20240220-Week1-数算课程概述

Updated 2153 GMT+8 Feb 19, 2024

2024 spring, Complied by Hongfei Yan

本周发布作业：

assignment1, assignmentP

https://github.com/GMyhf/2024spring-cs201

每次作业评分标准

每次作业评分标准				
标准	等级			得分
按时提交	1 得分 提交	0.5 得分 请假	0 得分 未提交	1 分
源码、耗时（可选）、解题思路（可选）	1 得分 4+题目	0.5 得分 2+题目	0 得分 无源码	1 分
AC代码截图	1 得分 4+题目	0.5 得分 2+题目	0 得分 无截图	1 分
清晰头像、pdf、md/doc	1 得分 三项全	0.5 得分 有二项	0 得分 少于二项	1 分
学习总结和收获	1 得分 有	0 得分 无		1 分
				总得分： 5 ，满分 5

一、课程安排

了解数据结构与算法是透彻理解计算机科学的前提，学习计算机科学与掌握其他高难度学科没什么不同。成功的唯一途径便是循序渐进地学习其中的核心思想。刚开始接触计算机科学的人，需要多多练习来加深理解，从而为学习更复杂的内容做好准备。此外，初学者需要建立起自信心。

学习计算机科学与掌握其他高难度学科没什么不同。成功的唯一途径便是循序渐进地学习其中的核心思想。刚开始接触计算机科学的人，需要多多练习来加深理解，从而为学习更复杂的内容做好准备。此外，初学者需要建立起自信心。

数据结构与算法通常是计算机科学专业的第二门课程，比第一门课程更深入。同学们已经准备好进一步探索这一领域并且进一步练习如何解决问题。

假定学生已经学过计算机科学的入门课程，但入门课程不一定是用Python讲解的。理解基本的结构体，比如分支、迭代以及函数定义。同时，学生也能够理解Python的基础数据结构，比如序列（列表和字符串）以及字典。

特点：

- 较早介绍基于大O记法的算法分析，并且通篇运用；
- 使用Python讲解，以促使初学者能够使用和掌握数据结构与算法。

学生首先学习线性数据结构，包括栈、队列、双端队列以及列表。我们用Python列表以及链表实现这些数据结构。然后学习与树有关的非线性数据结构，了解连接节点和引用结构（链表）等一系列技术。最后，将通过运用链式结构、链表以及Python字典的实现，学习图的相关知识。对于每一种结构，都尽力在使用Python提供的内建数据类型的同时展现众多的实现技巧。 这种讲法在向学生揭示各种主要实现方法的同时，也强调 Python 的易用性。

Python是一门非常适合于讲解算法的语言，语法干净简洁，用户环境直观，基本的数据类型十分强大和易用。

我们相信，对于初学者来说，投入时间学习与算法和数据结构相关的基本思想是非常有益的。

1 流水账事项

1.1

课程：2024spring 数据结构与算法B（12班）
时间：2024年2月~6月，每周二 15:00~18:00 北京
线下教室：理教410
线上教室：
https://teams.microsoft.com/l/meetup-join/19%3ameeting_NDFmZjdiOGUtODg3My00MjQ5LTg4NjMtMDNkMTA0MmMxMTkx%40thread.v2/0?context=%7b%22Tid%22%3a%22cd66712e-8aa9-44ad-9e42-02ea2d037e64%22%2c%22Oid%22%3a%227fae1325-5f9b-401d-bd3f-42dd9a7d27b8%22%7d

Meeting ID: 479 606 888 228
Passcode: nGujBW

全面介绍在经典计算机科学问题中出现的数据结构与算法。尽管在学习顺序上并无严格要求，但是许多话题之间都存在一定的依赖关系，所以建议顺序学习。下表是16周的课程内容安排：

1. 课程概述和导论	**2. 算法分析**
3. 基本数据结构	4. 递归
5. 二分搜索	6. 排序
7. 树及其算法（1/3）	8. 树及其算法（2/3）
9. 树及其算法（3/3）	10. 图及其算法（1/3）
11. 图及其算法（2/3）	12. 图及其算法（3/3）
13. 动态规划	14. 贪心
15. 课程总结（1/2）	16. 课程总结（2/2）

1.2

目前考虑：数算B机考试内容主要是在栈、树、图、排序，尤其是经典算法例如：shunting yard, dijkstra；笔试内容可以侧重概念、问答。平时练习可以加点其他的，如：贪心、动归、博弈、递归。

1.3 每周都有作业

Assignment1 和 AssignmentP已经发布。

我同大家一起学习，我的课件也刚开始。https://github.com/GMyhf/2024spring-cs201

1.4

信息管理系统里面，原有教学大纲我修正了，不知道大家能否看到。以我们自己的为主。

1.5

我们用到的编程平台主要是 http://cs101.openjudge.cn，https://codeforces.com 。寒假时候，汇总了102个数算相关的题目（包括了xzm老师的60+题目，增加了图的题目），http://cs101.openjudge.cn/dsapre/

1.6

说明：请同学修改课程微信群昵称为实名。请助教助在自己姓名前面加上 "TA-"，如：TA-彭亦男，TA-张以宁，TA-涂程颖，TA-黄涵优

1.7

2024/2/19 说明：根据目前收到平行班通知，期末笔试是统考。既然这样，其他的安排我们自己布置：1）3月、4月、5月，每个月初有一次月考（不计分，建议尽量在机房规定时间完成），如果时间冲突，可以自行完成，会留作当周作业。2）因为有月考，不安排期中考试。3）期末机考时间：是第16周上机时间 2024/6/5 15:08~17:00，地点：机房。

1.8

上机第16周，来机考就可以。

学生成绩评定方法
我们采用综合评定方法对学生成绩进行评定。学生成绩分为三部分：作业成绩来自每周1次的作业(其中有3次作业对应3次月考)成绩20%；期末上机考试40%；课程大作业、期末笔试成绩40%。
3月、4月、5月，每个月初有一次月考（不计分，建议尽量在机房规定时间完成），月考题目也会留作当周作业。因为有月考，不安排期中考试。
我们注重学期中的学习过程评定，因此成绩评定不仅考虑学生的考试成绩，还综合考虑学生的平时表现和参与度。
期末机考时间：第16周上机时间 2024/6/5 15:08~17:00
笔试时间：第2周 周二（2024.06.18）下午

1.9

1）请大家准备好可用的gpt，我用的是 poe.com 。课程相关问题，都可以在课程微信群里面问。例如：考试时间、作业、上机、签到、大作业有否等。

2）课程网站使用 canvas, https://pku.instructure.com。学校通知canvas，在3月1日导入选课名单后启用。

3）可能需要北大学长提供的Clash，请自己取用。
https://189854.xyz/verify/
https://blog.189854.xyz/blog/walless/2023/11/04/clash.html

1.10

教发中心通知，新学期北大Canvas系统课程：
本学期Canvas用户数量将严格控制，学生名单将在补退选结束后导入（本科生3月1日，研究生3月5日）。特殊情况，请发邮件说明。

O365（含teams）账号申请链接：https://www.wjx.cn/vm/Y5XwfHD.aspx#

1.11

如果方便，发邮件感谢教务老师，是她帮忙扩的开课人数。

信息科学技术学院教务：董晓晖，地点：理科1号楼1118，电话：010-62755414，邮箱：dxh@pku.edu.cn

二、导论

1.1 本章目标

❏ 复习计算机科学、编程以及解决问题方面的知识。
❏ 理解抽象这一概念及其在解决问题的过程中所发挥的作用。
❏ 理解并建立抽象数据类型的概念。
❏ 复习Python。

1.2 入门

自从第一台利用转接线和开关来传递计算指令的电子计算机诞生以来，人们对编程的认识历经了多次变化。与社会生活的其他许多方面一样，计算机技术的变革为计算机科学家提供了越来越多的工具和平台去施展他们的才能。高效的处理器、高速网络以及大容量内存等一系列新技术，要求计算机科学家掌握更多复杂的知识。然而，在这一系列快速的变革之中，仍有一些基本原则始终保持不变。计算机科学被认为是一门利用计算机来解决问题的学科。

你肯定在学习解决问题的基本方法上投入过大量的时间，并且相信自己拥有根据问题描述构建解决方案的能力。你肯定也体会到了编写计算机程序的困难之处。大型难题及其解决方案的复杂性往往会掩盖问题解决过程的核心思想。

本章将为后续各章重点解释两个重要的话题。首先，本章会复习计算机科学以及数据结构与算法的研究必须符合的框架，尤其是学习这些内容的原因以及为什么说理解它们有助于更好地解决问题。其次，本章会复习Python。尽管不会提供完整、详尽的Python参考资料，但是会针对阅读后续各章所需的基础知识及基本思想，给出示例以及相应的解释。

1.3 何谓计算机科学

要定义计算机科学，通常十分困难，这也许是因为其中的“计算机”一词。你可能已经意识到，计算机科学并不仅是研究计算机本身。尽管计算机在这一学科中是非常重要的工具，但也仅仅只是工具而已。

计算机科学的研究对象是问题、解决问题的过程，以及通过该过程得到的解决方案。给定一个问题，计算机科学家的目标是开发一个能够逐步解决该问题的**算法**。算法是具有有限步骤的过程，依照这个过程便能解决问题。因此，算法就是解决方案。

可以认为计算机科学就是研究算法的学科。但是必须注意，某些问题并没有解决方案。尽管这一话题已经超出了本书讨论的范畴，但是对于学习计算机科学的人来说，认清这一事实非常重要。结合上述两类问题，可以将计算机科学更完善地定义为：研究问题及其解决方案，以及研究目前无解的问题的学科。

在描述问题及其解决方案时，经常用到“**可计算**”一词。若存在能够解决某个问题的算法，那么该问题便是可计算的。因此，计算机科学也可以被定义为：研究可计算以及不可计算的问题，即研究算法的存在性以及不存在性。在上述任意一种定义中，“计算机”一词都没有出现。解决方案本身是独立于计算机的。

在研究问题解决过程的同时，计算机科学也研究**抽象**。抽象思维使得我们能分别从逻辑视角和物理视角来看待问题及其解决方案。举一个常见的例子。

试想你每天开车去上学或上班。作为车的使用者，你在驾驶时会与它有一系列的交互：坐进车里，插入钥匙，启动发动机，换挡，刹车，加速以及操作方向盘。从抽象的角度来看，这是从逻辑视角来看待这辆车，你在使用由汽车设计者提供的功能来将自己从某个地方运送到另一个地方。这些功能有时候也被称作**接口**。

另一方面，修车工看待车辆的角度与司机截然不同。他不仅需要知道如何驾驶，而且更需要知道实现汽车功能的所有细节：发动机如何工作，变速器如何换挡，如何控制温度，等等。这就是所谓的物理视角，即看到表面之下的实现细节。

使用计算机也是如此。大多数人用计算机来写文档、收发邮件、浏览网页、听音乐、存储图像以及打游戏，但他们并不需要了解这些功能的实现细节。大家都是从逻辑视角或者使用者的角度来看待计算机。计算机科学家、程序员、技术支持人员以及系统管理员则从另一个角度来看待计算机。他们必须知道操作系统的原理、网络协议的配置，以及如何编写各种脚本来控制计算机。他们必须能够控制用户不需要了解的底层细节。

上面两个例子的共同点在于，抽象的用户（或称客户）只需要知道接口是如何工作的，而并不需要知道实现细节。这些接口是用户用于与底层复杂的实现进行交互的方式。下面是抽象的另一个例子，来看看Python的math模块。一旦导入这一模块，便可以进行如下的计算。
>>> import math

>>> math.sqrt(16)

4.0

>>>
这是一个**过程抽象**的例子。我们并不需要知道平方根究竟是如何计算出来的，而只需要知道计算平方根的函数名是什么以及如何使用它。只要正确地导入模块，便可以认为这个函数会返回正确的结果。由于其他人已经实现了平方根问题的解决方案，因此我们只需要知道如何使用该函数即可。这有时候也被称为过程的“黑盒”视角。我们仅需要描述接口：函数名、所需参数，以及返回值。所有的计算细节都被隐藏了起来，如图1-1所示。

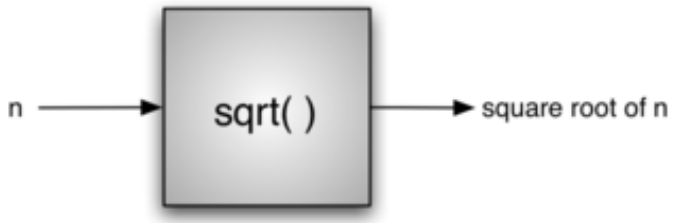

图1-1 过程抽象

1.3.1 何谓编程

编程是指通过编程语言将算法编码以使其能被计算机执行的过程。尽管有众多的编程语言和不同类型的计算机，但是首先得有一个解决问题的算法。如果没有算法，就不会有程序。

计算机科学的研究对象并不是编程。但是，编程是计算机科学家所做工作的一个重要组成部分。通常，编程就是为解决方案创造表达方式。因此，编程语言对算法的表达以及创造程序的过程是这一学科的基础。

通过定义表达问题实例所需的数据，以及得到预期结果所需的计算步骤，算法描述出了问题的解决方案。编程语言必须提供一种标记方式，用于表达过程和数据。为此，编程语言提供了众多的控制语句和数据类型。
Programming is often the way that we create a representation for our solutions. Therefore, this language representation and the process of creating it becomes a fundamental part of the discipline.

控制语句使算法步骤能够以一种方便且明确的方式表达出来。算法至少需要能够进行顺序执行、决策分支、循环迭代的控制语句。只要一种编程语言能够提供这些基本的控制语句，它就能够被用于描述算法。

计算机中的所有数据实例均由二进制字符串来表达。为了赋予这些数据实际的意义，必须要有**数据类型**。数据类型能够帮助我们解读二进制数据的含义，从而使我们能从待解决问题的角度来看待数据。这些内建的底层数据类型（又称原生数据类型）提供了算法开发的基本单元。

举例来说，大部分编程语言都为整数提供了相应的数据类型。根据整数（如23、654以及-19）的常见定义，计算机内存中的二进制字符串可以被理解成整数。除此以外，数据类型也描述了该类数据能参与的所有运算。对于整数来说，就有加减乘除等常见运算。并且，对于数值类型的数据，以上运算均成立。

我们经常遇到的困难是，问题及其解决方案都过于复杂。尽管由编程语言提供的简单的控制语句和数据类型能够表达复杂的解决方案，但它们在解决问题的过程中仍然存在不足。因此，我们需要想办法控制复杂度以利于找到解决方案。

1.3.2 为何学习数据结构及抽象数据类型

为了控制问题及其求解过程的复杂度，计算机科学家利用抽象来帮助自己专注于全局，从而避免迷失在众多细节中。通过对问题进行建模，可以更高效地解决问题。模型可以帮助计算机科学家更一致地描述算法要用到的数据。

如前所述，过程抽象将功能的实现细节隐藏起来，从而使用户能从更高的视角来看待功能。**数据抽象**的基本思想与此类似。**抽象数据类型**（有时简称为ADT）从逻辑上描述了如何看待数据及其对应运算而无须考虑具体实现。这意味着我们仅需要关心数据代表了什么，而可以忽略它们的构建方式。通过这样的抽象，我们对数据进行了一层封装，其基本思想是封装具体的实现细节，使它们对用户不可见，这被称为信息隐藏。

图1-2展示了抽象数据类型及其原理。用户通过利用抽象数据类型提供的操作来与接口交互。抽象数据类型是与用户交互的外壳。真正的实现则隐藏在内部。用户并不需要关心各种实现细节。

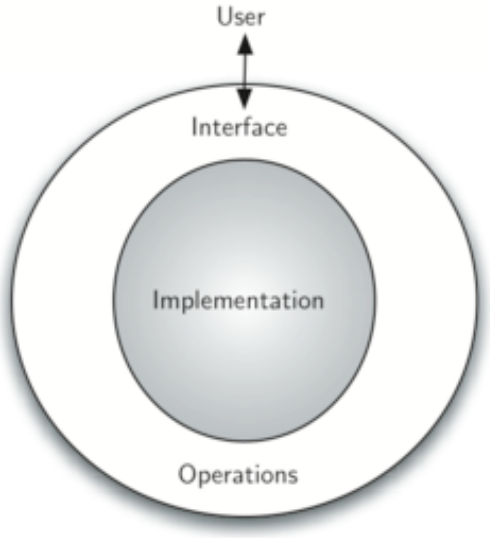

图1-2 抽象数据类型

抽象数据类型的实现常被称为**数据结构**，这需要我们通过编程语言的语法结构和原生数据类型从物理视角看待数据。正如之前讨论的，分成这两种不同的视角有助于为问题定义复杂的数据模型，而无须考虑模型的实现细节。这便提供了一个独立于实现的数据视角。由于实现抽象数据类型通常会有很多种方法，因此**独立于实现**的数据视角使程序员能够改变实现细节而不影响用户与数据的实际交互。用户能够始终专注于解决问题。

1.3.3 为何学习算法

计算机科学家通过经验来学习：观察他人如何解决问题，然后亲自解决问题。接触各种问题解决技巧并学习不同算法的设计方法，有助于解决新的问题。通过学习一系列不同的算法，可以举一反三，从而在遇到类似的问题时，能够快速加以解决。

各种算法之间往往差异巨大。回想前文提到的平方根的例子，完全可能有多种方法来实现计算平方根的函数。算法一可能使用了较少的资源，算法二返回结果所需的时间可能是算法一的10倍。我们需要某种方式来比较这两种算法。尽管这两种算法都能得到结果，但是其中一种可能比另一种“更好”——更高效、更快，或者使用的内存更少。随着对算法的进一步学习，你会掌握比较不同算法的分析技巧。这些技巧只依赖于算法本身的特性，而不依赖于程序或者实现算法的计算机的特性。

最坏的情况是遇到难以解决的问题，即没有算法能够在合理的时间内解决该问题。因此，至关重要的一点是，要能区分有解的问题、无解的问题，以及虽然有解但是需要过多的资源和时间来求解的问题。

在选择算法时，经常会有所权衡。除了有解决问题的能力之外，计算机科学家也需要知晓如何评估一个解决方案。总之，问题通常有很多解决方案，如何找到一个解决方案并且确定其为优秀的方案，是需要反复练习、熟能生巧的。

1.4 Python基础

本节将复习Python，并且为前一节提到的思想提供更详细的例子。如果你刚开始学习Python或者觉得自己需要更多的信息，建议你查看本书结尾列出的Python资源。本节的目标是帮助你复习Python并且强化一些会在后续各章中变得非常重要的概念。
Python是一门现代、易学、面向对象的编程语言。它拥有强大的内建数据类型以及简单易用的控制语句。由于Python是一门解释型语言，因此只需要查看和描述交互式会话就能进行学习。你应该记得，解释器会显示提示符>>>，然后计算你提供的Python语句。例如，以下代码显示了提示符、print函数、结果，以及下一个提示符。

```
>>> print("Algorithms and Data Structures")

>>> Algorithms and Data Structures

>>>
```

1.4.1 数据

前面提到，Python支持面向对象编程范式。这意味着Python认为数据是问题解决过程中的关键点。在Python以及其他所有面向对象编程语言中，类都是对数据的构成（状态）以及数据能做什么（行为）的描述。由于类的使用者只能看到数据项的状态和行为，因此类与抽象数据类型是相似的。在面向对象编程范式中，数据项被称作对象。一个对象就是类的一个实例。

1. 内建原子数据类型

我们首先复习原子数据类型。Python有两大内建数据类实现了整数类型和浮点数类型，相应的Python类就是int和float。标准的数学运算符，即+、-、＊、/以及＊＊（幂），可以和能够改变运算优先级的括号一起使用。其他非常有用的运算符包括取余（取模）运算符%，以及整除运算符//。注意，当两个整数相除时，其结果是一个浮点数，而整除运算符截去小数部分，只返回商的整数部分。

Python通过bool类实现对表达真值非常有用的布尔数据类型。布尔对象可能的状态值是True或者False，布尔运算符有and、or以及not。

布尔对象也被用作相等（==）、大于（>）等比较运算符的计算结果。此外，结合使用关系运算符与逻辑运算符可以表达复杂的逻辑问题。表1-1展示了关系运算符和逻辑运算符。

表1-1 关系运算符和逻辑运算符

Operation Name	Operator	Explanation
less than	<	Less than operator
greater than	>	Greater than operator
less than or equal	<=	Less than or equal to operator
greater than or equal	>=	Greater than or equal to operator
equal	==	Equality operator
not equal	!=	Not equal operator
logical and	and	Both operands True for result to be True
logical or	or	One or the other operand is True for the result to be True
logical not	not	Negates the truth value, False becomes True, True becomes False

标识符在编程语言中被用作名字。Python中的标识符以字母或者下划线（_）开头，区分大小写，可以是任意长度。需要记住的一点是，采用能表达含义的名字是良好的编程习惯，这使程序代码更易阅读和理解。

当一个名字第一次出现在赋值语句的左边部分时，会创建对应的Python变量。赋值语句将名字与值关联起来。变量存的是指向数据的引用，而不是数据本身。

赋值语句改变了变量的引用，这体现了Python的动态特性。同样的变量可以指向许多不同类型的数据。

2. 内建集合数据类型

除了数值类和布尔类，Python还有众多强大的内建集合类。列表、字符串以及元组是概念上非常相似的有序集合，但是只有理解它们的差别，才能正确运用。集（set）和字典是无序集合。

列表是零个或多个指向Python数据对象的引用的有序集合，通过在方括号内以逗号分隔的一系列值来表达。空列表就是[ ]。列表是异构的，这意味着其指向的数据对象不需要都是同一个类，并且这一集合可以被赋值给一个变量。

由于列表是有序的，因此它支持一系列可应用于任意Python序列的运算，如表1-2所示。

表1-2 可应用于任意Python序列的运算

Operation Name	Operator	Explanation
indexing	[ ]	Access an element of a sequence
concatenation	+	Combine sequences together
repetition	*	Concatenate a repeated number of times
membership	in	Ask whether an item is in a sequence
length	len	Ask the number of items in the sequence
slicing	[:]	Extract a part of a sequence

需要注意的是，列表和序列的下标从0开始。myList[1:3]会返回一个包含下标从1到2的元素列表（并没有包含下标为3的元素）。
如果需要快速初始化列表，可以通过重复运算来实现，如下所示。

```
>>> myList = [0] * 6

>>> myList

[0, 0, 0, 0, 0, 0]
```

非常重要的一点是，重复运算返回的结果是序列中指向数据对象的引用的重复。下面的例子可以很好地说明这一点。

```
>>> myList = [1,2,3,4]

>>> A = [myList]*3

>>> A

[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]

>>> myList[2] = 45

>>> A

[[1, 2, 45, 4], [1, 2, 45, 4], [1, 2, 45, 4]]
```

变量A包含3个指向myList的引用。myList中的一个元素发生改变，A中的3处都随即改变。
列表支持一些用于构建数据结构的方法，如表1-3所示。后面的例子展示了用法。

表1-3 Python列表提供的方法

ethod Name	Use	Explanation
`append`	`alist.append(item)`	Adds a new item to the end of a list
`insert`	`alist.insert(i,item)`	Inserts an item at the ith position in a list
`pop`	`alist.pop()`	Removes and returns the last item in a list
`pop`	`alist.pop(i)`	Removes and returns the ith item in a list
`sort`	`alist.sort()`	Modifies a list to be sorted
`reverse`	`alist.reverse()`	Modifies a list to be in reverse order
`del`	`del alist[i]`	Deletes the item in the ith position
`index`	`alist.index(item)`	Returns the index of the first occurrence of `item`
`count`	`alist.count(item)`	Returns the number of occurrences of `item`
`remove`	`alist.remove(item)`	Removes the first occurrence of `item`

你会发现，像pop这样的方法在返回值的同时也会修改列表的内容，reverse等方法则仅修改列表而不返回任何值。pop默认返回并删除列表的最后一个元素，但是也可以用来返回并删除特定的元素。这些方法默认下标从0开始。你也会注意到那个熟悉的句点符号，它被用来调用某个对象的方法。

range是一个常见的Python函数，我们常把它与列表放在一起讨论。range会生成一个代表值序列的范围对象。使用list函数，能够以列表形式看到范围对象的值。

范围对象表示整数序列。默认情况下，它从0开始。如果提供更多的参数，它可以在特定的点开始和结束，并且跳过中间的值。在第一个例子中，range(10)从0开始并且一直到9为止（不包含10）；在第二个例子中，range(5,10)从5开始并且到9为止（不包含10）；range(5,10,2)的结果类似，但是元素的间隔变成了2（10还是没有包含在其中）。

字符串是零个或多个字母、数字和其他符号的有序集合。这些字母、数字和其他符号被称为字符。常量字符串值通过引号（单引号或者双引号均可）与标识符进行区分。

由于字符串是序列，因此之前提到的所有序列运算符都能用于字符串。此外，字符串还有一些特有的方法，表1-4列举了其中一些。

表1-4 Python字符串提供的方法

Method Name	Use	Explanation
`center`	`astring.center(w)`	Returns a string centered in a field of size `w`
`count`	`astring.count(item)`	Returns the number of occurrences of `item` in the string
`ljust`	`astring.ljust(w)`	Returns a string left-justified in a field of size `w`
`lower`	`astring.lower()`	Returns a string in all lowercase
`rjust`	`astring.rjust(w)`	Returns a string right-justified in a field of size `w`
`find`	`astring.find(item)`	Returns the index of the first occurrence of `item`
`split`	`astring.split(schar)`	Splits a string into substrings at `schar`

split在处理数据的时候非常有用。split接受一个字符串，并且返回一个由分隔字符作为分割点的字符串列表。

列表和字符串的主要区别在于，列表能够被修改，字符串则不能。列表的这一特性被称为可修改性。列表具有可修改性，字符串则不具有。

由于都是异构数据序列，因此元组与列表非常相似。它们的区别在于，元组和字符串一样是不可修改的。元组通常写成由括号包含并且以逗号分隔的一系列值。与序列一样，元组允许之前描述的任一操作。

集（set）是由零个或多个不可修改的Python数据对象组成的无序集合。集不允许重复元素，并且写成由花括号包含、以逗号分隔的一系列值。空集由set()来表示。集是异构的。

尽管集是无序的，但它还是支持之前提到的一些运算，如表1-5所示。

表1-5 Python集支持的运算

Operation Name	Operator	Explanation
membership	in	Set membership
length	len	Returns the cardinality of the set
`|`	`aset | otherset`	Returns a new set with all elements from both sets
`&`	`aset & otherset`	Returns a new set with only those elements common to both sets
`-`	`aset - otherset`	Returns a new set with all items from the first set not in second
`<=`	`aset <= otherset`	Asks whether all elements of the first set are in the second

集支持一系列方法，如表1-6所示。在数学中运用过集合概念的人应该对它们非常熟悉。Note that `union`, `intersection`, `issubset`, and `difference` all have operators that can be used as well.

Note that `union`, `intersection`, `issubset`, and `difference` all have operators that can be used as well.

表1-6 Python集提供的方法

Method Name	Use	Explanation
`union`	`aset.union(otherset)`	Returns a new set with all elements from both sets
`intersection`	`aset.intersection(otherset)`	Returns a new set with only those elements common to both sets
`difference`	`aset.difference(otherset)`	Returns a new set with all items from first set not in second
`issubset`	`aset.issubset(otherset)`	Asks whether all elements of one set are in the other
`add`	`aset.add(item)`	Adds item to the set
`remove`	`aset.remove(item)`	Removes item from the set
`pop`	`aset.pop()`	Removes an arbitrary element from the set
`clear`	`aset.clear()`	Removes all elements from the set

字典是无序结构，由相关的元素对构成，其中每对元素都由一个键和一个值组成。这种键-值对通常写成键：值的形式。字典由花括号包含的一系列以逗号分隔的键-值对表达，

可以通过键访问其对应的值，也可以向字典添加新的键-值对。访问字典的语法与访问序列的语法十分相似，只不过是使用键来访问，而不是下标。添加新值也类似。

需要谨记，字典并不是根据键来进行有序维护的。键的位置是由散列来决定的，后续章节会详细介绍散列。len函数对字典的功能与对其他集合的功能相同。

字典既有运算符，又有方法。表1-7和表1-8分别展示了它们。keys、values和items方法均会返回包含相应值的对象。可以使用list函数将字典转换成列表。在表1-8中可以看到，get方法有两种版本。如果键没有出现在字典中，get会返回None。然而，第二个可选参数可以返回特定值。

表1-7 Python字典支持的运算

Operator	Use	Explanation
`[]`	`myDict[k]`	Returns the value associated with `k`, otherwise its an error
`in`	`key in adict`	Returns `True` if key is in the dictionary, `False` otherwise
`del`	del `adict[key]`	Removes the entry from the dictionary

表1-8 Python字典提供的方法

Method Name	Use	Explanation
`keys`	`adict.keys()`	Returns the keys of the dictionary in a dict_keys object
`values`	`adict.values()`	Returns the values of the dictionary in a dict_values object
`items`	`adict.items()`	Returns the key-value pairs in a dict_items object
`get`	`adict.get(k)`	Returns the value associated with `k`, `None` otherwise
`get`	`adict.get(k,alt)`	Returns the value associated with `k`, `alt` otherwise

1.4.2 输入与输出

程序经常需要与用户进行交互，以获得数据或者提供某种结果。目前的大多数程序使用对话框作为要求用户提供某种输入的方式。尽管Python确实有方法来创建这样的对话框，但是可以利用更简单的函数。Python提供了一个函数，它使得我们可以要求用户输入数据并且返回一个字符串的引用。这个函数就是input。

input函数接受一个字符串作为参数。由于该字符串包含有用的文本来提示用户输入，因此它经常被称为提示字符串。

不论用户在提示字符串后面输入什么内容，都会被存储在aName变量中。使用input函数，可以非常简便地写出程序，让用户输入数据，然后再对这些数据进行进一步处理。

需要注意的是，input函数返回的值是一个字符串，它包含用户在提示字符串后面输入的所有字符。如果需要将这个字符串转换成其他类型，必须明确地提供类型转换。

格式化字符串
print函数为输出Python程序的值提供了一种非常简便的方法。它接受零个或者多个参数，并且将单个空格作为默认分隔符来显示结果。通过设置sep这一实际参数可以改变分隔符。此外，每一次打印都默认以换行符结尾。这一行为可以通过设置实际参数end来更改。

更多地控制程序的输出格式经常十分有用。幸运的是，Python提供了另一种叫作格式化字符串的方式。格式化字符串是一个模板，其中包含保持不变的单词或空格，以及之后插入的变量的占位符。

1.4.3 控制结构

算法需要两个重要的控制结构：迭代和分支。Python通过多种方式支持这两种控制结构。程序员可以根据需要选择最有效的结构。

对于迭代，Python提供了标准的while语句以及非常强大的for语句。while语句会在给定条件为真时重复执行一段代码。

分支语句允许程序员进行询问，然后根据结果，采取不同的行动。绝大多数的编程语言都提供两种有用的分支结构：if else和if。

和其他所有控制结构一样，分支结构支持嵌套，一个问题的结果能帮助决定是否需要继续问下一个问题。

另一种表达嵌套分支的语法是使用elif关键字。将else和下一个if结合起来，可以减少额外的嵌套层次。注意，最后的else仍然是必需的，它用来在所有分支条件都不满足的情况下提供默认分支。

列表可以可以不通过迭代结构和分支结构来创建，这种方式被称为列表解析式。通过列表解析式，可以根据一些处理和分支标准轻松创建列表。Returning to lists, there is an alternative method for creating a list that uses iteration and selection constructs known as a **list comprehension**. A list comprehension allows you to easily create a list based on some processing or selection criteria.

1.4.4 异常处理

在编写程序时通常会遇到两种错误。第一种是语法错误，也就是说，程序员在编写语句或者表达式时出错。

第二种是逻辑错误，即程序能执行完成但返回了错误的结果。这可能是由于算法本身有错，或者程序员没有正确地实现算法。有时，逻辑错误会导致诸如除以0、越界访问列表等非常严重的情况。这些逻辑错误会导致运行时错误，进而导致程序终止运行。通常，这些运行时错误被称为异常。

许多初级程序员简单地把异常等同于引起程序终止的严重运行时错误。然而，大多数编程语言都提供了让程序员能够处理这些错误的方法。此外，程序员也可以在检测到程序执行有问题的情况下自己创建异常。
当异常发生时，我们称程序“抛出”异常。可以用try语句来“处理”被抛出的异常。

1.4.5 定义函数

之前的过程抽象例子调用了Python数学模块中的sqrt函数来计算平方根。通常来说，可以通过定义函数来隐藏任何计算的细节。函数的定义需要一个函数名、一系列参数以及一个函数体。函数也可以显式地返回一个值。

1.4.6 Python面向对象编程：定义类

前文说过，Python是一门面向对象的编程语言。到目前为止，我们已经使用了一些内建的类来展示数据和控制结构的例子。面向对象编程语言最强大的一项特性是允许程序员（问题求解者）创建全新的类来对求解问题所需的数据进行建模。

我们之前使用了抽象数据类型来对数据对象的状态及行为进行逻辑描述。通过构建能实现抽象数据类型的类，可以利用抽象过程，同时为真正在程序中运用抽象提供必要的细节。每当需要实现抽象数据类型时，就可以创建新类。

1. Fraction类

要展示如何实现用户定义的类，一个常用的例子是构建实现抽象数据类型Fraction的类。我们已经看到，Python提供了很多数值类。但是在有些时候，需要创建“看上去很像”分数的数据对象。

像 $\frac{3}{5}$ 这样的分数由两部分组成。上面的值称作分子，可以是任意整数。下面的值称作分母，可以是任意大于0的整数（负的分数带有负的分子）。尽管可以用浮点数来近似表示分数，但我们在此希望能精确表示分数的值。

Fraction对象的表现应与其他数值类型一样。我们可以针对分数进行加、减、乘、除等运算，也能够使用标准的斜线形式来显示分数，比如3/5。此外，所有的分数方法都应该返回结果的最简形式。这样一来，不论进行何种运算，最后的结果都是最简分数。

在Python中定义新类的做法是，提供一个类名以及一整套与函数定义语法类似的方法定义。以下是一个方法定义框架。

```
class Fraction

    #the methods go here
```

所有类都应该首先提供构造方法。构造方法定义了数据对象的创建方式。要创建一个Fraction对象，需要提供分子和分母两部分数据。在Python中，构造方法总是命名为`__init__`（即在init的前后分别有两个下划线），如代码清单1-2所示。

代码清单1-2 Fraction类及其构造方法

```
def __init__(self, top, bottom):

    self.num = top
    self.den = bottom
```

注意，形式参数列表包含3项。self是一个总是指向对象本身的特殊参数，它必须是第一个形式参数。然而，在调用方法时，从来不需要提供相应的实际参数。如前所述，分数需要分子与分母两部分状态数据。构造方法中的self.num定义了Fraction对象有一个叫作num的内部数据对象作为其状态的一部分。同理，self.den定义了分母。这两个实际参数的值在初始时赋给了状态，使得新创建的Fraction对象能够知道其初始值。

要创建Fraction类的实例，必须调用构造方法。使用类名并且传入状态的实际值就能完成调用（注意，不要直接调用`__init__`）。

```
myfraction = Fraction(3,5)
```

以上代码创建了一个对象，名为myfraction，值为3/5。图1-5展示了这个对象。

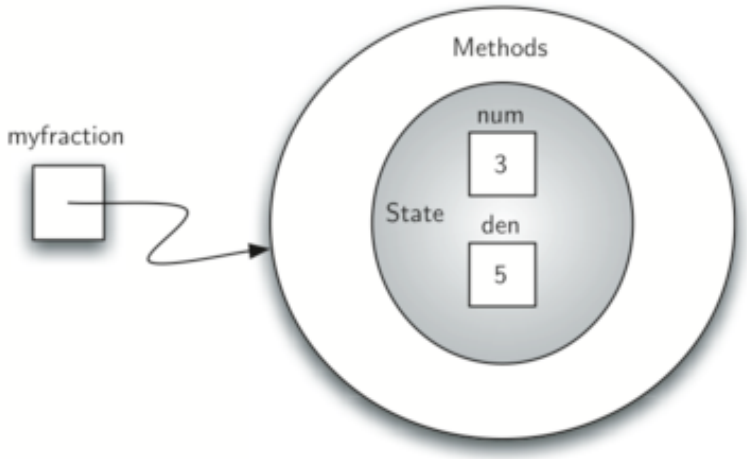

图1-5 Fraction类的一个实例

接下来实现这一抽象数据类型所需要的行为。考虑一下，如果试图打印Fraction对象，会发生什么呢?

```
>>> myf = Fraction(3,5)
>>> print(myf)
<__main__.Fraction instance at 0x409b1acc>
```

Fraction对象myf并不知道如何响应打印请求。print函数要求对象将自己转换成一个可以被写到输出端的字符串。myf唯一能做的就是显示存储在变量中的实际引用（地址本身）。这不是我们想要的结果。

有两种办法可以解决这个问题。一种是定义一个show方法，使得Fraction对象能够将自己作为字符串来打印。代码清单1-3展示了该方法的实现细节。如果像之前那样创建一个Fraction对象，可以要求它显示自己（或者说，用合适的格式将自己打印出来）。不幸的是，这种方法并不通用。为了能正确打印，我们需要告诉Fraction类如何将自己转换成字符串。要完成任务，这是print函数所必需的。

代码清单1-3 show方法

```
def show(self):
     print(self.num,"/",self.den)

>>> myf = Fraction(3,5)
>>> myf.show()
3 / 5
>>> print(myf)
<__main__.Fraction instance at 0x40bce9ac>
>>>
```

Python的所有类都提供了一套标准方法，但是可能没有正常工作。其中之一就是将对象转换成字符串的方法**str**。这个方法的默认实现是像我们之前所见的那样返回实例的地址字符串。我们需要做的是为这个方法提供一个“更好”的实现，即重写默认实现，或者说重新定义该方法的行为。

为了达到这一目标，仅需定义一个名为**str**的方法，并且提供新的实现，如代码清单1-4所示。除了特殊参数self之外，该方法定义不需要其他信息。新的方法通过将两部分内部状态数据转换成字符串并在它们之间插入字符/来将分数对象转换成字符串。一旦要求Fraction对象转换成字符串，就会返回结果。注意该方法的各种用法。

代码清单1-4 **str**方法

```
def __str__(self):
    return str(self.num)+"/"+str(self.den)
>>> myf = Fraction(3,5)
>>> print(myf)
3/5
>>> print("I ate", myf, "of the pizza")
I ate 3/5 of the pizza
>>> myf.__str__()
'3/5'
>>> str(myf)
'3/5'
>>>
```

可以重写Fraction类中的很多其他方法，其中最重要的一些是基本的数学运算。我们想创建两个Fraction对象，然后将它们相加。目前，如果试图将两个分数相加，会得到下面的结果。

```
>>> f1 = Fraction(1,4)
>>> f2 = Fraction(1,2)
f1+f2

Traceback (most recent call last):
  File "<pyshell#173>", line 1, in -toplevel-
    f1+f2
TypeError: unsupported operand type(s) for +:
          'instance' and 'instance'
>>>
```

如果仔细研究这个错误，会发现加号+无法处理Fraction的操作数。
可以通过重写Fraction类的`__add__`方法来修正这个错误。该方法需要两个参数。第一个仍然是self，第二个代表了表达式中的另一个操作数。

```
f1.__add__(f2)
```

以上代码会要求Fraction对象f1将Fraction对象f2加到自己的值上。可以将其写成标准表达式：f1 + f2。
两个分数需要有相同的分母才能相加。确保分母相同最简单的方法是使用两个分母的乘积作为分母。

$$\frac{a}{b}+\frac{c}{d}=\frac{ad}{bd}+\frac{cb}{bd}=\frac{ad+cb}{bd}$$

代码清单1-5展示了具体实现。`__add__`方法返回一个包含分子和分母的新Fraction对象。可以利用这一方法来编写标准的分数数学表达式，将加法结果赋给变量，并且打印结果。值得注意的是，第3行中的\称作续行符。当一条Python语句被分成多行时，需要用到续行符。

代码清单1-5 `__add__`方法

```
def __add__(self,otherfraction):

    newnum = self.num*otherfraction.den + \
               self.den*otherfraction.num
    newden = self.den * otherfraction.den

    return Fraction(newnum,newden)
>>> f1=Fraction(1,4)
>>> f2=Fraction(1,2)
>>> f3=f1+f2
>>> print(f3)
6/8
>>>
```

虽然这一方法能够与我们预想的一样执行加法运算，但是还有一处可以改进。1/4+1/2的确等于6/8，但它并不是最简分数。最好的表达应该是3/4。为了保证结果总是最简分数，需要一个知道如何化简分数的辅助方法。该方法需要寻找分子和分母的最大公因数（greatest common divisor, GCD），然后将分子和分母分别除以最大公因数，最后的结果就是最简分数。

要寻找最大公因数，最著名的方法就是欧几里得算法，第8章将详细讨论。欧几里得算法指出，对于整数m和n，如果m能被n整除，那么它们的最大公因数就是n。然而，如果m不能被n整除，那么结果是n与m除以n的余数的最大公因数。代码清单1-6提供了一个迭代实现。注意，这种实现只有在分母为正的时候才有效。对于Fraction类，这是可以接受的，因为之前已经定义过，负的分数带有负的分子，其分母为正。

代码清单1-6 gcd函数

```
def gcd(m,n):
    while m%n != 0:
        oldm = m
        oldn = n

        m = oldn
        n = oldm%oldn
    return n

print(gcd(20,10))
```

现在可以利用这个函数来化简分数。为了将一个分数转化成最简形式，需要将分子和分母都除以它们的最大公因数。对于分数6/8，最大公因数是2。因此，将分子和分母都除以2，便得到3/4。代码清单1-7展示了实现细节。

代码清单1-7 改良版`__add__`方法

```
def __add__(self,otherfraction):
    newnum = self.num*otherfraction.den + self.den*otherfraction.num
    newden = self.den * otherfraction.den
    common = gcd(newnum,newden)
    return Fraction(newnum//common,newden//common)

>>> f1=Fraction(1,4)
>>> f2=Fraction(1,2)
>>> f3=f1+f2
>>> print(f3)
3/4
>>>
```

Fraction对象现在有两个非常有用的方法，如图1-6所示。为了允许两个分数互相比较，还需要添加一些方法。假设有两个Fraction对象，f1和f2。只有在它们是同一个对象的引用时，f1 == f2才为True。这被称为浅相等，如图1-7所示。在当前实现中，分子和分母相同的两个不同的对象是不相等的。

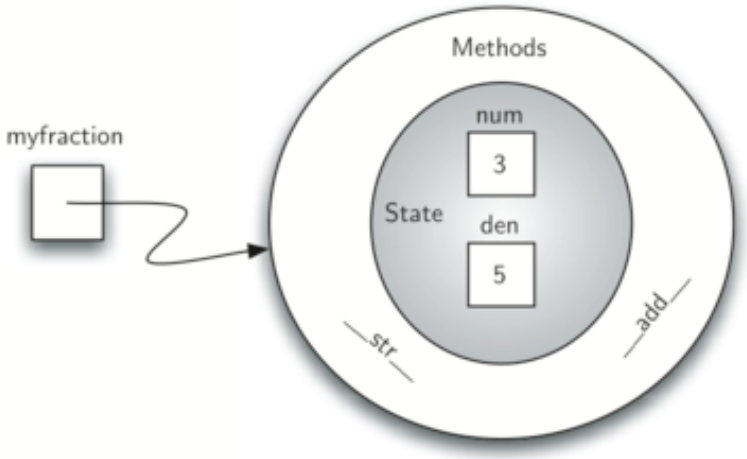

图1-6 包含两个方法的Fraction实例

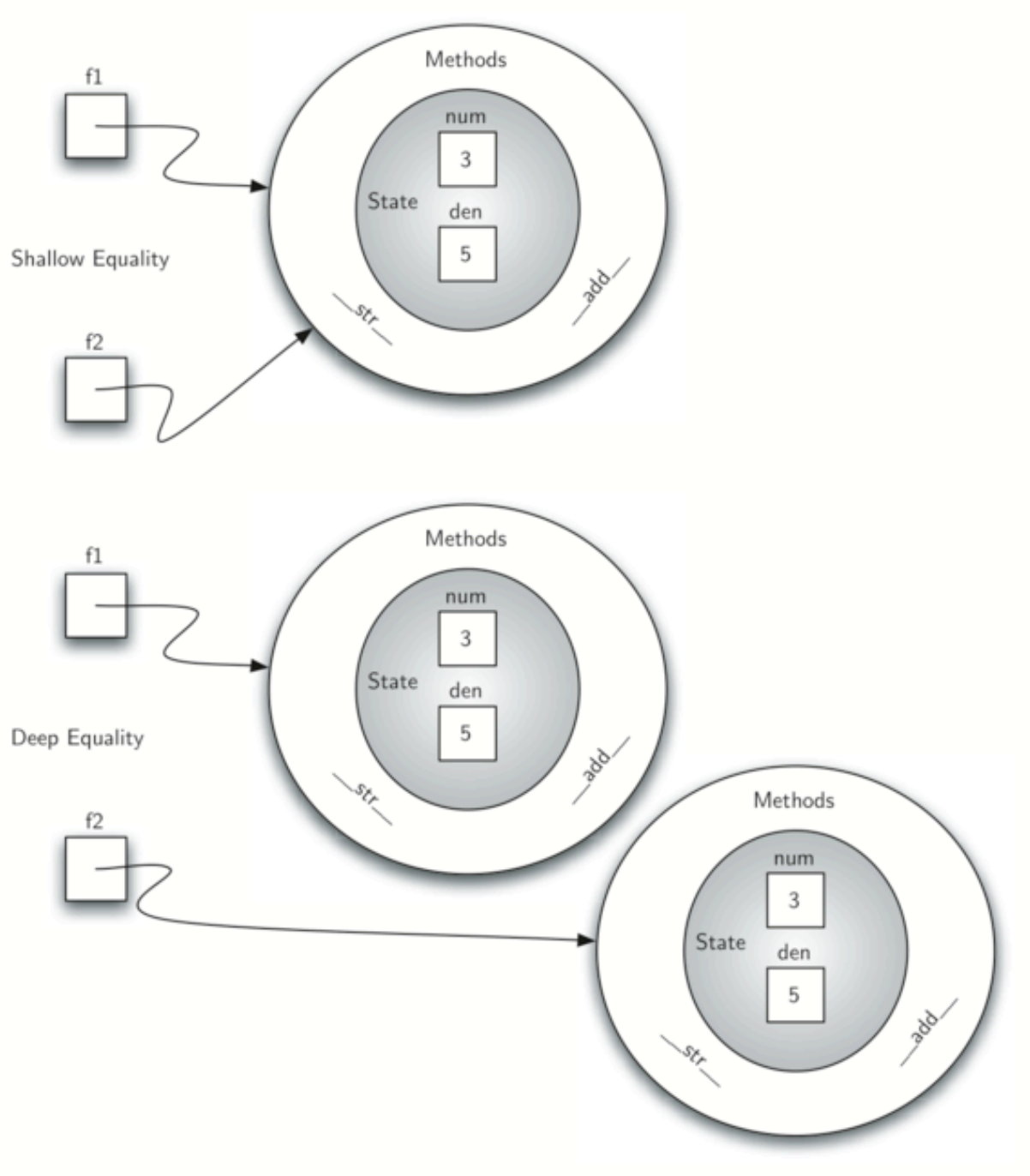

图1-7 浅相等与深相等

通过重写`__eq__`方法，可以建立深相等——根据值来判断相等，而不是根据引用。`__eq__`是又一个在任意类中都有的标准方法。它比较两个对象，并且在它们的值相等时返回True，否则返回False。
在Fraction类中，可以通过统一两个分数的分母并比较分子来实现`__eq__`方法，如代码清单1-8所示。需要注意的是，其他的关系运算符也可以被重写。例如，`__le__`方法提供判断小于等于的功能。

代码清单1-8 `__eq__`方法

```
def __eq__(self, other):
    firstnum = self.num * other.den
    secondnum = other.num * self.den

    return firstnum == secondnum
```

代码清单1-9提供了到目前为止Fraction类的完整实现。剩余的算术方法及关系方法留作练习。

代码清单1-9 Fraction类的完整实现

```
def gcd(m,n):
    while m%n != 0:
        oldm = m
        oldn = n

        m = oldn
        n = oldm%oldn
    return n

class Fraction:
     def __init__(self,top,bottom):
         self.num = top
         self.den = bottom

     def __str__(self):
         return str(self.num)+"/"+str(self.den)

     def show(self):
         print(self.num,"/",self.den)

     def __add__(self,otherfraction):
         newnum = self.num*otherfraction.den + \
                      self.den*otherfraction.num
         newden = self.den * otherfraction.den
         common = gcd(newnum,newden)
         return Fraction(newnum//common,newden//common)

     def __eq__(self, other):
         firstnum = self.num * other.den
         secondnum = other.num * self.den

         return firstnum == secondnum

x = Fraction(1,2)
y = Fraction(2,3)
print(x+y)
print(x == y)
```

2. 继承Inheritance

最后一节介绍面向对象编程的另一个重要方面。继承使一个类与另一个类相关联，就像人们相互联系一样。孩子从父母那里继承了特征。与之类似，Python中的子类可以从父类继承特征数据和行为。父类也称为超类。

图1-8展示了内建的Python集合类以及它们的相互关系。我们将这样的关系结构称为继承层次结构。举例来说，列表是有序集合的子。因此，我们将列表称为子，有序集合称为父（或者分别称为子类列表和超类序列）。这种关系通常被称为IS-A关系（IS-A意即列表是一个有序集合）。这意味着，列表从有序集合继承了重要的特征，也就是内部数据的顺序以及诸如拼接、重复和索引等方法。

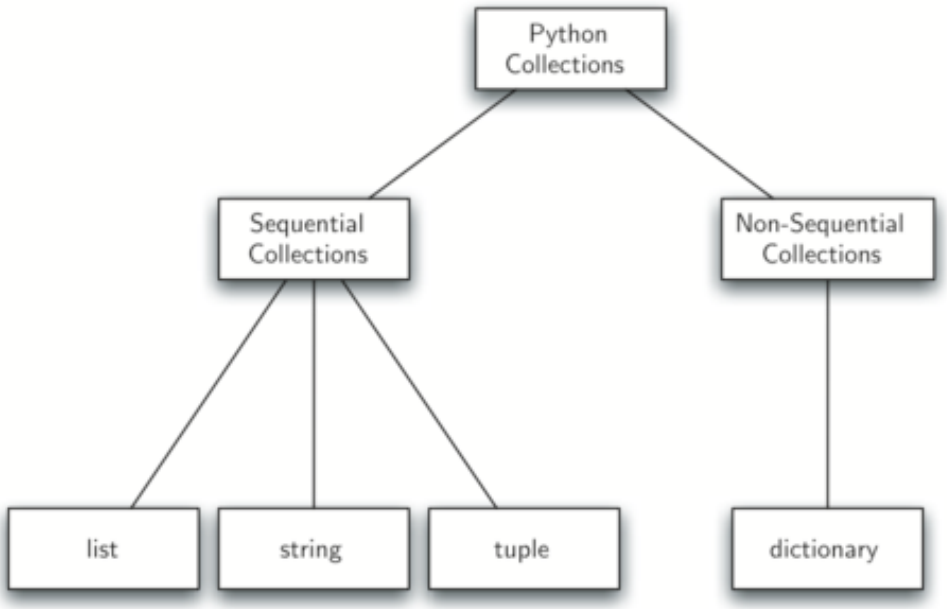

图1-8 Python集合类的继承层次结构

列表、字符串和元组都是有序集合。它们都继承了共同的数据组织和操作。不过，根据数据是否同类以及集合是否可修改，它们彼此又有区别。子类从父类继承共同的特征，但是通过额外的特征彼此区分。

通过将类组织成继承层次结构，面向对象编程语言使以前编写的代码得以扩展到新的应用场景中。此外，这种结构有助于更好地理解各种关系，从而更高效地构建抽象表示。

1.5 Summary

- Computer science is the study of problem solving.
- Computer science uses abstraction as a tool for representing both processes and data.
- Abstract data types allow programmers to manage the complexity of a problem domain by hiding the details of the data.
- Python is a powerful, yet easy-to-use, object-oriented language.
- Lists, tuples, and strings are built in Python sequential collections.
- Dictionaries and sets are nonsequential collections of data.
- Classes allow programmers to implement abstract data types.
- Programmers can override standard methods as well as create new methods.
- Classes can be organized into hierarchies.

- A class constructor should always invoke the constructor of its parent before continuing on with its own data and behavior.

1.6 Key Terms

abstract data type	**abstraction**	**algorithm**
class	computable	data abstraction
data structure	data type	deep equality
dictionary	encapsulation	exception
format operator	formatted strings	HAS-A relationship
implementation-independent	information hiding	inheritance
inheritance hierarchy	interface	IS-A relationship
list	list comprehension	method
mutability	object	procedural abstraction
programming	prompt	`self`
shallow equality	simulation	string
subclass	superclass	truth table

参考

Python数据结构与算法分析(第2版)，布拉德利·米勒 戴维·拉努姆/吕能,刁寿钧译，出版时间:2019-09

Brad Miller and David Ranum, Problem Solving with Algorithms and Data Structures using Python, https://runestone.academy/ns/books/published/pythonds/index.html