HZ BOOKS
华章教育

面向CS2013计算机专业规划教材

从问题到程序

用Python学编程和计算

裘宗燕
北京大学 著

From Problems to Programs

Learn Computing and Programming using Python

机械工业出版社
China Machine Press

面向CS2013计算机专业规划教材

从问题到程序

用Python学编程和计算

裘宗燕
北京大学 著

From Problems to Programs

Learn Computing and Programming using Python

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

从问题到程序：用 Python 学编程和计算 / 裘宗燕著 . —北京：机械工业出版社，2017.4
（面向 CS2013 计算机专业规划教材）

ISBN 978-7-111-56445-4

I. 从… II. 裘… III. 软件工具 – 程序设计 – 高等学校 – 教材 IV. TP311.561

中国版本图书馆 CIP 数据核字（2017）第 081844 号

本书是以 Python 为编程语言、面向计算机科学教育中的程序设计基础课程与编程初学者的入门教材和自学读物。本书以 Python 为工具，详细讨论了与编程有关的各方面问题，介绍了从初级到高级的许多重要编程技术。本书特别强调编程中的分析和思考、问题的严格化和逐步分解、语言结构的正确选择、程序结构的良好组织，以及程序的正确和安全。书中通过大量实例及其开发过程，展示了好程序的特征和正确的编程工作方法。此外，书中还介绍了 Python 语言的重要细节和工作原理。各章附有大量习题。

本书既可以作为高校程序设计相关课程的教材，也适合希望学习 Python 语言和编程技术的读者阅读和参考。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：朱 劼	责任校对：殷 虹
印 刷：北京瑞德印刷有限公司	版 次：2017 年 6 月第 1 版第 1 次印刷
开 本：185mm × 260mm 1/16	印 张：28.75
书 号：ISBN 978-7-111-56445-4	定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

前　言

计算机诞生至今不过六七十年，但它已经改变了世界，改变了每个人的生活。人们每天都在与计算机交流（如智能手机），各领域专业人员的大量日常工作都需要使用计算机，从事与计算机相关工作的人们已经发展为社会上最大的专业技术社团。计算机的研究和应用、互联网和其他相关领域，还在不断呼唤大量熟悉计算机的专业开发人才。计算机科学技术的开发和应用能力已被广泛认为是国家竞争力的重要组成部分。因此，学习计算机科学技术知识，不仅是社会发展的需要，而且已成为个人的重要职业竞争力。然而，要深入理解计算和计算机，使其成为得心应手的工具，就必须学习编程。

近年来，Python 语言的良好特征已经得到学界和业界的广泛认可，被世界上许多知名大学选作计算机科学教育的第一门语言，也被很多企业和实际项目作为开发语言。这些发展情况，使越来越多的人有了学习 Python 的兴趣和需要。

本书源自作者讲授基于 Python 的编程课程的经验，又经过认真整理，目标是提供一条清晰易行的学习路径。本书的结构和内容力图反映编程的本质，可以作为高校计算机基础课程教材，也适合希望学习 Python 语言和编程技术的读者阅读和参考。本书书名反映了作者对编程的基本认识：**编程是从要解决的问题开始，最终得到解决问题的程序的过程**。要学好编程，就要努力去理解这个过程，还必须反复实践。

编程就是为了指挥计算机（而不是我们自己），通过一个自动计算过程（而不是人的操作）去解决问题。为了指挥计算机工作，我们需要理解计算机，理解计算过程，理解指挥计算机完成计算的途径和方法。还需要一种计算机能处理的表达方式，也就是编程语言（或称程序设计语言，本书中用 Python），用于说明计算机应该怎样工作。

程序是人写出的一段文字，表达的是要求计算机完成的一个计算过程。完成编程工作需要多方面的理解。首先需要理解面对的问题：它究竟是什么、要求做什么、可能用什么方法和计算过程去解决？要理解自己写出的程序，理解它描述的计算过程。最后的理解也非常重要：理解两者之间的关系，即自己的程序是否正确解决了相应的问题？没有对问题和程序的深入思考和理解，就不可能开发出好的、功能正确的程序。

学习编程时必须关注两方面的问题：如何从**一个解决问题的需求**出发，逐步开发出一个**计算过程**；如何使用编程语言（如 Python）正确地写出解决问题的**程序**，描述（实现）这个计算过程。本书的做法是提出一个个问题，从简单到复杂，讨论从它们出发的程序开发过程：分析问题的细节，设法将其严格化，提出可能的解决方案，再经分解和设计，以及随后的编码和调试，最终开发出一个程序。完成了一个程序，常常不是工作的结束，还需要回过头去考察这个程序，分析其优点和缺点，研究改进或变化的可能性。这样的程序开发过程和其中的分析、思考、选择、决策等，反映了编程工作的本质和正确工作方法。当然，考虑到读者的经验积累，较前章节中的讨论包含了更多细节，后面逐渐简化，可能只关注工作中的一些要点。但是实际上，对任何一个问题，无论简单或复杂，上述的思考和工作过程都会再一次重复。只不过随着读者的经验积累，一些问题的分析和决策变得愈发自然顺畅，以至于其明显性逐步减弱。这种

趋势也反映了读者的进步。

在阅读本书的过程中，建议读者去亲身体验程序的开发过程：看到一个问题之后，最好先不去看书中的分析和解决方法，而是自己思考，设法解决这个问题，完成之后再与书中的做法比较。如果两种做法不同，就应该分析它们各自的长处和短处。采用这样的方法学习本书（和编程），读者将不再是被动的接受者，而是主动参与者。

实际上，对每个问题，都可能有很多（理论上是无穷多）不同解决方法，对不同方法和设计的比较、选择和决策，也是编程工作中最重要的内容。对一个具体问题，总可能做出很多正确程序。如果问题稍微复杂，常能做出一组程序，其中任何一个都不比别的程序“更好”。这些情况说明，在编程学习中，并没有必须背诵的标准答案。当然，另一方面，作者也希望通过书中实例，反映良好的编程工作和好程序的特征。书中程序都由作者开发（其间也参考了许多材料），并经过仔细修改和运行试验。程序的格式符合 Python 社团的考虑，结构良好，实现方法清晰简洁，可以作为读者参考的范例。

编程语言是编程的工具，也是人与计算机交流的工具。它既要反映思维的特点，使人易于使用，也要反映计算机的特点，使计算机能按它写出的程序高效工作。Python 被广泛认为是一种比较适合初学者的语言，其设计较好地反映了计算机和编程的性质和特点，具有比较平滑的学习曲线。最简单的 Python 程序可以看作数学和算术的自然延续，可以方便地运行，立刻看到结果（或效果）。初学者可以从这里开始一步步深入，在解决越来越复杂的问题的过程中，逐渐领悟编程和计算的性质和本质。另一方面，Python 也支持许多高级编程概念，从高阶函数、生成器函数，到复杂数据对象的描述式、面向对象的编程等，反映了现代编程技术和需求的发展。本书中详细介绍了这些机制和技术，最后还介绍了两个重要的、有很强实际意义的编程问题：图形用户界面编程和并发编程。由于 Python 既容易入门又支持许多高级编程概念，因此是学习编程和计算的很好选择。

本书的主要特点有：首先是包含了对许多问题的深入讨论，既有针对具体编程问题的讨论，也有结合具体问题而展开的一般性讨论。具体到编程和 Python 语言，本书通过实例介绍了大量重要的概念和技术，其中有些专门针对 Python 语言，更多概念和技术具有一般性。书中还介绍了 Python 语言的一些重要细节和工作原理，以帮助读者深入理解 Python 程序，在编程中做出正确的设计选择。

本书包含八章和若干附录，下面简单介绍它们的情况：

第 1 章介绍一些基本情况。这里主要讨论计算机和程序的关系，程序设计的基本问题，Python 语言的历史、现状和应用情况，还概述了基本的程序开发过程。

第 2 章是计算和编程入门。这一章首先介绍简单的数值表达式的描述和求值、类型的概念和意义、数学函数包的使用、字符串的概念和基本运算。然后讨论变量和赋值、顺序计算的脚本程序、条件控制（`if` 语句）、重复计算和循环（`for` 和 `while` 语句），最后讨论了函数的概念、意义，以及基本的函数定义技术。

第 3 章集中讨论基本的编程技术。其中首先讨论循环的开发和实现、输入控制的循环等，然后介绍函数定义的递归方法、递归和循环的关系，最后集中讨论定义函数的思路和技术。这一章还介绍了一些与所有程序都有关的重要概念和问题，包括程序运行时间和计时、程序中的断言、循环不变式、程序正确性问题、程序的终止性等。

第 4 章讨论 Python 函数定义、程序结构和编程中的调试问题。这里首先介绍与程序的结构

和行为有关的一些基本问题，包括作用域、状态和环境的概念，程序执行中环境和状态的变化，模块的执行和导入等。然后介绍 Python 函数定义的结构、函数的执行和 Python 丰富的函数参数机制，并介绍了以函数为参数的高阶函数、`lambda` 表达式等。最后集中讨论程序的测试和调试，以及 IDLE 支持程序调试的功能。

第 5 章讨论复杂数据的组织和 Python 的组合数据类型。这里首先讨论了表、元组、字符串及其在程序中的使用，然后从序列的概念出发，统一地介绍了序列的构造技术和各种序列操作。随后介绍了字典和集合，以及这两种组合类型在程序中的应用。这一章还讨论了对象的变动性问题，介绍了字符串的格式化功能，详细解释了逻辑判断和逻辑表达式的求值过程，并对 Python 函数的参数问题做了总结。

第 6 章介绍一些深入的编程技术。这里首先讨论文件的概念和文件使用技术，并介绍了两种重要的函数技术：生成器函数和闭包。然后讨论错误处理问题和 Python 处理异常的结构。最后介绍了数据持久性问题，以及 Python 标准库支持持久性的 pickle 包。

第 7 章集中讨论面向对象的编程和 Python 的相关机制。这里首先介绍了类和对象的概念，类定义的基本技术和基于类的程序开发。然后讨论了类继承和派生类的定义，以及多继承问题等。这一章的最后介绍了 Python 的特殊方法名，以及利用它们模拟容器、迭代器、上下文管理器等的技术，还介绍了 `with` 语句的意义和应用。

第 8 章讨论两个重要的编程问题。首先介绍图形用户界面（GUI）编程的问题，以及基于标准库包 tkinter 的 GUI 编程技术。然后介绍并发编程的问题，以及基于标准库包 threading 的并发编程。这两部分都包含许多实例，还说明了在实际应用开发中使用这些技术时需要解决的一些问题和处理方法。

本书的附录包括 Python 语言速查手册、标准函数表、书中使用的几个标准库包的简单介绍，以及 IDLE 开发环境的命令简介。

本书各章后附有很多习题，供使用本书开设课程的教师和自学的读者参考。某些章的最后有一节 “补充材料”，其中介绍一些与该章内容有关但比较具体或初学阶段不常用的 Python 语言机制，有时还总结了该章中讨论的一些编程技术。有几章中的若干节被标以星号，它们或者是技术性较强，或者内容涉及较深入的问题，作为基础课程的教学或者自学者初学时，可以跳过这些部分，留待课下或以后阅读。

作者在北京大学数学学院讲授相关课程时，与参加学习的同学有很多讨论，这些讨论帮助作者澄清了很多重要问题。参与课程辅导工作的胡婷婷和陈晨以及刘海洋，都提出了一些很好的建议。在本书的成书过程中，机械工业出版社的朱劼编辑非常认真，提出了很多改进意见。对所有这些帮助，作者在此一并表示衷心感谢。最后，作者希望得到读者的意见和反馈，并先在这里表示感谢。

裘宗燕

2017 年 3 月

目　录

CHAPTER 1

第 1 章 程序设计和 Python

我们已经生活在信息时代，环顾四周，信息技术的影响无处不在。由于信息科学技术的发展和应用，我们的世界的方方面面都与 20 年前大不相同了，例如：

- 个人生活：看看人们在每天生活中做的各种事情，有多少是在与屏幕键盘（可能是触摸屏）交互，这些都是 20 年前没有的事情。
- 人际交流：20 年前的人际交流方式很简单。除面对面交流外，只能通过纸笔写信或长途电话（要找专门的电话或者到电话局）。今天人手一部手机，可以通过电话、短信、各种网络即时消息相互交流。电子邮件也是私人之间的交流媒介，而在网上的各种地方发帖和回复，则带有公开性质，是人际交流的新方式。
- 学习和教育：课堂教育的方式和手段都改变了。互联网越来越多被用于学习和知识的发掘整理，通过电子手段获取信息和知识的比重越来越大。
- 产业和职业：出现了大量与信息处理有关的职业，信息技术在越来越多的业务和工作领域发挥着越来越大的作用，各种职业的工作方式都改变了。
- 信息技术已经渗入经济、政治、社会生活的所有方面，具体的事例不胜枚举。

为什么在这不长的几十年时间里，人类社会的方方面面会发生这么大变化？回答很简单：因为有了计算机，这种强大、灵活、威力无穷的通用信息处理工具（它从根本上有别于人类发明的其他工具），改变了人类社会的每个方面。

那么，计算机的威力从哪里来？为什么一台计算机（基于电子元器件组合起来的一套设备）能完成如此丰富多彩的不同工作？本章首先希望回答这个问题。

本章将帮助读者在比较直观的层面建立起对计算机、计算、程序、程序设计和编程语言的基本认识，然后简单介绍本书使用的编程语言——Python 语言，最后介绍在学习实际编程中必然遇到的一些情况和问题。

1.1 计算机和程序

人类发展的历史，也是积累知识的历史。今天的人们未必比古人更聪明，至少相差不会太多。但是，今天人们掌握的知识比古人多得多，而且一年多于一年，一代多于一代。正是由于这种知识积累，使今天人类的生活与古人大不相同了。

1.1.1 “是什么”和“怎样做”的知识

人类积累的知识，从某种角度看可分为两大类：

- 一类是有关“是什么”的知识，人们通过观察、分析、研究、试验等活动，设法认识所

见所闻的事物，理解这些事物，设法解释事物的方方面面。

- 另一类是有关“怎样做”的知识，这些知识告诉人们如何去改变各种现存的事物，或者去创造从未有过的新鲜事物。

显然，这两类知识相互有联系，但又有各自不同的特点。它们不仅内容不同，表述的方式和手段通常也大不相同。下面看两个例子。

两个实例

在日常生活和学习中，到处都可以看到这两类不同知识，现举两例：

【例 1.1 菜单和菜谱】以常见的菜肴——西红柿炒鸡蛋为例，在饭店菜单上可以看到对它的描述：通过自然语言写出，介绍其色香味方面的特点，可能再附以照片给人感官印象。这是有关该菜肴的一套说明性描述，说明了西红柿炒鸡蛋**是什么**。而翻开一本菜谱，其中有关同一菜肴的描述则完全不同。首先是一个配料表，说明制作这一菜肴需要准备哪些基本食材和配料，烹制前如何处理（准备工作）。而后是有关用火和烹制过程的详细说明，告诉人们应该**怎样做**，才能得到一盘合格的西红柿炒鸡蛋。

显然，上述两种说明都针对同一客体，即同一盘菜肴，但它们显然很不一样。前者传递的是有关“是什么”的知识，或称*说明性的知识*，而后者传递的是有关“怎样做”的知识（操作性的，或称为*过程性的知识*）。作为饭店顾客，查阅菜单弄清菜肴的特点，符合自己的需要，就可以选择它。而作为学做菜的人们则需要参考菜谱实际操作，才能做出合乎预期的菜肴。

【例 1.2 最大公约数】最大公约数是基础数学中的一个基本概念，出现在小学数学课本里。作为数学概念，最大公约数有严格的数学定义：

定义：两个正整数 x 和 y 的最大公约数（常简写为 GCD），就是能整除 x 并能整除 y 的正整数中最大的那个数（允许至多一个数为 0 是本定义的扩充）。

这个定义基于几个更基本的数学概念（正整数、整除、最大），严格定义了一个新概念：最大公约数。只要理解了这几个基本概念，就能理解这个新概念。显然这一定义传播的一些知识属于**是什么**的说明性知识。但是这个定义没有说明在遇到一对正整数时，如何把它们的最大公约数求出来。后一工作就要靠过程性的知识了。

求最大公约数的一种著名方法称为*辗转相除法*，又称为*欧几里得方法*。该方法说明了求两个正整数的最大公约数的一种计算过程，可严格描述如下：

0. 计算的基础（输入）：正整数 x 和 y
1. 求出 x 除以 y 的余数 r
2. 如果 r 等于 0，y 就是两个数的最大公约数，计算成功结束
3. 将 y 作为新的 x，r 作为新的 y，回到步骤 1 继续

这里的每一行描述一两个简单操作，做计算时只需一行行地往下做。最后两行有点特殊：第 3 行说明满足一定条件就得到了结果；第 4 行说明要转回前面继续。

如果给定一对正整数 x 和 y，实施上面描述的计算过程，最后就能得到它们的最大公约数。进一步说，要执行这个计算过程，需要知道如何完成两个基本操作：求一个整数除以另一个整数的余数，判断一个整数是否为 0。

过程性知识

说明性和过程性的知识都非常重要，前者是人们有关客观事物的朴素认识的深入、积累和总结，而后者则是人们对自己的各方面实践活动的总结。

本书主要关注后一类知识（过程性知识）。其表现形式可能是一段自然语言说明，也可能采用某些特殊的格式（例如上面有关求最大公约数的描述，或者菜谱里有关一个菜肴的描述）。但是，这类知识重要性，主要不在于其描述的形式，而在于它所描述的（操作、计算）过程可以实施，最终产生出所期望的结果或者效果。

历史上人们早已发明了许多有价值的计算过程，但实施却很困难，因为只能由人按照特定的方法一步步计算，需要大量人力，也极端费时。有很多著名的例子，例如，开普勒花了近10年时间计算太阳系的行星轨道；二战中美国陆军部为计算大炮的装药和弹道，雇佣了成百的美国妇女参与计算弹道手册等。

近几十年来，由于人类发明了计算机这种自动化的计算机器，过程性知识的重要性大大提升了。以计算机作为过程自动化的基础，社会生产生活的方方面面都已经发生了巨大变化，而且，这一变化趋势还在继续，没有看到丝毫减退的迹象。今天在几乎任何领域中，一旦人们弄清了一件事可以怎么做，就会考虑如何借助于计算机去完成它。

1.1.2 计算和程序

环顾四周，计算机已经是今天人类使用最广泛的一种机器了。它不但以独立可见的形式出现在家庭、办公室和许多其他地方，还被嵌入在几乎所有现代化的设备、仪器、家用器具中。为什么计算机这么有用？为什么计算机到处都能用？实际上，计算机就是一种能自动完成计算的机器，要理解计算机，就需要理解计算。人们从小学（或者学龄前）就开始接触计算，学习基本计算规则。那么计算是什么？

计算是一类**按规则操作抽象符号（序列）的过程**。例如，最基本的整数算术就是操作算术表达式（由10个数字符号和几个表示运算的符号构成的序列）。人们使用一组基本算术规则，从需要计算的符号序列出发，逐步变换，最后得到计算结果。人类社会发展（如计量、分配、度量等）产生了计算的需求，逐渐发明了与计算有关的各种记法和过程。在计算机出现前，计算由人（通过头脑和手等）实施，可能借助一些简单工具（手指、纸笔、算盘、计算尺）。越来越烦琐的计算，需要花费非常多的人力和物力，也限制了很多工程技术的发展。今天，计算机已经把人从包含大量细节的烦琐计算中解放出来了。

要理解计算机和计算，也必须理解程序的概念，理解计算、程序和计算机之间的关系。讨论程序和编程（程序设计）就是本书的主题。

程序与计算

“程序”一词来自生活，通常指为完成某项事务而确定的一套既定活动方式或者活动安排。在表述上，程序应看作是对一系列动作的进行过程的描述。日常生活中也可以找到许多“程序”实例。例如，下面序列描述了一个学生早晨起床后的活动：

1. 起床
2. 刷牙
3. 洗脸
4. 吃早饭
5. 去教室上课

这是一个直线型序列，包含一系列简单活动（步骤），是最简单的程序形式。如果按顺序实施序列中描述的动作，最后的效果就是完成了一项事务或者工作。

另一个复杂一些的过程是完成一门数学课程的作业，可以描述为：

1. 完成准备工作（如准备笔、打开作业本、打开书籍等）
2. 找到要做的一道习题
3. 如果习题简单，完成后转到第 5 步
4. 如果习题复杂
   4.1 仔细审题、思考、查阅教科书和参考书等
   4.2 完成习题
5. 如果还有未完成的习题，转到第 2 步继续

显然，这个程序比前一个复杂些。最主要复杂性在于它不是一个平铺直叙的序列，其中有些步骤有条件，需要根据情况处理，还可能出现重复的动作或动作序列。

仔细探究这个实例，可以看到它还可能进一步细化，以便处理实际中可能遇到的各种复杂情况。例如步骤 4.1，其中的审题、思考和查阅书籍又可能交替进行，可能出现多次反复。还可能遇到各种异常情况，例如作业本用完了、笔出了故障、电话铃突然响了，或者到了上课或吃饭时间等。由此可见，要把一个“程序”描述完全并不容易。

现实生活中有许多这样的程序性活动，典型的如一个会议的议程、一场演出的节目单、运动会的程序册。总而言之，对一项事务、活动等的进展过程的细节描述就是一个“程序”。一个程序的描述通常有开始与结束，动作者（人或机器）根据程序执行一系列动作，到达程序结束位置时，整个工作完成。在一个程序描述中，总有一批预先假定的“基本动作”。例如，上面有关作业的描述中把“查阅参考书”看作基本动作。但如果参考书在图书馆，那么这个动作就需要进一步分解。程序的精细化（精化，或称功能分解）也是在本书后面有关计算机程序设计的讨论中特别强调的最本质的东西。

在计算中做的都是程序性的工作，通过一系列较为简单的操作完成。以多位数加法为例，需要数字对位等准备，然后使用基本数字加法规则，从低位到高位逐位求和并处理进位，直至做完所有数位，最后收集结果。有关计算过程可以描述如下：

1. 准备工作：将需要求和的两个数按位对齐
2. 把个位作为当前位，进位值记为 0
3. 如果两个数的当前位都有值
   3.1 算出两个数的当前位再加进位值之和以及新的进位值（0 或 1）
   3.2 把得到的和记入和数的当前位，转到第 5 步继续
4. 如果只有一个数的当前位有值，用 A 表示这个数
   4.1 算出 A 的当前位加进位值之和及新进位值
   4.2 把得到的和记入和数的当前位
   4.3 把更高一位作为当前位
   4.4 如果 A 的当前位仍有值，转到第 4.1 步继续
   4.5 如果进位值是 0，结束
   4.6 把 1 记入和数的当前位，结束
5. 将更高一位作为当前位
6. 如果两数的当前位都已经无值
   6.1 如果当时进位值是 0，结束
   6.2 把 1 记入和数的当前位，结束
7. 转回第 3 步继续

这里的基本执行方式是执行完一步后执行下一步，除非当前步骤中明确要求下一步做什么。有些步骤有条件，如果当前步骤的条件不成立，就直接转去执行下一步的操作。虽然多位数加法是人们最熟悉的计算，将其详细地严格写出还是有点烦琐。显然这里也有开始和结束，有特定的基本操作（一位十进制数的加法，判断一个数字是否 0）。乘法的计算过程更复杂，其中需要做逐位的乘法、结果的对位和加法计算等。

日常生活中的程序可以看作计算机程序的直观对应物，在一些情况下也会明确写出实际文本，例如一次会议程序或演出节目单。但两者之间还是有巨大差异，主要在于后者的极端严格性。日常生活中程序性活动的描述可以含糊笼统，实际执行也可以有变化调整，不一定完全按程序走。而计算机程序的描述则要求绝对严格，计算机作为执行主体，对程序的执行一丝不苟，一步步按程序中的条目行事，丝毫没有商量的余地。

计算机和程序

为了研究计算的理论，数学家阿兰·图灵在 1936 年提出了一种计算模型，后来被人们称作*图灵机*。这是一种非常简单的抽象计算机器（可以看作一种数学模型），其中有一个控制部件，指挥一个读写头在一条记录数据的带子上读写数据。著名的*图灵-丘奇论题*说，对于任何可以通过计算解决的问题，都可以设计一台图灵机解决。人们用这个论题定义*什么是计算*。图灵还证明了计算的局限性，通过实例证明了*不可计算问题*的存在性。

图灵还有另一个特别重要的贡献：他在图灵机模型的基础上设计了一台称为*通用图灵机*的特殊图灵机，并证明了通用图灵机能模拟任何一台图灵机的工作过程，也就是说，能完成任何可能的计算工作。图灵的做法是把具体图灵机表达为通用图灵机可以处理的编码表示形式，实际上就是程序。因此，通用图灵机是一种可编程的通用机器，可以看作今天的通用电子计算机的理论模型，因此图灵被人们称为“计算机之父”。

有关图灵机（和*图灵-丘奇论题*）和通用图灵机的工作非常重要。前者告诉我们，如果需要考虑设计和制造完成计算的机器，只需要考虑计算能力“等价于”图灵机模型的机器就足够了。后者告诉我们，不需要考虑如何去设计能完成千奇百怪的具体计算的设备（例如加法机、乘法机、文字编辑机、超级玛丽游戏机等），只需要设计和制造出一种设备，其功能等价于通用图灵机，就能解决所有的计算问题了。

现实中的计算机是 20 世纪 40 年代人们发明的一种自动机器，基于电子技术，能完成各种计算工作。从 20 世纪 30 年代末开始，各国的研究者们开始基于电子技术开发出一些计算设备。早期的这类设备只能做一件专门计算工作，是功能固定的计算设备。例如 1942 年完成的 ABC 机器（Atanasoff-Berry Computer）专门求线性方程组的数值解，图灵领导开发的 bombe（意为密码破解）机器专门破解德国 Enigma 密码机生成的密码。

1946 年在美国宾州大学开始运行的 ENIAC 是第一台得到较多实际使用的可编程计算机，只需要给它换一个“程序”，它就能做另一种计算工作。但是，这台机器的程序由一些外部连线和开关设置组成，要想换一个程序，需要重新连接许多线路，改动很多开关。这件事通常需要许多人工作几天时间，非常麻烦。

著名数学家冯·诺依曼考察了 ENIAC 机器之后写出了一份报告，提出了采用*程序存储*方式的计算机的逻辑设计，为现代计算机的发展指明了正确方向。*程序存储计算机*的特点就是把需要执行的程序编码，像数据一样存入计算机的存储器，然后让计算机的执行部件自动提取程序的内容，执行相应操作。这样，一方面计算机能摆脱外部拖累，用自己的速度快速执行程序。

另一方面，给计算机换一个程序，就像是给它输入一组数据一样，非常方便。具有这种结构的计算机被称为冯·诺依曼计算机。1948 年英国曼切斯特大学研究者开发出 Mark I 计算机，通常被认为是第一台实际投入使用的程序存储计算机。

其实，所谓的冯·诺依曼计算机，也就是图灵理论中的通用图灵机的一种具体体现形式，是一台通用的计算设备。而完成具体计算的计算机程序，对应于图灵的通用图灵机理论中的具体图灵机的编码表示形式。也就是说，人们在设计和实现计算设备方面的多年实际探索，最终再次证明了图灵理论的重要意义。

计算机的基本原理

计算机能执行一组基本操作，每个操作完成一个简单动作，例如做一次整数运算、把一个数从这里搬到那里、比较两个数的大小等。计算机提供了一套指令，每种指令对应计算机的一个基本动作。最基本的计算机程序，就是一个这种指令的序列。

计算机的基本原理很简单，就是能自动地按照程序要求一步步工作。其工作方式如图 1.1 所示，这是一个循环，循环中不断地交替执行两个动作：第一个动作是取得程序里的一条指令，第二个动作是执行该指令要求的操作。完成一次基本循环后进入下一个循环，再次取得下一条应该执行的指令并执行它。周而复始，直至遇到一条明确要求它停止工作的指令。这个图表现了计算机运行的微观情况。

要指挥计算机工作，最基本的方式就是写出一个程序，把这个程序提供给计算机，命令计算机去执行它。此后计算机就会按照程序的描述，一丝不苟地执行其中的指令，直至整个程序执行结束。因此，从宏观上看到的计算机工作的情况如图 1.2 所示：获得一个程序，执行它，可能给出得到的结果；然后取得人们提供的下一个程序，执行并给出结果。同样是循环往复。人们常把计算机执行程序的过程简单说成是程序的执行过程。

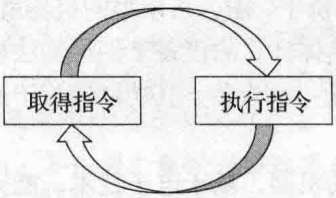

图 1.1 计算机的基本执行循环

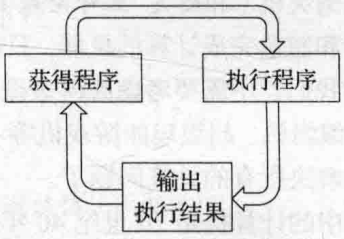

图 1.2 计算机的程序执行循环

计算机是一种通用计算机器，给了它一个或一组程序后，它就变成了一个处理某种专门问题，完成特殊工作的专用机器。例如，给它（在其上执行）一个文字处理程序，它就变成了一台文字处理机器；给它一个游戏程序，它就变成了一台游戏机器。通过技术，还可以同时在一台计算机上运行多个程序，使人可以把它当作多部不同的机器，同时使用。

这种通用性与专用性的统一非常重要：从一方面看，一种或不多几种计算机可以通过现代化工业的方式大量生产，使工程师们可以专心研究这几种设备的各方面改进，提高性能、降低成本、缩小体积等；另一方面，通过运行不同的程序，一台计算机可以在不同时候处理不同问题，甚至同时处理许多不同问题。这些就是计算机之威力的根源。这种通用性和专用性的统一，为信息科学技术领域的大发展奠定了坚实的基础。

今天，计算机的发展及其在各领域的广泛应用，对人类社会各方面的深刻影响已经是人人皆知的事实了。计算机之所以能产生这样大的影响，其原因不仅在于人们发明并大量制造了这

种令人敬畏的奇妙机器，更重要的是人们为计算机开发出了数量宏大、五彩缤纷、能指挥计算机完成各种简单或复杂工作的程序。目前正在使用的计算机并没有多少种，正是数量繁多、功能丰富多彩的程序给了计算机无穷无尽的“生命力”。

作为通用的机器，计算机有应用于各种领域、解决各种问题的潜力。但是，要真正将其应用于任何具体问题，都需要有完成这种具体工作的程序，需要有人把这种程序开发出来。人们把描述（编制、构造、开发）程序的工作称为*程序设计*或者*编程*，这种工作的产品就是程序。由于计算机的本质特征，从计算机诞生之初就有了程序设计工作。今天，编程已经变成了一个需要很多人去做的实际工作，而且还有更多的人多多少少需要做一些编程工作。学习计算和编程不仅可以成为一些人的职业基础，也是理解今天社会生产和生活方方面面的新特征，理解所有科学技术和社会发展的一条重要线索。

1.1.3 编程语言

要指挥计算机做某件工作，就要给它一个程序，其中说明计算机在工作过程中应该**怎样做**，给出完成所需工作的程序性活动的一步步细节，描述计算机执行中需要做的动作及其执行顺序。为了描述这种过程，需要有一种适当的描述方式。一套系统的描述方式就形成了一种*语言*。在这里需要的是一种人能使用、计算机也能处理的符号描述形式，这样一套描述形式就是一种*编程语言*。这是一类人造的语言。编程语言也常被称为*程序设计语言*，或简称为*程序语言*，本书中也经常简单将其说成*语言*。

有人可能说：小学生就学了数学，数学的一个基本部分是计算，小学生已经会用数学方式（或说用数学语言）描述计算过程，编程语言还有什么特殊之处吗？确实有！编程语言的突出特点就在于不仅人能懂得和使用它，计算机也能“懂得”它，能按用编程语言描述的程序去行动，完成所需要的计算工作。因此，程序语言既是人描述计算的工具，也是人与计算机交流信息的基本媒介。人通过用编程语言写程序的方式指挥计算机完成各种具体工作。当然，编程语言也是人与人交流有关计算的想法和过程的工具，一个人写的程序可能被其本人后来阅读，或者被其他人阅读。这方面的作用也影响着编程语言的发展。

机器语言和汇编语言

在计算机内部，一切信息都以二进制编码的形式存在。需要存入计算机，要求计算机执行的程序也不例外。计算机都规定了一套描述其指令的二进制编码形式，这种描述形式称为*机器语言*，用机器语言写的程序称为*机器语言程序*。计算机能直接“理解”并执行这种程序。下面是一台假想计算机上的一个机器语言程序，也就是一个指令的系列：

```
00000001000000001000    -- 将单元1000的数据装入寄存器0
00000001000100001010    -- 将单元1010的数据装入寄存器1
00000101000000000001    -- 将寄存器1的数据乘到寄存器0原有数据上
00000001000100001100    -- 将单元1100的数据装入寄存器1
00000100000000000001    -- 将寄存器1的数据加到寄存器0原有数据上
00000010000000001110    -- 将寄存器0里的数据存入单元1110
```

这里描述的程序包含 6 条指令，要求计算机做的事情就是计算算术表达式 $a \times b + c$ 的值。为便于理解，这里用 a、b、c 代表计算机里的三个存储单元，它们的地址分别为 1000、1010 和 1100。这个程序描述了完成这一计算，最后将结果存入单元 1110 的计算过程。上面提到的寄存器是一类存储数据的部件，计算机硬件可以直接操作其中数据，保存在其他地方（例如内存单元）的数据需要先装入寄存器，然后才能在计算中使用。

计算机诞生之初，人们只能用机器语言写程序。由上面的简单例子可见，对于人的使用而言，二进制编码形式的机器语言非常不方便：用它写程序非常困难，工作效率极低，写出的程序难以理解，正确性很难保证，发现程序有错也很难辨认和改正。

复杂的程序可能包含成百万、成千万条，甚至更多条指令，其中的执行流程错综复杂，理解一个二进制机器语言描述的复杂程序到底做了什么，很容易变成人力所不能及的事情。为了缓解这个问题，人们很快开发出符号形式的、相对更容易使用的*汇编语言*。用汇编语言写的程序需要通过专门软件（称为*汇编系统*）加工，翻译成机器语言后送给计算机执行。下面是用某种假想的汇编语言写的程序，它完成与上面程序同样的工作：

```
load 0 a  -- 将单元a的数据装入寄存器0
load 1 b  -- 将单元b的数据装入寄存器1
mult 0 1  -- 将寄存器1的数据乘到寄存器0原有数据上
load 1 c  -- 将单元c的数据装入寄存器1
add  0 1  -- 将寄存器1的数据加到寄存器0原有数据上
save 0 d  -- 将寄存器0里的数据存入单元d
```

汇编语言的每条指令对应一条机器语言指令，其中用助记的符号名表示操作和存储单元。这样，每条指令的意义都更容易理解和把握，理解整个程序也容易了许多。但是，汇编语言没有结构，一个程序是基本指令的一个长序列，是一盘散沙。用汇编语言编写复杂程序仍然很困难，难以理解，其中的错误也很难被定位和改正。

高级编程语言

1954 年诞生了第一个*高级程序语言* Fortran，程序设计进入了一个新时代。Fortran 采用完全符号化的描述形式，用类似数学表达式的形式描述数据和计算。语言中提供了有类型的变量，作为计算机存储单元的抽象。它还提供了一些高级流程控制机制，如循环和子程序等。这些高级机制使编程者摆脱了许多具体的计算机硬件细节，可以把复杂的程序分解为一些较小的较容易把握的部分，方便了复杂程序的书写。写出的程序更容易阅读，发现错误后也更容易辨认和改正。Fortran 语言诞生后受到广泛欢迎。

从 Fortran 语言诞生至今，人们提出的语言已经有数千种，其中大部分是试验性语言，只有少数语言得到广泛使用。随着时代的发展，今天绝大部分程序都是用高级语言写的，人们也已习惯于用*编程语言*这一术语特指各种高级程序语言了。用高级语言描述前面的程序只需要一行。例如，下面是用本书中将要使用的 Python 语言写出的完成同样计算的程序：

```
d = a * b + c
```

这一行程序要求计算机算出 = 符号右边表达式的值，最后用 d 记录计算的结果。这种表示方式接近人们熟悉的数学形式，明显更容易阅读和理解。

与机器语言和汇编语言相比，高级语言的形式人更习惯，更容易使用，这也使更多的人愿意参与到程序设计活动中。用高级语言写程序，工作效率更高，使人能开发出更多应用系统，这些又反过来推动了计算机应用的发展，进而推动了计算机工业的大发展。可以说，高级编程语言的诞生和发展，对计算机领域的发展起到决定性作用。

当然，计算机也不能直接执行高级语言描述的程序。人们在设计好一个高级语言后，还需要开发一套实现该语言的软件，这种软件被称作*高级语言系统*，也常说成是这一高级语言的*实现*。在研究开发各种高级语言的过程中，人们深入研究了实现语言的技术。高级语言的基本实现方式有两种，分别称为*编译*和*解释*：