

LESSON 05 - USER INPUT & BUILT-IN FUNCTIONS



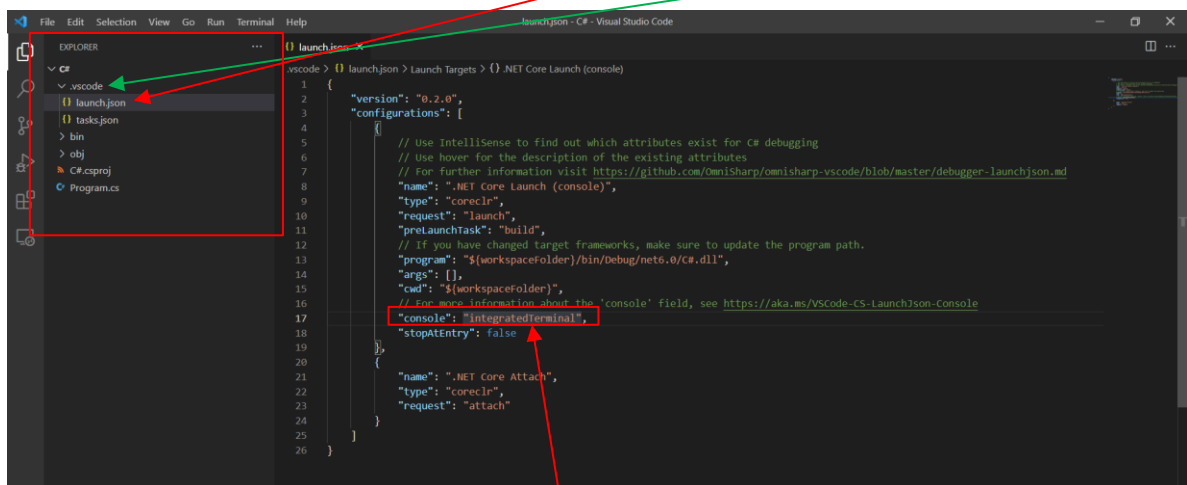
In this lesson we will explore how to receive input from the user which will make our programs much more dynamic. We will also look at some errors that can occur when dealing with user input, and we will examine the structure of built-in functions.

Sections:

- I. [SETTING UP THE 'TERMINAL' FOR INPUT..... PAGE 1](#)
- II. [USER INPUT 'Console.ReadLine\(\)' PAGE 2](#)
- III. [DEALING WITH INPUT ERRORS PAGE 5](#)
- IV. [BUILT-IN FUNCTIONS..... PAGE 6](#)

I. SETTING UP THE 'TERMINAL' FOR INPUT:

So far, we have been using the Visual Studio Code **'debug console'**. However, the debug console only allows for output. If we wish to receive input from the user, we must setup the Visual Studio Code **'terminal'** to be integrated with our program. To do this, expand the **'vscode'** folder in **Visual Studio Code Explorer** and open the **'launch.json'** by double clicking on it:



Now make sure that line 17 reads:

```
"console": "integratedTerminal",
```

Make sure to save the file by going to **'File->Save'** on the menu bar or hitting **CTRL-S** on your keyboard. You will now be using the **terminal** when we run our programs.

II. USER INPUT "Console.ReadLine()":

So far, our programs have been using **hardcoded values literal values**. That is, anytime we use a value (i.e., integer, floats, and strings) within our code we have been typing it directly into our code. For example:

```
int length = 7;
int width = 3;
int area = length * width;
Console.WriteLine("The area of your rectangle is: " + area);
```

Our programs can be more flexible by **asking the user** for values/data (input) instead of hardcoding the values/data. We can do this with the use of **Console.ReadLine()**. For example:

```
1  int length = 0;
2  int width = 0;
3
4  Console.Write("Please enter a length: ");
5  length = Console.ReadLine();
6
7  Console.Write("Please enter a width: ");
8  width = Console.ReadLine();
9
10 int area = length * width;
11 Console.WriteLine("\nThe area of your rectangle is: " + area);
```

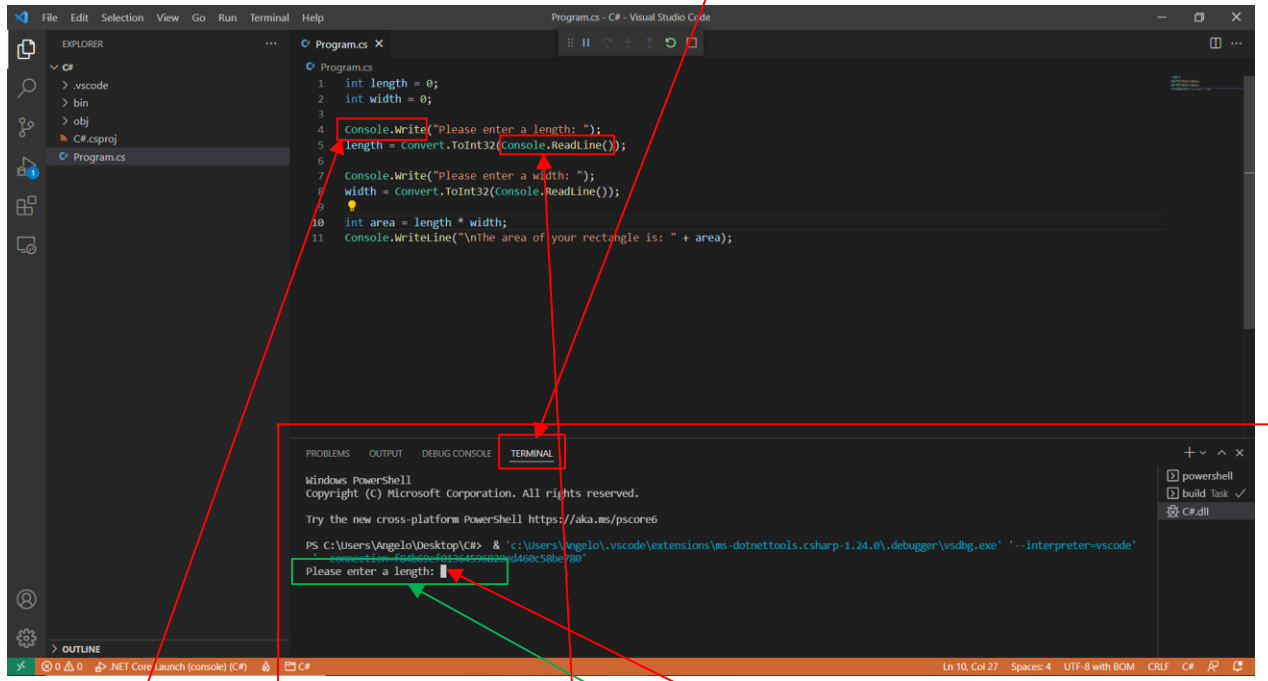
However, our **output** will be a **build error!**

The reason we got a build error is because **Console.ReadLine()** **only reads in a string** even if a number was typed, but **length & width** were **declared as integers**. What we need to do is **convert** the string that is captured by **Console.ReadLine()** into an **integer**. We can do so as follows:

```
1  int length = 0;
2  int width = 0;
3
4  Console.Write("Please enter a length: ");
5  length = Convert.ToInt32(Console.ReadLine());
6
7  Console.Write("Please enter a width: ");
8  width = Convert.ToInt32(Console.ReadLine());
9
10 int area = length * width;
11 Console.WriteLine("\nThe area of your rectangle is: " + area);
```

By putting our **Console.ReadLine()** statement inside the **Convert.ToInt32()** statement, our string that is typed by the user will be converted to an integer. Let's see this in action...

When we run our program, we will now be using the **'terminal'** and not the **'debug console'**. Here is what that looks like:



Notice how our **terminal** is waiting for us to **input a value**. Our program has paused at **line 5** and is **waiting for user input** because of our **Console.ReadLine()** statement. Also notice how we use a **Console.Write()** when alerting the user for input. This ensures that our **input cursor** stays on the **same line**. For example, changing this to **Console.WriteLine()**, will cause our input cursor to **not** stay on the same line:

```
Console.WriteLine("Please enter a length: ");
length = Convert.ToInt32(Console.ReadLine());
```

If we do this, our **input cursor** will appear on the **next line**:

```
Please enter a length:

```

Either way is fine but using **Console.Write()** and having the **input cursor** appear on the same line looks better.

Let's go through all the code so far:

```

1 int length = 0, width = 0;
2
3 Console.Write("Please enter a length: ");
4 length = Convert.ToInt32(Console.ReadLine());
5
6 Console.Write("Please enter a width: ");
7 width = Convert.ToInt32(Console.ReadLine());
8
9 int area = length * width;
10 Console.WriteLine("\nThe area of your rectangle is: " + area);

```

On **line 1** we declare two **integer** variables: '**length**' and '**width**'. On **line 3** we display a message to the user asking to enter a length. **Line 4** is where grab **input** from the **user** and store it in an **integer** variable named '**length**'.

When we run our code, the program will stop at **line 4** and **wait** for the user to type something:

Output will be:

Please enter your length:

Program **paused**. Waiting for the user to type something.

Once the user types a value and hits '**Enter**' the '**length**' variable will be **populated** with the value typed by the user.

After the user types a value for the '**length**' and hits '**Enter**', our program will then ask the user for a '**width**':

Output will be:

Please enter your length: 3
Please enter your width:

Program **paused**. Waiting for the user to type something.

The output above shows that the user typed '**3**' for the '**length**' and the program is now waiting for the user to type a value for the '**width**' variable.

Once the user types a value for the '**width**' and hits '**Enter**', **lines 9** will calculate the '**area**' variable and **line 10** will output the '**area**' variable to the screen. (Also note how we used the escape character '**\n**' in our Console.WriteLine() statement on **line 10** to add an **extra blank line** to make our output look nicer (we could have used another Console.WriteLine() to add a blank line, but using '**\n**' was easier).

Output will be:

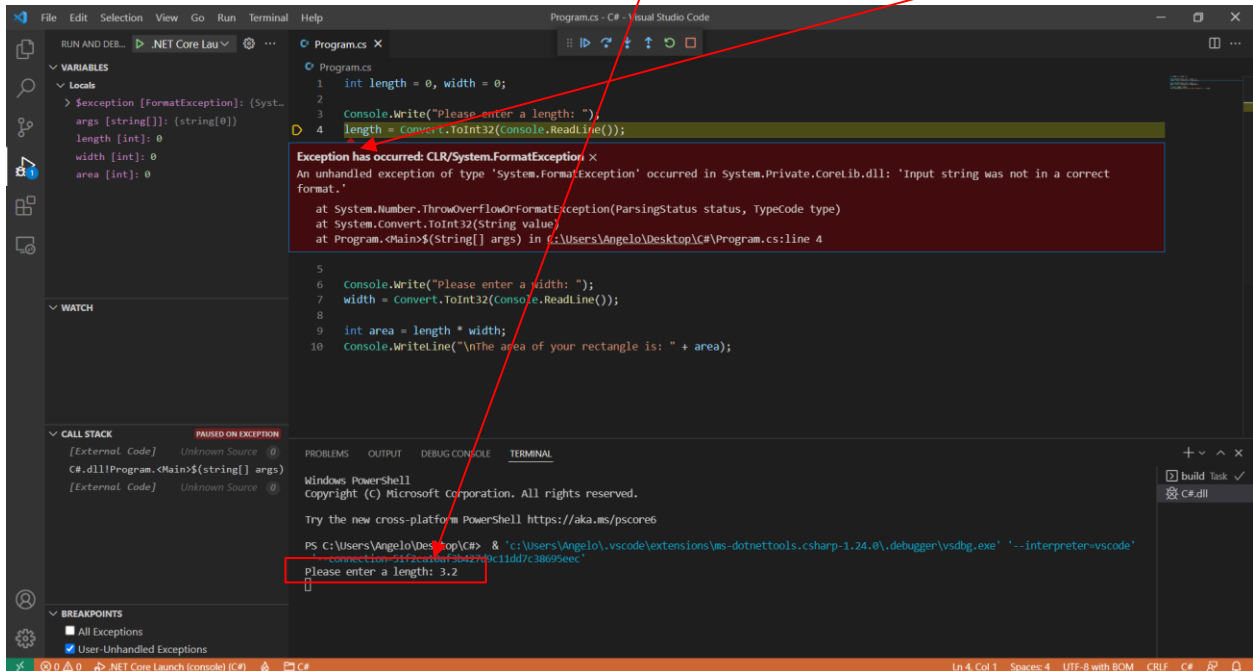
Please enter your length: 3
Please enter your width: 2

User **input**

The area of your rectangle is: 5

III. DEALING WITH INPUT ERRORS:

When the user inputs a value for a variable, the **input** must match the **data type** of the **variable**. For example, with the program from the previous section, if the user inputs decimal values our program will cause a **runtime error** (i.e., program will crash!) because the **input variables** were declared as **integers**. For example, if we try to input **'3.2'** for our **'length'** our program will **raise an exception** which means that an error occurred, and the program has crashed!



The input **'3.2'** cannot be stored into our **integer** variable **'length'**. An easy solution to this problem is to declare the **'length'**, **'width'**, and **'area'** variables as **doubles** instead of ints:

```

1  double length = 0, width = 0;
2
3  Console.WriteLine("Please enter a length: ");
4  length = Convert.ToDouble(Console.ReadLine());
5
6  Console.WriteLine("Please enter a width: ");
7  width = Convert.ToDouble(Console.ReadLine());
8
9  double area = length * width;
10 Console.WriteLine("\nThe area of your rectangle is: " + area);
11

```

Also, we now must use **Convert.ToDouble()** so that our **user input** will convert from a string to **double**. If you do not, then you will get a **runtime error**!

Note: We will not spend too much time on **error checking**, but from this point forward if you expect the user to **input** an **integer** then use an **'int'** variable and if you expect the user to **input** a **decimal** value then use a **'double'** variable.

IV. BUILT-IN FUNCTIONS:

A **built-in function** is a piece of code that performs a task in the background. They always contain, at the least, a **name** and a **set of brackets**. There are many built-in functions in C#. You have already seen a couple of built-in functions, for example: **Console.WriteLine()**, **Console.Write()**, **Console.ReadLine()**, **Convert.ToInt32()**, etc.

Functions are given values which we call **parameters**, and when we give a value to a function, we say that we are **passing** that function a **parameter**. We pass parameters by putting them inside the function brackets. In the case of **Console.WriteLine()** we pass a **string parameter** to output a message, for example:

```
Console.WriteLine("Hello World!");
```




We can also pass **no parameters** to **Console.WriteLine()** to output a blank line, for example:

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



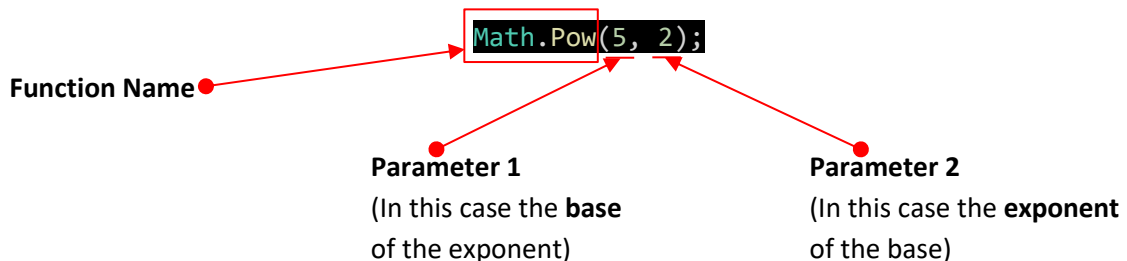
When we used the **Convert.ToInt32()** function we passed the entire **Console.ReadLine()** function into it so that it could convert our user input from a string to an integer. For example:

```
Convert.ToInt32(Console.ReadLine());
```



Some built-in functions allow for multiple parameters, for example let's look at the **Math.Pow()** function used to calculate the power of a number:

```
Math.Pow(5, 2);
```



Function Name

Parameter 1
(In this case the **base** of the exponent)

Parameter 2
(In this case the **exponent** of the base)

The above **Math.Pow()** function will calculate the power of a number. It takes two **parameters**: the **base** and **exponent**. Parameters of a function are always separate by a **comma (,)** unless there is only one parameter. Some built-in functions contain many parameters.

Note: It is important to keep in mind the **syntax** we use in C#. **Syntax** refers to the arrangement of the words and symbols used in our code. When using functions, the function names usually start with a capital letter. Functions also use brackets for the parameters where an open bracket always has an accompanying closed bracket at the end.

Here is how we can use the built-in **Math.Pow()** function:

```
double x = Math.Pow(5, 2);  
Console.WriteLine(x);
```

Output will be:

25

Functions always **return** a value. The value **returned** by the function is always of a specific **data type**. In the case of **Math.Pow()**, it returns a **double value**. If we did not declare the variable 'x' above as a **double**, then we would have gotten a **build error**!

We do not necessarily have to store the value **returned** by a built-in function into a variable. We can use it directly in our **Console.WriteLine()** statement. For example:

```
Console.WriteLine(Math.Pow(5, 2));
```

Output will be:

25

We could also **only** variables with our functions. For example:

```
int b = 5;  
int e = 2;  
double x = Math.Pow(b, e);  
Console.WriteLine(x);
```

Output will be:

25

As you can see, the C# language is flexible!