Exercise - Setting and getting values from variables

5 minutes

Because variables are temporary storage containers for data, they're meant to be *written to* and *read from*. You'll get a chance to do both in the following exercise.

Exercise - Working with variables

In this exercise, you'll declare a variable, assign it a value, retrieve its value, and more.

Step 1: Delete all of the code in the code editor

Use your mouse to highlight all of the text in the code editor, then select the backspace or `del` key to remove everything.

Later, we'll learn a different, less destructive technique for disabling the lines of code you no longer want to execute.

Step 2: Declare a variable and assign a value to it

To assign a value to a variable, you use the *assignment operator*, which is a single equals character `=`.

Add the following code in the code editor:

C# Copy Run

```
string firstName;
firstName = "Bob";
```

Assigning a value is also referred to as "setting the variable", or simply, a "set" operation.

Step 3: Attempt to improperly assign a value to a variable

It's important to notice that assignment happens from right to left. In other words, the C# compiler must first understand the value on the right side of the assignment operator, then it can perform the assignment to the variable on the left side of the assignment operator. If you reverse the order, you'll confuse the C# compiler.

Modify the code you wrote in Step 2 to match the following code:

C# Copy Run

```
string firstName;
"Bob" = firstName;
```

Now, run the code. The first error you would see in the output console:

Output Copy

```
(2,1): error CS0131: The left-hand side of an assignment must be a variable, property or indexer
```

Step 4: Improperly assign a value of the incorrect data type to the variable

Earlier, we declared that C# was designed to enforce types. When working with variables, *enforcing types* means you can't assign a value of one data type to a variable declared to hold a different data type.

Modify the code you wrote in Step 3 to match the following code:

C# Copy Run

```
int firstName;
firstName = "Bob";
```

Now, run the code. You'll see the following error in the output console:

Output Copy

```
(2,9): error CS0029: Cannot implicitly convert type 'string' to 'int'
```

The error message hints at what the C# compiler tries to do behind the scenes. It tried to "implicitly convert" the string "Bob" to be an int value; however, that is impossible. In other words, you can't put a square peg into a round hole. Even so, C# tried to force the square peg to fit into the round hole, but there's no numeric equivalent for the word "Bob", so it failed.

You'll learn more about implicit and explicit type conversion in other modules.

Step 5: Retrieve a value you stored in the variable

To retrieve a value from a variable, you just use the name of the variable. This example will set a variable's value, then retrieve that value and print it to the console.

Modify the code you wrote in Step 4 to match the following code:

C# Copy Run

```
string firstName;
firstName = "Bob";
Console.WriteLine(firstName);
```

Now, run the code. You'll see the following result in the output console:

Output Copy

```
Bob
```

Retrieving a value from a variable is also referred to as "getting the variable", or simply, a "get" operation.

Step 6: Reassign the value of a variable

You can reuse and reassign the variable as many times as you want. This example illustrates that idea.

Modify the code you wrote in Step 5 to match the following code:

C# Copy Run

```
string firstName;
firstName = "Bob";
Console.WriteLine(firstName);
firstName = "Beth";
Console.WriteLine(firstName);
firstName = "Conrad";
Console.WriteLine(firstName);
firstName = "Grant";
Console.WriteLine(firstName);
```

Now, run the code. You'll see the following result in the output console:

Output Copy

```
Bob
Beth
Conrad
Grant
```

Step 7: Initialize the variable

You must set a variable to a value before you can get the value from the variable. Otherwise, you'll see an error.

Modify the code you wrote in Step 6 to match the following code:

C# Copy Run

```
string firstName;
Console.WriteLine(firstName);
```

Now, run the code. You'll see the following result in the output console:

Output Copy

```
(2,19): error CS0165: Use of unassigned local variable 'firstName'
```

To avoid the possibility of an unassigned local variable, we recommend that you set the value as soon as possible after you declare it.

In fact, you can perform both the declaration and setting the value of the variable in one line of code. This technique is called *initializing* the variable.

Modify the code you wrote earlier in this step to match the following code:

C# Copy Run

```
string firstName = "Bob";
Console.WriteLine(firstName);
```

Now, run the code. You should see the following output:

Output Copy

```
Bob
```

Recap

Here's the main takeaways you learned about working with variables so far:

- You must assign (set) a value to a variable before you can retrieve (get) a value from a variable.
- You can initialize a variable by assigning a value to the variable at the point of declaration.
- Assignment happens from right to left.
- You use a single equals character as the assignment operator.
- To retrieve the value from the variable, you merely use the variable's name.

Next unit: Implicitly typed local variables

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

.NET Editor

Press CTRL+M, TAB to exit the editor

Clear Run

```
1
```

Output