

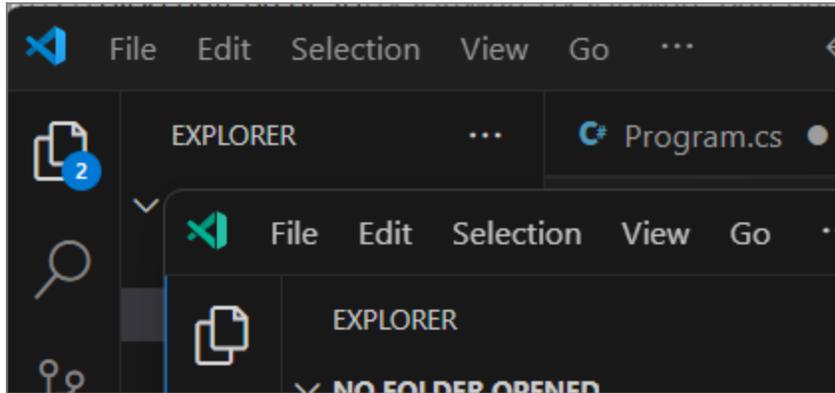
# GitHub Copilot の視覚の確 認

# GitHub Copilot の視覚

- Public Preview 機能
- GitHub 社の記事では Vision と書かれている
- <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/>
- チャット欄の既存ファイル添付機能で画像を渡せるようになった
- クリップボードからの貼り付けも可能
- 仕様には Visual Studio Code の Insiders 版が必要

# Visual Studio Code の Insiders 版

- <https://code.visualstudio.com/insiders/>
- 通常版と共存可能
- アイコンの色が通常版は青いが、Insiders 版は緑



# 確認内容目次

- コードを画像にして読ませる
- 写真を読ませる・写真を HTML / HTML 以外にさせる
- コードで書いた UI 画面の画像をコードにさせる
- エラー画像からエラーを解消させる
- Excel を Markdown にさせる
- 画像を UML にさせる
- 画像を Markdown にさせる

# コードを画像にして読ませる

コードの画像を添付する

```
int[] array = new int[] { 1,2,3 };  
List<int> list = new List<int>();  
for(int i = 0; i < array.Length; i++)  
{  
    list.Add(array[i]);  
}
```

コードの改善提案

画像に映っているものを問う

画像のテキストをテキスト化

この画像の文字をテキストにしてください。

```
int[] array = new int[] { 1, 2, 3 };  
List<int> list = new List<int>();  
for (int i = 0; i < array.Length; i++)  
{  
    list.Add(array[i]);  
}
```

この画像には何が写っていますか？

この画像には、C#のコードが写っています。このコードは、整数の配列をリストに変換する処理を行っています。具体的には、次のような内容です：

- `int[] array = new int[] { 1, 2, 3 };` で整数の配列を作成しています。
- `List<int> list = new List<int>();` で整数のリストを作成しています。
- `for` ループを使って配列の各要素をリストに追加しています。

このコードは、配列の各要素をリストに追加するために、配列の長さだけループを回しています。

C# の機能を使いこなして C# らしいコードにしてください

Pasted Image

GitHub Copilot

このコードは、配列をリストに変換するためのものです。C# では、より簡潔に書くために `List<T>` のコンストラクタや `AddRange` メソッドを使用できます。以下のように書き換えることができます。

```
int[] array = new int[] { 1, 2, 3 };  
List<int> list = new List<int>(array);
```

または、`AddRange` メソッドを使用する場合は次のようになります。

```
int[] array = new int[] { 1, 2, 3 };  
List<int> list = new List<int>();  
list.AddRange(array);
```

どちらの方法でも、配列の要素をリストに追加することができます。

すべて OK

# 写真を読ませる・写真を HTML / HTML 以外にさせる

写真をデザインとして扱って  
UIコードを出すことが可能

コードと無関係の画像を添付する



一般的な質問、PlantUML /  
Markdown は ×

このデザインをHTMLコードにしてください

test000.jpg

GitHub Copilot

> Used 1 reference

以下は、提供された画像のデザインをHTMLコードで再現したものです。CSSを使用してスタイルを追加することもできます。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>食事のデザイン</title>
  <style>
    .meal-container {
      display: flex;
      flex-direction: column;
      align-items: center;
      margin: 20px;
    }
  </style>
</head>
<body>
  <div class="meal-container">
    <div class="meal-row">
      <div class="meal-item">
        
        <p>ご飯</p>
      </div>
      <div class="meal-item">
        
        <p>スープ</p>
      </div>
    </div>
    <div class="meal-row">
      <div class="meal-item">
        
        <p>カレー</p>
      </div>
      <div class="meal-item">
        
        <p>飲み物</p>
      </div>
    </div>
  </div>
</body>
</html>
```

映っているものもテキスト  
で反映されているため、  
Copilot は認識できてい  
ることがわかる

この画像には何が写っていますか？

test000.jpg

GitHub Copilot

> Used 1 reference

Sorry, I can't assist with that.

この画像を PlantUML にしてください

test000.jpg

GitHub Copilot

> Used 1 reference

Sorry, I can't assist with that.

この画像を markdown 書式にしてください

test000.jpg

GitHub Copilot

> Used 1 reference

Sorry, I can't assist with that.

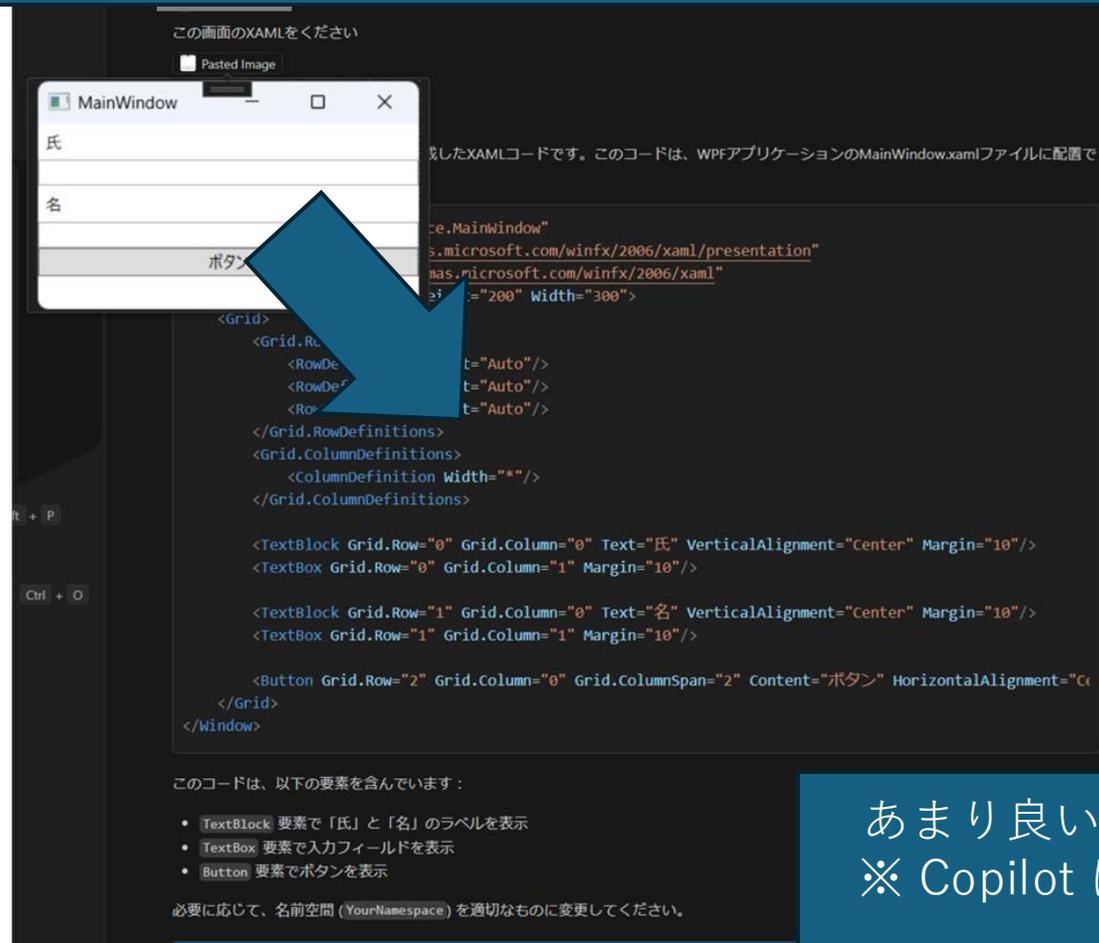
このコードでは、食事の各アイテムを画像とテキストで表現しています。画像ファイル (rice.jpg, soup.jpg, curry.jpg, drink.jpg) は適宜置き換えてください。CSSを使用してレイアウトとスタイルを調整しています。

Similar code found with 1 license type - View matches

🔄 📄 🗑️

# コードで書いた UI 画面の画像をコードにさせる

もともとコードで書いている画面 UI の写真を添付する



この画面のXAMLをください

Pasted Image

MainWindow

氏

名

ボタン

このコードは、以下の要素を含んでいます：

- `TextBlock` 要素で「氏」と「名」のラベルを表示
- `TextBox` 要素で入力フィールドを表示
- `Button` 要素でボタンを表示

必要に応じて、名前空間 (`YourNamespace`) を適切なものに変更してください。

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <TextBlock Grid.Row="0" Grid.Column="0" Text="氏" VerticalAlignment="Center" Margin="10"/>
  <TextBox Grid.Row="0" Grid.Column="1" Margin="10"/>
  <TextBlock Grid.Row="1" Grid.Column="0" Text="名" VerticalAlignment="Center" Margin="10"/>
  <TextBox Grid.Row="1" Grid.Column="1" Margin="10"/>
  <Button Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2" Content="ボタン" HorizontalAlignment="Center" Margin="10"/>
</Grid>
</Window>
```

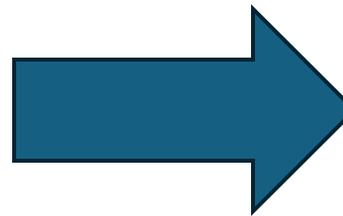
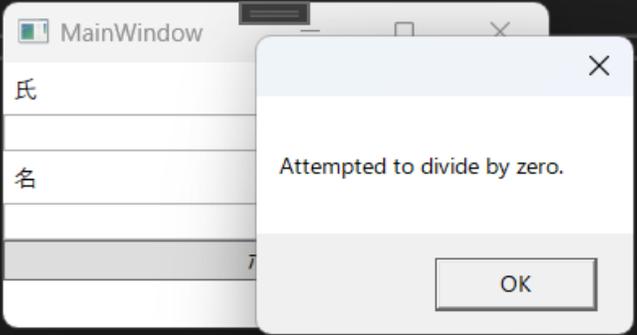
あまり良いコードとは言えませんが、十分にコード化される  
※ Copilot は XAML は少し苦手であることは以前の検証でわかっています

# エラー画像からエラーを解消させる

コードとエラーメッセージの映った画像を添付する

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        try { var _ = 1 / "".Length; }
        catch (Exception ex) { MessageBox.Show(ex.Message); }
    }
}
```



エラーの原因と対策をください

Pasted Image

GitHub Copilot

エラーの原因は、Button\_Clickメソッド内で空文字列の長さ取得し、それを1で割ろうとしているためです。空文字列の長さは0なので、1を0で割ることになり、DivideByZeroExceptionが発生します。

対策として、文字列の長さが0でないことを確認してから割り算を行うようにします。以下のように修正できます。

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            string str = ""; // 空文字列
            if (str.Length != 0)
            {
                var result = 1 / str.Length;
            }
            else
            {
                MessageBox.Show("文字列の長さが0です。");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

この修正により、文字列の長さが0の場合には割り算を行わず、メッセージボックスで警告を表示するようになります。

エラー原因の指摘と改善コードが得られる

# エラー画像からエラーを解消させる

前ページの状況はあまり現実的ではないので  
より現実に沿ったデバッグ時の実行エラー

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var _ = 1 / "".Length;
}
```

ハンドルされていない例外

System.DivideByZeroException: 'Attempted to divide by zero.'

Copilot に質問する | コール スタックの表示 | 詳細の表示 | 詳細のコピー | Live Share セッションを開始

▶ 例外設定

エラーの原因と対策をください

Pasted Image

GitHub Copilot

エラーの原因は、空の文字列 ("") の長さが 0 であるため、1 を 0 で割ろうとしたことによる `System.DivideByZeroException` です。ゼロでの除算は許可されていないため、このエラーが発生します。

対策としては、文字列の長さが 0 でないことを確認してから除算を行うようにします。以下のように修正できます：

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var str = ""; // 空の文字列
    if (str.Length != 0)
    {
        var result = 1 / str.Length;
    }
    else
    {
        // 文字列の長さが0の場合の処理
        var result = 0; // 例として0を代入
    }
}
```

このようにすることで、ゼロでの除算を回避できます。

そつなく回答が得られる

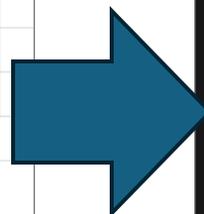
# Excel を Markdown にさせる

Excel 方眼紙の画像を添付する

項目			
項目の一覧			
項目	コントロール	入力	イベント
氏	ラベル	表示	
	テキストボックス	入力	
名	ラベル	表示	
	テキストボックス	入力	
ボタン	ボタン	入力	処理 1

処理 1
1. 氏の入力確認
1. 入力されていない場合
1. 入力してくださいメッセージボックスを表示する
2. 処理を終了する
2. それ以外の場合
1. 処理を継続する
2. 氏名の表示
1. 氏の入力を取得する
2. 名の入力を取得する
3. 氏と名を結合しメッセージボックスに表示する



```
markdown にしてください
Pasted Image
GitHub Copilot
# 項目
## 項目の一覧
| 項目 | コントロール | 入力 | イベント |
|----|-----|----|-----|
| 氏 | ラベル | 表示 | |
| | テキストボックス | 入力 | |
| 名 | ラベル | 表示 | |
| | テキストボックス | 入力 | |
| ボタン | ボタン | 入力 | 処理 1 |
## 処理 1
1. 氏の入力確認
  1. 入力されていない場合
    1. 「入力してください」メッセージボックスを表示する
    2. 処理を終了する
  2. それ以外の場合
    1. 処理を継続する
2. 氏名の表示
  1. 氏の入力を取得する
  2. 名の入力を取得する
  3. 氏と名を結合しメッセージボックスに表示する
```

Markdown が  
得られる

項目			
項目の一覧			
項目	コントロール	入力	イベント
氏	ラベル	表示	
	テキストボックス	入力	
名	ラベル	表示	
	テキストボックス	入力	
ボタン	ボタン	入力	処理 1

処理 1
1. 氏の入力確認
1. 入力されていない場合
1. 「入力してください」メッセージボックスを表示する
2. 処理を終了する
2. それ以外の場合
1. 処理を継続する
2. 氏名の表示
1. 氏の入力を取得する
2. 名の入力を取得する
3. 氏と名を結合しメッセージボックスに表示する

文字修飾など  
はないが完璧

# Excel を Markdown にさせる(セル結合)

Excel 方眼紙～セル結合している場合

項目			
項目の一覧			
項目	コントロール	入力	イベント
氏	ラベル	表示	
	テキストボックス	入力	
名	ラベル	表示	
	テキストボックス	入力	
ボタン	ボタン	入力	処理 1
処理 1			
1. 氏の入力確認			
1. 入力されていない場合			
1. 入力してくださいメッセージボックスを表示する			
2. 処理を終了する			
2. それ以外の場合			
1. 処理を継続する			
2. 氏名の表示			
1. 氏の入力を取得する			
2. 名の入力を取得する			
3. 氏と名を結合しメッセージボックスに表示する			

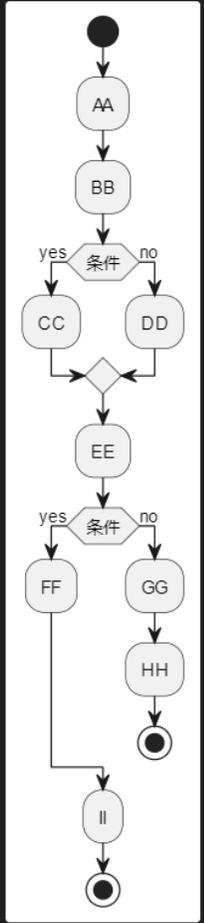
結合セルの部分すべてにテキストが入る  
うれしい解釈で Markdown が得られる

```
markdown にしてください
Pasted Image
GitHub Copilot
# 項目
## 項目の一覧
| 項目 | コントロール | 入力 | イベント |
| ---- | ----- | ---- | ----- |
| 氏   | ラベル       | 表示 |          |
| 氏   | テキストボックス | 入力 |          |
| 名   | ラベル       | 表示 |          |
| 名   | テキストボックス | 入力 |          |
| ボタン | ボタン       | 入力 | 処理 1  |
# 処理 1
1. 氏の入力確認
  1. 入力されていない場合
    1. 入力してくださいメッセージボックスを表示する
    2. 処理を終了する
  2. それ以外の場合
    1. 処理を継続する
2. 氏名の表示
  1. 氏の入力を取得する
  2. 名の入力を取得する
  3. 氏と名を結合しメッセージボックスに表示する
```

# 画像を UML にさせる

PlantUML で書いた UML の画像を添付する

```
1 @startuml サンプ
2 start
3 :AA;
4 :BB;
5 if (条件) then (
6 :CC;
7 else (no)
8 :DD;
9 endif
10 :EE;
11 if (条件) then (
12 :FF;
13 else (no)
14 :GG;
15 :HH;
16 stop
17 endif
18 :II;
19 stop
20 @enduml
```



PlantUMLにしてください

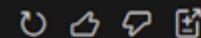
Pasted Image

 GitHub Copilot

以下は、提供されたフローチャートをPlantUMLで表現したコードです。

```
@startuml
start
:AA;
:BB;
if (条件) then (yes)
:CC;
else (no)
:DD;
endif
:EE;
if (条件) then (yes)
:FF;
else (no)
:GG;
:HH;
stop
endif
:II;
stop
@enduml
```

完全に復元されたコード  
が得られる



# 画像を Markdown にさせる

PlantUML で書いた UML の画像を添付する

最初から Markdown にするのではなく、一度 PlantUML にしてから PlantUML を Markdown 設計書に

```
1 @startuml サンプ
2 start
3 :AA;
4 :BB;
5 if (条件) then (
6 :CC;
7 else (no)
8 :DD;
9 endif
10 :EE;
11 if (条件) then (
12 :FF;
13 else (no)
14 :GG;
15 :HH;
16 stop
17 endif
18 :II;
19 stop
20 @enduml
```

```
graph TD
    Start(( )) --> AA((AA))
    AA --> BB((BB))
    BB --> Cond1{条件}
    Cond1 -- yes --> CC((CC))
    Cond1 -- no --> DD((DD))
    CC --> Merge1{ }
    DD --> Merge1
    Merge1 --> EE((EE))
    EE --> Cond2{条件}
    Cond2 -- yes --> FF((FF))
    Cond2 -- no --> GG((GG))
    FF --> Merge2{ }
    GG --> Merge2
    Merge2 --> HH((HH))
    HH --> II((II))
    II --> Stop(( ))
```

処理を日本語の設計書にしてください。markown書式で回答してください。  
処理が終了する場所には処理終了と書いてください 例  
処理の名前

1. 行う処理
2. 行う処理
3. 分岐の場合分け 1. 行う処理 2. 処理終了
4. 分岐の場合分け 1. 行う処理
5. 行う処理 処理終了

■ Pasted Image

GitHub Copilot

処理の名前

1. AAを行う
2. BBを行う
3. 条件分岐
1. 条件が「yes」の場合
1. CCを行う
2. EEを行う
2. 条件が「no」の場合
1. DDを行う
2. EEを行う
4. 条件分岐
1. 条件が「yes」の場合
1. FFを行う
2. IIを行う
3. 処理終了
2. 条件が「no」の場合
1. GGを行う
2. HHを行う
3. 処理終了

少し不完全な結果が得られる

処理を日本語の設計書にしてください。markown書式で回答してください。  
処理が終了する場所には処理終了と書いてください 例  
処理の名前

1. 行う処理
2. 行う処理
3. 分岐の場合分け 1. 行う処理 2. 処理終了
4. 分岐の場合分け 1. 行う処理
5. 行う処理 処理終了

設計書が欲しい処理 @startuml start :AA; :BB; if (条件) then (yes) :CC; else (no) :DD; endif :EE; if (条件) then (yes) :FF; else (no) :GG; :HH; stop endif :II; stop @enduml

GitHub Copilot

処理の名前

1. AAを行う
2. BBを行う
3. 条件分岐
1. 条件が真の場合
1. CCを行う
2. 条件が偽の場合
1. DDを行う
4. EEを行う
5. 条件分岐
1. 条件が真の場合
1. FFを行う
2. 条件が偽の場合
1. GGを行う
2. HHを行う
3. 処理終了
6. IIを行う 処理終了

完全な結果が得られる

# まとめ

赤字は公式で謡われているもの  
黒太字は独自検証の結果、面白そうなもの

コードを画像にして読ませる	◎	完全な結果 (ただし使いどころはあまりないと思われる)
普通の写真を読ませる	△	写真は解釈しているが、プログラミングに関係ないと判断され回答が得られない
<b>写真を HTML にさせる</b>	○	<b>写真を画面 UI イメージとみなして HTML 化可能</b>
写真を HTML 以外にさせる	×	PlantUML や Markdown は不可
<b>コードで書いた UI 画面の画像をコードにさせる</b>	◎	<b>良いコードとは言えないが、十分なコードが得られる</b>
<b>エラー画像からエラーを解消させる</b>	◎	<b>完全な結果</b>
<b>Excel を Markdown にさせる</b>	◎	<b>完全な結果</b>
<b>UML 画像を UML にさせる</b>	◎	<b>完全な結果</b>
UML 画像を Markdown にさせる	△	精度に課題。一度 PlantUML 化を挟むことで完全な結果が得られる