200 XP

Exercise - Examine the TryParse() method

12 minutes

When working with data, sometimes, you need to convert string data into a numeric data type. As you learned in the previous unit, because the string data type can hold a non-numeric value, it's possible that performing a conversion from a `string` into a numeric data type causes a runtime error.

For example, the following code:

C#

```
string name = "Bob";
Console.WriteLine(int.Parse(name));
```

causes the following exception:

Output

```
System.FormatException: 'Input string was not in a correct format.'
```

To avoid a format exception, use the TryParse() method on the target data type.

Use TryParse()

The TryParse() method does several things simultaneously:

- It attempts to parse a string into the given numeric data type.
- If successful, it stores the converted value in an **out parameter**, explained in following section.
- It returns a `bool` to indicate whether the action succeeded or failed.

You can use the Boolean return value to take action on the value (like performing some calculation), or display a message if the parse operation was unsuccessful.

Note

In this exercise, you'll use the `int` data type, but a similar `TryParse()` method is available on all numeric data types.

Out parameters

Methods can return a value or return "void" - meaning they return no value. Methods can also return values through `out` parameters, which are defined just like an input parameter, but include the `out` keyword.

TryParse() a string into an int

1. Delete or use the line comment operator `//` to comment out all of the code from the previous exercises.
2. Update your code in the Visual Studio Code Editor as follows:

   C#

   ```
   string value = "102";
   int result = 0;
   if (int.TryParse(value, out result))
   {
      Console.WriteLine($"Measurement: {result}");
   }
   else
   {
      Console.WriteLine("Unable to report the measurement.");
   }
   ```

3. Examine this line of code:

   ```
   if (int.TryParse(value, out result))
   ```

   When calling a method with an `out` parameter, you must use the keyword `out` before the variable, which holds the value. The `out` parameter is assigned to the `result` variable in the code `(int.TryParse(value,` **`out result`** `)`. You can then use the value the `out` parameter contains throughout the rest of your code using the variable `result`.

The `int.TryParse()` method returns `true` if it successfully converted the `string` variable `value` into an `int`; otherwise, it returns `false`. So, surround the statement in an `if` statement, and then perform the decision logic, accordingly.

The converted value is stored in the `int` variable `result`. The `int` variable `result` is declared and initialized before this line of code, so it should be accessible both *inside* the code blocks that belong to the `if` and `else` statements, as well as *outside* of them.

The `out` keyword instructs the compiler that the `TryParse()` method won't return a value the traditional way only (as a return value), but also will communicate an output through this two-way parameter.

When you run the code, you should see the following output:

Output

```
Measurement: 102
```

Use the parsed `int` later in code

1. To demonstrate that the `result` variable that was declared earlier, is populated by the `out` parameter and is also usable later in your code, update your code in the Visual Studio Code Editor as follows:

C#

```csharp
string value = "102";
int result = 0;
if (int.TryParse(value, out result))
{
   Console.WriteLine($"Measurement: {result}");
}
else
{
   Console.WriteLine("Unable to report the measurement.");
}
Console.WriteLine($"Measurement (w/ offset): {50 + result}");
```

2. On the Visual Studio Code **File** menu, select **Save**. The Program.cs file must be saved before building or running the code.

3. In the EXPLORER panel, to open a Terminal at your TestProject folder location, right-click **TestProject**, and then select **Open in Integrated Terminal**. A Terminal panel should open, and should include a command prompt showing that the Terminal is open to your TestProject folder location.

4. At the Terminal command prompt, to run your code, type **dotnet run** and then press Enter.

   Note

   If you see a message saying "Couldn't find a project to run", ensure that the Terminal command prompt displays the expected TestProject folder location. For example: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

   You should see the following output:

   Output

   ```
   Measurement: 102
   Measurement (w/ offset): 152
   ```

5. Examine the last line of code in the previous sample, `Console.WriteLine($"Measurement (w/ offset): {50 + result}");`, Since the `result` variable is defined outside of the if statement, it can be accessed later in your code.

Modify the string variable to a value that can't be parsed

Lastly, look at the other scenario - where the `TryParse()` is intentionally given a bad value that can't be converted into an `int`.

1. Modify the first line of code, reinitialize the variable `value` to a different value.

   C#

   ```
   string value = "bad";
   ```

2. Also, modify the last line of code to ensure that the result is greater than 0 before showing the second message.

   C#

```
if (result > 0)
   Console.WriteLine($"Measurement (w/ offset): {50 + result}");
```

3. The entire code example should now match the following code:

C#

```csharp
string value = "bad";
int result = 0;
if (int.TryParse(value, out result))
{
   Console.WriteLine($"Measurement: {result}");
}
else
{
   Console.WriteLine("Unable to report the measurement.");
}

if (result > 0)
   Console.WriteLine($"Measurement (w/ offset): {50 + result}");
```

4. Save your code file, and then use Visual Studio Code to run your code. You should get the following result:

Output

```
Unable to report the measurement.
```

5. Examine the last two lines of code added in the previous sample.

C#

```csharp
if (result > 0)
   Console.WriteLine($"Measurement (w/ offset): {50 + result}");
```

Since `result` is defined outside of the `if` statement, `result` can be accessed later in your code outside of the code blocks. So then `result` can be checked for a value greater than zero before allowing `result` + offset to be written as output. Checking for a `result` value greater than zero avoids printing an offset value after the `Unable to report the measurement.` message.

Recap

The `TryParse()` method is a valuable tool. Here are few quick ideas to remember.

- Use `TryParse()` when converting a string into a numeric data type.
- `TryParse()` returns `true` if the conversion is successful, `false` if it's unsuccessful.
- Out parameters provide a secondary means of a method returning a value. In this case, the `out` parameter returns the converted value.
- Use the keyword `out` when passing in an argument to a method that has defined an `out` parameter.

Check your knowledge

1. Which technique should be used to change `myInput`, a `string` value `"2.71828"`, into a `decimal` variable `myInputDecimal`? *

○ `decimal myInputDecimal = (decimal)(myInput);`

○ `decimal myInputDecimal = myInput + 0;`

○ `decimal.TryParse(myInput, out myInputDecimal)`

2. Which best describes the return type of `decimal.TryParse()`? *

○ `decimal`

○ `bool`

○ `out`

Check your answers