Extracted from:

Python Testing with pytest, Second Edition

Simple, Rapid, Effective, and Scalable

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Python Testing with pytest

Second Edition

Simple, Rapid, Effective, and Scalable

Brian Okken

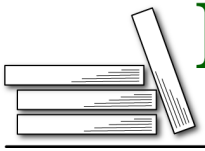
edited by Katharine Dvorak

Python Testing with pytest, Second Edition

Simple, Rapid, Effective, and Scalable

Brian Okken

The Pragmatic Bookshelf
Raleigh, North Carolina



Pragmatic Bookshelf

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-860-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—June 18, 2021

Preface

The use of Python is increasing not only in software development, but also in fields such as data science, machine learning, data analysis, research science, finance, and just about all other industries. The growth of Python in many critical fields also comes with the desire to properly, effectively, and efficiently put software tests in place to make sure the programs run correctly and produce the correct results. In addition, more and more software projects are embracing continuous integration and including an automated testing phase. There is still a place for exploratory manual testing—but thorough manual testing of increasingly complex projects is infeasible. Teams need to be able to trust the tests being run by the continuous integration servers to tell them if they can trust their software enough to release it.

Enter pytest. pytest is a robust Python testing tool that can be used for all types and levels of software testing. pytest can be used by development teams, quality assurance teams, independent testing groups, individuals practicing test-driven development, for both commercial and open-source projects. In fact, projects all over the Internet have switched from unittest or nose to pytest, including Mozilla and Dropbox. Why? Because pytest offers powerful features such as assert rewriting, a third-party plugin model, and a powerful yet simple fixture model that is unmatched in any other testing framework.

Why pytest?

pytest is a software testing framework, which means pytest is a command-line tool that automatically finds tests you've written, runs the tests, and reports the results. It has a library of goodies that you can use in your tests to help you test more effectively. It can be extended by writing plugins or installing third-party plugins. It can be used to test Python distributions. And it integrates easily with other tools like continuous integration and web automation.

Here are a few of the reasons pytest stands out above many other testing frameworks:

- Simple tests are simple to write in pytest.
- Complex tests are still simple to write.
- Tests are easy to read.
- Tests are easy to read. (So important it's listed twice.)
- You can get started in seconds.
- You use `assert` to fail a test, not things like `self.assertEqual()` or `self.assertLessThan()`. Just `assert`.
- You can use pytest to run tests written for unittest or nose.

pytest is being actively developed and maintained by a passionate and growing community. It's so extensible and flexible that it will easily fit into your work flow. And because it's installed separately from your Python version, you can use the same version of pytest on multiple versions of Python.

Learn pytest While Testing a Sample Application

In this book, you're going to learn pytest by writing tests against an example project that I hope has many of the same traits of applications you'll be testing after you read this book.

The sample application is called Cards. Cards is a minimal task-tracking application with a command-line user interface. It has enough in common with many other types of applications that I hope you can easily see how the testing concepts you learn while developing tests against Cards are applicable to your projects now and in the future.

Cards has a command-line interface (CLI). The CLI interacts with the rest of the code through an application programming interface (API). The API is the interface where we'll direct most of our testing. The API interacts with a database control layer, which interacts with a document database, TinyDB.

This isn't the most sophisticated task-management application, but it's complicated enough to use it to explore testing.

How This Book Is Organized

The book is organized into three parts.

In Part I, Primary Power, on page ?, you'll install pytest and start to explore its primary features, using the Cards project along the way. You'll learn how to run simple test functions on the command line. You'll then use pytest fixtures to push setup and teardown code out of the test functions. You'll learn how to use many of pytest's builtin fixtures to help with common testing problems like temporary directories. You'll also learn how to turn one test into many test cases with parametrization. And finally, you'll learn how to use markers to run a subset of tests.

In Part II, Working with Projects, on page ?, you'll look at some real-world issues around testing projects, as well as explore more of the power of pytest. You'll start by exploring a simple testing strategy process and applying it to the Cards project. You'll take a look at configuration files and all of the other non-test files involved in testing projects. You'll use coverage analysis to look at where our testing holes are with respect to Cards, and use mocking to help test the user interface and fill in some coverage gaps. Really all testing involves some debugging of both code and tests, so you'll take a look at some of the great feature pytest has to help us debug test failures. Many projects utilize continuous integration (CI). Tox is a popular framework to simulate a local CI system. You'll look at using pytest with tox and with hosted CI systems. Part II wraps up with a look at the Python search path. The Cards project is an installable Python package; however, not all testing projects involve installed packages. This final chapter in Part II looks at how you can tell pytest to find your source code.

In Part III, Booster Rockets, on page ?, you'll take your tests to the next level. You'll learn how to use third-party plugins to extend the capabilities of pytest and learn how to build your own plugins. You'll also learn some advanced parametrization techniques that build on what you learned in Part I. Finally, you'll take a look at some of the most powerful command-line options to make your testing effective and efficient.

What You Need to Know

Python

You don't need to know a lot of Python. The examples don't do anything super weird or fancy.

pip

You should use `pip` to install pytest and pytest plugins. If you want a refresher on `pip`, check out *Appendix 2: Pip*.

A command line

I wrote this book and captured the example output using zsh on a Mac laptop. However, the only commands I use in zsh are `cd` to go to a specific directory, and `pytest`, of course. Because `cd` exists in Windows `cmd.exe` and all Unix shells that I know of, all examples should be runnable on whatever terminal-like application you choose to use.

That's it, really. You don't need to be a programming expert to start writing automated software tests with pytest.

Why a Second Edition?

Both Python and pytest have changed since the first edition of this book was published in 2017. There have been updates to pytest that are now reflected in the book, such as:

- New builtin fixtures
- New flags (bet you can't wait to read that chapter now, huh?)
- The addition of package scope fixtures

There have also been updates to Python that are reflected in the book:

- The adoption of f-strings and `pathlib`
- The addition of dataclasses

Also, since publication of the first edition, I have taught many, many people about pytest, and I think I've learned how to be a better teacher. The second edition not only expands on what is covered in the first edition—it grew from 7 to 17 chapters!—but also it presents the material in what I think is a more gradual, digestible manner.

So what's in all of these new chapters?

- *More on parametrization, markers, coverage, mocking, tox and continuous integration, and third-party plugins.* All of these topics were covered in the first edition, but in this edition I expand that coverage. I pulled the discussion of parametrization into its own chapter and added a discussion of advanced parametrization techniques. I delve more deeply into markers and include an example of how to pass data from markers to fixtures (which is super cool). I also take you on a deeper dive into test coverage, mocking, and CI, and using and building your own plugins to extend pytest's capabilities.
- *A discussion of test strategy.* Feedback from the first edition was that the book was great for the mechanics of how to use pytest, but the "What test

do I write?" information was a bit lacking. The new *Chapter 7: Strategy* is push in the right direction of what tests to write. A complete treatment of test strategy would be a book in itself; however, this chapter will get you started.

- *Information about the Python search path.* A lot of readers reached out to me asking about how to get their tests to see their test code, and the first edition didn't cover it. The project in this book, Cards, doesn't have that problem because it's an installed Python package. However, lots of user projects are applications or scripts or lots of other things that are not installed packages. This chapter offers a focused look at the problem and provides some solutions.

I moved coverage of command-line flags and configuration settings in general to their own chapter at the end of the book where, after you've learned the basics of pytest, you can check out the ton of cool options that can help you run your tests more efficiently and effectively.

Also, I consolidated the information about debugging test failures into a chapter of its own. In the last edition, this information was spread all throughout the book. It is my hope that when you are faced with a deadline and a failing test suite, bringing this information together into one chapter will help you figure an answer out quickly and ease some stress.

Finally, the example project changed. The first edition used a project called Tasks to illustrate how to use pytest. Now it's called Cards. Here's why:

- It's easier to say out loud. (Try it. Say "tasks" three times, then "cards" three times. Right?)
- The new project itself is different because it uses Typer instead of Click for command-line functionality. Typer code is easier to read.
- The project also uses Rich for formatting the output. Rich didn't exist (neither did Typer) when the first edition was written.

The code examples have also been simplified. The directory structure of the first edition code examples followed a progression of a possible test directory within a project, with most of the project removed. Seriously, I think it made sense to me at the time. In this edition, there is a project in its own directory, `cards_proj`, with no tests. Then each of the chapters have test code (if appropriate) that either work on the one project, or on some local code. Trust me, I think you'll agree that it's way easier to follow along now.

Example Code and Online Resources

The examples in this book were written and tested using Python 3 and pytest 6. If you're reading this with later versions of pytest and wondering if this book still applies, the odds are that it does. I have worked with many core pytest contributors to make sure the content of this book will apply to future versions of pytest as well. There is also an errata page set up at both pytest-book.com[1] and at pragprog.com[2] that also note any updates you need to be aware of for future versions of pytest and this book.

The source code for the Cards project, as well as for all of the tests shown in this book, is available through a link on the book's web page.[3] You don't need to download the source code to understand the test code; the test code is presented in usable form in the examples. But to follow along with the Cards project, or to adapt the testing examples to test your own project (more power to you!), you must go to the book's web page to download the project.

To learn more about software testing in Python, you can also check out pythontest.com[4] and testandcode.com,[5] a blog and podcast I run that discuss the topic.

I've been programming for decades, and nothing has made me love writing test code as much as pytest. I hope you learn a lot from this book, and I hope you'll end up loving test code as much as I do.

1. https://pytestbook.com
2. https://pragprog.com/titles/bopytest2
3. https://pragprog.com/titles/bopytest2/source_code
4. http://pythontest.com
5. http://testandcode.com