

Python科学计算基础

罗奇鸣 编著

2022年10月22日

本书是中国科学技术大学“Python科学计算基础”课程的教材，已获教务处教材出版项目立项，目前正在编写和修改过程中。

本书的版权归编著者所有，仅供选课同学在校内使用，禁止对外传播。

目录

第一章 绪论	1
1.1 科学计算	1
1.2 利用计算机解决问题	1
1.3 程序设计语言的分类	1
1.4 过程式编程范式	2
1.5 面向对象编程范式	3
1.6 学习Python语言的理由	3
1.7 Python语言的发展历史	4
1.8 Python语言的特点	4
1.9 Python科学计算环境	5
1.10 选课须知	5
1.10.1 选课要求和课程安排	5
1.10.2 学习方法	6
1.10.3 大作业	6
1.11 实验1: 安装和使用Python开发环境	8
第二章 内建数据类型及其运算	11
2.1 变量和类型	11
2.2 数值类型	12
2.2.1 int类型	12
2.2.2 float类型	13
2.2.3 complex类型	14
2.2.4 数值类型的内建函数	14
2.2.5 math模块和cmath模块	15
2.3 bool类型	17
2.4 序列类型	17
2.4.1 list类型(列表)和tuple类型(元组)	18
2.4.2 str类型(字符串)	20
2.5 set类型(集合)	23
2.6 dict类型(字典)	24

2.7	实验2：内建数据类型及其运算	25
第三章 分支和迭代		27
3.1	if语句和if-else表达式	27
3.2	for语句	29
3.3	while语句	31
3.4	推导式	32
3.5	实验3：分支和迭代	33
第四章 函数和模块		35
4.1	定义和调用函数	35
4.2	局部变量和全局变量	36
4.3	默认值形参和关键字实参	37
4.4	函数式编程(functional programming)	38
4.4.1	函数作为实参	38
4.4.2	函数作为返回值	40
4.5	递归	41
4.5.1	阶乘	41
4.5.2	最大公约数	41
4.5.3	字符串反转	42
4.5.4	快速排序	42
4.6	创建和使用模块	43
4.7	实验4：函数和模块	48
第五章 类和继承		53
5.1	定义和使用类	53
5.1.1	二维平面上的点	53
5.1.2	复数	55
5.1.3	一元多项式	60
5.2	继承	62
5.3	实验5：类和继承	66
第六章 NumPy数组和矩阵计算		71
6.1	创建数组	71
6.1.1	已有数据存储在其他类型的容器中	71
6.1.2	没有数据但已知形状	72
6.1.3	改变数组的形状	73
6.1.4	数组的堆叠(Stacking)	74
6.1.5	数组的分割	74
6.2	数组的运算	75

目录	5
6.2.1 基本运算	75
6.2.2 函数运算	77
6.3 索引、切片和迭代	78
6.4 复制和视图	83
6.5 矩阵计算	85
6.6 实验6: NumPy数组和矩阵计算	88
第七章 错误处理和文件读写	89
7.1 错误处理	89
7.1.1 错误的分类	89
7.1.2 调试	90
7.1.3 异常处理	94
7.2 文件读写	96
7.2.1 打开和关闭文件	96
7.2.2 读写文本文件	97
7.2.3 读写CSV文件	98
7.2.4 读写JSON文件	100
7.2.5 读写pickle文件	101
7.2.6 读写NumPy数组的文件	102
7.3 实验7: 错误处理和文件读写	103
第八章 时间性能的分析 and 测量	105
8.1 算法和其时间性能的衡量	105
8.2 算法的时间复杂度	105
8.2.1 线性查找	106
8.2.2 插入排序	106
8.2.3 二分查找	107
8.2.4 归并排序	108
8.2.5 穷举法求解3-sum问题	109
8.2.6 穷举法求解subset-sum问题	110
8.3 时间性能的测量	111
8.4 Cython	114
8.5 实验8: 时间性能的分析 and 测量	117
参考文献	120

第一章 绪论

1.1 科学计算

科学计算(Scientific Computing)是以数学和计算机科学为基础形成的交叉学科, 是利用计算机的计算能力求解科学和工程问题的数学模型所需的理论、技术和工具的集合¹。随着计算机的计算能力的不断提升, 科学计算也得到了迅速发展, 并已成为当今科学发现的第三个支柱(前两个分别为理论分析和实验)。数值分析(Numerical Analysis)是科学计算的主要组成部分, 其特点包括: 计算对象是连续数值; 被求解的问题一般没有解析解或理论上无法在有限步求解。例如一元 N 次($N \geq 5$)方程和大多数非线性方程不存在通用的求根公式, 通常需要使用数值方法迭代求解。数值分析的目标是寻找计算效率高和稳定性好的迭代算法。

1.2 利用计算机解决问题

计算机是能够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。计算机通过运行程序解决问题。

1.3 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。

¹<https://www.scicomp.uni-kl.de/about/scientific-computing/>

机器语言用0和1组成的二进制串表示CPU(处理器)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来代替二进制串，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CPU，用它们编写的程序无法移植到不同的CPU上。1956年投入使用的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CPU，实现了可移植性。

近几十年来，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和Python等。
- 2. 声明式(declarative): 仅指明求解的结果，而不说明求解的过程，例如SQL等。
- 3. 过程式(procedural): 可进行过程调用的命令式，例如C、C++、Java和Python等。
- 4. 函数式(functional): 不修改状态的函数互相调用，例如Lisp、ML、Haskell和Scala等。
- 5. 逻辑式(logic): 基于已知事实和规则推断结果，例如Prolog等。
- 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和Python等。

这些编程范式并非互斥，一种语言可同时支持多种范式。

1.4 过程式编程范式

过程式的特点是基于输入和输出将一个较复杂的问题逐步分解成多个子问题。如果分解得到的某个子问题仍然较复杂，则继续对其分解，直至所有子问题都易于解决为止。过程式的程序由多个模块构成，每个模块解决一个子问题。这些模块形成一个树状结构。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，因此不适用于开发大型软件。

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

```
In [2]: run_month_calendar.py --year 2022 --month 10
2022 10
-----
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  29
30 31
```

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：

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

其中 $n = 0, 1, \dots, 6$ 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。

1.5 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。从同一类型的对象中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。例如在一个实现学生选课功能的软件系统中，每位学生是一个对象。从所有学生对象中提取出共同的属性(学号、姓名、所在系等)和操作(选课、退课等)，形成学生类。
2. 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，使软件系统更易于维护。
3. 通过继承可以重复利用已有类的代码，并根据需要进行扩展，从而提升了软件系统的开发效率。
4. 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，缺点是在程序的运行效率上不如过程式程序设计方法。

1.6 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对内存进行精细管理的场合使用C++，而在其他场合都使用Python。原因是用Python语言开发软件的效率更高，并且易于维护和复用。

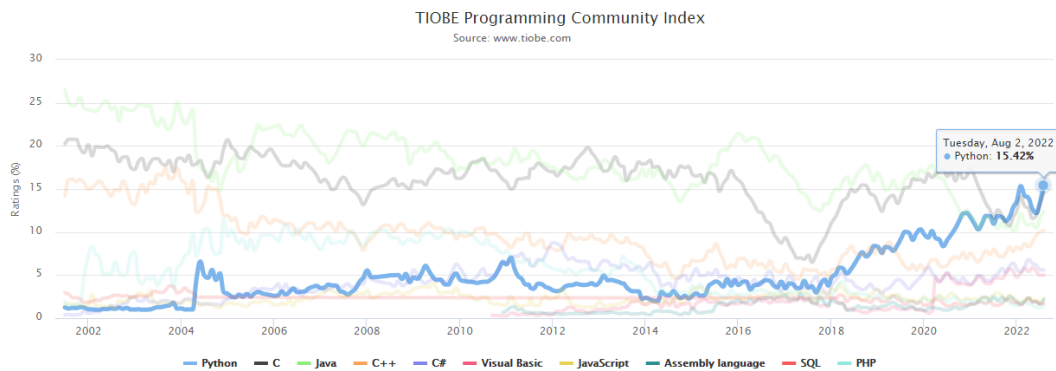


图 1.2: TIOBE指数2022年8月

1.7 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实现功能时也不得不耗费大量的时间。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的许多功能连接在一起。许多需要上百行语句的C语言程序实现的功能只需几行shell语句就可以完成。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总结的设计目标列举如下：

1. 一种简单直观的语言，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。

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。2021年10月，Python3.10发布。

1.8 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，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、图形用户界面、Web开发、系统管理等多个领域。
5. 面向对象, 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代码，可以把这部分代码用C或C++编写并编译成扩展库，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程序的运行效率更低。

1.9 Python科学计算环境

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运行效率更高，但开发耗时更长，软件维护代价也更高。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库。

Python语言及其众多的扩展库(NumPy、SciPy、SymPy和Matplotlib等)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实现了大量的数值计算算法，包括矩阵计算、插值、数值积分、代数方程求解、最优化方法、常微分方程求解、信号和图像处理等。SymPy实现了一种进行符号计算的计算机代数系统，可以进行矩阵计算、表达式展开和化简、微积分、代数方程求解和常微分方程求解等。Matplotlib提供了丰富的绘图功能，可绘制多种二维和三维图示，直观地呈现科学计算的输入数据和输出结果。

和美国公司开发的付费商业软件MATLAB相比，Python科学计算环境的优势是免费开源而且没有国际政治风险。

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；

- 对于学习计算机科学和数学具有浓厚兴趣；
- 自备笔记本电脑。

所有学习资料(讲义和源程序等)将发送至选课同学的电子信箱。每次课程包括两节理论课和一节实验课。考核方式是大作业(75%)、实验课作业(20%)和课堂参与(5%)。每次实验课作业需要在Blackboard系统提交，如果提交时间超过截止时间则扣除50%分数。

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”。

学习本课程的最有效方法是编写和调试程序，在实践中掌握知识和提升能力。

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 大作业

大作业的要求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。采用这种方式的目的：

- 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；
- 训练口头表达能力。

大作业由三部分组成：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。程序可以自己完全独立设计和编写的，也可以对于书籍或互联网上已有程序做一些改进。完成程序以后应使用已知正确结果的测试数据对程序进行充分测试。
2. 文档的格式是HTML。建议使用PythonLinks.html提供的HTML Composer编写文档。使用方法是安装SeaMonkey浏览器以后，在窗口菜单下选择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，学习心得和收获、参考文献等。其中科学技术原理部分必须引用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。课堂讲解的日期可以在本学期的最后四次课(12月9日，12月16日，12月23日和12月30日)中选择。选择在某一日期讲解的同学需要提前至少一天将打包文件通过电子邮件发送给教师，电子邮件的标题必须使用以下格式：Python+学号+姓名+东区(西区)。打包文件采用ZIP格式进行压缩，原则上长度不超过1兆字节。比较大的数据文件无需放在打包文件中，可以在讲解前用优盘复制到讲台电脑上。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；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、设计和实现等方面
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？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输入。
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：

- 在讲解日期当天才提交打包文件；
- 打包文件包含一些非必要文件；
- 文档内容不完整；
- 打包文件的长度超过1兆字节；

1.11 实验1：安装和使用Python开发环境

实验目的

本实验的目的是安装Python开发环境，熟悉其基本功能。

实验内容

1. 安装Python开发环境

安装Python开发环境的方式有两种：

- 下载和安装Anaconda：Anaconda是一个开源的Python发行版本，包含Python解释器、集成开发环境spyder、包管理器conda 和多个科学计算库（numpy、scipy等），可运行在Windows、Linux和Mac OS系统上。可从国内镜像网站²或Anaconda官网³下载Anaconda。安装过程中可指定安装路径，路径中不可包含中文字符。
- 从Python官网⁴下载Python解释器并安装。然后使用pip包管理工具根据自己的需要安装Python包。本课程需要安装的包有：numpy,scipy,sympy,matplotlib和spyder等。pip提供了对Python包的查找、安装和卸载等功能，在命令行运行。以下举例说明：
 - 查找所安装的所有包: `pip list`
 - 安装numpy: `pip install numpy`
 - 升级numpy: `pip install --upgrade numpy`
 - 卸载numpy: `pip uninstall numpy`

后三个命令以numpy为例，可将命令中的numpy替换成任何Python包。

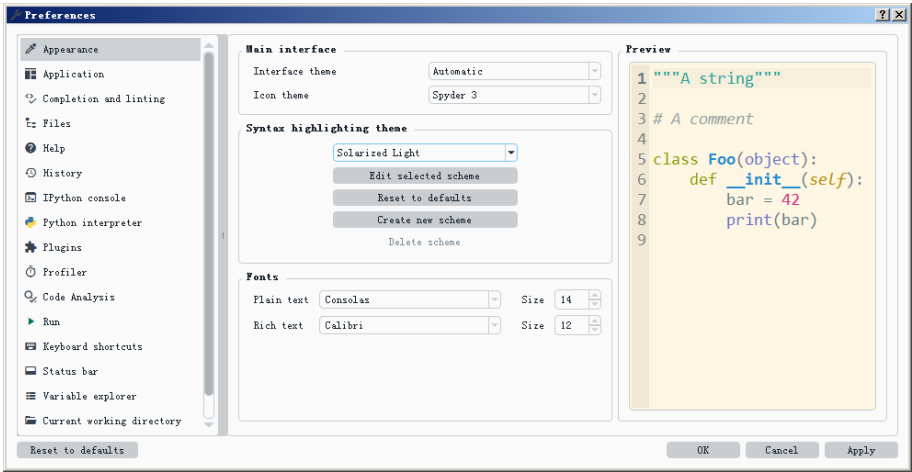


图 1.3: 设置spyder参数

²<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>

³<https://www.anaconda.com/products/individual>

⁴<https://www.python.org/>

2. 设置开发环境参数

安装完成以后，在Windows系统中从已安装程序的列表中可以找到Anaconda文件夹下的spyder的图标，点击此图标即可运行spyder。也可以通过在命令行(控制台)输入命令“spyder”运行spyder。如果需要设置spyder开发环境的参数，可以点击Tools菜单的Preference菜单项，此时出现一个对话框(图1.3)。对话框左边的列表列举了可以修改的参数的所属类别。其中Appearance表示界面的外观。选中Appearance，此时对话框中间的“Syntax highlighting theme”部分有一个下拉列表，其中的每个选项对应一种背景和语法高亮的颜色方案；“Fonts”部分可以设置字体类型和大小。

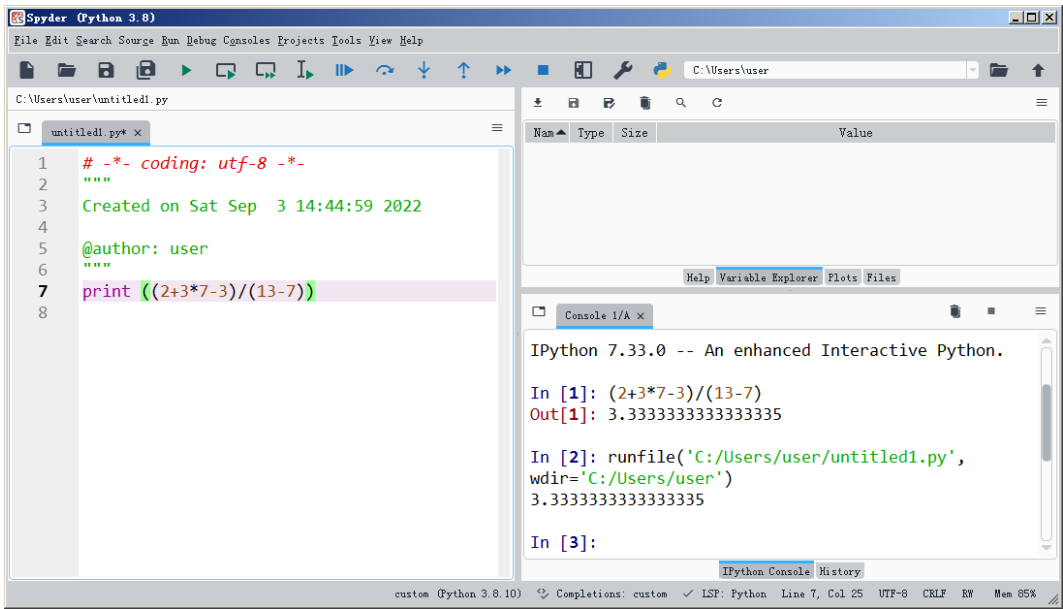


图 1.4: spyder界面

3. 运行Python程序

在spyder(图1.4)中运行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一个完整的程序，点击Run菜单的Run菜单项执行。运行结果显示在IPython窗口中。

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

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


第二章 内建数据类型及其运算

Python语言支持面向对象编程范式，Python程序中的所有数据都是由对象或对象之间的关系所表示。Python程序中的每个对象包括身份(identity，例如内存地址)，类型(type)和值(value)。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。动态类型的益处是程序的语法更简单，付出的代价是程序的运行效率不如静态类型语言(如C和Java等)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。

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运行完成以后，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。

本章介绍了Python语言的常用内建(built-in)数据类型及其运算。Python语言的官方文档[PythonDoc]详细说明了所有内建数据类型及其运算。

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。变量的类型由其存储的数据决定。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

```
1 and as assert break class continue def del elif else except
2 False finally for from global if import in is lambda None
3 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

```
1 In [1]: (32+4)/(23-13)
2 Out [1]: 3.6
3 In [2]: (32+4)//(23-13)
4 Out [2]: 3
5 In [3]: (32+4)%(23-13)
6 Out [3]: 6
7 In [4]: x=379516400906811930638014896080
8 In [5]: x**2
9 Out [5]: 144032698557259999607886110560755362973171476419973199366400
10 In [6]: y=12055735790331359447442538767
11 In [7]: 991*y**2
12 Out [7]: 144032698557259999607886110560755362973171476419973199366399
13 In [8]: Out [7]-Out [5]
14 Out [8]: -1
15 In [9]: x**2-991*y**2-1
16 Out [9]: 0
17 In [10]: bin(367), bin(1981)
18 Out [10]: ('0b101101111', '0b11110111101')
19 In [11]: bin(367 | 1981), bin(367 & 1981), bin(367 ^ 1981)
20 Out [11]: ('0b111111111111', '0b100101101', '0b11011010010')
21 In [12]: bin(~1981), bin(1981 << 3), bin(1981 >> 3)
22 Out [12]: ('-0b111101111110', '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 $\frac{1}{2}2^{-52}=2^{-53}\approx 1.11\times 10^{-16}$ ，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。

标准库以模块作为组成单位，使用某一模块之前需要用import语句将其导入。标准库的sys模块的float_info属性提供了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)。In[1]行用import语句导入sys模块。

Listing 2.3: float类型

```
1 In[1]: import sys
2 In[2]: sys.float_info
3 Out[2]: sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
4           max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
5           min_10_exp=-307, dig=15, mant_dig=53,
6           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]”行包含了多条语句，语句之间用分号(;)分隔，最后一条语句用来输出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

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

¹<https://www-users.cse.umn.edu/~arnold/disasters/patriot.html>

```

6 Out[3]: 0.099999999999999964      # true value should be 0.1
7 In[4]: import sys; max = sys.float_info.max; max
8 Out[4]: 1.7976931348623157e+308
9 In[5]: max*1.001
10 Out[5]: inf      # overflow leads to infinity
11 In[6]: sys.float_info.min*0.00000000000000001
12 Out[6]: 0      # underflow leads to 0
13 In[6]: 1.234567890987654321e38
14 Out[6]: 1.2345678909876543e+38      # number of significant digits <= 15
15 In[7]: import numpy as np
16 In[8]: (2**(2046-1023))*((1 + sum(0.5**np.arange(1, 53))))
17 Out[8]: 1.7976931348623157e+308
18 In[9]: (2**(1-1023))*(1+0)
19 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

```

1 In[1]: x = 3 - 5j;
2 In[2]: y = -(6 - 21j);
3 In[3]: (x+y)/(x - y**2)*(x**3 + y - 3j)
4 Out[3]: (-2.7021404738144748-6.422968879823101j)
5 In[4]: x.real
6 Out[4]: 3.0
7 In[5]: x.imag
8 Out[5]: -5.0
9 In[6]: x.conjugate()
10 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

```
1 In[1]: x = -15.6;
2 In[2]: y = int(x); y
3 Out[2]: -15
4 In[3]: type(y)
5 Out[3]: int
6 In[4]: x=float(y); x
7 Out[4]: -15.0
8 In[5]: type(x)
9 Out[5]: float
10 In[6]: z = complex(abs(x),(2 - y)); z
11 Out[6]: (15+17j)
12 In[7]: abs(z)
13 Out[7]: 22.671568097509265
14 In[8]: pow(z, 1.28)
15 Out[8]: (25.35612170271214+48.0468434395756j)
16 In[9]: pow(1.28, z)
17 Out[9]: (-20.006681963602528-35.28791909603722j)
```

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和以实数作为自变量和因变量的常用数学函数。
cmath模块定义了以复数作为自变量和因变量的常用数学函数。

表2.1列出了math模块的部分函数。

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
math.hypot(*coordinates)	n维欧式范数: math.hypot(3,4,5)值为7.0710678118654755
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模块的部分函数

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cmath模块的求复数平方根的函数。

Listing 2.7: 求解一元二次方程

```
1 In[1]: import math; a=2; b=6; c=1
2 In[2]: r1 = (-b + math.sqrt(b**2 - 4*a*c))/(2*a); r1
3 Out[2]: -0.17712434446770464
4 In[3]: r2 = (-b - math.sqrt(b**2 - 4*a*c))/(2*a); r2
5 Out[3]: -2.8228756555322954
6 In[4]: a=2; b = 6; c = 8;
7 In[5]: r1 = (-b + math.sqrt(b**2 - 4*a*c))/(2*a); r1
8 Out[5]: ValueError: math domain error
9 In[6]: import cmath;
10 In[7]: r1 = (-b + cmath.sqrt(b**2 - 4*a*c))/(2*a); r1
11 Out[7]: (-1.5+1.3228756555322954j)
12 In[8]: r2 = (-b - cmath.sqrt(b**2 - 4*a*c))/(2*a); r2
13 Out[8]: (-1.5-1.3228756555322954j)
```

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(or)和非(not)三种逻辑运算，规则列于表2.2。

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

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。

Listing 2.8: bool类型的运算

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

2.4 序列类型

序列类型(Sequence Types)可以看成是一个存储数据的容器，这些数据是有序的，每条数据对应一个索引值。用 n 表示序列的长度，则索引值的取值范围是区间 $[-n - 1, n - 1]$ 内的所有整数。序列中的第 k 条数据($1 \leq k \leq n$)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。

序列类型包括list(列表)、tuple(元组)、range(范围)、str(字符串)、bytes、bytearray、和memoryview等[PythonDoc]。range通常用于for语句(第三章)。bytes、bytearray和memoryview是存储二进制数据的序列类型。

2.4.1 list类型(列表)和tuple类型(元组)

list和tuple通常用来存储若干同一类型的数据。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。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2.3列出了list类型和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]。

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

表2.4列出了list类型的特有运算，每种运算运行之前s的初始值为[2,7,5,3,11]，t的值为[13,21,17]。

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

表 2.4: list类型的特有运算

2.9演示了list类型的一些运算。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]。

Listing 2.9: list类型的运算

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

2.4.2 str类型(字符串)

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是一个字符编码的国际标准，为人类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引号或双引号括起。单引号和双引号可以相互嵌套。用三引号括起的字符串可以包含程序中的多行语句(包括其中的换行符和空格)，常作为函数和模块的注释。

str类型除了提供表2.3列出的运算以外，还提供了特有运算。表2.5中列出了一些特有运算，示例中s的值为'abc123abc'，t的值为'bc'，u的值为'BC'。

运算名称	运算定义和示例
s.startswith(t)	若s以t为前缀，则返回True，否则返回False。 例: s.startswith(t)的值为False。
s.startswith(t, start)	若s[start:]以t为前缀，则返回True，否则返回False。 例: s.startswith(t, 7)的值为True。
s.startswith(t, start, end)	若s[start:end]以t为前缀，则返回True，否则返回False。 例: s.startswith(t, 7, 9)的值为True。
s.endswith(t[, start[, end]])	若s(或s[start:]或[start:end])以t为后缀， 则返回True，否则返回False。例: s.endswith(t, 4)的值为True， s.endswith(t, 1, 4)的值为False。
s.find(t[, start[, end]])	若s(或s[start:]或[start:end])中存在子串t， 则返回t第一次出现的索引值，否则返回-1。例: s.find(t)的值为1， s.find(t, 2)的值为7，s.find(t, 2, 8)的值为-1。
s.rfind(t[, start[, end]])	若s(或s[start:]或[start:end])中存在子串t， 则返回t最后一次出现的索引值，否则返回-1。
str.isalpha(s)	若s中包含至少一个字符并且所有字符都是字母，则返回True， 否则返回False。例: str.isalpha(s)的值为False。
str.isdecimal(s)	若s中包含至少一个字符并且所有字符都是数字，则返回True， 否则返回False。例: str.isdecimal(s)的值为False。
str.isalnum(s)	若s中包含至少一个字符并且所有字符都是字母或数字，则返回True， 否则返回False。例: str.isalnum(s)的值为True。
str.islower(s)	若s中包含至少一个字母并且所有字母都是小写，则返回True， 否则返回False。例: str.islower(s)的值为True。
str.isupper(s)	与str.islower(s)类似，区别在于判断是否所有字母都是大写。
s.lstrip(t)	若s有某个完全由t中的字符组成的最长前缀，则先从s中删除此前缀 再返回s，否则返回s。若t缺失或值为None，则删除s的最前面的 所有空白字符。例: s.lstrip(t)的值为'abc123abc'。
s.rstrip(t)	与s.lstrip(t)类似，区别在于删除后缀中的字符。 例: s.rstrip(t)的值为'abc123a'。
s.strip(t)	等同于先执行s.lstrip(t)，再执行s.rstrip(t)。
s.replace(t, u)	将s中出现的所有子串t替换成u。 例: s.replace(t, u)的值为'aBC123aBC'。
s.replace(t, u, k)	将s中前k个子串t替换成u。例: s.replace(t, u, 1)的值为'aBC123abc'。
s.split(t)	以t作为分隔符，将s分割成若干子串，再返回由这些子串构成的列表。 例: s.split(t)的值为['a', '123a', '']，其中""表示空串。
t.join(ss)	将列表ss中包含的所有字符串以t作为分隔符连接在一起， 再返回连接的结果。例: t.join(['a', '123a', ''])的值为s。

表 2.5: str类型的部分特有运算

2.10演示了str类型的一些运算。

Listing 2.10: str类型的运算

```

1 In[1]: a = 'allows_embedded_"double"_quotes'; a
2 Out[1]: 'allows_embedded_"double"_quotes'
3 In[2]: b = "allows_embedded_'single'_quotes"; b
4 Out[2]: "allows_embedded_'single'_quotes"
5 In[3]: c = """a=[1,3,5,7,9]; b=[2,4,6,8,10]
6     ...: c=a+b
7     ...: c.sort(); c"""
8 In[4]: c
9 Out[4]: 'a=[1,3,5,7,9];_b=[2,4,6,8,10]\nc=a+b\nc.sort();_c'
10 In[5]: d = [a.startswith('allow'), a.startswith('allou')]; d
11 Out[5]: [True, False]
12 In[6]: e = [a.startswith('embee', 7), a.endswith('quo', 3, -3)]
13 Out[6]: [False, True]
14 In[7]: f = [a.find('em', 3, 6), a.find('em', 7)]; f
15 Out[7]: [-1, 7]
16 In[8]: g = '_hello?!!'
17 In[9]: h = [g.lstrip(), g.rstrip('!?'), g.strip('_!')]; h
18 Out[9]: ['hello?!!', '_hello', 'hello?']
19 In[10]: a.replace('e', 'x', 1)
20 Out[10]: 'allows_xmbedded_"double"_quotes'
21 In[11]: a.replace('e', 'yy', 3)
22 Out[11]: 'allows_yymbyyddydd_"double"_quotes'
23 In[12]: a.split('e')
24 Out[12]: ['allows_', 'mb', 'dd', 'd_"doubl', '"_quot', 's']

```

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

```

1 In[1]: "%-16.8f" % 345.678987654321012
2 Out[1]: '345.67898765_____'
3 In[2]: "%16.8g" % 3.45678987654321012e34
4 Out[2]: '____3.4567899e+34'

```

```
5 In [3]: "%16X%-16d" % (345678987654, 987654321012)
6 Out [3]: '          507C12C186          987654321012          '
```

表2.6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 2.6: 常用的转换说明符

2.5 set类型(集合)

set类型可以看成是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。set类型的语法是用大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集合。

2.12演示了set类型的一些运算。

Listing 2.12: set类型的运算

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

frozenset类型是不可变的set类型。表2.7列出了set类型的常用运算，frozenset类型可进行表中除了add和remove以外的运算。

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。
s.add(x)	向s中添加数据x。若s中存在x，则s不发生变化。
s.remove(x)	从s中移除数据x。若s中不存在x，则报错。

表 2.7: set类型的常用运算

2.6 dict类型(字典)

dict类型可以看成是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

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

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

表2.8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。
items()	返回d中所有数据，对于每条数据返回一个由关键字和值组成的元组。
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 2.8: dict类型的常用运算

2.7 实验2：内建数据类型及其运算

实验目的

本实验的目的是掌握常用的类型(float和list)的运算。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

1. float类型的计算

等额本息是一种分期偿还贷款的方式，即借款人每月按相等的金额偿还贷款本息，每月还款金额*P*可根据贷款总额*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。

2. math模块

定义三个变量 “a=3; b=6; c=7” 表示一个三角形的三个边的长度，使用公式

$$a^2 = b^2 + c^2 - 2bc \cos \alpha$$

$$b^2 = a^2 + c^2 - 2ac \cos \beta$$

$$c^2 = a^2 + b^2 - 2ab \cos \gamma$$

分别计算三个内角 (α, β, γ) 的度数，然后检验等式 $\alpha + \beta + \gamma = 180$ 是否成立。

3. list类型的运算

定义两个列表 “s=[2,4,0,1,3,9,5,8,6,7]; t=[2,6,8,4]”，对于表2.4中的每种运算，先手工计算其结果，然后在Ipython中运行并记录输出结果。

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，以计算整数的绝对值为例说明。根据绝对值的定义，可以有三种实现方式：3.1，3.2和3.3。这三个程序的第1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: ”。int函数将用户的输(字符串)转换为int类型值后赋值给变量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的第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的第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，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，它实现了同样的功能。

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))
6 # The absolute value of -32 is 32
```

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内。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 $90 \leq x \leq 100$ ，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 $80 \leq x < 90$ 。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 $70 \leq x < 80$ 。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 $60 \leq x < 70$ 。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 $0 \leq x < 60$ ，此时无需给grade赋值，因为grade的值为默认值'F'。

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语句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))
14 # The grade of score 81 is B
```

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之后输出一个空格，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
```

第二章介绍的所有序列类型(包括list、tuple和str等)和set、map等类型的对象称为可遍历对象(iterable)，即可以在for循环中遍历其包含的所有数据。程序3.7输出一个由整数组成的集合中所包含的3的倍数。

Listing 3.7: for语句输出一个由整数组成的集合中所包含的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其电话号码。

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
3 for name, num in contacts.items():
```

```
4     print('%s->%d' % (name, num), end=';')
5 # Tom -> 12345; Jerry -> 54321; Mary -> 23415;
6 print()
7 for name in contacts.keys():
8     print('%s->%d' % (name, contacts[name]), end=';')
9 # Tom -> 12345; Jerry -> 54321; Mary -> 23415;
```

程序3.9使用for语句输出一个字符串中的所有字符和其对应的Unicode编码值。

Listing 3.9: for语句输出一个字符串中的所有字符和其对应的Unicode编码值

```
1 s = 'Python'
2 for c in s:
3     print('(%s->%d)' % (c, ord(c)), end=' ')
4 # (P -> 80) (y -> 121) (t -> 116) (h -> 104) (o -> 111) (n -> 110)
```

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，提高程序的执行效率并降低资源占用。

这个问题的设计方案如下：

- 1. 列举给定区间内的所有自然数。
 - (a) 对于每个自然数*i*，判断其是否质数。对于每个从2到*i* - 1的自然数*j*：
 - i. 检查*i*是否可以被*j*整除。
 - (b) 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，但是在运行效率上还有改进的余地。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3.10。第5行开始的外循环用range类型列举所有奇数。第7至第10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11行根据isPrime的值决定是否输出*i*。第7行至第10行的内循环作为一个整体相对于第5行的for必须有四个空格的缩进。

Listing 3.10: for语句和break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 if ub % 2 == 0: ub -= 1
5 for i in range(lb, ub + 1, 2):
6     isPrime = True
7     for j in range(2, math.ceil(math.sqrt(i)) + 1):
8         if i % j == 0:
9             isPrime = False
10            break
11    if isPrime: print(i, end=' ')
12 # 101 103 107 109 113 127 ... 199

```

程序3.11实现了和3.10相同的功能。区别在于第11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12行输出*i*。

Listing 3.11: 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 if ub % 2 == 0: ub -= 1
5 for i in range(lb, ub + 1, 2):
6     isPrime = True
7     for j in range(2, math.ceil(math.sqrt(i)) + 1):
8         if i % j == 0:
9             isPrime = False
10            break
11    if not isPrime: continue
12    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3.12用while语句实现了和3.5相同的功能。

Listing 3.12: 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3.13用while语句实现了和3.6相同的功能。

Listing 3.13: 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3.14用while语句实现了辗转相减法求两个正整数的最大公约数。

Listing 3.14: 辗转相减法求两个正整数的最大公约数

```

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3.15演示了推导式的用法。第2行创建一个列表multiplier_of_3，由集合nums中3的倍数构成。第4行创建一个集合square_of_odds，由nums中的奇数的平方构成。第8行基于从列表s转换到的集合set(s)创建一个字典sr，sr中的每条数据由集合中的每个数和其除以3得到的余数组成。第10行从字典sr创建另一个字典tr，由sr中3的倍数组成。

Listing 3.15: 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}
```

3.5 实验3: 分支和迭代

实验目的

本实验的目的是掌握分支和迭代的语句。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

1. 考拉兹猜想(Collatz conjecture)

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。输出的序列显示在一行，相邻的数之间用空格分隔。

2. 字符串加密

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。用户输入一个不小于-15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. 推导式转换为for语句

将程序3.15中的所有推导式转换为for语句。

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。把需要多次运行的代码写成函数，可以实现代码的重复利用。以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3.14可以改写成一个函数gcd，它接受两个自然数作为输入值(即形参，formal parameters)，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。第1行是函数头，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行至第9行构成函数体，相对函数头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。函数头后面通常写一个由三个(单或双)引号括起的字符串作为函数的注释。注释的内容包括函数的形参、实现的功能、返回值和设计思路等。第9行的return语句将变量a的值作为运行结果返回。

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，并且在顺序上一一对应。第10行用参数156和732调用函数gcd，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，函数的返回值是12。第11行用参数1280和800调用函数gcd，函数的返回值是160。

Listing 4.1: 定义一个辗转相减法求两个正整数的最大公约数的函数

```
1 def gcd(a, b):
2     """ compute the greatest common divisor of a and b using
3         repeated subtractions """
4     while a != b:
5         if a > b:
6             a -= b;
7         else:
8             b -= a;
9     return a
10
11 print (gcd(156, 732)) # 12
```

```
12 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的第1行至第8行中定义了一个函数max_min，它接受两个数作为参数，返回它们的最大值和最小值。

Listing 4.2: 定义一个求最大值和最小值的函数

```
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。局部变量只能在函数体内访问，在函数运行结束时即被销毁。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，除非某个函数中定义了同名的局部变量。

例如程序4.3的前两行定义的变量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。在函数f中可以访问第2行定义的全局变量c，但无法访问第1行定义的全局变量b。第6行的输出结果表明第5行中出现的b是第4行定义的局部变量。第7行的输出结果表明第4行是给局部变量b赋值，而不是给第1行定义的全局变量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某个全局变量，需要用global声明它。例如程序4.4的第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

函数头可以给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的顺序保持一致。

程序4.5定义了一个函数get_primes(1-13行)，它接受两个自然数作为形参，并返回以这两个自然数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并且和实参136对应的形参是lb，这里关键字实参的顺序和函数头中的形参顺序并不一致。

这个函数的返回值有多个而且数量未知，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(例如列表)中，最后返回整个容器。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     """ find the prime numbers within the interval [lb, ub] """
3     import math
4     primes = []
5     if lb % 2 == 0: lb += 1
6     if ub % 2 == 0: ub -= 1
7     for i in range(lb, ub + 1, 2):
8         isPrime = True
9         for j in range(2, math.ceil(math.sqrt(i)) + 1):
10             if i % j == 0:
11                 isPrime = False

```

```
12         break
13     if isPrime: primes.append(i)
14     return primes
15
16 print (get_primes(40, 50))    # [41, 43, 47]
17 print (get_primes(120, 140)) # [127, 131, 137, 139]
18 print (get_primes(80))      # [83, 89, 97]
19 print (get_primes(ub=150, lb=136)) # [137, 139, 149]
```

4.4 函数式编程(functional programming)

函数式编程的基本含义是函数具有和其他类型(如int、float等)同样的性质:被赋值给变量;作为实参传给被调用函数的形参;作为函数的返回值。

4.4.1 函数作为实参

内建函数sorted可以对一个可遍历对象(iterable)中的数据进行排序,排序的结果存储在一个新创建的列表中。关键字实参key指定一个函数,它从每条数据生成用于排序的比较值。关键字实参reverse的默认值为False,若设为True则表示从大到小的次序排序。

程序4.6演示了用函数作为实参调用sorted函数。In[1]行定义了由字符串构成的列表。In[2]行对其排序,Out[2]行显示了输出结果,默认的排序方式是按照两个字符串的字符序列的Unicode编码值从小到大排序,即首先比较两个字符串的第一个字符的Unicode编码值,若不等则已确定顺序,若相等则再比较第二个字符,以此类推。In[3]行在调用sorted函数时设置了关键字实参key为求字符串长度的内建函数len,Out[3]行显示了输出结果,即按照字符串的长度从小到大排序。In[4]行和In[3]行的区别在于设置了关键字实参reverse为True,Out[4]行显示了输出结果,即按照字符串的长度从大到小排序。In[5]行定义了一个函数m1,它返回一个字符串中的所有字符的Unicode编码值的最小值。In[6]行在调用sorted函数时设置了关键字实参key为m1,Out[6]行显示了输出结果,即按照字符串的所有字符的Unicode编码值的最小值从小到大排序。In[7]行定义了一个函数m2,它返回一个元组,由一个字符串中的所有字符的Unicode编码值的最小值(以下简称为最小编码值)和字符串的长度组成。元组在排序时看成组成元组的数据的序列,即先比较两个元组的第一条数据,若不等则已确定顺序,若相等则再比较第一条数据,以此类推。In[8]行在调用sorted函数时设置了关键字实参key为m2,Out[8]行显示了输出结果,即先按照最小编码值从小到大排序,若两个字符串具有相同的最小编码值,则按照长度从小到大排序。

Listing 4.6: 用函数作为实参调用sorted函数

```
1 In[1]: animals = ["elephant", "tiger", "rabbit", "goat", "dog",
2           "penguin"]
3 In[2]: sorted(animals)
```

```

4 Out[2]: ['dog', 'elephant', 'goat', 'penguin', 'rabbit', 'tiger']
5 In[3]: sorted(animals, key=len)
6 Out[3]: ['dog', 'goat', 'tiger', 'rabbit', 'penguin', 'elephant']
7 In[4]: sorted(animals, key=len, reverse=True)
8 Out[4]: ['elephant', 'penguin', 'rabbit', 'tiger', 'goat', 'dog']
9 In[5]: def m1(s): return ord(min(s))
10 In[6]: sorted(animals, key=m1)
11 Out[6]: ['elephant', 'rabbit', 'goat', 'dog', 'tiger', 'penguin']
12 In[7]: def m2(s): return ord(min(s)), len(s)
13 In[8]: sorted(animals, key=m2)
14 Out[8]: ['goat', 'rabbit', 'elephant', 'dog', 'tiger', 'penguin']

```

如果一个函数在定义以后只使用一次，并且函数体可以写成一个表达式，则可以使用Lambda函数语法将其定义成一个匿名函数：`g = lambda 形参列表: 函数体表达式`

例如程序4.7中定义的函数`map_fs`将函数`f`作用于列表`s`中的每条数据，用函数的返回值替代原来的数据，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 4.7: Lambda函数

```

1 def map_fs(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_fs(lambda x: x+1, a)) # [2, 4, 6, 8, 10]
7 print (map_fs(lambda x: x*x-1, a)) # [3, 15, 35, 63, 99]

```

程序4.7中定义的函数`map_fs`是内建函数`map`的简化。内建函数`map`可将它的第一个参数(一个函数)作用于其余参数(一个或多个可遍历对象)中的每条数据，并返回一个可遍历对象，它可以生成一个列表。内建函数`filter`类似一个过滤器，它的第一个参数(一个函数)作用于其余参数(一个或多个可遍历对象)中的每条数据时返回一个bool类型值。若值为True，则对应数据被保留在输出结果中，否则被舍弃。程序4.8演示了用函数作为实参调用内建函数`map`和`filter`。In[2]行定义了一个反转字符串的函数`reverse`。In[3]行使用`reverse`调用`map`函数，反转了In[1]行定义的列表`animals`中的每个字符串，结果显示在Out[3]行。In[4]行定义了程序4.6中定义的函数`m2`。In[5]行使用`m2`调用`map`函数，对列表`animals`中的每个字符串输出一个元组，结果显示在Out[5]行。In[6]行定义了一个函数`f`，它返回三个形参的和。In[7]行使用`f`调用`map`函数，对三个列表中对应位置的数据分别求和($1+10+100=111, 2+20+200=222, 3+30+300=333$)，结果显示在Out[7]行。In[9]行定义了一个函数`r3`，它判断形参是不是3的倍数。In[10]行使用`r3`调用`filter`函数，提取In[8]行定义的集合`nums`包含的3的倍数，结果显示在Out[10]行。

Listing 4.8: 用函数作为实参调用内建函数map和filter

```

1 In[1]: animals = ["elephant", "tiger", "rabbit", "goat", "dog",
2             "penguin"]
3 In[2]: def reverse(s): return s[::-1]
4 In[3]: list(map(reverse, animals))
5 Out[3]: ['tnahpele', 'regit', 'tibbar', 'taog', 'god', 'niugnep']
6 In[4]: def m2(s): return ord(min(s)), len(s)
7 In[5]: list(map(m2, animals))
8 Out[5]: [(97, 8), (101, 5), (97, 6), (97, 4), (100, 3), (101, 7)]
9 In[6]: def f(a, b, c): return a + b + c
10 In[7]: list(map(f, [1, 2, 3], [10, 20, 30], [100, 200, 300]))
11 Out[7]: [111, 222, 333]
12 In[8]: nums = {25, 18, 91, 365, 12, 78, 59}
13 In[9]: def r3(n): return n % 3 == 0
14 In[10]: list(filter(r3, nums))
15 Out[10]: [12, 78, 18]

```

4.4.2 函数作为返回值

程序4.9定义了一个函数key_fun，其中定义了两个用于字符串排序的函数m1和m2。列表ms存储了None、len和这些函数。key_fun以实参为索引值返回ms中的对应函数，即该函数的返回值是一个函数。第10行至第11行的循环依次使用这些函数对字符串进行排序，其中None表示默认的排序方式。输出结果显示在4.10。

Listing 4.9: 用函数作为作为返回值进行字符串排序

```

1 def key_fun(n):
2     def m1(s): return ord(min(s))
3     def m2(s): return ord(min(s)), len(s)
4
5     ms = [None, len, m1, m2]
6     return ms[n]
7
8 animals = ["elephant", "tiger", "rabbit", "goat", "dog", "penguin"]
9 for i in range(4):
10     print (sorted(animals, key=key_fun(i)))

```

Listing 4.10: 程序4.9的输出结果

```

1 ['dog', 'elephant', 'goat', 'penguin', 'rabbit', 'tiger']
2 ['dog', 'goat', 'tiger', 'rabbit', 'penguin', 'elephant']

```

```
3 ['elephant', 'rabbit', 'goat', 'dog', 'tiger', 'penguin']
4 ['goat', 'rabbit', 'elephant', 'dog', 'tiger', 'penguin']
```

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分析问题的递归模型，然后提出设计方案。以下举例说明。

4.5.1 阶乘

阶乘的定义本身就是一个递归模型：

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

程序4.11中的factorial函数计算阶乘。

Listing 4.11: 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
```

4.5.2 最大公约数

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4.12的gcd函数求解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，例如这个gcd函数对应的迭代形式是程序4.1中的gcd函数。

Listing 4.12: 计算最大公约数的递归函数

```

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
    
```

4.5.3 字符串反转

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4.13的reverse函数反转一个字符串。

Listing 4.13: 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
    
```

4.5.4 快速排序

快速排序是一种著名的排序算法，以下用一个实例描述其求解过程。要排序的原始数据集是列表[3, 6, 2, 9, 7, 3, 1, 8]。以第一条数据3为基准对其进行调整，把比3小的数据移动到3的左边，把比3大的数据移动到3的右边。调整的结果为[2, 1, 3, 3, 6, 9, 7, 8]，可看成是三个列表的连接: [2, 1], [3, 3],

和[6, 9, 7, 8]。这三个列表分别由小于3的数据、等于3的数据和大于3的数据组成。排序完成的结果是[1, 2, 3, 3, 6, 7, 8, 9]，也可看成是三个列表的连接: [1, 2], [3, 3], 和[6, 7, 8, 9]。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, 8]进行排序可得[6, 7, 8, 9]，这也是原问题的一个子问题。由此可总结出递归模型如下：

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

程序4.14的qsort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效实现。第4行定义的列表s1由s中比s[0]小的数据组成。第5行定义的列表s1由s中比s[0]大的数据组成。第6行定义的列表s0由s中等于s[0]的数据组成。

Listing 4.14: 实现快速排序的递归函数

```
1 def qsort(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 qsort(s1) + s0 + qsort(s2)
7
8 print (qsort([3, 6, 2, 9, 7, 3, 1, 8])) # [1, 2, 3, 3, 6, 7, 8, 9]
```

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1.3节提出的日历问题(输出给定年份和月份的日历)为例说明创建和使用模块的方法。程序4.15列出了模块month_calendar.py的代码。第1行至第17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。第20行至第21行定义了一个函数is_leap，它根据形参指定的某年是不是闰年返回True或False。第23行至第27行定义了一个函数test___is_leap用于测试函数is_leap。第24行定义了一个字典d，它的每条数据的关键字是一个年份，对应的值表示该年份是不是闰年。第26行至第27行的循环遍历字典d，对每条数据测试函数is_leap的输出结果是否和d中的正确答案一致，若不一致则报错。第29行至第32行定义了一个函数get_0101_in_week，它根据1.3节的公式计算形参指定的某年的元旦是星期几。这里0表示星期日，1至6分别表示星期一至星期六。第34行至第39行定义了一个函数test___get_0101_in_week用于测试函数get_0101_in_week。第35行定义了一个字典d，它的每条数据

的关键字是一个年份，对应的值表示该年份的元旦是星期几。

第41行至第42行定义了一个字典month_days，它的每条数据的关键字是一个月份，对应的值表示该月份有多少天，这里假定当前的年份不是闰年。第43行至第47行定义了一个函数get_num_days_in_month，它返回形参指定的某年某月所包含的天数，这里对闰年的二月单独处理。第49行至第53行定义了一个函数get_num_days_from_0101_to_m01，它调用函数get_num_days_in_month计算从形参指定的某年的元旦到形参指定的某年某月的第一天之间共经历了多少天。第55行至第59行定义了一个函数get_m01_in_week，它调用函数get_0101_in_week和函数get_num_days_from_0101_to_m01计算形参指定的某年某月的第一天是星期几。第61行至第66行定义了一个函数test___get_m01_in_week用于测试函数get_m01_in_week。第62行定义了一个字典d，它的每条数据的关键字是一个由某个年份和某个月份组成的元组，对应的值表示这个年份这个月份的第一天是星期几。第68行至第71行定义了一个函数print_header输出日历的标题。第73行至第78行定义了一个函数print_body，它调用函数get_m01_in_week并输出日历的主体。第80行至第82行定义了一个函数print_monthly_calendar，它调用函数函数print_header和函数print_body输出给定年份和月份的日历。第84行至第87行定义了一个函数test_all_functions，它运行所有的以“test___”为名称前缀的测试函数。

第89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。列表的第一条数据是模块名称，其余数据是用户输入的参数(若用户输入了至少一个参数)。第90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92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94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对一些关键函数进行测试。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96行else后面的语句块导入和使用argparse模块解析用户在命令行输入的所有参数(年份和月份)，然后调用函数print_monthly_calendar输出给定年份和月份的日历。这里的输入格式要求在每个参数的输入值前面都要用“-参数名称”的格式指定对应的参数名称(参数名称前面是两个减号)，因此参数的顺序无关紧要。

Listing 4.15: 输出给定年份和月份的日历的模块

```

1 """
2 Module for printing the monthly calendar for the year and
3 the month specified by the user.
4
5 For example, given year 2022 and month 9, the module prints
6 the monthly calendar of September 2022.
7
8 >>> run month_calendar.py --year 2022 --month 9
9 2022  9

```

```

10 -----
11 Sun Mon Tue Wed Thu Fri Sat
12         1   2   3
13 4     5   6   7   8   9  10
14 11    12  13  14  15  16  17
15 18    19  20  21  22  23  24
16 25    26  27  28  29  30
17 """
18 import sys, math
19
20 def is_leap(year):
21     return (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
22
23 def test___is_leap():
24     d = {1900:False, 2000:True, 2020:True, 2022:False}
25     for y in d.keys():
26         if d[y] != is_leap(y):
27             print("test failed: is_leap(%d) != %s" % (y, d[y]))
28
29 def get_0101_in_week(year):
30     return (year + math.floor((year - 1) / 4) -
31            math.floor((year - 1) / 100) +
32            math.floor((year - 1) / 400)) % 7
33
34 def test___get_0101_in_week():
35     d = {2008:2, 2014:3, 2021:5, 2022:6}
36     for y in d.keys():
37         if d[y] != get_0101_in_week(y):
38             print("test failed: get_0101_in_week(%d) != %s"
39                   % (y, d[y]))
40
41 month_days = {1:31, 2:28, 3:31, 4:30, 5:31, 6:30,
42              7:31, 8:31, 9:30, 10:31, 11:30, 12:31}
43 def get_num_days_in_month(year, month):
44     n = month_days[month]
45     if month == 2 and is_leap(year):
46         return n + 1
47     return n
48

```

```
49 def get_num_days_from_0101_to_m01(year, month):
50     n = 0
51     for i in range(1, month):
52         n += get_num_days_in_month(year, i)
53     return n
54
55 def get_m01_in_week(year, month):
56     n1 = get_0101_in_week(year)
57     n2 = get_num_days_from_0101_to_m01(year, month)
58     n = (n1 + n2) % 7
59     return n
60
61 def test___get_m01_in_week():
62     d = {(2022, 6):3, (2019, 10):2, (2016, 5):0, (2011, 7):5}
63     for y in d.keys():
64         if d[y] != get_m01_in_week(y[0], y[1]):
65             print("test failed: get_m01_in_week(%s) != %s"
66                   % (y, d[y]))
67
68 def print_header(year, month):
69     print ("%d_%d" % (year, month))
70     print ("-----")
71     print ("Sun_Mon_Tue_Wed_Thu_Fri_Sat")
72
73 def print_body(year, month):
74     n = get_m01_in_week(year, month)
75     print (n * 4 * ' ', end='')
76     for i in range(1, get_num_days_in_month(year, month) + 1):
77         print ('%-04d' % i, end='')
78         if (i + n) % 7 == 0: print ()
79
80 def print_monthly_calendar(year, month):
81     print_header(year, month)
82     print_body(year, month)
83
84 def test_all_functions():
85     test___is_leap()
86     test___get_0101_in_week()
87     test___get_m01_in_week()
```

```

88
89 if __name__ == '__main__':
90     if len(sys.argv) == 1:
91         print (__doc__)
92     elif len(sys.argv) == 2 and sys.argv[1] == '-h':
93         print (__doc__)
94     elif len(sys.argv) == 2 and sys.argv[1] == 'test':
95         test_all_functions()
96     else:
97         import argparse
98         parser = argparse.ArgumentParser()
99         parser.add_argument('--year', type=int, default=2022)
100        parser.add_argument('--month', type=int, default=1)
101        args = parser.parse_args()
102        year = args.year; month = args.month
103        print_monthly_calendar(year, month)

```

4.16列出了运行模块的示例。在IPython中运行程序时，首先进入文件calendar.py所在目录，然后在输入In[2]行的命令。也可以在操作系统的命令行窗口运行模块，首先进入文件calendar.py所在目录，然后输入命令“python month_calendar.py --year 2022 --month 10”。

Listing 4.16: 运行模块calendar.py的示例

```

1 In[1]: cd D:\Python\src
2 Out[1]: D:\Python\src
3 In[2]: run month_calendar.py --year 2022 --month 10
4 2022  10
5 -----
6 Sun Mon Tue Wed Thu Fri Sat
7                                     1
8  2   3   4   5   6   7   8
9  9   10  11  12  13  14  15
10 16  17  18  19  20  21  22
11 23  24  25  26  27  28  29
12 30  31

```

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。例如程序4.17调用month_calendar模块的get_m01_in_week函数以计算给定的某年某月某日是星期几。第1行也可以写成“from month_calendar import get_m01_in_week”，表示仅导入month_calendar模块的get_m01_in_week函数，此时第3行的函数调用需写成“get_m01_in_week(y, m)”。

Listing 4.17: 程序ydm.py调用month_calendar模块的get_m01_in_week函数

```
1 import month_calendar
2 y, m, d = 2022, 9, 18
3 n = (month_calendar.get_m01_in_week(y, m) + d - 1) % 7
4 dw = "Sun_Mon_Tue_Wed_Thu_Fri_Sat"
5 print(dw[4*n:4*n+4]) # Sun
```

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插入到列表sys.path中，然后可以导入模块。

Listing 4.18: 将模块所在目录加入到列表sys.path中

```
1 In[1]: run ymd.py
2 Out[1]: ... ModuleNotFoundError: No module named 'month_calendar'
3 In[2]: import sys; sys.path.insert(0, 'D:\Python\src')
4 In[3]: run ymd.py
5 Sun
```

本节以输出日历为例介绍创建和使用模块的方法。Python标准库的calendar模块的 TextCalendar类已提供了输出日历的功能。程序4.19的第3行的输出结果是2022年10月的日历。第4行的输出结果是2022年的日历，关键字实参m指定列数。

Listing 4.19: Python标准库的calendar模块的TextCalendar类

```
1 from calendar import TextCalendar
2 tc = TextCalendar()
3 print(tc.formatmonth(2022, 10))
4 print(tc.formatyear(2022, m=4))
```

4.7 实验4：函数和模块

实验目的

本实验的目的是掌握以下内容：定义和调用函数，创建和使用模块。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

1. 二分查找

编写一个程序使用二分法查找给定的包含若干整数的列表s中是否存在给定的整数k。若存在则输

出k的索引值，否则输出-1。4.20已列出了部分代码，需要实现函数is_sorted和递归函数binary_search。binary_search在列表s的索引值属于闭区间[low,high]的数据中查找k，若找到则返回k的索引值，否则返回-1。binary_search的设计方案是：首先判断low是否大于high；若是则返回-1；否则计算low和high的平均值mid；若k等于s[mid]，则返回mid；若k大于s[mid]或小于s[mid]，则分别确定合适的low和high值作为实参递归调用binary_search并返回结果。

Listing 4.20: 二分查找

```

1 def is_sorted(s):
2     # to be implemented
3
4 def qsort(s):
5     if len(s) <= 1: return s
6     s1 = [i for i in s if i < s[0]]
7     s2 = [i for i in s if i > s[0]]
8     s0 = [i for i in s if i == s[0]]
9     return qsort(s1) + s0 + qsort(s2)
10
11 def binary_search(s, low, high, k):
12     # to be implemented
13
14 s = [5, 6, 21, 32, 51, 60, 67, 73, 77, 99]
15 if not is_sorted(s):
16     s = qsort(s)
17 print (binary_search(s, 0, len(s) - 1, 5))    # 0
18 print (binary_search(s, 0, len(s) - 1, 31))   # -1
19 print (binary_search(s, 0, len(s) - 1, 99))   # 9
20 print (binary_search(s, 0, len(s) - 1, 64))   # -1
21 print (binary_search(s, 0, len(s) - 1, 51))   # 4

```

2. 有理数的四则运算

有理数的一般形式是 a/b ，其中 a 是整数， b 是正整数，并且当 a 非0时 $|a|$ 和 b 的最大公约数是1。编写一个模块rational.py实现有理数的四则运算。4.21已列出了部分代码，需要实现标注了“to be implemented”的函数。程序中用一个列表[n, d]表示有理数，其中n表示分子，d表示分母。函数reduce调用函数gcd进行约分。函数add、sub、mul和div分别进行加减乘除运算，运算的结果都需要约分，并且分母不出现负号。函数test_all_functions使用已知答案的数据对这些运算进行测试。函数output按照示例的格式输出有理数，例如[-13,12]表示的有理数的输出结果是字符串“-13/12”。用户在命令行输入三个命名参数。“-op”表示运算符，可以是“add”(加法)、“sub”(减法)、“mul”(乘法)或“div”(除法)。“-x”和“-y”表示进行计算的两个有理数。有理数以字符串的形式输入，必须用圆括号括起，分子和分母之间用“/”分隔。例如有理数“-20/-3”对应的输入形式是(-20/-3)，用户输入有理数可以在分母出现负号。函数get_rational从表示有理数的字符串中得到列表[n, d]，

例如从字符串 “(-20/-3)” 得到 [-20,-3]。

Listing 4.21: 有理数的四则运算

```
1 """
2 Module for performing arithmetic operations for rational numbers.
3
4 To run the module, user needs to supply three named parameters:
5 1. op stands for the operation:
6     add for addition
7     sub for subtraction
8     mul for multiplication
9     div for division
10 2. x stands for the first operand
11 3. y stands for the second operand
12
13 x and y must be enclosed in paired parentheses.
14
15 For example:
16
17 >>> run rational.py --op add --x (2/3) --y (-70/40)
18 -13/12
19 >>> run rational.py --op sub --x (-20/3) --y (120/470)
20 -976/141
21 >>> run rational.py --op mul --x (-6/19) --y (-114/18)
22 2/1
23 >>> run rational.py --op div --x (-6/19) --y (-114/-28)
24 -28/361
25 """
26
27 import sys, math
28
29 def test_all_functions():
30     # to be implemented
31
32 def gcd(a, b):
33     if a == b:
34         return a
35     elif a > b:
36         return gcd(a-b, b)
```



```
37     else:
38         return gcd(a, b-a)
39
40 def reduce(n, d):
41     # to be implemented
42
43 def add(x, y):
44     # to be implemented
45
46 def sub(x, y):
47     # to be implemented
48
49 def mul(x, y):
50     # to be implemented
51
52 def div(x, y):
53     # to be implemented
54
55 def output(x):
56     # to be implemented
57
58 def get_rational(s):
59     # to be implemented
60
61 if __name__ == '__main__':
62     if len(sys.argv) == 1:
63         print (__doc__)
64     elif len(sys.argv) == 2 and sys.argv[1] == '-h':
65         print (__doc__)
66     elif len(sys.argv) == 2 and sys.argv[1] == 'test':
67         test_all_functions()
68     else:
69         import argparse
70         parser = argparse.ArgumentParser()
71         parser.add_argument('--op', type=str)
72         parser.add_argument('--x', type=str)
73         parser.add_argument('--y', type=str)
74         args = parser.parse_args()
75         op = args.op
```

```
76     x = get_rational(args.x); y = get_rational(args.y)
77     f = {'add':add, 'sub':sub, 'mul':mul, 'div':div}
78     output(f[op](x, y))
```

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。以下举例说明。

5.1.1 二维平面上的点

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、以另一个点为中心旋转、计算与另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2行至第26行定义了一个Point2D类。类的定义由“class 类名(父类)”开始。第3行至第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 、 y 坐标和点的名称(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 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 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 y 坐标。第12行至第16行的方法`rotate`方法计算`self`以另一个点 p 为轴旋转角度 t 以后的坐标。第18行至第20行的`distance`方法计算`self`与另一个点 p 之间的距离。第22行至第26行的`__str__`方法返回`self`的字符串表示。Python规定了类的一些特殊的方法名称，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。例如对于Point2D类的对象 a 而言，`a.__str__()`等同于简化形式`'%s' % a`。

第28行至第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为中心旋转90度。第35行将点a以点b为中心旋转180度。

Listing 5.1: Point2D类

```

1 import math
2 class Point2D:
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)

```

5.1.2 复数

程序5.2定义了一个Complex类表示复数，并实现了复数的基本运算(部分代码来源于¹⁾。Complex的属性`re`和`im`分别表示复数的实部和虚部。第4行至第10行定义了构造方法。第5行至第6行判断传给形参`re`和`im`的数据的类型是不是`float`或`int`，若是则将它们分别赋值给`re`和`im`，否则在第9行以抛出异常的方式报错(异常将在第七章介绍)。`__dict__`是一个存储了对象的所有属性名和对应的属性值的字典。第10行将属性名`re`和`im`分别映射到`re`和`im`，例如若`c`是一个Complex类的对象，则`c.re`访问`re`。第12行至第13行定义的`__setattr__`方法在`self`的属性被试图赋值时被自动调用，其作用在于以抛出异常的方式禁止属性值被修改。第15行的`__str__`方法返回`self`的字符串表示。第17行定义了`__repr__`方法，它返回一个完整表示对象的字符串，通常是一个可以运行的表达式，运行的结果就是生成这个对象。第19行定义了`__abs__`方法，它返回复数的模。对于一个Complex类的对象`c`而言，`c.__abs__()`等同于语法上的简化形式`abs(c)`。第21行定义了`abs`方法等价于`__abs__`方法。第23行定义了`angle`方法，它返回复数的幅角。

第25行至第27行定义了`__add__`方法表示以当前对象为被加数的加法运算。对于一个Complex类的对象`u`而言：表达式`u+v`是`u.__add__(v)`的简化形式，这里`u`是被加数；表达式`v+u`是`u.__radd__(v)`的简化形式，这里`u`是加数。第26行将加数`other`转换为Complex类的对象，然后进行复数的加法运算。加法运算是可交换的，表达式`u+v`的值等于表达式`v+u`的值，因此第29行定义了`__radd__`方法等价于`__add__`方法。第31行至第33行定义了`__sub__`方法表示以当前对象为被减数的减法运算。对于一个Complex类的对象`u`而言：表达式`u-v`是`u.__sub__(v)`的简化形式，这里`u`是被减数；表达式`v-u`是`u.__rsub__(v)`的简化形式，这里`u`是减数。由于减法运算是不可交换的，第35行至第37行定义了`__rsub__`方法，它不等价于`__sub__`方法。第36行将被减数`other`转换为Complex类的对象。第37行等同于`return other.__sub__(self)`。第39行至第42行定义了`__mul__`方法表示以当前对象为被乘数的乘法运算。乘法运算是可交换的，因此第44行定义了`__rmul__`方法等价于`__mul__`方法。第46行至第51行定义了`__truediv__`方法表示以当前对象为被除数的除法运算。第46行至第51行定义了`__rtruediv__`方法表示以当前对象为除数的除法运算。

第46行至第51行定义了`__pow__`方法表示以当前对象为底数的乘方运算。第58行判断指数`n`是不是Complex类的对象。第59行在已知`n`是Complex类的对象的条件下判断`n`表示的复数的虚部是不是0。由于复数的实部和虚部的类型可以是`float`或`int`，`float`类型的计算结果可能存在误差，第59行调用的`is_zero`函数(定义在第76行)将一个实数等于0的条件定义为其绝对值不超过一个很小的阈值`tol`，其默认值为 10^{-15} 。第60行在已知`n`表示的复数的虚部非0的条件下判断当前对象`self`表示的复数的虚部是不是0。若虚部是0，第61行至第64行根据以下的演算计算乘方并返回结果：对于实数`t`和复数`n`，

¹<https://github.com/xbmc/python/blob/master/Demo/classes/Complex.py>

令 $n \ln t = a + bi$, 则 $t^n = e^{n \ln t} = e^{a+ib} = e^a(\cos b + i \sin b)$ 。第65行的else分支对应于底数和指数都是虚部非0的复数的乘方运算, 此时的结果是多值的, 第66行抛出异常表示未实现此种运算。第67行在已知n表示的复数的虚部为0的条件下将实部赋值给n。第68行至第70行根据以下的演算计算乘方并返回结果: 对于复数 t 和实数 n , 令 $t = re^{i\theta}$, 则 $t^n = r^n \cos n\theta + ir^n \sin n\theta$ 。第72行至第74行定义了__rpow__方法表示以当前对象为指数的乘方运算。第73行把底数转换为复数对象。第74行等同于return base.__pow__(self)。

第78行至第79行定义了is_Complex方法判断一个对象是不是Complex类的对象, 依据是它是否同时有re和im这两个属性。第81行至第90行定义了to_Complex方法将一个对象obj转换成Complex类的对象。若obj已经是Complex类的对象(第82行), 则返回obj(第83行)。若obj是一个元组, 并且其中包含的数据个数不超过2(超过2则报错), 则提取obj的数据(表达式*obj)作为实参调用Complex类的构造方法创建一个对象(第86行)。若以上条件都不满足, 则把obj作为实参调用Complex类的构造方法创建一个对象(第90行)。第92行至第93行定义了polar_to_Complex方法, 将极坐标表示的复数 $re^{i\phi}$ 转换为直角坐标表示 $r \cos \phi + ir \sin \phi$ 。第95行至第96行定义了Re方法, 它在确认形参obj是Complex类的对象后返回其re属性, 否则返回obj。第98行至第99行定义了Im方法, 它在确认形参obj是Complex类的对象后返回其im属性, 否则返回0。

第115行至第143行定义了test方法对测试Complex类实现的复数运算。第116行至第138行定义的字典testsuite包含了所有的测试数据。它的关键字是要测试的表达式, 每个表达式对应的值是一个由多个元组组成的列表。每个元组包含3条数据, 即表达式中的变量a和b的值以及表达式的正确求值结果。第139行至第143行的循环遍历testsuite, 对每个表达式和其对应的每个元组, 调用第101行至第113行定义的check函数进行测试。第104行调用内建函数eval对表达式求值, 求值环境包括当前可访问的所有全局和局部变量。例如第104行第一次运行时expr的值是'a+b', 形参a和b的值分别是Complex(0, 3)和2, eval将a和b的值代入expr中求值。第103行至第106行的try-except语句块表示若第104行在运行过程中抛出异常, 则在第106行输出出错信息并返回。第109行计算实际求值结果和正确求值结果之间的相对误差。第110行至第113行判断误差是否超过一个预先确定的阈值rel.tol(默认值为 10^{-6}), 若超过则报告此次测试发生错误。第146行定义的verbose变量是第147行调用test方法的实参, 用于控制程序的输出篇幅。若verbose的值设为True, 则会显示每条测试数据并在发生错误时报错。若verbose的值设为False, 则仅在发生错误时报错。

Listing 5.2: Complex类

```

1 import math
2
3 class Complex:
4     def __init__(self, re=0, im=0):
5         if isinstance(re, (float,int)) and \
6             isinstance(im, (float,int)):
7             _re = re; _im = im
8         else:

```

```
9         raise TypeError('Error: float or int expected')
10     self.__dict__['re'] = _re; self.__dict__['im'] = _im
11
12 def __setattr__(self, name, value):
13     raise TypeError('Error: Complex objects are immutable')
14
15 def __str__(self): return '(%g, %g)' % (self.re, self.im)
16
17 def __repr__(self): return 'Complex' + str(self)
18
19 def __abs__(self): return math.hypot(self.re, self.im)
20
21 abs = __abs__
22
23 def angle(self): return math.atan2(self.im, self.re)
24
25 def __add__(self, other):
26     other = to_Complex(other)
27     return Complex(self.re + other.re, self.im + other.im)
28
29 __radd__ = __add__
30
31 def __sub__(self, other):
32     other = to_Complex(other)
33     return Complex(self.re - other.re, self.im - other.im)
34
35 def __rsub__(self, other):
36     other = to_Complex(other)
37     return other - self
38
39 def __mul__(self, other):
40     other = to_Complex(other)
41     return Complex(self.re*other.re - self.im*other.im,
42                   self.re*other.im + self.im*other.re)
43
44 __rmul__ = __mul__
45
46 def __truediv__(self, other):
47     other = to_Complex(other)
```

```
48         d = float(other.re*other.re + other.im*other.im)
49         if is_zero(d): raise ZeroDivisionError('Error: division by 0')
50         return Complex((self.re*other.re + self.im*other.im) / d,
51                        (self.im*other.re - self.re*other.im) / d)
52
53     def __rtruediv__(self, other):
54         other = to_Complex(other)
55         return other / self
56
57     def __pow__(self, n):
58         if is_Complex(n):
59             if not is_zero(n.im):
60                 if is_zero(self.im):
61                     z = n * math.log(self.re)
62                     r = math.exp(z.re)
63                     return Complex(r * math.cos(z.im),
64                                   r * math.sin(z.im))
65             else:
66                 raise NotImplementedError('Complex to Complex')
67             n = n.re
68         r = self.abs() ** n
69         phi = n * self.angle()
70         return Complex(r * math.cos(phi), r * math.sin(phi))
71
72     def __rpow__(self, base):
73         base = to_Complex(base)
74         return pow(base, self)
75
76 def is_zero(x, tol = 1e-15): return abs(x) < tol
77
78 def is_Complex(obj):
79     return hasattr(obj, 're') and hasattr(obj, 'im')
80
81 def to_Complex(obj):
82     if is_Complex(obj):
83         return obj
84     elif isinstance(obj, tuple):
85         if len(obj) <= 2:
86             return Complex(*obj)
```



```

87         else:
88             raise TypeError('Error: ≤2 numbers expected')
89     else:
90         return Complex(obj)
91
92 def polar_to_Complex(r = 0, phi = 0):
93     return Complex(r * math.cos(phi), r * math.sin(phi))
94
95 def Re(obj):
96     return obj.re if is_Complex(obj) else obj
97
98 def Im(obj):
99     return obj.im if is_Complex(obj) else 0
100
101 def check(expr, a, b, value, verbose = False, rel_tol = 1e-6):
102     if verbose: print ('', a, 'and', b, end = '')
103     try:
104         result = eval(expr)
105     except:
106         print ('Error in evaluating' + expr); return
107
108     if verbose: print ('->', result)
109     rel_err = abs(result - value) / abs(value)
110     if rel_err > rel_tol:
111         print ('s for a=%s and b=%s = %s' % (expr, a, b, result))
112         print ('Correct value = %s Relative error = %f' %
113               (value, rel_err))
114
115 def test(verbose):
116     testsuite = {
117         'a+b': [ (Complex(0, 3), 2, Complex(2, 3)),
118                 (2, Complex(0, 3), Complex(2, 3)),
119                 (Complex(2,-3), Complex(-4,5), Complex(-2,2)) ],
120         'a-b': [ (Complex(0, 3), 2, Complex(-2, 3)),
121                 (2, Complex(0, 3), Complex(2, -3)),
122                 (Complex(2,-3), Complex(-4,5), Complex(6,-8)) ],
123         'a*b': [ (Complex(0, 3), 2, Complex(0, 6)),
124                 (2, Complex(0, 3), Complex(0, 6)),
125                 (Complex(2,-3), Complex(-4,5), Complex(7,22)) ],

```

```

126         'a/b': [ (Complex(0, 3), 2, Complex(0, 1.5)),
127                  (2, Complex(0, 3), Complex(0, -0.6666667)),
128                  (Complex(2, -3), Complex(-4, 5),
129                   Complex(-0.5609756, 0.04878049)) ],
130         'pow(a,b)': [ (Complex(2, -3), 2.3,
131                       Complex(-12.15244, -14.73536)),
132                       (2.3, Complex(2, -3),
133                        Complex(-4.234017, -3.171309)) ],
134         'polar_to_Complex(a.abs(), a.angle())': [
135             (Complex(0, -3), 0, Complex(0, -3)),
136             (Complex(2, -3), 0, Complex(2, -3)),
137             (Complex(-1, -3), 0, Complex(-1, -3)) ]
138     }
139     for expr in testsuite:
140         if verbose: print (expr + ':')
141         t = (expr,)
142         for item in testsuite[expr]:
143             check(*(t + item), verbose)
144
145 if __name__ == '__main__':
146     verbose = False
147     test(verbose)

```

5.1.3 一元多项式

程序5.3定义了一个类Polynomial表示一元多项式，并实现了一些基本运算(部分代码来源于[HL2020])。第3行至第6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阈值tol(默认值为 10^{-15})，则认为系数等于零，该项消失。第8行至第11行定义了__call__方法，对于一个多项式p(x)在x=t时求值可以写成简化形式“p(t)”，等同于“p.__call__(t)”。

第13行至第20行定义了多项式的加法运算。第14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的项，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20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。第22行至第32行定义了多项式的乘法运算。第23行定义了一个空的字典sum。第26行至第31行根据指数相加和系数相乘的规则执行乘法。第32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。第34行至第47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。

第49行和第51行分别创建了两个多项式对象p1和p2并输出其字符串表示。第53行计算它们的和并保存在多项式对象p3中。第55行计算它们的积并保存在多项式对象p4中。第57行输出对多项式p4在x=5时的求值结果。

Listing 5.3: Polynomial类

```
1 tol = 1E-15
2 class Polynomial:
3     def __init__(self, poly):
4         self.poly = {}
5         for power in poly:
6             if abs(poly[power]) > tol: self.poly[power] = poly[power]
7
8     def __call__(self, x):
9         value = 0.0
10        for power in self.poly: value += self.poly[power]*x**power
11        return value
12
13    def __add__(self, other):
14        sum = self.poly.copy()    # print (id(sum), id(self.poly))
15        for power in other.poly:
16            if power in sum:
17                sum[power] += other.poly[power]
18            else:
19                sum[power] = other.poly[power]
20        return Polynomial(sum)
21
22    def __mul__(self, other):
23        sum = {}
24        for self_power in self.poly:
25            for other_power in other.poly:
26                power = self_power + other_power
27                m = self.poly[self_power] * other.poly[other_power]
28                if power in sum:
29                    sum[power] += m
30                else:
31                    sum[power] = m
32        return Polynomial(sum)
33
34    def __str__(self):
```

```

35     s = ''
36     for power in sorted(self.poly):
37         s += '␣+␣%g*x^%d' % (self.poly[power], power)
38     s = s.replace('+␣-', '-␣')
39     s = s.replace('x^0', '1')
40     s = s.replace('␣1*', '␣')
41     s = s.replace('x^1␣', 'x␣')
42     # s = s.replace('x^1', 'x') replaces x^100 by x^00
43     if s[0:3] == '␣+␣': # remove initial +
44         s = s[3:]
45     if s[0:3] == '␣-␣': # fix spaces for initial -
46         s = '-' + s[3:]
47     return s
48
49 p1 = Polynomial({0: -1, 2: 1, 7: 3}); print (p1)
50 # -1 + x^2 + 3*x^7
51 p2 = Polynomial({0: 1, 2: -1, 5: -2, 3: 4}); print (p2)
52 # 1 - x^2 + 4*x^3 - 2*x^5
53 p3 = p1 + p2; print (p3)
54 # 4*x^3 - 2*x^5 + 3*x^7
55 p4 = p1 * p2; print (p4)
56 # -1 + 2*x^2 - 4*x^3 - x^4 + 6*x^5 + x^7 - 3*x^9 + 12*x^10 - 6*x^12
57 print (p4(5)) # -1353419826.0

```

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:

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

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

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

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

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[HL2020]。

程序5.4中第1行至第11行定义了Diff类。第2行至第5行定义了构造方法用以初始化所有属性。

dfdx_exact表示函数 $f(x)$ 的解析形式的一阶导函数 $f'(x)$ ，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。第7行至第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)。第13行至第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。第34行至第48行的table函数基于输入函数f、自变量x、步长值列表h_values、公式类列表methods和函数 $f(x)$ 的解析形式的一阶导函数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。第53行至第57行使用SymPy库(第十章)计算函数 $g(x) = e^{x \sin x}$ 的解析形式的一阶导函数 $g'(x) = (x \cos x + \sin x)e^{x \sin x}$ 并保存在dgdx中。第59行至第60行调用table函数，其输出结果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

```

1 class Diff:
2     def __init__(self, f, h=1E-5, dfdx_exact=None):
3         self.f = f
4         self.h = float(h)
5         self.exact = dfdx_exact
6
7     def get_error(self, x):
8         if self.exact is not None:
9             df_numerical = self(x)
10            df_exact = self.exact(x)
11            return abs( (df_exact - df_numerical) / df_exact )
12
13 class Forward1(Diff):

```

```
14     def __call__(self, x):
15         f, h = self.f, self.h
16         return (f(x+h) - f(x))/h
17
18 class Backward1(Diff):
19     def __call__(self, x):
20         f, h = self.f, self.h
21         return (f(x) - f(x-h))/h
22
23 class Central2(Diff):
24     def __call__(self, x):
25         f, h = self.f, self.h
26         return (f(x+h) - f(x-h))/(2*h)
27
28 class Central4(Diff):
29     def __call__(self, x):
30         f, h = self.f, self.h
31         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 ('%-8.6f' % 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, 7)],
60       methods=[Forward1, Central2, Central4], dfdx=dgdx)
```

Listing 5.5: 程序5.4的输出结果

1	h	1.00e-01	1.00e-02	1.00e-03	1.00e-04	1.00e-05	1.00e-06
2	Forward1	0.104974	0.010906	0.001095	0.000110	0.000011	0.000001
3	Central2	0.004611	0.000046	0.000000	0.000000	0.000000	0.000000
4	Central4	0.000080	0.000000	0.000000	0.000000	0.000000	0.000000

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 and 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
```

5.3 实验5：类和继承

实验目的

本实验的目的是掌握以下内容：定义和使用类，通过继承和覆盖实现代码复用。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

1. 表示有理数的类

有理数的一般形式是 a/b ，其中 a 是整数， b 是正整数，并且当 a 非0时 $|a|$ 和 b 的最大公约数是1。实现Rational类表示有理数和其运算。5.7已列出了部分代码，需要实现标注了“to be implemented”的函数。Rational类的属性nu和de分别表示分子和分母。函数__add__、__sub__、__mul__和__truediv__分别进行加减乘除运算，然后返回一个新创建的Rational对象作为运算结果。函数__eq__、__ne__、__gt__、__lt__、__ge__和__le__比较两个有理数，返回一个bool类型的值。这些函数对应的比较运算符分别是：==、!=、>、<、>=、<=。例如表达式Rational(6, -19) > Rational(14, -41) 在求值时被转换成方法调用Rational(6, -19).__gt__(Rational(14, -41))。函数test测试这些函数。gcd函数要求形参a和b都是正整数，如果其中出现0或负数，递归不会终止。

Listing 5.7: 表示有理数的类

```
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 class Rational:
10     def __init__(self, n=0, d=1):
11         # to be implemented
12         # e.g. transform 120/-64 to -15/8
13         _nu = n; _de = d
14         self.__dict__['nu'] = _nu; self.__dict__['de'] = _de
15
16     def __setattr__(self, name, value):
17         raise TypeError('Error: Rational objects are immutable')
18
19     def __str__(self): return '%d/%d' % (self.nu, self.de)
20
21     def __add__(self, other):
22         # to be implemented
23
24     def __sub__(self, other):
25         # to be implemented
26
27     def __mul__(self, other):
28         # to be implemented
29
30     def __truediv__(self, other):
31         # to be implemented
32
33     def __eq__(self, other):
34         # to be implemented
35
36     def __ne__(self, other):
37         # to be implemented
38
39     def __gt__(self, other):
40         # to be implemented
41
42     def __lt__(self, other):
43         # to be implemented
44
45     def __ge__(self, other):
46         # to be implemented
```

```

47
48     def __le__(self, other):
49         # to be implemented
50
51 def test():
52     testsuite = [
53         ('Rational(2, 3) + Rational(-70, 40)', Rational(-13, 12)),
54         ('Rational(-20, 3) - Rational(120, 470)', Rational(-976, 141)),
55         ('Rational(-6, 19) * Rational(-114, 18)', Rational(2, 1)),
56         ('Rational(-6, 19) / Rational(-114, -28)', Rational(-28, 361)),
57
58         ('Rational(-6, 19) == Rational(-14, 41)', False),
59         ('Rational(-6, 19) != Rational(-14, 41)', True),
60         ('Rational(6, -19) > Rational(14, -41)', True),
61         ('Rational(-6, 19) < Rational(-14, 41)', False),
62         ('Rational(-6, 19) >= Rational(-14, 41)', True),
63         ('Rational(6, -19) <= Rational(14, -41)', False),
64         ('Rational(-15, 8) == Rational(120, -64)', True),
65     ]
66     for t in testsuite:
67         try:
68             result = eval(t[0])
69         except:
70             print('Error in evaluating' + t[0]); continue
71
72         if result != t[1]:
73             print('Error: %s != %s' % (t[0], t[1]))
74
75 if __name__ == '__main__':
76     test()

```

2. 定积分的数值计算

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

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

其中 w_i 是函数值 $f(x_i)$ 的权重, 称为积分系数。不同的数值计算公式的区别体现在积分节点和积分系数上。

在程序5.8中实现Integrator类的integrate方法和它的三个子类, 分别对应表5.1中的三种公式。在每个

公式名称	积分节点的坐标和积分系数
复合梯形(Trapezoidal)公式	$x_i = a + ih \quad \text{for } i = 0, \dots, n, \quad h = \frac{b-a}{n},$ $w_0 = w_n = \frac{h}{2}, \quad w_i = h \quad \text{for } i = 1, \dots, n-1$
复合辛普森(Simpson)公式 n 必须是偶数 若输入的 n 是奇数, 则执行 $n = n + 1$	$x_i = a + ih \quad \text{for } i = 0, \dots, n, \quad h = \frac{b-a}{n},$ $w_0 = w_n = \frac{h}{3}, \quad w_i = \frac{2h}{3} \quad \text{for } i = 2, 4, \dots, n-2,$ $w_i = \frac{4h}{3} \quad \text{for } i = 1, 3, \dots, n-1$
复合高斯-勒让德(Gauss-Legendre)公式 n 必须是奇数 若输入的 n 是偶数, 则执行 $n = n + 1$	$x_i = a + \frac{i+1}{2}h - \frac{\sqrt{3}}{6}h \quad \text{for } i = 0, 2, \dots, n-1,$ $x_i = a + \frac{i}{2}h + \frac{\sqrt{3}}{6}h \quad \text{for } i = 1, 3, \dots, n,$ $h = \frac{2(b-a)}{n+1}, \quad w_i = \frac{h}{2}, \quad \text{for } i = 0, 1, \dots, n$

表 5.1: 定积分的几种数值计算公式

子类中只需覆盖父类的compute_points方法计算并返回两个列表，它们分别存储了所有积分节点的坐标和积分系数。test()函数用函数 $f(x) = (x \cos x + \sin x)e^{x \sin x}$ 和它的解析形式的积分函数 $F(x) = e^{x \sin x}$ 测试这三个公式的精确度。

Listing 5.8: 定积分的数值计算

```
1 import math
2
3 class Integrator:
4     def __init__(self, a, b, n):
5         self.a, self.b, self.n = a, b, n
6         self.points, self.weights = self.compute_points()
7
8     def compute_points(self):
9         raise NotImplementedError(self.__class__.__name__)
10
11     def integrate(self, f):
12         # to be implemented: do the summation in equation 5.9 using
13         # self.points and self.weights
14
15 class Trapezoidal(Integrator):
16     def compute_points(self):
17         # to be implemented
18
19 class Simpson(Integrator):
20     def compute_points(self):
21         # to be implemented
22
23 class GaussLegendre(Integrator):
```

```
24     def compute_points(self):
25         # to be implemented
26
27 def test():
28     def f(x): return (x * math.cos(x) + math.sin(x)) * \
29                     math.exp(x * math.sin(x))
30     def F(x): return math.exp(x * math.sin(x))
31
32     a = 2; b = 3; n = 200
33     I_exact = F(b) - F(a)
34     tol = 1E-3
35
36     methods = [Trapezoidal, Simpson, GaussLegendre]
37     for method in methods:
38         integrator = method(a, b, n)
39         I = integrator.integrate(f)
40         rel_err = abs((I_exact - I) / I_exact)
41         print ('%s: %g' % (method.__name__, rel_err))
42         if rel_err > tol:
43             print ('Error in %s' % method.__name__)
44
45 if __name__ == '__main__':
46     test()
```

第六章 NumPy数组和矩阵计算

NumPy扩展库[NumPyDoc]定义了由同类型的数据组成的多维数组`ndarray`及其常用运算，`ndarray`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`(维数)、`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)。

矩阵可以使用`numpy.matrix`类或`numpy.array`类表示。SciPy扩展库的`scipy.linalg`模块定义了常用的矩阵计算函数[SciPyDoc]。

6.1 创建数组

6.1.1 已有数据存储在其它类型的容器中

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

```
1 In[1]: import numpy as np
2 In[2]: a = np.array([2, 8, 64]); a
3 Out[2]: array([2, 8, 64])
4 In[3]: a.dtype, a.ndim, a.shape, a.size
5 Out[3]: (dtype('int32'), 1, (3,), 3)
6 In[4]: b = np.array([3.14, 2.71, 6.83, -8.34])
7 In[5]: b.dtype, b.ndim, b.shape, b.size
8 Out[5]: (dtype('float64'), 1, (4,), 4)
9 In[6]: c = np.array([(1, 2.4), (6, -3), (8, -5)])
10 In[7]: c.ndim, c.shape, c.size
11 Out[7]: (2, (3, 2), 6)
12 In[8]: d = np.array([95536, 2.71, 6, -8.34], dtype=np.int16); d
13 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。`np.arange`函数根据下界、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。`np.linspace`函数根据下界、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。

`np.random.default_rng`函数使用指定的种子创建一个随机数生成器，可用来初始化一个指定形状的随机数组。如果不提供种子，则由操作系统自动生成一个种子。`np.fromfunction`函数创建指定形状的数组，每个元素的值是这个元素的索引值的函数，该函数是传给`np.fromfunction`的第一个实参。

Listing 6.2: 创建数组

```
1 In[1]: a = np.zeros((2, 3)); a
2 Out[1]:
3 array([[0., 0., 0.],
4        [0., 0., 0.]])
5 In[2]: np.ones((3, 2))
6 Out[2]:
7 array([[1., 1.],
8        [1., 1.],
9        [1., 1.]])
10 In[3]: c = np.ones_like(a); c
11 Out[3]:
12 array([[1., 1., 1.],
13        [1., 1., 1.]])
14 In[4]: np.arange(2, 30, 7)
15 Out[4]: array([ 2,  9, 16, 23])
16 In[5]: np.arange(0.2, 3.01, 0.7)
17 Out[5]: array([0.2, 0.9, 1.6, 2.3, 3. ])
18 In[6]: np.arange(6)
19 Out[6]: array([0, 1, 2, 3, 4, 5])
20 In[7]: np.linspace(0, 3, 7)
21 Out[7]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. ])
22 In[8]: rg = np.random.default_rng(218); rg.random((3, 2))
23 Out[8]:
24 array([[0.98049392, 0.59814763],
25        [0.79896601, 0.63727534],
26        [0.15966292, 0.2596797 ]])
27 In[9]: def f(x, y): return (x + 2) ** 2 + y ** 3
28 In[10]: np.fromfunction(f, (2, 3), dtype=int)
29 Out[10]: array([[ 4,  5, 12],
```

```
[ 9, 10, 17]])
```

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方法从一个多维数组生成一维数组。reshape方法从原数组生成一个指定形状的新数组。T方法从一个数组h生成它的转置。以上方法不会改变原数组的形状。resize方法则将原数组改变为指定的形状。

Listing 6.3: 改变形状

```
1 In[1]: h=np.arange(1,13).reshape(3,4); h
2 Out[1]:
3 array([[ 1,  2,  3,  4],
4         [ 5,  6,  7,  8],
5         [ 9, 10, 11, 12]])
6 In[2]: h.flatten()
7 Out[2]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
8 In[3]: h.reshape(2, 6)
9 Out[3]:
10 array([[ 1,  2,  3,  4,  5,  6],
11         [ 7,  8,  9, 10, 11, 12]])
12 In[4]: h.T
13 Out[4]:
14 array([[ 1,  5,  9],
15         [ 2,  6, 10],
16         [ 3,  7, 11],
17         [ 4,  8, 12]])
18 In[5]: h
19 Out[5]:
20 array([[ 1,  2,  3,  4],
21         [ 5,  6,  7,  8],
22         [ 9, 10, 11, 12]])
23 In[6]: h.resize(2, 6); h
24 Out[6]:
25 array([[ 1,  2,  3,  4,  5,  6],
26         [ 7,  8,  9, 10, 11, 12]])
```

6.1.4 数组的堆叠(Stacking)

np.hstack函数沿第二个维度将两个数组堆叠在一起形成新的数组，np.vstack函数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

```
1 In[1]: a = np.arange(1, 7).reshape(2,3); a
2 Out[1]:
3 array([[1, 2, 3],
4        [4, 5, 6]])
5 In[2]: b = np.arange(7, 13).reshape(2,3); b
6 Out[2]:
7 array([[ 7,  8,  9],
8        [10, 11, 12]])
9 In[3]: np.hstack((a, b))
10 Out[3]:
11 array([[ 1,  2,  3,  7,  8,  9],
12        [ 4,  5,  6, 10, 11, 12]])
13 In[4]: np.vstack((a, b))
14 Out[4]:
15 array([[ 1,  2,  3],
16        [ 4,  5,  6],
17        [ 7,  8,  9],
18        [10, 11, 12]])
```

6.1.5 数组的分割

np.hsplit函数沿第二个维度将一个数组分割成为多个数组，可以指定一个正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各分割点的索引值。 np.vsplit函数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

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



```
9  array([[ 4,  5,  6],
10         [16, 17, 18]]),
11  array([[ 7,  8,  9],
12         [19, 20, 21]]),
13  array([[10, 11, 12],
14         [22, 23, 24]])]
15 In[3]: np.hsplit(c, (4, 7, 9))
16 Out[3]:
17 [array([[ 1,  2,  3,  4],
18         [13, 14, 15, 16]]),
19  array([[ 5,  6,  7],
20         [17, 18, 19]]),
21  array([[ 8,  9],
22         [20, 21]]),
23  array([[10, 11, 12],
24         [22, 23, 24]])]
25 In[4]: d = np.arange(1, 25).reshape(6,4); d
26 Out[4]:
27 array([[ 1,  2,  3,  4],
28        [ 5,  6,  7,  8],
29        [ 9, 10, 11, 12],
30        [13, 14, 15, 16],
31        [17, 18, 19, 20],
32        [21, 22, 23, 24]])
33 In[5]: np.vsplit(d, (2, 3, 5))
34 Out[5]:
35 [array([[1, 2, 3, 4],
36        [5, 6, 7, 8]]),
37  array([[ 9, 10, 11, 12]]),
38  array([[13, 14, 15, 16],
39        [17, 18, 19, 20]]),
40  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。两个二维数组之间的矩阵乘法可通过@运算符或dot方法完成。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。

Listing 6.6: 基本运算

```

1 In[1]: a = np.arange(6).reshape(2, 3); a
2 Out[1]:
3 array([[0, 1, 2],
4        [3, 4, 5]])
5 In[2]: b = np.arange(2,18,3).reshape(2, 3); b
6 Out[2]:
7 array([[ 2,  5,  8],
8        [11, 14, 17]])
9 In[3]: a+b, a-b, a*b, a/b, -a, -b+(a**np.e-0.818*b+6)**(-np.pi)
10 Out[3]:
11 (array([[ 2,  6, 10],
12         [14, 18, 22]]),
13  array([[ -2,  -4,  -6],
14         [-8, -10, -12]]),
15  array([[ 0,  5, 16],
16         [33, 56, 85]]),
17  array([[0.          , 0.2          , 0.25          ],
18         [0.27272727, 0.28571429, 0.29411765]]),
19  array([[ 0, -1, -2],
20         [-3, -4, -5]]),
21  array([[ -1.99023341,  -4.96511512,  -7.99647626],
22         [-10.99985895, -13.99998898, -16.99999851]]))
23 In[4]: c = b.reshape(3, 2); c
24 Out[4]:
25 array([[ 2,  5],
26        [ 8, 11],
27        [14, 17]])
28 In[5]: a@c, a.dot(c)
29 Out[5]:
30 (array([[ 36,  45],
31         [108, 144]]),
32  array([[ 36,  45],
33         [108, 144]]))
34 In[6]: d=a*3+b; b -= a; d, b

```

```

35 Out[6]:
36 (array([[ 2,  8, 14],
37        [20, 26, 32]]),
38  array([[ 2,  4,  6],
39        [ 8, 10, 12]]))
40 In[7]: rg = np.random.default_rng(); e = rg.random((2, 3)); e
41 Out[7]:
42 array([[0.31724113, 0.28135186, 0.19364362],
43        [0.70296297, 0.76202543, 0.95436982]])
44 In[8]: f = e + a - 2*b; f, f.dtype
45 Out[8]:
46 (array([[ -3.68275887,  -8.71864814, -13.80635638],
47        [-18.29703703, -23.23797457, -28.04563018]]),
48  dtype('float64'))

```

6.2.2 函数运算

sum、min和max方法分别返回一个数组包含的所有元素的总和、最大值和最小值。np.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

```

1 In[1]: g = np.array([[2,6,5],[4,1,3]]); g
2 Out[1]: array([[2, 6, 5],
3              [4, 1, 3]])
4 In[2]: g.sum(), g.max(), g.min()
5 Out[2]: (21, 6, 1)
6 In[3]: g.max(axis=0), g.max(axis=1)
7 Out[3]: (array([4, 6, 5]), array([6, 4]))
8 In[4]: g.min(axis=0), g.min(axis=1)
9 Out[4]: (array([2, 1, 3]), array([2, 1]))
10 In[5]: np.sort(g)                                # sort along the last axis
11 Out[5]: array([[2, 5, 6],
12              [1, 3, 4]])
13 In[6]: np.sort(g, axis=None)                      # sort the flattened array
14 Out[6]: array([1, 2, 3, 4, 5, 6])
15 In[7]: np.sort(g, axis=0)                          # sort along the first axis
16 Out[7]: array([[2, 1, 3],

```

```

17         [4, 6, 5]))
18 In[8]: np.sort?
19 Out[8]: Signature: np.sort(a, axis=-1, kind=None, order=None)
20 Docstring:
21 Return a sorted copy of an array.
22
23 Parameters
24 -----
25 a : array_like
26     Array to be sorted.
27 axis : int or None, optional
28     .....
29 In[9]: np.sqrt(b) + np.exp(a - 5) * np.cos(e**1.3 - f)
30 Out[9]:
31 array([[1.40958124, 2.22018241, 2.83965649],
32        [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

```

1 In[1]: a = np.arange(1, 16, 2)**2; a
2 Out[1]: array([ 1, 9, 25, 49, 81, 121, 169, 225], dtype=int32)
3 In[2]: a[3], a[1:7:2]
4 Out[2]: (49, array([ 9, 49, 121], dtype=int32))
5 In[3]: a[:6:3] = 361; a
6 Out[3]: array([361, 9, 25, 361, 81, 121, 169, 225], dtype=int32)
7 In[4]: a[::-1]
8 Out[4]: array([225, 169, 121, 81, 361, 25, 9, 361], dtype=int32)
9 In[5]: for i in a: print(np.sqrt(i), end=' ')
10 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。将二维数组看成是一个矩阵，对于二维数组的索引和迭代以矩阵的行为单位。对于一个二维数组a，a.flat是一个迭代器，可用在for循环中以每个元素为单位进行迭代。

数组不仅可以使⽤整数和切片作为索引，也可以使⽤整数数组(或列表)或布尔值数组(或列表)作为索引。使⽤一个整数数组a作为一个数组b的索引得到的结果是一个数组c，c中的每个元素是以a中的每个元素作为索引值所获取的数组b中的对应元素。使⽤一个布尔值数组a作为一个数组b的索引得到的结果是一个数组c，c中仅包含以数组a中的值为True的元素的索引值所获取的数组b中的对应元素。对于多维数组，可分别对每个维度索引。

对于一个二维数组，np.ix_函数使⽤两个整数数组作为行和列的索引，结果包含了行索引值属于第一个数组并且列索引值属于第二个数组的所有元素(Out[9])。对于一个二维数组，直接使⽤两个长度均为n的整数数组a和b作为行和列的索引，结果包含n个元素，其中第i个元素($0 \leq i \leq n-1$)的行索引值和列索引值分别为a[i]和b[i](Out[10])。

Listing 6.9: 多维数组的索引、切片和迭代

```

1 In[1]: def f(x, y): return x * 4 + y + 1
2 In[2]: h = np.fromfunction(f, (3, 4), dtype=int); h
3 Out[2]:
4 array([[ 1,  2,  3,  4],
5         [ 5,  6,  7,  8],
6         [ 9, 10, 11, 12]])
7 In[3]: h[1, 2], h[0, 3], h[2, 2]
8 Out[3]: (7, 4, 11)
9 In[4]: h[1:3], h[1:3,], h[1:3,:], h[0]
10 Out[4]:
11 (array([[ 5,  6,  7,  8],
12         [ 9, 10, 11, 12]]),
13  array([[ 5,  6,  7,  8],
14         [ 9, 10, 11, 12]]),
15  array([[ 5,  6,  7,  8],
16         [ 9, 10, 11, 12]]),
17  array([1, 2, 3, 4]))
18 In[5]: h[:, 1:4:2], h[:, 3:1:-1], h[:, -2]
19 Out[5]:
20 (array([[ 2,  4],
21         [ 6,  8],
22         [10, 12]]),
23  array([[ 4,  3],
24         [ 8,  7],
25         [12, 11]]),
26  array([ 3,  7, 11]))
27 In[6]: for row in h: print(row)

```

```
28 Out[6]:
29 [1 2 3 4]
30 [5 6 7 8]
31 [ 9 10 11 12]
32 In[7]: for element in h.flat: print(element, end=' ')
33 Out[7]: 1 2 3 4 5 6 7 8 9 10 11 12
34 In[8]: h[np.ix_([0,2], [1])] # rows 0 and 2, then column 1
35 Out[8]:
36 array([[ 2],
37        [10]])
38 In[9]: h[np.ix_([0, 2], [0, 2])] # rows 0 and 2, then column 0 and 2
39 Out[9]:
40 array([[ 1,  3],
41        [ 9, 11]])
42 In[10]: h[[0, 2], [0, 2]] # column 0 from row 0 and column 2 from row 2
43 Out[10]: array([ 1, 11])
44 In[11]: h[[0, 2]]
45 Out[11]:
46 array([[ 1,  2,  3,  4],
47        [ 9, 10, 11, 12]])
48 In[12]: h[:, [0,2]]
49 Out[12]:
50 array([[ 1,  3],
51        [ 5,  7],
52        [ 9, 11]])
53 In[13]: h[1:3, 0:3]
54 Out[13]:
55 array([[ 5,  6,  7],
56        [ 9, 10, 11]])
57 In[14]: j = np.arange(24).reshape(3, 2, 4); j
58 Out[14]:
59 array([[[ 0,  1,  2,  3],
60         [ 4,  5,  6,  7]],
61        [[ 8,  9, 10, 11],
62         [12, 13, 14, 15]],
63        [[16, 17, 18, 19],
64         [20, 21, 22, 23]])
```

```

67 In[15]: j[2, ...]
68 Out[15]:
69 array([[16, 17, 18, 19],
70        [20, 21, 22, 23]])
71 In[16]: j[:, 1:2, :]
72 Out[16]:
73 array([[[ 4,  5,  6,  7]],
74        [[12, 13, 14, 15]],
75        [[20, 21, 22, 23]])]
76
77
78 In[17]: j[..., 1:3]
79 Out[17]:
80 array([[[ 1,  2],
81        [ 5,  6]],
82        [[ 9, 10],
83        [13, 14]],
84        [[17, 18],
85        [21, 22]])]

```

Listing 6.10: 使用整数数组作为索引

```

1 In[1]: a = np.arange(1, 16, 2)**2; a
2 Out[1]: array([ 1,  9, 25, 49, 81, 121, 169, 225])
3 In[2]: i = np.array([3, 2, 7, 3, 5]); a[i]
4 Out[2]: array([ 49, 25, 225, 49, 121], dtype=int32)
5 In[3]: j = np.array([[3, 2, 4], [1, 5, 6]]); a[j]
6 Out[3]:
7 array([[ 49, 25, 81],
8        [ 9, 121, 169]], dtype=int32)
9 In[4]: b = a.reshape(4,2); b
10 Out[4]:
11 array([[ 1,  9],
12        [25, 49],
13        [81, 121],
14        [169, 225]], dtype=int32)
15 In[5]: b[np.array([2, 3, 1, 2])]
16 Out[5]:

```

```

17 array([[ 81, 121],
18        [169, 225],
19        [ 25,  49],
20        [ 81, 121]]], dtype=int32)
21 In[6]: b[np.array([[2, 3], [1, 2]])]
22 Out[6]:
23 array([[[ 81, 121],
24         [169, 225]],
25
26        [[ 25,  49],
27         [ 81, 121]]], dtype=int32)
28 In[7]: i1 = np.array([[3, 2], # row indices
29                      [2, 1]])
30 In[8]: i2 = np.array([[0, 1], # column indices
31                      [1, 0]])
32 In[9]: b[i1, i2]
33 Out[9]:
34 array([[169, 121],
35        [121,  25]], dtype=int32)
36 In[10]: b[i1, i2] = 36; a
37 Out[10]: 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`)的最大值的索引值。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: 使用整数数组作为索引

```

1 In[1]: data = np.cos(np.arange(103, 123)).reshape(5, 4); data
2 Out[1]:
3 array([[ -0.78223089, -0.94686801, -0.24095905,  0.68648655],
4        [ 0.98277958,  0.3755096 , -0.57700218, -0.99902081],
5        [-0.50254432,  0.4559691 ,  0.99526664,  0.61952061],
6        [-0.32580981, -0.97159219, -0.7240972 ,  0.18912942],
7        [ 0.92847132,  0.81418097, -0.04866361, -0.86676709]])
8 In[2]: maxind0 = data.argmax(axis=0); maxind0
9 Out[2]: array([1, 4, 2, 0], dtype=int32)
10 In[3]: data_max0 = data[maxind0, range(data.shape[1])]; data_max0
11 Out[3]: array([0.98277958, 0.81418097, 0.99526664, 0.68648655])
12 In[2]: maxind1 = data.argmax(axis=1); maxind1
13 Out[2]: array([3, 0, 2, 3, 0], dtype=int32)
14 In[3]: data_max1 = data[range(data.shape[0]), maxind1]; data_max1

```



```

15 Out[3]: array([0.68648655, 0.98277958, 0.99526664, 0.18912942,
16             0.92847132])

```

Listing 6.12: 使用布尔值数组作为索引

```

1 In[1]: a = np.arange(1, 16, 2)**2; a
2 Out[1]: array([ 1,  9, 25, 49, 81, 121, 169, 225], dtype=int32)
3 In[2]: g = a > 50; g
4 Out[2]: array([False, False, False, False, True, True, True, True])
5 In[3]: a[g] = 0; a
6 Out[3]: array([ 1,  9, 25, 49,  0,  0,  0,  0], dtype=int32)
7 In[4]: b = a.reshape(2, 4); b
8 Out[4]:
9 array([[ 1,  9, 25, 49],
10        [ 0,  0,  0,  0]], dtype=int32)
11 In[5]: i1 = np.array([False, True]); b[i1, :]
12 Out[5]: array([[0, 0, 0, 0]], dtype=int32)
13 In[6]: i2 = np.array([True, False, False, True]); b[:, i2]
14 Out[6]:
15 array([[ 1, 49],
16        [ 0,  0]], 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。

Out[4]行的输出结果是：k is h的值为True，并且k和h的id值相同。这说明In[3]行的赋值运算给已有数组h创建一个别名k，而不发生数据的复制。Python语言的函数调用对于可变实参是传引用而非传值，所以也不发生数据的复制。

view方法从已有数组创建一个新的数组(Out[6]行m is h的值为False)，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(Out[6]行m.base is h的值为True并且m.flags.owndata的值为False)，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方法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。reshape方法从原数组生成一个不同形状的视图。

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 (Out[12]行v.base is h的值为False并且v.flags.owndata的值为True)。copy方法的一个用途是复制数据以后可以用del回收原数组占用的内存空间。

Listing 6.13: 复制和视图

```
1 In[1]: def f(x, y): return x * 4 + y + 1
2 In[2]: h = np.fromfunction(f, (3, 4), dtype=int); h
3 Out[2]:
4 array([[ 1,  2,  3,  4],
5         [ 5,  6,  7,  8],
6         [ 9, 10, 11, 12]])
7 In[3]: k = h
8 In[4]: k is h, id(k), id(h)
9 Out[4]: (True, 186428160, 186428160)
10 In[5]: m = h.view()
11 In[6]: m is h, m.base is h, m.flags.owndata
12 Out[6]: (False, True, False)
13 In[7]: m.resize((2, 6)); h.shape
14 Out[7]: (3, 4)
15 In[8]: m[1, 3] = 16; m, h
16 Out[8]:
17 (array([[ 1,  2,  3,  4,  5,  6],
18         [ 7,  8,  9, 16, 11, 12]]),
19  array([[ 1,  2,  3,  4],
20         [ 5,  6,  7,  8],
21         [ 9, 16, 11, 12]]))
22 In[9]: t = h[0:2, 1:3]; t
23 Out[9]:
24 array([[2, 3],
25        [6, 7]])
26 In[10]: t[1, 0] = 20; h
27 Out[10]:
28 array([[ 1,  2,  3,  4],
29        [ 5, 20,  7,  8],
30        [ 9, 16, 11, 12]])
31 In[11]: v = h.copy()
32 In[12]: v is h, v.base is h, v.flags.owndata
33 Out[12]: (False, False, True)
34 In[13]: v[1, 1] = 36; v[1, 1], h[1, 1]
35 Out[13]: (36, 20)
36 In[14]: p = h.ravel(); p
37 Out[14]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 16, 11, 12])
38 In[15]: p[9]=99; h
```

```
39 Out[15]:
40 array([[ 1,  2,  3,  4],
41        [ 5,  6,  7,  8],
42        [ 9, 99, 11, 12]])
43 In[16]: a = np.arange(1000000); b = a[:100].copy()
44 In[17]: del a # the memory of array 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`扩展库的`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。

`inv`函数计算一个矩阵的逆矩阵。`det`函数计算一个矩阵的行列式。`norm`函数对于一个矩阵 \mathbf{A} 计算其Frobenius范数 $\sqrt{\text{trace}(\mathbf{A}^H \mathbf{A})}$ ，对于一个向量 \mathbf{x} 计算其欧式范数(各分量的平方和的平方根) $\sqrt{\sum_i |x_i|^2}$ 。`solve(\mathbf{A} , \mathbf{b})`函数求解线性方程组 $\mathbf{Ax} = \mathbf{b}$ 。`eig(\mathbf{A})`函数返回一个元组，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(singular value decomposition)
[TL2019]为 $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = [\mathbf{u}_1, \dots, \mathbf{u}_m]\mathbf{\Sigma}[\mathbf{v}_1, \dots, \mathbf{v}_n]^T$ ，其中 $\mathbf{\Sigma} = \left(\begin{array}{c|c} \text{diag}(\sigma_1, \dots, \sigma_r) & \mathbf{0}_{r, n-r} \\ \hline \mathbf{0}_{m-r, r} & \mathbf{0}_{m-r, n-r} \end{array} \right)$ ，奇异值为 $\sigma_1, \dots, \sigma_r$ ，奇异向量为 $\mathbf{u}_1, \dots, \mathbf{u}_m$ 和 $\mathbf{v}_1, \dots, \mathbf{v}_n$ ，且

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

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

`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。`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。

`lu(\mathbf{A})`实现了LU分解(LU decomposition)，即将一个矩阵 \mathbf{A} 分解为一个置换矩阵 \mathbf{P} 、一个对角线上元素均为1的下三角矩阵 \mathbf{L} 和一个上三角矩阵 \mathbf{U} 的乘积： $\mathbf{A} = \mathbf{PLU}$ 。`np.allclose`函数判断两个数组在指定的相对误差(用关键字参数`rtol`指定，默认值为 10^{-5})和绝对误差(用关键字参数`atol`指定，默认值为 10^{-8})下是否相等。

`scipy.linalg`模块还实现了Cholesky分解、QR分解和Schur分解[SciPyDoc]。

Listing 6.14: 矩阵计算

```
1 In[1]: import numpy as np
```

```
2 In[2]: A = np.mat('[4 3;2 1]'); A
3 Out[2]: matrix([[4, 3],
4           [2, 1]])
5 In[3]: A.I # inverse of A
6 Out[3]: matrix([[ -0.5,  1.5],
7           [ 1. , -2. ]])
8 In[4]: b = np.mat('[6 5]'); b
9 Out[4]: matrix([[6, 5]])
10 In[5]: b.T # matrix transpose of b
11 Out[5]:
12 matrix([[6],
13          [5]])
14 In[6]: A*b.T
15 Out[6]:
16 matrix([[39],
17          [17]])
18 In[7]: A = np.array([[4,3],[2,1]]); A
19 Out[7]:
20 array([[4, 3],
21        [2, 1]])
22 In[8]: from scipy import linalg; linalg.inv(A)
23 Out[8]:
24 array([[ -0.5,  1.5],
25        [ 1. , -2. ]])
26 In[9]: b = np.array([[6,5]]); b # 2D array
27 Out[9]: array([[6, 5]])
28 In[10]: b.T
29 Out[10]:
30 array([[6],
31        [5]])
32 In[11]: A*b # not matrix multiplication!
33 Out[11]:
34 array([[24, 15],
35        [12,  5]])
36 In[12]: A.dot(b.T) # matrix multiplication
37 Out[12]:
38 array([[39],
39        [17]])
40 In[13]: b = np.array([6,5]); b # 1D array
```

```

41 Out[13]: array([6, 5])
42 In[14]: b.T      # not matrix transpose!
43 Out[14]: array([6, 5])
44 In[15]: A.dot(b)
45 Out[15]: array([39, 17])
46 In[16]: A.dot(linalg.inv(A))
47 Out[16]: array([[1., 0.],
48                [0., 1.]])
49 In[17]: linalg.det(A)
50 Out[17]: -2.0
51 In[18]: linalg.norm(A), linalg.norm(b)
52 Out[18]: (5.477225575051661, 7.810249675906654)
53 In[19]: x = np.linalg.solve(A, b); x
54 Out[19]: array([ 4.5, -4. ])
55 In[20]: A.dot(x) - b
56 Out[20]: array([0., 0.])
57 In[21]: la, v = linalg.eig(A)
58 In[22]: la
59 Out[22]: array([ 5.37228132+0.j, -0.37228132+0.j])
60 In[23]: v
61 Out[23]:
62 array([[ 0.90937671, -0.56576746],
63        [ 0.41597356,  0.82456484]])
64 In[24]: A.dot(v[:, 0]) - la[0] * v[:, 0]
65 Out[24]: array([0.+0.j, 0.+0.j])
66 In[25]: np.sum(abs(v**2), axis=0)
67 Out[25]: array([1., 1.])
68 In[26]: A = np.array([[2,3,5],[7,9,11]])
69 In[27]: U,s,V = linalg.svd(A); s
70 Out[27]: array([16.96707058,  1.05759909])
71 In[28]: m, n = A.shape; S = linalg.diagsvd(s, m, n); S
72 Out[28]:
73 array([[16.96707058,  0.,          0.,          ],
74        [ 0.,          1.05759909,  0.,          ]])
75 In[29]: U.dot(S.dot(V))
76 Out[29]:
77 array([[ 2.,  3.,  5.],
78        [ 7.,  9., 11.]])
79 In[30]: tol = 1E-10; (abs(s) > tol).sum() # rank

```

```
80 Out[30]: 2
81 In[31]: C = np.array([[2,3,5,7],[9,11,13,17],[19,23,29,31]])
82 In[32]: p, l, u = linalg.lu(C); p, l, u
83 Out[32]:
84 (array([[0., 1., 0.],
85         [0., 0., 1.],
86         [1., 0., 0.]]),
87  array([[1., 0., 0.],
88         [0.10526316, 1., 0.],
89         [0.47368421, 0.18181818, 1.]]),
90  array([[19., 23., 29., 31.],
91         [ 0., 0.57894737, 1.94736842, 3.73684211],
92         [ 0., 0., -1.09090909, 1.63636364]]))
93 In[33]: np.allclose(C - p @ l @ u, np.zeros((3, 4)))
94 Out[33]: True
```

6.6 实验6: NumPy数组和矩阵计算

实验目的

本实验的目的是掌握使用NumPy数组进行矩阵计算。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

- 1. 求解线性方程组和矩阵的基本运算
生成一个由实数组成的秩为4的4行4列的矩阵**A**和一个由实数组成的包含4个元素的列向量**b**。求解线性方程组**Ax = b**。计算矩阵**A**的转置、行列式、秩、逆矩阵、特征值和特征向量。
- 2. 计算最小二乘解和矩阵分解
生成一个由实数组成的秩为4的6行4列的实矩阵**B**和一个由实数组成的包含6个元素的列向量矩阵**b**。计算线性方程组**Bx = b**的最小二乘解(**B^TB**)⁻¹**B^Tb**。计算矩阵**B**的奇异值分解并验证方程6.1。计算矩阵**B**的LU分解。

第七章 错误处理和文件读写

7.1 错误处理

7.1.1 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本意是对从1至10的所有整数求和，但由于对range函数的错误理解，实际是从1至9的所有整数求和。

Listing 7.1: for语句输出从1至10的所有整数的和

```
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，改正这两类错误比较容易。相比之下，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构思一个设计方案，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，在正确性和运行效率上优于自己编写的程序。

7.1.2 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。

设断点调试程序所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[HL2020]为例说明在Spyder中设断点调试程序的方法。

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($0 \leq j \leq n - 2$)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($j + 1 \leq i \leq m - 1$)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


```
1 [[ 2.          3.          5.          7.          ]
2  [ 0.          -3.5         -10.5        -19.5         ]
3  [ 0.          0.          -10.         -12.85714286]]
4 [[ 2.   3.   5.   7.]
5  [ 0.   0. -13. -23.]
6  [ nan nan -inf -inf]]
7 C:\Users\user\.spyder-py3\temp.py:7:
8   RuntimeWarning: divide by zero encountered in double_scalars
9   A[i, :] -= (A[i, j] / A[j, j]) * A[j, :]
10 C:\Users\user\.spyder-py3\temp.py:7:
11   RuntimeWarning: invalid value encountered in multiply
12   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”或按下快捷键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。 Debug菜单提供了多个菜单项用于调试运行，表7.1说明了一些常用菜单项的用途。

菜单项	用途
Set/Clear breakpoint	点击一次在当前行设置断点，再点击一次则清除断点。
Debug	开始调试运行。调试运行和普通运行的区别在于遇到断点会停止。
Step	单步运行，如果当前语句是函数调用语句则执行完函数调用，不会进入函数内部。
Step Into	单步运行，如果当前语句是函数调用语句则进入函数内部并停止在第一条语句。
Continue	继续运行直至遇到下一个断点，然后停止。
Stop	结束调试运行。
Clear breakpoints in all files	清除所有断点。

表 7.1: Debug菜单的常用菜单项

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”。

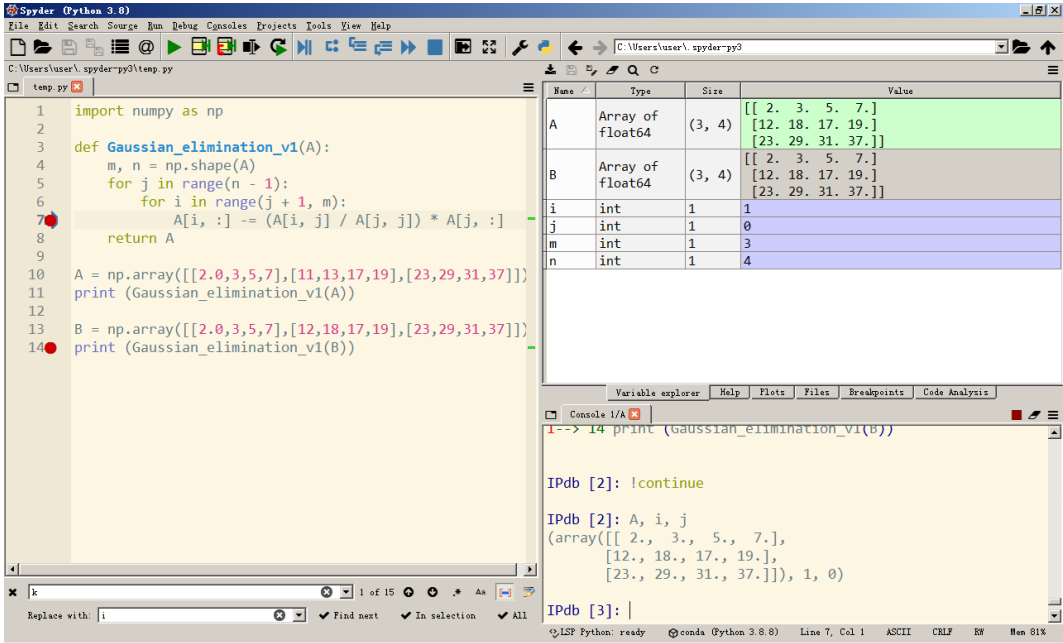


图 7.1: 设置断点进行调试

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
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 ...
```

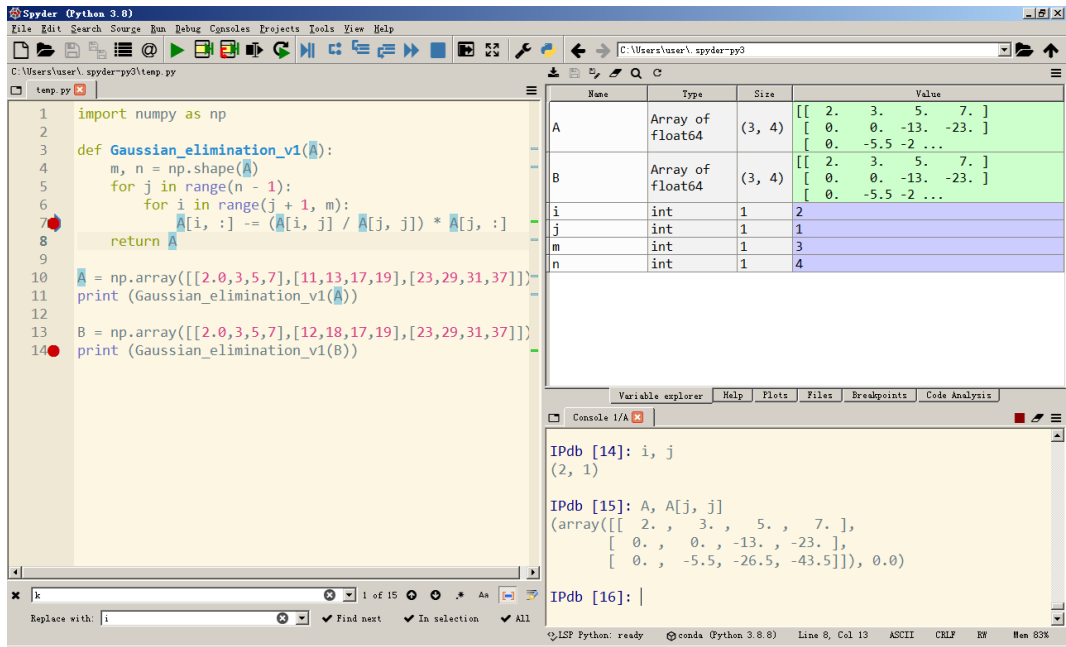


图 7.2: 设置断点进行调试

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行交换第p+i行和第i行。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:
9             # swap rows
10             A[[i, p + i]] = A[[p + i, i]]
11         if abs(A[i, j]) > tol:
12             # j is a pivot column
13             for r in range(i + 1, m):
14                 A[r, j:] -= (A[r, j] / A[i, j]) * A[i, j:]
15             i += 1
16             if i >= m: break
17     return A, i

```

7.1.3 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并难于理解。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，运行时系统会抛出异常，导致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，第13至14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，第15至16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，并且前两个都是整数，则程序不会发生异常，第12行运行完成以后跳转到第18行。

Listing 7.6: 异常处理

```

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

```

Listing 7.7: 程序7.6的运行结果

```

1 In[1]: run d:\python\src\gcd_ex.py 4
2 Out[1]: Two arguments must be supplied on the command line

```

```
3 In[2]: run d:\python\src\gcd_ex.py 4 60
4 Out[2]: Each argument should be an integer.
5 In[3]: run d:\python\src\gcd_ex.py 4 60
6 Out[3]: The greatest common divisor of 4 and 60 is 4
```

Python标准库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。这些异常类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，自定义的异常类以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22行判断用户输入的两个整数中是否存在负数，若是则在第23行抛出InputRangeError异常。该异常对象被第28行的except语句块捕获，然后在第29行输出出错信息。

第30至32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。第33至34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，通常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     while a != b:
3         if a > b:
4             a -= b;
5         else:
6             b -= a;
7     return a
8
9 class InputRangeError(Exception):
10     """Raised when an input is not in suitable range
11
```

```

12     Attributes:
13         message -- explanation of suitable range
14     """
15     def __init__(self, message):
16         self.message = message
17
18 import sys
19 try:
20     x = int(sys.argv[1])
21     y = int(sys.argv[2])
22     if x <= 0 or y <= 0:
23         raise InputRangeError('Each integer should be positive')
24 except IndexError:
25     print('Two arguments must be supplied on the command line')
26 except ValueError:
27     print('Each argument should be an integer.')
28 except InputRangeError as ex:
29     print(ex.message)
30 else:
31     print('The greatest common divisor of %d and %d is %d' %\
32           (x, y, gcd(x, y)))
33 finally:
34     print("executing finally clause")

```

Listing 7.9: 程序7.8的运行结果

```

1 In[1]: run d:\python\src\gcd_ex.py -48 126
2 Out[1]: Each integer should be positive
3         executing finally clause
4 In[2]: run d:\python\src\gcd_ex.py 48 126
5 Out[2]: The greatest common divisor of 48 and 126 is 6
6         executing finally clause

```

7.2 文件读写

7.2.1 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:”语句块打开的文件f会自动关闭。

7.2.2 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(7.10)记录了合肥市每月的平均降水量。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第2至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

```
1 Average rainfall (in mm) in HEFEI: 459 months between 1951 and 1990
2 Jan 32.2
3 Feb 53.2
4 Mar 71.8
5 Apr 92.5
6 May 101.5
7 Jun 117.3
8 Jul 175.7
9 Aug 117.7
10 Sep 85.6
11 Oct 60.7
12 Nov 51.2
13 Dec 27.6
14 Year 988.7
```

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行读取文件的第一行。第4行定义了一个空的字典rainfall。第5行至第8行的循环每次从文件中读取一行。第6行判断当前行是否包含子串'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行调用函数extract_data获取从文件中提取的字典。第14行至第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。第22行至第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.max; 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

```

1 The maximum rainfall of 175.7 occurs in Jul
2 The minimum rainfall of 27.6 occurs in Dec
3 The average rainfall is 82.3

```

7.2.3 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和LibreOffice Calc等)可以读入CSV文件并显示为电子表格。Python标准库的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的第2行至第3行从文件 scores.csv(7.14)中读取四位学生在三个科目上的考试成绩数据并存入嵌套列表table中。其中的每条数据都是字符串。第7行至第9行的双重循环将每条数据转换为float类型。第12行至第16行的双重循环计算每位学生的总分并将其追加到列表中对应该学生的行。第18行创建一个新的列表row。第19行至第23行的双重循环计算每个科目的平均成绩并将其追加到row中。第24行将row追加到table中。第26行使用pprint模块的pprint函数输出table。第28行至第31行将table写入文件scores2.csv(7.15)中。

Listing 7.13: 读写CSV文件

```
1 import csv, pprint
2 with open('D:/Python/src/scores.csv', 'r') as infile:
3     table = [row for row in csv.reader(infile)]
4
5 rows = len(table); cols = len(table[0])
6
7 for r in range(1, rows):
8     for c in range(1, cols):
9         table[r][c] = float(table[r][c])
10
11 table[0].append('Total')
12 for r in range(1, rows):
13     total = 0
14     for c in range(1, cols):
15         total += table[r][c]
16     table[r].append(total)
17
18 row = ['Average']
19 for c in range(1, cols):
20     avg = 0
21     for r in range(1, rows):
22         avg += table[r][c]
23     row.append(avg / (rows - 1))
24 table.append(row)
25
26 pprint.pprint(table)
27
28 with open('D:/Python/src/scores2.csv', 'w', newline='') as outfile:
29     writer = csv.writer(outfile)
```

```
30     for row in table:
31         writer.writerow(row)
```

Listing 7.14: `scores.csv`

```
1 Name,Math,Physics,English
2 Tom,95,91,81
3 Jerry,89,82,86
4 Mary,83,80,96
5 Betty,88,96,93
```

Listing 7.15: `scores2.csv`

```
1 Name,Math,Physics,English,Total
2 Tom,95.0,91.0,81.0,267.0
3 Jerry,89.0,82.0,86.0,257.0
4 Mary,83.0,80.0,96.0,259.0
5 Betty,88.0,96.0,93.0,277.0
6 Average,88.75,87.25,89.0
```

7.2.4 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标准库的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的第3行至第7行定义了一组通讯录数据contacts，它是一个字典的列表。第9行至第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)。第12行至第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

```
1 [{"Name": "Tom", "Phone": 12345, "Address": "100_Wall_St."},
2  {"Name": "Jerry", "Phone": 54321, "Address": "200_Main_St."},
3  {"Name": "Mary", "Phone": 23415, "Address": "300_Fifth_Ave."}]
```

Listing 7.18: 程序7.16的输出结果

```
1 [{'Address': '100_Wall_St.', 'Name': 'Tom', 'Phone': 12345},
2  {'Address': '200_Main_St.', 'Name': 'Jerry', 'Phone': 54321},
3  {'Address': '300_Fifth_Ave.', 'Name': 'Mary', 'Phone': 23415}]
```

7.2.5 读写pickle文件

pickle是一种Python定义的数据格式。Python标准库的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的第3行至第7行定义了一组通讯录数据contacts，它是一个字典的列表。第9行至第10行打开一个文件contacts.pickle，并将contacts的内容以pickle格式写入其中。第12行至第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

```
1 [{ 'Address': '100_Wall_St.', 'Name': 'Tom', 'Phone': 12345},
2  { 'Address': '200_Main_St.', 'Name': 'Jerry', 'Phone': 54321},
3  { 'Address': '300_Fifth_Ave.', 'Name': 'Mary', 'Phone': 23415}]
```

7.2.6 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。

Listing 7.21: 读写NumPy数组的文件

```
1 In[1]: import numpy as np; a = np.arange(1, 16, 2)**2; a
2 Out[1]: array([ 1, 9, 25, 49, 81, 121, 169, 225], dtype=int32)
3 In[2]: b = a.reshape(2, 4); b
4 Out[2]:
5 array([[ 1, 9, 25, 49],
6        [ 81, 121, 169, 225]], dtype=int32)
7 In[3]: np.savetxt('D:/Python/dat/b.txt', b)
8 In[4]: c = np.loadtxt('D:/Python/dat/b.txt'); c
9 Out[4]:
10 array([[ 1., 9., 25., 49.],
11        [ 81., 121., 169., 225.]])
12 In[5]: np.save('D:/Python/dat/b.npy', b)
13 In[6]: c = np.load('D:/Python/dat/b.npy'); c
14 Out[6]:
15 array([[ 1, 9, 25, 49],
16        [ 81, 121, 169, 225]])
17 In[7]: np.savez('D:/Python/dat/ab.npz', a, b)
18 In[8]: cd = np.load('D:/Python/dat/ab.npz')
19 In[9]: c = cd['arr_0']; c
20 Out[9]: array([ 1, 9, 25, 49, 81, 121, 169, 225])
21 In[10]: d = cd['arr_1']; d
```

```
22 Out[10]:
23 array([[ 1,    9,   25,   49],
24        [ 81,  121,  169,  225]])
```

7.3 实验7: 错误处理和文件读写

实验目的

本实验的目的是掌握以下内容：程序调试，异常处理和文件读写。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录第2题和第3题的源程序和运行结果。第1题无须提交。

实验内容

1. 程序调试

设断点单步运行本章的程序7.5、7.11和7.14，观察变量的值。

2. 异常处理

编写程序读入用户在命令行输入的三个float类型的数值，判断它们是否能构成一个三角形的三条边。若可以，则使用以下公式计算并输出三角形的面积。公式中的 a, b, c 为三角形的三条边的长度。

$$area = \sqrt{s(s-a)(s-b)(s-c)}, s = \frac{a+b+c}{2}$$

程序应处理以下类型的错误并输出出错信息：

- 用户在命令行输入的参数小于三个；
- 用户在命令行输入的三个参数不全是float类型；
- 用户在命令行输入的三个float类型的数值不能构成一个三角形的三条边，这里需要自定义异常类InvalidTriangleError。

3. 文件读写

编写程序读入一个存储了程序7.21的文本文件，从中提取用户输入的代码然后输出到一个文本文件中。输出的文件的内容应为：

Listing 7.22: 程序7.21中用户输入的代码

```
1 import numpy as np; a = np.arange(1, 16, 2)**2; a
2 b = a.reshape(2, 4); b
```

```
3 np.savetxt('D:/Python/dat/b.txt', b)
4 c = np.loadtxt('D:/Python/dat/b.txt'); c
5 np.save('D:/Python/dat/b.npy', b)
6 c = np.load('D:/Python/dat/b.npy'); c
7 np.savez('D:/Python/dat/ab.npz', a, b)
8 .....
```

第八章 时间性能的分析 and 测量

8.1 算法和其时间性能的衡量

算法设计与分析是计算机科学的核心领域。算法是求解问题的一系列计算步骤，用来将输入数据转换成输出结果。算法的每个步骤应当是精确定义的，不允许出现歧义。算法应在运行有穷步后结束。如果一个算法对所有输入数据都能输出正确的结果并停止，则称它是正确的。3.2节列出的求解某一给定自然数区间内的所有质数的设计方案描述了一个算法。解决同一个问题的算法可以有多种，运行这些算法所消耗的时间(称为时间性能)也有差别。需要衡量和比较不同算法的时间性能，选择最优算法。

一种衡量方法是测量该算法编写的程序在计算机上运行所消耗的时间。这种方法有一定局限性，因为一个高级语言程序的运行时间取决于多种因素，例如算法的策略、问题的规模、输入数据的特点、编译器的代码生成和优化、运行时系统的效率和CPU执行指令的速度等。只有当除了算法的策略以外的其他因素都相同时，基于测量的时间比较不同算法的性能才是合理的。

另一种衡量方法是理论分析，把算法的运行时间定义为问题规模的函数，称为时间复杂度。这种方法不需要考虑除了算法的策略以外的其他因素，但是需要复杂的数学推导和计算，较难得到精确的公式。有些算法的运行时间受问题输入数据的影响很大，例如排序算法。此时需要对最坏、平均和最好三种情形进行分析。在本章中只对最坏情形进行分析，即估计时间复杂度的上界。时间复杂度通常使用大O记号表示，其定义如下：设函数 $f(n)$ 和 $g(n)$ 是定义在非负整数集合上的正函数，如果存在两个正常数 c 和 n_0 ，使得当 $n \geq n_0$ 时 $f(n) \leq cg(n)$ 成立，则记做 $f(n) = O(g(n))$ 。使用大O记号的好处是当时间复杂度的表达式由多个项组成时，只需考虑数量级最大的项且可以忽略它的常数系数。例如当问题规模为 n 时，时间复杂度 $2n^3 + 4n^2 + 8n + 17 = O(n^3)$ 。

8.2 算法的时间复杂度

算法由一些不同类别的基本运算组成，算法的时间复杂度等于每种基本运算的运行时间和其对应运行次数的乘积的总和，其中运行次数是问题规模的函数。为了简化分析，一般只选取运行次数最多的基本运算进行分析，因为当问题规模较大时它是决定时间复杂度的主要因素。

大多数算法的时间复杂度属于以下七类，按照数量级从低到高的次序依次为： $O(1)$ ， $O(\log n)$ ，

$O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$ 。一般认为时间复杂度属于多项式函数的算法是有效(efficient)的, 即现实可行的[KT2006]。时间复杂度属于 $O(2^n)$ 的算法的运行时间随 n 的增长而急剧增长, 只适用于问题规模较小的情形。解决一个问题的算法通常不止一种, 在保证正确性的前提下应尽量选择时间复杂度最低的。以下列举一些属于这七类时间复杂度的常用算法。

8.2.1 线性查找

程序8.1实现了线性查找算法。第2行至第3行的循环在列表s中查找指定数据k是否出现, 若出现则返回其索引值, 否则在第4行返回-1表示未找到。运行次数最多的基本运算是第3行, 它在最坏情形下的运行次数为列表s的长度 n 。因此线性查找算法的时间复杂度为 $O(n)$ 。

Listing 8.1: 线性查找

```
1 def linear_search(s, k):
2     for i in range(len(s)):
3         if s[i] == k: return i
4     return -1
```

8.2.2 插入排序

程序8.2实现了插入排序算法。插入排序的基本思想是将待排序列表中的每条数据依次插入到合适的位置。第3行的外循环的循环变量i是列表中待插入数据的索引值。第6行至第8行的内循环将待插入数据与索引值为i-1,i-2,...,0的数据依次进行比较, 为其寻找一个合适的插入位置, 然后在第9行完成插入。运行次数最多的基本运算是内循环的比较和赋值。对于长度为 n 的列表s, 在最坏情况下的运行次数为 $1 + 2 + \dots + (n - 1) = n(n - 1)/2$ 。因此插入排序算法的时间复杂度为 $O(n^2)$ 。

Listing 8.2: 插入排序

```
1 def insertion_sort(s):
2     n = len(s)
3     for i in range(1, n):
4         value = s[i]
5         pos = i
6         while pos > 0 and value < s[pos - 1] :
7             s[pos] = s[pos - 1]
8             pos -= 1
9         s[pos] = value
10        print (s)
11
12 s = [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]; print (s)
13 insertion_sort(s)
```


Listing 8.3: 插入排序的运行过程

```

1 [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
2 [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
3 [6, 21, 73, 67, 99, 60, 77, 5, 51, 32]
4 [6, 21, 67, 73, 99, 60, 77, 5, 51, 32]
5 [6, 21, 67, 73, 99, 60, 77, 5, 51, 32]
6 [6, 21, 60, 67, 73, 99, 77, 5, 51, 32]
7 [6, 21, 60, 67, 73, 77, 99, 5, 51, 32]
8 [5, 6, 21, 60, 67, 73, 77, 99, 51, 32]
9 [5, 6, 21, 51, 60, 67, 73, 77, 99, 32]
10 [5, 6, 21, 32, 51, 60, 67, 73, 77, 99]

```

8.2.3 二分查找

程序8.4实现了二分查找算法。假定列表已经按照从小到大的次序排好序，查找范围的索引值的下界和上界分别为low和high，则可计算中间位置的索引值min。二分查找算法采用迭代方法，将指定数据k和列表s的中间位置的数据进行比较。如果比较的结果是相等，则已找到并返回。如果比较的结果是小于，则只需在左半边的子列表中继续查找。否则，只需在右半边的子列表中继续查找。迭代的终止条件是下界大于上界，如果此条件满足则表示未找到。每进行一次迭代，查找范围缩小一半。对于长度为n的列表s，迭代次数不超过 $\lceil \log n \rceil$ ，所以二分查找算法的时间复杂度为 $O(\log n)$ 。

Listing 8.4: 二分查找

```

1 def binary_search(s, k):
2     low = 0; high = len(s) - 1
3     while low <= high:
4         mid = (high + low) // 2
5         print ('(%2d, %2d) low=%d, mid=%d, high=%d'
6               % (k, s[mid], low, mid, high))
7         if k == s[mid]:
8             return mid
9         elif k < s[mid]:
10            high = mid - 1
11        else:
12            low = mid + 1
13    return -1
14
15 s = [5, 6, 21, 32, 51, 60, 67, 73, 77, 99]
16 print (binary_search(s, 77)); print (binary_search(s, 31))

```

Listing 8.5: 二分查找的运行过程

```
1 (77, 51) low = 0, mid = 4, high = 9
2 (77, 73) low = 5, mid = 7, high = 9
3 (77, 77) low = 8, mid = 8, high = 9
4 8
5 (31, 51) low = 0, mid = 4, high = 9
6 (31, 6) low = 0, mid = 1, high = 3
7 (31, 21) low = 2, mid = 2, high = 3
8 (31, 32) low = 3, mid = 3, high = 3
9 -1
```

8.2.4 归并排序

程序8.6实现了归并排序算法。算法的主函数是merge_sort，它把待排序的列表等分成左右两个子列表，分别对它们递归调用merge_sort进行排序，然后调用辅助函数merge_ordered_lists把两个已经排好序的子列表合并在一起成为一个排好序的列表。合并的方法是比较第一个列表s1中索引值为i的数据和第二个列表s2中索引值为j的数据，将其中较小的数据追加到列表t中，再修改i或j。如果s1或s2中的所有数据都已经追加到列表t中，则只需把另一个列表中的剩余数据追加到列表t中。对于长度为n的列表s，时间复杂度 $T(n)$ 满足方程 $T(n) = 2T(n/2) + n$ ，由此可以推导出归并排序算法的时间复杂度为 $O(n \log n)$ 。

Listing 8.6: 归并排序

```
1 def merge_ordered_lists(s1, s2):
2     t = []
3     i = j = 0
4     while i < len(s1) and j < len(s2):
5         if s1[i] < s2[j]:
6             t.append(s1[i]); i += 1
7         else:
8             t.append(s2[j]); j += 1
9     t += s1[i:]
10    t += s2[j:]
11    print ('%s_+_%s_=>_%s' % (s1, s2, t));
12    return t
13
14 def merge_sort(s):
15     if len(s) <= 1:
16         return s
17     mid = len(s) // 2
```

```
18     print ('%s->%s+%s' % (s, s[:mid], s[mid:]));
19     left = merge_sort(s[:mid])
20     right = merge_sort(s[mid:])
21     return merge_ordered_lists(left, right)
22
23 s = [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
24 print (s); print (merge_sort(s))
```

Listing 8.7: 归并排序的运行过程

```
1 [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
2 [21, 73, 6, 67, 99, 60, 77, 5, 51, 32] ->
3     [21, 73, 6, 67, 99] + [60, 77, 5, 51, 32]
4 [21, 73, 6, 67, 99] -> [21, 73] + [6, 67, 99]
5 [21, 73] -> [21] + [73]
6 [21] + [73] => [21, 73]
7 [6, 67, 99] -> [6] + [67, 99]
8 [67, 99] -> [67] + [99]
9 [67] + [99] => [67, 99]
10 [6] + [67, 99] => [6, 67, 99]
11 [21, 73] + [6, 67, 99] => [6, 21, 67, 73, 99]
12 [60, 77, 5, 51, 32] -> [60, 77] + [5, 51, 32]
13 [60, 77] -> [60] + [77]
14 [60] + [77] => [60, 77]
15 [5, 51, 32] -> [5] + [51, 32]
16 [51, 32] -> [51] + [32]
17 [51] + [32] => [32, 51]
18 [5] + [32, 51] => [5, 32, 51]
19 [60, 77] + [5, 32, 51] => [5, 32, 51, 60, 77]
20 [6, 21, 67, 73, 99] + [5, 32, 51, 60, 77] =>
21     [5, 6, 21, 32, 51, 60, 67, 73, 77, 99]
22 [5, 6, 21, 32, 51, 60, 67, 73, 77, 99]
```

8.2.5 穷举法求解3-sum问题

3-sum问题的描述如下：给定一个整数x和一个由整数构成的集合s，从s中找一个由三个元素构成的子集，该子集中的三个元素之和必须等于x。穷举法是一种简单直接的算法，即列举所有由三个元素构成的子集，逐个检查其是否满足条件。程序8.8实现了穷举法求解3-sum问题。运行次数最多的基本运算是三重循环的最内层循环的比较,次数为 $\sum_{i=0}^{n-3} \sum_{j=i+1}^{n-2} (n-j-1) = n(n-1)(n-2)/6$ 。所以该算法的时间复杂度为 $O(n^3)$ 。

Listing 8.8: 穷举法求解3-sum问题

```

1 def exhaustive_search_3Sum(s, x):
2     n = len(s)
3     for i in range(n - 2):
4         for j in range(i + 1, n - 1):
5             for k in range(j + 1, n):
6                 if s[i] + s[j] + s[k] == x:
7                     return s[i], s[j], s[k]
8     return ()
9
10 s = [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]
11 print (exhaustive_search_3Sum(s, 152)) # (21, 99, 32)

```

对以上穷举法可以进行改进得到时间复杂度为 $O(n^2 \log n)$ 的算法。假定存在一个子集 $\{a, b, c\}$ 满足 $x = a + b + c$, 则 $x - a = b + c$ 。原问题可以转换为另一个问题: 对于 s 中的任意两个元素 b 和 c , 在 $s - \{b, c\}$ 中查找一个等于 $x - b - c$ 的元素。为了提高查找的效率, 可以先将所有元素按照从小到大的顺序排序, 然后使用二分查找。

8.2.6 穷举法求解subset-sum问题

subset-sum问题的描述如下: 给定一个整数 x 和一个由整数构成的集合 s , 从 s 中找一个子集, 该子集中的所有元素之和必须等于 x 。使用穷举法列举 s 的所有子集, 逐个检查其是否满足条件。设 s 包含 n 个元素, 则 s 的每个子集 t 和 n 位二进制数存在一一映射。 n 位二进制数的第 k 位为1表示第 k 个元素在子集 t 中, 为0则表示不在。

程序8.9实现了穷举法求解subset-sum问题。第6行的表达式 $i \gg j$ 使用整数的比特移位运算符将 i 的二进制形式向右移动 j 次, 然后判断其是否是奇数。如果是, 则从右边数的第 j 位为1, 将第 j 个元素追加到子集中。对应于 s 的所有非空子集的 n 位二进制数有 $2^n - 1$ 个, 所以该算法的时间复杂度为 $O(2^n)$ 。

Listing 8.9: 穷举法求解subset-sum问题

```

1 def exhaustive_search_subset_Sum(s, x):
2     N = len(s)
3     for i in range(1, 2 ** N):
4         subset = []
5         for j in range(N):
6             if (i >> j) % 2 == 1: subset.append(s[j])
7             if sum(subset) == x: return subset
8     return []
9
10 s = [21, 73, 6, 67, 99, 60, 77, 5, 51, 32]

```

```
11 print (exhaustive_search_subset_Sum(s, 135)) # [73, 6, 5, 51]
```

8.3 时间性能的测量

timeit模块的timeit函数测量一个程序重复运行多次所需时间。程序8.10定义的两个字符串s1和s2表示两个程序。程序s1向一个空列表中添加10万个元素。程序s2生成一个指定长度的由随机数构成的列表，然后对其排序。第18行测量了程序s1运行10次的平均运行时间。第18行测量了程序s1运行10次的平均运行时间。第20行至第22行的循环使用不同的长度值运行程序s2，测量其运行10次的平均运行时间。

Listing 8.10: timeit单行语句

```
1 s1 = """\
2 a = []
3 for i in range(100000):
4     a.append(i)
5 """
6
7 s2 = """\
8 import random
9 def sort_random_list(n):
10     alist = [random.random() for i in range(n)]
11     alist.sort()
12
13 sort_random_list(%d)
14 """
15
16 import timeit
17 N = 10
18 print (timeit.timeit(stmt=s1, number=N) / N) # 0.01204275720001533
19
20 for n in [10000, 20000, 40000, 80000]:
21     t = timeit.timeit(stmt=s2 % n, number=N) / N
22     print ('%d: %.4f' % (n, t), end = ' ')
23 # 10000:0.0035 20000:0.0071 40000:0.0181 80000:0.0329
```

性能分析工具line_profiler可测量程序中加了@profile标记的函数中每一行语句的时间性能。

在Spyder的IPython窗口中运行命令“pip install line_profiler”安装line_profiler。以计算Julia集[SL2018]并绘图显示的程序8.11为例[MG2017]说明以行为单位的时间性能测量。

对于复平面上的每个点 z ，Julia集由迭代过程 $z_0 = z, z_{n+1} = z_n^2 + c$ 定义。迭代过程的终止条件是 $|z| \geq 2$ 或者 $n \geq N$ ，其中 N 是预先设定的最大迭代次数。通过迭代过程，可计算每个点 z 在终止时的迭代次数 n 。将以原点为中心的正方形区域内定义的等距网格上的每个点的迭代次数映射到某一灰度或颜色，即可生成一个灰度或彩色图片。图8.1展示了选取六个不同的 c 值所得到的有趣图片。程序8.11的`calculate_z_serial_purepython`函数计算每个点的迭代次数，`calc_Julia`函数对每个点调用`calculate_z_serial_purepython`函数。`time`模块的`time`函数返回从一个时间起点(1970年1月1日世界标准时间00:00:00)到当前时刻所经历的秒数。在运行一个函数之前和之后分别使用`time`函数记录两个数值，它们的差值即为该函数的运行时间。函数`show_color`将每个点的迭代次数转换为一个六进制的三位整数，它的每一位乘以50分别为组成一种颜色的三原色之一的数值。为了生成图片，需要将程序8.11的第47行的`False`改为`True`并且删除第6行和第16行的`@profile`标记。

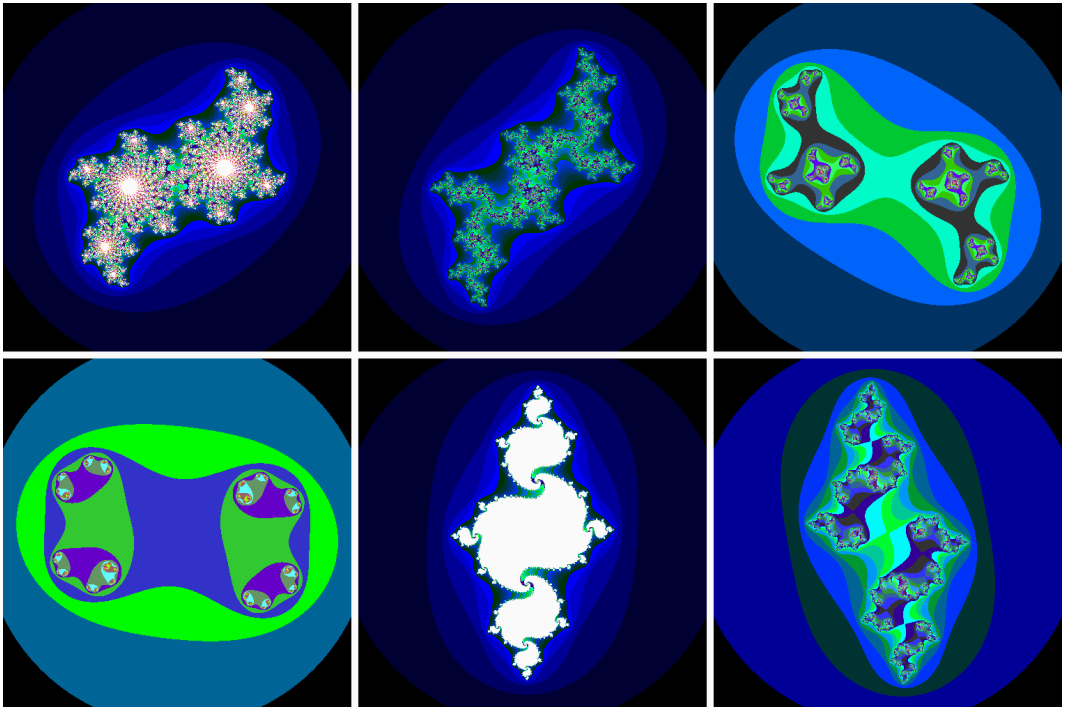


图 8.1: Julia集生成的图片

Listing 8.11: Julia集

```

1 import time; import array; import numpy as np
2
3 x1, x2, y1, y2 = -1.8, 1.8, -1.8, 1.8      # range of complex space
4 c_real, c_imag = -0.62772, -.42193
5
6 @profile
7 def calculate_z_serial_purepython(maxiter, zs, c):
8     output = [0] * len(zs)

```

```

9     for i in range(len(zs)):
10         z = zs[i]; n = 0
11         while abs(z) < 2 and n < maxiter:
12             z = z * z + c; n += 1
13         output[i] = n
14     return output
15
16 @profile
17 def calc_Julia(draw, desired_width, max_iterations):
18     xs = np.linspace(x1, x2, desired_width)
19     ys = np.linspace(y1, y2, desired_width)
20     zs = []; c = complex(c_real, c_imag)
21     for x in xs:
22         for y in ys:
23             zs.append(complex(x, y))
24     start_time = time.time()
25     output = calculate_z_serial_purepython(max_iterations, zs, c)
26     end_time = time.time()
27     print(end_time - start_time, "seconds")
28     if draw: show_color(output, len(xs), len(ys), max_iterations)
29
30 from PIL import Image
31 def show_color(output_raw, width, height, max_iterations):
32     # convert our output to PIL-compatible input
33     assert width * height == len(output_raw)
34     # rescale output_raw to be in the inclusive range [0..215]
35     max_value = float(max(output_raw))
36     output_raw_limited = [int(float(o) / max_value * 215) \
37                           for o in output_raw]
38     rgb = array.array('B')
39     for o in output_raw_limited:
40         r = o // 36; o = o % 36; g = o // 6; b = o % 6
41         # assert r<6 and b<6 and g<6
42         rgb.append(r*50); rgb.append(g*50); rgb.append(b*50);
43     im = Image.new("RGB", (width, height));
44     im.frombytes(rgb.tobytes(), "raw", "RGB")
45     im.show()
46
47 calc_Julia(draw=False, desired_width=500, max_iterations=200)

```

假定Anaconda的安装路径是”D:\Anaconda”，在控制台运行命令 “D:\Anaconda\Scripts\kernprof -l -v julia_nopil.py” 可显示程序中 calculate_z_serial_purepython和calc_Julia这两个函数的每行语句的运行时间(图8.2)。

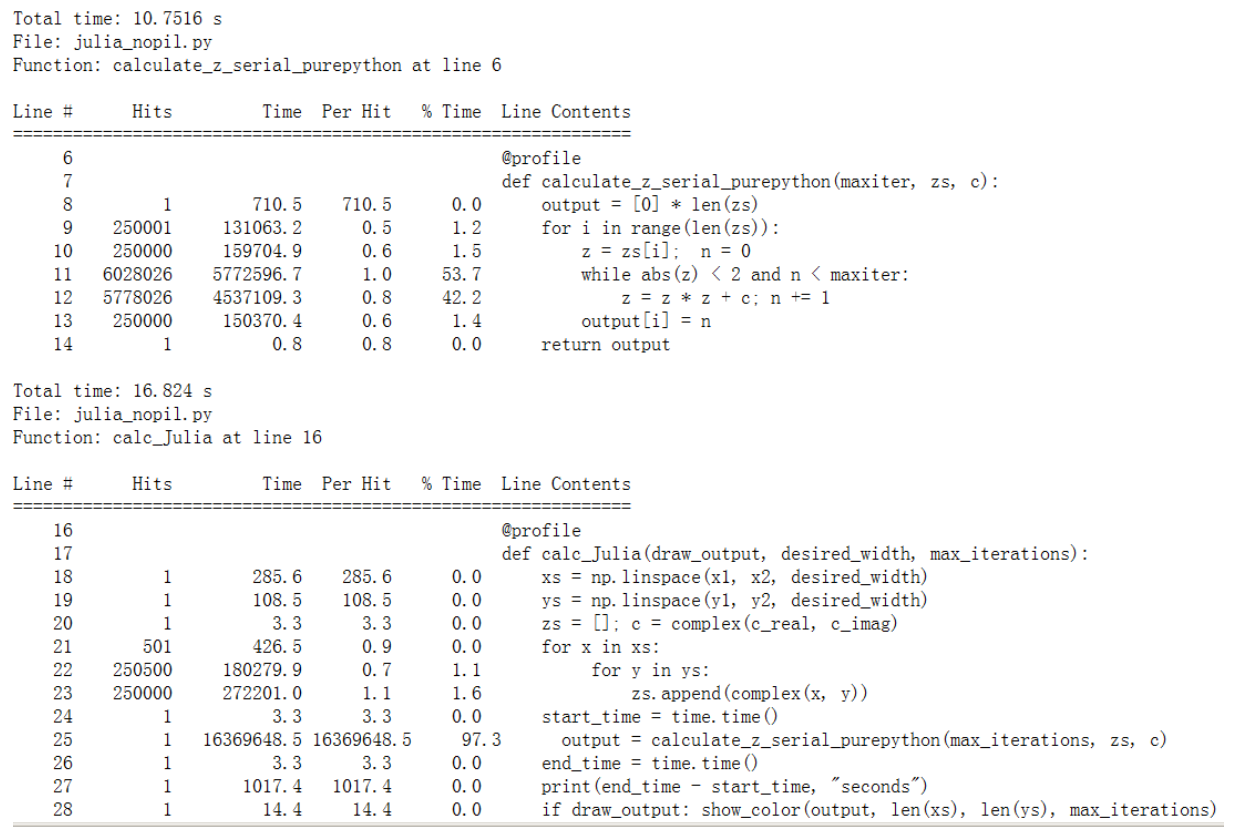


图 8.2: 以行为单位的时间性能测量

calc_Julia函数的第25行的函数调用消耗了整个函数大部分的运行时间，而 calculate_z_serial_purepython函数中的内循环消耗了整个函数大部分的运行时间，是整个程序的性能瓶颈。

8.4 Cython

和C语言相比，Python的动态特性虽然降低了编写程序的工作量，但也同时降低了程序的运行效率。使用Cython编译器可以将添加了类型声明的Python程序(称为Cython程序)编译成C语言程序，再使用C语言编译器将其编译成可从Python程序中调用的扩展模块，从而提高整个程序的运行效率。下面以计算圆周率 $\pi = 4(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots)$ 的程序为例说明使用Cython的主要步骤。

1. 从微软公司网站<https://visualstudio.microsoft.com/downloads/>下载和安装Visual Studio或Build Tools for Visual Studio。

- 假定Anaconda的安装路径是“D:\Anaconda”，将“D:\Anaconda”和“D:\Anaconda\Library\bin”这两个路径追加到操作系统的path环境变量中。
- 编写Python程序，按照Cython要求添加类型声明，保存为文件calc_pi.pyx(程序8.12)。

Listing 8.12: calc_pi.pyx

```

1 import cython; import numpy as np
2
3 @cython.cdivision(True)
4 def calc_pi(int n):
5     cdef double pi = 0
6     cdef int i
7     for i in range(1, n, 4):
8         pi += 4.0 / i
9     for i in range(3, n, 4):
10        pi -= 4.0 / i
11    return pi

```

- 编写一个文件setup.py(程序8.13)。

Listing 8.13: setup.py

```

1 from distutils.core import setup
2 from Cython.Build import cythonize
3 import numpy as np
4 setup(ext_modules=cythonize('calc_pi.pyx'),
5       include_dirs=[np.get_include()],
6       requires=['Cython', 'numpy'])

```

- 在控制台运行如下命令(8.14)将程序8.12编译成为扩展模块。

Listing 8.14: 将Cython程序编译成为扩展模块

```
1 python setup.py build_ext --inplace
```

- 在Spyder的IPython窗口中设置路径为生成的扩展模块calc_pi.cp38-win32.pyd所在文件夹。

Listing 8.15: 设置路径

```
1 cd D:\Python\src\calc_pi
```

- 编写程序8.16调用扩展模块。%timeit命令使用timeit模块对单行语句重复运行多次，计算出平均运行时间。

Listing 8.16: 调用扩展模块

```

1 from calc_pi import calc_pi
2 n = 1000000
3 %timeit calc_pi(n)
4 # 3.57 ms per loop (mean of 7 runs, 100 loops each)
5
6 import numpy as np
7 %timeit np.sum(4.0 / np.r_[1:n:4, -3:-n:-4])
8 # 6.75 ms per loop (mean of 7 runs, 100 loops each)

```

在本例中，扩展模块的运行效率是实现同样功能的Python程序的1.89倍。

为了提高运行效率，程序8.17用Cython实现calculate_z_serial_purepython函数。

Listing 8.17: 用Cython实现calculate_z_serial_purepython

```

1 #cython: boundscheck=False
2 def calculate_z(int maxiter, zs, c):
3     """Calculate output list using Julia update rule"""
4     cdef unsigned int i, n
5     cdef double complex z
6     output = [0] * len(zs)
7     for i in range(len(zs)):
8         n = 0
9         z = zs[i]
10        while n < maxiter and (z.real * z.real + z.imag * z.imag) < 4:
11            z = z * z + c
12            n += 1
13        output[i] = n
14    return output

```

编写一个文件setup.py(程序8.18)，然后在控制台运行命令8.14将程序8.17编译成为扩展模块。。

Listing 8.18: setup.py

```

1 from distutils.core import setup
2 from Cython.Build import cythonize
3 setup(ext_modules=cythonize("cythonfn.pyx"))

```

在程序8.11中添加语句“from cythonfn import calculate_z”，然后删除

calculate_z_serial_purepython函数，将对这个函数的调用改为对扩展模块中calculate_z函数的调用，保存为julia.cython.py。运行julia.cython.py耗时 0.04800271987915039秒。

8.5 实验8: 时间性能的分析 and 测量

实验目的

本实验的目的是掌握时间性能的分析 and 测量。

提交方式

在Blackboard提交一个文本文件(txt后缀)，文件中记录每道题的源程序和运行结果。

实验内容

1. 实现一个时间复杂度为 $O(n^2 \log n)$ 的算法，解决3-sum问题。
2. 生成长度为100,200,...,900,1000的由随机数构成的列表，分别测量插入排序(8.2)、归并排序(8.6)和快速排序(4.14)这三种排序算法的运行时间。对于每个长度测量10次计算平均值。在测量之前需要删除以上程序中的print语句。
3. 测量归并排序(8.6)的两个函数中的每条语句的时间性能。在测量之前需要删除以上程序中的print语句。

参考文献

- [HL2020] 汉斯.佩特.兰坦根 (挪)著, 张春元、刘万伟、毛晓光、陈立前、周会平、李墩译, 科学计算基础编程——Python版, 清华大学出版社, 2020.
- [JC2021] Jean-Pierre Corriou, Numerical Analysis and Optimization, Springer, 2012.
- [JD2011] Jay L. Devore, Kenneth N. Berk, Modern Mathematical Statistics with Applications, Springer, 2011.
- [JN2006] Jorge Nocedal, Stephen J. Wright, Numerical Optimization, Springer, 2006.
- [KS2021] Qingkai Kong, Timmy Siau, Alexandre M. Bayen, Python Programming And Numerical Methods: A Guide For Engineers And Scientists, Elsevier, 2021.
- [KT2006] Jon Kleinberg, Eva Tardos, Algorithm Design, Pearson Education, Inc., 2006
- [MG2017] Micha Gorelick, Ian Ozsvald(美)著, 胡世杰、徐旭彬译, Python高性能编程, 人民邮电出版社, 2017.
- [NL2019] Nicolas Lanchier, Stochastic Modeling, Springer, 2017.
- [NumPyDoc] <https://numpy.org/doc/stable>
- [PythonDoc] <https://docs.python.org/>
- [RJ2019] Robert Johansson, Numerical Python, Springer, 2019.
- [RS2017] Ramteen Sioshansi, Antonio J. Conejo, Optimization in Engineering: Models and Algorithms, Springer, 2017.
- [RV2014] Robert J. Vanderbei, Linear Programming, Springer, 2014.
- [SB2002] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, Springer, 2002.
- [SciPyDoc] <https://scipy.github.io/devdocs/index.html>
- [SL2018] Stephen Lynch, Dynamical Systems with Applications using Python, Springer, 2018.
- [SS2010] Seabold, Skipper, and Josef Perktold, statsmodels: Econometric and statistical modeling with python, Proceedings of the 9th Python in Science Conference, 2010.

- [TL2019] Tom Lyche, Numerical Linear Algebra and Matrix Factorizations, Springer, 2019.
- [WG2012] Walter Gautschi, Numerical Analysis, Springer, 2012.
- [YW2020] 喻文健, 数值分析与算法(第3版), 清华大学出版社, 2020.