CS6004NI
Application Development

samyush

Samyush.maharjan@islingtoncollege.edu.np

C# Method / Function

C# Method / Function

Introduction

A function is called method when it is a part of a object/class. A method/function let us use **name** and **reuse** a chunk of code.

```
returnType MethodName() {
  // method body
}
```

Example:

```
1. class Hello
2. {
3.     void PrintHelloWorld()
4.     {
5.         Console.WriteLine("Hello World!");
6.     }
7. }
```

C# Method / Function

Parameters and Arguments

Parameters are the **variable names** listed in the function's definition and **Arguments** are the **values** passed to the function.

Example:

```
1.  class Program
2.  {
3.      void SayHello(string name) // Method definition
4.      {
5.          Console.WriteLine($"Hello {name}!");
6.      }

7.      static void Main(string[] args)
8.      {
9.          var p = new Program();
10.         p.SayHello("John"); // Method call
11.     }
12. }
```

name is a **Parameter** of *SayHello* method

"John" is an **Argument** to a *SayHello* method call

C# Method / Function

Named Arguments

Named arguments allow us to specify an argument for a parameter by **matching the argument with its name rather than with its position** in the parameter list.

Example:

```
1.  class Program
2.  {
3.      void PrintIntro(string name, string companyName, string jobTitle)
4.      {
5.          Console.WriteLine($"Hi! I'm {name}. I work at {companyName} as a {jobTitle}.");
6.      }

7.      static void Main(string[] args)
8.      {
9.          var p = new Program();
10.         p.PrintIntro("John", "Google", "QA Engineer");
11.         p.PrintIntro("John", jobTitle:"QA Engineer", companyName:"Google"); // Same as above
12.     }
13. }
```

Named Arguments

C# Method / Function

Optional Arguments

Optional arguments allow us to **omit arguments for some parameters**. Both techniques can be used with methods, indexers, constructors, and delegates.

Example:

Optional Arguments

```
1.  class Program
2.  {
3.      void PrintIntro(string name, string companyName="Google", string jobTitle="QA Engineer")
4.      {
5.          Console.WriteLine($"Hi! I'm {name}. I work at {companyName} as a {jobTitle}.");
6.      }

7.      static void Main(string[] args)
8.      {
9.          var p = new Program();
10.         p.PrintIntro("John", "Google", "Software Engineer");
11.         p.PrintIntro("John", jobTitle:"Software Engineer"); // Same as above
12.     }
13. }
```

C# Method / Function

Controlling Parameters

A parameter can be passed in a method one of three ways:

1. By **value** (default): When passing a variable as a parameter by default, **its current value gets passed**, not the variable itself.

```
returnType MethodName(type parameterName)
```

1. By **reference** as a ref parameter: When passing a variable as a ref parameter, **a reference to the variable gets passed** into the method. It requires that the **variable be initialized before it is passed**.

```
returnType MethodName(ref type refParameterName)
```

1. As an **out** parameter: When passing a variable as an out parameter, **a reference to the variable gets passed** into the method. It requires that the **a value is assigned before the method returns**.

```
returnType MethodName(out type outParameterName)
```

C# Method / Function

Controlling Parameters

Example:

```
1. void PassingParameters(int x, ref int y, out int z)
2. {
3.     x++;
4.     y++;
5.     z = 30; // must be initialized inside the method
6.     z++;
7. }
8. ...
9. var p = new Program();
10.int a = 10;
11.int b = 20;
12.Console.WriteLine($"Before: a = {a}, b = {b}"); // Before: a = 10, b = 20
13.p.PassingParameters(a, ref b, out int c);
14.Console.WriteLine($"After: a = {a}, b = {b}, c = {c}"); // After: a = 10, b = 21, c = 31
```

C# Method / Function

Are the following statements True or False?

1. “Optional parameters must appear after all required parameters.”
2. “Named arguments must be passed in the correct positional order.”
3. “Named arguments can be used after positional arguments.”
4. “A ref or out parameter cannot have a default value.”

C# Method / Function

Are the following statements True or False?

1. “Optional parameters must appear after all required parameters.” **=> True**
2. “Named arguments must be passed in the correct positional order.” **=> False**
3. “Named arguments can be used after positional arguments.” **=> True**
4. “A ref or out parameter cannot have a default value.” **=> True**

C# Method / Function

Returning Value

A **return** value allows a **method to produce a result** when it completes.

Example:

```
1. string GetFullName(string firstName, string lastName)
2. {
3.     if (lastName == "")
4.         return firstName;
5.     if (firstName == "")
6.         return lastName;

7.     var fullName = $"{firstName} {lastName}";
8.     return fullName;
9. }
10. ...
11. var p = new Program();
12. string fullName = p.GetFullName("John", "Doe");
13. Console.WriteLine(fullName); // John Doe
```

C# Method / Function

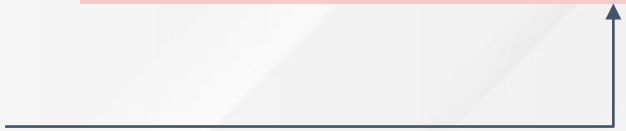
Throwing Exceptions

The other side of Try-catch, creating and throwing new exceptions.

Example:

```
1. string GetFullName(string firstName, string lastName)
2. {
3.     if (firstName == null || lastName == null)
4.         throw new Exception("I can't deal with null!");
5.     if (lastName == "")
6.     ...
7. }
8. ...
9. var p = new Program();
10.string fullName = p.GetFullName("John", null);
```

```
Unhandled exception. System.Exception: I can't deal with null!
   at Program.GetFullName(String firstName, String lastName)
   at Program.Main(String[] args)
Command terminated by signal 6
```



C# Method / Function

Throwing Exceptions

Re-throwing an Exceptions Example:

```
1.  string GetFullName(string firstName, string lastName)
2.  {
3.      if (firstName == null || lastName == null)
4.          throw new Exception("I can't deal with null!");
5.      if (lastName == "")
6.  ...
7.  try
8.  {
9.      var p = new Program();
10.     string fullName = p.GetFullName("John", null);
11.     Console.WriteLine(fullName);
12. }
13. catch (Exception ex)
14. {
15.     Console.WriteLine($"Error Message = {ex.Message}"); // Error Message = I can't deal with null!
16.     throw; // Re-throws the Exception
17. }
```

C# Method / Function

Common Exception Types

Exception Name	Meaning
NotImplementedException	The programmer hasn't written this code yet.
NotSupportedException	I will never be able to do this.
InvalidOperationException	I can't do this in my current state, but I might be able to in another state.
ArgumentOutOfRangeException	This argument was too big (too small, etc.) for me to use.
ArgumentNullException	This argument was null, and I can't work with a null value.
ArgumentException	Something is wrong with one of your arguments.
Exception	Something went wrong, but I don't have any real info about it.

C# Method / Function

Local Function

Local functions are nested in another method or function. They can only be called from their containing method/function.

Example:

```
1.  class Program
2.  {
3.      static void Main(string[] args)
4.      {
5.          int sum = Add(5, 10); // Local function call
6.          int product = Multiply(5, 10); // Local lambda function call
7.          Console.WriteLine($"The sum is {sum} and the product is {product}."); // The sum is 15 and the product is 50.

8.          int Add(int x, int y) // Local function
9.          {
10.             return x + y;
11.         }

12.         int Multiply(int x, int y) => x * y; // Local lambda function
13.     }
14. }
```

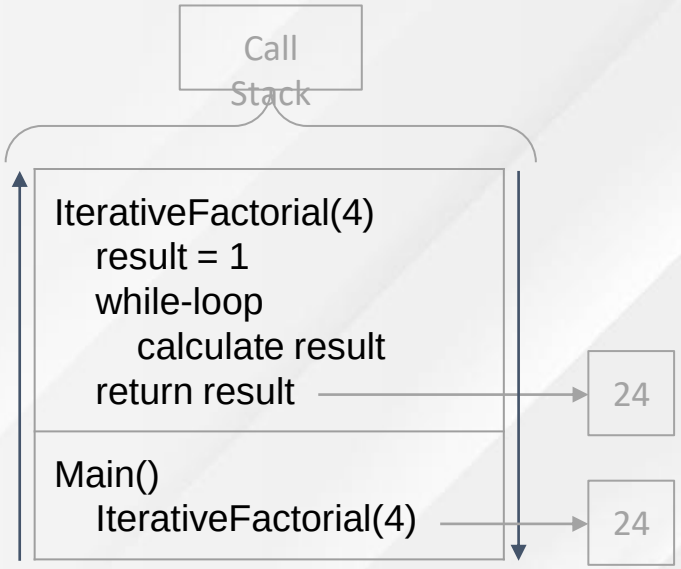
C# Method / Function

Recursion

Recursion is when a method calls itself.

Factorial Iterative Example:

```
1.  int IterativeFactorial(int n)
2.  {
3.      var result = 1;
4.      while (n > 0)
5.      {
6.          result *= n;
7.          n--;
8.      }
9.      return result;
10. }
11. ...
12. var p = new Program();
13. p.IterativeFactorial(4); // 4! = 4 × 3 × 2 × 1 = 24
```



C# Method / Function

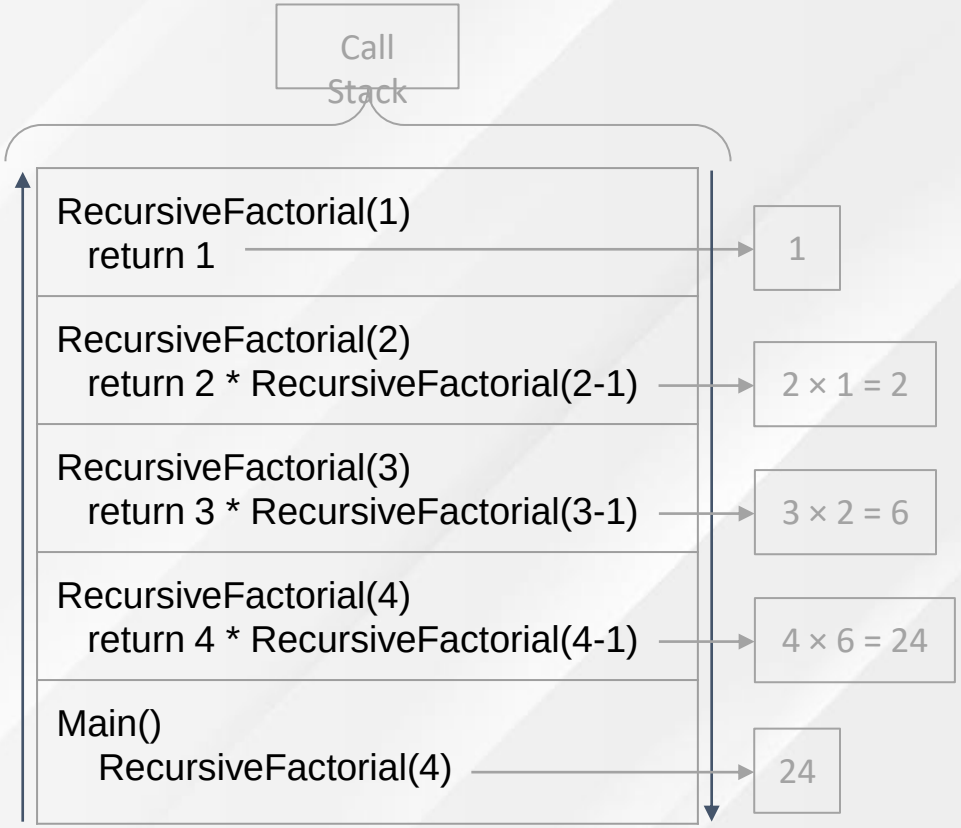
Recursion

Factorial Recursive Example:

```
1. int RecursiveFactorial(int n)
2. {
3.     if (n < 2)
4.         return 1;
5.     return n * RecursiveFactorial(n - 1);
6. }
7. ...
8. var p = new Program();
9. p.RecursiveFactorial(4); // 4! = 4 × 3 × 2 × 1 = 24
```

Using lambda expression with ternary operator:

```
int Factorial(int n) => n < 2 ? 1 : n * Factorial(n - 1);
```



C# Method / Function

Are the following statements True or False?

1. “A function can have only one return statement.”
2. “A function must return a value.”
3. “Unhandled exception terminates the program.”
4. “ The base case condition is optional in a recursive function.”

C# Method / Function

Are the following statements True or False?

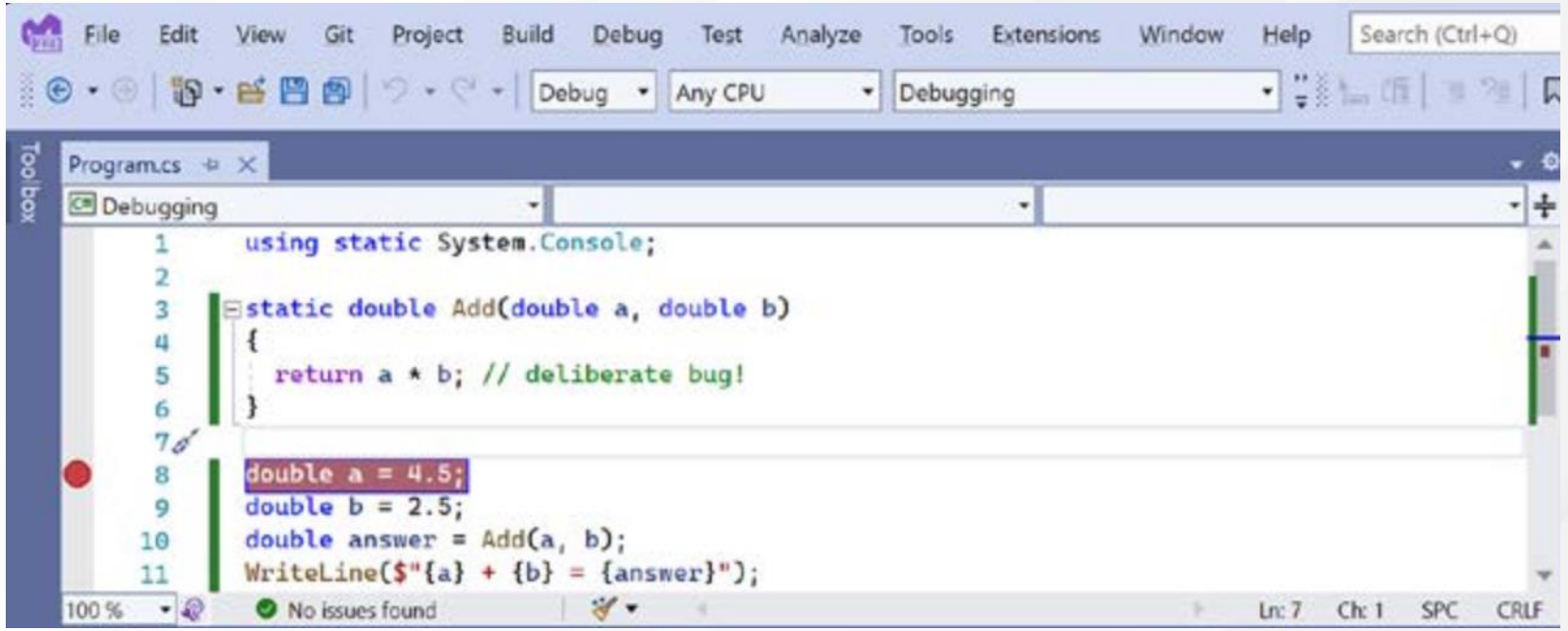
1. “A function can have only one return statement.” **=> False**
2. “A function must return a value.” **=> False**
3. “Unhandled exception terminates the program.” **=> True**
4. “ The base case condition is optional in a recursive function.” **=> False**

Debugging during Development

C# Debugging during Development

Using Visual Studio 2022

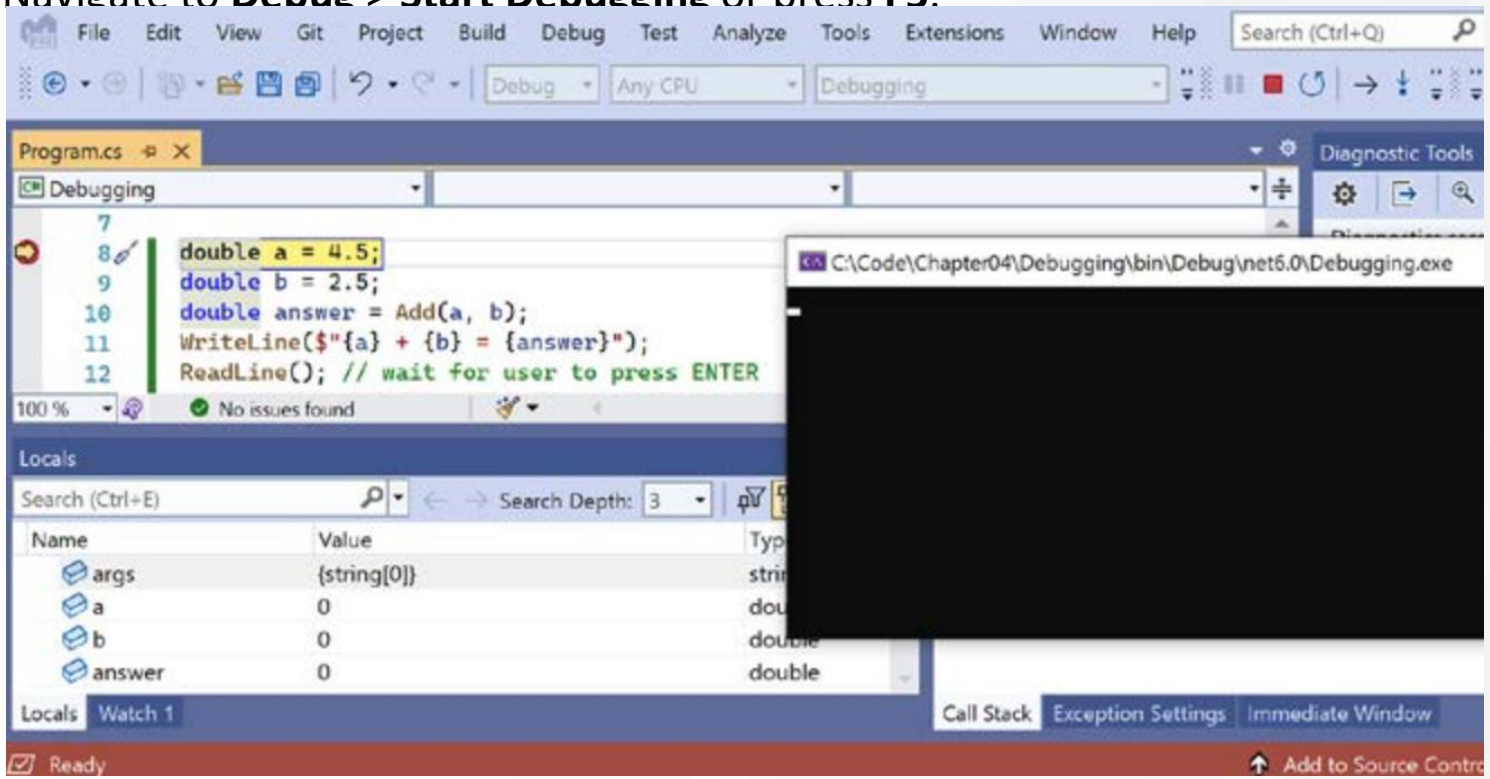
1. Click in the statement that declares the variable.
2. Navigate to **Debug > Toggle Breakpoint** or press **F9**.



C# Debugging during Development

Using Visual Studio 2022

3. Navigate to **Debug > Start Debugging** or press **F5**.

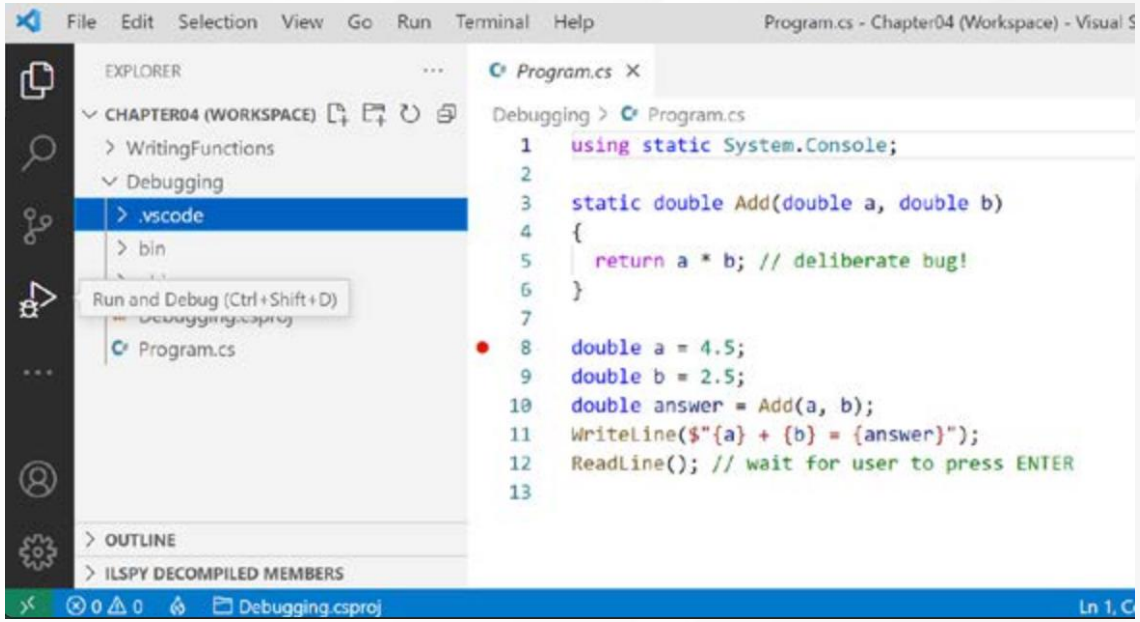


C# Debugging during Development

Using Visual Studio Code

For setting up Visual Studio Code for debugging follow the instructions from here.

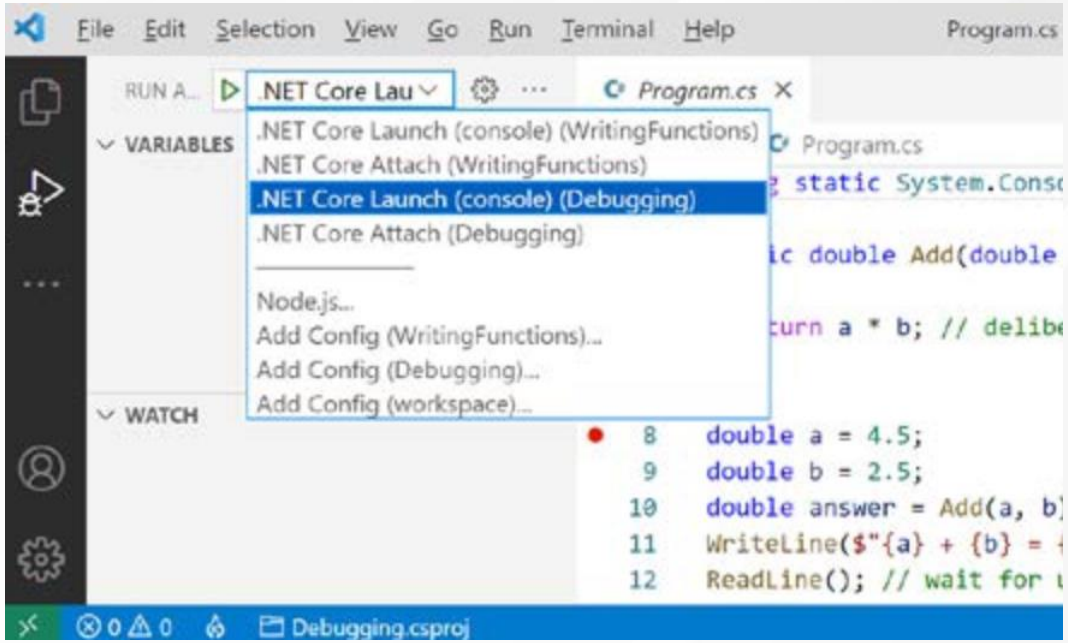
1. Click in the statement that declares the variable.
2. Navigate to **Run > Toggle Breakpoint** or press **F9**.



C# Debugging during Development

Using Visual Studio Code

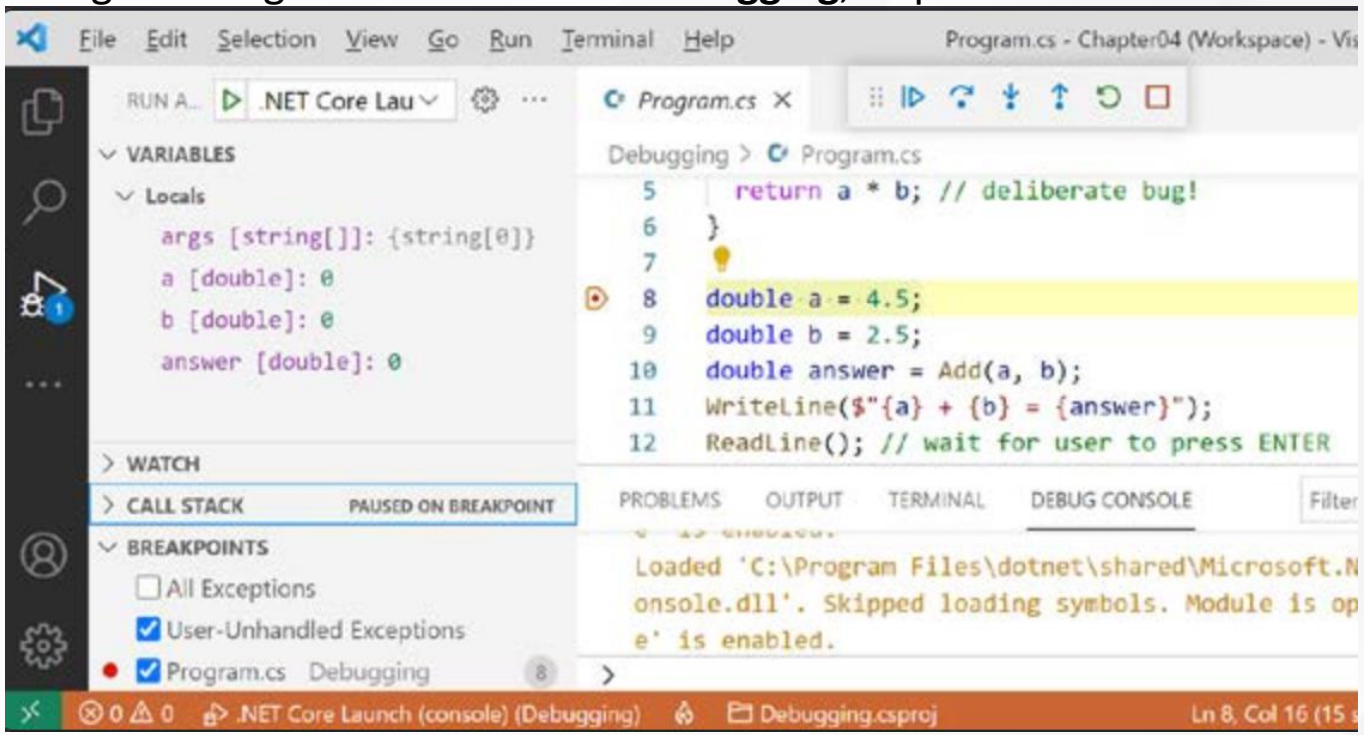
3. Navigate to **View** > **Run**.
4. At the top of the DEBUG window, click on the dropdown to the right of the **Start Debugging** button.
5. Select **.NET Core Launch (console) (Debugging)**.



C# Debugging during Development

Using Visual Studio Code

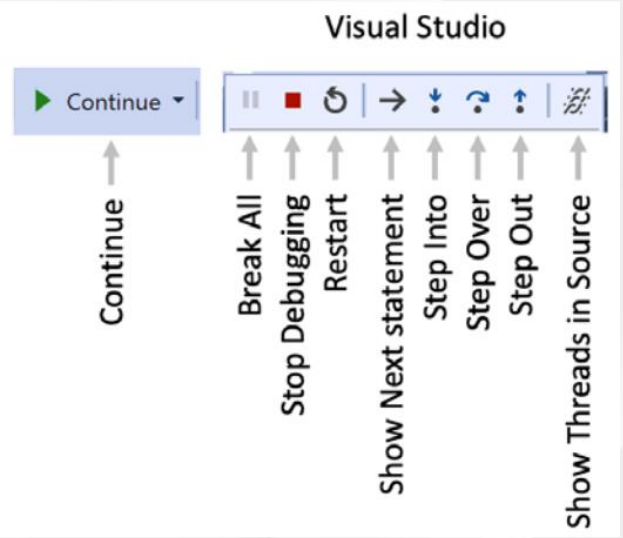
6. Navigate navigate to **Run** > **Start Debugging**, or press **F5**.

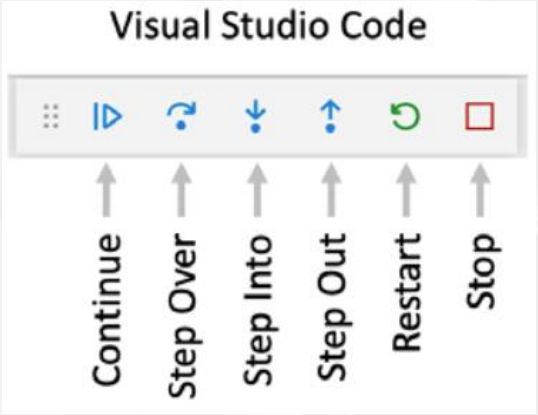


C# Debugging during Development

Debugging Toolbar

- **Continue**/*F5*: continue running the program from the current position until it ends or hits another breakpoint.
- **Step Over**/*F10*: the whole method is executed in one go and then suspends execution at the first line of code after the called function returns.
- **Step Into**/*F11*: steps into the method and allows to step through every line in that method.
    - **Step Out**/*Shift + F11*: continues running code and suspends execution when the current function returns.
- **Restart**/*Ctrl or Cmd + Shift + F5*: This button will stop and then immediately restart the program with the debugger attached again.
- **Stop**/*Shift + F5* (red square): This button will stop the debugging session.

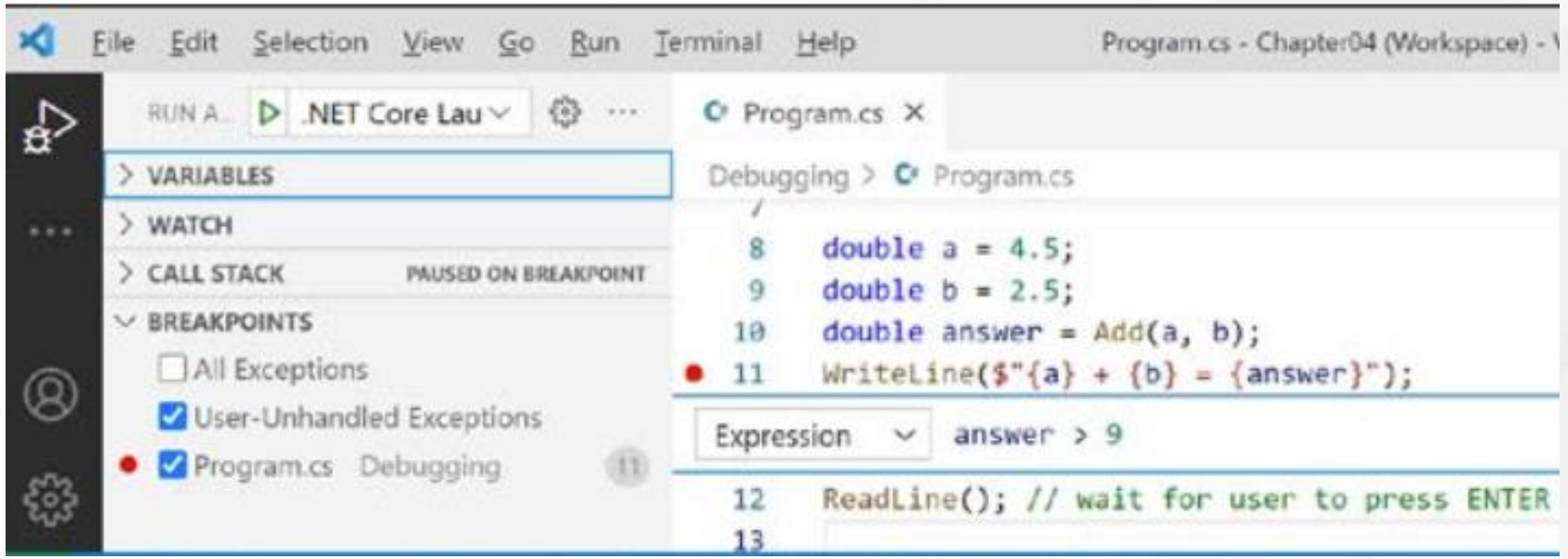




C# Debugging during Development

Customizing Breakpoints

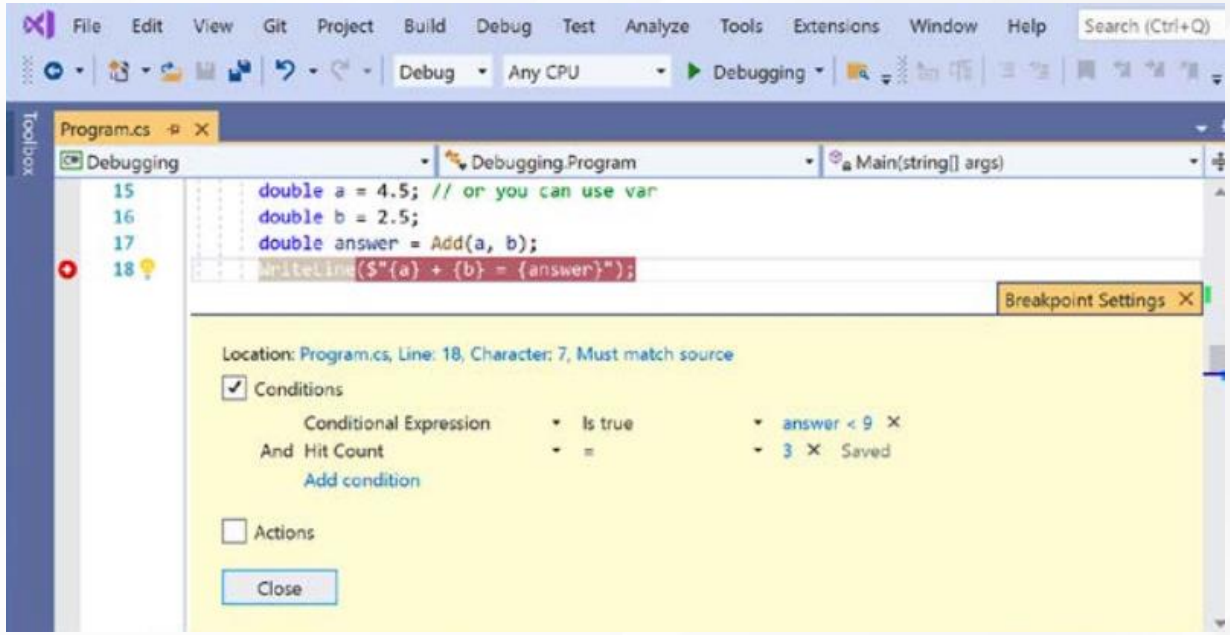
- Right-click the **breakpoint** and choose **Edit Breakpoint / Conditions**.
- Enter an **expression** which evaluate to a **boolean** value, such as the **answer** variable must be **greater than 9**.
- The break point is activated only when the **expression** which evaluate to **true**.



C# Debugging during Development

Customizing Breakpoints

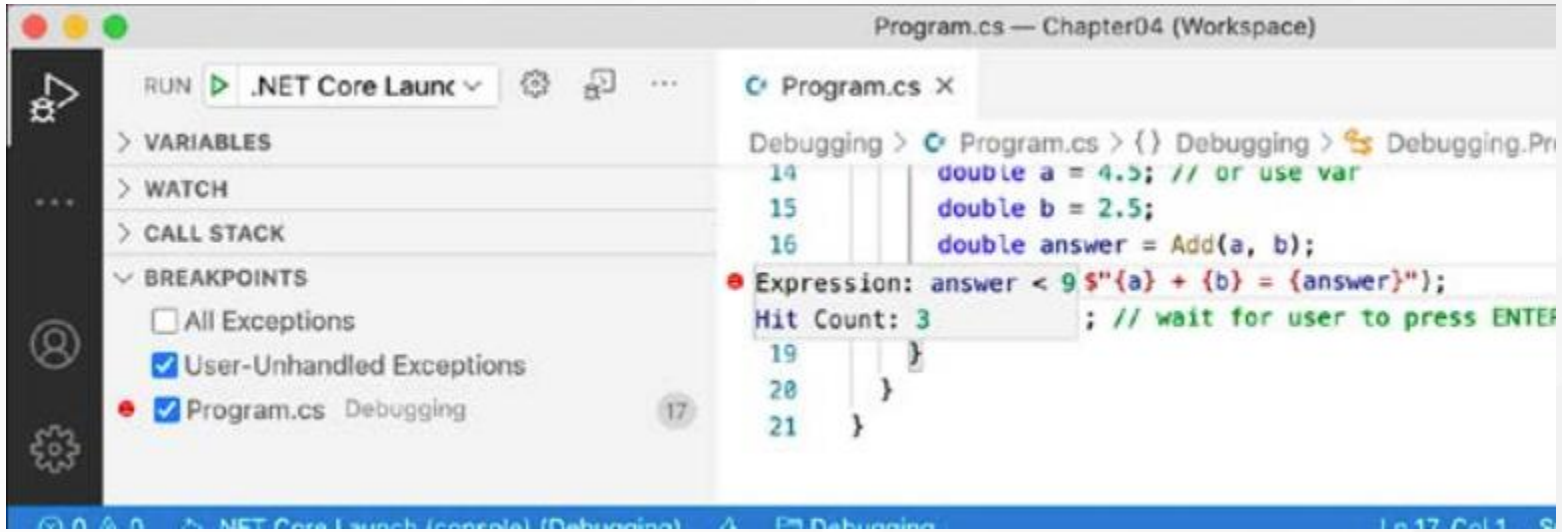
- Edit the breakpoint or its conditions.
- Click **Add condition** in Visual Studio or select **Hit Count** in Visual Studio Code.
- Then enter a number such as 3, meaning that you would have to hit the breakpoint three times before it activates.



C# Debugging during Development

Customizing Breakpoints

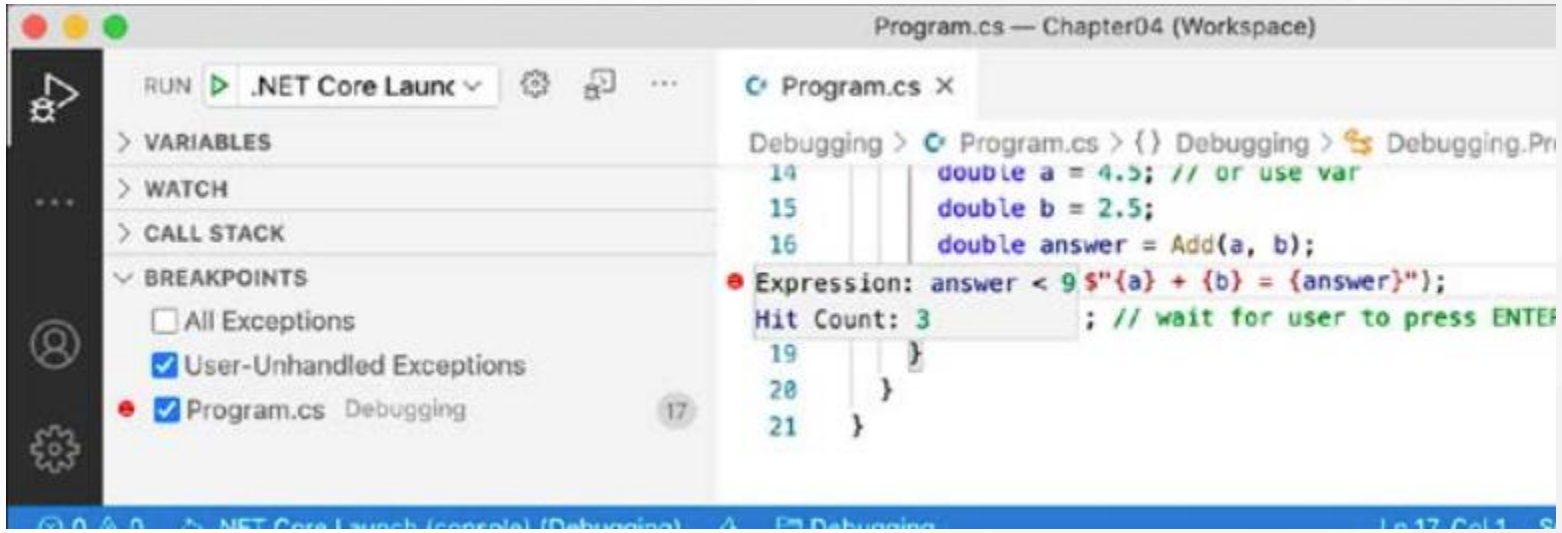
- Hovering mouse over breakpoint (red circle) would show a summary of **Expression** and **Hit Count**.



C# Debugging during Development

Customizing Breakpoints

- Hovering mouse over breakpoint (red circle) would show a summary of **Expression** and **Hit Count**.



Questions?