

Debug in Visual Studio



Before start

- Familiar in Visual Studio
- Understand conditional statements
- Understand loop statements

Lesson Objectives

- Debug in Visual Studio
- Debug Mode vs Release Mode
- #if DEBUG tag
- StackTrace
- Remote debug

Section 1

DEBUG IN VISUAL STUDIO

- ***Debugging*** means to run your code step by step in a debugging tool like Visual Studio, to find the exact point where you made a programming mistake. You then understand what corrections you need to make in your code, and debugging tools often allow you to make temporary changes so you can continue running the program.

Use code sample

1 reference

```
private static int AverageArray(int[] array)
{
    var length = array.Length;
    var sum = 0;
    for (int i = 0; i <= length; i++)
    {
        sum += array[i];
    }

    return sum / length;
}
```

0 references

```
static void Main(string[] args)
{
    Console.WriteLine("Sample of debug in Visual Studio");
    Console.Write("Enter length of array: ");
    var length = Convert.ToInt32(Console.ReadLine());
    int[] array = new int[length];
    for (int i = 0; i < length; i++)
    {
        Console.Write($"array[{i}]= ");
        array[i] = Convert.ToInt32(Console.ReadLine());
    }

    var average = AverageArray(array);
    Console.WriteLine("Average of array: " + average);
    Console.ReadLine();
}
```

Clarify the problem

- Before start debugging, make sure you've identified the problem you're trying to solve:
 - ✓ What did you expect your code to do?
 - ✓ What happened instead?

Examine your assumptions

- Ask yourself to challenge your assumptions:
 - ✓ Does your code contain any typos?
 - ✓ Did you make a change to your code and assume it is unrelated to the problem that you're seeing?
 - ✓ Did you expect an object or variable to contain a certain value (or a certain type of value) that's different from what really happened?
 - ✓ Do you know the intent of the code?

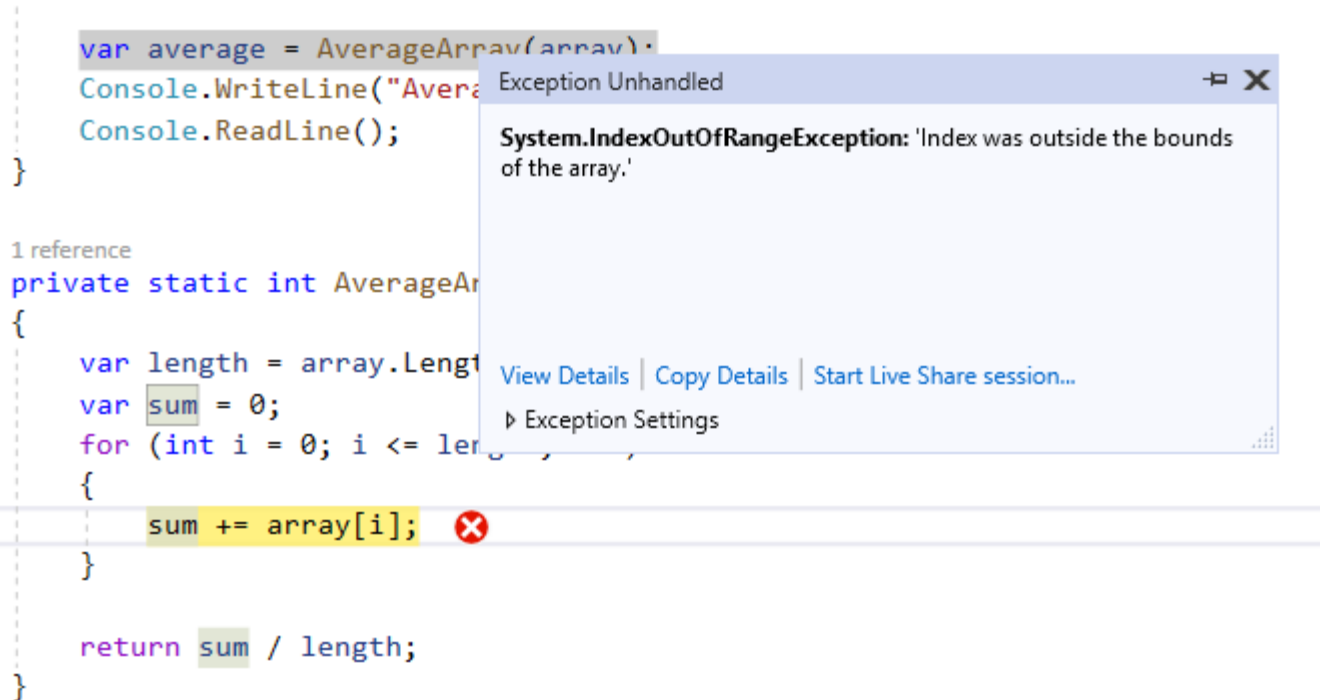
Examine your assumptions

- Check the common error:
 - ✓ Divide by zero
 - ✓ Invalid operation
 - ✓ Index out of range (array)
 - ✓ Wrong format
 - ✓ Null reference
- Clean your code
- Update code following coding conventions

- Running an app within a debugger, also called debugging mode, means that the debugger actively monitors everything that's happening as the program runs.
- It also allows you to pause the app at any point to examine its state, and to then step through your code line by line to watch every detail as it happens.

- *Note: If your solution has multi-projects, set project as StartUp Project*
- To start debugging mode
 - ✓ Press F5
 - ✓ From menu, select Debug > Start Debugging
 - ✓ Right click on the project, choose Debug > Start new instance

Debugging mode

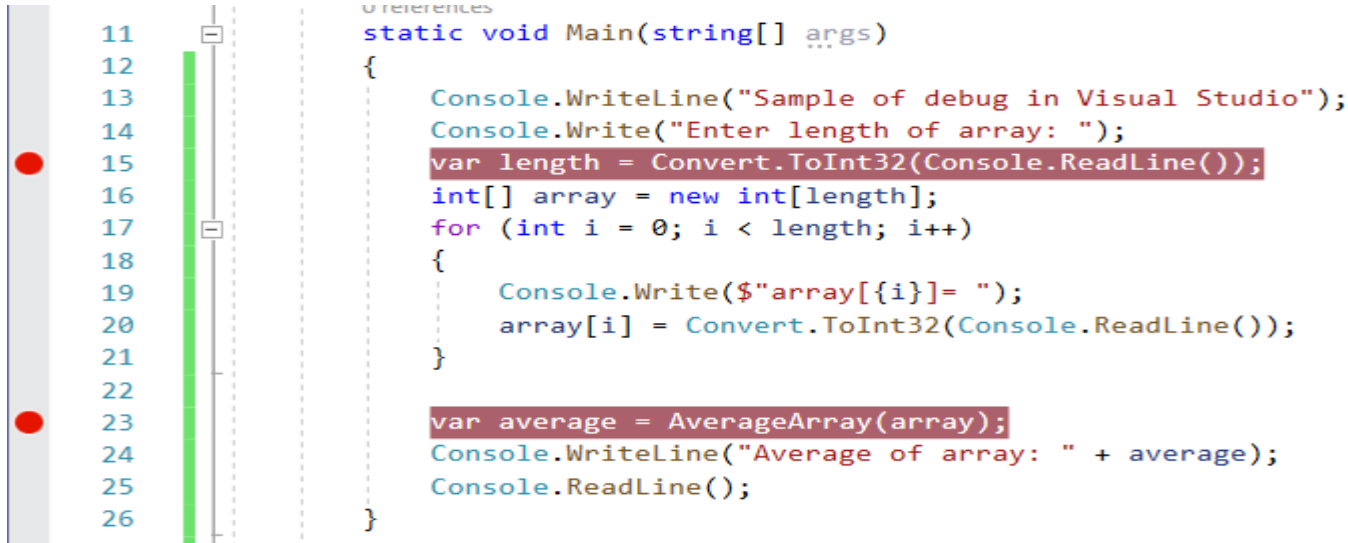


- Breakpoints are the most basic and essential feature of reliable debugging.
- A breakpoint indicates where Visual Studio should suspend your running code so you can take a look at the values of variables, or the behaviour of memory, or whether or not a branch of code is getting run.

- To set breakpoints:
 - ✓ Clicking in the margin to the left of a line of code to set breakpoint(s)
 - ✓ Focus your cursor in line of code then press F9
 - ✓ Focus your cursor in line of code, then from menu, select **Debug > Toggle Breakpoint**
 - ✓ Right-click and select **Breakpoint > Insert breakpoint.**
- To remove breakpoints:
 - ✓ Do the same one more time
 - ✓ Delete all breakpoints: from menu Debug > Delete all breakpoints or Ctrl + Shift + F9

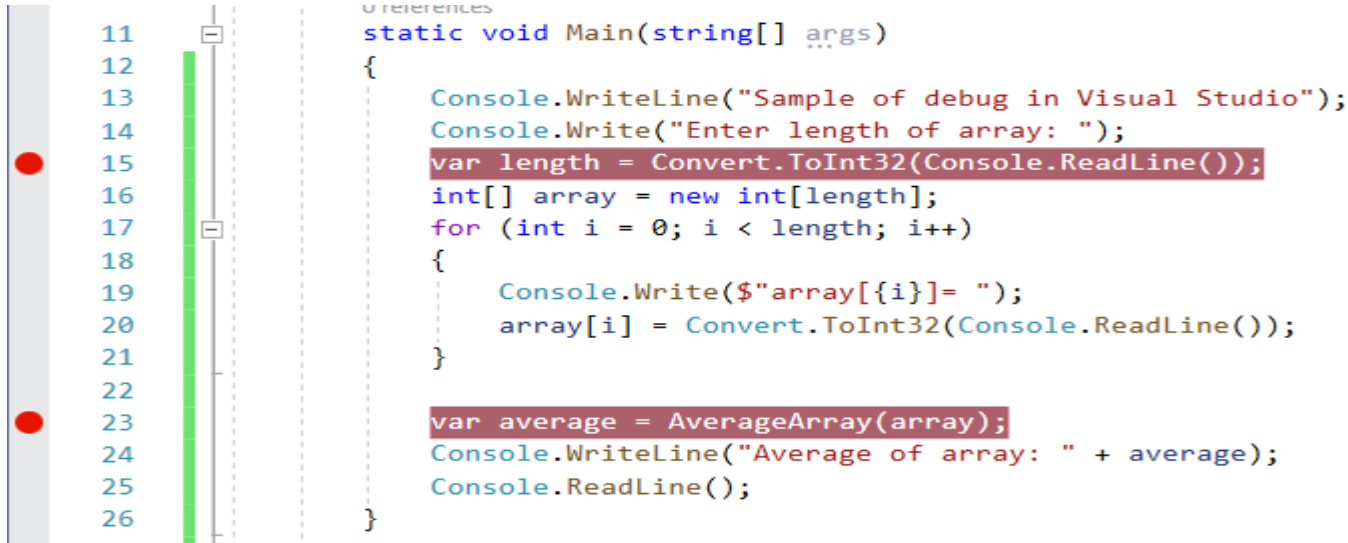
Breakpoints

The breakpoint appears as a red dot in the left margin and current execution lines are automatically highlighted



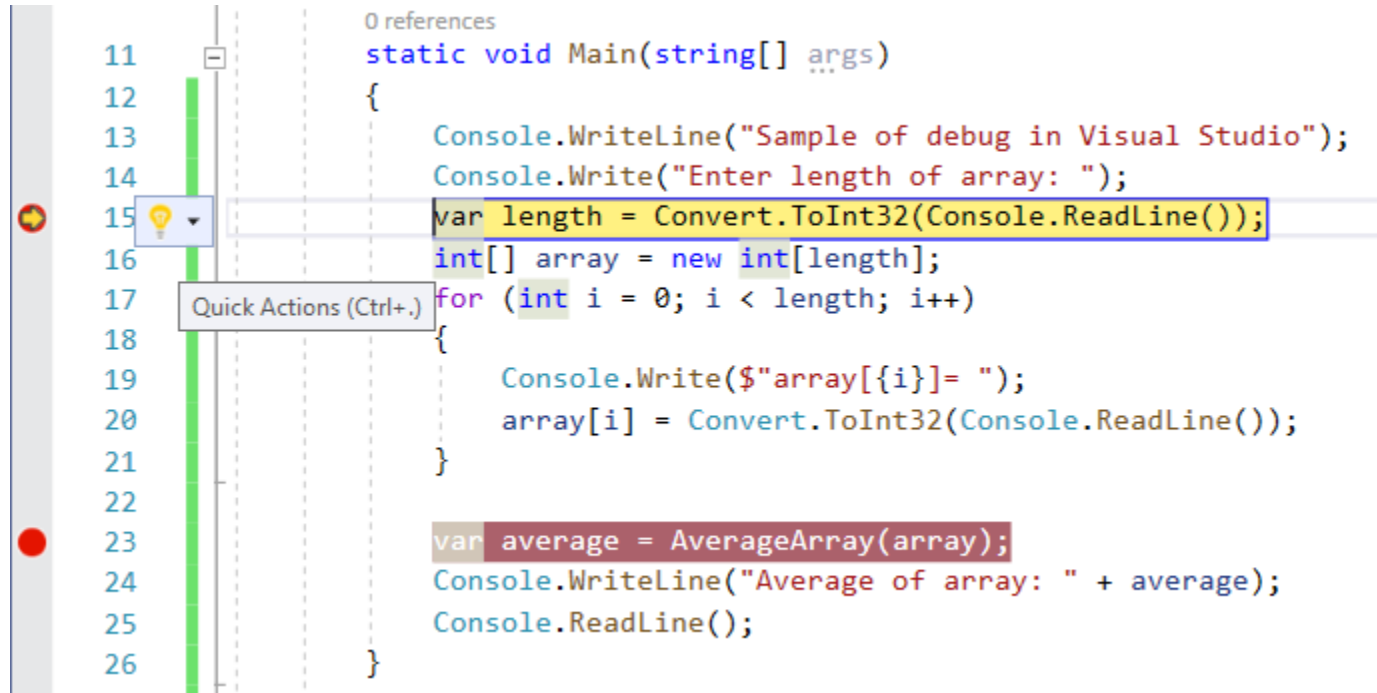
Breakpoints

The breakpoint appears as a red dot in the left margin and current execution lines are automatically highlighted



Breakpoints

When you debug, execution pauses at the breakpoint, before the code on that line is executed. The breakpoint symbol shows a yellow arrow.



- **F11 (Step Into)**
 - ✓ examine the execution flow
- **F10 (Step Over)**
 - ✓ skip over code that you're not interested in
- **Shift + F11 (Step Out)**
 - ✓ resumes app execution (and advances the debugger) until the current function returns.
- **Ctrl + F10 (Run to Cursor)**
 - ✓ quickly set a temporary breakpoint and execute app to the breakpoint

- Inspect data is most important features of the debugger
- Inspect data:
 - ✓ Hover over an object with the mouse and you see its data
 - ✓ Right click then select Add Watch
 - ✓ Right click then select Quick Watch

Inspect data

0 references

```
10 {
11     static void Main(string[] args)
12     {
13         Console.WriteLine("Sample of debug in Visual Studio");
14         Console.Write("Enter length of array: ");
15         var length = Convert.ToInt32(Console.ReadLine());
16         int[] array = new int[length];
17         for (int i = 0; i < length; i++)
18         {
19             Console.Write($"array[{i}]= ");
20             array[i] = Convert.ToInt32(Console.ReadLine());
21         }
22     }
23 }
```

119 % No issues found Ln: 16 Ch: 21 SPC CRLF

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
length	3	int
array	null	int[]

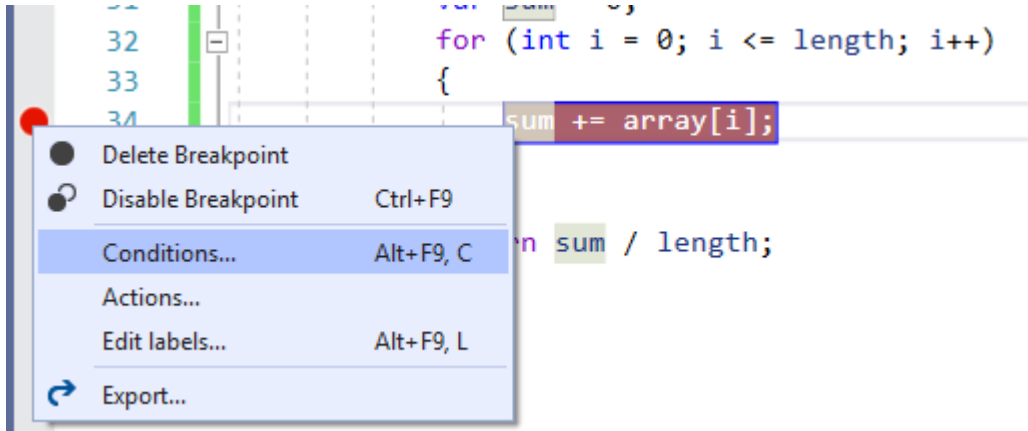
Add item to watch

- The **Immediate** window evaluates expressions by building and using the currently selected project.
 - ✓ evaluate expressions,
 - ✓ execute statements,
 - ✓ print variable values.




Breakpoint conditions

- Used to control when and where a breakpoint executes.
- The condition can be any valid expression that the debugger recognizes.



Breakpoint conditions

- Conditional Expression
- Hit Count
- Filter

 The Conditions option was automatically deselected because there's no condition.

Location: [Program.cs](#), Line: 34, Character: 17, [Must match source](#)

☒ Conditions

[Conditional Expression](#) [Is true](#)

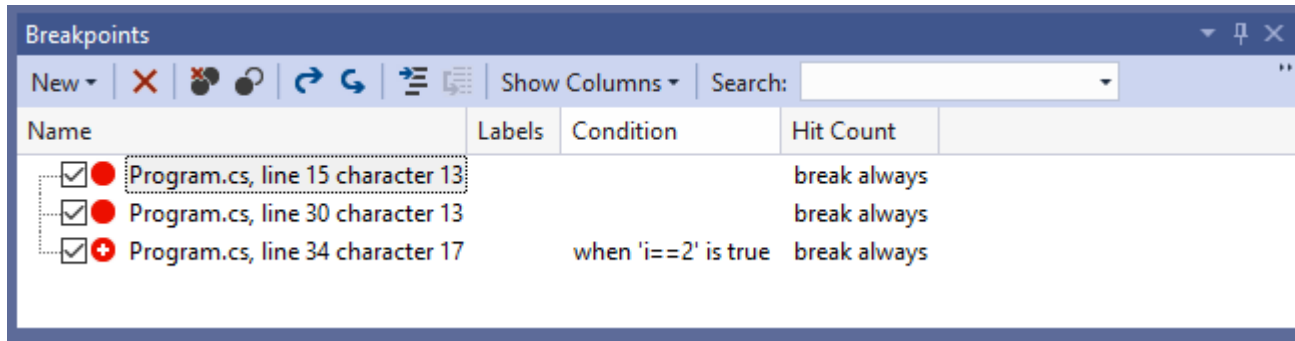
[Add condition](#)

☐ Actions

[Close](#)

Breakpoints window

- In the **Breakpoints** window, you can search, sort, filter, enable/disable, or delete breakpoints.
- You can also set conditions and actions, or add a new function or data breakpoint.



Section 3

DEBUG AND RELEASE CONFIGURATIONS

- In debug configuration, your program compiles with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex.
- The release configuration of your program has no symbolic debug information and is fully optimized.
- You can change build configuration in **Configuration Manager** or from toolbar

Debug and release configurations

Configuration Manager

Active solution configuration: Debug Active solution platform: Any CPU

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
DebugInVisualStudio	Debug	Any CPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>
HelloWorld	Debug	Any CPU	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Close

#if DEBUG tag

#if DEBUG:

///The code in here won't even reach the IL on release.

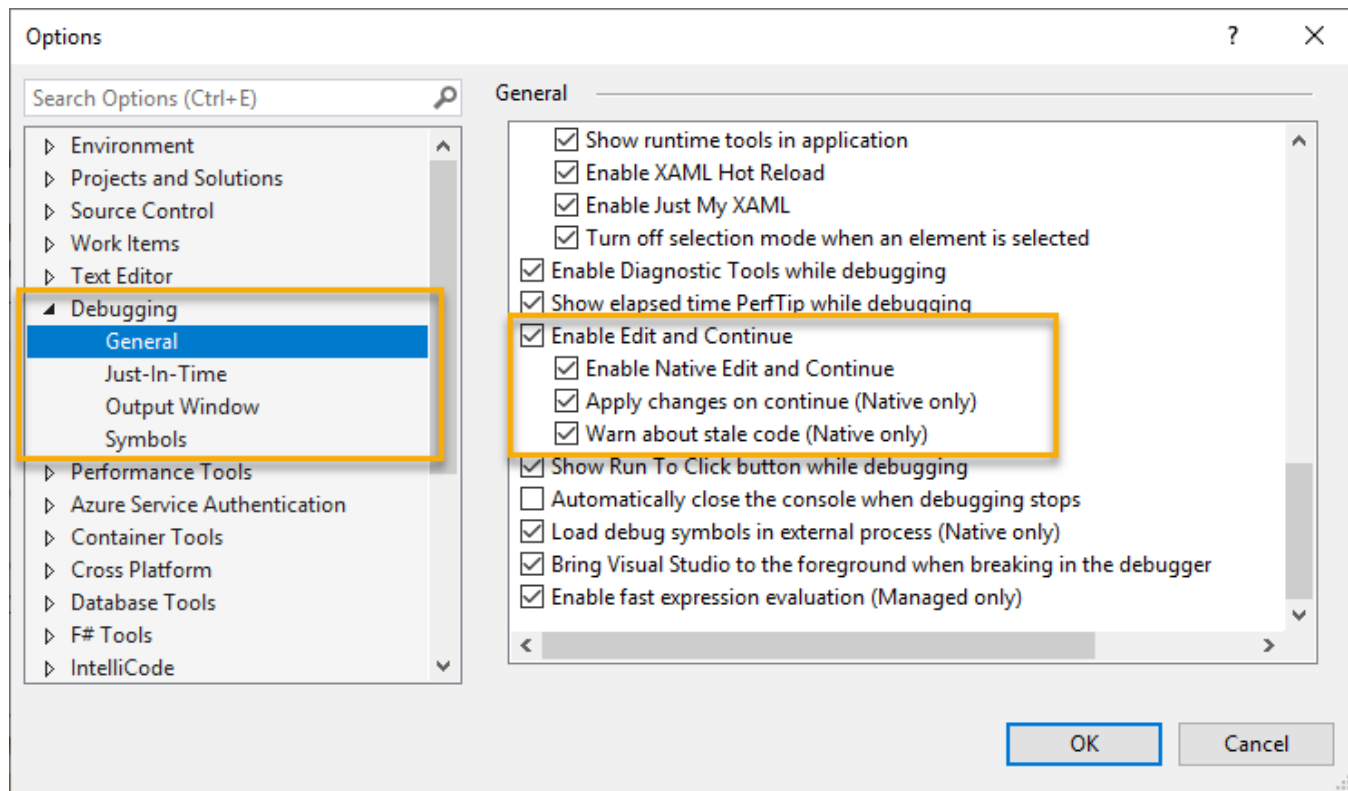
#endif

```
#if DEBUG
    for (int i = 0; i < length; i++)
    {
        Console.WriteLine("Current value of i: " + i);

        Console.Write($"array[{i}]= ");
        array[i] = Convert.ToInt32(Console.ReadLine());
    }
#endif
```

- Edit and Continue is a time-saving feature that enables you to make changes to your source code while your program is in break mode.
- When you resume execution of the program by choosing an execution command like **Continue** or **Step**, Edit and Continue automatically applies the code changes with some limitations.
- This allows you to make changes to your code during a debugging session, instead of having to stop, recompile your entire program, and restart the debugging session.

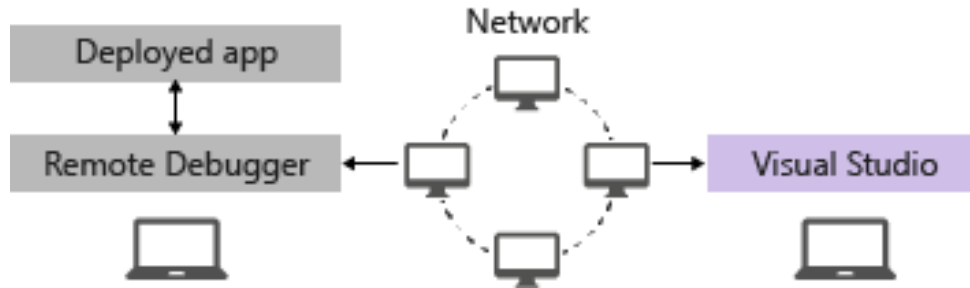
Edit code and continue



Section 4

REMOTE DEBUGGING

- To debug a Visual Studio application that has been deployed on a different computer, install and run the remote tools on the computer where you deployed your app, configure your project to connect to the remote computer from Visual Studio, and then run your app.



- Download and Install the remote tools
- Set up the remote debugger
 - ✓ Account to run
 - ✓ Port
- Debug package on server

Remote Debugging

- Step 1: Attach process
- Step 2: enter Connection target (include address and port)
- Step 3: enter credential
- Step 4: connect and debug

Lesson Summary



Thank you

