Converting Data Types

Because variables have to be strictly typed, there may be some circumstances where we want to change the type of data a variable is storing. This strategy is known as *data type conversion*.

For example, let's try converting a double to an integer:

```
double myDouble = 3.2;

// Round myDouble to the nearest whole number
int myInt = myDouble;
```

But if you tried to run this code, it wouldn't work. Sorry.

However, if you did the reverse and turned an int into a double:

```
int myInt = 3;
// Turn it into a decimal
double MyDouble = myInt;
```

It's fine! Why is that?

C# checks to make sure that when we convert data types from one to another that we're not losing any data, because that could cause problems in our code.

Because of that, there are a couple different ways to do data type conversion:

implicit conversion: happens automatically if no data will be lost in the conversion. That's why it's possible to convert an int (which can hold less data) to a double (which can hold more), but not the other way around.

explicit conversion: requires a cast operator to convert a data type into another one. So if we do want to convert a double to an int, we could use the operator (int).

So, if we're to fix the error in our previous code, we'd rewrite it as follows:

```
double myDouble = 3.2;

// Round myDouble to the nearest whole number
int myInt = (int)myDouble;
```

It's also possible to convert data types using built-in methods. For most data types, there is a <code>Convert.ToX()</code> method, like <code>Convert.ToString()</code> and <code>Convert.ToDouble()</code>. For a full list of <code>Convert</code> class built-in methods, check out the <code>Microsoft</code> <code>Documentation</code>.

✓Instructions

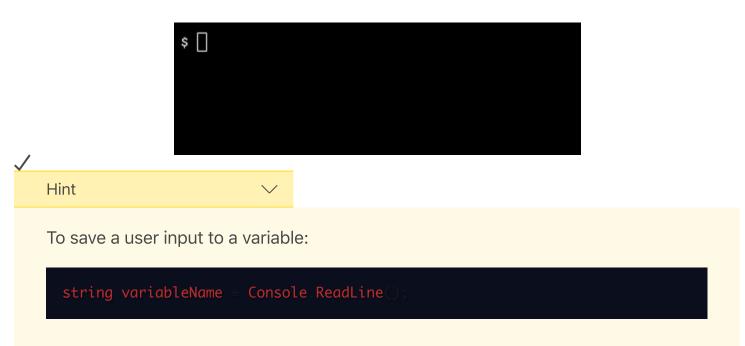
1.

One example of when we have to use conversions is when we ask a user to input a numerical value. Even if that value is an integer or a decimal, Console.ReadLine() will always return a string.

Let's write a program that asks a learner for their favorite number and see if we can *implicitly* convert their response to an int.

To start, below the Console.Write() statement, create an int variable named faveNumber and set it equal to Console.ReadLine().

To run the program, press the **Run** button to save your work, then type **dotnet** run into the console.

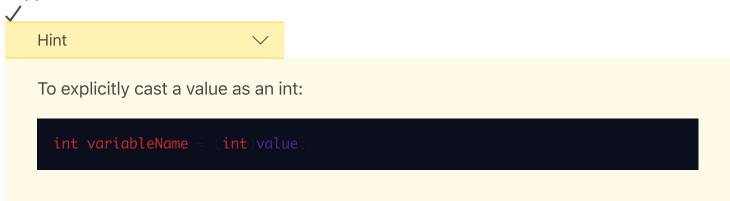


2. Hmm. That didn't work. Instead, we got the error message:

\$ dotnet run Program.cs(10,30): error CS0029: Cannot implici tly convert type 'string' to 'int' [/home/ccuse r/workspace/csharp-data-types-variables-convert ing-data-types-csharp/e7-workspace.csproj] The build failed. Please fix the build errors a nd run again.

Looks like we're going to have to cast their response as an int some other way!

Try explicitly casting the value of faveNumber as an int and rerun the program. What happens this time?



3. If you tried dotnet run again, you'll see that (int) didn't work either. That's because it is not possible to implicitly convert a string into an int (or vice versa) in C#. This time, let's try using a built-in method to do the conversion.

Look at <u>this article on converting strings to int</u>. It lists a few of the methods in the Convert class, including: <u>Convert.ToInt32()</u>. This method takes a string and outputs an integer. Let's try it!

Delete the explicit casting (int) from the code editor. Add the Convert.ToInt32() method so that it takes the user input as a string.

Run the code again. Did you run into any errors?

