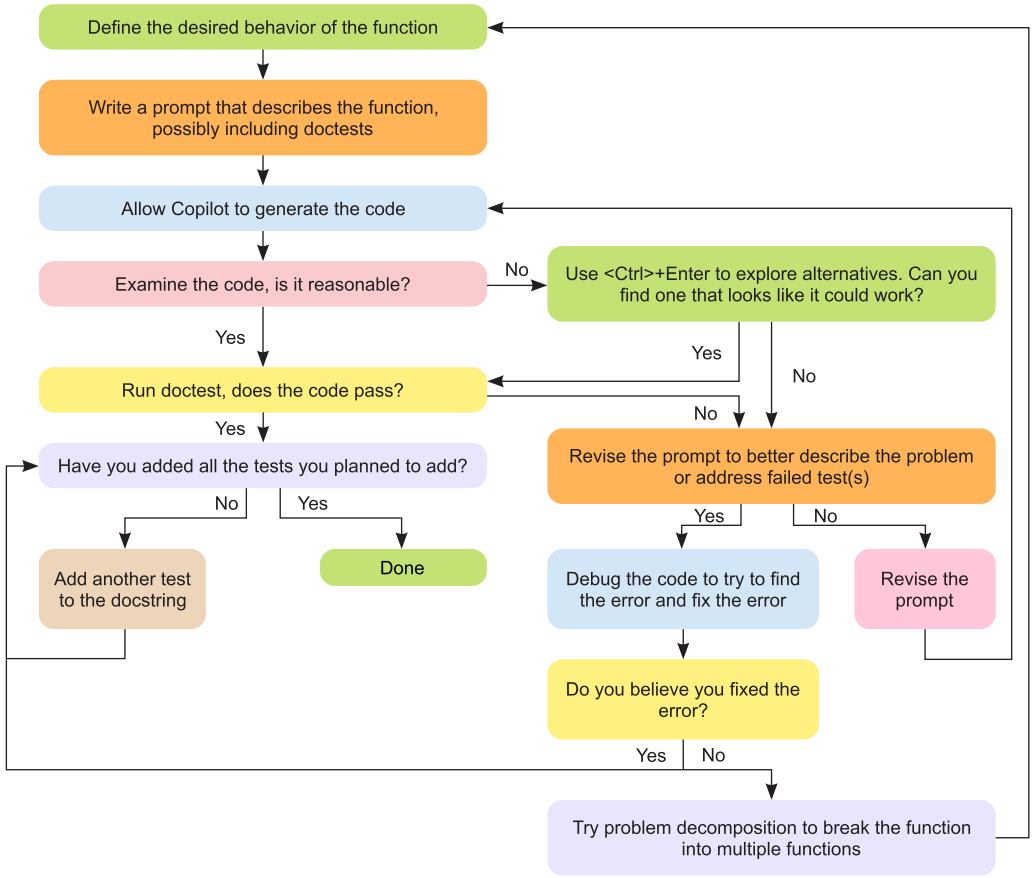
LEARN AI-Assisted Python Programming

With GitHub Copilot and ChatGPT

Leo Porter • Daniel Zingaro

Foreword by Beth Simon, Ph.D.

MANNING



The function design cycle with Copilot, augmented to include debugging

Learn AI-Assisted Python Programming

With GitHub Copilot and ChatGPT

LEO PORTER
DANIEL ZINGARO

MANNING
SHELTER ISLAND

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity.

For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

Development editor:	Rebecca Johnson
Technical editor:	Peter Morgan
Review editor:	Dunja Nikitović
Production editor:	Aleksandar Dragosavljević
Copy editor:	Katie Petito
Technical proofreader:	Mark Thomas
Typesetter:	Tamara Švelić Sabljić
Cover designer:	Marija Tudor

ISBN: 9781633437784
Printed in the United States of America

Dan thanks his wife, Doyali, for trading some of their time, again, to help this book exist.

Leo thanks his wife, Lori, and his children Sam and Avery for their love and support.

contents

foreword

It's an awesome time to learn programming. Why? Let me use an analogy to explain.

I like to make my own bread. I make it more frequently, and more reliably, when I use my stand mixer to knead the dough compared to kneading it by hand. Maybe you'd say that's lazy. I'd say it makes me more productive and more likely to actually make the bread. Maybe you have something that makes your life easier by taking over a tedious task, leaving you free to focus on more important or interesting things. Do you have a car that supports you in parallel parking? I recall when Gmail added spell and grammar checks in languages other than English. My husband's German relative were so excited that he was writing them longer emails—because the effort of remembering little-used German language specifics went away and allowed him to spend more time on the content!

Sadly, until recently, when learning programming, you had no equivalent of a stand mixer or grammar check to support you. And there are lots of tedious things to learn and remember when you start programming.

Good news! As of spring 2023, radically new and (we think) effective support is finally here. You are about to learn programming with one of the most exciting human task supporters so far this century: artificial intelligence. Specifically, this book seeks to support you in developing your ability to program in Python to solve computational problems more easily and faster by teaching you using a tool called GitHub Copilot. Copilot is a programming support tool that uses something called a LLM (large language model) to draw "help" from

a huge number of previously written programs. Once you learn how to direct it (sadly, it's more complicated than effectively using a stand mixer), Copilot can dramatically increase your productivity and success in writing programs to solve your problem.

But should you use Copilot? Are you really learning to program if you use it? Preliminary evidence looks positive—showing that students who learned with Copilot, when assigned a programming task to be done without the help of Copilot, did better than students who learned without Copilot (and also did the task without Copilot) [1]. That said, compared to what we used to teach in an introductory programming class, there are different skills you will need to focus on when programming with Copilot, specifically problem decomposition and debugging (it's OK if you don't know what those are). Just know, practicing programmers need to know those skills as well, but we previously weren't able to teach them explicitly or effectively in introductory courses, because students didn't have the brain space left for learning these "high-level skills" while focusing on nit-picky things like spelling and grammar (programming languages have these, just like real world languages).

Leo and Dan are expert computing educators and researchers; the decisions that they've made to guide your learning in this book are grounded in what we know about teaching and learning programming. I'm excited that, with this book, they're taking steps toward what the next wave of teaching programming will look like.

So, congratulations! Whether you have never done any programming or whether you started to learn before and got frustrated… we think you will find learning to program with Copilot transformative and will allow you to engage your brain in more meaningful and "expert-like" programming experiences!

—Beth Simon, Ph.D.

acknowledgments

Writing a book about technology in flux was new for us. Each day of writing started with us reading the new articles, opinion pieces, and capabilities of LLMs. Early plans had to be scrapped or revised. New ideas presented themselves for later chapters only after we'd written earlier chapters and had access to the latest LLM features. We thank the entire Manning Publications team for their agility and help throughout the process.

In particular, we thank our Development Editor Rebecca Johnson for her expertise, wisdom, and support.

Rebecca provided insightful feedback, constructive criticism, and creative suggestions that have greatly improved the quality and clarity of our work. Rebecca was supportive and encouraging and helped us manage book timelines and our busy schedules. Thank you, Rebecca—you went above and beyond for us.

We also thank our Technical Editor Peter Morgan and our Technical Proofreader Mark Thomas. Both offered valuable contributions to the quality of the book.

To all the reviewers: Aishvarya Verma, Andrew Freed, Andy Wiesendanger, Beth Simon, Brent Honadel, Cairo Cananea, Frank Thomas-Hockey, Ganesh Falak, Ganesh Swaminathan, Georgerobert Freeman, Hariskumar Panakmal, Hendrica van Emde Boas, Ildar Akhmetov, Jean-Baptiste Bang Nteme, Kalai C. E. Nathan, Max Fowler, Maya Lea-Langton, Mikael Dautrey, Monica Popa, Natasha Chong, Ozren Harlovic, Pedro Antonio Ibarra Facio, Radhakrishna Anil,

Snehal Bobade, Srihari Sridharan, Tan Wee, Tony Holdroyd, Wei Luo, Wondi Wolde, your suggestions helped make this a better book.

We thank our colleagues for supporting our work and offering their ideas for what such a book should attempt to do. Many of their ideas have informed our thinking as we sought to redefine what an introductory programming course looks like. We particularly thank Brett Becker, Michelle Craig, Paul Denny, Bill Griswold, Philip Guo, and Gerald Soosai Raj.

introduction

Software is essential today. It's hard to think of any industry where software isn't changing practically everything about how work is done. Manufacturing needs software to monitor production and shipping, let alone the robots that increasingly perform the actual task. Advertising, politics, and fitness, among others, are awash in big data and they routinely use software to make sense of it. Video games and movies are created using software. We could go on and on, but you get the point.

The result has been that more people than ever want to learn how to program. We're not just talking about the computer science, computer engineering, and data science majors at universities who have been in a perpetual "enrollment crisis" for the past decade. We're also talking about the scientist who needs to write software to evaluate their data, the office worker who wants to automate some of their tedious data processing tasks, and the hobbyist who wants to create a fun video game for their friends.

Despite the desire to learn programming, there are decades of research in our field (computing education) that have identified many reasons for why learning to write software is hard. Even after you figure out how to solve the problem, you have to tell a machine how to accomplish it in a programming language whose rules are unforgiving. Granted, writing programs in a language like Python is substantially easier than in machine code using punch cards, but it's still hard. We know it's hard because we've seen the failure rates of introductory computer science courses. We've seen first-hand as we've watched

motivated and intelligent students fail our courses, sometimes multiple times, before they succeed or, worse, give up.

But what if we could talk to computers in a better way? A way that doesn't require us to know all the detailed syntax rules that trip up most novices. That era has just begun thanks to AI assistants like Copilot that offer intelligent code suggestions in the same way ChatGPT can write reasonable text when prompted. This book is for everyone who wants to learn how to write software in the AI assistant era. We're excited to be on your learning journey with you.

AI assistants change how programming is done

We'll introduce you to your AI assistant, Copilot, in chapter 1, but we want to give you a brief overview now. If you read the news headlines or even opinion pieces by lauded software engineering professionals talking about Copilot or ChatGPT, you've seen that opinions run the gamut. Some people say that AI assistants mean the end of all programming jobs. Others say that AI assistants are so hopelessly flawed you are better without them. These views of the world are at such extremes that it's easy to poke holes in either argument. AI assistants learn from existing code, so if some new tool/technology is developed, humans will need to write the bulk of the initial code. As a recent article well expressed, there isn't a lot (or any) code out there for quantum computers since they are still in their infancy [1]. So human programmers aren't going away, at least not any time soon. At the same time, in our time working with Copilot, we've seen how powerful it is. Both of us have written software for decades and Copilot can often give us correct code much faster than we could write it on our own. To ignore such a powerful tool seems analogous to a carpenter refusing to use power tools.

As educators, the opportunity to help people learn to write software is instantly apparent. Why should students spend so much time fighting with syntax when writing code from scratch when the code suggested by an AI assistant is almost always syntactically correct? Why should students have to reach out to professors, instructional staff, friends, or internet forums for help explaining what a section of code is doing when AI assistants are really good at explaining code (particularly for questions asked by novices)? And if AI assistants often write correct code when solving common programming problems (by learning from huge volumes of code written in the past), why shouldn't students be using it to help them program?

Be warned that this doesn't mean that writing software is now just easy and that we can entirely offload the skill of programming onto the AI. Instead, the skills to write good software are evolving. Skills like problem decomposition, code specification, code reading, and code testing have become even more

important than they were in the past; skills like knowing library semantics and syntax become less important. We'll say more about this in the next chapters, but this book will teach you the skills that matter going forward. These skills will be valuable whether you dabble in writing software from time to time or you are starting a career in software engineering.

Audience

We have two primary audiences for the book. The first is everyone who has thought about writing software (and even tried and failed before) to make their lives better in some way. This includes the accountant who gets frustrated that their software can't do what they want so they are left solving problems by hand. Or scientists who want to analyze their data quickly, but existing tools aren't capable of doing what they want. We also imagine the office manager who feels limited by what their spreadsheet software can do and wants a better way to gain insight from their data. Additionally, we imagine the exec at a small company who wants to be notified when something is said publicly on social media about their company but can't afford to pay a software engineering team to write the tool for them. And we imagine the hobbyist of any age who just wants to write software for fun—whether it be for making their own small video games, storytelling with pictures, or creating fun family photo collages. These are just some of the people who want to write software to improve some element of their professional or personal lives.

The second is the student who is considering a career in software engineering or programming and wants to learn how to write software. They want to learn the basics and start creating interesting software, without the trappings of a classic computer science class. Certainly, there will be more courses or books that will follow this first book on the road to becoming a professional software developer, but this will hopefully be a fun and rewarding first step.

What we expect from you

This book requires no background whatsoever in programming. If you learned some programming and forgotten or it didn't go well the first time, we think this is a great place to resume your learning.

This book does require basic computer literacy. This means you should be comfortable installing software, copying files between folders, and opening files on your computer. If you don't have those skills, you could still start this book, but realize there may be moments when you need to look to outside resources (e.g., YouTube videos on how to copy a file from one folder to another).

You'll also need a computer where you have permission to install software so you can follow along and apply the ideas we're learning. Any Windows, Mac, or Linux personal computer or laptop will work.

What you will be able to do after reading this book

In this book, we're going to teach you how to use Copilot to write Python code. We'll teach you how to identify whether that code does what you want, and what to do when it doesn't. We'll teach you enough about Python to be able to read it for a general understanding of what it does and whether it is doing something potentially meaningful.

We won't, however, teach you how to program in Python entirely from scratch. You'll be in a good position to learn to do that with other resources following this book if you like—but for many tasks, as we will show you, it may not be necessary.

We don't know exactly what it will look like to be a professional programmer or software engineer in light of AI coding assistants. That role is already changing and will change further as the AI technology improves. For now, we will say that you need more than this book to be a professional programmer or software engineer. You'll need to know a great deal more about Python and other computer science topics to get there.

The good news is that learning how to program using Copilot will make you capable of writing basic software to address common needs. The software will be more complex than what we typically teach in an introductory course, and you'll be able to write these useful programs without banging your head on syntax or spending months learning just Python. If you wish to continue learning about writing professional software, this will be your first step toward mastery.

By the end of this book, you will be able to write basic software capable of data analysis, automating repetitive tasks, and creating simple games, among many others.

The challenge in working with AI assistants

We expect you're ready to jump into a technology that is maturing and changing quickly. *What you see from Copilot may not match what you see in the book.* Copilot is advancing and changing daily, and we cannot possibly keep up to the minute with such a moving target. More than that, Copilot is nondeterministic, which means that if you ask it to solve the same task multiple times, it may not give you the same code each time. And sometimes you'll get correct code for a task, but then if you ask again, you get code that is not correct. So even if you use the exact same prompts we do, you will likely see different code responses than we do. Much of this book is devoted to ensuring you learn how to determine

whether the answer from Copilot is right or not and, if it isn't, how to fix it. In short, we hope you're ready for what it means to learn on the leading edge of technology.

Why we wrote this book

Both of us have been professors for over a decade and programmers for a decade longer than that. Our care for our students' success led us to become researchers studying how students learn computing and how to improve their outcomes. Between the two of us, we've written nearly a hundred articles in our field exploring pedagogies, student beliefs, and assessments—all with the goal of improving the student experience.

We've also had students in our office hours who struggled to learn how to program, even when we are employing best practices in teaching computing. These are intelligent students who want to learn, but who are tripped up on some part of the programming process. The programming process has many steps, from understanding a problem, to coming up with a solution, to imparting the process of solving the problem to a computer. So, when we began working with AI assistants, specifically Copilot, we instantly saw how it could be a game changer for students, particularly in improving that last step "imparting the process of solving the problem to a computer". We want our students to succeed. We want you to succeed. And we believe AI assistants can help.

Warning: beware of elitism

One of the saddest things we see in our classes at our universities is students intimidating other students. We've heard students in our introductory Python programming courses try to show off how they already learned to program in such-and-such programming language and the affect that has on the other students in the course. Although we try to gently point these students to other, more appropriate courses, we've also seen that the students bragging in this way are often the students struggling to pass at the end of the term, having vastly over-estimated their proficiency at the start. And it doesn't take a licensed psychologist to see that this kind of posturing comes from a place of low self-esteem.

Beyond students in our introductory courses, we see how different kinds of programmers treat each other and their respective fields. For example, Human-Computer Interaction (HCI) professionals study how to improve the design of software to make it better for its human users. Sounds important, right? Unfortunately, that field was put down by computer scientists as merely "applied psychology" for years, and then major companies showed that maybe, just maybe, if you care about the users of your technology, those people might

just appreciate it more and be inclined to buy it. It's not surprising that HCI quickly became mainstream in computer science. This snobbery isn't limited to specific fields. We even see it occurring between programmers of different languages. For example, we've seen C++ (one programming language) programmers say silly things like JavaScript (another programming language) programming isn't real programming. (It definitely is real programming, whatever that might mean!)

All of this, in our opinion, is unproductive and unfortunate posturing that pushes people away from the field. A comic we both enjoy called XKCD, captured the ludicrousness of this posturing well in "Real Programmers" [2]. In the comic, programmers argue about what the best text editor app is for programming. Programmers need to use a text editor to enter their code, which is exactly what you'll start doing in chapter 2. There's been a long-standing, and mostly unserious, debate over the best editors ("emacs" is one of many editors). The comic is making light of the meaninglessness of the debate in a truly clever way.

The reason we're talking about this unfortunate aspect of our field is we know what some people will say about learning to program with Copilot. They'll say that to learn to write software, you have to learn how to write code entirely from scratch. And for future professional engineers, we actually agree that at some point in your career, you should learn to write code from scratch. But, for most people and even people starting their studies in software engineering, we wholeheartedly disagree that writing code entirely from scratch makes sense anymore as a starting place. So, if someone criticizes you for doing something to make yourself or your life or the world better, we encourage you to look to the immortal wisdom of Taylor Swift and just "Shake it off".

How this book is organized: a roadmap

This book is divided into 11 chapters. We recommend that you read this book from beginning to end, rather than skipping around. That's because most chapters introduce skills that will be assumed in later chapters:

- Chapter 1 describes what AI code assistants are, how they work, and why they are irrevocably changing how programming is done. It also explores the concerns we need to keep in mind when using AI coding assistants.
- Chapter 2 helps you set up your computer to be able to program with GitHub Copilot (that's your AI coding assistant) and Python (that's the programming language we'll use). Once your computer is set up, we'll use Copilot in our first programming example: doing some analysis on freely available sports data.