Better PYTHON CODE

David **Mertz**

Foreword by Alex Martelli

Praise for *Better Python Code*

"You'll not just be aspiring to be an expert anymore after practicing through *Better Python Code: A Guide for Aspiring Experts*, you'll be one of them! Learn from David Mertz, who's been making experts through his writing and training for the past 20 years."
—Iqbal Abdullah, past Chair, PyCon Asia Pacific, and past board member, PyCon Japan

"In *Better Python Code: A Guide for Aspiring Experts*, David Mertz serves up bite-sized chapters of Pythonic wisdom in this must-have addition to any serious Python programmer's collection. This book helps bridge the gap from beginner to advanced Python user, but even the most seasoned Python programmer can up their game with Mertz's insight into the ins and outs of Python."
—Katrina Riehl, President, NumFOCUS

"What separates ordinary coders from Python experts? It's more than just knowing best practices—it's understanding the benefits and pitfalls of the many aspects of Python, and knowing when and why to choose one approach over another. In this book David draws on his more than 20 years of involvement in the Python ecosystem and his experience as a Python author to make sure that the readers understand both *what* to do and *why* in a wide variety of scenarios."
—Naomi Ceder, past Chair, Python Software Foundation

"Like a Pythonic BBC, David Mertz has been informing, entertaining, and educating the Python world for over a quarter of a century, and he continues to do so here in his own pleasantly readable style."
—Steve Holden, past Chair, Python Software Foundation

"Being expert means someone with a lot of experience. David's latest book provides some important but common problems that folks generally learn only after spending years of doing and fixing. I think this book will provide a much quicker way to gather those important bits and help many folks across the world to become better."
—Kushal Das, CPython Core Developer and Director, Python Software Foundation

"This book is for everyone: from beginners, who want to avoid hard-to-find bugs, all the way to experts looking to write more efficient code. David Mertz has compiled a great set of useful idioms that will make your life as a programmer easier and your users happier."
—Marc-André Lemburg, past Chair, EuroPython, and past Director, Python Software Foundation

Better Python Code: A Guide for Aspiring Experts

Table of Contents

Cover
Half Title
Title Page
Copyright Page
Contents
Foreword
Preface
Acknowledgments
AbouttheAuthor
Introduction
1 Looping Over the Wrong Things
- 1.1 (Rarely) Generate a List for Iteration
- 1.2 Use enumerate() Instead of Looping Over an Index
- 1.3 Dont Iterate Over dict.keys() When You Want dict.items()
- 1.4 Mutating an Object During Iteration
- 1.5 for Loops Are More Idiomatic Than while Loops
- 1.6 The Walrus Operator for Loop-and-a-Half Blocks
- 1.7 zip() Simplifies Using Multiple Iterables
- 1.8 zip(strict=True) and itertools.zip_longest()
- 1.9 Wrapping Up

Pearson

Table of Contents

2 Confusing Equality with Identity

2.1 Late Binding of Closures

2.2 Overchecking for Boolean Values

2.3 Comparing x == None

2.4 Misunderstanding Mutable Default Arguments

2.4.1 First Approach, Usea Class

2.4.2 Second Approach, Use a None Sentinel

2.4.3 Third Approach, Take Advantage of Stateful Generators

2.5 Copies versus References to Mutable Objects

2.6 Confusing is with == (in the Presence of Interning)

2.7 Wrapping Up

3 A Grab Bag of Python Gotchas

3.1 Naming Things

3.1.1 Naming a File Identically to a Standard Library Module

3.1.2 Avoid Using import *

3.1.3 Bare or Overly Generic except Statements

3.2 Quadratic Behavior of Naive String Concatenation

3.3 Usea Context Manager to Opena File

3.3.1 First Danger

3.3.2 Second Danger

3.3.3 Correcting the Fragility

3.4 Optional Argument key to .sort() and sorted()

3.5 Usedict .get() for Uncertain Keys

3.6 Wrapping Up

4 Advanced Python Usage

4.1 Comparing type(x) == type(y)

4.2 Naming Things (Revisited)

Table of Contents

4.2.1 Overriding Names in Built-ins

4.2.2 Directly Accessing a Protected Attribute

4.3 Keep Less-Used Features in Mind

4.3.1 F-String Debugging

4.3.2 The Elegant Magic of Decorators

4.3.3 Use itertools (Sufficiently)

4.3.4 The more-itertools Third-Party Library

4.4 Type Annotations Are Not Runtime Types

4.4.1 Type Annotations Are Not Runtime Constraints

4.4.2 Mistaking typing.NewType() for a Runtime Type

4.5 Wrapping Up

5 Just Because You Can, It Doesnt Mean You Should

5.1 Metaclasses

5.2 Monkey patching

5.3 Getters and Setters

5.4 Its Easier to Ask for Forgiveness Than Permission

5.5 Structural Pattern Matching

5.6 Regular Expressions and Catastrophic Backtracking

5.7 Wrapping Up

6 Picking the Right Data Structure

6.1 collections. defaultdict

6.2 collections.Counter

6.2.1 The Solution

6.2.2 The Mistake

6.3 collections.deque

6.3.1 The Solution

6.3.2 The Mistake

Pearson

Table of Contents

- 6.4 collections.ChainMap
  - 6.4.1 The Solution
  - 6.4.2 The Mistake
- 6.5 Dataclasses and Namedtuples
  - 6.5.1 Using Namedtuples
  - 6.5.2 Static versus Dynamic
  - 6.5.3 Data Classes
- 6.6 Efficient Concrete Sequences
- 6.7 Wrapping Up

7 Misusing Data Structures
- 7.1 Quadratic Behavior of Repeated List Search
- 7.2 Deleting or Adding Elements to the Middle of a List
  - 7.2.1 More Efficient Data Structures
- 7.3 Strings Are Iterables of Strings
- 7.4 (Often) Use enum Rather Than CONSTANT
- 7.5 Learn Less Common Dictionary Methods
  - 7.5.1 The Dictionaries Defining Objects
  - 7.5.2 Back to Our Regularly Scheduled Mistake
- 7.6 JSON Does Not Round-Trip Cleanly to Python
  - 7.6.1 Some Back ground on JSON
  - 7.6.2 Data That Fails to Round-Trip
- 7.7 Rolling Your Own Data Structures
  - 7.7.1 When Rolling Your Own Is a Bad Idea
  - 7.7.2 When Rolling Your Own Is a Good Idea
  - 7.7.3 Takeaways
- 7.8 Wrapping Up

8 Security

Pearson

Table of Contents

8.1 Kinds of Randomness

8.1.1 Use secrets for Cryptographic Randomness

8.1.2 Reproducible Random Distributions

8.2 Putting Passwords or Other Secrets in Secure Source Code

8.3 Rolling Your Own Security Mechanisms

8.4 Use SSL/TLS for Microservices

8.5 Using the Third-Party requests Library

8.6 SQL Injection Attacks When Not Using DB-API

8.7 Dont Use assert to Check Safety Assumptions

8.8 Wrapping Up

9 Numeric Computation in Python

9.1 Understanding IEEE-754 Floating Point Numbers

9.1.1 Comparing NaNs (and Other Floating Point Numbers)

9.1.2 NaNs and statistics.median()

9.1.3 Naive Use of Floating Point Numbers: Associativity and Distributivity

9.1.4 Naive Use of Floating Point Numbers: Granularity

9.2 Numeric Datatypes

9.2.1 Avoid Floating Point Numbers for Financial Calculations

9.2.2 Nonobvious Behaviors of Numeric Datatypes

9.3 Wrapping Up

A Appendix: Topics for Other Books

A.1 Test-Driven Development

A.2 Concurrency

A.3 Packaging

A.4 Type Checking

A.5 Numeric and Dataframe Libraries

Index

Pearson

Table of Contents