掌握Python面向对象编程的精髓，构建强大的实际应用程序

Python 面向对象编程指南

Mastering Object-oriented Python

[美] Steven F. Lott 著
张心韬 兰亮 译

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Python面向对象编程指南

[美] Steven F. Lott 著
张心韬 兰亮 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Python面向对象编程指南 /（美）洛特（Lott, S. F.）著 ；张心韬，兰亮译. -- 北京 ：人民邮电出版社，2016.3
ISBN 978-7-115-40558-6

Ⅰ. ①P… Ⅱ. ①洛… ②张… ③兰… Ⅲ. ①软件工具－程序设计－指南 Ⅳ. ①TP311.56-62

中国版本图书馆CIP数据核字(2015)第310843号

版权声明

◆ 著 ［美］Steven F. Lott
译 张心韬 兰 亮
责任编辑 陈冀康
执行编辑 胡俊英
责任印制 张佳莹 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：29.25
字数：712千字 2016年3月第1版
印数：1-3 000册 2016年3月北京第1次印刷
著作权合同登记号 图字：01-2014-6030号

定价：79.00元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

Python 是一种面向对象、解释型的程序设计语言，它已经被成功应用于科学计算、数据分析以及游戏开发等诸多领域。

本书深入介绍 Python 语言的面向对象特性，全书分 3 个部分共 18 章。第 1 部分讲述用特殊方法实现 Python 风格的类，分别介绍了__init__()方法、与 Python 无缝集成——基本特殊方法、属性访问和特性及修饰符、抽象基类设计的一致性、可调用对象和上下文的使用、创建容器和集合、创建数值类型、装饰器和 mixin——横切方面；第 2 部分讲述持久化和序列化，分别介绍了序列化和保存、用 Shelve 保存和获取对象、用 SQLite 保存和获取对象、传输和共享对象、配置文件和持久化；第 3 部分讲述测试、调试、部署和维护，分别介绍了 Logging 和 Warning 模块、可测试性的设计、使用命令行、模块和包的设计、质量和文档。

本书深入剖析 Python，帮助读者全面掌握 Python 并构建出更好的应用程序，非常适合对 Python 语言有一定了解并想要深入学习 Python 的读者，也适合有一定开发经验并且想要尝试使用 Python 语言进行编程的 IT 从业人员。

译者简介

张心韬 新加坡国立大学系统分析硕士，北京航空航天大学北海学院软件工程学士。曾经就职于 NEC（新加坡）和 MobileOne（新加坡），目前投身金融领域，就职于 GoSwiff（新加坡），担任.NET 软件工程师，负责支付系统的研发工作。

他在编程领域耕耘数年，涉猎甚广，但自认“既非菜鸟，也非高人”。目前长期专注于.NET 平台，对 Python 也甚为喜爱。业余时间爱好甚广，尤其喜欢学习中医知识，对时间管理、经济和历史也略有涉猎。

兰亮 北京航空航天大学北海学院软件工程学士，IT 行业一线“码农”，曾获评“微软 2014 年度 MVP”和“微软 2015 年度 MVP”。曾一度混迹于飞信（中国）、NEC（新加坡）和 MobileOne（新加坡），现就职于 Keritos（新加坡），从事在线游戏研发工作。

他虽然涉猎广泛，但钟爱开源，长期关注前沿技术，并且对算法、函数式编程、设计模式以及 IT 文化等有着浓厚兴趣。工作之余，他喜欢在 Coursera 蹭课。作为一个热爱生活的人，他在钻研技术之余，还喜欢健身、旅行，立志成为一个阳光、向上的“码农”。

前言

本书主要介绍 Python 语言的高级特性，特别是如何编写高质量的 Python 程序。这通常意味着编写高性能且拥有良好可维护性的程序。同时，我们也会探究不同的设计方案并确定究竟是哪种方案提供了最佳性能。而对于一些正在寻找解决方案的问题，这也是一种很好的方式。

本书的大部分内容将介绍一种给定设计的不同替代方案。一些方案性能更好，另一些方案更加简单或者更加适合于特定领域的问题。最重要的是，找到最好的算法和最优的数据结构，以最少的开销换取最大的价值。时间就是金钱，高效的程序会为它们的用户创造更多的价值。

Python 的很多内部特性都可以直接被应用程序所使用。这意味着，我们的程序可以与 Python 现有的特性非常好地整合。充分利用这些 Python 特性，可以让我们的面向对象设计整合得很好。

我们经常会为一个问题寻找多种不同的解决方案。当你评估不同的算法和数据结构时，通常会设计几种不同的方案，它们在性能和内存的使用上不尽相同。通过评估不同的方案，最终合理地优化应用程序，这是一种重要的面向对象设计技巧。

本书一个更为重要的主题是，对于任何问题，没有所谓的唯一且最好的方法。相反，会有许多不同的方案，而这些方案也各有优劣。

关于编程风格的主题非常有趣。敏锐的读者会注意到，在一些非常细微的部分，例如在名称选择和符号的使用上，并非所有的例子都完全符合 PEP-8。

随着你能够越来越熟练地以面向对象的方式使用 Python，也将不得不花大量的时间去阅读各种 Python 源码。你会发现，甚至在 Python 标准库的模块中，都有很大的可变性。相比于展示完全一致的例子，我们更倾向于去关注那些不一致的部分，正如我们在各种开源项目中所看到的，一致性的缺乏，正是对代码更好的认可。

本书涵盖的内容

我们会用一些章节深入讲解 Python 的 3 个高级主题。

- 一些预备知识，主要讲解一些基本的主题，例如 `unittest`、`doctest`、`docstrings`

以及一些特殊的函数名。

第 1 部分“用特殊方法实现 Python 风格的类”，这个部分着重讲解面向对象编程以及如何更好地将 Python 内置的特性和我们的类进行集成，这个部分包括以下 8 章。

- 第 1 章“__init__()方法”，详细讲解了`__init__()`的功能和实现，我们会用不同的方式初始化一些简单的对象。接着，我们会尝试初始化更加复杂的对象，例如集合和容器。
- 第 2 章“与 Python 无缝集成——基本特殊方法”，讲解如何通过加入特殊函数来扩展一个简单的类。我们需要了解继承的默认行为，以便理解怎样的重写是必需的，以及什么时候应该使用重写。
- 第 3 章“属性访问、特性和修饰符”，主要讲解了默认情况下它们是如何工作的。我们需要决定在什么时候在什么地方重写默认行为。我们还将探讨描述符的细节，以便更好地理解 Python 的内部工作机制。
- 第 4 章“抽象基类设计的一致性”，主要关注 `collections.abc` 模块中的抽象基类。我们会探讨 collections 和 containers 的基本概念，主要关注那些常被扩展和修改的部分。类似地，我们还会探讨 numbers 的基本概念，主要关注那些常被实现的部分。
- 第 5 章“可调用对象和上下文的使用”，主要讲述了使用 `contextlib` 提供的方法以不同的方式来创建上下文管理器。我们会讲解可调用对象的一系列不同设计以及为什么有时候一个有状态的可调用对象会比一个简单的函数更加有用。在我们定制自己的上下文管理器之前，我们还会探讨如何使用 Python 中内置的上下文管理器。
- 第 6 章“创建容器和集合”，关注 container 类的基本使用。我们会探讨在创建容器过程中会调用的各种特殊函数。同时，我们也会探讨如何扩展内置容器以添加新特性。最后，我们将封装内置容器，然后通过委托方法让基础容器可以使用这些封装。
- 第 7 章“创建数值类型”，涵盖了这些基本的运算符：+、−、*、/、//、%和**。同时，我们也会介绍比较运算符，包括<、>、<=、>=、==和!=。最后，我们会总结一些在扩展和定制自己的数值类型时需要注意的设计要点。
- 第 8 章“装饰器和 mixin——横切方面”，涵盖了简单函数描述符，带参数的函数修饰符、类修饰符和方法修饰符。

第 2 部分“持久化和序列化”介绍一个序列化到存储介质的持久化对象，它可能是转换为 JSON 后写入文件系统的，也可能是通过 ORM 存储到数据库的。这个部分会着重探讨持久化的不同方法，包括以下 5 章。

- 第 9 章“序列化和保存——JSON、YAML、Pickle、CSV 和 XML”，涵盖了对不同数据格式做简单的持久化时可使用的现有的库，例如 JSON、YAML、Pickle、XML 和 CSV。
- 第 10 章“用 Shelve 保存和获取对象”，探讨了使用 Python 模块进行简单的数据库操作，例如 Shelve 和 dbm。
- 第 11 章“用 SQLite 保存和获取对象”，进入更加复杂的 SQL 和关系数据库的世界。因为 SQL 的特性并不符合面向对象设计的原则，我们会遇到“阻抗不匹配”问题。一个通用的

解决方案是使用 ORM 存储大量的领域对象。

- 第 12 章“传输和共享对象”，探讨 HTTP 协议以及使用 JSON、YAML 和 XML 来表示要传输的对象。
- 第 13 章“配置文件和持久化”，涵盖了 Python 应用程序使用配置文件的不同方法。

第 3 部分“测试、调试、部署和维护”，我们会展示如何收集数据来支持和调试高性能程序。其中包括创建尽可能完善的文档——减少技术支持的难度。这个部分包括最后 5 章。

- 第 14 章“Logging 和 Warning 模块”，探讨了如何使用 `logging` 和 `warning` 模块来记录审计和调试信息。相比于使用 `print()` 函数，这将是巨大的进步。
- 第 15 章“可测试性的设计”，涵盖了如何设计可测试的程序，以及如何使用 `unittest` 和 `doctest`。
- 第 16 章“使用命令行”，探讨如何使用 `argparse` 模块解析选项和参数。接着，我们会使用命令模式来编写易于整合和扩展的程序模块，而不是使用纯粹的 shell 脚本。
- 第 17 章“模块和包的设计”，涵盖了如何设计模块和包。这是一个更高级的主题，我们会探讨如何将相关的类组织在一个模块中，以及如何将相关的模块组合成一个包。
- 第 18 章“质量和文档”，涵盖了我们应该如何将设计文档化，以便让用户相信我们的软件是可靠的，并且是以正确的方式实现的。

阅读本书你需要准备什么

你需要下面的软件来编译和运行本书中的示例。

- 安装带有标准库的 Python 3.2 或者更高版本。我们会使用 Python 3.3，但是 Python 3.2 和 Python 3.3 之间的差别很小。
- 我们还会使用一些第三方的包，包括 `PyYaml`、`SQLAlchemy` 和 `Jinja2`。
  - http://pyyaml.org。
  - http://www.sqlalchemy.org。安装的时候，对照安装指南，http://docs.sqlalchemy.org/en/rel_0_9/intro.html#installation。用--without-cextensions 可以简化安装过程。
  - http://jinja.pocoo.org/。
- 根据需要，你也可以选择安装 Sphinx 或者 Docutils，因为我们也会介绍它们。
  - http://sphinx-doc.org。
  - http://docutils.sourceforge.net。

本书的目标读者

本书主要讲述 Python 的高级主题，所以要求读者熟悉 Python 3。通过解决大型的复杂问题，你

将会获益良多。

如果你非常熟悉其他的编程语言，但是想切换到 Python，那么你可能会发现本书对你很有帮助。本书不会介绍诸如语法之类的基本概念。

对于熟悉 Python 2 的程序员，本书可以帮助你切换到 Python 3。我们不会涉及任何版本切换的工具（例如，从版本 2 升级到版本 3），以及任何共用的库（例如 six）。书中重点讲述 Python 3 带来的新开发方式。

约定

在本书中，你会发现我们使用不同样式的文字来区分不同类别的信息，下面是这些样式的一些例子。

文本中涉及源码的单词会用以下这种样式："我们可以通过 `import` 来使用 Python 的其他模块"。

代码块的样式如下所示：

```
class Friend(Contact):
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone
```

当我们想提醒你注意一个代码块的特定部分时，我们对该部分使用粗体：

```
class Friend(Contact):
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone
```

以下是一个从命令行进行输入和输出的例子：

```
>>> e = EmailableContact("John Smith", "jsmith@example.net")
>>>Contact.all_contacts
```

新术语和重要的文字以粗体显示。在屏幕上看到的字，例如在菜单或对话框中出现的文字显示效果为："我们通过这个功能来实现在每次单击 Roll 按钮时，在标签中显示一个新的随机值"。

警告或重要信息会以这样的形式显示。

提示和技巧会以这样的形式显示。

读者反馈

非常欢迎读者对本书提供反馈和建议。让我们知道你关于本书的看法——你所喜欢和不喜欢的部分。哪部分内容使读者获得了最大收获，了解这点对于我们是重要的。

如果你要为我们提供反馈的话，只需要发送邮件到 feedback@packpub.com，并在消息标题中包含书名。

如果你有推荐让我们出版的书，请发送邮件至 suggest@packtpub.com。

如果有一个你所擅长的主题并有兴趣写书，可以联系 www.packtpub.com/authors。

客服支持

作为 Packt 图书的主人，我们会尽力为你提供帮助。

下载本书的示例代码

你可以从此处下载所有你通过 Packt 账号支付的 Packt 图书的示例代码：http://www.PacktPub.com。如果你是通过其他方式支付的，可以访问 http://www.PacktPub.com/support 并进行注册，我们将通过邮件的方式发送给你。

勘误

我们已经尽力保证本书内容的质量并避免错误的发生。如果你发现了本书的任何错误——可能是在文字描述上或代码中——我们将非常感谢你能联系我们。这样做可以为其他读者提供帮助并有助于我们提高本书后续的内容质量。如果你发现了任何勘误，请通过访问 http://www.packtpub.com/support 来联系我们，选择你的书并单击 let us know 链接，然后勘误会被上传到我们的网站，或是添加到该书名下的勘误列表。任何已有的勘误都可以通过在 http://www.packtpub.com/support 选择书名来进行查看。

版权

对各种媒体而言，互联网上受版权保护的各种材料都长期面临非法复制的问题。Packt 非常重视对版权和许可的保护。如果你在网络上发现了任何对我们的内容进行非法复制的情形，请立即为我们提供网址或网站名称，这样我们可以采取相应措施。

你也可以通过 copyright@packtpub.com 联系我们，提供盗版材料的链接。

我们感谢你能协助保护作者版权并帮助我们为你提供更有价值的内容。

其他问题

如果你对本书任何一方面存在疑问，可以通过 questions@packtpub.com 和我们联系，我们会尽力提供答复。

审阅者简介

Mike Driscoll 从 2006 年开始使用 Python 语言，他很喜欢在自己的博客中分享有关 Python 的使用技巧（博客地址：http://www.blog.pythonlibrary.org/）。他也是 DZone 出版的 *Core Python refcard* 一书的作者之一，同时也在 Packt 出版社出版过多本个人著作，例如 *Python 3 Object Oriented Programming*、*Python 2.6 Graphics Cookbook* 和 *Tkinter GUI Application Development Hotshot*。Mike 最近在写 *Python 101* 这本书。

我要感谢我的妻子 Evangeline 对我长期以来的支持，感谢我的朋友和家庭成员为我所做的一切。

Róman Joost 从 1997 年就开始从事开源项目的开发。作为 GIMP 用户文档的项目经理，他已经为 GIMP 和 Python/Zope 的开源项目做了 8 年的贡献。目前 Róman 在澳大利亚布里斯班的红帽公司工作。

Sakis Kasampalis 来自新西兰，目前在基于位置服务的 B2B 提供商担任软件开发工程师。他不赞成对编程语言和开发工具持过度追捧的态度。他的原则是把正确的技术应用在适合的问题上。Python 是他最喜爱的工具之一，他很看重 Python 的高效。在 FOSS 的项目工作期间，Kasampalis 维护 GitHub 上的一个与 Python 设计模式相关的项目，相关资料可以从 https://github.com/faif/python-patterns 下载。同时，他也是 Packt 出版的 *Learning Python Design Patterns* 一书的审阅者。

Albert Lukaszewski Ph.D 目前是苏格兰南部的 Lukaszewski 咨询服务中心的首席顾问。他已经有 30 多年的编程经验，现在是系统设计与实现方面的顾问。他之前还在爱可信欧洲有限公司担任过首席工程师。他的大部分经验都与文本处理、数据库系统和自然语言处理（**Natural Language Processing，NLP**）相关。同时，他还写过 *MySQL for Python* 一书，该书由 Packt 出版。除此之外，他之前还曾为纽约时报子公司 About.com 写过 Python 专栏。

Hugo Solis 是哥斯达黎加大学物理系的教授助理。他目前主要研究计算宇宙学、复杂性和氢对材料特性的影响。他在使用 C/C++和 Python 进行科研和可视化方面有着丰富的编程经验。他是自由软件基金会的成员之一，也做过一些开源项目。目前他主要负责管理 IFT，这是哥斯达黎加的一个非营利的科研机构，致力于物理学科的实践（http://iftucr.org）。

我要向我亲爱的母亲 Katty Sanchez 表示感谢，感谢她的支持以及诸多不错的创意。

作者简介

Steven F. Lott 的编程生涯开始于 20 世纪 70 年代，那时候计算机体积很大、昂贵并且非常少见。作为软件工程师和架构师，他参与了 100 多个不同规模的项目研发。在使用 Python 解决业务问题方面，他已经有 10 多年的经验了。

Steven 目前是自由职业者，居住在美国东海岸。他的技术博客是：http://slott-softwarearchitect.blogspot.com。

我要对 Floating Leaf 的支持和引导表示深深的感谢。

一些预备知识

为了使本书接下来的内容更清晰，我们先来看一些关心的问题。其中一项是 21 点游戏。我们将重点关注 21 点游戏的模拟，但并不赞成赌博。

然而，对于面向对象编程来说，模拟是最早的问题之一。这也是能够体现出面向对象编程优雅的一个情形。有关更多信息，可参见 http://en.wikipedia.org/wiki/Simula，以及 Rob Pooley 写的 *An Introduction to Programming*。

本章会介绍一些工具的背景，它们是编写出完整的 Python 程序和包的基础。在接下来的章中会使用它们。

我们会使用 `timeit` 模块将面向对象设计进行对比，找出性能更好的那个。在很多有关如何更好地写出适用于问题模型代码的主观考虑中，使用客观事实来进行说明是非常重要的。

我们将介绍如何在面向对象中使用 `unittest` 和 `doctest` 模块，它们是在开发过程中核对实际工作的基本工具。

一个良好的面向对象设计应当是清晰的并且可读性很强。为了确保良好的可读性，编写 Python 风格的文档是必要的。Docstrings 在模块、类和方法中都很重要。我们会在这里简单概括 RST 标记并会在第 18 章“质量和文档”中详细介绍。

此外，我们还要解决集成开发环境（Integrated Development Environment，IDE）的问题。常见的问题是 Python 开发最好的 IDE。

最后，我们会介绍 Python 中特殊基本方法的概念。关于特殊方法，在前 7 章都有介绍。在这里，我们会介绍一些有助于理解第 1 部分“用特殊方法实现 Python 风格的类”的背景知识。

在讨论 Python 面向对象编程过程中，将尽量避免一些题外话。我们会假设你已经读了 *Python 3 Object Oriented Programming* 这本书。我们不会重复在其他地方已经讲得很清楚的内容。在本书中，会完全关注 Python 3 的内容。

我们会引用很多常见的面向对象设计模式，也不会重复在 *Learning Python Design Patterns* 书中出现的内容。

关于 21 点游戏

如果你还不熟悉 21 点游戏，以下是大致的介绍。

游戏的最终目标是，从庄家手中拿到牌，将手中的牌组成和为在庄家点总数与 21 之间的数字。

在纸牌中数字牌（2 到 10）包含了牌的点数值。而非数字牌（J、Q、K）等同于 10 点。而 A 等于 11 点或 1 点。当把 A 当作 11 点使用时，手中牌的值被称为软手。当将 A 当作 1 点使用时，手中牌的值称为硬手。

如果手中牌中包含了 A 和 7，就可以当作 8 点硬手或 18 点软手。

有 4 种两张牌的组合可以构成 21 点。它们都称为 21 点，尽管其中一种组合包含了 J。

玩 21 点游戏

21 点游戏在不同的场合会有所不同，但主要流程类似，包含如下几点。

- 首先，玩家和庄家各自发两张牌，玩家会知道自己手中的牌面值。在有些场合中会需要将牌面朝上。
- 庄家的牌一张朝上一张朝下。因而玩家会了解一点庄家的牌，但不完全知道。
- 如果庄家朝上的那张牌是 A，那么另一张牌有 4/13 的概率为 10，因而构成总数 21。玩家可以选择做一个额外的保险下注。
- 下一步，玩家可以选择拿牌或停止拿牌。这两种常见的选择称为叫或停牌。
- 还会有更多选择。如果玩家手中的牌匹配，可以选择分牌。这算是额外的下注，分后的两手牌将分开进行游戏。
- 最终，玩家可以在拿最后一张牌前选择双倍下注，称为天生 21 点。如果玩家的牌总数为 10 或 11，这就是一种常见的下注方式。

手牌的最终评判如下。

- 如果玩家的牌大于 21 点，称为超出 21，玩家输并且庄家朝下的那张牌将不做考虑。
- 如果玩家的牌小于等于 21 点，那么根据一种简单的规则庄家拿牌。之后庄家手中的牌必须小于 18 点。手中牌的总数高出 18，庄家必须停叫。关于这点会略有不同，现在暂时忽略。
- 如果庄家超出 21，玩家赢。
- 如果庄家和玩家都小于等于 21 点，则比较他们手中牌的大小。

现在暂时不关心最终的收益。对不同玩法和下注策略的模拟过程来说，总收益关系不大。

21 点游戏策略

对于 21 点游戏来说，玩家必须使用以下两种策略。

- 一种策略用于决定玩法：保险、叫、停叫、分牌或双倍。
- 另一种策略用于决定下注大小。一种常见的谬误统计可以引导玩家提高或降低下注，进而最大限度地保证胜的概率并减少输的概率。任何模拟游戏的软件也必须对复杂下注策略进行模拟。它们是一些有趣的算法，通常包含状态，需要学习一些高级的 Python 编程技巧来完成。

这两种策略是介绍策略模式不错的例子。

21 点游戏模拟器对象的设计

我们将使用游戏中的元素，例如玩家手中的牌作为对象模型的例子。然而，不会对整个过程进行模拟。我们会重点关注游戏中的元素，因为它们会有细微的差别但不是特别复杂。

使用一个简单的容器：存放手中的牌对象，可以包含 0 个或多个。

介绍 `Card` 的子类：`NumberCard`，`FaceCard` 和 `Ace`。

介绍几种不同的方式来定义这种简单的类层次结构。由于层次结构很小（并且简单），可以简单对几种不同的实现方式进行尝试。

介绍几种实现玩家手中牌的方式。这只是一个简单的纸牌集合，包含了一些额外的功能。

从全局的视角来看玩家对象，玩家会有几手牌和下注策略以及 21 点游戏策略。这是一个复杂的组合对象。

我们也会对洗牌和发牌进行快速介绍。

性能——timeit 模块

我们会使用 `timeit` 模块来将不同面向对象设计和 Python 结构进行对比，`timeit` 模块包含了很多函数。重点关注的是 `timeit`，这个函数会为一些语句创建一个 `Timer` 对象，也会包含一些预备环境的安装代码，然后调用 `Timer` 的 `timeit()` 方法来执行一次安装过程并重复执行目标语句。返回值为运行语句所需的时间。

默认计数为 100000 次。这提供了一个有意义的平均时间值，来自其他计算机上 OS 级别活动的统计。对于复杂的或长时间运行的语句，需要谨慎使用小计数值。

以下是与 `timeit` 简单交互的示例代码：

```
>>> timeit.timeit( "obj.method()", """
... class SomeClass:
...     def method(self):
...         pass
... obj= SomeClass()
""")
0.1980541350058047
```

下载示例代码

你可以通过自己的帐号下载所有从 Packt 出版社所购买的书籍中的示例代码：http://www.packtpub.com。如果你是从其他地方购买的，可以访问 http://www.packtpub.com/support 并注册，我们会通过邮件形式发送给你。

obj.method()语句以字符串的形式提供给 timeit()，安装为类定义并且也由字符串的形式提供。语句中所需要的任何东西都必须在安装中提供，它包括所有的导入和所有的变量定义以及对象创建。

可能会需要多尝试几次来完成安装过程。当使用交互式 Python 时，经常会由于命令行窗口的翻屏导致无法追踪全局变量和导入信息。有一个例子是，10000 次空方法的调用，花了 0.198 秒。

以下是另一个使用 timeit 的例子：

```
>>> timeit.timeit( "f()","""
... def f():
...     pass
... """ )
0.13721893899491988
```

这个例子说明，空函数的调用会比空方法的调用略快一些，在这个例子中差不多为 44%。

在一些情况下，OS 的开销可以作为性能的测量组件，它们通常源自难以控制的因素。在这个模块中可以使用 repeat()函数来替代 timeit()函数。它会收集基本定时的多个样本，对 OS 在性能上的影响做进一步分析。

对于我们而言，timeit()函数会提供所有反馈信息，我们可用于在客观上对不同面向对象涉及的要素进行评估。

测试——unittest 和 doctest

单元测试当然是基本的。如果没有用于展示某个功能的单元测试，那么这个功能就不是真的存在。换句话说，对于一个功能来说，直到有测试可以说明它已经完成才算是完成。

我们只会对测试进行少量介绍。如果对每个面向对象设计功能的测试都进行深入介绍，那么这本书的厚度应该是现在的两倍。在忽略测试内容的细节上会存在一个误区，好的单元测试似乎只是