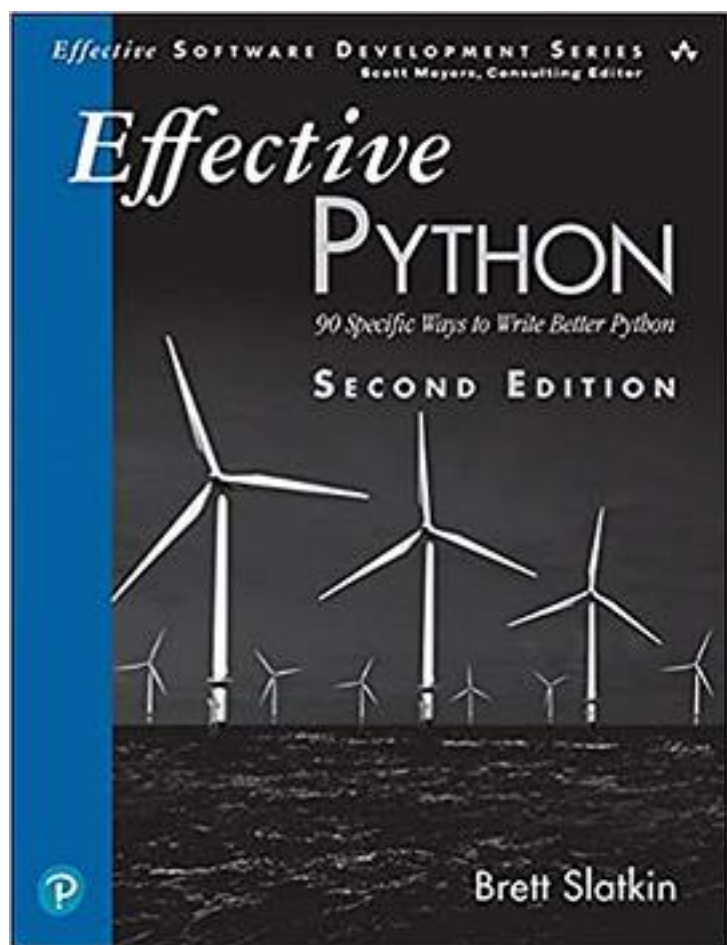


# Effective Python: 2nd Edition



[Effective Python: 2nd Edition 下载链接1](#)

著者:Brett Slatkin

出版者:Addison-Wesley Professional

出版时间:2019-12-2

装帧:Paperback

isbn:9780134854717

Updated and Expanded for Python 3

It's easy to start developing programs with Python, which is why the language is so

popular. However, Python's unique strengths, charms, and expressiveness can be hard to grasp, and there are hidden pitfalls that can easily trip you up.

This second edition of Effective Python will help you master a truly “Pythonic” approach to programming, harnessing Python's full power to write exceptionally robust and well-performing code. Using the concise, scenario-driven style pioneered in Scott Meyers' best-selling Effective C++, Brett Slatkin brings together 90 Python best practices, tips, and shortcuts, and explains them with realistic code examples so that you can embrace Python with confidence.

Drawing on years of experience building Python infrastructure at Google, Slatkin uncovers little-known quirks and idioms that powerfully impact code behavior and performance. You'll understand the best way to accomplish key tasks so you can write code that's easier to understand, maintain, and improve. In addition to even more advice, this new edition substantially revises all items from the first edition to reflect how best practices have evolved.

Key features include

30 new actionable guidelines for all major areas of Python

Detailed explanations and examples of statements, expressions, and built-in types

Best practices for writing functions that clarify intention, promote reuse, and avoid bugs

Better techniques and idioms for using comprehensions and generator functions

Coverage of how to accurately express behaviors with classes and interfaces

Guidance on how to avoid pitfalls with metaclasses and dynamic attributes

More efficient and clear approaches to concurrency and parallelism

Solutions for optimizing and hardening to maximize performance and quality

Techniques and built-in modules that aid in debugging and testing

Tools and best practices for collaborative development

Effective Python will prepare growing programmers to make a big impact using Python.

作者介绍:

Brett Slatkin is a principal software engineer at Google. He is the technical co-founder of Google Surveys, the co-creator of the PubSubHubbub protocol, and he launched Google's first cloud computing product (App Engine). Fourteen years ago, he cut his teeth using Python to manage Google's enormous fleet of servers. Outside of his day job, he likes to play piano and surf (both poorly). He also enjoys writing about programming-related topics on his personal website (<https://onebigfluke.com>). He earned his B.S. in computer engineering from Columbia University in the City of New York. He lives in San Francisco.

目录: Preface

Acknowledgments

About the Author

1. Pythonic Thinking

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

Item 2: Follow the PEP 8 Style Guide

Item 3: Know the Differences Between bytes and str

Item 4: Prefer Interpolated F-Strings Over C-style Format Strings and str.format

Item 5: Write Helper Functions Instead of Complex Expressions

Item 6: Prefer Multiple Assignment Unpacking Over Indexing

Item 7: Prefer enumerate Over range

Item 8: Use zip to Process Iterators in Parallel

Item 9: Avoid else Blocks After for and while Loops

Item 10: Prevent Repetition with Assignment Expressions

2. Lists and Dictionaries

Item 11: Know How to Slice Sequences

Item 12: Avoid Striding and Slicing in a Single Expression

Item 13: Prefer Catch-All Unpacking Over Slicing

Item 14: Sort by Complex Criteria Using the key Parameter

Item 15: Be Cautious When Relying on dict Insertion Ordering

Item 16: Prefer get Over in and KeyError to Handle Missing Dictionary Keys

Item 17: Prefer defaultdict Over setdefault to Handle Missing Items in Internal State

Item 18: Know How to Construct Key-Dependent Default Values with \_\_missing\_\_

3. Functions

Item 19: Never Unpack More Than Three Variables When Functions Return Multiple Values

Item 20: Prefer Raising Exceptions to Returning None

Item 21: Know How Closures Interact with Variable Scope

Item 22: Reduce Visual Noise with Variable Positional Arguments

Item 23: Provide Optional Behavior with Keyword Arguments

Item 24: Use None and Docstrings to Specify Dynamic Default Arguments

Item 25: Enforce Clarity with Keyword-Only and Positional-Only Arguments

Item 26: Define Function Decorators with functools.wraps

4. Comprehensions and Generators

Item 27: Use Comprehensions Instead of map and filter

Item 28: Avoid More Than Two Control Subexpressions in Comprehensions

Item 29: Avoid Repeated Work in Comprehensions by Using Assignment Expressions

Item 30: Consider Generators Instead of Returning Lists

Item 31: Be Defensive When Iterating Over Arguments

Item 32: Consider Generator Expressions for Large List Comprehensions

Item 33: Compose Multiple Generators with yield from

Item 34: Avoid Injecting Data into Generators with send

Item 35: Avoid Causing State Transitions in Generators with throw

Item 36: Consider itertools for Working with Iterators and Generators

5. Classes and Interfaces

Item 37: Compose Classes Instead of Nesting Many Levels of Built-in Types

Item 38: Accept Functions Instead of Classes for Simple Interfaces

Item 39: Use @classmethod Polymorphism to Construct Objects Generically

Item 40: Initialize Parent Classes with super

Item 41: Consider Composing Functionality with Mix-in Classes

Item 42: Prefer Public Attributes Over Private Ones

Item 43: Inherit from collections.abc for Custom Container Types

## 6. Metaclasses and Attributes

Item 44: Use Plain Attributes Instead of Setter and Getter Methods

Item 45: Consider @property Instead of Refactoring Attributes

Item 46: Use Descriptors for Reusable @property Methods

Item 47: Use \_\_getattr\_\_, \_\_getattribute\_\_, and \_\_setattr\_\_ for Lazy Attributes

Item 48: Validate Subclasses with \_\_init\_subclass\_\_

Item 49: Register Class Existence with \_\_init\_subclass\_\_

Item 50: Annotate Class Attributes with \_\_set\_name\_\_

Item 51: Prefer Class Decorators Over Metaclasses for Composable Class Extensions

## 7. Concurrency and Parallelism

Item 52: Use subprocess to Manage Child Processes

Item 53: Use Threads for Blocking I/O, Avoid for Parallelism

Item 54: Use Lock to Prevent Data Races in Threads

Item 55: Use Queue to Coordinate Work Between Threads

Item 56: Know How to Recognize When Concurrency Is Necessary

Item 57: Avoid Creating New Thread Instances for On-demand Fan-out

Item 58: Understand How Using Queue for Concurrency Requires Refactoring

Item 59: Consider ThreadPoolExecutor When Threads Are Necessary for Concurrency

Item 60: Achieve Highly Concurrent I/O with Coroutines

Item 61: Know How to Port Threaded I/O to asyncio

Item 62: Mix Threads and Coroutines to Ease the Transition to asyncio

Item 63: Avoid Blocking the asyncio Event Loop to Maximize Responsiveness

Item 64: Consider concurrent.futures for True Parallelism

## 8. Robustness and Performance

Item 65: Take Advantage of Each Block in try/except/else/finally

Item 66: Consider contextlib and with Statements for Reusable try/finally Behavior

Item 67: Use datetime Instead of time for Local Clocks

Item 68: Make pickle Reliable with copyreg

Item 69: Use decimal When Precision Is Paramount

Item 70: Profile Before Optimizing

Item 71: Prefer deque for Producer-Consumer Queues

Item 72: Consider Searching Sorted Sequences with bisect

Item 73: Know How to Use heapq for Priority Queues

Item 74: Consider memoryview and bytearray for Zero-Copy Interactions with bytes

## 9. Testing and Debugging

Item 75: Use repr Strings for Debugging Output

Item 76: Verify Related Behaviors in TestCase Subclasses

Item 77: Isolate Tests from Each Other with setUp, tearDown, setUpModule, and tearDownModule

Item 78: Use Mocks to Test Code with Complex Dependencies

Item 79: Encapsulate Dependencies to Facilitate Mocking and Testing

Item 80: Consider Interactive Debugging with pdb

Item 81: Use tracemalloc to Understand Memory Usage and Leaks

## 10. Collaboration

Item 82: Know Where to Find Community-Built Modules

Item 83: Use Virtual Environments for Isolated and Reproducible Dependencies

Item 84: Write Docstrings for Every Function, Class, and Module

Item 85: Use Packages to Organize Modules and Provide Stable APIs

Item 86: Consider Module-Scoped Code to Configure Deployment Environments

Item 87: Define a Root Exception to Insulate Callers from APIs

Item 88: Know How to Break Circular Dependencies

Item 89: Consider warnings to Refactor and Migrate Usage

Item 90: Consider Static Analysis via typing to Obviate Bugs

• • • • • ([收起](#))

[Effective Python: 2nd Edition 下载链接1](#)

## 标签

Python

计算机

英文原版

编程语言

编程

編程

程序設計

## 评论

新版抛弃了Python 2，介绍了一些Python 3的新特性(Python 3.8)；这本书比较好的地方是，每个知识点举例比较详细，并且通过对比告诉你为什么要这么做。

-----  
增加了一些python3.8的新特性比如walrus operator，第六章metaclass略难

-----  
如果要我选一本python开发者必看的书，那么不是fluent python，也不是python cookbook，而必须是这本effective python。简直太实用了！

-----  
这本书除了第六章的metaclass以外对于新手还是比较友好的，每节末尾的小总结很不错，可以快速回顾小节内容。  
另外，作者在讲某个概念的时候，往往会从问题开始，先给出初步的解决方案，然后再慢慢改进，最后提出关键的概念，比较循序渐进。  
几个之前不知道的技巧：enumerate, F-string, generator & itertools, decoration, classmethod, decorator & metaclass, ThreadPoolExecutor, try/except/else/finally, Cython, unittest.  
不足的是作者没有去和其他语言比较，或者去讨论Python核心的一些东西，比如everything is an object等等。

-----  
[Effective Python: 2nd Edition\\_ 下载链接1](#)

## 书评

大部分知识之前已经学习过，这次就写research project会需要到的代码知识过一遍。  
1. enumerate: `for i, a in enumerate(A)` 2. zip: `for a, b in zip(A, B)` 3. list comprehension `[x for x in a]` 4. generator `(x for x in a)` 5. try/except/else/finally: 6. not use...

-----  
Item 24: Use @classmethod polymorphism to construct object generically  
可以处理如何动态构建模型的问题。 Item 25: Use `super(\_\_class\_\_, self).\_\_init\_\_()` when inheriting  
Item 29: use plain attributes instead of getter and setters.  
Shortcoming of @property...

-----  
Effective Python 59 SPECIFIC WAYS TO WRITE BETTER PYTHON  
这本书终于读完了。从这本书里学到不少经验，以及之前忽略的知识。书中部分内容也是库的内容（这么说有失公允，大部分属都会有抄库文档的嫌疑的，因为文档包含了最多的信息），也有很多内容基本上是常识，比如七八章...

-----  
其他部分(并发, 模块, 部署)大部分都了解, 更愿意看一下 OOP 的想法.  
在这里稍微记一下. 尽量使用异常来表示特殊情况, 而不要 return None 现在看, 有两种比较棘手的情况: (1) 有时候一个方法里涉及数个含网络请求(which means 必须考虑失败)的调用, 会写成这样: def call0(): ...

-----

我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了  
我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了  
我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了 我看过了  
我看过了 我看...

-----

[Effective Python: 2nd Edition\\_ 下载链接1](#)