

Copilot and ChatGPT for Java and C# Developers

Essential prompting skills for
generative AI-enhanced code

<https://github.com/andrewstellman/genai-training>

©2025 Stellman and Greene Consulting LLC, all rights reserved



Andrew Stellman

I've been training people on software development and other technical topics for over 20 years. *Head First C#*, one of the six books I've written for O'Reilly, is going into its 5th edition. I'm also a full-time software developer and team lead, and I'm passionate about all things code.



What's in this course

A really quick overview of what we're going to learn...
and then we'll jump right into it.

What is generative AI?

Generative AI refers to AI models that can generate text, images, or code, by learning patterns from *existing data*.

- You can use generative AI to create stories, articles, song lyrics, poems, and even code based on a prompt.
- In coding, generative AI can help write and refactor code, generate documentation, and even suggest improvements.

Generative AI tools like ChatGPT and GitHub Copilot are trained on billions of lines of Java and C# code, so they can tap a vast knowledge base to generate code in response to your prompts.

“Just enough to be dangerous...”

- **Understanding Prompt Engineering:** Learn the basics of crafting effective prompts to get the best responses from AI chatbots.
- **Leveraging AI for Coding:** Discover how to use AI tools to generate code, add comments, and assist with debugging.
- **Exploring AI Capabilities:** Explore how AI can understand and interact with your code, providing valuable feedback and suggestions.
- **Practical AI Applications:** Gain hands-on experience with exercises in generating classes, solving problems, and writing unit tests using AI.

This is not a course in advanced AI!

This Course Is For:

- New developers looking to integrate AI into their C# and Java development.
- Developers interested in learning about the fundamentals of prompt engineering so they can use generative AI tools for their coding work.
- Anyone curious about using AI tools like chatbots and Copilot for coding.

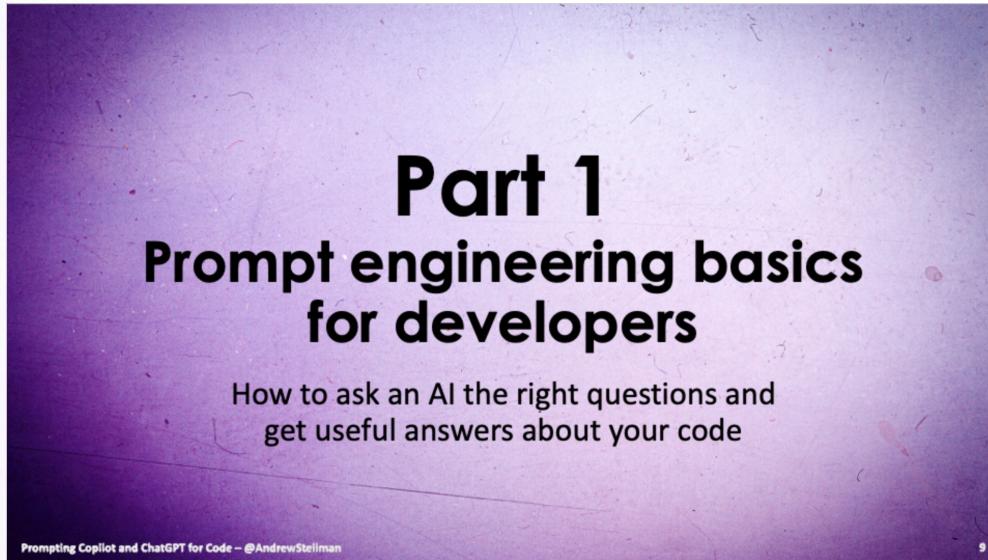
This Course Is Not For:

- Experts in AI or machine learning seeking advanced techniques.
- Those looking for in-depth theory on neural networks or deep learning.
- Developers seeking a comprehensive AI development course.

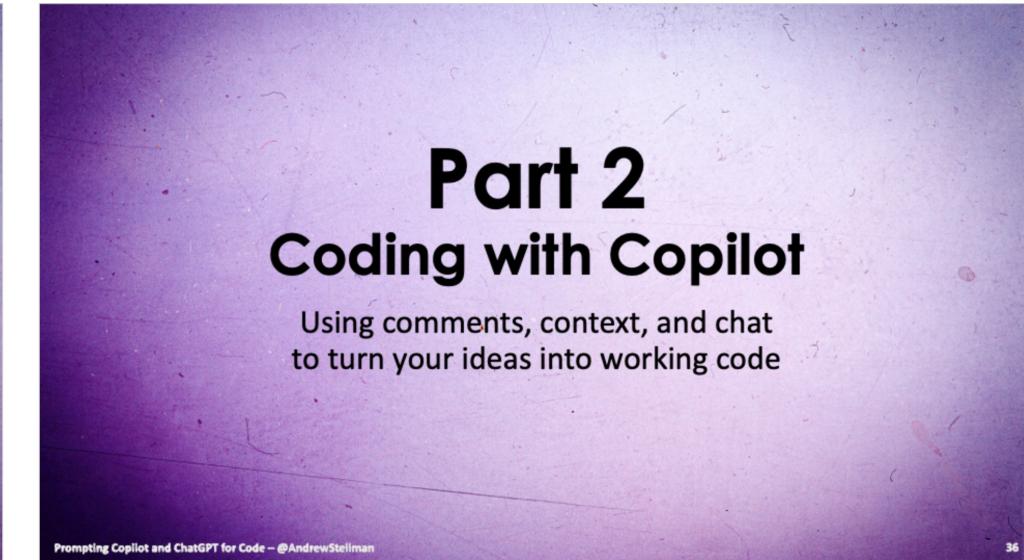
How this course is structured

- In the main **Presentation** sections, I'll do teaching and give demos using C# or Java. You're encouraged to follow along on your own.
- In the **Exercise** sections, I'll give you time to keep working on your own. You can download the code and slides from the GitHub page for this course: <https://bit.ly/coding-ai-course>
- This course is divided into two parts with a short break between them. Each part ends with a **Q&A** section where I answer your questions.

How this course is structured



We'll dive into the basics of using AI in coding, covering the basics of **prompt engineering**, and getting hands-on with an **exercise** to analyze classes and improve classes with AI.



We'll explore how AI can help you **write better code**. You'll learn how to steer GitHub Copilot using **context**, **comments**, and **partial code**, then practice with exercises that build full methods and small projects.

Part 1

Prompt engineering basics for developers

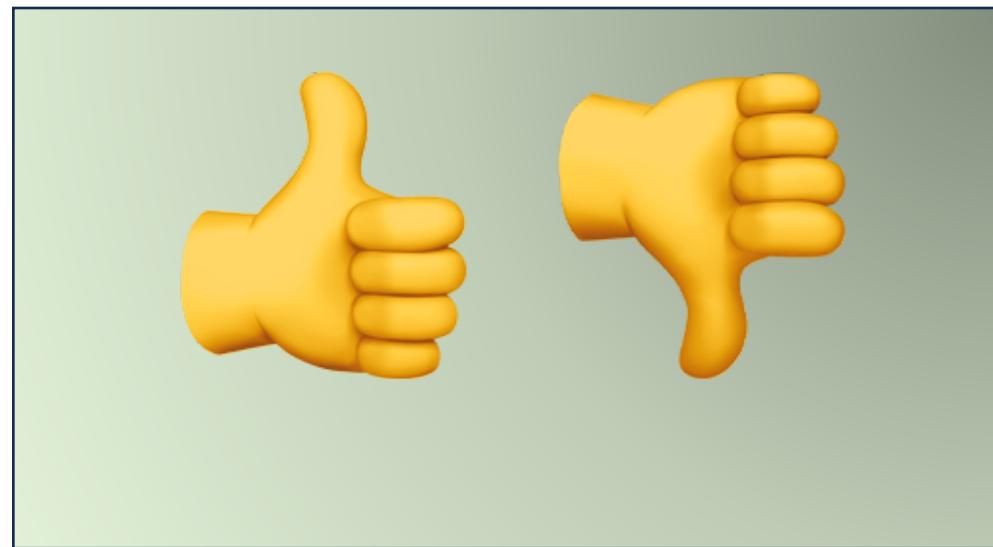
**How to ask an AI the right questions and
get useful answers about your code**

Introducing Generative AI

Unlocking the potential – and understanding the limitations – of generative AI

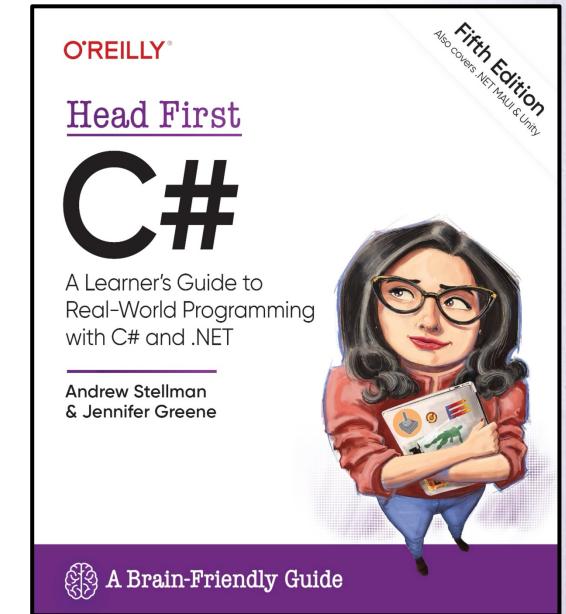
Pulse Check

Who here has used a Generative AI chatbot like ChatGPT, Copilot, Gemini, or Claude?



AI seems like magic (but it's not)

- When I was writing the latest edition of *Head First C#*, I used multiple AI chatbots to test the exercises.
- I pasted the instructions and code for the exercise into the chat as its **prompt**—or the instruction for an AI to generate specific output—and then copied the code back into my IDE to test it.
- If the AI chatbot was able to generate a correct answer that worked, it meant that I gave my readers enough information to complete the exercise.



Let's see a quick demo of what that looks like.

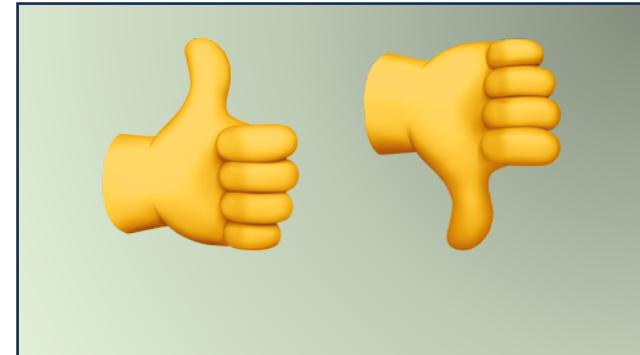
We'll also explore GitHub Copilot while we've got it open.

What happens if we ask a simple question about code?

- This is an exercise from Chapter 2 of *Head First C#*. People who just learned about loops usually get it right. How do you think an AI does?

```
int p = 2;
for (int q = 2; q < 32; q = q * 2) {
    while (p < q) {
        // How many times will
        // the next statement
        // be executed?
        p = p * 2;
    }
    q = p - q;
}
// How many times will the loop run?
```

Answer  if you think it gets the answer **right**



Answer  if you think it gets the answer **wrong**



andrewstellman

The code has comments that ask questions. Answer them.



GitHub Copilot

> Used 1 reference

Let's analyze the code step-by-step to answer the questions in the comments.

Code Analysis

```
int p = 2;
for (int q = 2; q < 32; q = q * 2) {
    while (p < q) {
        // How many times will
        // the next statement
        // be executed?
        p = p * 2;
    }
    q = p - q;
```

AI's will probably start getting this right when the new edition of *Head First C#* is added to their training data.

I entered the code into Visual Studio Code and used the GitHub Copilot extension to ask it to answer the questions in the comments. It gave the wrong answer. It always gives the wrong answer. I have yet to see an AI give the right answer to this question.

Answers to the Questions

1. How many times will the next statement (`p = p * 2`) be executed?

- The statement `p = p * 2` inside the `while` loop will be executed once for each iteration of the `for` loop where `q` is a power of 2 and `p < q`. This happens when `q` is 4, 8, 16, and 32. Therefore, it will be executed 4 times.

2. How many times will the loop run?

- The outer `for` loop will run until `q` is no longer less than 32. Given the doubling of `q` each iteration, the loop runs 5 times (for `q` values 2, 4, 8, 16, and 32).

Thus, the answers are:

- The statement `p = p * 2` will be executed 4 times.
- The outer `for` loop will run 5 times. *Not the right answer!*

GenAI isn't perfect (and that's okay!)

Start by understanding the capabilities—a limitations—of AI Chatbots.

- **Mimicking Patterns:** AI chatbots create responses by mimicking patterns from the vast amounts of text they've been trained on, aiming to sound like a natural conversation.
- **Code Generation:** They use patterns from many code examples to "auto-complete" or suggest code snippets.
- **Perceived Understanding:** It's easy to think that an AI understands the content it generates or is pulling correct answers from a database. In reality, it's crafting responses based on learned patterns, not actual understanding or knowledge.

Als don't inherently fact-check; they can produce convincing but sometimes incorrect or made-up answers. They do this a lot. A lot. ALL THE TIME.

An example that shows how AI works

A farmer needs to cross a river with a boat. He has a goat with him. The boat can only hold the farmer and one other item at a time. How does he get the goat to the other side of the river?



To get the goat across the river, the farmer can follow these steps:

1. **Take the goat across the river** and leave it on the other side.
2. **Return alone** to the original side.
3. **Take the boat back across the river** to the original side.
4. **Take the goat across the river again.**
5. **Leave the goat on the other side and return alone** to the original side.
6. **Finally, take the boat across the river** to the other side with the goat.

By repeating these steps, the farmer will successfully transport the goat across the river.



The current generation of AIs often generate a complicated answer to a question about getting a goat across a river, usually involving multiple unnecessary crossings.

The AI generates this weird answer because its training data included thousands of examples of the classic “River crossing puzzle” that involves getting a goat, cabbage, and a wolf across the river. It uses the solution to that puzzle as a pattern to generate its answer.

How to use AI safely: Trust but verify

AI doesn't give you the right answer. It generates an answer that sounds right.

- **Verify Information Accuracy:** Always double-check facts and information provided by AI, as it can sometimes be incorrect or incomplete.
- **Test Generated Code:** Before using AI-generated code, thoroughly review and test it to ensure it works as expected and meets your requirements.
- **Understand Limitations:** Recognize that AI chatbots can misunderstand prompts or lack context, leading to unexpected results.
- **Critical Examination:** Treat AI responses as a starting point for further research and validation, not as the final word.

Go to <https://bit.ly/coding-ai-course>
for the code in this exercise. It also has
a full PDF of this slide deck.

Exercise

This is your chance to try asking your favorite AI chatbot to work with some code.

Exercise: Use an AI chatbot to analyze Java or C# code (Part 1)

- Open an AI chatbot by going to <https://chat.openai.com/> (that's ChatGPT) or <https://claude.ai/> or <https://copilot.microsoft.com/> or <https://gemini.google.com/>
- Go to <https://bit.ly/coding-ai-course> and copy either the Java or C# version of the CustomSorter class. Paste it into the chat.
- Press shift-enter twice to add two lines to the end of the chat
- Finish the prompt by asking the AI to explain what this code does. Use a prompt like this (replace *[Java / C#]* with either Java or C#):
Can you explain what this *[Java / C#]* code does and how it works?

Ask it to suggest improvements (Part 2)

- Ask the AI to suggest a simple improvement to the code. You could use a prompt like this:

Can you suggest a simple improvement to make this code better?

Take a minute and read the response.

How accurate was the AI's explanation?

Did the AI catch the null check and empty string handling?

Was the improvement suggestion relevant and useful?

Als don't understand your code, but they sure look like they do

The AI may only be pattern matching, but it does a really good impression of understanding your code... and that's good enough for it to be *really* useful.

An AI chatbot can explain your code

AI chatbots might not truly "get" your code, but they can still provide useful insights that can help you.

- **Not true understanding:** AI chatbots don't "understand" because they don't have consciousness or context. They generate responses based on patterns in the data they've been trained on, not genuine comprehension (even if it feels like they're thinking!).
- **What they actually do:** AI chatbots analyze the structure and logic of your code, using patterns and rules to generate relevant comments that make the code easier to maintain.

Using AI in your coding workflow

Make the most of AI chatbots to simplify and improve your coding.

- **How you can use it:** You can ask AI to explain complex code sections, describe function purposes, or note TODOs or potential issues.
- **Why that's helpful:** The AI provides insightful comments that clarify the code's functionality, aiding both the original developer and others who might work on the code later.

Demo: AI-assisted code review

- I'm going to open code from *Head First C#* in Visual Studio Code and use **GitHub Copilot** to analyze it.
- GitHub Copilot isn't *exactly* like an AI chatbot. Instead, it's more like a coding assistant that lives in your IDE (like Visual Studio Code). It looks at the code you're working on and gives you helpful suggestions, snippets, and completions as you type. It uses the context of your project to predict what you might need next.
- You can also interact with it like an AI chatbot by using its chat panel. Your chat has access to all the code you're working on.
- I'll ask it to **explain** the code, add **comments**, and **refactor and improve** the code.

How AI Chatbots "Read" Code

Breaking down your code to spot patterns and potential issues.

- **Contextual Awareness:** AI chatbots analyze the structure and logic of your code to generate relevant comments, making the code easier to understand and maintain.
- **For example:** You can ask AI to explain complex code sections, describe function purposes, or note TODOs or potential issues.
- **Outcome:** The AI provides insightful comments that clarify the code's functionality, aiding both the original developer and others who might work on the code later.

The basics of prompt engineering

Crafting the right questions to get the best AI responses.

- **Prompt engineering** is the art and science of creating specific and clear inputs – *prompts* – to guide AI responses effectively.
- A well-crafted prompt can significantly improve the quality and relevance of the AI's output.
- AI chatbots rely on patterns from vast amounts of data, so making precise prompts helps them zero in on the data that best matches your problem and generate more accurate and useful responses.

Prompt engineering helps developers get better responses from AIs, making coding easier and faster.

Essential tips for crafting effective prompts

Clear and detailed prompts get you better results from your AI.

- **Be Specific and Clear:** Clearly state what you're looking for. Instead of asking "How do I use a for loop?" specify the language and context, like "How do I use a for loop in C# to iterate through an array?"
- **Provide Examples:** Give the AI a scenario or example to show the AI what you're working with. For example, "Given this code snippet [insert code], how can I optimize it for performance?"
- **Iterate, iterate, iterate:** Start with broader questions and then refine based on the AI's initial response. Keep refining your prompt until the AI gives you what you need.

Practice prompt engineering by asking an AI to refactor your code

Refactoring matters: It's so important for maintaining code quality and making sure your code is efficient, readable, and easy to change. AI is good at giving you suggestions that aren't immediately obvious. It can:

- **Identify Redundant Code:** Ask AI to review your code for redundancy:
“Do I have duplicated code in this file that can be moved to a method?”
- **Improve Readability:** Request AI to make code more readable:
“Refactor this function to make it more understandable and maintainable.”
- **Improve Performance:** Use AI to suggest performance optimizations:
“How can I optimize this method to reduce its execution time?”

Go to <https://bit.ly/coding-ai-course>
for the slides in this deck.

Exercise

Let's explore some of the more advanced analysis and refactoring capabilities of AI.

Exercise: Use an AI to analyze Java or C# code (Part 1)

- Open an AI chatbot by going to <https://chat.openai.com/> or <https://claude.ai/> or <https://copilot.microsoft.com/> or <https://gemini.google.com/>
- Go to <https://bit.ly/coding-ai-course> and copy the Java or C# code. (*Alternately, copy a class from your own code.*) Paste it into the chat.
- Press shift-enter twice to add two lines to the end of the chat
- Finish the prompt by asking the AI to explain what this code does and how it works and press enter. Here's a prompt you can use:
Here's a [Java/C#] implementation of a class. Can you explain what this code does and how it works?

Ask the AI to add comments and suggest improvements (Part 2)

- Ask the AI to give potential improvements or optimizations for the code. Here's an example prompt that you can use:

What potential improvements or optimizations could be made to this CustomSorter implementation?

- Request the AI to add comprehensive comments to the class and its methods. For Java, ask for JavaDoc comments. For C#, ask for XML documentation comments. Here's an example (replace *[JavaDoc / XMLDoc]* with JavaDoc or XMLDoc):

Add *[JavaDoc / XMLDoc]* comments to the code.

- Ask the AI to generate a simple main method (Java) or top-level statements (C#) that demonstrates how to use the class. You might give it this prompt:

Can you create *[a main method / top-level statements]* to demonstrate the usage of this code?

Refine and iterate (Part 3)

Let's get some practice using AI to analyze and explain complex code, so you can get some experience with prompt crafting.

- Iterate on the documentation request. If the AI's comments were too brief or missed important details, refine your prompt:

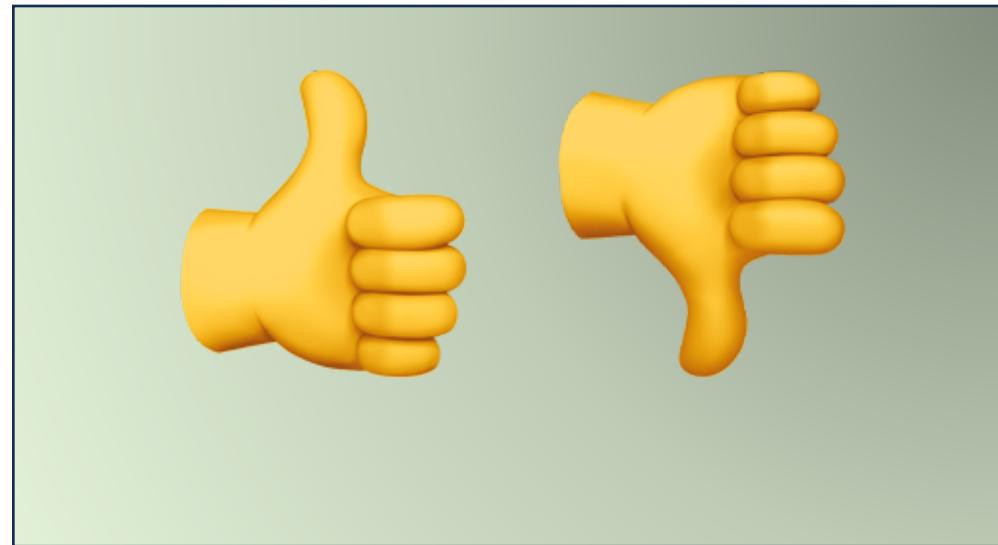
The comments you provided are a good start. Expand them to include a brief explanation in the class-level comment, parameter descriptions in all methods, and any preconditions or postconditions for each method.

- Refine your prompt for optimization. If the AI's initial response was vague or unsatisfactory, try a more specific prompt. Here's a starting point—try your own:

Suggest a specific optimization that would improve the time complexity of code. Please explain the optimization and show how to implement it in code.

Pulse Check

Were you able to get the AI to improve its output by asking it follow-up questions?



Q&A

If you have any questions, ask them now. Make sure you enter them in the Q&A widget, because that makes it easier for me to keep track of them.

5 minute break

Grab some water, check your email, have a look at your socials... we'll be back here in five minutes.

Part 2

Coding with Copilot

Using comments, context, and chat
to turn your ideas into working code

Welcome back

Think about the last time you looked at your own code and thought, “What was I doing here?!” Are there things you could have done to help future-you understand it faster?

Put your answers
in the chat 

Introducing GitHub Copilot

An AI that learns from your code and helps you build

Get to know GitHub Copilot

GitHub Copilot is an IDE tool that helps you write your code.

- GitHub Copilot is an AI-powered coding assistant that integrates directly into your IDE, providing **real-time code suggestions** as you type.
- It's really good at completing statements and methods, generating boilerplate code, and suggesting variable names, using machine learning trained on public code repositories.
- Beyond automatic suggestions, you can interact with Copilot by using **comments** to guide code generation, engaging in **chat** for explanations or debugging, and **prompting it for documentation or refactoring** to enhance your coding experience.

GitHub Copilot Chat Commands

Copilot Chat offers powerful commands to enhance your coding.

- You can use Copilot Chat like any generative AI. It also provides **shortcuts** for the most common things that you'd ask an AI about code, streamlining your workflow and boosting productivity.
- Key commands include **/tests** for generating unit tests, **/explain** for code explanations, and **/fix** to suggest code fixes.
- To use these commands, open Copilot Chat (usually with **Ctrl+I** or **⌘+I**), type the command, and optionally add specific instructions.

Use clear, descriptive language when using commands to get the most relevant and accurate responses from Copilot.

Behind the scenes of GitHub Copilot

- GitHub Copilot runs on **Codex**, a large language model derived from GPT models. Codex is fine-tuned on code repositories, making it specialized for coding tasks.
- Copilot uses your **code, comments, and file structure** as inputs instead of traditional natural language prompts. These inputs provide Codex with the context needed to generate relevant code suggestions.
- Codex processes this context to offer **real-time code suggestions** in your IDE. This method is more integrated and specific to coding than the prompt-response pattern used by chatbots like ChatGPT.
- **Copilot Chat** works like a traditional chatbot, engaging users with prompt-based interactions. It provides a conversational experience similar to models like ChatGPT, with shortcuts to help users easily generate prompts.

Refactoring with Copilot

Good things happen when you give Copilot context

Copilot is a *lot more* than just a “fancy” system for autocomplete suggestions

- Copilot uses AI to generate suggestions based on your actual code, not just predict the next token.
- It pays attention to method and class names, filenames, comments, and what's happening around your cursor.
- The more your code communicates intent, the more useful Copilot's suggestions become.

Think of Copilot as an AI coding partner—it's powerful, but it works best when your code gives it something meaningful to respond to.

How Copilot figures out what you and your code are trying to do

Copilot doesn't just guess. It looks at your code to *build context*. The more clues you give it, the better its suggestions become.

Here are a few things Copilot does to understand your code:

- Looks at the last 100 or so lines near your cursor (especially above it).
- Uses file names, method names, variable names, and comments to infer purpose.
- In Copilot Chat, it also sees the conversation and selected code block.
- Picks up on patterns across your project—naming conventions, function signatures, and code style.

Demo: Refactoring Code with Copilot

- We can rely on **the same skills we've used for prompt engineering** to help Copilot generate better, more meaningful code.
- I'll start with a short piece of code that's extremely hard to read—no clear names, no comments, no context.
- Then I'll use Copilot's **inline chat** to refactor the real-world system this code is from and provide documentation.
- Copilot needs context. Luckily, I found a document that will help it to suggest a much clearer, more readable refactoring.

Go to <https://bit.ly/coding-ai-course>
for the slides in this deck.

Exercise

Let's explore how Copilot responds when you give it just a little nudge—like a comment, a list, a bit of documentation, or a snippet of code.

Use comments to steer Copilot's suggestions

Copilot pays close attention to how you write your comments, especially when you start a list or documentation block. Even a few words can shape what it generates next.

- Start a comment like this: **// This method does the following:** and Copilot will suggest bullet points.
- Add Javadoc or XMLdoc comment and it often generates a full method based on it.
- Add code like this: **log.info(** and Copilot often fills in a useful log message based on what your code just did.

Here's a tip: Try phrasing your comment as if you're about to explain the next step—and let Copilot take a crack at it.

Tips for giving Copilot the right context

Copilot's suggestions depend heavily on the code around your cursor. These habits help it pick up on your intent:

- **Use meaningful names.** Clear variable and method names give Copilot clues about what you're building.
- **Write intentional comments.** Even quick notes can steer its suggestions.
- **Keep your structure clear.** Well-organized code helps Copilot (and your teammates) follow your thinking.
- **Stay near recent code.** Copilot's suggestions often depend on what you just wrote.

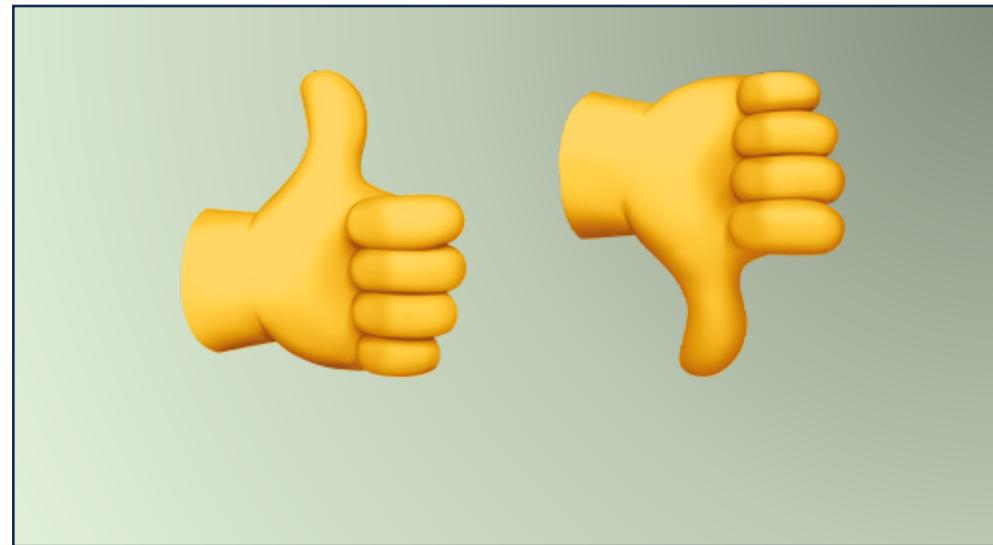
Exercise: Use Copilot to explore how comments guide its suggestions

Try experimenting with different kinds of comments to see how Copilot responds. You can use your own code, or download sample code for this exercise from <https://bit.ly/coding-ai-course>.

1. **Start a multi-line comment** and see if Copilot offers a bulleted or numbered list to continue it.
2. **Write a short documentation comment** above a method and check if Copilot tries to complete it—or even generate the method itself.
3. **Add a partial line of code**, like a call to a logging or validation method, and see if Copilot fills in something based on the code that came right before it.

Pulse Check

Could you start using these techniques today to help Copilot give you better suggestions for your own code?



Generating new code with GitHub Copilot

Your new AI skills will supercharge your use of GitHub Copilot. You're now primed to get the best out of this powerful tool. Now let's take it further and look at how to use Copilot to generate full methods and projects.

Strategies for getting great suggestions

Copilot can make a really big difference in handling complex coding challenges—but it works best when you use it **strategically**:

- **Start with structure.** Write an outline, comment, or partial method name to show Copilot where you're headed.
- **Break your task into smaller steps.** Copilot handles small, well-defined prompts much better than vague requests.
- **Be specific.** Use clear method names, well-named variables, and intentional comments to guide the suggestion.
- **Try writing the *call* first.** Let Copilot suggest the implementation after you write the call to the method.

Coding with Copilot is a partnership. Just remember who's in charge.

Simple tips to get the most out of Copilot and keep your code sharp.

- **You're the developer. Copilot is just here to help.** Don't treat Copilot's output as correct until you've read, tested, and understood it.
- **Refine, don't just accept.** Use Copilot's suggestions as a starting point—and then rewrite or improve as needed.
- **Give it context.** Good names, descriptive comments, and surrounding code all help Copilot suggest better results.
- **Iterate with it.** If the first suggestion doesn't work, adjust your code or comment and try again. And again. Keep trying until you're happy with it.

Go to <https://bit.ly/coding-ai-course>
for the slides in this deck.

Exercise

Let's use GitHub Copilot to create a simple project and explore its capabilities. You'll need GitHub Copilot installed and enabled in your IDE (I'll be using Visual Studio Code).

⚠ Generating code is uncertain! ⚠

- This exercise may take longer than the remaining course time. It's a take-home project for you to practice AI skills.
- Always review AI-generated code carefully. Copilot and other AI tools may sometimes produce code that doesn't compile or work as expected.
- You're in charge! Apply the best practices and critical thinking skills we've discussed throughout the course.
- I'm still here to help. If you encounter difficulties or want to see the process in action, a great way to get in touch with me is by raising an issue on the GitHub page for the course: <https://bit.ly/coding-ai-course>
- Every AI experience is different. But don't be discouraged if your AI doesn't work the way you want it to. Each interaction with AI can produce unique results! Remember—iterate, iterate, iterate.

Create a new project in VS Code

- **Before you begin:** Install the C# or Java installed in Visual Studio Code. Most people use Microsoft's *C# Dev Kit* or *Extension Pack for Java* extensions. Make sure a current version of .NET or JDK is installed. (You can use another IDE if you *really* want to—we'll include instructions to help you with that.)
- **For Java:** Choose “Java: Create Java Project...” from the Command Palette >> Maven “create from archetype” >> maven-archetype-quickstart. Rename the App class to MathQuiz. Open pom.xml and make sure the <maven.source.source> and <maven.source.target> tags match your JDK version.
- **For C#:** Choose “.NET: New Project...” from the Command Palette and create a new Console App project. Right-click on the solution in the Solution Explorer, choose “New Project...” and add an MSTest or Nunit project. Right-click on the new test project, choose “Add Project Reference” and add a reference to your console app project.

Use GitHub Copilot to code a simple game

- Create a new Java or C# app in your IDE. Remove all boilerplate from the Main method or top-level statements and add this comment:

```
// Create a console application that quizzes the user with  
// random math problems. Generate two random numbers from  
// 1 to 9 and pick either addition or multiplication.  
// Prompt for an answer, congratulate if correct, retry  
// if incorrect. Exit if user enters a non-numeric value.  
// Put the game in a class that is testable, with public  
// methods to generate operators, numbers, and questions.
```

- In Visual Studio Code you can press Ctrl + Enter to open Copilot Suggestions (there may be a pause before the Copilot displays its suggestions.)

Other IDEs may not have a Suggestions window, so just start creating the main class for your game—Copilot will generate a suggestion from the comment.

Carefully review the suggestions

- If you're using Visual Studio Code, Copilot opened a window with a list of suggestions. Find a suggestion that has public methods that can be tested, including a method that generates a question.
- *For other IDEs (or if you don't want to use the VSCode suggestions window), look closely at the suggested auto-complete code.*
- If you don't like your options, modify the prompt. Iterate until you have a suggestion you like.
- Accept the suggestion. Copilot will copy the code into your file. Run the app and make sure it works. If it does, you can delete the prompt comment at the top of the file.

Generate unit tests with Copilot

- Open the code file in your unit test project and add this comment:
`// Create unit tests for the math quiz game. Test
// random number generation, operation selection,
// and answer checking.`
- Press Ctrl+Enter to ask Copilot for suggestions. Find a suggestion that looks like it has good tests. Accept the suggestion.
- Choose “.NET: Rebuild” or “Java: Rebuild Projects” from the Command Palette. Make sure the code builds with no problems. If there are problems, either fix them by hand or delete the code and ask Copilot to regenerate the unit tests.

Run the unit tests and fix them if needed

- Choose *Testing* from the *View* menu or click the *Testing* button to open the *Testing* view in VS Code. If no tests are displayed, or if there's a message about installing extensions, make sure your code builds without problems.
- Run the tests.
- If a test doesn't pass, click inside the code for the test and open Copilot Chat (usually with **Ctrl+I** or **⌘+I**). Give it this prompt:
Why didn't this test pass?
- Read the response and fix the test so it passes.

Ask Copilot to test edge cases

You've been using the Copilot Suggestions window to create code, but there are other ways to use Copilot. We'll add more tests using two features, auto-completion and the `/tests` shortcut in Copilot Chat.

- Go to the bottom of the test class (above the closing bracket) and enter: `// Add a unit test to check an edge case`
- Copilot should give a suggestion for the annotation to start the test. Press Tab to accept it. Its next suggestion should be the body of the test. Accept it as well. Run the test and make sure it passes.
- Open Copilot Chat again and enter the following prompt:
`/tests add unit tests for additional edge cases`
- Copilot will suggest unit tests. Click Accept to accept them. Run the tests and make sure they pass. Fix any errors.

Refine and document the code

- Go back to your main code file and use Copilot Chat /doc command to add documentation. Open Copilot Chat and enter the following prompt:

C#: /doc add JavaDoc comments to all methods

Java: /doc add XMLDoc comments to all methods

- Review the generated documentation and make any necessary adjustments.
- Use Copilot Chat with the /explain command to get a full explanation of your refined code:
/explain this entire app

Use Copilot Chat to modify your code

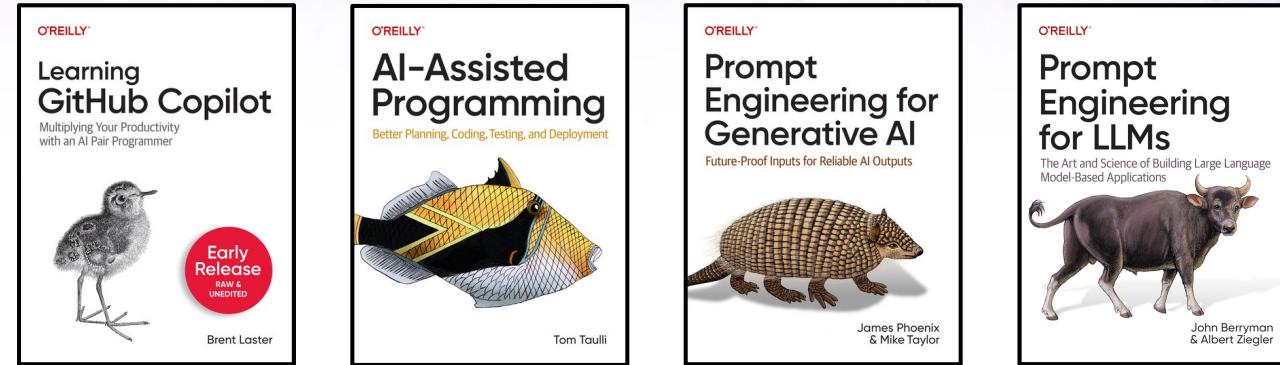
Copilot Chat can suggest changes to your code. Here are a few things to try:

- Split the code that generates the question into two public functions, one to generate a question and one to check the answer. Add unit tests for both.
- Modify the app so it keeps track of how many questions in a row the player gets right. Remember the longest question streak. Implement this with public methods and generate unit tests for those methods.
- Implement robust error handling and input validation. Create a separate method to validate user input, ensuring it handles various edge cases (e.g., non-numeric input, extremely large numbers, negative numbers). Generate unit tests for this validation method, including tests for different types of invalid input.

Q&A

If you have any questions, enter them in the Q&A widget.

Additional resources



Want to dive deeper into AI-assisted coding and prompt engineering?
Here are some great places to start.

- “Learning GitHub Copilot” by Brent Lester (O'Reilly, 2025) – This book is amazing! An early release is available today, and I wrote the foreword for it.
<https://learning.oreilly.com/library/view/learning-github-copilot/9781098164645/>
- “AI-Assisted Programming” by Tom Taulli (O'Reilly, 2024):
<https://learning.oreilly.com/library/view/ai-assisted-programming/9781098164553/>
- “Prompt Engineering for Generative AI” by James Phoenix & Mike Taylor (O'Reilly, 2024)
<https://learning.oreilly.com/library/view/prompt-engineering-for/9781098153427/>
- “Prompt Engineering for LLMs” by John Berryman and Albert Ziegler (O'Reilly, 2024)
<https://learning.oreilly.com/library/view/prompt-engineering-for/9781098156145/>



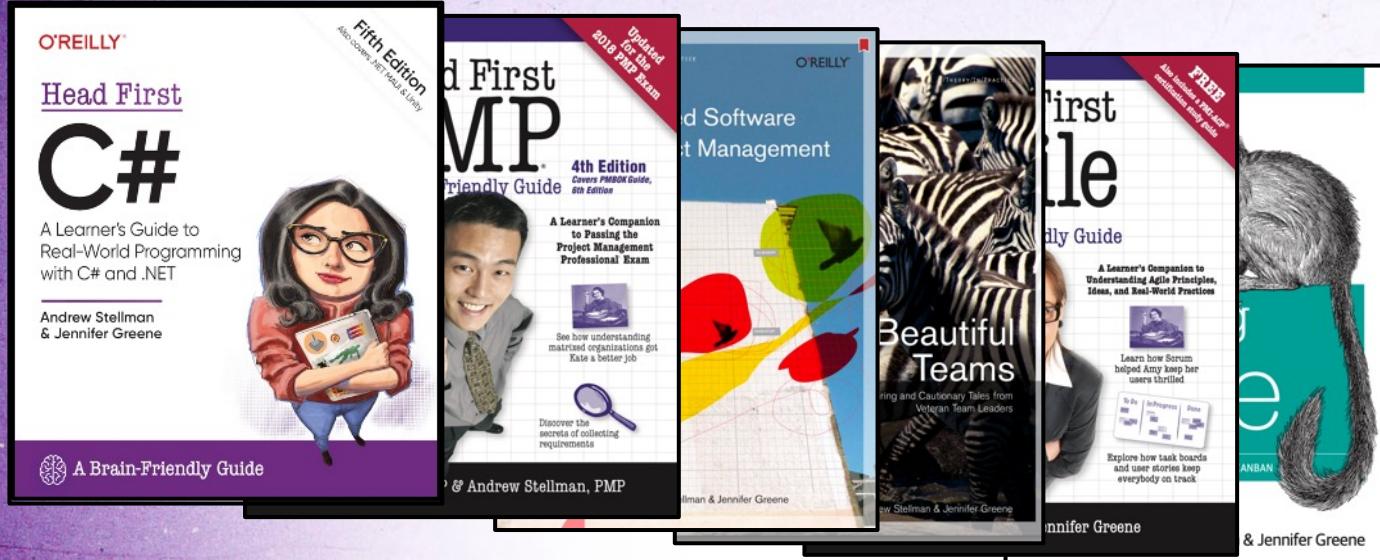
linkedin.com/in/andrewstellman/



github.com/andrewstellman



@andrewstellman.bsky.social



Thank you!

I hope you enjoyed my course! If you want to learn more, follow me on social media, and check out *Head First C#* and my other books on O'Reilly Learning.

Thanks so much for attending!