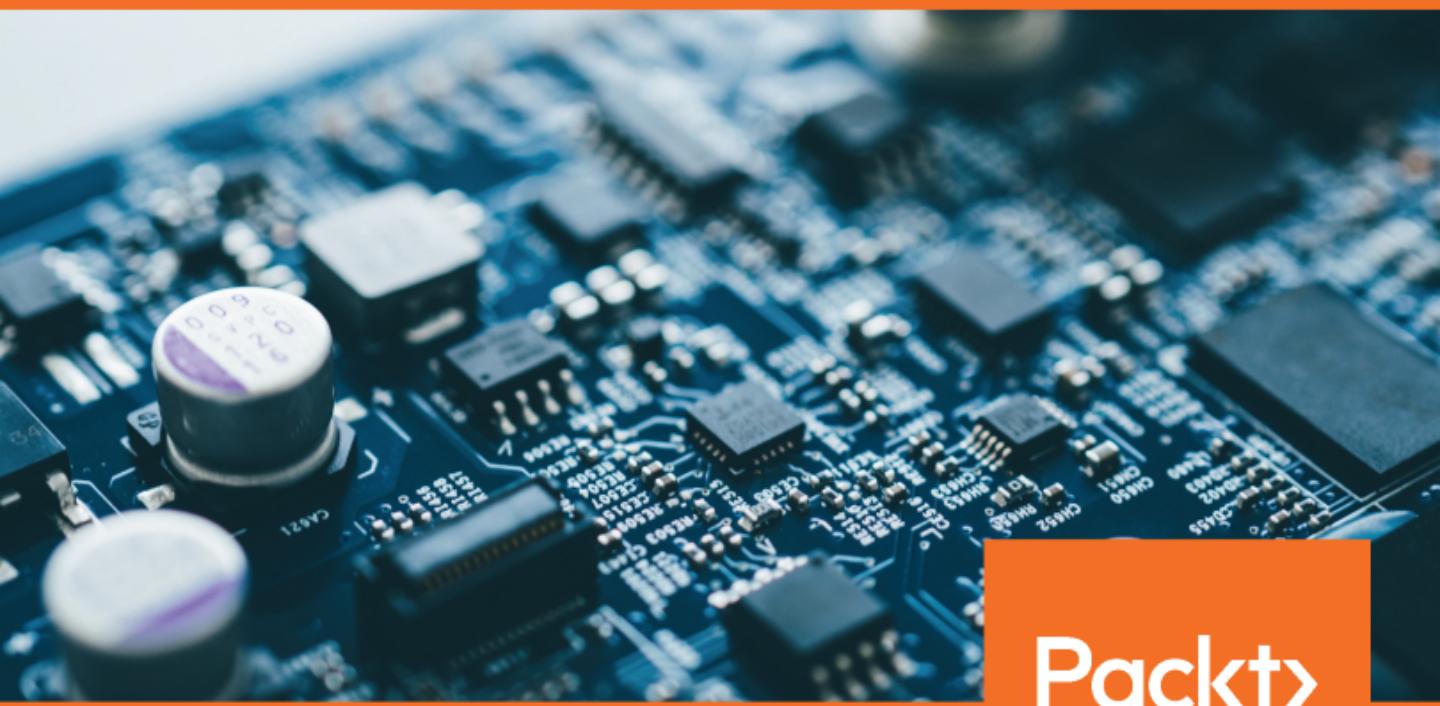


# Clean Code in Python



Refactor your legacy code base



Packt

[www.packt.com](http://www.packt.com)

By Mariano Anaya

# Clean Code in Python



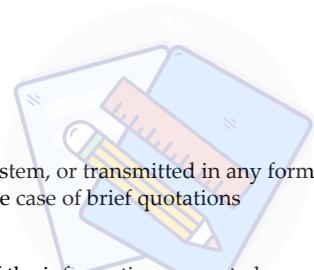
Refactor your legacy code base

**Mariano Anaya**

**Packt**

BIRMINGHAM - MUMBAI

# Clean Code in Python



Copyright © 2018 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.

**Commissioning Editor:** Merint Mathew

**Acquisition Editor:** Denim Pinto

**Content Development Editor:** Priyanka Sawant

**Technical Editor:** Gaurav Gala

**Copy Editor:** Safis Editing

**Project Coordinator:** Vaidehi Sawant

**Proofreader:** Safis Editing

**Indexer:** Rekha Nair

**Graphics:** Jason Monteiro

**Production Coordinator:** Shantanu Zagade

First published: August 2018

Production reference: 1270818

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78883-583-1

[www.packtpub.com](http://www.packtpub.com)



*To my family and friends, for their unconditional love and support.*

*– Mariano Anaya*



[mapt.io](https://mapt.io)

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

## Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

## PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](https://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](https://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

# Contributors



## About the author

**Mariano Anaya** is a software engineer who spends most of his time creating software with Python and mentoring fellow programmers. Mariano's main areas of interests besides Python are software architecture, functional programming, distributed systems, and speaking at conferences.

He was a speaker at Euro Python 2016 and 2017. To know more about him, you can refer to his GitHub account with the username [rmariano](#).

His speakerdeck username is [rmariano](#).

## About the reviewer

**Nimesh Kiran Verma** has a dual degree in Maths and Computing from IIT Delhi and has worked with companies such as LinkedIn, Paytm and ICICI for about 5 years in software development and data science.

He co-founded a micro-lending company, Upwards Fintech and presently serves as its CTO. He loves coding and has mastery in Python and its popular frameworks, Django and Flask.

He extensively leverages Amazon Web Services, design patterns, SQL and NoSQL databases, to build reliable, scalable and low latency architectures.

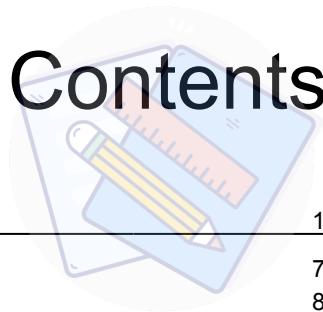
*To my mom, Nutan Kiran Verma, who made me what I am today and gave the confidence to pursue all my dreams. Thanks Papa, Naveen, and Prabhat, who motivated me to steal time for this book when in fact I was supposed to be spending it with them. Ulhas and the entire Packt team's support was tremendous. Thanks Varsha Shetty for introducing me to Packt.*



## Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors.packtpub.com](http://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

# Table of Contents



<b>Preface</b>	1
<b>Chapter 1: Introduction, Code Formatting, and Tools</b>	7
<b>The meaning of clean code</b>	8
<b>The importance of having clean code</b>	8
The role of code formatting in clean code	9
Adhering to a coding style guide on your project	10
<b>Docstrings and annotations</b>	12
Docstrings	13
Annotations	16
Do annotations replace docstrings?	18
Configuring the tools for enforcing basic quality gates	20
Type hinting with Mypy	20
Checking the code with Pylint	21
Setup for automatic checks	21
<b>Summary</b>	24
<b>Chapter 2: Pythonic Code</b>	25
<b>Indexes and slices</b>	26
Creating your own sequences	28
<b>Context managers</b>	29
Implementing context managers	32
<b>Properties, attributes, and different types of methods for objects</b>	35
Underscores in Python	35
Properties	38
<b>Iterable objects</b>	40
Creating iterable objects	40
Creating sequences	43
<b>Container objects</b>	45
<b>Dynamic attributes for objects</b>	46
<b>Callable objects</b>	48
<b>Summary of magic methods</b>	49
<b>Caveats in Python</b>	49
Mutable default arguments	50
Extending built-in types	51
<b>Summary</b>	53
<b>References</b>	53
<b>Chapter 3: General Traits of Good Code</b>	55
<b>Design by contract</b>	56

Preconditions	58
Postconditions	59
Pythonic contracts	59
Design by contract – conclusions	59
<b>Defensive programming</b>	60
Error handling	61
Value substitution	61
Exception handling	62
Handle exceptions at the right level of abstraction	64
Do not expose tracebacks	66
Avoid empty except blocks	67
Include the original exception	68
Using assertions in Python	69
<b>Separation of concerns</b>	70
Cohesion and coupling	71
<b>Acronyms to live by</b>	72
DRY/OAOO	72
YAGNI	74
KIS	74
EAFP/LBYL	76
<b>Composition and inheritance</b>	77
When inheritance is a good decision	78
Anti-patterns for inheritance	79
Multiple inheritance in Python	82
Method Resolution Order (MRO)	82
Mixins	84
<b>Arguments in functions and methods</b>	85
How function arguments work in Python	85
How arguments are copied to functions	86
Variable number of arguments	87
The number of arguments in functions	91
Function arguments and coupling	91
Compact function signatures that take too many arguments	92
<b>Final remarks on good practices for software design</b>	93
Orthogonality in software	94
Structuring the code	95
<b>Summary</b>	96
<b>References</b>	97
<b>Chapter 4: The SOLID Principles</b>	99
<b>Single responsibility principle</b>	99
A class with too many responsibilities	100
Distributing responsibilities	102
<b>The open/closed principle</b>	103
Example of maintainability perils for not following the open/closed principle	103
Refactoring the events system for extensibility	105
Extending the events system	107



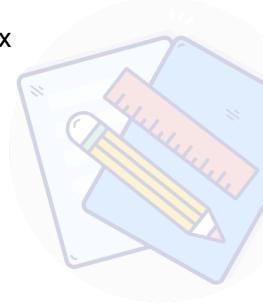
Final thoughts about the OCP	109
<b>Liskov's substitution principle</b>	110
Detecting LSP issues with tools	111
Detecting incorrect datatypes in method signatures with Mypy	111
Detecting incompatible signatures with Pylint	113
More subtle cases of LSP violations	113
Remarks on the LSP	116
<b>Interface segregation</b>	117
An interface that provides too much	118
The smaller the interface, the better	118
How small should an interface be?	119
<b>Dependency inversion</b>	119
A case of rigid dependencies	120
Inverting the dependencies	121
<b>Summary</b>	122
<b>References</b>	123
<b>Chapter 5: Using Decorators to Improve Our Code</b>	124
<b>What are decorators in Python?</b>	124
Decorate functions	126
Decorate classes	127
Other types of decorator	131
Passing arguments to decorators	131
Decorators with nested functions	132
Decorator objects	134
Good uses for decorators	135
Transforming parameters	135
Tracing code	136
<b>Effective decorators – avoiding common mistakes</b>	136
Preserving data about the original wrapped object	136
Dealing with side-effects in decorators	139
Incorrect handling of side-effects in a decorator	139
Requiring decorators with side-effects	141
Creating decorators that will always work	143
<b>The DRY principle with decorators</b>	146
<b>Decorators and separation of concerns</b>	147
<b>Analyzing good decorators</b>	149
<b>Summary</b>	150
<b>References</b>	151
<b>Chapter 6: Getting More Out of Our Objects with Descriptors</b>	152
<b>A first look at descriptors</b>	152
The machinery behind descriptors	153
Exploring each method of the descriptor protocol	156
__get__(self, instance, owner)	156
__set__(self, instance, value)	157



__delete__(self, instance)	159
__set_name__(self, owner, name)	161
<b>Types of descriptors</b>	163
Non-data descriptors	163
Data descriptors	165
<b>Descriptors in action</b>	168
An application of descriptors	168
A first attempt without using descriptors	168
The idiomatic implementation	169
Different forms of implementing descriptors	172
The issue of global shared state	172
Accessing the dictionary of the object	173
Using weak references	174
More considerations about descriptors	175
Reusing code	175
Avoiding class decorators	176
<b>Analysis of descriptors</b>	180
How Python uses descriptors internally	180
Functions and methods	180
Built-in decorators for methods	183
Slots	185
Implementing descriptors in decorators	185
<b>Summary</b>	186
<b>References</b>	187
<b>Chapter 7: Using Generators</b>	188
<b>Technical requirements</b>	188
<b>Creating generators</b>	189
A first look at generators	189
Generator expressions	192
<b>Iterating idiomatically</b>	193
Idioms for iteration	193
The next() function	195
Using a generator	196
Itertools	196
Simplifying code through iterators	197
Repeated iterations	198
Nested loops	198
The iterator pattern in Python	200
The interface for iteration	200
Sequence objects as iterables	201
<b>Coroutines</b>	203
The methods of the generator interface	204
close()	204
throw(ex_type[, ex_value[, ex_traceback]])	205
send(value)	206
More advanced coroutines	209
Returning values in coroutines	209



Delegating into smaller coroutines – the yield from syntax	210
The simplest use of yield from	211
Capturing the value returned by a sub-generator	212
Sending and receiving data to and from a sub-generator	213
<b>Asynchronous programming</b>	215
<b>Summary</b>	217
<b>References</b>	218
<b>Chapter 8: Unit Testing and Refactoring</b>	219
<b>Design principles and unit testing</b>	219
A note about other forms of automated testing	221
Unit testing and agile software development	222
Unit testing and software design	222
Defining the boundaries of what to test	225
<b>Frameworks and tools for testing</b>	226
Frameworks and libraries for unit testing	226
unittest	228
Parametrized tests	230
pytest	232
Basic test cases with pytest	233
Parametrized tests	234
Fixtures	235
Code coverage	236
Setting up rest coverage	236
Caveats of test coverage	237
Mock objects	238
A fair warning about patching and mocks	239
Using mock objects	239
Types of mocks	240
A use case for test doubles	241
<b>Refactoring</b>	243
Evolving our code	244
Production code isn't the only thing that evolves	246
<b>More about unit testing</b>	247
Property-based testing	248
Mutation testing	248
<b>A brief introduction to test-driven development</b>	251
<b>Summary</b>	252
<b>References</b>	252
<b>Chapter 9: Common Design Patterns</b>	253
<b>Considerations for design patterns in Python</b>	254
<b>Design patterns in action</b>	255
Creational patterns	256
Factories	256
Singleton and shared state (monostate)	257
Shared state	257
The borg pattern	260



Builder	262
Structural patterns	262
Adapter	263
Composite	264
Decorator	266
Facade	268
Behavioral patterns	269
Chain of responsibility	270
The template method	272
Command	273
State	274
<b>The null object pattern</b>	280
<b>Final thoughts about design patterns</b>	282
The influence of patterns over the design	282
Names in our models	283
<b>Summary</b>	284
<b>References</b>	285
<b>Chapter 10: Clean Architecture</b>	286
<b>From clean code to clean architecture</b>	286
Separation of concerns	287
Abstractions	288
<b>Software components</b>	290
Packages	290
Containers	293
<b>Use case</b>	295
The code	296
Domain models	296
Calling from the application	298
Adapters	300
The services	300
Analysis	304
The dependency flow	304
Limitations	305
Testability	305
Intention revealing	306
<b>Summary</b>	306
<b>References</b>	307
<b>Summing it all up</b>	307
<b>Other Books You May Enjoy</b>	308
<b>Index</b>	311

---





This is a book about software engineering principles applied to Python.

There are many books about software engineering, and many resources available with information about Python. The intersection of those two sets, though, is something that requires action, and that's the gap this book tries to bridge.

It would not be realistic to cover all possible topics about software engineering in a single book because the field is so wide that there are entire books dedicated to certain topics. This book focuses on the main practices or principles of software engineering that will help us write more maintainable code, and how to write it by taking advantage of the features of Python at the same time.

A word to the wise: there is no single solution to a software problem. It's usually about trade-offs. Each solution will have upsides and downsides, and some criteria must be followed to choose between them, accepting the costs and getting the benefits. There is usually no single best solution, but there are principles to be followed, and as long as we follow them we will be walking a much safer path. And that is what this book is about: inspiring the readers to follow principles and make the best choices, because even when facing difficulties, we will be much better off if we have followed good practices.

And, speaking of good practices, while some of the explanations follow established and proven principles, other parts are opinionated. But that doesn't mean it has to be done in that particular way only. The author does not claim to be any sort of authority on the matter of clean code, because such a title cannot possibly exist. The reader is encouraged to engage in critical thinking: take what works the best for your project, and feel free to disagree. Differences of opinions are encouraged as long as they yield an enlightening debate.

My intention behind this book is to share the joys of Python, and idioms I have learned from experience, in the hope that readers will find them useful to elevate their expertise with the language.

The book explains the topics through code examples. These examples assume the latest version of Python at the time of this writing is used, namely Python 3.7, although future versions should be compatible as well. There are no peculiarities in the code that bind it to any particular platform, therefore with a Python interpreter, the code examples can be tested on any operating system.

In most of the examples, with the goal of keeping the code as simple as possible, the implementations and their tests are written in plain Python using just the standard libraries. In some chapters, extra libraries were needed, and in order to run the examples of those cases, instructions have been provided along with the respective `requirements.txt` file.

Throughout this book we will discover all the features Python has to offer to make our code better, more readable, and easier to maintain. We do so not only by exploring the features of the language, but also by analyzing how software engineering practices can be applied in Python. The reader will notice that some of the reference implementations differ in Python, other principles or patterns change slightly, and others might not be even applicable all along. Understanding each case represents an opportunity to understand Python more deeply.

## Who this book is for

This book is suitable for all software engineering practitioners who are interested in software design or learning more about Python. It is assumed that the reader is already familiar with the principles of object-oriented software design and has some experience writing code.

In terms of Python, the book is suitable for all levels. It's good for learning Python because it is organized in such a way that the content is in increasing order of complexity. The first chapters will cover the basics of Python, which is a good way to learn the main idioms, functions, and utilities available in the language. The idea is not just to solve some problems with Python, but to do so in an idiomatic way.

Experienced programmers will also benefit from the topics in this book, as some sections cover advanced topics in Python, such as decorators, descriptors, and an introduction to asynchronous programming. It will help the reader discover more about Python because some of the cases are analyzed from the internals of the language itself.

It is worth emphasizing the word *practitioners* in the first sentence of this section. This is a pragmatic book. Examples are limited to what the case of study requires, but are also intended to resemble the context of a real software project. It is not an academic book, and as such the definitions made, the remarks made, and the recommendations given are to be taken with caution. The reader is expected to examine these recommendations critically and pragmatically rather than dogmatically. After all, practicality beats purity.