**Optimizing inefficient code**

Copilot can help you to optimize code - for example, to make the code run more quickly.

**Example code**

In the two sections below, we'll use the following example bash script to demonstrate how to optimize inefficient code:

#!/bin/bash

# Find all .txt files and count lines in each

for file in $(find . -type f -name "\*.txt"); do

wc -l "$file"

done

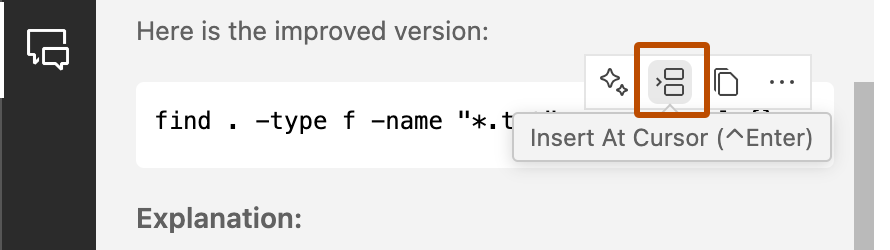
**Use the Copilot Chat panel**

Copilot can tell you whether code, like the example bash script, can be optimized.

1. Select either the for loop or the entire contents of the file.
2. Open Copilot Chat by clicking the chat icon in the activity bar or by using the keyboard shortcut:
   * **VS Code and Visual Studio:** Control+Command+i (Mac) / Ctrl+Alt+i (Windows/Linux)
   * **JetBrains:** Control+Shift+c
3. In the input box at the bottom of the chat panel, type: **Can this script be improved?**

Copilot replies with a suggestion that will make the code more efficient.

1. To apply the suggested change:
   * **In VS Code and JetBrains:** Hover over the suggestion in the chat panel and click the **Insert At Cursor** icon.



* + **In Visual Studio:** Click **Preview** then, in the comparison view, click **Accept**.

**Use Copilot inline chat**

Alternatively, if you already know that existing code, like the example bash script, is inefficient:

1. Select either the for loop or the entire contents of the file.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type optimize and press Enter.

Copilot suggests revised code. For example:

find . -type f -name "\*.txt" -exec wc -l {} +

This is more efficient than the original code, shown earlier in this article, because using -exec ... + allows find to pass multiple files to wc at once rather than calling wc once for each \*.txt file that's found.

1. Assess Copilot's suggestion and, if you agree with the change:
   * **In VS Code and Visual Studio:** Click **Accept**.
   * **In JetBrains:** Click the Preview icon (double arrows), then click the Apply All Diffs icon (double angle brackets).

As with all Copilot suggestions, you should always check that the revised code runs without errors and produces the correct result.

**Cleaning up repeated code**

Avoiding repetition will make your code easier to revise and debug. For example, if the same calculation is performed more than once at different places in a file, you could move the calculation to a function.

In the following very simple JavaScript example, the same calculation (item price multiplied by number of items sold) is performed in two places.

let totalSales = 0;

let applePrice = 3;

let applesSold = 100;

totalSales += applePrice \* applesSold;

let orangePrice = 5;

let orangesSold = 50;

totalSales += orangePrice \* orangesSold;

console.log(`Total: ${totalSales}`);

You can ask Copilot to move the repeated calculation into a function.

1. Select the entire contents of the file.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type: move repeated calculations into functions and press Enter.

Copilot suggests revised code. For example:

function calculateSales(price, quantity) {

return price \* quantity;

}

let totalSales = 0;

let applePrice = 3;

let applesSold = 100;

totalSales += calculateSales(applePrice, applesSold);

let orangePrice = 5;

let orangesSold = 50;

totalSales += calculateSales(orangePrice, orangesSold);

console.log(`Total: ${totalSales}`);

1. Assess Copilot's suggestion and, if you agree with the change:
   * **In VS Code and Visual Studio:** Click **Accept**.
   * **In JetBrains:** Click the Preview icon (double arrows), then click the Apply All Diffs icon (double angle brackets).

As with all Copilot suggestions, you should always check that the revised code runs without errors and produces the correct result.

**Making code more concise**

If code is unnecessarily verbose it can be difficult to read and maintain. Copilot can suggest a more concise version of selected code.

In the following example, this Python code outputs the area of a rectangle and a circle, but could be written more concisely:

def calculate\_area\_of\_rectangle(length, width):

area = length \* width

return area

def calculate\_area\_of\_circle(radius):

import math

area = math.pi \* (radius \*\* 2)

return area

length\_of\_rectangle = 10

width\_of\_rectangle = 5

area\_of\_rectangle = calculate\_area\_of\_rectangle(length\_of\_rectangle, width\_of\_rectangle)

print(f"Area of rectangle: {area\_of\_rectangle}")

radius\_of\_circle = 7

area\_of\_circle = calculate\_area\_of\_circle(radius\_of\_circle)

print(f"Area of circle: {area\_of\_circle}")

1. Select the entire contents of the file.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type: make this more concise and press Enter.

Copilot suggests revised code. For example:

import math

def calculate\_area\_of\_rectangle(length, width):

return length \* width

def calculate\_area\_of\_circle(radius):

return math.pi \* (radius \*\* 2)

print(f"Area of rectangle: {calculate\_area\_of\_rectangle(10, 5)}")

print(f"Area of circle: {calculate\_area\_of\_circle(7)}")

1. Assess Copilot's suggestion and, if you agree with the change:
   * **In VS Code and Visual Studio:** Click **Accept**.
   * **In JetBrains:** Click the Preview icon (double arrows), then click the Apply All Diffs icon (double angle brackets).

As with all Copilot suggestions, you should always check that the revised code runs without errors and produces the correct result.

**Splitting up complex units of code**

Large methods or functions that perform multiple operations are likely to offer fewer opportunities for reuse than smaller, simpler functions that are focused on performing a particular operation. They may also be more difficult to understand and debug.

Copilot can help you to split up complex blocks of code into smaller units that are more suitable for reuse.

The following Python code is a very simple example, but it shows the principle of splitting up a single function into two functions that perform particular operations.

import pandas as pd

from pandas.io.formats.style import Styler

def process\_data(item, price):

# Cleanse data

item = item.strip() # Strip whitespace from item

price = price.strip() # Strip whitespace from price

price = float(price) # Convert price to a float

# More cleansing operations here

# Create and print a DataFrame

data = {'Item': [item], 'Price': [price]}

df = pd.DataFrame(data)

print(df.to\_string(index=False))

# Example usage

item = " Apple "

price = " 1.25"

process\_data(item, price)

To split up the process\_data function:

1. Put the cursor in the function name.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type: split into 2 separate functions: one for cleansing data, the other for printing and press Enter.

Copilot suggests revised code. For example:

def cleanse\_data(item, price):

# Cleanse data

item = item.strip() # Strip whitespace from item

price = price.strip() # Strip whitespace from price

price = float(price) # Convert price to a float

return item, price

def print\_data(item, price):

# Create and print a DataFrame

data = {'Item': [item], 'Price': [price]}

df = pd.DataFrame(data)

print(df.to\_string(index=False))

def process\_data(item, price):

item, price = cleanse\_data(item, price)

print\_data(item, price)

Note

The example suggestion shown above replaces the existing function with three functions. The third function simply calls the other two functions, and allows for existing code that calls the process\_data function.

1. Assess Copilot's suggestion and, if you agree with the change:
   * **In VS Code and Visual Studio:** Click **Accept**.
   * **In JetBrains:** Click the Preview icon (double arrows), then click the Apply All Diffs icon (double angle brackets).

As with all Copilot suggestions, you should always check that the revised code runs without errors and produces the correct result.

**Rewrite conditional code for better readability**

There are often several ways to write code that does, or does not, get executed depending on various conditions. Some conditional structures are better suited than others to particular use cases, and choosing an alternative conditional structure can sometimes make the code easier to read.

This Java method uses a series of if and else if statements to determine which operation to perform:

public String getSound(String animal) {

if (animal == null) {

System.out.println("Oops! A null animal?");

} else if (animal.equalsIgnoreCase("Dog")) {

return "Bark";

} else if ( animal.equalsIgnoreCase("Cat")) {

return "Meow";

} else if ( animal.equalsIgnoreCase("Bird")) {

return "Tweet";

}

return "Unknown";

}

A switch statement might be a better way of applying the same logic.

1. Put the cursor in the method name.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type: rewrite the condition to use a switch and use Java 21 syntax with null case, but also add documentation and provide a better function name.

This demonstrates how you can use a single prompt to ask Copilot to make multiple changes.

1. Press Enter.

Copilot suggests revised code. For example:

/\*\*

\* Retrieves the sound made by a given animal.

\*

\* @param animal The name of the animal.

\* @return The sound made by the animal, or "Unknown" if the animal is not recognized.

\*/

public String getAnimalSound(String animal) {

return switch (animal) {

case null -> {

System.out.println("Oops! A null animal?");

yield "Unknown";

}

case String a when a.equalsIgnoreCase("Dog") -> "Bark";

case String a when a.equalsIgnoreCase("Cat") -> "Meow";

case String a when a.equalsIgnoreCase("Bird") -> "Tweet";

default -> "Unknown";

};

}

1. Assess Copilot's suggestion and, if you agree with the change:
   * **In VS Code and Visual Studio:** Click **Accept**.
   * **In JetBrains:** Click the Preview icon (double arrows), then click the Apply All Diffs icon (double angle brackets).

As with all Copilot suggestions, you should always check that the revised code runs without errors and produces the correct result.

**Reformat code to use a different structure**

Suppose you have this function in JavaScript:

function listRepos(o, p) {

return fetch(`https://api.github.com/orgs/${o}/repos?per\_page=${parseInt(p)}`)

.then((response) => response.json())

.then( (data) => data);

}

If your coding standards require you to use the arrow notation for functions, and descriptive names for parameters, you can use Copilot to help you make these changes.

1. Put the cursor in the function name.
2. Open inline chat:
   * **In VS Code:** Press Command+i (Mac) or Ctrl+i (Windows/Linux).
   * **In Visual Studio:** Press Alt+/.
   * **In JetBrains IDEs:** Press Control+Shift+i (Mac) or Ctrl+Shift+g (Windows/Linux).
3. Type: use arrow notation and better parameter names and press Enter.

Copilot suggests revised code. For example:

const listRepos = (org, perPage) => {

return fetch(`https://api.github.com/orgs/${org}/repos?per\_page=${parseInt(perPage)}`)

.then(response => response.json())

.then(data => data);

};

**Improving the name of a symbol**

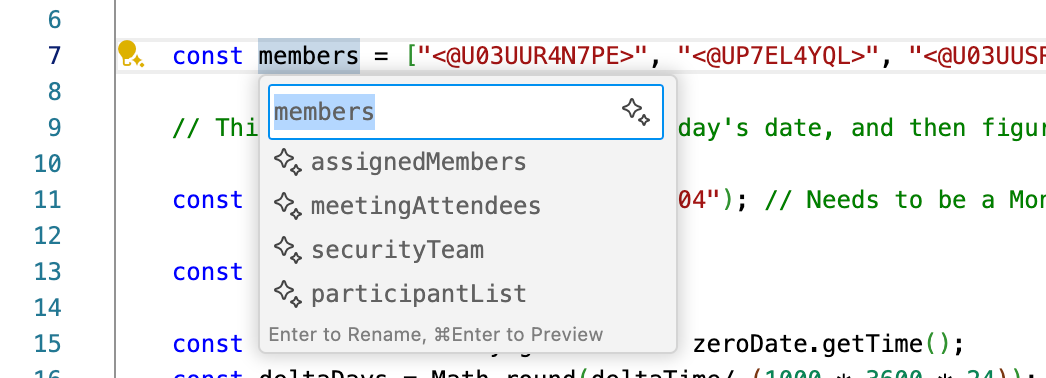
Note

* VS Code and Visual Studio only.
* Support for this feature depends on having the appropriate language extension installed in your IDE for the language you are using. Not all language extensions support this feature.

Well chosen names can help to make code easier to maintain. Copilot in VS Code and Visual Studio can suggest alternative names for symbols such as variables or functions.

1. Put the cursor in the symbol name.
2. Press F2.
3. **Visual Studio only:** Press Ctrl+Space.

Copilot suggests alternative names.



1. In the dropdown list, select one of the suggested names.

The name is changed throughout the project.