

Effective SOFTWARE DEVELOPMENT SERIES

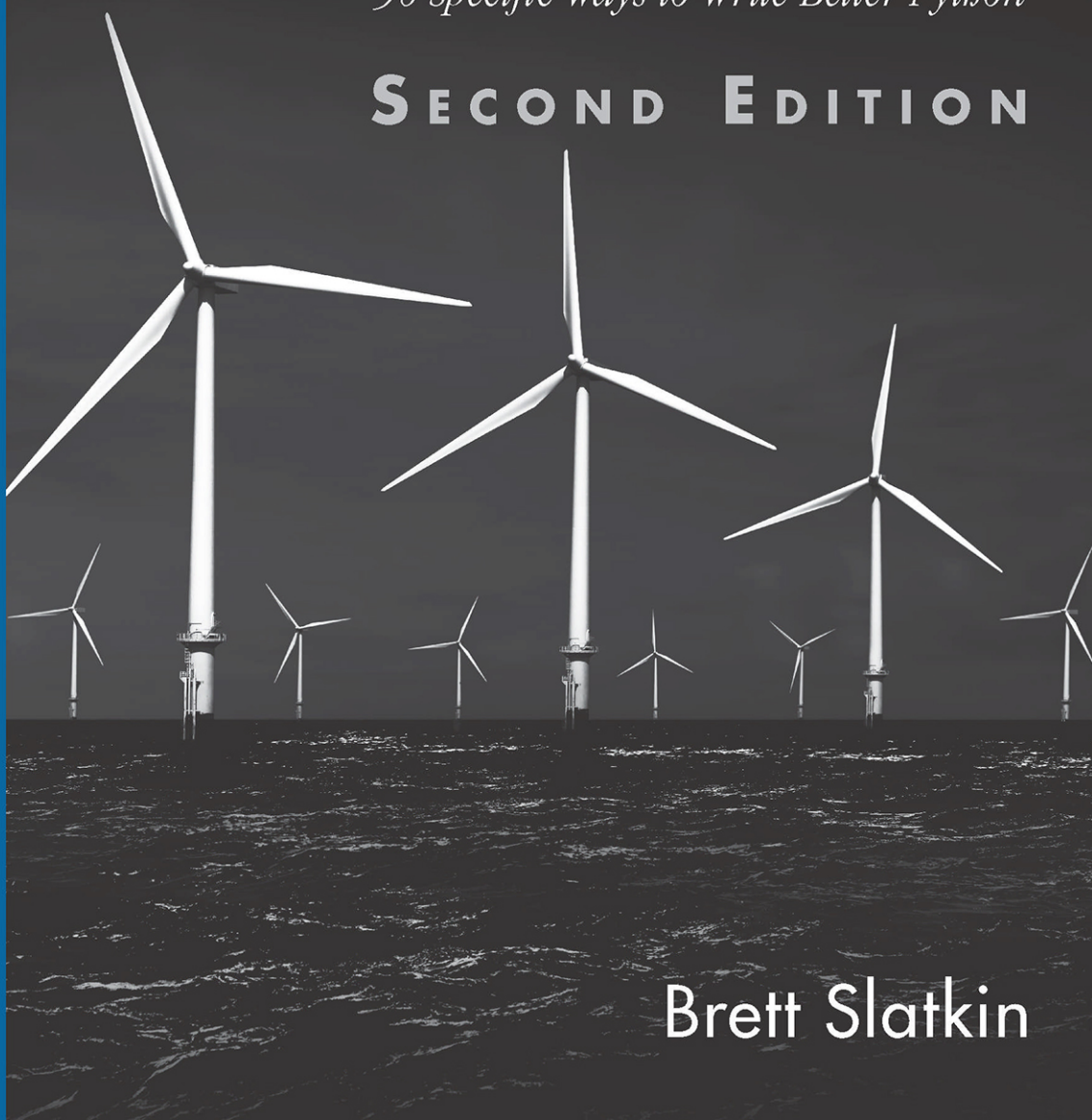
Scott Meyers, Consulting Editor



Effective PYTHON

90 Specific Ways to Write Better Python

SECOND EDITION



Brett Slatkin

Praise for *Effective Python*

"I have been recommending this book enthusiastically since the first edition appeared in 2015. This new edition, updated and expanded for Python 3, is a treasure trove of practical Python programming wisdom that can benefit programmers of all experience levels."

—Wes McKinney, *Creator of Python Pandas project, Director of Ursa Labs*

"If you're coming from another language, this is your definitive guide to taking full advantage of the unique features Python has to offer. I've been working with Python for nearly twenty years and I still learned a bunch of useful tricks, especially around newer features introduced by Python 3. *Effective Python* is crammed with actionable advice, and really helps define what our community means when they talk about Pythonic code."

—Simon Willison, *Co-creator of Django*

"I've been programming in Python for years and thought I knew it pretty well. Thanks to this treasure trove of tips and techniques, I've discovered many ways to improve my Python code to make it faster (e.g., using `bisect` to search sorted lists), easier to read (e.g., enforcing keyword-only arguments), less prone to error (e.g., unpacking with starred expressions), and more Pythonic (e.g., using `zip` to iterate over lists in parallel). Plus, the second edition is a great way to quickly get up to speed on Python 3 features, such as the walrus operator, f-strings, and the typing module."

—Pamela Fox, *Creator of Khan Academy programming courses*

"Now that Python 3 has finally become the standard version of Python, it's already gone through eight minor releases and a lot of new features have been added throughout. Brett Slatkin returns with a second edition of *Effective Python* with a huge new list of Python idioms and straightforward recommendations, catching up with everything that's introduced in version 3 all the way through 3.8 that we'll all want to use as we finally leave Python 2 behind. Early sections lay out an enormous list of tips regarding new Python 3 syntaxes and concepts like string and byte objects, f-strings, assignment expressions (and their special nickname you might not know), and catch-all unpacking of tuples. Later sections take on bigger subjects, all of which are packed with things I either didn't know or which I'm always trying to teach to others, including 'Metaclasses and Attributes' (good advice includes 'Prefer Class Decorators over Metaclasses' and also introduces a new magic method '`__init_subclass__`()' I wasn't familiar with), 'Concurrency' (favorite advice: 'Use Threads for Blocking I/O, but not Parallelism,' but it also covers `asyncio` and coroutines correctly) and 'Robustness and Performance' (advice given: 'Profile before Optimizing'). It's a joy to go through each section as everything I read is terrific best practice information smartly stated, and I'm considering quoting from this book in the future as it has such great advice all throughout. This is the definite winner for the 'if you only read one Python book this year...' contest."

—Mike Bayer, *Creator of SQLAlchemy*

Effective Python: 90 Specific Ways to Write Better Python

Table of Contents

Cover

Half Title

Title Page

Copyright Page

Dedication

Contents

Preface

Acknowledgments

About the Author

Chapter 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



Table of Contents

Chapter 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__`

Chapter 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`

Chapter 4 Comprehensions and Generators

- Item 27: Use Comprehensions Instead of map and filter
- Item 28: Avoid More Than Two Control Subexpressions in

Table of Contents

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

Chapter 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

Chapter 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

Table of Contents

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

Chapter 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

Chapter 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

Table of Contents

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 ProducerConsumer 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

Chapter 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

Chapter 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

Table of Contents

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

Index