Before you start

Exercise scenario

Examine GitHub Copilot settings in Visual Studio Code

Explore the GitHub Copilot user interface in Visual Studio Code

Summary

Clean up

Examine GitHub Copilot settings and user interface features

Visual Studio Code provides a seamless and customizable GitHub Copilot experience for developers. In this exercise you examine GitHub Copilot settings and explore the GitHub Copilot user interface in Visual Studio Code.

This exercise should take approximately **25** minutes to complete.

> **IMPORTANT**: To complete this exercise, you must provide your own GitHub account and GitHub Copilot subscription. If you don't have a GitHub account, you can sign up for a free individual account and use a GitHub Copilot Free plan to complete the exercise. If you have access to a GitHub Copilot Pro, GitHub Copilot Pro+, GitHub Copilot Business, or GitHub Copilot Enterprise subscription from within your lab environment, you can use your existing GitHub Copilot subscription to complete this exercise.

Before you start

Your lab environment must include the following:

- Git 2.48 or later
- Either .NET or Python:
  - **.NET SDK 9.0** or later with Visual Studio Code with the **C# Dev Kit** extension.
  - **Python 3.10** or later with Visual Studio Code with the **Python** extension
- Access to a GitHub account with GitHub Copilot enabled.

If you're using a local PC as a lab environment for this exercise:

- For help configuring your local PC as your lab environment, open the following link in a browser: Configure your lab environment resources.
- For help enabling your GitHub Copilot subscription in Visual Studio Code, open the following link in a browser: Enable GitHub Copilot within Visual Studio Code.

If you're using a hosted lab environment for this exercise:

- For help enabling your GitHub Copilot subscription in Visual Studio Code, paste the following URL into a browser's site navigation bar: Enable GitHub Copilot within Visual Studio Code.
- Open a command terminal and then run the following commands:

  To ensure that Visual Studio Code is configured to use the correct version of **.NET**, run the following command:

  Code Copy

  ```
  dotnet nuget add source https://api.nuget.org/v3/index.json -n nuget.org
  ```

  or

  To ensure that Visual Studio Code is configured to use the correct version of Python, verify your Python installation is version 3.10 or later:

  Code Copy

```
python --version
```

Exercise scenario

You're a developer working in the IT department of your local community. The backend systems that support the public library were lost in a fire. Your team needs to develop a temporary solution to help the library staff manage their operations until the system can be replaced. Your team chose GitHub Copilot to accelerate the development process.

This exercise includes the following tasks:

1. Examine GitHub Copilot settings in Visual Studio Code.
2. Explore the GitHub Copilot user interface in Visual Studio Code.

Examine GitHub Copilot settings in Visual Studio Code

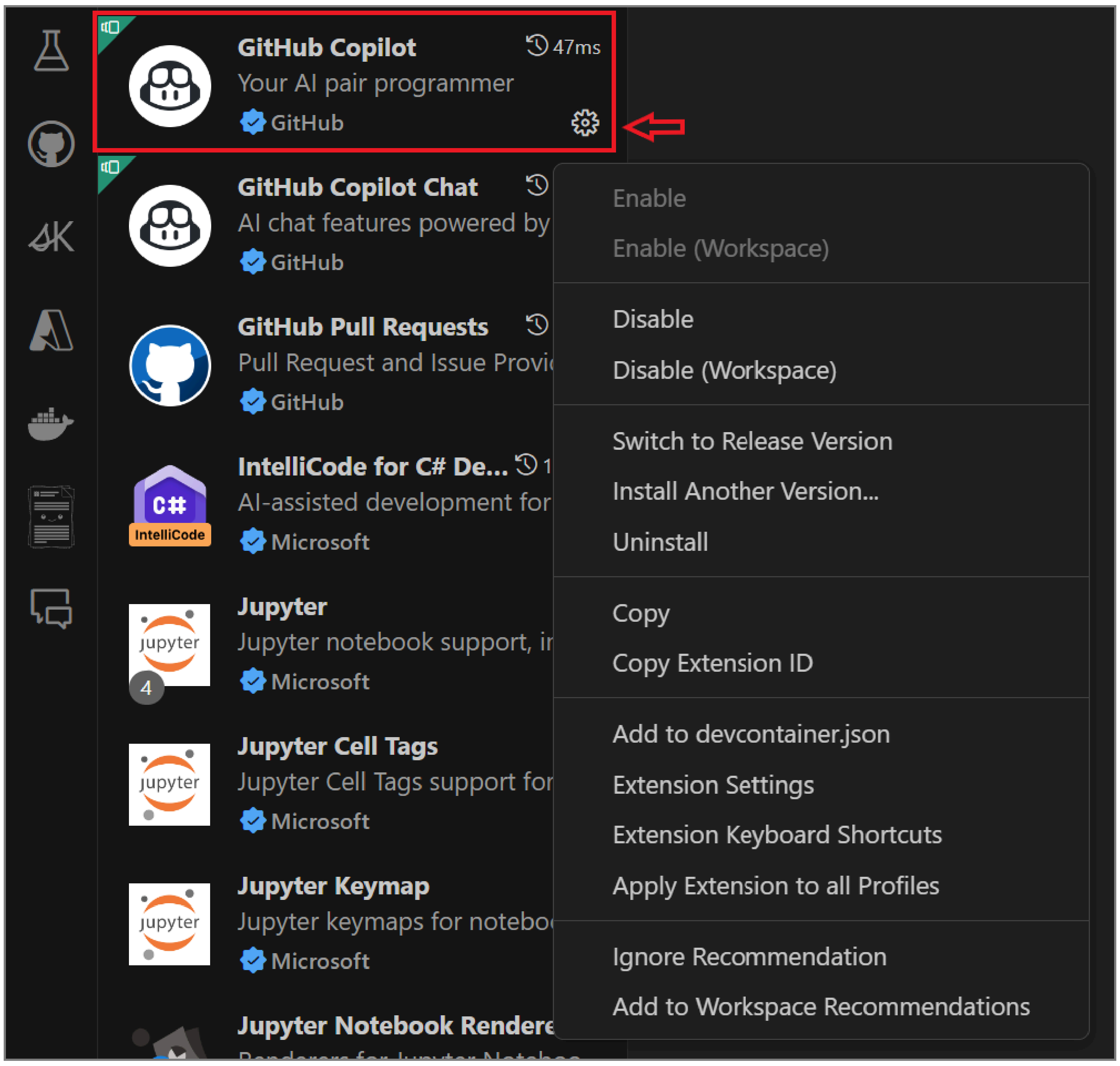
GitHub Copilot settings are split between your GitHub account and the Visual Studio Code environment. In Visual Studio Code, settings enable you to configure the behavior of GitHub Copilot and GitHub Copilot Chat. In your GitHub account, settings enable you to manage your GitHub Copilot subscription, configure the retention of prompts and suggestions, and allow or block suggestions matching public code.

Enable or disable GitHub Copilot in Visual Studio Code

The GitHub Copilot extensions in Visual Studio Code are enabled by default when you activate a plan. However, you can disable GitHub Copilot in Visual Studio Code for a period of time if you need to.

Use the following steps to complete this section of the exercise:

1. Open a new instance of Visual Studio Code.
2. In Visual Studio Code, open the **Extensions** view.
3. In the list of installed extensions, scroll down until you find **GitHub Copilot**.
4. To display a dropdown menu for the GitHub Copilot extension that lists Enable and Disable options, select on the gear icon for the GitHub Copilot extension.



If you want to test the enable/disable options, you can select the disable option. However, be sure to re-enable GitHub Copilot before you continue with this exercise.

Examine settings for the GitHub Copilot extensions

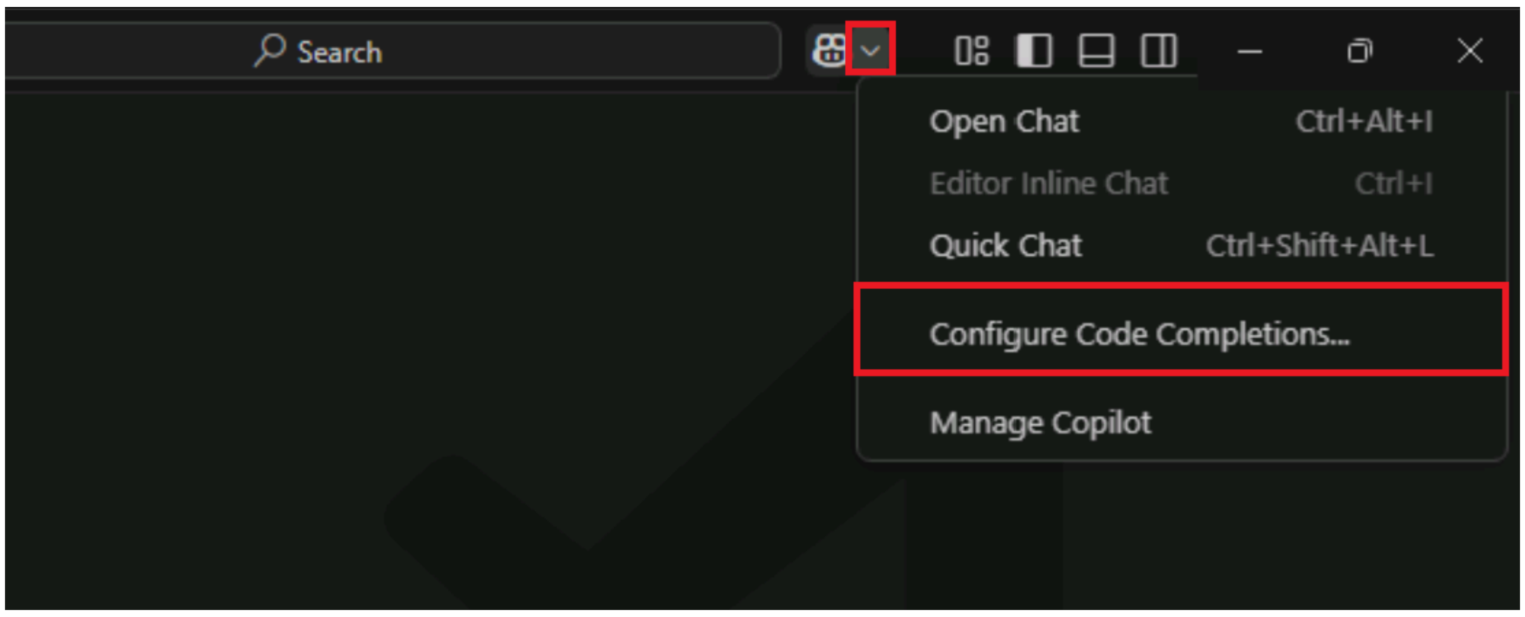
Default settings are configured for you when you activate GitHub Copilot in Visual Studio Code. The settings are organized under the Extensions label, which includes settings for both GitHub Copilot and GitHub Copilot Chat. You can customize settings for the GitHub Copilot extensions using Visual Studio Code's settings tab.

> **TIP**: You can use the Copilot Chat menu to open a Settings tab that's filtered for GitHub Copilot.

Use the following steps to complete this section of the exercise:

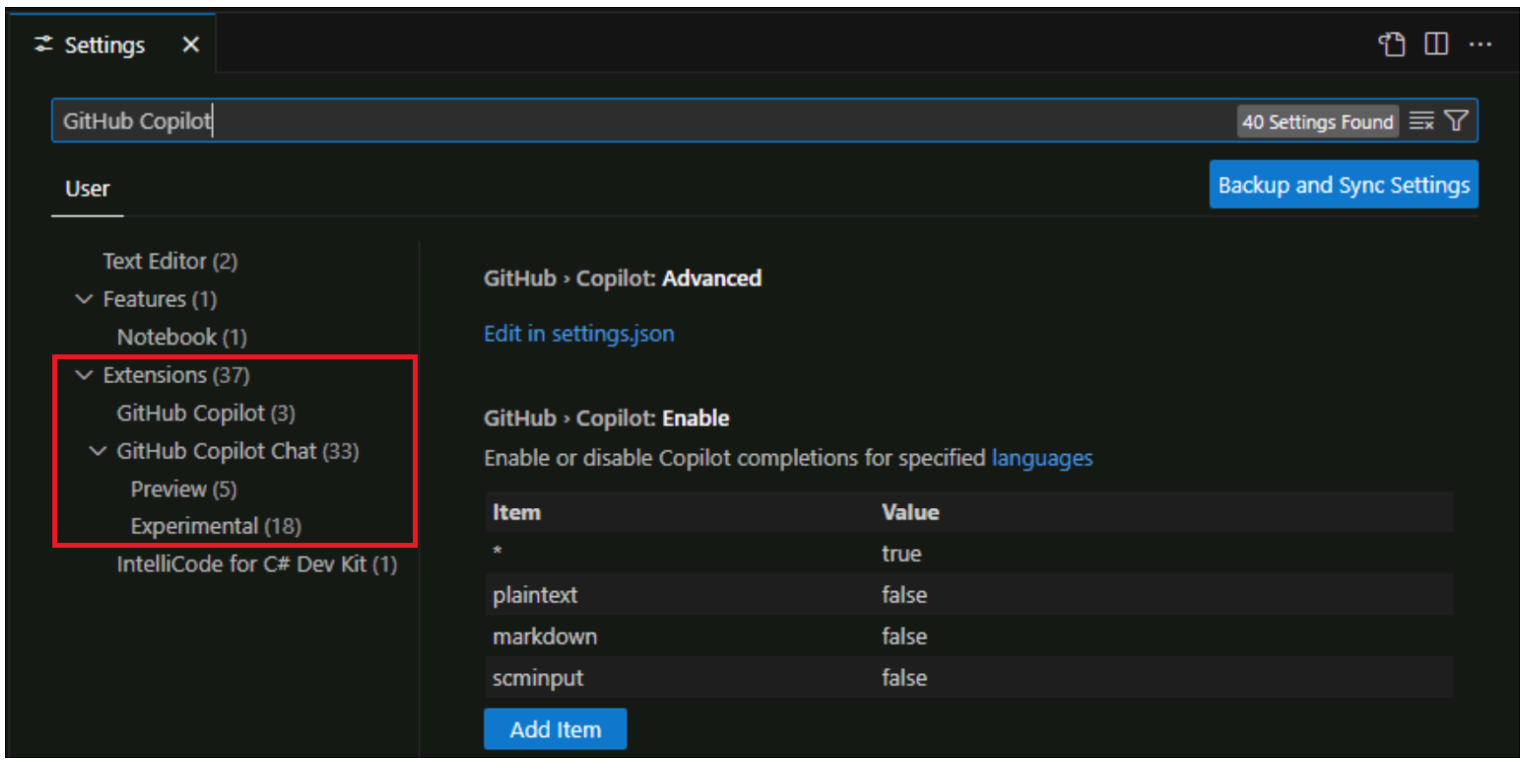
1. On Visual Studio Code's top menu bar, open the Copilot Chat menu.

   The Copilot Chat menu includes a **Configure Code Completions** option that provides access to a filtered list of GitHub Copilot settings.


Search
Open Chat Ctrl+Alt+I
Editor Inline Chat Ctrl+I
Quick Chat Ctrl+Shift+Alt+L
Configure Code Completions...
Manage Copilot

2. On the Copilot Chat menu, select **Configure Code Completions**, and then select **Edit Settings**.

3. Take a moment to review how the GitHub Copilot settings are organized.



Settings

GitHub Copilot | 40 Settings Found

User | Backup and Sync Settings

- Text Editor (2)
- Features (1)
  - Notebook (1)
- Extensions (37)
  - GitHub Copilot (3)
  - GitHub Copilot Chat (33)
    - Preview (5)
    - Experimental (18)
  - IntelliCode for C# Dev Kit (1)

GitHub › Copilot: **Advanced**

Edit in settings.json

GitHub › Copilot: **Enable**

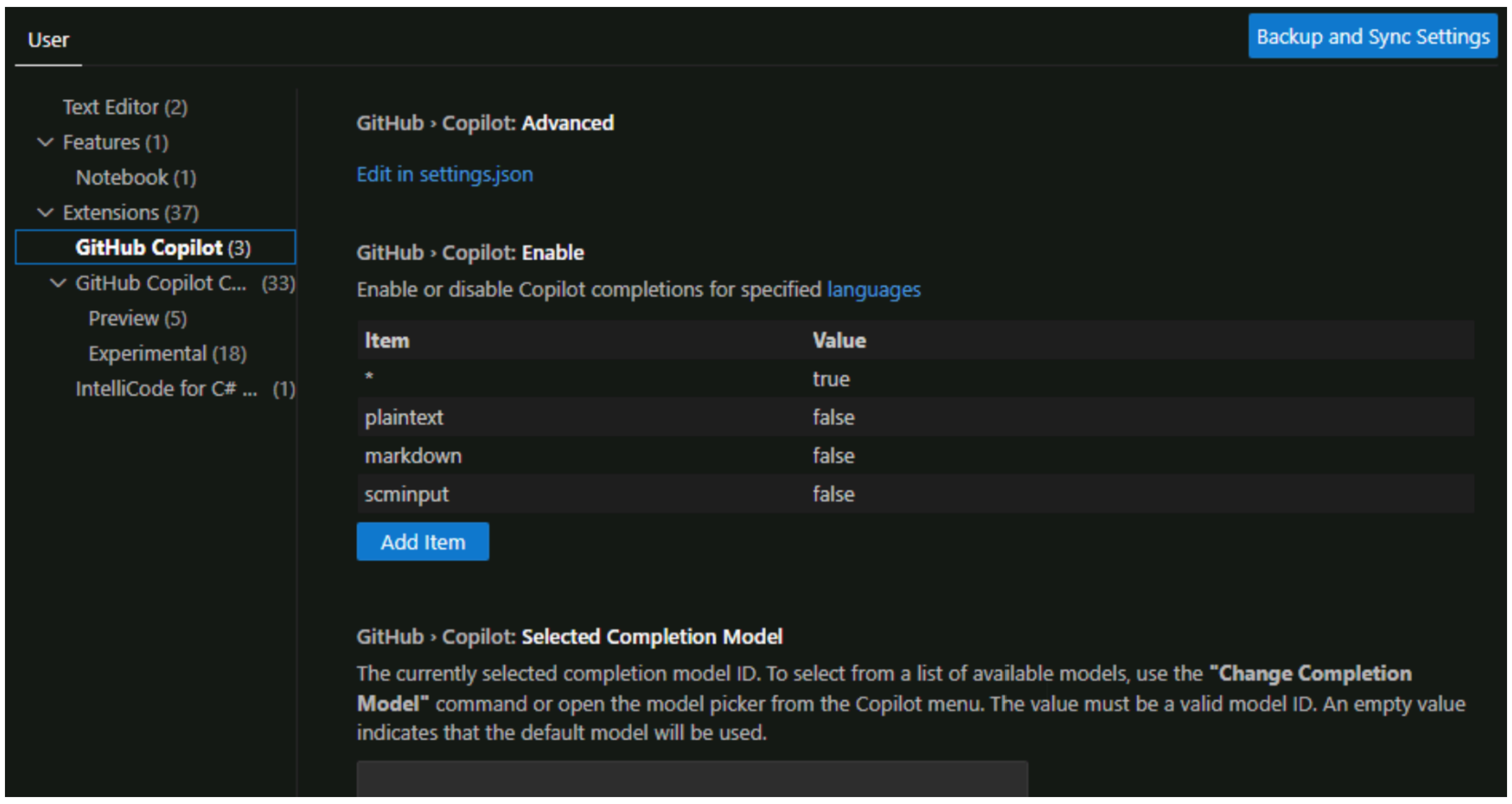
Enable or disable Copilot completions for specified languages

Item	Value
*	true
plaintext	false
markdown	false
scminput	false

Add Item

Notice that the settings for *GitHub Copilot* and *GitHub Copilot Chat* are separated. Also notice the *Preview* and *Experimental* categories.

4. Under the Extensions label, select **GitHub Copilot**.



User | Backup and Sync Settings

- Text Editor (2)
- Features (1)
  - Notebook (1)
- Extensions (37)
  - **GitHub Copilot** (3)
  - GitHub Copilot C... (33)
    - Preview (5)
    - Experimental (18)
  - IntelliCode for C# ... (1)

GitHub › Copilot: **Advanced**

Edit in settings.json

GitHub › Copilot: **Enable**

Enable or disable Copilot completions for specified languages

Item	Value
*	true
plaintext	false
markdown	false
scminput	false

Add Item

GitHub › Copilot: **Selected Completion Model**

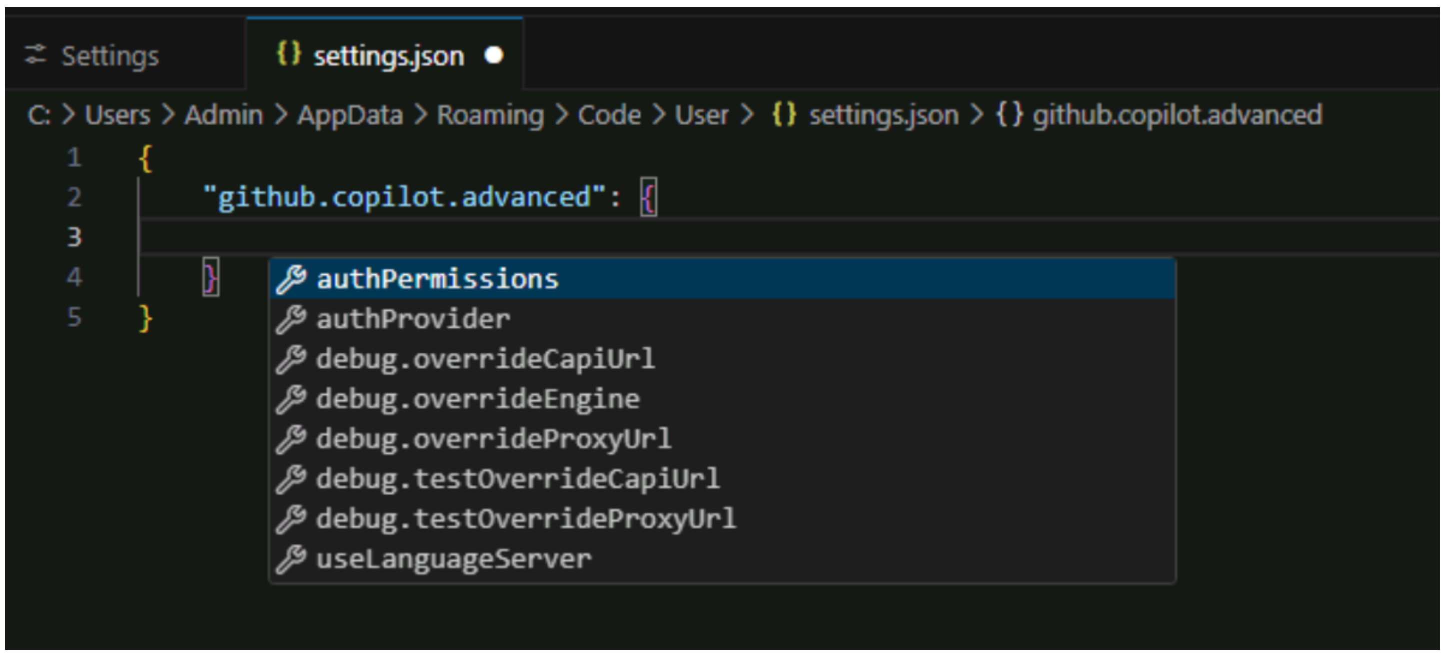
The currently selected completion model ID. To select from a list of available models, use the **"Change Completion Model"** command or open the model picker from the Copilot menu. The value must be a valid model ID. An empty value indicates that the default model will be used.

Notice that the settings list is now filtered for GitHub Copilot only.

The GitHub Copilot extension has the following setting options:

- **GitHub.Copilot.Advanced**: Configure advanced settings for GitHub Copilot in settings.json.

  Selecting **Edit in settings.json** opens the settings.json file, which is a JSON file that contains the settings for GitHub Copilot. The settings are organized into sections, and each section contains a list of settings.



Settings | settings.json

C: > Users > Admin > AppData > Roaming > Code > User > settings.json > github.copilot.advanced

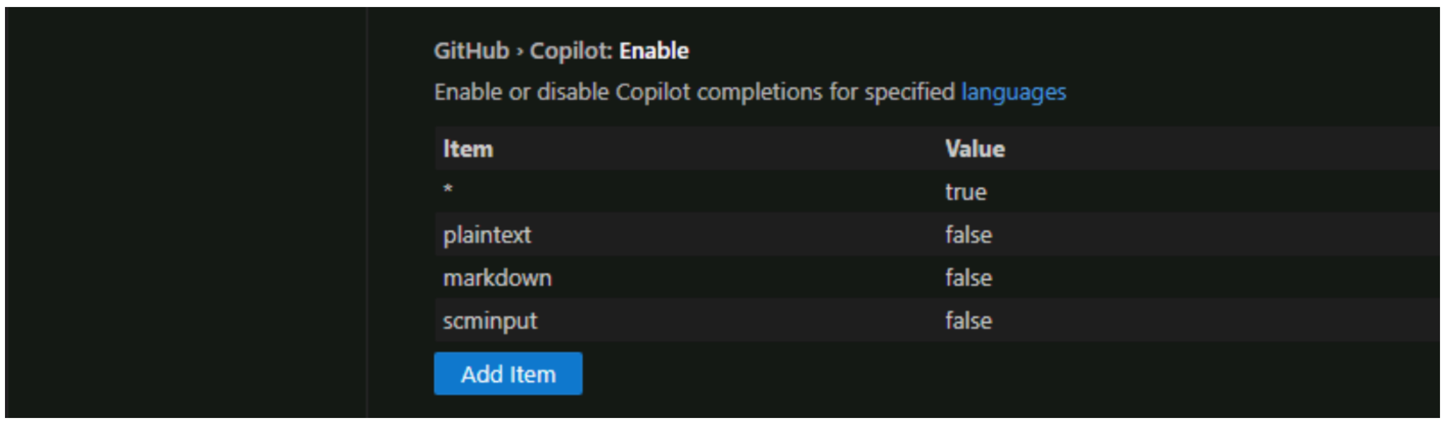
```
{
    "github.copilot.advanced": {

    }
}
```

- authPermissions
- authProvider
- debug.overrideCapiUrl
- debug.overrideEngine
- debug.overrideProxyUrl
- debug.testOverrideCapiUrl
- debug.testOverrideProxyUrl
- useLanguageServer

- **GitHub.Copilot.Enable**: Enable or disable Copilot completions for specified languages.

  The term *languages* refers to programming languages, but accommodates other file formats as well. The languages are specified using a list of languages and a value of **true** or **false** to enable or disable GitHub Copilot for each language. By default, GitHub Copilot is enabled for all languages. This setting is specified with the wildcard character * on the first row and the value **true**. The subsequent rows specify languages for which GitHub Copilot is enabled or disabled. For example, GitHub Copilot is enabled for **C#**, **JavaScript**, and **Python** and disabled for **Plaintext** and **Markdown**.



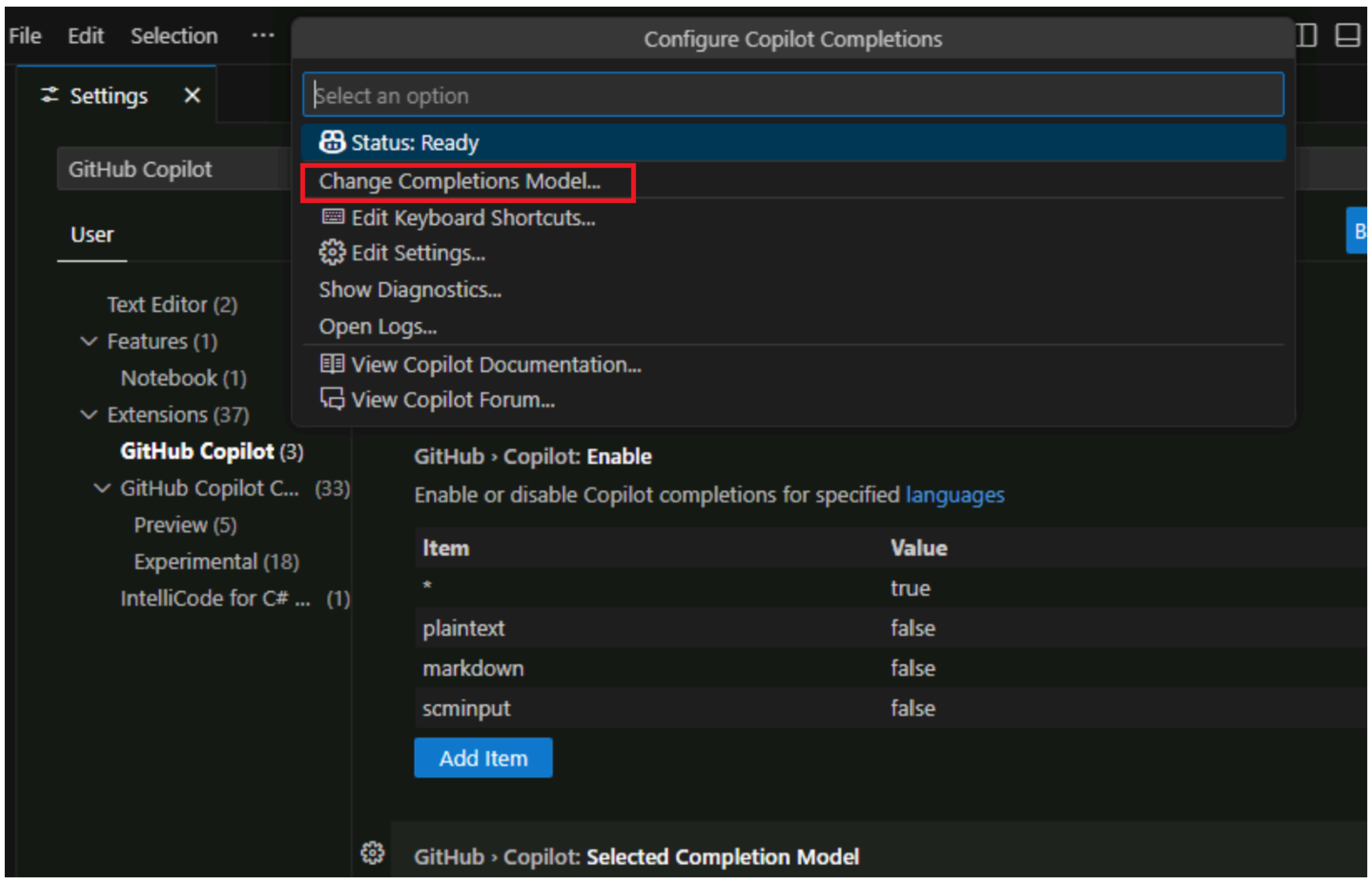
GitHub › Copilot: **Enable**
Enable or disable Copilot completions for specified languages

Item	Value
*	true
plaintext	false
markdown	false
scminput	false

Add Item

- **GitHub.Copilot.SelectedCompletionModel**: Select the model to use for Copilot completions.

  To select a completions model, open the Configure Copilot Completions menu, and then select the **Change Completions Model** menu option.



Configure Copilot Completions

Select an option

- Status: Ready
- Change Completions Model...
- Edit Keyboard Shortcuts...
- Edit Settings...
- Show Diagnostics...
- Open Logs...
- View Copilot Documentation...
- View Copilot Forum...

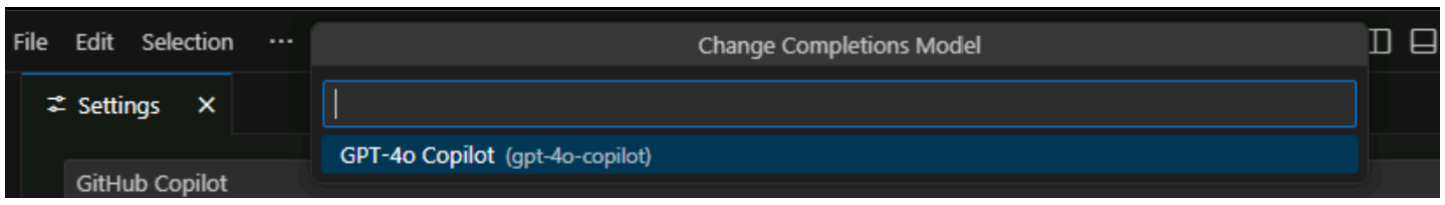
GitHub › Copilot: **Enable**
Enable or disable Copilot completions for specified languages

Item	Value
*	true
plaintext	false
markdown	false
scminput	false

Add Item

GitHub › Copilot: **Selected Completion Model**

The list of models may be limited by your GitHub Copilot subscription, the models available in your region, and the models available for your organization.



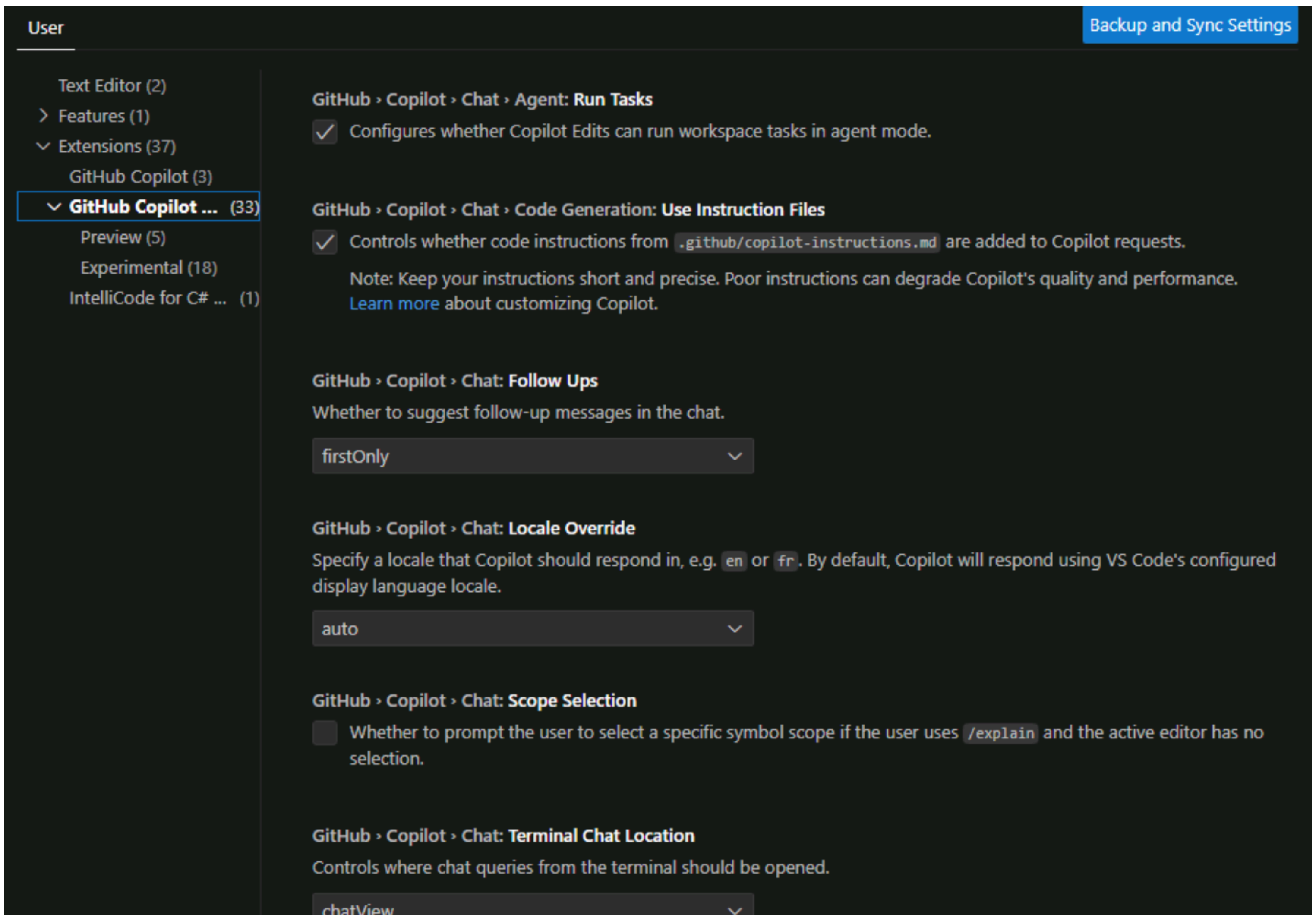
5. Under **Enable or disable Copilot completions for specified languages**, select **markdown**.

   The default value for Markdown is set to **false**. This means that GitHub Copilot completions are disabled for Markdown files.

6. To enable Copilot for Markdown files, select **Edit Item** (pencil icon), select **false**, change the value to **true**, and then select **OK**.

   You can now use GitHub Copilot to help you author or update Markdown files. For example, GitHub Copilot can generate code completion suggestions when you're working on project documentation.

7. Under the Extensions label, select GitHub Copilot Chat.



GitHub › Copilot › Chat › Agent: **Run Tasks**
✓ Configures whether Copilot Edits can run workspace tasks in agent mode.

GitHub › Copilot › Chat › Code Generation: **Use Instruction Files**
✓ Controls whether code instructions from `.github/copilot-instructions.md` are added to Copilot requests.
Note: Keep your instructions short and precise. Poor instructions can degrade Copilot's quality and performance. Learn more about customizing Copilot.

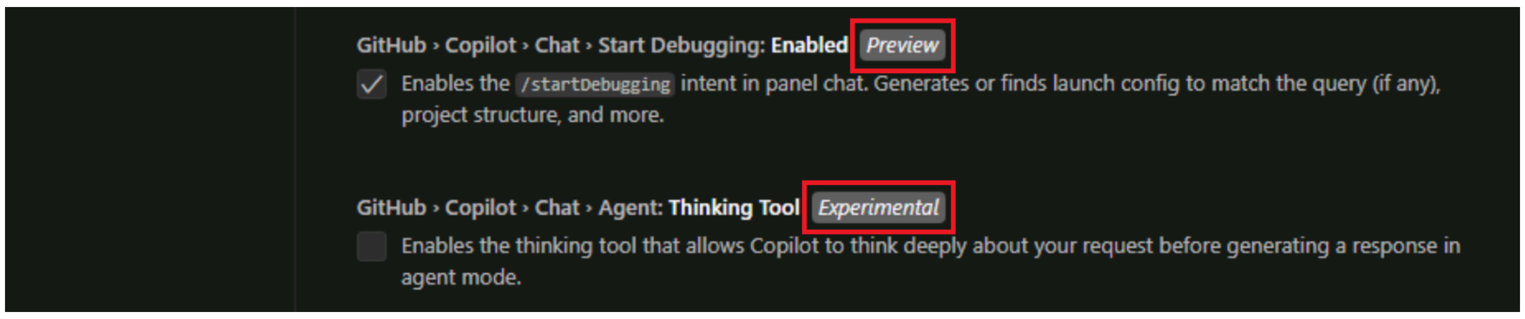
GitHub › Copilot › Chat: **Follow Ups**
Whether to suggest follow-up messages in the chat.
firstOnly

GitHub › Copilot › Chat: **Locale Override**
Specify a locale that Copilot should respond in, e.g. `en` or `fr`. By default, Copilot will respond using VS Code's configured display language locale.
auto

GitHub › Copilot › Chat: **Scope Selection**
Whether to prompt the user to select a specific symbol scope if the user uses `/explain` and the active editor has no selection.

GitHub › Copilot › Chat: **Terminal Chat Location**
Controls where chat queries from the terminal should be opened.
chatView

The GitHub Copilot Chat extension has a long list of available settings and it's updated regularly. The GitHub Copilot Chat extension also includes preview and experimental settings that are subject to change and could be discontinued. The preview and experimental settings are included at the end of the list and they're tagged as either **Preview** or **Experimental**.



GitHub › Copilot › Chat › Start Debugging: **Enabled** Preview
✓ Enables the `/startDebugging` intent in panel chat. Generates or finds launch config to match the query (if any), project structure, and more.

GitHub › Copilot › Chat › Agent: **Thinking Tool** Experimental
Enables the thinking tool that allows Copilot to think deeply about your request before generating a response in agent mode.

8. Take a few minutes to review the settings for GitHub Copilot Chat.

   We recommend keeping the default settings during this training. This helps to ensure that you have the expected experience when working through the exercises. When you have completed the training, you can experiment with these settings to customize your experience with GitHub Copilot and Copilot Chat.

9. Close the settings tab.

Explore the GitHub Copilot user interface in Visual Studio Code

Visual Studio Code seamlessly integrates GitHub Copilot's AI features into your development environment.

GitHub Copilot features align to the following categories:

- Natural language chat: GitHub Copilot provides three distinct chat interfaces: Chat view, Quick Chat, and Inline Chat.
- Code completions: GitHub Copilot integrates with the code editor to generate context-aware code completion suggestions as you type.
- Smart actions: GitHub Copilot automates common tasks with Smart actions to eliminate repetitive prompt writing.

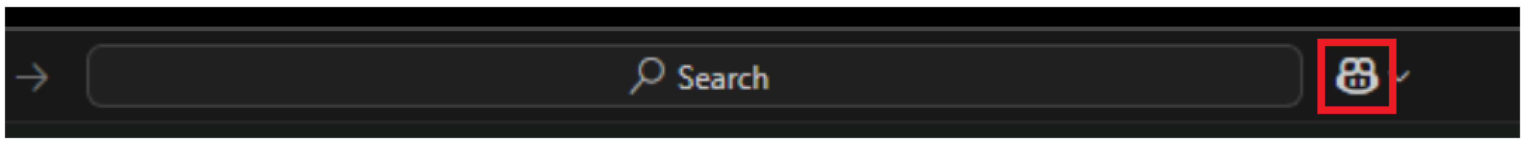
GitHub Copilot's productivity features are easy to access and fit seamlessly into your workflow without interrupting your coding experience.

Explore the Chat view features

Use the following steps to complete this section of the exercise:

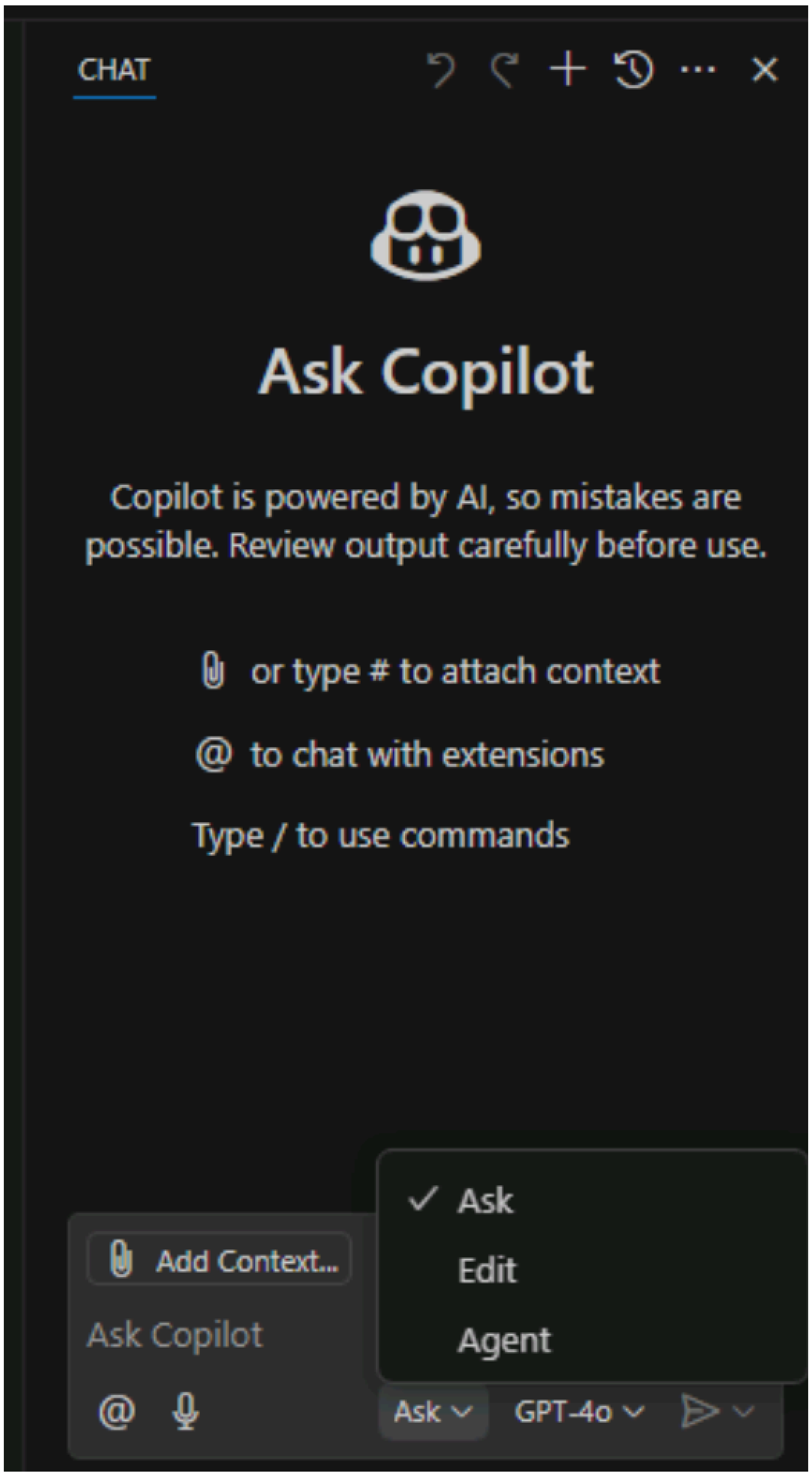
1. To open the Chat view, select the **Toggle Chat** button.

   The Toggle Chat button is located at the top of the Visual Studio Code window, just to the right of the search textbox.

   The GitHub Copilot Chat view opens in a side bar window on the right side of the Visual Studio Code window.

2. Take a few minutes to examine the Chat view interface.



CHAT

Ask Copilot

Copilot is powered by AI, so mistakes are possible. Review output carefully before use.

or type # to attach context

@ to chat with extensions

Type / to use commands

Starting from the top and moving down, the Chat view includes the following interface elements:

- Chat view toolbar: The Chat view toolbar is located in the top right corner of the Chat view. You can use the toolbar to manage the chat history, start a new chat, open the Chat view in another location, or hide the Chat view. Hover your mouse pointer over the toolbar button icons to see a description.
- Chat response area: The Chat response area is the space below the Chat view toolbar where GitHub Copilot displays responses. Responses include code suggestions, explanations, interactive elements, and other information related to your prompt.
- Add Context button: The Add Context button is located in bottom section of the Chat view. You can use this button search for resources that add context to Chat session. The resources can be anything from internal project files to public repositories on GitHub that are external to your organization.
- Ask Copilot: The Ask Copilot text box is where you enter your prompts. You can use the Ask Copilot text box to ask GitHub Copilot questions about your codebase, request code suggestions, or ask for help with specific tasks.
- Chat with Extension button: The Chat with Extension button is located below the Ask Copilot textbox. GitHub Copilot Extensions are a type of GitHub App that integrates the power of external tools into GitHub Copilot Chat. You can use this button to select from a list of Copilot Extensions.

The default extensions provide a combination of Chat participants and Slash commands that provide context for your prompts.

> **NOTE**: Anyone with a Copilot Pro, Copilot Pro+, or Copilot Free plan can use Copilot Extensions. For organizations or enterprises with a Copilot Business or Copilot Enterprise plan, organization owners and enterprise administrators can grant access to Copilot Extensions. Copilot Extensions is not available for GitHub Enterprise Server.

- Start Voice Chat button: The Start Voice Chat button is located below the Ask Copilot textbox. You can use this button to enable or disable voice interactions with GitHub Copilot. When voice interactions are enabled, you can use your voice to ask GitHub Copilot questions or request code suggestions.
- Chat Mode menu: The Chat Mode dropdown menu is located to the right of the Start Voice Chat button. Based on your specific needs, you can choose between different modes of chat:
  - **Ask**: Use this mode to ask GitHub Copilot questions about your codebase. You can ask GitHub Copilot to explain code, suggest changes, or provide information about the codebase.
  - **Edit**: Use this mode to edit code in your workspace. You can use GitHub Copilot to refactor code, add comments, or make other changes to your code.
  - **Agent**: Use this mode to run GitHub Copilot as an agent. You can use GitHub Copilot to run commands, execute code, or perform other tasks in your workspace.
- Pick Model menu: The Pick Model menu is located to the right of the Chat Mode menu. You can use this button to select the model that GitHub Copilot uses to generate code suggestions. The default mode is GPT-4o. Model selections may be limited based on your GitHub Copilot subscription, your GitHub Copilot settings, and the models available in your region.
- Send and Dispatch menu: The Send and Dispatch menu is located to the right of the Pick Model menu. You can use this button to submit your prompt to GitHub Copilot and receive a response. The menu includes several options for how your prompt is submitted.

3. Use the Ask Copilot textbox to enter the following prompt, and then submit the prompt:

   For C#:

   Code Copy

   ```
   Create a C# console app that prints Hello World to the console.
   ```

   Or for Python:

   Code Copy

   ```
   Create a Python console app that prints Hello World to the console.
   ```

4. Notice that GitHub Copilot's response includes an interactive list of files and a **Create Workspace** button.
5. To have GitHub Copilot create the code project, select **Create Workspace**.

   GitHub Copilot will prompt you for a parent folder before creating your workspace.

6. In the File dialog, select the **Desktop** folder, and then select **Select as Parent Folder**.
7. When prompted to open the created workspace, select **Open**.

   More advanced prompts can be used to create a workspace with multiple files or to create different project types. For example, you can use the following prompt specify additional conditions for the console app project:

   For C#:

Code Copy

```
Create a C# console app that prints Hello World to the console. Include a .gitignore file.
```

Or for Python:

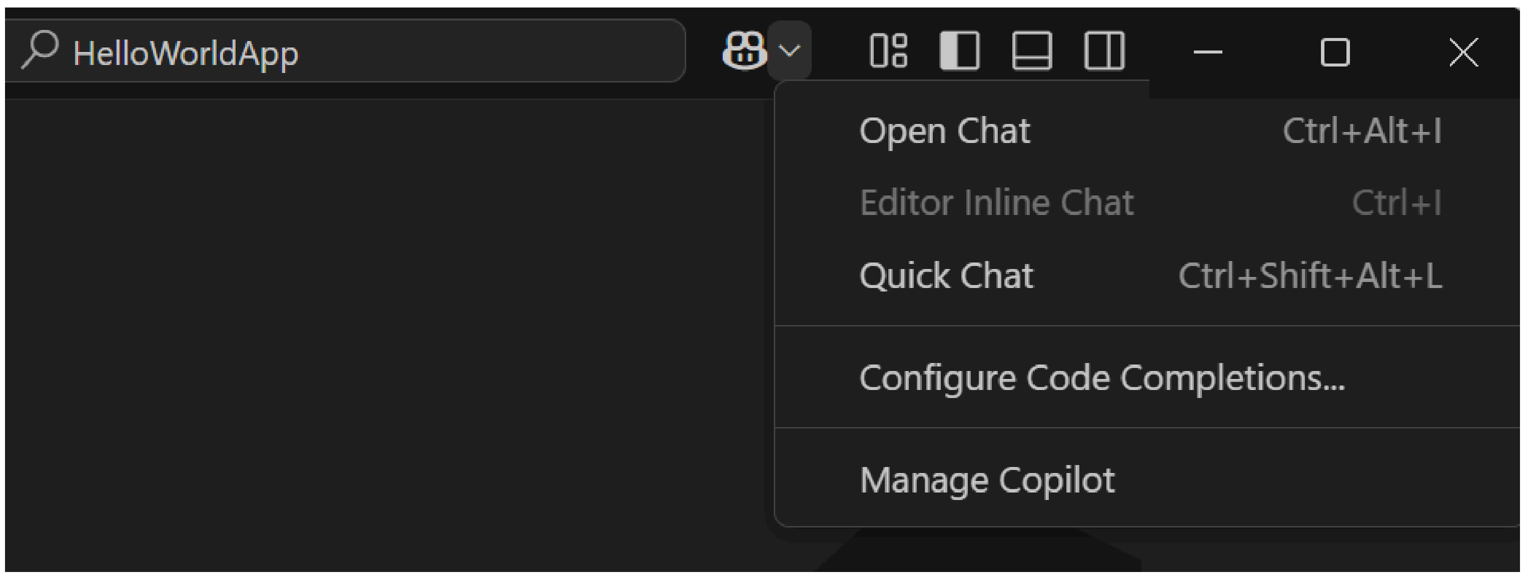
Code Copy

```
Create a Python console app that prints Hello World to the console. Include a .gitignore file.
```

Explore the Quick Chat features

The Quick Chat window is a simplified interface for interacting with GitHub Copilot. It provides a quick way to ask questions, request code suggestions, or get help with specific tasks without leaving the code editor.

1. Open the Copilot Chat menu and notice that three Chat options are listed.

   
   HelloWorldApp
   Open Chat Ctrl+Alt+I
   Editor Inline Chat Ctrl+I
   Quick Chat Ctrl+Shift+Alt+L
   Configure Code Completions...
   Manage Copilot

   The three Chat options are:

   - Open Chat: This option is used to open the Chat view.
   - Editor Inline Chat: This option is used to interact with GitHub Copilot directly in the code editor. The Inline Chat option is only enabled when there's a file open in the editor.
   - Quick Chat: This option is used to interact with GitHub Copilot using a simplified interface. You can use quick chat to ask questions, request code suggestions, or get help with specific tasks without leaving the code editor.

2. On the Copilot Chat menu, select **Quick Chat**.

   By default, the Quick Chat window opens at the top center of the Visual Studio Code window.

3. Notice that the Quick Chat window provides many of the same options provided with the Chat view.

4. Use the Quick Chat window to submit the following prompt:

   Code Copy

   ```
   Tell me about the Program.cs file
   ```

   Without additional context, the response is likely to be a description of how Program.cs files are used in various project types.

5. To add your Program.cs file to the Quick Chat context, drag-and-drop the Program.cs file from the Explorer view to the top of the Quick Chat window.

6. Notice that the Quick Chat window now includes **Program.cs** to the right of the **Add Context** button.

> **TIP**: Adding project files to the Chat context (Chat view, Quick Chat, or Inline Chat) helps GitHub Copilot provide more relevant suggestions. When adding project files to the Chat context, it’s often easier to use a drag-and-drop operation rather than the Add Context button.

7. Scroll to the top of the Quick Chat window and resubmit the same prompt:

   Code Copy
   ```
   Tell me about the Program.cs file
   ```

8. Notice that the new response describes the contents of your Program.cs file rather than a general description of how Program.cs files are used.

9. In the top-right corner of the Quick Chat window, select **Open in Chat View**.

   Notice that the Quick Chat window closes and the Chat view opens with responses that appeared the Quick Chat window. If the Chat view doesn’t display the Quick Chat session, use the Copilot Chat menu to open the Quick Chat window, and then select **Open in Chat View**.

   Switching to the Chat view is useful when you need to extend and manage a chat session that started in the Quick Chat window.

   > **TIP**: The Quick Chat window is great for quick questions and simple tasks. However, if you want a more dedicated Chat environment, you should use the Chat view. The Chat view provides a more comprehensive interface for interacting with GitHub Copilot and includes features like chat history, file management, and more advanced options for managing your chat sessions.

Explore the Inline Chat features

The Inline Chat interface is designed for code interactions directly within the editor. It provides a more focused and streamlined experience for working with code, allowing you to ask questions, request code suggestions, and make edits without leaving the code editor.

1. Open the Program.cs file in the editor.

2. Open the Copilot Chat menu and notice the following:

   - The Editor Inline Chat menu option is now enabled.
   - The **Ctrl+I** keyboard shortcut can be used to open an Inline Chat session.

3. In the code editor, select the following code:

   C# Copy
   ```csharp
   namespace HelloWorldApp
   {
       class Program
       {
           static void Main(string[] args)
           {
               Console.WriteLine("Hello World");
           }
       }
   }
   ```

4. To open an Inline Chat session, use the keyboard shortcut **Ctrl+I**.

5. Take a minute to examine the Inline Chat interface.

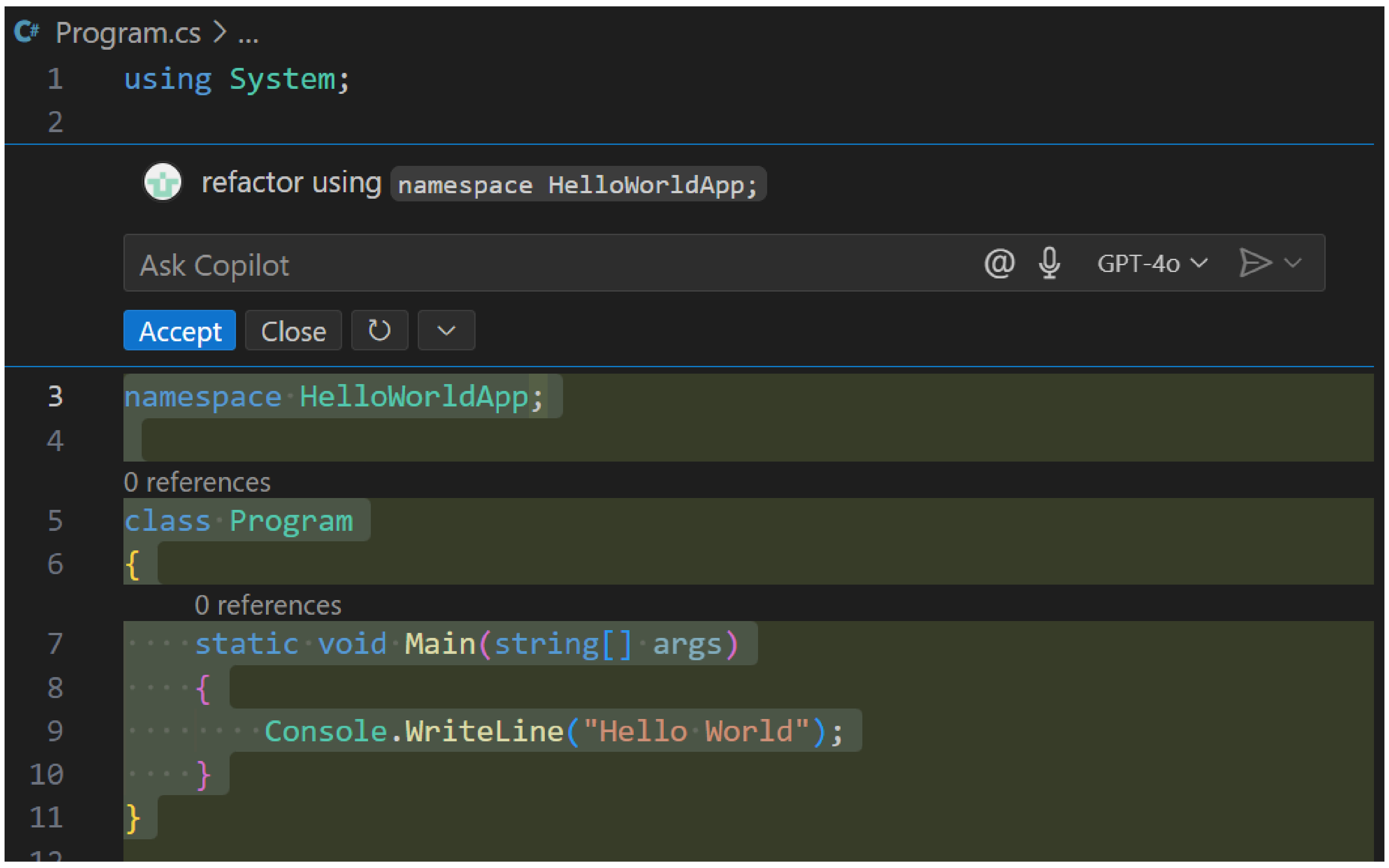
   The Inline Chat interface includes some of the features that you saw in the Chat view and Quick Chat interfaces, but it's designed for code interactions directly within the editor.

6. Enter the following prompt in the Ask Copilot textbox, and then submit the prompt:

   Code Copy

   ```
   refactor using `namespace HelloWorldApp;`
   ```

7. Take a minute to review the response generated by Inline Chat.

   ```
   using System;

   namespace HelloWorldApp;

   class Program
   {
       static void Main(string[] args)
       {
           Console.WriteLine("Hello World");
       }
   }
   ```

   Notice the following:

   - Inline Chat generates a suggestion by updating the code directly.
   - You can select **Accept** to implement the suggested update, select **Close** to reject the selected update, or select **Rerun** to generate a new suggestion.
   - There is also a **More Actions** menu that provides additional options for managing the Inline Chat session, and you can rate the suggestion as "helpful" or "unhelpful".

8. To accept the suggested code update, select **Accept**.

   The code is updated to use the file-scoped namespace syntax.

Compare the Chat view's Ask and Edit modes

The Chat view has three modes: **Ask**, **Edit**, and **Agent**. The Ask mode is designed for asking questions and generating code suggestions, while the Edit mode is designed for code refactoring and editing tasks. The Agent mode is designed for running commands and executing code.

1. Ensure that you have the Chat view open in the **Ask** mode.

2. Select the following code:

   C# Copy

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World");
}
```

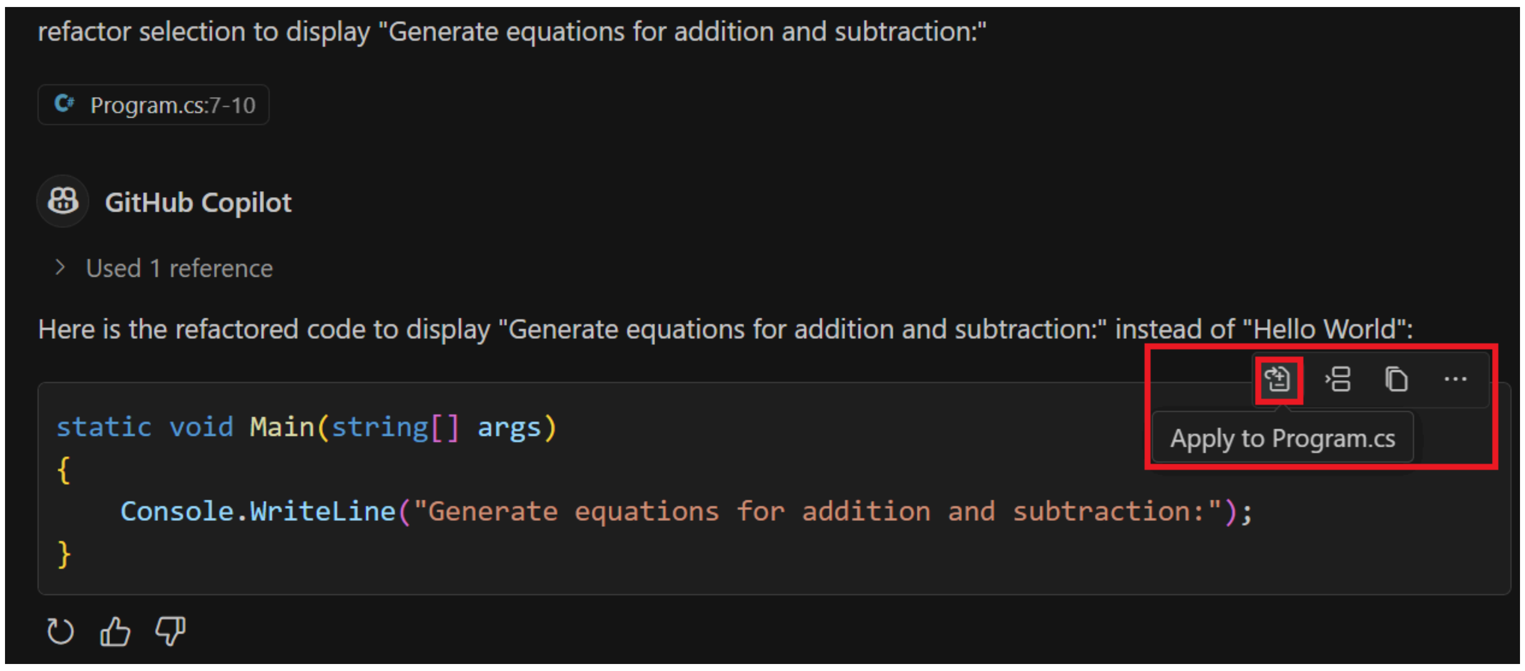
Notice that the Chat view context is updated to specify the selected code lines in the Program.cs file.

3. In the Chat view, enter the following prompt:

Code Copy

```
refactor selection to display "Generate equations for addition and subtraction:"
```

4. Take a minute to review response displayed in the Chat view.



refactor selection to display "Generate equations for addition and subtraction:"

Program.cs:7-10

GitHub Copilot

Used 1 reference

Here is the refactored code to display "Generate equations for addition and subtraction:" instead of "Hello World":

```
static void Main(string[] args)
{
    Console.WriteLine("Generate equations for addition and subtraction:");
}
```

Notice that when you hover the mouse pointer over the code, the Chat view displays the following options:

- **Apply to Program.cs**: Use the Apply to Program.cs option to apply the suggested code update to the Program.cs file.
- **Insert at Cursor**: Use the Insert at Cursor option to insert the suggested code update at the current cursor position in the editor.
- **Copy**: Use the Copy option to copy the suggested code update to the clipboard.

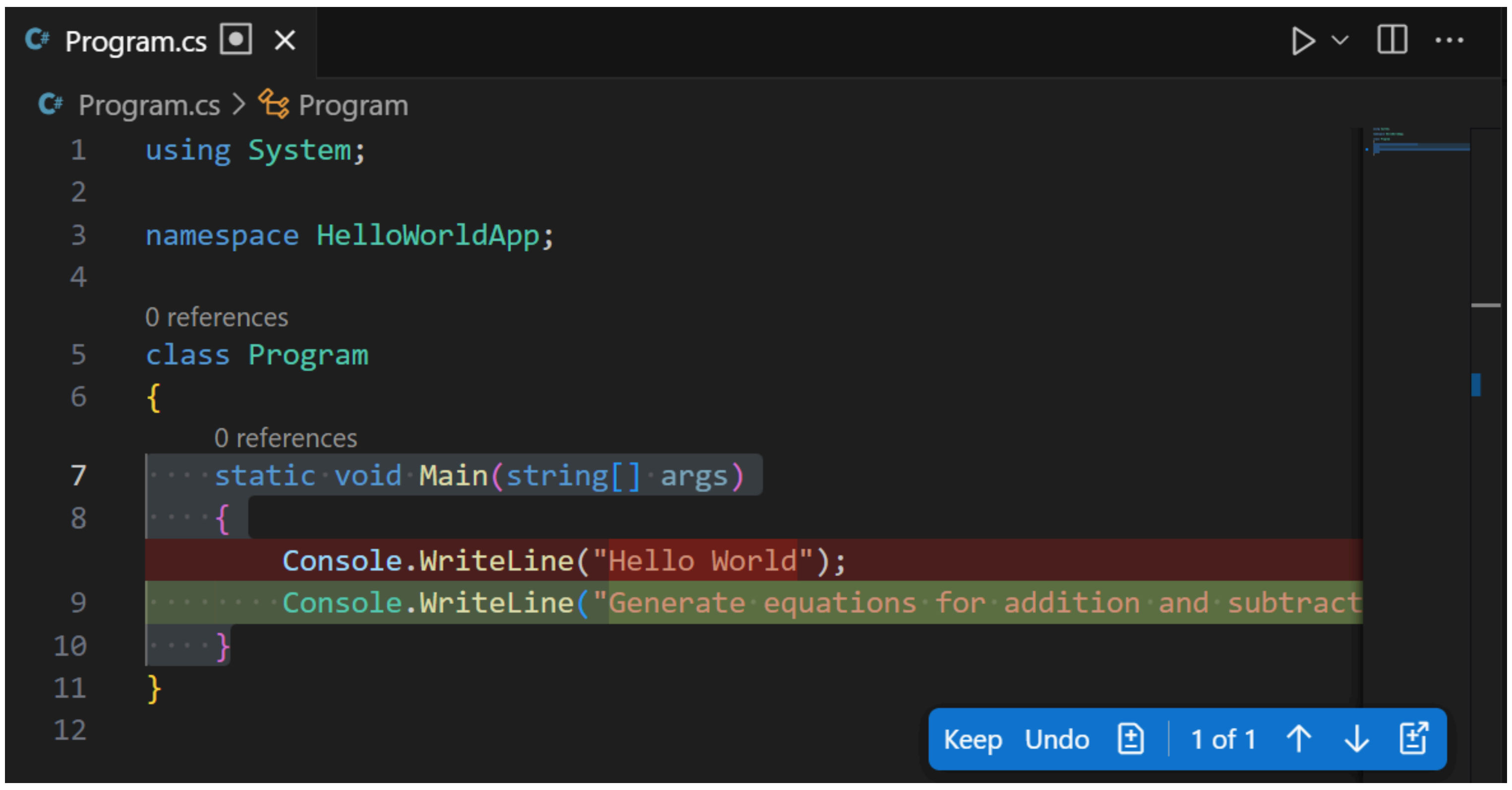
5. In the Chat view, select **Set Mode**, and then select **Edit**.

6. If a message is displayed telling you that changing the Chat mode terminates the current Chat session, select **Yes** to start a new Chat session, and then resubmit the following prompt:

Code Copy

```
refactor selection to display "Generate equations for addition and subtraction:"
```

7. Take a minute to review the updates suggested in the code editor.



```
using System;

namespace HelloWorldApp;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");
        Console.WriteLine("Generate equations for addition and subtract
    }
}
```

Notice the following:

- The code editor displays a *Diff-style* view that shows the changes made to the code, similar to the Diff view used in GitHub pull requests.
- The code editor displays **Keep** and **Undo** buttons that you can use to apply or reject the changes made to the code.
- The code editor displays additional buttons that can be used manage the suggested edits.

In addition to the edit controls displayed the editor tab, the Chat view displays a **Keep** button that you can use to apply all edits and an **Undo** button to cancel the edits, and an abbreviated description of the suggested update.

8. In the Chat view, select **Keep** to apply the suggested code updates.

NOTE: The Chat view's Agent mode is designed for running tasks using an autonomous agent. To conserve GitHub Copilot resources, the Agent mode isn't used in this exercise.

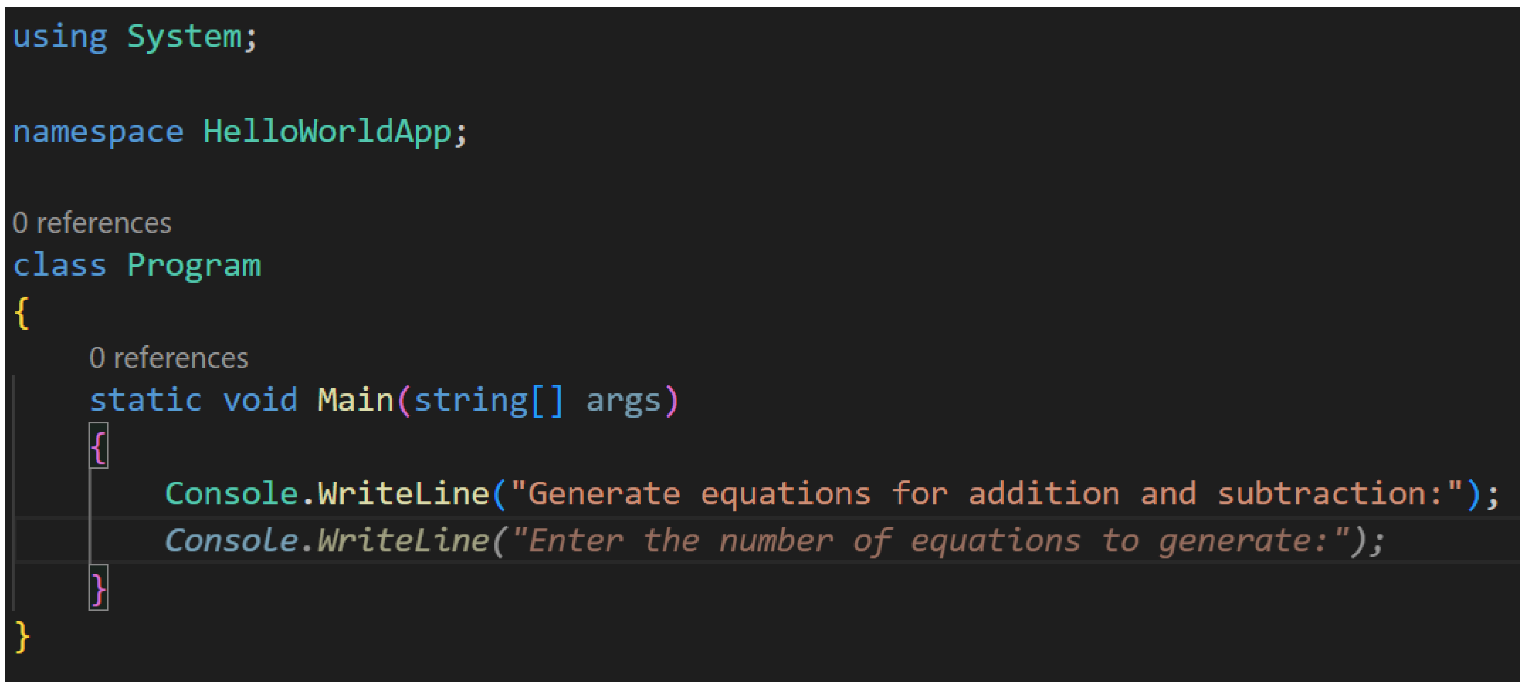
Explore code completion suggestions

GitHub Copilot provides code completion suggestions based on the context of the code in the editor. You can use code completion suggestions to quickly generate code snippets, refactor existing code, or implement new features.

1. With the Program.cs file open in the code editor, position the cursor at the end of the Console.WriteLine statement.

2. To generate a code completion suggestion, press **Enter**.

   GitHub Copilot generates a code completion suggestion based on the context of the code in the editor. In this case, GitHub Copilot suggests a new Console.WriteLine statement that's related to generating equations .



```
using System;

namespace HelloWorldApp;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Generate equations for addition and subtraction:");
        Console.WriteLine("Enter the number of equations to generate:");
    }
}
```

3. To accept the code completion suggestion, press **Tab**.

   The code in the editor is updated to include the new Console.WriteLine statement.

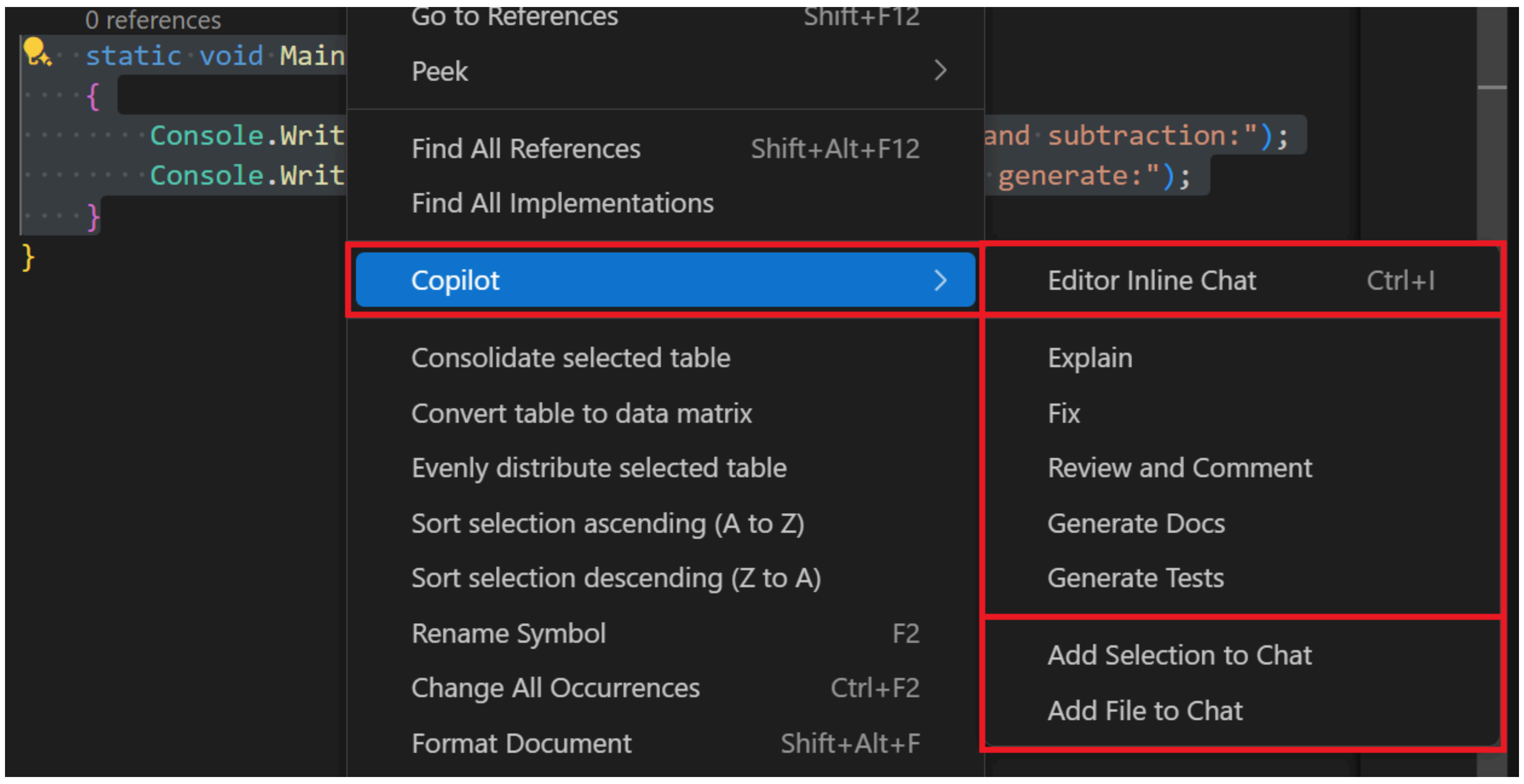
   When you accept a code completion suggestion, GitHub Copilot may suggest additional code lines. When this happens, you can press the **Tab** key to accept the suggestion, press the **Esc** key to reject the suggestion, or enter your own code to override the suggestion.

Access Smart Actions

Smart Actions are a set of predefined actions that are available from the Copilot context menu. You can use Smart Actions to quickly perform common tasks in Visual Studio Code without having to write prompts.

1. In the code editor, select the Main method, right-click the selected code, and then select **Copilot**.

   You should see a GitHub Copilot context menu that includes three sections.

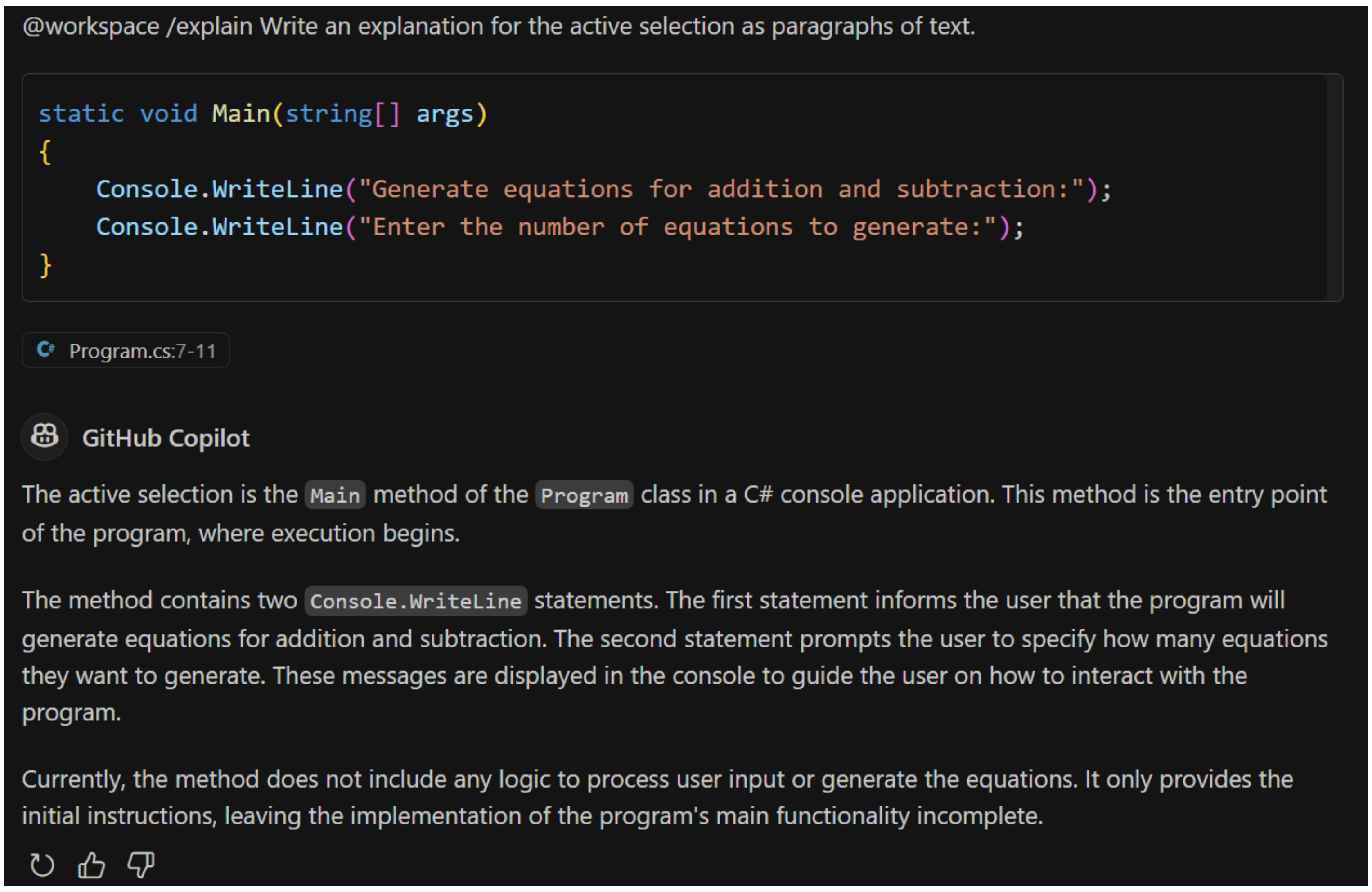

Go to References Shift+F12
Peek
Find All References Shift+Alt+F12
Find All Implementations
Copilot
Consolidate selected table
Convert table to data matrix
Evenly distribute selected table
Sort selection ascending (A to Z)
Sort selection descending (Z to A)
Rename Symbol F2
Change All Occurrences Ctrl+F2
Format Document Shift+Alt+F
Editor Inline Chat Ctrl+I
Explain
Fix
Review and Comment
Generate Docs
Generate Tests
Add Selection to Chat
Add File to Chat

   - The first section includes the option to open an Inline Chat session.
   - The second section includes a list of Smart Actions.
   - The third section includes options for adding context to a Chat session.

2. On the GitHub Copilot context menu, select **Explain**.

3. Take a minute to consider the results of the smart action.

   Notice that the Explain smart action constructs a prompt that's based on the code selection and submits the prompt in the Chat view.



@workspace /explain Write an explanation for the active selection as paragraphs of text.

```
static void Main(string[] args)
{
    Console.WriteLine("Generate equations for addition and subtraction:");
    Console.WriteLine("Enter the number of equations to generate:");
}
```

Program.cs:7-11

GitHub Copilot

The active selection is the `Main` method of the `Program` class in a C# console application. This method is the entry point of the program, where execution begins.

The method contains two `Console.WriteLine` statements. The first statement informs the user that the program will generate equations for addition and subtraction. The second statement prompts the user to specify how many equations they want to generate. These messages are displayed in the console to guide the user on how to interact with the program.

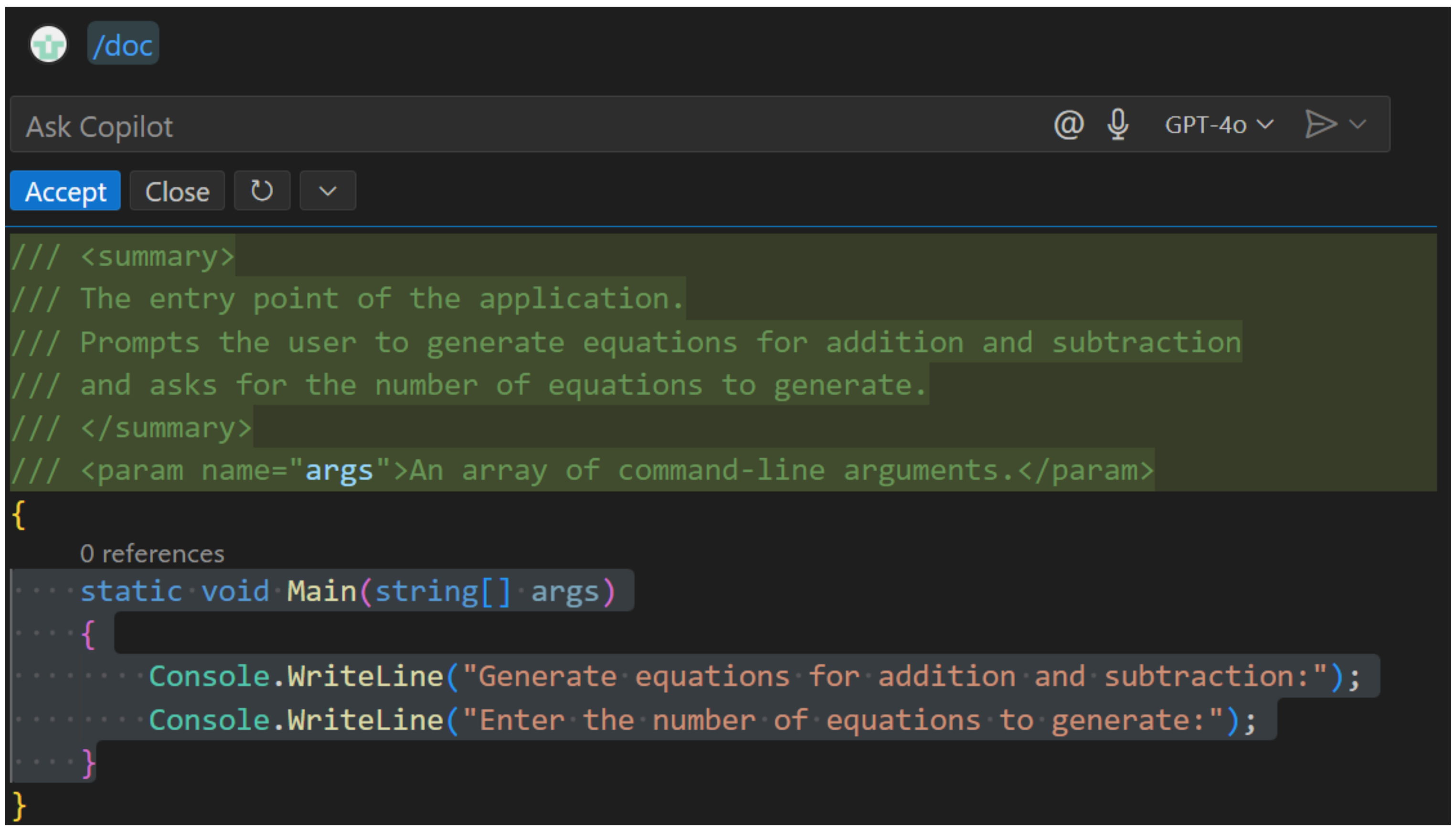
Currently, the method does not include any logic to process user input or generate the equations. It only provides the initial instructions, leaving the implementation of the program's main functionality incomplete.

The explanation includes a detailed description of the selected code, and may include suggested updates.

4. Open the GitHub Copilot context menu again, and then select **Generate Docs**.

5. Take a minute to consider the results of the smart action.

   Notice that the Generate Docs smart action opens an Inline Chat session and submits a prompt asking GitHub Copilot to suggest documentation for the selected code.



```
/// <summary>
/// The entry point of the application.
/// Prompts the user to generate equations for addition and subtraction
/// and asks for the number of equations to generate.
/// </summary>
/// <param name="args">An array of command-line arguments.</param>
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine("Generate equations for addition and subtraction:");
        Console.WriteLine("Enter the number of equations to generate:");
    }
}
```

6. In the Inline Chat window, select **Accept** to apply the suggested documentation.

Summary

In this exercise, you examined GitHub Copilot settings and explored the GitHub Copilot user interface in Visual Studio Code. You learned how to configure GitHub Copilot settings and you explored the Chat view, Quick Chat, and Inline Chat features. You also used GitHub Copilot to generate code completions and used Smart Actions to complete common tasks.

Clean up

Now that you’ve finished the exercise, take a minute to ensure that you haven’t made changes to your GitHub account or GitHub Copilot subscription that you don’t want to keep. If you made any changes, revert them now.