

GH-300

GitHub Copilot

part 2

本講義は以下のMicrosoft Learn教材に準拠しています。

[Course GH-300T00-A: GitHub Copilot - Training | Microsoft Learn](#)



Part 2 目次

- モジュール1 開発者のユースケース
 - ドキュメント作成
 - コード作成
- モジュール2 単体テストの開発
- モジュール3 GitHub Copilot Businessの概要
- モジュール4 GitHub Copilot Enterpriseの概要
- モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- モジュール6 PythonでGitHub Copilotを使用する（オプション）

開発者のユースケース: ドキュメント作成

- 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>
```

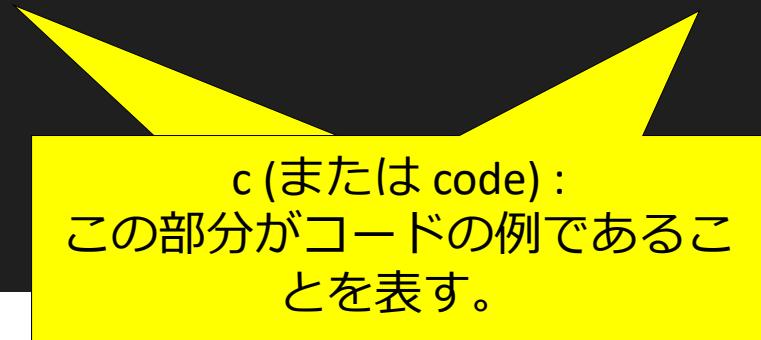
2 references

```
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  
= new Prim  
meService.IsPrime(input);  
lt);
```

Returns:
指定された数が素数の場合は `true`、それ以外の場合は `false`。



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。

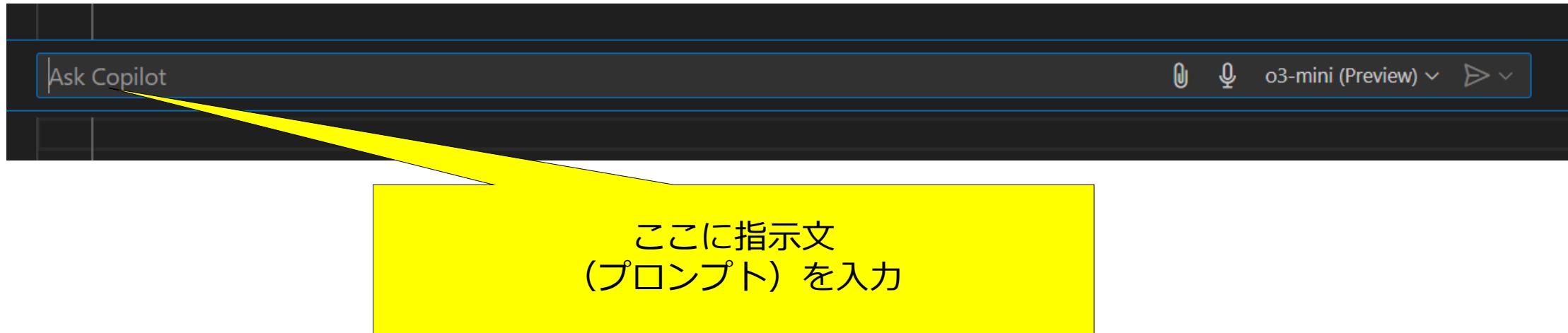
引数の説明が表示されるので、
ここに何を書けばよいのかわかりやすい！

開発者のユースケース: ドキュメント作成

- ・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] (Windows) / [Command + I] (Mac)
- 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 public bool IsPrime(int candidate)
9 {
10     if (candidate < 2)
11     {
12         return false;
13     }
14     for (int divisor = 2; divisor <= Math.Sqrt(candidate); ++divisor)
15     {
16         if (candidate % divisor == 0)
17         {
18             return false;
19         }
20     }
21
22 }
```

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] / [Command + 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 interface. In the top search bar, the text `/doc in Japanese` is being typed. Below the search bar, the text `2 references` is displayed. A code completion dropdown menu is open, showing the start of a method definition:

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

A yellow lightbulb icon is visible on the left side of the editor area, 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 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;
}
```

開発者のユースケース: ドキュメント作成

- ・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

開発者のユースケース: ドキュメント作成

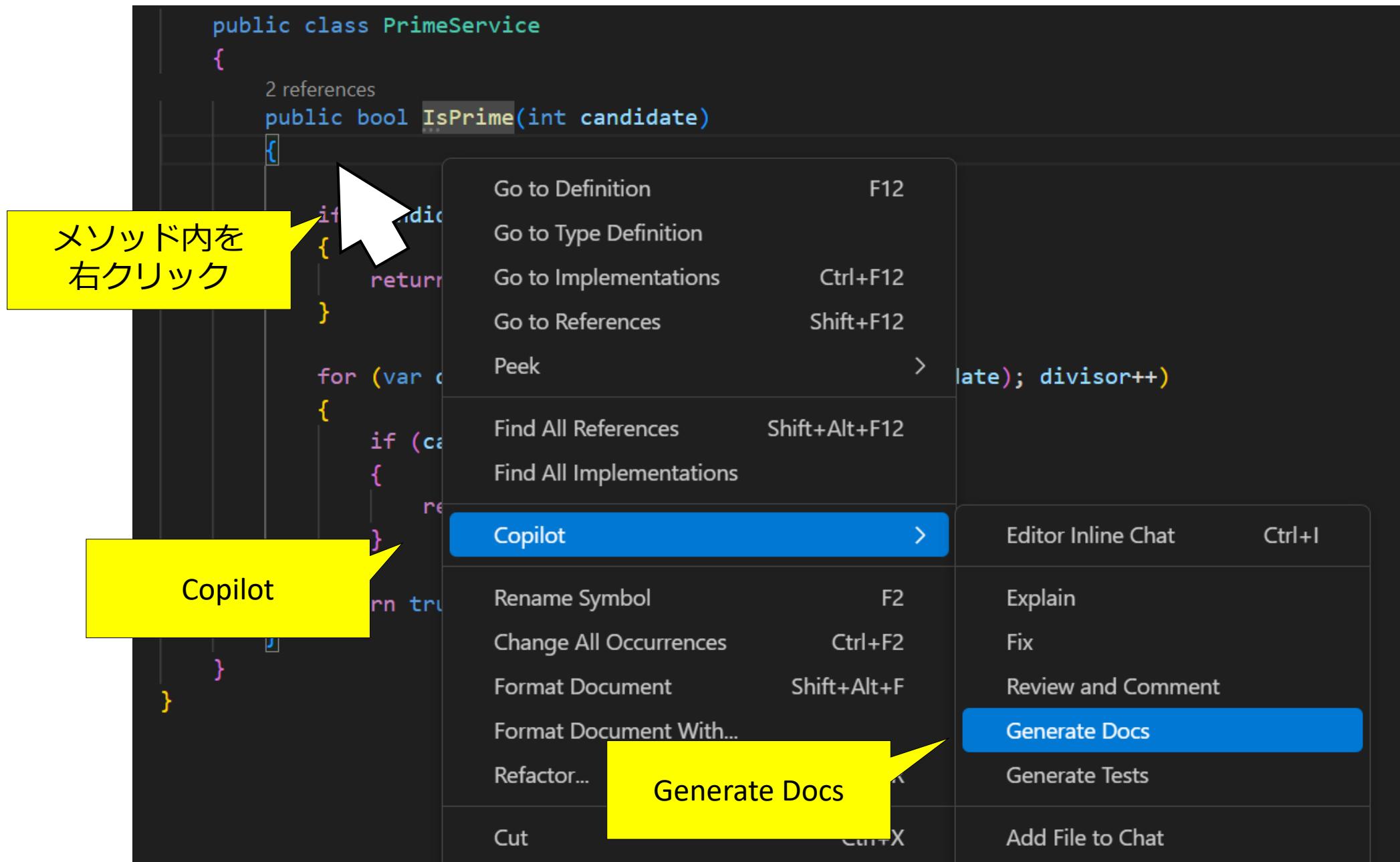
- ・ドキュメント（ドキュメントコメント）とは？
- ・ドキュメントの必要性
- ・ドキュメントの例
- ・インラインチャットを使ってドキュメントを書く
- ・スマートアクションを使ってドキュメントを書く
 - ・メソッドを右クリック
 - ・メソッド名をクリック→きらめきアイコン
- ・複数の候補の表示
- ・まとめ

■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。

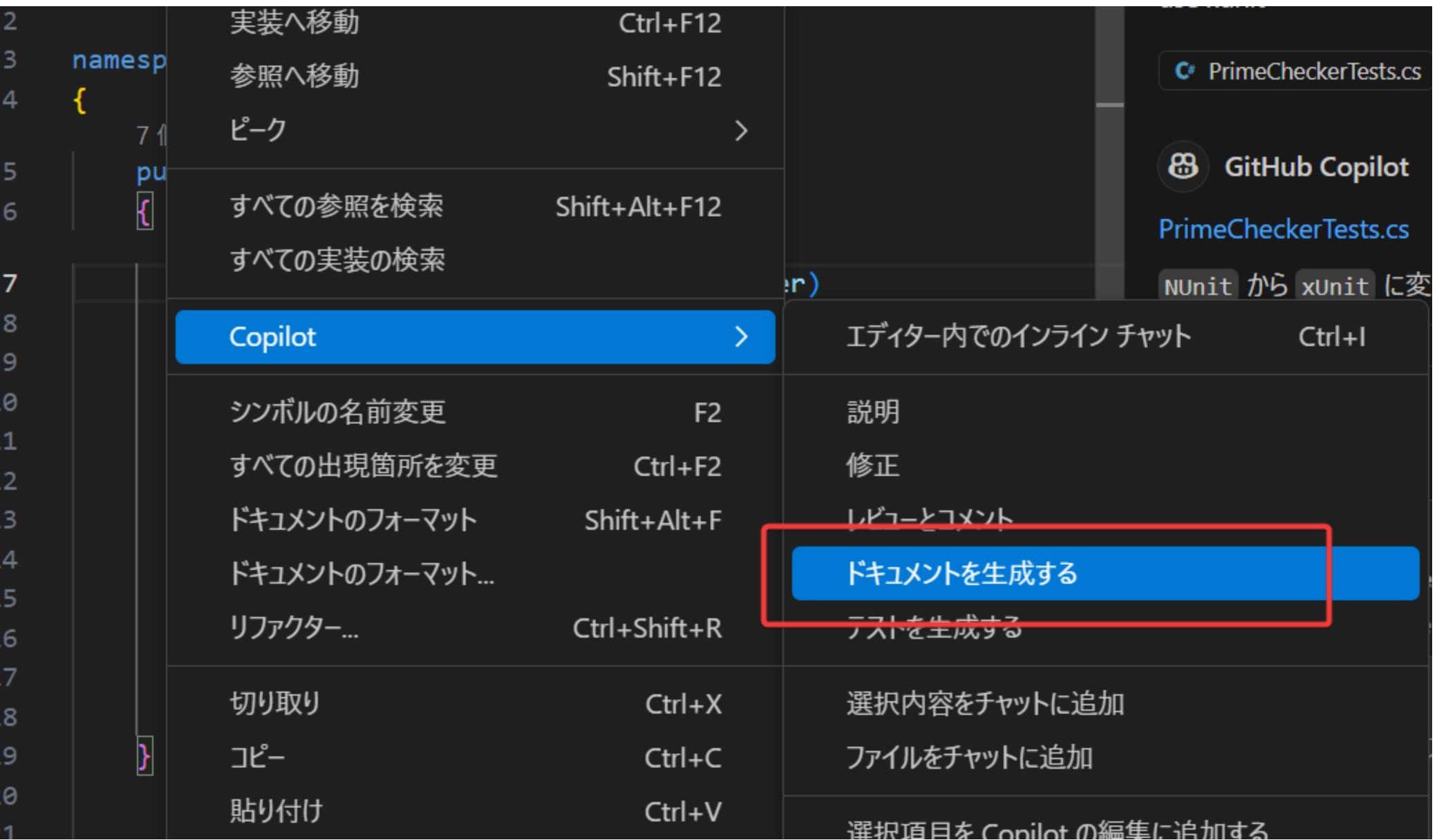
```
2 references
public bool IsPrime(int candidate)
{
    if (candidate < 2)
    {
        return false;
    }

    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 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 is presented in a dark-themed code editor with syntax highlighting for keywords like 'public', 'boolean', 'if', 'for', and 'Math.Sqrt'.

■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:

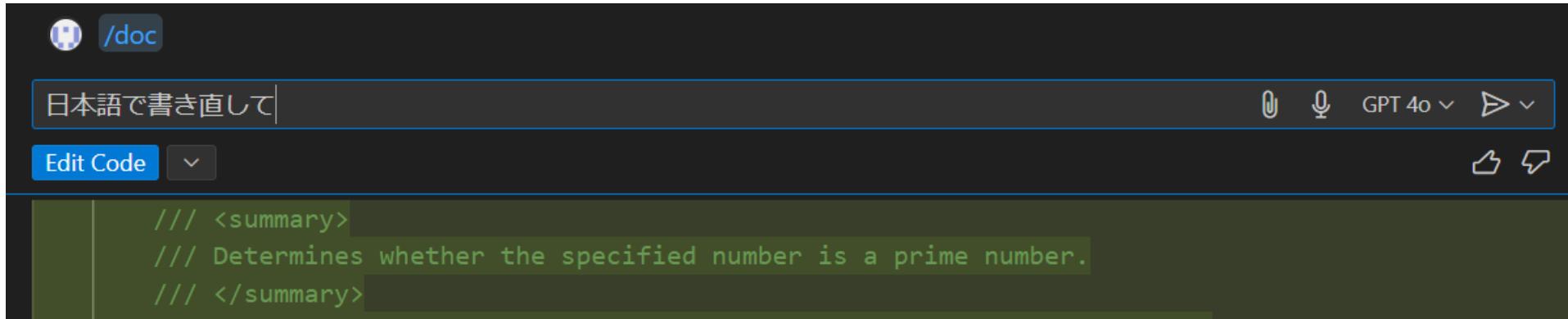
```
    /// <summary>
    /// Determines whether the specified number is a prime number.
    /// </summary>
    /// <param name="candidate">The number to check for primality.</param>
    /// <returns>
    /// <c>true</c> if the specified number is a prime number; otherwise, <c>false</c>.
    /// </returns>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

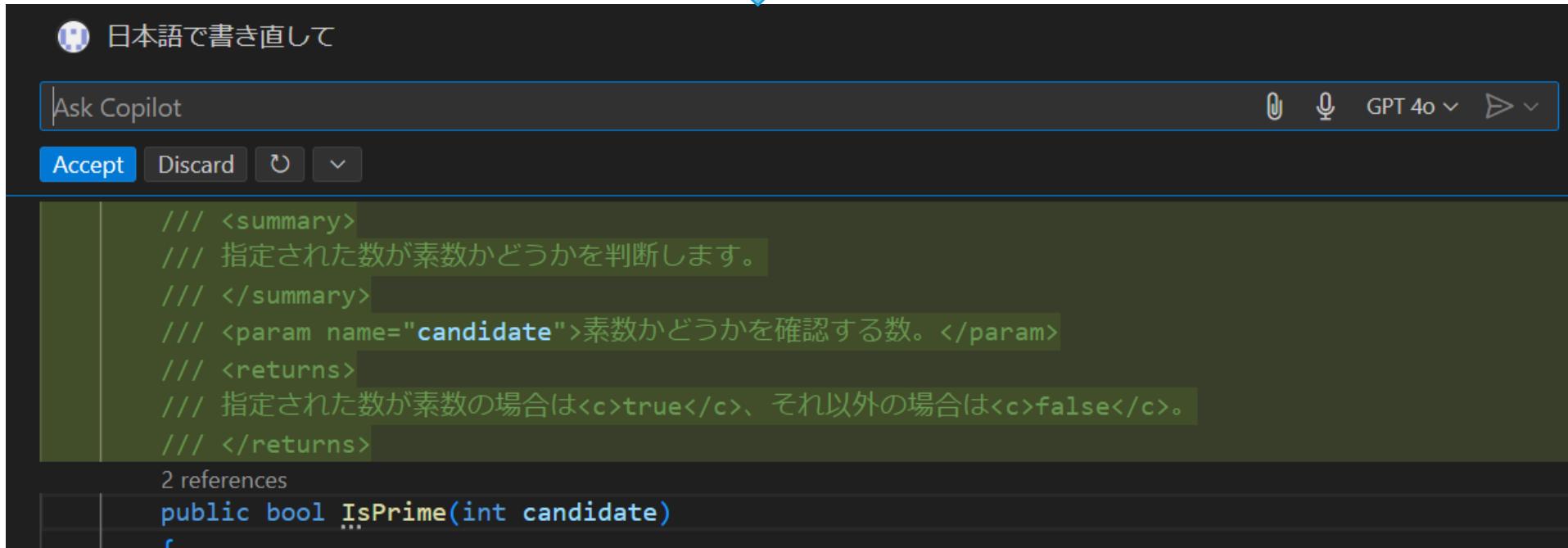
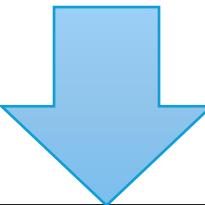
Below the code, it says '2 references'.

■ 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 microphone, 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 below it is a text input field labeled 'Ask Copilot'. The 'Accept' button is highlighted in blue, while 'Discard' and other buttons are in grey. The main code editor area now includes more Japanese documentation and parameters:

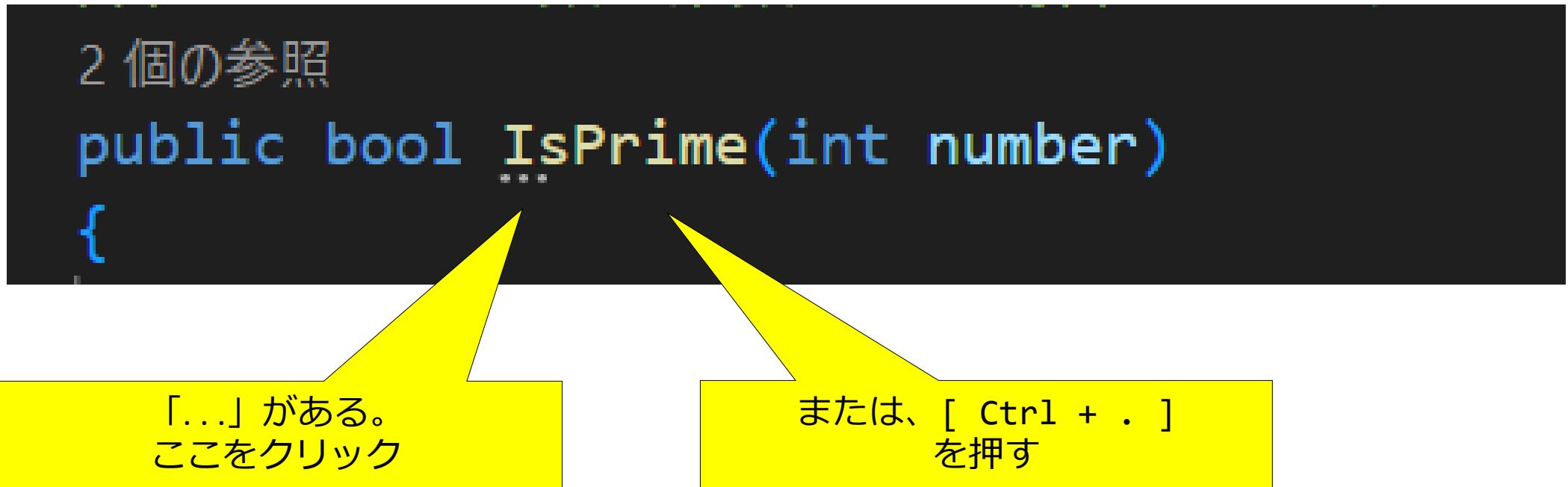
```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

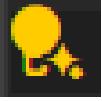
At the bottom of the code editor, there are two small boxes indicating '2 references' and a partial view of the code starting with 'public bool IsPrime(int candidate)'.

開発者のユースケース: ドキュメント作成

- 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

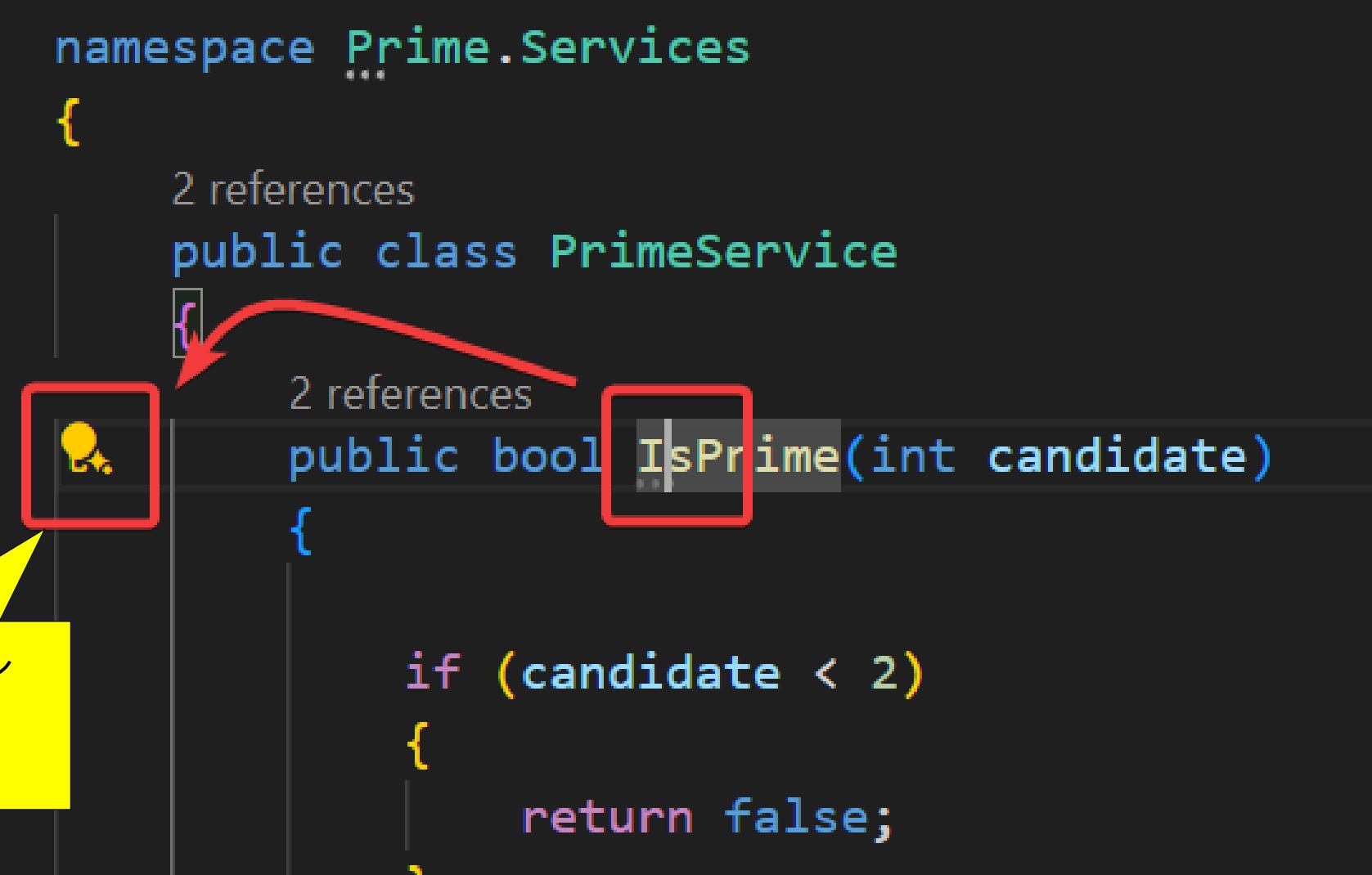
■メソッド名の最初の文字の下に「...」が表示されている



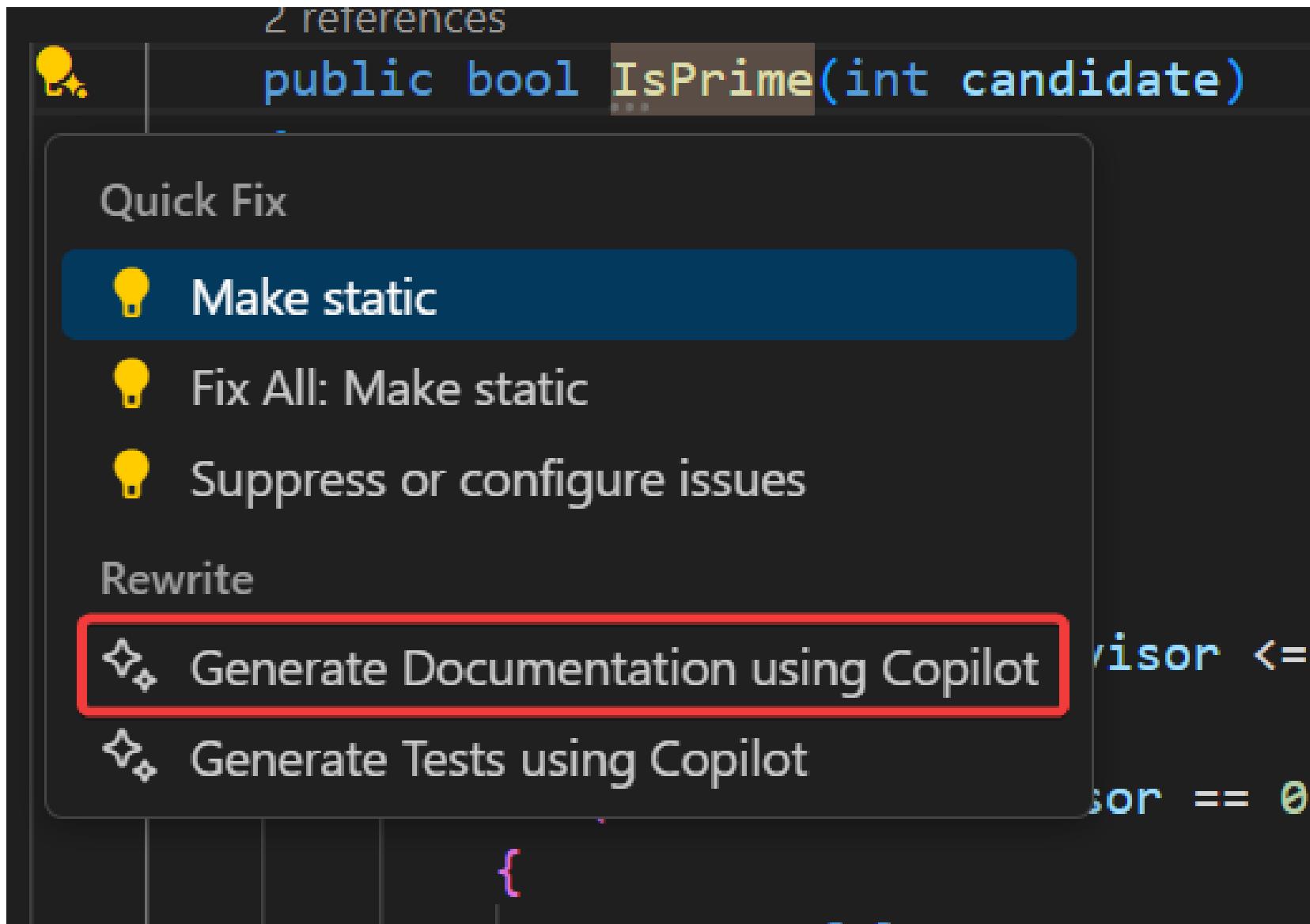
■ 続いて「きらめきアイコン」  をクリック

```
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:

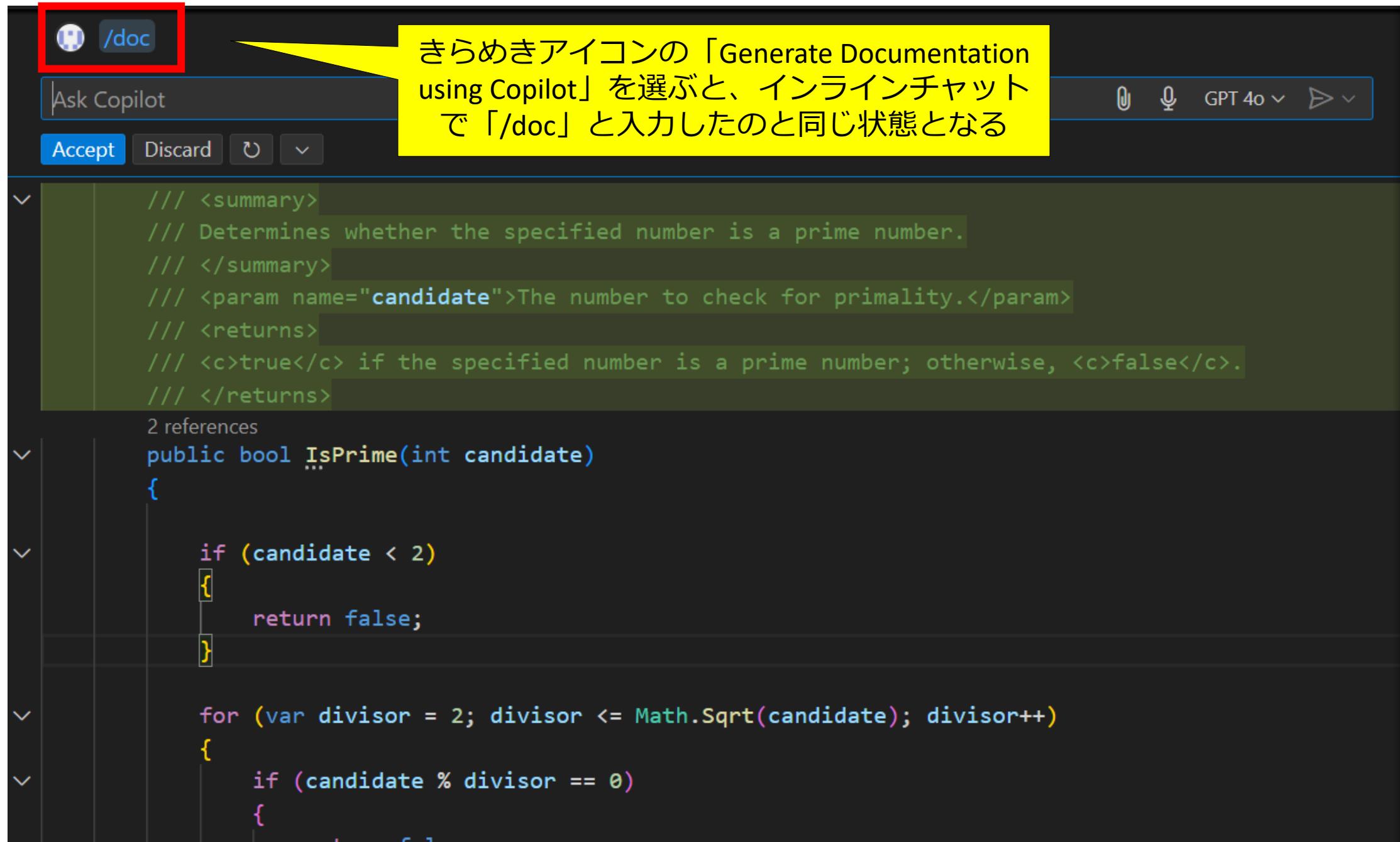
```
if (candidate < 2)
```

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。



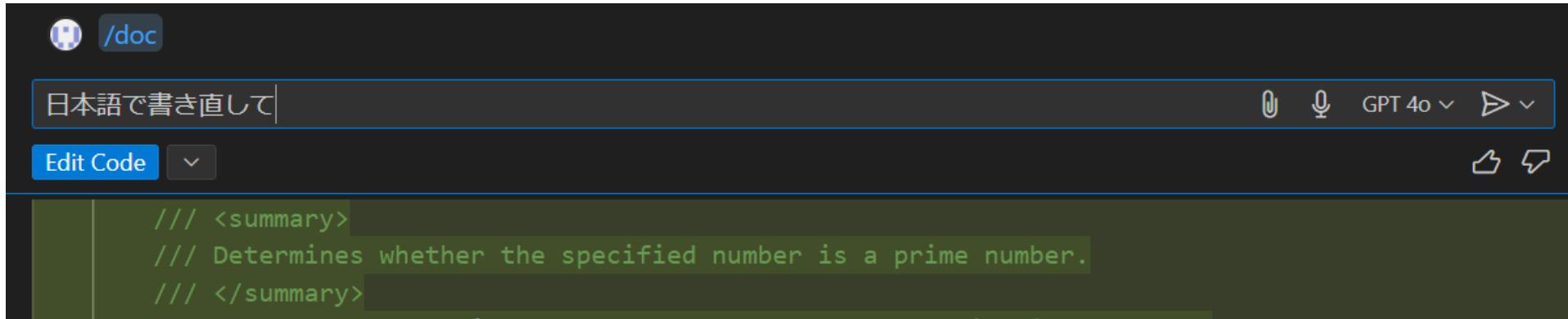
The screenshot shows the Microsoft Copilot interface. At the top left, there is a blue circular icon with a white 'H' and a red button labeled '/doc'. A yellow arrow points from this button to a yellow callout box containing Japanese text. The text in the callout box reads: 'きらめきアイコンの「Generate Documentation 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;
            }
        }

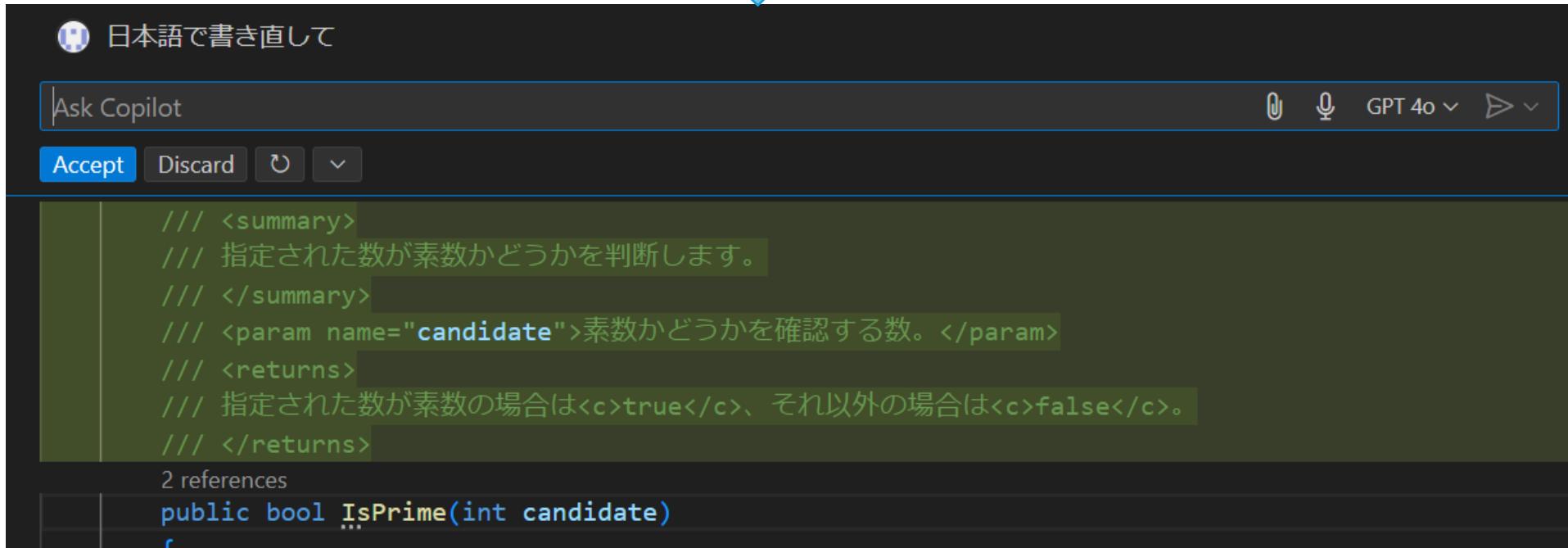
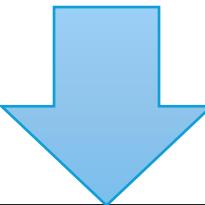
        return true;
    }
```

■ 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 microphone, 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 below it is a text input field labeled 'Ask Copilot'. The 'Accept' button is highlighted in blue, while 'Discard' and other buttons are in grey. The main code editor area now includes more Japanese documentation:

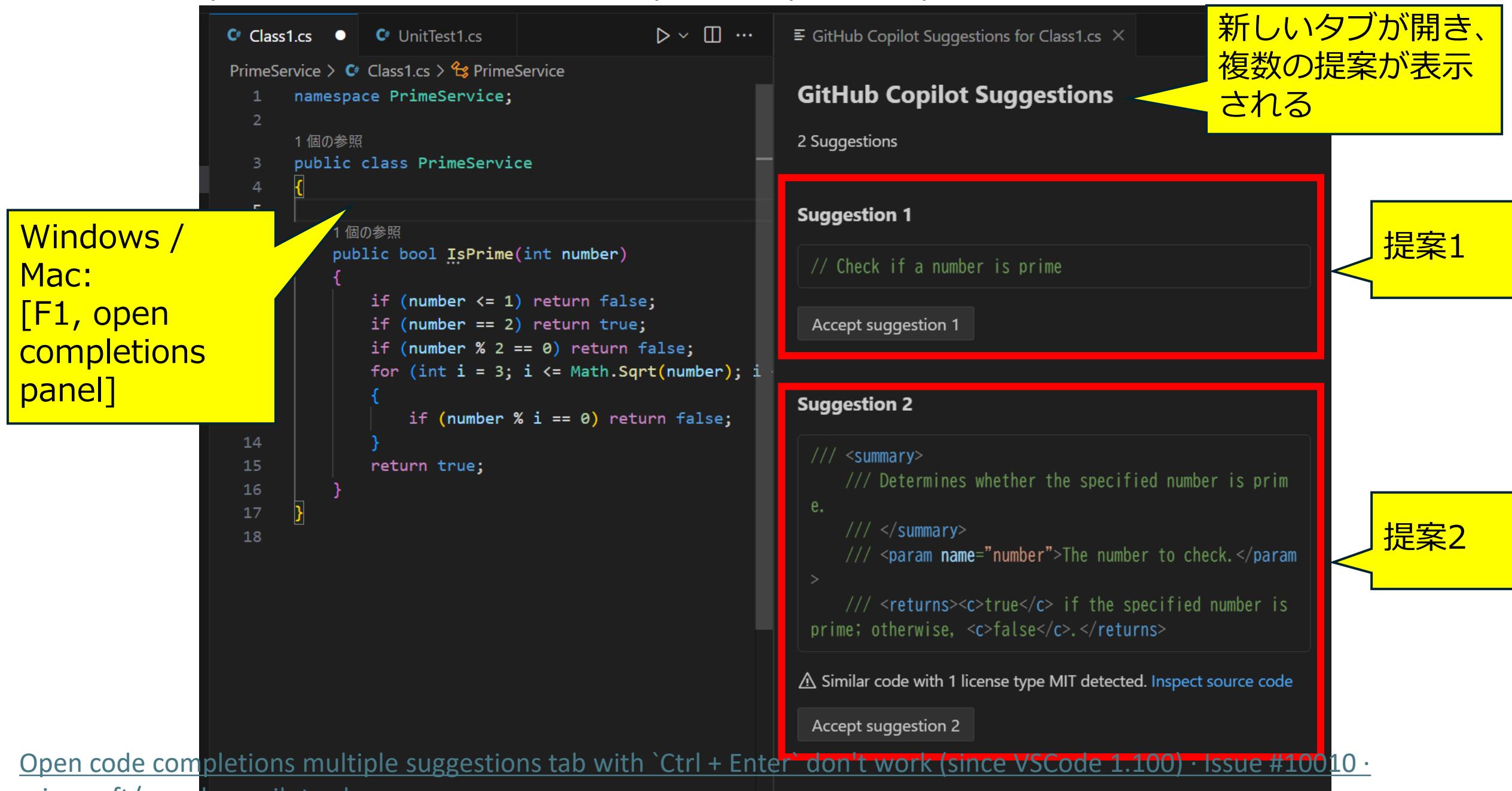
```
/// <summary>
/// 指定された数が素数かどうかを判断します。
/// </summary>
/// <param name="candidate">素数かどうかを確認する数。</param>
/// <returns>
/// 指定された数が素数の場合は<c>true</c>、それ以外の場合は<c>false</c>。
/// </returns>
```

Below the code, there is a note '2 references' and a partial view of the code starting with 'public bool IsPrime(int candidate)'.

開発者のユースケース: ドキュメント作成

- 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)



開発者のユースケース: ドキュメント作成

- ・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

開発者のユースケース: ドキュメント作成

- 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。
 - [F1, open completion panel]で複数の候補を別タブに表示し、そこから一つを選ぶ

Part 2 目次

- モジュール1 開発者のユースケース
 - ドキュメント作成
 - コード作成
- モジュール2 単体テストの開発
- モジュール3 GitHub Copilot Businessの概要
- モジュール4 GitHub Copilot Enterpriseの概要
- モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- モジュール6 PythonでGitHub Copilotを使用する（オプション）

開発者のユースケース: コードの作成

- 方法(1) **コード提案**[Alt + \](windows) [Option + \](Mac) 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](windows) [Command + I](Mac)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](windows) [Shift + Command + I](Mac)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;
}
```

開発者のユースケース: コードの作成

- ・方法(1) **コード提案[Alt + \](windows) [Option + \](Mac)** 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](windows) [Command + I](Mac)**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](windows) [Shift + Command + I](Mac)**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

A screenshot of a code editor window. The top status bar displays "2 references". The main code area shows the beginning of a class definition:

```
public class PrimeService {
```

Below the code area is a toolbar with the following icons from left to right: a magnifying glass, a question mark, "GPT 4o", and a right-pointing arrow. The "GPT 4o" icon is highlighted with a blue border.

■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:

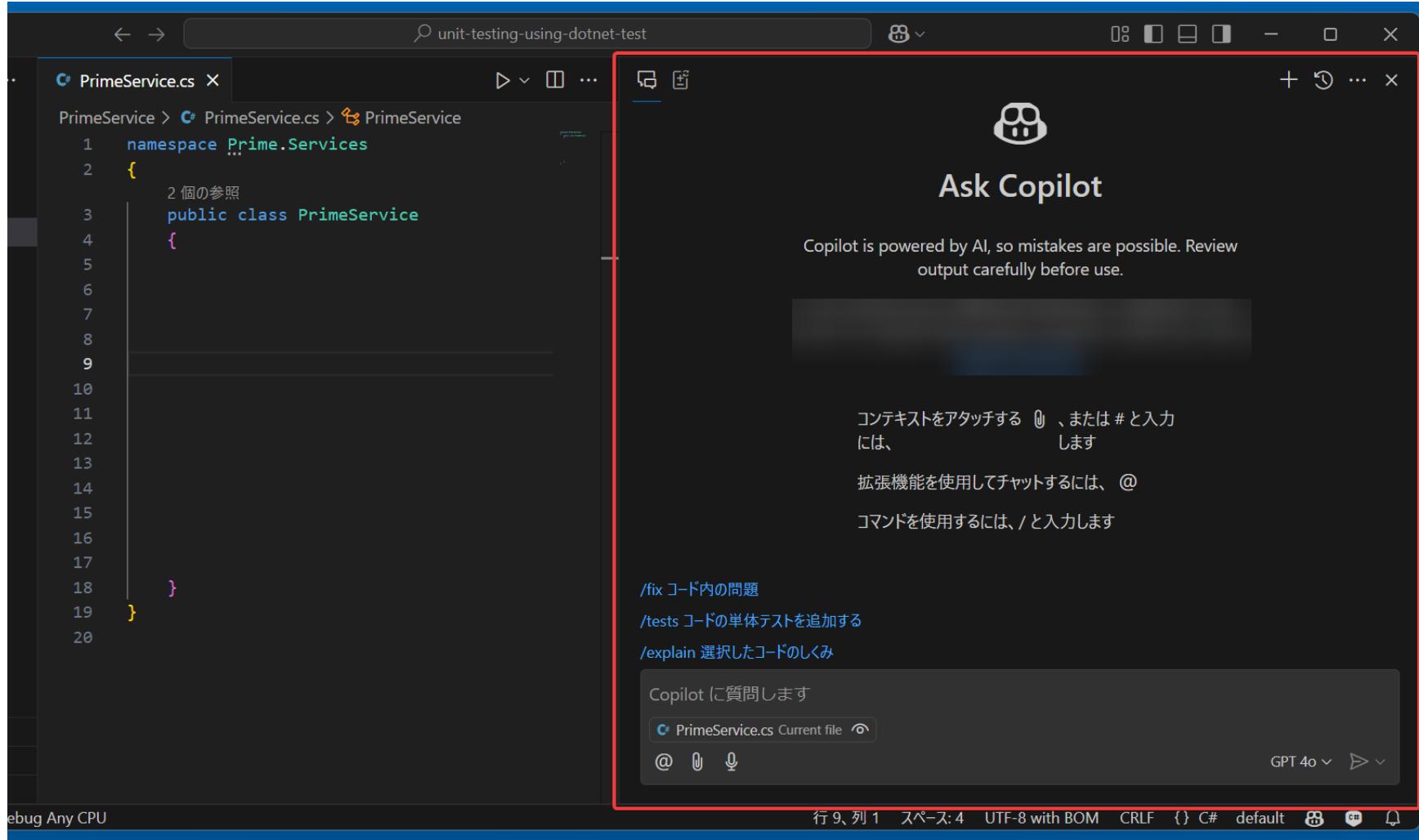
- Input:** A task prompt asking to write a method to determine if a number is prime, including a summary and parameter documentation.
- Output:** Generated C# code for a `IsPrime` method. The code includes:
 - Method signature: `public bool IsPrime(int number)`
 - Initial checks for numbers less than 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)".

開発者のユースケース: コードの作成

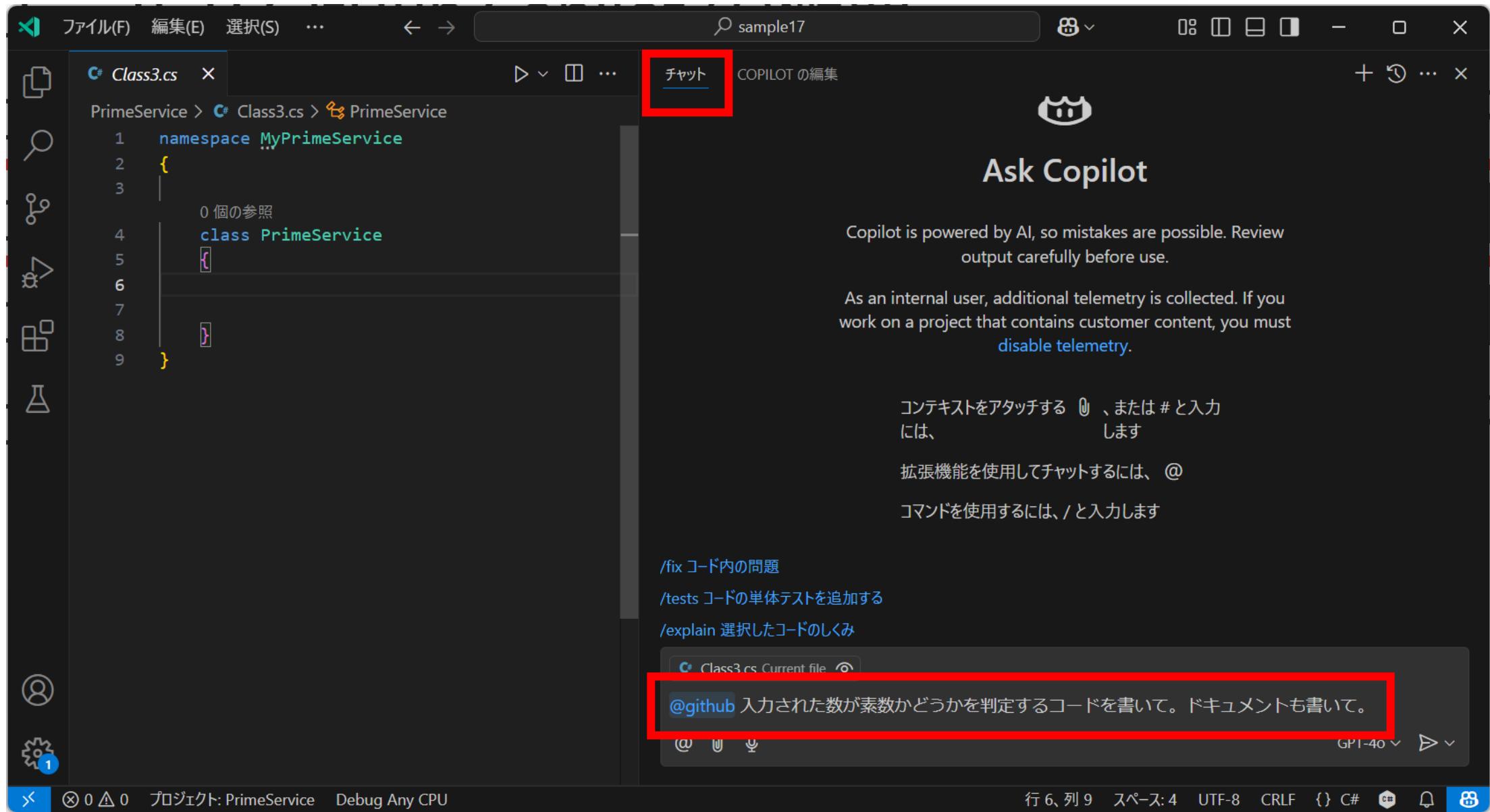
- 方法(1) **コード提案**[Alt + \](windows) [Option + \](Mac) を利用。ドキュメントを書き、コード提案により、ドキュメントに対応したコードを生成する。
- 方法(2) **インラインチャット**[Ctrl + I](windows) [Command + I](Mac)を利用。開発したいコードの仕様などを指定して、コードを生成する
- 方法(3) **チャットビュー**[Ctrl + Alt + I](windows) [Shift + Command + I](Mac)を利用。チャットで、開発したいコードの仕様などを入力して、コードを生成する
- 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 the Visual Studio IDE with the title bar "unit-testing-using-dotnet-test". A yellow callout box points from the text in the yellow box below to the GitHub Copilot interface.

C# PrimeService.cs X

PrimeService > C# PrimeService.cs > PrimeService

```
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
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>
    }
}
```

Copilot に質問します

PrimeService.cs Current file

@ ⌘ ⌘

GPT 4o

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

Any CPU

※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 code editor on the left contains a C# file named PrimeService.cs:

```
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 Chat View on the right displays a conversation with 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。

C# 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">検査対象の整数</param>
 /// <returns>数が素数である場合は true、それ以外の場合は false。</returns>
 }
}

「エディターで適用」ボタンをクリック

テ스트ケースも教えてください。

Copilot に質問します

C# PrimeService.cs Current file

GPT 4o

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

A red box highlights the 'Apply in Editor' button (the 'E' icon) in the Copilot preview bar. A yellow callout box points to it with the text '「エディターで適用」ボタンをクリック'.

■ 「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;
            }

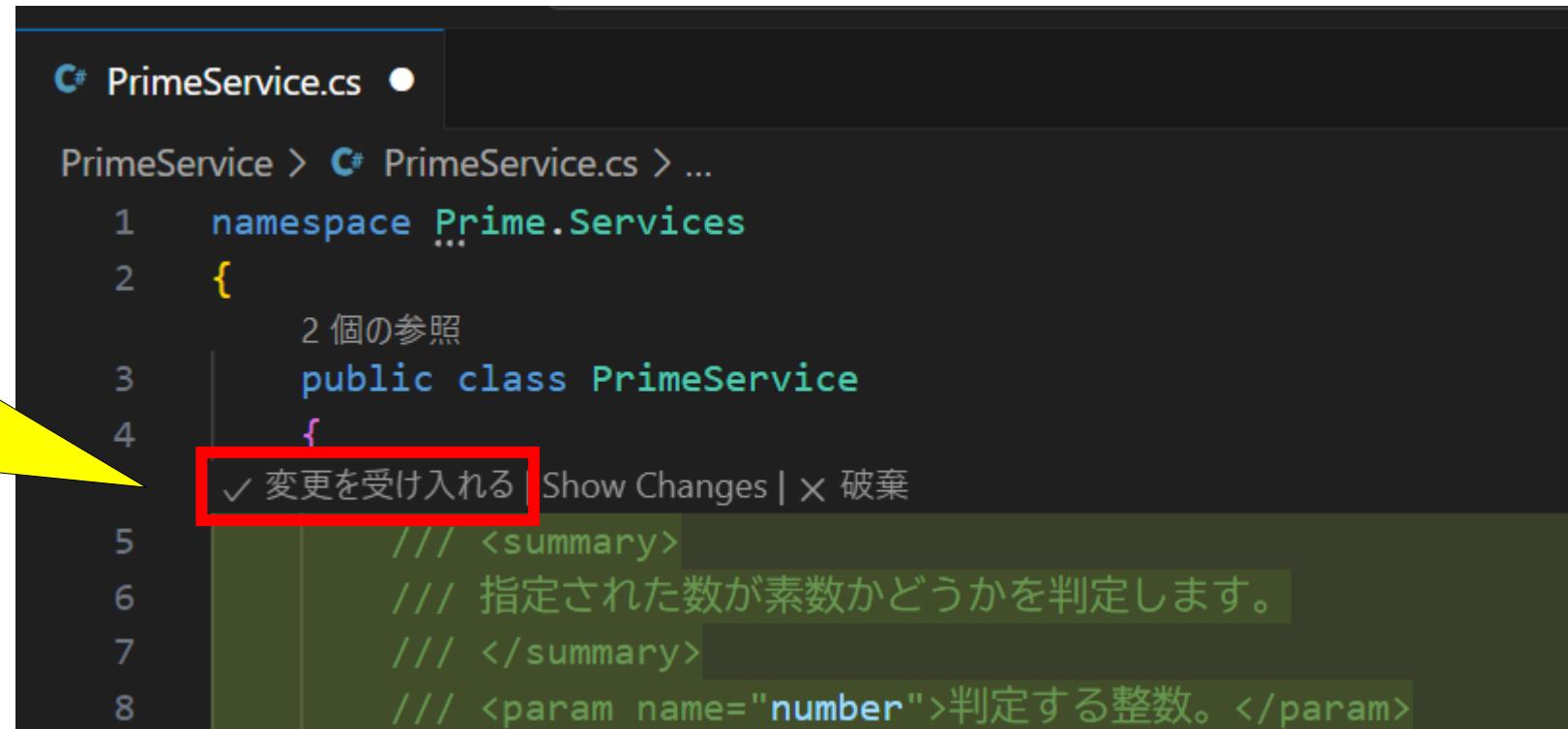
            return true;
        }
    }
}
```

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 shows: "Copilotに質問します" (Ask Copilot) and "PrimeService.cs Current file". 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。

The Chat View on the right shows the following interaction:

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。

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;
            }
        }
    }
}
```

テストケースも教えてください。

Copilot に質問します

PrimeService.cs Current file

GPT 4o

開発者のユースケース: コードの作成

- 方法(1) **コード提案[Alt + \](windows) [Option + \](Mac)** 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](windows) [Command + I](Mac)** 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](windows) [Shift + Command + I](Mac)** 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

開発者のユースケース: コードの作成

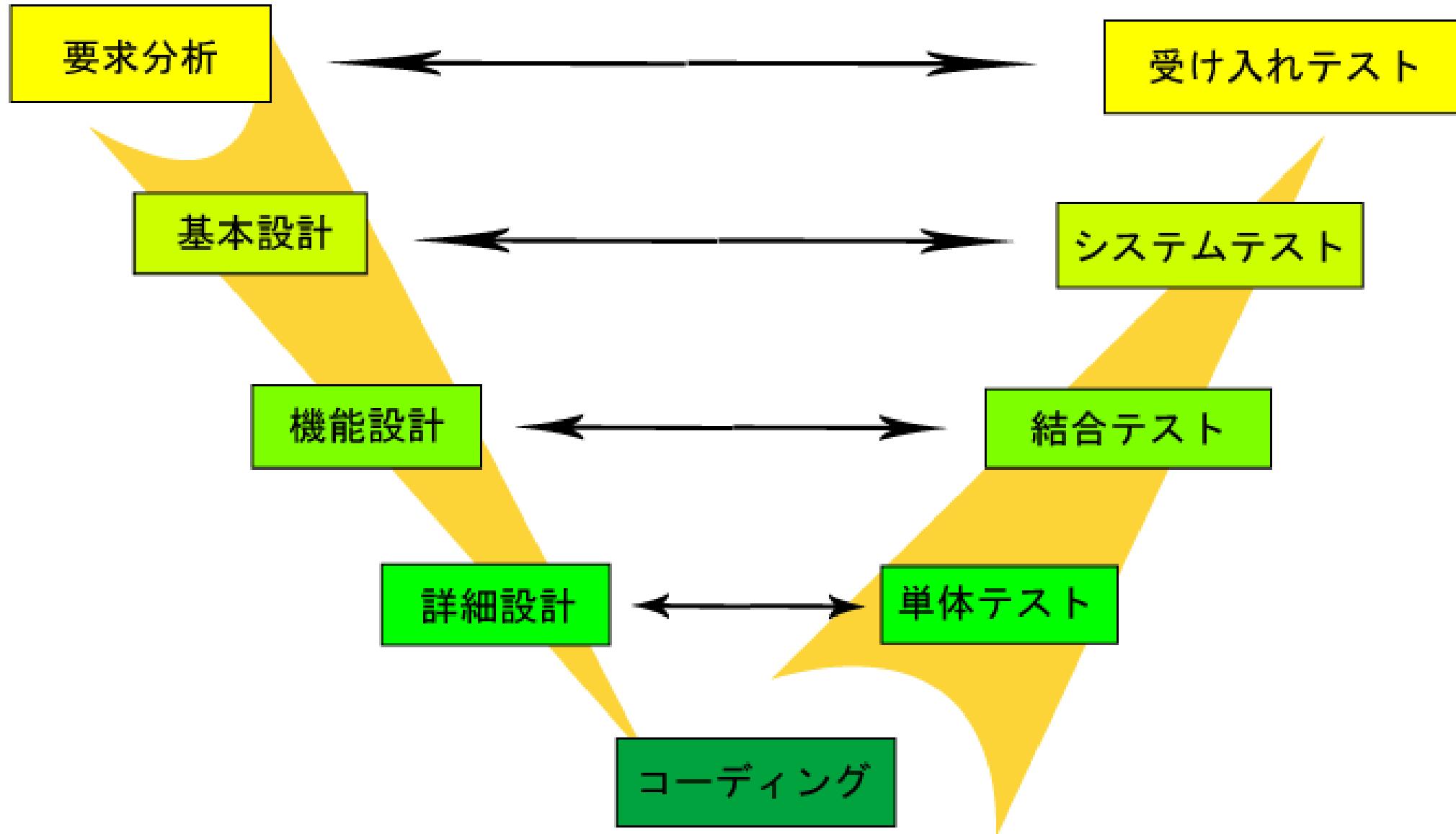
- ・コードの**提案**や**チャット**を使って、C#などのプログラムコードをすばやく正確に記述することができる。
- ・方法(1) **コード提案**を利用して、ドキュメントに対応したコードを生成する
- ・方法(2) **インラインチャット**を起動して、コードの仕様などをプロンプトで指定して、コードを生成する
- ・方法(3) **チャットビュー**を起動して、コードの仕様などをプロンプトで指定して、コードを生成する

Part 2 目次

- ・モジュール1 開発者のユースケース
 - ・ドキュメント作成
 - ・コード作成
- ・モジュール2 単体テストの開発
- ・モジュール3 GitHub Copilot Businessの概要
- ・モジュール4 GitHub Copilot Enterpriseの概要
- ・モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- ・モジュール6 PythonでGitHub Copilotを使用する（オプション）

モジュール2 単体テストの開発

- 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)

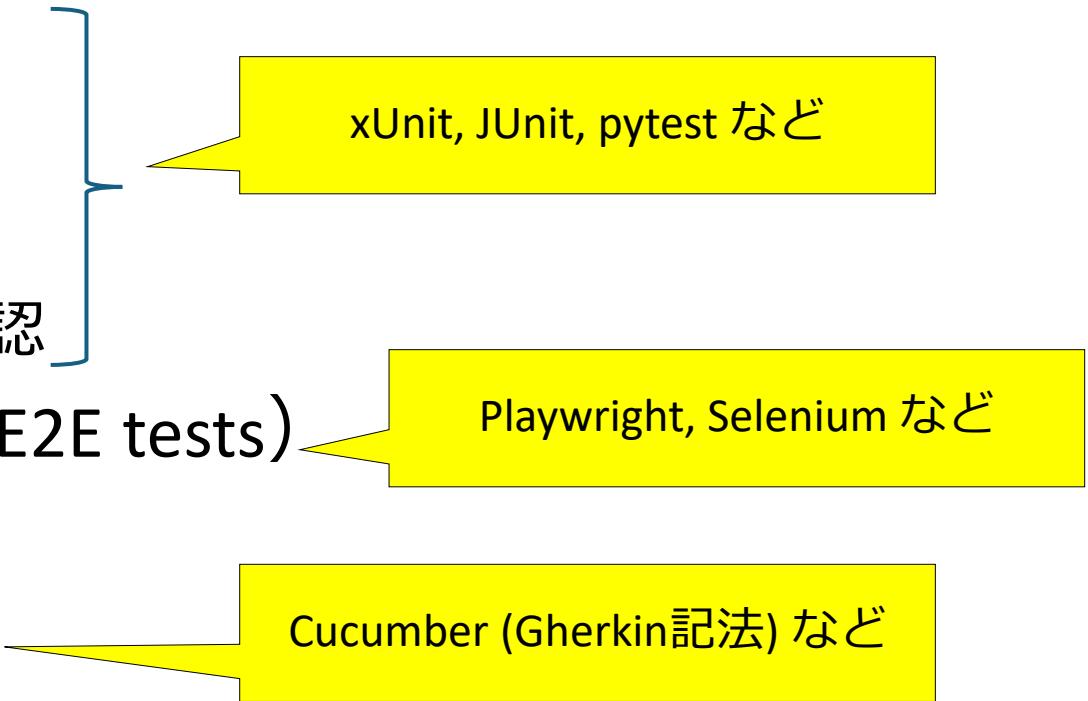
- システム全体の動作確認

- **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2 単体テストの開発

- ・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2 単体テストの開発

- ・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2 単体テストの開発

- ・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

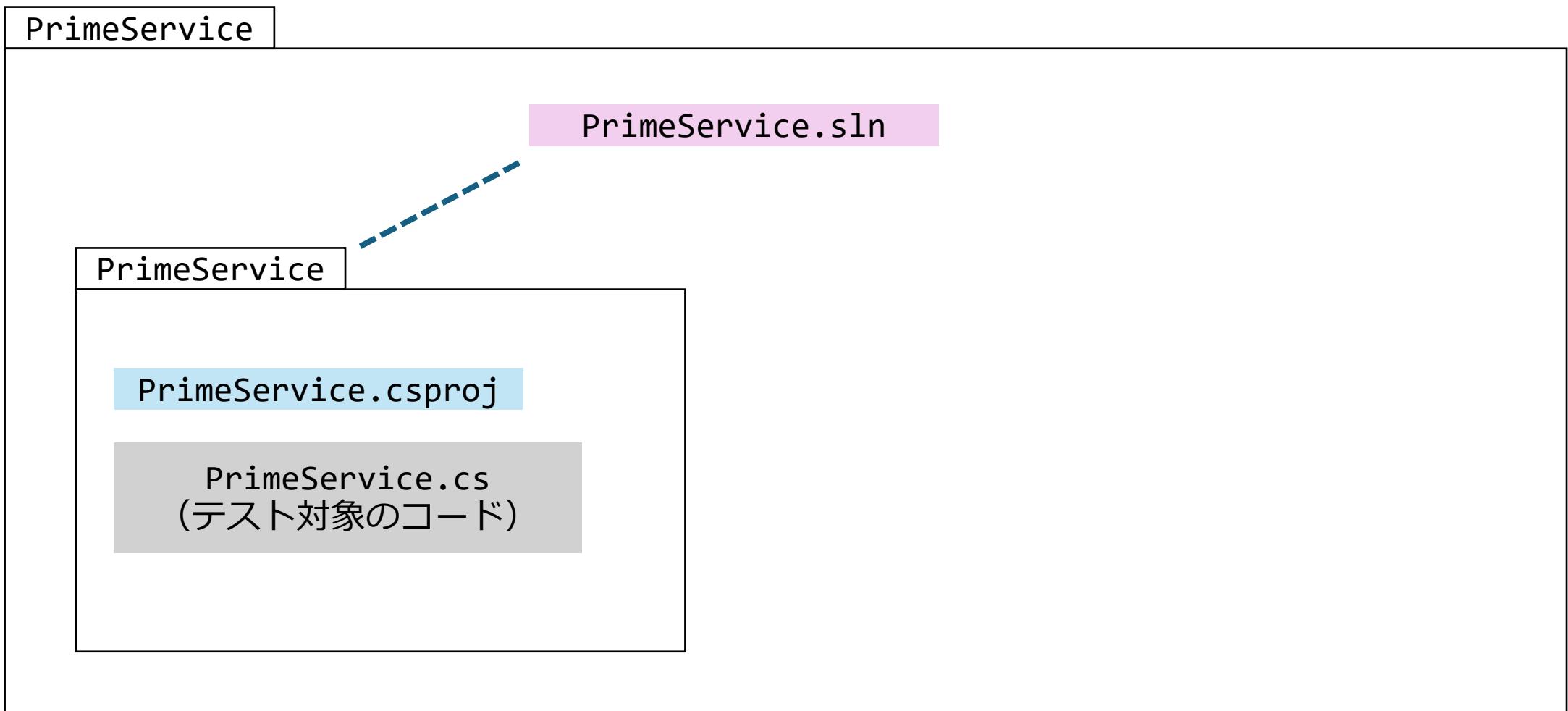
■ソリューションの作成

PrimeService

PrimeService.sln

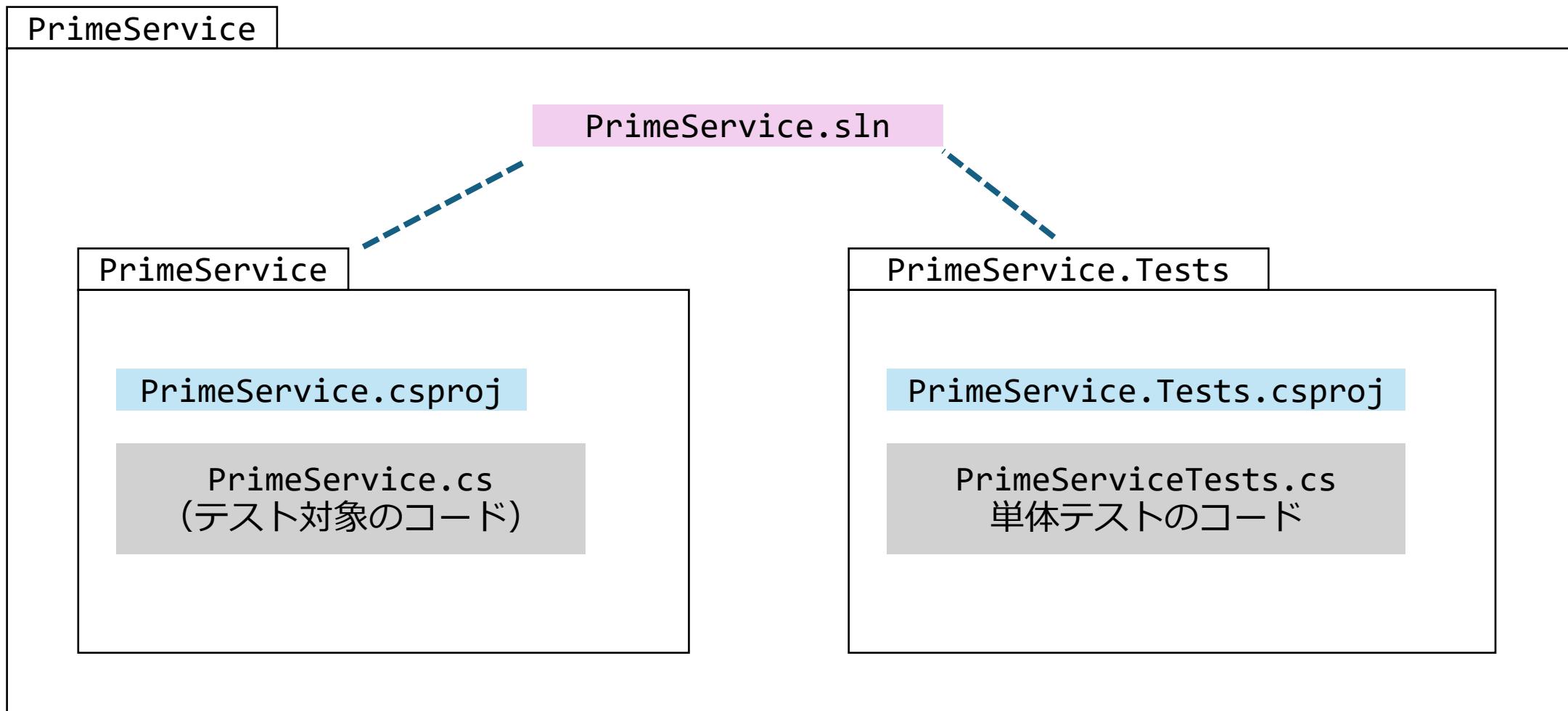
```
mkdir PrimeService  
cd PrimeService  
dotnet new sln -n PrimeService
```

■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



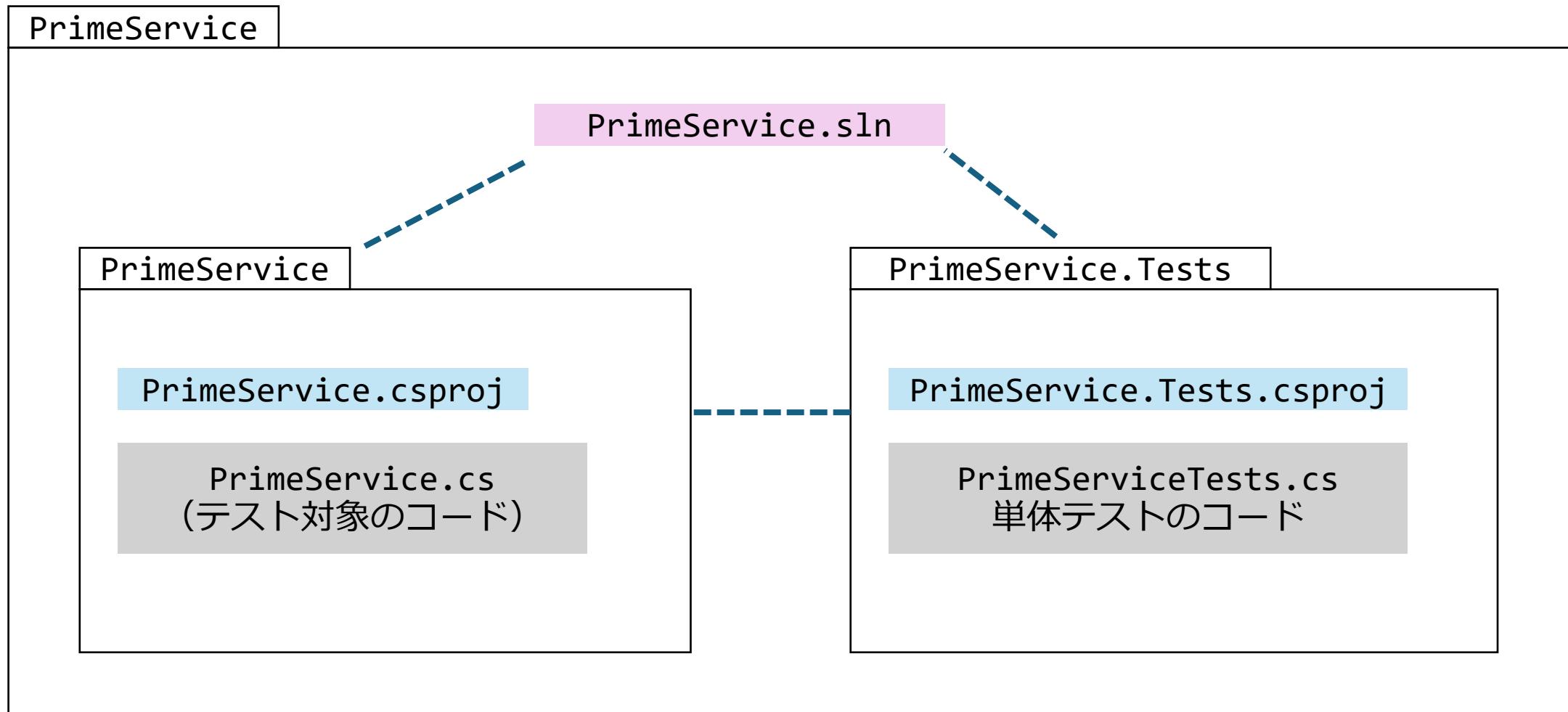
```
dotnet new classlib -n PrimeService  
dotnet sln add PrimeService
```

■xUnitテストプロジェクトの作成、ソリューションへの追加



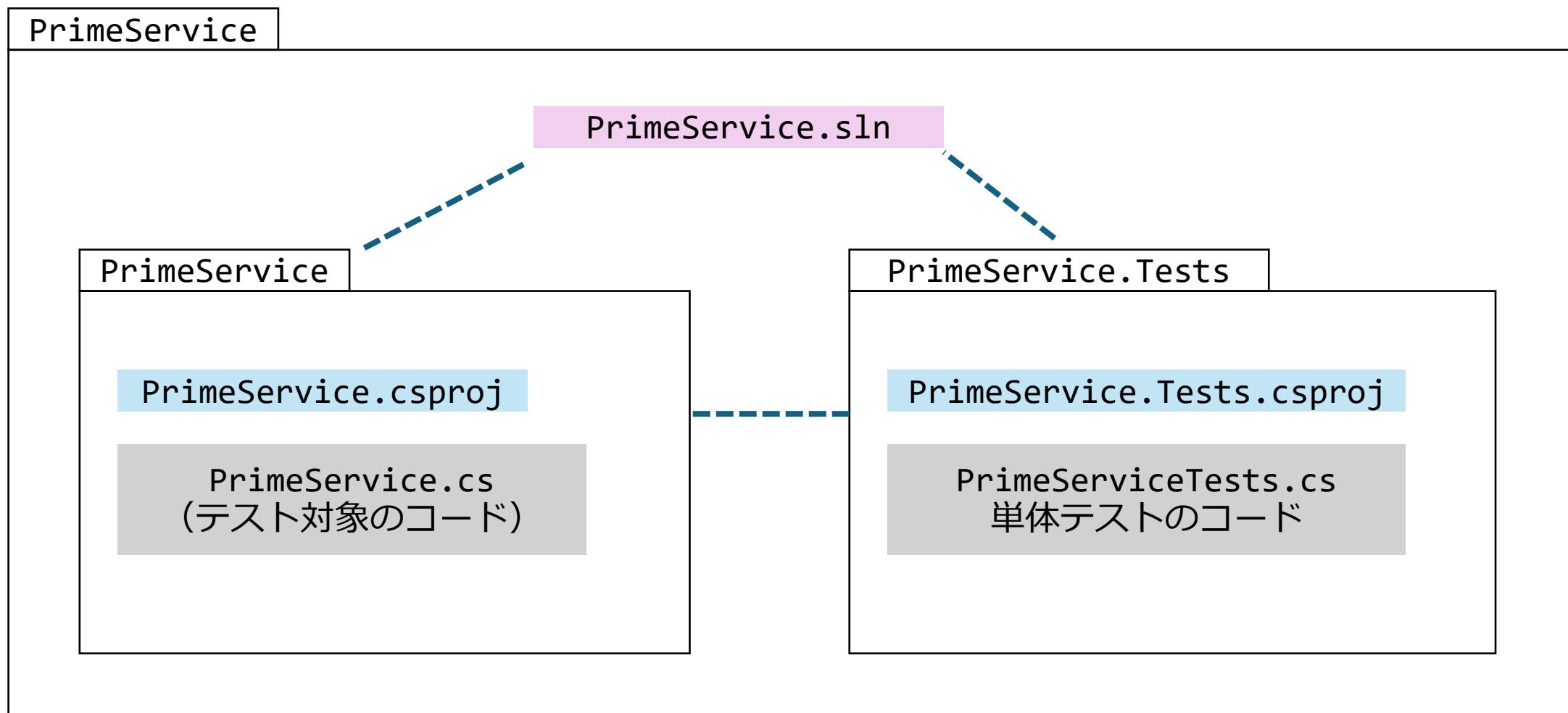
```
dotnet new xunit -n PrimeService.Tests  
dotnet sln add PrimeService.Tests
```

■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2 単体テストの開発

- ・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2 単体テストの開発

- ・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");
}
```

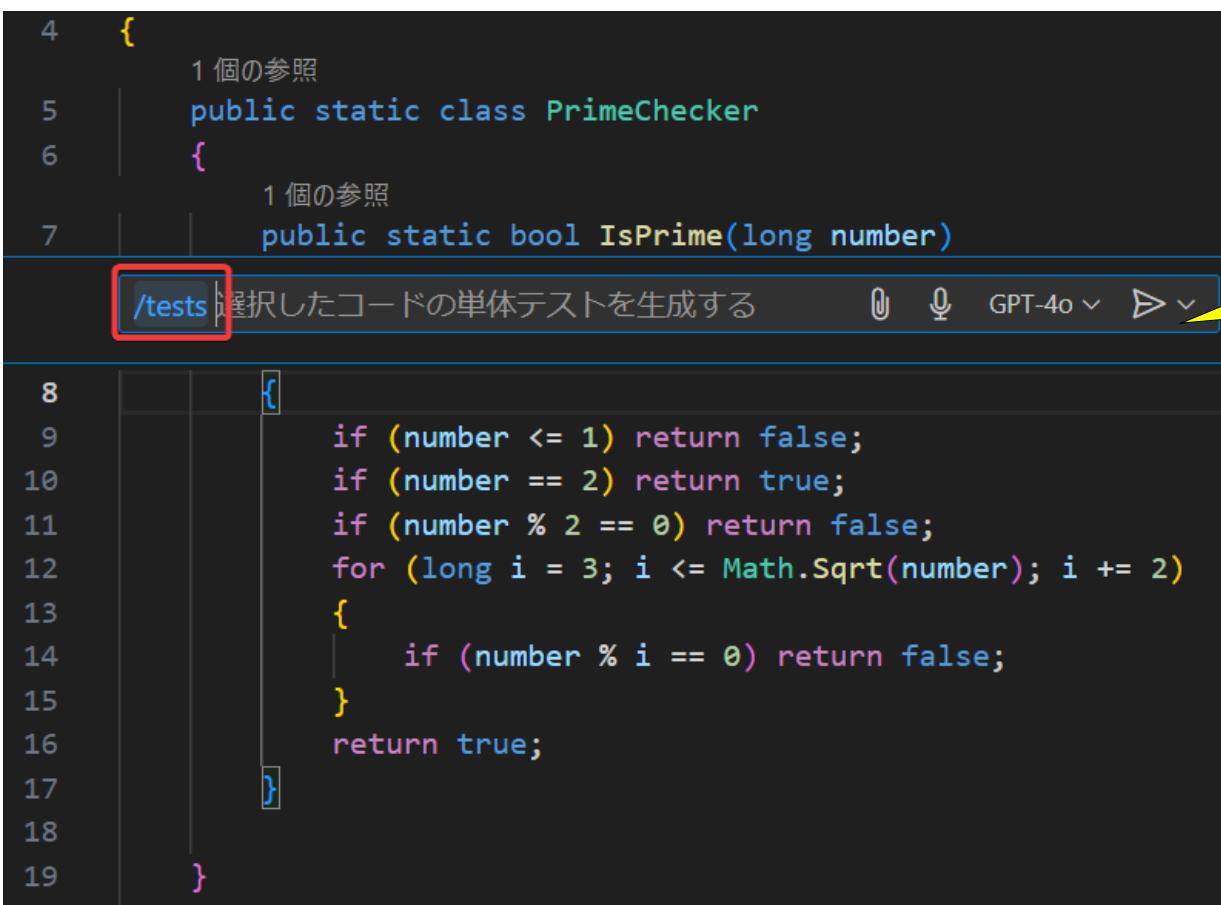
モジュール2 単体テストの開発

- ・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
 - チャットビュー(Ctrl + Alt + I)でモードを「Ask」から「Edit」に切り替え、「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



A screenshot of an IDE showing a code editor with C# code. The code defines a static class PrimeChecker with a static method IsPrime. The status bar at the bottom shows the text '/tests [選択したコードの単体テストを生成する]'. A red box highlights the '/tests' part of the text. The code itself is as follows:

```
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     }
```

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 }
```

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
 - チャットビュー(Ctrl + Alt + I)でモードを「Ask」から「Edit」に切り替え、「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

The screenshot shows the Visual Studio Code interface with the Chat view open. A yellow box highlights the command `/tests` in the Chat input field. A red arrow points from this box to a tooltip at the bottom right of the Chat view, which contains the text `@workspace /tests 選択したコードの単体テストを生成する`.

Ctrl + Alt + I でチャットビューを表示し、/tests と入力。
(自動的に @workspace /tests に展開される)

1 個の参照

```
public static bool IsPrime(long number)
{
    if (number <= 1) return false;
    if (number == 2) return true;
    if (number % 2 == 0) return false;
    for (long i = 3; i <= Math.Sqrt(number); i += 2)
    {
        if (number % i == 0) return false;
    }
    return true;
}
```

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、します

/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

C# Class1.cs Current file

@workspace /tests 選択したコードの単体テストを生成する

GPT-4o ▶

■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:

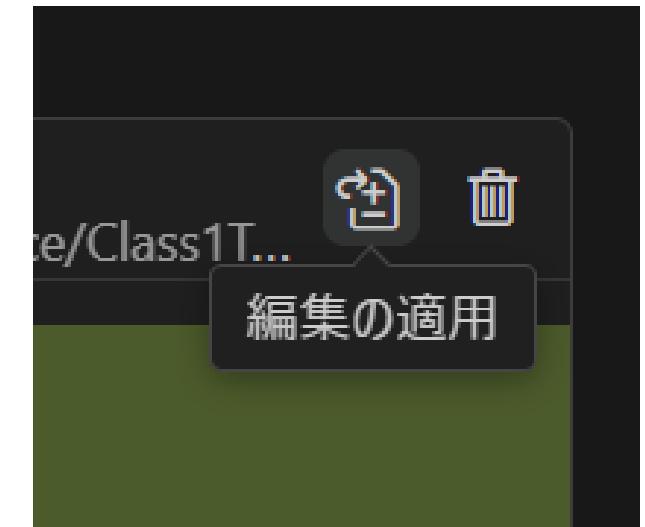
1. Create a new file `PrimeService/Class1Test.cs`.
2. Use a testing framework like xUnit.
3. Write test methods to cover various cases for the `IsPrime` method:
 - Test for numbers less than or equal to 1.
 - Test for the number 2.
 - Test for even numbers greater than 2.
 - Test for prime numbers.
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

The screenshot shows the Visual Studio IDE interface. On the left, there is a code editor window for 'Class1.cs' which contains C# code for a PrimeService. On the right, there is another code editor window for 'PrimeService.cs' which contains NUnit test code for 'PrimeCheckerTests'. A red box highlights a modal dialog titled 'Copilot に質問します' (Ask Copilot a question) with a blue button labeled '同意する' (Agree). The status bar at the bottom indicates '行 57 列 2 フォーマット: UTF-8 クリップボード: 1 C#'.

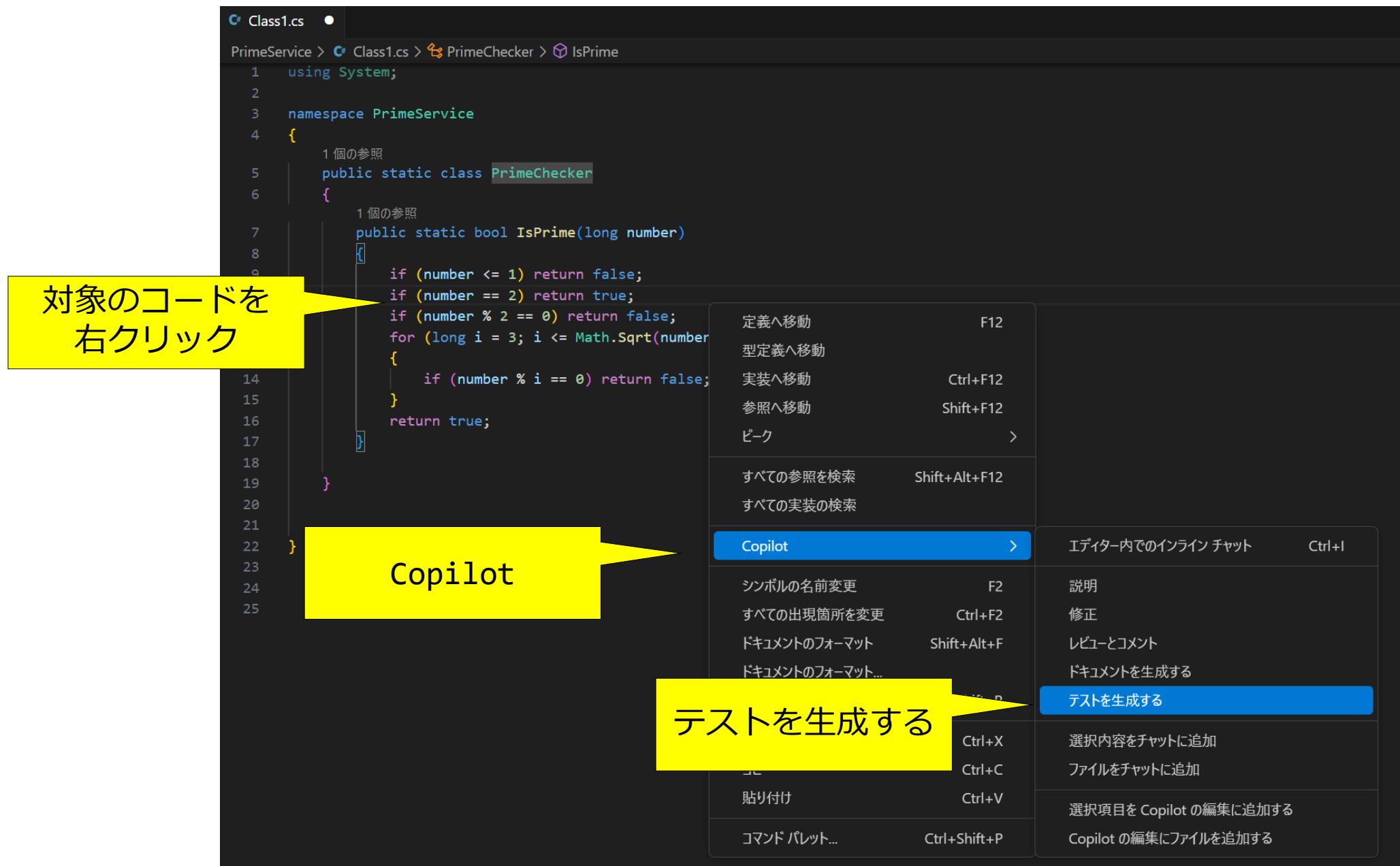
```
C# Class1.cs
C# Class1.cs > PrimeService > PrimeChecker > IsPrime
1  using System;
2
3  namespace PrimeService
4  {
5      1 個の参照
6      public static class PrimeChecker
7      {
8          1 個の参照
9          public static bool IsPrime(long number)
10         {
11             if (number <= 1) return 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 }

C# using NUnit.Framework; /c:/Users/hiryamada/github/sample17/PrimeService/PrimeService.cs
1  using NUnit.Framework;
2
3  /tests
4
5  Copilot に質問します
6  同意する 閉じる
7
8  namespace PrimeService.Tests
9  [
10  ]
11  public class PrimeCheckerTests
12  [
13  ]
14  public void IsPrime_InputIs1_ReturnsFalse()
15  [
16  ]
17  public void IsPrime_InputIs2_ReturnsTrue()
18  [
19
20 ]
```

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
 - チャットビュー(Ctrl + Alt + I)でモードを「Ask」から「Edit」に切り替え、「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 code editor interface with two tabs: 'Class1.cs' and 'PrimeService'. The 'Class1.cs' tab contains C# code for a PrimeChecker class. The 'PrimeService' tab shows generated NUnit test code for the IsPrime method.

```
C# using NUnit.Framework; /c:/Users/hiryamada/github/sample17/PrimeService/PrimeService.Tests.cs
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 }
```

The right pane shows the generated test code:

```
1 using NUnit.Framework;
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 }
```

A red box highlights the '同意する' (Agree) button in a Copilot modal window.

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
 - チャットビュー(Ctrl + Alt + I)でモードを「Ask」から「Edit」に切り替え、「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 extension open. A yellow callout box points from the top right to the 'Chat' tab in the top bar, with the Japanese text 'チャットビューを開く (Ctrl + Alt + I)' (Open Chat View) overlaid. Another yellow callout box points from the bottom left to the input field in the Copilot view, with the Japanese text '@github generate tests と入力する' (Input @github generate tests) overlaid. The code editor shows a C# file named 'Class1.cs' with a method 'IsPrime'.

```
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             i += 2;
15
16             return true;
17         }
18     }
```

チャットビューを開く
(Ctrl + Alt + I)

@github generate tests
と入力する

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

GPT-4o ▶

Class1.cs Current file

@ @ @

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 "Copilot" 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

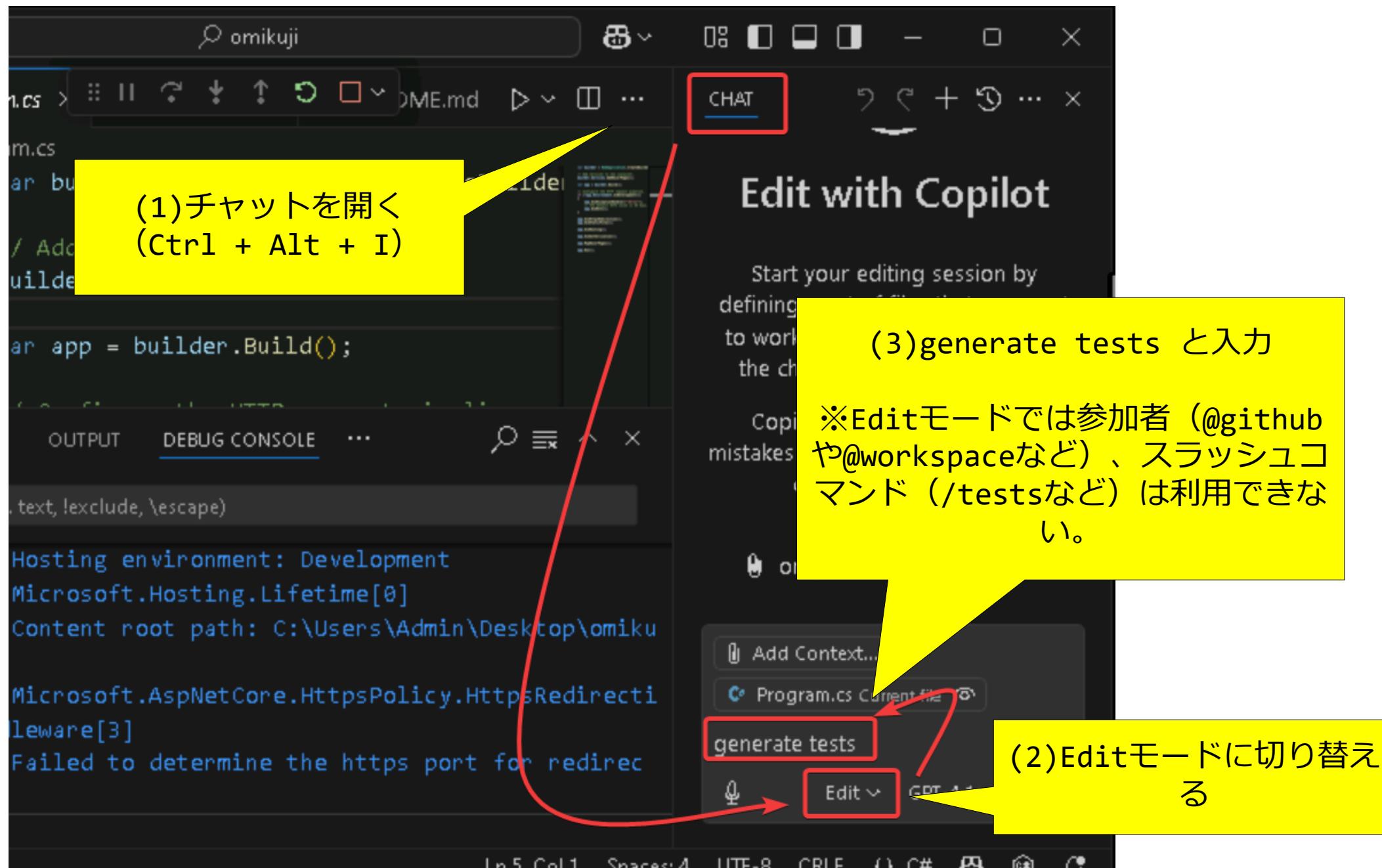
と入力した場合に使用できるボタン



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
 - チャットビュー(Ctrl + Alt + I)でモードを「Ask」から「Edit」に切り替え、「generate tests」のように指示

■ Copilot Edits (Editモード) で、generate tests を利用する



■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。

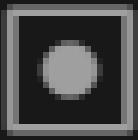
The screenshot shows the Visual Studio IDE interface. On the left, the Solution Explorer displays a project named 'SAMPLE17' with files like 'Class1.cs', 'PrimeService.csproj', and 'PrimeService.Tests'. A yellow callout box points to the 'PrimeCheckerTests.cs' file in the 'PrimeService.Tests' folder, which is highlighted with a red border. The code editor window shows the generated test 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     }
20     public void IsPrimeTest(long number, bool expected)
21     {
22         bool result = PrimeChecker.IsPrime(number);
23         Assert.Equal(expected, result);
24     }
}
```

A yellow callout box points to the generated test method 'IsPrimeTest' in line 20, with the text '生成されたテストコード' (Generated test code). In the top right corner, the GitHub Copilot interface is visible, showing the prompt 'Add a new method IsPrimeTest to test the IsPrime method.' and the resulting changes in 'Class1.cs'.

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



←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。

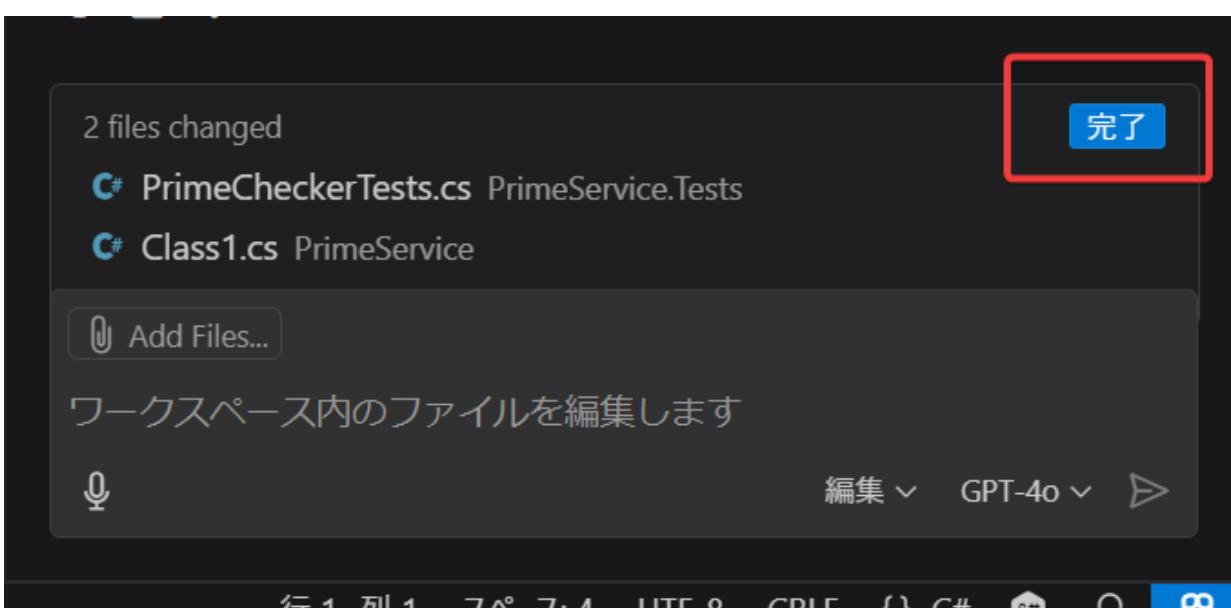
The screenshot shows the Visual Studio interface with the following details:

- Solution Explorer:** Shows a project named "SAMPLE17" with files like Class1.cs, Class2.cs, Class3.cs, Employee.cs, PrimeService.csproj, and PrimeService.Tests.csproj.
- Editor:** Displays the content of PrimeCheckerTests.cs. The code includes XUnit test cases for checking prime numbers.
- Status Bar:** At the bottom right, there is a toolbar with buttons for "保持" (Keep) and "元に戻す" (Undo). The "Keep" button has a red border, indicating it's active or highlighted.
- Code Content:**

```
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2 単体テストの開発

- ・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

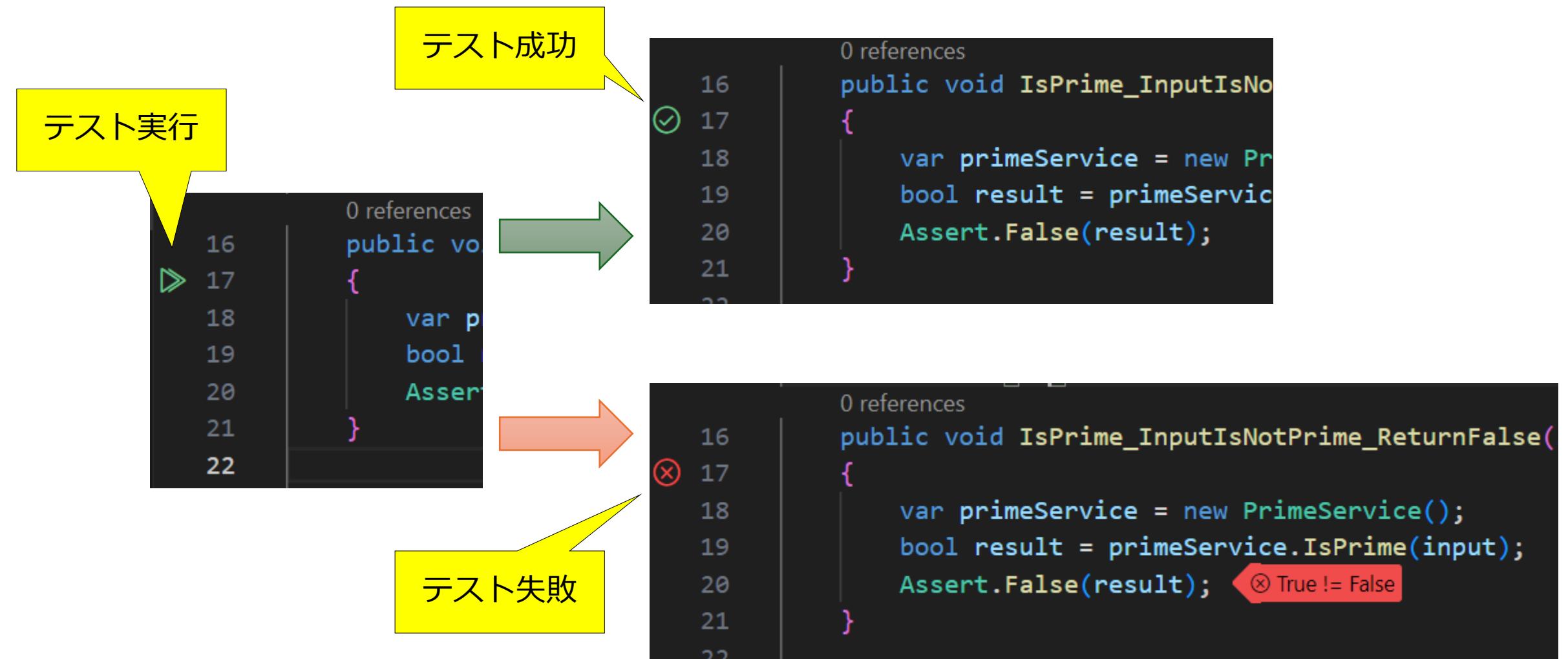
PrimeService.Tests テスト 成功しました (1.1 秒)

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

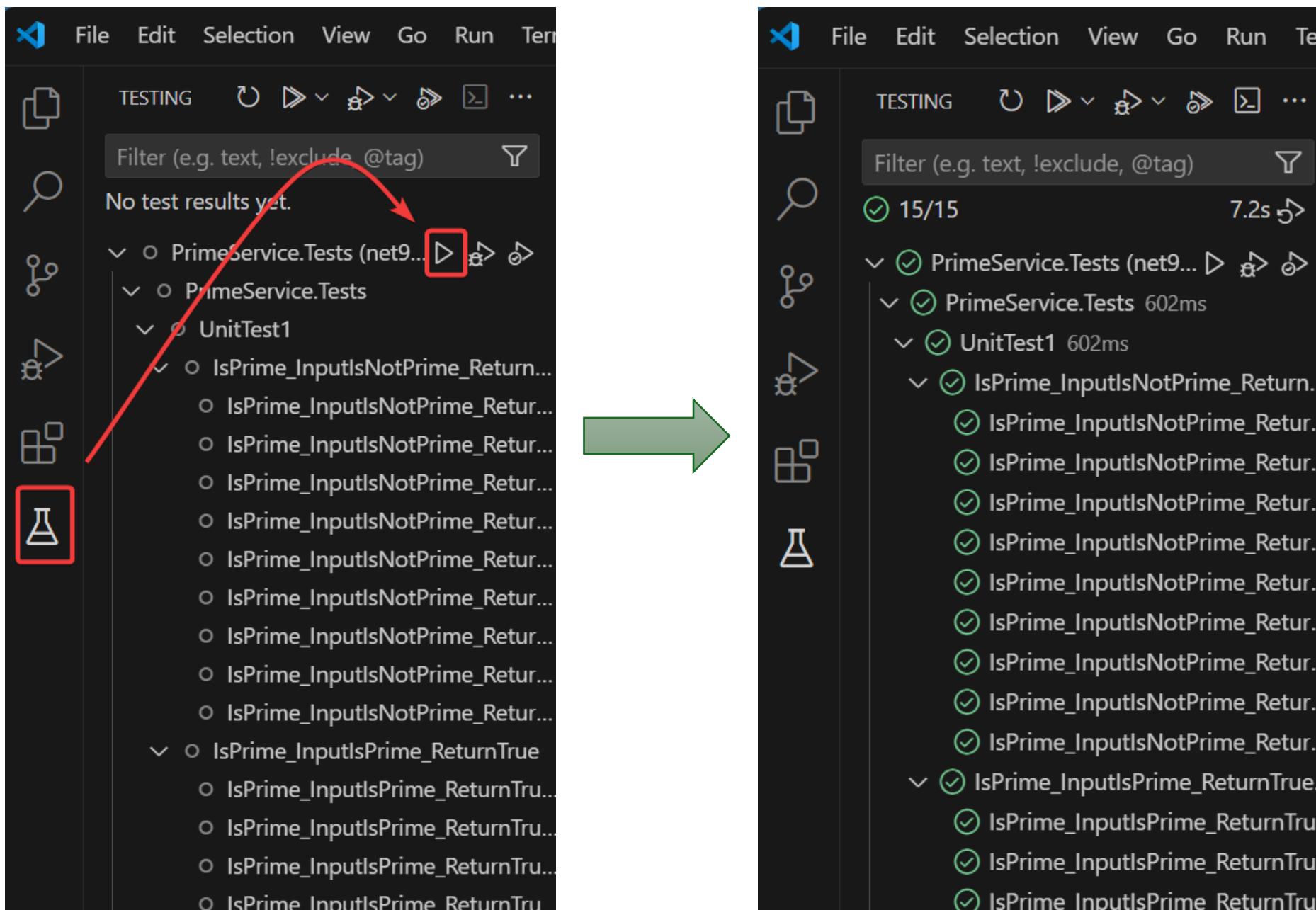
2.6 秒後に 成功しました をビルド

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2 単体テストの開発

- 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。

Part 2 目次

- ・モジュール1 開発者のユースケース
 - ・ドキュメント作成
 - ・コード作成
- ・モジュール2 単体テストの開発
- ・モジュール3 GitHub Copilot Businessの概要
- ・モジュール4 GitHub Copilot Enterpriseの概要
- ・モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- ・モジュール6 PythonでGitHub Copilotを使用する（オプション）

■価格レベルによる機能の違い

	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Organization-wide policy management	✗	✗	✓	✓
Audit logs	✗	✗	✓	✓
Increased GitHub Models rate limits ^[4]	✗	✗	✓	✓
Copilot knowledge bases	✗	✗	✗	✓
Fine tuning a custom large language model (limited public preview) ^[5]	✗	✗	✗	✓

■組織レベルのポリシーの例

Manage accounts and repositories



Policies



Enforce policies



About enterprise policies

Repository management policies

Projects policies

Restrict email notifications

GitHub Sponsors policies

Policies for security settings

GitHub Actions policies

GitHub Copilot policies

GitHub Codespaces policies

Security & analysis

Personal access token policies

Create rulesets

Manage rulesets

- [Copilot in GitHub.com](#)
- [GitHub Copilot in the CLI](#)
- [GitHub Copilot Chat in the IDE](#)
- [Editor preview features](#)
- [GitHub Copilot Chat in GitHub Mobile](#)
- [GitHub Copilot Extensions](#)
- [Suggestions matching public code](#)
- [Give Copilot access to Bing](#)
- [Copilot access to alternative AI models](#)

GitHub.comでチャットの利
用の可否（等）

プレビューの機能の利用の
可否

パブリックなGitHubリポジ
トリのコードに一致する
コードの提案の可否

デフォルトのモデル（GPT-
4o）以外のモデルの利用の
可否

Enterpriseポリシー内の
GitHub Copilotポリシー

■価格レベルによる機能の違い（の一部を簡単にご紹介）

「Business」
「Enterprise」では、監査ログ（組織内のどのユーザーがいつ、どの操作をしたかの記録）が参照できる。

CSV形式でのエクスポートも可能

Enterpriseの場合はREST API / GraphQL APIによるログアクセスも可能

	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Organization-wide policy management	✗	✗	✓	✓
Audit logs	✗	✗	✓	✓
Increased GitHub Models rate limits ^[4]	✗	✗	✓	✓
Copilot knowledge bases	✗	✗	✗	✓
Fine tuning a custom large language model (limited public preview) ^[5]	✗	✗	✗	✓

■ 「監査ログ」の種類 (Action)



Version: Free, Pro, & Team ▾

Search GitHub Docs

← Home

Organizations

Collaborate with groups

Manage membership

たとえばユーザーへの
「シート」（課金ユー
ザー）の割り当て、削
除、といった監査ロ
グが取得できる。

どのユーザーがどのよ
うなプロンプトを入力
したか、までは（今
のところ）監査ログには
出力されない模様。

Action	Description
<code>copilot.access_revoked</code>	Copilot access was revoked for the organization or enterprise due to its Copilot subscription ending, an issue with billing the entity, the entity being marked spammy, or the entity being suspended.
<code>copilot.cfb_org_settings_changed</code>	Copilot feature settings were changed at the organization level.
<code>copilot.cfb_seat_added</code>	A Copilot Business or Copilot Enterprise seat was added for a user and they have received access to GitHub Copilot. This can occur as the result of directly assigning a seat for a user, assigning a seat for a team, or setting the organization to allow access for all members.
<code>copilot.cfb_seat_assignment_created</code>	A Copilot Business or Copilot Enterprise seat assignment was newly created for a user or a team, and seats are being created.
<code>copilot.cfb_seat_assignment_refreshed</code>	A seat assignment that was previously pending cancellation was re-assigned and the user will retain access to Copilot.
<code>copilot.cfb_seat_assignment_reused</code>	A Copilot Business or Copilot Enterprise seat assignment was re-created for a user who already had a seat with no pending cancellation date, and the user will retain access to Copilot.
<code>copilot.cfb_seat_assignment_unassigned</code>	A user or team's Copilot Business or Copilot Enterprise seat assignment was unassigned, and the user(s) will lose access to Copilot at the end of the current billing cycle.

In this article

About audit log events for your organization account

actions_cache

advisory_credit

artifact

auto_approve_personal_access_token_requests

billing

checks

code_scanning

codespaces

commit_comment

copilot

custom_property_definition

custom_property_value

dependabot_alerts

dependabot_alerts_new_repos

dependabot_repository_access

dependabot_security_updates

dependabot_security_updates_new_repos

dependency_graph

dependency_graph_new_repos

discussion_post

discussion_post_reply

enterprise_installation

environment

git

hook

integration

<https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-security-settings-for-your-organization/audit-log-events-for-your-organization#copilot>

■価格レベルによる機能の違い（の一部を簡単にご紹介）

	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Organization-wide policy management	✗	✗	✓	✓
Audit logs	✗	✗	✓	✓
Increased GitHub Models rate limits ^[4]	✗	✗	✓	✓
Copilot knowledge bases	✗	✗	✗	✓
Fine tuning a custom large language model (limited public preview) ^[5]	✗	✗	✗	✓

「Business」
「Enterprise」では、
GitHub Modelsのレート
リミットが増加

■レートリミットとは？

GitHub Copilotで使用できるモデルは、「GitHub Models」に登録されているモデルの一部である。

「GitHub Models」に登録されたモデルは、Low、High、Embeddingの3種類の「Tier」（階層）に分類されている。

Tierにより、1日あたり・1分あたりのリクエスト数や、リクエストあたりの最大の入力・出力トークン数などが定められている（レートリミット）。

Rate limits

The playground and free API usage are rate limited by tokens per request, and concurrent requests. If you get an error message that says that you hit to reset before you can make more requests, you can wait a few minutes and try again.

Low, high, and embedding models have different rate limits. To learn more about the specific requirements for each model you are using, refer to the model's information in GitHub Marketplace.

BusinessやEnterpriseではより高いレートリミットが設定されているため、より大規模な利用が可能となっている。

Rate limit tier	Rate limits	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Low	Requests per minute	15	15	15	20
	Requests per day	150	150	300	450
	Tokens per request	8000 in, 4000 out	8000 in, 4000 out	8000 in, 4000 out	8000 in, 8000 out
High	Concurrent requests	5	5	5	8
	Requests per minute	10	10	10	15
	Requests per day	50	50	100	150
Embedding	Tokens per request	8000 in, 4000 out	8000 in, 4000 out	8000 in, 4000 out	16000 in, 8000 out
	Concurrent requests	2	2	2	4
	Requests per minute	15	15	15	20
	Requests per day	150	150	300	450
	Tokens per request	64000	64000	64000	64000
	Concurrent requests	5	5	5	8

■ 「GitHub Models」のモデルの詳細画面の例

The screenshot shows the GitHub Models interface for the OpenAI GPT-4o model. The model name 'OpenAI GPT-4o' is highlighted with a red box. A yellow callout box contains Japanese text: 'たとえばOpenAIの「GPT-4o」モデルのTierは「High」と定義されている。' A yellow arrow points from this text to the 'Rate limit tier High' section in the details panel, which is also highlighted with a red box. The details panel includes sections for About, Context, Training date, Rate limit tier, Provider support, Tags, and Languages.

Marketplace / Models / Azure OpenAI Service

OpenAI GPT-4o

Enter a message...

Can you explain the basics of machine learning?

What are some of the most famous works of Shakespeare?

Preview Give feedback Playground Use this model

About

OpenAI's most advanced multimodal model in the gpt-4o family. Can handle both text and image inputs.

Context

131k input · 16k output

Training date

Oct 2023

Rate limit tier
High

Provider support

Azure support site

Tags 4

multipurpose multilingual multimodal chat-completion

Languages 27

English, Italian, Afrikaans, Spanish, German, French, Indonesian, Russian, Polish, Ukrainian, Greek, Latvian

■価格レベルによる機能の違い（の一部を簡単にご紹介）

	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Organization-wide policy management	✗	✗	✓	✓
Audit logs	✗	✗	✓	✓
Increased GitHub Models rate limits ^[4]	✗	✗	✓	✓
Copilot knowledge bases	✗	✗	✗	✓
Fine tuning a custom large language model (limited public preview) ^[5]	✗	✗	✗	✓

「Enterprise」では、GitHub Copilot Chatが、組織レベルで指定した外部リポジトリに置かれているMarkdownファイルを参照して回答できる

■価格レベルによる機能の違い（の一部を簡単にご紹介）

	Copilot Free	Copilot Pro	Copilot Business	Copilot Enterprise
Organization-wide policy management	✗	✗	✓	✓
Audit logs	✗	✗	✓	✓
Increased GitHub Models rate limits ^[4]	✗	✗	✓	✓
Copilot knowledge bases	✗	✗	✗	✓
Fine tuning a custom large language model (limited public preview) ^[5]	✗	✗	✗	✓

「Enterprise」では、モデルにトレーニングデータを与えてファインチューニングし、カスタムモデルを作つてコード提案で利用できる。

現在限定パブリックレビューのためすべての組織が使えるわけではない。

<https://github.blog/news-insights/product-news/fine-tuned-models-are-now-in-limited-public-beta-for-github-copilot-enterprise/>

<https://docs.github.com/en/copilot/about-github-copilot/subscription-plans-for-github-copilot>

Part 2 目次

- モジュール1 開発者のユースケース
 - ドキュメント作成
 - コード作成
- モジュール2 単体テストの開発
- モジュール3 GitHub Copilot Businessの概要
- モジュール4 GitHub Copilot Enterpriseの概要
- モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- モジュール6 PythonでGitHub Copilotを使用する（オプション）

演習: JavaScriptでGitHub Copilotを使用する

- 演習ガイド
 - [演習 - GitHub Copilot を使用して JavaScript ポートフォリオを更新する - Training | Microsoft Learn](#)
- 演習環境: GitHub Codespaces
 - 個人のGitHubアカウントが必要です
 - 以下のリポジトリを開く
 - [MicrosoftDocs/mslearn-copilot-codespaces-javascript](#)
 - ページ内の案内に従い、 GitHub Codespacesを起動して演習を開始

Part 2 目次

- モジュール1 開発者のユースケース
 - ドキュメント作成
 - コード作成
- モジュール2 単体テストの開発
- モジュール3 GitHub Copilot Businessの概要
- モジュール4 GitHub Copilot Enterpriseの概要
- モジュール5 JavaScriptでGitHub Copilotを使用する（オプション）
- モジュール6 PythonでGitHub Copilotを使用する（オプション）

PythonでGitHub Copilotを使用する

- 演習ガイド
 - [演習 - GitHub Copilot を使用して Python Web API を更新する - Training | Microsoft Learn](#)
- 演習環境: GitHub Codespaces
 - 個人のGitHubアカウントが必要です
 - 以下のリポジトリを開く
 - [MicrosoftDocs/mslearn-copilot-codespaces-python](#)
 - ページ内の案内に従い、 GitHub Codespacesを起動して演習を開始