Learn Blockchain by Building One

A Concise Path to Understanding Cryptocurrencies

Daniel van Flymen

Apress®

Learn Blockchain by Building One

A Concise Path to Understanding Cryptocurrencies

Daniel van Flymen

Apress®

Learn Blockchain by Building One: A Concise Path to Understanding Cryptocurrencies

Daniel van Flymen
New York, NY, USA

ISBN-13 (pbk): 978-1-4842-5170-6 ISBN-13 (electronic): 978-1-4842-5171-3
https://doi.org/10.1007/978-1-4842-5171-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Shiva Ramachandran
Development Editor: Rita Fernando
Coordinating Editor: Rita Fernando

Cover designed by eStudioCalamar

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5170-6. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

*Dedicated to **Joshua**, who finishes what he starts.*

Table of Contents

About the Author

Daniel van Flymen is currently a Director of Engineering at Candid in New York City. As a seasoned Python veteran, he's a regular code contributor to popular open source projects and is a guest on the Software Engineering Daily podcast, having been on popular episodes such as Understanding Bitcoin Transactions and Blockchain Engineering. He frequently writes on Medium.com and has a number of popular articles, such as "Learn Blockchains by Building One" and "Learn Blockchains Using Spreadsheets"—he is passionate about increasing Bitcoin adoption because he believes it's the future.

About the Technical Reviewer

Federico Ulfo is a polyhedric software engineer and entrepreneur experienced in building high-scale API and ETL. He founded the Lightning Network NYC and the Learning Bitcoin meetups. His interests span from cryptocurrencies to economics, philosophy, gardening, and many more topics. You can reach out to him at ulfo.it.

Acknowledgments

This book is dedicated to my brother Joshua who always finishes what he starts. A major thanks to

- My friend and fellow Bitcoin educator, **Justin Moon**, for helping me clarify concepts, test code, and provide sound advice when needed.

  My friend **Federico Ulfo**, not just for the arduous task of reviewing and double-checking my work each week but also for trekking with me to countless Bitcoin and Lightning events and conferences over the last few years.

- **Rita Fernando** and **Shivangi Ramachandran** from Apress for believing in me and making this book a reality.

Introduction

Another book on blockchains? Why?

Understanding blockchains isn't easy. Or at least wasn't for me: when Bitcoin first made the news cycle, I tried to learn how it worked and discovered that there were too few resources addressed to programmers (like myself). There was always the Bitcoin reference wiki (https://en.bitcoin.it), but in those days, it wasn't as clearly organized as today, and although I read Satoshi's whitepaper, I didn't really understand it at first—at least not how the cryptographic parts worked.

I meandered through YouTube, completed porous tutorials, and felt the frustration of examples that didn't communicate the concepts clearly. So, I decided to try and build a blockchain myself, and document all the things I learnt along the way. In so doing, I discovered why cryptocurrencies are so hard to explain and understand; it's because you first need to define the ingredients of digital money:

- How does the money get created? (Mining)
- How does Alice send money to Bob? (Digitally signed transactions)
- Who keeps track of all these transactions and the generated money? (Everyone, via a distributed ledger)

These high-level points rely on distinct units of knowledge that must be understood before they can be combined into a set of commonly agreed-upon rules that everyone follows. And the best way to understand these disparate concepts is piece by piece—by practically using them to build your own cryptocurrency. So, I wrote this book for people who feel

the same frustration that I did, and overcome it by dealing with the subject matter at a code level—that's what really gets it to stick. If you follow through, and do the same, I'm certain that at the end of this book, you'll have a solid grasp of how they work.

Setting yourself up for success

GitHub repository

The finalized code is located at `https://github.com/dvf/blockchain-book`. But try do the coding yourself—the code is structured in such a way that methods are stubbed out at a high level with the details being filled incrementally. This code is kept updated, and so it's handy as a north star.

Take the time to set up your development environment

Use a good IDE (integrated development environment) like Microsoft VSCode or JetBrains PyCharm. They are both free and fantastic at spotting errors in your code before you do. And it's well worth the time to set your IDE up before you begin. Spend your time worrying about blockchains and not about syntax errors in your code.

Know where to get answers

Browse and ask questions on the GitHub repository's *Issues* page. The repository has a large community following, so you're likely to meet others with similar problems. And if you encounter errors or bugs, I implore you to open an *Issue*.

Don’t speak Python?

That’s OK. Python is known for its legibility; it’s a very easy language to transcribe. I have seen other programmers (C#, JavaScript, and Rust) do the examples in the book on the fly.

CHAPTER 1

Getting Ready for Application Development

For the unfamiliar, Python is one of the most popular languages. It's extensively used everywhere—from academia and the sciences to large-scale web applications, like Instagram. Part of its popularity is due to the plethora of libraries, packages, and extensions available for free online as well as ease of reading due to its resemblance to pseudocode.

In this chapter we'll make sure your computer is properly set up for application development and that Python is properly installed. Then, I'll show you how to create a pragmatic Python project and how to install dependencies.

Python Versions

Python comes in two flavors: version 2 and version 3. Version 2 is no longer supported by the Python Software Foundation, but it still ships preinstalled on most operating systems because it's used by plenty of internal tools. Another complication is that different operating systems install Python in different places in the file system. These factors make setting up a development environment tricky.

D. van Flymen, *Learn Blockchain by Building One*,
https://doi.org/10.1007/978-1-4842-5171-3_1

We'll try navigating these obstacles by installing and using tools which help us manage Python installations.

Note As a convention, throughout this book, we'll prefix a terminal command using the $ symbol. The output will be shown as plaintext.

Installing Python

Windows installation

Python.org contains downloadable binaries for Windows. Head over to www.python.org/downloads/windows/ and download the binary for Python 3.8.

Once downloaded, install Python 3.8, making sure to choose the options to

- Uninstall previous versions of Python.
- Install the pip (the Python package manager).
- Add Python to the PATH (allowing you to execute Python on the command line).

After installation, to confirm you've done everything correctly, open up your command line and check Python's version:

```
C:\Users\dan> python --version
3.8.3
```

macOS installation

Although macOS ships with a version of Python for internal purposes, we **don't want to modify it when we develop,** so we'll be installing a fresh version of Python using Homebrew—a tool used to help manage and install third party packages on macOS.

First, we'll need to make sure Apple's Command Line tools are installed, in your terminal:

```
$ xcode-select --install
```

You'll need to install *Homebrew*, a package installer for macOS. To install it, follow the instructions on https://brew.sh/, and ensure that Homebrew is correctly installed.

After you've installed Homebrew, let's install the latest version of Python:

```
$ brew install python
```

Once the installation completes, verify that Python has been installed correctly:

```
$ python --version

Python 3.8.3
```

Linux installation

If you're using a Debian-based version of Linux, you can install Python 3.8 using `apt` (or any other package manager):

```
$ sudo apt-get update
$ sudo apt-get install python3.8
```

Once the installation completes, verify that Python has been installed correctly:

```
$ python --version
Python 3.8.3
```

If you're not using a Debian-based Linux distribution, you can compile Python from the source: www.python.org/downloads/source/.

How Python programs run

When you install Python, you're actually installing an *interpreter*—a program that translates written Python code to instructions that your computer understands and executes. The interpreter you've installed is called CPython, a popular interpreter written in the C language.

You run a Python program by feeding it to the Python interpreter in your terminal:

```
$ python my_program.py
```

This converts your code to "computer instructions" and executes them.

How Does Your OS Know Where the Python Interpreter Is?

Your OS has a system-wide variable called `PATH`, containing a list of file paths to traverse when looking for programs. You can check what it's set to by running `echo $PATH` in your terminal. The Python interpreter resides in `/usr/local/bin/`. This is verified by calling `which python`.

Managing project dependencies

Every project you build is likely to use external libraries. These dependencies may be database access libraries or tools needed to parse documents or websites, but the important thing is that they're included in your project.

Managing project dependencies can be a tricky task, since different dependencies have different requirements—some dependencies require specific versions of Python, others may depend on sibling dependencies. Modern Python projects use package managers to cope with the arduous tasks of downloading, installing, and keeping up-to-date dependencies. In summary, it makes your life easier to use a package manager.

Poetry is one out of a handful of dependency managers for Python. There are other, more popular ones, like Pipenv. But after using both extensively, I've found that Poetry has a cleaner interface and is more pragmatic in its goals.

Installing Poetry

The recommended way of installing Poetry is to run the following in your terminal:

```
$ curl -sSL https://raw.githubusercontent.com/sdispater/poetry/
master/get-poetry.py |
python
```

If you run into any problems, please refer to the official documentation and installation instructions on the Poetry website: https://poetry.eustace.io/docs/