200 XP

Learn how it works

6 minutes

To understand how your code works, you need to step back and think about what a programming language is. Consider how your code communicates commands to the computer.

What is a programming language?

Programming languages like C# let you write instructions that you want the computer to carry out. Each programming language has its own syntax, but after learning your first programming language and attempting to learn another one, you'll quickly realize that they all share many similar concepts. A programming language's job is to allow a human to express their intent in a human-readable and understandable way. The instructions you write in a programming language are called "source code" or just "code". Software developers write code.

At this point, a developer can update and change the code, but the computer can't understand the code. The code first must be *compiled* into a format that the computer can understand.

What is compilation?

A special program called a **compiler** converts your source code into a different format that the computer's central processing unit (CPU) can execute. When you used the green **Run** button in the previous unit, the code you wrote was first compiled, then executed.

Why does code need to be compiled? Although most programming languages seem cryptic at first, they can be more easily understood by humans than the computer's *preferred* language. The CPU understands instructions that are expressed by turning thousands or millions of tiny switches either on or off. Compilers bridge these two worlds by translating your human-readable instructions into a computer-understandable set of instructions.

What is syntax?

The rules for writing C# code is called syntax. Just like human languages have rules regarding punctuation and sentence structure, computer programming languages also have rules. Those

rules define the keywords and operators of C# and how they are put together to form programs.

When you wrote code into the .NET Editor, you may have noticed subtle changes to the color of different words and symbols. Syntax highlighting is a helpful feature that you'll begin to use to easily spot mistakes in your code that don't conform to the syntax rules of C#.

How did your code work?

Let's focus on the following line of code you wrote:

```
C#

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

When you ran your code, you saw that the message `Hello World!` was printed to the output console. When the phrase is surrounded by double-quotation marks in your C# code, it's called a **literal string**. In other words, you literally wanted the characters `H`, `e`, `l`, `l`, `o`, and so on, sent to the output.

The `Console` part is called a **class**. Classes "own" methods; or you could say that methods live inside of a class. To visit the method, you must know which class it's in. For now, think of a class as a way to represent an object. In this case, all of the methods that operate on your output console are defined inside of the `Console` class.

There's also a dot (or period) that separates the class name `Console` and the method name `WriteLine()`. The period is the *member access operator*. In other words, the dot is how you "navigate" from the class to one of its methods.

The `WriteLine()` part is called a **method**. You can always spot a method because it has a set of parentheses after it. Each method has one job. The `WriteLine()` method's job is to write a line of data to the output console. The data that's printed is sent in between the opening and closing parenthesis as an input parameter. Some methods need input parameters, while others don't. But if you want to invoke a method, you must always use the parentheses after the method's name. The parentheses are known as the *method invocation operator*.

Finally, the semicolon is the *end of statement operator*. A **statement** is a complete instruction in C#. The semicolon tells the compiler that you've finished entering the command.

Don't worry if all of these ideas and terms don't make sense. For now, all you need to remember is that if you want to print a message to the output console:

- Use `Console.WriteLine("Your message here");`
- Capitalize `Console`, `Write`, and `Line`
- Use the correct *punctuation* because it has a special role in C#
- If you make a mistake, just spot it, fix it and re-run

> **Tip**
>
> Create a cheat sheet for yourself until you've memorized certain key commands.

Understand the flow of execution

It's important to understand the flow of execution. In other words, your code instructions were executed in order, one line at a time, until there were no more instructions to execute. Some instructions will require the CPU to wait before it can continue. Other instructions can be used to change the flow of execution.

Now, let's test what you've learned. Each module features a simple challenge, and if you get stuck, you'll be supplied with a solution. In the next unit, you'll get a chance to write some C# on your own.

Check your knowledge

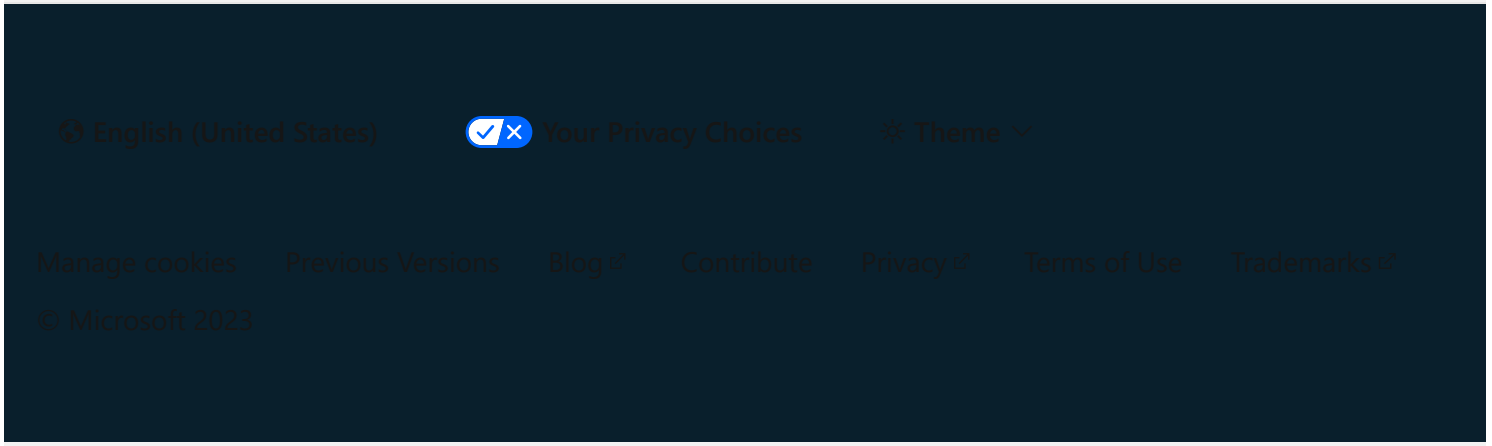
1. What is the difference between `Console.Write` and `Console.WriteLine`? *

- ◯ `Console.Write` prints the output on a new line.
- ◯ `Console.WriteLine` prints the output on a new line.
- ◯ `Console.WriteLine` appends a new line after the output.

Check your answers

Module complete:

Unlock achievement



.NET Editor

Press CTRL + M, TAB to exit the editor

Clear Run

```
1
```

Output