

# Generate code using GitHub Copilot code completion suggestions

6 minutes

GitHub Copilot can provide code completion suggestions for numerous programming languages and a wide variety of frameworks, but works especially well for Python, JavaScript, TypeScript, Ruby, Go, C# and C++. Code line completions are generated based on the context of the code you're writing. You can accept, reject, or partially accept the suggestions provided by GitHub Copilot.

GitHub Copilot provides two ways to generate code line completions:

- **From a comment:** You can generate code line completions by writing a comment that describes the code you want to generate. GitHub Copilot provides code completion suggestions based on the comment you write.
- **From code:** You can generate code line completions by starting a code line, or by pressing Enter after a completed code line. GitHub Copilot provides code completion suggestions based on the code you write.

## Use GitHub Copilot to generate code line completions from a comment

GitHub Copilot generates code completion suggestions based on the comment and the existing context of your app.

You can use comments to describe code snippets, methods, data structures, and other code elements.

Suppose you have the following code snippet:

C#

```
namespace ReportGenerator;  
  
class QuarterlyIncomeReport  
{  
    static void Main(string[] args)  
    {
```

```
// create a new instance of the class
QuarterlyIncomeReport report = new QuarterlyIncomeReport();

// call the GenerateSalesData method

// call the QuarterlySalesReport method

}

public void QuarterlySalesReport()
{

    Console.WriteLine("Quarterly Sales Report");

}
}
```

For example, the following comment could be used to create a data structure:

C#

```
// public struct SalesData. Include the following fields: date sold, de-
partment name, product ID, quantity sold, unit price
```

GitHub Copilot generates one or more code completion suggestions based on your code comment and the code files that are open in the editor.

```
/* public struct SalesData includes the following fields: date sold,
department name, product ID, quantity sold, unit price */
```

< 1/2 > Accept  Accept Word  ...

```
public struct SalesData
{
    public DateTime dateSold;
    public string departmentName;
    public int productID;
    public int quantitySold;
    public double unitPrice;
}
```

Notice the data types used to declare the fields of the data structure. GitHub Copilot selects data types and variable names based on your existing code and the code comment. GitHub Copilot tries to determine how the application uses variables and defines the data types accordingly.

When GitHub Copilot generates more than one suggestion, you can cycle through the suggestions by selecting the left or right arrows (> or <) located to the left of the **Accept** button. This allows you to review and select the suggestion that best fits your needs.

It's okay to accept a code completion suggestion that isn't an exact match for what you want. However, the changes required to "fix" the suggestion should be clear. In this case, some of the data types aren't what you want, but you can adjust them after accepting the suggested autocompletion.

If none of the suggested options resemble what you need, there are two things you can try. To open a new editor tab containing a list of other suggestions, press the **Ctrl + Enter** keys. This hotkey combination opens a new tab containing up to 10 more suggestions. Each suggestion is followed by a button that you can use to accept the suggestion. The tab closes automatically after you accept a suggestion. Your other option is to press the **Esc** key to dismiss the suggestions and try again. You can adjust the code comment to provide more context for GitHub Copilot to work with.

#### ⓘ Note

GitHub Copilot can occasionally propose a suggestion in stages. If this happens, you can press Enter to see additional stages of the suggestion after pressing the Tab key.

To accept a suggested data structure, press the Tab key or select **Accept**.

To modify the field data types, update your code as follows:

C#

```
public struct SalesData
{
    public DateOnly dateSold;
    public string departmentName;
    public int productID;
    public int quantitySold;
    public double unitPrice;
}
```

Making quick adjustments to code completion suggestions helps to ensure that you're building the code you want. It's especially important to make corrections early in your development process when large portions of your codebase still need to be developed. Subsequent code completions are based on the code you've already written, so it's important to ensure that your code is as accurate as possible.

# Use GitHub Copilot to generate code line completions from a comment

GitHub Copilot generates code completion suggestions based on the comment and the existing context of your app. You can use comments to describe code snippets, methods, data structures, and other code elements.

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

1. In the **Program.cs** file, create two empty code lines below the **Main** method.
2. To create a data structure that can be used to generate test data, create the following code comment, and then press Enter:

C#

```
// public struct SalesData. Include the following fields: date sold,  
department name, product ID, quantity sold, unit price
```

GitHub Copilot generates one or more code completion suggestions based on your code comment and any existing code that it finds in your app.

3. Take a minute to review the code completion suggestions provided by GitHub Copilot.

## ⓘ Note

If GitHub Copilot generates suggestions for a method rather than a data structure, type **public str** and wait for the code completion suggestion to update. GitHub Copilot uses the additional information to improve its suggestions.

```
/* public struct SalesData includes the following fields: date sold,  
department name, product ID, quantity sold, unit price */
```

< 1/2 > Accept [Tab] Accept Word [Ctrl] + [RightArrow] ...

```
public struct SalesData  
{  
    public DateTime dateSold;  
    public string departmentName;  
    public int productID;  
    public int quantitySold;  
    public double unitPrice;  
}
```

Notice the data types used to declare the fields of the data structure. GitHub Copilot selects data types and variable names based on your existing code and the code

comment. GitHub Copilot tries to determine how the application uses variables and defines the data types accordingly.

When GitHub Copilot generates more than one suggestion, you can cycle through the suggestions by selecting the left or right arrows (> or <) located to the left of the **Accept** button. This allows you to review and select the suggestion that best fits your needs.

It's okay to accept a code completion suggestion that isn't an exact match for what you want. However, the changes required to "fix" the suggestion should be clear. In this case, some of the data types aren't what you want, but you can adjust them after accepting the suggested autocomplete.

If none of the suggested options resemble what you need, there are two things you can try. To open a new editor tab containing a list of other suggestions, press the **Ctrl + Enter** keys. This hotkey combination opens a new tab containing up to 10 more suggestions. Each suggestion is followed by a button that you can use to accept the suggestion. The tab closes automatically after you accept a suggestion. Your other option is to press the **Esc** key to dismiss the suggestions and try again. You can adjust the code comment to provide more context for GitHub Copilot to work with.

#### ⓘ Note

GitHub Copilot can occasionally propose a suggestion in stages. If this happens, you can press **Enter** to see additional stages of the suggestion after pressing the **Tab** key.

4. To accept a suggested data structure, press the **Tab** key or select **Accept**.

5. To modify the field data types, update your code as follows:

C#

```
public struct SalesData
{
    public DateOnly dateSold;
    public string departmentName;
    public int productID;
    public int quantitySold;
    public double unitPrice;
}
```

Making quick adjustments to code completion suggestions helps to ensure that you're building the code you want. It's especially important to make corrections early in your development process when large portions of your codebase still need to be developed.

Code completions are based on your existing code, so it's important to ensure that your code is as accurate as possible.

6. Create two empty code lines below the `SalesData` data structure.
7. To create a method that generates test data using the `SalesData` data structure, write the following code comment and then press Enter:

C#

```
/* the GenerateSalesData method returns 1000 SalesData records. It assigns random values to each field of the data structure */
```

8. Take a minute to review the code completion suggestions provided by GitHub Copilot.

Notice that the `GenerateSalesData` method is designed to return an array of `SalesData` objects. The method generates 1,000 records of test data, with random values assigned to each field of the `SalesData` data structure.

```
/* the GenerateSalesData method returns 1000 SalesData records. It assigns random values to each field of the data structure */  
< 2/3 > Accept [Tab] Accept Word [Ctrl] + [RightArrow] ...  
public SalesData[] GenerateSalesData()  
{  
    SalesData[] salesData = new SalesData[1000];  
    Random random = new Random();  
    for (int i = 0; i < 1000; i++)  
    {  
        salesData[i].dateSold = new DateOnly(random.Next(1, 13), random.Next(1, 29));  
        salesData[i].departmentName = "Department " + random.Next(1, 11);  
        salesData[i].productID = random.Next(1, 101);  
        salesData[i].quantitySold = random.Next(1, 101);  
        salesData[i].unitPrice = random.NextDouble() * 100;  
    }  
    return salesData;  
}
```

You should always review the suggestions proposed by GitHub Copilot and GitHub Copilot Chat, even when they appear to be correct.

#### ⚠ Note

If GitHub Copilot suggests a single code line rather than a completed `GenerateSalesData` method, press **Ctrl + Enter** to open the GitHub Copilot Suggestions tab. Review the suggestions on the new tab. On the next step, use the "Accept suggestion #" button to accept the suggestion. GitHub Copilot presents suggestions incrementally on occasion. Although you can accept the code

completions incrementally, it's better to use the GitHub Copilot Suggestions tab to review full suggestion before making a decision to accept or discard.

9. Scroll through the code completion suggestions and select the best match for the requirements.

10. To accept the code completion, press the Tab key.

Notice that the code completion suggestion includes a syntax error in the code used to generate the `DateSold` field. `DateOnly` accepts three integer values that must be listed in the correct order: **Year, Month, Day**.

11. To specify a single year for the code used to generate the `DateSold` field, update the code line as follows:

C#

```
salesData[i].DateSold = new DateOnly(2023, random.Next(1, 13), random.Next(1, 29));
```

12. If necessary, adjust the other code lines to match the following code snippet:

C#

```
public SalesData[] GenerateSalesData()
{
    SalesData[] salesData = new SalesData[1000];
    Random random = new Random();

    for (int i = 0; i < salesData.Length; i++)
    {
        salesData[i].dateSold = new DateOnly(2023, random.Next(1, 13), random.Next(1, 29));
        salesData[i].departmentName = "Department " + random.Next(1, 11);
        salesData[i].productID = random.Next(1, 101);
        salesData[i].quantitySold = random.Next(1, 101);
        salesData[i].unitPrice = random.NextDouble() * 100;
    }

    return salesData;
}
```

The ability to generate code from code comments is a powerful feature of GitHub Copilot. With just two comments, you were able to generate a data structure and a method that generates test data.

# Use GitHub Copilot to generate code line completions

GitHub Copilot can generate code line completions based on the code you enter. You can generate code line completions in two ways:

- Start entering a code line, and then wait for GitHub Copilot to suggest an autocompletion for your unfinished code line.
- Enter a complete code line, press the **Enter** key, and then wait for GitHub Copilot to suggest an autocompletion for the next code line.

## ⓘ Note

GitHub Copilot generates suggested code completions based on the code you enter and the context defined by the code within your app. The more code you have available in your app, the more context GitHub Copilot has when generating a response. GitHub Copilot can base responses on your existing code, so the quality of your code is important. As the volume and quality of existing code increases, so does the quality and reliability of the code line completions suggested by GitHub Copilot. GitHub Copilot is good at generating code line completions for common programming tasks and patterns, especially when a sequence of related components needs to be generated.

In this portion of the exercise, you work on the `QuarterlySalesReport` method.

Here are the tasks you need to complete:

- Update the method constructor with a parameter that accepts your collection of `SalesData` objects.
- Use GitHub Copilot to generate code line completions that process sales data for the quarterly report.
- Run the app and review the quarterly sales report.

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

1. Update the method constructor for `QuarterlySalesReport` as follows:

```
C#
```

```
public void QuarterlySalesReport(SalesData[] salesData)
```

2. Take a minute to consider the code that you need to develop.

The concept is straight forward. You want your code to calculate quarterly sales based on your sales data and then write a report. To do that, your code needs to:



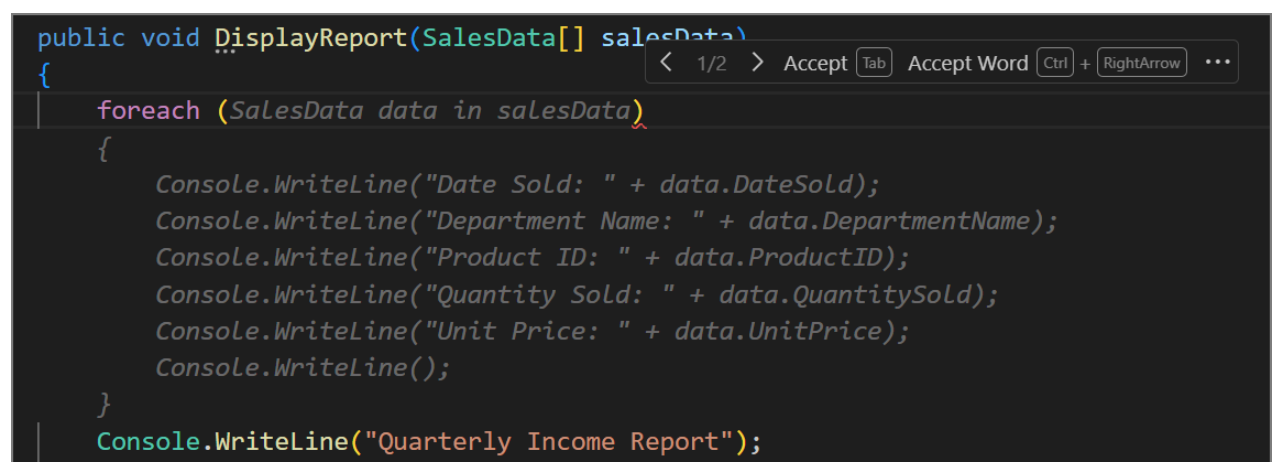
- Iterate through the `salesData` collection.
- Calculate the value of each sale based on the quantity sold and the unit price.
- Use the sales date to determine which quarter a sale belongs to.
- Sum the sales for each quarter.
- Write a report of the sales by quarter.

One option is to begin entering the code for a `foreach` loop and then see what GitHub Copilot suggests.

3. In the `QuarterlySalesReport` method, create a new code line at the top of the code block.

There should be at least one blank code line between the new code line and the code line containing `Console.WriteLine()`.

4. To generate a code line completion, type `foreach (` and then wait for GitHub Copilot to suggest code line completion options.
5. Review the code completion suggested by GitHub Copilot.



```
public void DisplayReport(SalesData[] salesData)
{
    foreach (SalesData data in salesData)
    {
        Console.WriteLine("Date Sold: " + data.DateSold);
        Console.WriteLine("Department Name: " + data.DepartmentName);
        Console.WriteLine("Product ID: " + data.ProductID);
        Console.WriteLine("Quantity Sold: " + data.QuantitySold);
        Console.WriteLine("Unit Price: " + data.UnitPrice);
        Console.WriteLine();
    }
    Console.WriteLine("Quarterly Income Report");
}
```

The suggested code completion isn't what you wanted.

Although GitHub Copilot suggests a `foreach` loop that iterates through the `salesData`, there's no analysis or calculations inside the loop. The suggested code includes `Console.WriteLine` statements that you don't want or need.

6. Take a minute to consider why GitHub Copilot is suggesting `Console.WriteLine` statements.

Recall that GitHub Copilot generates code completion suggestions based on the context of your code. In this case, you don't really have much code for GitHub Copilot to consider. And the situation gets worse.

The code that GitHub Copilot does see inside your method is a `Console.WriteLine` statement. With no other context available within the method and no similar methods in your codebase to draw from, GitHub Copilot concludes that you may *want* `Console.WriteLine` statements inside the `foreach` loop.

GitHub Copilot works best when your code is clean and focused. If you see superfluous code comments or statements in your code, you may want to remove them before you try using GitHub Copilot code completions.

7. To clean up your code before giving GitHub Copilot another try, complete the following steps:

- Cancel the suggested `foreach` ( code completion.
- Delete the partial `foreach` ( statement that you entered.
- Delete the `Console.WriteLine` statement from your `QuarterlySalesReport` method.

Now you should be ready to try GitHub Copilot again.

8. Ensure that your `QuarterlySalesReport` method looks similar to the following code:

```
C#  
  
public void QuarterlySalesReport(SalesData[] salesData)  
{  
  
  
}
```

9. Position the cursor on a blank code line inside the `QuarterlySalesReport` method, and then press Enter.

It may take a moment for GitHub Copilot to generate the suggested code completion.

10. Take a minute to review the suggested code completions.

#### Important

The code completions that you receive are likely to be different from the suggestions shown in the following screenshot. Although GitHub Copilot only has a method name and parameter to work with, that may be enough to generate useful suggestions. You should see suggestions that calculate sales by quarter. Rejecting the suggestions and trying again can provide different results.

```
public void QuarterlySalesReport(SalesData[] salesData)
{
    // create a dictionary to store the quarterly sales data
    Dictionary<string, double> quarterlySales = new Dictionary<string, double>();

    // iterate through the sales data
    foreach (SalesData data in salesData)
    {
        // calculate the total sales for each quarter
        string quarter = GetQuarter(data.dateSold.Month);
        double sales = data.quantitySold * data.unitPrice;

        if (quarterlySales.ContainsKey(quarter))
        {
            quarterlySales[quarter] += sales;
        }
        else
        {
            quarterlySales.Add(quarter, sales);
        }
    }

    // print the quarterly sales report
    Console.WriteLine("Quarterly Sales Report");
    Console.WriteLine("=====");
    foreach (KeyValuePair<string, double> entry in quarterlySales)
    {
        Console.WriteLine(entry.Key + ": $" + entry.Value);
    }
}
```

You can cycle through the suggestions by selecting > or <.

Notice that the suggested code completion iterates through the sales data and performs quarterly sales calculations.

11. To accept the code completion suggested, press the Tab key.

The suggested code completion calculates and displays the quarterly income based on sales data.

```
C#

// create a dictionary to store the quarterly sales data
Dictionary<string, double> quarterlySales = new Dictionary<string,
double>();

// iterate through the sales data
foreach (SalesData data in salesData)
{
    // calculate the total sales for each quarter
```

```
string quarter = GetQuarter(data.dateSold.Month);
double totalSales = data.quantitySold * data.unitPrice;

if (quarterlySales.ContainsKey(quarter))
{
    quarterlySales[quarter] += totalSales;
}
else
{
    quarterlySales.Add(quarter, totalSales);
}

// display the quarterly sales report
Console.WriteLine("Quarterly Sales Report");
Console.WriteLine("-----");
foreach (KeyValuePair<string, double> quarter in quarterlySales)
{
    Console.WriteLine(entry.Key + ": $" + entry.Value);
}
```

12. Notice that the `GetQuarter` method uses the sale's month to determine the sale's quarter.

The `GetQuarter` method is created next.

13. Create two blank code lines below the `QuarterlySalesReport` method.
14. Notice that GitHub Copilot suggests a code completion for the `GetQuarter` method.

With the context provided by the `QuarterlySalesReport` method, GitHub Copilot can easily generate a code completion for the `GetQuarter` method that determines the quarter based on the month of the sale.

15. Take a minute to review the suggested code line completion for the `GetQuarter` method.

```

public void QuarterlySalesReport(SalesData[] salesData)
{
    // create a dictionary to store the quarterly sales data
    Dictionary<string, double> quarterlySales = new Dictionary<string, double>();

    // iterate through the sales data
    foreach (SalesData data in salesData)
    {
        // calculate the total sales for each quarter
        string quarter = GetQuarter(data.dateSold.Month);
        double totalSales = data.quantitySold * data.unitPrice;

        if (quarterlySales.ContainsKey(quarter))
        {
            quarterlySales[quarter] += totalSales;
        }
        else
        {
            quarterlySales.Add(quarter, totalSales);
        }
    }

    // display the quarterly sales report
    Console.WriteLine("Quarterly Sales Report");
    Console.WriteLine("-----");
    foreach (KeyValuePair<string, double> quarter in quarterlySales)
    {
        Console.WriteLine("{0}: ${1}", quarter.Key, quarter.Value);
    }
}

```

< 1/1 > Accept Tab Accept Word Ctrl + RightArrow ...

```

public string GetQuarter(int month)
{
    if (month >= 1 && month <= 3)
    {
        return "Q1";
    }
    else if (month >= 4 && month <= 6)
    {
        return "Q2";
    }
    else if (month >= 7 && month <= 9)
    {
        return "Q3";
    }
    else
    {
        return "Q4";
    }
}

```

16. To accept the code completion suggested, press the Tab key.

C#

```
public string GetQuarter(int month)
{
    if (month >= 1 && month <= 3)
    {
        return "Q1";
    }
    else if (month >= 4 && month <= 6)
    {
        return "Q2";
    }
    else if (month >= 7 && month <= 9)
    {
        return "Q3";
    }
    else
    {
        return "Q4";
    }
}
```

17. Notice that the `Main` method needs to be completed before you can run the code.

You can use the comments in the `Main` method to update your code.

18. Position the cursor at the end of the `// call the GenerateSalesData method` code comment, and then press `Enter`.

GitHub Copilot uses the comment to propose a calling statement for the method.

19. Review and then accept the code completion suggested by GitHub Copilot.

20. Repeat the process for the `// call the QuarterlySalesReport method` code comment.

21. Your `Main` method should contain the following code:

C#

```
static void Main(string[] args)
{
    // create a new instance of the class
    QuarterlyIncomeReport report = new QuarterlyIncomeReport();

    // call the GenerateSalesData method
    SalesData[] salesData = report.GenerateSalesData();

    // call the QuarterlySalesReport method
    report.QuarterlySalesReport(salesData);
}
```

22. Take a minute to review the code in your `QuarterlyIncomeReport` class.

C#

```
namespace ReportGenerator
{
    class QuarterlyIncomeReport
    {
        static void Main(string[] args)
        {
            // create a new instance of the class
            QuarterlyIncomeReport report = new
QuarterlyIncomeReport();

            // call the GenerateSalesData method
            SalesData[] salesData = report.GenerateSalesData();

            // call the QuarterlySalesReport method
            report.QuarterlySalesReport(salesData);
        }

        /* public struct SalesData includes the following fields: date
sold, department name, product ID, quantity sold, unit price */
        public struct SalesData
        {
            public DateOnly dateSold;
            public string departmentName;
            public int productID;
            public int quantitySold;
            public double unitPrice;
        }

        /* the GenerateSalesData method returns 1000 SalesData
records. It assigns random values to each field of the data structure
*/
        public SalesData[] GenerateSalesData()
        {
            SalesData[] salesData = new SalesData[1000];
            Random random = new Random();

            for (int i = 0; i < 1000; i++)
            {
                salesData[i].dateSold = new DateOnly(2023,
random.Next(1, 13), random.Next(1, 29));
                salesData[i].departmentName = "Department " + ran-
dom.Next(1, 11);
                salesData[i].productID = random.Next(1, 101);
                salesData[i].quantitySold = random.Next(1, 101);
                salesData[i].unitPrice = random.NextDouble() * 100;
            }

            return salesData;
        }
    }
}
```

```
public void QuarterlySalesReport(SalesData[] salesData)
{
    // create a dictionary to store the quarterly sales data
    Dictionary<string, double> quarterlySales = new
Dictionary<string, double>();

    // iterate through the sales data
    foreach (SalesData data in salesData)
    {
        // calculate the total sales for each quarter
        string quarter = GetQuarter(data.dateSold.Month);
        double totalSales = data.quantitySold * data.unit-
Price;

        if (quarterlySales.ContainsKey(quarter))
        {
            quarterlySales[quarter] += totalSales;
        }
        else
        {
            quarterlySales.Add(quarter, totalSales);
        }
    }

    // display the quarterly sales report
    Console.WriteLine("Quarterly Sales Report");
    Console.WriteLine("-----");
    foreach (KeyValuePair<string, double> quarter in quar-
terlySales)
    {
        Console.WriteLine(entry.Key + ": $" + entry.Value);
    }
}

public string GetQuarter(int month)
{
    if (month >= 1 && month <= 3)
    {
        return "Q1";
    }
    else if (month >= 4 && month <= 6)
    {
        return "Q2";
    }
    else if (month >= 7 && month <= 9)
    {
        return "Q3";
    }
    else
    {
        return "Q4";
    }
}
}
```



```
}
```

This code was created, almost entirely, using code line completions generated by GitHub Copilot. However, your review of code suggestions is important, and corrections were required. You should always review the code completions suggested by GitHub Copilot to ensure that the code meets your requirements.

23. To review the report output, run the app.

Open a Terminal window in Visual Studio Code, and then enter the following command:

```
Bash
```

```
dotnet run
```

The output should display the quarterly income report, showing the department name, quarter, and income for each department and quarter represented in the test data.

24. Review the output in the Terminal window.

Although the quarterly results are based on random numeric values, you should see a report that's formatted similar to the following output:

```
Output
```

```
Quarterly Sales Report
```

```
-----  
Q3: $635637.5019563352  
Q4: $672247.315297204  
Q2: $667269.194630603  
Q1: $642769.2700531208
```

## Summary

Code line completions are a powerful feature of GitHub Copilot that can help you generate code quickly and efficiently. By using comments to describe the code you want to generate, you can create data structures, methods, and other code elements with minimal effort. Additionally, GitHub Copilot can generate code line completions based on the code you enter, allowing you to build complex applications with ease.

## Next unit: Generate code using GitHub Copilot Chat

[< Previous](#)[Next >](#)