

LESSON 02 - OUTPUTTING TEXT AND NUMBERS, & ERROR HANDLING



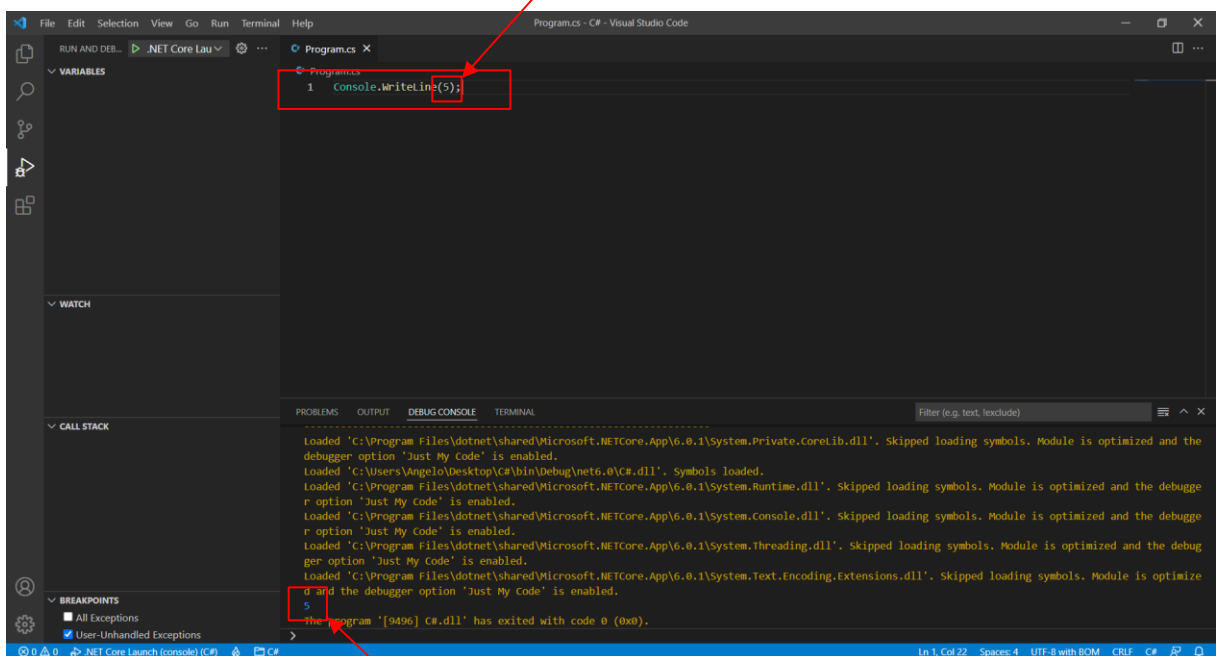
In this lesson we will demonstrate the difference between text (strings) and numbers, outputting to the console, and discuss building and basic errors you may encounter while programming with Visual Studio.

Sections:

- I. [YOUR COMPUTER IS A CALCULATOR \(STRINGS & NUMBERS\)..... PAGE 1](#)
- II. [MIXING STRINGS AND NUMBERS WITH 'Console.WriteLine\(\)' & 'Console.Write\(\)' PAGE 3](#)
- III. [BUILDING YOUR PROGRAM \(BUILD ERRORS, SYNTAX ERRORS & RUNTIME ERRORS\) .. PAGE 5](#)

I. YOUR COMPUTER IS A CALCULATOR (STRINGS & NUMBERS):

Let us start a new program (Program.cs) and output the number '5' to the debug console. And this time we will not use quotation marks " ":



Line 1 now reads:

```
Console.WriteLine(5);
```

The above line of code will still output a '5' to the console, however without the use of quotations. Let us see what happens if we put the '5' inside quotations " ":

```
Console.WriteLine("5");
```

Output will still be:

5

So, what is the difference? Let's use another example to explain:

```
Console.WriteLine(5 + 5);
```

Output will be:

10

Our program now **calculates** 5 plus 5. However, if we put '5 + 5' inside quotations " ":

```
Console.WriteLine("5 + 5");
```

Output will be:

5 + 5

This is because the compiler sees quotation marks and it thinks that you are trying to output text and not a number. Text in computer science is referred to as a **string**. Therefore, use double quotes (" ") when you wish to output a **string** (text) and **no quotations** when you wish to output **numbers/calculations**.

Notice how C# will calculate numbers like a calculator. We outputted '5 + 5' (without quotations) and the compiler calculated the output of '10' to the console. In fact, we can run any calculation we wish with the mathematical symbols on our keyboard. The following table shows the symbols on a keyboard associated with each **mathematical operation**:

Mathematical Operation	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/

Here are some examples using the above operations:

```
Console.WriteLine(5 + 5);
Console.WriteLine(10 - 3);
Console.WriteLine(8 / 2);
Console.WriteLine(3 * 2);
```

Output will be:

10
7
4
6

II. MIXING STRINGS AND NUMBERS WITH 'Console.WriteLine()' & 'Console.Write()':

You can also mix strings and numbers with the same Console.WriteLine() statement, for example:

```
Console.WriteLine("5 + 5 = " + 10);
```

Output will be:

5 + 5 = 10

Notice how we use a **plus sign '+'** to output the strings and numbers together. However, you need to be careful when doing this, for example:

```
Console.WriteLine("5 + 5 = " + 5 + 5);
```

Output will be:

5 + 5 = 55

How come '5 + 5' did not calculate? This is because the Console.WriteLine() statement encountered the text "5 + 5 = " first, so it treats anything after as text as well (even if you are trying to do a calculation). One way around this is to put brackets around '5 + 5':

```
Console.WriteLine("5 + 5 = " + (5 + 5));
```

Output will be:

5 + 5 = 10

Why did '5 + 5' now calculate? Because C# (and all programming languages) follows **BEDMAS (Brackets Exponents Division Multiplication Addition Subtraction)**. Recall from math class that this is the order of operations when doing calculations. Since your computer is a calculator, it also follows BEDMAS. Therefore, Console.WriteLine() will also follow **BEDMAS** and hence will compute the **brackets first** which in this case is the calculation of (5 + 5).

If a **calculation** occurs **before** text in Console.WriteLine() then we don't have to worry about putting brackets, for example:

```
Console.WriteLine(6 + 6 + " Hello " + 6 * 3);
```

Output will be:

12 Hello 18

In this case we did not need the brackets because Console.WriteLine() encountered a calculation first before it encountered the text, therefore it will treat the rest of the calculations properly.

Alternatively, we can use **Console.Write()** which will only output text without automatically going to a new line. For example:

```
Console.Write("5 + 5 = ");  
Console.Write(5 + 5);  
Console.WriteLine();
```

Output will be:

5 + 5 = 10

Note: We must have a blank **Console.WriteLine()** appear as the last statement when using **Console.Write()** statements, or nothing will appear on the screen (this is called flushing the buffer).

Note: You can use blank **Console.WriteLine()** statements to output **blank lines** to the console, for example:

```
Console.WriteLine("Hello World?");  
Console.WriteLine();  
Console.WriteLine();  
Console.WriteLine("How are you?");
```

Output will be:

Hello World!

How are you?

You can mix and match text any way you wish, and the above examples have shown you different techniques. You can even have one long **Console.WriteLine()** statement. For example:

```
Console.WriteLine("5 + 5 = " + (5 + 5) + ", 6 * 6 = " + (6 * 6));
```

Output will be:

5 + 5 = 10, 6 * 6 = 36


The possibilities are endless!

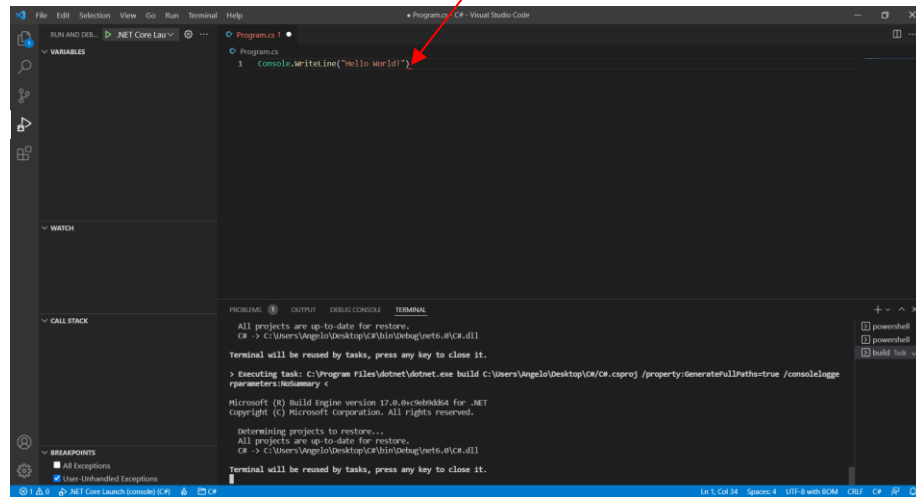
III. BUILDING YOUR PROGRAM (BUILD ERRORS, SYNTAX ERRORS & RUNTIME ERRORS):

When you run your program Visual Studio will first **build** your program. This is the process discussed in Unit 0 that compiles your source code (C#) into machine language that your computer hardware can run. Once your program is built, Visual Studio does not need to build your program again unless you make changes to your code.

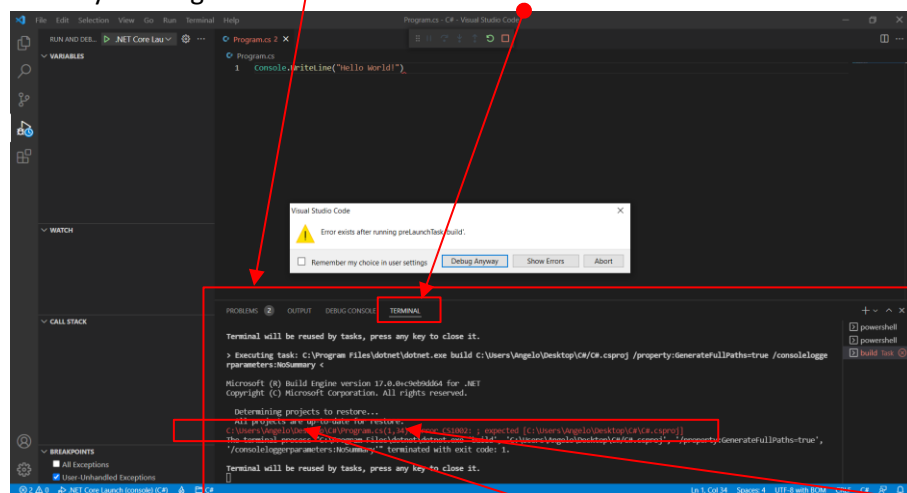
Before your program builds, Visual Studio will check for 2 types of errors: **Build Errors** and **Syntax Errors**.

Build Errors & Syntax Errors:

You have seen **syntax errors** occur when red lines  appears underneath your code (in this case we are missing a semicolon at the end of our statement). Syntax errors occur when you have an error in your code.



A syntax error will also cause a **build error**. Build errors occur **during compilation** of your code and are usually caused by syntax errors (but not always). For example, if you try to run the following code in Visual Studio you will get a build error in the **'Terminal'**:



The **'Terminal'** will show you some error information about the error including the **line number**. You can directly go to the error in your code by holding **CTRL** and **clicking** the error.

When your program is running you may experience **Runtime Errors**.

Runtime Errors:

Run-time errors can occur while the program is running, for example when a program has crashed. So far, our code is quite simple, but you may encounter runtime errors later in this course. It is up to the programmer to make sure that runtime errors never occur (you will learn all about this later in the course).