Python Workout
50 ten-minute exercises

每天10分钟学会Python

50次练习掌握一门语言

[美] Reuven M. Lerner 著
苏丹 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书以 “10 分钟集训” 为核心设计，旨在通过 50 个针对性练习，帮助读者高效掌握 Python。从数值类型、字符串等基础知识，到迭代器、生成器等进阶知识，本书将 Python 核心知识拆解为 10 章。为跳出传统教学模式，聚焦实战，本书设计的每个练习都由问题、解题思路、解答、解答视频、扩展练习等五部分组成，同时配套了代码资源。注意，扩展练习旨在帮助读者比较不同的解决方案、锻炼思维、内化知识，补充说明则指出了 Python 编程的常见问题，旨在帮助读者规避常见错误。零基础或基础薄弱的 Python 新用户与独立开发者，通过对本书练习的短时集训即可提升代码实战能力，在编程技能上实现质的飞跃。

版权贸易合同登记号 图字：01-2020-4040

图书在版编目（CIP）数据

每天 10 分钟学会 Python ： 50 次练习掌握一门语言 /（美）鲁文·勒纳（Reuven M. Lerner）著 ； 苏丹译 . 北京 ： 电子工业出版社 ， 2026. 1. -- ISBN 978-7-121-51461-6

Ⅰ . TP312.8

中国国家版本馆 CIP 数据核字第 2025FD4496 号

责任编辑：陈　丽
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　印张：15.75　字数：302.4 千字
版　　次：2026 年 1 月第 1 版
印　　次：2026 年 1 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：faq@phei.com.cn。

献给我的三个孩子，他们是我最好的老师

阿塔拉 · 玛格丽特

西柯玛 · 布鲁妮娅

阿莫兹 · 大卫

前言

在许多方面，学习编程语言都像学习一门（人类的）外语。你可以完成课程学习，理解阅读材料，甚至在期末考试中取得好成绩，但是到了实际使用这门语言的时候，你会发现自己很慌张，不确定该用什么语法，找不到最合适的表达方式——更别提自己根本听不懂那些母语人士在说什么了。

这就是练习的意义所在。练习外语，会让你讲得越来越流畅和自信，使你能够参与到更有深度、更有趣的对话中去。练习 Python 会让你更快、更容易地解决问题，同时能写出可读性更强、可维护性更佳的代码。这种进步是随着时间的推移逐步发生的，因为你总是在解决新的、各种不同场景下的问题时使用这门语言，所以你的进步往往不容易被看出来。然而，当你回顾几个月前的自己是怎样使用这门语言的时候，差别就很显著了。

这本书并不是要教你如何学 Python，而是让你熟练使用 Python，为此提供必需的练习。在完成本书的练习后——而不是跳过问题偷看答案——你将写出可读性更强、更符合语言习惯、可维护性更高的 Python 代码。

本书是我在面向企业的 Python 培训班上与学生沟通、交流的成果。每次课程一结束，他们总是会问，在哪里能找到更多有助于进一步提升技能的练习。本书大量借鉴了我为他们设计的动手实验，以及我在课上和课后跟他们进行的讨论。

这些练习旨在帮助你内化 Python 的一些核心思想：核心数据结构、函数、解析、面向对象编程，以及迭代器。这些课题看上去可能不难，或许对一本习题书来说甚至过于简单。但是一切 Python 程序，从最庞大的应用到最小巧的脚本，都是基于这些基础构件的。充分了解它们，是成为一名熟练的 Python 开发工程师的关键。我经常说，忽视这些基础构件，转而去探究更复杂的课题，就好比一名化学系的学生跳过学习化学元素，转而去研究所谓“真正”的化学制品。

作为一名 Python 讲师，同时作为一名学生，我的亲身经历可以证实练习的力量。近年来，我一直在学习中文，很大程度上是由于我每隔几个月就要去中国教授 Python 课程。我

上的每节中文课，做的每道习题，对我提升中文口语流利程度的帮助似乎都微乎其微。但是，每当我在离开几个月后再次回到中国时，我就会发现这些练习确实有帮助，让我能更轻松地与当地人交流。

我的中文仍然远没达到流利的水平，但我在不断进步。回顾过去，我对发现自己已经走了这么远感到很开心。我希望本书能为你做同样的事情，让你每天都能增进对 Python 的理解、提升使用 Python 的熟练程度。

致谢

写书是团队协作的成果。这种说法可能是陈词滥调，但事实的确如此。所以我想对一些人表示感谢和认可，没有他们就不可能有这本书。

我想感谢多年来我有幸在企业 Python 培训课程中教授过的数千名学生。多亏了他们提出的问题、建议、洞见和纠错，才有了现在这些练习，以及针对练习的解决方案与详细阐释。

还要感谢我在 *Better Developers* 邮件周刊（参见链接 1）上的众多订阅者，他们经常对我撰写的文章进行评论和纠错，我从他们那里学到了很多，并且经常将这些见解运用到教学中。

接下来，感谢 Philip Guo（参见链接 2），他是加利福尼亚大学圣地亚哥分校认知科学系的助理教授，同时是 Python Tutor 网站的作者和维护者。他的网站是极其有价值的工具，我在授课时经常使用，并且我也鼓励我的学生在解决他们的 Python 代码问题时使用。我在本书中使用了很多来自 Python Tutor 的截图，而且几乎在每个练习的解答中都附带了一个指向该网站的链接，这样你就可以自己去逐行运行和研究代码了。

感谢每一个从事 Python 相关工作的人，从核心开发者，到为这门语言撰写博客的作者，再到软件包贡献者。Python 生态系统是一项令人惊叹的技术成就，但更打动我的是我有幸结识的这些人——他们以体面和热情为这项成就提供了真正的帮助。

Manning 出版社的许多人都为本书做出了贡献，使它远比我自己能做到的更好。（证据是这本书的前身，即自行出版的版本，远远不如你正在读的这本好！）我曾跟他们中的一些人密切合作，他们的专业能力和耐心加速了本书的诞生。Michael Stephens 注意到这本以练习为主的书，看好它的出版前景，鼓励我与 Manning 出版社合作。Frances Lefkowitz 的文

字编辑技能十分娴熟，指出那些可以进一步修改、拆分或者插入图片的内容，在整个写书的过程中，她为我引路。Gary Hubbard 和 Ignacio Beltran Torres 在技术方面提供了无数的见解与修改意见，帮助我找出缺陷，删减糟糕的解释文字。Carl Quesnel 对最终文本的详细编辑给我留下了深刻的印象。

致所有的审稿人：Annette Dewind、Bill Bailey、Charles Daniels、Christoffer Fink、David Krief、David Moravec、David R. Snyder、Gary Hubbard、Geoff Craig、Glen Sirakavit、Jean-François Morin、Jeff Smith、Jens Christian B. Madsen、Jim Amrhein、Joe Justesen、Kieran Coote-Dinh、Mark Elston、Mayur Patil、Meredith Godar、Stefan Trost、Steve Love、Sushant Bhosale、Tamara L. Fultz、Tony Holdroyd，以及 Warren Myers。他们的建议让本书变得更好。

最后，我的家人一直都对我的事业和学术生涯很有耐心。在我发展培训业务、获授博士学位，并且开始游历世界教授 Python 时，他们给予了帮助和理解。关于这本书，他们曾两次展示了非凡的耐心：第一次是我在自己的网站上自行出版时；第二次是本书经过升级、扩展和显著的改进后，变成了你正在阅读的版本。感谢我的妻子 Shira，我的孩子 Atara、Shikma 和 Amotz，感谢他们的理解和欣赏。

关于本书

尽管我希望且相信你会在阅读过程中学到很多 Python 的相关知识，但写作本书的本意并不是教你如何使用 Python 语言，而是帮助你提升对 Python 的理解，提高利用它来解决实际问题的能力。你可以将其看作一本工作手册，它的力量和潜能有多大，完全取决于你。你在使用本书时下的功夫越多，就会得到越多的收获。

换言之，这不应该仅仅是一本你选择粗读或翻一翻的书，要想学到东西，你必须花时间解答问题，然后犯一些难以避免的错误。阅读别人的解决方案，跟写出自己的解决方案，二者有天壤之别。我希望你能在解答问题上花些时间，我保证这是一笔未来会给你带来丰厚回报的投资。

当你完成了本书的练习时，你应该已经解决了许多关于核心数据结构、函数、解析、模块、对象和迭代器的问题，理解了如何有效、规范地使用 Python。一旦完成了这些练习，你就会发现为工作或兴趣设计、编写 Python 程序变得容易了许多。

注意，在 Python 文档甚至 Stack Overflow（参见链接 3）这类网站中寻求帮助并不是作弊的行为。没有哪个开发者一定能够记住他们在日常工作中需要的所有知识点。不过，我确实希望随着你在本书中不断取得进展，以及在工作中不断使用 Python，你能越来越少地求助于这些文档，或者只需要在一些进阶的主题上寻求帮助。

谁应该阅读本书

本书的目标读者是那些上过 Python 课程，或者读过该语言的介绍性图书的开发者。实际上，这些练习的大部分都是为正在上或者刚上完我的 Python 入门课程的学生准备的，他们对 Python 语言的基本结构有所了解，比如 `if` 和 `for`，对核心数据结构也有一定的了解，比如字符串、列表、元组和字典等。

但是，对这些知识点略知一二，与能够将它们应用到实际问题中相比，在一定程度上还是有区别的。如果你有时候能用 Python 应对问题，但发现自己每天还是要多次访问 Stack Overflow，那么本书将帮助你在编写 Python 代码时变得更加自信与独立。我认为，如果你频繁使用 Python，但时间不足六个月，那么本书一定会让你有所收获。

本书是如何组织的：一个简要的路线图

本书有 10 章，分别专注于 Python 的 10 个不同方面，每一章的练习也会用到其他章中讲解的技术。例如，几乎每个练习都要求你编写一个函数或者一个类，但第 6 章才介绍函数，第 9 章才讲到类。你可以将章名当作你未来学习每一章并完成相应练习的一般指导原则，而不是严格的规则。

这些章是：

1. **数值类型**：介绍整数与浮点数，以及数字与字符串之间的转换。

2. **字符串**：使用字符串完成任务。你不仅要将其作为文本来处理，还要将其作为序列进行迭代。

3. **列表和元组**：创建和修改列表与元组（修改仅针对列表），并从中检索内容。

4. **字典和集合**：探索字典的各种使用方式，以及有关字典的最佳用法。另外，有些例子会用到集合，它与字典存在一定的相关性。

5. **文件**：从文件中读取内容和向文件写入内容。

6. **函数**：编写函数，包括嵌套函数。探索 Python 的作用域规则。

7. **使用解析式进行函数式编程**：通过列表解析、集合解析与字典解析来解决问题。

8. **模块和包**：在 Python 程序中编写和使用模块。

9. **对象**：创建 Python 的类，编写类方法，使用属性，以及理解继承。

10. **迭代器和生成器**：为类添加迭代器协议，编写生成器函数与生成器解析代码。

练习组成每章的主要内容。每个练习由以下五部分组成：

1. **问题：**描述一个需要由你解决的问题。

2. **解题思路：**详细探讨问题的细节及解决方案。

3. **解答：**给出最终解决方案的代码及其在 Python Tutor（参见链接 4）上的链接，以便你能直接运行。答案的代码及其测试代码也可以在 GitHub（参见链接 5）上进行下载。

4. **解答视频：**一个简短的视频演示，也就是以录屏的方式为你讲解题目的解决方案。你不仅可以在视频中看到答案，还能了解我寻求答案的整个过程。你要是在 Manning 的 Livebook 平台阅读本书，则可以在每个章节的答案部分找到附带的视频。如果是纸书或电子书，则请打开一个导航页面的链接（参见链接 6），然后通过练习的标题和编号找到视频。

5. **扩展练习：**三个相关的附加练习。书中并没有对其展开讨论，也没有提供解答（详细信息参见后文）。你可以在 Manning 的 Livebook 平台上，通过本书的在线论坛与其他读者一起讨论这些练习，并比较各种解决方案。

有的练习内容后面有补充说明，每个补充说明都解释了一个常令 Python 开发者困惑的问题。例如，有的补充说明讲解了 f- 字符串、变量作用域，以及当你在创建一个新对象时究竟会发生什么。书中还包含很多提示、技巧和注意事项——所有这些都旨在帮助你提升 Python 编程的熟练程度，并提醒你避开我多年以来犯过无数次的错误。

关于代码

本书包含大量的 Python 代码。跟其他大部分书不一样的是，这里的代码都应当是由你编写出来的，而不是仅需要你阅读。如果有一定的经验，某些读者（也许就是你）会写出更好、

更优雅，或者更正确的实现代码。如果是这样，请不要犹豫，联系我。

包括“扩展练习”在内的所有练习的解答，都可以在两个地方找到：一个是 Python Tutor（参见链接 7），它能提供直接运行代码的环境；另一个是 GitHub 代码库（参见链接 8），它能帮助你方便地下载代码。这里的代码库不仅包括所有的解答，还包括为每个解答编写的 pytest 测试（不熟悉 pytest？强烈建议你在参考链接 9 指向的网站学习 pytest，并用它来检查你的代码）。

GitHub 代码库中的代码与书中的代码有一点儿小小的区别。具体来说，书中的解答不包含函数、类和模块的文档字符串，而在可下载的代码库中则有关于文档字符串的说明。

本书中展示的代码，有的是带行号的代码清单，有的则是跟普通文本混在一起的行内代码。在两种情况下，源代码都会以等宽字体（`like this`）来展示。有时会把一部分代码加粗，以标记从上一步到这一步发生了什么变化，比如给一行已有的代码增添一项新功能。

在多数情况下，原始的代码格式会被修改，以适应书本页面大小，比如增加了断行或改写了缩进方式。在少数情况下，即使这样做也还是不够的，因此用续行标志（➥）来表示一行代码没有中断。此外，在文本中提到代码时，其中的注释通常会被去掉。代码清单里会有很多注释，用以解释重要的概念。

如之前所说的，购买本书后，你就能查看每道练习的解答视频。希望能够把书上的答案代码、解题思路、Python Tutor 链接、可下载的源代码、pytest 测试用例及解答视频组合起来，帮助你全面地理解每个练习的精髓，并把从中学到的经验应用到你自己的代码中去。

软件与硬件要求

首先，也是最关键的，本书要求你安装 Python。你可以从网站（参见链接 10）上轻松地下载和安装最新的可用版本。其他安装 Python 的途径包括 Windows Store 或者 macOS 的 Homebrew。

本书适用于 Python 3.6 及更高的版本，少数几处介绍了 3.7 和 3.8 版本引入的功能，但所有练习均使用 3.6 版本的功能提供解答。不同版本程序都可以跨操作系统运行，因此不论你使用什么平台，书中的练习都适用。

从技术上来说，你不需要为 Python 安装任何编辑器或者 IDE（集成开发环境），但有它

们肯定会更方便。两个最流行的 IDE 是 PyCharm（来自 JetBrains）和 VSCode（来自微软）。年纪大一点儿或更传统的 Python 开发者会使用 Vim 或 Emacs（个人的最爱）。但归根结底，你完全可以并且应该使用对你自己来说最适合的编辑器，Python 并不真的“在意”你用的是什么。

Livebook 论坛

购买本书英文版后，你可以获得免费访问本书私密论坛的权限，该论坛由 Manning 出版社提供，在此你可以发表对本书的评论，询问技术问题，并获得作者和其他用户的帮助。访问论坛可参见链接 11。在这里（参见链接 12）则可以详细了解 Manning 论坛及其行为准则。

Manning 出版社承诺为读者提供一个场所，让个人读者可以与作者及其他读者进行有意义的对话。不过，它并不承诺图书作者必须做到何种程度的参与，因为作者在论坛中的贡献是自愿（且无偿）的。建议你向作者提出一些有挑战性的问题，以吸引他的注意力！本书付印后，论坛和此前所有有关本书讨论的记录都可以从出版社的网站访问。

关于作者

鲁文·M. 勒纳（Reuven M. Lerner）是一名全职 Python 培训师，每年都会通过在线平台，向美国、欧洲部分国家、以色列、印度和中国的公司及个人教授课程。他经常发布有关 Python 的博客文章和推特文章（@reuvenmlerner），并且是“自由职业行业谈”（Business of Freelancing）播客小组的成员。鲁文和妻子及他们的三个孩子住在以色列的莫迪因，你可以通过本链接（参见链接 13）了解更多关于鲁文的信息。

关于封面插画

本书封面人物的名字叫作“来自火地岛的男人”（Homme de la Terre de Feu 或者 A man from the Tierra del Fuego）。插图摘自雅克·格拉塞·德·圣索弗（Jacques Grasset de Saint-Sauveur，1757–1810）的《世界各国已知民族的现代民用服饰》（*Costumes civils actuels de tous les peuples connus*）一书，该书于 1784 年在法国出版，其中每幅插图都是精心绘制和着色的。圣索弗拥有丰富多样且十分生动的藏品，提醒我们仅仅在 200 年前，世界各地的城镇和地区在文化上的差异何等巨大，人们彼此隔离，说着不同的语言或方言。不论

是在大街上还是在乡下，仅凭着装就能识别人们居住何处，从事什么职业，以及社会地位如何。

从那之后，人们的着装方式发生了巨变，曾经在穿着上呈现的如此丰富的地区多样性逐渐消失。现在，区分来自不同大陆的居民已经变得很难，更不用说来自不同的国家、地区、城镇的人们。我们牺牲了文化多样性，换来了个人生活的多样性——当然这是一种更加多元的、快节奏的高科技生活。

在这个甚至连计算机图书都难以彼此区分的时代，Manning 出版社将两个多世纪前充满多样性的地区生活图景用到了图书封面中，通过这些被圣索弗重现于世的精美画面，向计算机行业的积极进取与创造性致敬。

目 录

第1章 数值类型

1

无论是在计算工资、银行利息时，还是在计算手机信号频率时，都很难想象你的程序不会用各种各样的方式使用数字。Python有三种不一样的数值类型：整数（`int`），浮点数（`float`）和复数（`complex`）。对大多数人来说，能了解和使用`int`（整数）和`float`（带有小数部分的数字）就够了。

数值不仅是编程的基础，还是很好的入门指引，能让我们了解编程语言是如何运作的。理解整数和浮点数如何被用于变量赋值和函数参数，会帮助你进一步理解更复杂的数值类型，如字符串（string）、元组（tuple）和字典（dict）。

本章包含一些跟数值打交道的练习，练习中数值类型作为输入和输出。尽管使用数值是相当基础且简单直接的操作，但在它们之间进行转换，以及把它们跟其他数据类型熟练结合并应用，有时会需要一些时间。

有用的参考

你需要了解的知识请参见表 1.1。

表 1.1 你需要了解的知识

概念	它是什么	例子	了解更多
`random`	用来生成随机数和抽取随机元素的模块	`number = random.randint(1, 100)`	参见链接14
比较运算	对值进行比较的操作符	`x < y`	参见链接15
f-字符串	可以插入表达式值的字符串	`f'It is currently {datetime.datetime.now()}'`	参见链接16和17
for循环	在可迭代元素集中进行迭代	`for i in range(10): print(i*i)`	参见链接18
`input`	在命令行中提示用户输入一个字符串，并返回一个字符串	`input('Enter your name: ')`	参见链接19
`enumerate`	帮助我们在迭代的同时给元素编序号	`for index, item in enumerate('abc'): print(f'{index}: {item}')`	参见链接20
`reversed`	将可迭代对象的元素反序并返回新的可迭代对象	`r = reversed('abcd')`	参见链接21

练习1 猜数字游戏

设计第一个练习的目的是让你的手指活动一下，热一热身，为本书的其余部分做好准备。该练习引入了一些今后会在你的 Python 职业生涯中反复出现的主题：循环、用户输入、类型转换，以及对值的比较。

说得更具体一点儿，所有的程序都需要输入，才能实现一些有趣的任务，而这些输入通常都来自用户。因此，学习如何从用户那里得到输入不仅十分重要，还会促使我们思考，输入的是什么类型的数据，应该将其转换成什么格式，以及如何转换。

正如你可能知道的，Python 只提供两种循环：`for` 和 `while`。理解如何编写和使用它们，将在你的整个 Python 职业生涯中起到重要作用。事实上，几乎每种类型的数据都知道自己该如何配合 `for` 循环工作，这使得 `for` 循环十分有用且非常普遍。如果你正在处理数据库记录、XML 文件元素，或是使用正则表达式搜索文本后得到的结果，那么你会经常用到 `for` 循环。

在这个练习中，你需要做的是：

- 编写一个函数（`guessing_game`），不接受任何传入参数。
- 运行时，这个函数会从 0 到 100（含 100）中随机选出一个整数。
- 然后让用户猜测选出的数是哪个。
- 每当用户输入一个猜测结果时，程序会给出下面三个提示之一：

  —太高；

  —太低；

  —猜对了。
- 如果用户猜对了，则程序终止；否则会让用户再次尝试。
- 只有在用户猜对的情况下，程序才会退出。

我们会使用 `random` 模块中的 `randint` 函数（参见链接 22）来生成一个随机整数。比如，你可以这么写：

```
import random
number = random.randint(10, 30)
```

number 就会取从 10 到 30（含 30）的一个整数。接下来就可以用 number 来做任何想做的事情：打印、存储、把它传给一个函数，或者将其用于计算。

我们还会用 input 函数来提示用户输入一段文本。实际上，本书会经常用到 input 函数，以此提示用户输入信息。该函数接受一个字符串参数，它作为用户提示信息，然后等待用户输入。无论用户输入什么，函数都会把刚才用户输入的所有内容当作一个字符串返回。例如：

```
name = input('Enter your name: ')
print(f'Hello, {name}!')
```

注意，如果用户在 input 提示输入时仅按下了回车键，那么 input 函数会返回一个空字符串，而不是 None。实际上，不论用户输入什么，input 函数返回的值永远都是一个字符串。

注意，在 Python 2 中，你应该使用 raw_input 函数来读取用户输入。Python 2 的 input 函数被认为是危险的，因为它在让用户输入后，会使用 eval 函数把输入的字符串作为表达式进行求值（如果你对此感兴趣，可以参见链接 23）。Python 3 已经去掉了这个危险的函数，并且把安全的函数重命名为 input。

解题思路

这个程序的核心，实际上是在数字上简单地应用比较运算符（==，< 和 >）。通过比较，用户便能猜到计算机选出的随机整数。不过，对于这个程序，仍然有几个方面值得讨论。

首先，也是最重要的，我们使用了 random 模块来生成一个随机数。在导入 random 模块后，我们就可以调用 random.randint 了，它需要两个参数，可返回一个随机整数。一般来说，无论何时，当你需要生成一个随机数时，random 模块都是极为好用的工具。

注意，random.randint 中的最大值也在可选范围内，这在 Python 中是不常见的。大多数时候，Python 在表示范围时都是排除边界的，也就是说，较大的那个值不会被包含在范围之中。

提示，random 模块不仅可以用来生成随机数，其中的一些函数还可以用来从 Python 对象序列中随机取出一个或多个元素。

现在计算机已经选择了一个数字，轮到用户来猜测这个数字是什么了。我们在这里开始进入一个无限循环。要在 Python 中创建无限循环，最简单的方式是利用 while True，但

重要的是，这里还需要有一个可以退出循环的条件。在这个例子中，退出条件就是用户准确地猜到了答案的数值。当这种情况发生时，我们使用 `break` 命令来退出最内层的循环。

`input` 函数（参见链接 24）总是返回一个字符串，这意味着，如果我们想让用户猜的是一个数字，就必须把用户输入的字符串转换成整数。我们用的方法与 Python 中的其他转换方式都相同：把目标类型当作函数调用，把原始值当作参数传入。因此 `int(5)` 返回整数 `5`，`str(5)` 返回字符串 `'5'`。你也可以生成类型更复杂的实例，同样把对应的类当作函数进行调用，例如 `list('abc')` 或者 `dict([('a', 1), ('b', 2), ('c', 3)])`。

在 Python 3 中，你不能使用 `<` 和 `>` 来比较不同的类型。如果你一时疏忽，没有把用户的输入转换成整数，那么程序会退出并报错，提示不能对字符串（即用户的输入）和整数做比较。

> **注意**，在 Python 2 中，比较不同类型的对象并不会导致程序报错，如果你对此不了解，那么执行这个操作会给你带来出乎意料的结果。这是因为，Python 首先会按类型比较对象，然后比较相同类型的对象。换句话说，任何整数都比列表要小，而任何列表都比字符串要小。问题是，你为什么会想要使用 `<` 和 `>` 来比较不同类型的对象的大小呢？你一般不会这么做。这个功能更多地是让人们感到困惑，而不是帮助他们。在 Python 3 中，你不能再进行这样的比较了。想要检查 `1 < [10, 20, 30]` 是否成立，你只会得到一个 `TypeError` 的异常。

在这个练习及本书的其余部分中，我会使用 f- 字符串把变量里的值插入字符串。我是 f- 字符串的忠实粉丝，建议你也尝试一下（参见本章后文中讨论 f- 字符串的补充说明）。

被海象拯救

从其他编程语言转到 Python 时，程序员经常会对 `while True` 循环感到惊讶，因为循环可以实现用户输入和跳出。难道没有更好的办法了吗？有人会推荐如下的方法：

```
while s = input('Enter thoughts:'):
    print(f'Your thoughts are: {s}')
```

这段代码看起来相当符合逻辑：我们让用户输入，并把输入赋值给 `s`，然后把这个赋给 `s` 的值再传递给 `while`，后者会把它作为布尔值进行求值。如果输入的是一个空字符串，对应的布尔值会是 `False`，并且我们会从循环中退出。

这段代码只有一个问题：它根本行不通。这是因为 Python 中的赋值操作不是一个表达式，

也就是说，它不会返回任何值。既然没有返回值，也就没办法在 `while` 循环中使用。

从 Python 3.8 开始，情况发生了一些变化。这个版本引入了“赋值表达式运算符”，看起来是这个样子的：“`:=`”（一个冒号加一个等号）。但没有人会真的叫它“赋值表达式运算符”，很久以前，它就被叫作“海象运算符”。同样地，从很早以前开始，这个运算符就极具争议性。有些人认为它给语言引入了不必要的复杂性和潜在的缺陷。

前面提到的循环在 Python 3.8 中会是这个样子的：

```
while s := input('Enter thoughts:'):
    print(f'Your thoughts are: {s}')
```

Python 语言中有了海象运算符，我们终于可以摆脱 `while True` 循环及其潜在的危险了！但是等等，难道我们不需要担心 `while` 循环条件中的赋值会造成什么奇怪的影响吗？也许有，并且这也是争议的一部分。但我被海象运算符的支持方说服了，很大程度上，常规的赋值操作和赋值运算符是不能互换的，能用其中一个就不能用另一个，这有效地减少了潜在的滥用风险。

如果你想了解更多关于海象运算符的信息，包括它的争论，以及为什么它实际上非常有用，建议你观看达斯汀·英格拉姆（Dustin Ingram）在 PyCon 2019 中的演讲，他在演讲中提供了一个有效的案例（参见链接 25）。

你也可以在 PEP 572 中读到更多关于该运算符的介绍，它就是在这个 PEP 中被引入和定义的（参见链接 26）。

解答

```
import random


def guessing_game():
    answer = random.randint(0, 100)

    while True:
        user_guess = int(input('What is your guess? '))

        if user_guess == answer:
            print(f'Right!  The answer is {user_guess}')
            break

        if user_guess < answer:
            print(f'Your guess of {user_guess} is too low!')
```

```
        else:
            print(f'Your guess of {user_guess} is too high!')

guessing_game()
```

你可以在 Python Tutor 上逐行运行这段代码（参见链接 27）。

注意，在本练习中，我们假设用户只会输入有效数据，即整数。记住，`int` 函数通常都会假定我们传给它的是一个十进制数，这意味着它可能只接受数字参数。如果你真的想用教科书式的写法，可以先用 `str.isdigit` 方法（参见链接 28）检查一下该字符串中是否只包含数字，或者也可以通过捕获 `ValueError` 异常来检查，当你尝试将某些没法转为整数的内容传给 `int` 时，就会抛出这个异常。

在 Python Tutor 中一步步运行你的代码

在本书中，我提供了很多取自 Python Tutor（参见链接 29）的可视化图解。Python Tutor 是一个非常出色的 Python 在线教学工具（我经常在面授课程中使用它），你几乎可以把任何 Python 代码输入其中，并一步步地逐行运行。本书大多数解决方案的代码都已经被放在了 Python Tutor 上，并提供了访问链接，这样你就可以直接执行这段代码，而不用在该网站上重新输入一遍。

在 Python Tutor 里面，全局变量（包括函数和类）会在*全局区域*（global frame）中显示。记住，你如果要在函数外面定义变量，那么你同时会创建一个全局变量；而在函数内定义的变量都是一个局部变量，在 Python Tutor 中，局部变量会显示在它们自己的深色方框里。变量指向的简单数据结构，如整数和字符串等，会直接显示在变量旁边；指向的列表、元组或字典则会以图表的方式来展示。

解答视频

通过本地址（参见链接 30）的短视频可以了解解答的详细内容。

扩展练习

你经常需要从用户那里获取输入，而且因为获取的是字符串，所以你经常都得把它转成别的对象类型，比如（在这个练习中是）整数。下面是一些针对这个思路的附加练习。

- 修改这个程序，只给用户三次机会猜出正确的数字。如果用户尝试了三次都没成功，程序会告诉他们没有及时猜对，然后退出。

- 你不仅要选定一个随机数，还要从二进制到十六进制中选定一个随机的进制。用户会根据这个指定的进制提交猜测结果。比如用户输入了“10”，然后你得用正确的进制去解析它：“10”可能是 10（十进制），或者 2（二进制），或者 16（十六进制）。
- 尝试同样的任务，但是让程序从词典中随机选择一个单词，然后让用户去猜这个词（你可能希望单词的字母数量限制在二到五个，以避免难度过高）。与其提示用户数字的大小，不如提示单词在词典中出现的位置是更早还是更晚。

f- 字符串

很多人会在做“猜数字游戏”的练习时试图打印一个字符串和一个数字的组合，比如“你猜到了答案 5”。他们很快会发现 Python 不允许把字符串和整数“加”（使用 + 号）在一起。那么，如何才能把这两种类型放到同一行中输出呢？

这个问题长期以来一直困扰着从使用其他语言转至使用 Python 的新手。最早的方法是在字符串上使用 % 运算符：

```
'Hello, %s' % 'world'
```

当 C 语言程序员因为有了 `printf` 这样的工具而欢欣鼓舞时，其他人却觉得这种工具令人受挫。别的不说，% 至少对新人来说不是很直观，因为它要求在传入一个以上参数的时候必须用括号，而且让人没办法简单地多次引用相同的值。

因此，`str.format` 方法被引入 Python 的时候，被认为是一个巨大的进步，它让我们可以这样写：

```
'Hello, {0}'.format('world')
```

虽然我很喜欢用 `str.format`，但很多刚用 Python 的新人会觉得它有点儿难用，而且写起来特别长。他们尤其不喜欢在左边引用变量、在右边传入变量值。再加上 Python 独有的花括号语法让所有人都感到沮丧。

Python 3.6 引入了 f- 字符串，它类似于 Perl、PHP、Ruby 和 UNIX shell 程序员几十年来一直很喜欢的双引号字符串。f- 字符串的工作方式跟 `str.format` 基本相同，但不用传递参数：

```
name = 'world'
f'Hello, {name}'
```

实际使用中比这更方便，你可以把任何想要的表达式放在花括号里，在对字符串进行求值的时候，该表达式也会被求值。例如：

```
name = 'world'
x = 100
y = 'abcd'
f'x * 2 = {x*2}, and y.capitalize() is {y.capitalize()}'
```

你还可以通过在花括号中加上冒号（:）和格式描述符，来改变各种数据类型的格式化方式。例如，你可以让字符串向左或向右对齐，并且用哈希符（#）填充到十位，代码示例如下：

```
name = 'world'
first = 'Reuven'
last = 'Lerner'
        f'Hello, {first:#<10} {last:#>10}'
```

格式描述符 #<10 的意思是此字符串应向左对齐并填充至 10 位，使用 # 补齐文字位数不足的部分。格式描述符 #>10 的意思与 #<10 相同，不过是向右对齐

我极力推荐你去了解 f- 字符串并好好使用它们，在过去几年内，Python 发生了很多改变，f- 字符串是我最喜欢的改变之一。关于 f- 字符串的更多信息，请查看以下资源：

- Python 的不同格式化方式的比较，包括 f- 字符串（参见链接 31）。
- 一篇关于 f- 字符串及其使用方法的长文（参见链接 32）。
- 引入了 f- 字符串的 PEP（参见链接 33）。

如果你因使用 Python2 而无法使用 f- 字符串，那么应该怎么办呢？你可以使用 `str.format`，该字符串方法的工作方式和字符串大致相同，但灵活性较差。另外，你必须显式调用此方法，并通过数字或名称来引用参数。

练习 2 数字求和

我最喜欢的练习类型之一，是重新实现我们在其他地方（比如 Python 或者 UNIX 中）看到过的功能。下一个练习就是这样，你将重新实现 Python 中的 `sum` 函数（参见链接 34）。这个函数接受一个数字序列参数，并返回这些数字的和。所以，如果你调用 `sum([1,2,3])`，那么结果会是 6。

这里的挑战是编写一个 `mysum` 函数，它与内建的 `sum` 函数做的是同样的事情。但不同的是，它需要接受数量可变的参数，而不是将一个数字序列当作唯一的参数。因此，尽管你可以直接调用 `sum([1,2,3])`，但你现在需要改为调用 `mysum(1,2,3)` 或者 `mysum(10,20,30,40,50)`。

注意，内建的 `sum` 函数也接受可选的第二个参数，在这里我们先忽略它。

你能否用内建的 sum 函数来完成这个练习？答案是不行。（你如果知道在我讲课的时候大家多么频繁地问我这个问题，就会感到非常震惊。）

本练习旨在帮助你去思考如何操作数值，以及如何合理地设计函数。尤其应该注意 Python 中的函数可以接受怎样的参数列表。在很多语言中，你可以为同一个函数进行多次定义，每一次都带上不同的参数签名（也就是参数的数量和类型）。在 Python 中，对同一个函数的定义只有一次（也就是最后一次定义）会被保留下来。但通过传入不同的参数数量和类型，并正确识别和适当处理它们，传参的灵活性就可以实现。

提示，如果对此不熟悉，那你可以先了解一下星号运算符（splat 运算符，即 *），在这个 Python 教程（参见链接 35）里有详细介绍。

解题思路

mysum 函数是一个简单的例子，有助于说明如何使用 Python 的星号运算符（也就是 *）来允许一个函数接受任意数量的参数。我们在参数名 numbers 前面加上 *，也就是告诉 Python，这个参数将会接受所有的传入值，因此 numbers 永远是一个元组。

即使没有任何参数被传入这个函数，numbers 仍会是一个元组，即便是一个空元组，也始终是一个元组类型。

当你想接受数量未知的参数时，星号运算符就会变得特别有用。一般来说，你会期望每次传入的参数都是同样类型的，即便 Python 对此并没有强制的规则。根据我的经验，在这种情况下，你应该先拿到由所有的输入值组成的元组（在本例中是 numbers），然后用 for 循环或列表解析来遍历其中每一个元素。

注意，如果你发现自己在用数字索引从 *args 中取出想要的参数，那么你很可能正在做一件错误的事。如果你想单独取出某些特定参数来使用，那就应该定义独立的、有名称的参数列表。

由于我们期望所有的参数都是数字，所以在函数开始时将局部变量 output 设为 0，然后在一个 for 循环中把这些数字挨个儿加上去。一旦写好这个函数，我们就可以在任何时候使用它，对任意一个由数字构成的列表、集合或元组进行求和计算。

虽然你可能没有经常使用（或重新实现）sum 函数，但每当函数需要接受数量未知的参数时，*args 是一个极其常见的方式。

把可迭代对象转换为参数列表

如果我们想用 mysum 函数来计算一个数字列表（比如 [1,2,3]）的和，应该怎么做？我们不能简单地执行 mysum([1,2,3])，这样会让 numbers 参数更像一个元组，其第一个（也是唯一一个）元素是 [1,2,3]，看起来像：([1,2,3],)。

Python 将遍历这个单元素元组，试图将 0 和 [1,2,3] 相加。这会导致一个 TypeError 异常，Python 报错显示不能将一个整数和一个列表相加。

这种情况下的解决方法是调用函数时在参数前面加上 *。如果我们调用 mysum(*[1,2,3])，这个列表就会变成三个单独的参数，然后函数就会以正常的逻辑来运行了。

在调用函数时，这是一个常用的方法。如果你有一个可迭代对象，想把它所包含的元素传给一个函数，那么只需要在调用函数时在函数前面加上一个 *。

解答

```
def mysum(*numbers):
    output = 0
    for number in numbers:
            output += number
    return output

print(mysum(10, 20, 30, 40))
```

你可以在 Python Tutor 上逐行运行这段代码（参见链接 36）。

解答视频

点击本地址（参见链接 37）即可观看解答视频。

扩展练习

遍历一个列表或元组的所有元素，对它们施加一些操作，然后输出结果（例如求和），是一类极其常见的任务。下面是一些例子。

- 内建的 sum 函数接受可选的第二个参数，用来指定求和的初始值（这就是为什么第一个参数是由数字构成的列表，跟 mysum 函数的实现方式不一样）。因此运行 sum([1,2,3], 4) 返回结果 10，因为 1+2+3 等于 6，这个和会被加到初始值 4 上面。请重新实现你的 mysum 函数，

让它也能够这样工作。如果没有提供第二个参数，那么它的初始值应该是 0。注意，虽然你在 Python 3 里面编写函数的时候，还可以在 `*args` 后面再多加一个参数，但建议不要这么做，只提供两个参数就够了，即一个列表和一个可选的初始值。

- 编写一个函数，它可接受一个由数字构成的列表，并能返回这些数字的平均数（即算术平均值）。
- 编写一个函数，它可接受一个单词（字符串）列表，并能返回一个包含三个整数的元组，其中三个整数分别代表最短的词的长度、最长的词的长度和所有单词的平均长度。
- 编写一个函数，它可接受一个 Python 对象列表，将其中的整数，以及可被转为整数的对象相加求和，忽略其他对象。

练习3 跑步计时

系统管理员经常用 Python 来执行各种任务，包括根据用户的输入和文件生成报告。统计一个特定的错误信息发生的频率，或者最近哪些 IP 地址访问了某个服务器，抑或哪些用户最容易输错密码，这些都是很常见的工作。因此，学习如何随着时间推移，持续地跟踪和统计信息（如平均时间），并生成一些基础的报告，是非常重要和有用的。此外，了解如何处理浮点数，以及它们跟整数之间的区别，也十分重要。

在这个练习中，假设你每天跑 10km，作为跑步训练计划的一部分。你希望知道你的平均跑步时间是多少。

编写一个函数（`run_timing`），它会询问用户跑 10km 花费的时间。之后每次跑步还会继续询问同样的问题，并输入每次训练的成绩（以分钟为单位），直到直接输入回车为止。这时，程序会计算并显示用户跑 10km 的平均时间，然后退出。

例如，用户一共输入了三个数据，下面是程序运行的样子：

```
Enter 10 km run time: 15
Enter 10 km run time: 20
Enter 10 km run time: 10
Enter 10 km run time: <enter>

Average of 15.0, over 3 runs
```

注意，输入和输出的数都应该是浮点数。这个练习的目的是帮助你练习，将输入转换为适当的类型，以及在一段时间中持续记录信息。你以后需要记录的数据可

能比跑步时间和距离更加复杂，但随着时间的推移，你会发现不断累计数据的思路在编程中很常见，并且知道如何在 Python 中做到这点相当重要。

解题思路

在之前的练习中，`input` 作为一个函数，会基于用户的输入返回一个字符串。但在这个例子中，用户可能会提供两种不同类型的输入：他们也许会输入一个数字，或者仅输入一个空字符串。

由于空字符串和数字 0 在 `if` 语句中会被认为是 `False`，所以 Python 程序中经常会使用表达式来判断，正如以下的解答代码所示：

```
if not one_run:
    break
```

像下面这样写就不太常见，而且会显得有点儿奇怪：

```
if len(one_run) == 0:
    break
```

虽然这样也能运行，但从普遍接受的惯例来看，这会被认为是不太好的 Python 风格。遵循这些惯例，你的代码会更加凸显 Python 风格，从而也就更容易被其他开发者阅读。这里，在一个可能为空的变量前面使用 `not`，当其为空时给我们提供一个 `False` 值，是更为常见的用法。

在现实世界的 Python 应用中，如果你从用户处接收输入，并调用 `float` 函数（参见链接 38）去转换它，那你应该把它包在 `try`（参见链接 39）里面，以防用户给你一个非法的值。

```
try:
    n = float(input('Enter a number: '))
    print(f'n = {n}')
except ValueError as e:
    print('Hey! That's not a valid number!')
```

还要记住，浮点数不是完全准确的。对于测量程序的运行时间来说，它已经足够精确了；但如果要计算任何敏感的数值，比如用于科学计算或金融计算等，那么使用浮点数是一个糟糕的主意。

如果你还不知道这一点，那么我建议你打开本地的交互式 Python 命令行，问问它 `0.1 +0.2` 的值是多少。你会对结果感到吃惊的。（你还可以访问链接 40，看看在其他各种编程语言中会得到什么结果。）

解决这个问题的一个常见方案是使用整数。在金融计算中，与其记录美元和美分（作为浮点数），不如只记录美分（作为整数）。

通过 f- 字符串控制输出

如果你想在 Python 中打印一个浮点数，那么你可能想要使用 f- 字符串。为什么？因为使用这种方式，你就能指定被打印数字的数量。下面是一个例子：

```
>>> s = 0.1 + 0.7
>>> print(s)
0.7999999999999999
```

这可能不是你想要的，不过只要把 s 放在一个 f- 字符串里，你就能限制输出：

```
>>> s = 0.1 + 0.7
>>> print(f'{s:.2f}')
0.80
```

这里，告诉 f- 字符串我想读出 s 的值，然后把它显示为一个浮点数（f），小数点后最多保留两位数字。关于 f- 字符串的完整文档，以及可以对不同数据类型使用的格式描述符，请参见本章开头的表 1.1。

解答

```
def run_timing():
    number_of_runs = 0
     total_time = 0

     while True:
         one_run = input('Enter 10 km run time: ')

         if not one_run:
            break

            number_of_runs += 1
            total_time += float(one_run)

    average_time = total_time / number_of_runs

    print(f'Average of {average_time}, over {number_of_runs} runs')

run_timing()
```

看，这是一个无限循环！使用“while true”可能看起来很奇怪，而且如果这样的循环里没有任何“break”语句用于在满足条件时退出，则会非常糟糕。但要从用户那里获取未知数量的输入，使用无限循环就很常见了，我认为这样做完全没问题

如果 one_run 是一个空的字符串，就停止循环

你可以在 Python Tutor 中逐行运行这段代码（参见链接 41）。

解答视频

在本地址（参见链接 42）可以观看解答视频 。

扩展练习

在编程的世界中，浮点数不仅有存在的必要性，还有潜在的危险性；它的存在是必要的，是因为很多内容只能用小数表示，它有潜在的危险则因为浮点数并不精确。因此，你需要好好考虑应该何时何地使用它。在下面这两个练习中，你就需要用到 `float`。

- 编写一个函数，其接受一个 `float` 参数和两个整数参数（`before` 和 `after`）。这个函数会取第一个参数的小数点前 `before` 位整数，和小数点后 `after` 位小数，组成一个新的 `float` 并返回。所以，如果把 `1234.5678`，`2` 和 `3` 传入这个函数，返回值会是 `34.567`。
- 探索以另一种方式来表示浮点数的 `Decimal` 类（十进制数，参见链接 43），它能做到以任意精确度来表示十进制数。编写一个函数，从用户那里得到两个字符串，把它们转为 `decimal` 的实例，然后以浮点数的形式打印用户输入的这两个数的和。换句话说，让如下的事情成为可能：用户输入 `0.1` 和 `0.2`，得到答案 `0.3`。

练习 4 转换十六进制数

循环在 Python 中随处可见。由于大多数内建的数据结构都是可迭代的，我们可以很容易地把它们一个个地拿出来依次处理。不过，通常我们迭代一个对象的顺序是固定的，从前往后，从第一个元素到最后一个元素。另外，Python 不会自动提供元素的索引序号。在这个练习中，你会看到如何通过内建的 `reversed` 和 `enumerate` 函数做一点儿创造性的工作，帮你解决这里提到的问题。

十六进制数在计算机领域中理应极为常见，但实际情况却不完全是这样，有些程序员一直都在使用它们，但另一些程序员通常使用高级编程语言做网站开发这类工作，后者甚至不记得如何使用它们。

现在，我在日常工作中几乎不会用到十六进制数。即使需要用到它们，我也会用 Python 内建的 `hex` 函数（参见链接 44）和 `0x` 前缀来实现。前者接受一个整数并返回一个十六进制字符串；后者让我能够使用十六进制记号直接输入一个数字，这种方式更方便。也就是说，`0x50` 等于 `80`，`hex(80)` 会返回字符串 `0x50`。

在这个练习中，你需要编写一个函数（`hex_output`），其接受一个十六进制数，并返回与其相等的十进制数。也就是说，如果用户输入 `50`，你会认为它是一个十六进制数（即

`0x50`），并且在屏幕上输出 `80`。而且，虽然你不能用 `int` 函数一次性转换整个字符串，但允许你每次用 `int` 转换一个数字。

这个练习不是为了测试你的数学技能，如果你想把整数转为相等的十六进制数，直接用 `hex` 函数就行，更何况大部分人在日常生活中根本用不到这样的功能。不过，这个练习能让我们接触到多种常用的类型转换方式，这归功于 Python 中的序列（比如字符串）都是可迭代的。在解决这个问题时，建议考虑使用内建的函数，这比不得不从头开始写所有功能要容易得多。

提示，Python 的指数运算符是 `**`，例如 `2**3` 的结果是整数 `8`。

解题思路

Python 字符串的一个关键性质是，它是由字符组成的序列，所以我们可以用 `for` 循环（参见链接 45）在其中进行遍历。然而，Python 中的循环跟 C 语言中的循环不一样的是，它不会告诉我们（甚至根本就不用）当前字符的索引，而是在字符串本身进行迭代。

如果我们想要得到每个字符的数字索引，可以用内建的 `enumerate` 函数（参见链接 46）。这个函数每次迭代都会返回一个两元素元组，使用 Python 的多重赋值（或称“解包”）语法，我们可以得到每个元素的值，然后将其分别塞到代表幂和数字的 `power` 和 `digit` 变量中。

下面这个例子展示了如何使用 `enumerate` 函数来打印字母表的前四个字母，以及这些字母在字符串中的索引：

```
for index, one_letter in enumerate('abcd'):
    print(f'{index}: {one_letter}')
```

注意，Python 中存在一个 `enumerate` 函数，原因究竟是什么？因为在很多其他的语言（比如 C 语言）中，`for` 循环在数字序列中遍历，再用数字作为索引从另一个序列中取出对应的元素。但是在 Python 中，`for` 循环会直接取出这些元素，不需要任何显式的索引变量。然而，`enumerate` 函数则会基于这些元素来产生索引，这与其他语言的工作方式恰恰相反。

你还会看到这里使用了 `reversed` 函数（参见链接 47），这样我们就能从最后一个数字开始，一直到第一个数字。`reversed` 是一个内建函数，它把字符串的顺序反过来，返回一个新的字符串，其值与旧的字符串相反。我们也可以用切片语法得到同样的结果：

hexnum[::-1]，但我发现很多人对这种语法感到困惑。另外，切片返回的是一个新字符串，而 reversed 函数返回的是一个迭代器，它消耗的内存更少。

我们需要将以字符串形式输入的十进制数的每一位取出来，通过内建的 int 函数（参见链接 48）转换为整数。执行这个函数，可以认为是创建了 int 类或 int 类型的一个实例。我们还能看到，int 函数可以接受两个参数。第一个参数是强制要求的，也就是我们想要将其转换成整数的字符串。第二个参数是可选的，指定了进制。由于要转换的是十六进制数（即进制为 16），这里我们传入 16 作为第二个参数。

解答

```
def hex_output():
    decnum = 0
    hexnum = input('Enter a hex number to convert: ')
    for power, digit in enumerate(reversed(hexnum)):
        decnum += int(digit, 16) * (16 ** power)

    print(decnum)

hex_output()
```

“reversed”函数返回一个新的可迭代对象，后者以相反的顺序返回另一个可迭代对象的元素。把“reversed”函数返回的结果再传入“enumerate”函数，我们就能从“hexnum”里面一次取出一个值，同时取出其对应的从 0 开始的索引

Python 的“**” 运算符用于指数运算

你可以在 Python Tutor 的如下网页（参见链接 49）中逐行执行这段代码。

解答视频

在下面这个短视频（参见链接 50）中可以观看解答的详细过程。

扩展练习

每个 Python 开发者都应该对迭代器协议有深入的了解，for 循环和很多函数都会用到它。结合使用 for 循环和其他工具，例如 enumerate 函数和切片，可以使你的代码更简短，更易于维护。

- 为这个练习重新找一个解决方案，完全不使用 int 函数，而是使用内建的 ord 和 chr 函数来识别字符。按这个方法实现的程序，其健壮性应该更强，也可以基于指定的进制忽略不合法的字符。
- 编写一个程序，要求用户输入他们的名字，然后生成一个“名字三角”：先是名字的第一个字母，然后是前两个字母，然后是前三个字母，依次类推，直到最后写好完整的名字。

小结

一个不使用数字的 Python 程序是很难想象的。无论是作为（字符串、列表或元组的）数字索引，还是计算一个 IP 地址在日志文件中出现的次数，抑或是计算银行贷款的利率，你都会经常使用数字。

记住，Python 是强类型的，意味着整数和字符串永远都是不同的类型。你可以用 `int` 函数把字符串转换成整数，用 `str` 函数把整数变成字符串，还可以用 `float` 函数把这些类型中的任一个转换成浮点数。

在本章中，我们看到了一些处理不同数值类型的方法。你不太可能在真正的编程中仅以这些有限的方式使用数字，但了解它们如何工作，以及如何融入到更大的 Python 生态系统，是非常重要的。