How To 10X Your Python Skills

For Engineers, Data Scientists, And All Other Technology Professionals Writing Code in Python

Aaron Maxwell
https://powerfulpython.com

Current Trends

1. **You have less free time for deep focused learning**, due to changes in culture and technology. This makes it difficult to invest the self-study needed to reach elite status as a technology professional.

2. **Python is becoming the most important language** in all of technology. It is widely used in most engineering domains, and will continue to become more dominant over time. It has entrenched as *the* language of AI and data science, and is positioned to monopolize other fields in the coming years.

3. **AI coding tools** now fully automate basic tasks, and can partially replace more junior, lower-skilled coding jobs. As these tools improve, a higher level of coding skills will be necessary to meet rising professional standards, especially for the best roles.

4. **People are learning Python at an increasing rate**, with many university and introductory trainings focused on Python over other languages. This has pros and cons: it increases the value of knowing Python well, but also means you face greater competition, and demands a higher level of skill for you to stay ahead.

In summary: Recent trends have made it more difficult to develop your skills to the most rewarding levels. At the same time, Python has become the “lingua franca” of computing, with outsized benefits for those who master this language.

The Benefits Of 10X-ing Your Python Skills

Python is the most important programming language in history. Fluency in Python slams open the doors of opportunity, while poor Python skill bogs down your professional effectiveness and career opportunities.

No matter your job title or specialty, if you write Python code in your work, mastering Python gives you tremendous advantages over everyone competing with you for promotions and job offers, and helps ensure an increasingly fulfilling, brighter, and higher-income future in your career.

Some ways your life changes when you become a top 1% Pythonista:

- **You find yourself rocketing up learning curves**, for any new Python library, framework or other tooling you learn.
- **You spend less time debugging**. Because you make fewer bugs to begin with; and when you do make them, you can immediately see what went wrong, and how to most quickly fix it.
- **Interviews become relaxing and fun.** And you have better odds of standing out in interviewers' minds, among everyone else they interviewed that day.
- **Robust confidence,** naturally and easily.
- You can **write larger, more complex programs** than ever before.
- Other highly skilled pros **want to work with you.**
- You have **more opportunities for better jobs**…
- Which usually means **you earn more money**. It *definitely* means you'll have a more fulfilling career.

If you want these benefits for yourself, keep reading.

First Principles

Problem #1: You have too much to learn.
Problem #2: Society has evolved to reduce your time and energy for deep focused learning.

This seems like a recipe for misery. But there is a way out: by focusing on what are called *first principles*.

In any field of human activity - including Python coding - there are foundational concepts which everything builds on. Certain powerful distinctions, abstractions, and mental models. When you learn what these first principles are, and how to work with them, you find you can cut through the noise and get ahead much more easily.

These first principles are *accelerators*, in that they give you the tools, inner resources and capabilities to solve nearly any problem. It effectively creates a "95/5" rule: there is a 5% you can focus on learning, which makes the remaining 95% fall like dominos. That 5% is the first principles of Python.

This notion of first principles dates back to Aristotle. It has more recently been championed by physicists like Dr. Richard Feynman, and technology entrepreneurs like Reed Hastings (Netflix) and Elon Musk (SpaceX).

When you learn these first principles of Python, the impact is difficult to overstate. It exponentially accelerates everything you do involving Python code.

Crucially, it makes FUTURE LEARNING faster and easier, for anything involving Python. You simply find yourself climbing learning curves more quickly, with less mental effort. You pick up the remaining 95% rapidly and naturally.

Because Python is so foundational for all modern technology work, learning the first principles of Python is transformative. This is the magic key to getting ahead, and stop feeling overwhelmed.

The DateInterval Example

Let's look at a class called `DateInterval`. This class was originally created as part of an internal marketing analytics application.

`DateInterval` represents a range or interval of days, and some operations you might want to do on that interval. It makes use of several advanced Python features, working in harmony to provide a sophisticated interface. Being a single class, it is also simple enough to quickly understand.

You use it like this:

```
>>> interval = DateInterval(date(2050, 1, 1), date(2059, 12, 31))
>>> some_day = date(2050, 5, 3)
>>> another_day = date(2060, 1, 1)
>>> some_day in interval
True
>>> another_day in interval
False
>>> for day in interval:
...     process_date(day)
```

Notice you can use it with Python's binary "in" operator, and also in “for” loops. Here is the full source:

```
from datetime import (
    date,
    MINYEAR,
    MAXYEAR,
    timedelta,
    )

class DateInterval:
    BEGINNING_OF_TIME = date(MINYEAR, 1, 1)
    END_OF_TIME = date(MAXYEAR, 12, 31)

    def __init__(self, start=None, end=None):
        if start is None:
            start = self.BEGINNING_OF_TIME
        if end is None:
            end = self.END_OF_TIME
        if start > end:
            raise ValueError(
                f"Start {start} cannot be after end {end}")
        self.start = start
        self.end = end

    def __contains__(self, when):
        return self.start <= when <= self.end

    def __iter__(self):
        num_days = 1 + (self.end - self.start).days
        for offset in range(num_days):
            yield self.start + timedelta(days=offset)
```

This is not "hello world" level stuff. `DateInterval` uses class methods, the iterator protocol, generator functions, magic methods, and a couple of important yet subtle design tradeoffs that are not obvious.

If you are not familiar with those features yet, that is fine. What's important to understand is that those Python language features work together, to provide a flexible abstraction which simplifies code performing common operations. It creates a new data type, integrating smoothly with well-known Python syntax and idioms, while encapsulating and hiding complexity the developer does not need to think about.

First principles thinking allows you to see these kinds of patterns in complexity, and create novel code structures that dramatically improve every aspect of the coding process. It's like you can see a `DateInterval`-shaped hole in your codebase, that no one else can see except you. Until you create that class to fill the hole… solving their problem, and yours.

Ultimately, You Want Status

As technology professionals, status comes from what we contribute. From our demonstrated ability to move the project forward, as a valued member of the team.

In other professions, status is often NOT based on ability, nor on positive contribution. In terms of game theory: for us, **seeking status is positive-sum.** Whereas it is zero-sum or even negative-sum for others.

This is something to celebrate! We do not need to choose between status and excellence. For us, they are strongly correlated, hand-in-hand.

Status means your professional peers - other developers, data scientists, et cetera - look up to you and respect you. Because of your abilities, your skills, and the success you clearly bring to the team.

It is important to recognize this is relative. The brutal truth is that it's not enough to be highly skilled; rather, you need to be *better than everyone else*. That is why we emphasize the idea of a "top 1% Pythonista". The greatest benefits go to the top 1%, and are not available to the 99%, regardless of how good those 99% are.

No matter what realm of technology you are in, **Python coding skills are foundational**. They amplify your productivity, impact, and sometimes even your speed of learning (e.g., for Python libraries). Substantially improving your Python skills as soon as possible yields outsized benefits, compounding exponentially over your future career.

And decisive speed matters. Your benefits compound over time; any delay forfeits much exponential gain. For your short, medium, and long-term goals, it matters that you handle this, and handle it soon.

The Solution

So how do you learn these first principles of Python? And more importantly, apply them in real-world coding tasks?

The good news: there is a solution, called Powerful Python Bootcamp. By focusing on the first principles of Python, its rigorously proven curriculum maximizes the impact of your invested learning effort. Combined with expert mentoring support, a community of skilled Pythonistas,

and a remarkable “back catalog” of advanced Python seminars, Powerful Python Bootcamp is the fastest and most certain path for busy Python pros to master using the language in their chosen domain.

Case Studies

Powerful Python students work in the best engineering and data science teams in the world, including:

Juan Arambula, Software Engineer

"I tried other premium Python content, and they didn't go deep enough. I found they pretty much repeated each other. And when I tried Powerful Python, it was quite different. I was able to learn in a matter of weeks all I needed to learn about Python to use it proficiently, something that I feel would have taken me months to learn on my own. So I really recommend it to everybody." (Link to Juan's full video review)

Jeffrey Smart, Risk Manager & Software Engineer

"I just had a round of interviews for a software engineering position, and I'd say about half of the programming examples were all but taken directly from Powerful Python materials. So when I would see these questions I really could hit the ground running." (Link to Jeffrey's full video review)

Erik Engstrom, Embedded Systems Developer

“Powerful Python's first principle approach was the primary thing that really attracted me to it. The idea of mastering is something that was very attractive and something that I wanted. And so it's been very valuable. And I would highly recommend it to anyone else. I really encourage you to consider it.” (Link to Erik’s full video review)

Many Dozens More

Watch and read dozens of other case studies here.

Who This Is For

Powerful Python Bootcamp is for working technology professionals who want to quickly ramp up their Python skills. With a time investment of 5-10 hours per week, on your own schedule.

This is for Data Engineers & Scientists; Software Engineers & Developers; Security Engineers; Web Developers; Test Engineers; Devops/SREs; QA Automation; and all other technology professionals writing Python code in their work.

The Bootcamp includes:

- **The Powerful Python Curriculum:** The most in-depth, realistic, and advanced Python education in the world
- **Elite Group Coaching:** Get live expert feedback & mentoring every week

- **Private Slack Community:** Access to one of the most exclusive networks of elite Python professionals
- **100+ Hours of Elite Coaching Archives**: Hours of advanced Python Q&A where Powerful Python instructors screen-share live coding to answer Bootcamp students' toughest questions.

Let me explain each aspect in detail.

The Curriculum

Powerful Python's curriculum comes out of Aaron Maxwell's partnership with O'Reilly Media in 2018-2020. In this partnership, he developed a comprehensive and rigorous curriculum for intermediate to advanced Python, and taught it to over 10,000 technology professionals worldwide… with students from nearly every culture, continent, and domain of engineering.

He also wrote a book, then decided to further develop and teach this curriculum independently, as Powerful Python Bootcamp.

This makes Powerful Python Bootcamp the most well-tested professional Python curriculum ever taught in the history of Python... and arguably the history of ANY programming language. It is unlikely anything will ever surpass its quality, rigor, or track record for results.

Modules include:

Module 1: Pythonic OOP

(3.5 hours of course video)

- Review of Python's Syntax for Classes
- The Single-Responsibility Principle
- Inheritance
- Refactoring For Inheritance
- Pythonic Interfaces
- Methods And Inheritance
- Access Control
- Properties
- Properties And Refactoring
- Factories, Class Methods And Static Methods
- Pythonic Design Patterns

Module 2: Test-Driven Python

(4 hours of course video)

- Foundations of Automated Testing

- Types of test: Unit Tests, Integration Tests, and More
- Test-Driven Development
- Detailed Test Assertion Types & Strategies
- Fixtures
- Parameterized Tests (And Subtests)
- Mock Objects
- Patching With Mocks
- Mocking Strategies: Pros and Cons

Module 3: Scaling Python With Generators

(2 hours of course video)

- Foundations of Generators
- Generator Design Patterns
- The Iteration Protocol in Python
- List Comprehensions
- Generator Comprehensions (And Other Comprehensions)
- Passing Data Into Generators (Coroutines)
- Overview of AsyncIO

Module 4: Next-Level Python

(2.5 hours of course video)

- Variable Arguments and Argument Unpacking
- Functions As Objects
- Writing Simple Decorators
- Stateful Decorators
- Higher-Order Decorators
- Class-Based Decorators

Module 5: Python Code Walkthroughs

(5 hours of course video)

- A detailed narrated walkthrough of real-world production Python codebases
- DateInterval class
- Program: lookupemails.py
- DownloadDir class
- EmailAnonymizer class
- "Out of context" Analysis of A Special Method
- Bonus Q&A Session Recordings

Module 6: Practical Python Engineering

(4 hours of course video)

- Python Module Organization
- Logging in Python

- Errors And Exceptions
- Python Dependency Management
- Command-Line Python Programs

Module 7: RESTful API Server Project

(4 hours of course video)

- The Most Challenging Python Course Ever Created!
- Project where you fully code a realistic, complex software application...
- Starting with nothing but a short spec document, and an empty folder...
- Building out the entire application using test-driven development and version control...
- To a standard of quality that will pass a rigorous code review in a top-tier engineering team.
- And it is HARD. But you can do it, by the time you get here. Because the preceding modules prepare you.

Group Mentoring

The Bootcamp includes weekly group mentoring sessions, led by an expert Pythonista. This allows you to get your questions answered live. These sessions are recorded, so if you are not able to attend live, you can submit your questions before and they will be answered on the recording for you.

The best part of group mentoring is that you learn from other student's questions. Most who join Powerful Python Bootcamp are ambitious and skilled technology professionals. They will ask AMAZING questions you never would have thought to ask on your own. This makes the group mentoring sessions a priceless learning experience for everyone. Plus it's a lot of fun!

You have full access to Group Mentoring for a minimum of 8 weeks. In practice, we allow students in good standing to continue attending indefinitely. While the minimum is guaranteed, there is currently no *maximum*. Some Bootcamp students have been actively participating in the group mentoring for *years*.

"Good standing" means you are professional in your behavior, and contribute to the positive learning environment for everyone. This is another way of saying we kick out troublemakers. So far, we have only had to do that to one person; Bootcamp students are generally very good people.

Professional Student Community

Powerful Python Bootcamp students enjoy access to the private Bootcamp Slack channel. Because Bootcamp students are ambitious and skilled technology professionals, this gives you

remarkable networking opportunities and a higher caliber of discussion. Genuine friendships have formed in this Bootcamp community.

Extensive Group Mentoring Archive

Powerful Python Bootcamp gives you full access to over 100 hours of video from past Elite Group Mentoring sessions. This trove is filled with priceless insights and live coding demonstrations, for a wide range of practical, real-world topics in Python, software engineering, data science, and much more.

A partial list of topics:

- Advanced uses of super()
- A live troubleshooting session, figuring out a bug in a lab under Pytest
- Optional arguments in dataclasses
- Spilling some secrets from the Coder Dream Job course
- The three ways to use type annotations
- Aaron's opinions on Javascript
- Reducing operations in Pyspark
- The decorator pattern that should be ILLEGAL
- Peering under the hood of Powerful Python's marketing automation code
- The C3 Linearization algorithm
- How to inspect the source of your imported libraries
- Why we never use f-strings in logging
- Module organization
- Is Python high level or low level? How does it compare to other languages? What does ‘high level' even mean? And what does one of the most famous Python libraries of all time have to teach us about this?
- Sentinel values, and how they relate to None, pandas.NA, and np.nan
- Higher-order function tricks
- The @functools.cached_property decorator (which is totally and completely different)
- When to use notebooks, and when to avoid them
- Walking through the codebase of a Django web application I recently cooked up, demonstrating many advanced techniques from Higher-Order Python and other Powerful Python courses.. A real-world production application with a STACK of money on the line
- Exploratory mutations of the webviews.py lab
- Core concepts in the Twisted framework
- Levels of technological abstraction, and how understanding lower levels affects the effectiveness with higher levels
- How to learn a new programming language that's different from what you are familiar with
- A detailed walkthrough of how to do the labs in the most systematic way, and working with mocks in unittest
- Deep dive into the webframework.py lab from Next-Level Python

- How to handle it when you start to become better than your peers
- How IDEs can get in the way of your learning (and how to deal with it)
- Testing functions that are inherently difficult to test
- Bash wrappers for Python programs
- The evolution of built-in exceptions from Python 2 to Python 3, and the lessons that evolution has for our coding today
- Higher-level strategies of automated testing
- Subclassing built-in types, and why str is different there
- Unconventional decorator patterns
- How to read PEPs
- Deferred Reference
- The security risks and other hazards of eval()
- The essence of super()
- Revealing the source of a script I wrote to generate the HTML version of the Powerful Python book
- Strategy for navigating multiple layers of abstractions
- A small distributed Pythonic app to get around a corporate firewall
- Details of the iterator protocol (and how generator objects fit into that)
- Understanding the Liskov Substitution Principle, and how it relates to Python's OOP syntax
- How to think about mocks
- Covering many advanced details of working with web services and the HTTP protocol
- Complexity trade-offs for code calling APIs
- Talking through a challenging network-engineering issue with strong security considerations
- The @functools.cache decorator, and how it relates to the memoization labs in Next-Level Python
- Adding exception context in the "catch and re-raise" pattern
- Why DownloadDir.wait_for_chrome_download() (from Code Walkthroughs) is like a state machine
- The three pillars of concurrency in Python
- How @functools.cached_property relates to the memoize.py lab in Next-Level Python
- Object lifecycle in Python, and the __del__() magic method
- A partial code walkthrough of the metaleads.py program, including comments on environment variables
- Prompt engineering troubleshooting
- Many fascinating details of generators and decorators
- A non-Python database design question
- Learning more advanced Git
- Deep distinctions on Python's generator model
- Is the GIL finally going away? And why it is important for Python concurrency. How threading, asyncio, and multiprocessing compare now... and how PEP 703 will change that.
- Looking at a multithreaded production codebase

- how object identity is defined in Python's language model
- My blasphemous opinion on Linus Torvald's greatest achievement
- Scoping rules around nonlocal variables
- A broad survey of the different collection types in Python, beyond lists and dicts
- The different roles of type annotations
- System paths vs Python paths vs other paths
- Debugging diabolical decorators
- The concept of "cognitive cost" when coding
- Method Resolution Order
- Deeper understanding of lambdas (anonymous functions), including how Python does them differently than other languages
- Choosing good inheritance hierarchies
- Book recommendations for Python pros
- The right way to use Stack Overflow (and how to avoid mis-using it)
- Generator objects/comprehensions/functions
- Troubleshooting a bug so subtle, that we had to dig elbows-deep into the Pandas source code to crack it
- Peering into the Asciidoc source of the Powerful Python slide decks (the same format used by O'Reilly for its books)
- How instance, static and class methods differ in Python, and how that's all different from how it works in languages like Java
- Defensive asserts
- Talking about defaultdict... not just how to use it, but going deep into WHY it was designed the way it was, and the lessons it has for your own code
- Multithreaded/multiprocessor programming, and the best book for learning it deeply
- Diving deep into exception patterns
- Troubleshooting a high-throughput messaging architecture problem
- Many insights about automated testing
- What to do when you are facing an overwhelmingly complex coding problem
- A demonstration of expanding the test suite with more refined requirements, and how that affects the evolution of the application
- Strategies for cramming for a FAANG interview
- Classmethod vs. staticmethod
- Should you be worried about being replaced by a bot? In fact, many Python developers SHOULD be worried, but a small fraction will come out of this as big winners. Today's session tells you how to be one of them
- Generator algorithms
- The "walrus operator", formally called "assignment expressions" (PEP 572)
- Generator-based coroutines
- The intersection of module organization + dependency management
- Inter-Python-process communication
- How Github Copilot, ChatGPT, and other AI tools coming in the future affect your career
- A stateful low-level parsing algorithm
- Pyspark

- The why and how of functools.wraps()
- How to catch exceptions in Twisted

To our knowledge, this is the most extensive repository of realistic and advanced discourse for Python professionals in the world.

Qualifications

Powerful Python Bootcamp is for busy technology professionals. It is designed to fit in your life.

For best results, we recommend you invest 8-10 focused hours per week, for at least 6 weeks. 5 hours per week is the minimum to maintain good forward momentum.

The course material is self-paced, so the schedule and time invested is up to you. We strongly recommend you attend at least one group mentoring session per week; more is better.

For technical prerequisites, you should be comfortable writing simple Python programs with functions, dicts and lists, and working with them on the command line. Experience writing simple classes and methods is helpful and recommended, but not strictly required.

The best way to ensure you are qualified is to complete the sample coding exercises at https://powerfulpython.com/sample/ .

Enrollment

The first step is to fill out a brief application, to ensure you and the Bootcamp are a good fit for each other. Do that here: https://powerfulpython.com/apply/

Questions & Answers

If your question is not answered below, ask us at service@powerfulpython.com and we will reply directly. We are also happy to do a short call, if you want to speak with us live; just email us to ask for the scheduling link.

What if I cannot attend the group mentoring sessions live?

You can submit questions before, and the instructor will answer them on the recording for you. If you have followup questions, you can ask in Slack, or you can submit it for the next session.

How do I know if the Bootcamp is right for me?

Several ways you can check:

1. Read the Powerful Python book (https://powerfulpython.com/book/). This will clarify for you the higher, more elite level of Python coding skills we are aiming for. If you want to learn this level more quickly, more deeply, and with more certainty of success, the Bootcamp is your best way to achieve that.
2. Do the sample coding exercises here: https://powerfulpython.com/sample/
   This will give you a taste of the Bootcamp learning experience, and how it is uniquely different from other trainings.
3. Watch these publicly-available videos of Group Mentoring sessions:
   a. Diving Into Pandas' Source Code To Figure Out A Bug
   b. Why does Python's defaultdict work that way?!

How long are the group mentoring sessions?

As long as they need to be, up to a point. Typically they range from 45-75 minutes. When there are many questions, the session may go 90 minutes or even longer. Rarely, there are not many questions, and the session will wrap up in under 15 minutes.

You can always drop in or drop out, and catch the parts you missed in the recording. That said, attending live will maximize your learning, let you interact with the instructor and other students, and is a lot of fun. So we recommend you attend live as often as possible.

What kinds of questions can I ask in the group mentoring?

In each session, the first priority is to answer all questions which directly relate to the course material. We do this to ensure everyone can keep strong forward momentum of their learning.

Once those are answered, we open it up to “any question the instructor is able to answer”. It does not even have to be about Python.

How long do I have access to the Group Mentoring and Slack community?

A minimum of 8 weeks.

In practice, we currently allow students in good standing to continue attending indefinitely, at no extra cost. Some Bootcamp students have been actively participating in our group mentoring for *years* now.

“Good standing” means you are professional in your behavior, and contribute to the positive learning environment for everyone. This is another way of saying we kick out troublemakers. So far, we have only had to do that to one person; Bootcamp students are generally great people.

One other point: While we have been doing these group mentoring sessions for years, there is no guarantee they will continue indefinitely. Powerful Python instructors are active engineers,

and we may decide to focus on other projects eventually. The sooner you enroll, the longer your potential “bonus time”.

What is the Group Mentoring schedule?

The normal schedule is currently

- Tuesdays at 5pm Pacific
- Fridays at noon Pacific

Please note these times occasionally shift, and may be different when you enroll. You can ask if there have been any recent changes by emailing us at service@powerfulpython.com.

How long do I have access to the course material? Including the Group Mentoring archives?

Our terms of service say “a minimum of 5 years”. Unofficially, we plan to host them for many years longer. For legal reasons, we cannot say “lifetime access”, but that is likely what will amount to.

Do I get future course updates too?

Yes, as a student in Powerful Python Bootcamp, you get full access to all future updates of the Bootcamp curriculum, for free. This includes future new Bootcamp modules, and all updates to existing modules.

I purchased a Bootcamp course separately. Can this be credited towards the Bootcamp fee?

Absolutely. The entire amount you paid for all courses and learning materials (and which are part of the Bootcamp curriculum) are fully applied toward your enrollment fee.

I have another question.

Ask us by email: service@powerfulpython.com