《Python 超能学习手册》

[美] 本·福达(Ben Forta) 什穆尔·福达(Shmuel Forta) 著

周子衿 译（https://bookzhou.com）

中文试读版 1-6 章，仅供参考，

更多精彩内容，请购买正式版：京东>>

配套资源和试读下载： ys168 网盘>> 百度网盘>>

访问中文版主页，获取最新资讯

清华大学出版社

北 京

PYTHON

超能学习手册

[美] 本·福达 (Ben Forta)　什穆尔·福达 (Shmuel Forta)◎著　周子衿◎译

清華大學出版社
北京

内 容 简 介

本书是作者 5 年 Python 编程教学成果的结晶，采用了布鲁姆教育目标来精心设计全书的结构和内容，同时还结合读者的认知水平和需求，在注重知识性的同时深度融入了趣味性，从做游戏的角度来激发读者学习编程的兴趣，聚焦于编程技能以及逻辑、计算和创新思维的培养与提升。

全书共 3 个部分 24 章，从积极正面的游戏（比如文字冒险类和图形类游戏）入手，以快速、有趣和目标为导向，着眼于帮助读者通过学习 Python 编程来掌握高效率的底层思维框架，从而懂得如何规划、解决问题、沟通，如何培养逻辑思维、同理心、观察力、耐心、适应力、毅力和创造力等。此外，穿插于全书的术语、技巧提示、补充说明及编程挑战等，可以帮助读者进一步理解和应用各个知识点，也是本书很重要的特色之一。

本书适合作为 Python 的入门教材，尤其适合不具备任何编程经验的读者。

北京市版权局著作权版权合同登记号　图字：01-2022-1565

图书在版编目(CIP)数据

Python超能学习手册 / (美) 本·福达 (Ben Forta)，(美) 什穆尔·福达 (Shmuel Forta) 著；周子衿译. —北京：清华大学出版社， 2022.6

书名原文：Captain Code: Unleash Your Coding Superpower with Python

ISBN 978-7-302-60812-7

Ⅰ. ①P…　Ⅱ. ①本… ②什… ③周…　Ⅲ. ①软件工具—程序设计　Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2022)第080024号

责任编辑：文开琪
封面设计：李　坤
责任校对：周剑云
责任印制：杨　艳
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者：小森印刷霸州有限公司
经　　销：全国新华书店
开　　本：170mm×230mm　　**印　　张**：21.25　　**字　　数**：456千字
（附赠全彩不干胶贴纸）
版　　次：2022年8月第1版　　**印　　次**：2022年8月第1次印刷
定　　价：128.00元

产品编号：097098-01

推荐序 1

最近几年少儿编程以及青少年编程整个行业算是经历了一个跌荡起伏的过程，从最初炙手火热的资本赛道，到现在的风平浪静，散的多半儿是热钱，留下来的一如既往地踌躇满志和充满热情。

其实这是件好事儿，因为教育从来不是，也不应该是一个快打快冲的领域。它是一个需要不断耕耘，长期积累，持续发挥价值的领域。毕竟，我们面对的是孩子，我们需要的是及时播下种子，准时疏枝散叶和施好肥，然后静待花开！

借这篇序也好好捋了一下自己从这一路以来的变化，第一个阶段，作为一名专业的程序员，面对的主要是业务与代码；第二个阶段，作为一名创业者，面对的主要是产品和市场；第三个阶段，作为一名编程教育推广者和老师，面对的更多是孩子和家长，还有其他老师。

在面对孩子、家长和老师时，我思考和回答得最多的是这样几个问题：“孩子为什么要学编程？怎样学才能学得好？老师如何教才能更有效？”

在阅读这本《Python 超能学习手册》时，我有这样一种感觉，作者也把自己对这几个问题的理解完全融入到书中去了。

再遇 Ben 大叔

作为一名曾经的 Flex 和 ColdFusion 开发者，我看过作者（Ben Forta） 的书，听过他的演讲，也是他的粉丝。多年以后，我全身心投入青少年编程的推广与教育，一直期待着有一本适合青少年学习的 Python 编程书，不仅要教青少年如何编程，更要教他们如何思考、如何分析和解决问题！当然，有趣更是必不可少的！

这个时候，Ben 大叔的《Python 超能学习手册》出现了，这本书完全符合甚至超出了我和学生们的期待！

两位作者在前言中谈到了他们对孩子学编程的看法以及为什么要写这本书，篇幅不长，却很好地回答了我在前面所提到的几个经常在思考的问题。

为什么要学编程

作者讲到了编程能给你带来一种强大的“超能力”。也和朋友聊到过一个相关的话题：“编程给自己带来的最大影响是什么？”虽然每个人对具体带来的影响有不同的描述，但比较有共识的一点是，在这个数字化的互联网时代，正是有了对编程的理解，我们才可以从更抽象的底层逻辑去观察、认识和理解这个世界，从这一点来说，我们确实拥有了“超能力”，我们是幸运的。

作为一名编程老师，我也很清楚，现在的孩子学习编程，并不是都要成为程序员，编程要作为一种通识教育进入课堂。学编程，并不是会用一种语言去编程，更是通过编程去学习，编程能给孩子带来完全不同的思考方式，从这个角度来说，每个人都应该学习编程，孩子更是如此。

就像作者描述的那样，人人都应该学习编程，就如同每个人都应该学习绘画、学习乐器一样，因为这些都是创造性的工作，意味着是在真正地创造，在工作中发挥创造力会带来不同的思考方式，对于这种理解，我简直不能赞同更多！

能学会编程吗

“任何一门学科，在孩子的任何发展阶段，都能以某种智识上诚实的方式，有效地教授给任何孩子。”杰罗姆·布鲁纳曾经说过这句话。

好吧，对于家长来说，观念已经在开始转变，哪怕编程并不属于一个与升学考试直接挂钩的学科（至少目前是这样），大部分家长觉得还是有必要让孩子学点编程，但问题又出来了：“孩子能学会编程吗？”毕竟在孩子父母读书那时候，大多是从大学才开始接触编程的，而且还很头痛，感觉编程很难，大人都不是那块料，孩子能学会吗？

做了编程教育很长一段时间后，我也越来越认同布鲁纳的这句话。作为一个编程教育工作者，首先并不是要回答“孩子能学会吗？”这样的问题，而是应该回答：“我们以一种什么样诚实的方式，有效地去教会孩子学习编程？”

要回答好这个问题很不容易。目前孩子学习编程，大致上可以分为两个大的阶段，图形化编程和代码编程。图形化编程基本上是基于麻省理工学院 Scratch 来学习的，而从图形化编程过渡到代码编程，Python 语言以它简洁、接近自然语言的语法结构，自然成为首选。

然而，市面上的 Python 书很多，大多还是写给专业程序员的，写给青少年教他们学会编程的很少，由真正懂编程、懂教育并熟悉孩子的人来写的就更少了，本•福达（Ben Forta）作为 Adobe 的资深教育主管，已经帮助超过 100 万人学会编程；而什穆尔 • 福达（Shmuel Forta）更是拥有程序员、工程师、创客和教师多重身份，而且他在高中教授孩子们学习编程长达 5 年，由他们俩来合写这样的一本书再合适不过了。

本书的组织结构更像是一份桌游指南，带着大家去玩转 Python 编程，从最开始的小游戏开始，让大家先玩起来；再逐步过渡到复杂一点的冒险类文字类游戏，让大家认真玩；最后再到更复杂的带界面的赛车游戏，带大家好好玩，玩法难度逐步提升，让大家在玩中学习和成长。

全书的内容递进也是基于布鲁姆教育目标精心做了设计，所有章节都是围绕着一些要解决的游戏问题不断展开，提出问题 → 尝试捣鼓 → 解决问题 → 再提出问题 → 优化问题，最后留下挑战与思考，把自主权又给到学生，甚至还讲到了复用、测试、重构这些最佳实践，非常专业，也不失趣味性！

如何更好地教呢

如果是正在教孩子学习 Python 编程的老师，本书也一定能帮上你，书中的内容结构也很适合分解或组合成课程，如果想将本书的内容用到课堂教学中，Ben 大叔还贴心了为编程老师准备了教学资源，为每个项目都提供了使用说明和讨论要点。我在读完本书后，我也第一时间将本书的内容和思路分享给了一些老师，并且应用到了自己的教学中，效果非常好。

当然，教和学都不一件简单的事，在接下来的时间里，我也计划以这本书为基础，作为起点，结合自己在青少年编程教学中的一点经验，多做一些探索和分享，也希望有更多想在这方面做一些工作的程序员、编程老师一起加入进来，让更多的孩子享受编程的乐趣！

期待这本书可以为正在学习编程的孩子和为正在教孩子学习编程的老师，打开一扇窗，释放自己在编程方面的超能力，领略编程带来的无限美好风光！

是不是等不及了？赶快来释放你的超能力吧！

大圣不是圣（陈显军）

2022 年 8 月于成都

推荐序 2

通过制作游戏，让编程初学者对优雅的程序设计有一个整体上的认识，《Python 超能学习手册》就是这样的一本书。

日常碎片化的语言充斥着整个社会，也分离了个体对自我的认识。如果能依托于书中使用的 Python 语言，回归到计算思维的原点，我们将有可能从整体上、系统上感知、认识和理解自我、社会、自然乃至整个宇宙。

作者选取了我们熟悉的经典游戏：掷骰子、剪刀石头布（包剪锤）、猜数字、文字类冒险游戏以及赛车等，规则从简单逐渐变得复杂，要完成整个游戏的制作，需要涉及关键的基础概念和最佳实践，从整体上正确认识程序设计，具备如此 MVP（最小可行性）特性的认知骨架特别适合希望一开始就做对的初学者。

作者提到：“学习编程是不够的，相反，我们要帮助你学习如何像程序员一样思考，像他们那样思考和分析问题，规划和快速迭代，最后得到优雅的解决方案。”此话于我心有戚戚焉，借此机会与广大正在从事编程教育的老师们共勉，同时也邀请各位有志于深耕编程教育的教育工作者积极参与教学研讨，共同为下一代的编程教育贡献我们的群体智慧。

李端

四川质量发展研究院区块链中心

前　言

请以最低沉的嗓音，气音实际上也可以，缓缓地念出下面这段话：

“传说中，有一群超人。他们拥有超能力，散居在全球各地。他们有激活潜能、唤醒僵尸亡灵的能力。他们能用不同的语言发布指令，可以让或近或远的机器服从他们的意志，听从他们的命令。这些人优秀，强大，他们是传说中的……程序员！”

＜咳＞不好意思！

好吧，我得承认自己刚才的表演可能有些用力过猛。不过，话又说回来，咱们这些程序员啊，真的个个都算得上是高手，是超人，是美国队长那样的超人。我们都明白，我们是程序员，并且认为自己又酷又厉害（这可真不是在吹牛）。事实上，对大多数程序员而言，我们和《哈利·波特》中的甘道夫、《蜘蛛侠》里面的布鲁斯·韦恩、《星球大战》中的卢克·天行者、《冰雪奇缘》里面的女王艾尔莎、《钢铁侠》中的托尼·史塔克、神奇女侠或死侍[①]。最相似的地方，莫过于我们个个都有超能力，能通过编程来指挥机器，让它们为我们人类服务。

我知道，这么说可能显得有些（一丢丢）夸张。但说句老实话，编程就是能够让我们拥有这么强大的能力。也就是说，“超能力”是很容易通过学习编程来获得的。

本书带着大家一起学习编程，将帮助大家掌握这些技能。此外，更重要的是，我们想要帮助大家通过正确、高效的学习方式来成为美国队长那样的超人程序员。

为什么要学习编程

在此之前，首先请大家考虑这个问题：“为什么要学编程？”如果问问身边的人或者上网一搜，我们会得到各种各样的回答。

最常见的回答是，编程是一种面向未来的技能，非常重要。也就是说，如果我们

① 译注：漫威旗下的反派英雄，拥有远超于金刚狼的治愈能力和一个可以让自己实现瞬间移动的腰带。

掌握了编程，未来就更容易找到一份好的工作。虽然这种说法可能有些道理，但是说真的，我并不认为这是学习编程最好的理由。为什么我会这么说呢？

首先，并不是每个人都需要成为一名程序员。这是不可能的，就像不可能每个人都是医生、厨师、教师、飞行员，或者又都是穿过下水管道拯救公主的马里奥一样，懂我的意思了吧？为了维持社会的正常运转，需要有不同的人去做不同的事情，所以说呢，虽然很遗憾，但是，我们未来真的不需要 80 亿人个个都是程序员。

此外，技术领域（包括编程）的发展日新月异，程序员现在的工作和 10 年前的工作不同了，而且，下一个 10 年的变化更大。因此，大家现在学的并不一定是将来成为程序员后会用到的。优秀的程序员永远不会停止学习、提升或拓展自己的技能。基于本书锁定的是基础知识，这些知识始终重要且实用，只不过具体的细节经常会随着应用场景的不同而变化。再说了，编程这个技能并不是学了之后立刻就能上手的，如果有人真的这么以为，那就只能说他是大错特错了。

最重要的是，如果对编程有兴趣完全是出于对未来职业的考虑，可能就会觉得它是工作而不是乐趣。如果没有兴趣，就不会有热爱，就不可能坚持下去，而且肯定缺乏沉迷于编程的动力。这样就太可惜了，因为编程这件事儿，真的很好玩儿。

我并不是说编程领域没有好的工作。肯定是有的，而且未来几十年内会有许多好的工作。但坦白地说，对未来职业的考虑不应该是大家选择成为程序员唯一的原因。

说一千，道一万，到底为什么要学习编程呢？每个人都应该学吗？我认为，即使不打算以编程为职业，也应该学习编程。我相信这一点，如同我相信每个人都应该学习绘画和素描，学习演奏乐器，学习烹饪，学习拍照和拍视频，等等。这些都是创造性的工作，意味着是在真正创造事物，而创造会让人充满成就感和满足感。诚然，花几个小时在手机上浏览别人的创作很有意思，但相比自己的个人作品可以供别人消费和使用时所获得的快乐和满足感，前者完全不值一提。

除此之外，在学习编程的过程中，还可以发展出编程之外的各种不可预期的技能和品质，其中包括规划能力、解决问题的能力、沟通能力、逻辑思维、同理心、对细节的关注、耐心、适应能力、毅力和创造能力。

实际上，对未来的工作和职业生涯而言，这些能力特别重要，尤其是创造能力和创造性解决问题的能力。所以，没错，即使不打算成为程序员，编程也确实可以为大家未来的职业生涯提供帮助。

如何学习编程

现在，我们确定了学习编程是大势所趋，是刚需。但从哪里开始学呢？根据我的经验，许多书籍、视频和课程都过于关注编程的机制，比如语法和使用特定语言元素的具体细节。种种细枝末节让人感觉像是填鸭式教学，并不是在鼓励大家动手尝试捣鼓代码，很无聊。以这样的书作为教材，就好比花几个小时学习字典里的单词和语法，然后通过模仿来使用这些单词和语法，完全没有机会带入自己的话语和声音。这太离谱了，对吧？然而，大多数人都是以这种方式第一次接触编程的。

我从事编程教学已经有很多年的历史了。事实上，我已经帮助 100 多万人成为了程序员，包括许多年轻人。我知道如何帮助大家培养这些技能，因为我就是以这种方式自学成才的。我的教学特点是快速、有趣但同时又以目标与结果为导向，强调成效，力求帮助学生融会贯通，从想要知道、参与做到、进而得到以及最后精通，从头到尾真正掌握编程这门手艺。

以上就是我写这本书的原因，即帮助大家学习编程，并且更重要的是，帮助大家充分释放自己在编程方面的超能力，让大家变身成为擅长于思考和行动的高效率程序员。

本书内容

本书不会只专注于讲解如何编程，那样的书多得是，其中有一些甚至还真的不错。

但是，仅仅学会编程是远远不够的。本书还将帮助大家学会像程序员一样思考，像程序员一样分析问题，像程序员一样制订计划，像程序员一样增量迭代，像程序员一样设计优雅的解决方案……事实上，在完成本书的学习后，你将变成（此处应响起击鼓声）一名让人刮目相看的超人程序员！

为了实现这个目的，本书与其他书籍迥然相异。本书的创作动机是帮助大家在快速成为一名超能程序员的同时深度沉浸于编程的乐趣之中。

全书一共 3 个部分 24 章，各个部分相辅相成，具体如下所述。

第 I 部分“Python 玩起来：小游戏，大欢乐”

这部分涵盖一些基础知识（也有一些不那么基础的知识）。学完本部分的内容后，大家将掌握所有主要的编程概念，具备编写任何应用程序都需要的基础知识。

本部分包含 10 章内容，具体如下所述。

- 第 1 章的主要内容是安装和运行，包括如何帮助大家安装好必要的软件并为使用软件做好准备。
- 第 2 章到第 7 章介绍如何创建各种小游戏和其他程序。每章都会讲解新的编程概念，并立即在新的项目中应用这些概念。每一章中，都有机会调整、修改代码并让代码成为你独有的“资产”。
- 接下来，第 8 章将创建一个更复杂的游戏，并在第 9 章中完成这个游戏。
- 第 10 章讨论各种可供自行尝试的点子，以此来作为第 I 部分的收尾。

这样设计章节是考虑到各个主题需要相得益彰。在某一章中新开发的技能随即可以在后续章节中派上用场。同时，这些章节也设计得短小精悍，大部分章节都只涉及一些小型的独立程序。

学习这部分内容时，请慢慢来。请自行尝试每节课和每个案例，用玩儿的心态放开胆子去修改、调整和捣鼓代码。请随心所欲地对书中提供的代码进行修改，看看程序会有哪些变化。因为随时可以撤销操作，所以完全不必担心这样玩儿代码会破坏程序。在第 I 部分中学到的东西会是大家以后最常用到的，无论是在学习本书时还是在今后的任何项目中。

第 II 部分“Python 认真玩：文字冒险类游戏”

完成第 I 部分的学习后，我们离开浅水区，来到深水区。在本部分中，将创建一个更大型也更有趣的游戏。一开始，先着手构建框架，然后逐步向其中增加功能。要创建一个什么样的游戏呢？答案是一个很酷的古风文字冒险类游戏，它能给你的家人和朋友留下深刻的印象，而且可以做得相当复杂，足以把硬核玩家给难哭。

本部分共有 8 章内容。

- 第 11 章涉及正式开始制作游戏前的准备工作。
- 第 12 章将开始创建游戏，并逐步添加功能和复杂性，一直延续到第 17 章。
- 第 18 章将给出各种改进游戏的点子。

与第 I 部分不同，在本部分中，我们希望大家踏上自己的冒险之旅，讲述自己的故事，编写自己的游戏。我们会帮助大家启航，展示要用到的技术。大家可以自由地使用书中的代码。我们甚至会介绍怎样下载其他故事的开头，但随后我们会把一切交给大家，让大家创造出自己的游戏大作。

第 III 部分“Python 好好玩：赛车竞速类游戏”

和第 II 部分相似，在这部分中，我们将循序渐进地创建一个更大型的游戏。这次要创建一个图形游戏，有图像、运动、用户交互和得分等。

本部分共有 6 章。

- 第 19 章将引入并介绍如何使用游戏引擎以及解释什么是游戏引擎。
- 在第 20 章到第 23 章中，我们将构建一个完整的、可玩的游戏。书中会提供可用的图片（是的，我们就是这么体贴）。
- 第 24 章总结了许多可以添加到游戏中的有趣的点子。

在这个部分，可复制的代码会变少（因为学到这里时，大家都已经是专家了）。同时，要讲解如何改动和更新代码来得到自己想要的效果。

哦，对了，还要提一下第 25 章。是的，我们就是这么宠溺大家，因此额外添加了第 25 章。在本书的网页中可以找到。访问前言末尾的链接或扫描二维码即可访问。

特别说明

本书包含许多下面这样带有图标的方框。它们分别有以下的含义。

新术语

标题　我们不仅要讲解如何编程，还要帮助大家像一名真正的程序员那样说行话。一旦接触到新的单词或短语，我们就会在这样的文本框中进行释义。

小贴士

标题　程序员总是喜欢想方设法寻求效率和节省时间。这种文本框中包含一些捷径和节约时间的点子或者能让编程变得更简单的小知识。

补充说明

这种文本框中包含数不胜数的实用注释，还有一些不那么实用但很有趣的注释。

挑战

这个方框一出现，就意味着我们即将开始额外布置任务（加分项）。不过，这可不是什么家庭作业，而是一些有趣的任务。如前所述，在这本书中，希望大家不是通过阅读而是通过实践来学习编程。我们将帮助大家创建许多程序，有些是简单的小程序，有些是更复杂的程序，其中许多程序后面都带有一个供大家自行钻研和解决的挑战练习。不要担心，若是遇到了困难，则随时可以查看本书的在线提示和解决方案。

最后，请留意书中显示的二维码，比如下面这个。扫描它们即可访问本书英文版配套网站的页面，其中包含着实用的链接、可下载的代码、挑战练习的解决方案等。

英文版配套网站

获得帮助

在阅读本书的过程中，偶尔可能需要一些帮助。碰到这种情况时，可以采取下面这几种方式。

- 在浏览器的地址栏，输入 https://forta.com/books/0137653573，访问本书英文版网站。也可以扫描下面的二维码。网站包含着针对英文原书的很多提示、解决方案和更多拓展内容。也可以通过小助手，加入 Python 社群。

扫码添加小助手

- 当然，也可以像大多数程序员那样，用浏览器搜索。输入具体的问题，例如，完整的编程语言名称，就能找到答案。
- 也可以随时联系我们，在 https://forta.com/ 和前面的提示中，可以找到我们的联系方式。

好了，欢迎来到 Python 编程世界！请翻过这一页，让我们正式开始吧！

致　谢

本·福达（Ben Forta）

我从事写作和出版有 25 年的历史了，真是让人难以置信！ 1996 年，我在培生出版了我的处女作。自那以后，我们的合作成果是累计出版了 40 多部书。我们一起教育并激励

着世界各地的众多开发人员。回顾这四分之一个世纪，我发自内心地感谢培生这些年来的奉献和支持。《Python 超能学习手册》是我为青少年群体所写的第一本书，因此，我要特别感谢培生能信任我的眼光，为我们作者赋予了充分的自由，让我们可以按照自己的想法来进行创作。

特别感谢指导我们从想法到成果的金·斯宾塞利（Kim Spencely），再次感谢克里斯·扎恩（Chris Zahn）为我们提供了开发方面的支持。

在过去的几年，我有幸在密歇根州南菲尔德的法伯希伯来高中给学生们上机器人课程。 COVID 来袭之后，我们转为线上授课，我利用这个机会开始教学生 Python，希望借此来提高他们的编程技能。作为一种尝试，学生们在课堂上的表现激励我写下了这本书。

千言万语总结为一句话，感谢法伯希伯来高中给我一个可以启发学生们的机会，也感谢学生们帮助我学会了怎样更好地教书育人。

感谢我的儿子伊莱（Eli），他是一位才华横溢的设计师和崭露头角的建筑师，他为本书提供了图片资源。

最后，感谢我的儿子什穆尔（Shmuel），他是一位出色的工程师和充满热情的教育工作者，也是我这本书的合著者。回顾过去，我有一半的书都是和合著者一起完成的。不过说实话，我更喜欢独立写作。但这次合作是个例外。什穆尔（Shmuel）的经验帮助打磨了这本书，每一页都能看出他独到的见解，我们父子俩的合作让我感到快乐和自豪。“谢谢你，什穆尔（Shmuel）！”

什穆尔·福达（Shmuel Forta）

对我而言，这本书的写作是一段令人振奋，也让人戒骄戒躁的经历。我很荣幸能有机会与世界各地的读者一起分享我五年以来给七八年级学生上课的教学心得。

特别感谢培生。依托于他们的信任，我们作者能够按照自己的构想来创作这本书。还要感谢金·斯宾塞利（Kim Spencely）和克里斯·扎恩（Chris Zahn），没有他们，就不会有这本书，也不会编辑和出版后分享给大家。

还要感谢我的妻子查娜·米娜（Chana Mina），感谢你的所作所为。谢谢你平时对我的包容（这本身就是一项了不起的壮举）。我知道，写作占据了我大量的时间，但你始终如一地支持着我。没有你的支持（以及你对这本书提出的建议），我真不知道自己会变成什么样子。感恩一路上有你。

还要感谢我的家人。感谢我的母亲、兄弟和岳父岳母，他们帮助和支持我度过了一切难关。他们的鼓励和力量，帮助我完成了这本书的写作。

最后特别感谢（虽然这个词远远不足以表达我的感激之情）我的父亲兼合著者。在我还不到 10 岁的时候，我的父亲就开始教我用 Visual Basic 写代码。在我最美好的童年回忆中，包括我们父子俩挤在老式的笔记本电脑前，他耐心地引导我找出代码中的错误（比如， 一个等号 = 而不是两个 ==）。我还记得，小时候的我欣喜若狂地跑下楼，向他展示自己做的猜数字游戏、那个简陋的只有下拉菜单的计算器以及自己动手做的第一个图形游戏（一个太空射击游戏），当其时，我感受到的是一种纯粹的快乐。父亲对我的作品表现出极大的自豪感和爱，使得小小的我创作热情高涨。我遇到过许多拥有各种技能的天才程序员，但很少有人能像我们父子俩一样打心眼儿里真正热爱着编程。而我对编程的热爱，来自我的父亲并且受到他的感染。在这里，我想对他说：“谢谢您与我合作写成这本书，但更重要的是，谢谢您把我培养成为今天的我，并与我分享您对编程的热爱，对此，我将永远心存感激。”

简明目录

详细目录

第 I 部分

Python 玩起来：小游戏，大欢乐

第 1 章

预备知识

嗨，欢迎大家来到 Python 的世界！我热爱编程，相信大家在看完这本书之后，也会跟我一样爱上编程的。我将帮助大家以我的方式——也就是实践——来学习编程。没有长篇大论，没有大量的说明，也没有复杂、枯燥的讲解。每一章都提供了动手实践的机会，让大家在实践的过程中学习如何编程。

不过这一章例外，抱歉啦 :-(。在开始充分释放编程超能力之前，大家一定要先了解什么是编程，所以我们接下来要花几分钟时间介绍一下这方面的内容。但在这之后，会有很多的实践项目，我保证。

了解计算机编程

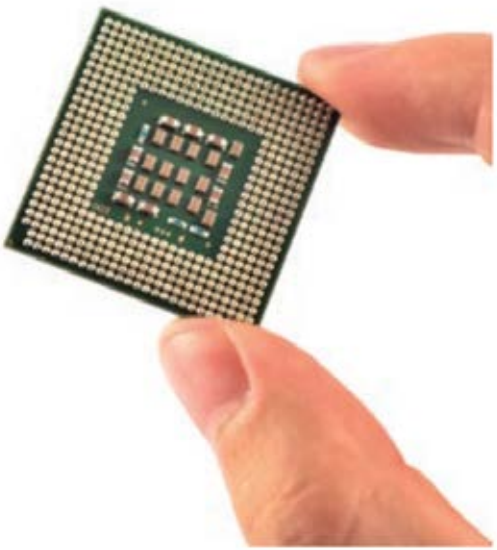
我们先花几分钟时间来了解到底什么是计算机编程。首先，来看什么是计算机。

什么是计算机

大家肯定见过不少计算机，而且它们基本上都差不多，有屏幕、键盘、鼠标或触摸板，笔记本计算机还可以对折合上。计算机都是这样的，对吧？错！实际上，大家见过和用过的大多数计算机看起来根本就不是那样的。真的。

举个例子，游戏机实际上也是一种计算机。智能手机、智能手表和智能电视也是。实际上，任何名称中带有“智能”二字的设备基本上都是计算机。就连能显示谁在你家门前的智能可视门铃也是计算机。扫地机器人、花里胡哨的触摸屏温控器以及车上的显示控制台也都是计算机。无人机是带有螺旋桨的计算机，而特斯拉汽车是带有车座和车轮的计算机。美国宇航局派往火星的那些很酷的机器人也是计算机。银行里的自动柜员机，还有超市的自助收银机，这些都是计算机。明白我的意思了吧。世界上有数以亿计的计算机，而且大多数看起来都不同于我们日常所说的计算机。

那么，是什么让所有这些设备成为计算机的呢？答案是内置的微处理器，也就是一个作为设备大脑来运转的计算机芯片。微处理器控制着设备的所有元器件，显示屏、马达、输入、传感器、扬声器等都由一个或多个微处理器来控制。



现在，明白什么是计算机后，不妨试着回答一下这几个问题：“计算机有智能吗？游戏机有智能吗？智能手机或平板计算机呢？”

很遗憾，答案都是否定的，它们根本没有智能。计算机并不聪明。实际上恰恰相反，尽管名字中有“智能”一词，但计算机其实是相当愚蠢且无能的。

为什么说计算机不聪明呢？这是因为尽管它们很强大，但是它们完全不知道怎么自主决定做任何事情。它们不能自行在屏幕上显示视频，不能自行响应鼠标单击或摇杆控制，不能自行连接到互联网，不能自行理解用户输入的内容，不能自行运行游戏或者加入视频会议。单凭它自己的话，计算机根本做不了任何有用的事情……所以我们可以说，它并不聪明。

但计算机确实能做许多了不起的事儿，那它们究竟是怎么知道该如何做这些事情的呢？因为有人在指挥它们。有人给了计算机非常具体的指令，教它们做那些电子设备该做的事情，而这些指令的确很聪明。

真正的智能来自创建指令的人。指挥世界上所有计算机来做有趣和有用的事情的，是计算机程序员。而计算机编程的本质就是指挥计算机为我们做事情。

如何与计算机交流

和朋友交谈时，他们能理解你所说的话（嗯，希望如此）。这是因为你们在用双方都能理解的语言进行交流。这非常重要。和一个不会说同一种语言的人讲话，很难称得上是真正意义上的交流（鸡同鸭讲，了解一下）。

名字有什么意义？

计算机程序员也称为程序员、软件工程师、应用开发人员和软件开发人员。虽然头衔很多，但是意思其实都是一样的。

与计算机进行交流也如此。想让计算机做什么事情时，需要使用计算机能理解的语言——也就是用计算机编程语言——来给出指令。和我们人类的语言一样，编程语言也是有单词和语法的。

计算机语言有很多。一些语言有特定的用途，另一些则比较通用。大多数程序员都会学习并使用多种语言，能根据各种特定的情况选择最合适的语言。若是觉得需要学习很多语言让人望而生畏的话，别担心，有下面这些好消息。

- 确切的语言细节，比如用词和语法（程序员称之为语法）因语言而异。但和人类语言不同的是，计算机编程语言的单词和规则往往都很少，我们基本上很快就能掌握。

新术语

语法（syntax） 在人类语言中，语法指的是组合单词和短语来构成整个句子的规则。在编程语言中，语法一词的含义也与之类似，指的是语言元素的使用规则。

- 此外，几乎所有编程语言的基础操作都是共通的（本书将全部介绍）。这意味着，掌握一种编程语言后，再学习另一种语言就会容易得多。
- 对于编程语言，永远不要尝试去死记硬背。想不起怎样用某个语言执行某项特定任务时，可以参考专业程序员的做法——上网搜索。
- 人类语言和专业编程语言之间，有一个重要的不同：听众。给朋友发信息时，你可以写错字（不过最好不要）、省略标点符号（也最好不要）甚至发送不完整的句子（唉，真的别这么做），但朋友仍然能看得懂。计算机的宽容度非常低（毕竟它们不聪明，记得吗？），但凡漏掉一个句点 . 或一个花括号 }，计算机就会不知所措。这个地方可能最容易让新手程序员产生挫败感。那么，为什么要把这一点列为好消息呢？因为编辑器（用来写代码的工具……很快就要进行详细说明了）非常善于捕捉这些错误，它们大大减轻了程序员的负担。

知道了什么是编程语言以及它们为什么和人类语言相似后，现在是时候告诉大家一个秘密了（嗯，实际上是两个秘密）：计算机编程的本质以及程序员工作的真相。

- 假设你知道母语中的每个字，已经把《新华字典》背得滚瓜烂熟，还掌握了每个字和词的发音及定义。难道这样就能成为一名畅销书作家了吗？认得所有的字就意味着能写出一部大片的剧本了吗？当然不能。知道这些字是一回事儿，而知道如何创造性地将这些字组合成好文章却是另外一回事儿。显然，两者截然不同。计算机语言也是如此。掌握语法是轻而易举的（毕竟计算机语言的词汇量比人类语言少得多）。经验丰富的程序员，典型特征是知道如何运用这些语言元素来巧妙地解决问题，这正是我们想帮助大家学会和掌握的技能。这是一项需要时间和实践才能打磨出来的技能。
- 再想想人类语言。分享想法是否只有唯一一种正确的方式呢？当然不是。如

果真的是这样，所有电影和书籍都会是一模一样的，那也未免太可怕了！语言是一种工具，而作者要用这种工具来创造各种美妙而独特的体验。使用编程语言时也如此。写代码或解决任何具体问题都没有唯一的正解，相反，有多少个程序员，就会有多少个解决方案。本书将展示各种技术和解决方案，大家可以在编程的时候自由使用。但随着时间的推移，大家将能找到自己独特、新颖的解决方案，就像专业的程序员那样，因为创造性地解决问题是编程的真谛。

什么是 Python

我们选择 Python 这种语言来介绍如何编程。

没错，Python 是一种用来向计算机“发号施令”的语言。Python 不算是比较新的编程语言，它实际上已经问世 30 多年了。它特别流行，大家最喜欢的一些网站和应用程序可能都是由 Python 来驱动的。为什么 Python 如此受欢迎呢？

- Python 真的很好用，用不着复杂的工具或设置，写几行代码就可以了。真的！实际上，在本章中，我们就要开始动手写代码。
- Python 的创造者非常努力地创造出了这种语法非常容易理解的语言。读 Python 代码就像读英语一样。其他大多数编程语言读起来要复杂得多（真的是这样）。
- 不必担心前面提到的那些让大多数新手程序员感到沮丧的语法规则。因为 Python 的语法是所有编程语言中最简单的，没有之一。规则越少越好！
- 好用、相对宽松的语法规则固然不错，但真正让 Python 用起来如此有趣的是它所提供的各种库。本书的后面部分将详细讨论库，现在，我们只需要知道，库是其他程序员编写的代码，下载即可使用。因此，那些本来很复杂的任务（比如在应用程序中嵌入谷歌地图和导航，或在游戏中检测汽车何时撞上障碍物）就变得简单多了。不需要什么都自己从头开始做，有了 Python，我们可以基于其他聪明的程序员所分享的代码进行构建。

接下来，我们要一起学习 Python。不过，正如前面所解释的那样，我们要学习的是所有语言都用得上的概念和技术。学完这本书后，大家可以把辛苦学会的知识优势和专业知识应用到其他任何一种语言中。

Python 和其他语言

正如前面所说，不同的编程语言有不同的用途。Python 通用性强，并且能为各种网站提供支持。但它不适合用来构建移动客户端应用，因为其他语言更适合。尽管如此，但是在用 Java 或 Swift 创建 Android 或 iOS 应用时，学习 Python 过程中掌握的技术绝对也是有关联且能派上用场的。

安装和设置

好了，言归正传。完成安装并设置好之后，我们就可以开始动手编程了。需要用到两样东西：Python 语言和一个编辑器。步骤不少，但是只需要做一次就行。我保证。

小贴士

请留意二维码 书中提到的所有链接和下载地址都可以在本书作者提供的英文版配套网站上找到，网站的网址是 https://forta.com/books/0137653573/（直接扫描二维码也可以访问）。扫描书中出现的二维码，即可指向原书作者为英文版开发的网站，可在此获得下载链接、提示和更多内容（后期可能有中文相关内容）。

安装 Python

大多数计算机都没有自带 Python 语言，所以我们首先要做的事便是安装 Python，可以免费下载。请按照以下步骤进行操作。

1. 打开浏览器，进入 https://python.org。
2. 单击屏幕上方的 Download（下载）。

用的是 Chromebook 吗？

如果用的是 Chromebook，也可以使用 Python，不过安装步骤有些复杂。扫描二维码，即可在本书网页中找到详细的信息（目前为英文版）。

3. 页面上有一个下载 Python 的选项。Windows 和 macOSX 的安装程序不同，但下载界面会自动显示对应的程序，如若不然，请手动选择。

   下载最新版本的 Python。写作本书时，最新版本是 3.9。

4. 下载完成后，双击下载文件，启动安装程序。可以直接保持所有默认选项不变，安装程序会履行它的职责。

小贴士

用的是 Windows 操作系统吗？　在安装过程中，可能会有一个复选框 Add Python to PATH（将 Python 添加到路径）。为了方便以后使用，请一定要勾选这个复选框。

安装完成后，计算机上就有可用的 Python 了。但在开始编程之前，还需要执行两个步骤。

安装和配置 Visual Studio Code

想要编写文档时，我们需要用到谷歌文档或微软 Word 这样的文档编辑器。想要制作视频时，则需要用工具来拍摄和编辑视频。对吧？编程也不例外。写代码也需要用到编辑器，用来写代码和编辑代码。

市面上有许多款编辑器，可以用来创建和打开文件、输入代码以及保存文件。这都是很常规的功能，因此，大多数程序员都会用特殊的编辑器来做更多的事情。这些特殊的编辑器称为 IDE（“集成开发环境”的英文缩写）。IDE 不仅能打开、编辑和保存文件，还有一些非常实用的功能。它能高亮显示代码中的错误，还能为代码着色，使其可读性更强，等等。如此说来，是的，我们需要安装一个 IDE。

Python 有一个名为 IDLE 的内置 IDE，虽然用起来还行，但是，另外还有更好的选择。我非常喜欢的 IDE 是 Microsoft Visual Studio Code（或简称 VS Code）。我自己用的就是它，也强烈推荐给大家。

Visual Studio Code

VS Code 是 Microsoft Visual Studio 的兄弟，后者需要付费购买。如果能用完整版的 Visual Studio，那么可以选择用它而不是用 VS Code。

如果用的是 Chromebook？

如果使用的是 Chromebook，那么只有在 Chromebook 支持 Linux 的情况下，才能用 VS Code。如果想知道怎样检查是否支持 Linux 以及如何在 Chromebook 上安装 VS Code，那么可以扫描二维码进入本书的页面（英文说明）。

喜欢 VS Code 的原因有很多：运行速度很快；提供大量的内置帮助和语法支持；还支持包括 Python 在内的各种语言。噢，而且它是免费的。

请按照以下步骤安装 VS Code。

1. 打开浏览器，在地址栏输入并访问 https://code.visualstudio.com/。
2. 单击右上方的蓝色 Download 按钮。像 Python 一样，Windows 和 macOS 的安装程序不同，下载页面会自动显示针对不同平台的链接。否则，单击左侧 Download for Windows 按钮旁边的下箭头，手动选择。
3. 下载最新版本的 VS Code（即 Stable Build）。
4. 下载完成后，双击下载的文件，启动安装程序。
5. 如果出现一个 Edit with Code（用代码编辑）复选框，那么请勾选它。此外，可以保留所有默认选项不变。
6. 安装程序完成后，请确保勾选 Launch Visual Studio Code，然后单击 Finish（结束）。安装程序关闭后，VS Code 将会启动。

很快就要搞定了。记住，VS Code 支持各种语言。而我们要用的是 Python，所以现在要做的最后一件事就是让 VS Code 知道。VS Code 知道我们要用 Python 后，就会安装额外的软件来提供 Python 支持。

VS Code 启动后，会显示一个欢迎界面，其中有教程、文件和更多内容的链接。在 VS Code 界面的左上角，有下图所示的几个图标。

这些图标是常用图标，不过现在要关注最下面那个图标。单击它，然后扩展面板会显示出来，里面列出的扩展应用可以安装到 VS Code 中。VS Code 支持多种不同的语言。每次需要支持一种新的语言时，安装正确的扩展即可。扩展面板如下图所示。

在扩展面板的顶部，有一个部分名为“已安装”。在还没有安装扩展的情况下，右侧的计数可能显示为 0。

如果“已安装”右侧的数字不是 0，而且 Python 已经在“已安装”部分中，则就意味着 Python 扩展已经安装好了。

如果 Python 扩展不在“已安装”列表内（大概率是这样），就需要安装它。过程很简单。Python 通常列在“推荐”一栏中（如果没有的话，请在上方搜索栏中输入 Python 进行查找），单击 Python 右侧的蓝色安装链接，就可以开始安装了。在这之后，Python 将被列入“已安装”列表内，VS Code 可能会显示新的 Python 欢迎界面。

新建工作文件夹

程序员将代码保存在文件夹中。他们通常有一个用于存储所有项目的主文件夹，其中有为每个特定项目或应用程序新建的子文件夹。现在，我们来做这件事吧。

Windows 用户

Windows 用户有一个“文档”文件夹，其中存放着各种文档。这也是个存放代码的好地方。请按照以下步骤进行操作。

1. 单击 Windows“开始”按钮（通常在计算机屏幕的左下方），然后会出现下图所示的几个图标（不过，这些图标可能看起来有些不同，取决于 Windows 的版本）。

最上面的图标（看起来像是有折的纸）就是“文档”图标。把鼠标悬停在上面，如果浮窗中显示“文档”的话，就说明你找对了。

用的是 Chromebook？

扫描下面这个二维码，即可在本书英文版的网站上找到 Chromebook 对应的具体说明。

2. 单击“文档”图标，打开 Windows 文件资源管理器中的“文档”文件夹。
3. 找到窗口顶部的新文件夹图标。它应该看起来像下图这样。

4. 单击“新建文件夹”图标，新建一个文件夹。新建的文件夹的默认名称如下图所示。

5. 将文件夹的名称改为 Python，然后按 Enter 键保存。

好了，新的文件夹成功创建完成了，干得不错！

Mac 用户

Mac 用户的“文档”文件夹中存放着各种文档，也是个存放代码的好地方。请按照以下步骤操作。

1. 找到 Finder（访达），通常在计算机屏幕的底部，如下图所示。

2. 单击打开 Go（前往）栏，选择下拉菜单中的 Documents（文档），随即打开文件夹，如下图所示。

3. 打开文件夹后，就可以开始新建文件夹了。单击打开 File（文件），选择 New Folder（新建文件夹），如下图所示。

4. 一个新的文件夹就这样创建好了。
5. 如下图所示，将新建文件夹命名为 Python，然后按 Enter 键保存。

就这样，新的文件夹成功创建好了，干得不错！

编写第一个 Python 程序

为了确保一切都能正常工作，我们要写一个很简单的程序来进行判断。为此，需要先新建一个工作文件夹，然后再写一些代码。

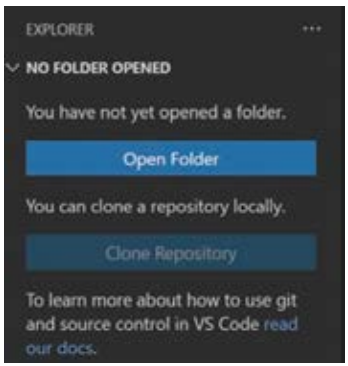
选择工作文件夹

我们知道，程序员把代码文件放在文件夹中，所以我们刚才要新建文件夹。现在，我们需要让 VS Code 知道这个新的工作文件夹在哪里。

第一，回到 VS Code 窗口左上方的按钮，如下图所示。

单击最上面的按钮可以打开或关闭资源管理器面板，在其中可以看到所有的文件。单击这个按钮来显示资源管理器（如果还没有打开的话）。

由于我们还没有告诉 VS Code 工作文件夹在哪里，因此会显示信息，表明尚未打开文件夹，如下图所示。



第二，单击 Open Folder（打开文件夹）后，屏幕上会跳出常规的（Windows 或 Mac）文件夹界面。

第三，定位到刚才创建的 Python 文件夹，然后单击 Choose Folder（选择文件夹）。

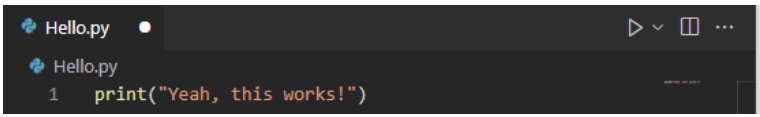
现在，我们有一个打开的文件夹了，不过里面什么都还没有，如下图所示。

创建好工作文件夹后，就可以开始编程了。

编程时间

现在就来开始新建文件并编写代码吧！请记住新建新文件的步骤，因为后面经常需要这样做（从下一章开始）。

1. 把鼠标移到资源管理器中的 PYTHON 一栏。看到 PYTHON 右边的四个图标了吗？单击第一个图标即可新建一个文件。单击它并为新建文件命名为 Hello.py（.py 这个扩展名是不可或缺的，创建每个 Python 文件时，都必须以 .py 作为扩展名）。
2. 按下 Enter 键后，文件将被保存。除了资源管理器面板以外，还需要了解 IDE 界面的另一部分，也是最重要的部分：编辑器，也就是右侧大的方框，输入代码的地方。新建的文件应该已经自动打开，随时可以开始编程了。如果没有的话，就在资源管理器面板中双击文件名打开。
3. 开始动手写代码吧！请在屏幕的编辑器部分输入下图中显示的内容：

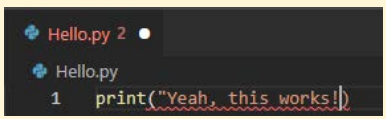


先不要关注代码本身。注意到了吗？编辑器上方显示的是文件名。当同时打开很多个文件时，这一点尤为重要，可以通过文件名来区分不同的文件。

4. 最后要注意的一个重点是代码右上方的箭头，它是用来运行代码的。如果把鼠标悬停在箭头上的话，屏幕上就会显示 Run Python file。可以把鼠标悬停在 VS Code 的任何选项上，看看它们的用途。

IDE 很好用！

请留意，VS Code 自动为代码着色，因而代码更容易阅读。如果代码中有错误的话，VS Code 还会将其标注出来。比方说，如果去掉第二个引号，就会看到提示错误的红色波浪线（请随意尝试，但试完之后，记得把引号加回去）：



红色代表出问题了。文件名变成红色意味着文件中有错误，VS Code 会在代码中添加一条红色的波浪线，把出错的位置显著标注出来。看到了吧，IDE 真的很好用。

执行代码

大家可能听程序员讨论过与执行（executing）代码相关的话题。这并不是一件坏事。没有人会“干掉”这些代码[①]。execute 是 run 的另一种说法，执行代码意味着运行代码。

5. 单击运行按钮，Python 将会运行代码。那么在哪里能看到运行代码的结果呢？答案是在“终端”窗口，在编辑器窗口的下方。我们让 Python 打印（意味着展示）了一些文本，于是，它就在终端窗口中依言照办了，如下图所示。

```
Yeah, this works!
```

如果以上文本成功显示在终端窗口中，就意味着 Python 已经安装完成并在正常运行，同时，VS Code 也已经安装完成并可以与安装的 Python 交互了。我们已经彻底准备好开始编程了。恭喜大家！

小结

本章介绍了什么是编程、程序员有哪些职责以及什么是 Python。我们安装了 Python 和 Visual Studio Code （并简单了解和使用了后者），我们还写了第一个出色的（嗯，可能也没有那么出色）程序。现在，我们已经准备好正式用 Python 进行编程了。

① 译注：execute 在英文中有“处决”的意思。

第 2 章

填字游戏：函数和变量

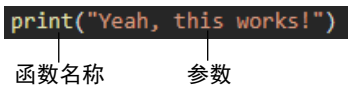
一切准备就绪，是时候真正开始动手编程了。本章首先介绍一个重要的知识点——变量。我们之后所写的每个程序都会用到它。同时，我们还将学习函数并创建一个简单的游戏。大家准备好了吗？

函数

在编程语言中，函数（function）指的是一些执行特定任务的代码。实际上，前面已经出现过一个函数，第 1 章末尾用到的 `print()` 函数。如大家所见，`print()` 的作用就是打印（也就是显示）文本。

和其他大多数编程语言一样，在 Python 语言中，函数的用法是在名称后加上一对圆括号。程序员把使用函数称为调用函数。

在第 1 章中，我们调用了 `print()` 函数，如下图所示。

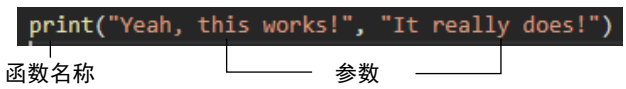


`print()` 打印的文本称为参数（argument），参数总是出现在一对圆括号 () 之间。程序员将这个过程称为“传递”（passing）参数。

一个函数能接受多少个参数呢？嗯，这取决于函数本身。有的函数不接受参数，有些函数接受一个或多个参数，在后面这种情况下，参数将在调用函数时被传入其中。无论是否有参数，圆括号都是必须要有的。

让我们快速看一看如何向一个函数传递多个参数。在 VS Code 中新建一个文件。之前已经有一个名为 Hello.py 的文件，所以这次不妨把新文件命名为 Hello2.py。

现在，Hello2.py 文件已经打开了，在其中输入以下代码：



属于自己的函数

现在我们用的都是 Python 的内置函数。在后面的章节中，将有机会学习如何创建属于自己的函数。

只此一次的友情提示

需要回忆一下新建文件的方法吗？切换到 VS Code 的资源管理器面板，然后单击文件名上方的新建文件图标，就可以新建一个文件并为其命名了，接着，文件自动打开供大家编辑。也可以打开“文件”菜单，选择“新建文件”，不过用这种方式新建文件的话，需要在保存文件时为它命名。两种方式都可以。请记住这些步骤，因为将来我们要新建很多文件。希望在这之后，一看到书中写着“新建一个名为 Hello2.py 的文件”，大家马上就会知道该怎么做。

完成后，打开“文件”菜单并单击“保存”，保存文件并运行它（单击代码右上方的箭头）。下面终端窗口中显示的是结果。

回顾一下刚才含有 print() 函数的那条语句。它有两个参数，print() 在终端窗口中显示了这两个参数。是的，这并不是一个特别有用的例子。我们知道，输入以下代码也可以得到同样的结果：

```
print("Yeah, this works!", "It really does!")
```

之所以前面那行代码要加引号和逗号，是为了说明一个很重要的点：向函数传递一个以上的参数时，必须用逗号来分隔每个参数。若是只有一个参数，就不需要逗号，但如果有两个或更多的参数，请确保每个参数要用一个逗号来隔开。

总而言之，必须使用函数时，要调用它，并在必要时传递参数。如果传递多个参数的话，就得确保用逗号来分隔每个参数。

变量

好了，说了这么多，是时候有请“主角”变量登场了。变量是重中之重，所以我们将用本章余下的后半部分进行详细的讲解。

请想象自己正坐在一张办公桌前完成一个项目。因为要处理大量的信息，所以有许多贴着整整齐齐的标签的存储容器，以便把需要记住的东西随时放入其中。无论何时想要获取一个容器的内容，只需看一下名称，就能获取存储在其中的任何东西。

这个简单的比喻说明了变量的工作原理。为变量命名，把信息放进去，然后随时可以把这些信息取出来。

创建变量

下面来动手实践吧！新建一个文件，命名为 Hello3.py。这里就不提醒大家该怎么做了，毕竟大家都已经是专家了。

然后输入以下代码（请大家像我一样把自己的名字输入到引号中）。

```
firstName = "Shmuel"
```

这段代码新建了一个变量，名为 firstName，并在其中存储了我们指定的名字。

使用变量

如果想要使用刚刚新建的变量，该怎么做呢？只需要用名字来引用它即可。在新建变量那行代码下面添加以下代码：

```
print(firstName)
```

保存并运行代码，然后可以看到下面的终端窗口中打印出了名字。

现在，我们来让它变得更有趣一些。以下面这种形式来更新代码（还是像我一样用自己的名字），然后保存并运行。

```
firstName = "Shmuel"

print("Hello, my name is", firstName, "and I'm a coder!")
```

这里有几件重要的事情需要注意。

传递给 `print()` 的参数有三个。第一个参数和最后一个参数包含我们输入的文本。中间的参数则是 `firstName` 变量，但 Python 没有打印 `firstName`，而是打印了我们存储在 `firstName` 变量中的信息。

留空

注意到了吗？创建变量的那行代码和 `print()` 那行代码之间空出了一行。这个额外的空行称为空白，它可以让代码的可读性更强。实际上，`print()` 函数中每个逗号后面的空格也是一种空白。所有空白都是选择性地留出来的，Python 运行时会忽略，但恰当地利用空白能使代码的可读性更强。

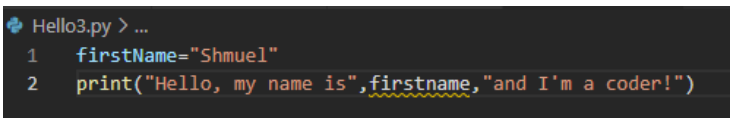
另外还要注意代码的着色。正如我们在第 1 章中提到的那样，VS Code 会自动为代码着色。其实，代码只是一段文本，它并不是真正有颜色。但彩色的代码真的很有帮助。原因有两个：首先，颜色可使代码更易于阅读；更重要的是，因为所有函数都是一种颜色，所有文本是另一种颜色，而所有变量又是另一种颜色，所以我们能通过颜色的不同很快定位到错误。

重要的变量规则

在进一步展开讨论之前，需要先了解 Python 中与变量相关的几条重要规则。

- 变量名可以包含字母和数字，但不能以数字开头。举个例子，`pet1` 可以用作变量名，但 `1pet` 就不行。

- 变量名中不能有空格。如果想在变量名中用多个单词，那么可以用混合大小写的形式（如 firstName）或使用下画线（如 first_name）。
- 需要记住一个最重要的规则，变量名是区分大小写的。也就是说，如果新建的变量名为 firstName，那么就不能用 firstname 来指代它。两者截然不同，一个有大写字母，另一个没有。如果好奇的话，可以尝试一下，看看会得到什么样的结果。把 print() 语句中的 firstName 改为 firstname。VS Code 会识别出这个错误，并且在 firstname 下面加一条波浪线，如下图所示。



如果把鼠标悬停在 firstname 上，VS Code 会显示“firstname is not defined”，意思是 firstname 变量未被定义，这意味着我们正试图使用一个不存在的变量。

在遵守以上规则的前提下，可以自由地以任何方式为变量命名。不过，一般来说，描述性的名字会比较好。比如，firstName 这个变量名就很不错，因为这个名字明确表明了它的用途。相比之下，fn 这个变量名就显得有些指代不明。它可能代表着“肥牛”“风能”“飞鸟”什么的，太容易让人混淆了。使用清晰且具有描述性的名字是专业开发者的特征之一（看起来专业点儿总是好的，对吧？）

变量，更多的变量，更多更多的变量

请看下面这段代码，它有什么作用呢？

```
firstName = "Shmuel"
firstName = "Ben"

print("Hello, my name is", firstName, "and I'm a coder!")
```

变量名是大小写敏感的

由于变量名是大小写敏感的，所以其实可以创建若干个只有大小写不同的变量。比如下面的代码：

```
FirstName = "Shmuel"

firstname = "Ben"
```

不过，建议大家最好不要这样做。尽量保证变量名恰当且独特，以免以后不得不花好几个小时去排查程序错误。

不妨直接在 VS Code 中试试。修改 Hello3.py 中的代码，让 firstName 被定义两次。保存并运行代码，显示出来的结果是否如大家所料呢？

第一行代码 firstName="Shmuel"，新建了一个名为 firstName 的变量，并把 Shmuel 这个名字放入其中。第二行代码 firstName="Ben"，并没有新建变量，也没有把 "Ben" 添加到已有变量中，而是覆盖了第一个值，用 "Ben" 替换了 "Shmuel"。

好，我们来看看最后一个例子。新建一个文件，并为其命名为 Hello4.py（还是像我一样在代码中用自己的名字）：

```
firstName ="Shmuel"
lastName ="Forta"
fullName = firstName + " " + lastName

print("Hello, my name is", fullName, "and I'm a coder!")
```

保存并运行代码，看看会怎样。

Python 每次都是逐行处理代码，所以它做的第一件事是新建一个名为 firstName 的变量，并在其中存储一个值。第二行告诉 Python 新建一个名为 lastName 的变量，并且在这个变量中存储一个值。

小贴士

“另存为”可以省下很多时间 这里有个实用的小提示。Hello4.py 文件只在 Hello3.py 的基础上做了少许改动。可以选择新建一个文件，然后像平时一样输入代码。或者，也可以在 Hello3.py 中点开“文件”菜单并选择“另存为”来新建一个名为 Hello4.py 的副本，然后再进行编辑。

一个变量中可以存储多少个值？

如大家所见，变量一次只能存储一个值。如果试图存储第二个值的话，这第二个值就会取代第一个值。所以一般来讲，变量总只能存储一个值。但实际上，有一些特殊类型的变量中可以存储多个值。后续的章节中将深入研究这些变量。

第三行代码很有意思。它新建了一个名为 fullName 的变量，并在其中存储了一个值。这个值由用加号 + 连接起来的 3 个部分组成。它用 firstName 加上了一个空位符（也就是 " "），然后又加上了 lastName。因此，如果 firstName 变量是 "Shmuel"，lastName 变量是 "Forta" 的话，fullName 就是 "Shmuel Forta"。这也就是 print() 函数中第

二个参数的内容。

> **新术语**
>
> 拼接（concatenation）　将变量连接在一起，称为拼接。想让自己听起来非常聪明的时候，就可以念一念这个英文单词。

获取用户输入

大家已经很擅长用 `print()` 函数了，所以现在是时候认识新的函数了。恰如其名，`input()` 用来要求用户输入（input）一些内容。

新建一个名为 Hello5.py 的文件并输入以下代码：

```
name = input("What is your name? ")
print("Hello", name, "nice to meet you!")
```

保存并运行。

代码在终端窗口中运行，显示“`What is your name？`”并等待我们在终端窗口中输入回复。输入回答并按下 Enter 键后，`print()` 函数会以你的名字问候你。

如大家所见，`input()` 接受要显示的文本，就像 `print()` 一样。但 `input()` 还做了其他事情，也就是从用户那里获得输入。还记得吗？ Python 会逐行运行代码，运行到 `input()` 时，它会停下来，等到用户输入完成后再继续运行。而用户在提示符处输入的任意内容都会提供给大家使用。这个过程称为返回值（returning a value），`input()` 返回的值可以保存到变量中，就像前面那样。如此，`name` 变量中包含用户在终端窗口中输入的内容。

变量在哪里？

请看下面这行代码：

```
input("What is your name? ")
```

大家觉得这行代码有问题吗？其实，这段代码是合法的。运行它后，终端窗口将出现输入提示符。但尽管如此，用户输入的内容并不会被保存下来，这并非我们所愿。因此，必须要用 `name=input()`，只有这样，才能让 Python 把 `input()` 返回的信息保存在 `name` 变量中。

另一方面，`print()` 并不返回任何值，所以不需要将它分配给变量。

小贴士

对 input() 提示符做出响应时，请确保在输入之前单击一下终端窗口。否则，光标可能留在编辑器窗口中，导致编辑的是代码而不是输入。

挑战 2.1

要想成为一名代码超人，唯一的途径就是刻意练习写代码。写的代码越多，就越像是一名优秀的程序员。课程和练习虽然是个很好的开端，但大家真的要试着自己动手写代码。因此，本书中有许多这样的小挑战。在这部分中，我会根据前面讲过的课程设立一些挑战。书中不会直接提供答案，全靠大家独立去解决。别担心，每个挑战都是基于前面学过的知识来设计的，大家肯定可以完美解决。

是时候迎接第一个挑战了。Hello4.py 中，新建的两个变量分别是 firstName 和 lastName。然后，我们将它们合并成一个新的变量，名为 fullName。修改这段代码，使其让用户输入姓和名，而不是用硬编码的值。小提示：只需修改前两行代码，每一行都使用 input() 函数。大家看看是否能够解决这个问题。

填字游戏

填字游戏（比如 Mad Libs[①]）可以根据玩家提供的单词来编出不同的故事。游戏提示玩家给出一些单词（动词、名词和形容词等），这些单词会被插入故事中，以有趣（也可能并不有趣）的方式来编出不同的故事。

自己创造故事

下面我们来实践一下吧。先用“老朋友”print() 函数展现一个简单的故事。既可以选择本书给出的故事，也可以自己编一个。富有创造力的你不妨试着自己原创一个故事。

① 译注：Mad Libs 游戏发明于 1953 年，但正式确定下名称却是 1958 年的事情。当时，斯特恩和普莱斯在美国纽约一家餐厅吃饭，邻桌是一个经纪人和一名演员。演员想要通过即兴表演的方式去参加试镜（即 ad-lib），经纪人却认为这种行为很疯狂（mad）。说者无意，听者有心，于是就得到了有喜剧效果的聚会游戏名称 Mad Libs，其实也有疯狂图书馆的意思。到现在，这套填字游戏累计售出了 1.1 亿册。

新建一个名为 Story.py 的文件，并用 `print()` 函数输入自己编的故事，如下所示：

```
print("I have a pet iguana named Spike.")
print("He is long, green, and lazy.")
print("Spike eats leaves, flowers, and fruit.")
print("His favorite toy is a small yellow ball.")
```

保存并运行。故事将显示在终端窗口中。

添加变量

现在，我们来让事情变得更有趣一些。试着把故事中的一个词替换成一个变量，如下所示：

```
animal = "iguana"

print("I have a pet", animal, "named Spike.")
print("He is long, green, and lazy.")
print("Spike eats leaves, flowers, and fruit.")
print("His favorite toy is a small yellow ball.")
```

Mad Libs®

这是我们对填字游戏的诠释。真正的 Mad Libs® 已经成为企鹅兰登书屋的注册商标。

如大家所见，我们改动了第一条 `print()` 语句。不再直接显示动物的种类，而是用变量来代替。

保存并运行代码，得到的输出和之前一样。

现在，可能还不太有趣。让我们继续吧。接下来要改变很多文本，用变量来替代硬编码的词。在这个示例故事中，有 11 个词被替换成了变量，如下所示：

```
animal="iguana"
name="Spike"
adjective1="long"
color1="green"
adjective2="lazy"
noun1="leaves"
noun2="flowers"
noun3="fruit"
adjective3="small"
color2="yellow"
```

```
noun4="ball"

print("I have a pet", animal, "named", name, ".")
print("He is", adjective1, ",", color1, ", and", adjective2, ".")
print(name, "eats", noun1, ",", noun2,", and", noun3, ".")
print("His favorite toy is a", adjective3, color2, noun4, ".")
```

修改完成后保存代码。

运行代码，得到的输出结果应该和之前的一模一样。

值得注意的是，name 变量在故事中使用了两次。新建一个变量后，可以根据需要反复使用。

注意引号和逗号

使用引号和逗号时，要小心。文本需要加引号，但变量名不需要。还要确保用逗号来分隔所有参数。这就是彩色编程的妙用了。如果颜色不对，就意味着引号或逗号可能错了。

获取用户输入

有了变量之后，就能简单改变它们来显示用户输入的文本。只需像本章前面那样将每个变量改为使用 input()。以我的故事为例：

```
print("Hello, please answer the following prompts.")
print()
animal=input("Enter an animal: ")
name=input("Enter a name: ")
adjective1=input("Enter an adjective: ")
color1=input("Enter a color: ")
adjective2=input("Enter an adjective: ")
noun1=input("Enter a noun: ")
noun2=input("Enter a noun: ")
noun3=input("Enter a noun: ")
adjective3=input("Enter an adjective: ")
color2=input("Enter a color: ")
noun4=input("Enter a noun: ")

print("Thank you. Here is your story.")
print()
print("I have a pet", animal,"named",name, ".")
print("He is", adjective1, ", ", color1, ", and", adjective2, ".")
print(name, "eats",  noun1, ",", noun2, ", and", noun3, ".")
print("His favorite toy is a", adjective3, color2, noun4, ".")
```

我在代码开头处添加了一条用于提供游玩指示的 `print()` 语句，并在代码间穿插了几个空的 `print()` 函数。这些函数添加了空行，使得输出更容易理解。

保存代码并运行。完成所有输入后，代码会生成一个故事。每次重新运行代码时，程序都会根据用户提供的输入生成一个新的故事。

测试完代码后，不妨让朋友或家人试着玩一玩，他们肯定会对你的编程技术刮目相看的。

挑战 2.2

准备好迎接下一个挑战了吗？

要求用户输入至少 15 个不同的单词，让填字游戏变得更有趣。同时，再让游戏更加个性化。在给出游玩说明前，让用户提供他们的名字，并在说明中使用这个名字，创造一个个性鲜明的游戏体验。

小结

本章介绍了关于变量的知识以及使用变量的方法。我们掌握了两个函数 `print()` 和 `input()`，前者用于显示文本，后者用于提示输入文本。随后，我们结合所有学到的知识来创建了一个功能齐全的应用程序。恭喜！大家现在已经成为一名真正合格的、如假包换的程序员了。

第 3 章

掷骰子游戏：库和随机性

学会使用函数和变量后，我们将通过引入库和随机性来使编程变得更有趣。

库的使用

第 1 章中简要地提到过库。我们可以把库当成代码的集合——通常是函数的集合，比如前面用到的 `print()` 函数和 `input()` 函数。库很好用。只需向 Python 说明需要用到什么库，就可以用其中的函数了。就这么简单。没错，别的人把艰苦的准备工作都搞定了，我们可以直接拿来就用。很方便，对吧？

Python 包含很多库。比如，datetime 库提供了各种处理日期的函数，math 库用于进行数学运算，还有许多库用于处理计算机上的文件、访问网站、处理密码学等。而且，库中也可以包含更多函数（后面会讲到）。

如果 Python 中没有你需要的库，或许可以在网上找，找到后下载即可使用。本书的第III部分将学习如何使用第三方库（第三方库意味着库来自于别人，并非 Python 自带的）。

random 库

我们要探索的第一个库是 random，正如其名，它是用来为代码添加随机性的。想获得 1 到 100 之间的任意一个随机数吗？ random 库可以帮上忙。创建游戏时，想让敌人以随机时间间隔和随机血量出现吗？也可以用 random 库来搞定。甚至模拟抛硬币这样简单的事，也可以通过 random 库来实现。

那么，如何告诉 Python 我们想用什么库呢？答案是 `import` 语句。我们来动手实践一下。新建一个名为 Random1.py 的文件并输入以下语句：

```
import random
```

了解 PyPI

Python 库的官方存储库是 PyPI（Python Package Index），包含的库超过 30 万个。

关于随机的真相

我要指出关于计算机和随机性的一个真相：计算机不能随机行事，它们做不到。人类可以随机，但计算机喜欢有条不紊地按指令办事，它们不知道怎么做没有意义或没有特定顺序的事。因此，它们无法真正地随机做事。当计算机看起来像是在随机做事时，实际上是在依靠复杂的算法和不断变化的因素（如当前的日期和时间）来模拟随机性。这可能听起来有些复杂（好吧，实际上确实也是很复杂的）。这就是我们喜欢 random 库的原因。只需使用库中的函数，就可以让库中的代码为我们代劳所有的麻烦事儿。

这行代码向 Python 表明要导入 random 库。保存代码并运行。

发生什么了？是不是好像啥也没有发生？可能看上去是这样，但实际上发生了一件大事。记住，Python 只会逐行运行代码。当运行到有 `import` 语句的那一行时，它会找来 random 库并将其拉到代码中，让其随时待用。现在还没有开始使用这个库。但没有关系，接下来会用的。

生成随机数

再添加如下代码：

```
import random

num=random.randrange(1, 11)
print("Random number is", num)
```

保存并运行。然后，再反复运行几次。注意到了吗？每次运行后，显示的数字都是随机的。

下面来看看代码，了解它是怎么运行的。第一行代码 `import random` 的作用是导入 random 库，最后一行代码的作用是打印文本以及随机数。

把 import 语句放在开头处

一般情况下，要把所有 `import` 语句都放在代码的最上方，以便查看都导入了什么。

现在，把注意力集中在中间那行代码上。它调用一个名为 `randrange()` 的函数，我们把两个定义了范围的数字作为参数传给它，然后，它生成了一个随机数（在这两个数字的范围之间）。不管生成什么数字，都会保存在 `num` 变量中。

这段代码与前面用到的 `input()` 相似。我们根据需要把参数传给它，不管返回什么，都保存在一个变量中。不过，这里有一个重要的区别。

请仔细看代码。`randrange()` 函数并不仅仅是通过函数名来调用的，库的名称也包含在内。`random.randrange()` 告诉 Python 要用 random 库中的 `randrange()` 函数。这一点至关重要。如果不指明库的话，Python 就不知道去哪里找 `randrange()`。

大家如果觉得好奇的话，可以自己试一试。删除 random 库，然后保存并运行代码，会看到 VS Code 在 `randrange()` 下面画了一条波浪线，如果把鼠标悬停在代码上，就会有浮窗显示“`randrange is not defined`”，意为 `randrange()` 函数未被定义。因此，请记住，必须有库的名称。

选择随机项目

我们已经知道怎样生成随机数了，但如果想做别的，比如抛硬币的话，该怎么办呢？在这种情况下，我们希望程序能随机返回正面或反面，而不是返回数字。这时 randrange() 就派不上用场了。但不用担心，可以用 random 库中的另一个函数来实现。

注意范围

randrange（1，11）返回 1 到 10 之间的一个随机数。为什么最大是 10 呢？因为第二个参数是 11，也就意味着最大数小于 11。这有些令人困惑，因为第一个参数 1 包括在范围内，但第二个参数 11 却不然。也就是说，如果想得到介于 3 和 8 之间的一个数字，范围就要设置为（3，9）。好消息是，其他 Python 函数同理，所以大家会慢慢习惯的。

新建一个名为 Random2.py 的文件并输入以下内容：

```
import random

choices="HT"
coinToss=random.choice(choices)
print("It's", coinToss)
```

保存文件并运行代码。每次运行这段代码时，终端窗口中都会显示 H（代表正面）或 T（代表反面）。

那么，这段代码是怎么运行的呢？我们已经知道第一行和最后一行代码的作用了。

choices="HT" 新建了一个名为 choices 的变量，并将文本 "HT" 存储在其中。

下一行将随机选择 H（代表正面）或 T（代表反面）。为此，它使用的是 random 库中的另一个函数 choice()。与接受数字范围作为参数的 randrange() 函数不同，choice() 函数接受一个带有选项列表的参数。这里，我们把包含 "HT" 的 choices 变量传给它，因此，choice() 函数将返回 H 或 T 中的一个。

很简单，对不对？

如果想显示 Heads（正面）或 Tails（反面）而不是 H 或 T 呢？不能像下面这样把 Heads 和 Tails 直接作为选项，因为这是行不通的：

```
choices="HeadsTails"
coinToss=random.choice(choices)
```

另一个选择

即使不用 choices 变量，也能写出同样的代码。像下面的代码一样直接将 "HT" 传入 choice()：

```
coinToss=random.choice("HT")
```

最终得到的结果和前面一样。

之所以行不通，是因为 choice() 会将其视为 10 个选项：字母 H、字母 e、字母 a 等。最终得到的返回值可能是 i，这显然不是我们想要的。

有几种方法可以达到我们的目的。其中之一就是使用一种特殊类型的变量。

还记得第 2 章提到的包含值的变量吗？一些特殊的变量可以包含很多个值。我们未来的课程将更深入地使用这些变量。现在，先试着用这些特殊变量来解决硬币正面或反面的挑战。

只需要对代码进行一个小小的改动可。把下面这行代码：

```
choices="HT"
```

改成下面这样：

```
choices=["Heads","Tails"]
```

保存并运行代码。现在的输出就是 "Heads" 或 "Tails" 了。

那么，这行代码究竟带来了什么样的改变呢？在 Python 中，方括号 [ ] 用于创建一个列表。举例来说，[10,20,33] 将创建一个包含三个项目（三个数字）的列表。同理，["ant","bat","cat","dog","eel"] 将创建一个包含五种动物的列表。

在后面的章节中，会经常用到列表。不过，现在只需要知道列表存储多个项目，每个项目之间都用逗号来分隔就行。

好了，回到我们的代码。choices=["Heads", "Tails"] 这条语句创建了一个包含两个项目的列表，两个项目分别是 Heads 和 Tails。在这种情况下，不必再改变任何其他代码，因为 random.choice() 函数非常聪明。给它一些文本，它就知道我们想从文本中随机抽取一个字符。但如果给它一个列表，那么它就会知道我们想要的是列表中一个随机的项目。

完美搞定！

挑战 3.1

这个挑战的难度比较大，但肯定是可以搞定的，我保证！看到那个有五种动物的列表了吗？编写代码，创建两个列表，其中一个是如下所示的动物名称列表：

```
animals=["ant","bat","cat","dog","eel"]
```

喜欢什么动物就往列表中添加什么动物吧，甚至可以添加五种以上的动物，越多越好。

接下来，新建一个类似的形容词列表，比如 big、green、smelly、cute 等，也是越多越好。两个列表中的项目数量是否相同无所谓。

然后，随机挑选一个形容词和一种动物，并分别将其保存在变量中，需要两个变量：一个是动物，另一个是形容词。然后加上 print()，就可以得到“I have a cute eel”（意思是我有一条可爱的鳗鱼）这样的输出了。每次运行这个应用程序，都会得到不同的组合。

"3" 不等于 3

在进一步探索之前，需要先讨论一个重要的话题。注意到我们在输入变量名的值时有一些不同了吗？提示一下，请仔细看看下面的代码，它们都在前面出现过：

```
lastName = "Forta"
fullName = firstName +" "+ lastName
name=input("What is your name? ")
num=random.randrange(1,11)
choices=["Heads","Tails"]
```

可以看出，有时我们会在值的前后加双引号，有时则不会。这是为什么呢？

因为是文本块（程序员称之为字符串）需要用引号来标记。数字的前后不需要加引号。Python 知道 1 和 11 是数字，它们不可能是其他东西。但是 lastName 呢？它既可以是文本，也可以是一个变量的名称，甚至是一个函数的名称，具体是什么呢？全由程序员来指定。Heads 也是一样，它既可以是一个字符串，也可以是其他东西。计算机不喜欢模棱两可，什么事儿都要清清楚楚地讲明白。因此，使用文本并希望它被计算机视为纯文本时，我们必须要用双引号。

总结如下。

- 变量的前后不能有引号。
- 数字的前后不需要加引号。
- 字符串的前后总是需要加引号。

好了，我们来让这个问题变得有趣一些。`"3"` 是一个数字还是一个字符串呢？能用它乘以 5 吗？

对我们人类来说，这个问题很简单。在我们看来，`"3"` 当然是一个数字，用它乘以 5 会得到 15。但是 Python 并不知道 `"3"` 是一个数字。因为有引号，所以 Python 会认为 `"3"` 是一个字符串。

如果让 Python 用 `"3"` 乘以 5，会怎样呢？我知道这听起来会让人有些吃惊，但是我们会得到 `"33333"`！ Python 会让字符串重复 5 次，而不是把字符串中的数字乘以 5。真的！

字符串需要引号

如果字符串忘记用引号，那么 Python 会认为你指的是一个变量，并且会显示一条错误信息，提示这个变量不存在。

对比 "3" 和 3

想自己试着对比一下吗？在 Python 中，星号 * 是乘号。可以新建一个文件试试，先输入 `print(3 * 5)`，再输入 `print("3"* 5)`，看看两者的输出结果有何不同。`"3"*5` 会将 5 个字符串 `"3"` 连接在一起。

Python 使得新建变量变得简单

在大多数语言中，我们在新建变量时必须告诉计算机变量的数据类型是什么。Python 在这方面很友好，它会根据我们提供的值自动识别数据类型。

数据类型之间可以进行转换

后面的课程要介绍如何把数据从一种数据类型转换为另一种数据类型。举例来说，这样就可以把字符串 `"3"` 转换为数字 3。

前面的例子引入了数据类型（data types）的概念。简单地说，数据类型就是一个变量所能存储的信息类型。数据类型有很多种，但最常用的是字符串和数字。`lastName = "Forta"` 这条语句新建了一个具有字符串数据类型的名为 `lastName` 的变量。`num = random.randrange(1,11)` 这条语句新建了一个具有数值数据类型的名为 `num` 的变量。

那么，`"3"` 和 `3` 是一样的吗？答案是否定的。前者是一个字符串，后者是一个数字，虽然看起来很像，但是它们的数据类型并不相同。

代码注释

在开始讨论本章的最后一个例子之前，还需要介绍一个重要的知识点。

到目前为止，我们写的所有代码都很简单，通常只有几行。但在之后的课程中，我们要写几十行，甚至几百行代码。为了让代码更容易阅读和理解，程序员通常会在代码中加入注释。

怎么添加注释呢？答案是像下面这样（以下代码来自前面的例子，不过这次添加了注释）：

```
# Import needed libraries
import random

# Define the choices
choices=["Heads","Tails"]

# Pick a random choice
coinToss=random.choice(choices)

# And display it
print("It's",coinToss)
```

在 Python 中，注释以井号 `#` 开头。VS Code 用特殊颜色显示注释，让用户能轻而易举地识别出哪些是注释。

重中之重是要明白 Python 会完全忽略注释。当 Python 看到一个井号 `#` 时，它就会忽略其后的任何内容。注释是为程序员而写的，而不是为 Python 写的。

写注释可能看起来是在浪费时间。但相信我，它真的很重要，优秀的程序员会对所有代码进行注释。主要有下面几个原因。

- 注释可以帮助你阅读自己的代码。
- 注释能提醒你之前做过什么以及为什么要这样做。
- 注释可以帮助别人理解你的代码是用来做什么的。
- 注释有助于其他程序员理解并处理你写的代码。
- 可以在注释中解释各种假设或依赖这些让代码运作所需要的东西。

苏格拉底对话录：关于注释

而且，注释还有一个重要的用法，它可以用来隐藏代码。例如，我们之前修改了一下代码，把 `choices="HT"` 改成 `choices=["Heads", "Tails"]`。这是一个很简单的改动，但要是做了更复杂的改动呢？因此，最好在测试完新的版本之前，把旧的版本保留下来。请看以下代码：

```
# Import needed libraries
import random

# Define the choices
# choices="HT"
choices=["Heads","Tails"]

# Pick a random choice
coinToss=random.choice(choices)

# And display it
print("It's",coinToss)
```

注意，原来的 `choices="HT"` 仍然在文件中，并没有被删除或编辑，只是前面多了一个井号 `#`。于是，这行代码就变成了被 Python 忽略的注释。要想回到先前的版本，只需把这行代码中的井号 `#` 去掉并放在下一行的开头，把第二行代码变成注释。

程序员们称之为“把代码注释掉”，在修改或测试代码时，这是一个相当重要的技巧。

新术语

将代码注释掉（commenting out）　利用注释可以暂时隐藏代码，使其不被执行。

好了，从现在开始，我们要为所有的代码添加注释。

一个骰子，两个骰子

接下来，我们将通过本章的最后一个例子来复习前面学过的全部知识。实际上，不如把例子分解成两个。

大家应该玩过骰子，很多游戏都会用到它。骰子很酷，但计算机骰子更有趣。因此，我们接下来将创建两个程序：一个掷一个骰子，另一个掷两个骰子。

以下是 Dice1.py 的代码：

```
# Imports
import random

# Roll and print
print("You rolled a", random.randrange(1,7))
```

看上去很简单，而且都是前面出现过的代码。保存并运行代码，终端窗口中会出现 1 和 6 之间的一个数字（记住，7 不会包含在范围内）。

唯一不同的是，通过 random.randrange() 返回的数字没有被保存到变量中，而是被作为参数直接传给了 print()。

需要掷一个骰子时，随时可以运行这个程序。

需要用变量吗？

请看下面的代码：

```
import random

print("You rolled a", random.randrange(1,7))
```

它和下面的代码有什么区别呢？

```
import random

num=random.randrange(1,7)
print("You rolled a", num)
```

从功能上讲，这两段代码没有区别，两者都生成并显示一个随机数。

第一段代码在 print() 语句中直接生成随机数。其结果值——即 randrange() 生成的数字——作为参数传给 print()。

第二段代码生成一个随机数，并将其保存在名为 num 的变量中。然后，num 变量被作为参数传给 print()。

两者最后打印的结果一样。唯一不同的是有没有使用变量。

那么，应该使用哪个版本呢？在当前情况下，两个版本之间分不出高下。只有当这个随机数用于其他目的时，例如另一个 `print()` 或一些计算时，这种差异才会变得重要起来。那时，把生成的数字保存到变量中肯定是更好的选择，因为能重复使用。

但需要掷两个骰子的时候，该怎么办呢？可以运行两次前面的程序，然后自己把这些数字加起来。但这并不是我们程序员的做法。我们会选择另外写一个掷两个骰子的程序。

下面是 Dice2.py 的代码：

```
# Imports
import random

# Roll both dice
dice1=random.randrange(1,7)
dice2=random.randrange(1,7)

# Display total and individual dices
print("You rolled", dice1, "and", dice2, "- that's", dice1+dice2)
```

运行这段代码，终端窗口中会显示两个骰子分别的数值以及它们的和。

对已经掌握了 `print()` 函数和 random 库的大家而言，这段代码的作用不言自明。这段代码中新建了两个变量，每个变量都包含一个掷出的骰子的值。结果语句显示了这些值，然后进行下面这样的操作：

```
dice1+dice2
```

这是个简单的数学运算：`dice1` 和 `dice2` 相加，打印出它们的和。没错，Python 可以飞速完成数学运算。

运算符 +

运算符 + 的作用取决于数据类型。在前面的掷骰子代码中，`dice1` 和 `dice2` 都是数值数据类型的变量，因此，当输入 `dice1+dice2` 时，Python 知道这是要把两者加起来。

但如果这两个变量是字符串，Python 就会把两者串起来，因为对字符串做加法显然没有意义。

Python 在这方面很聪明。它会尝试弄清楚程序员的意图。

现在似乎可以趁此机会向大家说明 Python 中可以使用哪些数学运算符，如下所示。

运算符	说明
+	加法运算符。示例：`print(5+5)` 将显示 10
-	减法运算符。示例：`print(12-7)` 将显示 5
*	乘法运算符，本章前面部分中曾介绍过。示例：`print(10*3)` 将显示 30
/	除法运算符。示例：`print(10/3)` 将显示 3.333（实际上，小数点后会有无数个 3）
//	也是除法运算符，但只返回整数，不带有余数。示例：`Print(10/3)` 将显示 3
%	取模运算符，用于从除法中获得余数。示例：`print(10%3)` 将显示 1

很明显，这些运算符可以应用到不同的代码和函数中。如果好奇的话，可以尝试运行表格中的示例 `print()` 语句，看看这些运算符的实际效用。

那么，为什么我们在 Dice2.py 中使用变量来存储骰子的值，但在 Dice1.py 中却没有使用变量呢？其实，我们完全可以在两个文件中都使用变量。但只有在数值会被多次使用的情况下，才真正需要变量。在 Dice1.py 中，值只在显示的时候才使用一次，所以尽管可以使用变量，但在这种情况下，变量是可有可无的。在 Dice2.py 中，骰子的值用了两次：一次是在两个骰子的点数显示的时候，另一次是在两个点数加起来以获取总和的时候。在这种情况下，有必要把骰子的点数保存到变量中。

挑战 3.2

平时，我们最常用的是六面骰子。这样的骰子是一个正立方体（正六面体），各个面分别有一到六个孔（或数字），对立面两个数相加为七。在桌面游戏中，常见的正多面体骰子有四面、八面、十面、十二面和二十面。[①] 事实上，古希腊人和古罗马人用的就是十二面。大家不妨用 Python 来写一个掷骰子游戏，以便时空穿越到过去和古希腊人及古罗马人切磋一下十二面的骰子怎么玩。

小结

这一章涵盖了很多内容！我们学会了使用库，尤其是 random 库和它的两个函数。我们探索了数据类型，还学会了为代码添加注释。在下一章中，我们将研究如何教会代码做决定。

① 译注：此外还有一些少见的多面骰子，例如十四面、三十面、六十面、一百二十面，以及不太实用的一面的（莫比乌斯带）、五面的、一百面的和球形的。国内最常用的骰子习惯于把一点和四点涂上红色，据清代赵翼考证，最早使用红四点骰子的是唐玄宗。

第 4 章

计算时间差：datetime 库

现在，我们已经学会使用变量、函数和库了。接下来，要探索最为重要的编程工具之一：决策机制，让计算机来做决定。

与日期打交道

如大家所见，Python 是逐行处理代码的。Python 从程序的顶部开始，一行一行地（除了注释之外）按照我们的要求做事。

这其实是很无聊的。如果每个程序都逐行执行的话，那每次运行时做的事情不就完全相同了吗？想象一下，每次访问一个网站时都以同样的顺序显示同样的内容。或者在一个游戏中只能以同样的顺序一步一步地操作。又或者在一个聊天软件中，永远只能输入并发送同样的信息。明白我的意思了吗？这样未免也太无聊了！

很明显，所有实用的程序都必须能以不同顺序做很多不同的事情。这意味着，作为程序员的我们需要想办法告诉计算机如何做决定。

这就轮到至关重要的 `if` 语句出场了，它是本章（和下一章）的重点。

datetime 库

为了给人留下深刻的印象，数学家们很喜欢的一个“戏法”是，询问对方的生日，然后在几秒钟内说出那一天对应的是星期几。这并不是他们瞎蒙出来的（毕竟如果只是蒙的话，只有七分之一的概率能蒙对，难以给人留下深刻的印象），而是在脑海中快速计算出来的。

我们当然可以学习数学家们的计算方法，但身为程序员，我们可以用自己的编程技术来给人留下深刻的印象，让计算机给出结果。

首先，我们需要了解 Python 内置的另一个库 datetime。正如其名，这个库可以处理各种有关日期的事情，如下所示。

- 获取当前日期。
- 计算出未来和过去的一个日期的相关细节（比如当天是星期几）。
- 计算两个日期之间的间隔（若是需要考虑月长和闰年的因素，实际上相当难算）。

对了，datetime 库也可以对时间进行同样的操作。

datetime 库

之所以选择 datetime 库作为我们接触的第二个库，一部分原因是它真的很实用。还有个原因在于，它的工作方式与前面的 random 库略有不同。体验使用各种不同的库对学习编程是很有益的。

新建一个文件，命名为 Date1.py，然后输入以下代码：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()

# Print it
print("It is", today)
```

保存并运行，终端窗口中将显示当前的日期和包括毫秒在内的时间（我知道，这很实用）。

我们已经知道了 `import` 关键字和 `print()` 函数的用途，所以要把注意力放在中间这一行上。嗯，说实话，这行代码看起来有点奇怪。

`today=datetime.datetime.now()` 这行代码获取当前日期，并将其保存在 `today.now()` 中。显然，`datetime.datetime.now()` 是个返回当前日期和时间的函数。与迄今为止用过的函数不同，这个函数不需要任何参数。不过，括号仍然是必须要有的。当调用函数时，我们必须提供括号，但可以不传入参数，而是括号内留空，也就是让括号中不含任何内容。

但是 `datetime.datetime` 是怎么回事呢？为什么不能像第 3 章使用 random 库函数时那样直接用 `datetime.now()` 呢？

函数与变量

请记住，当函数执行时，后面总是带有括号。变量后面则没有括号。

第一个 `datetime` 确实是库的名称。它与第一行代码 `import datetime` 中提到的库相匹配。

第二个 `datetime` 并不是库或函数。实际上，它是一个类（class）。本书的第 II 部分将详细介绍类，并且我们还要自己创建类。现在只需要明白，程序员利用类来整理代码，使函数和信息片段存储于一处。类中包含着可以像其他函数一样调用的函数。

新名词

方法（method）　类中的函数称为“方法”。但它们仍然是函数，与前面出现的那些函数一样。

datetime 库中有一个名为 datetime 的类（是的，我和大家一样，也觉得两者的名字如果不一样的话，更好理解）。也就是说，datetime 是导入的库，datetime.datetime 指向 datetime 库中的 datetime 类。

now() 则是一个函数，包含在 datetime 库中，它返回的当前日期和时间保存在 today 变量中。

type() 函数

第 3 章介绍了数据类型。那么，我们刚刚创建的 today 变量的数据类型是什么呢？它不是字符串或数字类型，而是 datetime 类数据类型。

如果想要知道一个变量的数据类型是什么，那么可以使用 Python 的 type() 函数。type() 所做的就是查看变量，并返回其数据类型的对应值。type(3) 将返回 int（表示整数），因为 3 是个数字。type（"3"）将返回 str（表示字符串）。而 type(today)（前面创建的变量）返回的类型是 datetime.datetime。

在后面的章节中，将更深入地研究类型和 type() 函数。

使用 datetime 类

在本章的第一个例子中，我们仅仅打印了 today 变量中的内容。但由于 today 是个类，所以它的许多数据和函数其实都可以使用。

下面试着再来做一个练习。新建一个名为 Date2.py 的文件，输入以下代码：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()

# Print today's year, month, and day
print("The year is", today.year)
print("The month is", today.month)
print("The day is", today.day)
```

保存并运行该代码，终端窗口中将分行显示当前的年、月、日。

today.year 指的是 today 类中的 year 值。month 和 day 同理。

如大家所见，today 变量包含很多信息。先前，我们打印了整个 today 变量（没有特别指明是其中的哪一项），如以下代码所示：

```
print("It is", today)
```

这里，Python 帮了个忙，以默认的可读格式显示了所有信息。但我不推荐这样做。一般来说，最好只显示需要用到的项目，以便能更好地控制输出。

方法与属性

为什么 year、month 和 day 之后没有括号呢？因为它们不是函数（或者说方法），而是可以使用的数据片段，和类中的变量比较相似。它们实际上被称为属性。我们将在第 II 部分中深入讨论属性（和方法）。

引用属性时，不需要使用括号。使用方法（又称函数）时，则需要使用括号。后面很快就会讲到这方面的例子，我们后面要用到 weekday()。

挑战 4.1

修改 Date2.py，使其也可以显示当前时间。需要添加两个属性，分别是 hour 和 minute。

做决定

知道如何获取日期后，是时候回到主题上了。接下来，我们将探索 if 语句，研究如何用这些语句来帮助计算机做决定。

if 语句

我知道，大家恨不得把一天中的每分每秒都用来写代码。但没有办法，还得做其他事，比如说上学，对吧？所以，让我们写一个能识别出当前是星期几的程序，然后为不同日子显示不同的实用信息。很明显，这需要靠计算机来决定。计算机不能只是死板地逐行打印信息，而是必须根据日期的不同做不同的事。

新建一个名为 Date3.py 的文件并输入以下内容：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()
```

```
# Display the right message for different days of week
if today.weekday() == 6:
    print("It's the weekend, no school today!")
    print("We can code all day long!")
```

保存并运行代码。发生什么了呢？嗯，如果今天碰巧是星期天的话，终端窗口中会显示信息（两条 `print()` 语句）。但如果今天不是星期天的话，就什么都不显示。

神奇的是下面这行代码：

```
if today.weekday() == 6:
```

`if` 是用来创建条件的。条件是从 `if` 后开始直到句尾的冒号之前的内容。`today.weekday()` 方法返回的是当前是星期几，`0` 代表星期一，1 代表星期二，以此类推。这个条件非常简单。它告诉 Python 调用 `today` 变量中的 `weekday()` 方法，然后将其返回的结果与数字 `6`（代表星期日）进行比较。所以，以上语句创建的条件是“如果今天是星期天，就……”

请特别注意，`weekday()` 和 `6` 之间是两个等号，而不是一个。双等号 == 意味着检查两个东西是否相等。它与等号 = 不一样，后者的作用是将值保存到变量中，正如前文所述。

传递给 `if` 语句的条件必须是一个能被判定为 `True` 或 `False` 的条件。就这个例子而言，如果今天确实是星期天，那么条件就是 `True`。如果不是，那么条件就是 `False`。

= 与 ==

= 和 == 是不一样的，许多程序员会花好几个小时试图弄清楚代码为什么会出错，到最后却发现是错把 = 写成了 ==，反之亦然。

因此，为了把 = 与 == 明确区分开，需要记住它们各自的定义和用途。首先，= 是赋值运算符，它的作用是赋值，也就是将 = 右边的内容保存到左边的变量中。举个例子，`x = 3` 创建一个名为 x 的变量，并将数字 3 存入其中。然后，== 是等号，用于比较。举个例子，`x == 3` 的作用是检查变量 x 的值是否为 3。

千万不要记混了！

Python 中的一周有些奇怪

Python 的一周是从星期一开始的。没错，Python 认为星期一是一周中的第一天，而星期天是最后一天。

而且，和几乎所有编程语言一样，Python 是从 0 开始计数的。举个例子，如果有一个项目列表的话，它们的编号会是 0、1、2，以此类推。第一个项目的位置是 0，而不是 1。

把这两个点结合起来，也就意味着 0 是星期一，1 是星期二，5 是星期六，6 是星期日。

注意缩进

要留心缩进。假设前面的代码像下面这样：

```
if today.weekday() == 6:
    print("It's the weekend, no school today!")
print("We can code all day long!")
```

在这种情况下，如果今天是星期天的话，第一条 `print()` 将被执行。但无论是星期几，第二条 `print()` 语句将总会被打印。为什么呢？因为第二条 `print()` 语句不在 `if` 语句的作用域中，它只是一行普普通通的代码，总是要被执行的。

那么，Python 怎么知道在条件为 `True` 时要运行什么代码呢？答案是 Python 会寻找并运行 `if` 语句下所有缩进的代码行。当 Python 遇到没有缩进的代码行时，就知道对 `if` 语句的处理已经完成了。

else 语句

目前的代码能在星期日时打印出信息。但如果今天不是星期天，Python 就不会打印出任何内容。现在来解决这个问题。以下是改动后的代码，主要是在底部增加了两条新的语句：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()

# Display the right message for different days of week
```

```
if today.weekday() == 6:
    print("It's the weekend, no school today!")
    print("We can code all day long!")
else:
    print("It's a school day.")
```

保存并运行代码。现在，终端窗口将在星期日时显示前两条 print() 语句，其他时候则显示最后一条。

else 用来定义当 if 语句为 False 时（在这个例子中，只要不是星期天就为 False）要运行的代码。无需给 else 设置条件，只需输入 else:，然后输入要运行的代码。在那之后，当 if 条件为 False 时，else 后的缩进内容就会被执行。

改进 if 语句

if 语句还有个问题。它只检查 weekday() 是否会返回 6（也就是星期日），那星期六怎么办呢（也就是 5）？

我们需要将 if 语句的条件改为检查星期六或星期天。如下所示，更新后的代码中只有 if 语句发生了变化：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()

# Display the right message for different days of week
if today.weekday() == 5 or today.weekday() == 6:
    print("It's the weekend, no school today!")
    print("We can code all day long!")
else:
    print("It's a school day.")
```

保存并运行代码。

发生了什么变化呢？修改后的 if 语句的条件变成了两部分，它检查两件事：weekday() 返回 5（星期六）或者 weekday() 返回 6（星期日）。or 意味着只有在其中一个条件为 True 时，if 语句才为 True（而且其下的缩进代码行会被执行）。现在，星期六和星期日时显示的信息（前两条 print() 语句）都是正确的了。

当为 if 语句提供多个检查时，总是得用 and 或 or 将它们连接起来。and 和 or 有什么区别呢？请看下表中有关条件的例子（我将用文字而不是代码来讲解）。

条件	描述
if 午餐是比萨 and 甜点是冰淇淋	if 语句在什么情况下为 True 呢？只有当午餐确实是比萨且甜点也确实是冰淇淋时才为 True。条件的两部分都必须为 True，整个条件才会为 True。如果午餐不是比萨的话，那无论甜点是不是冰淇淋，这个条件都是 False。and 意味着只有当满足了条件的每部分时，整个条件才为 True。缺一不可
if 今天是星期日 or 今天学校放假	if 语句在什么情况下为 True 呢？这里的条件是用 or，而不是用 and 连接的。因此这两部分中的一个为 True，整个条件才为 True。如果今天是星期天，但学校没有放假，那么这个条件就为 True。同样，如果学校放假但不是星期天，条件也是 True。如果在学校放假的时候恰好是星期天呢？条件还是 True。使用 or 时，只要条件的一个部分为 True，那么整个条件就为 True。只有当所有部分都是 False 时，or 条件语句才为 False
if 今天是星期一 or 今天是星期二 or 今天是星期三	if 语句在什么情况下为 True 呢？这个条件包含三个部分，每个部分之间都有一个 or。如果今天是星期一，或星期二，或星期三的话，这个条件就为 True。只要满足了这三个部分中任何一个，那么整个条件就为 True

我们的代码中，两个判断都是在判断是否相等（也就是 == 左右两边的内容是否相同）。下表总结了可以使用的比较运算符。

运算符	描述
==	判断是否等于，如前所述
!=	判断是否不等于，也就是两边是否不一样，和 == 完全相反
>	判断是否大于。如果左边的值大于右边的值，则为 True
<	判断是否小于。如果左边的值小于右边的值，则为 True
>=	判断是否大于或等于。如果左边的值大于右边的值或者与右边的值相同，则为 True
<=	判断是否小于或等于。如果左边的值小于右边的值或者与右边的值相同，则为 True

不要混淆 and 和 or

很明显，在我们的例子中用 and 并不合适，因为某一天不可能既是星期六（5）又是星期天（6）。如果像以下代码这样用 and 的话：

```
if today.weekday() ==5 and today.weekday() ==6:
```

以上 if 语句的评估结果永远是 False，因为这个条件永远不可能成立。这就是为什么我们要用 or 的原因。在本章后面部分中，我们还将多次用到 or。

还有一些其他比较运算符，实际上，我们很快就会接触到其中之一。不过，表格中这些运算符的使用频率最高。

判断其他选项

总之，if 对一个条件进行判断，如果判断结果是 True，那么 if 下面缩进的代码行就会被执行。else 负责提供 if 语句为 False（不为 True）时要执行的代码。

要是还想判断其他条件呢？现在，我们的代码在星期六和星期日显示一段信息，在其他日子显示另一段信息。如果想在星期五时显示一条特别的信息，该怎么办呢？这时候该 elif（else if 的缩写）出场了。

和前面的代码相比，以下代码在 if 语句块和 else 语句块之间多了两行代码：

```
# Imports
import datetime

# Get today's date
today=datetime.datetime.now()

# Display the right message for different days of week
if today.weekday() == 5 or today.weekday() == 6:
    print("It's the weekend, no school today!")
    print("We can code all day long!")
elif today.weekday() == 4:
    # Display this on Friday
    print("It's Friday, tomorrow we'll have tons of time to code!")
else:
    print("It's a school day.")
```

保存代码并运行。现在，程序在星期六和星期日显示一种信息，在星期五显示另一种信息，在所有其他日子时显示第三种信息。

那么，这里发生了什么变化？首先，多了一行解释代码意图的注释。

但最重要的改变是新增了一条 elif 语句：

```
elif today.weekday() == 4:
```

这一行本质上是另一条 if 语句，但因为它是 elif 语句，所以只有在第一条 if 语句为 False 时才会被调用。这行代码判断 weekday() 是否返回 4，也就是星期五。如果判断结果为 True，那么就执行 elif 下缩进的代码行。

下面回顾一下 if、elif 和 else 的用途。

- 如果要用代码判断什么条件的话，总是先从 if 语句开始写起。
- 如果想进行额外的判断，可以使用 elif。elif 总是选择性使用的，既可以完全不添加 elif 语句，也可以添加很多，实际上，想添加多少就添加多少 elif 语句。

- 如果想让代码在 if 或 elif 判断都不为 True 的情况下执行，那么就可以使用 else。else 是可选的，并不是必须要有的。不必为 else 设置条件。else 语句只能有一条，并且它必须是序列中最后的条件语句。

使用 in

在展开下一个话题之前，我想先展示另一种判断是否为多个值之一的方法。请回顾第一条 if 语句：

```
if today.weekday() == 5 or today.weekday() == 6:
```

如大家所知，以上 if 语句判断两个条件，只要条件之一为 True，整条 if 语句就为 True。

这两个条件都是在判断 == 两边是否相等。首先检查 today.weekday() 是否为 5，然后检查 today.weekday() 是否为 6，都是在用 today.weekday() 与数值进行比较。因此，我们可以用另一种方式写这一条 if 语句。

请看以下 if 语句：

```
if today.weekday() in [5,6]:
```

第 3 章介绍过方括号 [ ] 的作用，它创建一个项目列表。以上代码创建了一个包含 5 和 6 两个值的列表。在 Python 中可以用一个名为 in 的特殊检测，如果要找的值在列表中，就会返回 True。所以，如果 today.weekday() 是 5 或 6，那么它就在（in）列表中，判断将返回 True。如果 today.weekday() 是任何其他值，那它就不在列表中，if 语句就会返回 False。

很不错，是吧？想尝试一下的话，将代码中的 if 语句替换成上面的语句即可。

总之，两种方法都可以完成这项任务。用 in 或 or 运算符都是可以的。按个人偏好来选择就是了。

战胜数学家

我们已经学会了写好一个程序所需要的一切知识。接下来要写一个程序，询问用户的生日，然后告诉他们那天是星期几。瞧，我们算得比数学家还快呢！

处理数字输入

不过，在开始编程之前，还需要明确一点。请看下面的代码：

```
year = input("What year were you born? ")
```

大家都明白以上代码有什么用。它要求用户输入一些信息，然后将用户的输入保存在一个名为 year 的变量中。

但是，问题来了。正如第 3 章所提到的那样，字符串和数字是两码事。input() 总是返回一个字符串。如果用户输入是 2011，那么 year 变量就是字符串 "2011"，但我们希望它是数字 2011，因为 datetime 想要的是数字，而不是字符串。

后面的章节中，我们要花更多时间研究数据类型以及如何在数据类型之间进行转换。现在，只需要知道有这么一个奇妙的函数 int()。把包含数字的字符串传给 int() 函数，它就会把这个数字作为真正的数字返回。因此，这行代码会存储字符串 "2011"：

```
 year="2011"
```

但下面这行代码则会将数字 2011 存入 year 变量中（字符串 "2011" 被转换成数字 2011）：

```
year=int("2011")
```

综合应用

现在，可以正式动手编程序了。新建一个名为 Birthday.py 的文件，并输入以下代码：

```
# Import
import datetime

#Get user input
year = input("What year were you born ?  ")
year = int(year)
month = input("What month were you born? ")
month = int(month)
day = input("What day were you born? ")
day = int(day)

#Build the date object
```

```
bday = datetime.datetime(year, month, day)

#Display the results
if bday.weekday() == 6:
    print("You were born on Sunday")
elif bday.weekday() == 5:
    print("You were born on Saturday")
elif bday.weekday() == 4:
    print("You were born on Friday")
elif bday.weekday() == 3:
    print("You were born on Thursday")
elif bday.weekday() == 2:
    print("You were born on Wednesday")
elif bday.weekday() == 1:
    print("You were born on Tuesday")
elif bday.weekday() == 0:
    print("You were born on Monday")
```

保存并运行该程序。它将提示输入出生年、月、日，然后显示出那天是星期几。

那么，这个程序是如何工作的呢？

我们需要用到 datetime 库，所以第一条语句是 `import datetime`。

接着，我们要求用户输入出生年、月、日。请看以下代码：

```
year = input("What year were you born？  ")
year = int(year)
```

第一行是 `input()`，它将提示用户输入年份，并将用户输入的内容作为字符串保存在变量 `year` 中。

第二行通过使用 `int()`，将字符串形式的年份转换为数字形式，并将其保存在同一变量中（用数字年份覆盖字符串 `year`）。

实际上，可以像下面这样把两行代码合并为一行：

```
year = int(input("What year were you born? "))
```

在这行代码中，`int()` 将整个 `input()` 函数包含在内，因此它转换了 `input()` 返回的内容，并将其保存到变量中。最终得到的结果和前面的两行代码得到的结果一样。

获得年、月和日后，就要用它们来创建 Python 日期了。前面，我们用如下代码创建了一个包含当前日期的 Python 日期：

```
today=datetime.datetime.now()
```

怎样才能用变量来新建一个日期呢？不是用 now()，而是将年、月、日的值传入 datetime，如以下代码所示：

```
bday = datetime.datetime(year, month, day)
```

这样，bday 变量中就包含随时可用的 Python 日期了。

接下来是 if 和 elif 语句，和前面的 if 和 elif 语句很相似：

```
if bday.weekday() == 6:
    print("You were born on Sunday")
elif bday.weekday() == 5:
    print("You were born on Saturday")
elif bday.weekday() == 4:
    print("You were born on Friday")
```

我们先检查星期天，然后是星期六，然后是星期五，以此类推。总结一下就是，首先是一条 if 语句（判断是否为星期天），然后是一系列 elif 语句（逐一判断其他日期）。这里就不再逐一细说了，聪明如你，想必早就已经明白了。

else 去哪里了？

这个 if 语句块中为什么没有 else？因为一个星期只有 7 天，weekday() 只可能返回 7 个数字。我们已经在 if 和 elif 语句块中处理了所有 7 个选项。我们当然可以添加一条 else 语句，但由于它永远不会被执行，所以也就没必要添加了。

请记住，else 永远是可以选择性添加的。

另一种解决方案

我还有最后一个想法。现在的代码中有 7 条 print() 语句，每个 if 和 elif 下都有一条。若是想简化为只用一条 print() 语句（这样就不用重复输入相同的文本了），可以像下面这样做（用以下代码来替换当前代码中的 if 语句块）：

```
#Display the results
if bday.weekday() == 6:
    dow="Sunday"
elif bday.weekday() == 5:
    dow="Saturday"
elif bday.weekday() == 4:
    dow="Friday"
elif bday.weekday() == 3:
    dow="Thursday"
```

```
elif bday.weekday() == 2:
    dow="Wednesday"
elif bday.weekday() == 1:
    dow="Tuesday"
elif bday.weekday() == 0:
    dow="Monday"

# Display the results
print("You were born on", dow)
```

编程的方式是多种多样的。在上面的版本中，`if` 语句和 `elif` 语句并不打印任何内容，而是都在名为 `dow`（day of week 的缩写）的变量中存储一个日期。然后，这个 `dow` 变量被传入一条 `print()` 语句中。最终结果是相同的，只是组织代码的方式略有不同。

小结

本章介绍了所有编程语言中最重要的一部分，`if` 语句是用来做决定的，下一章将继续研究它。

第 5 章

剪刀石头布

第 4 章讲解了如何用 `if` 语句创建条件，这是个非常重要的主题，因此本章将继续讨论，并利用学到的各种知识来创建一个剪刀石头布小游戏。

更多字符串

在前面几章中，我们已经用过很多很多字符串了。友情提示，字符串就是文本块。大家应该已经非常熟悉下面这样的代码了：

```
name="Ben"
print(name)
```

在这段代码中，name 是个变量，准确地说，它是个字符串。

既然我们已经接触过类（比如第 4 章中的 datetime 类），那么现在是时候告诉大家一个秘密了：name 字符串实际上也是一个类，它是 str 类。如果运行以下代码的话：

```
name="Ben"
print(type(name))
```

会看到 name 变量属于 str 类型（这是 Python 对字符串的称呼）。

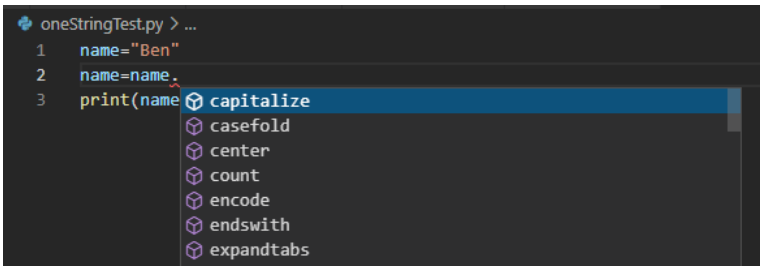
而且，正如大家所知，类有一些方法可供使用。

大家可以试着做接下来这件有趣的事。新建一个文件，命名为 oneStringTest.py，然后输入以下内容（大家要像我一样用自己的名字……除非和我名字一样，如果那样的话，你的名字真好听，一定得用）：

```
name="Ben"
name=name
print(name)
```

是的，中间那行代码并没有什么用：它把 name 设置成当前 name。

但是，一个小小的改动就会让这行代码变得有用起来。在 name=name 后面加一个句点 . 后稍等片刻，VS Code 会弹出下图这样的浮窗。



name 是个字符串，也就是 str 类，对吧？输入句点后，VS Code 会显示出所有可用的方法（记住，方法就是函数）。选择任何一个方法，将鼠标悬停在方法名上，就会出现与该方法对应的帮助浮窗。

下面来实践一下。把中间那行改成下面这样：

```
name=name.upper()
```

保存并运行代码。可以看到，name 中的值被转换成大写字母。这就是 upper() 方法的作用。相对地，lower() 方法可以将文本转换为小写字母。还有一个很有用的方法叫 strip()，可以用来移除文本前后的额外字符。

如果想把文本转换为大写字母并同时移除所有多余的空位，就可以用下面两个函数：

```
name=" Ben "
name=name.upper()
name=name.strip()
print(name)
```

上面的代码中，名字一开始是 Ben （前后有空格）。下一行把它变成了 BEN （空格仍然在）。第三行把空格移除了。

> 小贴士
>
> 移除空格（stripping whitespace）　实际上，有三种不同的方法可以移除多余的文本。rstrip() 从字符串右边（指文本末尾）移除多余的文本。lstrip() 从字符串左边（字符串的开头处）移除文本。strip() 差不多就是 rstrip() 和 lstrip() 的合体，用于删除字符串两端的空格。

实际上，Python 还允许我们叠加使用这些函数。以下代码的作用和前面的代码一样，只不过把两个方法合并到了一行中：

```
name=" Ben "
name=name.upper().strip()
print(name)
```

我们将在游戏中使用这些方法。

游戏时间

掌握了如何使用 if 语句后，是时候创建石头剪刀布游戏了。这通常是两个人玩儿的猜拳游戏，每人各出三种手势中的一种。石头克剪刀，剪刀克布，布克石头。我

们将创建的是这个游戏的计算机版本。我们和计算机将分别从三种手势中挑选一种，然后看看是谁赢了，也有可能是平局。

处理用户输入

这次要做的事情有些不同，不再是创建多个程序，而是创建一个程序，并利用前面学到的所有知识来逐步增加它的功能。

新建一个文件，命名为 rps.py（rps 就是石头布剪刀的英文首字母缩写）。先输入以下代码：

```
# Imports
import random

# Computer picks one
cChoice = random.choice("RPS")

# Get user choice
print("Rock, Paper, or Scissors?")
uChoice = input("Enter R, P, S: ")

#Test it
print("You:", uChoice)
print("Computer", cChoice)
```

保存并运行代码。程序将提示我们输入 R、P 或 S，然后显示我们和计算机的选择。

和我们在第 3 章中所做的练习类似，这段代码首先导入 random 库，然后用 `choice()` 方法从 R、P 或 S 这三个选项中随机挑选一个。

接着，我们用 `input()` 函数要求用户输入 R、P 或 S。

到这里为止，我们已经获取了计算机的（随机）选择以及用户的选择。

最后几行（从 `# Test it` 注释开始直到结尾）是暂时的，它们的作用是在进一步编写程序之前先测试一下前面的代码。确定前几行代码合法之后，可以删除这段测试代码。

像专业人士那样，随时进行测试

在工作过程中经常测试代码是个非常好的习惯。需要测试的代码比较少的时候，更容易发现和定位问题。确定一部分代码没有问题之后，就可以继续编写下部分代码了。所有专业程序员都是这么做的。

怎么样？代码能正常运行吗？用户需要输入三个字母中的一个，我们将在 `if` 语句中使用这些字母。代码要求用户输入 `R`，`P` 或者 `S`。如果用户输入小写的 s 或在字母后加了一个空格的话，`if` 语句就不能正常运行了。

我们知道怎么解决这个问题。在 `input()` 语句下方加上下面这行代码：

```
uChoice=uChoice.upper().strip()
```

另外，因为 Python 允许叠加使用方法，所以我们可以把 `input()` 代码行改成下面这样：

```
uChoice=input("Enter R, P, S: ").upper().strip()
```

最终结果都一样：`uChoice` 包含没有多余空格的大写文本。

再次测试代码。输入大写或小写文本，并添加一些空格，以确保代码能按照我们的意图运行。确定一切正常后，就可以删除测试代码了（从 `# Test it` 到最后一行）。

游戏的代码

现在要利用 `if` 语句来查看游戏赢家是谁。在文件底部添加下面的代码：

```
# Compare choices
if cChoice == uChoice:
    print("It's a tie!")
elif uChoice == "R" and cChoice == "P":
    print("You picked rock, computer picked paper. You lose.")
elif uChoice == "P" and cChoice == "R":
    print("You picked paper, computer picked rock. You win.")
elif uChoice == "R" and cChoice == "S":
    print("You picked rock, computer picked scissors. You win.")
elif uChoice == "S" and cChoice == "R":
    print("You picked scissors, computer picked rock. You lose.")
elif uChoice == "P" and cChoice == "S":
    print("You picked paper, computer picked scissors. You lose.")
elif uChoice == "S" and cChoice == "P":
    print("You picked scissors, computer picked paper. You win.")
else:
    print("Not very good at listening to instructions. Huh?")
```

保存代码并运行程序。每次游戏时，程序会比较我们和计算机的选择，并显示哪方赢了或是双方平了。

这段代码应该不难理解，不过我们还是从头到尾把它捋一遍吧。

第一条 if 语句检查用户的选择（uChoice）和计算机的选择（cChoice）是否相同。如果相同，那么就是平局。

然后是 6 条 elif 语句，负责检查每一个可能的选择组合。注意，这次的条件表达式使用了 and，而不是 or，因为每条 elif 语句都要测试两个选择是否匹配。

最后是 else 语句。只有当 if 或 elif 语句没有一条为 True 时，才会轮到 else 语句出场。唯一可能发生的情况是用户并没有输入 R、P 或 S 这三个字母之一，所以 else 语句打印的信息反映了这一点。

我们的游戏已经略具雏形了！

用户从来不遵循指示

代码要求用户输入 R、P 或 S，而我们在 if 语句块中加入了一条 else 语句，后者只有在用户输入 R、P 或 S 以外的东西时才会被执行。

这是个好习惯。因为用户往往不善于遵循指示。我们身为程序员，应该总是假设用户不会遵照指示，提前考虑到这一点，并写出能预见和处理这种情况的代码。这样一来，代码就不会因为用户的错误而中断了。

最后一次调整

游戏已经做好了。每次和计算机猜拳的时候，你都有相同的概率赢、输或和计算机打平。

环顾四周，确保没有人在看着你。我是认真的。现在，我们要变身为只顾牟取个人私利而篡改代码的超级大坏蛋了。准备好了吗？作为程序员，我们对应用程序有完全的控制权，而且，不是说这种行为应该被纵容，但我们可以利用这种控制权来为自己取得优势。

先从个性化的游戏体验开始。在 import 这行代码下，在计算机的选择之前添加这段代码：

```
# Ask the user for their name
name=input("What is your name?: ")
```

接着，为了显得更有个性，把获取用户输入的上一行改成下面这样：

```
print("Hello", name, "let's play Rock, Paper, or Scissors")
```

现在，运行游戏时，程序会询问用户的名字，并向他们问好。没错，我们就是为了提升用户的个性化游戏体验而特地添加代码的……（才怪）。

代码知道用户是谁后，就可以（咳咳）根据用户输入的内容来调整计算机的选择了。;-)

在获取用户输入的代码行之后，在 `# Compare choices`（比较选择）之前添加下面这段代码（要像我这样用自己的名字）：

```
# TOP SECRET CODE
if name == "Ben":
    if uChoice == "R":
        cChoice = "S"
    elif uChoice == "P":
        cChoice = "R"
    elif uChoice == "S":
        cChoice = "P"
```

保存并运行代码。现在的你将所向披靡。玩这个游戏（输入你的名字）的时候，你肯定每次都会赢，而其他人的胜率却只有三分之一。

那么，我们添加的代码实际上起着什么作用呢？首先是一条判断“你的”名字的 `if` 语句。如果输入的是你的名字（也就是说，你的名字被存储在 `name` 变量中），Python 就会运行缩进的代码。如果不是， Python 就会跳过这个 `if` 语句块中所有缩进的代码。

缩进的代码很有趣。它是另一条 `if` 语句，但它缩进了，所以被包含在第一条 `if` 语句里面。这是一条 `if` 语句中的 `if` 语句，程序员称之为嵌套 `if` 语句。`if` 和 `elif` 语句查看用户输入的内容（`uChoice`），然后，嗯，让我们把这种行为叫调整，它对计算机的选择（`cChoice`）进行调整，让计算机总是输给你。计算机一开始做出的选择仍然是随机的，只不过我们用另一个选择覆写了它。

和家人或朋友一起玩玩这个游戏，让他们看看你的运气有多好。

新术语

嵌套（nested）　当一条 `if` 语句位于另一条 `if` 语句内部时，我们就称之为嵌套（有点儿像俄罗斯套娃）。以后的章节要介绍其他可以嵌套的命令。

挑战 5.1

我们可不想让和自己同名同姓的人也占到便宜。这段代码该怎么修改才能实现只有我们自己才能作弊的目标呢？或许可以要求名字以特定的方式输入（比如全部小写），或者在结尾处加一两个空格。想出一个方案并修改 `if` 语句来检查它是否可行。

小结

`if` 语句非常重要，不使用它们的话，几乎不可能写出任何实用的代码。因此，本章讲解了很多使用 `if` 语句的例子，其中含有不同的判断和条件，这将把我们带到下一个话题：循环。

第 6 章

加密解密：for 循环

掌握了 if 语句后，就只剩下一个主要的知识点需要学习了。本章中，我们将开始探索循环。

列表

循环是至关重要的。接下来的几章将重点介绍不同类型的循环，以及它们的正确用法。

不过，在开始研究循环之前，我们需要先花些时间来回顾一种特殊的变量类型：列表。前面简要介绍过列表，比如第 3 章出现的这行代码：

```
choices=["Heads", "Tails"]
```

正如前面解释的那样，变量通常存储一个单一的值。列表是一种特殊类型的变量，它可以存储大量的值，也可以不存储任何值。

前面这行来自第 3 章的代码新建了一个名为 `choices` 的列表，其中包含两项，分别是 `"Heads"` 字符串和 `"Tails"` 字符串。

创建列表

那么，如何在 Python 中创建列表呢？嗯，我们知道，只需要简单地创建一个变量，就可以创建列表了。是什么让它成为列表的呢？只要把存储在变量中的值放入一对方括号 [ ]，就算是创建了一个列表。

因此，下面这行代码将创建一个空的列表：

```
animals=[]
```

`animals` 是个变量，也是个列表，但它是空的。以下代码创建了一个包含五个条目的列表：

```
animals=["ant","bat","cat","dog","eel"]
```

列表可以包含各种各样的东西

第 3 章中创建的列表是一个字符串列表。为了简单起见，我们在这里使用的例子都是字符串列表。但值得一提的是，Python 中的列表强大且灵活。它们可以存储数字、日期甚至列表（是的，可以创建列表的列表，我们将在本书的第 II 部分展开实践）。

列表中的各个条目必须全部放在方括号内，而且各个条目要用逗号来分隔开。

下面来动手实践吧。新建一个名为 List1.py 的文件并输入以下代码：

```
# Create a list
animals=[]
# How many items in it?
```

```
print(len(animals), "animals in list")
```

保存并运行代码，终端窗口中将显示“0 animals in list”，意思是列表中有 0 个动物。

这段代码是怎么运行的呢？"animals=[]" 创建了一个空的列表。print() 语句则负责显示列表中有多少个条目。为了做到这一点，我们使用了 len() 函数。len() 返回传递给它的任何条目的长度，比如列表中的条目数。因此，len(animals) 返回列表 animals 中的条目数。因为列表是空的，所以返回的是 0。

向列表中添加一些动物名称，如以下代码所示（可以用自己喜欢的动物，不一定要和我的列表一模一样）：

```
# Create a list
animals=["ant","bear","cat","dog","elephant"]
# How many items in it?
print(len(animals), "animals in list")
```

保存并运行更新后的代码。输出结果显示，列表中包含 5 种动物。

len() 函数

这里使用 len() 函数来获得列表中的条目数。但除了列表之外，len() 函数还能返回其他东西的长度。len() 的一个常见用途是用来获取字符串的长度。举例来说，len("hello") 将返回 5。

len() 与索引值的区别

不要把 len() 返回的值和索引值混淆。还记得吗？Python 是从 0 开始计数的，所以，一个有 5 个元素的列表，其索引值为 0 到 4。

初始化列表

和 Python 中的所有数据类型一样，列表实际上是一个 list 类，没错！运行下面的代码时：

```
animals=[]
```

在内部，Python 正在创建一个列表并在没有值的情况下初始化这个列表。Python 是通过使用 init() 方法完成的。我们自己也能这么做。创建空列表的那行代码也可以像下面这样写：

```
animals=list()
```

前面两行代码的作用完全一样。

访问

列表用来存储列表项，比如前面的 animal 列表。为了让列表发挥作用，我们需要能够访问存储在其中的条目。为此，我们要再次使用方括号，并在其中指定想获取的项目的编号。

下面来动手实践一下。新建文件并将其命名为 List2.py，然后输入以下代码：

```
# Create a list
animals= ["ant","bat","cat","dog","eel"]
# Display a list item
print(animals[1])
```

保存并运行代码。终端窗口中会显示什么动物的名字呢？一个条目在列表中的位置被称为索引。我们在 print() 函数中指定了索引 1，对应的是列表中的 bat。为什么呢？因为，众所周知，Python 总是从 0 开始计数，所以 ant 所在的位置是 0，而 bat 所在的位置是 1。

新术语

索引（index） 一个条目（举个例子，一个列表项）在列表中的位置就是它的索引。

一些用得比较少的索引

想从一个列表中返回一系列条目时，可以提供一个范围的起始值和结束值，用冒号 : 分隔。如以下代码所示：

```
print(animals[2:4])
```

这行代码将在终端窗口中打印 "['cat', 'dog']"，cat 的索引是 2（因为 Python 从 0 开始计数），dog 的索引是 3。一如既往地，结束值并不包含范围在内，2:4 意味着从 2 开始，并在 4 之前停止。

想从一个列表的末尾处开始计数的话，可以使用减号 -，如以下代码所示：

```
print(animals[-1])
```

这将显示什么呢？ -1 表示列表中的最后一个条目，-2 是倒数第二个条目，以此类推。因此，这段代码将显示“eel”，意为鳗鱼。

试着输入不同的索引值并运行这段代码。然后尝试输入一个超出范围的索引（比如 5，如果和我用的是同一个例子的话）。Python 不喜欢这种行为，它会提示“`list index out of range`”，意为列表索引超出范围，这意味着我们提供的索引是无效的。

修改

如你所见，索引可以用来引用列表中特定的条目。我们利用这个方法从列表中获取一个条目，但同样的语法其实也可以用来更新列表中的一个条目。

新建一个名为 List3.py 的文件，输入以下代码：

```
# Create a list
animals = ["ant","bat","cat","dog","eel"]
# Display the list
print(animals)
# Update item 2
animals[2] = "cow"
# Display the list
print(animals)
```

保存并运行代码。终端窗口中将显示两个 `animals` 列表，其中，第二个列表中的 `cat` 被替换为 `cow`。下面这行代码：

```
animals[2] = "cow"
```

将 `cow` 保存在 `animals` 列表的第三个位置（重申一下，Python 从 0 开始计数，所以 2 对应的是第三个条目）。它并没有增加新的条目，而是覆写了已有的值。

添加和删除

到目前为止，我们使用的所有列表都是用一组值来创建和初始化的。但如果需要临时添加或删除值的话，该怎么办呢？

先来看看如何添加一个条目吧。创建 List4.py，并输入以下代码：

```
# Create a list
animals = ["ant","bat","cat","dog","eel"]
# How big is the list?
print(len(animals), "animals in list")
# Add an item
animals.append("fox")
# Now how big is the list?
print(len(animals), "animals in list")
```

保存并运行。程序将报告列表中有 5 种动物，然后再报告列表中有 6 种动物。

为什么会这样呢？ animal 列表一开始确实只有 5 种动物，所以 print() 语句中的 len() 返回 5。

但请看接下来的这一行：

```
animals.append("fox")
```

append() 函数将一个条目添加到列表的末尾。所以下一条 print() 语句说有 6 个动物，因为 len() 函数返回的值就是 6。

将一个列表添加到另一个列表中

想在一个列表中追加多个条目时，可以选择多次调用 append() 函数，也可以选择使用 extend() 函数直接添加一个列表，如以下代码所示：

```
list2=["goat","hippopotamus","iguana"]
animals.extend(list2)
```

这段代码创建了第二个列表（也就是 list2），然后通过使用 extend() 函数将 list2 中的所有条目附加到 animals 列表的末尾。

我们经常发现，完成一项任务的方法远远不止一种，作为程序员，我们可以自由选择最适合自己的方案。

如何删除列表中的条目呢？有两个函数可供使用。如果知道想删除的条目的确切索引，可以用 pop() 函数，像下面这样：

```
animals.pop(5)
```

如果想根据条目的值来删除它，则可以这样：

```
animals.remove("fox")
```

查找

怎么检查一个值是否在列表中呢？方法有几种。

如果只是想知道列表中是否包含一个值，并不关心它在列表中的确切位置，那么使用 if 语句就可以了。

新建 List5.py 文件，并输入以下代码：

```
# Create a list
animals = ["ant","bat","cat","dog","eel","fox","goat"]
# Is "goat" in the list?
```

```
if "goat" in animals:
    # Yes it is
    print("Yes, goat is in the list.")
else:
    # No it isn't
    print("Nope, no goat, sorry.")
```

保存并运行。终端窗口将显示“Nope, no goat, sorry.”意为“抱歉，没有山羊。”将 "goat" 添加到第 2 行的 animals 列表中，然后再次运行该代码，这次会显示“Yes, goat is in the list.”意为“是的，山羊在列表中。”

前面的章节中出现过很多 if 语句。这里的 if 包含成员运算符 in，正如其名，如果列表中有 in 左边的值，则返回 True，否则返回 False。

若是想知道一个条目在列表中的确切位置，可以使用 index() 函数。将代码改成下面这样（只改动第 6 行的第一条 print() 语句）：

```
# Create a list
animals = ["ant","bat","cat","dog","eel","fox","goat"]
# Is "goat" in the list?
if "goat" in animals:
    # Yes it is
    print("Yes, goat is item", animals.index("goat"))
else:
    # No it isn't
    print("Nope, no goat, sorry.")
```

保存并运行代码。如果 animals 列表中没有 goat 的话，那么返回的结果就会和之前的一样。但如果有 goat 的话，程序将返回 goat 在列表中的位置，因为 animals.index("goat") 返回 goat 的索引。

排序

列表没有任何特定的顺序，列表以条目添加顺序来存储和显示条目。

本章中用到的所有列表都是按字母顺序排列的，但不是必须如此。这么做是为了更方便阅读和编写代码。

但要是需要给列表排序的话，该怎么做呢？比如，现在有以下代码：

```
# Create a list
animals= ["iguana","dog","bat","eel","goat","ant","cat"]
```

这个列表显然不是按字母顺序排列的。如果它有必要按字母排序呢？

尽管这并不是一个很好的例子，因为我们可以直接按字母顺序输入动物。但如果

这个列表不是硬编码的，而是由用户输入动物名称，然后我们得对列表中的用户输入进行排序，怎么办呢？

新术语

硬编码（hard coding） 当值（数字、文本、日期、各种内容）被直接输入到代码中时，我们就称之为“硬编码”。而且，原则上来讲，硬编码是非常不提倡的。

新建一个文件，命名为 List6.py，参考输入以下代码（可以随意向列表中添加更多动物，越多越好）：

```
# Create a list
animals= ["iguana","dog","bat","eel","goat","ant","cat"]
# Display the list
print(animals)
# Sort the list
animals.sort()
# Display the list
print(animals)
```

保存并运行代码。你会看到完整的 `animals` 列表显示了两次，第一次是按照它们放入列表的顺序显示，第二次则是按照字母顺序显示的。

神奇的是下面这一行：

```
animals.sort()
```

`sort()` 是一个函数，它的作用就是对列表进行排序。在默认情况下，它是按字母顺序排序的，但如果需要的话，也可以让它按降序排列。

很有意思吧？那么这一切和循环有什么关系呢？

只能对相同的类型进行排序

如前所述，我们可以对字符串列表进行排序。其实，还可以给数字列表排序，像下面这样：

```
[98,1,65,43,1]
```

但混合数据类型的列表不能排序。举个例子，如果试图对下面的列表进行排序：

```
[98,"car",1,65,"plane",43,1,"boat"]
```

Python 不知道该怎么排序，所以会显示错误信息。

列表函数会直接修改列表

注意到列表函数有一些有趣和不同之处了吗？它们的表现和之前用过的函数略有不同。为什么呢？请看下面的代码：

```
name="Shmuel"
name.upper()
```

这段代码有什么作用呢？它新建了一个名为 name 的变量，然后建了 name 变量的大写字母版本，对吧？但是 upper() 返回的是 name 的大写副本，并没有真正改变 name 变量。不信你自己测试一下。输入上面的代码，然后再加上 print(name)。可以看到，name 没有什么变化。如果真的想把 name 转换为大写字母，则需要把 upper() 返回的内容保存在 name 变量中，覆写原来的内容，如以下代码所示：

```
name="Shmuel"
name=name.upper()
```

列表函数不是这样工作的。比如，animals.append() 真的会在 animals 列表中添加一个值。

这个区别非常重要，请务必记牢。

其他实用函数

如你所见，len() 会显示一个列表中有多少个项目。想要知道列表中某个项目有多少时，可以使用 count() 函数。举个例子，想知道列表中有几个 cow 时，可以输入 animals.count("cow")，然后就能知道列表中有多少个 cow 了。

需要复制列表时，可以使用 copy() 函数。需要在列表的中间位置插入一个项目时（将所有后面的项目往后移一位），可以使用 insert() 函数。

之后的章节会讲解这些函数的相关例子。

循环

现在，我们已经知道 Python 是逐行执行代码的。它从文件的顶部开始，略过注释，按照顺序处理每行代码。第 4 章介绍了 if 语句，它有效地处理过程中包含或排除的各行代码。

但是，怎样才能不断地重复一个代码块呢？假设我们正在编写一个游戏，需要反复地移动，直到碰到障碍物为止。或者我们想让用户反复地自拍并发送信息，直到结束聊天会话。又或者我们想要开发一个简单的计算器，并允许用户在单击计算按钮之前反复输入数字。

所有这些例子都有一个共同点，一个功能可以反复使用，直到过程结束为止。

实现这个目的需要使用循环，Python 支持两种类型的循环。

- 可以循环一组定义好的选项。比如从 1 到 10 的循环，或者循环显示一组上传的图像，又或者是循环当前正在阅读的文件中的段落。在这种类型的循环中，迭代次数（即循环的次数）是有限的。对一组选项进行循环时，一旦循环完最后一个选项，循环就会结束。本章将重点讨论这种类型的循环。
- 也可以循环到一个条件发生变化为止。例如，允许玩家在游戏中移动，直到玩家的角色死亡为止，只要条件（角色仍然活着）为 `True`，循环就会一直重复，而当条件不为 `True` 时（角色死翘翘）就会结束。或者允许用户反复自拍，直到他们单击发送为止，只要条件（用户还没有单击发送）为 `True`，循环就允许用户使用相机并拍照；单击发送后，条件就变成 `False`，循环也就结束了。在这种类型的循环中，迭代的次数是未知的。循环只是不断地进行重复下去，直到条件发生变化。下一章将重点研究这种类型的循环。

遍历

从最简单的循环开始吧。试着在以下列表中进行循环。

```
animals=["ant","bat","cat","dog","eel"]
```

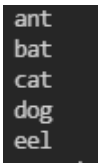
我们知道，这是一个名为 `animal` 的列表，其中有 5 个条目。

本章的前面部分介绍过访问单个列表条目的方法。但如果想循环浏览这个列表，并单独打印每个条目的话，该怎么做？嗒哒！下面有请循环闪亮登场！

新建一个文件，命名为 Loop1.py，并输入以下代码：

```
# List of animals
animals=["ant","bat","cat","dog","eel"]
# Loop through the list
for animal in animals:
    # Display one per loop iteration
    print(animal)
```

保存并运行代码。应该可以看到下面这样的输出：



我们已经知道第一行代码的作用了，接下来看一看循环部分的代码。如下所示：

```
for animal in animals:
```

这行代码让 Python 循环处理 `animals` 列表，并在每次迭代时，将一个项目放入名为 `animal` 的变量中。注意，这里有两个变量：`animal` 和 `animals`。`animals` 是我们新建的列表，`animal` 是什么呢？我们没有单独创建过 `animal` 变量，而是交给 `for` 循环代劳了。循环在每次迭代时都会自动改变 `animal` 的值。我们通过指定 `for animal` 来告诉 `for` 循环为这个变量起什么名字给这个变量起什么名字都可以，但对于一个从 `animal` 列表中存储动物的变量来说，`animal` 这个名字似乎是个不错的选择。

> 新术语
>
> 迭代（iteration）　一个循环的每个周期称为“迭代”。程序员说“代码在迭代”时，意味着它在循环。

和 `if` 语句一样，循环以冒号 : 结尾，在循环下面缩进的代码就是每次迭代时都会被调用一次的代码。

那么，在前面的例子中，`print()` 语句被调用了多少次？答案是 5 次，因为 `animal` 列表中有 5 个条目。试着添加一个或多个项目，然后再次运行代码。缩进的代码将被列表中的每个条目调用一次。

和 `if` 语句一样，需要留心循环下的缩进。如果把缩进弄错了，程序就不能如期工作了。举个例子，请看下面的 `for` 循环：

```
# Loop through the list
for animal in animals:
    # Display one per loop iteration
    print("Here is the next animal:")
print(animal)
```

得到的输出如下图所示：

```
Here is the next animal:
Here is the next animal:
Here is the next animal:
Here is the next animal:
Here is the next animal:
eel
```

为什么会是这样呢？缩进的 `print()` 语句在每个迭代中被调用一次，所以“`Here is the next animal:`”重复了 5 次。但是最后一条 `print()` 语句没有缩进（程序员会说它在循环之外），所以它在循环结束之前不会被处理。因此，最后一条 `print()` 只被调用一次，而 `animal` 变量将包含最后被放入的值（列表中的最后一个条目）。

列表能有多小？

当对一个列表进行循环时，最少能迭代多少次？答案是 0 次。如果列表是空的，那么缩进的代码根本不会被调用。

什么时候会用到空的列表呢？之后的章节中会讲解相关的例子。

循环处理数字

接下来看数字循环。和刚才的列表循环有些相似，但这次不是在列表中循环，而是指定一组数字后对它们进行循环处理。

创建 Loop2.py 文件，并输入以下代码：

```
# Loop from 1 to 10
for i in range(1, 11):
    # Display i in each loop iteration
    print(i)
```

保存并运行代码，终端窗口中会逐行显示数字 1 到 10。

`range()` 指定了数字的范围，和第 3 章中的 `randrange()` 一样，结束值不包含在内，所以 `range(1, 11)` 意味着从 1 开始，在 11 之前停止。

`for i` 创建了一个名为 `i` 的变量，在循环中，`i` 包含着范围内的下一个条目。`i` 在第一次迭代中是 1，第二次迭代中是 2，然后是 3，依此类推。

试着改变范围值并运行代码。多试几次。

挑战 6.1

range() 接受选择性的第三个参数：步长（step）。如果指定 range(1, 11, 2)，循环计数器将每次增加 2，因此循环将运行 5 次而不是 10 次（输出 1，3，5，7 和 9）。试着创建一个循环，显示数字 10、20、30，最后直到 100。

嵌套循环

现在来增加一些趣味性。在第 4 章末尾，我们探索了嵌套 if 语句，也就是一条 if 语句包含在另一条 if 语句中。循环也是可以嵌套的。

接下来的例子或许会让你回想起小学时候的美好时光。还记得乘法口诀表吗？很有趣，对吧？你当时可能花了些时间才能把乘法口诀表倒背如流的，但只需要短短的三行代码，就能让 Python 展示 12×12 的乘法口诀表！

循环嵌套

可以在循环中嵌套循环，用 if 语句嵌套 if 语句，在 if 语句中嵌套循环，在循环中嵌套 if 语句，还可以在嵌套中嵌套。不过，嵌套得太多可能会加大阅读和维护代码的难度。

创建一个名为 Loop3.py 的文件并输入以下代码：

```
# Loop from 1 to 12
for i in range(1, 13):
    # Loop from 1 to 12 within each iteration of the outer loop
    for j in range(1, 13):
        # Print both loop values and multiple them
        print(i, "x", j, "=", i*j)
```

保存并运行代码。可以看到终端窗口中很快显示出 144 行输出，从 1x1 = 1 开始，一直到 12x12 = 144。

那么这段代码是如何工作的呢？这里有两个循环，为了加以区分，我们分别称之为外循环和内循环。

外循环使用的范围是 range(1, 13)，所以它下面的所有内容都被调用了 12 次，每一次，i 变量都包含当前外循环的迭代数（首先是 1，然后是 2，以此类推）。

内循环使用的范围也是 `range(1, 13)`，它下面缩进的内容都被调用了 12 次，每一次，`j` 变量都包含当前内循环的迭代数。

那么 `print()` 语句被调用了多少次呢？外循环让内循环执行了 12 次。而内循环每次被执行时，它都会执行 12 次 `print()` 语句。就这样，`print()` 语句总共被执行了 144 次（12x12）。

`print()` 语句本身非常简单：

```
print(i, "x", j, "=", i*j)
```

第一次运行时，`i` 是 1，`j` 也是 1，所以 `print()` 语句实际上是下面这样的：

```
print(1,"x", 1, "=", 1*1)
```

`print()` 快速进行数学运算，并显示 `1x1 = 1`。

第二轮循环时，`i` 将是 1，`j` 将是 2，所以输出将是 1x2 = 2，如此循环往复，直到 j 是 12，显示的文本是 1x12 = 12。

然后，内循环就结束了，外循环将第二次重启内循环。这次的 `i` 将是 2。也就是说，在第二次迭代中，将从 2x1 = 1 开始输出，一直到 `i` 是 12，`j` 是 12，输出是 12x12 = 144。

而要显示这一切，只需要三行代码（是的，我知道上面有六行，但其中有三行是注释。Python 忽略了它们，所以我们也可以忽略它们！）

破解代码

知道如何使用循环后，是时候创建能加密和解密文本的程序了。没错，我们将创建两个程序。第一个程序将要求用户提供一些文本，并显示文本的加密版本。另一个程序将要求用户提供加密的文本，然后解密并显示原始文本。只要在加密和解密时使用的是相同的密钥（稍后再细讲），一切都能顺利进行。

因此，如果收到这样的文本：

```
Fphjsp#jw!hxrm%
```

我们就可以解密它，这段文本隐含的真正信息是……（抱歉，现在还不能告诉你！不过，你马上就知道了。）

友情提示，接下来的代码看起来可能会很复杂。但别怕，代码中用到的都是我们前面已经学过的。准备好了吗？

编程与加密

严格来说，我们接下来要做的并不是真正的加密。真正的加密对我们简单的需求而言有些复杂，所以我们要做的是加密（encoding），也就是用密码替换明文。我们还要使用一个密钥来使其更难被破译。

Python 非常适合用来进行真正的加密，有很多更高级的库可以用。

加密字符

将字符替换成加密版本需要进行一些数学运算。但字母又不是数字，怎样对它们进行数学运算呢？

实际上，在计算机中，字母确实是数字。每个字母和字符都有一个内部编号。这通常和我们无关。对我们而言，字母 A 就是 A，B 就是 B，而 3 就是 3。但在计算机里，显示为 A 的字母是 65，B 是 98，3 是 51。每个字符都有自己对应的编码，a 和 A 有不同的编码，因为它们是不同的字符。

我知道，这听起来有点奇怪，但现在只要求大家知道有这么回事儿就行了。每个字符都有一个唯一对应的编码（美国信息交换标准代码，也称 ASCII 码）。

小贴士

使用测试文件　每个程序员都有几十个测试文件（通常命名为 test42.py）。有时，程序员会有个用于存储测试文件的测试文件夹（是的，文件夹通常命名为 test）。测试文件非常适合用来进行修改和尝试。

在 Python 中，可以用 `ord()` 函数获得任何字符的 ASCII 码。在测试文件中，运行下面的代码：

```
print(ord('A'))
```

终端窗口将显示 65，这就是 A 的 ASCII 代码。把 A 改成 B，就会显示 66，以此类推。

ASCII 字符编程

ASCII（发音同“ASS-key”）是美国信息交换标准代码的缩写。它是一种电子信息交换的字符编程标准，比互联网和所有现代设备的概念久远得多。

ord() 返回一个数字，所以可以用它来做数学运算。下面这个例子算不上特别实用，但请看下面的代码：

```
print(ord('A') + 1)
```

终端窗口将显示 66。ord('A') 返回 65，程序又加了 1，最终得到 66。

那么，怎么把这个数字变回字符呢？和 ord() 函数相对应的是 chr() 函数。下面的代码将显示字母 A：

```
print(chr(65))
```

所以，如果想用 A 加 1 得到 B 的话，可以像下面这样：

```
print(chr(ord('A')+1))
```

这行代码有什么作用呢？最简单的解读方法是先看内部在看外部。ord('A') 返回 65，如前所示。65 加 1 得到 66。然后 66 被传递给 chr()，最终返回 B。

我们可以用这个技术来加密文本。想要加密 HELLO 的话，我们只需要知道要给每个字符的 ASCII 码加上多少（或减去多少）即可。如果加 10，那么 HELLO 就会被加密为 ROVVY（H 变成 R，E 变成 O，以此类推）。反之，用后者减去 10，就解密得到了 HELLO。

取模

在刚才的例子中，10 是加密密钥。我们就是用它来更改每个字母的。

但用这样一个简单的数字进行加密并不是很安全。用户可以先尝试 1，再尝试 2，再尝试 3，然后最终猜出真正的代码。为了使之更具安全性，最好使用其他密钥。

假设密钥是 314159。为了加密 HELLO，我们给 H 的 ASCII 码加上 3，给 E 加上 1，给第一个 L 加上 4，给第二个 L 加上 1，然后给 O 加上 5。这样，HELLO 就变成了 KFPMT。这种密钥很难被猜出来，因为每个字母使用的密钥都不同。

密钥越复杂越好

这种加密手段可以通过寻找模式和重复来破解。简单的密钥会导致大量的重复，复杂的密钥则不然。所以，密钥越复杂，就越难被破解。拿出小本本记下来！

但如果文本比密钥长呢？假设我们要加密 `Hello World` 这个文本，需要使用 11 位数的密钥（因为空格也有 ASCII 码），然而，现在的密钥却只有 6 位数。在这种情况下，该怎么做呢？

答案是重复使用密钥。如果密钥是 6 位数，我们就用这 6 位数来先加密前 6 个字母，然后再从头开始使用密钥。也就是对第 7 个字母使用密钥的第一个数字，对第 8 个字母使用密钥中的第二个数字，并根据需要不断地重复使用密钥。

如何计算出要使用哪位数字呢？答案是用除法并查看余数。举个例子，想找出第 8 个字符（我们需要密钥中的第二位数字）对应的密钥时，只需用字符索引（8，因为这是第 8 个字符）除以密钥长度（6），就能得到余数 2。 第 3 章介绍过取模运算符 `%`，可以用它来找到余数。具体用法如下：

```
print(8%6)
```

`8` 除以 `6`，余数是 `2`。

利用取模，我们总是可以用字符位置（8 代表第 8 个字符，42 代表第 42 个字符，以此类推）除以密钥的长度，得到的余数就是要使用的密钥对应的位数。

加密代码

好了，新建一个文件，命名为 Encrypt.py，然后输入以下代码：

```
# ASCII range of usable characters - anything out of
# this range could throw errors
asciiMin = 32   # Represents the space character - " "
asciiMax = 126  # Represents the tilde character - "~"
# Secret key
key = 314159    # Top secret! This is the encryption key!
key = str(key)  # Convert to string so can access individual digits
# Get input message
message = input("Enter message to be encrypted: ")
# Initialize variable for encrypted message
messEncr = ""
# Loop through message
for index in range(0, len(message)):
    # Get the ASCII value for this character
    char = ord(message[index])
    # Is this character out of range?
    if char < asciiMin or char > asciiMax:
        # Yes, not safe to encrypt, leave as is
        messEncr += message[index]
```

```
    else:
        # Safe to encrypt this character
        # Encrypt and shift the value as per the key
        ascNum = char + int(key[index % len(key)])
        # If shifted past range, cycle back to the beginning of the range
        if ascNum > asciiMax:
            ascNum -= (asciiMax - asciiMin)
        # Convert to a character and add to output
        messEncr = messEncr + chr(ascNum)
# Display result
print("Encrypted message:", messEncr)
```

保存并运行代码。程序将要求你输入一条要加密的信息，然后显示出加密后的信息。至于解密，则是下一部分要讨论的主题。

那么，这段代码是如何工作的呢？

不是所有 ASCII 字符都能被好好打印出来的，因此为了保险起见，我们需要定义想使用的字符范围，像下面这样：

```
asciiMin = 32   # Represents the space character - " "
asciiMax = 126  # Represents the tilde character - "~"
```

接下来设置密钥：

```
# Secret key
key = 314159    # Top secret! This is the encryption key!
key = str(key)  # Convert to string so can access individual digits
```

key 是数字式的加密密钥，我的密钥是六位数的，不过可以自由设置。这里的关键在于，必须用同样的密钥来加密和解密文本。

我们需要单独使用密钥中的每位数字，记得吗？我们就是在这样用不同密钥数字加密每个字符的。为了做到这一点，需要先把密钥转换为字符串，于是，数字 314159 就变成了字符串 "314159"。从字符串中获取字符就简单多了，就像是从列表中获取列表中的条目一样。

接下来，程序用 input() 函数从用户处那里获取要进行加密的文本，我们已经非常熟悉这个函数了。

接着是下面这行代码：

```
messEncr = ""
```

这将创建一个名为 messEncr（加密信息的英文缩写）的空字符串变量。代码将逐个加密文本的字符，同时把加密后的字符添加到 messEncr 变量中。

str() 和 int()

str() 与第 4 章中的 int() 函数相反。int() 将字符串转换为数字，str() 将数字转换为字符串。我们将在第 7 章中更详细地研究 int() 函数。

然后，开始循环处理信息：

```
for index in range(0, len(message)):
```

这里使用了一个 for 循环，从 0 到文本的长度的范围内进行循环。这个循环是怎么知道文本有多长的呢？答案是通过 len() 函数。如果文本是 10 个字符，len() 就返回 10，所以循环的范围就是 range(0, 10)，也就是说从 0 到 9 循环，正如我们所期望的那样。在每次迭代中，index 变量将包含迭代次数：0 是第一次，1 是第二次，以此类推。

在 for 语句下缩进的代码将对每个要加密的字符都执行一次。在每个循环开始时，我们需要得到待处理字母所对应的 ASCII 码，如以下代码所示：

```
char = ord(message[index])
```

message[index] 让我们访问一个单一的字符。在第一个循环中，index 是 0，因此第一次迭代时，message[index] 将返回第一个字符。第二次迭代中，则返回第二个字符，以此类推。ord() 获得的数字 ASCII 码被保存在 char 变量中。

第二条 if 语句检查这个字符的 ASCII 码是否在安全范围内。如果不在，就不对它进行编程。如果在，就执行以下这段代码：

```
ascNum = char + int(key[index % len(key)])
```

这行代码就是真正负责加密的。index 是当前的字符数（由 for 循环设置）。index % len(key) 用 index 除以密钥的长度，得到一个余数，也就是要使用的密钥数位。对应的密钥将和 char（当前 ASCII 码）相加，得到的结果存储到 ascNum 中。举个例子，如果循环目前的索引是 9，而密钥是 6 位数，index % len(key) 将变成 9 % 6，返回余数 3，密钥的第 3 位数字将被使用。

编码后得到的字符会附加到 messEncr 上，如下所示：

```
messEncr = messEncr + chr(ascNum)
```

chr(ascNum) 将计算并编码后得到的字符转换为一个字符串，并将其追加到 essEncr 中（请记住，字符串相加的意思是拼接）。

正如前面提到的那样，一些 ASCII 字符无法打印出来，因此我们需要把这些字符

排除在外。下面这段代码检查编码后得到的字符是否在安全范围内，如果不在的话，就将其转换成安全范围内的值：

```
if ascNum > asciiMax:
    ascNum -= (asciiMax - asciiMin)
```

最后是用来显示加密的文本的 `print()` 语句。

```
print("Encrypted message:", messEncr)
```

这样就搞定了。输入文本，程序将使用密钥中的数字对其进行加密。给别人发送一条加密信息后，他们需要匹配的密钥才能解读它。可以为不同的人设置不同的密钥。

解密代码

很好。但怎样才能对加密的信息进行解密呢？实际上，这个过程和加密代码的过程差不多，只需要在 Encrypt.py 文件的基础上做一些小小的改动。

小贴士

使用另存为 使用 VS Code 的“另存为”选项（在“文件”菜单中）来将 Encrypt.py 另存为 Decrypt.py，就会有两个完全相同的文件了。试试吧！

打开 Decrypt.py，我们将对它进行稍作改动。首先，把 `input()` 的提示语句改为解密：

```
message = input("Enter message to be decrypted: ")
```

接下来，找到下面这行负责加密的代码：

```
ascNum = char + int(key[index % len(key)])
```

回顾一下，我们在加密文本时，加上了一个密钥数字。在解密时，减去相同的数字就可以了。因此，把代码改成下面这样：

```
ascNum = char - int(key[index % len(key)])
```

就这样，加号改成了减号。

接下来看刚刚那行代码下面的 `if` 语句。它负责检查 ASCII 码是否超出安全范围，如果有必要，就将其转换为安全范围内的值。我们需要把情况反过来，如以下代码所示：

```
if ascNum < asciiMin:
    ascNum += (asciiMax - asciiMin)
```

if 语句中的小于号 > 被改成了大于号 <，赋值语句中的 -= 变成了 +=。现在不用担心解码过程中会产生一个低于安全范围的数字了。

记得把需要修改的注释改掉。

完美搞定！现在可以加密和解密信息了，只要双方都有相同的密钥。所有这些功能都是通过几个简单的 for 循环实现的。

使用密钥 314159，能解密本节一开始提到的那串加密文本吗？

挑战 6.2

如你所见，Encrypt.py 和 Decrypt.py 大同小异。事实上，应该把它们整合为同一个程序。之所以分成两个文件，是为了使代码看上去简单一点儿。

但这可以改进。尝试创建一个既能加密又能解密的程序。它需要询问用户要执行哪种操作，像下面这样：

```
action = input("Encrypt or decrypt? Enter E or D: ")
```

然后，可以用 if 语句根据 action 是 E 还是 D 来选择程序是要加密还是解密。

小结

本章讲解了如何用 for 循环来循环处理已知的项目。下一章将研究条件循环以及如何结合使用这两种循环。