Clean Code in Python

Second Edition

Develop maintainable and efficient code

Mariano Anaya

Packt

BIRMINGHAM - MUMBAI

Clean Code in Python

Second Edition

Copyright © 2020 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Producers: Tushar Gupta
Acquisition Editor – Peer Reviews: Divya Mudaliar
Content Development Editor: Bhavesh Amin
Technical Editor: Karan Sonawane
Project Editor: Mrunal Dave
Copy Editor: Safis Editing
Proofreader: Safis Editing
Indexer: Tejal Soni
Presentation Designer: Ganesh Bhadwalkar

First published: August 2018
Second Edition: December 2020

Production reference: 2281220

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-80056-021-5

`www.packt.com`

Table of Contents

Preface **ix**

Chapter 1: Introduction, Code Formatting, and Tools **1**

Introduction **2**
The meaning of clean code 2
The importance of having clean code 3
Some exceptions 5
Code formatting **6**
Adhering to a coding style guide on your project 6
Documentation **9**
Code comments 9
Docstrings 10
Annotations 13
Do annotations replace docstrings? 17
Tooling **19**
Checking type consistency 20
Generic validations in code 22
Automatic formatting 23
Setup for automatic checks 26
Summary **28**
References **28**

Chapter 2: Pythonic Code **29**

Indexes and slices **30**
Creating your own sequences 32
Context managers **34**
Implementing context managers 37
Comprehensions and assignment expressions **40**

Properties, attributes, and different types of methods for objects **43**
Underscores in Python 44
Properties 46
Creating classes with a more compact syntax 49
Iterable objects 52
Creating iterable objects 53
Creating sequences 56
Container objects 57
Dynamic attributes for objects 59
Callable objects 61
Summary of magic methods 63
Caveats in Python **64**
Mutable default arguments 64
Extending built-in types 66
A brief introduction to asynchronous code **68**
Summary **70**
References **71**

Chapter 3: General Traits of Good Code 73

Design by contract **74**
Preconditions 76
Postconditions 76
Pythonic contracts 77
Design by contract – conclusions 77
Defensive programming **78**
Error handling 78
Value substitution 79
Exception handling 80
Using assertions in Python 88
Separation of concerns **90**
Cohesion and coupling 91
Acronyms to live by **92**
DRY/OAOO 92
YAGNI 94
KIS 95
EAFP/LBYL 97
Inheritance in Python **99**
When inheritance is a good decision 99
Anti-patterns for inheritance 100
Multiple inheritance in Python 103
Method Resolution Order (MRO) 104
Mixins 106
Arguments in functions and methods **107**

How function arguments work in Python 107
How arguments are copied to functions 107
Variable number of arguments 109
Positional-only parameters 114
Keyword-only arguments 115
The number of arguments in functions 117
Function arguments and coupling 117
Compact function signatures that take too many arguments 118
Final remarks on good practices for software design 120
Orthogonality in software 120
Structuring the code 122
Summary 123
References 124
Chapter 4: The SOLID Principles 125
The single responsibility principle 126
A class with too many responsibilities 126
Distributing responsibilities 128
The open/closed principle 129
Example of maintainability perils for not following the OCP 130
Refactoring the events system for extensibility 133
Extending the events system 135
Final thoughts about the OCP 137
Liskov's substitution principle 137
Detecting LSP issues with tools 138
Using mypy to detect incorrect method signatures 138
Detecting incompatible signatures with pylint 140
More subtle cases of LSP violations 141
Remarks on the LSP 144
Interface segregation 144
An interface that provides too much 146
The smaller the interface, the better 146
How small should an interface be? 148
Dependency inversion 149
A case of rigid dependencies 150
Inverting the dependencies 150
Dependency injection 152
Summary 154
References 155
Chapter 5: Using Decorators to Improve Our Code 157
What are decorators in Python? 158
Function decorators 159
Decorators for classes 160

Other types of decorator 165
More advanced decorators 165
Passing arguments to decorators 165
Decorators with nested functions 166
Decorator objects 168
Decorators with default values 169
Decorators for coroutines 172
Extended syntax for decorators 175
Good uses for decorators 176
Adapting function signatures 176
Validating parameters 178
Tracing code 178
Effective decorators – avoiding common mistakes 178
Preserving data about the original wrapped object 179
Dealing with side effects in decorators 181
Incorrect handling of side effects in a decorator 182
Requiring decorators with side effects 184
Creating decorators that will always work 186
Decorators and clean code 189
Composition over inheritance 189
The DRY principle with decorators 192
Decorators and separation of concerns 193
Analysis of good decorators 195
Summary 197
References 197
Chapter 6: Getting More Out of Our Objects with Descriptors 199
A first look at descriptors 200
The machinery behind descriptors 200
Exploring each method of the descriptor protocol 203
The get method 203
The set method 205
The delete method 207
The set name method 210
Types of descriptors 211
Non-data descriptors 212
Data descriptors 214
Descriptors in action 217
An application of descriptors 217
A first attempt without using descriptors 217
The idiomatic implementation 219
Different forms of implementing descriptors 222
The issue of shared state 222
Accessing the dictionary of the object 223

Using weak references 224
More considerations about descriptors 225
Reusing code 225
An alternative to class decorators 226
Analysis of descriptors 230
How Python uses descriptors internally 230
Functions and methods 230
Built-in decorators for methods 234
Slots 235
Implementing descriptors in decorators 237
Final remarks about descriptors 237
Interface of descriptors 237
Object-oriented design of the descriptors 238
Type annotations on descriptors 238
Summary 239
References 240
Chapter 7: Generators, Iterators, and Asynchronous Programming 241
Technical requirements 242
Creating generators 242
A first look at generators 242
Generator expressions 246
Iterating idiomatically 247
Idioms for iteration 247
The next() function 249
Using a generator 250
Itertools 251
Simplifying code through iterators 252
The iterator pattern in Python 255
Coroutines 259
The methods of the generator interface 260
close() 260
throw(ex_type[, ex_value[, ex_traceback]]) 261
send(value) 263
More advanced coroutines 266
Returning values in coroutines 266
Delegating into smaller coroutines – the 'yield from' syntax 268
Asynchronous programming 273
Magic asynchronous methods 275
Asynchronous context managers 276
Other magic methods 277
Asynchronous iteration 278
Asynchronous generators 280
Summary 281

References 282

Chapter 8: Unit Testing and Refactoring 285

Design principles and unit testing 286

A note about other forms of automated testing 287

Unit testing and agile software development 288

Unit testing and software design 289

Defining the boundaries of what to test 293

Tools for testing 293

Frameworks and libraries for unit testing 294

unittest 295

pytest 301

Code coverage 306

Mock objects 309

Refactoring 315

Evolving our code 316

Production code isn't the only one that evolves 318

More about testing 319

Property-based testing 320

Mutation testing 320

Common themes in testing 323

Boundaries or limit values 323

Classes of equivalence 323

Edge cases 324

A brief introduction to test-driven development 325

Summary 325

References 326

Chapter 9: Common Design Patterns 327

Design pattern considerations in Python 328

Design patterns in action 329

Creational patterns 330

Factories 330

Singleton and shared state (monostate) 331

Builder 336

Structural patterns 337

Adapter 337

Composite 339

Decorator 340

Facade 343

Behavioral patterns 344

Chain of responsibility 345

The template method 348

Command 349

State 350

The null object pattern **355**
Final thoughts about design patterns **358**
The influence of patterns over the design 358
Design patterns as theory 359
Names in our models 360
Summary **360**
References **361**
Chapter 10: Clean Architecture **363**
From clean code to clean architecture **364**
Separation of concerns 364
Monolithic applications and microservices 366
Abstractions 367
Software components **368**
Packages 369
Managing dependencies 372
Other considerations when managing dependencies 374
Artifact versions 376
Docker containers 376
Use case 378
The code 379
Domain models 380
Calling from the application 382
Adapters 383
The services 384
Analysis 388
The dependency flow 388
Limitations 389
Testability 389
Intention revealing 390
Summary **390**
References **391**
Summing it all up **391**
Share your experience 392
Other Books You May Enjoy **393**
Index **395**