UNIT 1
Walkthrough Visual Studio Part 2
(Debugging Basics)

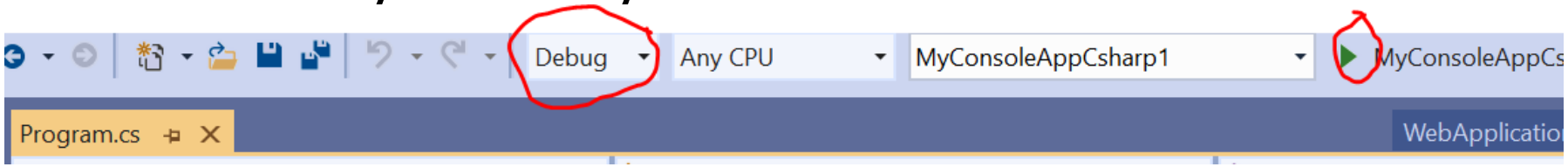
Presented By : Aditya Joshi
Asst. Professor
Department of Computer Application
Graphic Era Deemed to be University
Dated : 27-01-2021

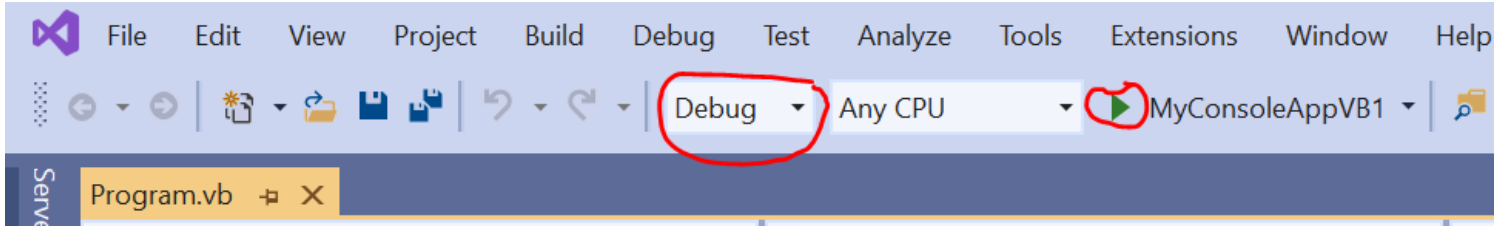
Debugging

- Debugging means removing bugs from your code.(finding and fixing bugs)
- You might debug your code by scanning your code or by code analyser.
- Your may debug your code using visual studio debugger.
- A debugger is a very specialized developer tool that attaches to your running app and allows you to inspect your code.

Debug mode vs. running your app

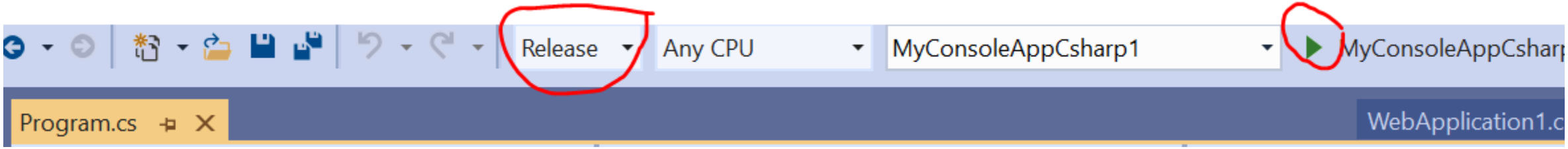
- When you run your app in Visual Studio for the first time, you may start it by pressing the green arrow button Start Debugging in the toolbar (or F5). By default, the Debug value appears in the drop-down to the left. If you are new to Visual Studio, this can leave the impression that debugging your app has something to do with running your app--which it does--but these are fundamentally two very different tasks.





- A **Debug** value indicates a debug configuration. When you start the app (press the green arrow or **F5**) in a debug configuration, you start the app in *debug mode*, which means you are running your app with a debugger attached. This enables a full set of debugging features that you can use to help find bugs in your app.

If you have a project open, choose the drop-down selector where it says **Debug** and choose **Release** instead.



When you switch this setting, you change your project from a debug configuration to a release configuration. Visual Studio projects have separate release and debug configurations for your program. You build the debug version for debugging and the release version for the final release distribution. A release build is optimized for performance, but a debug build is better for debugging.

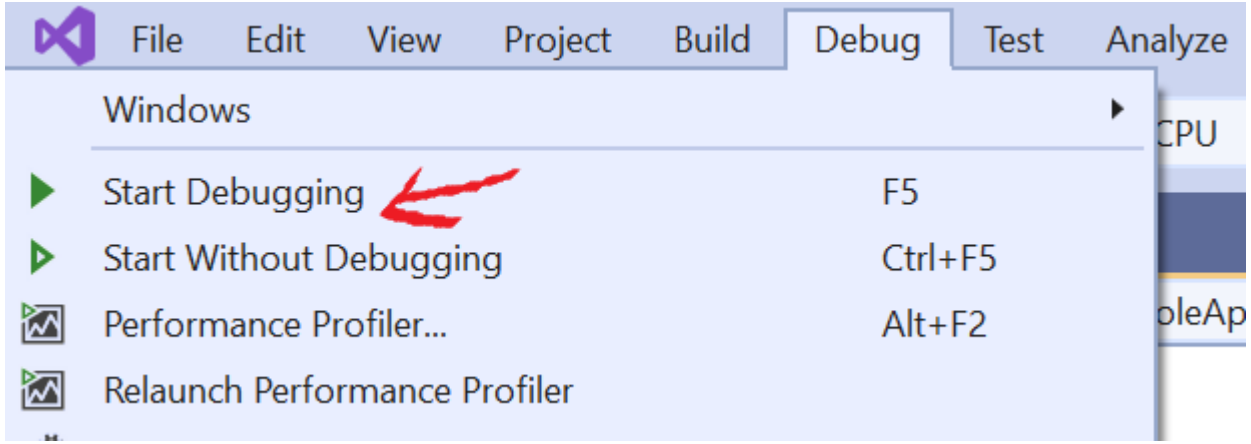
Agenda

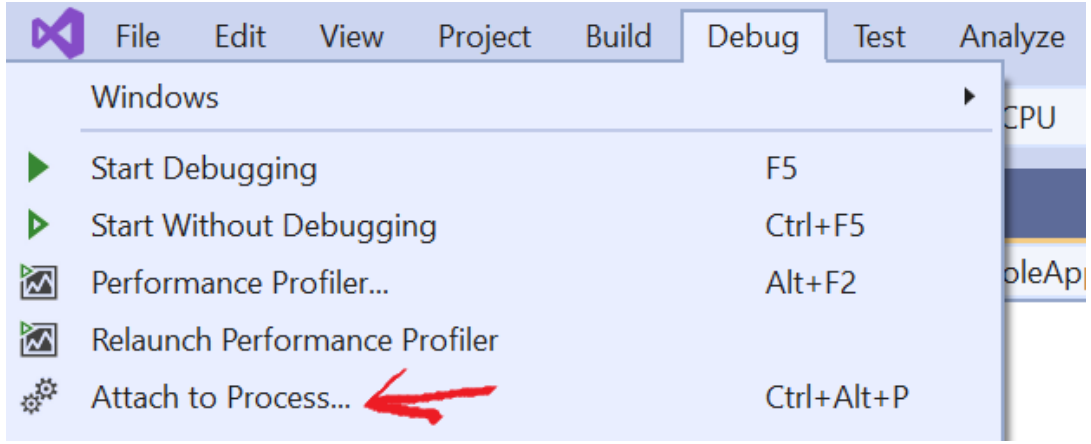
- Start Debugging
- Break Points
- Code Navigations
- Investigating Variables
- DataTip and QuickWatch Features
- BreakPoint Advanced Features

1. Attaching the debugger

Debugging in Visual Studio occurs automatically when you run from Visual Studio with F5 or select **Debug | Start Debugging**.

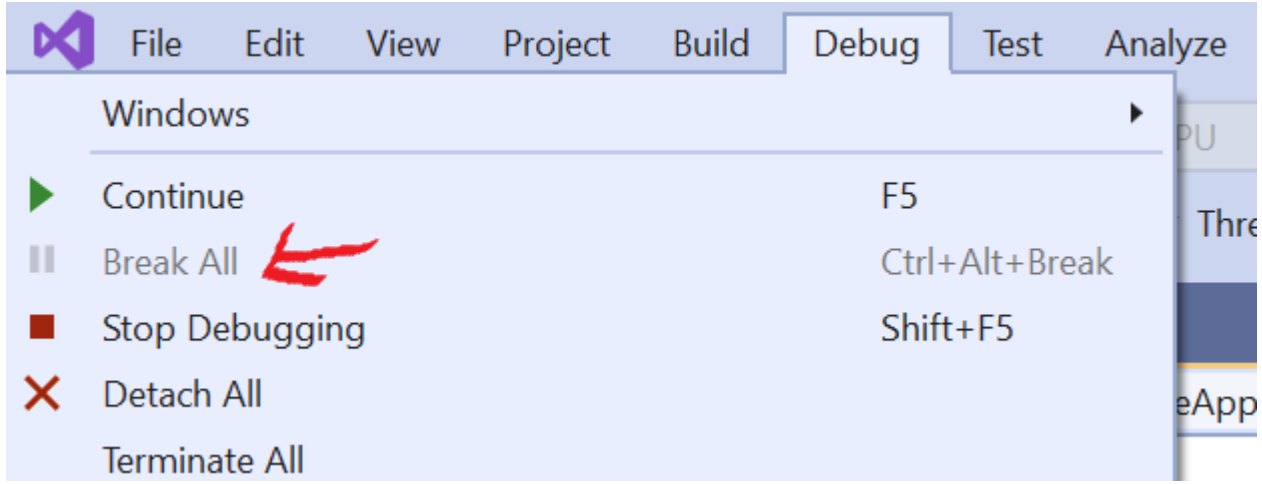
When doing, so Visual Studio attached itself as a debugger to the program. Alternatively, you can **attach to a running process** with **Debug | Attach to Process...** (Ctrl+Alt+P).





2. Debugger Break Mode

- The debugged process has to be in "Break Mode" for debugging. That means the program is currently paused on a specific line of code. More accurately, all the running threads are paused on a specific line of code.
- You can get to break mode with **Debug | Break All** menu item (Ctrl+Alt+Break) or by placing breakpoints.



We're usually going to use breakpoints because in most debugging scenarios we will want to debug when the program reaches a certain line of code.

You can place breakpoints by clicking on the margin, pressing F9, or **Debug | Toggle Breakpoint**. Once set, when your program reaches that line of code, Visual Studio's debugger will stop execution and enter "Break Mode":

```
6   {
    0 references
7   static void Main(stri
8   {
9       int i, fact = 1,
10      Console.Write("Er
11      number = int.Pars
12      for (i = 1; i <=
13      {
```

```
9       int i, fact = 1, number;
10      Console.Write("Enter any Number: ");
11      number = int.Parse(Console.ReadLine());   ≤13
12      for (i = 1; i <= number; i++)
13      {
14          fact = fact * i;
```

```
    0 references
4   Sub main()
5       Dim n, i, f As Int
6       Console.Write("Ent
7       n = CInt(Console.F
```

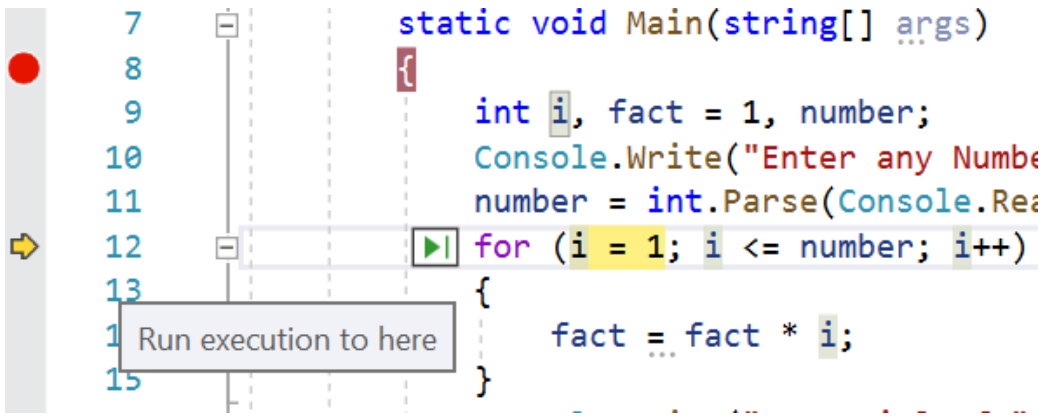
```
6       Console.Write("Enter a Number: ")
7       n = CInt(Console.ReadLine())   ≤10ms elapsed
8       f = 1
```

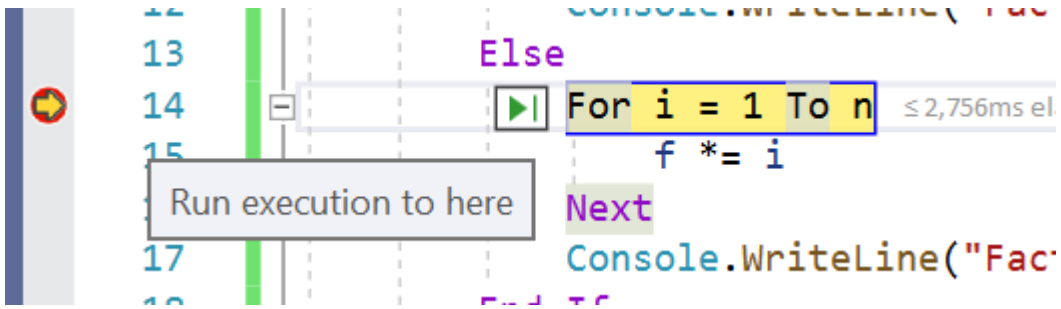
3. While in Break Mode – Navigate through code

- When in break mode, you can debug interactively and see how your execution of code progresses. The basic features of code navigation are:

1. **Continue** (F5) will quit break mode and continue the program's execution until the next breakpoint is hit, entering break-mode again.
2. **Step Over** (F10) will execute the current line and break on the next line of code.
3. **Step Into** (F11) is used when the next execution line is a method or a property. When clicked, the debugger will step into the method's code first line. By default, properties are skipped. To enable stepping into property code go to **Tools | Options | Debugging** and uncheck **Step over properties and operators**.
4. **Run execution to here** allows you to continue execution, and break in a specified location without a breakpoint. It's like creating a breakpoint and removing it after first break. You can do it in 3 ways:

I. Hover and click on the green arrow that appears on the start of each line





I. Stand on the desired line of code and click Ctrl + F10

II. Right click on the desired line of code and click on **Set next statement**

5. Run to a cursor location allows you to forcefully set the next line of code to execute. The current (yellow) line will **not** be executed. It can be a line that was executed before or after, but it's best for the new line of code to stay **in the current scope**. There are 2 ways to do this:

- Drag the yellow arrow to any line of code
- Stand on the desired line of code and click Ctrl+Shift+F10

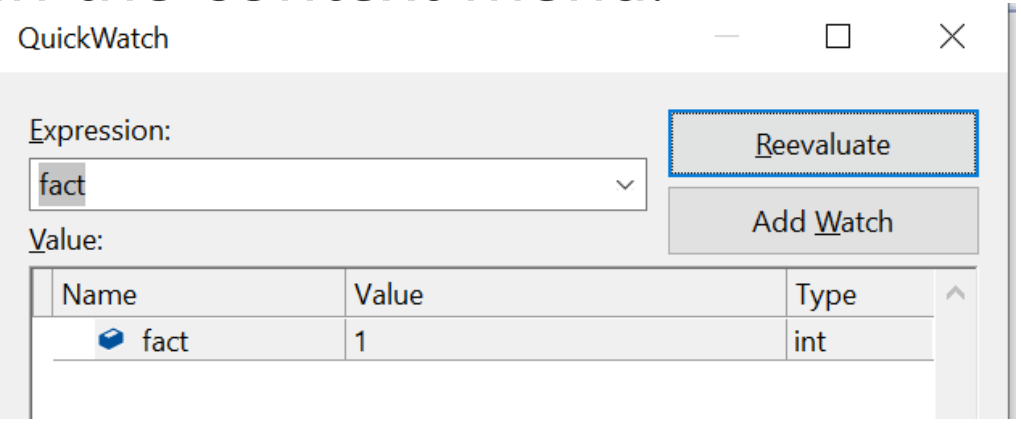
4. Investigate variables

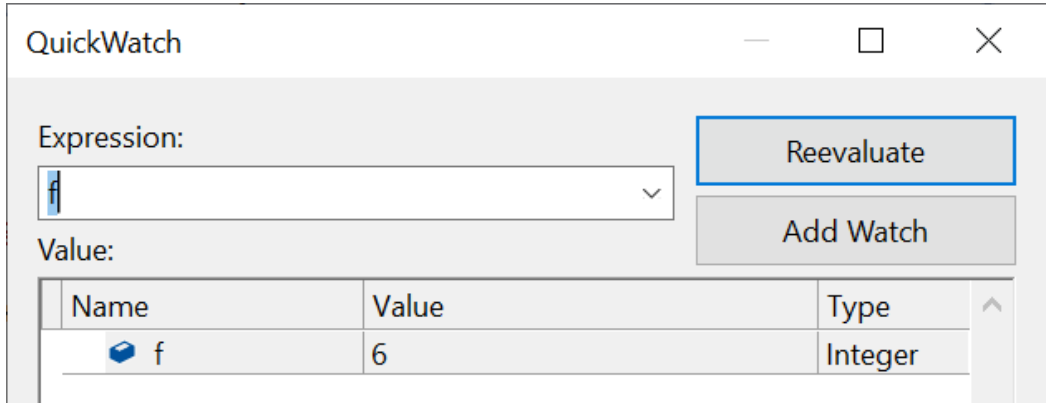
- When in break mode, you can investigate the value of **local variables** and **class members**. This is as easy as hovering over a variable:

```
12  for (i = 1; i <= number; i++)
13  {
14      fact = fact * i;   ≤1ms elapsed
15  }                      fact | 1
16  Console.Write("Factorial of "
```

```
13  Else
14      For i = 1 To n
15          f *= i   ≤1ms elap
16      Next                f | 2
17      Console.WriteLine
```

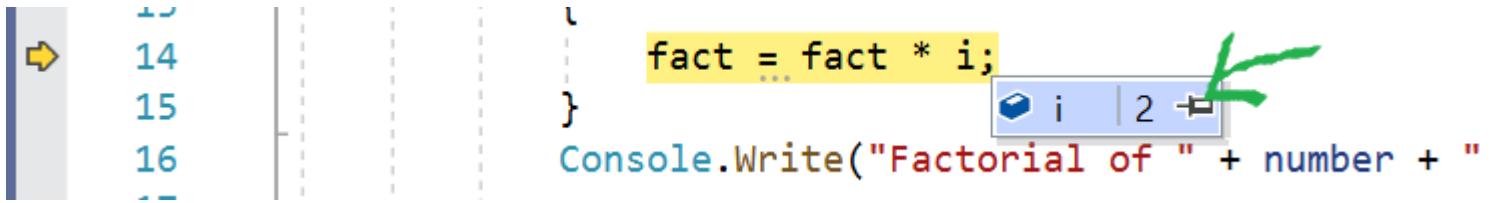
The hover popup is called a **Data Tip**. You can get to the same popup by right-clicking the variable and select **QuickWatch** (Shift+F9 or Ctrl+D, Q) in the context menu.

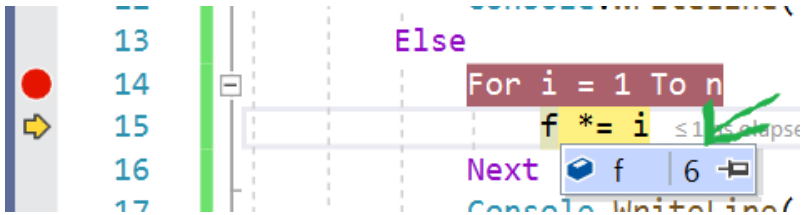




5. DataTip and QuickWatch notable Features

- The DataTip and QuickWatch have several useful features:
- **Pinning DataTips** – You can leave a DataTip pinned to the editor by clicking on the pin icon. Useful when you hit the same breakpoint many times (maybe in a loop)



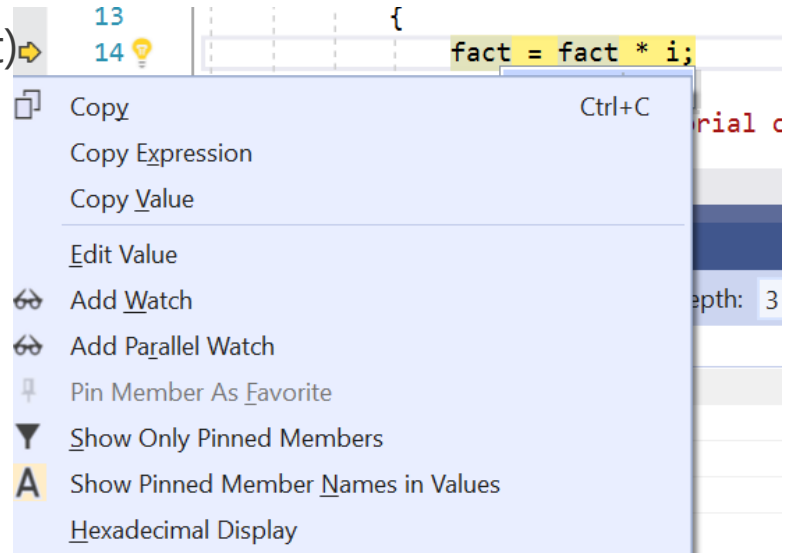


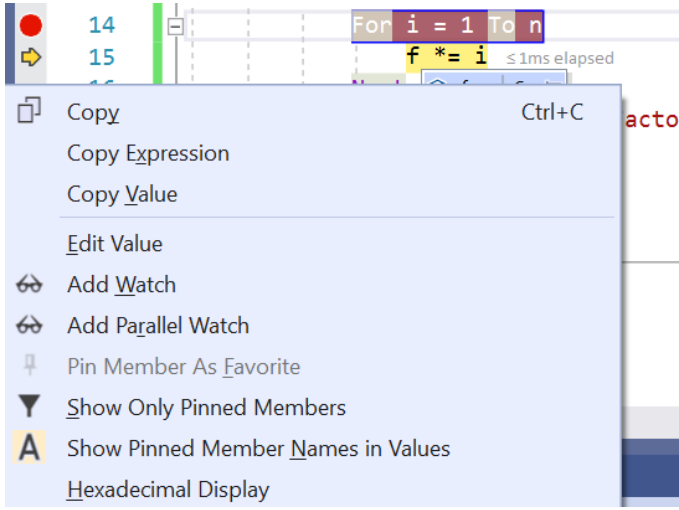
- Holding **Ctrl** will make the DataTip **transparent**
- By right clicking on an expression in the DataTip, you can open a context menu with several options:

Copy – Copies to clipboard both Expression and Value(fact =1)

Copy Expression – Copies expression(fact)

Copy Value – Copies Value(1)





•Edit Value – A useful feature where you can change the value during debugging. Most useful for primitives (strings, integers, etc.).

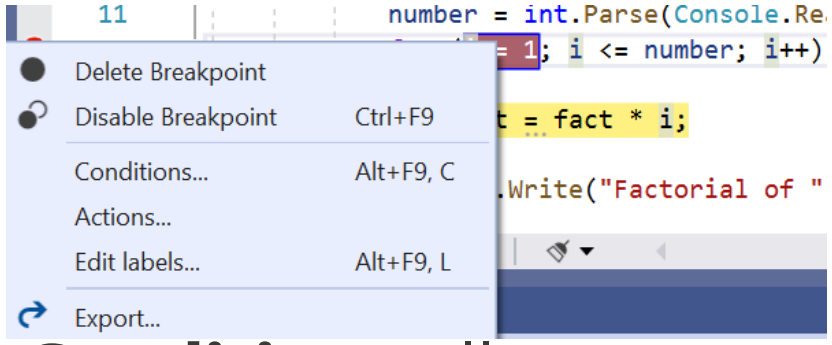
•Add Watch – Adds expression to Watch window (more on that later)

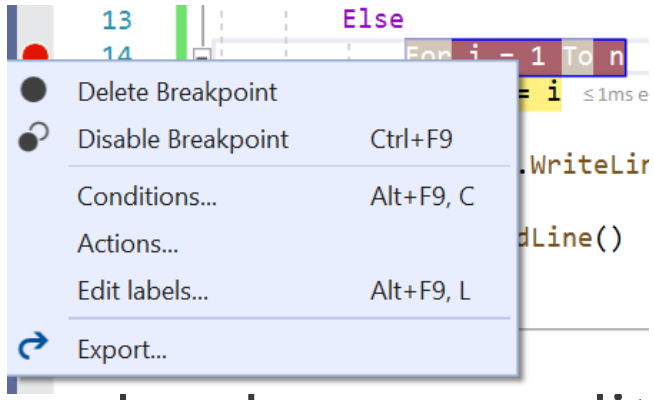
•Add Parallel Watch – Adds expression to Parallel Watch window (more on that later)

•Make Object ID – Creates a scope-independent expression that starts with '$' ($1, $2, and so on). You can evaluate this expression at any time, regardless of the current scope, in QuickWatch, Watch Window or the Immediate Window. A very useful feature and can be used to detect memory leaks.

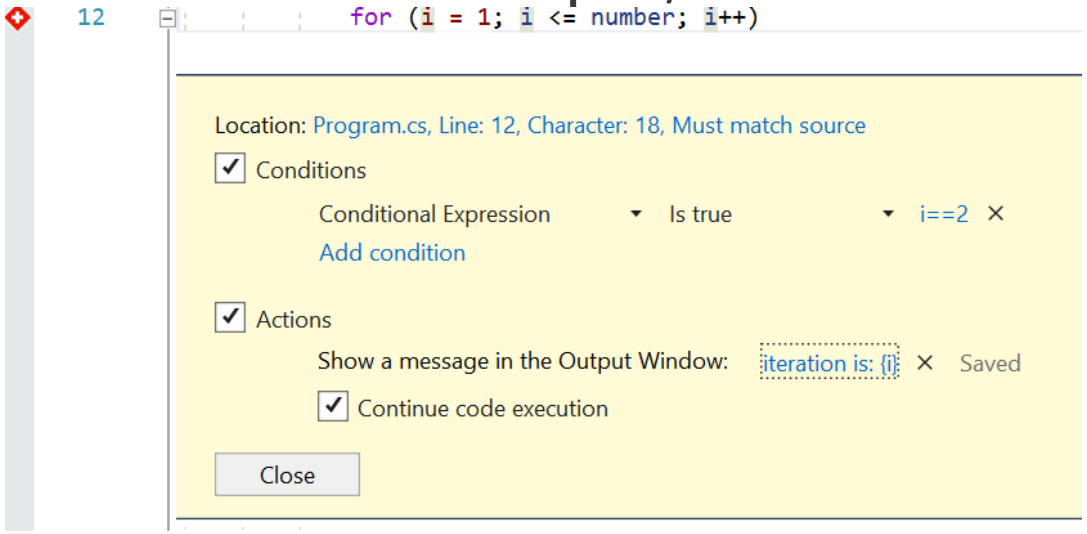
6. Breakpoint advanced features

- The breakpoints has several very useful lesser known features. By right-clicking a break you will see a context menu:





- **Conditions** allows you to break on this breakpoint only when a condition is met. For example, when in a loop I can break only on even numbers:



This is most useful when debugging **multi-threaded scenarios**. More on multi-threading debugging later.

- **Edit labels...** allows to categorize breakpoints into labels. This makes it easier later to organize them in the breakpoints tool window: