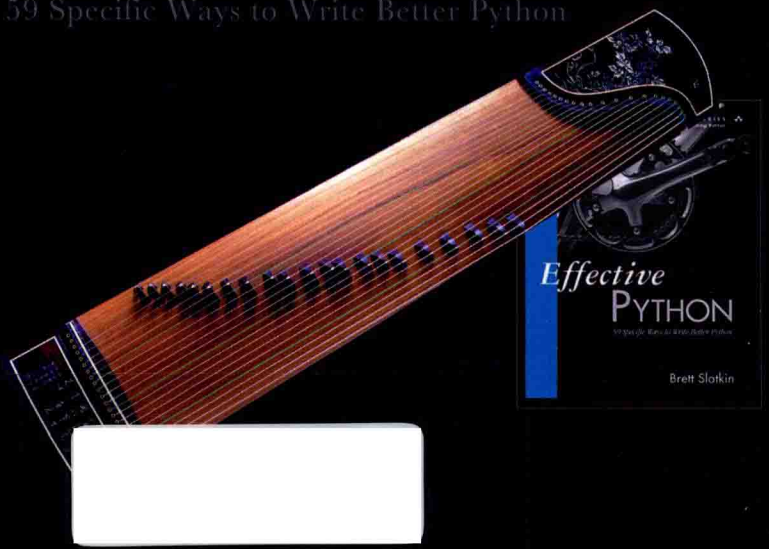
Broadview
www.broadview.com.cn
PEARSON
· 原味精品书系

Effective Python（英文版）

编写高质量Python代码的59个有效方法

Effective Python: 59 Specific Ways to Write Better Python

[美] Brett Slatkin 著



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

· 原味精品书系 ·

Effective Python（英文版）

编写高质量Python代码的59个有效方法

Effective Python: 59 Specific Ways to Write Better Python

[美] Brett Slatkin 著

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

内容简介

本书不是要讲述 Python 的基础编程，而是要帮你掌握 Python 独特的优势和魅力。书中总结了 59 个 Python 编程的优秀实践、贴士和捷径，并用真实代码示例进行了解释。全书共分 8 章，第 1 章讲述 Python 的风格思想，介绍了 Python 中常见问题的推荐解决方案；第 2 章讲述如何使用 Python 函数来阐明意图、提升可重用性，并减少错误；第 3 章介绍如何使用类和继承来表达你对对象的预期行为；第 4 章介绍了使用这些元类和属性的常用语法；第 5 章讲述如何在并行和并发的场景下利用好 Python；第 6 章讲述 Python 中必要的内置模块；第 7 章教你如何合作开发 Python 程序；第 8 章介绍如何使用 Python 调试、优化和测试程序。

版权贸易合同登记号　图字：01-2015-6099

图书在版编目（CIP）数据

Effective Python：编写高质量 Python 代码的 59 个有效方法 =Effective Python: 59 Specific Ways to Write Better Python：英文 /（美）斯拉特金（Slatkin,B.）著. — 北京：电子工业出版社，2016.4
（原味精品书系）
ISBN 978-7-121-27262-2
I. ① E…II. ① 斯…III. ① 软件工具－程序设计－英文 IV. ① TP311.56
中国版本图书馆 CIP 数据核字（2015）第 227471 号

策划编辑：张春雨
责任编辑：徐津平
印　　刷：北京中新伟业印刷有限公司
装　　订：三河市皇庄路通装订厂
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：15.25　　字数：366 千字
版　　次：2016 年 4 月第 1 版
印　　次：2016 年 4 月第 1 次印刷
定　　价：65.00 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。
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。
服务热线：（010）88258888。

本书赞誉

本书中的每一个项目都有自己的源代码，都是一个自包含的课程。这就让这本书具有了随机可读的特性：无论读者需要按照什么顺序，都可以很容易地浏览和学习项目。我会把 *Effective Python* 推荐给学生们，因为它以紧凑的形式向 Python 程序员提供了各方面的建议。

——Brandon Rhodes，Dropbox 软件工程师，PyCon 2016~2017 主席

我已经用 Python 编程好多年了，我认为我已经很了解 Python 了。感谢这本书中的技巧和建议让我意识到我可以让我的 Python 代码运行得更快（比如使用 Python 内建的数据结构），让我的代码更容易阅读（比如强制只使用关键字参数），并且让代码更符合 Python 语言编程理念的代码风格（比如用 zip 并行地迭代列表）。

——Pamela Fox，Khan 学院教育工作者

如果在我第一次从 Java 转向 Python 的时候，我就拥有这本书，那么我将节省很多个月的写重复代码的时间，那时候每当我意识到自己做的事情不符合 Python 编程理念时，我就会重写代码。这本书收集了 Python 中必须知道的绝大多数基础知识，不需要我们再通过课程对它们进行数月甚至数年的学习。本书涉及的范围之广是令人印象深刻的，以 PEP8 的重要性及 Python 的习惯用法开篇，然后贯穿函数、方法和类的设计，以及标准库的高效使用、高质量 API 的设计、测试及性能测量，这本书真的包含了一切。无论对 Python 新手还是有经验的开发人员来说，这都是一本出色的介绍读本。

——Bayer，SQLAlchemy 的创建者

Effective Python 这本书通过明确的准则帮你优化 Python 代码风格和功能，进而让你的 Python 技能达到更高的级别。

——Leah Culver，Dropbox 主开发者

对于那些其他语言经验丰富的、正寻求快速开启 Python 学习，并且要超越基本语言构造以编写更符合 Python 规范的代码的开发人员来说，本书是一个非常宝贵的资源。这本书的组织清晰、简洁、易于消化，而且每个项目、每一章都可以独立地作为一个特定主题。本书用纯 Python 的知识涵盖了 Python 语言结构的广度，而没有用更广泛的 Python 生态系统让读者感觉混淆。对于那些经验丰富的开发人员来说，本书为他们提供了他们之前可能没遇到过的深入的语言结构示例，以及一些不常用的语言特性的例子。可以很明显地看出，

本书作者使用 Python 时得心应手，他用他的专业经验提醒读者去注意微妙的错误和常见的错误方式。此外，这本书还做了一个出色的工作，那就是指出了 Python 2.X 和 Python 3.X 的微妙差异，因此可以作为 Python 不同版本之间过渡的参考资料。

——Katherine Scott，Tempo Automation 软件工程师主管

这本书对新手和有经验的程序员来说都是一本好书，示例代码和说明都是作者经过深思熟虑的，并且阐释简明透彻。

——C. Titus Brown，加州大学戴维斯分校副教授

对于使用 Python 高级的用法构建一个清晰的可维护的软件来说，本书是一个非常实用的资源，那些想要把他们的 Python 技能提高到一个新水平的人将受益于本书的建议与实践。

——Wes McKinney，Python pandas 库创建者，
Python for Data Analysis 一书作者，
Cloudera 软件工程师

献给我的家人，无论还在不在，爱你们！

前言

Python 编程语言具有独特的优势和魅力，可能很难被掌握。很多熟悉其他语言的程序员经常带着思维定势去接近 Python，这样就不能感受它的全部表现力了。还有些程序员在其他方向走得太远，过度使用 Python 的特性可能会在以后引起严重问题。

本书提供了编写符合 Python 规范的代码的方式：Python 的最佳使用方式。阅读本书需要对该语言有一个基本的了解。新手程序员可以学习 Python 能力的最佳实践，有经验的程序员可以学习如何自信地接受一个陌生的新工具。

本书的目标是让你做好准备，迎接 Python 将为你带来的巨大影响。

这本书覆盖的内容

本书的每章都包含了一个广泛但是相关联的项目。你可以根据自己的兴趣在项目间任意跳跃。每个项目都包含简洁具体的指导，用以说明如何更有效地编写 Python 程序。每个项目包括建议做什么、需要避免什么、如何取得适当的平衡，以及为什么这是最好的选择。

无论你使用的 Python 2 还是 Python 3，这本书中的项目都是适用的（见 Item 1：Know Which Version of Python You're Using）。对于使用运行时的 Jython、IronPython 或者 Pypy 的程序员，也会发现本书中的大多数项目是适用的。

第 1 章　Python 风格思想（Pythonic Thinking）

Python 社区已经发展到了用形容词 Pythonic 来描述遵循特定样式的代码的阶段。长期以来，Python 的惯用语法在语言上以及和其他语言合作上经受住了考验。本章介绍了 Python 中最常见事情的最佳做法。

第 2 章　函数（Functions）

Python 函数有各种各样的额外功能，这使得程序员的生活更轻松了。有些函数的功能与其他编程语言类似，但很多函数的功能是 Python 独有的。本章介绍如何使用函数来阐明意图、提升可重用性，并减少错误。

第 3 章　类和继承（Classes and Inheritance）

Python 是一门面向对象语言。想要在 Python 中把事情做好，经常需要写新类并定义它们如何通过接口和层次交互。本章介绍如何使用类和继承来表达你对对象的预期行为。

第 4 章 元类和属性（Metaclasses and Attributes）

元类和动态属性是 Python 的强大功能。但是它们也能够让你的实现出现离奇和意想不到的行为。本章介绍了使用这些机制常用的语法，以确保让你遵循最少意外的原则（rule of least surprise）。

第 5 章 并行和并发（Concurrency and Parallelism）

Python 可以很容易地编写在同一时间内做很多不同事情的并发程序。Python 也可以通过系统调用做并行工作。比如使用 subprocess 模块和 C 扩展。本章介绍了如何在这些不同的微妙场景下最好地利用 Python。

第 6 章 内建模块（Built-in Modules）

Python 安装了很多写程序时可能会用到的重要模块。这些标准包与 Python 的惯用语法之间的联系是如此紧密，以至于它们也可能成为语言规范的一部分。本章讲述了必要的内置模块。

第 7 章 合作（Collaboration）

Python 程序合作中要求你思考如何编写代码。即使独自工作，你也需要了解如何使用别人写好的模块。本章介绍的标准工具和最佳实践将教你如何合作开发 Python 程序。

第 8 章 生产（Production）

Python 提供可以适应多种部署环境的便利。它还具有内置的模块，有助于强化程序，使之健壮无比。本章将介绍如何使用 Python 调试、优化和测试程序，以便在运行时最大限度地提高质量和性能。

本书中一些约定

本书中的 Python 代码段使用了等宽字体，并且有语法高亮。我采取了 Python 风格指南的一些艺术性的许可，使代码示例能够更好地适应一本书的格式，并且能够更突出最重要的部分。

当一行代码太长时，我用 ➥ 字符换行；截断片段用省略号注释（#...）来表示，代表某些非核心内容的代码；我略去了嵌入的文档，以减小代码示例的大小。我强烈建议你不要在自己的项目中这样做，相反，你应该遵循风格指南（见 Item 2：Follow the PEP 8 Style Guide），并撰写文档（见 Item 49：Write Docstrings for Every Function, Class, and Module）。

本书中大多数的代码片段都配有相应的运行结果输出。这里说的“输出”是指在控制台或者终端输出。当在交互式的解释器中运行 Python 程序时，你看到了什么？输出部分是等宽字体，并且前面有 >>> 行（Python 交互式提示）。我们的想法是，你可以将键入的代码片段转换为 Python shell，再次产生期望的输出。

最后，还有一些前面没有 >>> 行的其他等宽字体。这些代表除了运行 Python 解释器之外的程序的输出。这些例子往往以 $ 开头，表明我正在运行一个命令行 shell 的程序（如 Bash）。

从何处获取代码和勘误表

查看本书中示例的完整代码是很有用的，这样能够更好地理解自己的代码，以及为什么程序会按照描述的那样运行。可以在本书的网站（http://www.effectivepython.com）找到本书中所有代码片段的源代码。本书相关的任何勘误也会在该网站上更正。

致谢

这本书的成功出版离不开生活中很多人的指导、支持和鼓励。

首先感谢 Scott Meyers 的高效软件开发系列图书（Effective Software Development series）。第一次看 *Effective C++* 的时候，我 15 岁，那时候我爱上了 C++。毫无疑问，Scott 的书籍让我积累了学术经验，并在谷歌找到了第一份工作。我很高兴有机会写这本书。

感谢我的核心技术审校者提供的深刻而缜密的反馈，他们是 Brett Cannon、Tavis Rudd 和 Mike Taylor。感谢 Leah Culver 和 Adrian Holovaty 对本书的认可。感谢我的朋友们 Michael Levine、Marzia Niccolai、Ade Oshineye 和 Katrina Sostek，他们耐心阅读了本书的早期版本。感谢谷歌同事的审校。如果没有你们的帮助，这本书远没有今天清晰可读。

感谢每一位帮助这本书成为现实的人。感谢我的编辑 Trina MacDonald 对本书自始至终的支持。感谢帮助过我的整个编辑团队：开发编辑 Tom Cirtin 和 Chris Zahn、编辑助理 Olivia Basegio、销售经理 Stephane Nakib、文字编辑 Stephanie Geels 和生产编辑 Julie Nahil。

感谢我知道并且合作过的出色的 Python 程序员 Anthony Baxter、Brett Cannon、Wesley Chun、Jeremy Hylton、Alex Martelli、Neal Norwitz、Guido van Rossum、Andy Smith、Greg Stein 和 Ka-Ping Yee，我很欣赏你们的修养和领导力。Python 有一个很好的社区，很荣幸我能够成为其中的一分子。

感谢我的团队成员们，多年来你们一直允许我这个最差的成员留在队中。感谢 Kevin Gibbs 帮我承担风险。感谢 Ken Ashcraft、Ryan Barrett 和 Jon McAlister 教我如何去做。感谢 Brad Fitzpatrick 让我们的团队到达一个新的水准。感谢 Paul McDonald 共同创办了这个疯狂的项目。感谢 Ginsberg 和 Jack Hebert 让本书成为现实。

感谢曾经鼓励我的老师：Ben Chelf、Vince Hugo、Russ Lewin、Jon Stemmle、Derek Thomson 和 Daniel Wang。没有你们的指导，我可能永远不会达到现在的水平，不会具备教导别人的能力。

感谢我的母亲，她让我坚持目标，并鼓励我成为一名程序员。感谢我的兄弟、祖父母，以及其他的家人和童年伙伴们，你们在我成长的过程中扮演着重要的角色。

最后，感谢我的妻子 Colleen，感谢生活中一直以来的爱与支持，还有乐观的笑声。

关于作者

Brett Slatkin 是一名谷歌高级软件工程师，他是谷歌消费者调查的工程主管兼联合创始人之一。他曾供职于 Google App Engine 的 Python 基础设置部门。他是 PubSubHubbub 协议的创建者之一。十年前，年纪轻轻的他就已经在用 Python 来管理谷歌庞大的服务器机群了。

除了日常工作之外，他还研究开源的工具，会在他的个人网站（http://onebigfluke.com）写软件、自行车等方面的文章。他获得了哥伦比亚大学（纽约）计算机工程学士学位。目前住在旧金山。

目录

1 Pythonic Thinking

The idioms of a programming language are defined by its users. Over the years, the Python community has come to use the adjective *Pythonic* to describe code that follows a particular style. The Pythonic style isn't regimented or enforced by the compiler. It has emerged over time through experience using the language and working with others. Python programmers prefer to be explicit, to choose simple over complex, and to maximize readability (type `import this`).

Programmers familiar with other languages may try to write Python as if it's C++, Java, or whatever they know best. New programmers may still be getting comfortable with the vast range of concepts expressible in Python. It's important for everyone to know the best—the *Pythonic*—way to do the most common things in Python. These patterns will affect every program you write.

Item 1: Know Which Version of Python You're Using

Throughout this book, the majority of example code is in the syntax of Python 3.4 (released March 17, 2014). This book also provides some examples in the syntax of Python 2.7 (released July 3, 2010) to highlight important differences. Most of my advice applies to all of the popular Python runtimes: CPython, Jython, IronPython, PyPy, etc.

Many computers come with multiple versions of the standard CPython runtime preinstalled. However, the default meaning of `python` on the command-line may not be clear. `python` is usually an alias for `python2.7`, but it can sometimes be an alias for older versions like `python2.6` or `python2.5`. To find out exactly which version of Python you're using, you can use the `--version` flag.

```
$ python --version
Python 2.7.8
```

Python 3 is usually available under the name `python3`.

```
$ python3 --version
Python 3.4.2
```

You can also figure out the version of Python you're using at runtime by inspecting values in the `sys` built-in module.

```
import sys
print(sys.version_info)
print(sys.version)
```

```
>>>
sys.version_info(major=3, minor=4, micro=2,
➥releaselevel='final', serial=0)
3.4.2 (default, Oct 19 2014, 17:52:17)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.51)]
```

Python 2 and Python 3 are both actively maintained by the Python community. Development on Python 2 is frozen beyond bug fixes, security improvements, and backports to ease the transition from Python 2 to Python 3. Helpful tools like the `2to3` and `six` exist to make it easier to adopt Python 3 going forward.

Python 3 is constantly getting new features and improvements that will never be added to Python 2. As of the writing of this book, the majority of Python's most common open source libraries are compatible with Python 3. I strongly encourage you to use Python 3 for your next Python project.

Things to Remember

- There are two major versions of Python still in active use: Python 2 and Python 3.
- There are multiple popular runtimes for Python: CPython, Jython, IronPython, PyPy, etc.
- Be sure that the command-line for running Python on your system is the version you expect it to be.
- Prefer Python 3 for your next project because that is the primary focus of the Python community.

Item 2: Follow the PEP 8 Style Guide

Python Enhancement Proposal #8, otherwise known as PEP 8, is the style guide for how to format Python code. You are welcome to write Python code however you want, as long as it has valid syntax.

However, using a consistent style makes your code more approachable and easier to read. Sharing a common style with other Python programmers in the larger community facilitates collaboration on projects. But even if you are the only one who will ever read your code, following the style guide will make it easier to change things later.

PEP 8 has a wealth of details about how to write clear Python code. It continues to be updated as the Python language evolves. It's worth reading the whole guide online (http://www.python.org/dev/peps/pep-0008/). Here are a few rules you should be sure to follow:

Whitespace: In Python, whitespace is syntactically significant. Python programmers are especially sensitive to the effects of whitespace on code clarity.

- Use spaces instead of tabs for indentation.
- Use four spaces for each level of syntactically significant indenting.
- Lines should be 79 characters in length or less.
- Continuations of long expressions onto additional lines should be indented by four extra spaces from their normal indentation level.
- In a file, functions and classes should be separated by two blank lines.
- In a class, methods should be separated by one blank line.
- Don't put spaces around list indexes, function calls, or keyword argument assignments.
- Put one—and only one—space before and after variable assignments.

Naming: PEP 8 suggests unique styles of naming for different parts in the language. This makes it easy to distinguish which type corresponds to each name when reading code.

- Functions, variables, and attributes should be in `lowercase_underscore` format.
- Protected instance attributes should be in `_leading_underscore` format.
- Private instance attributes should be in `__double_leading_underscore` format.
- Classes and exceptions should be in `CapitalizedWord` format.
- Module-level constants should be in `ALL_CAPS` format.