✓ 100 XP

Examine GitHub Copilot best practices

8 minutes

The GitHub Copilot extensions for Visual Studio Code help developers write code faster and with fewer errors.

However, GitHub Copilot is an AI pair programmer, it's not a programmer. To get the most out of the GitHub Copilot tools, you need to follow best practices.

Choose the right Copilot tool for the job

While GitHub Copilot's code completion features and GitHub Copilot's chat features share some functionality, the two tools are best used in different circumstances.

Code completions provide the most value when they're used to:

- Complete code snippets, variable names, and functions as you write them.
- Generate repetitive code.
- Generate code from inline comments in natural language.
- Generate tests for test-driven development.

Alternatively, GitHub Copilot's chat features provide the most value when they're used to:

- Answer questions about code in natural language.
- Generate large sections of code, then iterate on that code to meet your needs.
- Accomplish specific tasks with keywords and skills. GitHub Copilot Chat uses built-in keywords (and skills designed to provide important context) within prompts to accomplish common tasks quickly. Different types of keywords and skills are available in different Copilot Chat platforms.
- Complete a task as a specific persona. For example, you can tell Copilot Chat that it's a Senior C++ Developer who cares greatly about code quality, readability, and efficiency. After establishing the persona, you can ask GitHub Copilot to review your code.

Get the most out of Copilot inline suggestions

The GitHub Copilot extension presents suggestions automatically to help you code more efficiently. GitHub Copilot provides better suggestions when you follow certain actions and behaviors. You may already be following some of these actions and behaviors, since they also help you and your colleagues understand your code.

Provide context for GitHub Copilot

GitHub Copilot works best when it has sufficient context to know what you're doing and what you want help with. Just as you provide a colleague with relevant context when asking for help, you need to provide GitHub Copilot with context.

Open files

For code completions, GitHub Copilot looks at the files that are open in your editor to establish context and create appropriate suggestions. Having related files open in Visual Studio Code while using GitHub Copilot helps to define context and lets Copilot see the bigger picture of your project.

Top level comment

Just as you would give a brief, high-level introduction to a coworker, a top level comment in a code file can help GitHub Copilot understand the overall context of the code you're creating.

Appropriate includes and references

It's best to manually set the includes or module references you need for your work. GitHub Copilot can make suggestions, but you likely know best what dependencies you need to include. This can also inform GitHub Copilot about the frameworks, libraries, and their versions you'd like it to use when crafting suggestions.

Meaningful function names

Just as a method called fetchData() won't mean much to a coworker (or you after several months), fetchData() won't help GitHub Copilot to understand your code. Using meaningful function names helps Copilot to generate suggestions that do what you want.

Specific and well-scoped function comments

A function name can only be so descriptive without being overly long. Function comments can help fill in details that Copilot might need to know.

Prime Copilot with sample code

One trick to get Copilot on the right page, is to copy and paste desirable code samples into an open editor tab. The sample should represent the code that you want GitHub Copilot to emulate. Providing an example can help GitHub Copilot generate suggestions that match the language and tasks you want it to emulate. Once Copilot has helped you create some code that matches your goal, you can delete the sample code. This approach is especially helpful when you need to "train" Copilot. For example, suppose GitHub Copilot defaults to suggesting code snippets that implement an older version of a library. You can insert code into your project that uses the latest version of the library. This helps Copilot to begin suggesting code snippets that use the latest version of the library. Once GitHub Copilot has learned to use the new library version, you can delete the sample code.

Be consistent and keep the quality bar high

GitHub Copilot is going to latch on to your code to generate suggestions that follow the existing pattern, so the adage "garbage in, garbage out" applies.

Always keeping a high quality bar can take discipline. Especially when you're coding fast and loose to get something working. You might want to disable GitHub Copilot completions while in "hacking" mode. You can temporarily disable completions from the GitHub Copilot status menu (accessible from Visual Studio Code's status bar).

Get the most out of GitHub Copilot Chat

When you're using GitHub Copilot's chat features, there are several things you can do to optimize your experience.

Start general, then get specific

When writing a prompt for GitHub Copilot, first give Copilot a broad description of the goal or scenario. Then list any specific requirements. Consider the following prompts:

1. Write a function that tells me if a number is prime.
2. The function should take an integer and return true if the integer is prime.
3. The function should throw an error if the input isn't a positive integer.

Give examples

Use examples to help GitHub Copilot understand what you want. You can provide example input data, example outputs, and example implementations.

Break complex tasks into simpler tasks

If you want GitHub Copilot to complete a complex or large task, break the task into multiple simple, small tasks. For example, instead of asking Copilot to generate a word search puzzle, break the process down into smaller tasks, and ask GitHub Copilot to accomplish them one by one:

1. Write a function to generate a 10 by 10 grid of letters.
2. Write a function to find all words in a grid of letters, given a list of valid words.
3. Write a function to that uses the previous functions to generate a 10 by 10 grid of letters that contains at least 10 words.
4. Update the previous function to print the grid of letters and 10 random words from the grid.

Avoid ambiguity

Avoid ambiguous terms. For example, don’t ask "what does this do" when "this" could be the current file, the last GitHub Copilot response, or a specific code block. Instead of using ambiguous terms, be specific: What does the createUser function do?

Indicate relevant code

If you're using GitHub Copilot Chat in your IDE, open the file(s) or highlight the code that you want GitHub Copilot to reference. Use chat participants, slash commands, and chat variables in prompts to define context. For example, specify which files GitHub Copilot Chat should reference.

Use chat participants, slash commands, and chat variables

Chat participants are designed to collect extra context either about a code base or a specific domain or technology. When you specify the appropriate participant, GitHub Copilot Chat can find and provide better information to send to the Copilot backend. For example, use `@workspace` when you ask questions about your open project, or `@vscode` when you ask questions about Visual Studio Code features and APIs.

Slash commands help GitHub Copilot Chat to understand your intent when you ask a question. Are you learning about a code base (`/explain`), do you want help with fixing an issue (`/fix`), or are you creating test cases (`/tests`)? By letting Copilot Chat know what you're trying to do, it can tune its reply to your task and provide helpful commands, settings, and code snippets.

Chat participants, such as `@workspace` or `@vscode`, can contribute chat variables that provide domain-specific context. You can reference a chat variable in your chat prompt by using the `#` symbol. You can use chat variables to be more specific about the context of your prompt.

For example, the `#file` chat variable lets you reference specific files from your workspace in your chat prompt. This helps make the answers from GitHub Copilot Chat more relevant to your code by providing context about the file you're working with. You can ask questions like "`Can you suggest improvements to #file:package.json?`" or "`How do I add an extension in #file:devcontainer.json?`". Using the `#file` variable can help you get more targeted and accurate responses from Copilot.

Experiment and iterate

If you don't get the result that you want, iterate on your prompt and try again. Reference the previous response in your next request. You can also delete the previous response and start over.

Keep history relevant

GitHub Copilot Chat uses the chat history to get context about your request. To ensure that GitHub Copilot is using a relevant chat history:

- Use threads to start a new conversation for a new task.
- Delete requests that are no longer relevant or that didn't give you the desired result.

Follow good coding practices

If you aren't getting the responses that you want when you ask GitHub Copilot for suggestions, make sure that your existing code follows best practices and is easy to read. For example:

- Use a consistent code style and patterns.
- Use descriptive names for variables and functions.
- Comment your code.
- Structure your code into modular, scoped components.
- Include unit tests.

Summary

GitHub Copilot is a powerful tool that can help you write code faster and with fewer errors. However, to get the most out of GitHub Copilot, you need to follow best practices. By choosing the right Copilot tool for the job, providing context, and following good coding practices, you can optimize your experience with GitHub Copilot.

Next unit: Generate code using GitHub Copilot code completion suggestions

< Previous Next >