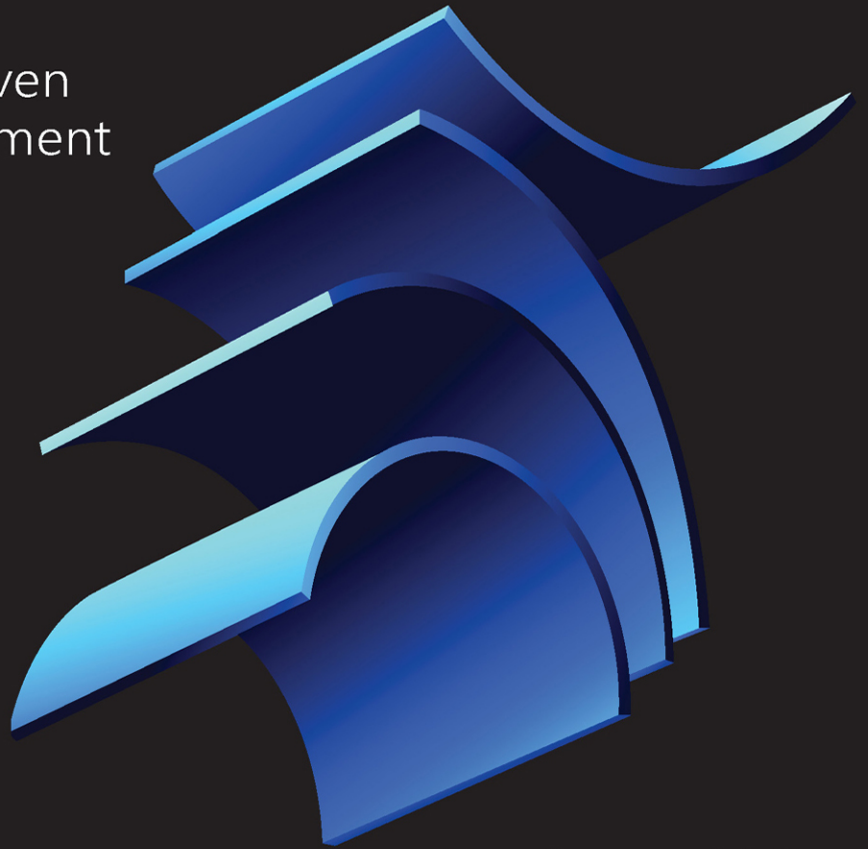


GitHub Copilot

Step by Step

Navigating AI-driven
software development



Dr. Gomathi S.

FREE SAMPLE CHAPTER |





Github Copilot Step by Step: Navigating AI-driven software development

Dr. Gomathi S.

Github Copilot Step by Step

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

Copyright © 2026 by Pearson Education, Inc.
Hoboken, New Jersey

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/global-permission-granting.html.

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-13-549304-5
ISBN-10: 0-13-549304-8

Library of Congress Control Number: On File

\$PrintCode

Trademarks

Microsoft and the trademarks listed at <http://www.microsoft.com> on the "Trademarks" webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

Cover: BonkersArt/stock.adobe.com
Figures 1.1, 1.4, 2.1, 2.2, 8.1–8.12, 9.1–9.14, 10.1: © 2025 GitHub, Inc
Figure 1.3: © 2000–2025 JetBrains s.r.o.

Editor-in-Chief

Julie Phifer

Portfolio Manager

Loretta Yates

Acquisitions Editor

Shourav Bose

Development Editor

Songlin Qiu

Managing Editor

Sandra Schroeder

Production Editor

Mary Roth

Copy Editor

Kitty Wilson

Technical Editor

Charles Pluta

Indexer

Timothy Wright

Proofreader

Barbara Mack

Compositor

codeMantra

Contents

Foreword	xvii
Acknowledgments	xix
About the author	xxi



Introduction	xxiii
Who this book is for	xxiii
How the book is organized	xxiv
Download the practice files	xxv
Errata, updates, and support	xxv



Introduction to GitHub Copilot	1
Practice files	1
Understand GitHub Copilot and its core functionality	2
How does it work?	2
Why does it matter?	3
Where can you use it?	3
Responsible AI in Copilot	5
Explore the AI technology behind GitHub Copilot	5
What is OpenAI codex?	6
From prompts to programs: Natural language understanding	6
How Copilot is different from IntelliSense	7
How it learns: Machine learning in action	8
Continuous improvement	8
Identify the key benefits of GitHub Copilot for developers	8
Core capabilities of GitHub Copilot	9
Real-world use cases of GitHub Copilot	13
Why Copilot matters for developers	17
Where Copilot fits into the developer workflow	20

Learn how GitHub Copilot handles privacy and applies responsible AI	23
Ethical considerations	23
Source code licensing and open-source debates	24
Responsibility: The developer is accountable	24
Copilot’s usage guidelines and best practices	25
Recognize the limitations and ethical considerations of GitHub Copilot	25
Benefits	25
Limitations.	27
Skills review.	29
Practice tasks	30
Understand GitHub Copilot and its core functionality.	30
Explore the AI technology behind GitHub Copilot	30
Identify the key benefits of GitHub Copilot for developers	31
Learn how GitHub Copilot handles privacy and applies responsible AI . . .	31
Recognize the limitations and ethical considerations of GitHub Copilot . .	31

2

Setting up GitHub Copilot for your development workflow.	33
Practice files	33
Set up your GitHub account and select a Copilot plan.	34
Prerequisites and account setup.	34
GitHub Copilot plans: Free vs. paid	35
Getting free access to GitHub Copilot Pro as a student, teacher, or open-source maintainer	37
Install Copilot in your development environment.	39
GitHub Codespaces	39
Visual Studio Code (VS Code)	40
Visual Studio (2022 and later)	42
VS Code for Web (GitHub.dev)	42
JetBrains IDEs (IntelliJ IDEA, PyCharm, WebStorm, etc.)	43
Neovim.	43
Azure Data Studio.	43
Eclipse.	44
Configure permissions and personal settings.	44

Exploring the GitHub Copilot interface in VS Code	46
Test Copilot with sample prompts across languages	50
Step 1: Creating a file for each language	50
Step 2: Writing a sample function with a comment	51
Step 3: Testing suggestions in multiple languages	51
Troubleshoot installation issues and common errors	53
Copilot extension not appearing in the IDE	53
Authentication or sign-in issues	54
Copilot suggestions not appearing	54
Extension installation fails or is blocked	55
Copilot Chat or Labs features not available	55
Slow or laggy suggestions	55
Apply best practices for using Copilot effectively	56
When to accept, modify, or reject suggestions	56
Using comments effectively to prompt Copilot	57
Setting realistic expectations as you onboard	58
The importance of security and compliance checks	59
Explore Copilot's functionality in online and offline modes	59
GitHub Codespaces integration	59
How Copilot works without an internet connection (limitations)	60
Skills review	62
Practice tasks	63
Set up your GitHub account and select a Copilot plan	63
Install Copilot in your development environment	63
Configure permissions and personal settings	63
Test Copilot with sample prompts across languages	64
Troubleshoot installation issues and common errors	65
Apply best practices for using Copilot effectively	65
Explore Copilot's functionality in online and offline modes	66

3

Writing effective prompts for AI-powered coding	67
Practice files	67
Understand what makes a prompt for AI code generation effective	68

What makes a prompt effective?	68
Compare vague vs. effective prompts	68
Prompt writing tips for Python	69
Use comment-based, function signature, and docstring prompts.....	71
Comment-based prompts	72
Function signature prompts.....	72
Docstring prompts	73
Prompt style comparison	73
Refine or rewrite prompts to improve suggestion quality	74
Why refining prompts matters	74
Techniques to refine prompts	75
Apply prompt templates for algorithms, tests, refactoring, and more	77
Algorithm prompt template.....	78
Unit test prompt template	78
Refactoring prompt template	79
Documentation prompt template	80
Prompt templates at a glance	81
Customize prompts for specific languages, libraries, and frameworks.....	81
Mention the library or framework name	82
Follow the patterns of the framework.....	83
Reference common functions or idioms.....	83
Include initialization code where needed	83
Templates for specific libraries and frameworks at a glance.....	84
Identify and fix vague and inefficient prompts	84
Why vague prompts fail	85
Recognizing inefficiency in prompts.....	85
Add context when needed.....	86
Prompt quality checks	87
Build an iterative workflow to continuously improve prompt results	88
Start small and build up.....	88
Test and evaluate the output.....	88
Refine based on what you get.....	89
Use comments and structure for context.....	89
Maintain a prompt log.....	90

Experiment and compare	90
Keep improving through feedback	90
Skills review	91
Practice tasks	92
Understand what makes a prompt for AI code generation effective.	92
Use comment-based, function signature, and docstring prompts	92
Refine or rewrite prompts to improve suggestion quality	93
Apply prompt templates for algorithms, tests, refactoring, and more	93
Customize prompts for specific languages, libraries, and frameworks	93
Identify and fix vague and inefficient prompts	94
Build an iterative workflow to continuously improve prompt results	94

4

Enhancing code efficiency with AI assistance	95
Practice files	95
Apply advanced prompting to guide GitHub Copilot	96
Rule of thumb: Write prompts the way you'd explain the task to a smart intern.	96
Prompting strategies to try	96
Common prompting styles	103
Automate repetitive tasks using AI	104
What counts as a repetitive task?	104
How to use Copilot to automate repetitive tasks	105
Real-world use cases for automating repetitive tasks with Copilot	107
Refactor inefficient or legacy code	112
How Copilot helps with refactoring	112
Boosting performance with Copilot	116
Implement clean coding practices	118
The CLEAR framework to guide clean coding	118
Examples of using Copilot to clean code	120
Common clean coding mistakes Copilot helps avoid	121
Copilot-powered developer toolkit	123
Improve code readability and structure	124
What makes code readable?	125

Strategies to improve readability and structure with Copilot	126
Evaluate AI suggestions for quality	128
Adopt a Reviewer’s Mindset	128
Copilot code quality review checklist	132
Skills review	134
Practice tasks	135
Apply advanced prompting to guide GitHub Copilot	135
Automate repetitive tasks using AI	135
Refactor inefficient or legacy code	136
Implement clean coding practices	136
Improve code readability and structure	137
Evaluate AI suggestions for quality	137

5

Debugging and troubleshooting code with Copilot.....	139
Practice Files	139
Identify common code issues with GitHub Copilot.....	140
Common code issues Copilot can help identify.....	140
Use Copilot with a debugging checklist	142
Strategy: Using the 5 Whys method for root cause analysis.....	142
How Copilot detects issues in real time	144
Use Copilot to suggest and apply bug fixes	145
Understand Copilot’s role in fixing bugs	145
Best practices for applying Copilot bug fix suggestions	146
Use Copilot with test failures	148
Interactive troubleshooting flow: A step-by-step guide	149
Improve error handling by using AI-assisted recommendations	150
Use Copilot to insert try-except or try-catch blocks.....	151
Enhance validation logic and input checking	152
Use AI to handle edge cases and provide fallback logic	152
Optimize code performance through Copilot-driven insights	153
Recognize inefficient code patterns	154
Use built-in functions and language idioms	155
Optimize loops and recursive logic	156

Skills review	158
Practice tasks	159
Identify common code issues with GitHub Copilot	159
Use Copilot to suggest and apply bug fixes	159
Improve error handling by using AI-assisted recommendations	160
Optimize code performance through Copilot-driven insights	161

6

Writing and automating tests with GitHub Copilot.....163

Practice files	163
Generate unit tests and integration tests using GitHub Copilot	164
What are unit tests?	164
What are integration tests?	166
Languages and frameworks Copilot supports for testing	168
Benefits of generating tests with Copilot	168
Automate test case creation to reduce repetitive coding	169
Use comments to describe multiple test scenarios	169
Automate table-driven tests	170
Extend existing test files	172
Generate negative and edge case tests	173
Copy and paste refactoring in a prompt	173
Suggested structure for reusable test prompts	174
When to automate test creation with Copilot	174
Work with popular testing frameworks	175
Use Copilot with unittest	175
Use Copilot with pytest	176
When to use unittest vs. pytest with Copilot	177
Use framework-specific prompts to guide Copilot	178
Write clear and structured test prompts for Copilot to follow	178
Start with a clear intent	179
Use structure and test language	179
Mention the framework, if needed	180
Combine comments and code for better context	180

Add one example to steer the output	181
What to avoid in prompts	181
Suggested structure for effective test prompts	182
Apply best practices to improve the reliability of AI-generated tests	182
Always review AI-generated test logic	183
Validate against real use cases	183
Avoid over-trusting happy paths	184
Be consistent with naming and structure	184
Use fixtures and setup blocks when needed	185
Keep tests isolated	185
Document assumptions in comments or docstrings	186
Don't forget security and edge conditions	186
Summary of best practices	187
Skills review	187
Practice tasks	188
Generate unit tests and integration tests using GitHub Copilot	188
Automate test case creation to reduce repetitive coding	189
Work with popular testing frameworks	189
Write clear and structured test prompts for Copilot to follow	189
Apply best practices to improve the reliability of AI-generated tests	190

7

Using GitHub Copilot for code reviews and collaboration	191
Practice files	191
Explore Copilot's help with code reviews	192
Copilot code review vs. manual code review	192
Use Copilot to spot inefficiencies	193
Detect outdated or deprecated patterns	194
Highlight potential bugs and code smells	195
Balance AI assistance with human judgment	196
Common review prompts and their benefits	196
Generate refactoring suggestions	197

Prompt Copilot for refactoring ideas	198
Simplify conditionals and remove redundancy	199
Rename variables and functions consistently	199
Identify opportunities to extract helper functions	200
Generate refactoring suggestions	202
Create clear inline documentation	203
Generate docstrings automatically	203
Add meaningful comments to complex logic	204
Keep comments and docstrings up to date	205
Avoid overly generic or incorrect documentation	206
Create clear inline documentation	206
Use Copilot for alternative implementations	207
Prompt Copilot for different solutions	208
Explore performance improvements	209
Compare alternatives before choosing	210
Discover new Python idioms and patterns	210
Use alternative implementations	212
Collaborate with teammates using AI-enhanced feedback	212
Use Copilot to suggest improvements in pull requests	213
Share AI-generated snippets in chat or documentation	213
Improve pair programming efficiency	214
Keep your team aligned with consistent patterns	214
Encourage thoughtful discussions	215
Best practices for collaborating with Copilot in code reviews	216
Common Q&A	218
Skills review	219
Practice tasks	220
Explore Copilot's help with code reviews	220
Generate refactoring suggestions	220
Create clear inline documentation	221
Use Copilot for alternative implementations	221
Collaborate with teammates using AI-enhanced feedback	221

8

Using AI-powered development workflows in real-world scenarios	223
Practice files	223
Discover real-world Copilot use cases.....	224
Web development	224
Backend development.....	227
Data science and analytics	230
DevOps and automation.....	234
Testing and debugging	236
Explore AI workflows	239
The AI-assisted development lifecycle	239
Manual vs. Copilot-assisted workflow.....	244
Best practices for Copilot in your workflow.....	245
Understand the role of AI across industries.....	248
Healthcare and life sciences.....	249
Finance and fintech.....	252
Education	254
Manufacturing and logistics	257
Apply practical examples of AI in your projects.....	259
Web project example: Contact form handler	259
Data analysis example: Clean a DataFrame	260
Backend logic example: Calculate user discounts	260
DevOps example: Bash script for file cleanup	261
Testing example: Write unit tests	261
Reflect on Copilot's impact on teams	262
Improved collaboration and shared understanding.....	263
Faster onboarding and mentorship	263
Reduced time to implementation.....	263
AI as a second reviewer	263
Changing developer roles and team dynamics	264
Skills review	264
Practice tasks	265

Discover real-world Copilot use cases	265
Explore AI workflows	266
Understand the role of AI across industries	266
Apply practical examples of AI in your projects	267
Reflect on Copilot’s impact on teams	268

9

Avoiding common pitfalls with GitHub Copilot 269

Practice files	269
Spot and fix common Copilot mistakes	270
Blindly accepting suggestions	270
Overusing Copilot	271
Letting Copilot guess without enough context	271
Failing to check for edge cases or validation	272
Forgetting to match your team’s style	273
Understand why Copilot mistakes happen	274
Copilot doesn’t understand your goal; it predicts patterns	274
Copilot makes it easy to move fast	275
You’re not giving enough context	275
Prompting is still a new skill for many developers	277
Copilot doesn’t know your project’s architecture, data, or rules	277
Fix and refine AI-generated code	279
Break long suggestions into smaller pieces	279
Rename variables for clarity	280
Add comments and documentation	282
Use linters and formatters	283
Review suggestions line by line	283
Guide Copilot effectively	284
Start with clear, specific comments	284
Use examples or constraints in comments	285
Write descriptive function names and docstrings	286
Break logic into steps	287
Don’t settle; try again	287

Practice real-world examples.	288
Scenario 1: Clean up messy logic.	288
Scenario 2: Write helper functions from prompts	290
Scenario 3: Add security and error handling	291
Scenario 4: Use Copilot for SQL, YAML, and HTML	291
Scenario 5: Combine multiple steps.	294
Taking your knowledge into the real world.	295
Skills review.	298
Practice tasks	299
Spot and fix common Copilot mistakes	299
Understand why Copilot mistakes happen	299
Fix and refine AI-generated code.	300
Guide Copilot effectively.	301
Practice real-world examples.	301

10

Exploring the future of AI in software development. 303

Practice files	303
Explore upcoming AI trends in coding	304
Understand the shift to natural language programming	304
Embrace voice-first coding environments.	304
AI-powered coding trends vs. traditional coding approaches.	305
Use AI as a code architect, not just a code generator.	306
Explore immersive and spatial coding environments.	306
Prepare for cross-modal development.	307
The rise of real-time code optimization	308
What developers need to do now	308
Understand the evolution of GitHub Copilot	309
From autocomplete to AI pair programming	309
Copilot Chat: Context-aware conversations	309
The evolution of GitHub Copilot capabilities	310
The rise of AI documentation and learning assistants	310
Integration across the toolchain	312

Copilot for teams and enterprises	312
AI and team collaboration: Beyond the individual developer	313
Integration of Copilot into CI/CD and DevOps pipelines.....	315
Continuous improvement and the future roadmap	316
Discover new developer roles in the AI era	316
AI-augmented developer roles	317
From coder to AI supervisor.....	318
Becoming prompt engineers.....	318
AI-powered collaboration facilitators	319
Responsible AI champions	319
Continuous learners and AI integrators	320
Use cases for Copilot in DevOps and CI/CD.....	320
Embracing vibe coding and AI flow states.....	321
Skills review	322
Practice tasks	323
Explore upcoming AI trends in coding	323
Understand the evolution of GitHub Copilot	323
Discover new developer roles in the AI era	324
Index.....	325

This page intentionally left blank

Foreword

I still remember the day—December 5, 2022—when I used GitHub Copilot for the first time, and it wrote a complete function before I finished writing the comment. It wasn't just faster; it felt different. I felt like I wasn't coding alone anymore, and that moment changed how I thought about writing software. Not because AI replaced my thinking, but because it helped me to think bigger, build faster, and learn continuously while shipping real code.

In the last few years, I've had the privilege of being at the forefront of the Copilot revolution, advising my teams at Microsoft, integrating AI-powered workflows, and sharing best practices with the global developer community. Fast forward to now: we have a matured, powerful AI coding assistant that has evolved far beyond a tool for autocomplete suggestions. GitHub Copilot transformed into a sophisticated pair programmer that understands context, engages in natural language conversations, generates unit tests, refactors legacy code, and even integrates into our development workflows. It is not just a productivity hack anymore, but a foundational tool in reshaping how millions of developers today write and learn from GitHub Copilot. I highly recommend reading my recent *Business Insider** article about how as a Principal Software Engineering Manager at Microsoft, AI helped me in my day-to-day activities.

Dr. Gomathi, a Microsoft MVP, Microsoft certified trainer (MCT), and passionate AI advocate, has poured her deep technical expertise and teaching experience into every chapter, from setting up Copilot across multiple IDEs to crafting effective prompts, automating tests, avoiding common pitfalls, and preparing for the future of AI-powered development. This isn't just another tech book. It's comprehensive with hands-on practical knowledge to help those who want to master GitHub Copilot. I'm grateful she wrote this book. And I'm grateful you're reading it.

We're at the beginning of something transformative. AI-powered development is no longer experimental; it's becoming standard practice in every company. Developers who learn to work *with* AI, who understand how to prompt effectively, iterate intelligently, and maintain human oversight, will have a massive advantage in the years

* <https://www.businessinsider.com/microsoft-manager-ai-reduces-busywork-not-daily-workload-2024-10>

ahead. This book is your foundation for that future. Whether you're a student learning to code, a professional looking to boost productivity, or a team leader exploring AI-assisted workflows, the lessons here will serve you well.

There is a truth I've come to embrace in this new AI and agentic landscape:

"AI will not replace your job, but a person who is leveraging AI can."

Start leveraging GitHub Copilot and go write some code. Copilot is waiting—and so is the future.

Happy coding!

Naga Santhosh Reddy Vootukuri
Principal Software Engineering Manager
Microsoft, Azure SQL Server
(Cloud+ AI division)

Acknowledgments

This book is the result of not just my work but the collective strength, love, and encouragement of many people who have been part of my journey.

I dedicate this book to my beloved mother, whose love and guidance continue to inspire me every day. With deep gratitude, I honor my father, V. Srinivasan, and my husband, Anantha Krishnan A, for their unwavering support and encouragement through every challenge. To my twin gems, Vishanth and Vishwanth, your boundless joy lights up my world.

I extend heartfelt thanks to my in-laws, Lakshmi and Appadurai, for their constant encouragement, and to Mahalakshmi and Padmanaban, whose warmth resembles that of my parents. I am also grateful to my brother, Viswanathan, and my sisters-in-law, Gomathy and Anantha Kalyani, for their love and support.

Special thanks to the publishing and editorial team who have kept me on track and made this book better at every stage. Thank you to Shourav Bose for guiding me through the entire process, Songlin Qiu for shaping the structure, and Dan Foster for refining the narrative with clarity and precision. My sincere thanks also go to Mary Roth, Content Producer, and Jayaprakash P., Senior Project Manager, for their invaluable assistance and coordination during the production phase, ensuring a smooth and successful completion of this book. I am especially grateful to my technical editor, Charles Pluta, for ensuring accuracy, providing valuable tips, and offering positive feedback throughout this journey.

I am equally indebted to my students, mentors, friends, and the global Microsoft community, who constantly inspire me to share knowledge and keep learning.

Finally, to the readers of this book: Your curiosity and commitment to learning are the true motivation behind my work. May this book help you embrace AI-powered development and achieve your aspirations.

This page intentionally left blank

About the author



Dr. Gomathi S. is a Microsoft Most Valuable Professional (MVP), a Microsoft Certified Trainer (MCT) Community Lead, and Microsoft Learn Expert, with more than 14 years of experience in teaching, mentoring, and technology leadership. She has trained thousands of professionals and students globally in Power BI, Business Central, Power Apps, Microsoft Fabric, data science, and AI-driven tools.

As the author of two technical books on Microsoft Dynamics 365 Business Central, she has established herself as a trusted voice in ERP and AI-powered development. Through her YouTube channel *Gom Tech Talks*, global workshops, and community sessions, she continues to inspire professionals, students, and educators alike to embrace technology with confidence.

An accomplished researcher and innovator, Dr. Gomathi holds six Australian and five national patents, has published more than 30 research papers in Scopus and international journals, and serves as a reviewer for leading indexed journals. She is also a record holder in the Asia and India Book of Records and a recipient of the prestigious Enrique Lima Award (Asia Region, 2024), Young Scientist, and Women Scientist Awards.

Beyond her professional achievements, Dr. Gomathi is a passionate mentor, dedicated to empowering women in ERP and technology, and takes pride in balancing her career with being a mother of twin children.

This page intentionally left blank

Introduction



GitHub Copilot is an AI-powered coding assistant designed to help developers write better code faster. Built on OpenAI's Codex and deeply integrated with popular IDEs such as Visual Studio Code, JetBrains, and Neovim, it works alongside you like a virtual pair-programmer. Copilot suggests entire functions, generates unit tests, assists with debugging, and even helps refactor legacy code. It reduces repetitive work, accelerates learning, and makes complex coding tasks more approachable, whether you are building web apps, automating scripts, or exploring data pipelines.

Copilot is more than an autocomplete tool; it is a true productivity partner. You can describe your intent in plain English, and it translates that into working code across multiple programming languages. From writing queries in SQL to creating responsive web layouts in HTML and JavaScript, or developing enterprise logic in Python and C#, Copilot adapts to your project's context and provides intelligent, context-aware suggestions. It helps you focus on the logic and design of your solutions while handling much of the boilerplate code.

Who this book is for

GitHub Copilot Step by Step and other books in the Step by Step series are designed for beginning to intermediate developers. This book is for anyone who wants to understand and use Copilot effectively, regardless of whether you are a student just learning to code, a professional developer seeking to boost productivity, or a team member working in collaborative enterprise environments.

It is especially valuable for:

- Software developers (front-end, back-end, full-stack)
- Data engineers, analysts, and AI/ML practitioners
- Cloud developers and DevOps engineers

- Solution architects and automation engineers
- Students, freelancers, and hobbyists who want to code smarter, not harder

How the book is organized

This book is organized into ten chapters, each building your knowledge of GitHub Copilot in a structured way.

- **Chapter 1** introduces GitHub Copilot, explaining its core features, benefits, limitations, and the responsible AI principles behind it.
- **Chapter 2** walks you through setting up Copilot across popular IDEs such as Visual Studio Code, JetBrains, and GitHub Codespaces, covering account setup, installation, and configuration.
- **Chapter 3** focuses on writing effective prompts, teaching you how to guide Copilot to generate accurate, context-aware code.
- **Chapter 4** shows how to enhance code efficiency with Copilot by automating repetitive tasks, refactoring existing code, and applying best practices for cleaner design.
- **Chapter 5** explores Copilot's role in debugging and troubleshooting, helping you detect errors, suggest fixes, and optimize performance.
- **Chapter 6** demonstrates how to write and automate tests with Copilot, including unit tests and integration testing for different frameworks.
- **Chapter 7** highlights Copilot's value in collaboration by assisting with code reviews, documentation, and shared coding practices.
- **Chapter 8** presents real-world scenarios, showing how Copilot supports workflows in areas like data science, cloud development, and enterprise applications.
- **Chapter 9** identifies common pitfalls developers face when using Copilot and provides strategies to avoid them, ensuring responsible and effective use.
- Finally, **Chapter 10** looks ahead to the future of AI in software development, exploring emerging trends, ethical considerations, and how tools like Copilot will continue to evolve.

This structure supports two types of readers. If you are new to GitHub Copilot or generative AI, the early chapters will help you build a strong foundation with clear explanations, setup instructions, and guided exercises. If you are already familiar with Copilot or prompt engineering, you can skip ahead to the later chapters to focus on advanced use cases, workflow integration, collaboration, and professional practices.

The book has been designed to lead you step by step through the tasks you're most likely to perform with Copilot in your everyday development work. Because generative AI is creative by nature, you will discover new possibilities as you experiment with prompts, explore different coding contexts, and apply Copilot to your own projects. Each topic is self-contained, so you can jump in wherever you need specific skills or reinforcement.

Download the practice files

Before you can complete the exercises in this book, you need to download the book's practice files to your computer. These practice files, and other resources, can be downloaded from the following page:

[MicrosoftPressStore.com/gitcopilot/downloads](https://microsoftpressstore.com/gitcopilot/downloads)

The following table lists the practice files for this book.

Chapter	File
Chapter 5: Debugging and Troubleshooting Code with Copilot	buggy_functions.py
	error_handling_cases.py
	identify_issues.py
	performance_review.py

Errata, updates, and support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed at:

[MicrosoftPressStore.com/gitcopilot/errata](https://microsoftpressstore.com/gitcopilot/errata)

If you find an error that is not already listed, you can report it to us through the same page.

For additional book support and information, please visit *MicrosoftPressStore.com/Support*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *support.microsoft.com*.

Setting up GitHub Copilot for your development workflow

Practice files

There are no practice files for this chapter.

Getting started with GitHub Copilot begins with setting up the right environment for your development needs. Whether you're a beginner exploring AI-powered coding for the first time or a seasoned developer integrating Copilot into an existing workflow, the setup process lays the foundation for a smooth and productive experience. This chapter walks you through everything you need to get Copilot up and running—from choosing a suitable subscription plan to installing Copilot in popular IDEs like Visual Studio Code, JetBrains IDEs, and GitHub Codespaces.

You'll also learn how to configure Copilot to suit your coding preferences, verify that it's working properly, and write your first AI-assisted code. The chapter includes tips on troubleshooting common issues and using best practices to ensure a successful onboarding experience.

2

In this chapter

- Set up your GitHub account and select a Copilot plan
- Install Copilot in your development environment
- Configure permissions and personal settings
- Test Copilot with sample prompts across languages
- Troubleshoot installation issues and common errors
- Apply best practices for using Copilot effectively
- Explore Copilot's functionality in online and offline modes

Whether you're working online or offline, you'll gain a clear understanding of how to optimize your setup to make the most of what Copilot has to offer.

Set up your GitHub account and select a Copilot plan

Before you begin using GitHub Copilot, you need to set up a GitHub account and choose an appropriate subscription plan that fits your development needs. Whether you're a student trying Copilot for the first time or a professional exploring AI-powered development, this section guides you through the account creation process, explores available pricing tiers, and explains eligibility for free access to premium features. With your account and plan in place, you'll be ready to integrate Copilot into your development environment.

Prerequisites and account setup

Before you can start coding with GitHub Copilot, there are a few key steps to complete. This section guides you through setting up your GitHub account, choosing the right plan for your needs, and understanding the differences between free trials and paid subscriptions.

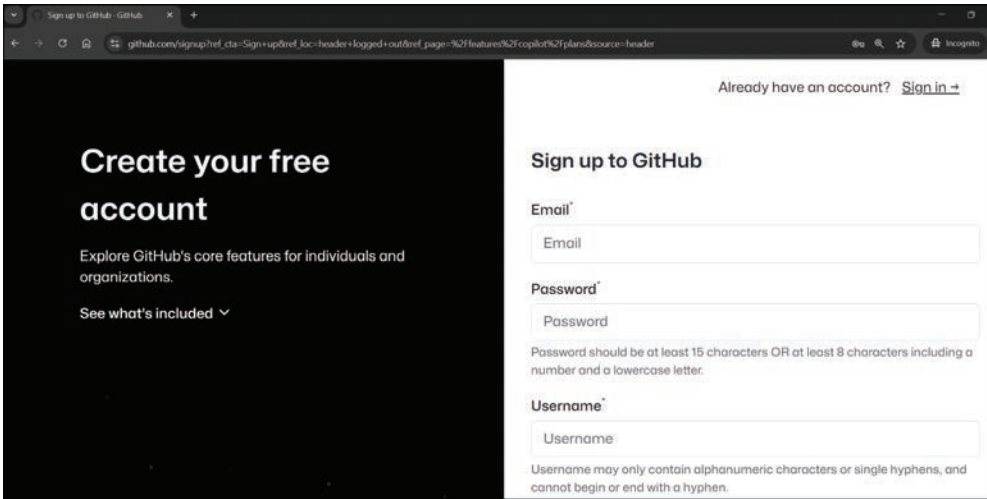


SEE ALSO The next section walks you through installing GitHub Copilot in your preferred development environment—whether on the desktop or in the cloud.

Creating a GitHub account

GitHub Copilot requires a GitHub account. If you don't have one already, follow these steps:

1. Visit <https://github.com>.
2. Click on Sign Up.
3. Enter your email address, password, and username.



The GitHub free account sign-up page

4. Choose whether to receive updates and offers and solve the CAPTCHA.
5. Verify your email address by clicking the link GitHub sends to your inbox.
6. Complete your profile and preferences. (This is optional but recommended for new developers.)

Once you're registered, you'll have access to all of GitHub's features, including repositories, issue tracking, project management tools, and, of course, GitHub Copilot.



TIP You can use the same GitHub account across multiple devices and IDEs. If you're a VS Code user on both desktop and the web, your settings will sync automatically when you're logged in.



IMPORTANT While GitHub Copilot supports multiple IDEs, all examples and demonstrations in this book are provided using Visual Studio Code (VS Code). This ensures consistency and ease of replication for most developers, as VS Code is free and widely adopted and has seamless Copilot integration.

GitHub Copilot plans: Free vs. paid

GitHub Copilot now offers three distinct subscription tiers to suit individual learners, developers, and power users.

The following table details the key differences between GitHub Copilot's Free, Pro, and Pro+ plans, including pricing, feature access, and AI model availability. Please note that features and availability may change over time. For the most up-to-date plan comparison, visit the official GitHub Copilot plans page:

<https://docs.github.com/en/copilot/get-started/plans-for-github-copilot>.

Feature	Free	Pro (Most Popular)	Pro+
Cost	\$0 USD	\$10/month or \$100/year	\$39/month or \$390/year
Best for	Beginners, casual users	Most developers, students, individual pros	Advanced users, power coders, AI tinkerers
Agent/chat access	50 requests/month	Unlimited agent mode and chats with GPT-4o	Access to all models, including GPT-4.5
Completions	2,000/month	Unlimited code completions	30 times more premium requests than the Free plan
Models available	Claude 3.5, Sonnet, GPT-4o	All in the Free plan plus Claude 3.7 Sonnet, o1, and more	All in the Pro plan plus and GPT-4.5
Premium model usage	Limited	6 times more requests for latest models than with the Free plan	30 times more requests than with the Free plan; upgradable
Free trial	Always free	30-day free trial available	No free trial



TIP Students, teachers, and verified maintainers of popular open-source projects can access the Pro plan for free under GitHub Education benefits.

Which plan should you choose?

Which GitHub Copilot plan is right for you depends on your development goals and frequency of use, as well as the level of AI capabilities you need. Here's a quick guide to help you decide:

- Choose the Free plan if you're just exploring Copilot, experimenting, or coding occasionally.
- Upgrade to Pro if you're coding frequently and need more completions, advanced model access, and unlimited usage.
- Go for Pro+ if you want full model access, including GPT-4.5, and you work with large-scale or high-performance AI tasks.



IMPORTANT The Pro+ plan is best for enterprise developers and advanced individual users who want maximum AI capability with minimal restrictions. It offers significantly more compute and model access.

Getting free access to GitHub Copilot Pro as a student, teacher, or open-source maintainer

GitHub Copilot Pro is a premium AI coding assistant that brings additional features like enhanced suggestions, chat support (in some editors), and priority access to infrastructure. While it's generally available via paid subscription, GitHub offers free access to Copilot Pro for a select group of eligible users, including verified students, teachers, and open-source maintainers, in recognition of their contribution to the tech community.

How to qualify for free access to GitHub Copilot Pro

The following table outlines the eligibility criteria for students, educators, and open-source maintainers to access GitHub Copilot Pro for free.

User category	How to qualify
Verified student	Students can apply at https://education.github.com/students . Proof of academic enrollment (such as a student email address or documents) is required.
Verified teacher	Educators can apply at https://education.github.com/teachers . Verification of teaching status is required.
Open-source maintainer	Maintainers of widely used public repositories may qualify. GitHub automatically evaluates repository popularity and contribution metrics. You can check your eligibility at https://github.com/settings/copilot .



TIP GitHub reevaluates eligibility monthly. So, if your student or maintainer status changes, your access might be revoked unless reverified.

What if you're not eligible?

If you don't meet the free access criteria, you still have two solid options:

- Try GitHub Copilot Pro free for 30 days: Anyone can sign up for a one-time 30-day free trial. After the trial ends, you'll need to subscribe to a paid plan to continue using Pro features.
- Use the GitHub Copilot Free plan: While more limited in functionality, the free Copilot experience still offers AI coding suggestions within select IDEs. It's a good starting point if you're exploring.

How to access Copilot Pro for free (if eligible)

Follow these steps to activate your free Copilot Pro access:

1. Sign in to GitHub by going to github.com and logging in with your GitHub credentials.
2. Go to your Copilot settings. In the upper-right corner, click your profile picture and select Your Copilot from the dropdown.
3. Check for eligibility. If you qualify for free access, you'll see the message "GitHub Copilot Pro" informing you of your eligibility.
4. Activate access by clicking the Get Access to GitHub Copilot button and then configure your usage preferences (e.g., telemetry settings, suggestion behavior), and click Save and Complete.



IMPORTANT You must be signed in to the same GitHub account that has been verified as a student, teacher, or maintainer account. The system will not recognize eligibility across multiple accounts.

Install Copilot in your development environment

GitHub Copilot is compatible with a wide range of integrated development environments (IDEs), enabling developers to integrate AI-assisted coding seamlessly into their existing workflows. From web-based editors to powerful desktop environments, Copilot enhances productivity across languages and platforms. The following table lists the supported IDEs and briefly describes each of them, highlighting how Copilot functions within these environments. The sections after the table provide further information on the IDEs.

IDE Name	Description
GitHub Codespaces	Supports Copilot in browser via GitHub.dev or Codespaces for quick editing.
Visual Studio Code	Provides full-featured support for Copilot, including inline suggestions and Copilot Chat.
Visual Studio	Works with Visual Studio 2022+ for .NET, C#, VB.NET, and enterprise development.
VS Code for Web	Supports browser-based use of Copilot with Provides VS Code (GitHub.dev) for light development.
JetBrains IDEs	Includes IntelliJ, PyCharm, WebStorm, etc. and supports Copilot via official plugin.
Neovim	Provides an advanced text editor with Copilot using community plugins.
Azure Data Studio	Enables SQL/database development through use of Copilot with extensions.
Eclipse	Enables the Java IDE with Copilot support via a plugin. (This is less seamless than VS Code.)

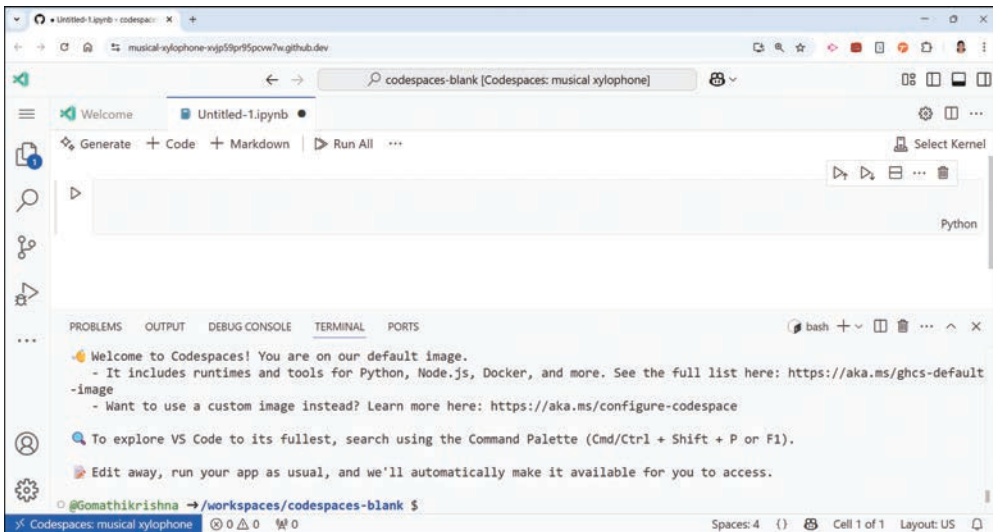
GitHub Codespaces

Installation required: No

Access method: Cloud based

How to use:

1. Create or sign in with a GitHub account that has an active Copilot subscription or trial.
2. Open <https://github.com/codespaces> In your browser.
3. Launch or create a Codespace. Copilot will already be integrated and available in this cloud-hosted development environment.



The GitHub Codespaces environment, displaying a Jupyter Notebook interface with Python kernel and welcome terminal instructions

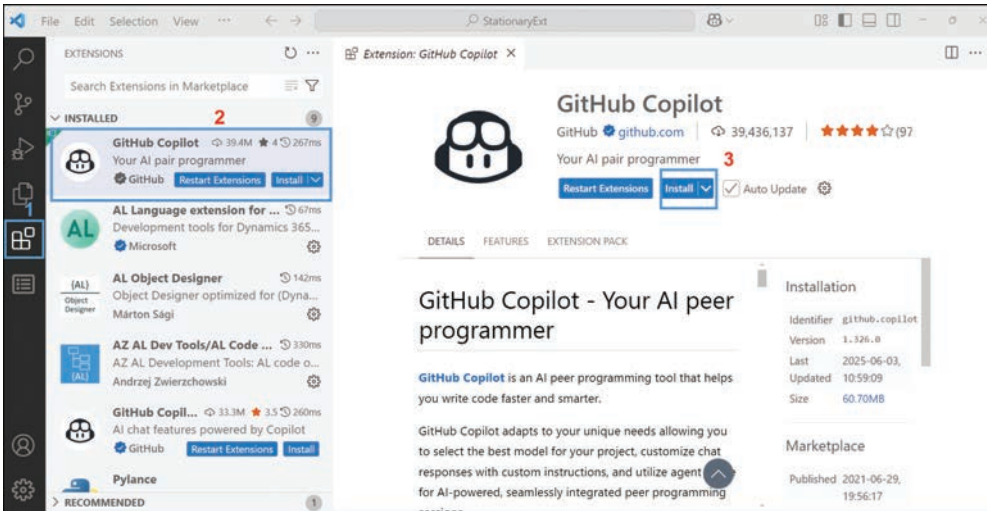
Visual Studio Code (VS Code)

Installation required: Yes

Access method: Desktop app

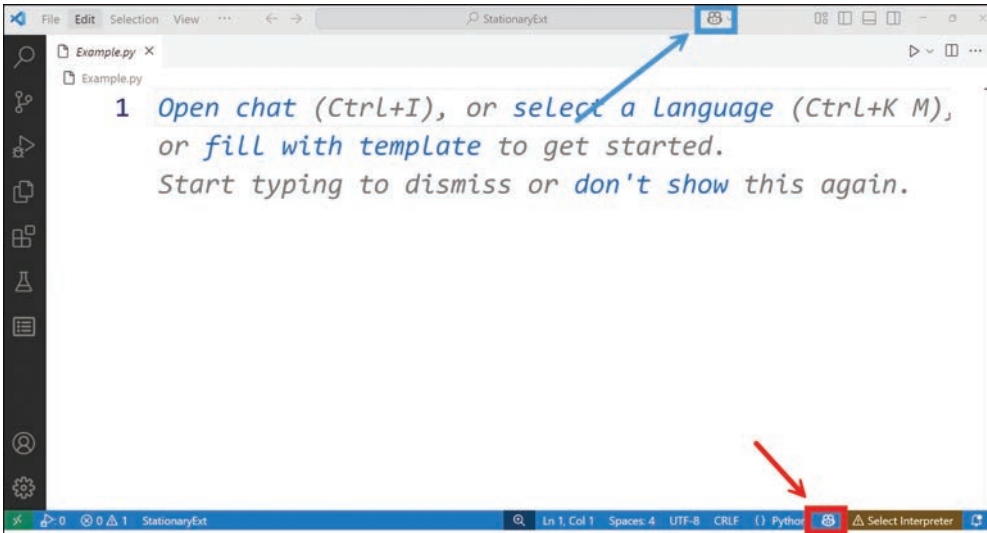
How to use:

1. Download and install VS Code from <https://code.visualstudio.com>.
2. Open VS Code and select the Extensions panel.
3. Search for GitHub Copilot.
4. Click Install as shown in the following image



The GitHub Copilot Extensions panel in Visual Studio Code, highlighting the Install button, extension details, and version information

5. Sign in with your GitHub account to activate Copilot.
6. Begin coding. Copilot suggests completions as you type.



Visual Studio Code interface, showing the GitHub Copilot icons highlighted in both the top menu and bottom status bar, indicating ways to launch the AI assistant

Visual Studio (2022 and later)

Installation required: Yes

Access method: Desktop app

How to use:

1. Install Visual Studio from <https://visualstudio.microsoft.com>.
2. Go to Extensions, search for GitHub Copilot, and install it.
3. Sign in with your GitHub account to enable Copilot.
4. Start coding in supported languages (e.g., C#, VB.NET). Copilot provides suggestions.

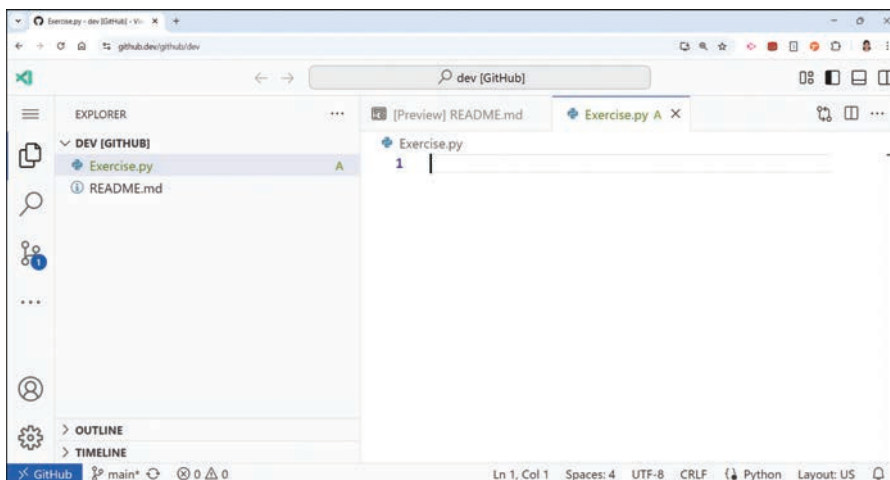
VS Code for Web (GitHub.dev)

Installation required: No

Access method: Cloud based

How to use:

1. Go to <https://github.dev> and create or sign in with a GitHub account.
2. Navigate to any GitHub repository



The Visual Studio Code browser-based interface at github.dev, with a Python file open in the editor and the file explorer displaying the project structure

JetBrains IDEs (IntelliJ IDEA, PyCharm, WebStorm, etc.)

Installation required: Yes

Access method: Desktop app

How to use:

1. Download the JetBrains IDE of your choice from <https://www.jetbrains.com>.
2. Go to Settings > Plugins, search for GitHub Copilot, and click Install.
3. Log in with your GitHub account to activate Copilot. Copilot suggestions appear inline as you write code.

Neovim

Installation required: Yes

Access method: Terminal/desktop

OS: Linux

How to use:

1. Download and install Neovim from <https://neovim.io/>
2. Install the `github/copilot.vim` plugin using a plugin manager (such as `vim-plug`).
3. Launch Neovim and authenticate Copilot through terminal login. Start coding; Copilot will suggest completions inline as you type.

Azure Data Studio

Installation required: Yes

Access method: Cloud based

How to use:

1. Download Azure Data Studio from <https://learn.microsoft.com/en-us/azure-data-studio/>.
2. Install the Azure Data Studio in Visual Studio Code.
3. Go to Extensions, search for GitHub Copilot, and click Install.

4. Sign in with your GitHub account.
5. Begin working with SQL. Copilot provides AI assistance.

Eclipse

Installation required: Yes

Access method: Desktop app

How to use:

1. Download Eclipse from <https://www.eclipse.org>.
2. Go to Help > Eclipse Marketplace, search for GitHub Copilot, and install the plugin.
3. Log in with your GitHub account.
4. Start coding. Copilot provides AI-powered suggestions for Java and other supported languages.

Configure permissions and personal settings

Once GitHub Copilot is installed and running in your IDE, you can tailor its behavior to match your development style and project needs. GitHub offers a variety of customization options that help improve usability, reduce distractions, and maintain control over your development environment. This section walks you through the key settings you can adjust, including how suggestions are presented, how to control Copilot on a per-language basis, and how your data is handled, how to change suggestion behavior (e.g., number of suggestions, inline vs. in a panel), and how to use keyboard shortcuts with GitHub Copilot.

GitHub Copilot offers a set of intuitive keyboard shortcuts that are designed to streamline and accelerate your coding workflow. These shortcuts allow you to interact with Copilot's AI-generated suggestions more efficiently, helping you accept, reject, explore alternatives, and manage how suggestions appear in your editor. Knowing these shortcuts enhances productivity and also reduces the friction between ideation and implementation.

The shortcuts may vary slightly depending on the IDE you're using (such as VS Code, JetBrains IDEs, or Neovim), but most of the commonly used shortcuts are available in Visual Studio Code, which is the IDE used throughout this book.



IMPORTANT The ability to cycle through suggestions using Ctrl+] or Ctrl+[may no longer be available in the latest VS Code Copilot extension. Instead, you can manually trigger suggestions using Alt+/,.

Let's say you're writing a function, and Copilot offers a suggestion. To accept it, all you need to do is press the Tab key. If the suggestion isn't helpful, press Esc to dismiss it.

To boost productivity, GitHub Copilot includes several keyboard shortcuts. The following table lists and describes the most useful shortcuts.

Action	Keyboard Shortcut	Notes
Accept current suggestion	Tab	Inserts the AI-generated code suggestion inline.
Reject current suggestion	Esc	Dismisses the current suggestion.
Trigger Copilot manually (inline)	Alt+/,	Useful when Copilot doesn't auto-suggest or when starting a new line.
Show Copilot Labs feature list	Ctrl+Shift+Alt+E	Opens experimental tools and enhancements from Copilot Labs (if enabled).
Accept suggestion word-by-word	Ctrl+→	Accepts the suggestion one word at a time.
Show multiple suggestions (panel view)	Ctrl+Enter	Opens a list of alternative completions (if supported).
Toggle Copilot sidebar	Ctrl+Shift+A	Opens or closes the Copilot sidebar in VS Code.
Show next Copilot suggestion	Alt+]]	Cycles to the next suggestion (only if cycling mode is available).
Show previous Copilot suggestion	Alt+[[Cycles to the previous suggestion.



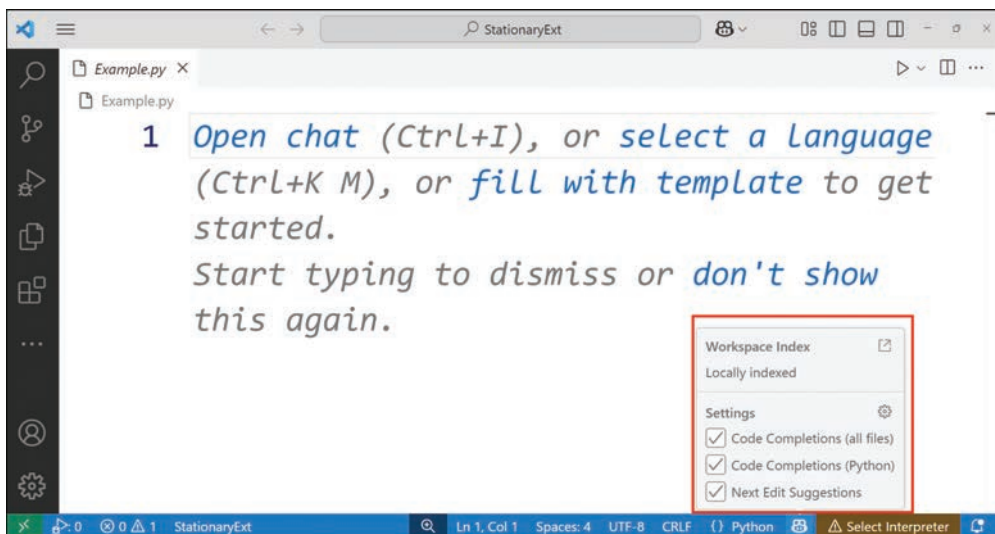
TIP You can customize or view these keyboard shortcuts in Visual Studio Code by navigating to File > Preferences > Keyboard Shortcuts and searching for Copilot. Keeping these shortcuts at your fingertips ensures a smoother, faster, and more productive coding experience with GitHub Copilot.

Exploring the GitHub Copilot interface in VS Code

GitHub Copilot's integration with Visual Studio Code isn't just about code suggestions. VS Code with GitHub Copilot provides a complete interface that empowers you to control when, how, and where suggestions appear. Next, we will look at the key elements of the interface along with practical use cases.

Copilot in the status bar: Real-time control over code suggestions

The Copilot status bar icon, located at the bottom right of the VS Code window (next to the language selector), gives you access to quick toggles and settings. Let's look at the options available when you click the status bar icon and consider some use cases.



Visual Studio Code, showing a Copilot-related settings pop-up, including code completion and next edit suggestions, with workspace indexing set to locally indexed

Locally Indexed (Workspace Index)

What it does: Tells you whether the Copilot suggestions are enhanced with your current project files.

Use case: When you're working in a large codebase like a Django project, local indexing helps Copilot provide more context-aware suggestions, such as referencing existing models, views, or utility functions.

Example:

```
def get_user_email(user_id):
```

Copilot suggests:

```
return User.objects.get(id=user_id).email  
    ...
```

Because your project is indexed, Copilot knows you're using Django ORM.

Code Completions (All Files)

What it does: Enables Copilot suggestions across all file types.

Use case: You're working in a full-stack project with both Python and JavaScript files, and you want Copilot to help in both.

Example: In `app.js`, you begin typing a `fetch()` call, and Copilot suggests a full API call with error handling.

Code Completions (Python)

Use case: You're preparing for a data science assignment and want Copilot's help specifically in `.py` files but not in README files.

Example:

```
import pandas as pd  
df = pd.read_csv(
```

Copilot suggests available CSV files in the project folder.

Next Edit Suggestions

What it does: Suggests what to do next after a code change.

Use case: You just wrote a new class, and Copilot immediately recommends test cases or docstrings.

Example:

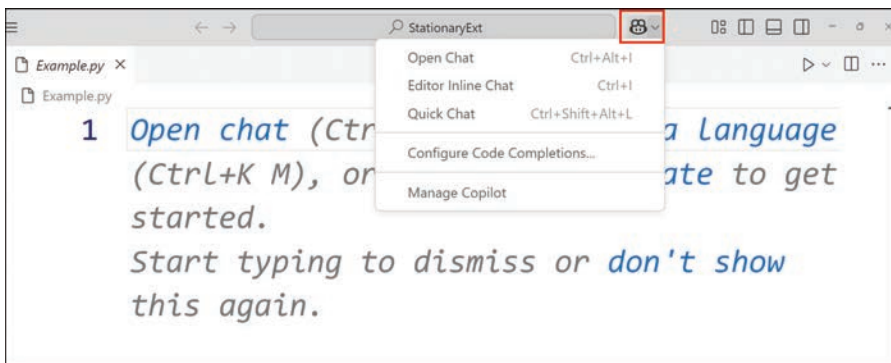
```
class InvoiceProcessor:  
    def process(self):  
        pass
```

Copilot suggests:

```
# def test_invoice_processor():  
#     processor = InvoiceProcessor()  
#     ...
```

Top bar Copilot menu: Interact, chat, and configure

At the top right of VS Code (next to the split editor and layout icons) is the Copilot icon, which you can click to open the GitHub Copilot menu. Next, we'll look at the options available in this menu.



Visual Studio Code with the GitHub Copilot menu expanded, showing multiple interaction and configuration options for using the AI assistant

Open Chat (Ctrl+Alt+I)

Use case: You need help understanding a block of unfamiliar code or generating a function on the fly.

Sample prompts:

"What does this regex do?"

"Generate a function to validate email format."

Benefit: Think of this as your coding assistant; it is conversational, context-aware, and fast.

Editor Inline Chat (Ctrl + I)

Use case: You're editing a class and want to ask for a method without opening the sidebar.

Example: Highlight a method and then type the following prompt:

```
"Convert this function to async and add error handling."
```

Benefit: Perfect for in-place refactoring or micro-improvements.

Quick Chat (Ctrl+Shift+Alt+L)

Use case: You're writing a loop and are stuck. Use Quick Chat for a one-line prompt without switching focus.

Sample prompt:

```
"Generate a for-loop to iterate over a list of files and print the filenames."
```

Configure Code Completions

Use case: You're working on a sensitive codebase and want to limit suggestions to only certain file types.



TIP You can toggle completions for file types like .ipynb, .json, and .sql.

Manage Copilot

Use case: You need to access your Copilot account, disable telemetry, switch to Copilot for Business, or set up organization-wide preferences. Or you need to customize how Copilot handles suggestions, integrates with other extensions, or operates offline.



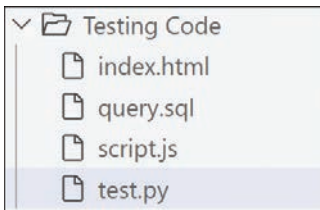
TIP Combine chat, inline chat, and indexing for best results. Chat gives high-level contextual help, inline chat offers precision edits, and indexing ensures that suggestions are project aware.

Test Copilot with sample prompts across languages

This section guides you through the process of verifying GitHub Copilot's installation by writing your first prompt and observing suggestions in multiple programming languages. It also covers how to use keyboard shortcuts to control suggestions.

Step 1: Creating a file for each language

To enable Copilot to understand the language context, you must first create a file with the correct extension in Visual Studio Code.



Folder structure in Visual Studio Code, displaying HTML, SQL, JavaScript, and Python files under the Testing Code directory

To verify Copilot functionality across programming languages, you can start by creating test files. The following table shows the file extension and steps to create them in Visual Studio Code.

Language	File extension	Steps to create a file
Python	.py	File > New File > Save As > test.py
JavaScript	.js	File > New File > Save As > script.js
SQL	.sql	File > New File > Save As > query.sql
HTML	.html	File > New File > Save As > index.html



TIP The file extension tells Copilot which programming language you're working in, enabling it to generate accurate code.

2

Step 2: Writing a sample function with a comment

Once your file is created and open, you can write a natural language comment that describes the functionality you want. Copilot will respond with code suggestions in real time. Let's start with the Python file `test.py`.

Example:

```
#function to calculate factorial of a number
```

Copilot suggestion:

```
def factorial(n):
    if n < 0:
        return "Invalid input"
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
```

The screenshot shows a code editor window with several tabs: `Example.py`, `index.html`, `query.sql`, `script.js`, and `test.py`. The active tab is `test.py`. The editor content shows a prompt and a Copilot suggestion:

```
.vscode > Testing Code > test.py > factorial
1 #function to calculate factorial of a number
2 def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Python code generated by GitHub Copilot, based on a comment to calculate the factorial of a number, demonstrating AI-assisted coding with recursion

Step 3: Testing suggestions in multiple languages

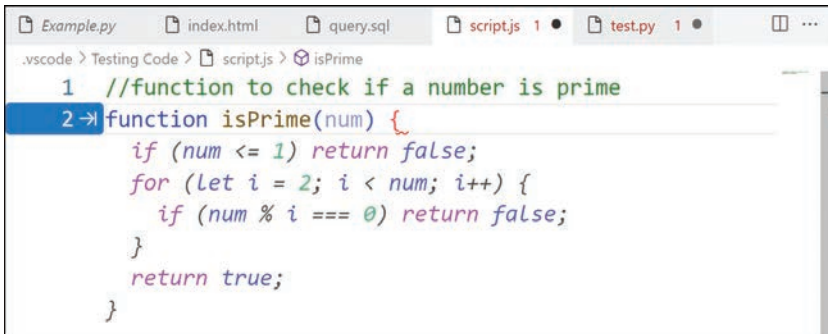
You can try the same idea with other files to verify that Copilot works across languages.

JavaScript (script.js)

Prompt:

```
//function to check if a number is prime
```

Suggested output:



The screenshot shows a VS Code editor window with several tabs: Example.py, index.html, query.sql, script.js (1), and test.py (1). The active file is script.js, and the cursor is at line 2. The code is as follows:

```
1 //function to check if a number is prime
2 → function isPrime(num) {
    if (num <= 1) return false;
    for (let i = 2; i < num; i++) {
        if (num % i === 0) return false;
    }
    return true;
}
```

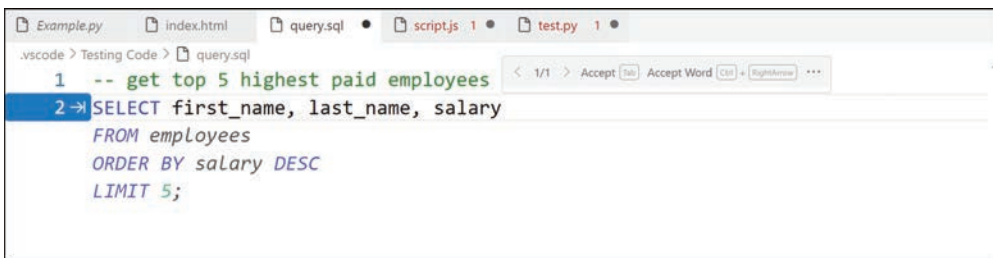
JavaScript function generated by GitHub Copilot from a comment to determine if a number is prime, highlighting AI-assisted code generation with syntax coloring

SQL (query.sql)

Prompt:

```
-- get top 5 highest paid employees
```

Expected output:



The screenshot shows a VS Code editor window with tabs for Example.py, index.html, query.sql, script.js (1), and test.py (1). The active file is query.sql, and the cursor is at line 2. The code is as follows:

```
1 -- get top 5 highest paid employees
2 → SELECT first_name, last_name, salary
    FROM employees
    ORDER BY salary DESC
    LIMIT 5;
```

SQL query generated by GitHub Copilot to select the top five highest-paid employees from an employee table, with Copilot's in-line suggestion options visible

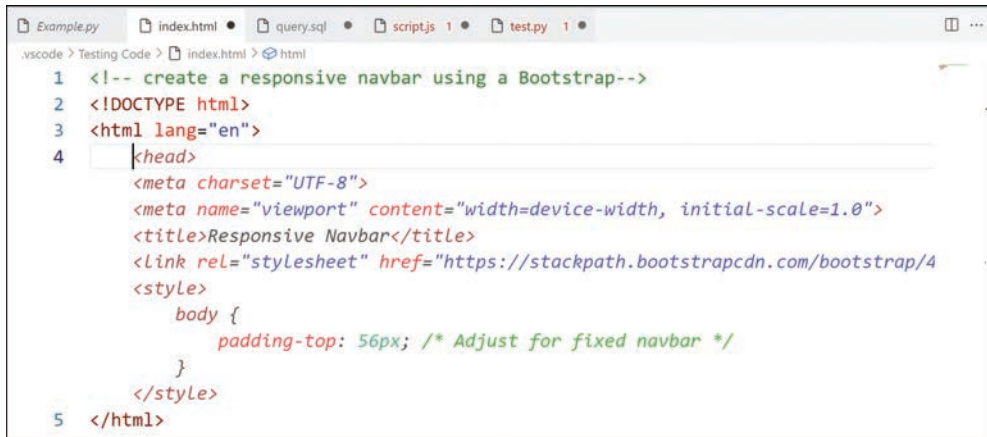
HTML (index.html)

Prompt:

```
<!-- create a responsive navbar using a Bootstrap -->
```

2

Suggested output:



```

1 <!-- create a responsive navbar using a Bootstrap-->
2 <!DOCTYPE html>
3 <html lang="en">
4   <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Responsive Navbar</title>
      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4
      <style>
        body {
          padding-top: 56px; /* Adjust for fixed navbar */
        }
      </style>
5 </html>

```

HTML boilerplate generated by GitHub Copilot based on a comment to create a responsive navbar using a bootstrap, illustrating AI assistance in web development

Troubleshoot installation issues and common errors

Even with a streamlined setup process, you may occasionally encounter issues when installing or activating GitHub Copilot. This section outlines common problems and provides solutions to help you get up and running quickly.

Copilot extension not appearing in the IDE

Problem:

You've installed GitHub Copilot, but the extension doesn't show up in Visual Studio Code or another IDE.

Solutions:

- Ensure that you are using a supported version of the IDE.
- Restart the IDE and check the Extensions panel again.
- Manually trigger Copilot by pressing Ctrl+Shift+P and then search for “Copilot.”

Authentication or sign-in issues

Problem:

You're prompted to sign in repeatedly, or GitHub Copilot refuses to authenticate.

Solutions:

- Make sure you are signed in with a GitHub account that has an active Copilot subscription.
- Open the Command Palette (Ctrl+Shift+P) and run GitHub: Sign Out. Then run GitHub: Sign In again.
- Check your network connection and firewall settings (especially in enterprise environments).

Copilot suggestions not appearing

Problem:

You're writing code, but Copilot does not provide any suggestions.

Solutions:

- Ensure that you are in a supported file type (.py, .js, .html, etc.).
- Press Alt+/ (or Ctrl+Enter) to manually trigger suggestions.
- Check to see if the Copilot icon in the status bar shows Disabled. If it does, re-enable Copilot through the status bar menu.
- Check to see if the Copilot icon in the status bar shows Offline. If it does, reconnect to the internet.

Extension installation fails or is blocked

Problem:

The GitHub Copilot extension fails to install or update.

Solutions:

- Run VS Code as Administrator (Windows) or with elevated permissions (macOS/Linux).
- Try uninstalling and reinstalling the extension.
- Check if your network proxy or antivirus is blocking extension downloads.

Copilot Chat or Labs features not available

Problem:

You can use Copilot suggestions, but advanced features like Chat or Labs don't appear.

Solutions:

- Verify that you are on the Pro or Pro+ plan as some features require higher-tier subscriptions.
- Make sure the GitHub Copilot Chat extension (which is separate from the core Copilot extension) is installed.
- Check experimental feature toggles in your IDE settings.

Slow or laggy suggestions

Problem:

Suggestions take a long time to load, or typing feels delayed.

Solutions:

- Disable other heavy extensions to improve IDE performance.
- Switch to a lightweight file or reduce the number of open tabs.
- Ensure that you have a stable internet connection, as Copilot fetches suggestions from the cloud.



TIP Use the GitHub Copilot status icon in the bottom-right corner of VS Code to get real-time updates on Copilot's connectivity and activity.



SEE ALSO For additional help, refer to the official GitHub Copilot troubleshooting guide:<https://docs.github.com/en/copilot/how-tos/troubleshoot/troubleshooting-common-issues-with-github-copilot>.

Apply best practices for using Copilot effectively

As you begin your journey with GitHub Copilot, understanding how to work strategically and thoughtfully with the tool is key to maximizing its value. Copilot is not just a code-completion assistant; it's a generative AI model that interprets your comments, code structure, and intent to suggest possible implementations. While it can save time and reduce repetitive coding tasks, it's not perfect or infallible.

The initial days of using Copilot are critical for developing productive habits and setting the right expectations. It's important to learn how to interpret the suggestions Copilot provides, how to effectively prompt it using comments, and when to trust or modify its output. Without proper guidance, developers may either over-rely on suggestions or fail to utilize the tool to its full potential.

This section offers a practical set of best practices tailored for first-time users, ensuring that your early experiences with GitHub Copilot are effective, secure, and aligned with your development goals. Whether you're a student, a professional, or an open-source contributor, these practices will help you build confidence and mastery in using AI-assisted development tools.

When to accept, modify, or reject suggestions

Copilot generates code based on context, which means its suggestions may not always be accurate, secure, or optimal. Knowing when to accept, modify, or reject suggestions is a vital part of the learning curve.

Accept suggestions when:

- The code output is syntactically and logically corrected for your needs.

- The suggestion matches your intended structure or logic.
- You've reviewed the suggestion and verified that it meets any security or performance standards.

Modify suggestions when:

- The structure is correct, but the logic needs refinement.
- Variable names, formatting, or comments need to follow your project's conventions.
- The code is almost correct but requires context-specific adjustments.

Reject suggestions when:

- The code is irrelevant, incorrect, or inefficient.
- The code does not align with the best practices (e.g., security, architecture).
- Copilot introduces patterns you do not want junior developers or your team to follow.



IMPORTANT Always review suggestions critically, especially in production-grade or security-sensitive applications.

Using comments effectively to prompt Copilot

Copilot heavily relies on natural language comments to understand what you intend to write. Writing clear, descriptive comments before you start to code can significantly improve the relevance of suggestions.

Consider the following tips for writing effective prompts.

Be specific

A specific prompt like this:

```
# Calculate the factorial of a number using recursion
```

yields better results than this:

```
# Factorial
```

Include structure

If you know the structure you want, include it. For example:

```
# Create a Python function that accepts a list of integers and returns the even numbers.
```

Break down logic

Writing step-by-step logic in comments helps Copilot understand complex tasks. Here's an example:

```
# Step 1: Check if the list is not empty  
  
# Step 2: Filter elements greater than 10  
  
# Step 3: Return the filtered list
```

Use docstrings

Starting with a function name and a docstring helps Copilot infer functionality.



SEE ALSO Chapter 3: Writing effective prompts for AI-powered coding dives deeper into writing clear and effective prompts to get the most out of Copilot's AI suggestions.

Setting realistic expectations as you onboard

Copilot is not a silver bullet. It's a coding assistant, not a replacement for human judgment or experience.

Keep in mind these key points:

- Copilot doesn't understand your business logic. Copilot can't fully grasp your project's domain context or long-term architecture. Always cross-check logic and performance.
- Quality depends on context. The more context you provide—like descriptive comments, file structure, or function names—the better the suggestions.
- Trial and error is part of the process. Don't expect perfect results immediately. Use Copilot iteratively: Try, observe, and modify.

- Monitor code quality and style. Copilot may use patterns or syntax styles that differ from your team's guidelines. Use linters, formatters, and manual reviews to maintain consistency.

The importance of security and compliance checks

Never assume that Copilot-generated code is safe for production. Always run security scans and perform audits as needed.

The following table summarizes common best practices and common pitfalls when using GitHub Copilot effectively.

Do	Don't
Write clear and specific comments	Use vague prompts like // do something
Review suggestions before accepting them	Blindly copy and paste Copilot code
Customize suggestions to fit project standards	Assume that Copilot understands business logic
Use Copilot to speed up the creation of boilerplate code	Use Copilot to replace deliberate design decisions
Start with simpler use cases	Jump straight into complex systems with Copilot

Explore Copilot's functionality in online and offline modes

GitHub Copilot is a cloud-powered AI tool that depends heavily on access to online resources for its real-time suggestions. However, developers today work in a mix of environments—some entirely cloud based (like GitHub Codespaces) and others locally installed and sometimes even offline. Understanding how Copilot behaves in both scenarios is essential to plan your workflow, avoid surprises, and maximize efficiency.

GitHub Codespaces integration

GitHub Codespaces is a cloud-hosted development environment that runs entirely in the browser or from Visual Studio Code. It allows you to spin up container-based dev

environments with all your tools, dependencies, and extensions—on demand. Copilot integrates natively and seamlessly into GitHub Codespaces, making it ideal for developers who want to:

- Start coding instantly without local setup.
- Work from anywhere with consistent environments.
- Get real-time Copilot suggestions with minimal latency.

Use case example

A student working on a group project can open Codespaces directly from the GitHub repo, write a comment like:

```
// function to calculate prime numbers in JavaScript
```

and get an instant, intelligent suggestion from Copilot without any local setup. All suggestions are served from GitHub's cloud infrastructure.



TIP GitHub Codespaces ensures that Copilot remains active as long as internet access is available, since the environment is inherently online.

How Copilot works without an internet connection (limitations)

GitHub Copilot requires an active internet connection for generating suggestions. This is because Copilot doesn't run its model locally; it sends your prompt (e.g., comments or code context) to the GitHub Copilot servers, which return relevant completions. However, it is available in offline mode, which can be useful if you temporarily lose connectivity (such as on a plane or in a restricted network environment) and still need to keep working with your existing code.

These are the key limitations in offline mode:

- No suggestions: If your machine goes offline, Copilot cannot provide new completions or suggestions. The Copilot icon typically shows a warning or "offline" message in such cases.
- No updates or learning: Copilot cannot fetch new context or respond to changes in real time.

- **Cached data is limited:** Even if you had previous suggestions, they are not stored locally in a meaningful way for reuse.
- **Feature access is blocked:** Features like Labs, feedback submission, and telemetry are inaccessible offline.

Use case caution

If you're coding on a plane or in a remote area with no internet, Copilot won't function. Developers in such scenarios should prepare code templates, snippets, or reference materials in advance, or they can temporarily rely on traditional IDE features like IntelliSense.

GitHub Copilot mode comparison

The following table compares GitHub Copilot's availability and performance across different development environments, both online and offline.

Mode	Copilot availability	Performance	Ideal for
GitHub Codespaces (online)	Full support	Fast, cloud based	Quick setup, remote collaboration, cloud-first teams
Local IDE (online)	Full support	Slight latency	Personal projects, enterprise workstations
Offline (local)	Not available	No suggestions	Not recommended for Copilot usage



IMPORTANT If you're concerned about data privacy when working online, you can configure Copilot's telemetry and data-sharing settings under your IDE's GitHub Copilot preferences.



SEE ALSO If you're curious about how Copilot compares across real-world projects and environments, check out Chapter 8, which offers in-depth use cases and developer workflows.

Skills review

In this chapter, you learned how to:

- Set up a GitHub account and choose the appropriate Copilot subscription plan.
- Understand the differences between the Free, Pro, and Pro+ tiers of GitHub Copilot.
- Install GitHub Copilot in supported IDEs, including Visual Studio Code, JetBrains IDEs, and GitHub Codespaces.
- Configure Copilot settings to match your development style and file preferences.
- Use keyboard shortcuts and interface features to control Copilot's suggestions and behavior.
- Test your setup by writing prompts in different languages, like Python, JavaScript, SQL, and HTML.
- Troubleshoot common installation and activation issues with GitHub Copilot.
- Apply best practices for prompting Copilot effectively and securely.
- Understand how Copilot functions differently in online and offline environments, including GitHub Codespaces integration.

Practice tasks

No practice files are necessary to complete the practice tasks in this chapter. Follow the steps below to reinforce the concepts and tools introduced in each section.



Set up your GitHub account and select a Copilot plan

Create your GitHub account and explore available Copilot plans:

1. Open your browser and go to <https://github.com>.
2. Select Sign Up and enter your email, password, and username.
3. Complete the CAPTCHA and email verification steps.
4. Navigate to <https://github.com/features/copilot>.
5. Compare the Free, Pro, and Pro+ subscription options.
6. Select Start Free Trial if applicable or subscribe to the plan that suits your needs.

Install Copilot in your development environment

Set up GitHub Copilot in your preferred IDE:

1. Open <https://code.visualstudio.com> and install Visual Studio Code.
2. Launch VS Code and open the Extensions panel (Ctrl+Shift+X).
3. Search for GitHub Copilot and then click Install.
4. Sign in with your GitHub account when prompted.
5. Verify that the Copilot icon appears in the bottom-right corner of the IDE.
6. Repeat steps 2–5 in another supported IDE, like JetBrains or Codespaces, if desired.

Configure permissions and personal settings

Customize Copilot behavior in VS Code:

1. In VS Code, go to File > Preferences > Settings.
2. Search for Copilot to view all available settings.

3. Adjust the suggestion frequency, panel view, and language preferences.
4. Open Keyboard Shortcuts and test:
 - a. Alt+/ to trigger a suggestion.
 - b. Tab to accept a suggestion.
 - c. Ctrl+Enter for multiple options.
5. Explore the Copilot sidebar and Labs features (if available).

Test Copilot with sample prompts across languages

Try prompts in Python, JavaScript, SQL, and HTML:

2. Create the following new files in VS Code:
 - a. test.py
 - b. script.js
 - c. query.sql
 - d. index.html
3. In each file, write a comment that describes a function or layout:
 - a. In test.py, try this prompt:

```
# function to calculate factorial
```

- b. In script.js, try this prompt:

```
// check if number is prime
```

- c. In query.sql, try this prompt:

```
-- select top 5 salaries
```

- d. In index.html, add this prompt:

```
<!-- responsive navbar using Bootstrap -->
```

4. Observe Copilot's suggestions and accept or modify them.

Troubleshoot installation issues and common errors

Simulate and resolve common Copilot setup issues:

1. Disconnect from the internet and confirm that Copilot goes offline.
2. Reconnect and restart VS Code.
3. Open the Command Palette (Ctrl+Shift+P) and run GitHub: Sign Out and then GitHub: Sign In.
4. Try triggering suggestions in an unsupported file type (e.g., a .txt file) to observe limitations.
5. Visit your GitHub Copilot dashboard to confirm your subscription status.

Apply best practices for using Copilot effectively

Learn to prompt Copilot for optimal code suggestions:

1. In a Python file, write vague and specific comments to compare results:

```
# function
```

vs.

```
# function to reverse a list
```

2. Try adding a step-by-step comment block:

```
# Step 1: Filter even numbers
```

```
# Step 2: Square them
```

3. Accept and test suggestions, modifying variable names and logic.
4. Create a docstring-based prompt and observe Copilot's understanding.
5. Write insecure code and observe whether Copilot flags or reinforces bad practices.

Explore Copilot's functionality in online and offline modes

Compare Copilot behavior in cloud vs. local setups:

1. Open GitHub Codespaces and confirm that Copilot provides suggestions immediately as you start typing code.
2. Use inline comments in Codespaces to test real-time completions.
3. Switch to local VS Code and simulate working offline.
4. Try using Copilot Chat (if on the Pro plan) and note what the chat experience is like.
5. Review telemetry settings in your GitHub Copilot preferences.
6. Reflect on how cloud-first environments differ from local development.

Index

Symbols and Numerics

""" (triple quotes), writing a docstring, 73
5 Whys technique, 142

A

accountability, developer, 24–25
advanced prompting, 96
 add examples inside comments, 98–100
 automation, 104–105
 clean missing values, 108–109
 create class templates quickly, 107
 create new columns from existing data, 109
 exception handling, 110
 form field validation, 108
 generating a flask route, 108
 looping, 111
 repetitive structures, 106
 reusing comments, 106
 SQL query, 109
 use partial code as a trigger, 105–106
break large tasks into smaller comments, 100–101
setting the tone, 102–103
specify the format you want, 101–102
use “what” and “how” together, 96–98

AI

agents, 306
-assisted development cycle
 code stage, 240
 deploy stage, 242
 documentation, 241
 plan stage, 239–240
 test stage, 240–241
emerging trends
 cross-modal development, 307
 holographic assistants, 306–307
 real-time code optimization, 308
 voice-first coding environments, 304–305
machine learning, 8
natural language programming, 6–7, 304
-powered coding, 305–306
responsible, 5, 23

algorithm prompt template, 78
analytics. *See* data science and analytics, use cases
Apache License 2.0, 24
AR (augmented reality), 306–307
authentication, troubleshooting, 54
autocomplete, 9
automation, 104–105. *See also* DevOps and automation, use cases
 clean missing values, 108–109
 creating class templates, 107
 creating new columns from existing data, 109
 defining a class with attributes, 111
 exception handling, 110
 form field validation, 108
 generating a flask route, 108
 generating docstrings, 203–204
 generating unit test for calculator function, 110
 looping, 111
 repetitive structures, 106
 reusing comments, 106
 SQL query, 109
 test creation, 169, 174–175
 Copilot-suggested code, 169–170
 copy and paste refactoring in a prompt, 173–174
 extending existing test files, 172
 table-driven tests, 170–172
 using comments to describe multiple scenarios, 169
 use partial code as a trigger, 105–106
Azure Data Studio, installing Copilot, 43–44

B

backend development, use cases, 227
 calculate user discounts, 260–261
 contact form handler, 259–260
 creating RESTful endpoints, 228
 handling database queries, 228–229
 implementing middleware, 229–230
best practices

best practices

- bug fixes, 147
 - code review, 216–217
 - security and compliance checks, 59
 - setting realistic expectations as you onboard, 58–59
 - testing, 187
 - using comments effectively, 57–58
 - when to accept, modify, or reject suggestions, 56–57
 - workflow, 245–247
- BSD License, 24
- bug fixes, 139, 145
- best practices
 - applying fixes incrementally, 147
 - context-rich prompts, 147
 - cross-reference with logs or tests, 147
 - documenting fixes as you go, 147
 - checklist, 146
 - Copilot's role in, 145–146
 - prompt writing, 147–148
 - test failures, 148–149
- built-in functions, 155

C

- chat, 50
- checklist
- code review, 196–197
 - debugging, 142
 - error handling, 151
 - refactoring, 202–203
 - reviewing code for quality, 132–133
- CI/CD
- Copilot use cases, 320
 - workflow, creating, 292–294
- class
- defining with attributes, 111
 - templates, 107
- clean coding practices
- break long logic into helper functions, 121
 - common mistakes Copilot helps avoid, 121–123
 - improving function clarity, 121
 - toolkit, 123–124
- CLEAR framework, 118–119. *See also* clean coding practices
- VS Code for Web, installing Copilot, 42
- code stage, AI-assisted development lifecycle, 240
- code/coding. *See also* clean coding practices
- AI-powered, 305–306
 - bug fixes, 145
 - checklist, 146
 - context-rich prompts, 147
 - Copilot's role in, 145–146
 - cross-reference with logs or tests, 147
 - documenting, 147
 - incremental, 147
 - prompt writing, 147–148
 - clean, 118
 - break long logic into helper functions, 121
 - common mistakes Copilot helps avoid, 121–123
 - improving function clarity, 121
 - CLEAR framework, 118–119
 - context, 145
 - debugging, 139, 140–141. *See also* debugging; troubleshooting
 - 5 Whys technique, 142–144
 - best practices, 144–145
 - checklist, 142
 - detecting issues in real time, 144
 - error handling, 150
 - checklist, 151
 - enhance validation logic and input checking, 152
 - handling edge cases and providing fallback logic, 152–153
 - try-except/try-catch blocks, 151
 - generation, 9–10
 - HTML, generating, 224–225
 - legacy, refactoring, 14–15
 - messy logic, cleaning up, 288–290
 - optimizing for performance, 116–117
 - reducing memory usage, 117–118
 - using built-in functions and language idioms, 155–157
 - using Copilot-driven insights, 153–154
 - using sets for faster lookups, 117
 - readability, 124–128
 - real-time optimization, 308
 - redundant loop operation, 154
 - refactoring
 - copy and paste method, 173–174
 - legacy code, 14–15, 112–116
 - prompt template, 79–80
 - prompts, 197–198
 - renaming variables and functions, 199–200

- simplifying conditionals and removing redundancy, 199–200
 - reviewing, 128, 192, 214, 263–264, 314
 - alternative implementations, 208–209, 212
 - ask Copilot to improve its own output, 131–132
 - balance AI assistance with human judgment, 196
 - best practices, 216–217
 - check logic against sample inputs, 129
 - checklist, 132–133, 196–197
 - common Q&As, 218
 - Copilot code review vs. manual, 192
 - detecting outdated or deprecated patterns, 194–195
 - highlighting potential bugs and code smells, 195
 - large datasets, 129–130
 - line-by-line, 283
 - look for hidden assumptions, 130
 - prompts, 196
 - security gaps, 131
 - spotting inefficiencies, 192–194
 - using linters and formatters, 283
 - validate with tests, 130–131
 - security, 291
 - test failures, 148–149. *See also* tests/testing
 - vibe, 321–322
 - voice-first, 304–305
 - Codex, 1, 2, 6, 8
 - collaboration, 212–213, 214, 215, 263
 - facilitating, 319
 - pair programming, 214
 - pull requests, 213
 - sharing AI-generated snippets in chat or documentation, 213
 - team, 312–313
 - using Copilot feedback, 217
 - using Visual Studio Code, 215–216
 - columns, creating from existing data, 109
 - comment/s, 3, 89–90, 296–297. *See also*
 - prompts/prompt writing
 - based prompts, 72, 282–283
 - best practices, 19–20
 - to-code, 11
 - describing multiple test scenarios, 169
 - explaining complex logic, 204–205
 - natural language, 70, 284–285
 - reuse, 106
 - updating, 205
 - using examples or constraints, 98–100, 285–286
 - writing a function using, 51
 - contact form handler, building, 259–260
 - context
 - code, 145
 - rich prompts, 147, 271–272, 275–278
 - Copilot Chat, 309–310, 311
 - copy and paste refactoring in a prompt, 173–174
 - core capabilities, Copilot, 12–13
 - autocomplete, 9
 - code generation, 9–10
 - comment-to-code, 11
 - function completion, 11–12
 - writing tests or documentation, 12
 - creating a GitHub account, 34–35
 - creativity, boosting, 19
 - cross-modal development, 307
- ## D
- data science and analytics, use cases
 - building machine learning models, 232–233
 - data cleaning and transformation, 230–231, 260
 - generating visualizations, 231–232
 - Jupyter notebook support, 233–234
 - database queries, handling, 228–229
 - debugging, 139, 140–141
 - 5 Whys technique, 142–144
 - applying bug fixes, 145
 - checklist, 146
 - context-rich prompts, 147
 - Copilot’s role in, 145–146
 - cross-reference with logs or tests, 147
 - documenting, 147
 - incremental, 147
 - prompt writing, 147–148
 - best practices, 144–145
 - checklist, 142
 - detecting issues in real time, 144
 - suggesting fixes while, 238–239
 - decision tree, creating, 232–233
 - deploy stage, AI-assisted software development lifecycle, 242
 - developer, 90. *See also* software development
 - accountability, 24–25
 - AI-augmented roles, 317–318, 319

- AI supervisor, 318
- collaboration facilitators, 319
- prompt engineer, 318–319
- changing roles, 264
- continuous learning, 320
- productivity, 17–18, 263
- DevOps and automation, use cases, 320
 - Bash script for file cleanup, 261
 - creating GitHub Actions workflows, 235
 - IaC (infrastructure as code), 236
 - scripting with Bash or PowerShell, 235–236
 - writing Dockerfiles, 234
- Docker files, writing, 15–16, 234
- docstrings, 73
 - descriptive, 286–287
 - generating automatically, 203–204
 - generic, avoiding, 206
- documentation. *See also* comment/s
 - AI-generated snippets, 213
 - educational, 256
 - inline, 203, 206–207, 311
 - prompt template, 80–81
 - regulatory compliance, 250–251
 - software development lifecycle, 241
 - writing, 12

E

- Eclipse, 20, 44
- edge cases, testing, 173, 184, 272–273
- education, Copilot use cases, 254
 - build educational platforms, 256
 - generate documentation and guides, 256
 - generate sample code and assignments, 254–255
 - help students practice programming logic, 255
- effective prompts, 68, 71
- error handling, 139, 150, 291
 - checklist, 151
 - enhance validation logic and input checking, 152
 - handling edge cases and providing fallback logic, 152–153
 - try-except/try-catch blocks, 151
- ethical considerations
 - developer accountability, 24–25
 - source code licensing, 24
- exception handling, 110

F

- feedback, 217. *See also* suggestions
 - AI-enhanced, 212–213
 - prompt writing, 90
- finance and fintech, Copilot use cases, 252
 - build risk models and scoring systems, 253–254
 - build secure APIs for account transaction and management, 252
 - scaffold compliance-related logic and audit logs, 253
 - write SQL queries for fraud detection and financial reporting, 252–253
- formatters, 283
- framework/s
 - CLEAR, 118–119
 - following the patterns of, 83
 - specific prompts, 82, 178
 - testing, 175
 - pytest, 176–177
 - unittest, 175–176
 - when to use unittest vs. pytest, 177
- Free Copilot plan, 38
- f-string, 194–195
- function/s
 - alternative implementation, 210
 - built-in, 155
 - completion, 11–12
 - generating tests for, 14
 - helper, 200–202, 290
 - improving clarity, 121
 - naming, 286–287
 - performance improvements, 209–210
 - prompt writing, 83
 - renaming, 199–200
 - signature prompts, 72
 - writing, 13–14, 51, 74–75

G

- GitHub account, setting up, 34–35
- GitHub Codespaces, 20, 59–60
- GitHub Copilot, 1, 2–3
 - autocomplete, 9, 21–22
 - benefits
 - enhances creativity, 26
 - learning support, 27
 - reduces context switching, 26

- speeds up development, 26
 - best practices
 - security and compliance checks, 59
 - setting realistic expectations as you onboard, 58–59
 - using comments effectively, 57–58
 - when to accept, modify, or reject suggestions, 56–57
 - boosting developer productivity, 17–18
 - bug fixes, 145–146
 - checklist, 146
 - context-rich prompts, 147
 - cross-reference with logs or tests, 147
 - documenting, 147
 - incremental, 147
 - capabilities, 310
 - code generation, 9–10
 - in Codespaces, 5
 - comment-to-code, 11
 - comparing with IntelliSense, 7
 - continuous improvement, 8
 - core capabilities, 9–13
 - debugging code issues. *See* debugging
 - diagnosing a test failure, 148–149
 - early versions, 309
 - error handling. *See* error handling
 - ethical considerations, 23
 - developer accountability, 24–25
 - source code licensing, 24
 - experimental features, 316
 - function completion, 11–12
 - home page, 2
 - IDE support, 3–5, 20–21, 39, 312.
 - See also* IDEs
 - installation issues, troubleshooting
 - authentication or sign-in issues, 54
 - Copilot Chat or Labs feature not available, 55
 - Copilot extension not appearing in the IDE, 53–54
 - extension installation fails or is blocked, 55
 - slow or laggy suggestions, 55–56
 - suggestions not appearing, 54
 - integration into CI/CD and DevOps pipelines, 315–316
 - keyboard shortcuts, 45–46
 - language and framework support, 21–22, 168
 - as learning aid, 18
 - limitations
 - accuracy, 27
 - lacks business context, 28
 - requires human judgment, 28
 - sometimes suggests insecure or outdated code, 28
 - multi-language support, 16–17
 - natural language understanding, 6–7
 - offline mode, 60–61
 - online and offline modes, 59
 - overusing, 271
 - permissions and personal settings, 44–45
 - plans
 - comparing, 36–37
 - Free, 38
 - free access to GitHub Copilot Pro, 37–38
 - responsible AI principles, 5, 23
 - suggestions. *See* suggestions
 - for teams and enterprises, 312–313
 - testing with sample prompts across languages, 50
 - create a file for each language, 50–51
 - testing suggestions in multiple languages, 51–53
 - write a sample function with a comment, 51
 - training, 3
 - usage guidelines and best practices, 25
 - use cases
 - creative code brainstorming, 19
 - generating tests for a function, 14
 - refactoring legacy code, 14–15
 - writing a function from a comment prompt, 13–14
 - writing SQL queries, Docker files, or YAML configs, 15–16
 - writing tests or documentation, 12, 168
 - comments, 169
 - integration tests, 166–168
 - prompts, 169
 - unit tests, 164–166
 - GPL (GNU General Public License), 24
- ## H
- healthcare, Copilot use cases
 - build and test APIs, 250
 - cleaning and analyzing datasets, 249
 - generate documentation for regulatory compliance, 250–251

- scaffold notebooks for diagnostics and prediction, 251
- helper functions, 200–202, 290
- holographic pair programmers, 306–307
- home page, GitHub Copilot, 206
- HTML, generating code, 224–225

I

- laC (infrastructure as code) files, 236
- IDEs, 3–4, 20–21, 39. *See also* installing Copilot in your IDE
- installing Copilot in
 - Azure Data Studio, 43–44
 - VS Code for Web, 42
 - Eclipse, 44
 - GitHub Codespaces, 39–40
 - JetBrains, 43
 - Neovim, 43
 - Visual Studio (2022 and later), 42
 - Visual Studio Code, 40–41
- idiomatic Python, 210–211
- incremental bug fixes, 147
- indexing, suggestions, 50
- inefficient prompts, 85–86
- initialization code, including in a prompt, 83–84
- inline documentation, 203, 206–207, 311
- input validation, 152
- installation, troubleshooting, 53
 - authentication or sign-in issues, 54
 - Copilot Chat or Labs feature not available, 55
 - Copilot extension not appearing in the IDE, 53–54
 - extension installation fails or is blocked, 55
 - slow or laggy suggestions, 55–56
 - suggestions not appearing, 54
- installing Copilot in your IDE
 - Azure Data Studio, 43–44
 - VS Code for Web, 42
 - Eclipse, 44
 - GitHub Codespaces, 39–40
 - JetBrains, 43
 - Neovim, 43
 - Visual Studio (2022 and later), 42
 - Visual Studio Code, 40–41
- integration tests
 - generating, 237–238
 - writing, 166–167
- IntelliSense, comparing with GitHub Copilot, 7

J-K

- JetBrains, 4, 20, 43
- Jupyter notebook support, 233–234, 251
- keyboard shortcuts, 45–46

L

- language functions, 155
- learning aid, Copilot as, 18
- legacy code, refactoring, 14–15, 112–116
- licenses, 24
- linters, 283
- LLMs (large language models), 1
- looping, 111

M

- machine learning, 8
- manufacturing and logistics, Copilot use cases, 257
 - estimate delivery windows and logistics delays, 258
 - generate dashboards and monitoring logic, 258–259
 - inventory tracking and stock validation, 257
 - process IoT sensor data, 258
- matplotlib graph, 84, 232
- mentorship, AI-assisted, 263
- middleware, implementing, 229–230
- mistakes commonly made using Copilot, 295–296
 - blindly accepting suggestions, 270–271, 275
 - failing to check for edge cases or validation, 272–273
 - inconsistent style, 273
 - overusing Copilot, 271
 - providing inadequate context, 271–272, 275–278
 - reasons for, 274–275
- MIT License, 24
- ML (machine learning) models, building, 232–233
- multi-language support, 16–17

N

- naming functions, 286–287
- natural language

- comment-to-code, 11
- prompts, 6–7
- Neovim, 4, 20, 43

O

- offline mode, 59, 60–61
- online mode, 59, 61
- OpenAI, Codex. *See* Codex
- overusing Copilot, 271

P

- pair programming
 - efficiency, 214
 - holographic, 306–307
 - mindset, 23
- parameterized tests, automating, 170–172
- performance
 - code, 116–117
 - built-in functions and recursive logic, 155–157
 - clean coding practices, 118
 - loops and recursive logic, 154
 - optimizing through Copilot-driven insights, 153–154
 - reducing memory usage, 117–118
 - using sets for faster lookups, 117
 - function, 209–210
- permissions, GitHub Copilot, 44–45
- plan stage, AI-assisted development lifecycle, 239–240
- plans, GitHub Copilot
 - comparing, 36–37
 - free access to GitHub Copilot Pro, 37–38
 - Pro+, 37
- productivity, developer, 17–18
- prompts/prompt writing, 68, 69, 81–82, 96
 - adding context, 76, 86–87
 - automation, 104–105
 - clean missing values, 108–109
 - create class templates quickly, 107
 - create new columns from existing data, 109
 - defining a class with attributes, 111
 - exception handling, 110
 - form field validation, 108
 - generate unit test for calculator function, 110
 - generating a flask route, 108
 - looping, 111
 - repetitive structures, 106
 - reusing comments, 106
 - SQL query, 109
 - use partial code as a trigger, 105–106
 - break complex tasks into smaller parts, 76, 88–89, 100–101, 279–280
 - clarity checklist, 87
 - code readability, 124–128
 - code review, 196
 - collecting, 262
 - comment-based, 72, 282–283
 - adding examples, 98–100
 - natural language, 70
 - context-rich, 147, 271–272, 275–278
 - copy and paste refactoring, 173–174
 - describing a troubleshooting issue, 149
 - docstring, 73
 - effective, 68, 71
 - experiment and compare, 90
 - extending existing test files, 172
 - fixing a bug, 147–148
 - framework
 - following the patterns of, 83
 - specifying the name of, 82
 - function signature, 72
 - inefficient, 85–86
 - initialization code, 83–84
 - input and output, 70
 - integration test, 166
 - keep improving through feedback, 90
 - maintaining a prompt log, 90
 - natural language, 6–7
 - optimizing code for performance, 116–117
 - clean coding practices, 118
 - reducing memory usage, 117–118
 - using sets for faster lookups, 117
 - refactoring legacy code, 112–116
 - reference common functions or idioms, 83
 - refining, 74–75, 89, 279
 - renaming variables, 199–200, 280–282
 - setting the tone, 102–103
 - specify language or tools, 70
 - specify the format you want, 101–102
 - state the goal clearly, 69
 - style comparison, 73–74
 - styles, 103
 - template, 77, 81
 - algorithm, 78

- documentation, 80–81
- refactoring, 79–80
- unit test, 78–79
- test, 178–179
 - combining comments and code for better context, 180–181
 - framework-specific, 178, 180
 - ineffective, 181–182
 - steering the output, 181
 - structure and language, 179–180, 182, 184–185
 - using fixtures and setup blocks, 185
- toolkit, 123–124
- unit test, 165
- use “what” and “how” together, 96–98
- vague, 68–69, 84–85, 86
- weak vs. refined, 76–77
- Python
 - idiomatic, 210–211
 - prompt writing
 - mention input and output, 70
 - specify language or tools, 70
 - state the goal clearly, 69
 - use natural, complete sentences in comments, 70
 - pytest, 176–177, 185
 - unit tests, 169–170
 - unittest, 175–176, 185

Q-R

- readability, code, 124–128
- real-time code optimization, 308
- recursive logic, optimizing, 156–157
- refactoring. *See also* debugging; performance checklist, 202–203
 - copy and paste method, 173–174
 - extracting helper functions, 200–202
 - generating suggestions, 202
 - legacy code, 14–15, 112–116
 - prompt template, 79–80
 - prompts, 197–198
 - renaming variables and functions, 199–200
 - simplifying conditionals and removing redundancy, 199–200
- refining prompts, 74–75, 89
 - adding context, 76
 - break complex tasks into smaller parts, 76, 279–280

- regulatory compliance
 - finance and fintech, 253
 - healthcare industry, 250–251
- renaming variables, 199–200, 280–282
- responsible AI, 5, 23
- responsive components, CSS, 225–226
- RESTful endpoints, creating, 228
- reviewing code, 128, 192, 263–264
 - alternative implementations, 208–209, 212
 - ask Copilot to improve its own output, 131–132
 - balance AI assistance with human judgment, 196
 - best practices, 216–217
 - checking logic against sample inputs, 129
 - checklist, 132–133, 196–197
 - common Q&As, 218
 - Copilot code review vs. manual, 192
 - detecting outdated or deprecated patterns, 194–195
 - highlighting potential bugs and code smells, 195
 - large datasets, 129–130
 - line-by-line, 283
 - looking for hidden assumptions, 130
 - prompts, 196
 - security gaps, 131
 - spotting inefficiencies, 192–194
 - using linters and formatters, 283
 - validate with tests, 130–131
- root cause analysis, 5 Whys technique, 142–144

S

- scikit-learn, creating a decision tree using, 232–233
- scripting, Bash and PowerShell, 235–236
- security, 59, 131
 - code, 291
 - test, 186
- setting up your GitHub account, 34–35
- software development, 307
 - in the AI era, 316–317
 - cross-modal, 307
 - GitHub Copilot workflow tips, 242–243
 - lifecycle, AI-assisted
 - code stage, 240
 - deploy stage, 242
 - documentation, 241

- plan stage, 239–240
 - test stage, 240–241
- sorting function, writing, 74–75
- source code licensing, 24
- spatial computing, 306–307
- SQL query
 - automation, 109
 - generating, 228–229
 - retrieving top 10 customers, 292
 - writing, 15–16
- students, free access to GitHub Copilot Pro, 37–38
- styles
 - inconsistent, 273
 - prompting, 103
- suggestions, 22. *See also* prompts/prompt writing
 - accepting, modifying, or rejecting, 56–57, 287–288
 - applying, 150
 - blindly accepting, 270–271, 275
 - debugging, 144
 - evaluating for quality, 128
 - ask Copilot to improve its own output, 131–132
 - check logic against sample inputs, 129
 - checklist, 132–133
 - large datasets, 129–130
 - look for hidden assumptions, 130
 - security gaps, 131
 - validate with tests, 130–131
 - generating for multiple programming languages, 50
 - create a file for each language, 50–51
 - write a sample function with a comment, 51
 - generating negative and edge case tests, 173
 - indexing, 50
 - integration test, 167
 - Next Edit, 47–48
 - pull request, 213
 - redundant loop operation, 154
 - refactoring, 197–198, 202
 - reviewing, 150, 183
 - tests, 51–53
 - extending existing test files, 172
 - unit, 165–166
 - triggering manually, 45
 - troubleshooting, 54
 - unit test, 169–170

T

- table-driven tests, automating, 170–172
- tags, HTML, generating, 224–225
- Tailwind CSS, creating responsive components, 225–226
- teachers, free access to GitHub Copilot Pro, 37–38
- template/s
 - class, 107
 - prompt, 77, 81
 - algorithm, 78
 - documentation, 80–81
 - refactoring, 79–80
 - unit test, 78–79
 - for specific libraries, 84
- test stage, AI-assisted development lifecycle, 240–241
- tests/testing, 164
 - automation, 169, 174–175
 - Copilot-suggested code, 169–170
 - copy and paste refactoring in a prompt, 173–174
 - table-driven tests, 170–172
 - using comments to describe multiple scenarios, 169
 - best practices, 187
 - documenting assumptions in comments or docstrings, 186
 - edge cases, 173, 184
 - extending existing test files, 172
 - failures, using Copilot with, 148–149
 - framework/s, 175
 - pytest, 176–177
 - specific prompts, 178
 - unittest, 175–176
 - when to use unittest vs. pytest, 177
 - generating for a function, 14
 - generating multiple test cases, 169
 - integration, 166–167, 237–238
 - isolating, 185–186
 - language and framework support, 168
 - prompts, 178–179
 - combining comments and code for better context, 180–181
 - framework-specific, 178, 180
 - ineffective, 181–182
 - steering the output, 181

- structure and language, 179–180, 182, 184–185
- using fixtures and setup blocks, 185
- security, 186
- suggestions, 51–53, 183
- unit, 12, 164–166, 237, 261
- validating against real use cases, 183–184
- toolkit, prompt writing, 123–124
- training, GitHub Copilot, 3
- troubleshooting, 139. *See also* debugging; error handling; mistakes commonly made using Copilot
 - applying Copilot’s suggestions, 150
 - describe the issue with a prompt, 149
 - identify the problem, 149
 - installation issues, 53
 - authentication or sign-in issues, 54
 - Copilot Chat or Labs feature not available, 55
 - Copilot extension not appearing in the IDE, 53–54
 - extension installation fails or is blocked, 55
 - slow or laggy suggestions, 55–56
 - suggestions not appearing, 54
 - review Copilot’s suggestion, 150
 - validation, 150
- try-except/try-catch blocks, 151

U

- unit test
 - generating, 110, 237
 - prompt template, 78–79
 - writing, 12, 164–166
- unittest, 175–176, 185
- updating, comments, 205
- use cases, Copilot
 - backend development, 227
 - calculate user discounts, 260–261
 - contact form handler, 259–260
 - creating RESTful endpoints, 228
 - handling database queries, 228–229
 - implementing middleware, 229–230
 - creative code brainstorming, 19
 - data science and analytics
 - building machine learning models, 232–233
 - data cleaning and transformation, 230–231, 260
 - generating visualizations, 231–232
 - Jupyter notebook support, 233–234
 - DevOps and automation, 320
 - Bash script for file cleanup, 261
 - creating GitHub Actions workflows, 235
 - laC (infrastructure as code), 236
 - scripting with Bash or PowerShell, 235–236
 - writing Dockerfiles, 234
 - in education, 254
 - build educational platforms, 256
 - generate documentation and guides, 256
 - generate sample code and assignments, 254–255
 - help students practice programming logic, 255
 - in finance and fintech, 252
 - build risk models and scoring systems, 253–254
 - build secure APIs for account transaction and management, 252
 - scaffold compliance-related logic and audit logs, 253
 - write SQL queries for fraud detection and financial reporting, 252–253
 - generating tests for a function, 14
 - in health and life sciences
 - build and test APIs, 250
 - cleaning and analyzing datasets, 249
 - generate documentation for regulatory compliance, 250–251
 - scaffold notebooks for diagnostics and prediction, 251
 - in manufacturing and logistics, 257
 - estimate delivery windows and logistics delays, 258
 - generate dashboards and monitoring logic, 258–259
 - inventory tracking and stock validation, 257
 - process IoT sensor data, 258
 - refactoring legacy code, 14–15
 - testing and debugging
 - creating integration tests with mocks, 237–238
 - generating unit tests, 237
 - suggesting fixes while debugging, 238–239
 - writing unit tests, 261
 - web development, 224
 - CSS, responsive components, 225–226
 - HTML, generating, 224–225
 - JavaScript validation logic, 226–227

- writing a function from a comment prompt, 13–14
- writing SQL queries, Docker files, or YAML configs, 15–16

V

- vague prompts, 68–69, 84–85, 86
- variables, renaming, 199–200, 280–282
- vibe coding, 321–322
- Visual Studio Code, 3–4, 20
 - code generation, 9–10
 - collaborating with teammates, 215–216
 - Copilot status bar icon
 - Code Completions (All Files), 47
 - Code Completions (Python), 47
 - Locally Indexed (Workspace Index), 46–47
 - Next Edit suggestions, 47–48
 - IntelliSense, comparing with GitHub Copilot, 7
 - keyboard shortcuts, customizing, 46
 - testing Copilot with sample prompts across languages, 50
 - create a file for each language, 50–51
 - testing suggestions in multiple languages, 51–53
 - write a sample function with a comment, 51
 - top bar Copilot menu, 48
 - configure code completions, 49
 - Editor Inline Chat (Ctrl+I), 49
 - Open Chat (Ctrl+Alt+I), 48–49
 - Quick Chat (Ctrl+Shift+Alt+L), 49
 - Visual Studio, installing Copilot, 42
 - visualizations, generating, 231–232
 - voice-first coding, 304–305

W

- weak prompts, 76–77
- web development, use cases
 - CSS, responsive components, 225–226
 - generating HTML code, 224–225
 - JavaScript validation logic, 226–227
- workflows
 - best practices, 245–247
 - CI, 292–294
 - GitHub Actions, 235
 - manual vs. Copilot-assisted, 244–245
 - software development lifecycle, 242–243
 - code stage, 240
 - deploy stage, 242
 - plan stage, 239–240
 - test stage, 240–241
- writing, 172. *See also* prompts/prompt writing
 - Docker files, 234
 - functions, 74–75
 - helper functions, 290
 - SQL queries, Docker files, and YAML configs, 15–16
 - tests. *See also* tests/testing
 - integration, 166–167
 - unit, 164–166

X-Y-Z

Xcode, 20

YAML

- config, writing, 15–16
- creating a CI workflow, 292–294