

# How it works

4 minutes

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

## What is a programming language?

Programming languages like C# enable you to write instructions that you want the computer to carry out. Each programming language has a different syntax, but after you learn your first programming language and then attempt to learn a second one, you'll quickly realize that they all share many similar ideas. A programming language's job is to enable a human to write instructions in a human-readable and understandable way. The instructions you write in a programming language is called "source code", or just "code".

At this point, the code can be updated and changed by a software developer, but the computer can't understand the code. It 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 is executable by the computer's CPU. When you used the green **Run** button in the previous unit, the code you wrote was first compiled, then executed.

Why do we need to compile our code? Even though most programming languages seem cryptic at first, they can be more easily understood by humans than the computer's *preferred* language, which is 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 syntax of a programming language includes the keywords, the operators (those special keyboard characters like the semicolon or parenthesis), and other grammar rules that the

compiler enforces. The lines of code you typed followed about a dozen different syntax rules and used at least four different operators. There's much to learn, but fortunately each concept is simple. Don't give up! You can learn it!

When you entered 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#. In fact, a similar (and even more robust) version of this feature is available in Visual Studio Code and the full Visual Studio IDE.

## How did your code work?

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

C#	 Copy
<pre>Console.WriteLine("Hello World!");</pre>	

When you ran your code, you saw that the message `Hello World!` was printed to the output pane. When the phrase is surrounded by double-quotation marks in your C# code, it's called a **literal string**. In other words, we literally wanted the characters `H`, `e`, `l`, `l`, `o`, and so on, sent to the output. You'll learn about literal strings in the module titled "Store and retrieve data using literal and variable values in C#".

The `WriteLine()` part is called a **method**. You can always spot a method because it has a set of parenthesis after it. Each method has one job. The `WriteLine()` method's job is to write a line of data to the output window. The data that's printed is sent in between the opening and closing parenthesis as an input parameter. Some methods need input parameters, others don't. But if you want to invoke a method you must always use the parenthesis after the method's name. The parentheses are known as the **method invocation operator**. You'll learn more about calling methods in the module titled "Harness libraries of functionality by calling methods from the .NET Class Library in C#".

The `Console` part is called a **class**. Classes "own" methods, or perhaps a better way to say it is 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 store and organize all of the methods that do similar things. In this case, all of the methods that operate on your Output pane are defined inside of the `Console` class.

There is 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.

Finally, the semi-colon is the *end of statement operator*. A **statement** is a complete instruction in C#. The semi-colon tells the compiler that we're finished entering the command.

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

- Use `Console.WriteLine("Your message here");`
- Capitalize `Console`, `Write`, and `Line`
- Use the correct *punctuation* because they have a special role in C#
- If you make a mistake, just spot it, fix it and rerun ... you can't really fail

### Tip

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

## Understand the flow of execution

Also, 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. You'll learn about these special situations as you learn more C# syntax and methods in the .NET Class Library.

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

## Next unit: Challenge

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

.NET Editor

Press  + ,  to exit the editor

 Clear

 Run



1

Output 