Try agent mode (vscode://GitHub.Copilot-Chat/chat?mode=agent&referrer=vscode-agentbanner) in VS Code!

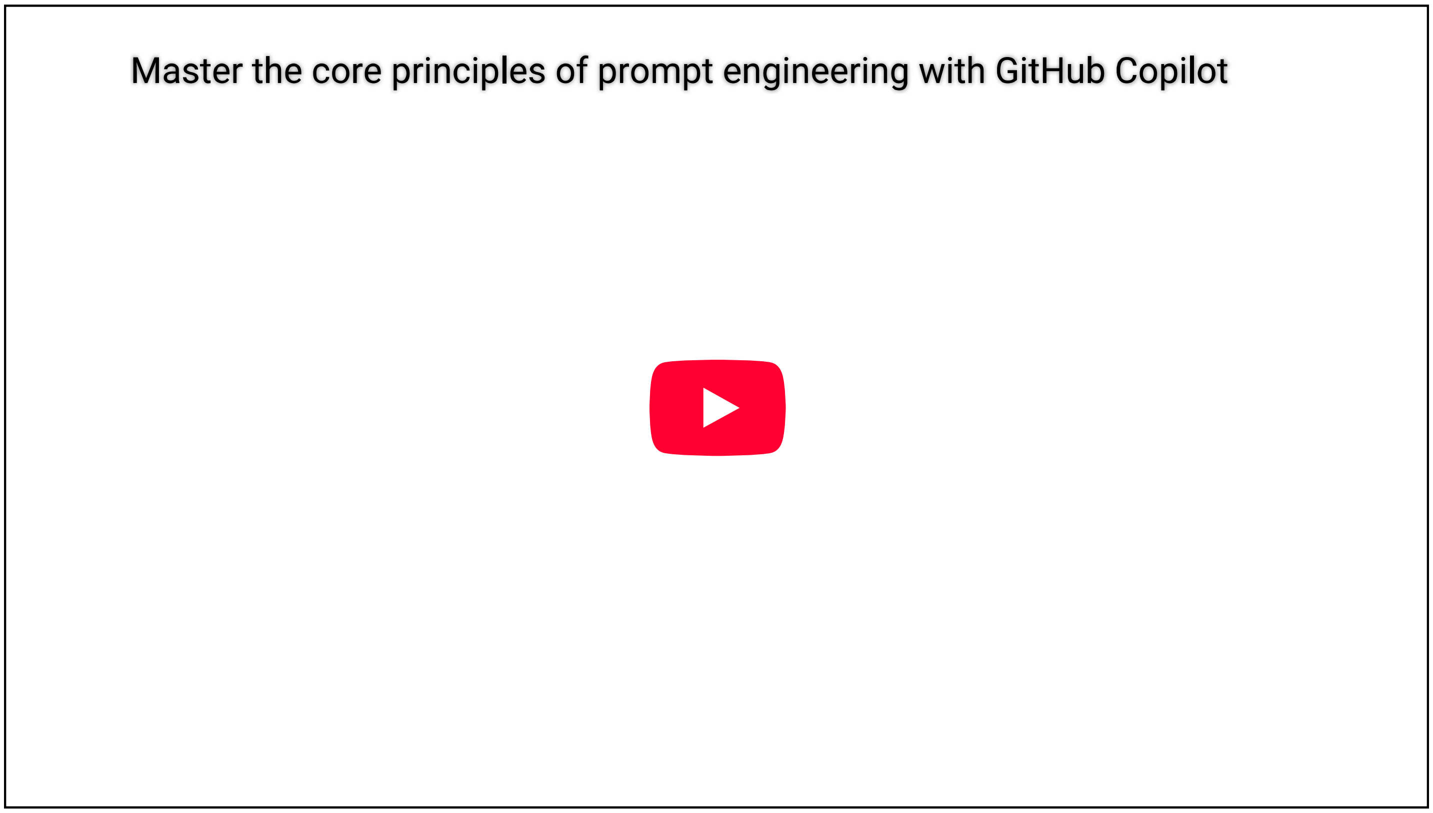
TOPICS Prompt Engineering

IN THIS ARTICLE Getting the most out of Copilot inli

(https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/copilot/chat/prompt-crafting.md)

Prompt engineering for Copilot Chat

This article covers tips to write prompts to get better and more relevant responses from Copilot Chat in Visual Studio Code. *Prompt engineering* or *prompt crafting* is a common phrase you'll hear when discussing AI and refers to how and what information is packaged and sent to an AI API endpoint.


Master the core principles of prompt engineering with GitHub Copilot

If you are new to VS Code or GitHub Copilot, you might want to review the GitHub Copilot Overview (/docs/copilot/overview) article first or dive straight into the Getting started (/docs/copilot/getting-started) tutorial.

There are different options for optimizing your Copilot experience for inline suggestions and chat:

- Get the most out of inline suggestions

- Get the most out of Copilot Chat

Getting the most out of Copilot inline suggestions

The GitHub Copilot (https://marketplace.visualstudio.com/items?itemName=GitHub.copilot) extension presents suggestions (/docs/copilot/overview#_Code-completions-in-the-editor) automatically to help you code more efficiently. There are things you can do to help ("prompt") Copilot to give you the best possible suggestions. And the good news is that you are probably already doing these right now, since they help you and your colleagues understand your code.

Provide context to Copilot

Copilot works best when it has sufficient context to know what you're doing and what you want help with. Just as you would provide a colleague with the context when asking for help with a specific programming task, you can do the same with Copilot.

Open files

For code completions, Copilot looks at the current and open files in your editor to analyze the context and create appropriate suggestions. Having related files open in VS Code while using Copilot helps set this context and lets the Copilot see a bigger picture of your project.

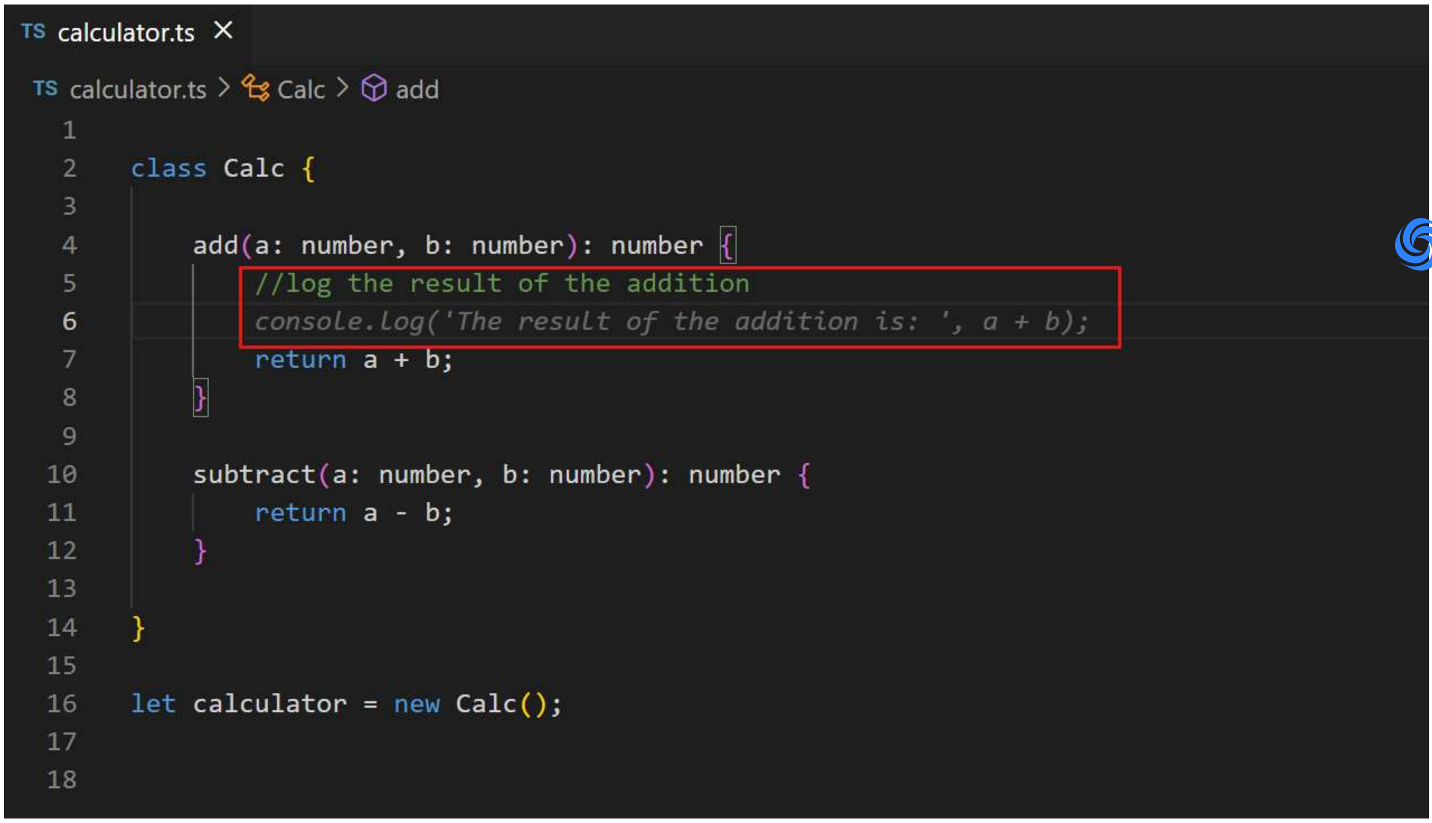
Top level comment

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

Appropriate includes and references

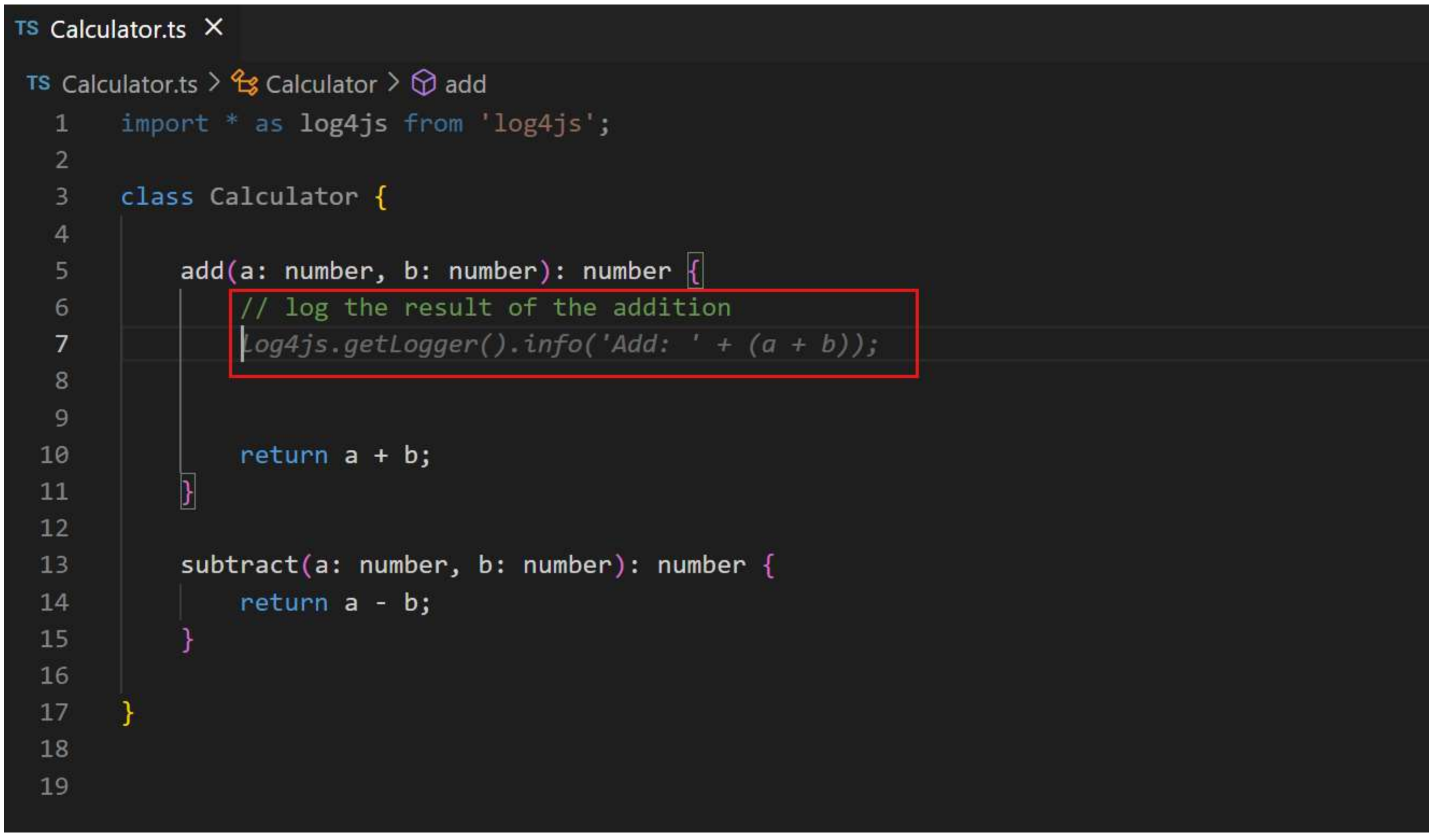
It's best to manually set the includes or module references you need for your work. Copilot can make suggestions, but you likely know best what dependencies you need to include. This can also help let Copilot know what frameworks, libraries, and their versions you'd like it to use when crafting suggestions.

In the following TypeScript example, we want to log the output of the `add` method. When we don't have any includes, Copilot suggests using `console.log`:



```
TS calculator.ts > Calc > add

class Calc {

    add(a: number, b: number): number {
        //log the result of the addition
        console.log('The result of the addition is: ', a + b);
        return a + b;
    }

    subtract(a: number, b: number): number {
        return a - b;
    }

}

let calculator = new Calc();
```

On the other hand, when you add a reference to `Log4js`, Copilot suggests using that framework for logging the output:



```
TS Calculator.ts > Calculator > add
import * as log4js from 'log4js';

class Calculator {

    add(a: number, b: number): number {
        // log the result of the addition
        Log4js.getLogger().info('Add: ' + (a + b));


        return a + b;
    }

    subtract(a: number, b: number): number {
        return a - b;
    }

}
```

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 Copilot either. Using meaningful function names helps Copilot provide a body that does 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.

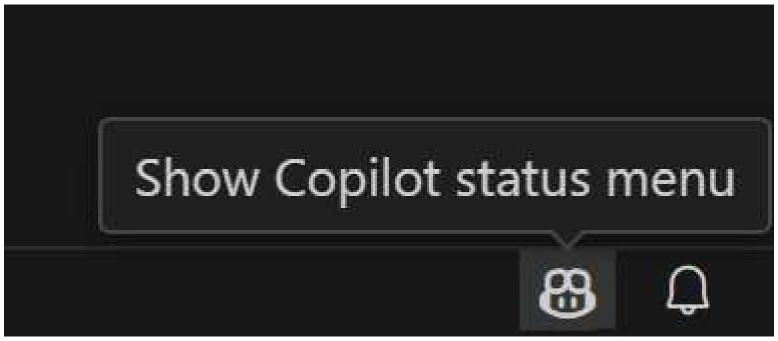
Prime Copilot with sample code

One trick to get Copilot on the right page, is to copy and paste sample code that is close to what you are looking for into your open editor. Providing a small example can help Copilot generate suggestions that match the language and tasks you want to achieve. Once Copilot begins providing you with the code you want and will actually use, you can delete the sample code from the file. This can be especially helpful to jump start Copilot to a newer library version when it defaults to providing older code suggestions.

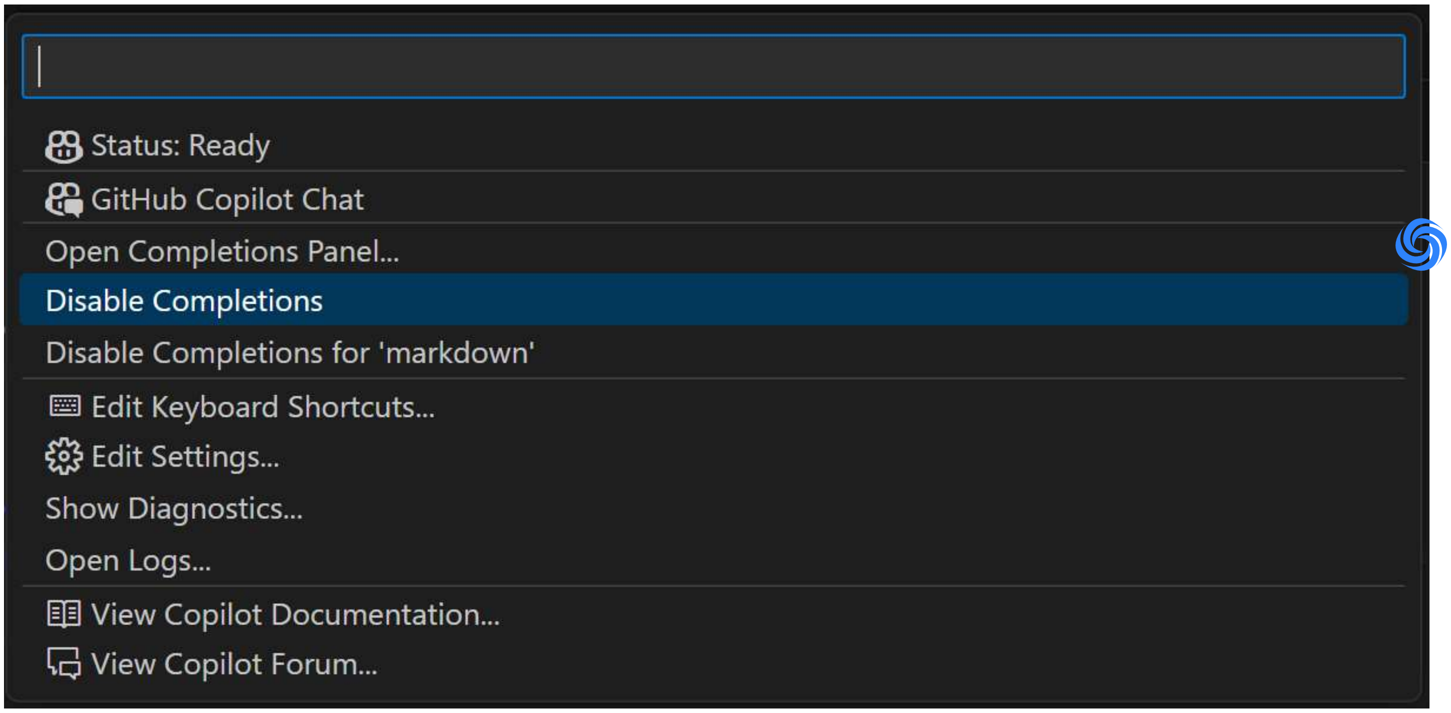
Be consistent and keep the quality bar high

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 Copilot completions while in "hacking" mode. You can temporarily disable completions from the Copilot status menu. Bring up the Copilot status menu dropdown by selecting the Copilot Status bar item.



From the dropdown, you can disable completions entirely, or disable just for the active file type, for example Markdown files.



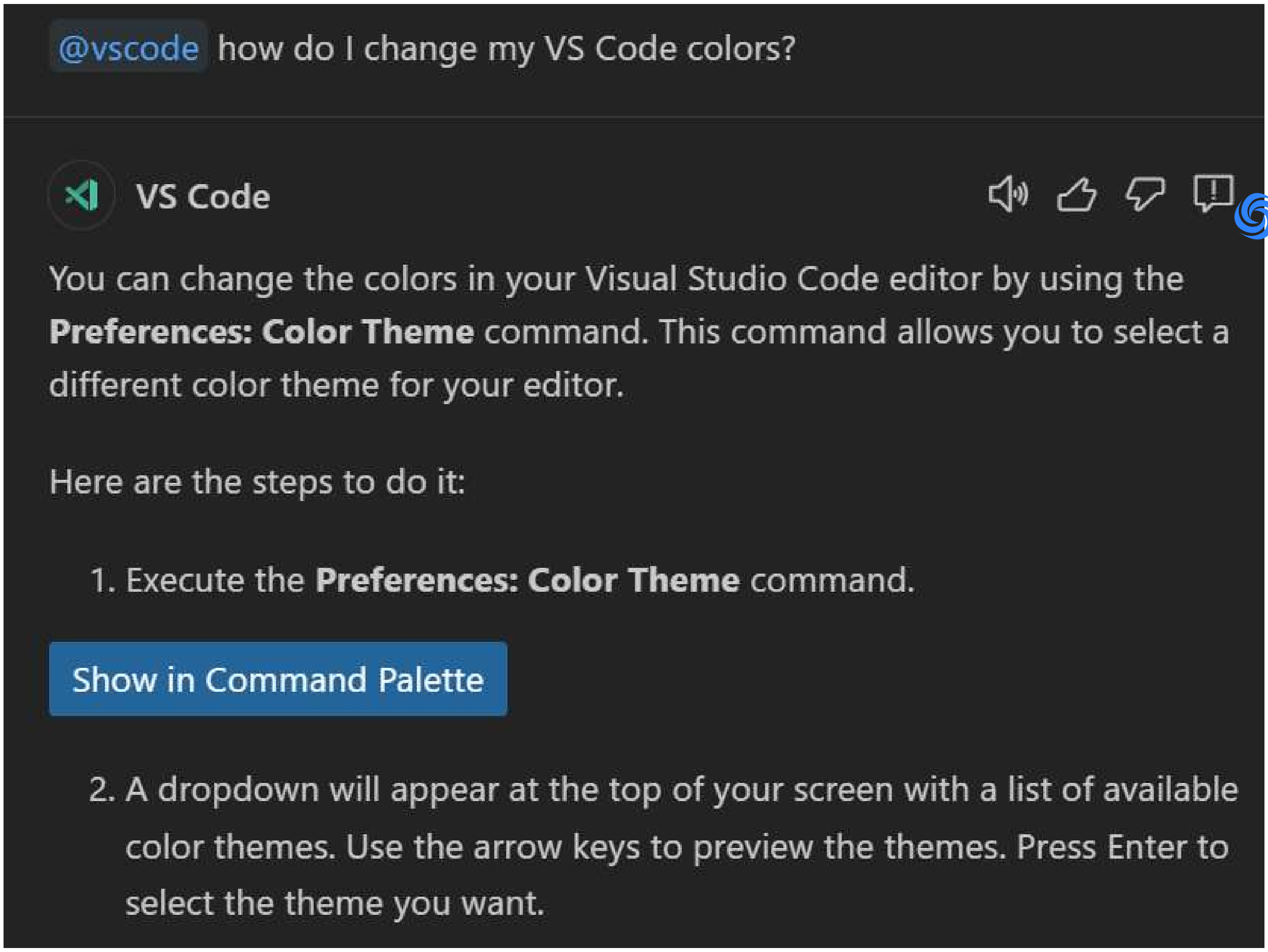
Getting the most out of Copilot Chat

You can also get assistance from Copilot via a chat interface (/docs/copilot/overview#_Answer-coding-questions) by installing the GitHub Copilot Chat (https://marketplace.visualstudio.com/items?itemName=GitHub.copilot-chat) extension.

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

Use chat participants and slash commands

Chat participants are designed to collect extra context either about a code base or a specific domain or technology. By using the appropriate participant, Copilot Chat can find and provide better information to send to the Copilot backend. For example, use `@workspace` if you want to ask questions about your open project, or `@vscode` to know more about VS Code features and APIs.



@vscode how do I change my VS Code colors?

VS Code

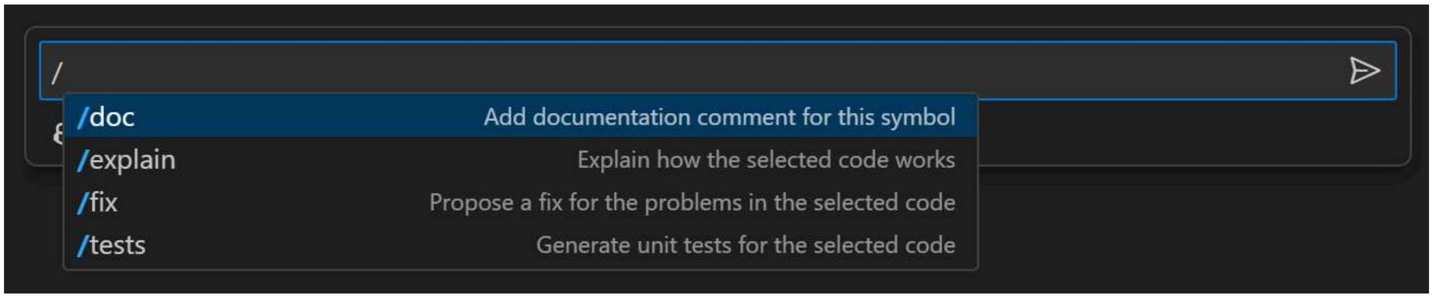
You can change the colors in your Visual Studio Code editor by using the **Preferences: Color Theme** command. This command allows you to select a different color theme for your editor.

Here are the steps to do it:

1. Execute the **Preferences: Color Theme** command.

2. A dropdown will appear at the top of your screen with a list of available color themes. Use the arrow keys to preview the themes. Press Enter to select the theme you want.

Slash commands help Copilot Chat 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.



/doc	Add documentation comment for this symbol
/explain	Explain how the selected code works
/fix	Propose a fix for the problems in the selected code
/tests	Generate unit tests for the selected code

You could write out your project scope or current task with a natural language query but using chat participants and slash commands is more concise and explicit.

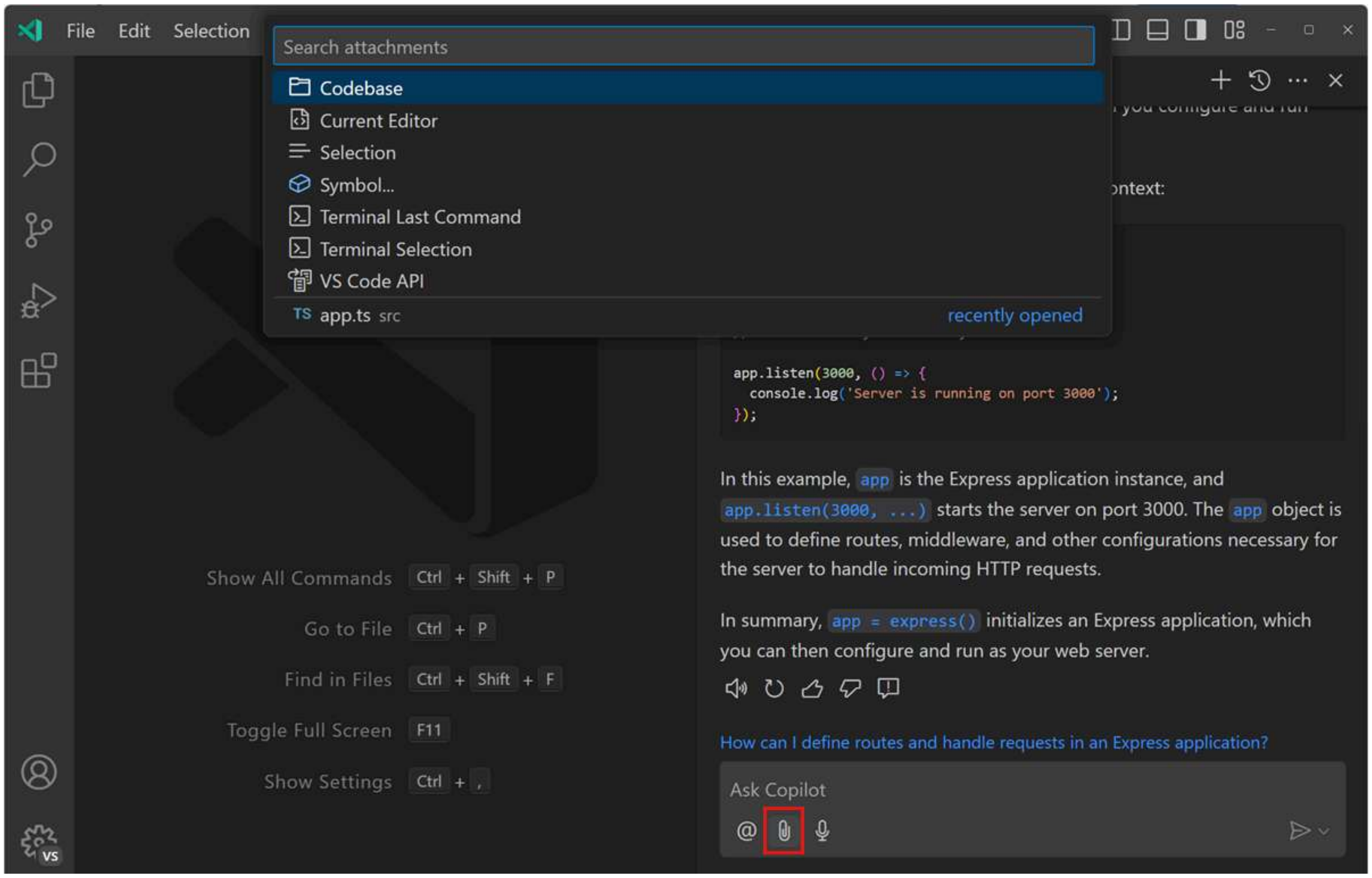
Learn more about chat participants (/docs/copilot/chat/chat-ask-mode#_special-keywords) and slash commands (/docs/copilot/chat/chat-ask-mode#_special-keywords) in Copilot Chat.

Use chat variables for context

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. By using a chat variable, you can be more specific about the context that you include in your chat prompt.

For example, the `#file` variable lets you reference specific files from your workspace in your chat prompt. This helps make the answers from Copilot Chat more relevant to your code by providing context about the file you are 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?". By using the `#file` variable, you can get more targeted and accurate responses from Copilot.

You can also add context to your chat message by using the **Attach Context** button in the Chat view. You can then select the specific type of context from a Quick Pick, such as the current selection, one or more files from the workspace, or one or more symbols from your source code.



```
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

In this example, `app` is the Express application instance, and `app.listen(3000, ...)` starts the server on port 3000. The `app` object is used to define routes, middleware, and other configurations necessary for the server to handle incoming HTTP requests.

In summary, `app = express()` initializes an Express application, which you can then configure and run as your web server.

Learn more about [using context variables with Copilot Chat (/docs/copilot/chat/chat-ask-mode#_special-keywords)](/docs/copilot/chat/chat-ask-mode#_special-keywords).

Be specific and keep it simple

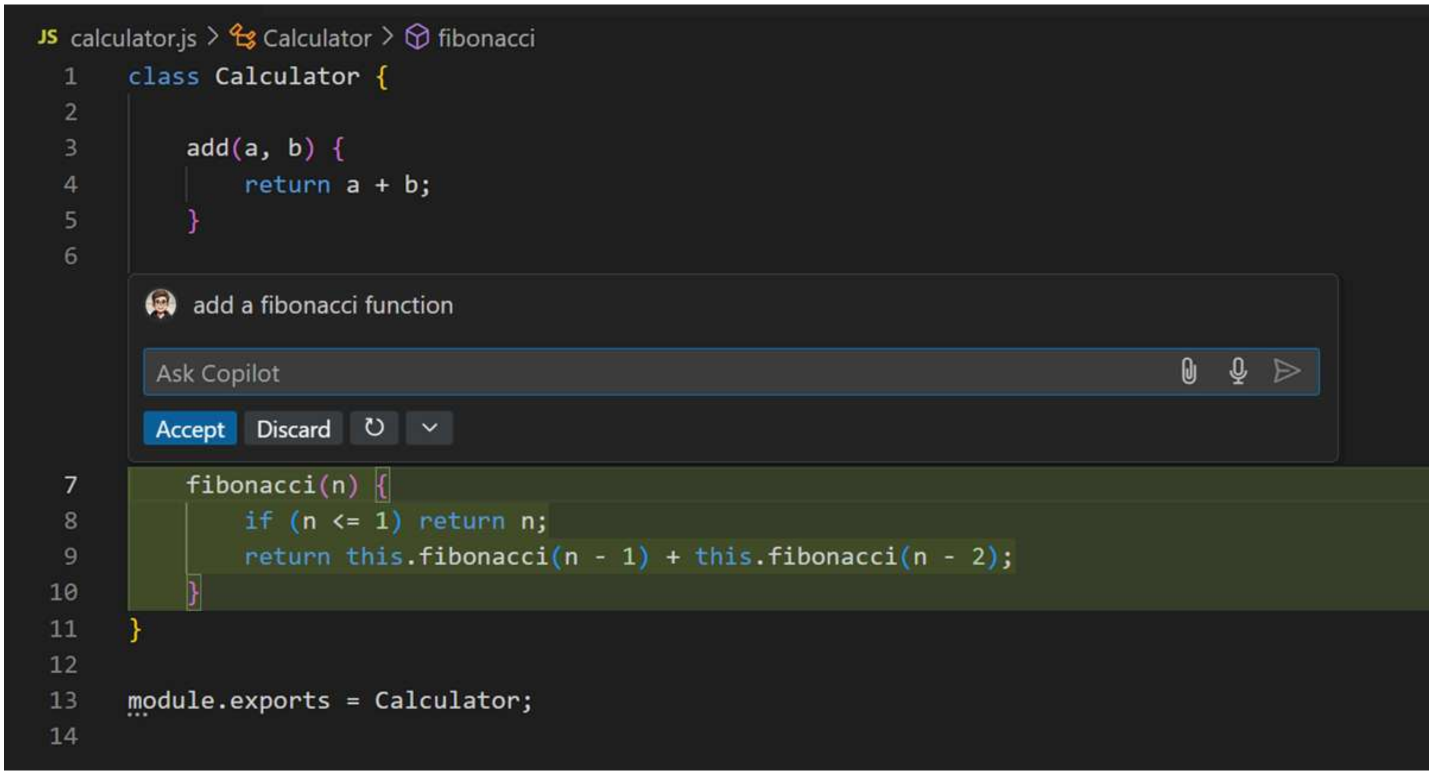
When you ask Copilot to do something, be specific in your ask and break down a large task into separate, smaller tasks. For example, don't ask Copilot to create an Express app, that uses TypeScript and Pug, and that has a products page that retrieves data from a MongoDB database. Instead, first ask Copilot to create the Express app with TypeScript and Pug. Next, ask to add a products page, and finally ask to retrieve the customer data from a database.

When you ask Copilot to do a specific task, be specific about the inputs, outputs, APIs, or frameworks you want to use. The more specific your prompt is, the better the outcome will be. For example, instead of "read product data from the database", use "read all products by category, return the data in JSON format, and use the Mongoose library".

Iterate on your solution

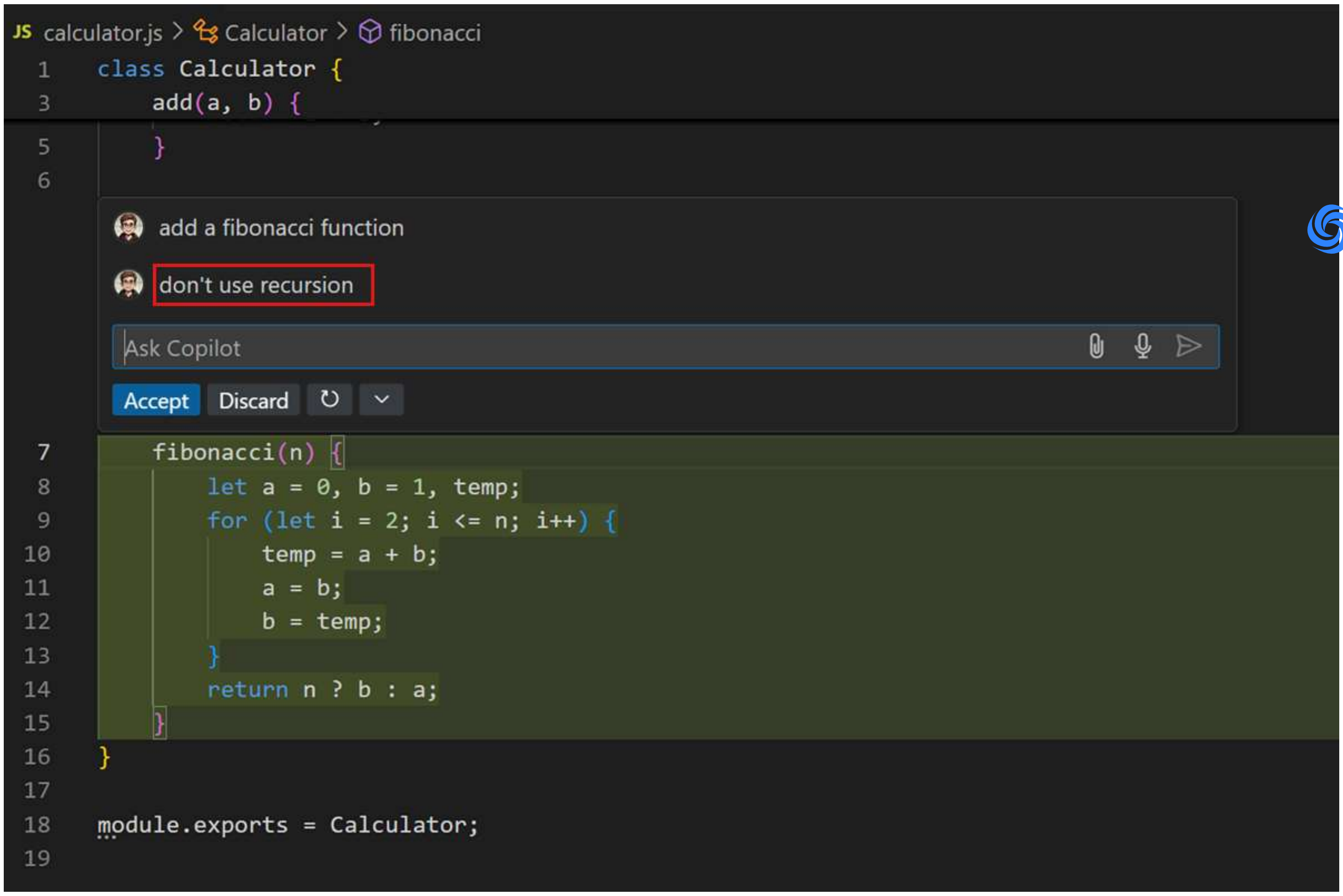
When asking Copilot Chat for help, you aren't stuck with the first response. You can iterate and prompt Copilot to improve the solution. Copilot has both the context of the generated code and also your current conversation.

Here's an example using Inline Chat to create a function to calculate Fibonacci numbers:



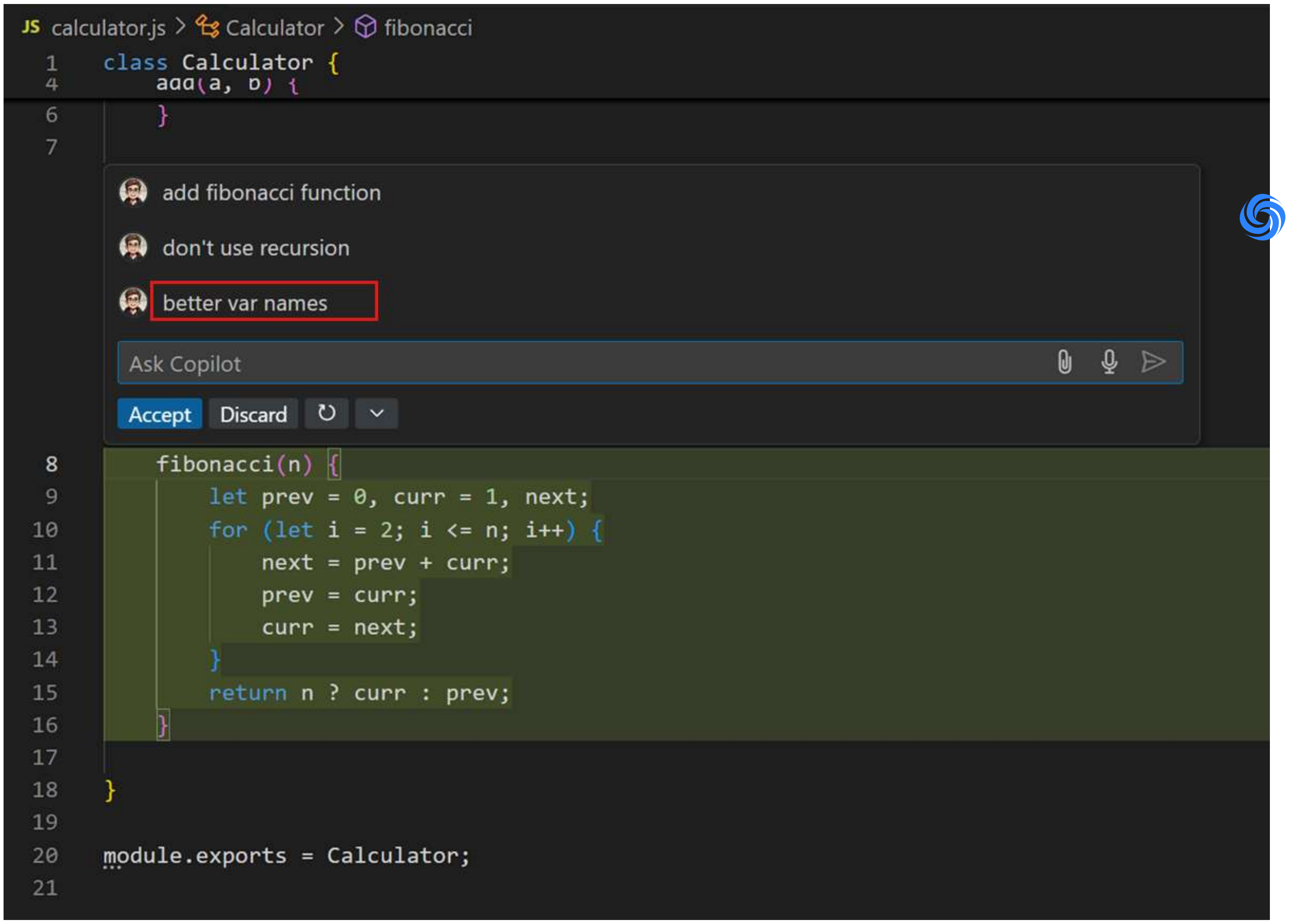
```
class Calculator {

    add(a, b) {
        return a + b;
    }

    fibonacci(n) {
        if (n <= 1) return n;
        return this.fibonacci(n - 1) + this.fibonacci(n - 2);
    }
}

module.exports = Calculator;
```

Maybe you prefer a solution that doesn't use recursion:



```
class Calculator {
    add(a, b) {
    }

    fibonacci(n) {
        let a = 0, b = 1, temp;
        for (let i = 2; i <= n; i++) {
            temp = a + b;
            a = b;
            b = temp;
        }
        return n ? b : a;
    }
}

module.exports = Calculator;
```

You can even ask Copilot to follow coding conventions or improve variable names:



```
class Calculator {
    add(a, b) {
    }

add fibonacci function
don't use recursion
better var names

    fibonacci(n) {
        let prev = 0, curr = 1, next;
        for (let i = 2; i <= n; i++) {
            next = prev + curr;
            prev = curr;
            curr = next;
        }
        return n ? curr : prev;
    }

}

module.exports = Calculator;
```

Even if you've already accepted a result, you can always ask Copilot to iterate on the code later.

More resources about prompting for Copilot

If you'd like to learn more about productively using GitHub Copilot, you can follow up with these videos and blog posts:

- Effective Prompting for GitHub Copilot (https://www.youtube.com/watch?v=ImWfIDTxn7E)
- Pragmatic techniques to get the most out of GitHub Copilot (https://www.youtube.com/watch?v=CwAzIpc4AnA)
- Best practices for prompting GitHub Copilot in VS Code (https://www.linkedin.com/pulse/best-practices-prompting-github-copilot-vs-code-pamela-fox)
- How to use GitHub Copilot: Prompts, tips, and use cases (https://github.blog/2023-06-20-how-to-write-better-prompts-for-github-copilot/)

Was this documentation helpful?

Yes No

04/03/2025

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://www.microsoft.com)

(https://go.microsoft.com/fwlink/?LinkID=533687)

(https://github.com/microsoft/vscode)

(https://www.youtube.com/@code)