

Contents

Preface	xiv
----------------	------------

Acknowledgments	xxi
------------------------	------------

About the Author	xxiii
-------------------------	--------------

Chapter 1 Pythonic Thinking	1
------------------------------------	----------

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

Item 2: Follow the PEP 8 Style Guide	2
--------------------------------------	---

Item 3: Know the Differences Between bytes and str	5
--	---

Item 4: Prefer Interpolated F-Strings Over C-style Format Strings and str.format	11
---	----

Item 5: Write Helper Functions Instead of Complex Expressions	21
--	----

Item 6: Prefer Multiple Assignment Unpacking Over Indexing	24
---	----

Item 7: Prefer enumerate Over range	28
-------------------------------------	----

Item 8: Use zip to Process Iterators in Parallel	30
--	----

Item 9: Avoid else Blocks After for and while Loops	32
---	----

Item 10: Prevent Repetition with Assignment Expressions	35
---	----

Chapter 2 Lists and Dictionaries	43
---	-----------

Item 11: Know How to Slice Sequences	43
--------------------------------------	----

Item 12: Avoid Striding and Slicing in a Single Expression	46
--	----

Item 13: Prefer Catch-All Unpacking Over Slicing	48
--	----

Item 14: Sort by Complex Criteria Using the key Parameter	52
---	----

Item 15: Be Cautious When Relying on dict Insertion Ordering	58
Item 16: Prefer get Over in and KeyError to Handle Missing Dictionary Keys	65
Item 17: Prefer defaultdict Over setdefault to Handle Missing Items in Internal State	70
Item 18: Know How to Construct Key-Dependent Default Values with __missing__	73

Chapter 3 Functions 77

Item 19: Never Unpack More Than Three Variables When Functions Return Multiple Values	77
Item 20: Prefer Raising Exceptions to Returning None	80
Item 21: Know How Closures Interact with Variable Scope	83
Item 22: Reduce Visual Noise with Variable Positional Arguments	87
Item 23: Provide Optional Behavior with Keyword Arguments	90
Item 24: Use None and Docstrings to Specify Dynamic Default Arguments	94
Item 25: Enforce Clarity with Keyword-Only and Positional-Only Arguments	97
Item 26: Define Function Decorators with functools.wraps	102

Chapter 4 Comprehensions and Generators 107

Item 27: Use Comprehensions Instead of map and filter	107
Item 28: Avoid More Than Two Control Subexpressions in Comprehensions	109
Item 29: Avoid Repeated Work in Comprehensions by Using Assignment Expressions	111
Item 30: Consider Generators Instead of Returning Lists	114
Item 31: Be Defensive When Iterating Over Arguments	117
Item 32: Consider Generator Expressions for Large List Comprehensions	122
Item 33: Compose Multiple Generators with yield from	124
Item 34: Avoid Injecting Data into Generators with send	127
Item 35: Avoid Causing State Transitions in Generators with throw	133

Item 36: Consider <code>itertools</code> for Working with Iterators and Generators	138
--	-----

Chapter 5 Classes and Interfaces 145

Item 37: Compose Classes Instead of Nesting Many Levels of Built-in Types	145
Item 38: Accept Functions Instead of Classes for Simple Interfaces	152
Item 39: Use <code>@classmethod</code> Polymorphism to Construct Objects Generically	155
Item 40: Initialize Parent Classes with <code>super</code>	160
Item 41: Consider Composing Functionality with Mix-in Classes	165
Item 42: Prefer Public Attributes Over Private Ones	170
Item 43: Inherit from <code>collections.abc</code> for Custom Container Types	175

Chapter 6 Metaclasses and Attributes 181

Item 44: Use Plain Attributes Instead of Setter and Getter Methods	181
Item 45: Consider <code>@property</code> Instead of Refactoring Attributes	186
Item 46: Use Descriptors for Reusable <code>@property</code> Methods	190
Item 47: Use <code>__getattr__</code> , <code>__getattribute__</code> , and <code>__setattr__</code> for Lazy Attributes	195
Item 48: Validate Subclasses with <code>__init_subclass__</code>	201
Item 49: Register Class Existence with <code>__init_subclass__</code>	208
Item 50: Annotate Class Attributes with <code>__set_name__</code>	214
Item 51: Prefer Class Decorators Over Metaclasses for Composable Class Extensions	218

Chapter 7 Concurrency and Parallelism 225

Item 52: Use <code>subprocess</code> to Manage Child Processes	226
Item 53: Use Threads for Blocking I/O, Avoid for Parallelism	230
Item 54: Use Lock to Prevent Data Races in Threads	235
Item 55: Use Queue to Coordinate Work Between Threads	238
Item 56: Know How to Recognize When Concurrency Is Necessary	248

Item 57: Avoid Creating New Thread Instances for On-demand Fan-out	252
Item 58: Understand How Using Queue for Concurrency Requires Refactoring	257
Item 59: Consider ThreadPoolExecutor When Threads Are Necessary for Concurrency	264
Item 60: Achieve Highly Concurrent I/O with Coroutines	266
Item 61: Know How to Port Threaded I/O to asyncio	271
Item 62: Mix Threads and Coroutines to Ease the Transition to asyncio	282
Item 63: Avoid Blocking the asyncio Event Loop to Maximize Responsiveness	289
Item 64: Consider concurrent.futures for True Parallelism	292

Chapter 8 Robustness and Performance 299

Item 65: Take Advantage of Each Block in try/except /else/finally	299
Item 66: Consider contextlib and with Statements for Reusable try/finally Behavior	304
Item 67: Use datetime Instead of time for Local Clocks	308
Item 68: Make pickle Reliable with copyreg	312
Item 69: Use decimal When Precision Is Paramount	319
Item 70: Profile Before Optimizing	322
Item 71: Prefer deque for Producer–Consumer Queues	326
Item 72: Consider Searching Sorted Sequences with bisect	334
Item 73: Know How to Use heapq for Priority Queues	336
Item 74: Consider memoryview and bytearray for Zero-Copy Interactions with bytes	346

Chapter 9 Testing and Debugging 353

Item 75: Use repr Strings for Debugging Output	354
Item 76: Verify Related Behaviors in TestCase Subclasses	357
Item 77: Isolate Tests from Each Other with setUp, tearDown, setUpModule, and tearDownModule	365
Item 78: Use Mocks to Test Code with Complex Dependencies	367

Item 79: Encapsulate Dependencies to Facilitate Mocking and Testing	375
Item 80: Consider Interactive Debugging with pdb	379
Item 81: Use tracemalloc to Understand Memory Usage and Leaks	384

Chapter 10 Collaboration **389**

Item 82: Know Where to Find Community-Built Modules	389
Item 83: Use Virtual Environments for Isolated and Reproducible Dependencies	390
Item 84: Write Docstrings for Every Function, Class, and Module	396
Item 85: Use Packages to Organize Modules and Provide Stable APIs	401
Item 86: Consider Module-Scoped Code to Configure Deployment Environments	406
Item 87: Define a Root Exception to Insulate Callers from APIs	408
Item 88: Know How to Break Circular Dependencies	413
Item 89: Consider warnings to Refactor and Migrate Usage	418
Item 90: Consider Static Analysis via typing to Obviate Bugs	425

Index **435**

What This Book Covers

Each chapter in this book contains a broad but related set of items. Feel free to jump between items and follow your interest. Each item contains concise and specific guidance explaining how you can write Python programs more effectively. Items include advice on what to do, what to avoid, how to strike the right balance, and why this is the best choice. Items reference each other to make it easier to fill in the gaps as you read.

This second edition of this book is focused exclusively on Python 3 (see Item 1: "Know Which Version of Python You're Using"), up to and including version 3.8. Most of the original items from the first edition have been revised and included, but many have undergone substantial updates. For some items, my advice has completely changed between the two editions of the book due to best practices evolving as Python has matured. If you're still primarily using Python 2, despite