1.5+1.3228756555322954j)
In[8]: r2 = (-b - cmath.sqrt(b**2 - 4*a*c))/(2*a); r2
Out[8]: (-1.5-1.3228756555322954j)
```

表2.1列出了math模块的一些常用函数。

函数名称	函数定义和示例
math.ceil(x)	大于等于x的最小整数: math.ceil(-5.3)值为-5
math.floor(x)	小于等于x的最大整数: math.floor(-5.3)值为-6
math.factorial(x)	x的阶乘: math.factorial(5)值为120
math.sqrt(x)	x的平方根: math.sqrt(3)值为1.7320508075688772
math.exp(x)	以自然常数e为底的指数函数: math.exp(2)值为7.38905609893065
math.log(x)	以自然常数e为底的对数函数: math.log(7.38905609893065)值为2.0
math.log(x, base)	以base为底的对数函数: math.log(7.38905609893065, math.e)值为2.0
math.log2(x)	以2为底的对数函数: math.log2(65536)值为16.0
math.log10(x)	以10为底的对数函数: math.log10(1e-19)值为-19.0
三角函数	math.sin(x) math.cos(x) math.tan(x)
反三角函数	math.asin(x) math.acos(x) math.atan(x)
双曲函数	math.sinh(x) math.cosh(x) math.tanh(x)
反双曲函数	math.asinh(x) math.acosh(x) math.atanh(x)

表 2.1: math模块的常用函数

2.3 bool类型

int类型和float类型的数据可以使用以下这些关系运算符进行比较: >(大于)、<(小于)、>=(大于等于)、<=(小于等于)、 ==(等于)、!=(不等于)。比较的结果属于bool类型，只有两种取值：True和False。bool类型的数据可以进行与(and)、或(and)和非(not)这三种逻辑运算，规则列于表2.2。

2.8演示了使用比较运算符和逻辑运算符判断一个年份是否闰年。

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

表 2.2: 逻辑运算的规则

Listing 2.8: **bool类型的运算**

```
In[1]: year = 1900
In[2]: (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
Out[2]: False
In[3]: year = 2020
In[4]: (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
Out[4]: True
In[5]: year = 2022
In[6]: (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
Out[6]: False
```

2.4 序列类型

序列类型(Sequence Types)可以看成一个存储数据的容器，包括list(列表)、tuple(元组)和 range(范围)。list和tuple通常用来存储若干同一类型的数据，range通常用来存储若干循环中使用的数值。
序列中存储的数据是有序的，每条数据对应一个索引值。设序列的长度为n，则索引值的取值范围从$-n-1$到$n-1$。序列中的第k条数据($1 \le k \le n$)对应的索引值有两个: $k-1$和$k-n-1$。例如，索引值0和$-n$都对应第1条数据，索引值$n-1$和-1都对应第n条数据。

list类型的语法是用方括号括起的用逗号分隔的若干数据，例如[2,3,5,7,11]是一个存储了五个质数的列表。tuple类型的语法是用圆括号括起的用逗号分隔的若干数据，例如(2,3,5,7,11)是一个存储了五个质数的元组。

表2.3列出了list和tuple类型的共有运算。示例中s的值为[2,3,5,7,11]，t的值为[13,17]。 list是可变的，即其中存储的数据可以被修改。tuple和range是不可变的。表2.4列出了list类型的特有运算，每个示例中s的初始值为[2,7,5,3,11]，t的值为[13,21,17]。
2.9演示了list的一些运算。其中组成列表d的每条数据(a和b)本身也是一个列表，类似d这样的列表称为嵌套列表。获取d中的数据需要使用两层索引。对于d而言，d[0]等同于a，d[1]等同于b。因此，d[0][4]等同于a[4]，d[1][0:5:2]等同于b[0:5:2]。

运算名称	运算定义和示例
x in s	若s中存在等于x的数据，则返回True，否则返回False。例: 5 in s 值为True
x not in s	若s中存在等于x的数据，则返回False，否则返回True。 例: 7 not in s 值为False
s + t	返回将s和t连接在一起得到的序列。例: s + t 值为[2,3,5,7,11,13,17]
s*n 或 n*s	返回将s重复n次得到的序列。例: s*3值为[2,3,5,7,11,2,3,5,7,11,2,3,5,7,11]
s[n]	返回s中索引值为n的数据。例: s[4]值为11
s[i:j]	返回s中索引值从i到j-1的所有数据构成的序列。例: s[1:4]值为[3, 5, 7]
s[i:]	返回s中索引值从i开始到最后的所有数据构成的序列。例: s[1:]值为[3, 5, 7, 11]
s[:j]	返回s中索引值从0开始到j-1的所有数据构成的序列。例: s[:3]值为[2, 3, 5]
s[i:j:k]	返回s中索引值属于从i到j-1且等差为k的等差数列的所有数据构成的序列。 例: s[1:4:2]值为[3, 7]
len(s)	返回s的长度。例: len(s)值为5
min(s)	返回s中的最小数据。例: min(s)值为2
max(s)	返回s中的最大数据。例: max(s)值为11
s.index(x)	若s中存在数据x，则返回数据x第一次出现的索引值，否则报错。 例: s.index(7)值为3
s.index(x, i)	在s中从索引值i开始向后查找，若存在数据x则返回其索引值，否则报错。 例: s.index(7,1)值为3
s.index(x, i, j)	在s中从索引值i开始向后查找到索引值j-1为止， 若存在数据x则返回其索引值，否则报错。例: s.index(7,1,4)值为3
s.count(x)	返回x在s中出现的次数。例: s.count(8)值为0

表 2.3: list和tuple类型的共有运算

Listing 2.9: **list类型的运算**

```
In[1]: a=[1,3,5,7,9]; b=[2,4,6,8,10]
In[2]: c=a+b; c
Out[2]: [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
In[3]: c.sort(); c
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In[4]: d=[a,b]; d
Out[4]: [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
In[5]: c[2:10:3]
Out[5]: [3, 6, 9]
In[6]: c[-1:-9:-4]
Out[6]: [10, 6]
In[7]: d[0][4]
Out[7]: 9
In[8]: d[1][0:5:2]=d[0][2:5]; d
Out[8]: [[1, 3, 5, 7, 9], [5, 4, 7, 8, 9]]
In[9]: b
Out[9]: [5, 4, 7, 8, 9]
```

2.5 str类型

str类型表示不可变的使用Unicode编码的字符构成的序列，即字符串。Unicode是一个字符编码的标准，为每种语言中的每个字符设定了统一并且唯一的二进制编码。ord函数可获取一个字符对应的Unicode编码值，例如ord('A')值为65。chr函数可获取一个Unicode编码值对应的字符，例如chr(90)值为'Z'。UTF-8是一种国际上广泛使用的 Unicode实现方式，即用一个或多个字节存储二进制编码。

字符串可用单引号、双引号或三个相同的单(双)引号括起。单引号和双引号可相互嵌套。用三引号括起的字符串可以跨越程序中的多行语句(包括其中的换行符和空格), 常作为函数和模块的注释。

字符串除了支持表2.3列出的运算以外，还有一些特有运算，包括判断和查找子串(startswith, endswith, find, rfind)，判断组成字符串的所有字符是否属于某一类别(isalpha, isascii, isdecimal, islower, isspace, isupper)，在首尾删除特定字符(lstrip和rstrip)，使用分隔符分割(split, rsplit)和替换子串(replace)等。

2.10演示了字符串的一些常用运算。

运算名称	运算定义和示例
s[i] = x	将s中索引值为i的数据修改为x。例: 执行s[3] = 8以后s值为[2,3,5,8,11]
s[i:j] = t	将s中索引值从i到j-1的所有数据构成的序列修改为t。 例: 执行s[1:4] = t以后s值为[2, 13, 21, 17, 11]
del s[i:j]	删除s中索引值从i到j-1的所有数据构成的序列。 例: 执行del s[2:4]以后s值为[2, 7, 11]
s[i:j:k] = t	将s中索引值属于从i到j-1且等差为k的等差数列的所有数据构成的序列 修改为x。例: 执行s[0:5:2] = t以后s值为[13, 7, 21, 3, 17]
del s[i:j:k]	删除s中索引值属于从i到j-1且等差为k的等差数列的所有数据构成 的序列。例: 执行del s[0:5:2]以后s值为[7, 3]
s.append(x)	添加数据x到列表s中使其成为最后一条数据。 例: 执行s.append(13)以后s值为[2, 7, 5, 3, 11, 13]
s.clear()	删除s中所有数据。例: 执行s.clear()以后s值为[]
s.copy()	返回s的一个副本。例: s.copy()返回[2,7,5,3,11]
s.extend(t)	将序列t添加到s的后面。 例: 执行s.extend(t)以后s值为[2, 7, 5, 3, 11, 13, 21, 17]
s += t	同上
s *= n	修改s为将s重复n次得到的序列 例: 执行s *= 3以后s值为[2, 7, 5, 3, 11, 2, 7, 5, 3, 11, 2, 7, 5, 3, 11]
s.insert(i, x)	添加数据x到列表s中使其成为索引值为i的数据。 例: 执行s.insert(2, 19)以后s值为[2, 7, 19, 5, 3, 11]
s.pop(i)	删除s中索引值为i的数据并将其返回。 例: 执行s.pop(3)返回3，且s的值为 [2, 7, 5, 11]
s.remove(x)	删除s中第一次出现的数据x。例: 执行s.remove(11)以后s值为[2, 7, 5, 3]
s.reverse()	将s中所有数据的次序反转。例: 执行s.reverse()以后s值为[11, 3, 5, 7, 2]
s.sort()	将s中所有数据按从小到大的次序排序。 例: 执行s.sort()以后s值为[2, 3, 5, 7, 11]
s.sort(reverse =True)	将s中所有数据按从大到小的次序排序。 例: 执行s.sort(reverse=True)以后s值为 [11, 7, 5, 3, 2]

表 2.4: list类型的特有运算

Listing 2.10: **str类型的运算**

```
In[1]: a = 'allows␣embedded␣"double"␣quotes'; a
Out[1]: 'allows␣embedded␣"double"␣quotes'
In[2]: b = "allows␣embedded␣'single'␣quotes"; b
Out[2]: "allows␣embedded␣'single'␣quotes"
In[3]: c = """A=[1,3,5,7,9]; B=[2,4,6,8,10]
  ...: C=A+B
  ...: C.SORT(); C"""
In[4]: c
Out[4]: 'a=[1,3,5,7,9];␣b=[2,4,6,8,10]\nc=a+b\nc.sort();␣c'
In[5]: d = [a.startswith('allow'), a.startswith('allou')]; d
Out[5]: [True, False]
In[6]: e = [a.startswith('embee', 7), a.endswith('quo', 3, -3)]
Out[6]: [False, True]
In[7]: f = [a.find('em', 3, 6), a.find('em', 7)]; f
Out[7]: [-1, 7]
In[8]: g = '␣␣hello?!!'
In[9]: h = [g.lstrip(), g.rstrip('!?'), g.strip('␣!')]; h
Out[9]: ['hello?!!', '␣␣hello', 'hello?']
In[10]: a.replace('e', 'x', 1)
Out[10]: 'allows␣xmbedded␣"double"␣quotes'
In[11]: a.replace('e', 'yy', 3)
Out[11]: 'allows␣yymbyyddyyd␣"double"␣quotes'
In[12]: a.split('e')
Out[12]: ['allows␣', 'mb', 'dd', 'd␣"doubl', '"␣quot', 's']
In[13]: a.rsplit('d', 2)
Out[13]: ['allows␣embedde', '␣"', 'ouble"␣quotes']
```

str类型的%运算符可以使用多种转换说明符为各种类型的数据生成进行格式化输出。%左边是一个字符串，其中可包含一个或多个转换说明符。%右边包含一个或多个数值，这些数值必须和转换说明符一一对应。如果有多个数值，这些数值必须置入一个元组中。 2.11中的"%-16.8f"中的负号表示左对齐(无负号表示右对齐)，16表示所生成的字符串的长度，在点以后出现的8表示输出结果在小数点以后保留8位数字。

Listing 2.11: **格式化输出**

```
In[1]: "%-16.8f" % 345.678987654321012
```

```
Out[1]: '345.67898765␣␣␣␣'
In[2]: "%16.8g" % 3.45678987654321012e34
Out[2]: '␣␣␣3.4567899e+34'
In[3]: "%16X␣%-16d" % (345678987654, 987654321012)
Out[3]: '␣␣␣␣␣␣507C12C186␣987654321012␣␣␣␣'
```

表2.5列出了常用的转换说明符及其定义。

转换说明符名称	转换说明符定义
%d、%i	转换为带符号的十进制整数
%o	转换为带符号的八进制整数
%x、%X	转换为带符号的十六进制整数
%e、%E	转换为科学计数法表示的浮点数（e小写、E 大写）
%f、%F	转换为十进制浮点数
%g、%G	智能选择使用 %f或%e格式(%F或 %E格式)
%c	将整数转换为单个字符的字符串
%r	使用 repr()函数将表达式转换为字符串
%s	使用 str()函数将表达式转换为字符串

表 2.5: 常用的转换说明符

2.6 set类型

set类型可以看成一个存储数据的容器，存储的数据是无序且不可重复的，类似于数学中定义的集合。

2.12演示了一些set类型的运算。

Listing 2.12: **set类型的运算**

```
In[1]: l = [2,3,5,3,9,2,7,8,6,3]; (l, type(l))
Out[1]: ([2, 3, 5, 3, 9, 2, 7, 8, 6, 3], list)
In[2]: s = set(l); (s, type(s))
Out[2]: ({2, 3, 5, 6, 7, 8, 9}, set)
In[3]: t = set([11, 2, 7, 3, 5, 13])
In[4]: s.union(t)
Out[4]: {2, 3, 5, 6, 7, 8, 9, 11, 13}
In[5]: s.intersection(t)
Out[5]: {2, 3, 5, 7}
```

```
In[6]: s.difference(t)
Out[6]: {6, 8, 9}
```

表2.6列出了set类型的常用运算。

运算名称	运算定义
len(s)	返回s的长度，即其中存储了多少条数据
x in s	若s中存在等于x的数据，则返回True，否则返回False。
x not in s	若s中存在等于x的数据，则返回False，否则返回True。
s.isdisjoint(t)	若s和t的交集非空，则返回True，否则返回False。
s.issubset(t)	若s是t的子集，则返回True，否则返回False。
s<=t、s<t	若s是t的子集(真子集)，则返回True，否则返回False。
s.issuperset(t)	若s是t的超集，则返回True，否则返回False。
s>=t、s>t	若s是t的超集(真超集)，则返回True，否则返回False。
s.union(t)	返回s和t的并集
s.intersection(t)	返回s和t的交集
s.difference(t)	返回s和t的差集

表 2.6: set类型的常用运算

2.7 dict类型

dict类型可以看成一个存储数据的容器，存储的每条数据由两部分组成: 关键字和其映射到的值。dict类型的语法是用大括号括起的多条数据，每条数据内部用冒号分隔关键字和值，数据之间用逗号分隔。

2.13演示了一个dict类型的实例(通讯录)，其中每个联系人的姓名映射到其电话号码。

Listing 2.13: **dict类型的运算**

```
In[1]: contacts={"Tom":12345, "Jerry":54321, "Mary":23415}; contacts
Out[1]: {'Tom': 12345, 'Jerry': 54321, 'Mary': 23415}
In[2]: contacts["Jerry"]=54123; contacts["Betty"]=35421; contacts
Out[2]: {'Tom': 12345, 'Jerry': 54123, 'Mary': 23415, 'Betty': 35421}
In[3]: contacts.keys()
Out[3]: dict_keys(['Tom', 'Jerry', 'Mary', 'Betty'])
In[4]: ['Tommy' in contacts, 'Betty' in contacts]
Out[4]: [False, True]
In[5]: (contacts.pop('Jerry'), contacts)
```

```
Out[5]: (54123, {'Tom': 12345, 'Mary': 23415, 'Betty': 35421})
In[6]: (contacts.pop('Tommy', None), contacts)
Out[6]: (None, {'Tom': 12345, 'Mary': 23415, 'Betty': 35421})
```

表2.7列出了dict类型的常用运算。

运算名称	运算定义
len(d)	返回d的长度，即其中存储了多少条数据
key in d	若s中存在关键字等于key的数据，则返回True，否则返回False。
key not in d	若s中存在关键字等于key的数据，则返回False，否则返回True。
d[key] = value	若d中存在关键字等于key的数据，则将其对应的值修改为value。 否则在d中添加一条数据，其关键字和值分别为key和value。
del d[key]	若d中存在关键字等于key的数据则将其删除，否则报错。
clear()	删除所有数据。
get(key[, default])	若d中存在关键字等于key的数据则返回其对应的值，否则返回default。 若未提供default，则返回None。
pop(key[, default])	若d中存在关键字等于key的数据则将其删除，然后返回其对应的值， 否则返回default。若未提供default，则返回None。
keys()	返回d中所有数据的关键字
values()	返回d中所有数据的值

表 2.7: dict类型的常用运算

习题

1. 假设一个程序的运行时间T满足$T = knlog_2n$，其中k是一个常数，n是输入数据的规模。若程序在$n = 10$时的运行时间为1秒，计算$n = 50$和$n = 100$时的运行时间，分别以秒和小时作为单位。

2. 假设一个程序的运行时间T满足$T = kn^2$，其中k是一个常数，n是输入数据的规模。若程序在$n = 10$时的运行时间为1秒，计算$n = 50$和$n = 100$时的运行时间，分别以秒和小时作为单位。

3. 假设一个程序的运行时间T满足$T = kn^3$，其中k是一个常数，n是输入数据的规模。若程序在$n = 10$时的运行时间为1秒，计算$n = 50$和$n = 100$时的运行时间，分别以秒和小时作为单位。

4. 假设一个程序的运行时间T满足$T = k1.2^n$，其中k是一个常数，n是输入数据的规模。若程序在$n = 10$时的运行时间为1秒，计算$n = 50$和$n = 100$时的运行时间，分别以秒、小时和天作为单位。

5. 假设一个程序的运行时间T满足$T = k2^n$，其中k是一个常数，n是输入数据的规模。若程序在$n = 10$时的运行时间为1秒，计算$n = 50$和$n = 100$时的运行时间，分别以秒、小时、天和年作为单

位。

6. 贷款的每月还款金额P可根据贷款金额A、年利率r和贷款月数n计算得到，公式为$P = \frac{\frac{r}{12}A}{1-(1+\frac{r}{12})^{-n}}$。计算当贷款金额为1000000，贷款时间为30年，年利率分别为4% 、5%和6%时的每月还款金额和还款总额。

7. 以自己设定的s和t进行表2.3、2.4、2.6和2.7列出的每项运算，理解运算结果。

第三章　分支和迭代

分支和迭代是程序中常用的流程控制结构。分支的语义是根据若干条件是否满足从多个分支中选择一个执行，由if语句和if-else表达式实现。迭代的语义是当某一条件满足时反复执行一个语句块，由for语句、while语句和推导式实现。

3.1　if语句和if-else表达式

if语句可以根据若干条件是否满足从多个分支中选择一个执行。if语句可以有多种形式，以实现一个程序计算整数的绝对值为例说明。根据绝对值的定义，可以有以下三种实现方式：3.1，3.2和3.3。这三个程序的第一行使用内建函数input提示用户输入一个整数，提示信息是"Please enter an integer: "。用户输入的内容作为一个字符串由int函数转换为整型值后赋值给变量x。最后一行的内建函数print输出字符串表达式"The absolute value of %d is %d" %(x, y)的值。

程序3.1的3到4行是只有一个分支的if语句，当条件表达式(x<0)的值为True时执行if后面的分支。程序3.2的2到5行是有两个分支的if语句，当表达式(x<0)的值为True时执行if后面的分支，当表达式(x<0)的值为False时执行else后面的分支。程序3.3的第2行右侧是一个if-else表达式，它实现了相同的功能。

Listing 3.1: **单分支if语句计算绝对值**

```
1 x = int(input("Please␣enter␣an␣integer:␣"))
2 y = x
3 if x < 0:
4     y = -x
5 print ("The␣absolute␣value␣of␣%d␣is␣%d" %(x, y))
```

Listing 3.2: **多分支if语句计算绝对值**

```
1 x = int(input("Please␣enter␣an␣integer:␣"))
2 if x < 0:
3     y = -x
4 else:
```

```
5     y = x
6 print ("The␣absolute␣value␣of␣%d␣is␣%d" %(x, y))
```

Listing 3.3: **if-else表达式计算绝对值**

```
1 x = int(input("Please␣enter␣an␣integer:␣"))
2 y = -x if x < 0 else x
3 print ("The␣absolute␣value␣of␣%d␣is␣%d" %(x, y))
```

程序3.4利用多分支if语句将百分制成绩转换为等级分。第2行将等级分的默认值设置为'F'。第3行判断用户输入的百分制成绩是否在有效范围。若不在，则第4行将等级分设置为一个特殊标记'Z'。第5行的条件等价于90<=x<=100，如果此条件成立，则第6行将等级分设置为'A'。如果第5行的条件不成立，则继续判断第7行的条件，该条件等价于80<=x<90。如果此条件成立，则第8行将等级分设置为'B'。如果第7行的条件不成立，则继续判断第9行的条件，该条件等价于70<=x<80。如果此条件成立，则第10行将等级分设置为'C'。如果第9行的条件不成立，则继续判断第11行的条件，该条件等价于60<=x<70。如果此条件成立，则第12行将等级分设置为'D'。如果第11行的条件不成立，x必然满足0<=x<60，此时无需给grade赋值，因为grade的值为默认值'F'。

需要注意的是，if语句包含的每个分支可以是一条语句，也可以是多条语句组成的语句块。这些分支相对if必须有四个空格的缩进。唯一的例外情形是一个分支的if语句，例如程序3.1的3到4行可以写成一行: if x < 0: y = -x 。

Listing 3.4: **百分制成绩转换为等级分**

```
1 x = int(input("Please␣enter␣a␣score␣within␣[0,␣100]:␣"))
2 grade = 'F';
3 if x > 100 or x < 0:
4     grade = 'Z';
5 elif x >= 90:
6     grade = 'A';
7 elif x >= 80:
8     grade = 'B';
9 elif x >= 70:
10    grade = 'C';
11 elif x >= 60:
12    grade = 'D';
13 print ("The␣grade␣of␣score␣%d␣is␣%c" %(x, grade))
```

3.2 for语句

for语句可对某一序列中的每条数据执行一个语句块，以下举例说明。

程序3.5输出1到10之间的所有自然数的和，第2行的range(1, n+1)生成一个range类型的序列(1,2,3...,n)。

程序3.6输出100到110之间的所有偶数，第2行的range(lb, ub+1, 2)生成一个range 类型的序列(100,102,...,120), 第3行的print函数的参数(end=' ')的作用是在每输出一个数后输出一个空格，而不是换行。

Listing 3.5: **for语句输出1到10之间的所有自然数的和**

```
1 sum = 0; n = 10
2 for i in range(1, n+1):
3     sum += i
4 print("The␣sum␣of␣1␣to␣%d␣is␣%d" % (n, sum))
5 # THE SUM OF 1 TO 10 IS 55
```

Listing 3.6: **for语句输出100到120之间的所有偶数**

```
1 lb = 100; ub = 120
2 for i in range(lb, ub+1, 2):
3     print(i, end='␣')
4 # 100 102 104 106 108 110 112 114 116 118 120
```

程序3.7输出一个由整数组成的列表中所包含的3的倍数。

Listing 3.7: **for语句输出一个由整数组成的列表中所包含的3的倍数**

```
1 nums = {25, 18, 91, 365, 12, 78, 59}
2 for i in nums:
3     if i % 3 == 0: print(i, end='␣')
4 #  12 78 18
```

程序3.8采用两种方式输出一个通讯录中的每个联系人的姓名和对应的电话号码。

Listing 3.8: **for语句输出一个通讯录中的每个联系人的姓名和对应的电话号码**

```
1 contacts = {"Tom":12345, "Jerry":54321, "Mary":23415}
2
```

```
for name, num in contacts.items():
    print('%s␣->␣%d' % (name, num), end=';␣')
#  Tom -> 12345; Jerry -> 54321; Mary -> 23415;
print()
for name in contacts.keys():
    print('%s␣->␣%d' % (name, contacts[name]), end=';␣')
#  Tom -> 12345; Jerry -> 54321; Mary -> 23415;
```

怎样使用for语句实现一个程序输出某一给定自然数区间内的所有质数？这个问题比我们之前所解决的问题更加复杂。当问题比较复杂时，在编写程序之前应提出一个设计方案，这样便于对解决问题的策略和步骤进行深入而细致的思考，避免错误。此外，还可以在保证正确性的前提下选择最优解决方案，提高程序的执行速度并降低资源占用。

设计方案如下：

1. 列举给定区间内的所有自然数。
2. 对于每个自然数i，判断其是否质数，如果是则输出。怎样判断i是否质数？
   (a) 对于每个从2到$i-1$的自然数j，检查i是否可以被j整除。若存在这样的j，则i非质数。否则i为质数，输出i。

根据质数的定义，这个设计方案是正确的，而且每个步骤都易于实现，但是在性能上还有改进的余地。在步骤1列举自然数时，只需列出奇数，因为偶数肯定不是质数。在步骤1(a)查找i的因子j时，j的取值范围的上界可以缩小为$\lceil\sqrt{i}\rceil$。因为若$i = j \times k$，则j和k中至少有一个不大于$\lceil\sqrt{i}\rceil$。

基于以上改进的设计方案，可以使用嵌套for语句写出程序3.9。第4行开始的外循环用range类型列举所有奇数。6到9行的内循环检查i是否有因子，如果有则将isPrime的值设为False，并使用break语句跳出内循环。第10行根据isPrime的值决定是否输出i。6到9行的内循环作为一个整体相对于第4行的for必须有四个空格的缩进。

Listing 3.9: **for语句和break语句输出100到200之间的所有质数**

```
import math
lb = 100; ub = 200
if lb % 2 == 0: lb += 1
for i in range(lb, ub + 1, 2):
    isPrime = True
    for j in range(2, math.ceil(math.sqrt(i)) + 1):
        if i % j == 0:
            isPrime = False
```

```
9             break
10    if isPrime: print(i, end='␣')
11 #  101 103 107 109 113 127 ... 199
```

程序3.10实现了和3.9相同的功能。区别在于第10行如果确认isPrime的值为False，则使用 continue语句跳过本次外循环的剩余语句并开始下一次外循环，否则在第11行输出i。

Listing 3.10: **for语句和continue语句输出100到200之间的所有质数**

```
1  import math
2  lb = 100; ub = 200
3  if lb % 2 == 0: lb += 1
4  for i in range(lb, ub + 1, 2):
5      isPrime = True
6      for j in range(2, math.ceil(math.sqrt(i)) + 1):
7          if i % j == 0:
8              isPrime = False
9              break
10     if not isPrime: continue
11     print(i, end='␣')
```

3.3 while语句

while语句包含一个条件表达式和一个语句块。while语句的执行过程如下：

1. 对条件表达式求值。
2. 若值为False，则while语句执行结束。
3. 若值为True，则执行语句块，然后跳转到1。

程序3.11用while语句实现了和3.5相同的功能。

Listing 3.11: **while语句输出1到10之间的所有自然数的和**

```
1  sum = 0; n = 10; i = 1
2  while i <= n:
3      sum += i
4      i += 1
5  print("The␣sum␣of␣1␣to␣%d␣is␣%d" % (n, sum))
```

程序3.12用while语句实现了和3.6相同的功能。

Listing 3.12: **while语句输出100到120之间的所有偶数**

```
1 i = lb = 100; ub = 120
2 while i <= ub:
3     print(i, end='␣')
4     i += 2
```

for语句常用于循环次数已知的情形，而while语句也适用于循环次数未知的情形。程序3.13用while语句实现了辗转相减法求最大公约数。

Listing 3.13: **辗转相减法求最大公约数**

```
1 a = 156; b = 732
2 str = 'The␣greatest␣common␣divisor␣of␣%d␣and␣%d␣is␣' % (a, b)
3 while a != b:
4     if a > b:
5         a -= b;
6     else:
7         b -= a;
8 print (str + ('%d' % a))
9 # THE GREATEST COMMON DIVISOR OF 156 AND 732 IS 12
```

3.4 推导式

list、dict和set等容器类型都提供了一种称为推导式(comprehension)的紧凑语法，可以通过迭代从已有容器创建新的容器。

程序3.14演示了推导式的用法。第2行创建一个列表，由原列表中3的倍数构成。第4行创建一个集合，由原列表中的奇数的平方构成。第8行从一个集合创建一个字典sr，sr中的每条数据由原集合中的每个数和其除以3得到的余数组成。第10行从sr创建另一个字典tr，由sr中3的倍数组成。

Listing 3.14: **推导式的用法**

```
1 nums = {25, 18, 91, 365, 12, 78, 59}
2 multiplier_of_3 = [n for n in nums if n % 3 == 0]
3 print(multiplier_of_3) # [12, 78, 18]
4 square_of_odds = {n*n for n in nums if n % 2 == 1}
5 print(square_of_odds) # {133225, 3481, 625, 8281}
```

```
6
7  s = [25, 18, 91, 365, 12, 78, 59, 18, 91]
8  sr = {n:n%3 for n in set(s)}
9  print (sr) # {18: 0, 25: 1, 91: 1, 59: 2, 12: 0, 365: 2, 78: 0}
10 tr = {n:r for (n,r) in sr.items() if r==0}
11 print (tr) # {18: 0, 12: 0, 78: 0}
```

习题

1. 定义一个从给定正整数n构建一个整数序列的过程如下。开始时序列只包含n。如果序列的最后一个数m不为1则根据m的奇偶性向序列追加一个数。如果m是偶数，则追加$m/2$，否则追加$3 \times m + 1$。考拉兹猜想(Collatz conjecture)认为从任意正整数构建的序列都会以1终止。编写程序读取用户输入的正整数n，然后在while循环中输出一个以1终止的整数序列。

2. 编写程序实现基于偏移量的字符串加密。加密的过程是对原字符串中的每个字符对应的Unicode值加上一个偏移量，然后将得到的Unicode值映射到该字符对应的加密字符。用户输入一个绝对值不超过60的非零整数和一个由大小写字母或数字组成的字符串，程序生成并输出加密得到的字符串。例如用户输入10和字符串“Attack at 1600”得到的加密字符串是“K kmu*k *;@::”。需要思考的问题是：怎样对加密得到的字符序列进行解密？怎样改进这个加密方法(例如对每个字符设置不同的偏移量)？

3. 将程序3.14中的所有推导式转换为for语句。

4. 编写程序解决1.3节的输出日历问题。

第四章　函数和模块

4.1　定义和调用函数

函数是一组语句，可以根据输入参数计算输出结果。把需要多次执行的代码写成函数，可以避免重复代码。函数作为程序的组成单位，使程序更易理解和维护。

函数的定义包括函数头和函数体两部分。例如程序3.13可以改写为一个函数gcd。该函数接受两个自然数作为形参(formal parameter)，计算其最大公约数并返回。在程序4.1中，函数gcd的定义包括前9行语句。定义函数的第1行语句是函数头，以关键字def开始，之后是空格和函数的名称(gcd)。函数名称后面是用括号括起的一个或多个形参。如果有多个形参，它们之间用逗号分隔。第2行语句开始的剩余语句(第2行到第9行)构成函数体。函数体相对def所在行需要有四个空格的缩进。函数体由一条或多条语句构成，完成函数的功能。def行后面通常写一个由三个引号括起的字符串作为函数的注释，说明函数的输入参数、所实现的功能、返回结果和设计思路等内容。第9行的return语句返回变量a的值作为运行结果。

调用函数的语法是在函数的名称后面加上用括号括起一个或多个实参(argument)。如果有多个实参，它们之间用逗号分隔。这些实参必须和函数的形参在数量上相同，且在顺序上一一对应。第11行和第12行分别用两组整数作为实参分别调用gcd函数。以第11行为例，实参156赋值给了函数的形参a，实参732赋值给了函数的形参b。

第11行用参数156和732调用函数gcd，得到的结果是12。类似，第10行用参数1280和800调用函数gcd，得到的结果是160。

Listing 4.1: 定义一个求最大公约数的函数

```
1 def gcd(a, b):
2     """ COMPUTE THE GREATEST COMMON DIVISOR OF A AND B USING REPEATED
          SUBTRACTIONS """
3     while a != b:
4         if a > b:
5             a -= b;
```

```
6         else:
7             b -= a;
8     return a
9
10 print (gcd(156, 732)) # 12
11 print (gcd(1280, 800)) # 160
```

函数可以返回多个结果，这些结果之间用逗号分隔，构成一个元组。例如程序4.2中定义一个函数max_min(1-8行)。该函数接受两个数作为参数，返回它们的最大数和最小数。

Listing 4.2: 定义一个求最大数和最小数的函数

```
1 def max_min(a, b):
2     """ COMPUTE THE MAXIMUM AND MINIMUM OF A AND B """
3     if a > b:
4         return a, b
5     else:
6         return b, a
7
8 print (max_min(156, 34)) # (156, 34)
9 print (max_min(12, 800)) # (800, 12)
```

4.2 局部变量和全局变量

函数的形参和在函数体内定义的变量称为局部变量。局部变量在函数体内可访问，在函数执行结束时即被销毁。局部变量在函数体外无法访问。在函数体外定义的变量称为全局变量。全局变量在任何函数中都可以被访问，除非函数中定义了同名的局部变量。

例如程序4.3中前两行定义的变量b和c是全局变量。第3行函数f的形参a和第4行定义的变量b是函数f的局部变量。函数f中可以访问全局变量c，但无法访问全局变量b。第6行的输出结果表明第5行中的b是局部变量。第7行的输出结果表明第4行是给局部变量b赋值。

Listing 4.3: 局部变量和全局变量

```
1 b = 10
2 c = 15
3 def f(a):
4     b = 20
```

```
5     return a + b + c
6 print (f(5)) # 40
7 print ('b␣=␣%d' % b) # B = 10
```

如果需要在在函数体中修改全局变量，则需要用global声明全局变量。程序4.4中第4行用global声明了全局变量b。第8行的输出结果表明第5行是给全局变量b赋值。

Listing 4.4: 函数中修改全局变量

```
1 b = 10
2 c = 15
3 def f(a):
4     global b
5     b = 20
6     return a + b + c
7 print (f(5)) # 40
8 print ('b␣=␣%d' % b) # B = 20
```

4.3 默认值形参和关键字实参

在函数头中可以一个或多个形参赋予默认值，这些形参称为默认值形参(default parameters)。这些默认值形参的后面不能出现普通的形参。在调用函数的语句中，可以在一个或多个实参的前面写上其对应的形参的名称。这些实参称为关键字实参(keyword arguments)。此时实参的顺序不必和函数头中的形参顺序一致。

程序4.5中定义一个函数get_primes(1-13行)。该函数接受两个自然数作为参数，返回以这两个自然数作为下界和上界的区间中的所有质数。这里表示上界的形参ub设置了默认值100，所以第17行的调用get_primes(80)等同于get_primes(80, 100)。第18行的调用get_primes(ub=150, lb=136)使用了两个关键字实参，即和实参150对应的形参是ub且和实参136对应的形参是lb。关键字实参的顺序不必和函数头中的形参顺序严格一致。

根据用户提供的区间范围，这个函数需要返回的质数的有多个而且数量不确定，对于类似的情形可以把所有需要返回的结果存储在一个容器中，最后返回整个容器。具体来说，第3行定义了一个空列表primes。第11行确认isPrime为True时将i追加到primes中。第12行返回列表primes。

Listing 4.5: 定义一个求给定取值范围内的所有质数的函数

```
1 def get_primes(lb, ub=100):
2     """ COMPUTE THE PRIME NUMBERS WITHIN THE INTERVAL [LB, UB] """
```

```
3     import math
4     primes = []
5     if lb % 2 == 0: lb += 1
6     for i in range(lb, ub + 1, 2):
7         isPrime = True
8         for j in range(2, math.ceil(math.sqrt(i)) + 1):
9             if i % j == 0:
10                isPrime = False
11                break
12        if isPrime: primes.append(i)
13    return primes
14
15 print (get_primes(40, 50)) # [41, 43, 47]
16 print (get_primes(120, 140)) # [127, 131, 137, 139]
17 print (get_primes(80)) # [83, 89, 97]
18 print (get_primes(ub=150, lb=136)) # [137, 139, 149]
```

4.4 函数作为实参

进行数值微分计算的程序需要把一个函数作为调用另一个函数时提供的实参。例如要实现一个Python函数使用中心差分公式近似计算函数f(x)的二阶导数。

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{4.1}$$

程序4.6中的diff2nd函数实现了公式4.2，它的形参f是要计算二阶导数的函数。第3行从math模块导入了exp函数和sin函数。第5行使用这些函数定义了函数$g(x) = e^{xsin(x)}$。第8行到第10行的循环初始化一个列表存储了不同的步长值，并输出这些值。第12行定义了一个全局变量x0作为计算二阶导数的自变量值。第13行到第15行的循环以函数g、x0和不同的步长值作为实参调用函数diff2nd计算x0处的二阶导数函数值。为了把这些近似计算值和精确值做比较，第18行到22行使用sympy符号计算模块获得函数g的二阶导数的解析形式，然后计算x0处的函数值作为精确值。

从比较的结果可以看出，当步长h不断减小时，近似计算值和精确值的误差也不断减小。当h在1e-03到h=1e-05的范围内时，近似计算值很接近精确值。但之后当步长进一步减小时，误差反而急剧增大，最终出现完全错误的计算结果。原因是当h很小时，公式中的分子是非常接近的数值的差值。减法导致差值的有效数字的个数很少，相对误差很大，而且误差又因为除以很小的分母而放大了。在实际科学和工程问题中，很多输入数据带有误差，所以在进行类似的计算时选择适度小的h值(1e-03到1e-05范围)可以获得足够精确的结果。

Listing 4.6: 改变步长h的值，使用中心差分公式近似计算二阶导数

```
1  def diff2nd(f, x, h):
2      return (f(x+h) - 2*f(x) + f(x-h)) / (h * h)
3
4  from math import exp, sin
5  def g(x): return exp(x*sin(x))
6
7  hs = []
8  for k in range(1, 9):
9      h = 10**(-k); hs.append(h)
10     print (('%-10.0e') % h, end='␣')
11 print ()
12 x0 = 1.4
13 for h in hs:
14     d2g = diff2nd(g, x0, h)
15     print (('%-10.6f') % d2g, end='␣')
16 print ()
17
18 from sympy import Symbol, diff, lambdify, exp, sin
19 sym_x = Symbol('x'); sym_gx = g(sym_x)
20 dgdx = diff(sym_gx, sym_x); d2gdx2 = diff(dgdx, sym_x)
21 print (d2gdx2)
22 print ("Exact␣value␣=␣%10.6f" % lambdify([sym_x], d2gdx2)(x0))
```

Listing 4.7: 程序4.6的输出结果

```
1e-01     1e-02     1e-03     1e-04     1e-05     1e-06     1e-07     1e-08
1.754065  1.815320  1.815936  1.815942  1.815943  1.815437  1.687539  -4.440892
(-x*sin(x) + 2*cos(x))*exp(x*sin(x)) + (x*cos(x) + sin(x))**2*exp(x*sin(x))
Exact value =  1.815942
```

如果一个函数在定义以后只使用一次且函数体可以写成一个表达式，则可以使用Lambda函数语法定义一个匿名函数。其一般形式为：
g = lambda 形参列表: 函数体表达式

例如程序4.8中定义的函数map将函数f作用于列表s中的每条数据，用结果替代原来的数据，最后返回s。第6行调用函数map时提供的第一个实参是一个Lambda函数，它的功能是对于形参x返回x+1。第7行的Lambda函数对于形参x返回x*x-1。

Listing 4.8: **Lambda函数**

```
1 def map(f, s):
2     for i in range(len(s)): s[i] = f(s[i])
3     return s
4
5 a = [1, 3, 5, 7, 9]
6 print (map(lambda x: x+1, a)) # [2, 4, 6, 8, 10]
7 print (map(lambda x: x*x-1, a)) # [3, 15, 35, 63, 99]
```

4.5 递归

递归就是一个函数调用自己。当要求解的问题满足以下三个条件时，递归是有效的解决方法。

1. 原问题可以分解为一个或多个结构类似但规模更小的子问题。
2. 子问题的规模足够小时可以直接求解，称为递归的终结条件。
3. 原问题的解可由子问题的解合并而成。

用递归方法解决问题的过程是先通过分析提出问题的递归模型作为设计方案，再编写程序实现。

阶乘的定义本身就是一个递归模型，在程序4.8中由factorial函数实现。

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n * (n-1)! & \text{if } n > 1 \end{cases}$$

Listing 4.9: **计算阶乘的递归函数**

```
1 def factorial(n):
2     if n == 1:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 print (factorial(10)) # 3628800
```

设a和b表示两个正整数。若a>b，则易证a和b的公约数集合等于a-b和b的公约数集合，因此a和b的最大公约数等于a-b和b的最大公约数。若a=b，则a和b的最大公约数等于a。由此可总结出递归模型如下。

$$gcd(a,b) = \begin{cases} a & \text{if a=b} \\ gcd(a-b,b) & \text{if a>b} \\ gcd(a,b-a) & \text{if a<b} \end{cases}$$

程序4.10的gcd函数实现了求两个正整数的最大公约数。递归函数都可以转换成与其等价的迭代形式，例如gcd函数对应的迭代形式是程序3.13。

Listing 4.10: 计算字符串反转的递归函数

```
1 def gcd(a, b):
2     if a == b:
3         return a
4     elif a > b:
5         return gcd(a-b, b)
6     else:
7         return gcd(a, b-a)
8
9 print (gcd(156, 732)) # 12
```

字符串反转就是将原字符串中的字符的先后次序反转，例如"ABCDE"反转以后得到"EDCBA"。问题的分析过程如下。原字符串"ABCDE"可看成是两个字符串"ABCD"和"E"的连接。反转以后的字符串"EDCBA"可看成是两个字符串"E"和"DCBA"的连接。 "DCBA"是"ABCD"的反转，是原问题的子问题。"E"是"E"的反转，也是原问题的子问题。递归的终结条件是:由单个字符构成的字符串的反转就是原字符串。由此可总结出递归模型如下。

$$reverse(s) = \begin{cases} s & \text{if len}(s)= 1 \\ s[-1] + reverse(s[:-1]) & \text{if len}(s)> 1 \end{cases}$$

程序4.11的reverse函数实现了字符串反转。

Listing 4.11: 计算字符串反转的递归函数

```
1 def reverse(s):
2     if len(s) == 1:
3         return s
4     else:
5         return s[-1] + reverse(s[ : -1])
6
7 print (reverse("ABCDE")) # EDCBA
```

快速排序是一种著名的排序算法，以下用一个实例分析其求解过程。要排序的原始数据集是列表[3, 6, 2, 9, 7, 19, 1, 8],可看成是三个列表的连接: [2, 1], [3], 和[6, 9, 7, 19, 8]。排完序的结果是[1, 2, 3, 6, 7, 8, 9, 19],也可看成是三个列表的连接: [1, 2], [3], 和[6, 7, 8, 9, 19]。对[2, 1]进行排序可得[1, 2]，这是原问题的一个子问题。对[6, 9, 7, 19, 8]进行排序可得[6, 7, 8, 9, 19]，这也是原问题的一个子问题。由此可总结出递归模型如下。

$$sort(s) = \begin{cases} s & \text{if len}(s)<=1 \\ sort(\{i \in s | i < s[0]\}) + \{i \in s | i = s[0]\} + sort(\{i \in s | i > s[0]\}) & \text{if len}(s)>1 \end{cases}$$

程序4.12的sort函数实现了一个易于理解的快速排序算法，但并非这个算法的高性能实现。

Listing 4.12: 实现快速排序的递归函数

```
1 def sort(s):
2     if len(s) <= 1: return s
3     s1 = [i for i in s if i < s[0]]
4     s2 = [i for i in s if i > s[0]]
5     s0 = [i for i in s if i == s[0]]
6     return sort(s1) + s0 + sort(s2)
7
8 print (sort([3, 6, 2, 9, 7, 19, 1, 8])) # [1, 2, 3, 6, 7, 8, 9, 19]
```

4.6 创建和使用模块

模块是一个包含了若干函数和语句的文件，文件名是模块的名称加上".py"后缀。一个模块实现了某类功能，是规模较大程序的组成单位，易于重复利用代码。每个模块都有一个全局变量__name__。模块的使用方式有两种。

1. 模块作为一个独立的程序运行，此时变量__name__的值为'__main__'。
2. 被其他程序导入以后调用其中的函数，此时变量__name__的值为模块的名称。

以下通过一个计算银行存款的本金、年利率、存款天数和当前余额的实例说明创建和使用模块的方法。设A_0为本金，p为年利率，n为存款天数，A为当前余额。以下公式可从其中任意三个变量计算剩余的变量。

$$A = A_0(1 + \frac{p}{100 \times 360})^n \tag{4.2}$$

$$A_0 = A(1 + \frac{p}{100 \times 360})^{-n} \tag{4.3}$$

$$n = \ln(A/A_0)/\ln(1 + \frac{p}{100 \times 360}) \tag{4.4}$$

$$p = (100 \times 360)((A/A_0)^{\frac{1}{n}} - 1) \tag{4.5}$$

程序4.13列出了模块(interest.py)的代码。1-18行是模块的注释，解释了模块的功能和用法,其内容可以通过变量__doc__获取。23-33行的四个函数实现了以上四个公式。

直接运行模块的示例如下4.14。首先进入文件interest.py所在目录，然后运行文件并输入四个参数中的任意三个。这些参数的默认值为-1，因此用户未输入的参数值为-1，由此确定调用哪个函数计算并输出该参数的值。

第68行的条件如果成立，则模块作为一个独立的程序运行，sys.argv是一个记录了用户在命令行输入的所有参数的列表(包括模块名称在内)。第69行的条件如果成立，则用户未输入参数，此时程序输出使用说明。第71行的条件如果成立，则用户输入了一个参数'-h'需要帮助信息，此时程序输出使用说明。第73行的条件如果成立，则使用已知正确结果的测试数据调用函数test_all_functions对四个实现公式的函数进行测试。软件开发完成以后，难免会有各种错误。在软件交付使用前应通过充分测试尽可能查找和改正错误。第75行else后面的语句块导入和使用argparse模块解析用户输入的所有参数，然后调用函数compute_missing_parameter计算未输入的参数。这里的输入格式要求每个参数的数值前面都要用"–参数名称"指定对应的参数名称，因此参数的顺序无关紧要。

Listing 4.13: **计算银行存款相关参数的模块interest.py**

```
1  """
2  Module for computing with interest rates.
3  Symbols: A is present amount, A0 is initial amount,
4  n counts days, and p is the interest rate per year.
5
6  Given three of these parameters, the fourth can be
7  computed as follows:
8
9      A  = present_amount(A0, p, n)
10     A0 = initial_amount(A, p, n)
11     n  = days(A0, A, p)
12     p  = annual_rate(A0, A, n)
13
14 For example, given A0, p and n, the module outputs A.
15
16 >>> run interest.py --A0 1000 --p 3 --n 365
17 A = 1030.8826730421133
18 """
19
```

```
20 import sys
21 from math import log as ln
22
23 def present_amount(A0, p, n):
24     return A0*(1 + p/(100*360))**n
25
26 def initial_amount(A, p, n):
27     return A*(1 + p/(100*360))**(-n)
28
29 def days(A0, A, p):
30     return ln(A/A0)/ln(1 + p/(100*360))
31
32 def annual_rate(A0, A, n):
33     return 100*360*((A/A0)**(1.0/n) - 1)
34
35 def compute_missing_parameter():
36     if A<0:
37         print ('A␣=', present_amount(A0, p, n))
38     elif A0<0:
39         print ('A0␣=', initial_amount(A, p, n))
40     elif n<0:
41         print ('n␣=', days(A0, A, p))
42     elif p<0:
43         print ('p␣=', annual_rate(A0, A, n))
44
45 def test_all_functions():
46     # Compatible values
47     A = 2.2133983053266699; A0 = 2.0; p = 5; n = 730
48     # Given three of these, compute the remaining one
49     # and compare with the correct value (in parenthesis)
50     A_computed = present_amount(A0, p, n)
51     A0_computed = initial_amount(A, p, n)
52     n_computed = days(A0, A, p)
53     p_computed = annual_rate(A0, A, n)
```

```
54
55      def float_eq(a, b, tolerance=1E-12):
56          """Return True if a == b within the tolerance."""
57          return abs(a - b) < tolerance
58
59      success = float_eq(A_computed, A) and \
60                float_eq(A0_computed, A0) and \
61                float_eq(p_computed, p) and \
62                float_eq(n_computed, n)
63      msg = """Computations failed (correct answers in parenthesis):
64               A=%g (%g) A0=%g (%.1f) n=%d (%d) p=%g (%.1f)""" \
65          % (A_computed, A, A0_computed, A0, n_computed, n, p_computed, p)
66      assert success, msg
67
68  if __name__ == '__main__':
69      if len(sys.argv) == 1:
70          print (__doc__)
71      elif len(sys.argv) == 2 and sys.argv[1] == '-h':
72          print (__doc__)
73      elif len(sys.argv) == 2 and sys.argv[1] == 'test':
74          test_all_functions()
75      else:
76          import argparse
77          parser = argparse.ArgumentParser()
78          parser.add_argument('--A0', type=float, default=-1)
79          parser.add_argument('--A', type=float, default=-1)
80          parser.add_argument('--p', type=float, default=-1)
81          parser.add_argument('--n', type=float, default=-1)
82          args = parser.parse_args()
83          A0 = args.A0; A = args.A; p = args.p; n = args.n
84          compute_missing_parameter()
```

Listing 4.14: **直接运行interest.py**

```
In[1]: cd D:\Python\src
```

```
Out[1]: D:\Python\src
In[2]: run interest.py --A0 1000 --p 3 --n 365
Out[2]: A = 1030.8826730421133
In[3]: run interest.py --A0 1000 --A 1030.88 --n 365
Out[3]: p = 2.9997442339233515
In[4]: run interest.py --A0 1000 --A 1030.88 --p 3
Out[4]: n = 364.96888308991
In[5]: run interest.py --n 365 --A 1030.88 --p 3
Out[5]: A0 = 999.9974070355598
```

模块除了可以作为一个独立的程序运行，也可以被其他程序导入以后调用其中的函数。如果使用模块的程序和模块文件在同一个目录下时，使用import语句导入模块即可使用。例如运行程序doubling.py(4.15)的结果为4.16。

Listing 4.15: **doubling.py导入和使用interest模块的days函数**

```
1 from interest import days
2 # HOW MANY DAYS DOES IT TAKE TO DOUBLE AN AMOUNT WHEN THE INTEREST
     RATE IS P=2,3,...8?
3 for p in range(2, 9):
4     years = days(1, 2, p)/365.0
5     print ('p=%d%% implies %.1f years to double the amount' \
6         % (p, years))
```

Listing 4.16: **doubling.py的运行结果**

```
In[1]: run D:/Python/src/doubling.py
p=2% implies 34.2 years to double the amount
p=3% implies 22.8 years to double the amount
p=4% implies 17.1 years to double the amount
...
```

如果使用模块的程序和模块文件不在同一个目录下时，使用import语句导入模块会报错。此时需要将模块所在目录加入到列表sys.path中，然后可以导入和使用模块。

Listing 4.17: **将模块所在目录加入到列表sys.path中**

```
1 In[1]: run D:\Python\c\doubling.py
2 Out[1]: ... ModuleNotFoundError: No module named 'interest'
3 In[2]: import sys; sys.path.insert(0, 'D:\Python\src')
```

```
In[3]: run D:\Python\c\doubling.py
p=2% implies 34.2 years to double the amount
p=3% implies 22.8 years to double the amount
p=4% implies 17.1 years to double the amount
...
```

习题

1. 仿照程序4.13创建一个模块解决1.3节的输出日历问题。定义六个函数。第一个函数判断某个年份是否是闰年。第二个函数计算指定的某年y的一月一日是星期几。第三个函数计算指定的某年某月有多少天。第四个函数计算指定的某年某月的一日是星期几。第五个函数根据年和月输出日历。第六个函数测试前四个函数。

2. 编写一个递归函数实现对一个存储了一些整数的有序列表的二分查找。将要查找的数和列表中间的数进行比较。如果相等，则返回列表中间的数的索引值。如果大于，对列表的后半边进行递归查找。否则，对列表的前半边进行递归查找。递归的终结条件是什么？

3. 国际象棋中的皇后可以攻击同一行或者同一列或者斜线（左上左下右上右下四个方向）上的棋子。n皇后问题指如何在一个n行n列的国际象棋棋盘上的n个不同的方格里放置n个皇后，使得任意两个皇后互相之间不能处于攻击状态。每个方格的位置用(i, j)表示，其中i和j分别表示行号和列号，i和j的取值范围都是闭区间[0, n-1]。

编写一个递归函数NQueen求解n皇后问题。NQueen假定当前已经放置了row个皇后，它们的位置是(i, board[i])，其中0≤i≤row-1，且这些皇后互相之间不处于攻击状态。要完成的任务是在row到n-1行中的每一行选择一列放置一个皇后。这个任务的完成分为两个步骤：先在第row行选择一列放置一个皇后，且这个皇后不能被已有的皇后攻击；然后递归调用NQueen在row+1到n-1行中的每一行选择一列放置一个皇后。isAttacked函数用来检查目前打算放置在(row, col)位置的皇后是否被已有的位于(i, board[i])位置的皇后所攻击。printBoard函数用来输出一个正确的放置方案，其中'Q'表示皇后放置的位置。需要思考的问题是：什么情况下调用printBoard函数？

Listing 4.18: **递归求解N皇后问题**

```
def isAttacked(board, row, col):
    for i in range(0, row):
        if col == board[i] or abs(row - i) == abs(col - board[i]):
            return True
    return False

def NQueen(board, row):
```

```
8  ???
9
10 def printBoard(board):
11     global m, n
12     m += 1
13     print ('Solution␣%d␣' % m)
14     for col in board:
15         print('+␣' * col + 'Q␣' + '+␣' * (n - 1 - col))
16     print ()
17
18 n = 8; m = 0
19 board = [0] * n
20 NQueen(board, 0)
21
22 ''' one Solution
23 Solution 92
24 + + + + + + + Q
25 + + + Q + + + +
26 Q + + + + + + +
27 + + Q + + + + +
28 + + + + + Q + +
29 + Q + + + + + +
30 + + + + + + Q +
31 + + + + Q + + +
32 '''
```

第五章　类和继承

面向对象的软件设计和开发技术是当前软件开发的主流技术，使得软件更易维护和复用。规模较大的软件大多是基于面向对象技术开发，以类作为基本组成单位。

5.1　定义和使用类

在使用面向对象技术开发软件时，首先通过对软件需求的分析找到问题域中同一类的客观事物，称为对象。把对象共同的属性和运算封装在一起得到的程序单元就是类。类可作为一个独立单位进行开发和测试。

例如要解决的问题是进行二维平面上的点的运算，如平移、旋转等。将二维平面上的点视为对象，抽象出其共同的属性和运算，定义一个类表示点。点的属性包括 x坐标和y坐标。点的运算包括：给定坐标创建一个点、沿x轴平移、沿y轴平移、以另外一个点为中心旋转和计算与另外一个点之间的距离等。这些运算通过函数实现。定义在类内部的函数称为方法(method)。

程序5.1的第22行定义了一个类Point2D。类的定义由“class 类名(父类)”开始。第2行中的object是Python程序中所有自定义类的父类。3到5行的方法__init__称为构造方法，用来初始化新创建的对象的所有属性。在类的所有方法中出现的属性都需要使用“对象名.”进行限定。 self是一个特殊的对象名，表示当前对象。类中的所有方法的第一个形参都是self。构造方法的另外三个形参分别表示点的x和y坐标，以及点的名称(默认值为空串)。第4行和第5行分别用形参x和y初始化当前对象self的x坐标self.x和当前对象的y坐标self.y。第8行的move_x方法计算self沿x轴平移一段距离delta_x以后的新的x坐标。第10行的move_y方法计算self沿y轴平移一段距离delta_y以后的新的y坐标。 12到16行的方法rotate方法计算self以另外一个点p为轴旋转角度t以后的新坐标。 18到20行的distance方法计算self与另外一个点p之间的距离。22到26行的__str__方法返回self的字符串表示。

28到36行的语句演示了怎样从类创建对象和调用方法。调用一个对象所属类的构造方法的语法是“类名(实参列表)”，调用一个对象所属类的的方法的语法是“对象名.方法名(实参列表)”。第28行创建了一个点a并调用构造方法初始化它的坐标值和名称，然后输出这个点的字符串表示。第29行调用点a的move_x方法将点a沿x轴平移，然后输出其字符串表示。第30行将点a沿y轴平移以后输出其字符串表示。第31行创建了一个点b并调用构造方法初始化它的坐标值和名称，然后输出这个点的字符串表示。第32行计算并输出这两个点之间的距离。第33行将点b以点a为轴旋转90度。第35行将点a以

点b为轴旋转180度。

Listing 5.1: **Point2D类**

```
1  import math
2  class Point2D(object):
3      def __init__(self, x, y, name=''):
4          self.x = x
5          self.y = y
6          self.name = name
7
8      def move_x(self, delta_x): self.x += delta_x
9
10     def move_y(self, delta_y): self.y += delta_y
11
12     def rotate(self, p, t):
13         xr = self.x - p.x; yr = self.y - p.y
14         x1 = p.x + xr * math.cos(t) - yr * math.sin(t)
15         y1 = p.y + xr * math.sin(t) + yr * math.cos(t)
16         self.x = x1; self.y = y1;
17
18     def distance(self, p):
19         xr = self.x - p.x; yr = self.y - p.y
20         return math.sqrt(xr * xr + yr * yr)
21
22     def __str__(self):
23         if len(self.name) < 1:
24             return '(%g, %g)' % (self.x, self.y)
25         else:
26             return '%s: (%g, %g)' % (self.name, self.x, self.y)
27
28 a = Point2D(-5, 2, 'a'); print (a) # A: (-5, 2)
29 a.move_x(-1); print (a)          # A: (-6, 2)
30 a.move_y(2); print (a)           # A: (-6, 4)
31 b = Point2D(3, 4, 'b'); print (b) # B: (3, 4)
32 print ('The distance between a and b is %f' % a.distance(b))
```

```
33 # THE DISTANCE BETWEEN A AND B IS 9.000000
34 b.rotate(a, math.pi/2)
35 print (a); print (b)   # A: (-6, 4) B: (-6, 13)
36 a.rotate(b, math.pi)
37 print (a); print (b)   # A: (-6, 22) B: (-6, 13)
```

程序5.2定义了一个类Complex表示复数，并实现了复数的基本运算。7到11行定义了构造方法，虚部的默认值为0。 Python规定了类的一些特殊的一元和二元运算符方法名称，这些方法的名称都以“__”开始和结束。如果一个类定义了这些方法，则调用这些方法时可以在语法上简化，使之符合数学上的表达习惯。例如7到11行定义了__add__方法，则两个复数对象被加数u和加数v进行加法运算的表达式可以写成“u+v”，其实等同于“u.__add__(v)”。为了支持复数和整数或浮点数的加法，第9行判断加数other为float或int类型时将其转换为 Complex类型。第10行检查other必须具有real和imag这两个属性。如果缺少至少一个，则无法进行复数的加法运算，因此第11行抛出运行时错误。第12行按照复数的加法规则计算结果。14到15行定义的__radd__方法是为了支持被加数为float或int类型的情形，例如“3+u”。由于加法运算是可交换的，即“3+u = u+3”，所以返回将被加数和加数交换以后调用__add__方法的运算结果。

7到11行定义了__sub__方法进行减法运算。第14行定义的__rsub__方法是为了支持被减数为float或int类型的情形，例如“3-u”。由于减法运算是不可交换的，所以不能仿照__radd__方法的做法，而是需要像__sub__那样对被减数做必要的转换和判断之后按照复数的减法规则计算结果。29到33行定义了乘法运算，和加法类似。35到47行定义了除法运算，和减法类似。

第49行定义的__abs__方法计算复数的模。53到54行定义了相等运算符。考虑到浮点数的误差，如果两个浮点数的差的绝对值小于一个预先定义的很小的常数则认定为相等。第56行定义了不等运算符作为相等运算符的否定。第58行定义了self的字符串表示。第60行定义了__repr__方法，可将self转换为一个字符串，它是self所包含的内容的完整表示。由于未定义乘方运算，第63行抛出运行时错误。由于复数之间无法进行大小的比较，66到69行定义了这些运算符是非法的。

Listing 5.2: **Complex类**

```
1 from math import sqrt
2 tol = 1e-15
3
4 class Complex(object):
5     def __init__(self, real, imag=0.0):
6         self.real = real; self.imag = imag
7
8     def __add__(self, other):
9         if isinstance(other, (float,int)): other = Complex(other)
```

```
10          elif not (hasattr(other, 'real') and hasattr(other, 'imag')):
11              raise TypeError('other must have real and imag attr.')
12          return Complex(self.real + other.real, self.imag + other.imag)
13
14      def __radd__(self, other): # DEFINES OTHER + SELF
15          return self.__add__(other)
16
17      def __sub__(self, other):
18          if isinstance(other, (float,int)): other = Complex(other)
19          elif not (hasattr(other, 'real') and hasattr(other, 'imag')):
20              raise TypeError('other must have real and imag attr.')
21          return Complex(self.real - other.real, self.imag - other.imag)
22
23      def __rsub__(self, other): # DEFINES OTHER - SELF
24          if isinstance(other, (float,int)): other = Complex(other)
25          elif not (hasattr(other, 'real') and hasattr(other, 'imag')):
26              raise TypeError('other must have real and imag attr.')
27          return other.__sub__(self)
28
29      def __mul__(self, other):
30          return Complex(self.real*other.real - self.imag*other.imag,
31                         self.imag*other.real + self.real*other.imag)
32      def __rmul__(self, other): # DEFINES OTHER + SELF
33          return self.__mul__(other)
34
35      def __truediv__(self, other):
36          if isinstance(other, (float,int)): other = Complex(other)
37          elif not (hasattr(other, 'real') and hasattr(other, 'imag')):
38              raise TypeError('other must have real and imag attr.')
39          ar, ai, br, bi = self.real, self.imag, other.real, other.imag
40          r = float(br*br + bi*bi)
41          return Complex((ar*br+ai*bi)/r, (ai*br-ar*bi)/r)
42
43      def __rtruediv__(self, other): # DEFINES OTHER / SELF
```

```
44          if isinstance(other, (float,int)): other = Complex(other)
45          elif not (hasattr(other, 'real') and hasattr(other, 'imag')):
46              raise TypeError('other must have real and imag attr.')
47          return other.__truediv__(self)
48
49      def __abs__(self):
50          return sqrt(self.real*self.real + self.imag*self.imag)
51
52      def __neg__(self): return Complex(-self.real, -self.imag)
53
54      def __eq__(self, other):
55          return abs(self.real-other.real) < tol and abs(self.imag-other.
                imag) < tol
56
57      def __ne__(self, other): return not self.__eq__(other)
58
59      def __str__(self): return '(%g, %g)' % (self.real, self.imag)
60
61      def __repr__(self): return 'Complex' + str(self)
62
63      def __pow__(self, power):
64          raise NotImplementedError('pow is not yet implemented')
65
66      def _illegal(self, op):
67          print ('illegal operation "%s" for complex numbers' % op)
68
69      def __gt__(self, other): self._illegal('>')
70      def __ge__(self, other): self._illegal('>=')
71      def __lt__(self, other): self._illegal('<')
72      def __le__(self, other): self._illegal('<=')
73
74  u = Complex(2,-3); v = Complex(1, -4)
75  print (u + v, u - v, u * v,  u / v)
76  #     (3, -7) (1, 1) (-10, -11) (0.823529, 0.294118)
```

```
77 print (4 + v, 4 - v, 4 * v, 4 / v)
78 #     (5, -4) (3, 4) (4, -16) (0.235294, 0.941176)
79 print (u > v)
80 # ILLEGAL OPERATION ">" FOR COMPLEX NUMBERS
81 print (u ** 3)
82 # NotImplementedError: pow is not yet implemented
```

程序5.3定义了一个类Polynomial表示一元多项式，并实现了一些运算。2到5行定义了构造方法。多项式中的每一项的指数和其对应的系数存储在一个字典poly中。如果在运算结果中某一项的系数的绝对值小于一个预先定义的很小的常数，则认为系数等于零，该项消失。 7到10行定义了__call__方法，对于一个多项式p(x)在x=t时求值可以写成“p(t)”，其实等同于“p.__call__(t)”。

12到19行定义了多项式的加法运算，13行先从self.poly复制一个副本sum，然后对于other中每一项的指数查找在sum中是否存在相同指数项，若存在则执行这两项的系数的加法，否则创建一个新的项并设置其系数为other中这一项的系数。第19行把计算结果sum作为实参调用构造方法创建一个新的多项式，在这个过程中删除系数等于零的项。 21到27行定义了多项式的乘法运算。第22行定义了一个空的字典sum。23到26行根据指数相加和系数相乘的规则执行乘法。第27行把计算结果sum作为实参调用构造方法创建一个新的多项式。29到41行定义了多项式的字符串表示，这里判断和处理了多种特殊情形使得输出结果符合数学上的表达习惯。

Listing 5.3: **Polynomial类**

```
1  class Polynomial(object):
2      def __init__(self, poly):
3          self.poly = {}
4          for power in poly:
5              if abs(poly[power]) > tol: self.poly[power] = poly[power]
6
7      def __call__(self, x):
8          value = 0.0
9          for power in self.poly: value += self.poly[power]*x**power
10         return value
11
12     def __add__(self, other):
13         sum = self.poly.copy() # PRINT (ID(SUM), ID(SELF.POLY))
14         for power in other.poly:
15             if power in sum:
16                 sum[power] += other.poly[power]
```

```
17            else:
18                sum[power] = other.poly[power]
19        return Polynomial(sum)
20
21    def __mul__(self, other):
22        sum = {}
23        for self_power in self.poly:
24            for other_power in other.poly:
25                sum[self_power + other_power] = \
26                    self.poly[self_power] * other.poly[other_power]
27        return Polynomial(sum)
28
29    def __str__(self):
30        s = ''
31        for power in sorted(self.poly):
32            s += ' + %g*x^%d' % (self.poly[power], power)
33        s = s.replace('+ -', '- ')
34        s = s.replace('x^0', '1')
35        s = s.replace(' 1*', ' ')
36        s = s.replace('x^1 ', 'x ')
37        # S = S.REPLACE('X^1', 'X') REPLACES X^100 BY X^00
38        if s[0:3] == ' + ': # REMOVE INITIAL +
39            s = s[3:]
40        if s[0:3] == ' - ': # FIX SPACES FOR INITIAL -
41            s = '-' + s[3:]
42        return s
43
44 tol = 1E-15
45 p1 = Polynomial({0: -1, 2: 1, 7: 3})
46 print (p1)     #  -1 + x^2 + 3*x^7
47 p2 = Polynomial({0: 1, 2: -1, 5: -2, 3: 4})
48 print (p2)     #  1 - x^2 + 4*x^3 - 2*x^5
49 p3 = p1 + p2
50 print (p3)     #  4*x^3 - 2*x^5 + 3*x^7
```

```
51 p4 = p1 * p2
52 print (p4)
53 #  -1 + x^2 - 4*x^3 - x^4 + 4*x^5 + 3*x^7 - 3*x^9 + 12*x^10 - 6*x^12
54 print (p4(5)) #  -1353269851.0
```

5.2 继承

利用有限差分可以近似计算函数$f(x)$的一阶导数。根据泰勒公式可将$f(x)$在x的邻域展开如下:

$$f(x-2h) = f(x) - 2hf'(x) + \frac{4h^2f''(x)}{2} - \frac{8h^3f'''(x)}{6} + \frac{16h^4f''''(x)}{24} - \frac{32h^5f'''''(x)}{120} + \cdots \quad (5.1)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2f''(x)}{2} - \frac{h^3f'''(x)}{6} + \frac{h^4f''''(x)}{24} - \frac{h^5f'''''(x)}{120} + \cdots \quad (5.2)$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2f''(x)}{2} + \frac{h^3f'''(x)}{6} + \frac{h^4f''''(x)}{24} + \frac{h^5f'''''(x)}{120} + \cdots \quad (5.3)$$

$$f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2f''(x)}{2} + \frac{8h^3f'''(x)}{6} + \frac{16h^4f''''(x)}{24} + \frac{32h^5f'''''(x)}{120} + \cdots \quad (5.4)$$

由此可以推导出以下这些按照精确度从低到高次序列出的计算数值一阶导数的有限差分公式。这些公式依次称为一阶向前差分、一阶向后差分、二阶中心差分和四阶中心差分。

$$f'(x) = \frac{f(x+h)-f(x)}{h} + O(h) \quad (5.5)$$

$$f'(x) = \frac{f(x)-f(x-h)}{h} + O(h) \quad (5.6)$$

$$f'(x) = \frac{f(x+h)-f(x-h)}{2h} + O(h^2) \quad (5.7)$$

$$f'(x) = \frac{4}{3}\frac{f(x+h)-f(x-h)}{2h} - \frac{1}{3}\frac{f(x+2h)-f(x-2h)}{4h} + O(h^4) \quad (5.8)$$

如果每个公式都用一个类实现，则这些类都有数据属性f和h，并且它们的构造方法是相同的。这就导致了大量重复代码。面向对象程序编程范式提供了继承机制，新类可从已有类获得属性和方法并进行扩展，实现了代码复用。新类称为子类或派生类。已有类称为父类或基类。对于每个公式，都需要比较其计算结果和精确结果的差别。可为这些公式类定义一个共同的父类Diff，其中包含了属性f和h，以及比较计算结果的方法。通过继承，这些公式类可以获得这些属性和方法。

程序5.4中1到11行定义了Diff类。2到5行定义了构造方法用以初始化所有属性。 dfdx_exact表示函数f的一阶导数的解析形式的函数，其默认值为None。如果调用构造方法时提供了dfdx_exact，则可计算微分的精确结果。7到11行定义了get_error方法计算数值结果和精确结果之间的相对误差(百分比)。 13到32行定义了对应这四个公式的四个类。这些类的定义的第一条语句中的“(Diff)”表示其父类是Diff。由于从父类继承了所有属性和构造方法，这些类只需实现__call__方法进行公式计算。37到51行的table方法基于输入函数f、自变量x、步长值列表h_values、公式类列表methods和函数f的一阶导数的解析形式的函数dfdx生成一个表格。表格中的每一项数据表示对于一个特定公式和特定步

长值，数值结果和精确结果之间的相对误差(百分比)。输出结果5.5表明这些有限差分公式的精确度符合理论预期的从低到高次序。

Listing 5.4: **数值微分类**

```
class Diff(object):
    def __init__(self, f, h=1E-5, dfdx_exact=None):
        self.f = f
        self.h = float(h)
        self.exact = dfdx_exact

    def get_error(self, x):
        if self.exact is not None:
            df_numerical = self(x)
            df_exact = self.exact(x)
            return 100 * abs( (df_exact - df_numerical) / df_exact )

class Forward1(Diff):
    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x+h) - f(x))/h

class Backward1(Diff):
    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x) - f(x-h))/h

class Central2(Diff):
    def __call__(self, x):
        f, h = self.f, self.h
        return (f(x+h) - f(x-h))/(2*h)

class Central4(Diff):
    def __call__(self, x):
        f, h = self.f, self.h
        return (4./3)*(f(x+h) - f(x-h)) /(2*h) - \
```

```
32              (1./3)*(f(x+2*h) - f(x-2*h))/(4*h)
33
34  def table(f, x, h_values, methods, dfdx=None):
35      print ('%-10s' % 'h', end='␣')
36      for h in h_values: print ('%-8.2e' % h, end='␣')
37      print()
38      for method in methods:
39          print ('%-10s' % method.__name__, end='␣')
40          for h in h_values:
41              if dfdx is not None:
42                  d = method(f, h, dfdx)
43                  output = d.get_error(x)
44              else:
45                  d = method(f, h)
46                  output = d(x)
47              print ('%-8.4f' % output, end='␣')
48          print()
49
50  import math
51  def g(x): return math.exp(x*math.sin(x))
52
53  import sympy as sym
54  sym_x = sym.Symbol('x')
55  sym_gx = sym.exp(sym_x*sym.sin(sym_x))
56  sym_dgdx = sym.diff(sym_gx, sym_x)
57  dgdx = sym.lambdify([sym_x], sym_dgdx)
58
59  table(f=g, x=-0.65, h_values=[10**(-k) for k in range(1, 9)],
60        methods=[Forward1, Central2, Central4], dfdx=dgdx)
```

Listing 5.5: 程序**5.4**的输出结果

```
h           1.00e-01 1.00e-02 1.00e-03 1.00e-04 1.00e-05 1.00e-06 1.00e-07 1.00e-08
Forward1    10.4974  1.0906   0.1095   0.0110   0.0011   0.0001   0.0000   0.0000
Central2    0.4611   0.0046   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
Central4    0.0080   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

子类可以对父类进行功能上的扩展，例如在子类中定义父类中没有的属性和方法。子类还可以重新定义从父类继承的方法，称为覆盖(overriding)。覆盖的规则是子类中定义的某个方法和父类中的某个方法在名称、形参列表和返回类型上都相同，但方法体不同。覆盖体现了子类和父类在功能上的差异。例如程序5.6中定义了一个父类Parent和它的三个子类：Child1、Child2和Child3。其中Child1和Child2都覆盖了Parent的方法method，而Child3则从Parent继承了方法method。从输出结果可以看出，如果子类覆盖了父类的方法，则子类对象调用的方法method是子类中重新定义的方法。如果子类未覆盖父类的方法，则子类对象调用的方法是从父类中继承的方法method。

Listing 5.6: **覆盖**

```
1  class Parent:          # DEFINE PARENT CLASS
2     def method(self):
3        print ('Calling␣method␣in␣class␣Parent')
4
5  class Child1(Parent): # DEFINE CHILD1 CLASS
6     def method(self):
7        print ('Calling␣method␣in␣class␣Child1')
8
9  class Child2(Parent): # DEFINE CHILD2 CLASS
10    def method(self):
11       print ('Calling␣method␣in␣class␣Child2')
12
13 class Child3(Parent): # DEFINE CHILD3 CLASS
14    def method2(self):
15       print ('Calling␣method2␣in␣class␣Child3')
16
17 c1 = Child1()         # INSTANCE OF CHILD1
18 c2 = Child2()         # INSTANCE OF CHILD2
19 c3 = Child3()         # INSTANCE OF CHILD3
20 p = Parent()          # INSTANCE OF PARENT
21 c1.method()           # CALLING METHOD IN CLASS CHILD1
22 c2.method()           # CALLING METHOD IN CLASS CHILD2
23 c3.method()           # CALLING METHOD IN CLASS PARENT
24 p.method()            # CALLING METHOD IN CLASS PARENT
```

习题

1. 函数$f(x)$在区间$[a,b]$上的定积分可用区间内选取的n个点x_i $(i=0,1,...,n-1)$上的函数值的加权和近似计算：

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} w_i f(x_i) \tag{5.9}$$

其中w_i是函数值$f(x_i)$的权重。常用的近似计算定积分的几种方法的区别体现在点的选取和权重上。

方法名称	点的选取和权重
中点(Midpoint)法	$x_i = a + \frac{h}{2} + ih, \quad w_i = h, \quad h = \frac{b-a}{n}$
梯形(Trapezoidal)法	$x_i = a + ih \quad for \quad i = 0,...,n-1, \quad h = \frac{b-a}{n-1},$ $w_0 = w_{n-1} = \frac{h}{2}, \quad w_i = h \quad for \quad i = 1,...,n-2$
辛普森(Simpson’ s)法 n必须是奇数 若输入的n是偶数，则执行$n = n+1$	$x_i = a + ih \quad for \quad i = 0,...,n-1, \quad h = \frac{b-a}{n-1},$ $w_0 = w_{n-1} = \frac{h}{3}, \quad w_i = \frac{2h}{3} \quad for \quad i = 2,4,...,n-3,$ $w_i = \frac{4h}{3} \quad for \quad i = 1,3,...,n-2$
高斯-勒让德(Gauss-Legendre)法 n必须是偶数 若输入的n是奇数，则执行$n = n+1$	$x_i = a + (i+\frac{1}{2})h - \frac{1}{\sqrt{3}}\frac{h}{2} \quad for \quad i = 0,2,...,n-2,$ $x_i = a + (i+\frac{1}{2})h + \frac{1}{\sqrt{3}}\frac{h}{2} \quad for \quad i = 1,3,...,n-1,$ $h = \frac{b-a}{n}, \quad w_i = h, \quad for \quad i = 0,1,...,n-1$

表 5.1: 近似计算定积分的几种方法

在以下程序中实现Integrator类的四个子类，分别对应表5.1中的四种方法。在每个子类中只需覆盖父类的 construct_method方法初始化父类的所有属性。test_Integrate()函数用已知的数据测试四种方法。

Listing 5.7: **近似计算定积分的几种方法**

```
1  import numpy as np
2  import math
3
4  class Integrator(object):
5      def __init__(self, a, b, n):
6          self.a, self.b, self.n = a, b, n
7          self.points, self.weights = self.construct_method()
8
9      def construct_method(self):
10         raise NotImplementedError('no␣rule␣in␣class␣%s' \
11                                   % self.__class__.__name__)
12
13     def integrate(self, f):
```

```
14          s = 0
15          for i in range(len(self.weights)):
16              s += self.weights[i]*f(self.points[i])
17          return s
18
19  class Midpoint(Integrator):
20  ???
21
22  class Trapezoidal(Integrator):
23  ???
24
25  class Simpson(Integrator):
26  ???
27
28  class GaussLegendre(Integrator): # HINT: USE NP.LINSPACE
29  ???
30
31  # A LINEAR FUNCTION WILL BE EXACTLY INTEGRATED BY ALL
32  # THE METHODS, SO SUCH AN F IS THE CANDIDATE FOR TESTING
33  # THE IMPLEMENTATIONS
34
35  def test_Integrate():
36      """CHECK THAT LINEAR FUNCTIONS ARE INTEGRATED EXACTLY."""
37      def f(x): return x + 2
38      def F(x): return 0.5*x**2 + 2*x
39
40      a = 2; b = 3; n = 4   # TEST DATA
41      I_exact = F(b) - F(a)
42      tol = 1E-15
43
44      methods = [Midpoint, Trapezoidal, Simpson, GaussLegendre]
45      for method in methods:
46          integrator = method(a, b, n)
47
```

```
48          I = integrator.integrate(f)
49          assert abs(I_exact - I) < tol, 'bug in class %s' % method.
                __name__
50
51  if __name__ == '__main__':
52      test_Integrate()
```

第六章　NumPy数组和矩阵计算

Python的列表可以包含不同类型的数据，NumPy定义了由同类数据组成的多维数组 ndarray及其常用运算，是科学计算中最常用的数据类型。NumPy数组相对列表的优势是运算速度更快和占用内存更少。 ndarray是一个类，它的别名是array。它的主要属性包括ndim(维度数量)、shape(形状，即由每个维度的长度构成的元组) 、size(元素数量)和dtype(元素类型，可以是Python的内建类型，也可以是NumPy定义的类型，例如numpy.int32、numpy.int16和 numpy.float64等)。 NumPy的官方文档(https://numpy.org/doc/stable)详细说明了NumPy数组。

矩阵可以使用numpy.matrix类或numpy.array类表示。scipy.linalg模块定义了常用的矩阵计算函数(https://scipy.github.io/devdocs/index.html)。

6.1　创建数组

6.1.1　已有数据存储在其他类型的容器中

array函数可以从存储了数据的列表、元组或它们的嵌套创建数组，其类型取决于数据的类型，也可以使用dtype关键字实参指定类型。如果数据无法使用指定类型表示，可能会发生溢出或精度损失。

Listing 6.1: 创建数组

```
In[1]: import numpy as np
In[2]: a = np.array([2, 8, 64])
Out[2]: array([ 2, 8, 64])
In[3]: a.dtype,a.ndim,a.shape,a.size
Out[3]: (dtype('int32'), 1, (3,), 3)
In[4]: b = np.array([3.14, 2.71, 6.83, -8.34])
In[5]: b.dtype,b.ndim,b.shape,b.size
Out[5]: (dtype('float64'), 1, (4,), 4)
In[6]: c = np.array([(1, 2.4), (6, -3), (8, -5)])
In[7]: c.ndim,c.shape,c.size
Out[7]: (2, (3, 2), 6)
```

```
In[8]: d = np.array([95536, 2.71, 6, -8.34], dtype=np.int16); d
Out[8]: array([30000,  2,    6,   -8], dtype=int16))
```

6.1.2 没有数据但已知形状

np.zeros函数和np.ones函数创建指定形状的数组，并分布用0和1填充所有元素。zeros_like函数和ones_like函数创建和已有数组具有相同形状的数组，并分布用0和1填充所有元素。arange根据下界、上界和步长生成一个由等差数列组成的数组，包括下界但不包括上界。linspace根据下界、上界和数量生成一个由包含指定数量的数据的等差数列组成的数组，包括下界和上界。

Listing 6.2: **创建数组**

```
In[1]: a = np.zeros((2, 3)); a
Out[1]:
array([[0., 0., 0.],
       [0., 0., 0.]])
In[2]: np.ones((3, 2))
Out[2]:
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
In[3]: c = np.ones_like(a); c
Out[3]:
array([[1., 1., 1.],
       [1., 1., 1.]])
In[4]: np.arange(2, 30, 7)
Out[4]: array([ 2, 9, 16, 23])
In[5]: np.arange(0.2, 3.01, 0.7)
Out[5]: array([0.2, 0.9, 1.6, 2.3, 3. ])
In[6]: np.arange(6)
Out[6]: array([0, 1, 2, 3, 4, 5])
In[7]: np.linspace(0, 3, 7)
Out[7]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. ])
```

6.1.3 改变数组的形状

改变形状是指改变各维度的长度，但不改变组成数组的元素。 flatten函数从一个多维数组生成一维

数组。reshape从原数组生成一个指定形状的新数组。“h.T”生成数组h的转置。以上运算都返回一个新数组，而不会改变原数组。resize函数则改变原数组为指定的形状。

Listing 6.3: 改变形状

```
In[1]: h=np.arange(12).reshape(3,4) + 1; h
Out[1]:
array([[ 1, 2, 3, 4],
       [ 5, 6, 7, 8],
       [ 9, 10, 11, 12]])
In[2]: h.flatten()
Out[2]: array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
In[3]: h.reshape(2, 6)
Out[3]:
array([[ 1, 2, 3, 4, 5, 6],
       [ 7, 8, 9, 10, 11, 12]])
In[4]: h.T
Out[4]:
array([[ 1, 5, 9],
       [ 2, 6, 10],
       [ 3, 7, 11],
       [ 4, 8, 12]])
In[5]: h
Out[5]:
array([[ 1, 2, 3, 4],
       [ 5, 6, 7, 8],
       [ 9, 10, 11, 12]])
In[6]: h.resize(2, 6), h
Out[6]:
(None,
 array([[ 1, 2, 3, 4, 5, 6],
        [ 7, 8, 9, 10, 11, 12]]))
```

6.1.4 数组的堆叠

hstack和c_沿第二个维度将两个数组堆叠在一起形成新的数组，vstack和r_沿第一个维度将两个数组堆叠在一起形成新的数组，

Listing 6.4: **堆叠(Stacking)**

```
In[1]: a = np.arange(1, 7).reshape(2,3); a
Out[1]:
array([[1, 2, 3],
       [4, 5, 6]])
In[2]: b = np.arange(7, 13).reshape(2,3); b
Out[2]:
array([[ 7, 8, 9],
       [10, 11, 12]])
In[3]: np.hstack((a, b))
Out[3]:
array([[ 1, 2, 3, 7, 8, 9],
       [ 4, 5, 6, 10, 11, 12]])
In[4]: np.vstack((a, b))
Out[4]:
array([[ 1, 2, 3],
       [ 4, 5, 6],
       [ 7, 8, 9],
       [10, 11, 12]])
```

6.1.5 数组的分割

hsplit沿第二个维度将一个数组分割成为多个数组，可以指定一个整数表示均匀分割得到的数组的数量或指定一个元组表示分割的索引值。 vsplit沿第一个维度将一个数组分割成为多个数组，参数和hsplit类似。

Listing 6.5: **分割**

```
In[1]: c = np.arange(1, 25).reshape(2,12); c
Out[1]:
array([[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])
In[2]: np.hsplit(c, 4)
Out[2]:
[array([[ 1, 2, 3],
        [13, 14, 15]]),
```

```
 array([[ 4, 5, 6],
        [16, 17, 18]]),
 array([[ 7, 8, 9],
        [19, 20, 21]]),
 array([[10, 11, 12],
        [22, 23, 24]])]
In[3]: np.hsplit(c, (4, 7, 9))
Out[3]:
[array([[ 1, 2, 3, 4],
        [13, 14, 15, 16]]),
 array([[ 5, 6, 7],
        [17, 18, 19]]),
 array([[ 8, 9],
        [20, 21]]),
 array([[10, 11, 12],
        [22, 23, 24]])]
In[4]: d = np.arange(1, 25).reshape(6,4) + 1; d
Out[4]:
array([[ 1, 2, 3, 4],
       [ 5, 6, 7, 8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
In[5]: np.vsplit(d, (2, 3, 5))
Out[5]:
[array([[1, 2, 3, 4],
        [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12]]),
 array([[13, 14, 15, 16],
        [17, 18, 19, 20]]),
 array([[21, 22, 23, 24]])]
```

6.2 数组的运算

6.2.1 基本运算

数组的一元运算(取反、乘方)和二元运算(加减乘除)对于每个元素分别进行。二元运算要求两个数组具有相同的形状。数组和单个数值之间的运算等同于数组的每个元素分别和数值之间进行运算。“+=”、“-=”、“*=”、“/=”和“**=”运算符不返回新的数组，而是用运算结果替代原数组。如果参与运算的多个数组的数据类型不同，则结果的类型为取值范围最大的类型。
两个二维数组之间进行矩阵乘法可通过“@”运算符或dot函数实现。
np.random.default_rng使用指定的种子创建一个随机数生成器，可用来初始化一个给定形状的随机数组。。如果不提供种子，则由操作系统自动生成一个种子。

Listing 6.6: 基本运算

```
In[1]: a = np.arange(6).reshape(2, 3); a
Out[1]:
array([[0, 1, 2],
       [3, 4, 5]])
In[2]: b = np.arange(2,18,3).reshape(2, 3); b
Out[2]:
array([[ 2, 5, 8],
       [11, 14, 17]])
In[3]: a+b, a-b, a*b, a/b, -a, -b+(a**np.e-0.818*b+6)**(-np.pi)
Out[3]:
(array([[ 2, 6, 10],
        [14, 18, 22]]),
 array([[ -2, -4, -6],
        [ -8, -10, -12]]),
 array([[ 0, 5, 16],
        [33, 56, 85]]),
 array([[0.       , 0.2      , 0.25     ],
        [0.27272727, 0.28571429, 0.29411765]]),
 array([[ 0, -1, -2],
        [-3, -4, -5]]),
 array([[ -1.99023341, -4.96511512, -7.99647626],
        [-10.99985895, -13.99998898, -16.99999851]]))
In[4]:c=b.reshape(3, 2); c
Out[4]:
```

```
array([[ 2, 5],
       [ 8, 11],
       [14, 17]])
In[5]: a@c, a.dot(c)
Out[5]:
(array([[ 36, 45],
        [108, 144]]),
 array([[ 36, 45],
        [108, 144]]))
In[6]: d=a*3+b; b -= a; d, b
Out[6]:
(array([[ 2, 8, 14],
        [20, 26, 32]]),
 array([[ 2, 4, 6],
        [ 8, 10, 12]]))
In[7]: rg = np.random.default_rng()
In[8]: e = rg.random((2, 3)); e
Out[8]:
array([[0.31724113, 0.28135186, 0.19364362],
       [0.70296297, 0.76202543, 0.95436982]])
In[9]: f = e + a - 2*b; f, f.dtype
Out[9]:
(array([[ -3.68275887, -8.71864814, -13.80635638],
        [-18.29703703, -23.23797457, -28.04563018]]),
 dtype('float64'))
```

6.2.2 函数运算

sum、min和max可返回一个数组包含的所有元素的总和、最大值和最小值。sort函数可对数组进行排序。对于二维数组，可通过axis关键字实参指定对每行或每列分别计算。NumPy提供了很多数学函数(sin、cos、exp)，这些函数可分别作用于数组的每个元素。输入函数名和问号可以获取该函数的详细说明。

Listing 6.7: 函数运算

```
In[1]: g = np.array([[2,6,5],[4,1,3]]); g
```

```
Out[1]: array([[2, 6, 5],
               [4, 1, 3]])
In[2]: g.sum(), g.max(), g.min()
Out[2]: (21, 6, 1)
In[3]: g.max(axis=0), g.max(axis=1)
Out[3]: (array([4, 6, 5]), array([6, 4]))
In[4]: g.min(axis=0), g.min(axis=1)
Out[4]: (array([2, 1, 3]), array([2, 1]))
In[5]: np.sort(g)             # SORT ALONG THE LAST AXIS
Out[5]: array([[2, 5, 6],
               [1, 3, 4]])
In[6]: np.sort(g, axis=None)  # SORT THE FLATTENED ARRAY
Out[6]: array([1, 2, 3, 4, 5, 6])
In[7]: np.sort(g, axis=0)     # SORT ALONG THE FIRST AXIS
Out[7]: array([[2, 1, 3],
               [4, 6, 5]])
In[8]: np.sort?
Out[8]: Signature: np.sort(a, axis=-1, kind=None, order=None)
Docstring:
Return a sorted copy of an array.

Parameters
----------
a : array_like
    Array to be sorted.
axis : int or None, optional
......
In[9]: np.sqrt(b) + np.exp(a - 5) * np.cos(e**1.3 - f)
Out[9]:
array([[1.40958124, 2.22018241, 2.83965649],
       [3.45077796, 3.87699024, 3.31766545]])
```

6.3 索引、切片和迭代

一维数组可以像列表一样进行索引、切片和迭代。

Listing 6.8: 一维数组的索引、切片和迭代

```
In[1]: a = np.arange(1, 16, 2)**2; a
Out[1]: array([ 1,  9,  25,  49,  81, 121, 169, 225], dtype=int32)
In[2]: a[3], a[1:7:2]
Out[2]: (49, array([ 9, 49, 121], dtype=int32))
In[3]: a[:6:3] = 361; a
Out[3]: array([361, 9,  25, 361, 81, 121, 169, 225], dtype=int32)
In[4]: a[::-1]
Out[4]: array([225, 169, 121, 81, 361, 25, 9, 361], dtype=int32)
In[5]: for i in a: print(np.sqrt(i), end=' ')
Out[6]: 19.0 3.0 5.0 19.0 9.0 11.0 13.0 15.0
```

多维数组的每个维度都有一个索引，这些索引用逗号分隔共同构成一个完整的索引元组。如果提供的索引的数量小于维度，则等同于将缺失的维度全部选择(即冒号":")。省略号("...")代表多个可省略的冒号。对于二维数组的迭代以行为单位，使用flat属性进行迭代则以元素为单位。

数组不仅可以用整数和切片进行索引，也可以用整数的数组或布尔值的数组进行索引。对于多维数组，可分别对每个维度索引。np.ix_函数使用整数数组对行和列进行索引。

Listing 6.9: 多维数组的索引、切片和迭代

```
In[1]: def f(x, y): return x * 4 + y + 1
In[2]: h = np.fromfunction(f, (3, 4), dtype=int); h
Out[2]:
array([[ 1, 2, 3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
In[3]: h[1, 2], h[0, 3], h[2, 2]
Out[3]: (7, 4, 11)
In[4]: h[1:3], h[1:3,], h[1:3,:], h[0]
Out[4]:
(array([[ 5, 6, 7,  8],
        [ 9, 10, 11, 12]]),
 array([[ 5, 6, 7,  8],
```

```
       [ 9, 10, 11, 12]]),
 array([[ 5, 6, 7, 8],
       [ 9, 10, 11, 12]]),
 array([1, 2, 3, 4]))
In[5]: h[:, 1:4:2], h[:, 3:1:-1], h[:, -2]
Out[5]:
(array([[ 2, 4],
       [ 6, 8],
       [10, 12]]),
 array([[ 4, 3],
       [ 8, 7],
       [12, 11]]),
 array([ 3, 7, 11]))
In[6]: for row in h: print(row)
Out[6]:
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
In[7]: for element in h.flat: print(element, end='␣')
Out[7]: 1 2 3 4 5 6 7 8 9 10 11 12
In[8]: h[np.ix\_([0,2], [1])] # ROWS 0 AND 2, THEN COLUMN 1
Out[8]:
array([[ 2],
       [10]])
In[9]: h[np.ix\_([0, 2], [0, 2])]
Out[9]:
array([[ 1, 3],
       [ 9, 11]])
In[10]: h[[0, 2], [0, 2]] # COLUMN 0 FROM ROW 0 AND COLUMN 2 FROM ROW
   2
Out[10]: array([ 1, 11])
In[11]: h[[0,2]]
Out[11]:
array([[ 1, 2, 3, 4],
```

```
       [ 9, 10, 11, 12]])
In[12]: h[:,[0,2]]
Out[12]:
array([[ 1, 3],
       [ 5, 7],
       [ 9, 11]])
In[13]: h[1:3, 0:4]
Out[13]:
array([[ 5, 6, 7, 8],
       [ 9, 10, 11, 12]])
In[14]: j = np.arange(24).reshape(3, 2, 4); j
Out[14]:
array([[[ 0, 1, 2, 3],
        [ 4, 5, 6, 7]],

       [[ 8, 9, 10, 11],
        [12, 13, 14, 15]],

       [[16, 17, 18, 19],
        [20, 21, 22, 23]]])
In[15]: j[2, ...]
Out[15]:
array([[16, 17, 18, 19],
       [20, 21, 22, 23]])
In[16]: j[:, 1:2, :]
Out[16]:
array([[[ 4, 5, 6, 7]],

       [[12, 13, 14, 15]],

       [[20, 21, 22, 23]]])
In[17]: j[..., 1:3]
Out[17]:
array([[[ 1, 2],
```

```
        [ 5,  6]],

       [[ 9, 10],
        [13, 14]],

       [[17, 18],
        [21, 22]]])
```

Listing 6.10: 用整数数组进行索引

```
In[1]: i = np.array([3, 2, 7, 3, 5]); a[i]
Out[1]: array([ 49, 25, 225, 49, 121], dtype=int32)
In[2]: j = np.array([[3, 2, 4], [1, 5, 6]]); a[j]
Out[2]:
array([[ 49, 25, 81],
       [  9, 121, 169]], dtype=int32)
In[3]: b = a.reshape(4,2); b
Out[3]:
array([[ 1,  9],
       [ 25, 49],
       [ 81, 121],
       [169, 225]], dtype=int32)
In[4]: b[np.array([2, 3, 1, 2])]
Out[4]:
array([[ 81, 121],
       [169, 225],
       [ 25, 49],
       [ 81, 121]], dtype=int32)
In[5]: b[np.array([[2, 3], [1, 2]])]
Out[5]:
array([[[ 81, 121],
        [169, 225]],

       [[ 25, 49],
        [ 81, 121]]], dtype=int32)
```

```
In[6]: i1 = np.array([[3, 2], # INDICES FOR THE FIRST DIMENSION
                      [2, 1]])
In[7]: i2 = np.array([[0, 1], # INDICES FOR THE SECOND DIMENSION
                      [1, 0]])
In[8]: b[i1, i2]
Out[8]:
array([[169, 121],
       [121,  25]], dtype=int32)
In[9]: b[i1, i2] = 36; a
Out[9]: array([ 1,  9, 36, 49, 81, 36, 36, 225], dtype=int32)
```

对于一个二维数组，argmax函数可以返回一个整数数组表示每列(axis=0)或每行(axis=1)的最大值的索引。用这个整数数组作为索引可以获取二维数组中每列或每行的最大值。

Listing 6.11: 整数数组作为索引

```
In[1]: data = np.cos(np.arange(103, 123)).reshape(5, 4); data
Out[1]:
array([[-0.78223089, -0.94686801, -0.24095905, 0.68648655],
       [ 0.98277958, 0.3755096 , -0.57700218, -0.99902081],
       [-0.50254432, 0.4559691 , 0.99526664, 0.61952061],
       [-0.32580981, -0.97159219, -0.7240972 , 0.18912942],
       [ 0.92847132, 0.81418097, -0.04866361, -0.86676709]])
In[2]: maxind0 = data.argmax(axis=0); maxind0
Out[2]: array([1, 4, 2, 0], dtype=int32)
In[3]: data_maxind0 = data[maxind0, range(data.shape[1])]; data_maxind0
Out[3]: array([0.98277958, 0.81418097, 0.99526664, 0.68648655])
In[2]: maxind1 = data.argmax(axis=1); maxind1
Out[2]: array([3, 0, 2, 3, 0], dtype=int32)
In[3]: data_maxind1 = data[range(data.shape[0]), maxind1]; data_maxind1
Out[3]: array([0.68648655, 0.98277958, 0.99526664, 0.18912942,
   0.92847132])
```

Listing 6.12: 布尔数组作为索引

```
In[1]: a = np.arange(1, 16, 2)**2; a
Out[1]: array([ 1,  9,  25,  49,  81, 121, 169, 225], dtype=int32)
```

```
In[2]: g = a > 50; g
Out[2]: array([False, False, False, False, False, False, False, False])
In[2]: a[g] = 0; a
Out[2]: array([ 1, 9, 36, 49, 0, 36, 36, 0], dtype=int32)
In[3]: b = a.reshape(2, 4); b
Out[3]:
array([[ 1,  9, 25, 49],
       [ 81, 121, 169, 225]], dtype=int32)
In[4]: i1 = np.array([False, True]); b[i1, :]
Out[4]: array([[ 81, 121, 169, 225]], dtype=int32)
In[5]: i2 = np.array([True, False, False, True]); b[:, i2]
Out[199]:
array([[ 1, 49],
       [ 81, 225]], dtype=int32)
```

6.4 复制和视图

在对数组进行运算时，有时数据会复制到一个新数组中，有时不发生复制。

从id函数返回值可知，赋值运算给已有数组创建一个别名，不会发生数据的复制。Python的函数调用对于可变实参是传引用而非传值，所以不发生数据的复制。

view方法从已有数组创建一个新的数组，可以理解为原数组的一个视图。新数组和已有数组共享数据，但可以有不同形状。从原数组切片得到的新数组也是原数组的一个视图。ravel函数从一个多维数组生成一个一维数组，也是原数组的一个视图。

copy方法将已有数组的数据复制到新创建的数组中，新数组和原数组不共享数据。copy方法的一个用途是复制数据以后可以用del释放原数组占用的内存空间。

Listing 6.13: **复制和视图**

```
In[1]: k = h
In[2]: k is h, id(k), id(h)
Out[2]: (True, 186428160, 186428160)
In[3]: def get_id(x): return id(x)
In[4]: get_id(k)
Out[4]: 186428160
```

```
In[5]: m = h.view()
In[6]: m is h, m.base is h, m.flags.owndata
Out[6]: (False, True, False)
In[7]: m.resize((2, 6)); h.shape
Out[7]: (3, 4)
In[8]: m[1, 3] = 16; m, h
Out[8]:
(array([[ 1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 16, 11, 12]]),
 array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 16, 11, 12]]))
In[9]: t = h[0:2, 1:3]; t
Out[9]:
array([[2, 3],
       [6, 7]])
In[10]: t[1, 0] = 20; h
Out[10]:
array([[ 1,  2,  3,  4],
       [ 5, 20,  7,  8],
       [ 9, 16, 11, 12]])
In[11]: v = h.copy()
In[12]: v is h, v.base is h, id(v), id(h)
Out[12]: (False, False, 186709872, 186427488)
In[13]: v[1, 1] = 36; v[1, 1], h[1, 1]
Out[13]: (36, 20)
In[14]: a = np.arange(1000000); b = a[:100].copy()
In[15]: del a # THE MEMORY OF ``A`` CAN BE RELEASED.
```

6.5 矩阵计算

np.mat(numpy.matrix的别名)和numpy.array这两个类都表示矩阵。”*”运算符对于前者是矩阵乘法，对于后者是两个矩阵的对应元素的乘法。 scipy.linalg模块中定义的矩阵计算函数对这两个类的对象都适用。

linalg.inv函数计算一个矩阵的逆矩阵。linalg.det函数计算一个矩阵的行列式。linalg.norm函数对于一个矩阵$\mathbf{A}$计算其Frobenius范数$\sqrt{trace(\mathbf{A}^H\mathbf{A})}$，对于一个向量$\mathbf{x}$计算其各分量的平方和的平方根$\sqrt{\sum_i |x_i|^2}$。 linalg.solve($\mathbf{A}$, $\mathbf{b}$)求解线性方程组$\mathbf{Ax} = \mathbf{b}$。
linalg.eig($\mathbf{A}$)返回一个元组，由一个向量和一个矩阵组成。向量的第i个分量是矩阵的第i个特征值，矩阵的第i个列向量则是第i个特征值对应的特征向量。

一个秩为r的$m \times n$的实矩阵$\mathbf{A}$的奇异值分解[TL2019]为$\mathbf{A} = \mathbf{U\Sigma V^T} = [\mathbf{u}_1, ..., \mathbf{u}_m]\mathbf{\Sigma}[\mathbf{v}_1, ..., \mathbf{v}_n]^T$，其中 $\mathbf{\Sigma} = \left[\begin{array}{c|c} diag(\sigma_1, ..., \sigma_r) & \mathbf{0}_{r,n-r} \\ \hline \mathbf{0}_{m-r,r} & \mathbf{0}_{m-r,n-r} \end{array}\right]$，奇异值为$\sigma_1, ..., \sigma_r$，奇异向量为$\mathbf{u}_1, ..., \mathbf{u}_m$和$\mathbf{v_1}, ..., \mathbf{v_n}$，且

$$\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i, \quad i = 1, ..., r, \quad \mathbf{A}\mathbf{v}_i = 0, \quad i = r+1, ..., n, \tag{6.1}$$

$$\mathbf{A}^T\mathbf{u}_i = \sigma_i\mathbf{v}_i, \quad i = 1, ..., r, \quad \mathbf{A}^T\mathbf{u}_i = 0, \quad i = r+1, ..., m. \tag{6.2}$$

linalg.svd($\mathbf{A}$)返回一个元组($\mathbf{U}$, $\mathbf{s}$, $\mathbf{V^T}$)，其中$\mathbf{s}$是由所有奇异值组成的向量。linalg.diagsvd函数返回$\mathbf{\Sigma}$。矩阵的秩可通过计数非零奇异值的个数获得。

Listing 6.14: **矩阵计算**

```
In[1]: import numpy as np
In[2]: A = np.mat('[4␣3;2␣1]'); A
Out[2]:matrix([[4, 3],
               [2, 1]])
In[3]: A.I  #  INVERSE OF A
Out[3]: matrix([[-0.5, 1.5],
                [ 1. , -2. ]])
In[4]: b = np.mat('[6␣5]'); b
Out[4]: matrix([[6, 5]])
In[5]: b.T  #  MATRIX TRANSPOSE OF B
Out[5]:
matrix([[6],
        [5]])
In[6]: A*b.T
Out[6]:
matrix([[39],
        [17]])
In[7]: A = np.array([[4,3],[2,1]]); A
Out[7]:
array([[4, 3],
```

```
       [2, 1]])
In[8]: from scipy import linalg; linalg.inv(A)
Out[8]:
array([[-0.5, 1.5],
       [ 1. , -2. ]])
In[9]: b = np.array([[6,5]]); b # 2D ARRAY
Out[9]: array([[6, 5]])
In[10]: b.T
Out[10]:
array([[6],
       [5]])
In[11]: A*b  #  NOT MATRIX MULTIPLICATION!
Out[11]:
array([[24, 15],
       [12,  5]])
In[12]: A.dot(b.T) # MATRIX MULTIPLICATION
Out[12]:
array([[39],
       [17]])
In[13]: b = np.array([6,5]); b # 1D ARRAY
Out[13]: array([6, 5])
In[14]: b.T  #  NOT MATRIX TRANSPOSE!
Out[14]: array([6, 5])
In[15]: A.dot(b)
Out[15]: array([39, 17])
In[16]: A.dot(linalg.inv(A)
Out[16]: array([[1., 0.],
                [0., 1.]])
In[17]: linalg.det(A)
Out[17]: -2.0
In[18]: linalg.norm(A), linalg.norm(b)
Out[18]: (5.477225575051661, 7.810249675906654)
In[19]: x = np.linalg.solve(A, b); x
Out[19]: array([ 4.5, -4. ])
```

```
In[20]: A.dot(x) - b
Out[20]: array([0., 0.])
In[21]: la, v = linalg.eig(A)
In[22]: la
Out[22]: array([ 5.37228132+0.j, -0.37228132+0.j])
In[23]: v
Out[23]:
array([[ 0.90937671, -0.56576746],
       [ 0.41597356, 0.82456484]])
In[24]: A.dot(v[:, 0]) - la[0] * v[:, 0]
Out[24]: array([0.+0.j, 0.+0.j])
In[25]: np.sum(abs(v**2), axis=0)
Out[25]: array([1., 1.])
In[26]: A = np.array([[2,3,5],[7,9,11]])
In[27]: U,s,V = linalg.svd(A); s
Out[27]: array([16.96707058, 1.05759909])
In[28]: m, n = A.shape; S = linalg.diagsvd(s, m, n); S
Out[28]:
array([[16.96707058, 0.        , 0.        ],
       [ 0.        , 1.05759909, 0.        ]])
In[29]: U.dot(S.dot(V))
Out[29]:
array([[ 2., 3., 5.],
       [ 7., 9., 11.]])
In[30]: tol = 1E-10; (abs(s) > tol).sum() # RANK
Out[30]: 2
```

习题

1. 生成一个由实数组成的秩为4的4行4列的矩阵$\mathbf{A}$和一个由实数组成的包含4个元素的列向量$\mathbf{b}$。求解线性方程组$\mathbf{Ax} = \mathbf{b}$。计算$\mathbf{A}$的转置、行列式、秩、逆矩阵、特征值、特征向量和奇异值分解。

2. 生成一个由实数组成的秩为4的6行4列的实矩阵$\mathbf{A}$和一个由实数组成的包含6个元素的列向量$\mathbf{b}$。计算最小二乘解$(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$。计算$\mathbf{A}$的奇异值分解并验证6.1。

第七章　错误处理和读写文件

7.1　错误的分类

程序发生的错误可分为三大类：语法错误、逻辑错误和运行时错误。

- 语法错误是指程序违反了程序设计语言的语法规则，例如语句“if 3>2 print('3>2')”因冒号缺失导致语法解析器(parser)报错“SyntaxError: invalid syntax”。
- 逻辑错误是指程序可以正常运行，但结果不正确。例如程序7.1本意是对1到10的所有整数求和，但由于对range函数的错误理解，实际是对1到9的所有整数求和。

Listing 7.1: **for语句输出1到10之间的所有自然数的和**

```
1 sum = 0
2 for i in range(1, 10):
3     sum += i
4 print("The␣sum␣of␣1␣to␣10␣is␣%d" % sum) # THE SUM OF 1 TO 10 IS 45
```

- 运行时错误也称为异常(exception),是指程序在运行过程中发生了意外情形而无法继续运行。例如语句“a = 1/0 + 3”在运行过程中报错“ZeroDivisionError: division by zero”并终止。

语法错误和运行时错误都有明确的出错信息，容易改正。与前两种错误相比，改正逻辑错误的难度更大。

避免发生错误的基本方法列举如下。

1. 在编写较复杂程序之前应给出一个设计方案，把要完成的任务分解成为一些子任务，各子任务分别由一个模块完成。每个模块内部根据需要再进行功能分解，实现一些类和函数。这样使得整个程序有清晰合理的结构，容易修改和维护。
2. 对于每个类、函数和模块进行充分的测试。
3. 实现某一功能之前，先了解Python标准库和扩展库是否已经实现了该功能。如果有，则可以直接利用。这些库由专业软件开发人员实现，在正确性和性能上比自己写的程序更可靠。
4. 理解并遵从Python的编程规范(https://www.python.org/dev/peps/pep-0008/)。

7.2 调试

对于比较简单的程序，可以通过反复阅读程序和输出中间步骤的变量值查找逻辑错误。对于比较复杂的程序，上述方法的效率较低。更为有效的方法是设断点调试程序。

设断点调试依据的原理是基于命令式编程范式编写的程序的运行过程可以理解为状态转换的过程。状态包括程序中所有变量的值和正在运行的语句编号。每条语句的运行导致某些变量的值发生变化，可以理解为发生了一步状态转换。程序的输出结果是最终状态。从程序开始运行到结束经历了多次状态转换。如果程序结束时的输出结果有错，则错误必定发生在某一次状态转换中。在可能出错的每一条语句之前设断点。程序运行到断点停下以后，单步运行程序以观察每一步状态转换并与预期结果对照，这样最终一定会找到出错的语句。下面以实现高斯消去法的程序为例说明在Spyder中设断点调试的方法。

高斯消去法可用来求解线性方程组$\mathbf{Ax}=\mathbf{b}$。通过一系列的初等行变换，高斯消去法将增广矩阵$(\mathbf{A},\mathbf{b})$转变成一个上三角矩阵。设矩阵$\mathbf{A}$的行数和列数分别为m和n。程序的设计方案如下：

1. 外循环对第j列$(0\le j\le n-2)$运行，每次循环完成后第j列处于主对角线下方的元素变为0。
   (a) 内循环对第i行$(j+1\le i\le m-1)$运行，将第j行乘以$-a_{ij}/a_{jj}$的结果从第i行减去，目的是使得a_{ij}变为0。

程序7.2的运行结果(7.3)显示它对矩阵$\mathbf{A}$输出了正确的结果，但对$\mathbf{B}$输出的结果有错误并且报告除以零的警告。

Listing 7.2: 高斯消去法第一个版本

```
1  import numpy as np
2
3  def Gaussian_elimination_v1(A):
4      m, n = np.shape(A)
5      for j in range(n - 1):
6          for i in range(j + 1, m):
7              A[i, :] -= (A[i, j] / A[j, j]) * A[j, :]
8      return A
9
10 A = np.array([[2.0,3,5,7],[11,13,17,19],[23,29,31,37]])
11 print (Gaussian_elimination_v1(A))
12
13 B = np.array([[2.0,3,5,7],[12,18,17,19],[23,29,31,37]])
14 print (Gaussian_elimination_v1(B))
```

Listing 7.3: **程序7.2的运行结果**

```
[[ 2.          3.          5.          7.        ]
 [ 0.         -3.5       -10.5       -19.5       ]
 [ 0.          0.        -10.        -12.85714286]]
[[ 2.   3.   5.   7.]
 [ 0.   0. -13. -23.]
 [ nan  nan -inf -inf]]
C:\Users\user\.spyder-py3\temp.py:7: RuntimeWarning: divide by zero
    encountered in double_scalars
  A[i, :] -= (A[i, j] / A[j, j]) * A[j, :]
C:\Users\user\.spyder-py3\temp.py:7: RuntimeWarning: invalid value
    encountered in multiply
  A[i, :] -= (A[i, j] / A[j, j]) * A[j, :]
```

程序中只有第7行有除法运算。为了查明出错的原因，需要在第7行设置断点进行调试运行。由于程序对**B**的输出有错，所以先在第14行以**B**作为实参调用函数的语句处设置断点。首先用鼠标点击左边编辑窗口中的第14行使得光标在该行跳动，然后点击Debug菜单的菜单项"Set/Clear breakpoint"或使用快捷键F12。此时编辑窗口第14行行号的右边出现了一个红色的圆点，表示这一行已经设置断点。设置断点的操作类似电灯的开关，再进行一次以上操作则会取消断点。

和正常运行不同，调试运行会在所有设置的断点处停下，以便检查程序的中间状态。点击Debug菜单的菜单项"Debug"或使用快捷键Ctrl+F5使程序开始调试运行。此时第14行行号的右边出现一个蓝色箭头，表示已在这一行停下。右下角的控制台也显示了部分程序，并在第14行左边显示一个箭头。此时需要在第7行设置断点。为了使程序继续运行直到下一个断点(即第7行)，点击Debug菜单的菜单项"Continue"或使用快捷键Ctrl+F5。程序停下后，为了观察程序中变量的值，在控制台输入"A, i, j"，输出结果如图7.1所示。点击右上角窗口下边界处的"Variable explorer"可使右上角窗口自动显示所有变量的值。有时矩阵无法完整显示，可以用鼠标右键点击窗口，在弹出的菜单中选中"Resize rows to contents"或"Resize columns to contents"。

为了使程序单步运行，点击Debug菜单的菜单项"Step Into"或使用快捷键Ctrl+F11。此时编辑窗口第6行行号的右边出现一个蓝色箭头，表示已在这一行停下。继续上述操作，则程序又停在第7行。反复进行以上"Step Into"或"Continue"操作若干次，当$i = 2$且$j = 1$时程序的状态如图7.2所示。 此时作为除数的A[j,j]的值为0，导致了错误。已经找到出错原因后为了终止调试运行，点击Debug菜单的菜单项"Stop"或使用快捷键Ctrl+Shift+F12。找到以上出错处的另一种效率更高的方式是在第7行之前加上一个条件语句判断A[j,j]的值是否为0。在判断为是的分支处第8行设置断点(程序7.4)，

Listing 7.4: **设置条件断点**

```
1 import numpy as np
```

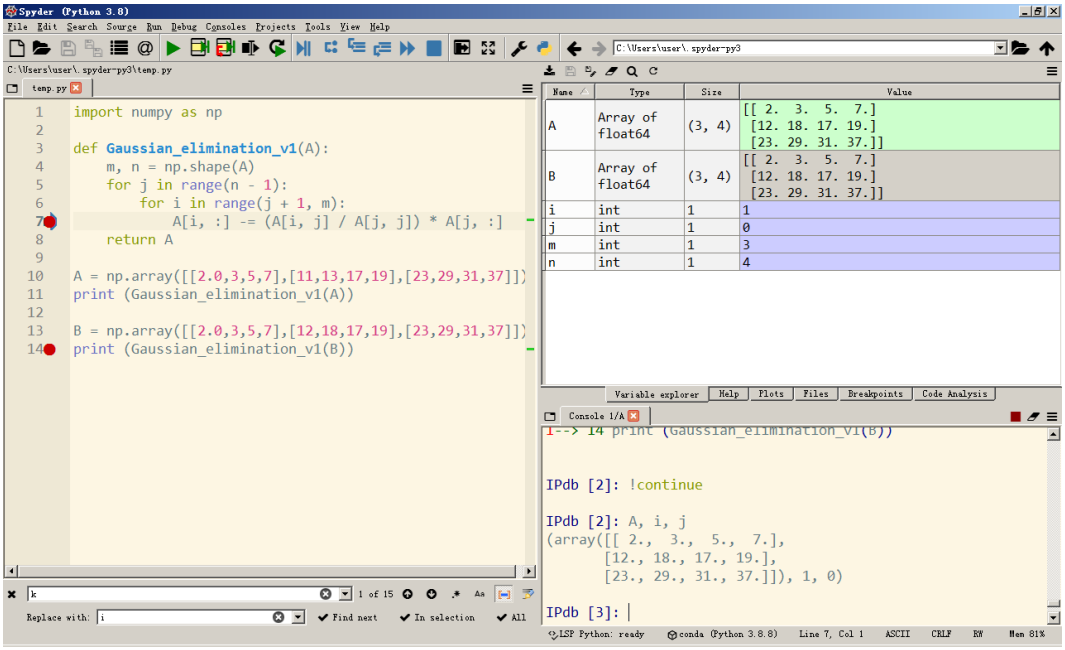

图 7.1: 设置断点进行调试

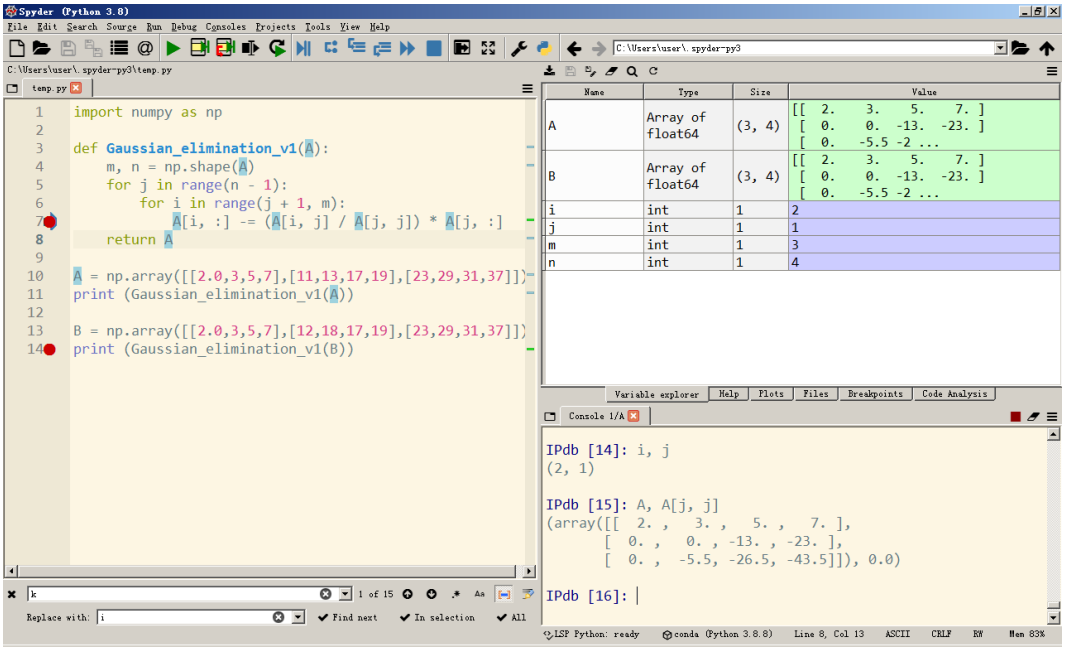

图 7.2: 设置断点进行调试

```
2
3  def Gaussian_elimination_v1(A):
4      m, n = np.shape(A)
5      for j in range(n - 1):
6          for i in range(j + 1, m):
7              if abs(A[j, j]) < 1e-10:
8                  i += 0
9              A[i, :] -= (A[i, j] / A[j, j]) * A[j, :]
10     return A
11 ...
```

程序7.5处理了A[j, j]值为0的情形。第7行在第j列的第i行及以下的元素构成的向量中找到绝对值最大的元素所在行的索引值p，这个索引值是相对于i的。第8行判断如果p大于0，则在第9行将第$p+i$行和第i行进行交换。第10行判断若 A[i, j]不为0再进行消去。返回的值i是矩阵的秩。

Listing 7.5: 高斯消去法第二个版本

```
1  import numpy as np
2
3  def Gaussian_elimination_v2(A, tol = 1e-10):
4      m, n = np.shape(A)
5      i = 0
6      for j in range(n):
7          p = np.argmax(abs(A[i:m, j]))
8          if p > 0:                    # SWAP ROWS
9              A[[i, p + i]] = A[[p + i, i]]
10         if abs(A[i, j]) > tol:       # J IS A PIVOT COLUMN
11             for r in range(i + 1, m):
12                 A[r, j:] -= (A[r, j] / A[i, j]) * A[i, j:]
13             i += 1
14             if i >= m: break
15     return A, i
```

7.3 异常处理

若程序中某一语句块在运行过程中可能发生运行时错误，可以利用if语句对每一种出错情形进行判断和处理。当出错情形较多时，这些if语句导致程序结构不清晰，难于理解。异常处理是比if语句更好

的错误处理方式，体现在将程序的主线和错误处理分离。异常处理为每种出错的情形定义一种异常，然后将可能出错的语句置于try语句块中。如果这些语句在运行时出错，系统会抛出异常，程序跳转到对应这种异常的except语句块中处理异常。

例如程序7.6要求用户在命令行输入两个整数作为参数，然后计算它们的最大公约数。如果用户输入的参数个数少于两个，或者某个参数不是整数，都会导致错误。 sys.argv是一个列表，存储了用户在命令行输入的所有字符串。索引值为0的字符串是程序的名称，其余字符串是用户输入的参数。如果输入的参数个数少于两个，则读取sys.argv[2]导致IndexError，14-16行的except语句块对这种异常进行处理，即输出具体的出错信息。如果某个参数不是整数，则int函数报错ValueError， 17-18行的except语句块对这种异常进行处理。如果用户输入了至少两个参数，且前两个都是整数，则程序不会发生异常，第13行运行完成以后跳转到第20行。

Listing 7.6: **异常处理**

```
1  def gcd(a, b):
2      """ COMPUTE THE GREATEST COMMON DIVISOR OF A AND B USING REPEATED
          SUBTRACTIONS """
3      while a != b:
4          if a > b:
5             a -= b;
6          else:
7              b -= a;
8      return a
9
10 import sys
11 try:
12     x = int(sys.argv[1])
13     y = int(sys.argv[2])
14 except IndexError:
15     print('Two␣arguments␣must␣be␣supplied␣on␣the␣command␣line')
16 except ValueError:
17     print('Each␣argument␣should␣be␣an␣integer.')
18
19 print ('The␣greatest␣common␣divisor␣of␣%d␣and␣%d␣is␣%d' %\
20        (x, y, gcd(x, y)))
```

Listing 7.7: **程序7.6的运行结果**

```
In[1]: run d:\python\src\gcd_ex.py 4
Out[1]: Two arguments must be supplied on the command line
In[2]: run d:\python\src\gcd_ex.py 4 6o
Out[2]: Each argument should be an integer.
In[3]: run d:\python\src\gcd_ex.py 4 60
Out[3]: The greatest common divisor of 4 and 60 is 4
```

Python语言定义了很多内建异常类，它们都是Exception类的直接或间接子类，构成一个继承层次结构。这些异常针对的错误类型包括算术运算、断言、输入输出和操作系统等。 except语句块中声明某一个异常类时，可以处理对应于该异常类或其子类的运行时错误。

用户在程序中可以使用这些内建异常类，也可以根据需要自定义异常类，自定义异常类以Exception作为父类。例如程序7.8中针对用户输入的整数为负数的出错情形定义了异常类InputRangeError。调用gcd时提供的两个实参中如果存在负数，循环不会终止。 InputRangeError类只有一个属性，即出错信息。第23行判断用户输入的两个整数中是否存在负数，若有则在第24行抛出InputRangeError异常。该异常对象被第30行的except语句块捕获，然后在第31行输出出错信息。

32到33行的else语句块是可选的。else语句块包含不发生任何异常时必须运行的语句，必须位于所有except语句块之后。 34到35行的finally语句块是可选的。finally语句块必须位于所有其他语句块之后。无论是否发生异常，finally语句块都会运行，主要用于回收系统资源等善后工作。finally语句块的语义规则如下：

1. 如果try语句块在运行过程中抛出了异常，且未被任何except语句块处理，则运行finally语句块后会重新抛出该异常。
2. 如果except语句块和else语句块在运行过程中抛出了异常，则运行finally语句块后会重新抛出该异常。
3. 如果try语句块中即将运行break、continue或return等跳转语句，则会先运行finally语句块再运行跳转语句。

Listing 7.8: **自定义异常类**

```
1 def gcd(a, b):
2     """ COMPUTE THE GREATEST COMMON DIVISOR OF A AND B USING REPEATED
          SUBTRACTIONS """
3     while a != b:
4         if a > b:
5             a -= b;
6         else:
```

```
7             b -= a;
8     return a
9
10 class InputRangeError(Exception):
11     """Raised when an input is not in suitable range
12
13     Attributes:
14        message -- explanation of suitable range
15     """
16     def __init__(self, message):
17         self.message = message
18
19 import sys
20 try:
21     x = int(sys.argv[1])
22     y = int(sys.argv[2])
23     if x < 0 or y < 0:
24         raise InputRangeError('Each integer should be positive')
25 except IndexError:
26     print('Two arguments must be supplied on the command line')
27 except ValueError:
28     print('Each argument should be an integer.')
29 except InputRangeError as ex:
30     print(ex.message)
31 else:
32     print ('The greatest common divisor of %d and %d is %d' %\
33            (x, y, gcd(x, y)))
34 finally:
35     print("executing finally clause")
```

Listing 7.9: 程序7.8的运行结果

```
In[1]: run d:\python\src\gcd_ex.py -48 126
Out[1]: Each integer should be positive
        executing finally clause
```

```
In[2]: run d:\python\src\gcd_ex.py 48 126
Out[2]: The greatest common divisor of 48 and 126 is 6
        executing finally clause
```

7.4 打开和关闭文件

如果程序需要输入大量数据，应从文件中读取。如果程序需要输出大量数据，应写入文件中。文件可分为两类：文本文件和二进制文件。文本文件存储采用特定编码方式(例如UTF-8、GBK等)编码的文字信息，以字符作为基本组成单位，可在文本编辑器中显示内容。二进制文件存储图片、视频、音频、可执行程序或其他格式的数据，以字节作为基本组成单位，在文本编辑器中显示为乱码。

读写文件之前，先要使用“f = open(filename, mode)”语句打开文件f，其中filename是文件名，mode是打开方式，可以是'r'(读)、'w'(写)或'a'(追加)。mode中如果有‘b'表示以二进制方式打开。读写一个文件f完成以后，需要使用“f.close() ”语句关闭文件。使用“with open(filename, mode) as f:”语句打开的文件f会自动关闭。

7.5 读写文本文件

从普通文本文件读取数据的基本方法是分析文件中的数据格式，采用合适的方法提取有效数据。文件rainfall.dat(7.11)记录了罗马每月的平均降水量。需要从中读取这些数据，然后计算最大值、最小值和平均值并写入文件rainfall_stat.dat中。文件中的有效数据在2-13行，每行的格式是“月份名称+空格+降水量”。

Listing 7.10: **文本文件rainfall.dat**

```
Average rainfall (in mm) in Rome: 1188 months between 1782 and 1970
Jan  81.2
Feb  63.2
Mar  70.3
Apr  55.7
May  53.0
Jun  36.4
Jul  17.5
Aug  27.5
Sep  60.9
Oct  117.7
Nov  111.0
Dec  97.9
Year 792.9
```

程序7.11的前9行定义了一个函数extract_data，它的形参为文件名。第3行忽略文件的第一行。第4行定义了一个空的字典rainfall。5到8行的循环从文件中每次读取一行。第6行判断当前行是否存在子

串'Year'。若存在，则已读完所有月份的数据，可以跳出循环。第7行将当前行以空格作为分隔符分解成为两部分(月份和对应的降水量)，然后存储在一个列表words中。第8行将从月份到降水量的映射添加到rainfall中。

主程序的第12行调用extract_data获取从文件中提取的字典。14到20行的循环计算最大值、最小值和平均值，并保存最大值和最小值的对应月份。22到25行输出以上信息到文件rainfall_stat.dat(7.12)中。

Listing 7.11: 读写文本文件

```
1  def extract_data(filename):
2      with open(filename, 'r') as infile:
3          infile.readline() # SKIP THE FIRST LINE
4          rainfall = {}
5          for line in infile:
6              if line.find('Year') >= 0: break
7              words = line.split() # WORDS[0]: MONTH, WORDS[1]: RAINFALL
8              rainfall[words[0]] = float(words[1])
9      return rainfall
10
11 import sys
12 rainfall = extract_data('D:/Python/src/rainfall.dat')
13 max = sys.float_info.min; min = sys.float_info.max; sum = 0
14 for month in rainfall.keys():
15     rainfall_month = rainfall[month]
16     sum += rainfall_month
17     if max < rainfall_month:
18         max = rainfall_month; max_month = month
19     if min > rainfall_month:
20         min = rainfall_month; min_month = month
21
22 with open('D:/Python/src/rainfall_stat.dat', 'w') as outfile:
23     outfile.write('The␣maximum␣rainfall␣of␣%.1f␣occurs␣in␣%s\n' %\
24                   (max, max_month))
25     outfile.write('The␣minimum␣rainfall␣of␣%.1f␣occurs␣in␣%s\n' %\
26                   (min, min_month))
```

```
27      outfile.write('The␣average␣rainfall␣is␣%.1f' % (sum / 12))
```

Listing 7.12: **rainfall_stat.dat**

```
The maximum rainfall of 117.7 occurs in Oct
The minimum rainfall of 17.5 occurs in Jul
The average rainfall is 66.0
```

7.6 读写CSV文件

CSV是一种简单的电子表格文件格式，其中的数据值之间用逗号分隔。办公软件(如Excel, LibreOffice Calc) 可以读入CSV文件并显示为电子表格。 Python的csv模块可将CSV文件中的数据读入一个嵌套列表中，也可以将一个嵌套列表写入CSV文件中。

程序7.13的2-3行从文件budget.csv(7.14)中读取数据并存入嵌套列表table中。其中的每条数据都是字符串。5-7行的双重循环将每条数据转换为float类型。第9行创建一个长度等于table的列数的新的列表row，且每条数据的初始值为0.0。 5-7行的双重循环计算每列数据的总和并存入row中。第9行将row追加到table中。第17行使用pprint模块的pprint函数输出table。19-23行将table写入文件budget2.csv(7.15)中。

Listing 7.13: **读写CSV文件**

```
1  import csv, pprint
2  with open('D:/Python/src/budget.csv', 'r') as infile:
3      table = [row for row in csv.reader(infile)]
4
5  for r in range(1,len(table)):
6      for c in range(1, len(table[0])):
7          table[r][c] = float(table[r][c])
8
9  row = [0.0]*len(table[0])
10 row[0] = 'sum'
11 for c in range(1, len(row)):
12     s = 0
13     for r in range(1, len(table)):
14         s += table[r][c]
15     row[c] = s
16 table.append(row)
```

```
17 pprint.pprint(table)
18
19 with open('D:/Python/src/budget2.csv', 'w', newline='') as outfile:
20     writer = csv.writer(outfile)
21     for row in table:
22         writer.writerow(row)
```

Listing 7.14: **budget.csv**

```
,"year␣1","year␣2","year␣3"
"person␣1",651000,651000,651000
"person␣2",1100500,950100,340000
"person␣3",740000,780000,800000
```

Listing 7.15: **budget2.csv**

```
,year 1,year 2,year 3
person 1,651000.0,651000.0,651000.0
person 2,1100500.0,950100.0,340000.0
person 3,740000.0,780000.0,800000.0
sum,2491500.0,2381100.0,1791000.0
```

7.7 读写JSON文件

JSON是“JavaScript Object Notation”的缩写，是一种常用的应用程序间数据交换格式。Python的json 模块可将结构化数据(字典、列表或它们的组合)转换成为一个JSON格式的字符串并写入一个文件中，也可以从一个JSON文件中读取结构化数据。

程序7.16的3-7行定义了一组通讯录数据contacts，它是一个字典的列表。9-10行打开一个文件contacts.json，并将contacts的内容以JSON格式写入其中(7.17)。12-13行从contacts.json中读取数据，然后由第15行输出(7.18)。

Listing 7.16: **读写JSON文件**

```
1 import json, pprint
2
3 contacts = [
4     {"Name":"Tom", "Phone":12345, "Address":"100␣Wall␣St."},
5     {"Name":"Jerry", "Phone":54321, "Address":"200␣Main␣St."},
6     {"Name":"Mary", "Phone":23415, "Address":"300␣Fifth␣Ave."}
```

```
7      ]
8
9  with open('D:/Python/src/contacts.json', 'w') as outfile:
10     json.dump(contacts, outfile)
11
12 with open('D:/Python/src/contacts.json', 'r') as infile:
13     x = json.load(infile)
14
15 pprint.pprint(x)
```

Listing 7.17: **contacts.json**

```
[{"Name": "Tom", "Phone": 12345, "Address": "100␣Wall␣St."}, {"Name": "Jerry", "Phone":
    54321, "Address": "200␣Main␣St."}, {"Name": "Mary", "Phone": 23415, "Address": "300␣Fifth
    ␣Ave."}]
```

Listing 7.18: **程序7.16的输出结果**

```
[{'Address': '100␣Wall␣St.', 'Name': 'Tom', 'Phone': 12345},
 {'Address': '200␣Main␣St.', 'Name': 'Jerry', 'Phone': 54321},
 {'Address': '300␣Fifth␣Ave.', 'Name': 'Mary', 'Phone': 23415}]
```

7.8 读写pickle文件

pickle是一种Python定义的数据格式。Python的pickle 模块可将结构化数据(字典、列表或它们的组合以及类的对象)转换成为一个字节流并写入一个二进制文件中，也可以从一个pickle文件中读取结构化数据。JSON格式适用于使用多种程序设计语言编写的程序之间的数据交换，而pickle格式只适用于使用Python语言编写的程序之间的数据交换。

程序7.19的3-7行定义了一组通讯录数据contacts，它是一个字典的列表。 9-10行打开一个文件contacts.pickle，并将contacts的内容以pickle格式写入其中。 12-13行从contacts.pickle中读取数据，然后由第15行输出(7.20)。

Listing 7.19: **读写pickle文件**

```
1  import pickle, pprint
2
3  contacts = [
4      {"Name":"Tom", "Phone":12345, "Address":"100␣Wall␣St."},
5      {"Name":"Jerry", "Phone":54321, "Address":"200␣Main␣St."},
```

```
6     {"Name":"Mary", "Phone":23415, "Address":"300␣Fifth␣Ave."}
7     ]
8
9  with open('D:/Python/src/contacts.pickle', 'wb') as outfile:
10    pickle.dump(contacts, outfile)
11
12 with open('D:/Python/src/contacts.pickle', 'rb') as infile:
13    x = pickle.load(infile)
14
15 pprint.pprint(x)
```

Listing 7.20: **程序7.19的输出结果**

```
[{'Address': '100␣Wall␣St.', 'Name': 'Tom', 'Phone': 12345},
 {'Address': '200␣Main␣St.', 'Name': 'Jerry', 'Phone': 54321},
 {'Address': '300␣Fifth␣Ave.', 'Name': 'Mary', 'Phone': 23415}]
```

7.9 读写NumPy数组的文件

np.savetxt函数可以把一个NumPy数组保存为一个文本文件。np.loadtxt函数可以从一个文本文件中读入一个数组。

np.save函数可以把一个数组保存为一个后缀为“npy”的二进制文件。np.load函数可以从一个后缀为“npy”的二进制文件中读入一个数组。 np.savez函数可以把多个数组保存为一个后缀为“npz”的二进制文件。np.savez_compressed可以把多个数组保存为一个后缀为“npz”的压缩二进制文件。

数组可以转换为Pandas模块的DataFrame格式，然后写入一个CSV文件中。从CSV文件可以读取DataFrame格式的数据，再转换为数组。

Listing 7.21: **读写NumPy数组的文件**

```
In[1]: import numpy as np; a = np.arange(1, 16, 2)**2; a
Out[1]: array([ 1,  9,  25,  49,  81, 121, 169, 225], dtype=int32)
In[2]: b = a.reshape(2, 4); b
Out[2]:
array([[ 1,  9,  25,  49],
       [ 81, 121, 169, 225]], dtype=int32)
In[3]: np.savetxt('D:/Python/dat/b.txt', b)
```

```
In[4]: c = np.loadtxt('D:/Python/dat/b.txt'); c
Out[4]:
array([[ 1.,  9., 25., 49.],
       [ 81., 121., 169., 225.]])
In[5]: np.save('D:/Python/dat/b.npy', b)
In[6]: c = np.load('D:/Python/dat/b.npy'); c
Out[6]:
array([[ 1,  9, 25, 49],
       [ 81, 121, 169, 225]])
In[7]: np.savez('D:/Python/dat/ab.npz', a, b)
In[8]: cd = np.load('D:/Python/dat/ab.npz')
In[9]: c = cd['arr_0']; c
Out[9]: array([ 1,  9, 25, 49, 81, 121, 169, 225])
In[10]: d = cd['arr_1']; d
Out[10]:
array([[ 1,  9, 25, 49],
       [ 81, 121, 169, 225]])
In[11]: import pandas as pd
In[12]: bf = pd.DataFrame(b); bf
Out[12]:
    0    1    2    3
0   1    9   25   49
1  81  121  169  225
In[13]: bf.to_csv('D:/Python/dat/pb.csv')
In[14]: df = pd.read_csv('D:/Python/dat/pb.csv'); df
Out[14]:
   Unnamed: 0  0    1    2    3
0           0  1    9   25   49
1           1  81  121  169  225

In[15]: d = np.array(df); d
Out[15]:
array([[ 0,  1,  9, 25, 49],
       [  1,  81, 121, 169, 225]], dtype=int64)
```

习题

1. 设断点单步运行本章的程序7.5、7.11和7.13，观察变量的值。

2. 运行本章的程序并理解运行结果。

参考文献

[HL2020] 汉斯.佩特.兰坦根 (挪), 科学计算基础编程——Python版, 清华大学出版社, 2020.

[RJ2019] Robert Johansson, Numerical Python, Springer, 2019.

[NL2019] Nicolas Lanchier, Stochastic Modeling, Springer, 2017.

[JD2011] Jay L. Devore, Kenneth N. Berk, Modern Mathematical Statistics with Applications, Springer, 2011.

[TL2019] Tom Lyche, Numerical Linear Algebra and Matrix Factorizations, Springer, 2019.