# Practices of the Python Pro

Dane Hillard





Practices of the Python Pro

# Practices of the Python Pro

DANE HILLARD



For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department Manning Publications Co. 20 Baldwin Road PO Box 761 Shelter Island, NY 11964 Email: orders@manning.com

©2020 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.



Manning Publications Co. 20 Baldwin Road PO Box 761 Shelter Island, NY 11964 Development editor: Toni Arritola Technical development editor: Nick Watts Review editor: Aleks Dragosavljević Production editor: Lori Weidert Copy editor: Andy Carroll Proofreader: Carl Quesnel Technical proofreader: Jens Christian Bredahl Madson Typesetter: Gordan Salinovic Cover designer: Marija Tudor

ISBN 9781617296086 Printed in the United States of America

# brief contents

<ul> <li>1 The bigger picture 3</li> <li>PART 2 FOUNDATIONS OF DESIGN.</li> <li>2 Separation of concerns 19</li> <li>3 Abstraction and encapsulation 41</li> </ul>	1
<ul> <li>PART 2 FOUNDATIONS OF DESIGN.</li> <li>2 Separation of concerns 19</li> <li>3 Abstraction and encapsulation 41</li> </ul>	
<ul> <li>2 Separation of concerns 19</li> <li>3 Abstraction and encapsulation 41</li> </ul>	17
<ul> <li>4 Designing for high performance 58</li> <li>5 Testing your software 77</li> </ul>	
<ul> <li>PART 3 NAILING DOWN LARGE SYSTEMS</li> <li>6 Separation of concerns in practice 103</li> <li>7 Extensibility and flexibility 127</li> <li>8 The rules (and exceptions) of inheritance 143</li> <li>9 Keeping things lightweight 160</li> <li>10 Achieving loose coupling 177</li> </ul>	. 101
PART 4 WHAT'S NEXT?	.197

### contents

preface xiii acknowledgments xv about this book xvii about the author xxi about the cover illustration xxii

#### PART 1 WHY IT ALL MATTERS......1

#### The bigger picture 3

1.1 Python is an enterprise language 4				
	The times they are a-changin' 5 • What I like about Python	5		

1.2 Python is a teaching language 5

- 1.3 Design is a process 6 The user experience 7 • You've been here before 8
- 1.4 Design enables better software 8 Considerations in software design 9 • Organically grown software 10
- 1.5 When to invest in design 11
- 1.6 New beginnings 12

- 1.7 Design is democratic 12 Presence of mind 13
- 1.8 How to use this book 15

#### Part 2 Foundations of design ......17

Separation of concerns 19

- 2.1 Namespacing 20 Namespaces and the import statement 20 • The many masks of importing 23 • Namespaces prevent collisions 24
- 2.2 The hierarchy of separation in Python 25 Functions 26 - Classes 32 - Modules 37 - Packages 37

#### $\bigcirc$ Abstraction and encapsulation 41

- What is abstraction? 42
   The "black box" 42 Abstraction is like an onion 43
   Abstraction is a simplifier 45 Decomposition enables abstraction 46
- 3.2 Encapsulation 47 Encapsulation constructs in Python 47 • Expectations of privacy in Python 48
- 3.3 Try it out 48 Refactoring 50
- 3.4 Programming styles are an abstraction too 51 Procedural programming 51 • Functional programming 51 Declarative programming 53
- 3.5 Typing, inheritance, and polymorphism 54
- 3.6 Recognizing the wrong abstraction 56 Square pegs in round holes 56 - Clever gets the cleaver 57

#### **Designing for high performance** 58

4.1 Hurtling through time and space 59
 Complexity is a little . . . complex 59 • Time complexity 60
 Space complexity 63

## 4.2 Performance and data types 64 Data types for constant time 64 • Data types for linear time 65 Space complexity of operations on data types 65

	4.3	Make it work, make it right, make it fast 68 Making it work 68 • Making it right 68 • Making it fast 71
	4.4	Tools 72 timeit 72 • CPU profiling 73
	4.5	Try it out 75
5	Testin	ng your software 77
	5.1	What is software testing? 78
		Does it do what it says on the tin? 78 - The anatomy of a functional test 78
	5.2	Functional testing approaches 79 Manual testing 80 • Automated testing 80 • Acceptance testing 80 • Unit testing 82 • Integration testing 83 The testing pyramid 84 • Regression testing 85
	5.3	Statements of fact 86
	5.4	Unit testing with unittest 86
		Test organization with unittest 86 • Running tests with unittest 87 • Writing your first test with unittest 87 Writing your first integration test with unittest 90 • Test doubles 92 • Try it out 93 • Writing interesting tests 96
	5.5	Testing with pytest 96
		Test organization with pytest 97 - Converting unittest tests to pytest 97
	5.6	Beyond functional testing 98 Performance testing 98 • Load testing 99
	5.7	Test-driven development: A primer99It's a mindset100 • It's a philosophy100
PART 3	VAILIN	ig down large systems10

#### Separation of concerns in practice 103

- 6.1 A command-line bookmarking application 104
- 6.2 A tour of Bark 105 The benefits of separation: Reprise 105
- 6.3 An initial code structure, by concern 106 The persistence layer 107 • The business logic layer 115 The presentation layer 119

1

Extensibility and flexibility 127 7.1What is extensible code? 127 Adding new behaviors 128 • Modifying existing behaviors 130 Loose coupling 131 7.2133Solutions for rigidity Letting go: Inversion of control 133 • The devil's in the details: Relying on interfaces 136 • Fighting entropy: The robustness principle 137 7.3An exercise in extension 138 The rules (and exceptions) of inheritance 143 8.1 The inheritance of programming past 143 The silver bullet 144 • The challenges of hierarchies 144 8.2 The inheritance of programming present 146 What is inheritance for, really? 146 • Substitutability 147 The ideal use case for inheritance 148 8.3 Inheritance in Python 150Type inspection 150 • Superclass access 151 • Multiple inheritance and method resolution order 152 - Abstract base classes 155 8.4 Inheritance and composition in Bark 157 Refactoring to use an abstract base class 157 - A final check on your inheritance work 159 Keeping things lightweight 160 9.1 How big should my class/function/module be? 161Physical size 161 • Single responsibility 161 • Code complexity 162 9.2 Breaking down complexity 166 Extracting configuration 166 • Extracting functions 168 9.3 Decomposing classes 170 Initialization complexity 171 • Extracting classes and forwarding calls 173 Achieving loose coupling 177 10.1Defining coupling 177The connective tissue 178 - Tight coupling 178 - Loose coupling 181

10	0.2 Recognizing coupling 184 Feature envy 184 • Shotgun surgery 184 • Leaky abstractions 185
10	0.3 Coupling in Bark 186
10	0.4 Addressing coupling 188 User messaging 189 • Bookmark persistence 191 • Try it out 192
PART 4 WH	at's next?197
11 01	ward and upward 199
	1.1 What now? 199 Develop a plan 200 • Execute the plan 201 • Track your progress 203
1	.2 Design patterns 204 Ups and downs of design patterns in Python 206 • Terms to start with 206
13	1.3 Distributed systems 206 Modes of failure in distributed systems 207 • Addressing application state 208 • Terms to start with 208
13	<ul> <li>.4 Take a Python deep dive 208</li> <li>Python code style 208 - Language features are patterns 209 Terms to start with 210</li> </ul>
11	.5 Where you've been 210 There and back again: A developer's tale 210 • Signing off 212
appen	dix Installing Python 213 index 217

preface

Python, like me, was born in December of 1989. Although I've accomplished a great deal in the subsequent three decades, Python's success is prolific. More people than ever before are picking it up to accomplish fascinating things in data science, machine learning, and more. Since I learned Python, this "second-best language for every-thing" has in reality been my first choice for many endeavors.

I had a rather traditional path into programming through the Electrical Engineering and Computer Science Department at the University of Michigan. At that time, the coursework focused mainly on C++ and MATLAB—languages I continued to use in my first job out of school. I developed some shell scripting and SQL chops in my next position, processing big data for bioinformatics. I also started using PHP to work on a personal WordPress site from scratch.

Although I was getting results (and cool ones, in some cases), none of the languages I was using *resonated* with me. But I was oblivious. I assumed that programming languages were purely means to an end, and they had little chance of being *fun* to work with. Around this time, a friend invited me to join him in a hackathon project to build a Ruby library.

The world exploded with color, fruits tasted sweeter, and all that. The ease of using an interpreted language and the human-friendly syntax of Ruby really made me think about the tools I'd been using. Although I didn't stick with Ruby for too long, I decided to give Python and the Django web framework a try for the next iteration of my personal site. It gave me the same joy and shallow learning curve I'd seen with Ruby, and I haven't looked back since! Now that Python is recognized widely as a language of choice for many tasks, folks coming into software development don't need to go through the trial and error process I did. New and interesting pathways into a career in software are opening up all around too. Despite these differences, I hope we can all share in the common experience of finding joy in programming with Python. I also hope this book can contribute to that joy.

Come along on the wonderful Python journey I fell into somewhat haphazardly. I want to see you build a website, a data pipeline, or an automated plant-watering system. Whatever you fancy. Python's got your back. Send photos and code samples of your projects to python-pro-projects@danehillard.com.

## acknowledgments

I didn't write this book alone. My appreciation runs deep for everyone who helped me along the way, at every stage and in every capacity. You are loved.

Most anyone who's been involved in the production of a book can tell you that it's always more work than you think. I heard this many times throughout the process, and it certainly was a lot of work. What's not always clear is that the real struggle is balancing all that extra work with your existing life.

To my partner, Stefanie: your support, encouragement, and tolerance of my ranting and raving were paramount in making this book a reality. Thank you for judging my neglect lightly and extricating me from this project during the roughest times. I could not have done this without you.

Thank you to my parents, Kim and Donna, for always funneling my energy toward curiosity, creativity, and compassion.

Thanks to my dear friend Vincent Zhang for spending countless nights at the coffee shop coding by my side. You were there when the concept for this book was born, and your validation helped spur me to take on this endeavor.

Thank you to James Nguyen for persevering as you changed paths to become a developer. You embody the audience for this book, and your input has been invaluable. I'm proud of your accomplishments.

My gratitude goes to all my colleagues at ITHAKA and beyond for your input and support. I thank you for enduring what has undoubtedly been a flighty period for me.

#### ACKNOWLEDGMENTS

To Toni Arritola, my editor: thank you for your determination in pushing me ever toward higher-quality teaching. The writing process is fraught with many unexpected snags, but you provided me consistency and stability. Thank you.

To Nick Watts, my technical editor: your feedback has pushed the content of this book from frantic ramblings to plausible software teachings. Your candor and insight are much appreciated.

Thank you to Mike Stephens and Marjan Bace at Manning for believing in this idea and trusting me as its shepherd. Thank you to everyone at Manning for working tirelessly to bring authors' ideas to life.

To all the reviewers—Al Krinker, Bonnie Bailey, Burkhard Nestmann, Chris Wayman, David Kerns, Davide Cadamuro, Eriks Zelenka, Graham Wheeler, Gregory Matuszek, Jean-François Morin, Jens Christian Bredahl Madsen, Joseph Perenia, Mark Thomas, Markus Maucher, Mike Stevens, Patrick Regan, Phil Sorensen, Rafael Cassemiro Freire, Richard Fieldsend, Robert Walsh, Steven Parr, Sven Stumpf, and Willis Hampton—your suggestions helped make this a better book.

A final thank you to anyone and everyone else who has had a positive influence directly, intentionally, or otherwise—on my journey in programming and this book. I cannot hope to produce an exhaustive list; names not appearing here are due expressly to the limitations of my own mind. Thank you to Mark Brehob, Dr. Andrew DeOrio, Jesse Sielaff, Trek Glowacki, everyone at SAIC (in our little Ann Arbor office), everyone at Compendia Bioscience (and friends), Brandon Rhodes, Kenneth Love, Trey Hunner, Jeff Triplett, Mariatta Wijaya, Ali Spittel, Chris Coyier, Sarah Drasner, David Beazley, Dror Ayalon, Tim Allen, Sandi Metz, and Martin Fowler.

## about this book

*Practices of the Python Pro* introduces several concepts that software developers in almost any language can use to improve their work. This would be a great book to read after learning the fundamentals of the Python language.

#### Who should read this book

*Practices of the Python Pro* is for anyone in the early stages of their programming journey. In fact, people outside the software industry altogether who use software to supplement their work can find value in this book. The concepts contained in these pages will help readers build software that's more maintainable, which in turn makes their software easier to collaborate on.

In the sciences, reproducibility and provenance are important aspects of the research process. As more research comes to rely on software, code that people can understand, update, and improve is a major consideration. But college curricula are still catching up to this intersection of software with other disciplines. For those with limited experience in formal software development, this book provides a set of principles for producing shareable, reusable software.

If you're seasoned in object-oriented programming and domain-driven design, you may find this book too introductory for your benefit. On the other hand, if you're relatively new to Python, software, or software design, give this book a try. There's something in here for you.

#### How this book is organized: A roadmap

*Practices of the Python Pro* consists of 11 chapters in 4 parts. Parts 1 and 2 provide discussion along with short examples and an occasional exercise. Part 3 builds on what you've learned in earlier chapters and contains a variety of exercises. Part 4 provides strategies for learning more, along with recommendations about what to try after reading this book.

Part 1, "Why it all matters," sets the stage for Python's rise to fame and why software design is valuable.

• Chapter 1 covers some recent history of Python and why I enjoy developing Python programs. It goes on to explain software design, why it's important, and how it manifests in your day-to-day work.

Part 2, "Foundations of design," covers the high-level concepts that underpin software design and development.

- Chapter 2 covers separation of concerns, a fundamental activity that provides a basis for several others in the book.
- Chapter 3 explains abstraction and encapsulation, showing you how hiding information and providing simpler interfaces to more complex logic helps you keep a handle on your code.
- Chapter 4 prompts you to think about performance, covering different data structures, approaches, and tools to help you build speedy programs.
- Chapter 5 teaches you about testing your software, using a variety of approaches, from unit testing to end-to-end testing.

Part 3, "Nailing down large systems," walks you through building a real application using the principles you've learned.

- Chapter 6 introduces the application you'll build in the book and provides exercises for creating a program's foundation.
- Chapter 7 covers the concepts of extensibility and flexibility and includes exercises that add extensibility to the application.
- Chapter 8 helps you understand class inheritance, providing recommendations about where and when it should be used. It continues on with exercises that examine inheritance in the application you're building.
- Chapter 9 steps back a bit, introducing tools and an approach for keeping code from growing too large as you go along.
- Chapter 10 explains loose coupling, providing some final exercises to reduce the coupling in the application you're building.

Part 4, "What's next?" gives you some recommendations for how and what to learn next.

 Chapter 11 shows you how I map out new learning material and gives you a few areas of study to try if you're interested in going deeper into software development.