

# AZ-2007

# GitHub Copilot を使用して アプリ開発を高速化する

本講義は以下のMicrosoft Learn教材に準拠しています。

[GitHub Copilot を使用してアプリ開発を高速化する - Training | Microsoft Learn](#)



GitHub Copilot を使用してアプリ開発を高  
速化する

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する
- ・補足: GitHub Copilotの最新機能のご紹介

2025/2/6にプレビューとなった  
Next edit suggestions (NES) と agent modeを  
ご紹介

2024/4/30にテクニカルプレビューとなった  
GitHub Copilot Workspaceをご紹介

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する
- ・補足: GitHub Copilotの最新機能のご紹介

# モジュール1



## GitHub Copilot の概要

10 分 残り • モジュール • 8/9 ユニットが完了しました

このモジュールでは、GitHub Copilot 製品、GitHub Copilot が開発者に提供する利点、GitHub Copilot と GitHub Copilot Chat の製品機能、Visual Studio Code 用の GitHub Copilot 拡張機能について説明します。

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ



# GitHub (ギットハブ) とは？

- ・ソフトウェア開発のプラットフォーム
- ・インターネット上での多数の開発者とのコラボレーションが容易
- ・Git (バージョン管理ツール) を使用してソースコードのバージョン管理を行う
- ・2018/6/4, マイクロソフトはGitHubの買収を発表、同10月に買収を完了。現在GitHubはMicrosoft傘下となっている。

## ■ GitHubのトップページ <https://github.com/>

The screenshot shows the GitHub homepage with a dark blue background. At the top, there is a banner with the text "GitHub Copilot is now available for free. [Learn more](#)". Below the banner is the GitHub logo and a "Sign in" button. The main headline reads "Build and ship software on a single, collaborative platform". Below the headline, a subtext says "Join the world's most widely adopted AI-powered developer platform." There are three buttons at the bottom: a white button with "Enter your email", a green button with "Sign up for GitHub", and a dark blue button with "Try GitHub Copilot". A yellow callout box at the bottom left contains the Japanese text "GitHubの利用をこれから開始する場合、ここからアカウントを作成できる". A small circular arrow icon with an upward arrow is in the bottom right corner.

GitHub Copilot is now available for free. [Learn more](#)

Sign in

# Build and ship software on a single, collaborative platform

Join the world's most widely adopted AI-powered developer platform.

Enter your email

Sign up for GitHub

Try GitHub Copilot

GitHubの利用をこれから開始する場合、  
ここからアカウントを作成できる

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilot とは？

- ・プログラマのコードの記述作業を支援する機能
- ・2021/6/29 プレビュー開始、2022/6/22 正式リリース
- ・「AIペアプログラマ」である
  - ・人間のプログラマのコード記述をAIが助けてくれる
  - ・プログラマが効率的にコードを記述するのに役立つ
- ・Visual Studio Codeなど、さまざまな開発ツールと連携が可能
- ・**利用するにはGitHubアカウントが必要**

<https://github.blog/jp/2022-06-22-github-copilot-is-generally-available-to-all-developers/>

<https://www.itmedia.co.jp/news/articles/2106/30/news063.html>

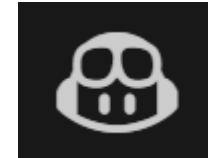
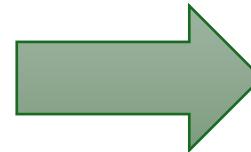
# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHubのCopilotのマスコットキャラ 「Copilot」

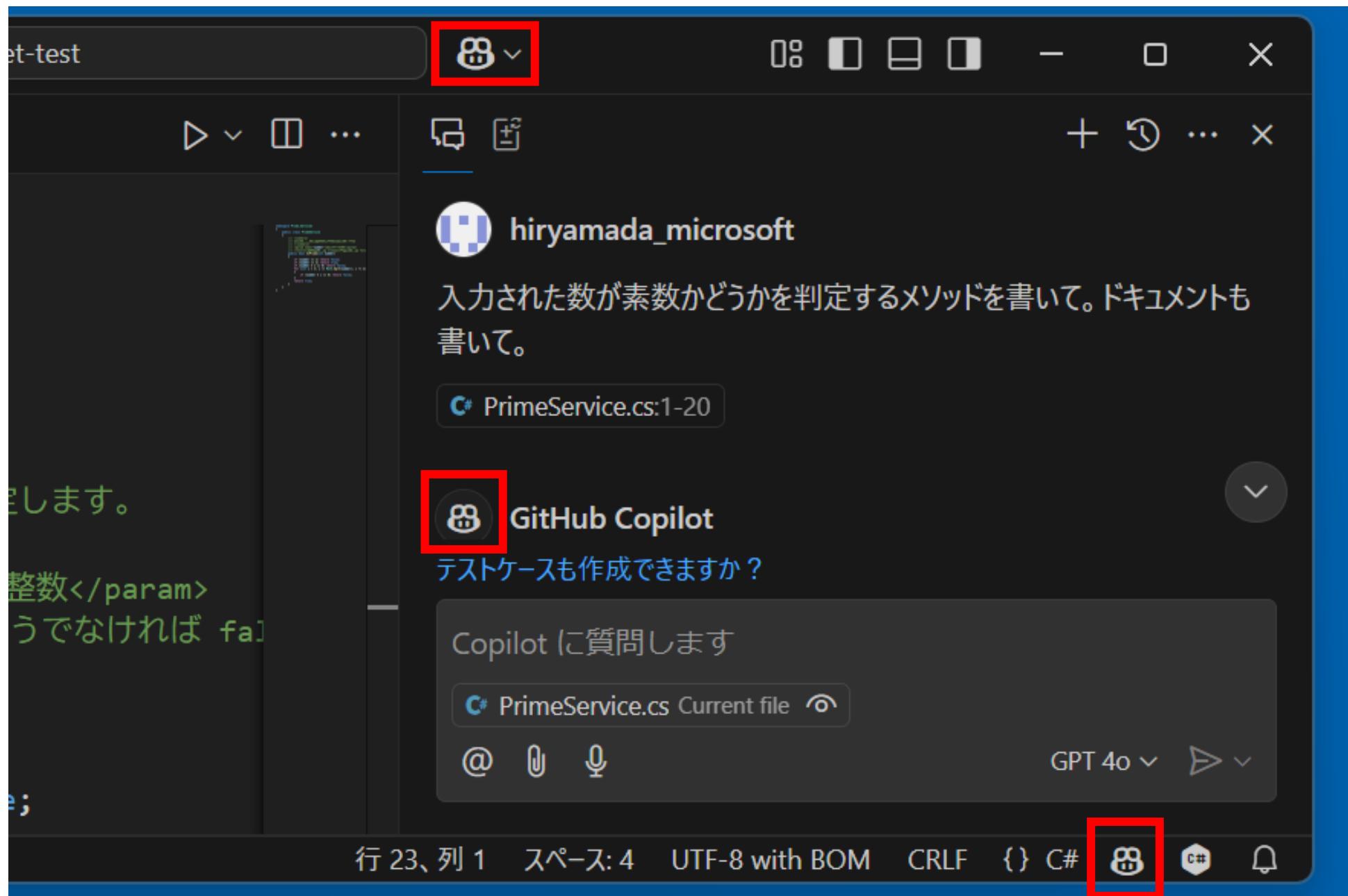


ヘルメットとゴーグルを装着した  
「コパイロット」（副操縦士）の  
イメージ



開発ツールなどに  
表示される白黒アイコン

## ■Visual Studio Codeの中に表示されるCopilotアイコンの例



# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilotは無料で利用可能

- 2024/12/18～ **GitHub Copilot Free** が利用可能になった
- 個人のGitHubユーザーで、組織や企業を通じてCopilotを利用していない方が対象
- 利用制限がある
  - 1ヶ月に 2,000 コード補完が利用可能
  - 1ヶ月に 50 チャットメッセージが利用可能
- 制限なく利用したい場合は GitHub Copilot Pro （月額10ドル）などを契約

# 参考: GitHub Copilotのサブスクリプションプラン

- GitHub Copilot Free
  - 個人ユーザー向け。無料
- GitHub Copilot Pro
  - 個人のプロフェッショナル開発者向け。10ドル/月
  - (以前は「GitHub Copilot **Individual**」と呼ばれていた)
- GitHub Copilot Business
  - チームや小規模な組織向け。19ドル/ユーザー/月
- GitHub Copilot Enterprise
  - 大規模な企業向け。39ドル/ユーザー/月

[GitHub Copilot のサブスクリプションプラン - GitHub Docs](#)

[GitHub Copilot Enterprise の概要 - Training | Microsoft Learn](#)

詳しくはこちらの  
ページを参照

## ■GitHub Copilot のサブスクリプションの選択画面

Pricing

# Take flight with GitHub Copilot

**New**

**Free**  
For developers looking to get started with GitHub Copilot.

**\$0 USD**

**Get started >**

Already have  VS Code? [Open now](#)

Includes up to 2,000 completions and 50 chat requests per month.

**Pro**  
For developers who want unlimited access to GitHub Copilot.

**\$10 USD**  
per month. First 30 days free.

**Get started >**

Free for verified students, teachers, and maintainers of popular open source projects. [Learn more](#)

**Popular for teams**

**Business**  
For teams ready to accelerate their workflows with GitHub Copilot.

**\$19 USD**  
per user / month

**Get started >**

**Contact sales >**

**Enterprise**  
For organizations seeking a fully customized GitHub Copilot experience.

**\$39 USD**  
per user / month

**Get started >**

**Contact sales >**

## ■GitHub Copilot の各サブスクリプションプランで提供される機能の違い (表の一部)

Compare features	Free \$0	Pro \$10 per month	Business \$19 per user / month	Enterprise \$39 per user / month
<b>Chat</b>				
Messages and interactions	⟳ Up to 50 per month	♾️ Unlimited	♾️ Unlimited	♾️ Unlimited
Access to OpenAI GPT-4o	✓	✓	✓	✓
Preview	✓	✓	✓	✓
Access to Anthropic Claude 3.5 Sonnet	✓	✓	✓	✓
Preview	—	✓	✓	✓
Access to OpenAI o1	—	✓	✓	✓
Preview	—	✓	✓	✓
Access to OpenAI o1-mini	—	✓	✓	✓
Context-aware coding support and explanations	✓	✓	✓	✓
Debugging and security remediation assistance	✓	✓	✓	✓

各サブスクリプションプランで  
利用できる機能の詳細を確認できる

## ■GitHub Copilot の各サブスクリプションプランで提供される機能の違い (表の一部)

Compare features	Free \$0	Pro \$10 per month	Business \$19 per user / month	Enterprise \$39 per user / month
<b>Chat</b>				
Messages and interactions	Up to 50 per month		たとえば、OpenAI o1 / o1-miniなどの最新モデルはPro以上で利用可能	
Access to OpenAI GPT-4o	✓			
<b>Preview</b>				
Access to Anthropic Claude 3.5 Sonnet	✓	✓	✓	✓
<b>Preview</b>				
Access to OpenAI o1	—	✓	✓	✓
<b>Preview</b>				
Access to OpenAI o1-mini	—	✓	✓	✓
Context-aware coding support and explanations	✓	✓	✓	✓
Debugging and security remediation assistance	✓	✓	✓	✓

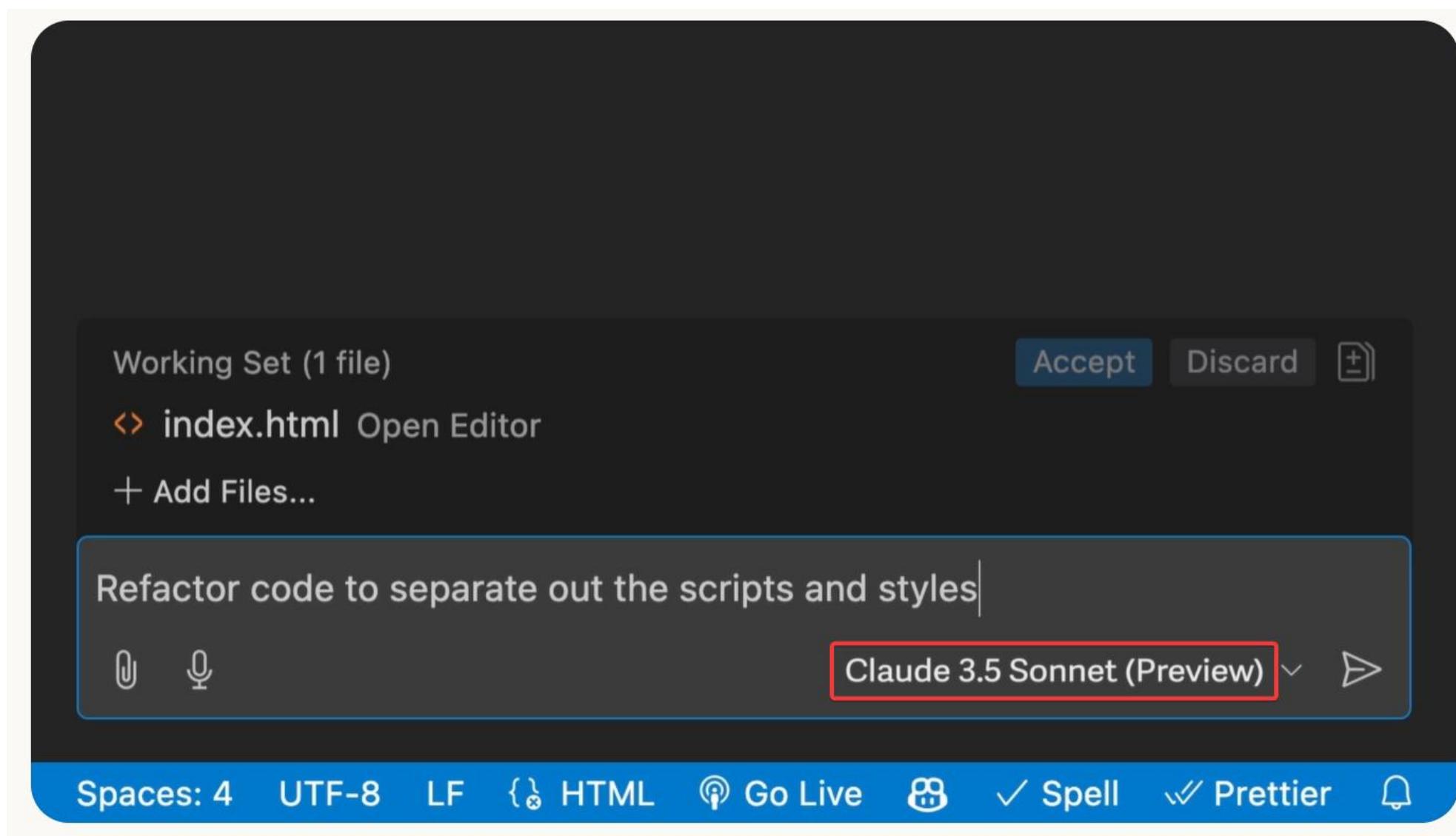
## ■GitHub Copilot の各サブスクリプションプランで提供される機能の違い (表の一部)

Compare features	Free \$0	Pro \$10 per month	Business \$19 per user / month	Enterprise \$39 per user / month
<b>Chat</b>				
Messages and interactions	Up to 50 per month	Unlimited	Unlimited	Unlimited
Access to OpenAI GPT-4o	✓	✓	✓	✓
Access to Anthropic Claude 3.5 Sonnet	✓	✓	✓	✓
Access to OpenAI o1	Freeでは、GPT-4oが利用できる			
Access to OpenAI o1-mini	—	✓	✓	✓
Context-aware coding support and explanations	✓	✓	✓	✓
Debugging and security remediation assistance	✓	✓	✓	✓

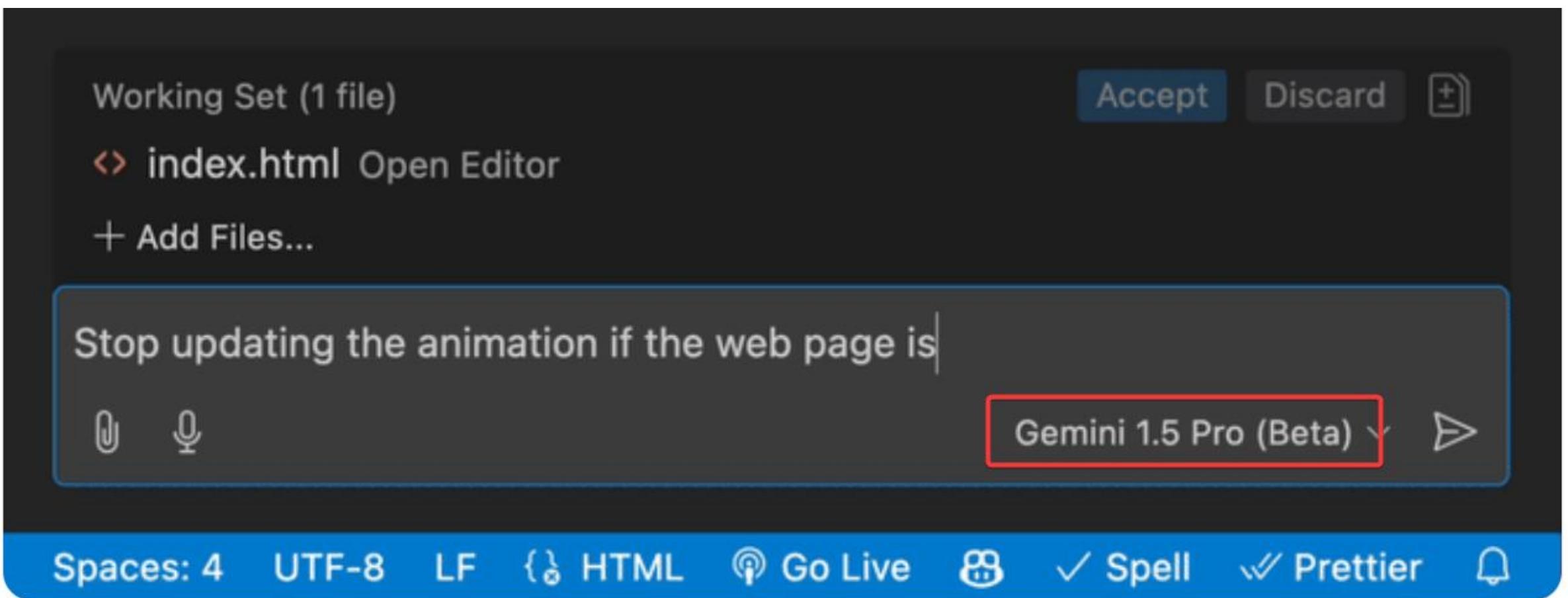
## ■GitHub Copilot の各サブスクリプションプランで提供される機能の違い (表の一部)

Compare features	Free \$0	Pro \$10 per month	Business \$19 per user / month	Enterprise \$39 per user / month
<b>Chat</b>				
Messages and interactions	⟳ Up to 50 per month	♾️ Unlimited	♾️ Unlimited	♾️ Unlimited
Access to OpenAI GPT-4o	✓	✓	✓	✓
Preview Access to Anthropic Claude 3.5 Sonnet	✓	✓	✓	✓
Preview Access to OpenAI o1	—	✓	✓	✓
Preview Access to OpenAI o1-mini	Anthropic Claude 3.5 Sonnet なども選択できる (現在プレビュー)		✓	✓
Context-aware coding support and explanations	✓	✓	✓	✓
Debugging and security remediation assistance	✓	✓	✓	✓

■ご参考: Anthropic の Clause 3.5 Sonnet などのモデルが利用可能に (2024/10/30~)



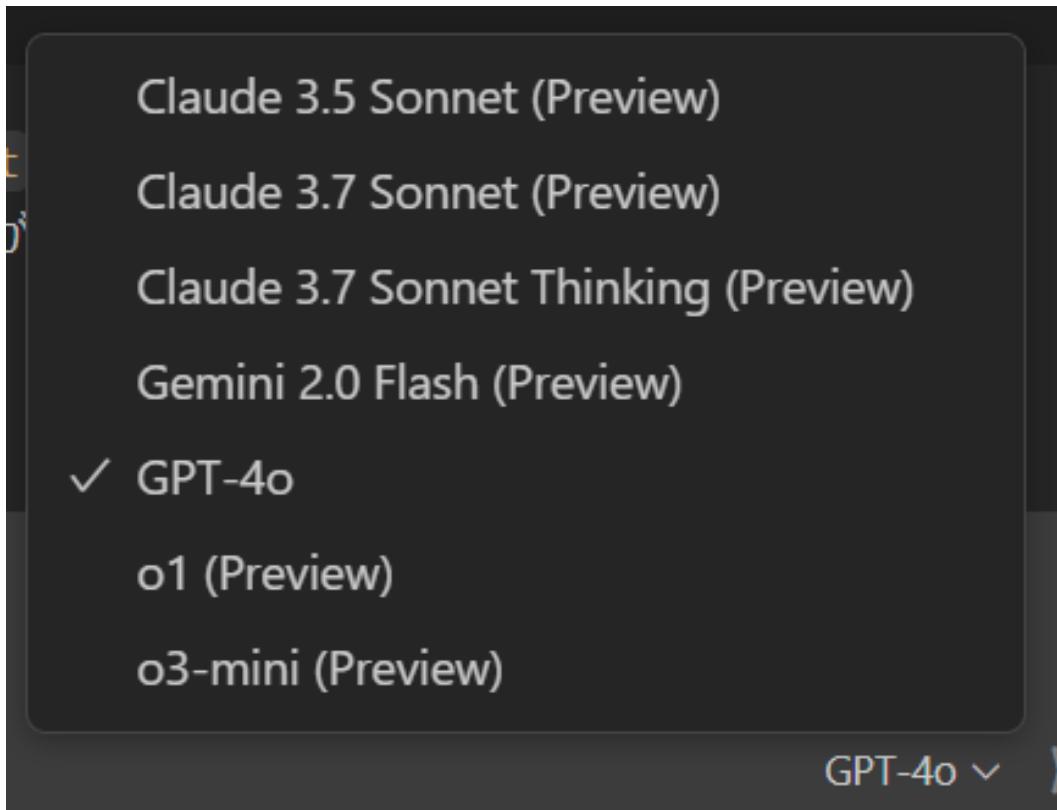
## ■ご参考: Google の Gemini 1.5 Pro モデルが利用可能に (2024/10/30~)



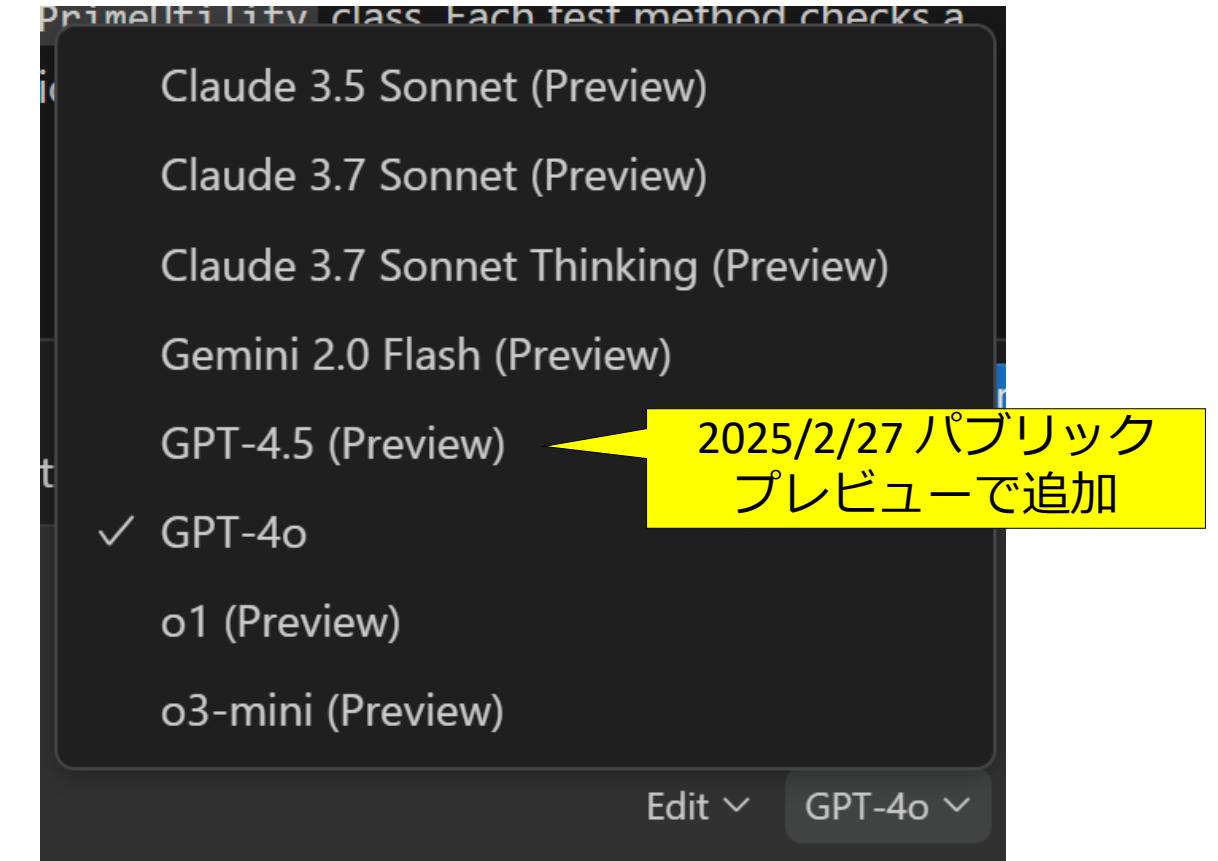
Developers will soon be able to select **Gemini 1.5 Pro** in GitHub Copilot's new model picker to assist in coding related use cases.

## ■ご参考: 選択可能なモデル

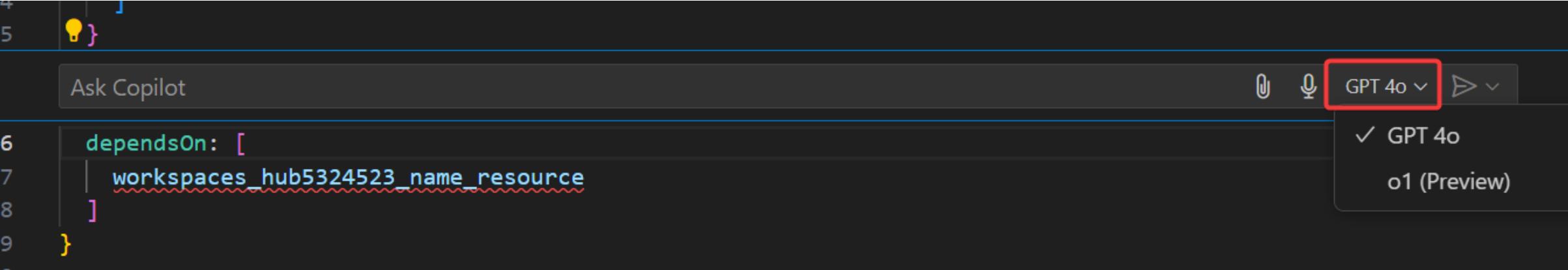
2025/2/5時点



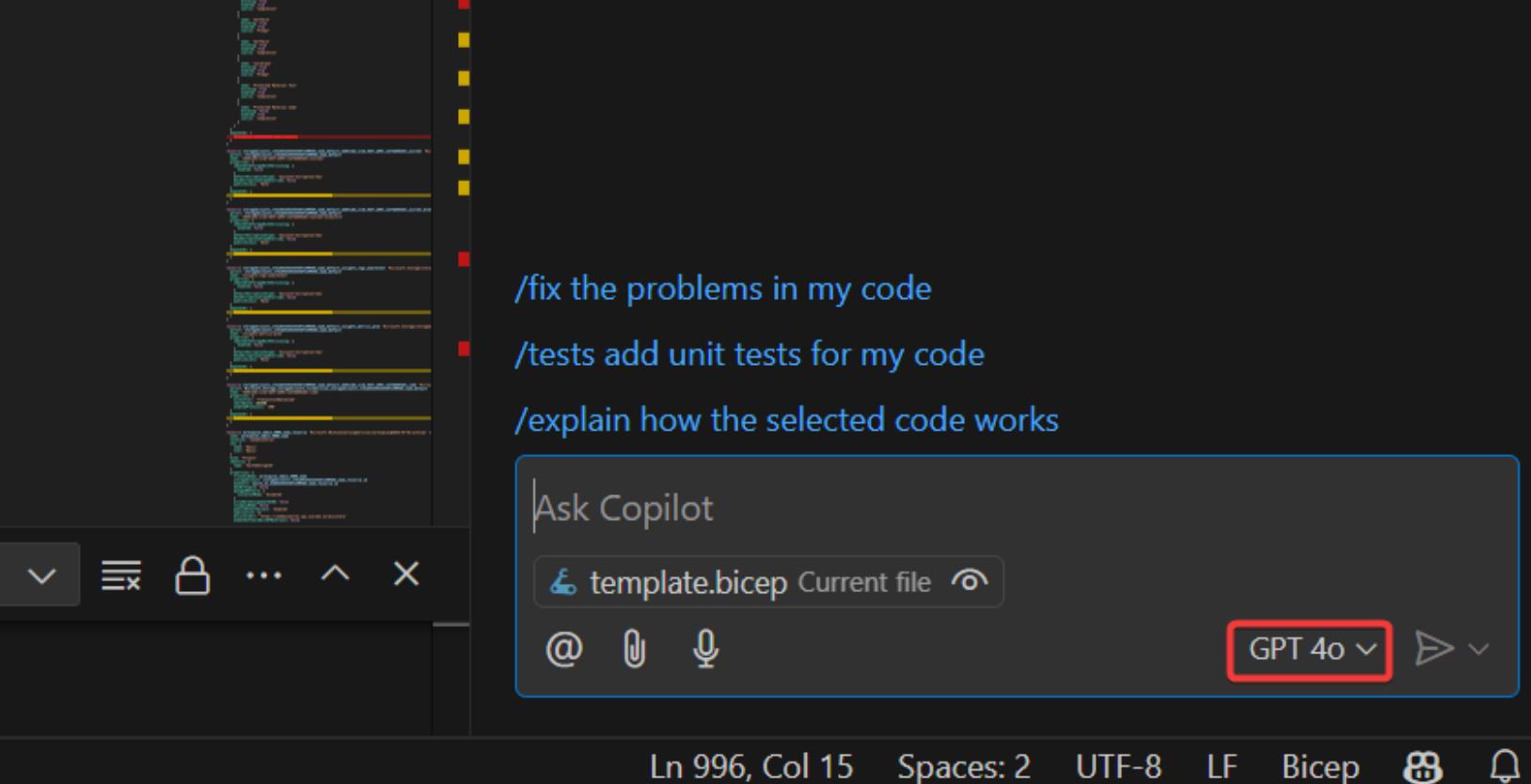
2025/3/1時点



## ■ご参考: モデルの切り替えはいつでも可能



```
95 }  
96 dependsOn: [  
97 | workspaces_hub5324523_name_resource  
98 ]  
99 }
```



/fix the problems in my code  
/tests add unit tests for my code  
/explain how the selected code works

Ask Copilot  
template.bicep Current file

@ 0 0 GPT 4o ▶

Ln 996, Col 15 Spaces: 2 UTF-8 LF Bicep 📁 📲

# モジュール1 GitHub Copilotの概要

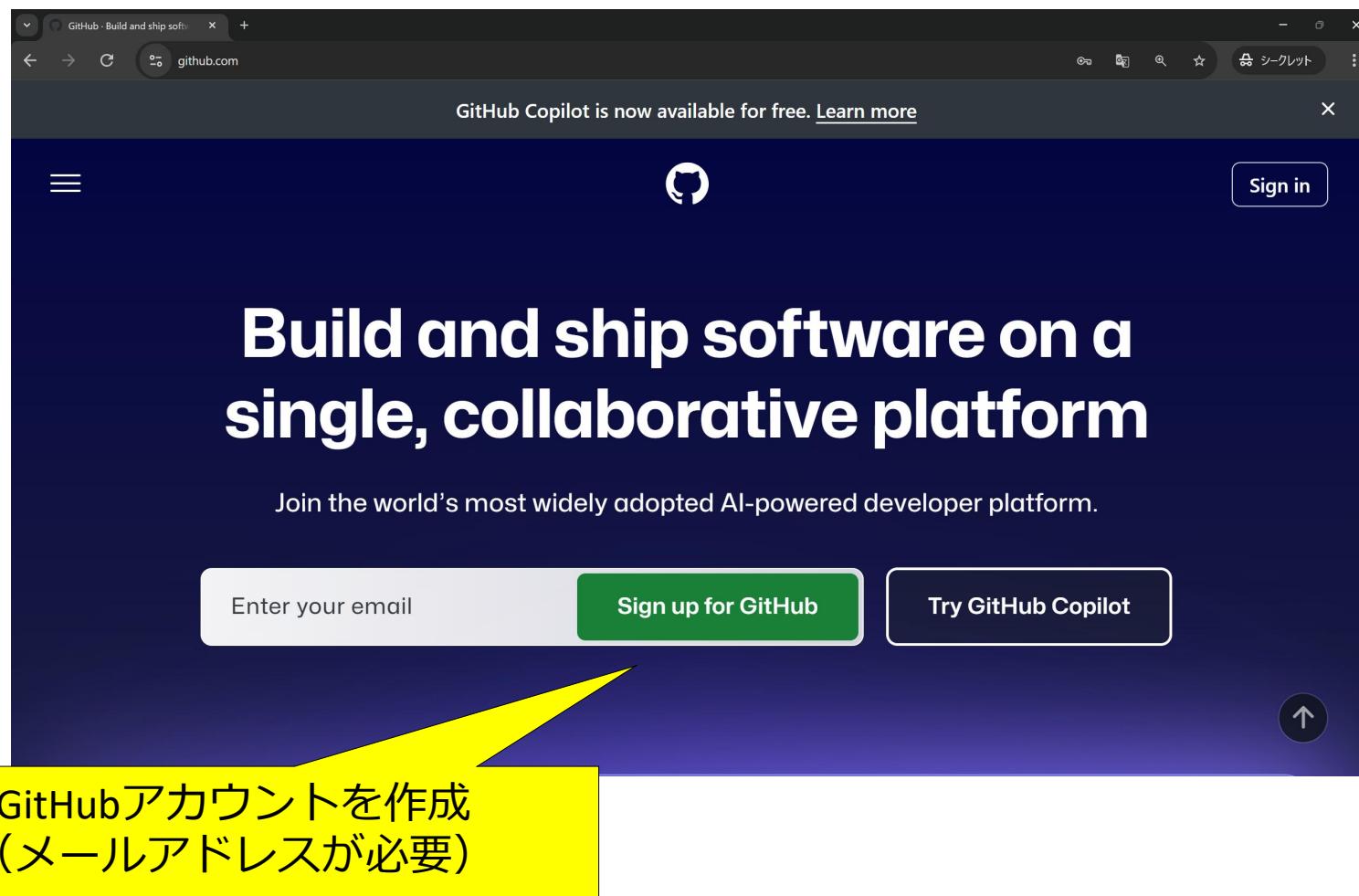
- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilotの使用を開始する手順

- ・※Visual Studio Code と GitHub Copilot for Free を使う場合
- ・GitHubアカウントを持っていない場合は作成
- ・Visual Studio Code をインストールして起動
- ・Visual Studio Code 画面上部の「Copilot」アイコンをクリック
- ・「Sign in to Use Copilot for Free」をクリック
- ・GitHubアカウントにサインイン

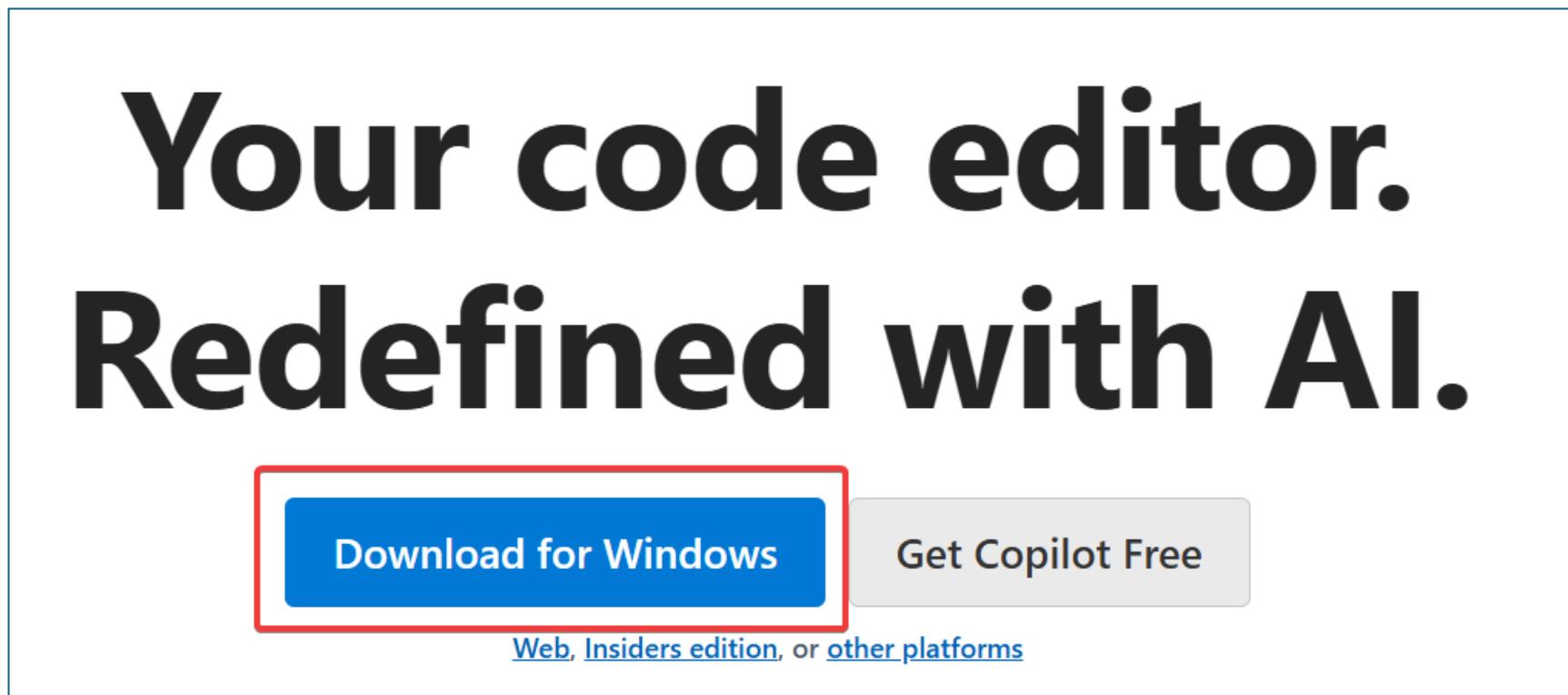
## ■GitHub Copilotの使用を開始する手順(1)GitHubアカウントの作成

<https://github.com/> にアクセス

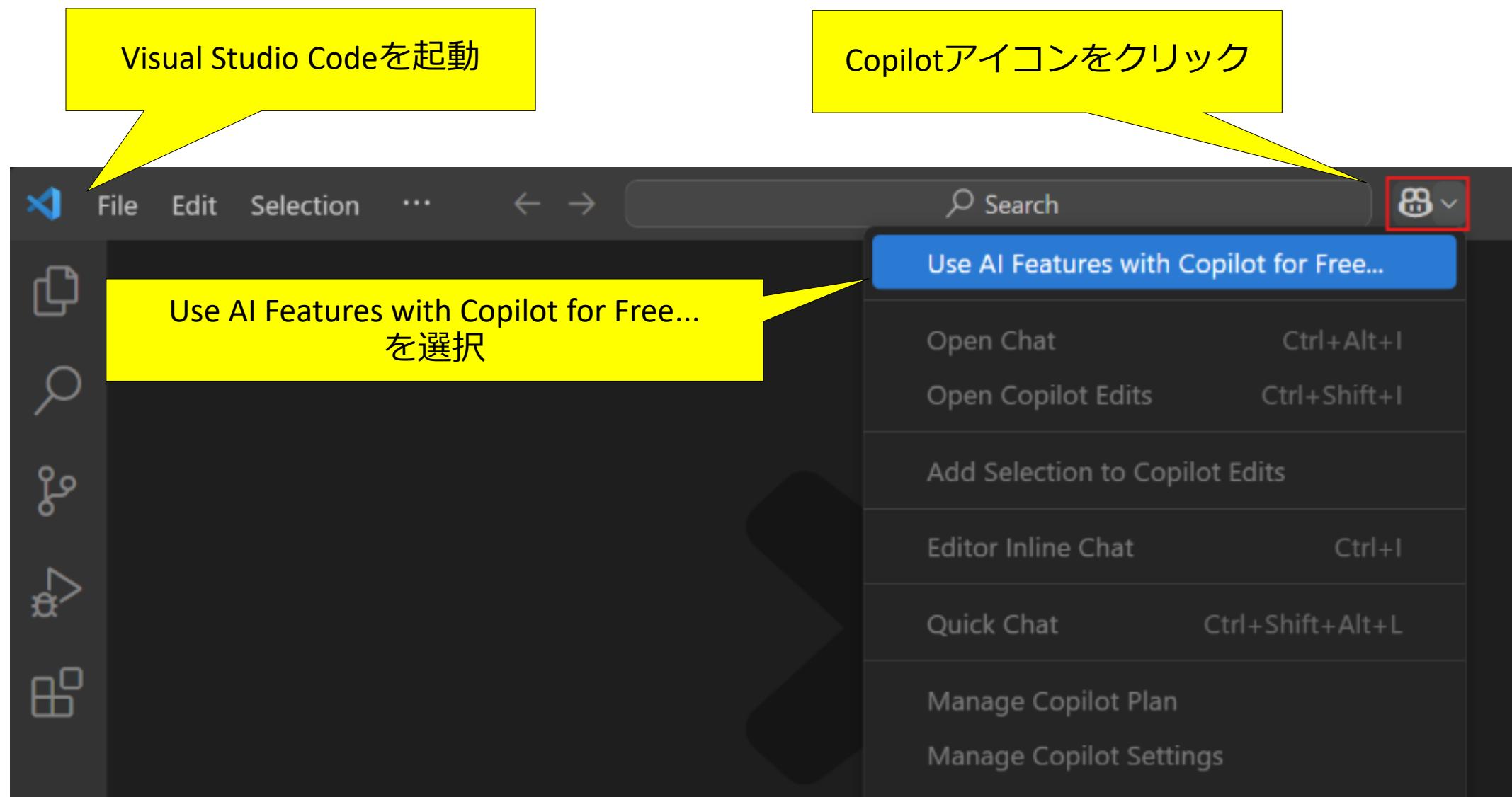


## ■GitHub Copilotの使用を開始する手順(2)Visual Studio Codeのインストール

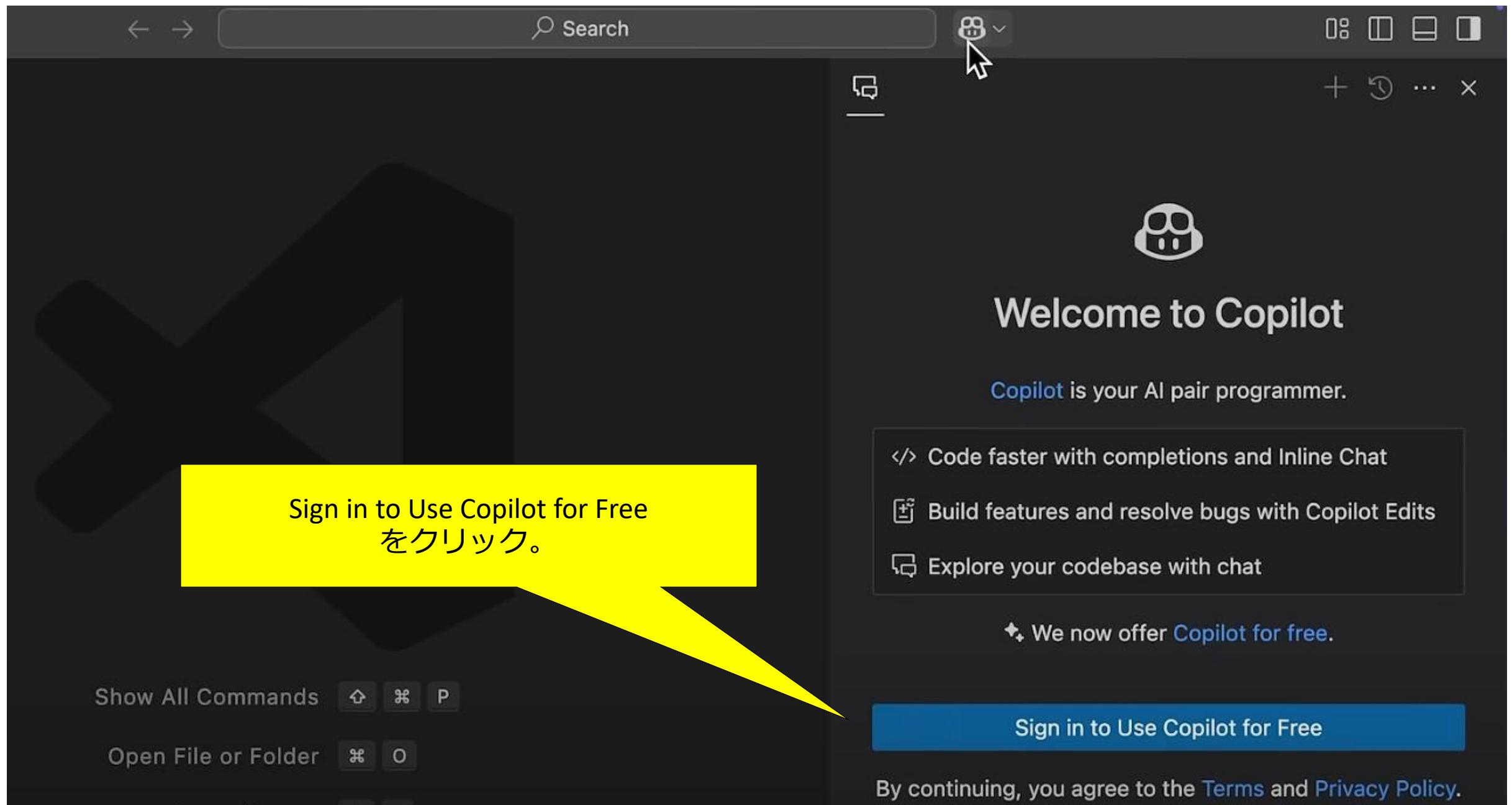
<https://code.visualstudio.com/> から、Visual Studio Code をダウンロードする。Windows/Mac/Linuxに対応。



## ■GitHub Copilotの使用を開始する手順(3)Visual Studio CodeでGitHub Copilot for Freeを選択



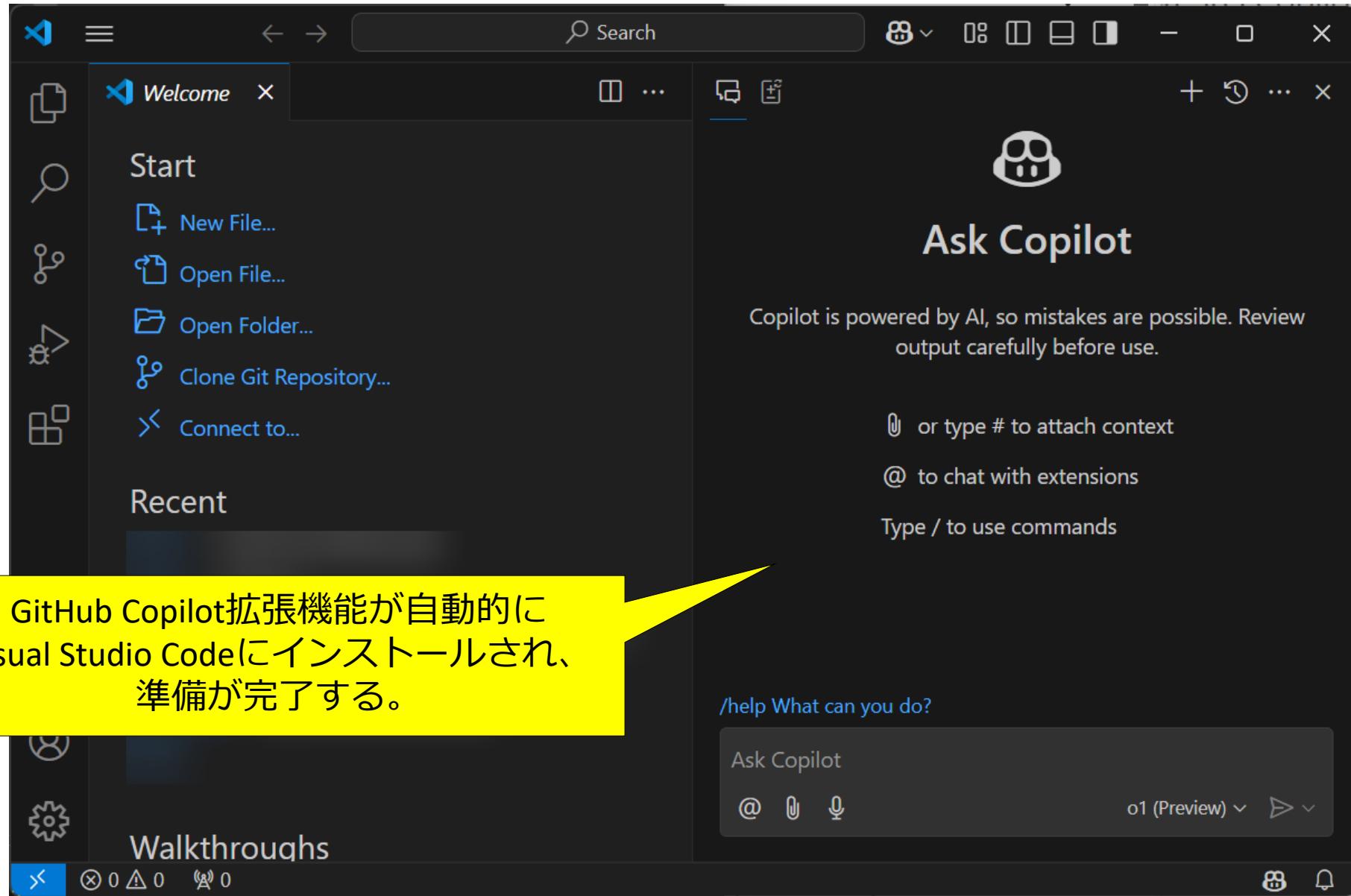
## ■GitHub Copilotの使用を開始する手順(4)GitHubアカウントにサインイン



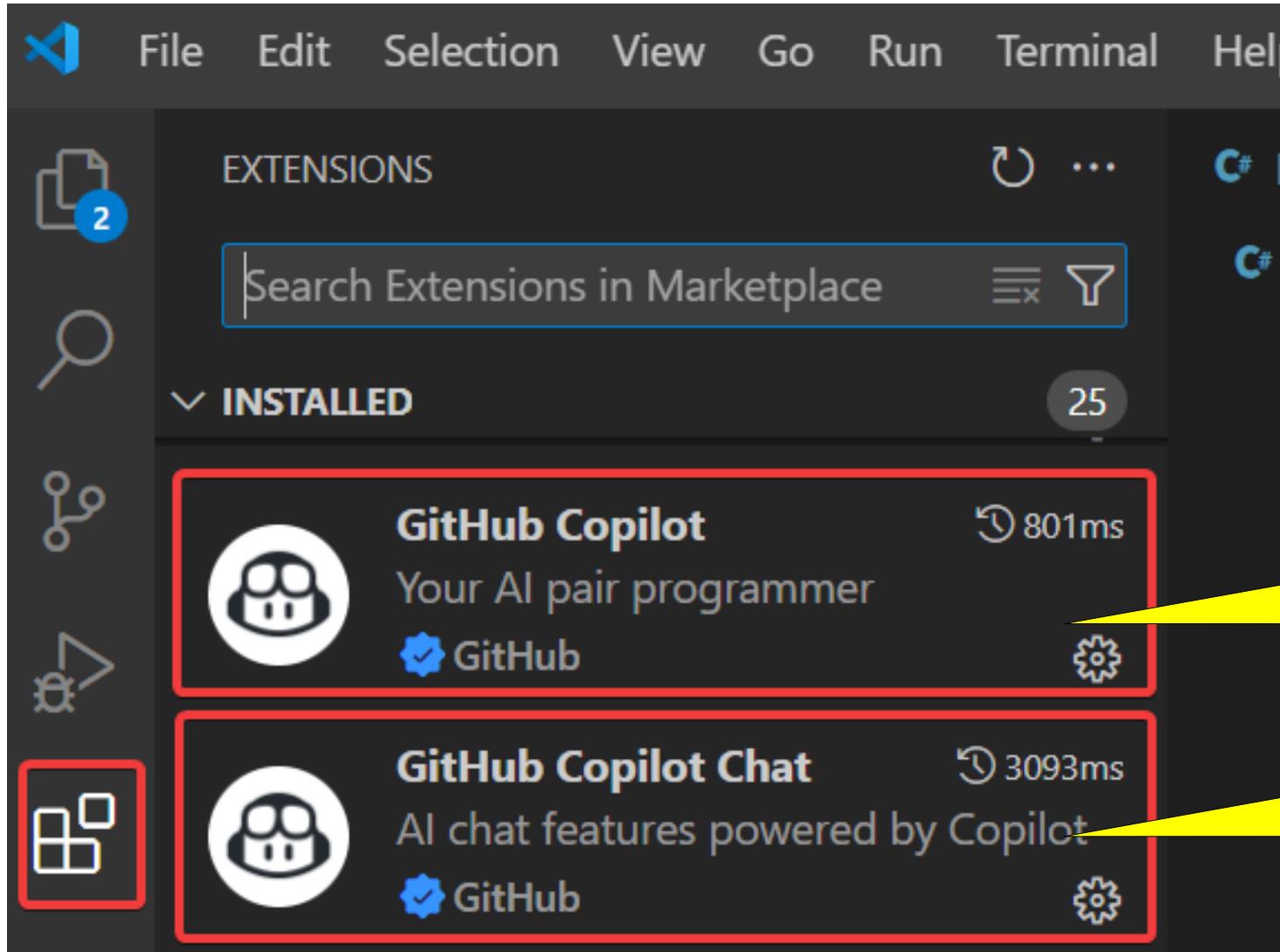
## ■GitHub Copilotの使用を開始する手順(4)GitHubアカウントにサインイン



## ■GitHub Copilotの使用を開始する手順(5)準備完了



## ■このときVisual Studio Codeに2つの「拡張機能」がインストールされる



コード提案（Code Suggestion）を  
提供する拡張機能

チャット、Copilot Editsを  
提供する拡張機能

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilot の基本的な動作: **提案**

- GitHub Copilotは、プログラマが次に打ち込むであろうコードを予測して**提案**してくれる (code **suggestion**)
  - この動作は「**コード補完**」(code completion) とも呼ばれる
- **提案**は少し薄い色で表示される
- **提案**が表示されたら**Tabキー**を押して確定する。
- または**ESCキー**を押して**提案**を却下する。
- GitHub Copilot Freeでは月2000回までコード**提案**を利用できる

## ■GitHub Copilotの基本的な動作

```
60  class IDatabaseServiceImpl : IDatabaseService
61  {
62      public string GetData(int id) => id switch
63      {
64          1 => "りんご",
65          2 => "みかん",
66          _ => "(見つかりませんでした)"
67      };
68  }
69
70 // IDatabaseServiceを実装したクラスで、文房具のデータを返す
71 
```

プログラマが  
コメントを  
書くと・・・

## ■GitHub Copilotの基本的な動作

```
60  class IDatabaseServiceImpl : IDatabaseService
61  {
62      2 references
63      public string GetData(int id) => id switch
64      {
65          1 => "りんご",
66          2 => "みかん",
67          _ => "(見つかりませんでした)"
68      };
69
70 // IDatabaseServiceを実装したクラスで、文房具のデータを返す
71 class StationeryDatabaseService : IDatabaseService
{
    public string GetData(int id) => id switch
    {
        1 => "ボールペン",
        2 => "シャープペンシル",
        _ => "(見つかりませんでした)"
    };
}
```

コメントに沿って、文房具のデータを返すコードが生成される（ここでは、すでに書かれている上のコードを手本として、同じパターンのコードが生成されている）

GitHub Copilotがコードを提案（コメントに続く適切なコードを生成し、候補として表示）

コメントに沿って、インターフェースも実装されている

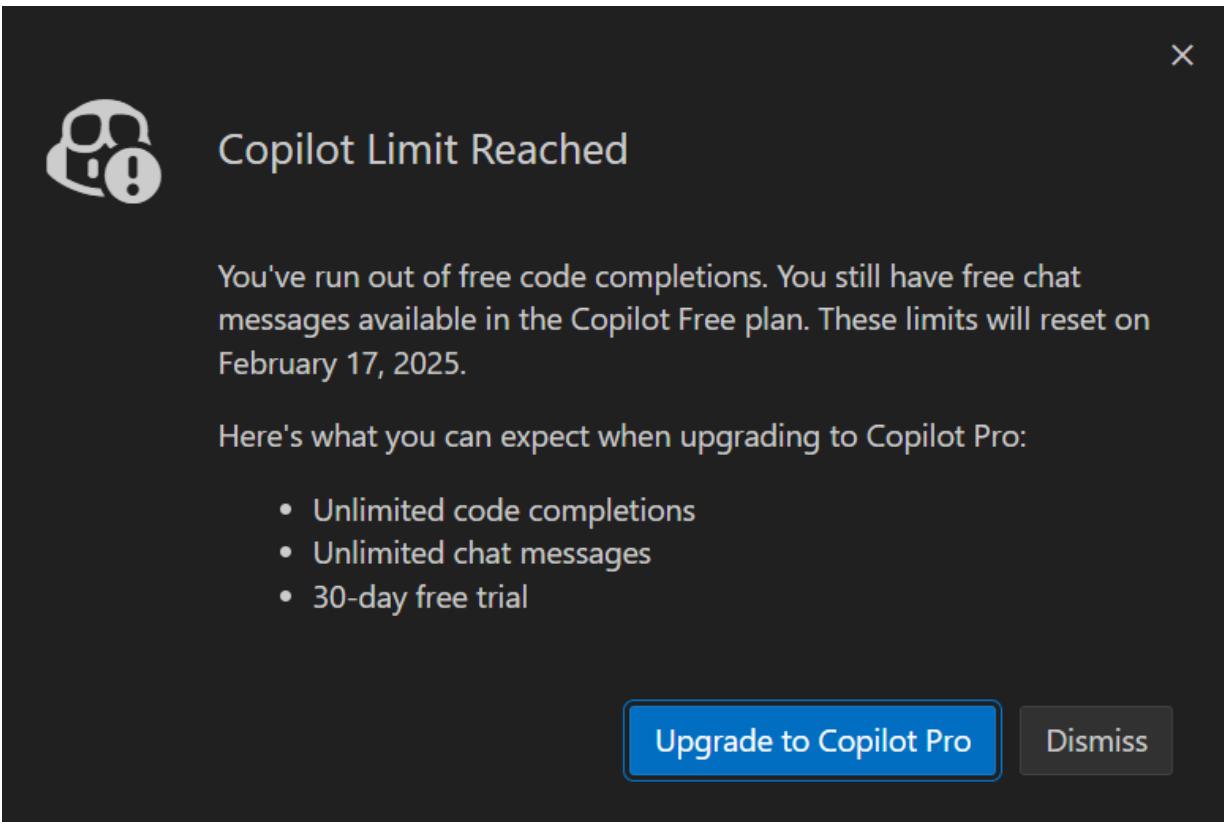
## ■GitHub Copilotの基本的な動作

```
60  class IDatabaseServiceImpl : IDatabaseService
61  {
62      2 references
63      public string GetData(int id) => id switch
64      {
65          1 => "りんご",
66          2 => "みかん",
67          _ => "(見つかりませんでした)"
68      };
69
70 // IDatabaseServiceを実装したクラスで、文房具のデータを返す
71 // 0 references
72 class StationeryDatabaseService : IDatabaseService
73 {
74     2 references
75     public string GetData(int id) => id switch
76     {
77         1 => "ボールペン",
78         2 => "シャープペンシル",
79         _ => "(見つかりませんでした)"
80     };
81 }
```

Tabキーを押して  
提案を確定

## ■GitHub Copilot Freeで利用できる制限に達した場合の表示例

### Copilot の制限に達しました



**無料のコード補完**が利用できなくなりました。この制限は 20XX 年 M 月 N 日にリセットされます。

1ヶ月経過すると  
また2000回の補完が利用できるようになる

GitHub Copilot Pro にアップグレードすると、以下のものが利用できる：

- 無制限のコード補完
- 無制限のチャット メッセージ

GitHub Copilot Pro は月10ドルの費用が発生（最初の30日は無料で利用可能）

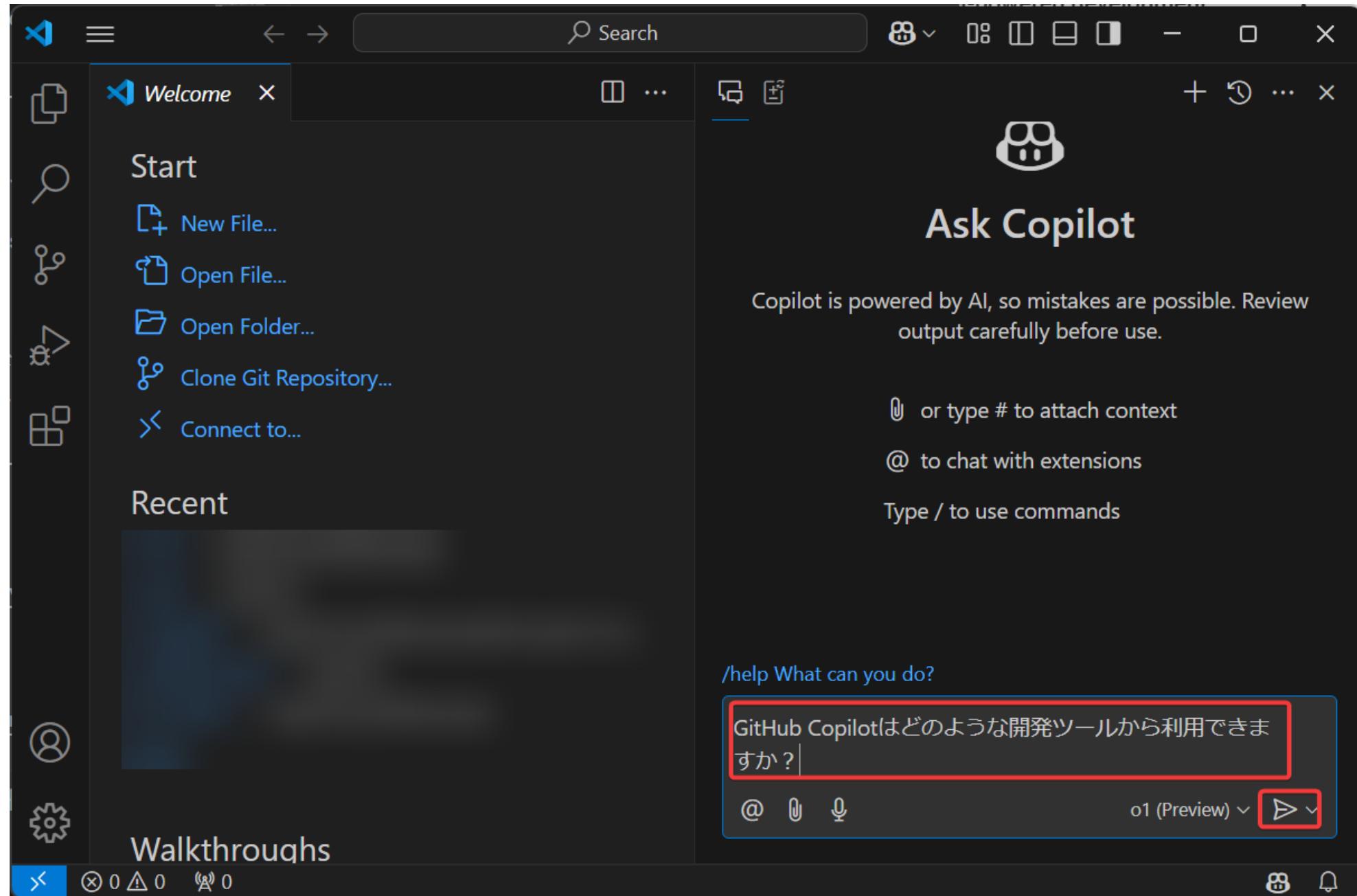
# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

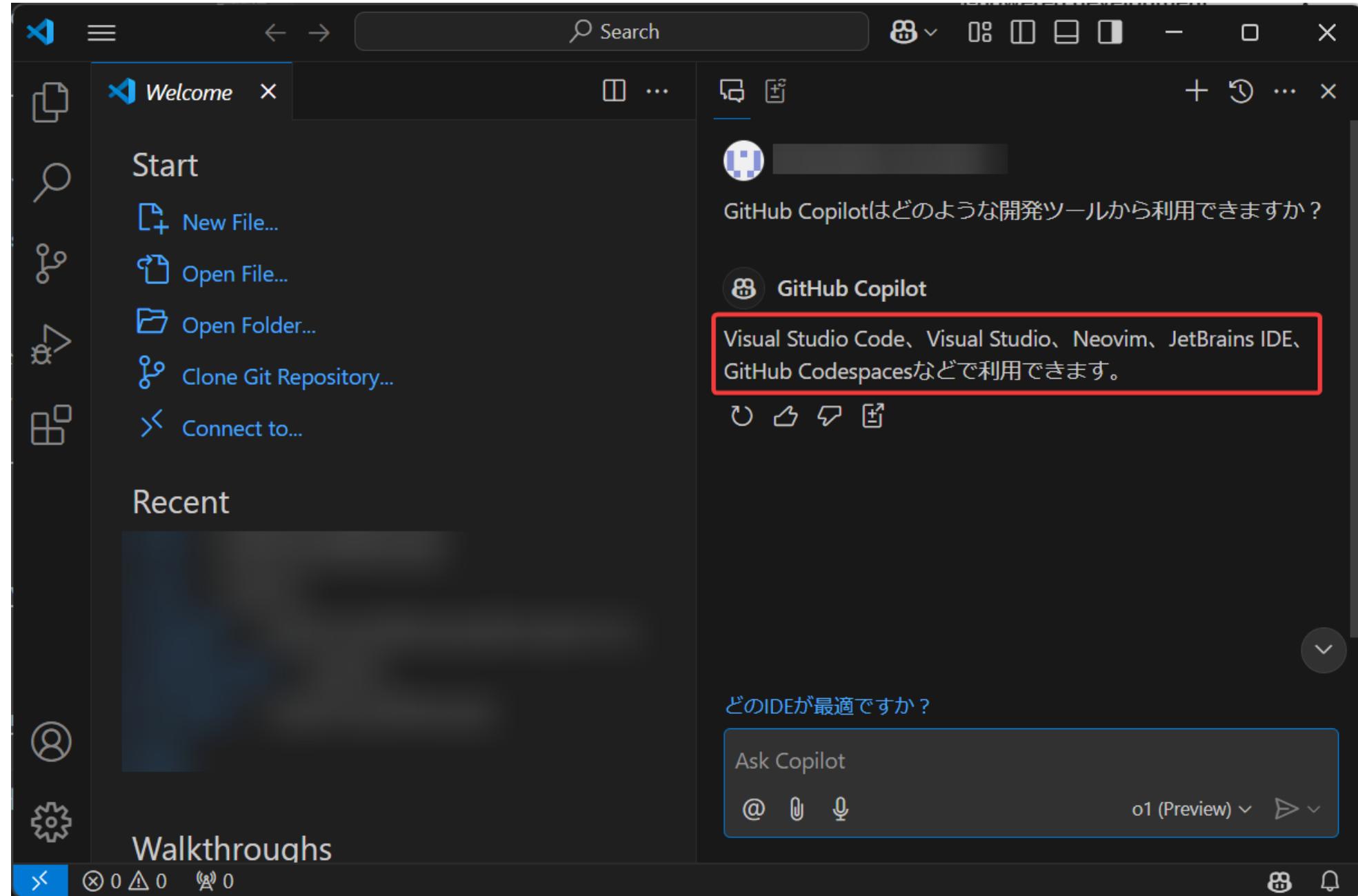
# GitHub Copilot **Chat** とは？

- ・チャット形式でAIに質問したり、コードの説明や修正案をもらったりできる機能
- ・2023/12/29 一般提供開始
- ・GitHub Copilot Freeの場合は月50メッセージまで使用可能
- ・利用方法
  - ・Visual Studio Codeなどの開発ツールから
  - ・GitHubのトップ画面（ダッシュボード）から
  - ・GitHub モバイルアプリから

## ■ Visual Studio Code内から利用する例



## ■Visual Studio Code内から利用する例



## ■GitHubのトップページ（ダッシュボード）から利用する例

The screenshot shows the GitHub Dashboard interface. On the left, there's a sidebar with 'Top repositories' and a 'New' button. Below it is a search bar with 'Find a repository...'. The main area is titled 'Home' and features a prominent 'Ask Copilot' button, which is highlighted with a red rectangular box. Other visible elements include a search icon, user profile icons, and links for 'Find issues assigned to me' and 'Python password endpoint'.

Ask Copilot

Find issues assigned to me

Python password endpoint

Home

microsoft/semantic-kernel released 5 days ago

dotnet-1.34.0

Changes:

## ■GitHubのトップページ（ダッシュボード）から利用する例

The screenshot shows the GitHub Dashboard at <https://github.com>. On the left, there's a sidebar with 'Top repositories' and a 'New' button. Below it is a search bar with 'Find a repository...'. The main area is titled 'Home' and features a prominent callout box with the Japanese text 'GitHub Copilotはどのような開発ツールから使えますか?' (Which development tools can GitHub Copilot be used with?). This callout is highlighted with a red border. To the right of the callout, it says 'Copilot uses AI. Check for mistakes.' and has a 'Filter' button. Below the callout, there's a repository card for 'microsoft/semantic-kernel' with a release note: 'released 5 days ago dotnet-1.34.0'. At the bottom, there's a 'Changes:' section.

GitHub Copilotはどのような開発ツールから使えますか?

Copilot uses AI. Check for mistakes.

Home

microsoft/semantic-kernel released  
5 days ago

dotnet-1.34.0

Changes:

## ■GitHubのトップページ（ダッシュボード）から利用する例

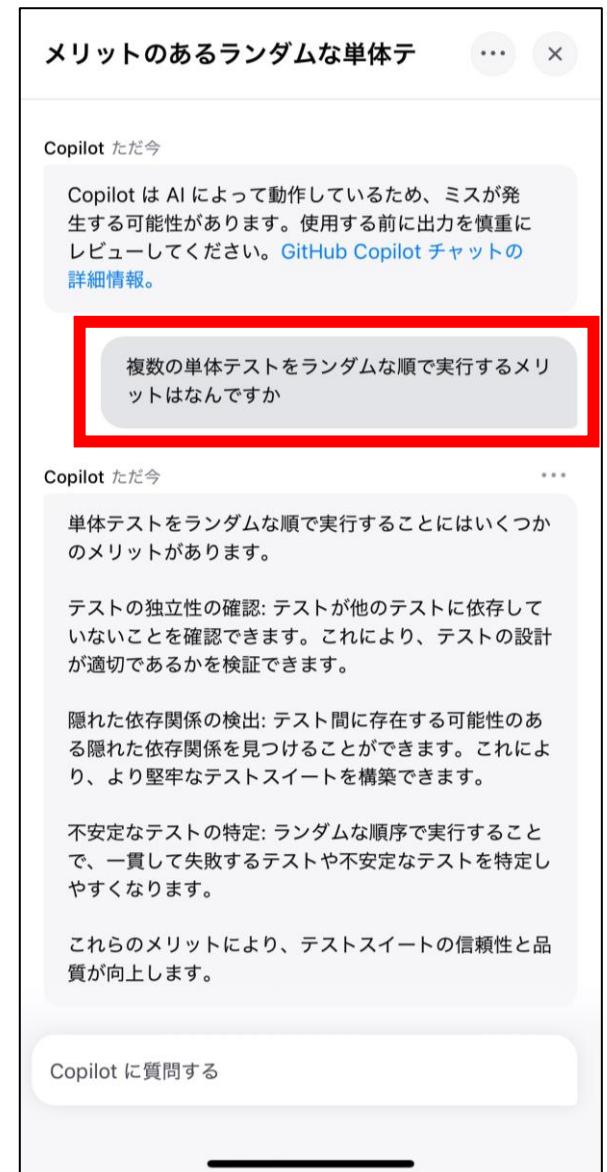
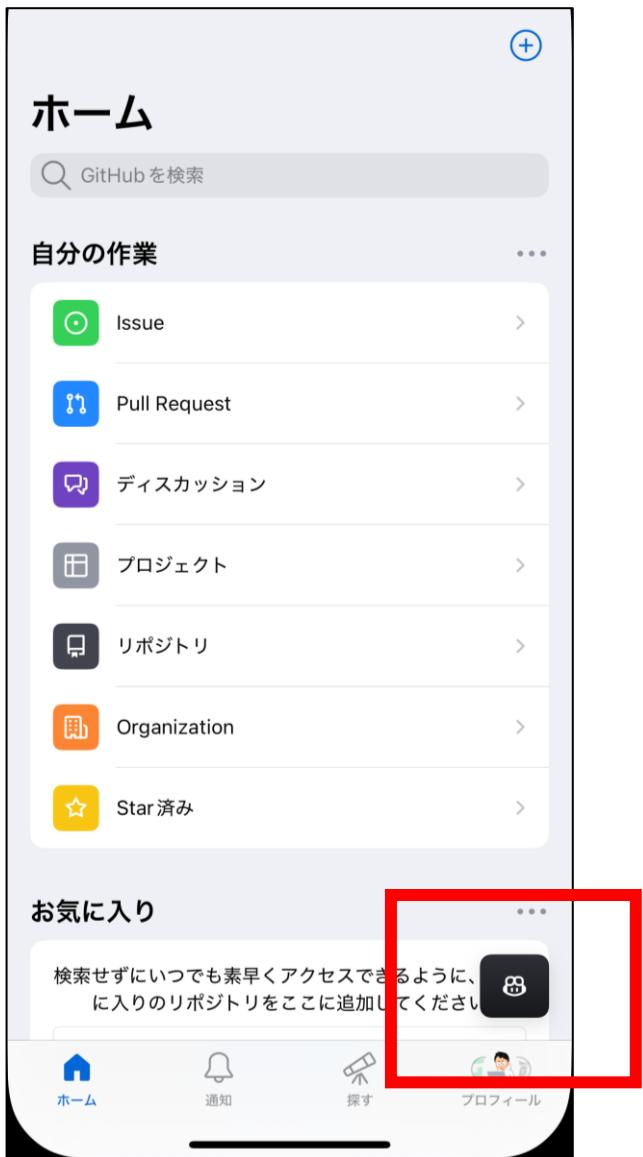
The screenshot shows the GitHub Copilot interface. At the top, there's a navigation bar with icons for three, GitHub logo, Copilot, search, and other options. Below it is a toolbar with icons for refresh, edit, and more. The main area displays a question "GitHub Copilotはどのような開発ツールから使えますか？" (What development tools can GitHub Copilot be used from?) followed by an answer and a summary at the bottom.

**GitHub Copilotは以下の開発ツールから使用できます：**

- 1. Visual Studio Code (VS Code):**
  - GitHub CopilotはVisual Studio Codeの拡張機能として提供されています。インストール後、コーディング中に自動補完やコード提案を受けることができます。
- 2. Visual Studio:**
  - Visual Studio 2022以降のバージョンでGitHub Copilotを使用することができます。VS Codeと同様に、コード補完や提案機能を利用できます。
- 3. JetBrains製品:**
  - IntelliJ IDEA、PyCharm、WebStormなどのJetBrains製品でもGitHub Copilotを使用することができます。JetBrains Marketplaceからプラグインをインストールすることで利用可能です。

これらのツールを使用して、効率的なコーディング体験を得ることができます。

# ■GitHubモバイルアプリでGitHub Copilotのチャットを利用する例



# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilot では どのようなことができるのか？

- ・コードのドキュメントを生成する（モジュール2で解説）
- ・コメントからコードを生成する（モジュール3で解説）
- ・単体テストのコードを生成する（モジュール4で解説）
- ・コードの品質やセキュリティを改善する（モジュール5で解説）
- ・コマンドを提案・実行する（「補足」で解説）

## ■ご参考: 本コースではすべてを解説しませんが、以下のような使い方も可能です

- 一般的な質問への回答（開発に関する質問に答えてもらうなど）
- プロンプトの生成、改善
- コードのアップデート（Python 2で書かれたコードをPython 3にアップデートするなど）、コンバート（COBOLのプログラムをJavaに書き換えるなど）
- 選択部分やGitの変更部分のレビューの生成（GitHub Code Review）
- HTML・CSS・JavaScriptの生成
- テストデータの生成（CSVなど）
- 設定ファイルの生成（YAML、JSONなど）
- IaCの定義ファイルの生成（Terraform、ARM テンプレート、Bicepなど）
- GitHub Actionsのワークフローの生成（YAML）
- UMLダイアグラムなどの図の生成（SVG、Mermaidなど）
- SQL文の生成
- コマンドやスクリプトの生成
- プログラムのエラーログや、コマンドのエラーメッセージなどの原因究明と対処方法の提案
- コミットメッセージの生成、プルリクエストのタイトルと説明文の生成
- @azure や @askstackoverflow といった「GitHub Copilot Extensions」の追加
- 独自の「GitHub Copilot Extensions」の開発

# ■ご参考: 「Copilot Chat クックブック」で、GitHub Copilot のチャットを使った作業の例（プロンプト例）が確認できる

The screenshot shows the GitHub Docs interface for the Copilot Chat Cookbook. The left sidebar has a navigation menu with sections like Home, GitHub Copilot, Quickstart, and Copilot Chat Cookbook. The Copilot Chat Cookbook section is expanded, showing a list of prompts such as All prompts, Debugging code, Intermediate, and Advanced. The main content area is titled "Copilot Chat クックブック" and describes examples of prompts used in GitHub Copilot Chat. It features a "Spotlight" section with three cards: "Generate unit tests", "Improving code readability and maintainability", and "Debugging invalid JSON". Below this, there's a search bar and filter options for "記事を検索する", "カテゴリ: All", "複雑性: All", and "フィルターのリセット". At the bottom, there are three more cards: "無効な JSON のデバッグ", "API のレート制限の処理", and "考えられる機能実装について確認する". Each card includes a brief description and tags like "Debugging code" and "Intermediate".

[Copilot Chat クックブック - GitHub Docs](#)

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# GitHub Copilot と連携できる開発ツール

- Visual Studio / Visual Studio Code 本コースではVisual Studio Codeでの使用方法を解説
- JetBrains IDE（IntelliJ IDEA、PyCharm、Rider、WebStorm等）
- Vim / Neovim
- Xcode
- Eclipse
- Azure Data Studio
- GitHub CLI (`gh copilot explain` / `gh copilot suggest`)
- GitHub Mobile アプリ
- Windows Terminal

## ■各開発ツールでの導入方法は公式サイトのドキュメントで確認できる

The screenshot shows the GitHub Docs page for "GitHub Copilot を使用して IDE でコードの提案を取得する". The page header includes the GitHub logo and "GitHub Docs", with a dropdown for "Version: Free, Pro, & Team". The main title is "GitHub Copilot を使用して IDE でコードの提案を取得する". Below it, a sub-section title reads "GitHub Copilot を使用して、エディターでコードの提案を取得する". A horizontal navigation bar lists supported tools: Azure Data Studio, JetBrains IDEs, Vim/Neovim, Visual Studio, Visual Studio Code, and Xcode. A green button at the bottom left says "無料で始める". A yellow callout bubble points to the "JetBrains IDEs" link, stating "各開発ツールでの GitHub Copilot 導入方法のドキュメントが利用可能". Another yellow callout bubble points to the bottom right, stating "詳しくはこちらのページを参照". The footer contains the URL "GitHub Copilot を使用して IDE でコードの提案を取得する - GitHub Docs".

GitHub Docs

Version: Free, Pro, & Team

三 GitHub Copilot / GitHub Copilot を使用する / コードの提案を取得する

## GitHub Copilot を使用して IDE でコードの提案を取得する

GitHub Copilot を使用して、エディターでコードの提案を取得する

Azure Data Studio    JetBrains IDEs    Vim/Neovim    Visual Studio    Visual Studio Code    Xcode

無料で始める

各開発ツールでの GitHub Copilot 導入方法のドキュメントが利用可能

詳しくはこちらのページを参照

[GitHub Copilot を使用して IDE でコードの提案を取得する - GitHub Docs](#)

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# よくあるご質問

- GitHub Copilotが生成するコードは正確ですか？

→必ずしも正確とは限らないため、プログラマ（人間）が結果をよく確認する必要がある。

# よくあるご質問

■ GitHub Copilotは無料ですか、有料ですか？

→無料（Free）でも利用できますが、提案・チャットの回数に制限があるため、業務で使用するのであれば有料プラン（Pro、Business、Enterprise）がおすすめです。

■ 有料プランは従量課金ですか？

→いいえ、定額料金です。ユーザーあたり・月あたりで 10 USD（Proプランの場合）といったように価格が決められています。

# よくあるご質問

- GitHub CopilotのBusinessとEnterpriseは何が違いますか?  
→以下のMicrosoft Learnで詳しく学習できます

[GitHub Copilot の基礎 - AIペアプログラマについて理解する - Training | Microsoft Learn](#)

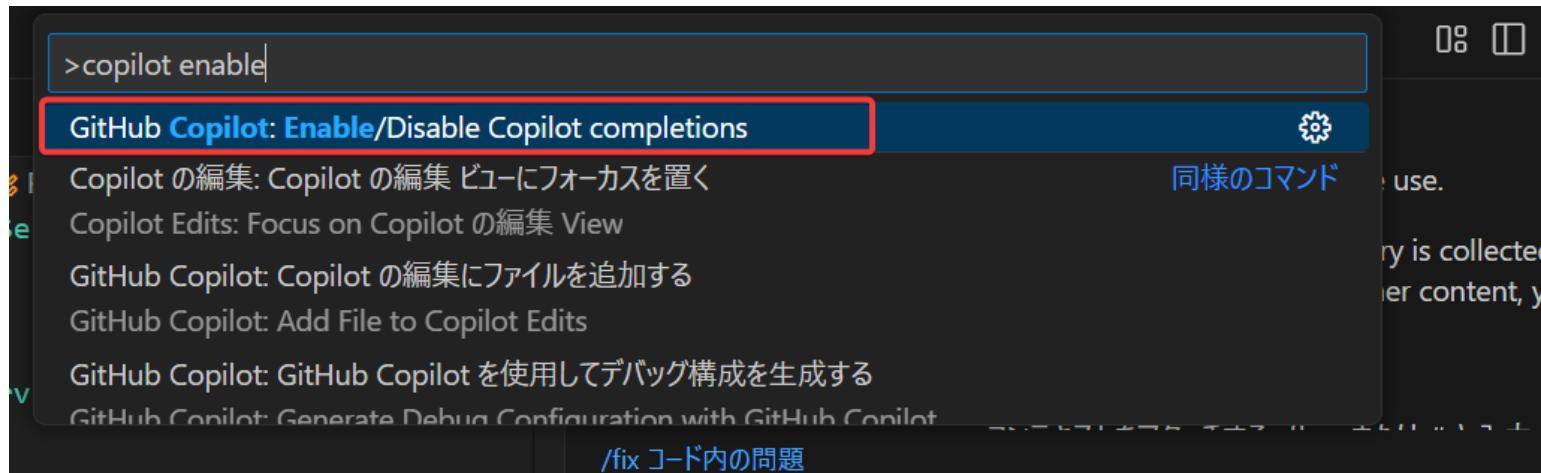
The screenshot shows two training modules from Microsoft Learn:

- GitHub Copilot Business の概要**:
  - 700 XP
  - 5 分 残り • モジュール • 4/6 ユニットが完了しました
  - GitHub Copilot Business と GitHub Copilot Individuals の違いについて学習します。また、GitHub Copilot Business の具体的なユース ケースと顧客事例について、および実際の使用方法についても学習します。
  - 概要 ▾
- GitHub Copilot Enterprise の概要**:
  - 600 XP
  - 5 分 残り • モジュール • 3/5 ユニットが完了しました
  - GitHub Copilot for Enterprise、for Business、for Individuals の違いについて説明します。GitHub Copilot Enterprise を有効にして使用する方法など、特定のユース ケースを確認します。
  - 概要 ▾

# よくあるご質問

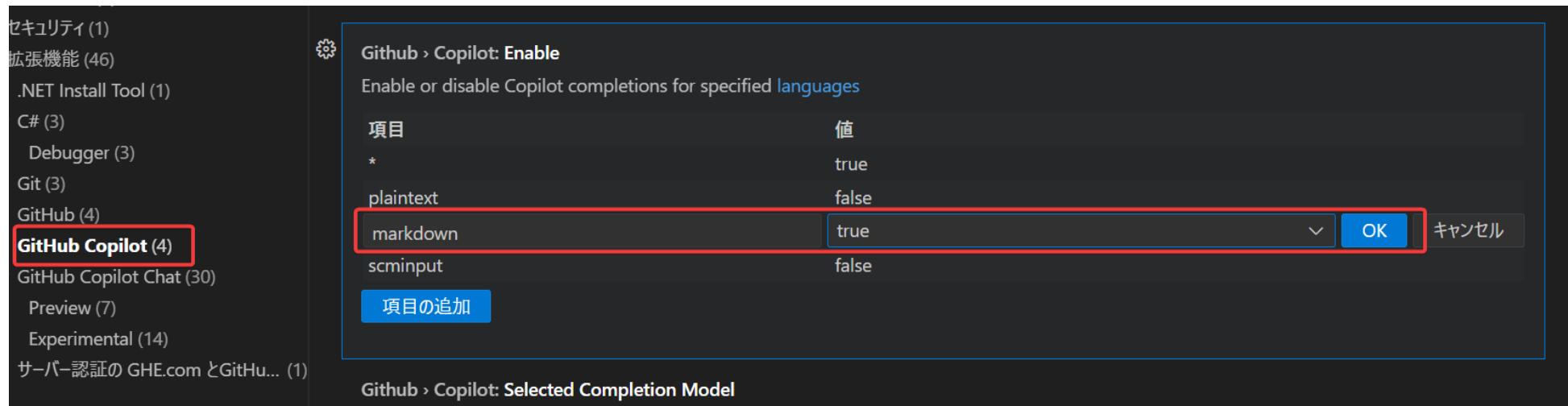
■ GitHub Copilotのコード提案を普段はOFFにしておき、必要な場合にのみONにして利用できますか？

→可能です。[F1]または[Ctrl + Shift + P]を押してコマンドパレットを表示し、[GitHub Copilot: Enable/Disable Copilot completions]を呼び出すことで、提案（補完）をON/OFFできます



# よくあるご質問

■テキストファイルやMarkdownファイルではGitHub Copilotの提案が表示されないです  
→設定で変更できます。



# よくあるご質問

■ GitHub Copilotはどのようなプログラミング言語で使用できますか？

→C#、Java、Python、JavaScript、TypeScriptなど、非常に多くの言語に対応しています。

# よくあるご質問

- GitHub Copilotのショートカットキーがたくさんあって覚えるのが難しい  
→ 「チートシート」（ショートカットキーまとめ）を活用

## GitHub Copilot in VS Code cheat sheet

### Chat with GitHub Copilot

Use natural language to chat with GitHub Copilot and get help with coding tasks. For example, ask Copilot to explain a block of code or a programming concept. Get more information about using [Copilot Chat](#).

Action	Description
<code>Ctrl+Alt+I</code>	Open the <b>Chat</b> view and start a chat conversation with Copilot by using natural language.
<code>Ctrl+Shift+I</code>	Open the <b>Copilot Edits</b> view and start a code editing session across multiple files.

→一週間くらいGitHub Copilotを使うと指が自然にショートカットを覚えると思います。

# よくあるご質問

■ GitHub Copilotの企業での採用事例はありますか？  
→非常にたくさんあります。

Github Copilot 採用企業・レポートをまとめてみた (2023年08月07日時点) #GitHub - Qiita

TC3株式会社: GitHub Copilot for Businessを導入  
no plan株式会社: 「ChatGPT & GitHub Copilot手当」を全員に導入  
株式会社M&Aクラウド: GitHub Copilot導入におけるハードルの整理  
STORES株式会社: GitHub Copilot for Businessを導入  
株式会社SmartHR: GitHub Copilotを導入し、開発体験の向上に貢献  
OLTA株式会社: ChatGPTとGitHub Copilotを全社導入  
株式会社Leaner Technologies: GitHub Copilot for Businessを導入  
株式会社Gaudiy: GitHub Copilotを導入して1ヶ月経過  
ミスリル株式会社: ゲーム開発の現場でGitHub Copilotを検証  
NewCyte合同会社: GitHub Copilotを使ったプログラミングにチャレンジ  
株式会社ワントゥーテン: GitHub Copilotを全社導入  
株式会社スマートバンク: GitHub Copilot for Businessを導入  
株式会社カケハシ: ChatGPTやGitHub Copilotの活用に関する考慮点  
株式会社ZOZO: GitHub Copilotの全社導入とその効果  
レバレジーズ株式会社: GitHub Copilotの全社導入過程でのノウハウ  
FastLabel株式会社: 開発社員向け福利厚生としてGitHub Copilotを導入  
ナイル株式会社: GitHub Copilotを全エンジニアに導入

株式会社スカイディスク: GPT-4を搭載したGitHub Copilot Xを導入  
株式会社ツクリバ: GitHub Copilotの利用環境を全エンジニアに提供開始  
ラグナロク株式会社: エンジニア希望者全員にGitHub Copilotを支給  
株式会社TOKIUM: ChatGPT PlusとGitHub Copilotを導入  
ENECHANGE株式会社: GPT-4とGitHub Copilotの導入  
株式会社ヘッドウォータース: GitHub Copilot for Businessの全社導入  
Crezit Holdings株式会社: GitHub Copilotの導入  
株式会社メタップス: GitHub Copilot Xを導入  
株式会社ワンキャリア: GitHub Copilotを導入  
株式会社サーバーワークス: ChatGPT PlusとGitHub Copilotの補助制度を導入  
株式会社Ridge-i: GitHub CopilotとChatGPT Plusを導入  
株式会社アイスリーデザイン: GitHub Copilotを導入  
BRANU株式会社: GitHub Copilotを導入  
株式会社アブリズム: Microsoft 365 CopilotとGitHub Copilotを導入  
株式会社エブリー: ChatGPT PlusとGitHub Copilotを導入  
株式会社スタディスト: AIとのセキュアな相談を可能にし、従業員の生産性を向上  
パーソルホールディングス株式会社: GitHub Copilotを導入

# よくあるご質問

- GitHub Copilotのすべての機能は、どのツール（vsCode, Visual Studio, Eclipse, Xcodeなど）でも同じように利用可能ですか？  
→いいえ。
  - ・現在Visual Studio Codeが最も進化が早く、多くの機能がすばやく投入され、すぐに最新機能を利用できます。
  - ・Visual Studio でも多くの機能を利用でき、Visual Studioでしか利用できない「GitHub Copilotによるアプリデバッグ」といった機能もあります。
  - ・その他のツール・IDEについては、「チャットのみ」「補完のみ」など、サポート状況はまちまちで、今後のアップデートにより徐々に改善が進むと思われます。

# よくあるご質問

■ GitHub Copilotは、GitHub Codespaces（GitHubが提供するクラウドベースの開発環境）で使用できますか？

→はい、利用できます。

# よくあるご質問

■ GitHub Copilotは、[vscode.dev](https://vscode.dev)（web版のVisual Studio Code）で使用できますか？

→いいえ。GitHub Copilotの拡張機能がセットアップできないため、利用できません。

# よくあるご質問

- GitHub Enterprise Server（セルフホストバージョンのGitHub）を使用していますが、GitHub Copilotは使用できますか？

→いいえ、GitHub CopilotはGitHub Enterprise Serverの環境では使用できません。

# よくあるご質問

■ GitHub Copilotは、インターネット接続がない環境でも使用できますか？

→いいえ、使用できません。インターネット接続（GitHubのサーバーへのHTTPS接続）が必須となります。

HTTPプロキシが利用できる環境では使用できます。

# よくあるご質問

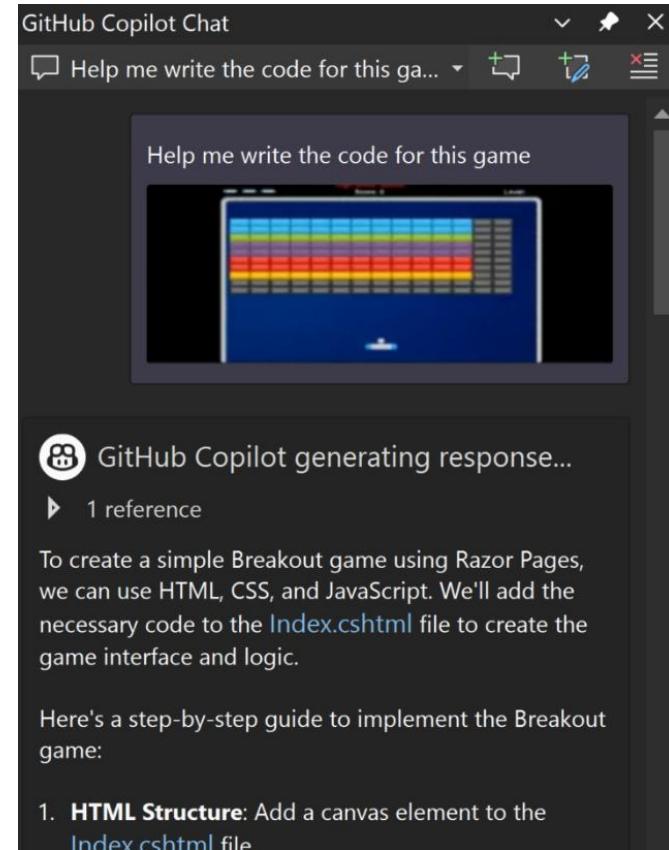
- GitHub Copilotでは、ローカル環境で稼働しているAIモデル（たとえばOllamaなどを使用）を利用できますか？

→いいえ、GitHub Copilotでは、GitHub社が提供するAIモデルと組み合わせて利用できますが、それ以外のAIモデルは利用できません。

# よくあるご質問

■ GitHub Copilotのチャットに画像を送信できますか？（画像を参考に作業をしてもらう、トラブルシューティングをしてもらうなど）

→はい、画像添付が可能です。



# よくあるご質問

- GitHub Copilotは、音声での入力・出力には対応していますか？

→はい、対応しています。Visual Studio CodeでGitHub Copilotが使える状態にして、さらに「VS Code Speech」拡張機能をセットアップすることにより、マイクから音声入力でGitHub Copilotに指示を出したり、回答を読み上げてもらったりすることが可能です。日本語にも対応しています。

# よくあるご質問

- GitHub Copilotを使用すると、ユーザーのデータ（コード）はモデルのトレーニングに使用されますか？  
→個人での使用（Free、Proプラン）の場合：設定できる

Allow GitHub to use my code snippets for product improvements \*

Allow GitHub and its affiliates to use my code snippets to research and improve Copilot suggestions, related models and product features. More information in [Privacy FAQ](#).

Visual Studio Code  
の設定

- 企業での利用（Business、Enterpriseプラン）の場合：いいえ

**GitHubは、GitHubのモデルをトレーニングするためにCopilot BusinessまたはEnterpriseデータを使用しますか？**

いいえ。GitHubはモデルのトレーニングにCopilot BusinessデータもEnterpriseデータも使用しません。

# よくあるご質問

- GitHub Copilotによって、プログラマの仕事はなくなりますか？

→GitHub Copilotはプログラマの仕事を支援し、生産性を向上させるツールと位置づけられています。プログラマの仕事をなくすことがGitHub Copilotの目的ではありません。

# よくあるご質問

## ■ GitHub Copilotの応答や生成は英語？日本語？

→Visual Studio Codeの表示言語設定に従うはずですが、実際動かしてみると必ずしも言語設定に従わない場合があります。

チャットのプロンプトで「in English」「in Japanese」「日本語で」といったように追加で指示するとよいです。

VSCodeの設定で明示的に「応答ロケール」を指定することも可能です。



# よくあるご質問

■ GitHub Copilotが提案するコードやチャットの動作はカスタマイズできますか？（規則や指示を追加するなど）

→はい、いくつかの方法でカスタマイズが可能です。

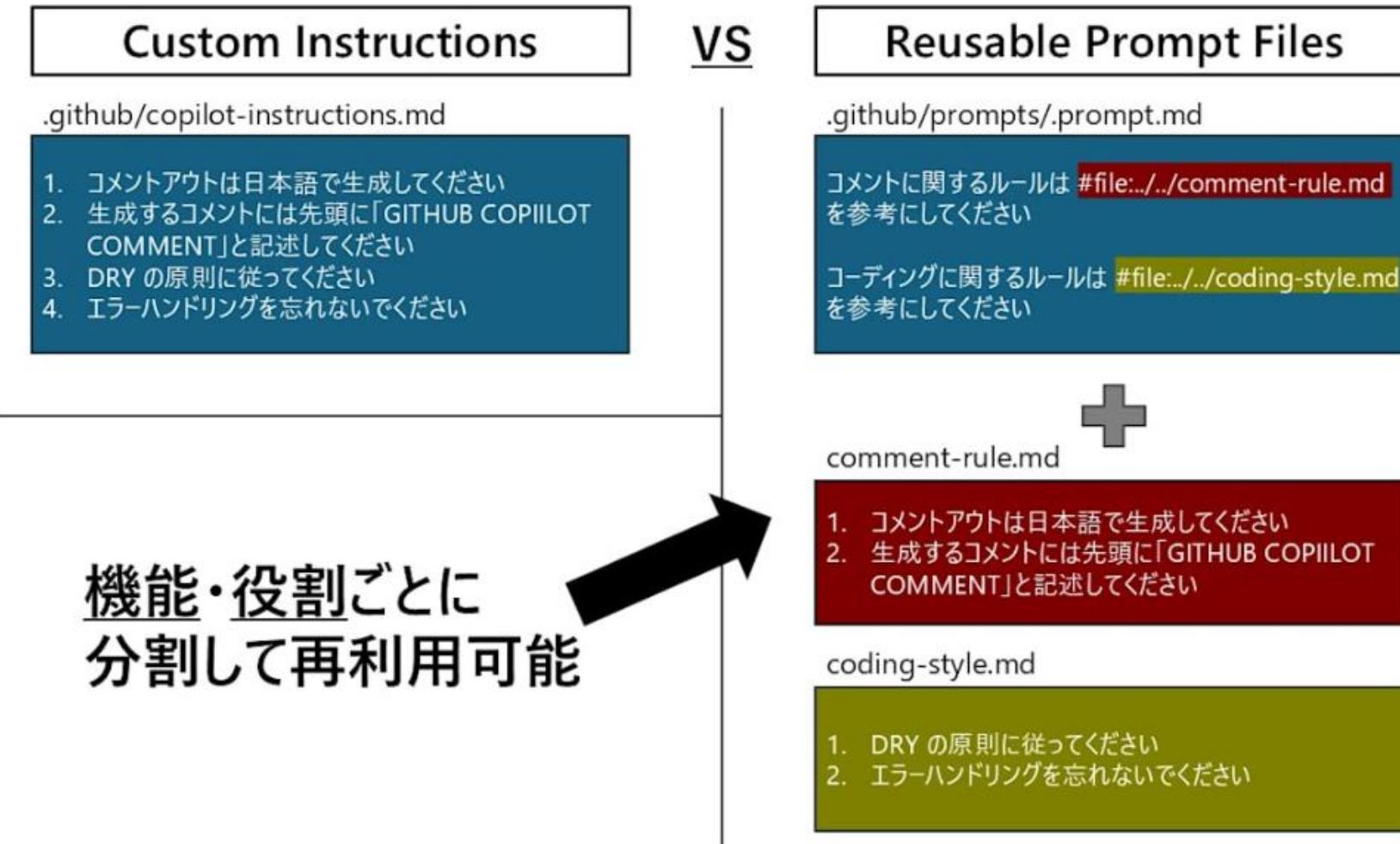
- GitHubリポジトリレベルのカスタマイズ: リポジトリのルートに`.github/copilot-instructions.md`ファイルを作成
- 個人レベルのカスタマイズ: GitHubの画面右上のCopilotアイコンをクリックして「Customize GitHub Copilot > Personal Instructions」から設定
- VSCodeの設定で、コード生成・テスト生成・コードレビュー・コミットメッセージ生成のカスタムインストラクションを設定

[Custom instructions for GitHub Copilot in VS Code](#)

[Copilot のカスタマイズ - GitHub Docs](#)

■参考 「Reusable Prompt Files」により、プロンプトを複数ファイルに分割して定義し、プロンプトファイルを再利用できるようになった。

2025/2/6～



[Vision, agent mode, next edit suggestions, and more for GitHub Copilot in VS Code January release \(v0.24\) - GitHub Changelog](#)

図の出典: カスタムインストラクションの一歩先！GitHub Copilotへの指示を分割管理しよう

# よくあるご質問

## ■ GitHub Copilotのトークン数の制限は？

→選択したプランとモデルによって決まりますが、たとえば入力4000トークン・出力4000トークンなどです。

### 転送率の制限 ☞

プレイグラウンドと無料の API の使用には、1 分あたりの要求数、1 日あたりの要求数、要求あたりのトークン数、同時要求数のレート制限があります。レート制限を受けた場合、さらに要求を行うには、該当したレート制限がリセットされるまで待つ必要があります。

低、高、埋め込みモデルのレート制限は異なります。使用しているモデルの種類を確認するには、GitHub Marketplace のモデルの情報を参照してください。

レート制限レベル	転送率の制限	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Azure OpenAI o1-mini	1分あたりの要求数	適用なし	2	3	3
	1日あたりの要求数	適用なし	12	15	20
	要求あたりのトークン数	適用なし	4000 入力、 4000 出力	4000 入力、 4000 出力	4000 入力、 4000 出力
同時要求数	適用なし	1	1	1	1

詳しくはこちらの  
ページを参照

■参考 VSCode + GPT-4oで64K / 128K コンテキストウィンドウが使用可能に。

2024/12/6～

- GitHub.com、GitHub Mobile、GitHub CLI、および公式にサポートされている IDE 上の Copilot Chat では、OpenAI GPT-4o を使用する際に **64k トークン ウィンドウ**が利用できるようになりました。
- Visual Studio Code Insiders で GitHub Copilot を使用している場合は、さらに大きな **128k のコンテキスト ウィンドウ**にアクセスできます。これは、OpenAI GPT-4o でサポートされている最大値です。
- ※トークン ウィンドウ = コンテキスト ウィンドウ = 入力トークンと出力トークンの合計値。

# よくあるご質問

- GitHub Copilotに意図したコードをうまく生成させるには?  
→関連ファイルをエディターで開いておく、チャットで #file を使用して関連ファイルを明示的に指定する、などの操作で、GitHub Copilotに関連ファイルを認識させると、GitHub Copilotはその内容を参考にコードを生成することができる。

# よくあるご質問

- GitHub Copilotに関する認定試験や資格はありますか？

→はい。GitHub社は、「GitHub Copilot 認定試験」を提供しています。

※詳しくは本講義の最後に解説



# よくあるご質問

- GitHub Copilotが生成したコードの著作権・所有権はどうなりますか？

→GitHub Copilotはただのツールであり、著作権・所有権は主張しません。エディターやIDEを使ってコードを書いた場合と同様に、コードは常にコードを書いた人間に属します。

# よくあるご質問

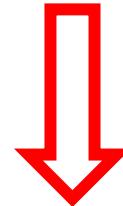
■ GitHub Copilot利用におけるベストプラクティス集はありますか？

→はい。GitHubドキュメント (<https://docs.github.com>) 内に、「ベストプラクティス」や「プロンプトエンジニアリング」（良いプロンプトの書き方）が記載されています。

[Copilot Chat のプロンプトエンジニアリング - GitHub Docs](#)

[GitHub Copilot の使用についてのベストプラクティス - GitHub Docs](#)

その他のご不明点  
(特に企業での利用における懸念点など)  
については、  
「GitHub Copilot Trust Center」に情報がまとまっています。ご確認ください。



[GitHub Copilot Trust Center - GitHub Resources](#)

<https://resources.github.com/ja/copilot-trust-center/>

企業でのGitHub Copilotの導入の推進方法については、GitHubドキュメントの「Copilotの導入を推進する」ページをご確認ください。

The screenshot shows a navigation sidebar on the left with sections like 'GitHub Copilot' and 'Copilot の導入を推進する'. The main content area has a title '会社での Copilot の導入の推進' and text explaining the process for promoting Copilot in an organization.

**会社での Copilot の導入の推進**

Copilot の導入を推進するための効果的な有効化プロセスを計画する方法について説明します。

効果的な有効化プロセスは、organization への Copilot の導入を推進するために不可欠です。このプロセスを organization のニーズと目標に合わせて調整し、チームが Copilot の効果的な使い方を理解するのに役立つよう設計する必要があります。

有効化プロセスは、フィードバックと結果に基づいて進化する可能性があります。プロセスのレビューと更新を定期的に行って、organization のニーズが満たされ続けるようにする必要があります。

GitHub Copilot の有効化プロセスは、次のステージに分ることができます。

- ライセンスの付与
- ユーザーによる環境の設定のサポート
- Copilot の効果的な使用のサポート

# モジュール1 GitHub Copilotの概要

- GitHub とは ?
- GitHub Copilot とは ?
- GitHub Copilot のマスコットキャラクター
- GitHub Copilot は無料で利用可能
- GitHub Copilot の使用を開始する手順
- GitHub Copilot の基本的な動作: 提案
- GitHub Copilot Chat とは ?
- GitHub Copilot ではどのようなことができるのか ?
- GitHub Copilot と連携できる開発ツール
- よくあるご質問
- まとめ

# モジュール1まとめ

- **GitHub Copilot** は「AIペアプログラマ」であり、プログラマのコードの記述作業を支援する。
- Visual Studio Codeなど、さまざまな開発ツールで利用できる。
- 利用するには**GitHubアカウント**が必要。
- 無料でも利用できるが、コード提案が月2000回まで・チャットメッセージが月50回まで、といった制限がある。より上位のサブスクリプションプランに変更が可能。
- 基本的には、**GitHub Copilot** は、コードの続きを提案してくれる。タブキーを押すことで提案を確定できる。
- **GitHub Copilot Chat**では、チャット形式でAIに質問したり、コードの説明や修正案をもらったりできる。

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する

# モジュール2



## GitHub Copilot を使用してドキュメントを生成する

1 時間 10 分 残り • モジュール • 1/9 ユニットが完了しました

このモジュールでは、Visual Studio Code 用の GitHub Copilot Chat 拡張機能を使用して、コードの説明、プロジェクトドキュメント、オンラインコードコメントドキュメントを生成する方法について説明します。

# モジュール2

- ドキュメント（ドキュメントコメント）とは？
- ドキュメントの必要性
- ドキュメントの例
- インラインチャットを使ってドキュメントを書く
- スマートアクションを使ってドキュメントを書く
  - メソッドを右クリック
  - メソッド名をクリック→きらめきアイコン
- 複数の候補の表示
- まとめ

# ドキュメント（ドキュメントコメント）とは？

- ・メソッドなどに対する説明文
- ・C#の場合は **XML形式** で記述される

```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
2 references
public bool IsPrime(int candidate)
{
```

# ドキュメントの必要性

- ・ドキュメントがないと、他のプログラマがコードを理解したりメンテナンスしたりするのが難しくなる
  - ・コードを書いた本人ですら、時間が経つと、自分が書いたコードの理解が難しくなる場合がある
- ・一般的に、プログラムコードを書く場合は、コードそのものだけではなく、そのドキュメントも書いておくことが求められる
- ・ただし、きちんとドキュメントを書くのはそれなりに大変。
- ・GitHub Copilotを活用して、ドキュメントをすばやく生成しましょう！

## ■参考: C#のコメントの種類

XMLの「ドキュメントコメント」ではない普通のコメント（説明文）を記述する場合

XMLの「ドキュメントコメント」を記述する場合

```
// コメント
```

```
/*
 * 複数行のコメント
 */
```

```
/// ドキュメントコメント
```

```
/** 
 * 複数行のドキュメントコメント
 */
```

## ■ ドキュメント（ドキュメントコメント）の例

```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
2 references
public bool IsPrime(int candidate)
{
```

summary:  
メソッドの説明文

## ■ ドキュメント（ドキュメントコメント）の例

```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

2 references

```
public bool IsPrime(int candidate)
{
```

param:  
引数の説明文

## ■ ドキュメント（ドキュメントコメント）の例

```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

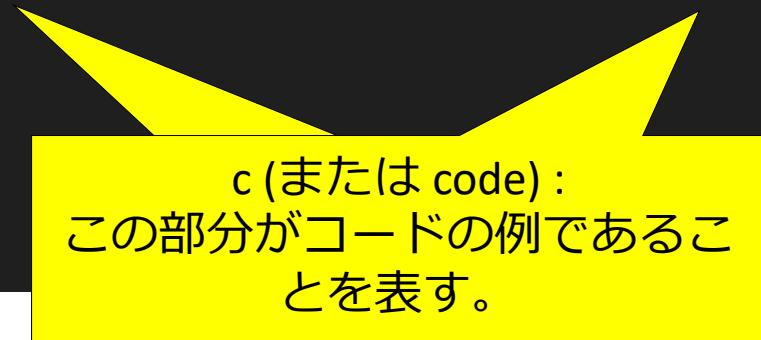
2 references

```
public bool IsPrime(int candidate)
{
```

returns:  
メソッドの戻り値の説明文

## ■ ドキュメント（ドキュメントコメント）の例

```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
2 references
public bool IsPrime(int candidate)
{
```



c (または code) :  
この部分がコードの例であることを表す。

- ドキュメントコメントが書かれたメソッド名にカーソルを当てると  
**メソッドの説明文や戻り値の説明文**がポップアップ表示される。  
コードリーディングの際に便利。

```
bool PrimeService.IsPrime(int candidate)  
指定された数が素数かどうかを判断します。  
nputIsNotP  
Returns:  
= new Prim  
meService.IsPrime(input);  
lt);
```



説明が表示されるので  
コードが理解しやすい！

- コーディングの途中では、引数を記述するタイミングで、  
**引数の説明文**がpopupアップ表示される。  
コーディングの際に便利。

```
public void IsPrime_Input  
{  
    var primeService = ne  
    bool result = primeSe  
    primeService.IsPrime()
```

bool PrimeService.IsPrime(int candidate)  
 素数かどうかを確認する数。  
 指定された数が素数かどうかを判断します。

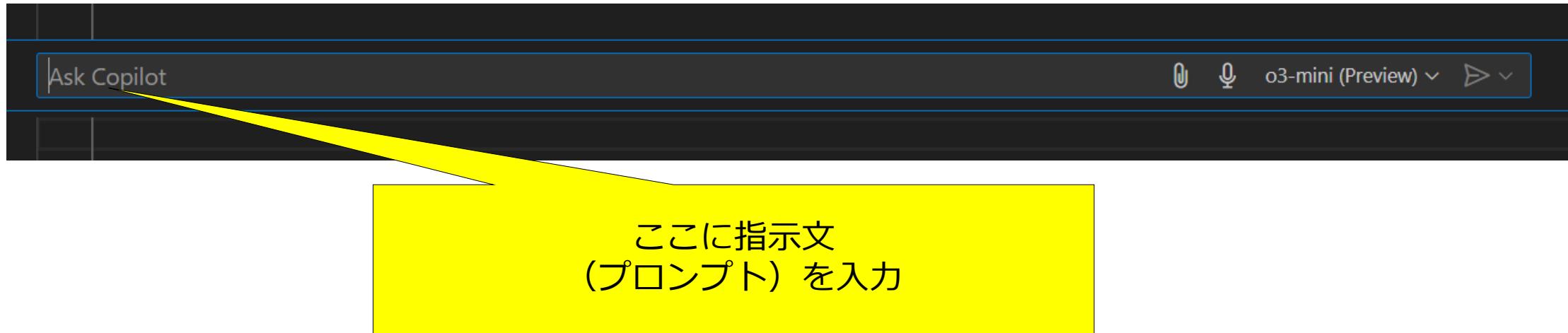
引数の説明が表示されるので、  
ここに何を書けばよいのかわかりやすい！

# モジュール2

- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

# インラインチャット

- [Ctrl + I]
- エディター内でCopilotにすばやく指示を出せる



[AI-powered chat conversations with GitHub Copilot](#)

[IDE で GitHub Copilot に質問する - GitHub Docs](#)

## ■ ドキュメントを生成したいメソッド全体を選択

```
5
6 2 references
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

```
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

この場合、行番号6番でマウス左ボタンを押し、  
そのまま行番号20番までマウスを動かして  
マウスを離すとうまく選択できる

## ■ [Ctrl + I]を押してINLINEチャットを起動

Ask Copilot



GPT 4o



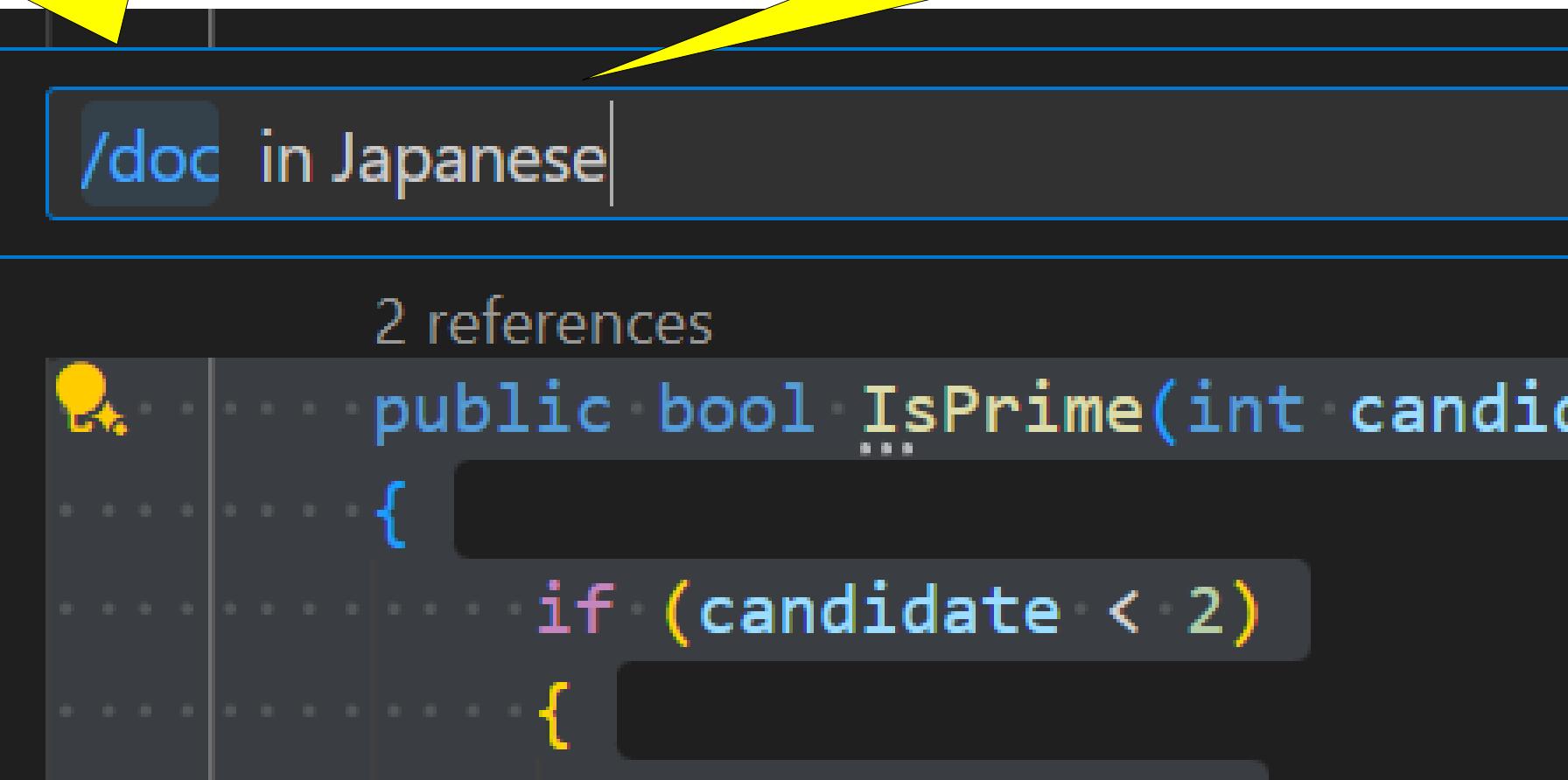
2 references

```
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

## ■インラインチャットに **/doc in Japanese** と入力してエンター

/docのようなスラッシュで始まるものを  
スラッシュコマンド、またはアクションという。  
/docは「ドキュメントを書いて」という指示。

追加で「in Japanese」を指定して、日本語のドキュ  
メントコメントが生成されるようにしている



The screenshot shows a dark-themed code editor or IDE interface. In the top navigation bar, there is a blue button containing the text '/doc in Japanese'. Below the navigation bar, the main code editor area displays the following code:

```
2 references
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
```

A yellow lightbulb icon is visible on the left side of the code editor, indicating a potential issue or suggestion.

## ■ ドキュメントが生成された。

確定するには Accept (同意する) をクリック (または Ctrl + Enter) 。  
取り消したい場合は Discard (閉じる) をクリック (または ESC) 。

The screenshot shows the Microsoft Copilot interface. At the top, it says "/doc in Japanese". Below that is a search bar labeled "Ask Copilot". Underneath the search bar are two buttons: "Accept" (highlighted in blue) and "Discard". To the right of these buttons are icons for refresh and dropdown.

The main area displays generated documentation and code. The documentation includes:

```
/// <summary>
/// 指定された数が素数かどうかを判定します。
/// </summary>
/// <param name="candidate">判定対象の整数。</param>
/// <returns>素数であれば true、そうでなければ false。</returns>
```

Below the documentation, it says "2 references". The generated C# code is:

```
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

# モジュール2

- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

# スマートアクションとは？

- ・プロンプト（指示文）を書かずに、メニューから機能を選択することで、GitHub Copilotに指示を出す方法。
- ・GitHub Copilotでよく使われる機能がすぐに呼び出せる
- ・他の方法に比べて簡単に利用できる

# モジュール2

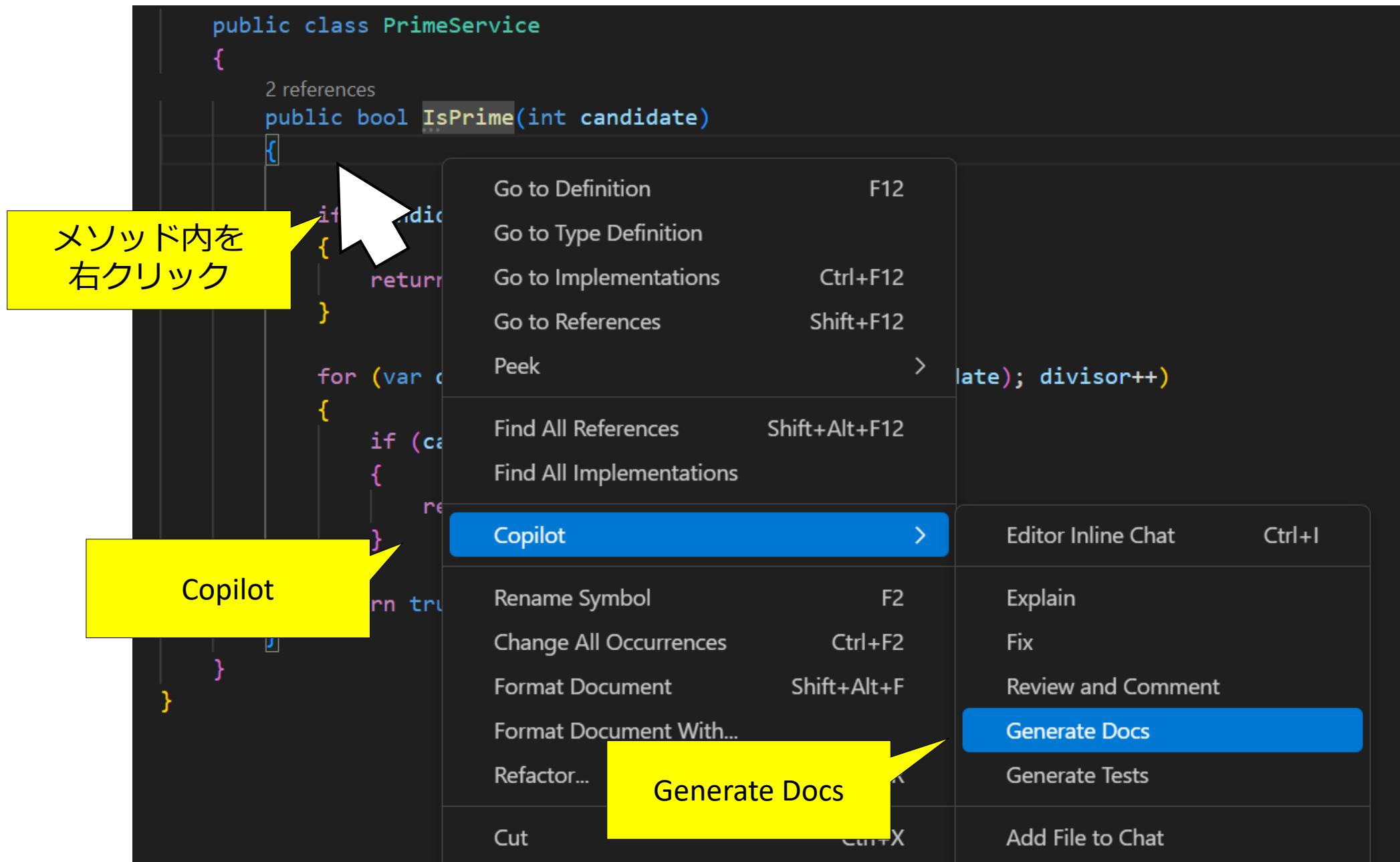
- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

## ■あるメソッドが定義されている。これに対するドキュメントを追記する。

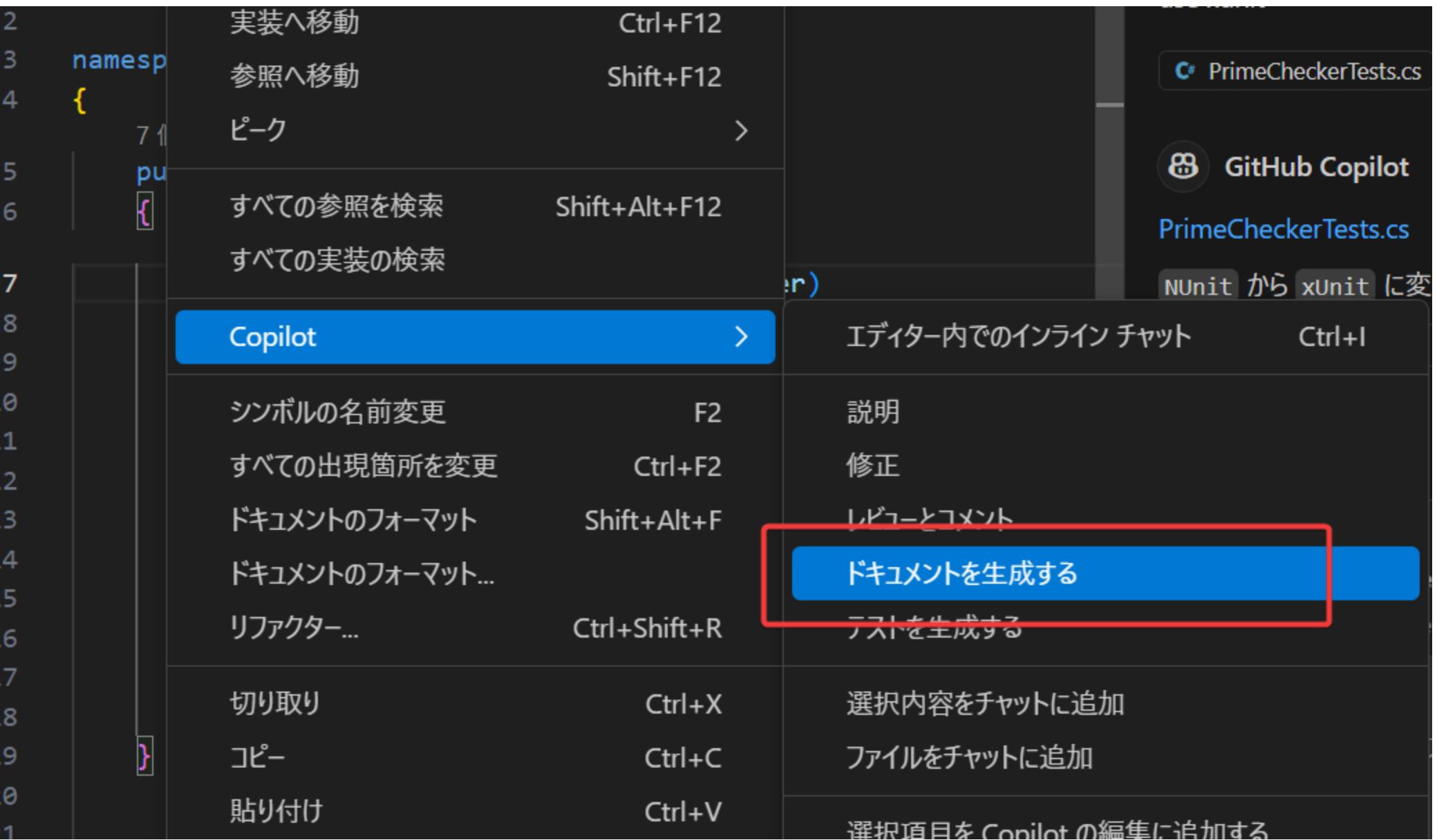
```
2 references
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }

    for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

## ■スマートアクション「Generate Docs」を呼び出す



## ■日本語UIの場合



## ■ GitHub Copilotがドキュメントを生成してくれる。Accept（同意する）で確定。

The screenshot shows the GitHub Copilot interface. At the top, there's a header with a profile icon, the path '/doc', and a search bar labeled 'Ask Copilot'. To the right of the search bar are icons for pinned items, a microphone, 'GPT 4o', and a dropdown menu. Below the header, there are buttons for 'Accept' (highlighted in blue), 'Discard', and a refresh icon.

The main area displays a Java-like code snippet with generated documentation:

```
    /// <summary>
    /// Determines whether the specified number is a prime number.
    /// </summary>
    /// <param name="candidate">The number to check for primality.</param>
    /// <returns>
    /// <c>true</c> if the specified number is a prime number; otherwise, <c>false</c>.
    /// </returns>
```

Below the documentation, the code starts with a method definition:

```
public boolean IsPrime(int candidate)
{
```

Then it includes an if statement:

```
    if (candidate < 2)
    {
        return false;
    }
```

Finally, it shows a for loop:

```
    for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
    {
        if (candidate % divisor == 0)
        {
            // ...
        }
    }
```

## ■これは、インラインチャットの「/doc」と同じ。

The screenshot shows the Microsoft Copilot interface. At the top, there's a toolbar with a logo, the text '/doc' (which is highlighted with a red box), 'Ask Copilot', 'Accept', and 'Discard'. A yellow arrow points from the text 'スマートアクションの「Generate Docs」を選ぶと、インラインチャットで「/doc」と入力したのと同じ状態となる' to the '/doc' button. The main area displays a C# code snippet with XML documentation comments:

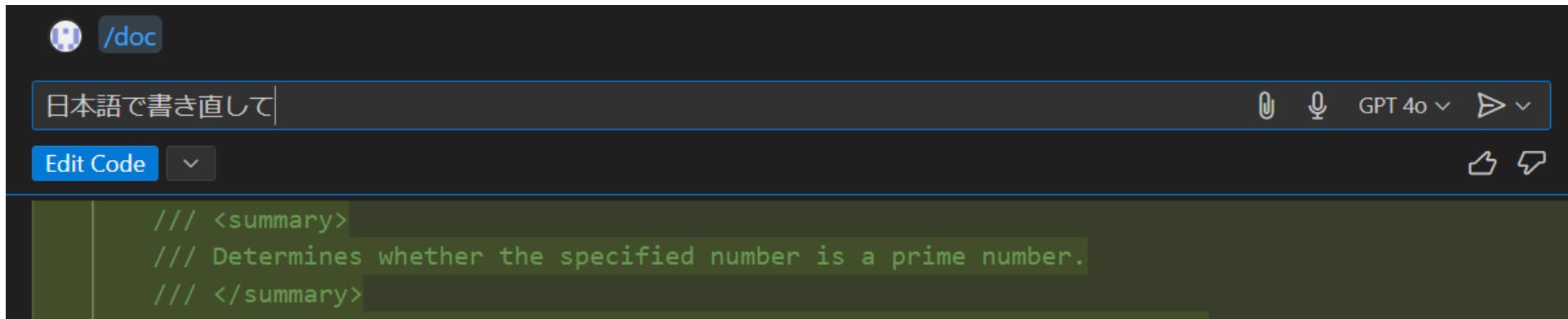
```
    /// <summary>
    /// Determines whether the specified number is a prime number.
    /// </summary>
    /// <param name="candidate">The number to check for primality.</param>
    /// <returns>
    /// <c>true</c> if the specified number is a prime number; otherwise, <c>false</c>.
    /// </returns>
    public bool IsPrime(int candidate)
    {
        if (candidate < 2)
        {
            return false;
        }

        for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
        {
            if (candidate % divisor == 0)
            {
                return false;
            }
        }

        return true;
    }
```

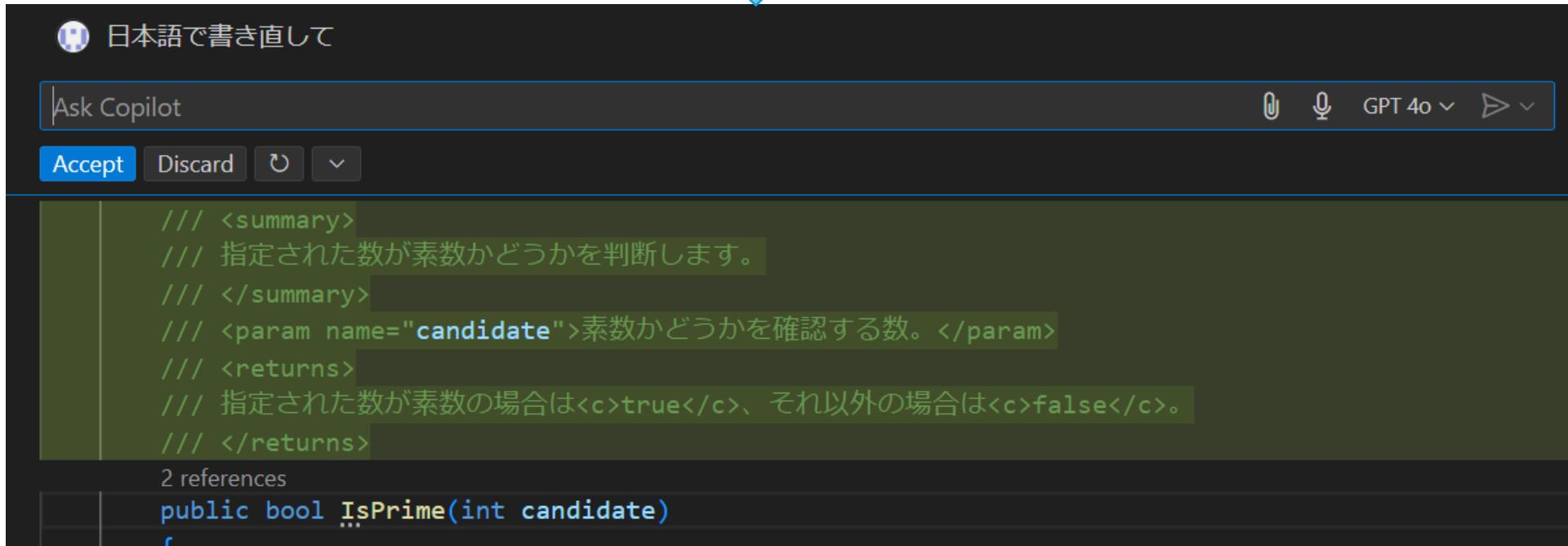
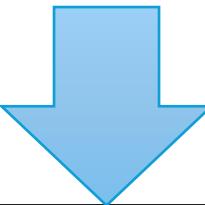
Below the code, it says '2 references'.

## ■ ここでさらにプロンプトを入力してドキュメントに変更を加えさせることもできる



The screenshot shows the Microsoft Copilot interface in a browser window. At the top, there's a header with a profile icon, the text '/doc', and various status icons like battery level and signal strength. Below the header is a search bar containing the Japanese text '日本語で書き直して'. To the right of the search bar are buttons for 'GPT 4o' and a refresh arrow. Underneath the search bar is a button labeled 'Edit Code' with a dropdown arrow. The main content area displays a code snippet with three Japanese comments:

```
/// <summary>
/// Determines whether the specified number is a prime number.
/// </summary>
```



This screenshot shows the Microsoft Copilot interface after a prompt has been entered. The search bar now contains '日本語で書き直して' and includes an 'Ask Copilot' button. Below the search bar are buttons for 'Accept' (highlighted in blue) and 'Discard'. The main content area shows the same code snippet, but with more detailed Japanese documentation added:

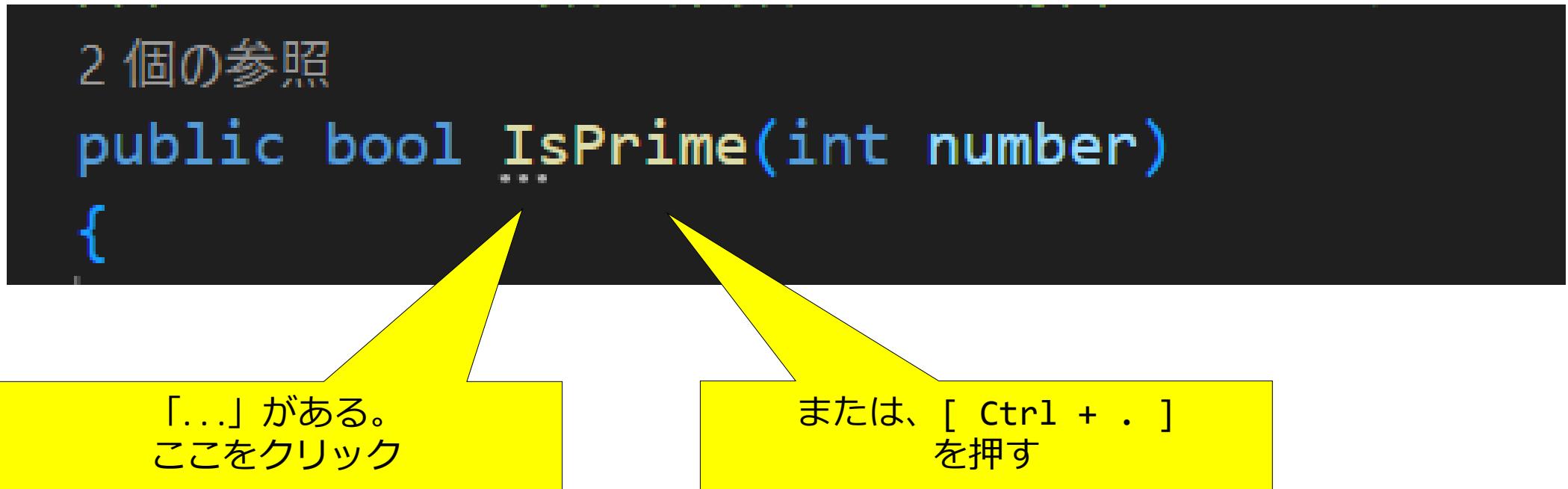
```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

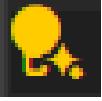
At the bottom of the code editor, there are two small boxes indicating '2 references' and a partial view of the code starting with 'public bool IsPrime(int candidate)'.

# モジュール2

- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

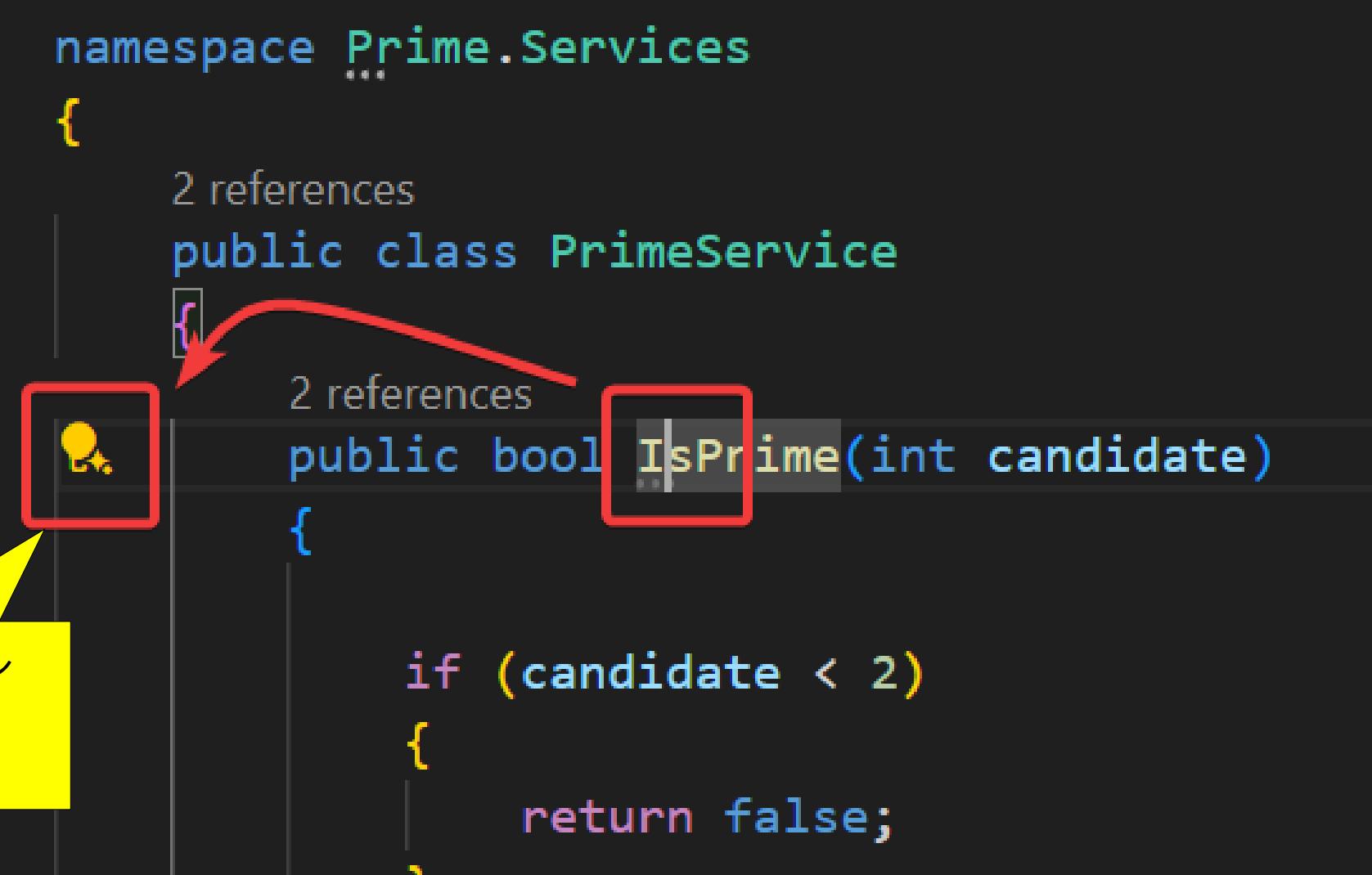
## ■メソッド名の最初の文字の下に「...」が表示されている



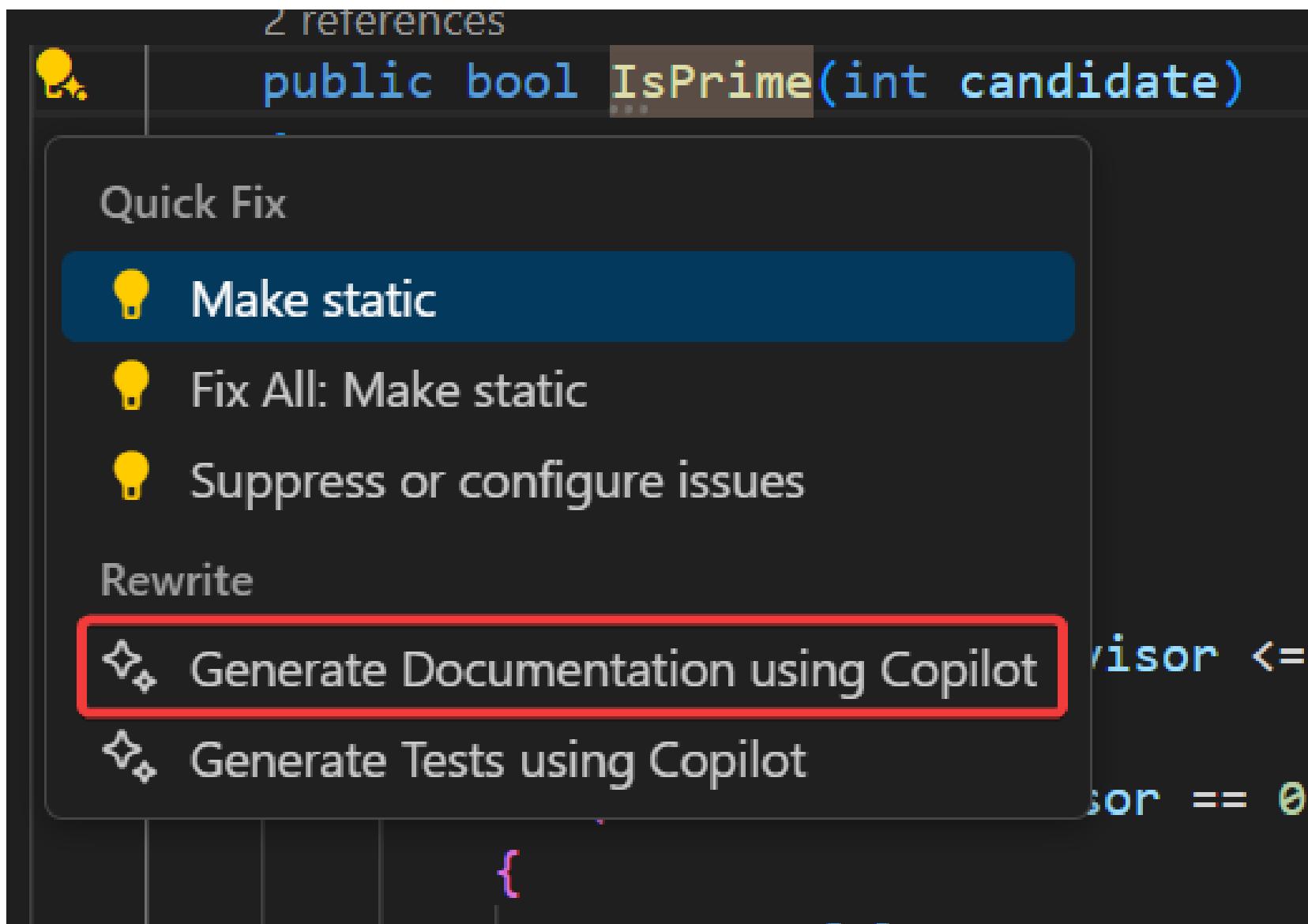
■ 続いて「きらめきアイコン」  をクリック

```
namespace Prime.Services
{
    2 references
    public class PrimeService
    {
        2 references
        public bool IsPrime(int candidate)
        {
            if (candidate < 2)
            {
                return false;
            }
        }
    }
}
```

きらめきアイコン  
(sparkle icon)  
(twinkle icon)



■表示されるメニューで「Generate Documentation using Copilot」を選択



## ■GitHub Copilotがドキュメントを生成してくれる。Acceptで確定。

The screenshot shows the GitHub Copilot interface. At the top, there's a header with a profile icon, the path '/doc', and a search bar labeled 'Ask Copilot'. To the right of the search bar are icons for pinned items, a microphone, 'GPT 4o', and a dropdown menu. Below the header, there are buttons for 'Accept' (highlighted in blue), 'Discard', and a refresh icon.

The main area displays a Java code snippet with generated Javadoc comments:

```
    /**
     * <summary>
     * Determines whether the specified number is a prime number.
     * </summary>
     * <param name="candidate">The number to check for primality.</param>
     * <returns>
     * <c>true</c> if the specified number is a prime number; otherwise, <c>false</c>.
     * </returns>
    
```

Below the comments, the code starts with a public method declaration:

```
public boolean IsPrime(int candidate)
```

Then it includes an if statement:

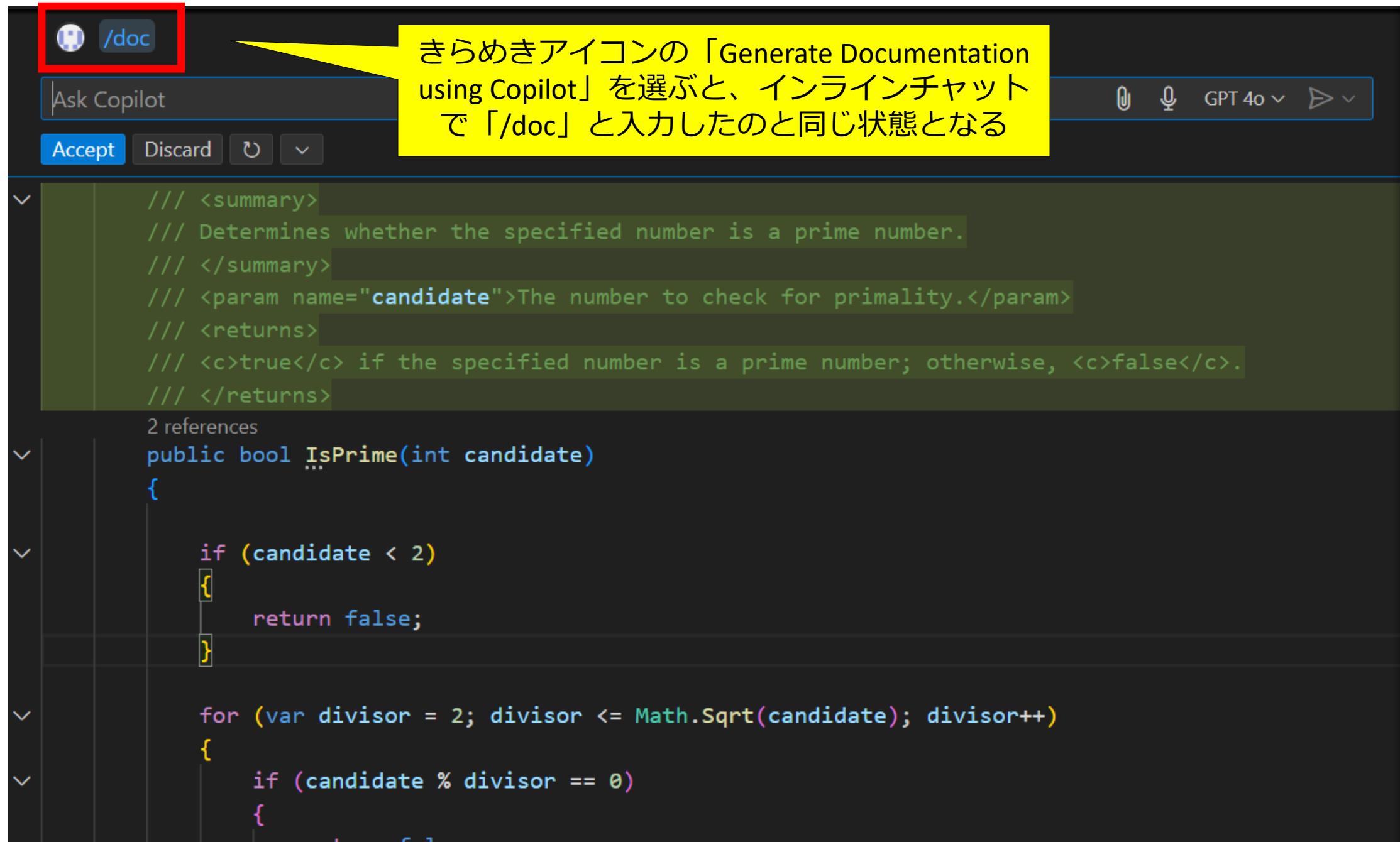
```
if (candidate < 2)
```

And a for loop:

```
for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
```

The code uses color-coded syntax highlighting for keywords like 'public', 'boolean', 'if', 'for', and 'var', and for identifiers like 'IsPrime', 'candidate', 'divisor', and 'Math.Sqrt'.

## ■ これも、インラインチャットの「/doc」と同じ。



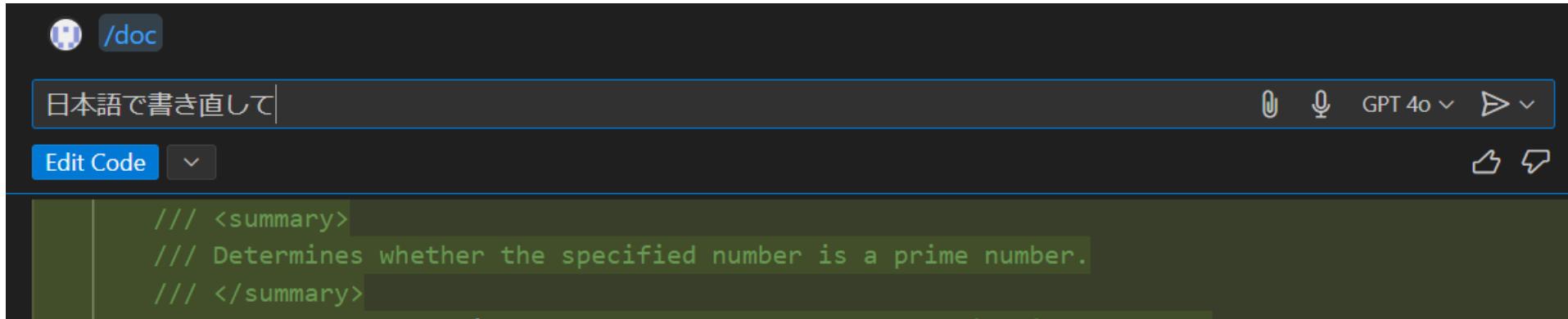
The screenshot shows the Microsoft Copilot interface. At the top left, there is a blue circular icon with a white 'H' and a red button labeled '/doc'. A yellow arrow points from this button to a yellow callout box containing Japanese text. The text in the callout box reads: 'きらめきアイコンの「Generate Documentation using Copilot」を選択すると、インラインチャットで「/doc」と入力したのと同じ状態となる' (Selecting the 'Generate Documentation using Copilot' icon will result in the same state as entering '/doc' in the inline chat). Below the header, there are buttons for 'Accept' and 'Discard'. The main area displays a code editor with C# code. The code defines a method 'IsPrime' that checks if a given number is prime. The code includes XML documentation blocks (///) describing the method's purpose, parameters, and return value.

```
    /// <summary>
    /// Determines whether the specified number is a prime number.
    /// </summary>
    /// <param name="candidate">The number to check for primality.</param>
    /// <returns>
    /// <c>true</c> if the specified number is a prime number; otherwise, <c>false</c>.
    /// </returns>
    public bool IsPrime(int candidate)
    {
        if (candidate < 2)
        {
            return false;
        }

        for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
        {
            if (candidate % divisor == 0)
            {
                return false;
            }
        }

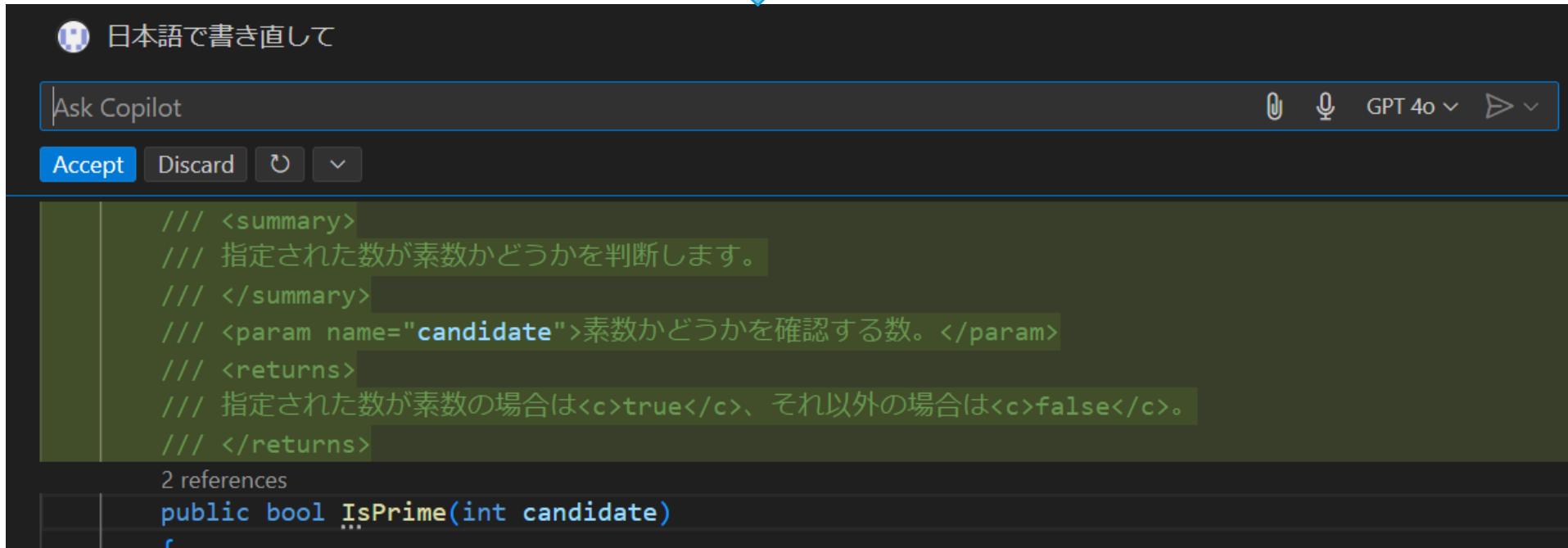
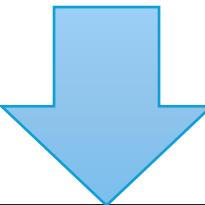
        return true;
    }
```

## ■ ここでさらにプロンプトを入力してドキュメントに変更を加えさせることもできる



The screenshot shows the Microsoft Copilot interface in a browser window. At the top, there's a header with a profile icon, the text '/doc', and various status icons like battery level and signal strength. Below the header is a search bar containing the Japanese text '日本語で書き直して'. To the right of the search bar are buttons for 'GPT 4o' and a refresh arrow. Underneath the search bar is a button labeled 'Edit Code' with a dropdown arrow. The main content area displays a code snippet with three Japanese comments:

```
/// <summary>
/// Determines whether the specified number is a prime number.
/// </summary>
```



This screenshot shows the Microsoft Copilot interface after a prompt has been entered. The search bar now contains '日本語で書き直して' and includes an 'Ask Copilot' button. Below the search bar are buttons for 'Accept' (highlighted in blue) and 'Discard'. The main content area shows the same code snippet, but with more detailed Japanese documentation added:

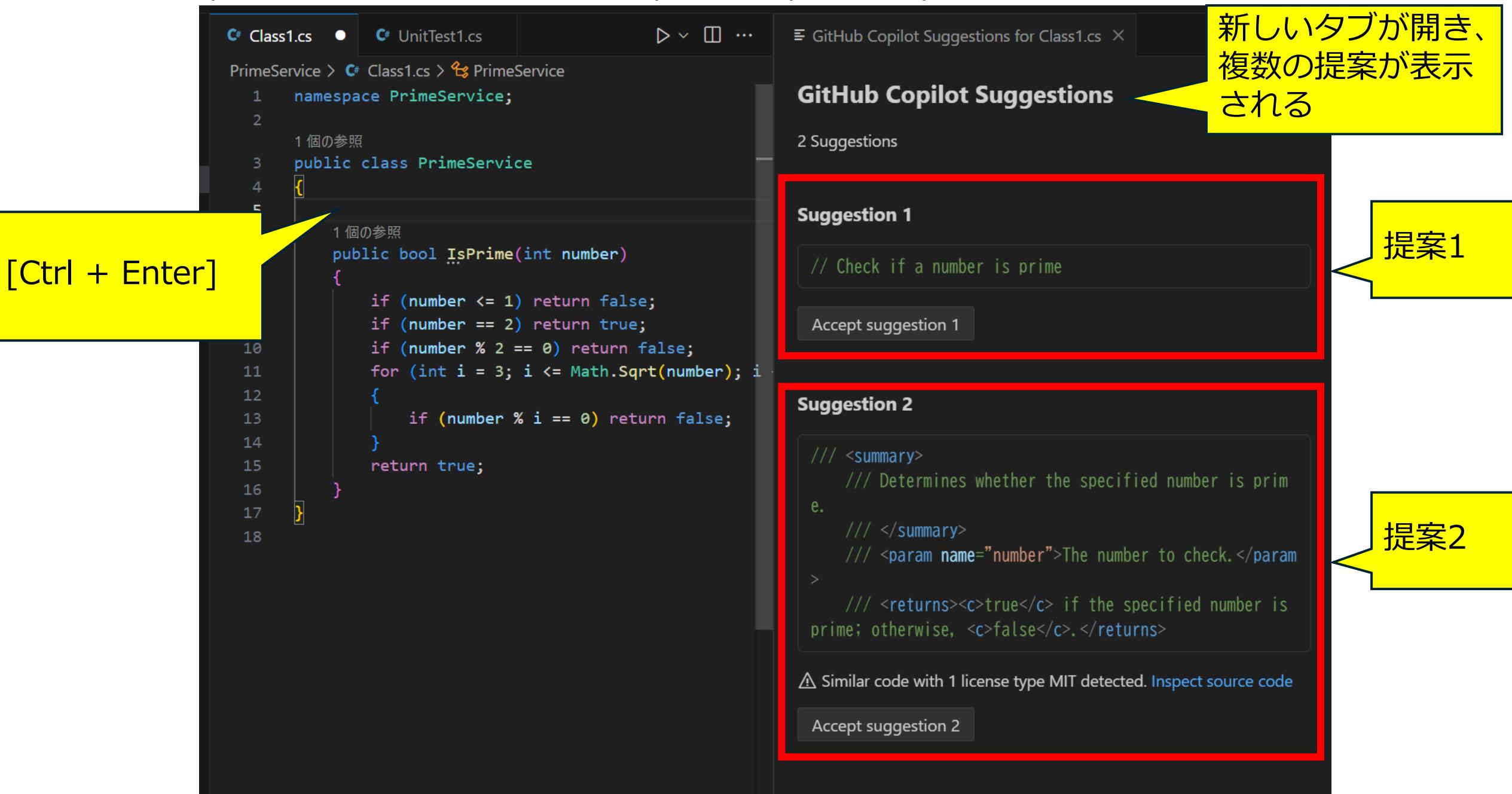
```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

At the bottom of the code editor, there are two small boxes indicating '2 references' and a partial view of the code starting with 'public bool IsPrime(int candidate)'.

# モジュール2

- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

## ■ GitHub Copilot 複数の候補の表示 (open completions panel)



# モジュール2

- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
  - ・メソッドを右クリック
  - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

# モジュール2 まとめ

- ・メソッドなどのドキュメント（ドキュメントコメント）をGitHub Copilotで生成できる
- ・ドキュメントを書くことでコードが理解しやすくなる
- ・コードにドキュメントを追加する方法
  - ・インラインチャットを起動し、`/doc` アクションを使用する
  - ・スマートアクションを使用する
    - ・メソッド名をクリックして「Generate Docs」を選ぶ
    - ・メソッドを右クリックして「きらめきアイコン」から「Generate Documentation using Copilot」を選ぶ
  - ・スマートアクションのドキュメント生成機能を選択すると、インラインチャットで`/doc` アクションを実行した場合と同じ事が起こる。
  - ・[Ctrl + Enter]で複数の候補を別タブに表示し、そこから一つを選ぶ

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する

# モジュール3



## GitHub Copilot を使用してコード機能を開発する

1 時間 10 分 残り • モジュール • 1/9 ユニットが完了しました

このモジュールでは、GitHub Copilot と GitHub Copilot Chat の候補を使用して新しいコードを作成する方法について説明します。オートコンプリートとコード更新の候補は、Visual Studio Code 用の GitHub Copilot 拡張機能を使用して生成、管理、実装されます。

# モジュール3

## GitHub Copilotを活用したコードの開発

- 方法(1) **コード提案[Alt + ¥]**を利用。ドキュメントを書き、コード提案により、ドキュメントに対応したコードを生成する。
- 方法(2) **インラインチャット[Ctrl + I]**を利用。開発したいコードの仕様などを指定して、コードを生成する
- 方法(3) **チャットビュー[Ctrl + Alt + I]**を利用。チャットで、開発したいコードの仕様などを入力して、コードを生成する
- まとめ

## ■ ドキュメントの最初の部分を書く

```
/// <summary>入力された数が素数か
```

■ すると、続きの部分の候補が表示される。Tabキーを押して確定。

```
/// <summary>入力された数が素数かどうかを判定します。</summary>
```

もし表示されない場合は  
Alt + ¥  
(trigger inline suggestion)

■改行すると、次の行の候補が表示されるので、またTabキーを押して確定

```
/// <summary>入力された数が素数かどうかを判定します。</summary>
/// <param name="candidate">素数かどうかを判定する数</param>
/// <returns>入力された数が素数の場合は true、そうでない場合は false</returns>
```

## ■メソッドのドキュメントが書けた。

```
/// <summary>入力された数が素数かどうかを判定します。</summary>
/// <param name="candidate">素数かどうかを判定する数</param>
/// <returns>入力された数が素数の場合は true、そうでない場合は false</returns>
```



## ■さらに、その下に、コード候補が表示される

```
/// <summary>入力された数が素数かどうかを判定します。</summary>
/// <param name="candidate">素数かどうかを判定する数</param>
/// <returns>入力された数が素数の場合は true、そうでない場合は false</returns>
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

もし表示されない場合は  
Alt + ¥  
(trigger inline suggestion)

## ■ タブキーを押して候補を確定。

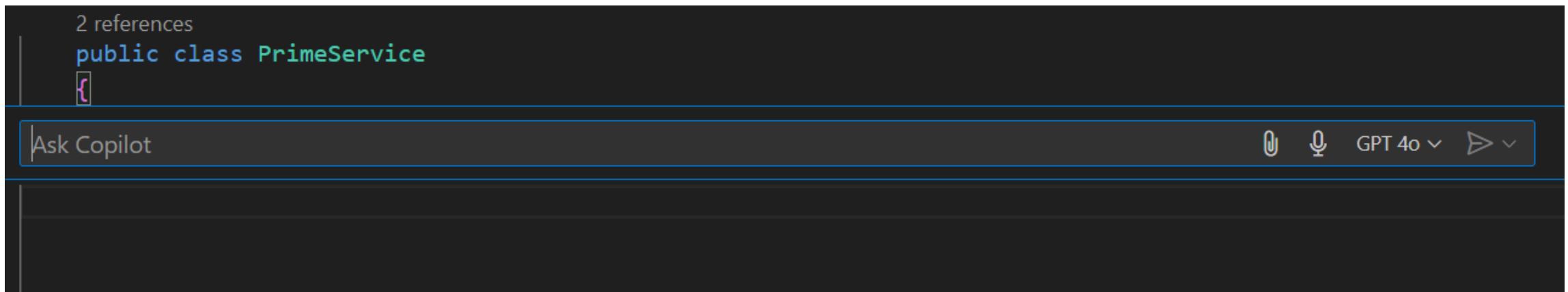
```
/// <summary>入力された数が素数かどうかを判定します。</summary>
/// <param name="candidate">素数かどうかを判定する数</param>
/// <returns>入力された数が素数の場合は true、そうでない場合は false</returns>
2 references
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }
    for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
    {
        if (candidate % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
```

# モジュール3

## GitHub Copilotを活用したコードの開発

- ・方法(1) **コード提案[Alt + ¥]**を利用。ドキュメントを書き、コード提案により、ドキュメントに対応したコードを生成する。
- ・方法(2) **インラインチャット[Ctrl + I]**を利用。開発したいコードの仕様などを指定して、コードを生成する
- ・方法(3) **チャットビュー[Ctrl + Alt + I]**を利用。チャットで、開発したいコードの仕様などを入力して、コードを生成する
- ・まとめ

■エディター内で[Ctrl + I]を押して「インラインチャット」を開く



## ■プロンプトを入力してEnterキーを押す

```
2 references
3     public class PrimeService
4     {
5
6
7
8
9
10
11 }
```

入力された数が素数であるかどうかを判定するメソッド IsPrime を書いて。適切なドキュメントも書いて。

0 0 GPT 4o ▶

# ■コードとドキュメントが生成された。Acceptをクリックして確定。

The screenshot shows the Microsoft Copilot interface with the following details:

- Input:** A task prompt asking to write a method to determine if a number is prime, including a summary and parameter documentation.
- Output:** Generated C# code for a `IsPrime` method. The code includes:
  - Method signature: `public bool IsPrime(int number)`
  - Initial checks for numbers less than 2 and 2 itself.
  - A check for even numbers greater than 2.
  - A loop from 3 to  $\sqrt{\text{number}}$  checking for divisibility by odd numbers.
  - A final `return true;` statement.
- User Interface:** The interface includes a toolbar with "Ask Copilot", "Accept" (which is highlighted), "Discard", and other icons. The status bar shows "o3-mini (Preview)".

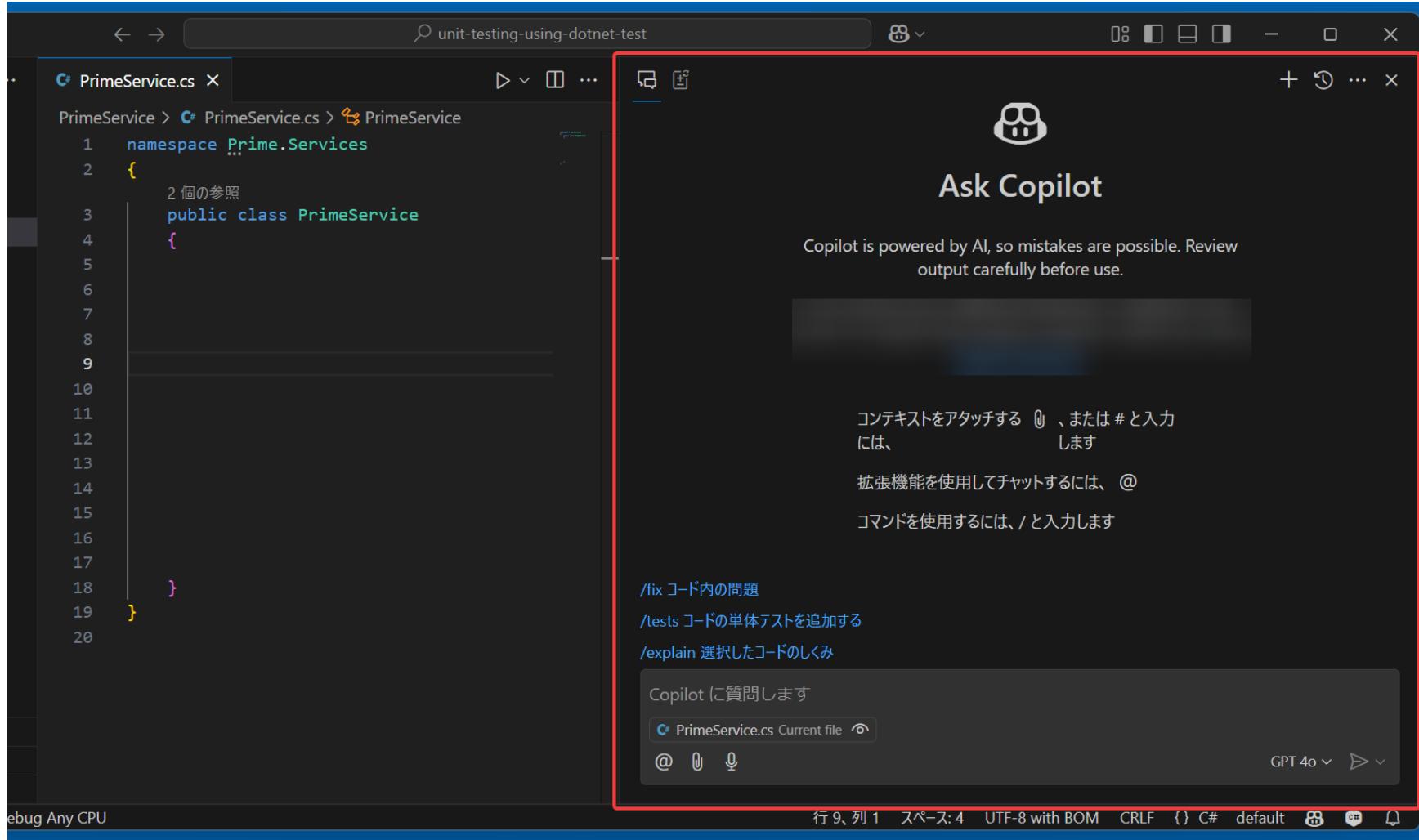
# モジュール3

## GitHub Copilotを活用したコードの開発

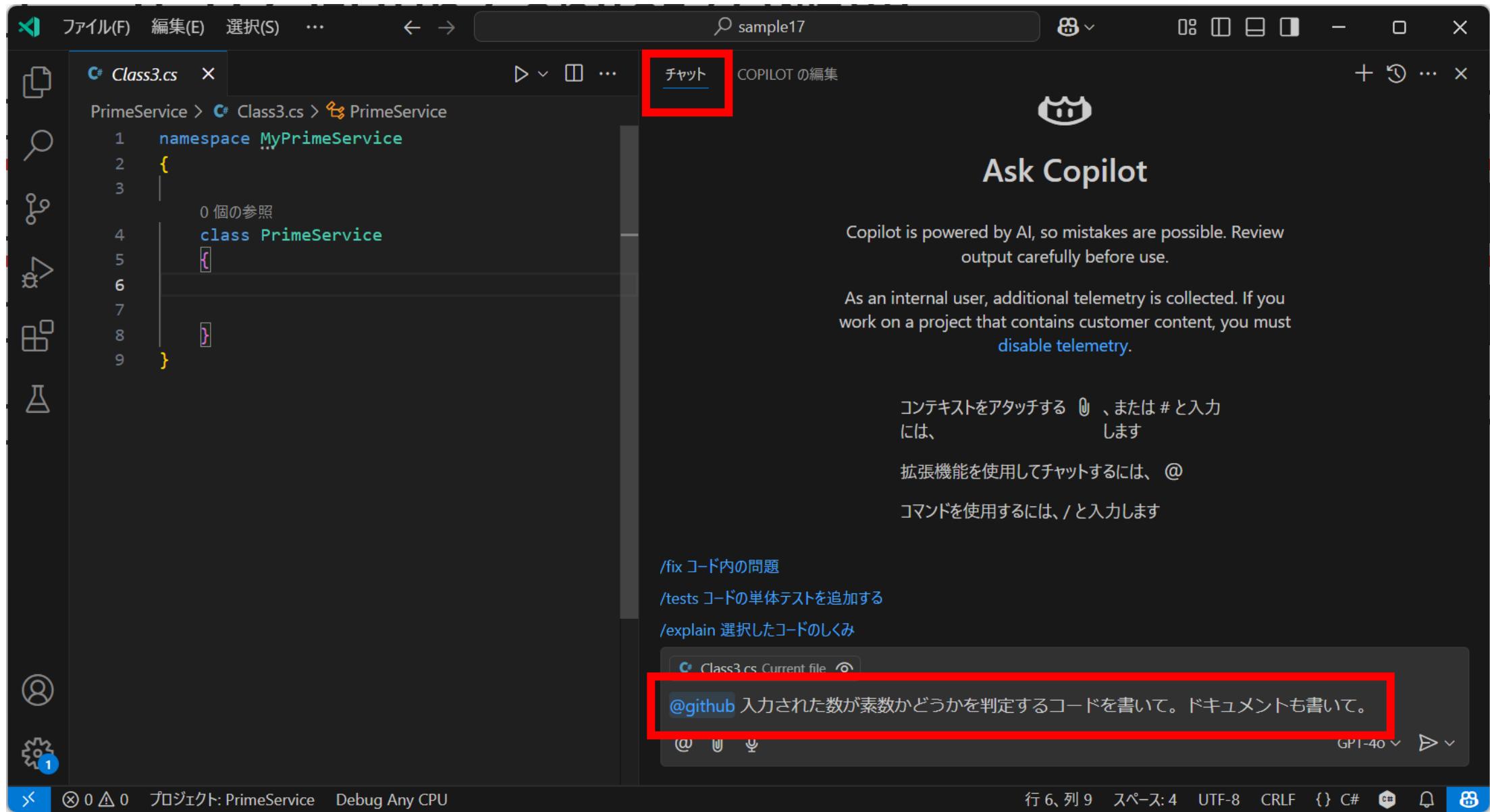
- ・方法(1) **コード提案[Alt + ¥]**を利用。ドキュメントを書き、コード提案により、ドキュメントに対応したコードを生成する。
- ・方法(2) **インラインチャット[Ctrl + I]**を利用。開発したいコードの仕様などを指定して、コードを生成する
- ・方法(3) **チャットビュー[Ctrl + Alt + I]**を利用。チャットで、開発したいコードの仕様などを入力して、コードを生成する
- ・まとめ

# チャットビュー

- [Ctrl + Alt + I]を押し、エディターの右側に表示される「チャットビュー」内でCopilotとチャット形式で対話



## 「チャットビュー」での作業の例



## ■ 「チャットビュー」での作業の例

The screenshot shows a Visual Studio interface with the following details:

- Title Bar:** unit-testing-using-dotnet-test
- Left Panel:** C# PrimeService.cs (PrimeService.cs > PrimeService.cs > PrimeService)
- Code Editor:** Shows the beginning of a PrimeService class definition.
- GitHub Copilot Chat View:** A red box highlights this area.
  - Header:** GitHub Copilot (with 1 reference used)
  - Text:** A message from GitHub Copilot suggesting to add an IsPrime method to the PrimeService class and providing a sample implementation with documentation comments.
  - Code Preview:** A snippet of generated code is shown, including a summary and parameters for the IsPrime method.
- Bottom Status Bar:** 行 9、列 1 スペース: 4 UTF-8 with BOM CRLF {} C# default

A yellow callout box points to the GitHub Copilot section with the following text:

GitHub Copilotがチャット内で  
コードとドキュメントを生成  
してくれた  
※この時点ではまだファイル  
には反映されていない

## ■「チャットビュー」での作業の例

The screenshot shows the Visual Studio Code interface with the GitHub Copilot extension open in the Chat View. The left pane displays a C# file named `PrimeService.cs` with the following code:

```
1  namespace Prime.Services
2  {
3      2 個の参照
4      public class PrimeService
5      {
6
7
8
9
10
11
12
13
14
15
16
17
18    }
19  }
```

The right pane shows the Chat View with the following interaction:

GitHub Copilot: 入力された数が素数かどうかを判定するコードを書いて。ドキュメントも書いて。  
PrimeService.cs:1-20

GitHub Copilot: 1 参照 使用済み

以下のように、入力された数が素数かどうかを判定するメソッド `IsPrime` を `PrimeService` クラスに追加します。また、メソッドのドキュメントコメントも追加します。

namespace Prime.Services  
{  
 public class PrimeService  
 {  
 /// <summary>  
 /// 指定された数が素数かどうかを返す  
 /// </summary>  
 /// <param name="number">  
 /// <returns>数が素数である場合は true、それ以外の場合は false。</returns>  
 }  
}

A yellow callout box points to the 'Apply in Editor' button (a small square icon) in the toolbar of the Chat View, which is highlighted with a red border. The text inside the callout box says: 「エディターで適用」ボタンをクリック (Click the 'Apply in Editor' button).

Copilot に質問します  
PrimeService.cs Current file

行 9、列 1 スペース: 4 UTF-8 with BOM CRLF {} C# default GPT 4o

# ■ 「チャットビュー」での作業の例

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `PrimeService.cs` file under the `PrimeService` project. A red box highlights the code area. On the right, the Chat View window is open, showing a conversation with Copilot. A yellow arrow points from the text "ファイルに、コードの提案が反映された。" (The code suggestion was reflected in the file) to the right-hand window. The Chat View contains the following text:

```
namespace Prime.Services
{
    public class PrimeService
    {
        /// <summary>
        /// 指定された数が素数かどうかを判定します。
        /// </summary>
        /// <param name="number">判定する整数。</param>
        /// <returns>数が素数である場合は true、それ以外の場合は false。</returns>
        public bool IsPrime(int number)
        {
            if (number <= 1) return false;
            if (number == 2) return true;
            if (number % 2 == 0) return false;

            var boundary = (int)Math.Floor(Math.Sqrt(number));

            for (int i = 3; i <= boundary; i += 2)
            {
                if (number % i == 0) return false;
            }

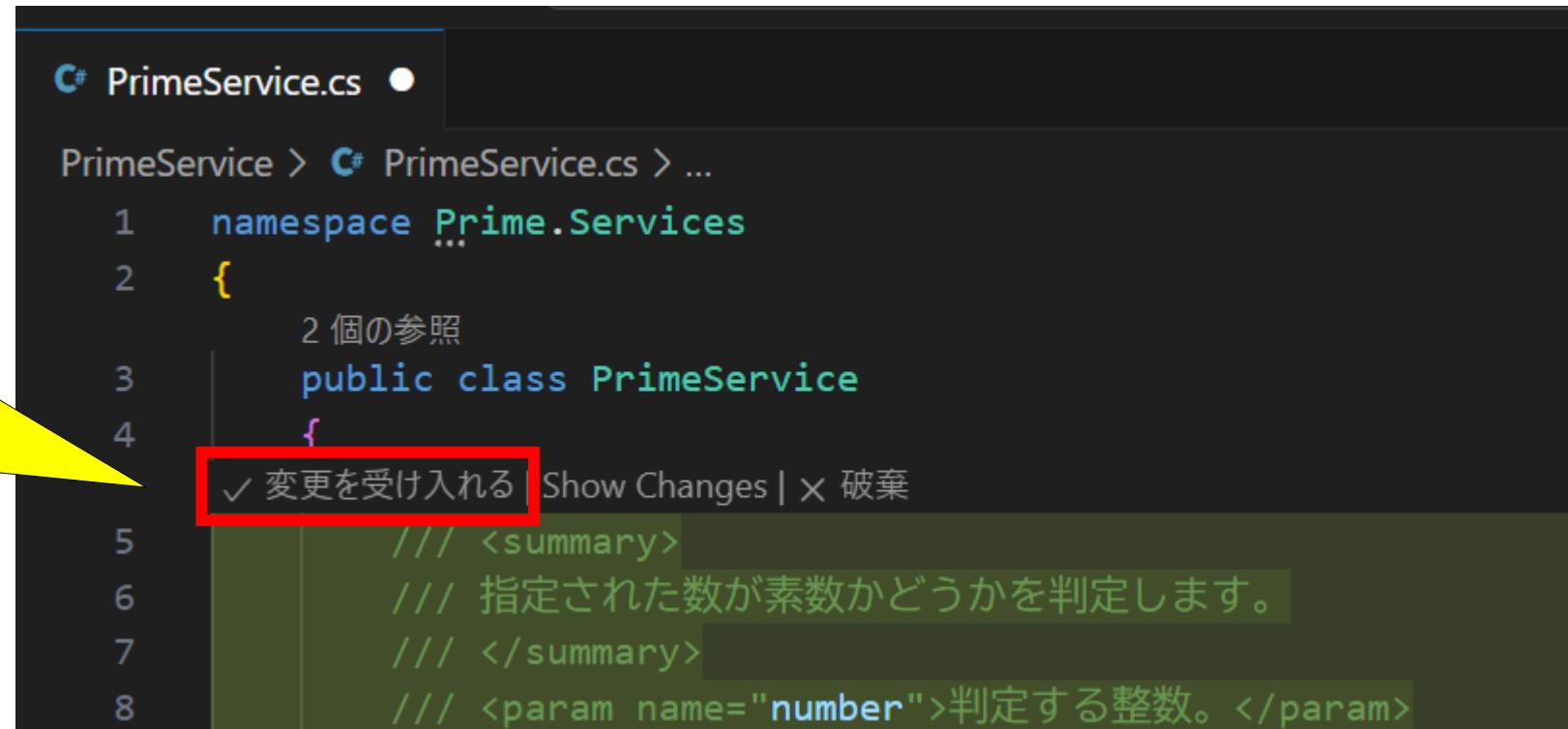
            return true;
        }
    }
}
```

Below the code, a message from Copilot reads: "以下のように実装できます。また、テストケースも教えてください。" (You can implement it like this. Also, please tell me about test cases.)

Copilot's input field says: "Copilotに質問します" (Ask Copilot). The status bar at the bottom indicates: 行 27、列 1 スペース: 4 UTF-8 with BOM CRLF {} C# 🎯 🔒

## ■ 「チャットビュー」での作業の例

問題がなければ、エディターで「変更を受け入れる」をクリック。



The screenshot shows a code editor window for a C# file named PrimeService.cs. The code defines a class PrimeService with a constructor and some documentation comments. A context menu is open at the bottom of the code area, with the '受け入れる' (Accept) option highlighted by a red box. The menu also includes 'Show Changes' and '破棄' (Discard) options.

```
C# PrimeService.cs
PrimeService > C# PrimeService.cs > ...
1 namespace Prime.Services
2 {
3     2 個の参照
4     public class PrimeService
5     {
6         /// <summary>
7         /// 指定された数が素数かどうかを判定します。
8         /// </summary>
9         /// <param name="number">判定する整数。</param>
```

# ■「チャットビュー」での作業の例

The screenshot shows the Visual Studio interface with the GitHub Copilot extension open in the Chat View. On the left, the code for `PrimeService.cs` is displayed:

```
1  namespace Prime.Services
2  {
3      2 個の参照
4      public class PrimeService
5      {
6          /// <summary>
7          /// 指定された数が素数かどうかを判定します。
8          /// </summary>
9          /// <param name="number">判定する整数。</param>
10         /// <returns>数が素数である場合は true、それ以外の場合は false。</returns>
11     2 個の参照
12     public bool IsPrime(int number)
13     {
14         if (number <= 1) return false;
15         if (number == 2) return true;
16         if (number % 2 == 0) return false;
17
18         var boundary = (int)Math.Floor(Math.Sqrt(number));
19
20         for (int i = 3; i <= boundary; i += 2)
21         {
22             if (number % i == 0) return false;
23         }
24
25         return true;
26     }
27 }
```

A yellow callout box in the bottom-left corner contains the text:

GitHub Copilotによる変更案が  
確定された。  
必要に応じてファイルを保存  
し、Ctrl + Alt + Iでチャット  
ビューを閉じる。

In the Chat View on the right, the task is defined as "Write code to determine if the input number is prime. Write a comment too." The Copilot suggestion is shown:

```
namespace Prime.Services
{
    public class PrimeService
    {
        /// <summary>
        /// 指定された数が素数かどうかを判定します。
        /// </summary>
        /// <param name="number">判定する整数。</param>
        /// <returns>数が素数である場合は true、それ以外の場合は false。</returns>
        public bool IsPrime(int number)
        {
            if (number <= 1) return false;
            if (number == 2) return true;
            if (number % 2 == 0) return false;

            var boundary = (int)Math.Floor(Math.Sqrt(number));

            for (int i = 3; i <= boundary; i += 2)
            {
                if (number % i == 0) return false;
            }

            return true;
        }
    }
}
```

Below the code, there is a message: "Please teach me test cases." At the bottom, there is a text input field for asking questions to Copilot.

# モジュール3

## GitHub Copilotを活用したコードの開発

- ・方法(1) **コード補完[Alt + ¥]**を利用。ドキュメントを書き、コード補完により、ドキュメントに対応したコードを生成する。
- ・方法(2) **インラインチャット[Ctrl + I]**を利用。開発したいコードの仕様などを指定して、コードを生成する
- ・方法(3) **チャットビュー[Ctrl + Alt + I]**を利用。チャットで、開発したいコードの仕様などを入力して、コードを生成する
- ・まとめ

# モジュール3 まとめ

- ・コードの**提案**や**チャット**を使って、C#などのプログラムコードをすばやく正確に記述することができる。
- ・方法(1) **コード提案[Alt + ¥]**を利用して、ドキュメントに対応したコードを生成する
- ・方法(2) **インラインチャット[Ctrl + I]**を起動して、コードの仕様などをプロンプトで指定して、コードを生成する
- ・方法(3) **チャットビュー[Ctrl + Alt + I]**を起動して、コードの仕様などをプロンプトで指定して、コードを生成する

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する
- ・補足: GitHub Copilotの最新機能のご紹介

# モジュール4

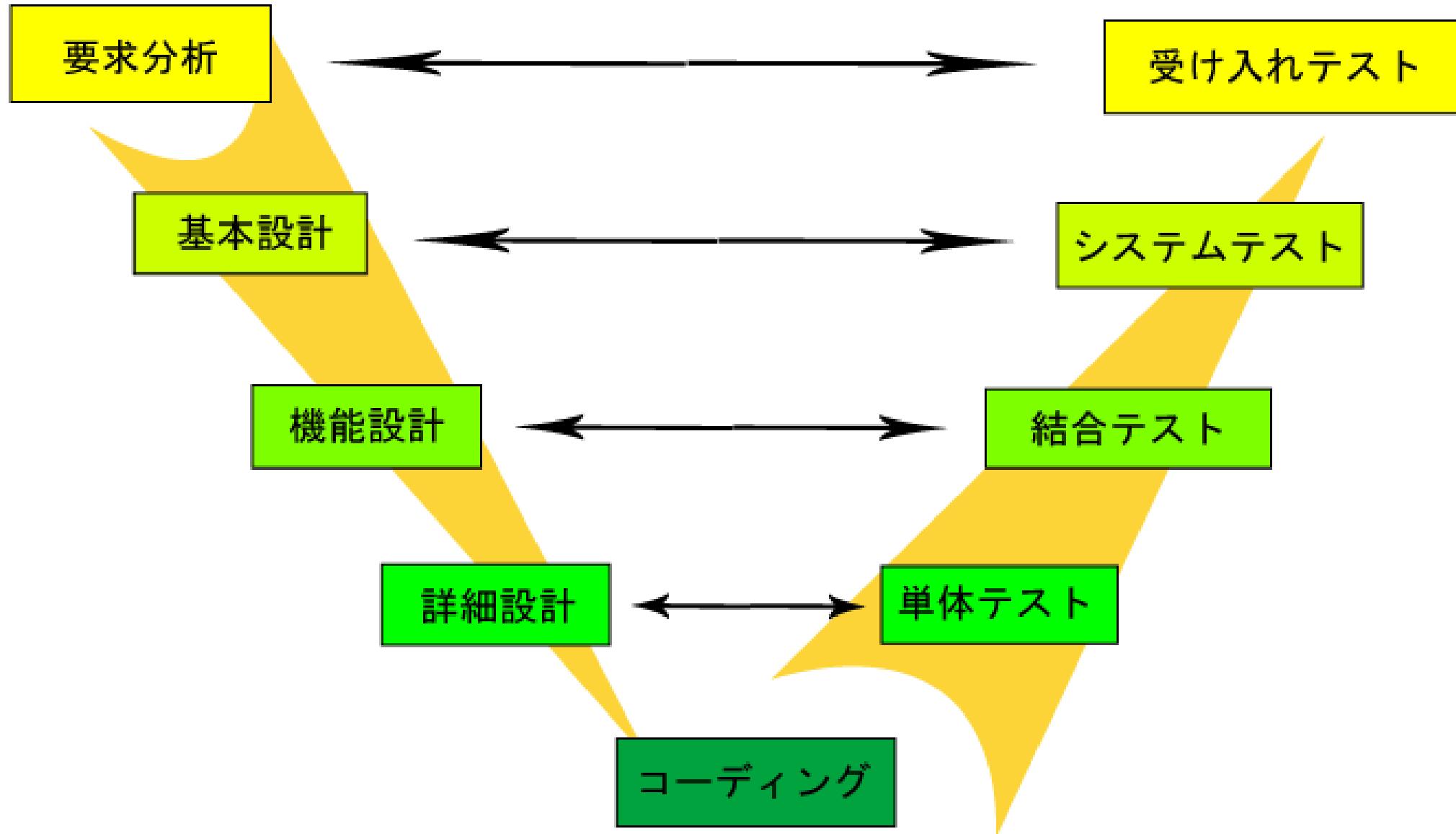


## [GitHub Copilot を使用して単体テストを開発する](#)

このモジュールでは、GitHub Copilot と GitHub Copilot Chat を使用して単体テストを作成する方法について説明します。

# モジュール4

- ソフトウェアテストの主な種類
- C#のテストフレームワーク
- 単体テストとは？
- xUnitを使用するソリューション/プロジェクトの準備
- xUnit 単体テストの実装例
- GitHub Copilotによる単体テストの生成
- 単体テストの実行例
- まとめ



# ソフトウェアテストの主な種類

- **単体テスト** (unit tests)

- メソッドなどの単位での動作確認

- **結合テスト** (integration tests)

- 複数のプログラムを組み合わせた動作確認

- **システム / エンドツーエンドテスト** (E2E tests)

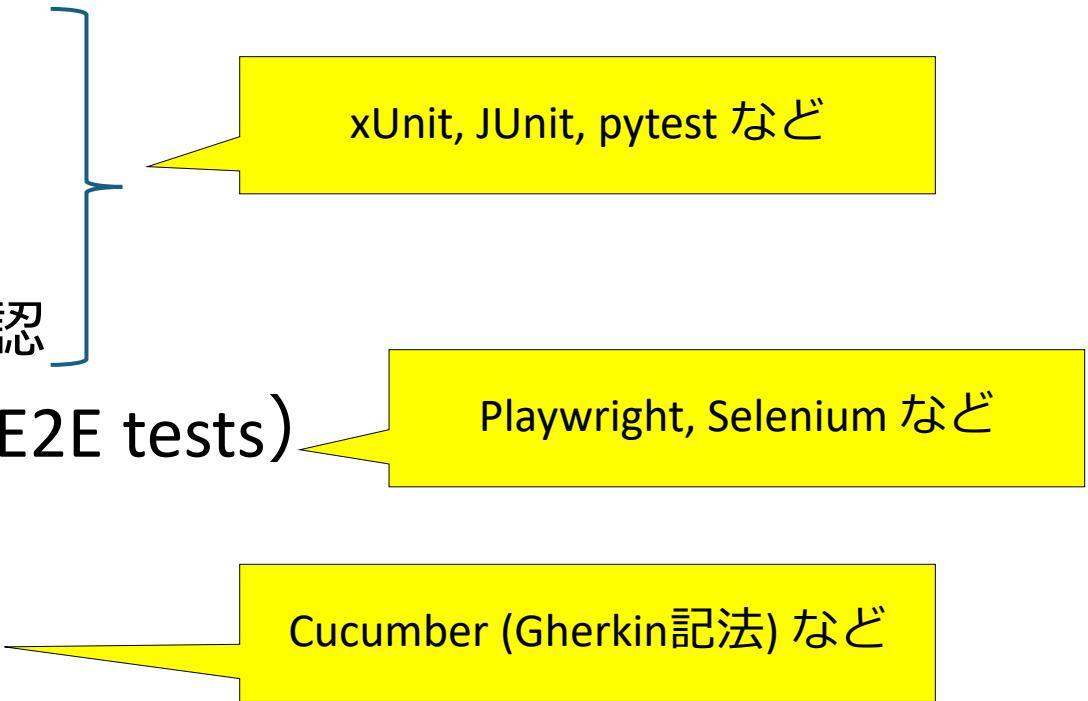
- システム全体の動作確認

- **受け入れテスト** (acceptance tests)

- ユーザーの要求を満たしているかの確認

- など

- GitHub Copilotは、これらすべてのテストコード生成が可能。



[Writing tests with GitHub Copilot - GitHub Docs](#)

[Create end-to-end tests for a webpage - GitHub Docs](#)

[ソフトウェアテスト - Wikipedia](#)

[Testing with GitHub Copilot \(YouTube\)](#)

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

# C#のテストフレームワーク

- xUnit.net (<https://xunit.net/>)
  - NUnit, MSTestと比べて様々な優れた点がある
  - GitHubのスター数やNuGetパッケージのダウンロード数が最も多い
  - C#のテストフレームワークとして最も人気がある
- NUnit (<https://nunit.org/>)
- MSTest (<https://github.com/microsoft/testfx>)
- など
- GitHub Copilotは、これらすべてのテストフレームワークに対応。

C# のテストフレームワークを MSTest から xUnit.net に乗り換えたい時にその理由を同僚に説明できるようにする #Microsoft - Qiita

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・**単体テストとは？**
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

# 単体テストとは？

- 単体テスト (unit tests)
  - メソッドなどの単位での動作確認
  - あるメソッドが仕様通りに正しく動くかどうかを（別のテストメソッドを使用して）確認する
  - 最も基礎的なテスト
- 単体テスト用のテストフレームワークを使用して実装
  - C#: xUnit.net など
  - python: pytest など
  - Java: JUnit など

# ここまでまとめ

- GitHub Copilotは、
  - さまざまなプログラミング言語に対応できる
  - さまざまなソフトウェアテスト（単体、結合、E2E、受け入れ）に対応できる。
  - さまざまなテストフレームワークのテストコードを生成できる。
- 本コースでは、C#のソースコードに対するxUnitテストフレームワークを使った単体テストの生成に焦点を当てて解説する。

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

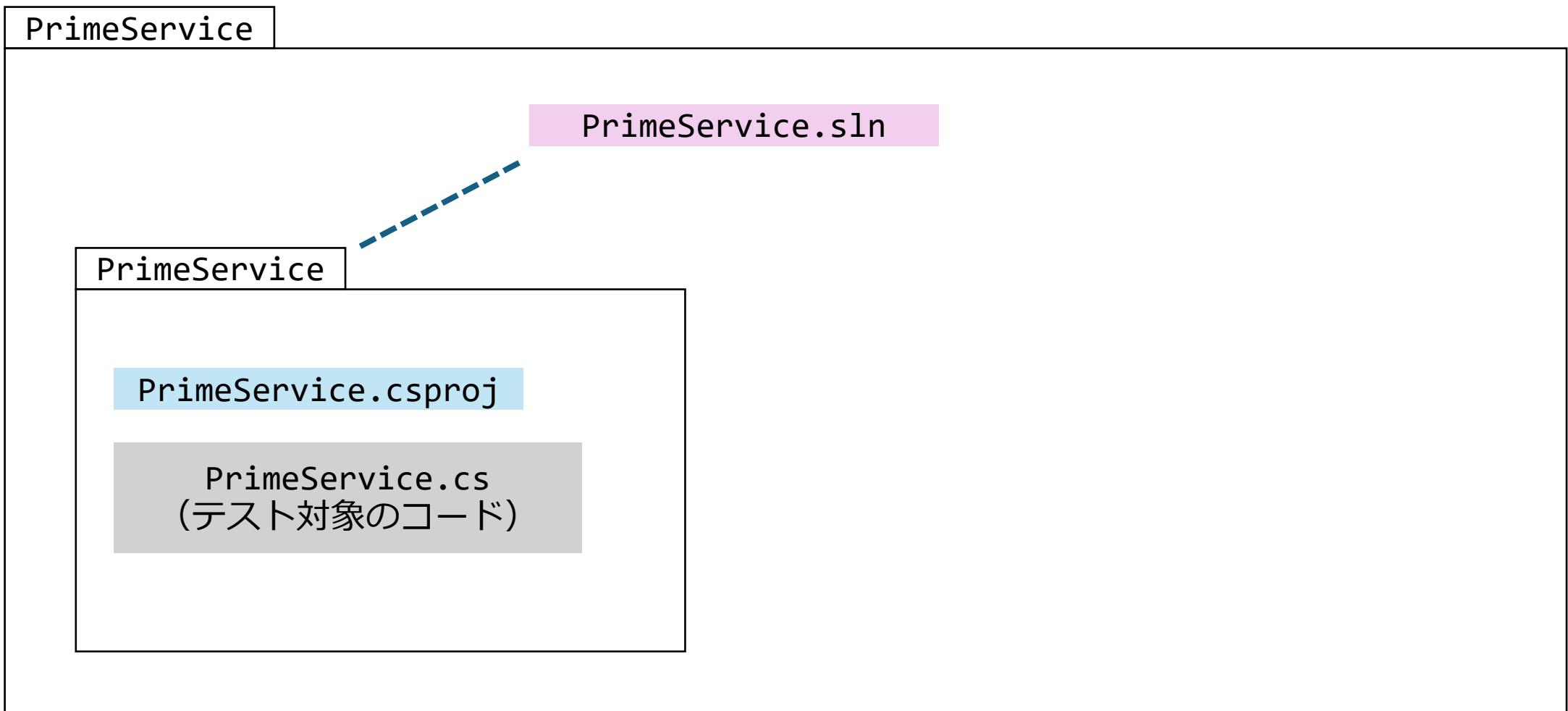
## ■ソリューションの作成

PrimeService

PrimeService.sln

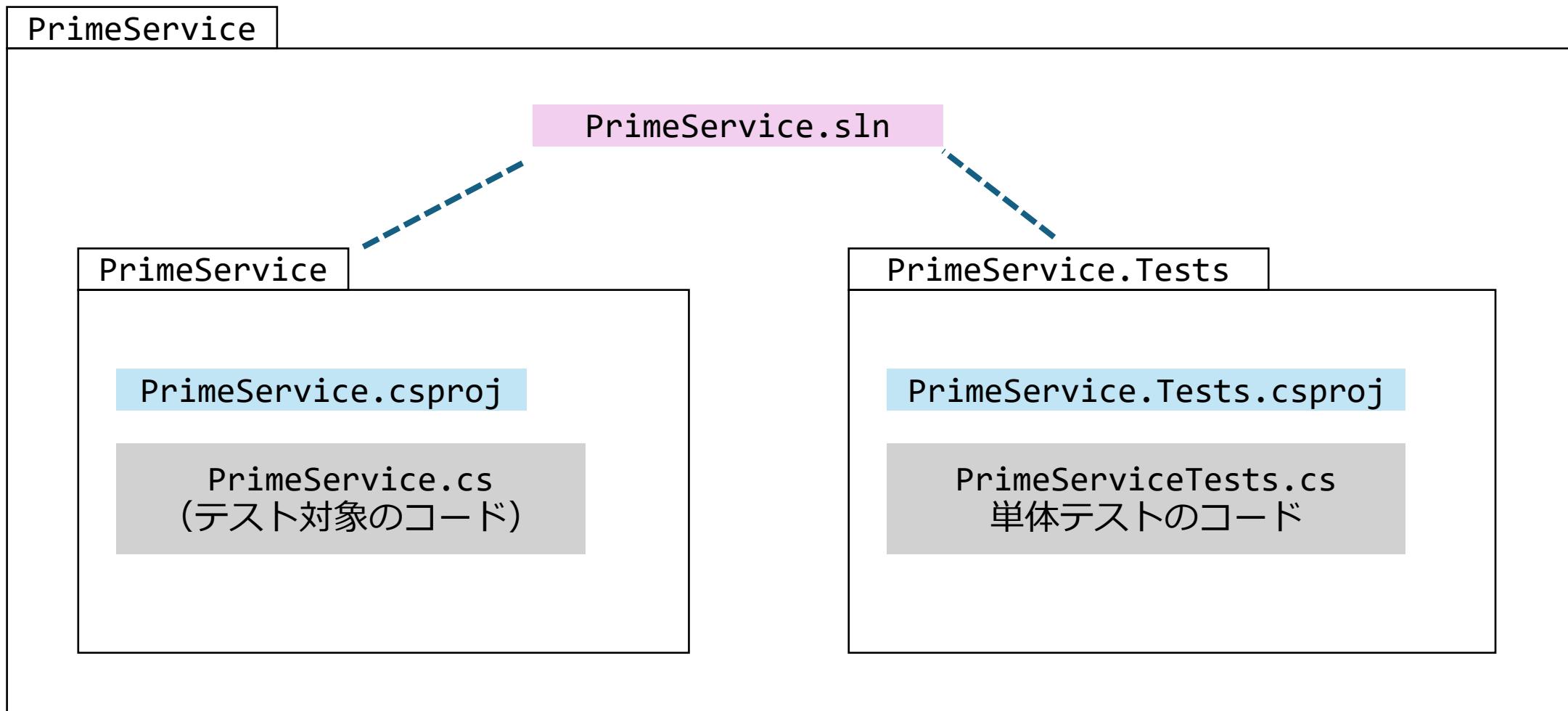
```
mkdir PrimeService  
cd PrimeService  
dotnet new sln -n PrimeService
```

## ■クラスライブラリプロジェクトの作成、ソリューションへの追加



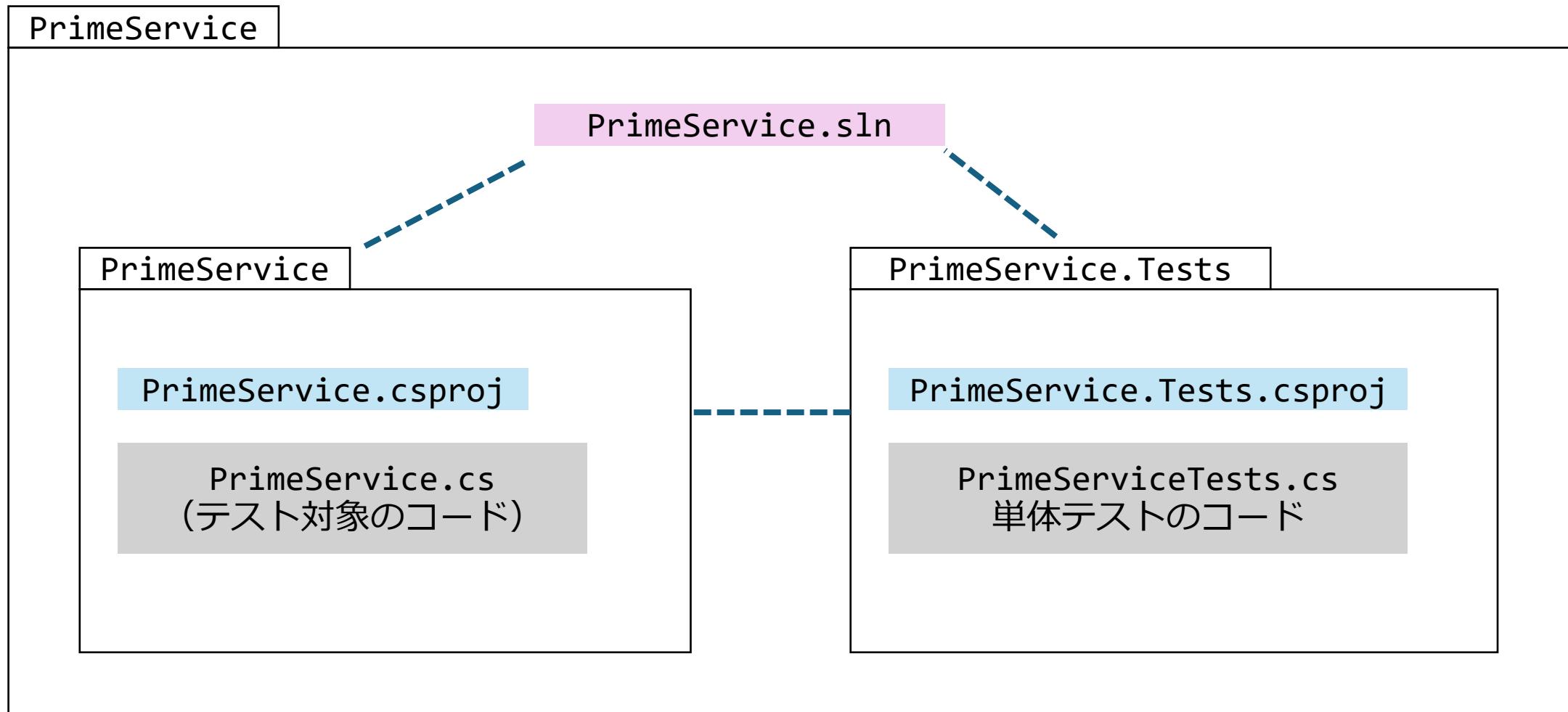
```
dotnet new classlib -n PrimeService  
dotnet sln add PrimeService
```

## ■xUnitテストプロジェクトの作成、ソリューションへの追加



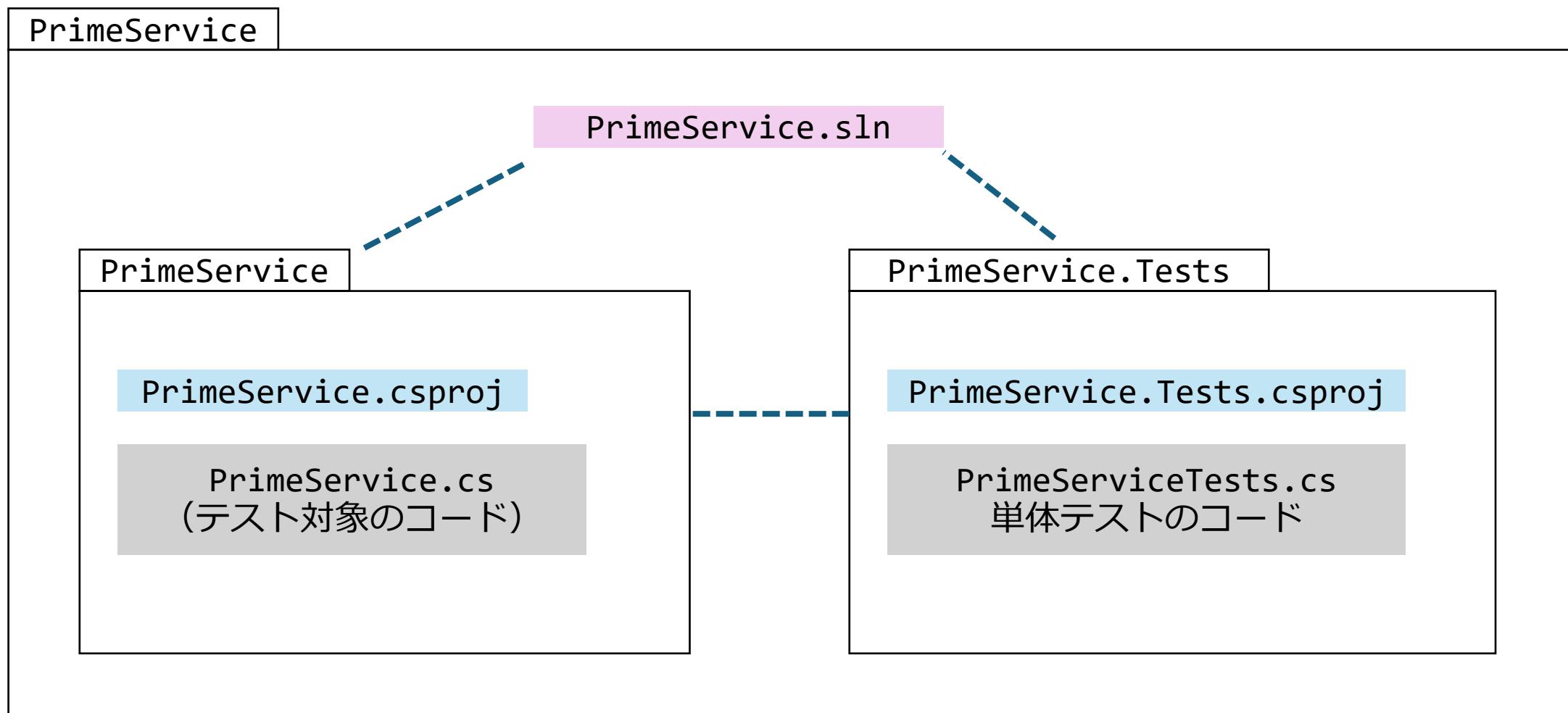
```
dotnet new xunit -n PrimeService.Tests  
dotnet sln add PrimeService.Tests
```

## ■テストプロジェクトに、テスト対象のプロジェクトへの「プロジェクト参照」を追加



```
dotnet add PrimeService.Tests reference PrimeService
```

■以上で、ソリューション/プロジェクトの構成が完成。  
dotnet build でソリューション全体のビルド、dotnet test でテストの実行が可能



dotnet build  
dotnet test

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・**テスト対象のコードの実装**
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

## ■テスト対象のコードの実装例

```
public class PrimeService
{
    2 references
    public bool IsPrime(int candidate)
    {
        if (candidate < 2)
        {
            return false;
        }

        for (var divisor = 2; divisor <= Math.Sqrt(candidate); divisor++)
        {
            if (candidate % divisor == 0)
            {
                return false;
            }
        }
        return true;
    }
}
```

与えられた数 (candidate) が素数かどうかを判定する IsPrime というメソッドを定義。

このメソッドが正しく動くのかどうかを、単体テストを使って検証する

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・テスト対象のコードの実装
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

## ■単体テストの実装例（**xUnit** を使用）

```
0 references
public class UnitTest1
{
    [Fact]
    0 references
    public void IsPrime_InputIs1_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(1);

        Assert.False(result, "1 should not be prime");
    }
}
```

「1」が素数かどうかを判定  
(結果は false になるはず)

実際に、結果が false となつた  
ことを確認

```
[Fact]
0 references
public void IsPrime_InputIs13_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(13);

    Assert.True(result, "13 should be prime");
}
```

「13」が素数かどうかを判定  
(結果は true になるはず)

実際に、結果が true となつた  
ことを確認

■参考: テストは「Arrange」「Act」「Assert」で書く（AAAパターン）

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

Arrange: オブジェクトを初期化するなどの準備

## ■参考: テストは「Arrange」「Act」「Assert」で書く（AAAパターン）

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);
    Assert.False(result, "1 should not be prime");
}
```

Act: テスト対象のメソッドを呼び出す

## ■参考: テストは「Arrange」「Act」「Assert」で書く（AAAパターン）

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

Assert: メソッドが仕様通りの  
戻り値を返したかを検証

## ■参考: テストメソッド名のベストプラクティス

xUnit のベストプラクティス（命名規約）に沿った適切な名前を付ける。

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

## ■参考: xUnitにおけるテストメソッドの名前の規約

### テストの名前付け

テストの名前は、次の 3 つの部分で構成される必要があります。

- テスト対象のメソッドの名前。
- それがテストされるシナリオ。
- シナリオが呼び出されたときに想定される動作。

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);
    Assert.False(result, "1 should not be prime");
}
```

## ■参考: xUnitにおけるテストメソッドの名前の規約

### テストの名前付け

テストの名前は、次の 3 つの部分で構成される必要があります。

- テスト対象のメソッドの名前。
- **それがテストされるシナリオ。**
- シナリオが呼び出されたときに想定される動作。

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

## ■参考: xUnitにおけるテストメソッドの名前の規約

### テストの名前付け

テストの名前は、次の 3 つの部分で構成される必要があります。

- テスト対象のメソッドの名前。
- それがテストされるシナリオ。
- シナリオが呼び出されたときに想定される動作。

```
public void IsPrime_InputIs1_ReturnFalse()
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(1);

    Assert.False(result, "1 should not be prime");
}
```

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・テスト対象のコードの実装
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・単体テストの実行例
- ・まとめ

# GitHub Copilotを使用した単体テストの生成

- (1) **スラッシュコマンド**を使用する
  - インラインチャット(Ctrl + I)で /tests
  - チャットビュー(Ctrl + Alt + I)で @workspace /tests
- (2) **スマートアクション**を使用する
  - 右クリックメニューから Copilot > Generate Testsを選択
- (3) **チャットビュー**で @github にテストコード生成を指示する
  - チャットビュー(Ctrl + Alt + I)で @github generate tests のように指示
- (4) **Copilot Edits**を使用する ※おすすめ
  - Copilot Edits(Ctrl + Shift + I)で generate tests のように指示

## ■ インラインチャットで スラッシュコマンド /tests を利用する

```
4  {
5      1 個の参照
6      public static class PrimeChecker
7      {
8          1 個の参照
9          public static bool IsPrime(long number)
10         /tests 選択したコードの単体テストを生成する
11         if (number <= 1) return false;
12         if (number == 2) return true;
13         if (number % 2 == 0) return false;
14         for (long i = 3; i <= Math.Sqrt(number); i += 2)
15         {
16             if (number % i == 0) return false;
17         }
18     }
19 }
```

Ctrl + I でインライン  
チャットを表示し、  
/tests と入力

■すると新しいタブ（新規ファイル）に、テストコードが生成される。  
「同意する」をクリックして候補を確定し、  
ファイルをテストプロジェクト以下に適切な名前で保存する。

The screenshot shows a Visual Studio interface with two tabs open. The left tab, titled 'Class1.cs', contains the implementation of the `PrimeChecker` class. The right tab, titled 'sample17', shows the generated test code for the same class.

```
Class1.cs (Implementation):
C# Class1.cs
PrimeService > C# Class1.cs > PrimeChecker > IsPrime
1  using System;
2
3  namespace PrimeService
4  {
5      1 個の参照
6      public static class PrimeChecker
7      {
8          1 個の参照
9          public static bool IsPrime(long number)
10         {
11             if (number <= 1) return false;
12             if (number == 2) return true;
13             if (number % 2 == 0) return false;
14             for (long i = 3; i <= Math.Sqrt(number); i += 2)
15             {
16                 if (number % i == 0) return false;
17             }
18             return true;
19         }
20     }
21 }
22 }
```

```
sample17 (Test Generation):
C# using NUnit.Framework; /c:/Users/hiryamada/github/sample17/PrimeService
1  using NUnit.Framework;
2
3  Copilot に質問します
4  GPT-4o
5  同意する 閉じる
6
7  namespace PrimeService.Tests
8  {
9      [TestFixture]
10     public class PrimeCheckerTests
11     {
12         [Test]
13         public void IsPrime_InputIs1_ReturnsFalse()
14         {
15             bool result = PrimeChecker.IsPrime(1);
16             Assert.IsFalse(result);
17         }
18
19         [Test]
20         public void IsPrime_InputIs2_ReturnsTrue()
21         {
22             bool result = PrimeChecker.IsPrime(2);
23             Assert.IsTrue(result);
24         }
25     }
26 }
```

A red box highlights the '同意する' (Agree) button in the Copilot dialog box, which is used to confirm the completion of the test code.

# GitHub Copilotを使用した単体テストの生成

- (1) **スラッシュコマンド**を使用する
  - インラインチャット(Ctrl + I)で /tests
  - チャットビュー(Ctrl + Alt + I)で @workspace /tests
- (2) **スマートアクション**を使用する
  - 右クリックメニューから Copilot > Generate Testsを選択
- (3) **チャットビュー**で @github にテストコード生成を指示する
  - チャットビュー(Ctrl + Alt + I)で @github generate tests のように指示
- (4) **Copilot Edits**を使用する ※おすすめ
  - Copilot Edits(Ctrl + Shift + I)で generate tests のように指示

## ■チャットビューでスラッシュコマンド /tests を利用する



■するとチャットビューの中にに、テストコードが生成される。  
テストコード右上の「**編集の適用**」ボタンをクリックすると、  
このテストコードが新しいタブ（新しいファイル）へと出力される。

The screenshot shows the Visual Studio Code interface. On the left, the 'Class1.cs' file is open, containing C# code for a 'PrimeChecker' class. On the right, the 'Copilot' tab is active, displaying a plan for generating test code and the generated 'Class1Test.cs' file.

**Class1.cs:**

```
1 using System;
2
3 namespace PrimeService
4 {
5     public static class PrimeChecker
6     {
7         public static bool IsPrime(long number)
8         {
9             if (number <= 1) return false;
10            if (number == 2) return true;
11            if (number % 2 == 0) return false;
12            for (long i = 3; i <= Math.Sqrt(number); i += 2)
13            {
14                if (number % i == 0) return false;
15            }
16            return true;
17        }
18    }
19
20
21
22
23
24
25 }
```

**Class1Test.cs:**

```
1+using System;
2+using Xunit;
3+using PrimeService;
4+
5+namespace PrimeService.Tests
```

**Copilot Chat:**

Plan:

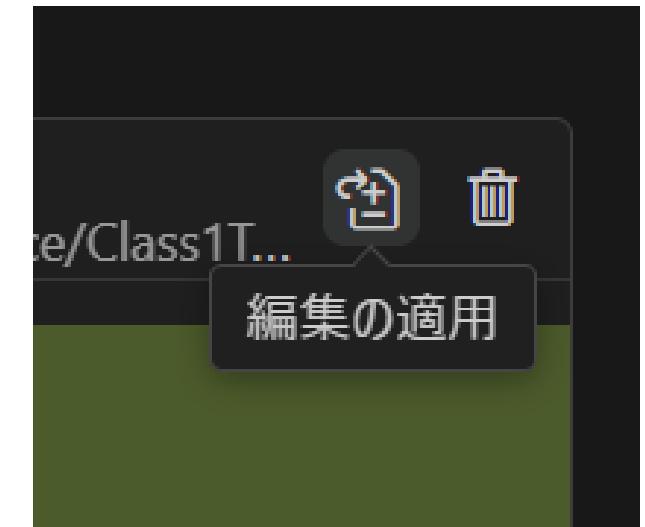
1. Create a new file `PrimeService/Class1Test.cs`.
2. Use a testing framework like xUnit.
3. Write test methods to cover various cases for the `IsPrime` method:
  - Test for numbers less than or equal to 1.
  - Test for the number 2.
  - Test for even numbers greater than 2.
  - Test for prime numbers.
  - Test for non-prime odd numbers.

テス~~ト~~ケースを追加できますか？

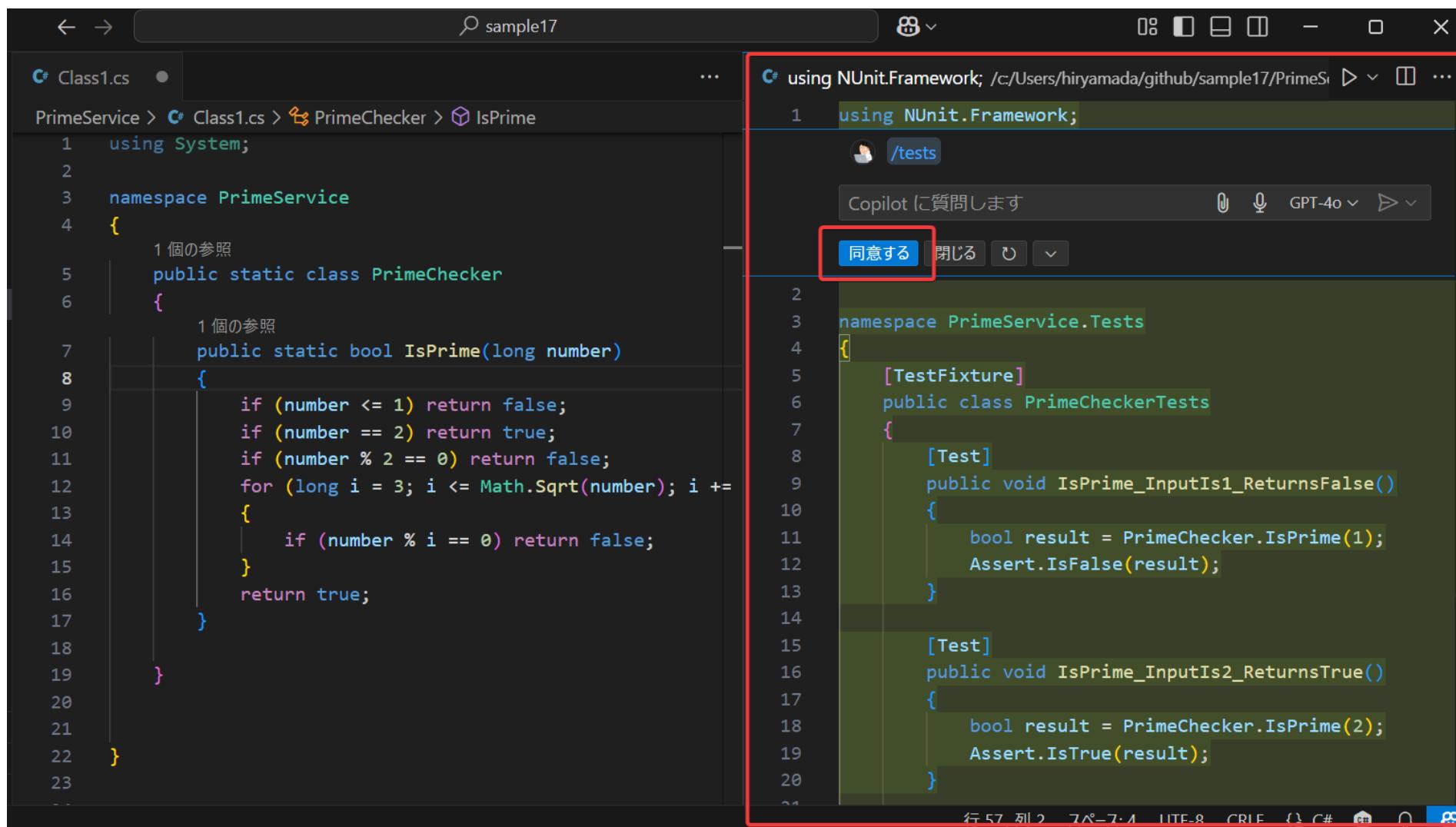
Class1.cs Current file

Copilotに質問します

GPT-4o



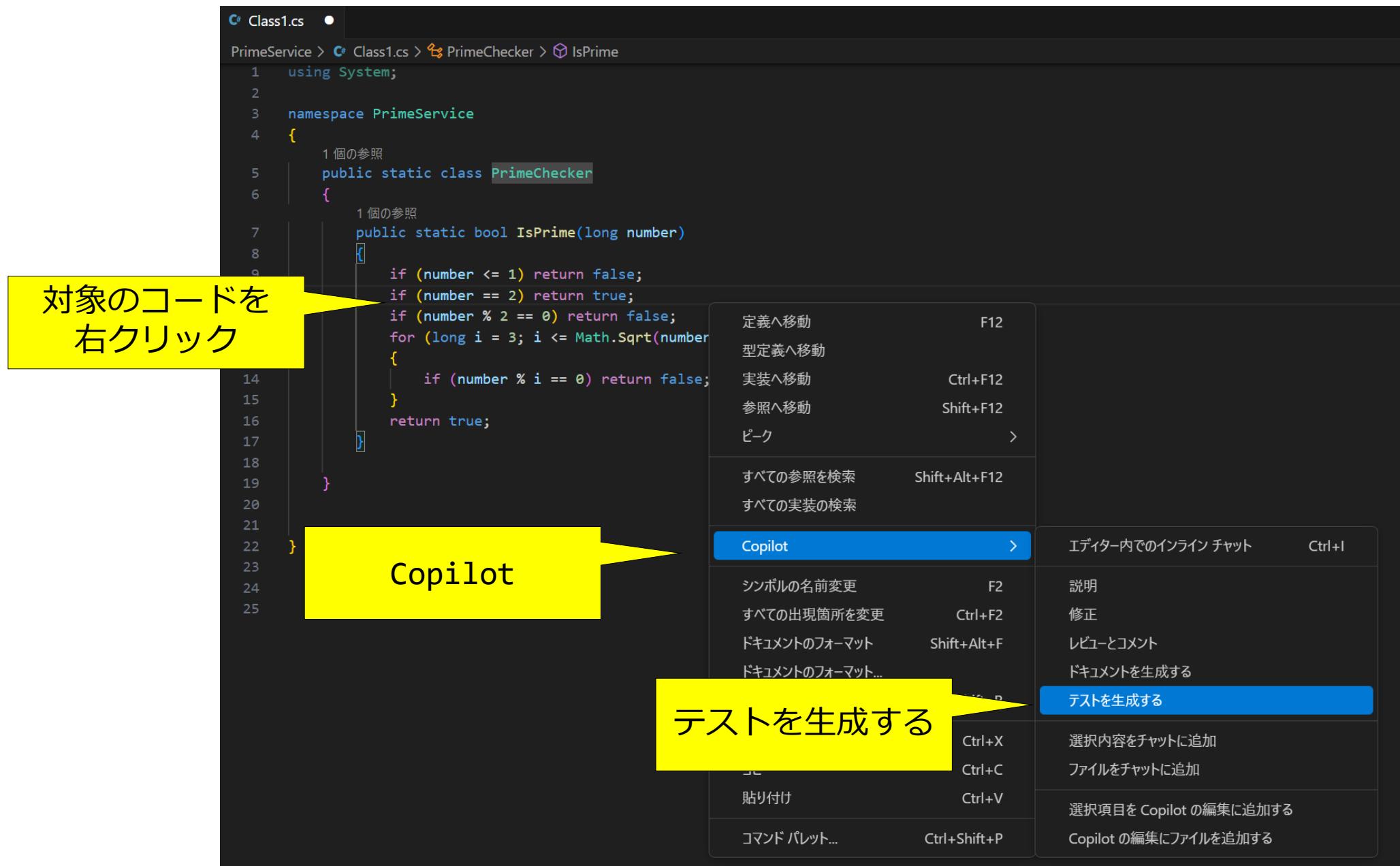
- 「同意する」をクリックして候補を確定し、ファイルをテストプロジェクト以下に適切な名前で保存する。



# GitHub Copilotを使用した単体テストの生成

- (1) **スラッシュコマンド**を使用する
  - インラインチャット(Ctrl + I)で /tests
  - チャットビュー(Ctrl + Alt + I)で @workspace /tests
- (2) **スマートアクション**を使用する
  - 右クリックメニューから Copilot > Generate Testsを選択
- (3) **チャットビュー**で @github にテストコード生成を指示する
  - チャットビュー(Ctrl + Alt + I)で @github generate tests のように指示
- (4) **Copilot Edits**を使用する ※おすすめ
  - Copilot Edits(Ctrl + Shift + I)で generate tests のように指示

# ■スマートアクション（GitHub > テストを生成する）を利用する



■すると新しいタブ（新規ファイル）に、テストコードが生成される。  
「同意する」をクリックして候補を確定し、  
ファイルをテストプロジェクト以下に適切な名前で保存する。

The screenshot shows a Visual Studio interface with two tabs open. The left tab, titled 'Class1.cs', contains the implementation of the `PrimeChecker` class. The right tab, titled 'sample17', shows the generated test code for the `PrimeChecker` class.

The generated test code is as follows:

```
1 using NUnit.Framework; /c:/Users/hiryamada/github/sample17/PrimeService.Tests
2
3 namespace PrimeService.Tests
4 {
5     [TestFixture]
6     public class PrimeCheckerTests
7     {
8         [Test]
9         public void IsPrime_InputIs1_ReturnsFalse()
10        {
11            bool result = PrimeChecker.IsPrime(1);
12            Assert.IsFalse(result);
13        }
14
15         [Test]
16         public void IsPrime_InputIs2_ReturnsTrue()
17        {
18            bool result = PrimeChecker.IsPrime(2);
19            Assert.IsTrue(result);
20        }
21    }
22}
```

A red box highlights the '同意する' (Agree) button in the Copilot UI overlay, which is used to confirm the completion of the test code.

# スマートアクションの動作

- ・スマートアクションで「Copilot>テストの生成」を選択すると、  
**インラインビュー (Ctrl + I) で「/tests」と入力してEnterを押し  
た場合と同じ結果**になる。

# GitHub Copilotを使用した単体テストの生成

- (1) **スラッシュコマンド**を使用する
  - インラインチャット(Ctrl + I)で /tests
  - チャットビュー(Ctrl + Alt + I)で @workspace /tests
- (2) **スマートアクション**を使用する
  - 右クリックメニューから Copilot > Generate Testsを選択
- (3) **チャットビュー**で @github にテストコード生成を指示する
  - チャットビュー(Ctrl + Alt + I)で @github generate tests のように指示
- (4) **Copilot Edits**を使用する ※おすすめ
  - Copilot Edits(Ctrl + Shift + I)で generate tests のように指示

## ■チャットビューで、@github generate tests を利用する

The screenshot shows the Visual Studio Code interface with the GitHub Copilot Chat view open. The code editor on the left contains C# code for a PrimeService class. A yellow callout box points to the 'Chat' button in the top right toolbar, which is highlighted with a red border. Another yellow callout box points to the '@github generate tests' command in the chat input field, also highlighted with a red border.

Class1.cs

PrimeService > Class1.cs > PrimeChecker > IsPrime

```
1  using System;
2
3  namespace PrimeService
4  {
5      1 個の参照
6      public static class PrimeChecker
7      {
8          1 個の参照
9          public static bool IsPrime(long number)
10         {
11             if (number <= 1) return false;
12             if (number == 2) return true;
13             if (number % 2 == 0) return false;
14             @github generate tests
15             i += 2)
16             return true;
17     }
```

チャットビューを開く  
(Ctrl + Alt + I)

チャット

COPilot の編集

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

As an internal user, additional telemetry is collected. If you work on a project that contains customer content, you must disable telemetry.

コンテキストをアタッチする @ 、または # と入力には、

/fix コード内の問題

/tests コードの単体テストを追加する

/explain 選択したコードのしくみ

Class1.cs Current file

@ @ @

GPT-4o ▶

行 16、列 25 スペース: 4 UTF-8 with BOM CRLF {} C# 🔍 🔔 📁

■するとチャットビュー内部に、テストコードが生成される。  
「エディターに適用」ボタンをクリックして、指定したエディターへとこの候補を適用したり、「Insert At Cursor（カーソル位置に挿入）」ボタンを使用して、エディター内の任意の位置にコードを挿入したりできる。

The screenshot shows a Visual Studio Code window with the following details:

- Title Bar:** sample17
- Left Panel:** Shows the file structure: PrimeService > Class1.cs > PrimeChecker.
- Code Editor:** Displays the following C# code:

```
1  using System;
2
3  namespace PrimeService
4  {
5      1 個の参照
6      public static class PrimeChecker
7      {
8          1 個の参照
9          public static bool IsPrime(long number)
10         {
11             if (number <= 1) return false;
12             if (number == 2) return true;
13             if (number % 2 == 0) return false;
14             for (long i = 3; i <= Math.Sqrt(number); i += 2)
15             {
16                 if (number % i == 0) return false;
17             }
18         }
19     }
20
21
22 }
```
- Right Panel (Chat View):**
  - User: hiryamada\_microsoft
  - AI Response: @github generate xunit tests for IsPrime
  - Code Snippet:

```
using System;
using Xunit;
using PrimeService;

namespace PrimeService.Tests
{
    public class PrimeCheckerTests
    {
        [Theory]
        [InlineData(1, false)]
        [InlineData(2, true)]
        [InlineData(3, true)]
    }
}
```
  - Buttons: "エディターで適用します" (Apply to Editor), "@github Web 検索、コード検索、エンタープライズのナレッジベース", "GPT-4o", and a refresh icon.
- Bottom Status Bar:** 行 18、列 9 スペース: 4 UTF-8 with BOM CRLF {} C# ⚙️ 🔍

## ■ @github と @workspace の違い: @github のほうがより多くのボタンが利用できる

### @github generate tests

と入力した場合に使用できるボタン



### @workspace /tests

と入力した場合に使用できるボタン



# GitHub Copilotを使用した単体テストの生成

- (1) **スラッシュコマンド**を使用する
  - インラインチャット(Ctrl + I)で /tests
  - チャットビュー(Ctrl + Alt + I)で @workspace /tests
- (2) **スマートアクション**を使用する
  - 右クリックメニューから Copilot > Generate Testsを選択
- (3) **チャットビュー**で @github にテストコード生成を指示する
  - チャットビュー(Ctrl + Alt + I)で @github generate tests のように指示
- (4) **Copilot Edits**を使用する ※おすすめ
  - Copilot Edits(Ctrl + Shift + I)で generate tests のように指示

## ■ Copilot Edits で、 generate tests を利用する

The screenshot shows the Visual Studio Code interface with the Copilot Edits extension active. The title bar includes tabs for 'Class1.cs' and 'PrimeService > Class1.cs > PrimeChecker > IsPrime'. The main editor window displays C# code for a prime number checker. A yellow callout box points to the status bar message 'Copilot Editsを開く (Ctrl + Shift + I)'.

**Copilot Editsを開く (Ctrl + Shift + I)**

**generate tests と入力**

※Copilot Editsでは参加者 (@githubや@workspaceなど)、スラッシュコマンド (/testsなど) は利用できない。

The status bar also shows 'COPilot の編集' (Copilot Edit) with a red border. A red arrow points from the status bar to the 'Copilot' interface. The 'Copilot' interface is shown in a separate window, which includes an 'Add Files...' button, a file list for 'Class1.cs Current file', and a text input field containing 'generate tests' with a red border. The status bar at the bottom of the interface shows '行 8、列 10 スペース: 4 UTF-8 with BOM CRLF {} C# ⚙️ 🔍'.

**Copilot で編集する**

作業する一連のファイルを定義して、編集セッションを開始します。次に、行う変更を Copilot に依頼します。

Copilot は AI によって動作しているため、ミスが発生する可能性があります。使用する前に出力を慎重にレビューしてください。

内部ユーザーとして、追加のテレメトリが収集されます。カスタマーコンテンツを含むプロジェクトで作業する場合は、テレメトリを無効にする必要があります。

コンテキストをアタッチする ⌂ 、または # と入力には、

■すると、Copilot Editsはプロジェクトの適切な位置に、適切なファイル名で、テストコードを生成してくれる。

The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays a project named 'SAMPLE17' with files like 'Class1.cs', 'PrimeService.csproj', and 'PrimeServiceTests.cs'. A yellow callout box points to 'PrimeServiceTests.cs' with the text '適切なフォルダー以下に  
適切なファイル名で  
ファイルが生成される！' (A file is generated in the appropriate folder with the appropriate file name!). The 'PrimeCheckerTests.cs' tab is selected in the editor, which contains the following C# code:

```
1  using Xunit;
2  using PrimeService;
3
4  namespace PrimeService.Tests
5  {
6      public class PrimeCheckerTests
7      {
8          [Theory]
9          [InlineData(1, false)]
10         [InlineData(2, true)]
11         [InlineData(3, true)]
12         [InlineData(4, false)]
13         [InlineData(5, true)]
14         [InlineData(16, false)]
15         [InlineData(17, true)]
16         [InlineData(18, false)]
17         [InlineData(19, true)]
18         [InlineData(20, false)]
19         0 個の参照
20         public void IsPrimeTest(long number, bool expected)
21         {
22            保持 元に戻す 1/1 ↑ ↓ ↴
23             bool result = PrimeChecker.IsPrime(number);
24             Assert.Equal(expected, result);
25         }
26     }
27 }
```

A red box highlights the 'PrimeCheckerTests.cs' tab in the editor. To the right, the GitHub Copilot interface shows a suggestion: 'Add a new method IsPrimeTest to test the IsPrime method.' Below it, a preview shows the changes: 'Class1.cs +9 -3' and 'PrimeCheckerTests.cs +26 -1'. A yellow callout box points to the right pane with the text '生成された  
テストコード' (Generated test code). At the bottom, a status bar shows '2 files changed' and the file names.

適切なフォルダー以下に  
適切なファイル名で  
ファイルが生成される！

生成された  
テストコード

2 files changed

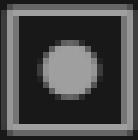
C# PrimeCheckerTests.cs PrimeService.Tests

C# Class1.cs PrimeService

保持 元に戻す

ワークスペース内のファイルを編集します

行 1、列 1 スペース: 4 UTF-8 CRLF {} C# 🔍



←このアイコンは、Copilot Editsが、このファイルに対して候補を書き込んだ、という状態を表している。  
「保持」ボタンでこれを受理したり、  
「元に戻す」ボタンでこの変更を取り消したりできる。

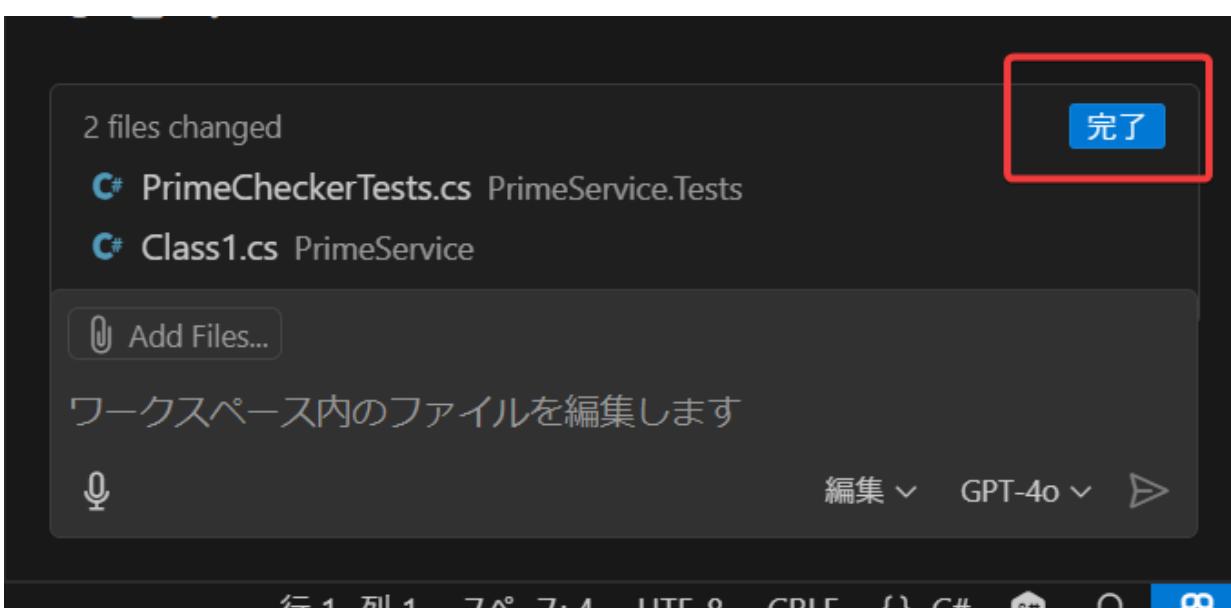
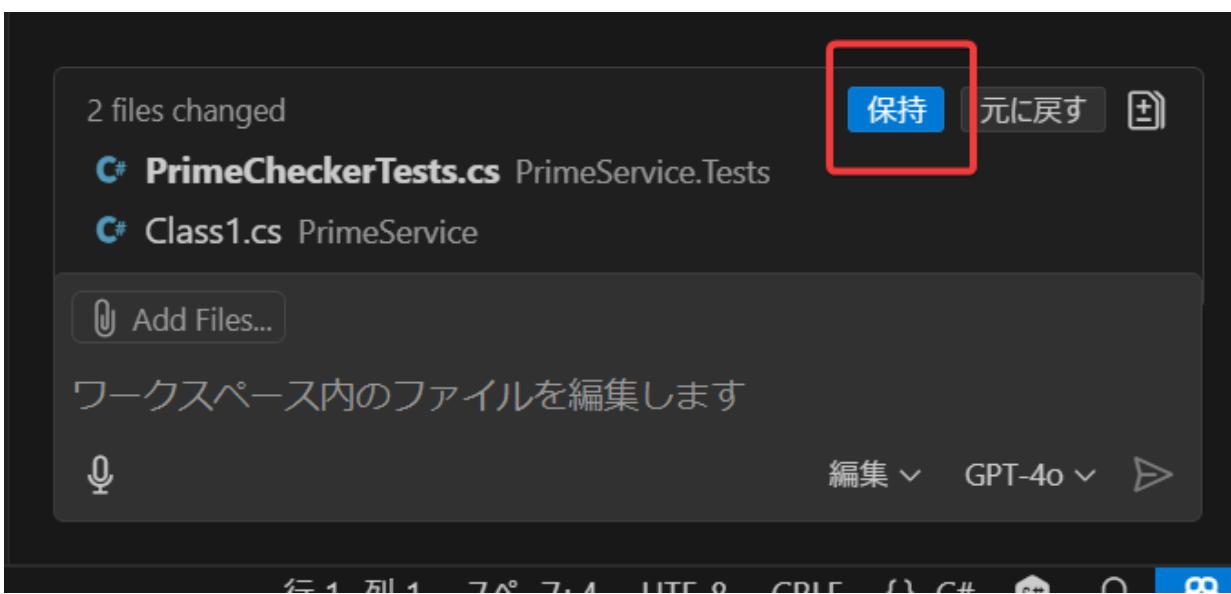
The screenshot shows the Visual Studio interface with the following details:

- Solution Explorer:** Shows a project named "SAMPLE17" with files like Class1.cs, Class2.cs, Class3.cs, Employee.cs, PrimeService.csproj, and PrimeService.Tests.csproj.
- Toolbox:** Standard Visual Studio icons for file operations, search, and other tools.
- Status Bar:** Shows "保持 元に戻す" (Keep/Undo) buttons and navigation information (1/1).
- Copilot Edits Status:** Three icons are highlighted with red boxes:
  - A small square with a circle inside, located in the gutter next to PrimeCheckerTests.cs.
  - A small square with a dot inside, located in the gutter next to PrimeService.Tests.cs.
  - A small square with a dot inside, located in the gutter next to PrimeCheckerTests.cs.
- Code Editor:** Displays C# code for PrimeService.Tests.

```
1  using Xunit;
2  using PrimeService;
3
4  namespace PrimeService.Tests
{
    [Fact]
    public void IsPrime_InputIs1_ReturnsFalse()
    {
        Assert.False(PrimeChecker.IsPrime(1));
    }

    [Fact]
    public void IsPrime_InputIs2_ReturnsTrue()
    {
        Assert.True(PrimeChecker.IsPrime(2));
    }
}
```

## ■問題がなければ「保持」「完了」とクリックしてEditsの候補を確定する。



なお、生成または変更されたファイルはこの時点では**保存されていない**ので、  
保存操作を行うこと！

Ctrl + S でこのファイルを保存。  
Ctrl + K, S で全ファイルを保存。

```
1 using Xunit;
2 using PrimeService;
3
4 namespace PrimeService.Tests
5 {
6     0 個の参照
7     public class PrimeCheckerTests
8     {
9         [Theory]
10        [InlineData(1, false)]
11        [InlineData(2, true)]
12        [InlineData(3, true)]
13        [InlineData(4, false)]
```

# モジュール4

- ・ソフトウェアテストの主な種類
- ・C#のテストフレームワーク
- ・単体テストとは？
- ・xUnitを使用するソリューション/プロジェクトの準備
- ・テスト対象のコードの実装
- ・xUnit 単体テストの実装例
- ・GitHub Copilotによる単体テストの生成
- ・**単体テストの実行例**
- ・まとめ

## ■単体テストの実行例

dotnet test で  
テストを実行

● \$ dotnet test

復元が完了しました (0.6 秒)

PrimeService 成功しました (0.3 秒) → PrimeService\bin\Debug\net9.0\PrimeService.dll

PrimeService.Tests 成功しました (0.3 秒) → PrimeService.Tests\bin\Debug\net9.0\PrimeService.Tests.dll

[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 9.0.1)

[xUnit.net 00:00:00.08] Discovering: PrimeService.Tests

[xUnit.net 00:00:00.13] Discovered: PrimeService.Tests

[xUnit.net 00:00:00.13] Starting: PrimeService.Tests

[xUnit.net 00:00:00.22] Finished: PrimeService.Tests

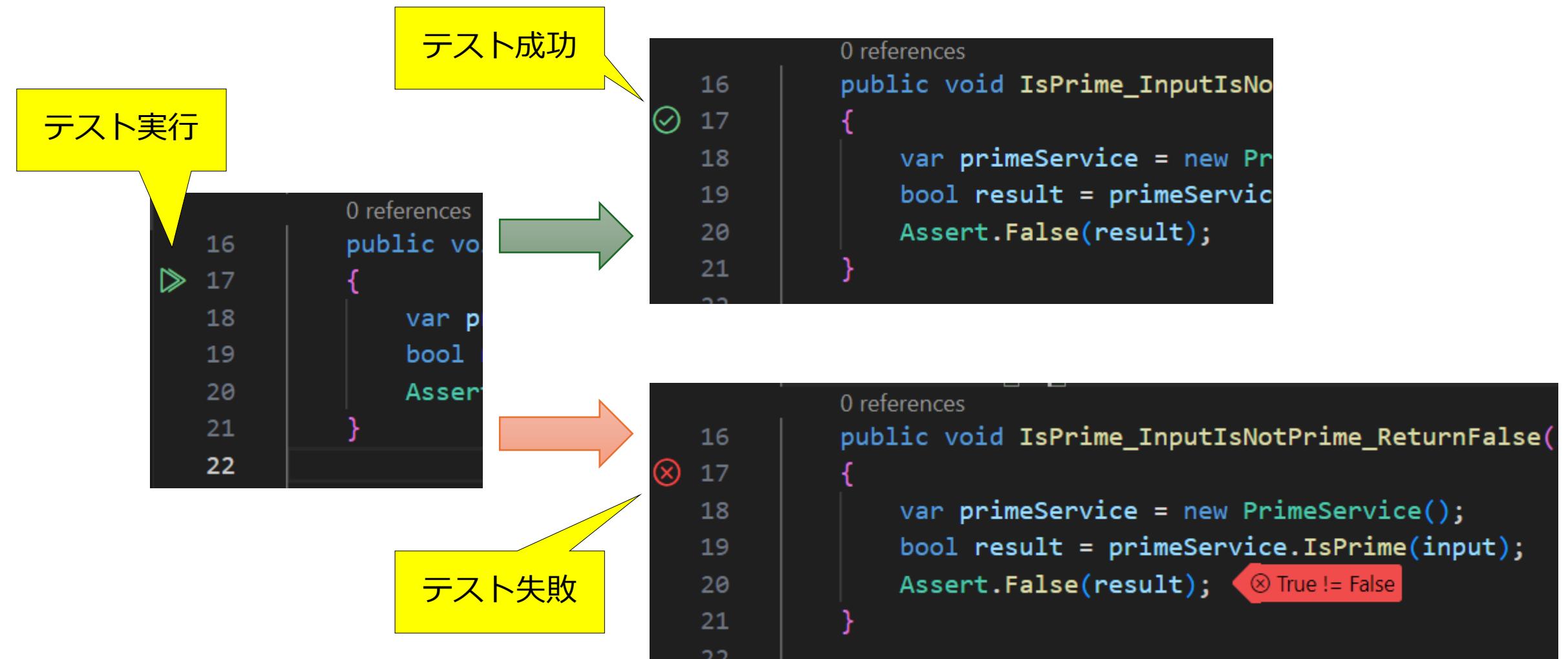
PrimeService.Tests テスト 成功しました (1.1 秒)

テスト概要: 合計: 2, 失敗数: 0, 成功数: 2, スキップ済み数: 0, 期間: 1.1 秒

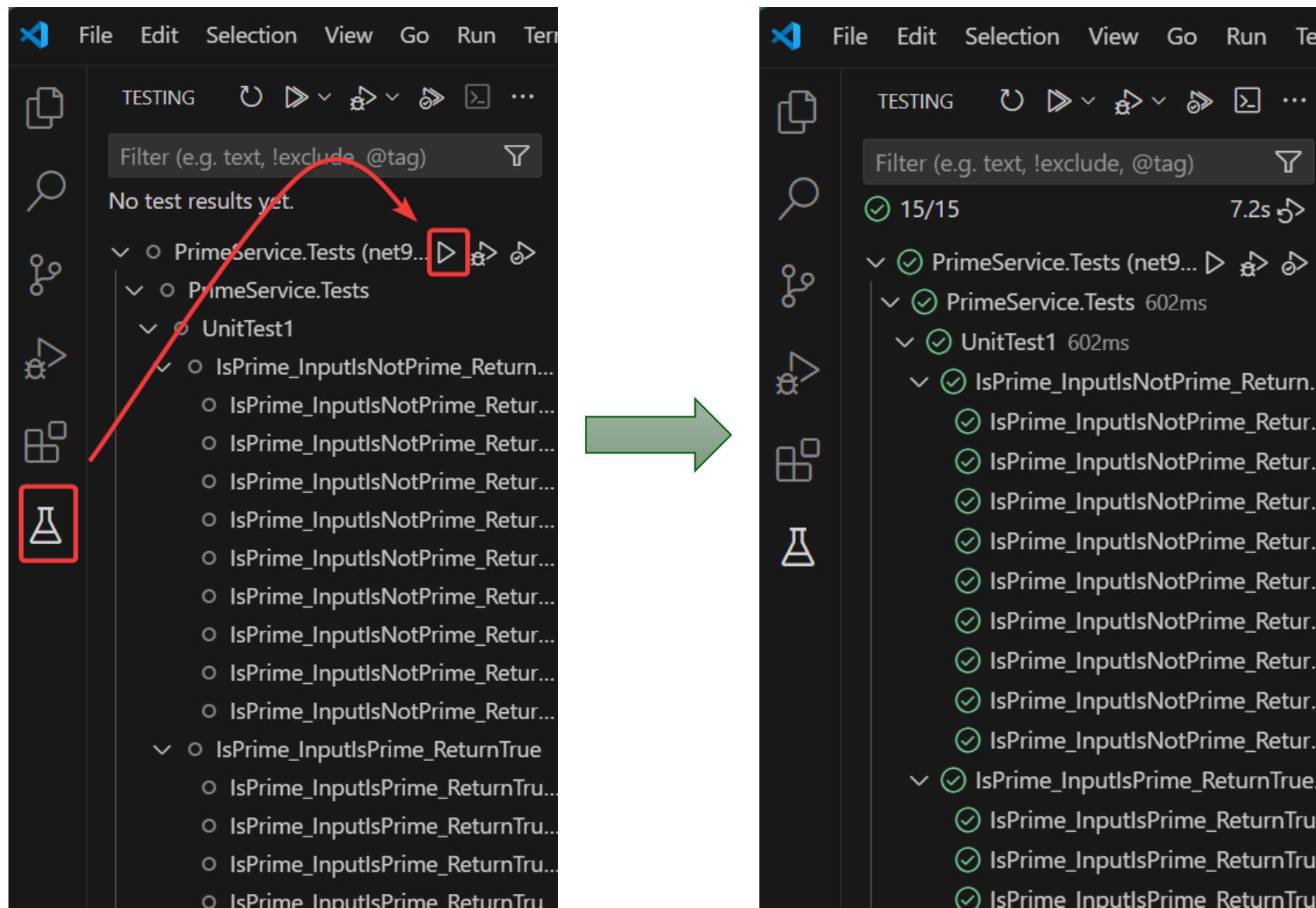
2.6 秒後に 成功しました をビルド

2件のテストが成功。IsPrimeメソッドが  
(少なくともこの2件のテストケースに対しては) 期待通りに動作しているといえる。

■参考: 行番号部分の  マークをクリックしてテストを実行することもできる



## ■参考: アクティビティバーの「テスト」からテストを実行することもできる



# モジュール4 まとめ

- GitHub Copilotは、さまざまなプログラミング言語の、さまざまなソフトウェアテスト（単体、結合、システム/E2E、受け入れ）のテストコードを生成できる。また、さまざまなテストフレームワークに対応できる。
- GitHub Copilotでは、**インラインチャットやチャットビュー**内でスラッシュコマンド(/tests)を使う、**スマートアクション**（右クリック>GitHub>Generate Tests）を選ぶ、**Copilot Edits**を使う、などの方法で、テストコードをすばやく生成できる。

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する
- ・補足: GitHub Copilotの最新機能のご紹介

# モジュール5



## GitHub Copilot を使用してコードの改善を実装する

1 時間 10 分 残り • モジュール • 1/9 ユニットが完了しました

このモジュールでは、GitHub Copilot Chat を使用して、既存のコードベースの改善策を開発する方法について説明します。演習により、コードの品質、信頼性、パフォーマンス、およびセキュリティを向上させる GitHub Copilot の提案を実装する実用的な経験が得られます。

# モジュール5

- ・インラインチャットを使用したコード改善の例
- ・コードの改善のその他の例
  - ・セキュリティ
  - ・リファクタリング
  - ・パフォーマンス
- ・まとめ

# インラインチャットを使った改善の例

- ・改善したい部分を選択
- ・[Ctrl + I]でインラインチャットを起動
- ・コードの改善をプロンプトで指示
- ・GitHub Copilotがコードを書き直す
  - ・この時点ではまだコードは保存されておらず、確定もされていない
  - ・もし問題があれば Discard（取り消し）
- ・ファイルを保存し、テストを実行
- ・テスト実行結果に問題がなければ Accept
  - ・もし問題があれば Discard（取り消し）

## ■改善前のコード: IsPrimeの単体テスト (テストケースは現在2件しかなく、十分とは言えない)

```
public class UnitTest1
{
    [Fact]
    public void IsPrime_InputIs1_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(1);

        Assert.False(result, "1 should not be prime");
    }

    [Fact]
    public void IsPrime_InputIs13_ReturnTrue()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(13);

        Assert.True(result, "13 should be prime");
    }
}
```

1が素数と判定されないことのテスト

13が素数と判定されることのテスト

## ■改善したい部分を選択（ここではテストコード全体を選択している）

```
public class UnitTest1
{
    [Fact]
    0 references
    public void IsPrime_InputIs1_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(1);

        Assert.False(result, "1 should not be prime");
    }

    [Fact]
    0 references
    public void IsPrime_InputIs13_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(13);

        Assert.True(result, "13 should be prime");
    }
}
```

## ■Ctrl + Iでオンラインチャットを起動し、改善の指示をプロンプトで与える

The screenshot shows a code editor window with a dark theme. At the top, there's a header bar with icons for file operations and a user profile. Below the header, a search bar contains the Japanese text "1や13といったテストの素数をパラメータ化して". To the right of the search bar are icons for a trash can, a document, and "GPT 4.0", along with a dropdown arrow.

The main area displays a C# unit test class named `UnitTest1`:

```
0 references
4 public class UnitTest1
5 {
6     [Fact]
7     public void IsPrime_InputIs1_ReturnFalse()
8     {
9         var primeService = new PrimeService();
10        bool result = primeService.IsPrime(1);
11
12        Assert.False(result, "1 should not be prime");
13    }
14
15    [Fact]
16    public void IsPrime_InputIs13_ReturnFalse()
17    {
18        var primeService = new PrimeService();
19        bool result = primeService.IsPrime(13);
20
21        Assert.True(result, "13 should be prime");
22    }
23 }
```

The code editor highlights several lines of code in blue, indicating they are selected or being processed by the AI. The AI prompt at the top suggests parameterizing the prime number tests (1 and 13) into parameters.

## ■ GitHub Copilotにより、改善されたコードが示される

1や13といったテストの素数をパラメータ化して

Ask Copilot

0 0 GPT 4o ▶ ▾

Accept Discard ⌂ ▾

0 references

```
public class UnitTest1
{
    [Theory]
    [InlineData(1, false)]
    [InlineData(13, true)]
```

0 references

1は素数ではない(false),  
13は素数である(true)

```
    public void IsPrime_InputIsParam_ReturnExpectedResult(int input, bool expected)
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(input);

        Assert.Equal(expected, result);
    }
}
```

## ■ 続けて、別の改善の指示をプロンプトで与えることもできる



1や13といったテストの素数をパラメータ化して

1～15の数をそれぞれテストするようにして

0 0 GPT 4o ▶

Edit Code ▾

0 references

public class UnitTest1

{

... [Theory]

... [InlineData(1, false)]

... [InlineData(13, true)]

0 references

... public void IsPrime\_InputIsParam\_ReturnExpectedResult(int input, bool expected)

{

... var primeService = new PrimeService();

... bool result = primeService.IsPrime(input);

... Assert.Equal(expected, result);

}

}

## ■ GitHub Copilotにより、改善されたコードが示される

The screenshot shows the GitHub Copilot interface with the following text:

```
Ask Copilot
Accept Discard ⚙️ GPT 4o ▶️
```

0 references

```
public class UnitTest1
{
    [Theory]
    [InlineData(1, false)]
    [InlineData(2, true)]
    [InlineData(3, true)]
    [InlineData(4, false)]
    [InlineData(5, true)]
    [InlineData(6, false)]
    [InlineData(7, true)]
    [InlineData(8, false)]
    [InlineData(9, false)]
    [InlineData(10, false)]
    [InlineData(11, true)]
    [InlineData(12, false)]
    [InlineData(13, true)]
    [InlineData(14, false)]
    [InlineData(15, false)]
}
0 references
public void IsPrime_InputIsParam_ReturnExpectedResult(int input, bool expected)
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(input);

    Assert.Equal(expected, result);
}
```

## ■ さらに別の改善を指示

- 1や13といったテストの素数をパラメータ化して
- 1~15の数をそれぞれテストするようにして

素数とそうでない数でメソッドを分けて

GPT 4o ▶

Edit Code

0 references

```
public class UnitTest1
{
    [Theory]
    [InlineData(1, false)]
    [InlineData(2, true)]
    [InlineData(3, true)]
    [InlineData(4, false)]
    [InlineData(5, true)]
    [InlineData(6, false)]
    [InlineData(7, true)]
    [InlineData(8, false)]
    [InlineData(9, false)]
    [InlineData(10, false)]
    [InlineData(11, true)]
    [InlineData(12, false)]
    [InlineData(13, true)]
    [InlineData(14, false)]
    [InlineData(15, false)]
```

0 references

```
public void IsPrime_InputIsParam_ReturnExpectedResult(int input, bool expected)
{
    var primeService = new PrimeService();
    bool result = primeService.IsPrime(input);

    Assert.Equal(expected, result);
}
```

## ■ GitHub Copilotにより、改善されたコードが示される

```
public class UnitTest1
{
    [Theory]
    [InlineData(1)]
    [InlineData(4)]
    [InlineData(6)]
    [InlineData(8)]
    [InlineData(9)]
    [InlineData(10)]
    [InlineData(12)]
    [InlineData(14)]
    [InlineData(15)]
    0 references
    public void IsPrime_InputIsNotPrime_ReturnFalse(int input)
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(input);

        Assert.False(result);
    }

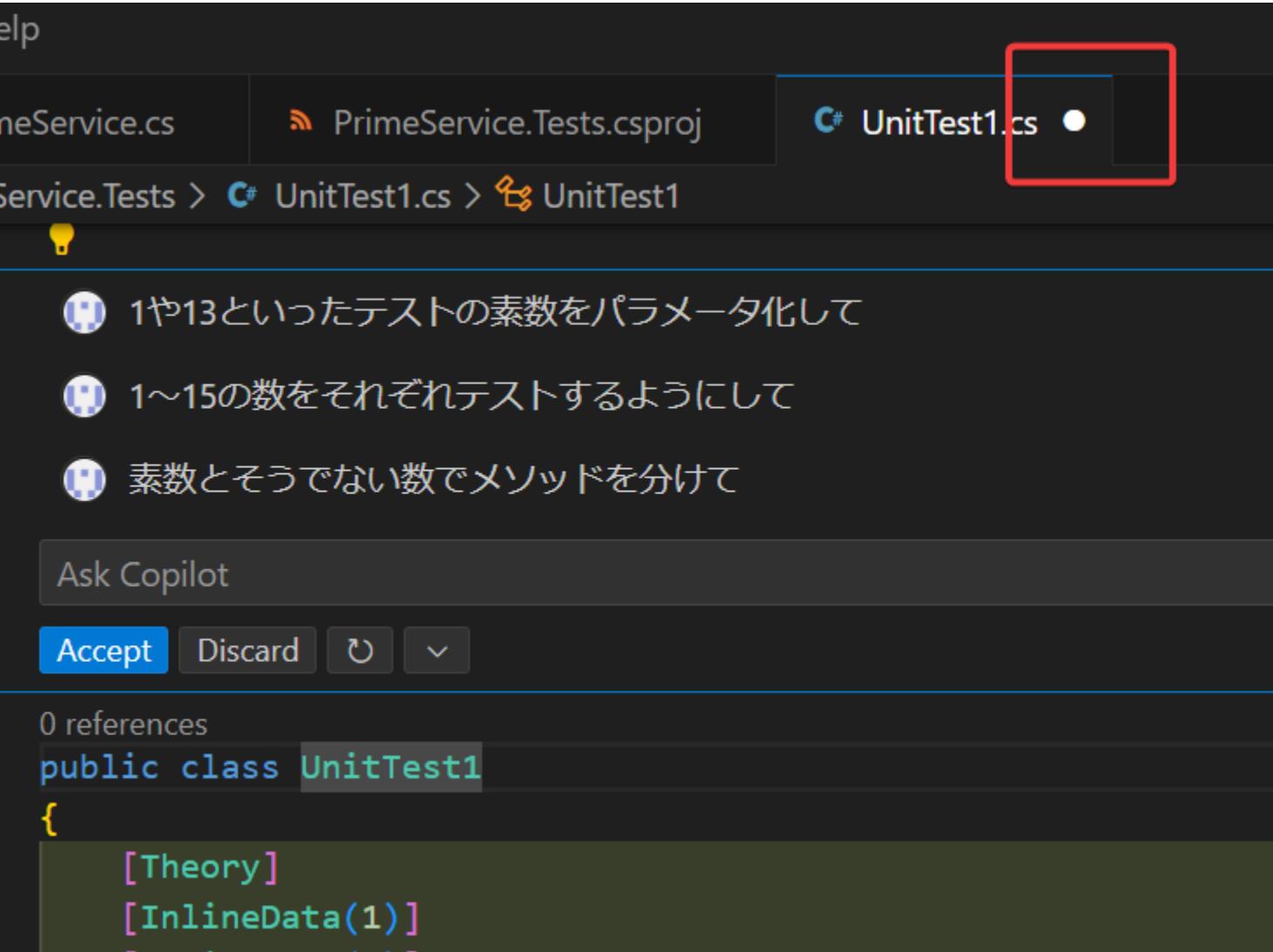
    [Theory]
    [InlineData(2)]
    [InlineData(3)]
    [InlineData(5)]
    [InlineData(7)]
    [InlineData(11)]
    [InlineData(13)]
    0 references
    public void IsPrime_InputIsPrime_ReturnTrue(int input)
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(input);

        Assert.True(result);
    }
}
```

1, 4, 6, 8, 9, 10, 12,  
14, 15が素数と判  
定されないこと  
のテスト

2, 3, 5, 7, 11, 13が  
素数と判定され  
ることのテスト

■この時点ではまだこの変更がファイルに保存されていないことに注意。



The screenshot shows the Visual Studio interface with the following details:

- File Tab:** The "UnitTest1.cs" tab is highlighted with a red box.
- Project Explorer:** Shows "PrimeService.cs", "PrimeService.Tests.csproj", and "UnitTest1.cs".
- Task List:** Three items are listed:
  - 1や13といったテストの素数をパラメータ化して
  - 1~15の数をそれぞれテストするようにして
  - 素数とそうでない数でメソッドを分けて
- Ask Copilot:** A button to interact with Microsoft Copilot.
- Accept/Discard Buttons:** Buttons for accepting or discarding changes.
- Code Editor:** Displays the following C# code:

```
public class UnitTest1
{
    [Theory]
    [InlineData(1)]
```

## ■ ここでいったん Ctrl + S でファイルを保存して、テストを実行してみる

● \$ dotnet test

復元が完了しました (1.7 秒)

PrimeService 成功しました (0.5 秒) → PrimeService\bin\Debug\net9.0\PrimeService.dll

PrimeService.Tests 成功しました (3.5 秒) → PrimeService.Tests\bin\Debug\net9.0\PrimeService.Tests.dll

[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 9.0.1)

[xUnit.net 00:00:01.68] Discovering: PrimeService.Tests

[xUnit.net 00:00:01.74] Discovered: PrimeService.Tests

[xUnit.net 00:00:01.74] Starting: PrimeService.Tests

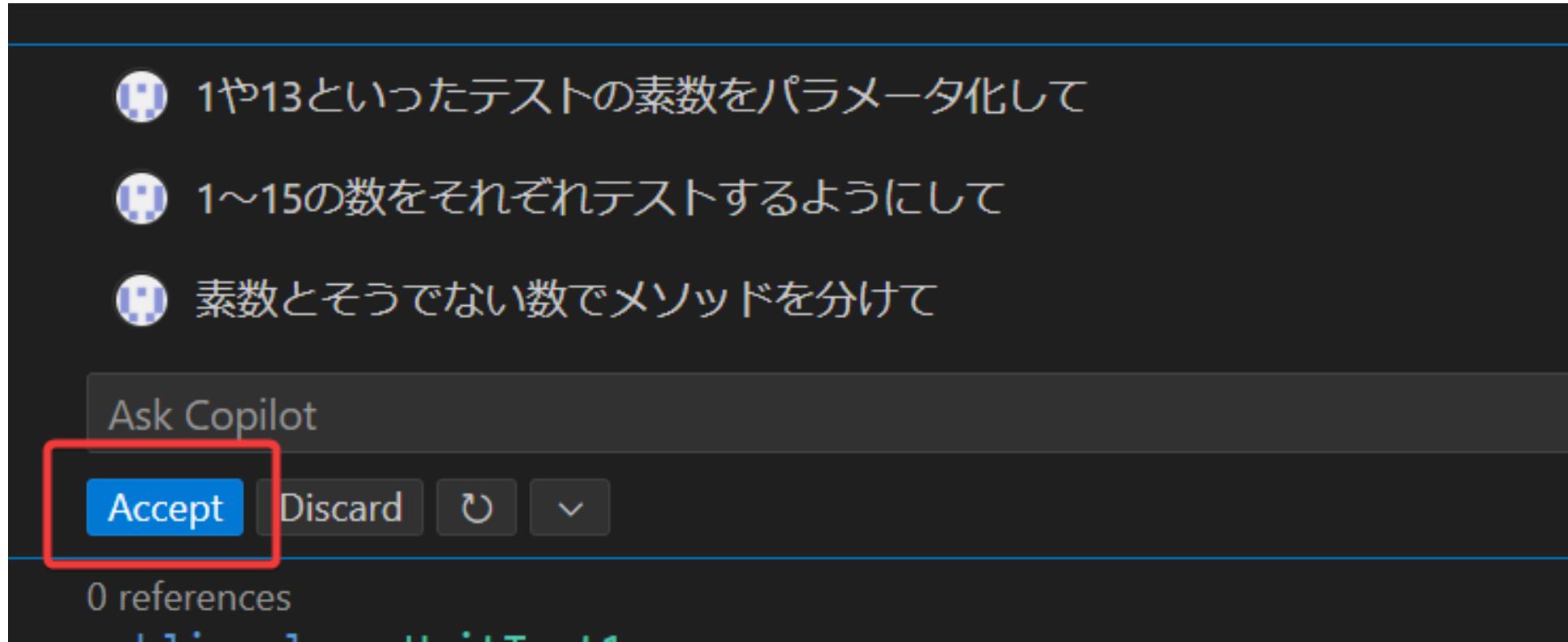
[xUnit.net 00:00:02.40] Finished: PrimeService.Tests

PrimeService.Tests テスト 成功しました (4.0 秒)

テスト概要: 合計: 15, 失敗数: 0, 成功数: 15, スキップ済み数: 0, 期間: 3.9 秒

10.3 秒後に 成功しました をビルド

■問題がなかったため、Accept をクリックして、これらの改善提案を受け付ける。



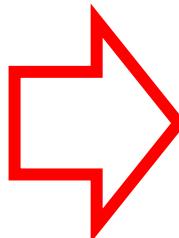
## ■ ここまで操作により、コードがこのように改善された。

```
public class UnitTest1
{
    [Fact]
    0 references
    public void IsPrime_InputIs1_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(1);

        Assert.False(result, "1 should not be prime");
    }

    [Fact]
    0 references
    public void IsPrime_InputIs13_ReturnFalse()
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(13);

        Assert.True(result, "13 should be prime");
    }
}
```



```
public class UnitTest1
{
    [Theory]
    [InlineData(1)]
    [InlineData(4)]
    [InlineData(6)]
    [InlineData(8)]
    [InlineData(9)]
    [InlineData(10)]
    [InlineData(12)]
    [InlineData(14)]
    [InlineData(15)]
    0 references
    public void IsPrime_InputIsNotPrime_ReturnFalse(int input)
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(input);

        Assert.False(result);
    }

    [Theory]
    [InlineData(2)]
    [InlineData(3)]
    [InlineData(5)]
    [InlineData(7)]
    [InlineData(11)]
    [InlineData(13)]
    0 references
    public void IsPrime_InputIsPrime_ReturnTrue(int input)
    {
        var primeService = new PrimeService();
        bool result = primeService.IsPrime(input);

        Assert.True(result);
    }
}
```

[Theory]と[InlineData(...)]を使用して、より多くのテストケースを追加した。新しいテストケースの追加も簡単にできる形となった。

# モジュール5

- ・インラインチャットを使用したコード改善の例
- ・コードの改善のその他の例
  - ・セキュリティ
  - ・リファクタリング
  - ・パフォーマンス
- ・まとめ

# コードの改善のその他の例

- ・セキュリティの強化
  - ・脆弱性のあるコードの発見と修正
- ・リファクタリング（品質向上）
  - ・複雑なコードをよりシンプルな形にする
  - ・コードの重複を排除する
  - ・適切な名前に変更する
  - ・リテラル（コード中に直接書かれた文字列）の定数化
  - ・コーディング規約に従うようにする（改行・空白の使い方など）
- ・パフォーマンスの改善
  - ・適切なアルゴリズムの使用、SIMD命令の使用などによる高速化
  - ・メモリの使用量を減らす

Single Instruction, Multiple Data。  
1回の命令で複数のデータを同時に処理するCPU命令。

# コードの改善のその他の例

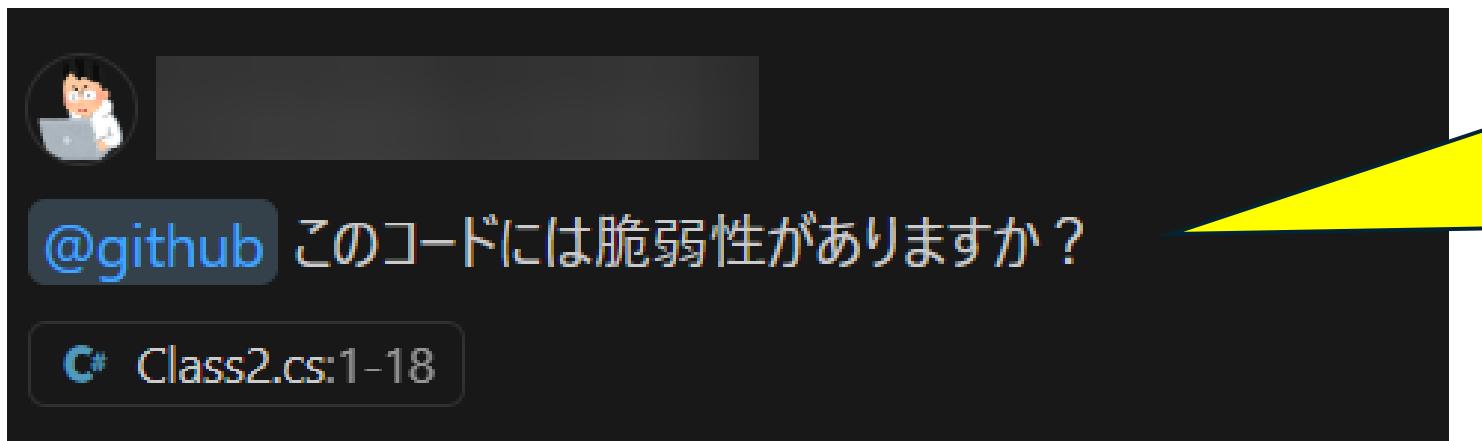
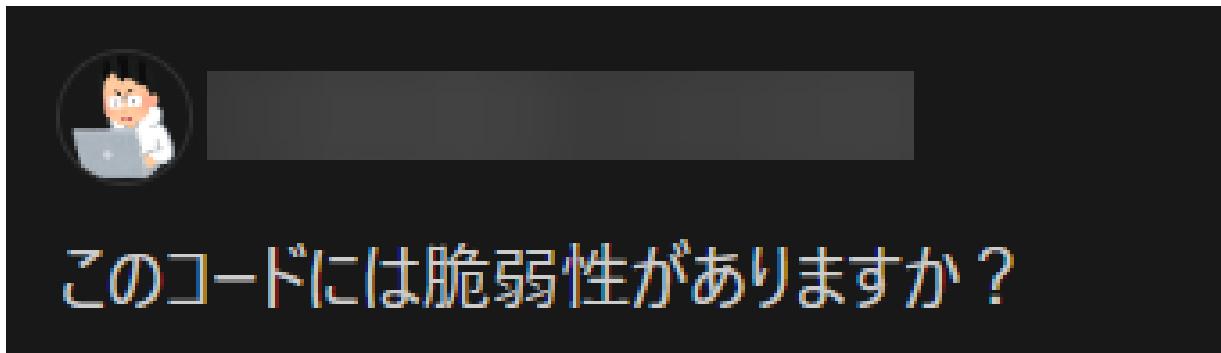
- セキュリティの強化
  - 脆弱性のあるコードの発見と修正
- リファクタリング（品質向上）
  - 複雑なコードをよりシンプルな形にする
  - コードの重複を排除する
  - 適切な名前に変更する
  - リテラル（コード中に直接書かれた文字列）の定数化
  - コーディング規約に従うようにする（改行・空白の使い方など）
- パフォーマンスの改善
  - 適切なアルゴリズムの使用、SIMD命令の使用などによる高速化
  - メモリの使用量を減らす

Single Instruction, Multiple Data。  
1回の命令で複数のデータを同時に処理するCPU命令。

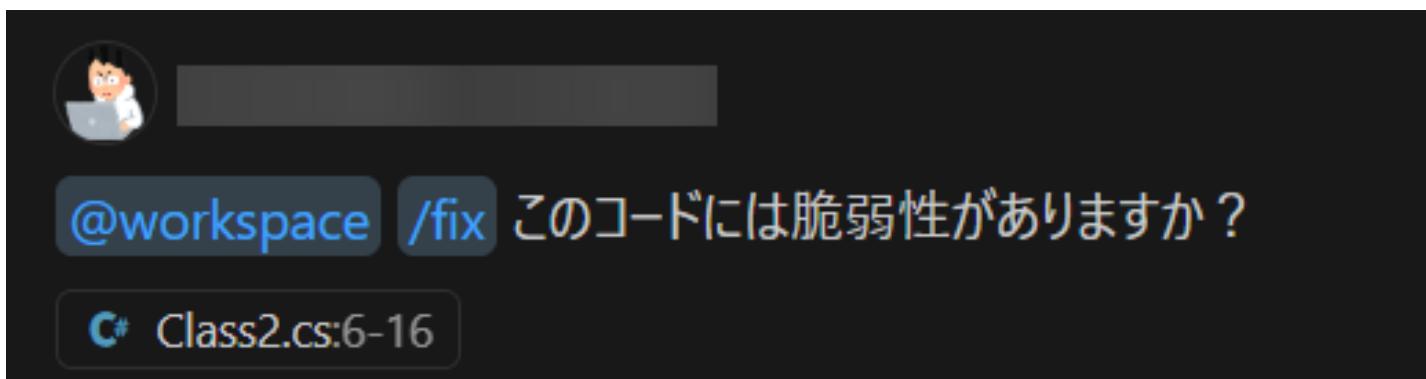
## ■脆弱性があるコードの例

```
class DatabaseUtil
{
    public static Employee FindEmployeeByName(string name)
    {
        string sql = "SELECT * FROM Employees WHERE Name = " + name;
        // TODO : Implement database access logic here
        return new Employee();
    }
}
```

## ■チャットビュー（Ctrl + Alt + I）を起動し、脆弱性があるかどうか確認してみる



@githubや@workspaceなどの「参加者」を指定したり、「/fix」といったスラッシュコマンドを組み合わせることもできるが、この例では、どれも同じような結果となる。



## ■GitHub Copilotが「SQLインジェクション脆弱性」を指摘し、訂正後のコードを提案してくれる。

The screenshot shows a GitHub Copilot interface with the following elements:

- GitHub logo** and **GitHub** text.
- A navigation bar with **>**, **1 参照**, and **使用済み**.
- A warning message: "このコードにはSQLインジェクションの脆弱性があります。特に、FindEmployeeByName メソッドで名前を直接SQLクエリに埋め込んでいる部分が問題です。攻撃者は名前にSQLコードを含めて、データベースを操作することができます。"
- A note: "脆弱性を修正するために、パラメータ化されたクエリを使用することをお勧めします。例えば、次のように修正します：
- A code editor window containing C# code for a **DatabaseUtil** class. The code includes a **FindEmployeeByName** method with a direct SQL query and a TODO comment for implementing a parameterized query.
- A button labeled **エディターで適用します** (Apply in Editor) with a context menu icon above it.

```
class DatabaseUtil
{
    public static Employee FindEmployeeByName(string name)
    {
        string sql = "SELECT * FROM Employees WHERE Name = @name";

        // TODO: Implement database access logic here using parameterized query

        // Example with ADO.NET (Assuming you are using SQL Server)
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            SqlCommand command = new SqlCommand(sql, connection);
            command.Parameters.AddWithValue("@name", name);
        }
    }
}
```

# コードの改善のその他の例

- ・セキュリティの強化
  - ・脆弱性のあるコードの発見と修正
- ・リファクタリング（品質向上）
  - ・複雑なコードをよりシンプルな形にする
  - ・コードの重複を排除する
  - ・適切な名前に変更する
  - ・リテラル（コード中に直接書かれた文字列）の定数化
  - ・コーディング規約に従うようにする（改行・空白の使い方など）
- ・パフォーマンスの改善
  - ・適切なアルゴリズムの使用、SIMD命令の使用などによる高速化
  - ・メモリの使用量を減らす

Single Instruction, Multiple Data。  
1回の命令で複数のデータを同時に処理するCPU命令。

## ■名前（大文字・小文字の使い方）が適切ではないコードの例

```
class Employee
{
    public int id { get; set; }
    public string name { get; set; }
}
```

## ■GitHub Copilotがコードの改善点を教えてくれる。

このコードでなにか改善点はありますか？

C# Employee.cs:1-10

GitHub Copilot

> 1 参照 使用済み

はい、いくつかの改善点があります。

1. プロパティ名のキャメルケースをパスカルケースに変更する。
2. `[✉] id` プロパティを `Id` に、`[✉] name` プロパティを `Name` に変更する。

■さらに改善後のコードを提案してくれる。右上のボタンをクリックして、現在のエディターにこの提案を適用できる。

以下は改善後のコードです。

```
/// <summary>
/// 従業員を表すクラス
/// </summary>
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

PrimeService\Employee.cs に適用

これにより、C# の命名規則に従ったコードになります。

戻る リセット 確認

# コードの改善のその他の例

- ・セキュリティの強化
  - ・脆弱性のあるコードの発見と修正
- ・リファクタリング（品質向上）
  - ・複雑なコードをよりシンプルな形にする
  - ・コードの重複を排除する
  - ・適切な名前に変更する
  - ・リテラル（コード中に直接書かれた文字列）の定数化
  - ・コーディング規約に従うようにする（改行・空白の使い方など）
- ・パフォーマンスの改善
  - ・適切なアルゴリズムの使用、SIMD命令の使用などによる高速化
  - ・メモリの使用量を減らす

Single Instruction, Multiple Data。  
1回の命令で複数のデータを同時に処理するCPU命令。

## ■実行時間を改善できるコードの例

```
1 reference
static List<int> RemoveDuplicates(List<int> list)
{
    List<int> result = [];
    foreach (int item in list)
    {
        if (!result.Contains(item))
        {
            result.Add(item);
        }
    }
    return result;
}
```

※このメソッドは、リストから重複する要素を取り除いた新しいリストを作る。  
たとえば [1, 1, 2, 2, 2, 3, 3, 3, 3, 3] を与えると [1, 2, 3] が返される。

※`result.Contains(item)`は`result`の要素数に比例した探索時間がかかる。  
ループを繰り返すごとに`result`の要素が増え、`Contains`の計算量は $O(0)$ 、 $O(1)$ 、 $O(2)$ 、 $\dots$   $O(n)$  と増加する。  
全体の計算量は  $O(0 + 1 + 2 + \dots + n) = O(n * (n + 1) / 2) \asymp O(n^2)$  となる。  
つまり要素数の2乗に比例した時間がかかる。

ビッグオーバー表記  
詳しくはアルゴリズム  
の教科書などを参照

## ■GitHub Copilotが、改善されたコードを提案してくれる

The screenshot shows the GitHub Copilot interface. At the top, there's a user icon and a dark bar. Below that, a message asks if the performance of the `RemoveDuplicates` method can be improved. A button indicates the code is at `Program.cs:21-42`. The main area shows the GitHub Copilot logo and the status "Generating...". It notes that one reference was used. A button labeled `RemoveDuplicates` is highlighted. A text box contains a note about using a `HashSet` to remove duplicates efficiently. Below this, the generated code is shown:

```
static List<int> RemoveDuplicates(List<int> list)
{
    HashSet<int> set = new HashSet<int>(list);
    return new List<int>(set);
}
```

※ハッシュセットは重複を許さない順不同の集合であり、要素の追加は $O(1)$ 、すなわち一定時間で実行できる。  
listの要素  $n$  個がハッシュセットに追加される場合、全体の計算量は  $O(1) * n = O(n)$  となる。  
つまり要素数に比例した時間で処理を実行できる。

## ■要素の重複の削除処理: 改善前と改善後の比較

要素数 (n)	改善前 リストを使用 計算量= $O(n^2)$	改善後 ハッシュセットを使用 計算量= $O(n)$
10	100	10
100	10,000	100
1,000	1,000,000	1,000
10,000	100,000,000	10,000
100,000	10,000,000,000	100,000

要素数が多い場合、ハッシュセットのほうがはるかに効果的!!  
(要素数10万の場合リストに比べて10万分の1の計算量)

# モジュール5

- ・インラインチャットを使用したコード改善の例
- ・コードの改善のその他の例
  - ・セキュリティ
  - ・リファクタリング
  - ・パフォーマンス
- ・まとめ

# モジュール5 まとめ

- GitHub Copilotを使用して、以下のようなコードの改善をすばやく実行できる。
- コードの改修: テストコードにテストケースを追加したり、コードを理解しやすい形にリファクタリングしたりする
- セキュリティの強化: コードの脆弱性を発見し、修正する
- コード品質の維持・改善: 一般的なコーディング規約に沿って、不適切な名前を修正する
- コードの性能の改善: 不適切なアルゴリズムの使用などの問題を発見し、改善する

# コースの構成



GitHub Copilot を使用してアプリ開発を高  
速化する

- ・モジュール1: GitHub Copilot の概要
- ・モジュール2: GitHub Copilot を使用してドキュメントを生成する
- ・モジュール3: GitHub Copilot を使用してコード機能を開発する
- ・モジュール4: GitHub Copilot を使用して単体テストを開発する
- ・モジュール5: GitHub Copilot を使用してコードの改善を実装する
- ・**補足: GitHub Copilotの最新機能のご紹介**

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

# ■ GitHub Copilot **Code Suggestion**: 2022/6/6 一般提供開始

The screenshot shows a code editor window with a dark theme. The file is named 'Class3.cs' and is part of a project 'PrimeService'. The code defines a class 'PrimeService' with a static method 'IsPrime' that checks if a number is prime. A yellow callout box with the text '[Alt + ¥]' points to the cursor position in the code editor, indicating the keyboard shortcut for accepting the suggested code.

```
1  namespace MyPrimeService
2  {
3  }
4  class PrimeService
5  {
6      public static bool IsPrime(Long number)
7      {
8          if (number <= 1) return false;
9          if (number == 2) return true;
10         if (number % 2 == 0) return false;
11         for (Long i = 3; i <= Math.Sqrt(number); i += 2)
12         {
13             if (number % i == 0) return false;
14         }
15         return true;
16     }
17 }
```

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

# ■ GitHub Copilot Chat: 2023/12/29 一般提供開始

A screenshot of a code editor showing a C# file named Class3.cs. The code defines a namespace `MyPrimeService` and a class `PrimeService`. A cursor is at the start of the class definition. Below the code, there is an inline chat interface with a text input field containing `/generate IsPrime`. The interface includes icons for copy, paste, microphone, and a dropdown menu labeled `GPT-4o`.

```
1 namespace MyPrimeService
2 {
3     0 個の参照
4     class PrimeService
5     {
6
7         }
8     }
9 }
```

オンラインチャット  
[Ctrl + I]

A screenshot of a code editor showing a chat view for a file named sample17. The title bar says "COPILOT の編集". The main area displays a message from GPT-4o: "output carefully before use." followed by "As an internal user, additional telemetry is collected. If you work on a project that contains customer content, you must disable telemetry." Below this, there are three command suggestions: `/fix`, `/tests`, and `/explain`. A dropdown menu shows the current file is "Class3.cs Current file". The bottom status bar shows "行 7, 列 1 スペース: 4 UTF-8 CRLF {} C#".

チャットビュー  
[Ctrl + Alt + I]

<https://github.blog/news-insights/product-news/github-copilot-chat-now-generally-available-for-organizations-and-individuals/>

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

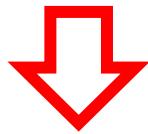
## ■ GitHub Copilot Edits: 2025/2/6 一般提供開始



<https://github.blog/news-insights/product-news/github-copilot-the-agent-awakens/#copilot-edits-now-ga-in-vs-code-%f0%9f%8e%89>

## ■ GitHub Copilot **Next edit suggestions (NES)**: 2025/2/6 プレビュー開始

```
0 個の参照  
static int Add(int a, int b)  
{  
    return a + b;  
}
```



第3引数 int c を  
手動で追加

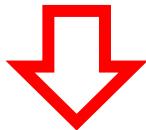
```
0 個の参照  
💡 static int Add(int a, int b, int c)  
{  
    return a + b;  
}
```

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

## ■ GitHub Copilot **Next edit suggestions (NES)**: 2025/2/6 プレビュー開始

```
0 個の参照  
static int Add(int a, int b)  
{  
    return a + b;  
}
```



第3引数 int c を  
手動で追加

```
0 個の参照  
💡 static int Add(int a, int b, int c)  
{  
    return a + b;  
}
```



```
4 {  
0 個の参照  
5 static int Add(int a, int b, int c)  
6 {  
7     return a + b + c;  
8 }  
9 }
```

GitHub Copilotがユーザーの  
「次の編集」を予測して提案  
を表示。Tabで確定。

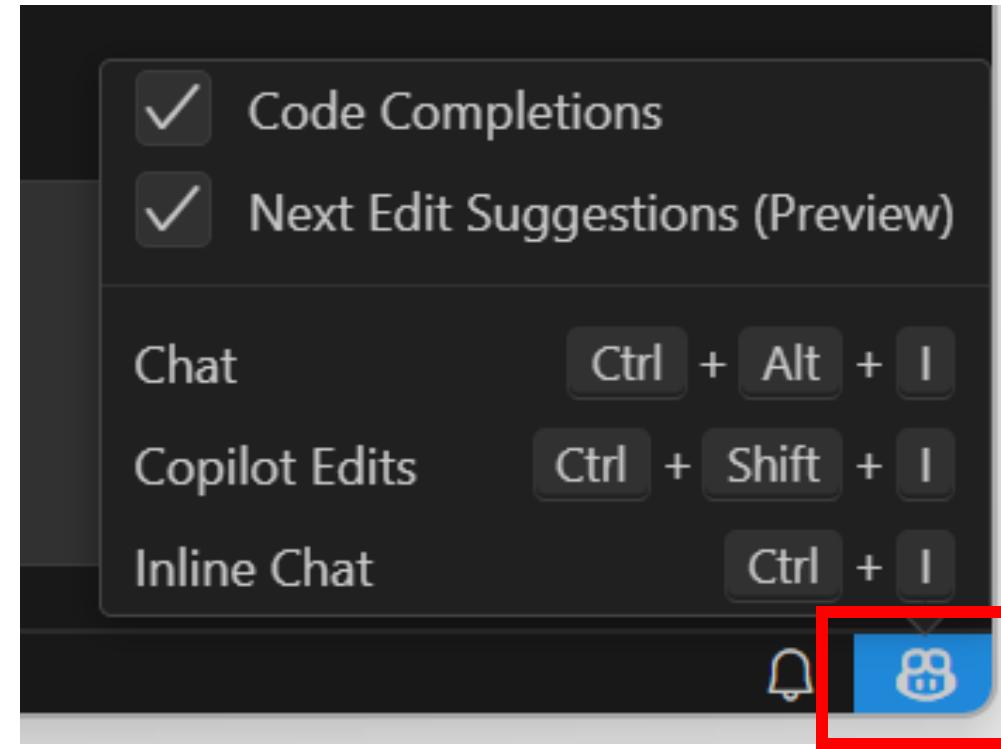
<https://github.blog/changelog/2025-02-06-next-edit-suggestions-agent-mode-and-prompts-files-for-github-copilot-in-vs-code-january-release-v0-24/>

■ GitHub Copilot **Next edit suggestions (NES)**を使用するには、  
Visual Studio Codeの設定で有効に設定する。

The screenshot shows the Visual Studio Code settings interface. On the left, there is a sidebar with categories like '機能 (3)', 'ノートブック (3)', '拡張機能 (37)', 'C# (2)', 'Debugger (1)', 'OmniSharp (1)', 'GitHub Copilot (4)', 'GitHub Copilot C... (30)', 'Preview (7)', 'Experimental (14)', and 'マージの競合 (1)'. The 'GitHub Copilot (4)' item is highlighted with a red box. On the right, there are several configuration sections:

- Github › Copilot › Next Edit Suggestions: Enabled** プレビュー  
✓ 次の編集候補 (NES) を有効にするかどうか。
- Notebook: Cell Execution Time Verbosity**  
セルステータスバーのセル実行時間の詳細を制御します。  
default
- Notebook › Diff › Experimental: Toggle Inline**  
✓ 試験的なノートブックのインライン diff エディターを切り替えるには、コマンドを有効にします。

■Visual Studio Code **Insiders** (Visual Studio Codeの最新開発版)  
を使用している場合は、画面右下のCopilotボタンから  
Code CompletionsやNESが簡単にON・OFF切り替えできる



Visual Studio Code Insidersはここからダウ  
ンロードできる。一つのPCに、通常版と  
Insidersを両方インストールして、どちらも  
利用できる。

[Download Visual Studio Code Insiders](#)

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
  - GitHub Copilot Chat
  - GitHub Copilot Edits
  - GitHub Copilot Next edit suggestions
  - GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

■ GitHub Copilot **agent mode**: 2025/2/6 プレビュー開始。**Insiders**で利用可能。  
Editsの機能に加え、**コマンドの提案・実行**ができるようになっている。



<https://github.blog/changelog/2025-02-06-next-edit-suggestions-agent-mode-and-prompts-files-for-github-copilot-in-vs-code-january-release-v0-24/>

■GitHub Copilot **agent mode**で、プロンプトを入力。かなり複雑な手順を指示できる。

The screenshot shows the GitHub Copilot Agent Mode interface. At the top left is a button labeled "Add Files...". The main area contains three paragraphs of Japanese text:

C#のソリューションを作成し、ソリューションに PrimeService クラスライブラリプロジェクトと PrimeService.Tests xunit テストプロジェクトを追加してください。  
PrimeService.Tests は PrimeService プロジェクトを参照します。PrimeService プロジェクトに、入力された整数が素数かどうかを判定する IsPrime メソッドを作成します。  
PrimeService.Tests プロジェクトに、IsPrime メソッドの単体テストを記述します。単体テストでは1から15までの整数をテストしてください。最後に dotnet tests を実行して単体テストを実行し、テストが成功することを確認してください。

At the bottom right of the main panel are three buttons: "エージェント" (Agent), "GPT-4o", and a blue arrow icon. Below the main panel is a dark footer bar with a bell icon and a gear icon.

## ■ GitHub Copilot agent mode

The screenshot shows the GitHub Copilot agent mode interface. At the top, there are tabs for "チャット" and "COPilot の編集". On the right side, there are icons for refresh, settings, plus, and close. Below the tabs, there is a small profile picture of a cat.

A red box highlights a block of Japanese text from the AI's response:

C#のソリューションを作成し、ソリューションに PrimeService クラスライブラリプロジェクトと PrimeService.Tests xunit テストプロジェクトを追加してください。PrimeService.Tests は PrimeService プロジェクトを参照します。PrimeService プロジェクトに、入力された整数が素数かどうかを判定する IsPrime メソッドを作成します。PrimeService.Tests プロジェクトに、IsPrime メソッドの単体テストを記述します。単体テストでは1から15までの整数をテストしてください。最後に dotnet tests を実行して単体テストを実行し、テストが成功することを確認してください。

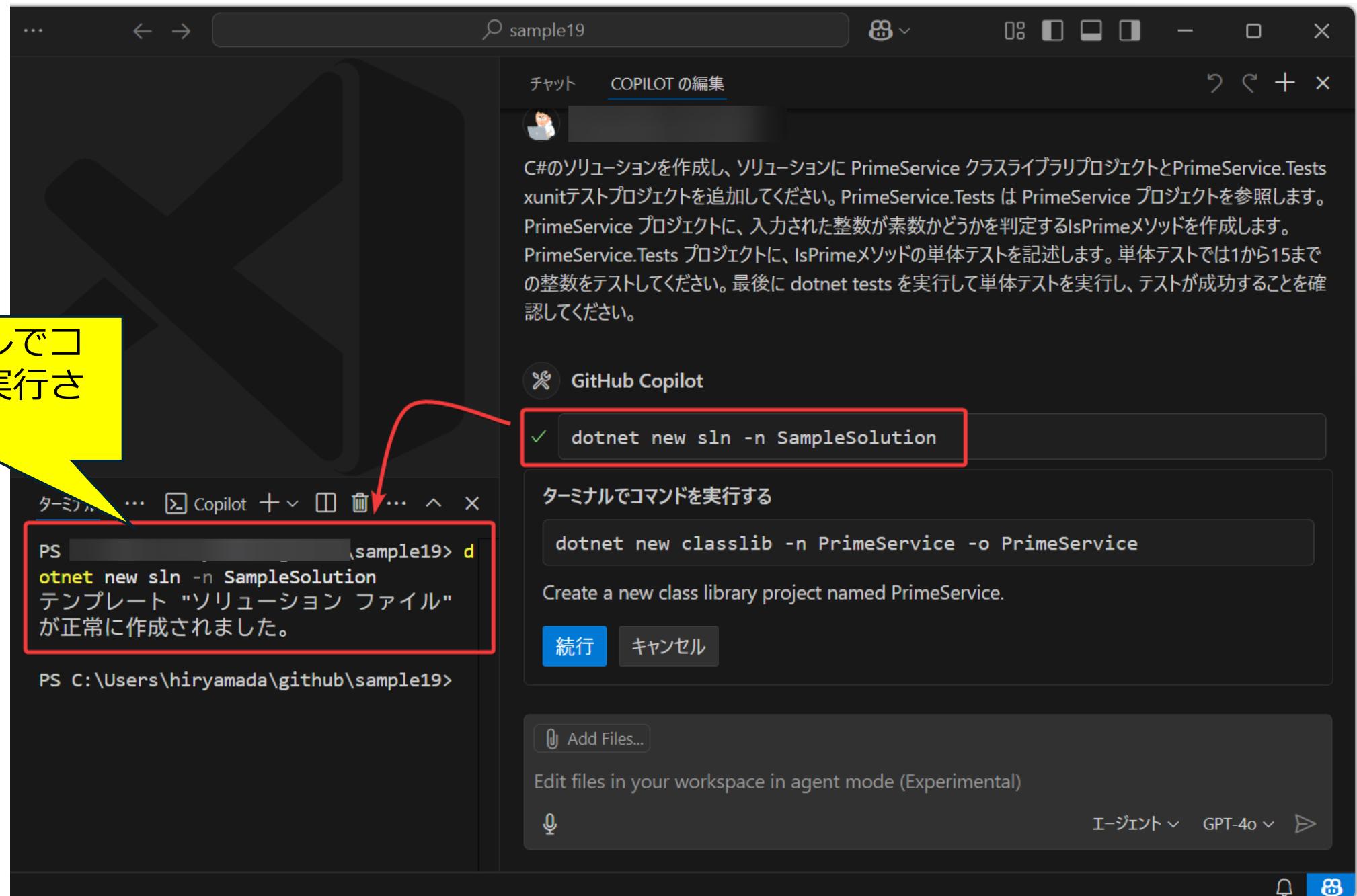
In the center, the GitHub Copilot logo is shown next to the text "GitHub Copilot". Below it, a terminal window displays the command "dotnet new sln -n SampleSolution". A yellow callout bubble points to this command with the text "必要なコマンドを提案してくる！".

At the bottom left, another yellow callout bubble points to the "続行" (Continue) button in the terminal window with the text "「続行」をクリックして承認".

At the bottom right, there is a "Add Files..." button, a message "Edit files in your workspace in agent mode (Experimental)", and a status bar with "エージェント" and "GPT-4o" dropdowns, along with a right-pointing arrow icon.

## ■ GitHub Copilot agent mode

ターミナルでコマンドが実行された！



## ■ GitHub Copilot agent mode

The screenshot shows the GitHub Copilot agent mode interface. At the top, there's a search bar with 'sample19' and a toolbar with various icons. The main area has tabs for 'チャット' (Chat) and 'COPilot の編集' (Edit). A message from GitHub Copilot suggests creating a C# solution with a PrimeService class library and a PrimeService.Tests unit test project. Below this, a command history shows 'dotnet new sln -n SampleSolution' with a green checkmark. A yellow callout points to the right side of the screen with the text 'さらに次のコマンドを提案してくる！' (It will also propose the next command!). In the terminal pane, a command 'dotnet new classlib -n PrimeService -o PrimeService' is shown with a red box around it. A yellow callout points to the '続行' (Continue) button in the terminal window with the text '「続行」をクリックして承認' (Click 'Continue' to accept). The bottom of the interface shows workspace editing tools and a status bar indicating 'エージェント v GPT-4o'.

sample19

チャット COPilot の編集

C#のソリューションを作成し、ソリューションに PrimeService クラスライブラリプロジェクトとPrimeService.Tests xunitテストプロジェクトを追加してください。PrimeService.Tests は PrimeService プロジェクトを参照します。PrimeService プロジェクトに、入力された整数が素数かどうかを判定するIsPrimeメソッドを作成します。PrimeService.Tests プロジェクトに、IsPrimeメソッドの単体テストを記述します。単体テストでは1から15までの整数をテストしてください。最後に dotnet tests を実行して単体テストを実行し、テストが成功することを確認してください。

GitHub Copilot

✓ dotnet new sln -n SampleSolution

ターミナルでコマンドを実行する

dotnet new classlib -n PrimeService -o PrimeService

Create a new class library project named PrimeService.

続行 キャンセル

「続行」をクリックして承認

Add Files...

Edit files in your workspace in agent mode (Experimental)

エージェント v GPT-4o ▶

## ■ GitHub Copilot agent mode

ターミナルでコマンドが実行された！

ターミナル

```
dotnet new classlib -n PrimeService -o PrimeService
テンプレート "クラス ライブラリ" が正常に作成されました。
```

作成後の操作を処理しています...
\sample19\PrimeService\PrimeService.csproj を復元しています:
正常に復元されました。

Copilot

sample19

チャット COPILOT の編集

このノードークをトマホークに PrimeService ノードークノードーク。 PrimeService.Tests は PrimeService プロジェクトを参照します。 PrimeService プロジェクトに、入力された整数が素数かどうかを判定する IsPrime メソッドを作成します。 PrimeService.Tests プロジェクトに、 IsPrime メソッドの単体テストを記述します。 単体テストでは 1 から 15 までの整数をテストしてください。 最後に dotnet tests を実行して単体テストを実行し、 テストが成功することを確認してください。

GitHub Copilot

✓ dotnet new sln -n SampleSolution

✓ dotnet new classlib -n PrimeService -o PrimeService

ターミナルでコマンドを実行する

```
dotnet new xunit -n PrimeService.Tests -o PrimeService.Tests
```

Create a new xUnit test project named PrimeService.Tests.

続行 キャンセル

Add Files...

Edit files in your workspace in agent mode (Experimental)

エージェント GPT-4o

PS \sample19>

## ■ GitHub Copilot agent mode

複数ファイルの編集が可能な点についてはEditsと同様。

プロジェクト構造:

- PrimeService.Tests
- bin
- obj
- Class1.cs
- PrimeService.csproj
- PrimeService.Tests.csproj
- bin
- obj
- PrimeService.Tests.cs
- UnitTest1.cs
- SampleSolution.sln

コード:

```
using Xunit;
using PrimeService;

namespace PrimeService.Tests
{
    public class UnitTest1
    {
        [Fact]
        public void Test1()
        {
            [Theory]
            [InlineData(1, false)]
            [InlineData(2, true)]
            [InlineData(3, true)]
            [InlineData(4, false)]
            [InlineData(5, true)]
            [InlineData(6, false)]
            [InlineData(8, false)]
            [InlineData(9, false)]
        }
    }
}
```

チャット:

- Class1.cs +13 -3
- dotnet test
- don't forget to add xunit test cases for UnitTest1.cs
- Github Copilot
- UnitTest1.cs +26 -5

ターミナル:

```
PS C:\Users\hiriyamada\github\sample19> dotnet sln add PrimeService.Tests/PrimeService.Tests.csproj
プロジェクト `PrimeService.Tests\PrimeService.Tests.csproj` をソリューションに追加しました。
PS C:\Users\hiriyamada\github\sample19> dotnet add PrimeService.Tests/PrimeService.Tests.csproj reference PrimeService/PrimeService.csproj
参照 `..\PrimeService\PrimeService.csproj` がプロジェクトに追加されました。
PS C:\Users\hiriyamada\github\sample19>
```

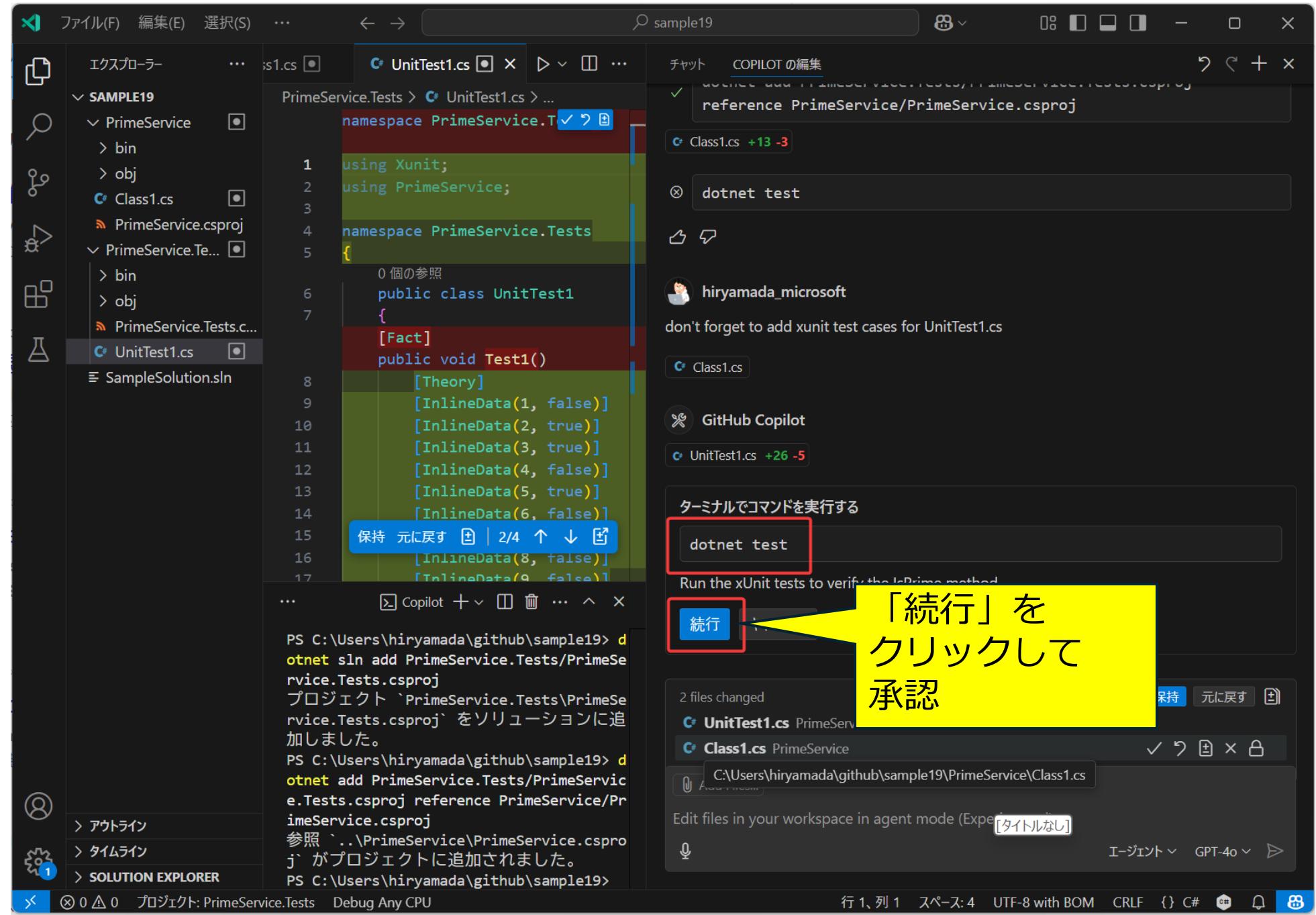
GitHub Copilot のメッセージ:

ここで、GitHub Copilotが、なぜか単体テストのコードの生成をせずに、単体テストを実行しようとしている！

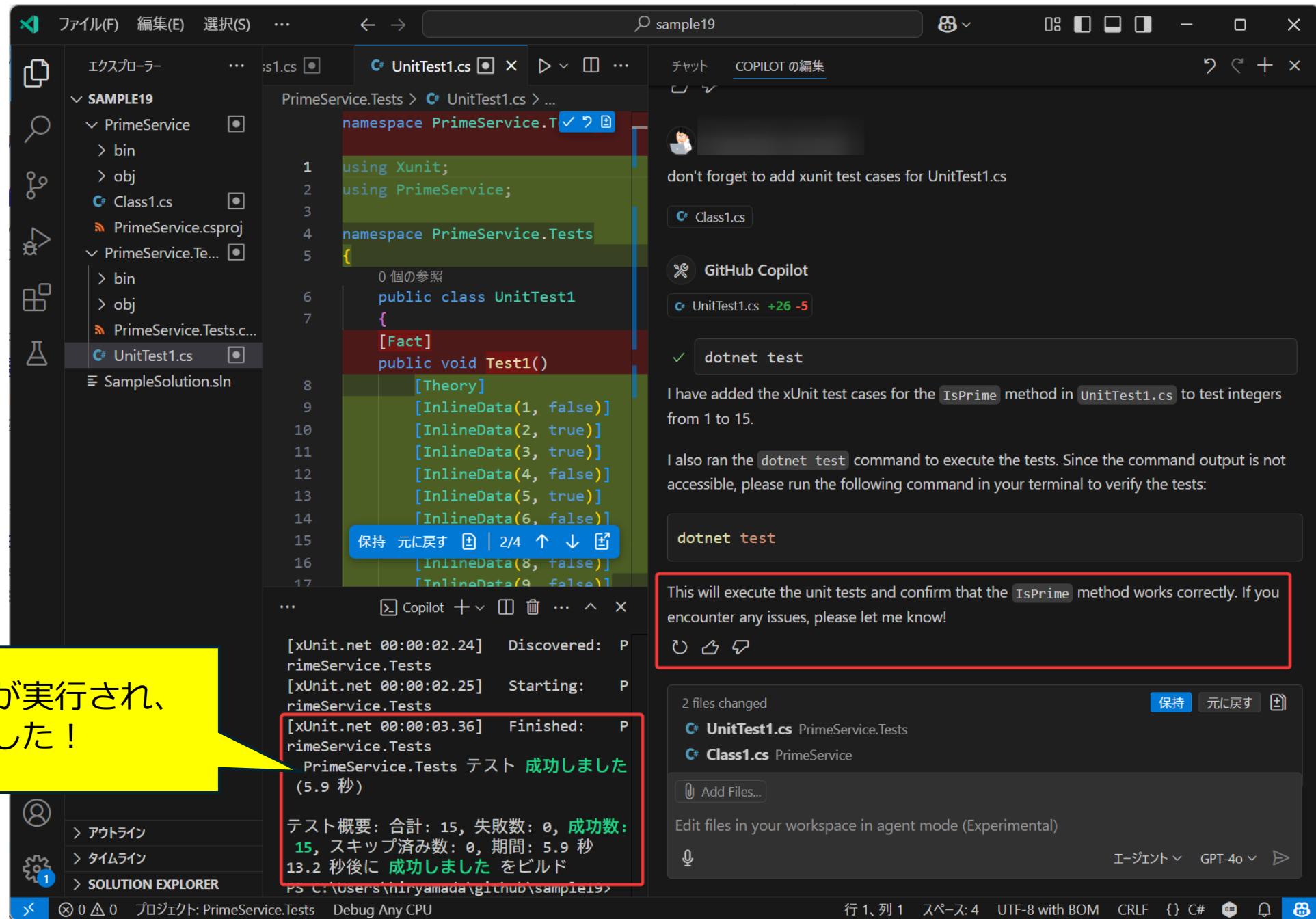
そこで、ユーザー（私）が、テストケースをちゃんと生成するようにチャットで追加の指示を出した

するとGitHub Copilotがその指示を受けて、単体テストのコードを生成

# ■ GitHub Copilot **agent mode**



# ■ GitHub Copilot **agent mode**



## ■ GitHub Copilot agent mode

The screenshot shows a Visual Studio Code workspace for a project named "SAMPLE19". The code editor displays `UnitTest1.cs` with XUnit test cases for the `IsPrime` method. A yellow callout box highlights the GitHub Copilot panel, which provides feedback and execution commands. The terminal at the bottom shows the output of a `dotnet test` command.

エージェントの動作はここまで。  
あとはEditsと同様に「保持」、  
「完了」とクリックして、すべて  
の編集結果を受理する。

```
namespace PrimeService.Tests
{
    using Xunit;
    using PrimeService;

    namespace PrimeService.Tests
    {
        public class UnitTest1
        {
            [Fact]
            public void Test1()
            {
                [Theory]
                [InlineData(1, false)]
                [InlineData(2, true)]
                [InlineData(3, true)]
                [InlineData(4, false)]
                [InlineData(5, true)]
                [InlineData(6, false)]
                [InlineData(8, false)]
                [InlineData(9, false)]
            }
        }
    }
}

[xUnit.net 00:00:02.24] Discovered: PrimeService.Tests
[xUnit.net 00:00:02.25] Starting: PrimeService.Tests
[xUnit.net 00:00:03.36] Finished: PrimeService.Tests
PrimeService.Tests テスト 成功しました (5.9 秒)

テスト概要: 合計: 15, 失敗数: 0, 成功数: 15, スキップ済み数: 0, 期間: 5.9 秒
13.2 秒後に 成功しました をビルド
PS C:\Users\hiriyamada\github\sample19>
```

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

# ご参考: GitHub Copilot **Workspace**

- 2024/4/30 テクニカルレビュー開始
- Visual Studio Code の中では**なく**、GitHubのWebサイト上で利用する機能。
- 一口で言えば「AIでプルリクエストを作成する」機能。
  - リポジトリのコードの修正や新機能の追加などの作業のための「ワークスペース」を作る
  - 「ワークスペース」内で、GitHub Copilotが、開発者の指示にしたがって、ファイルの作成や編集を行う
  - 最後にプルリクエストの作成を行う
- これらすべての作業がすべて「ワークスペース」上で完結する

## ■ GitHub Copilot Workspace

<https://copilot-workspace.githubnext.com/> にアクセスする

The screenshot shows the GitHub Copilot Workspace interface. On the left, there's a sidebar with 'Recent' (1), 'Bookmarked' (0), and 'Completed' (0) sections. The main area has a 'Start new session' section with a 'Choose a repository' button (which is highlighted with a red box and arrow) and a 'Create new repository' link. Below this is a 'Recent sessions' section with a card for 'Add PrimeService' (by hirayamada\_microsoft/test, #1 pull request created 9 minutes ago). On the right, a modal window titled 'Choose a repository' is open, featuring a search bar and a list of repositories. One repository, 'hirayamada\_microsoft/test' (with a yellow exclamation mark icon), is highlighted with a red box. A large yellow callout box points to this repository with the text '操作対象のリポジトリを選択する' (Select the target repository).

Copilot Workspace Preview Give feedback

Recent 1

Bookmarked 0

Completed 0

Start new session

hiriyamada\_microsoft/test

Choose a repository

Create new repository

Recent sessions

Add PrimeService

hiriyamada\_microsoft/test #1

Created pull request #1 9 minutes ago

Choose a repository

Search repositories...

hiriyamada\_microsoft/test

操作対象のリポジトリを選択する

## ■ GitHub Copilot Workspace

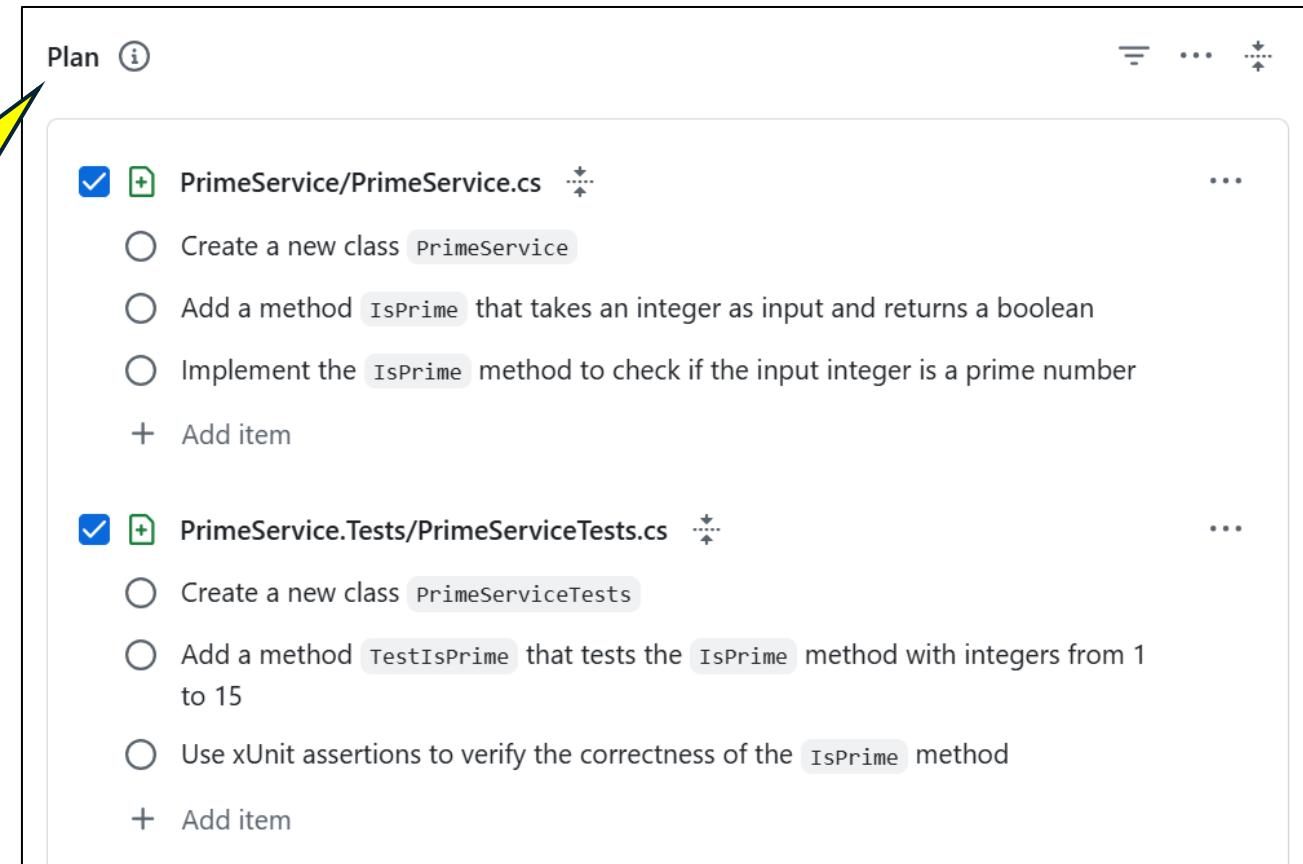
The screenshot shows the GitHub Copilot Workspace interface. At the top, there's a browser-like header with a profile icon, the text 'Copilot Workspace · hiryamada\_m', and a search bar containing the URL [https://copilot-workspace.githubnext.com/hiryamada\\_microsoft/test?s=e8cf](https://copilot-workspace.githubnext.com/hiryamada_microsoft/test?s=e8cf). Below the header, the title 'Copilot Workspace' is displayed with a 'Preview' button and a 'Give feedback' link. A navigation bar shows the repository 'hiryamada\_microsoft / test' and the branch 'main'. The main area is titled 'Task' and contains two tabs: 'Write' (selected) and 'Preview'. The 'Write' tab contains the following text:

C#のソリューションを作成し、ソリューションに PrimeService クラスライブラリプロジェクトとPrimeService.Tests xunitテストプロジェクトを追加してください。PrimeService.Tests は PrimeService プロジェクトを参照します。PrimeService プロジェクトに、入力された整数が素数かどうかを判定するIsPrimeメソッドを作成します。PrimeService.Tests プロジェクトに、IsPrimeメソッドの単体テストを記述します。単体テストでは1から15までの整数をテストしてください。最後に dotnet tests を実行して単体テストを実行し、テストが成功することを確認してください。

Below the text are two buttons: 'Brainstorm' (green) and 'Generate plan' (gray). A yellow callout box points to the 'Generate plan' button with the text '「プラン」を生成する'. Another yellow callout box points to the right side of the interface with the text 'この「ワークスペース」で解決したい「タスク」を記述する'.

## ■ GitHub Copilot Workspace

ファイルの作成・編集の  
「プラン」が作成される。  
「プラン」を確認し、実行  
する。



# ■ GitHub Copilot Workspace

The screenshot shows the GitHub Copilot Workspace interface. At the top, there's a navigation bar with icons for preview, giving feedback, and user profile. Below it, a header bar shows the repository 'hiryamada\_microsoft / test' and the branch 'main'. On the right side of the header, there's a 'Create pull request' button with a red border.

The main area is divided into sections: 'Task' (containing a Japanese instruction), 'Plan' (listing tasks for 'PrimeService/PrimeService.cs' and 'PrimeService.Tests/PrimeServiceTests'), and a code editor.

**Task:** C#のソリューションを作成し、ソリューションに PrimeService クラスライブラリproj... (C# Solution を作成し、ソリューションに PrimeService クラスライブラリプロジェクトを追加)

**Plan:**

- PrimeService/PrimeService.cs
  - ✓ Create a new class `PrimeService`
  - ✓ Add a method `IsPrime` that takes an integer as input and returns a boolean
  - ✓ Implement the `IsPrime` method to check if the input integer is a prime number
  - + Add item
- PrimeService.Tests/PrimeServiceTests
  - ✓ Create a new class `PrimeServiceTests`
  - ✓ Add a method `TestIsPrime` that tests the `IsPrime` method with integers from 1 to 15
  - ✓ Use xUnit assertions to verify the correctness of the `IsPrime` method
  - + Add item

A yellow callout box points to the 'PrimeService/PrimeService.cs' section of the plan, containing the text: 「プラン」が実行され、ソースファイルが作成された。

**Code Editor:**

```
1+using System;
2+
3+namespace PrimeService
4+{
5+    public class PrimeService
6+    {
7+        public bool IsPrime(int number)
8+        {
9+            if (number <= 1) return false;
10+           ...
11+        }
12+    }
13+}
```

# ■ GitHub Copilot Workspace

The screenshot shows the GitHub Copilot Workspace interface. At the top left is the Copilot Workspace logo and a preview/give feedback button. The top right features a user icon, a notifications badge (0), and a three-dot menu. A yellow callout box with black text is overlaid on the interface, pointing towards the top right. The main area displays a task card for a 'PrimeService' project, listing steps to implement a prime number checker. To the right is a code editor showing the implementation of the PrimeService class.

この変更を含む「プルリクエスト」を作成する

Task C#のソリューションを作成し、ソリューションに PrimeService クラスを追加する

Plan

PrimeService/PrimeService.cs

- Create a new class `PrimeService`
- Add a method `IsPrime` that takes an integer as input and returns a boolean
- Implement the `IsPrime` method to check if the input integer is a prime number
- + Add item

PrimeService.Tests/PrimeServiceTests.cs

- Create a new class `PrimeServiceTests`
- Add a method `TestIsPrime` that tests the `IsPrime` method with integers from 1 to 15
- Use xUnit assertions to verify the correctness of the `IsPrime` method
- + Add item

Create pull request

changed 2 Commands

```
1+using System;
2+
3+namespace PrimeService
4+{
5+    public class PrimeService
6+    {
7+        public bool IsPrime(int number)
8+        {
9+            if (number <= 1) return false;
+        }
+    }
+}
```

## ■ GitHub Copilot Workspace

The screenshot shows the GitHub Copilot Workspace interface. On the left, there's a sidebar with 'Copilot Workspace' and 'Preview' buttons. Below that are sections for 'Task' and 'Plan'. In the 'Plan' section, several tasks are listed with checkmarks: 'Create a new class PrimeService', 'PrimeService.Tests/PrimeServiceTests.cs', 'Create a new class PrimeServiceTests', and 'Add a method IsPrime that tests the IsPrime method with int'. A large yellow box highlights the text '適切な名前のブランチが作成される' (A branch with an appropriate name is created). Another yellow box highlights the text 'プルリクエストのタイトルが生成される。' (The title of the pull request is generated). A third yellow box highlights the text 'プルリクエストの説明文を手動で入力、または生成する' (Enter the pull request description manually or generate it). A fourth yellow box highlights the text 'プルリクエストを作成' (Create pull request).

適切な名前のブランチが作成される

プルリクエストのタイトルが生成される。

プルリクエストの説明文を手動で入力、または生成する

プルリクエストを作成

Create pull request

Copilot Workspace Preview Give feedback

Branch name \*

add-prime-service

Title \*

Add PrimeService class library and unit tests

Description

Describe the changes made in this pull request

Generate description

Attach share link

Commands

dotnet new classlib -o PrimeService

Revise...

dotnet new xunit -o PrimeService.Tests

Create pull request

## ■ GitHub Copilot **Workspace**

Code Issues Pull requests 1 Actions Projects Wiki Security Insights ...

# Add PrimeService class library and unit tests #1

**Open** hiryamada\_microsoft wants to merge 1 commit into **main** from **add-prime-service**

Conversation 0 Commits 1 Checks 0 Files changed 2 +50 -0

hiryamada\_microsoft commented now • edited

For more details, open the [Copilot Workspace session](#).

Add PrimeService class library and unit tests ... 5b12086

**No conflicts with base branch**  
Merging can be performed automatically.

Merge pull request You can also merge this with the command line.  
[View command line instructions](#)

Reviewers   
Suggestions  
**Copilot** Request  
Still in progress? Learn about draft PRs

Assignees   
No one—[assign yourself](#)

Labels   
None yet

Projects   
None yet

# 補足: GitHub Copilotの最新機能のご紹介

- GitHub Copilot Code Suggestion
- GitHub Copilot Chat
- GitHub Copilot Edits
- GitHub Copilot Next edit suggestions
- GitHub Copilot agent mode
- ご参考: GitHub Copilot Workspace
- まとめ

# 「補足: GitHub Copilotの最新機能のご紹介」のまとめ

- GitHub Copilotは、以下の順で進化してきました
  - Code suggestions
  - Chat
  - Edits
  - Next edit suggestions (NES) ※現在パブリックプレビュー中
  - agent mode ※現在パブリックプレビュー中
  - (ご参考: GitHub Copilot Workspaces: ※現在テクニカルプレビュー中)
- 特に Next edit suggestions (NES) と agent mode は 2025/2/6 に追加されたばかりの最新機能で、お手元ですぐに試せます
  - どちらも非常に便利で強力な機能であるため、ぜひ使いこなしていただきたいです

# コース全体のまとめ

- ・モジュール1: GitHub Copilotの概要（できること、プラン）
- ・モジュール2: コードのドキュメントを生成する
  - ・/doc スラッシュコマンド、スマートアクションなど
- ・モジュール3: コメントからコードを生成する
  - ・コメントからコードの「提案」を表示、確定
- ・モジュール4: 単体テストのコードを生成する
  - ・/tests スラッシュコマンド、チャット、Edits、テストの実行
- ・モジュール5: コードの品質やセキュリティを改善する
  - ・チャットを使用してコードを改善
- ・補足: GitHub Copilotの最新機能
  - ・Next edit suggestions で効率よくコードを編集
  - ・agent mode でコマンドを提案・実行