**Writing your first Javascript code**

In this reading, you'll learn about comments in JavaScript. Additionally, you'll learn about the semi-colon in JavaScript: what it does and why it is used. You will then download a browser if you don't have one installed and run your first piece of JavaScript using the Console.

**Comments in JavaScript**

I've chosen comments as the starting point for two reasons:

1. Their syntax - the way comments are written is easy to understand.
2. Writing comments can empower you as a developer.

First, I'll explain the syntax, and after that, I'll discuss why being able to write comments is so empowering.

**Comments in JavaScript: the syntax**

There are two varieties of comments in JavaScript:

1. Single-line comments
2. Multi-line comments

A single-line comment is created when you add two forward-slash characters one after the other, without spaces.

// this is a comment!

Anything that follows a single-line comment in JavaScript is ignored by the browser.

This means that, essentially, you can write any kind of text, code, characters, emojis, whatever - and the browser will ignore it.

A multi-line comment, as its name says, spans for several lines of code and is created with a forward slash and a star. For example:

/\*

this

is

a

multi-line

comment

\*/

You can also use the multi-line comment syntax on a single line of code, as follows:

/\* this is a multi-line comment on a single line \*/

**Why writing comments is empowering**

In this course, it is assumed that you've never written a single line of JavaScript code.

With this assumption in mind, consider the effects of what you've just learned, that is, the effects of learning how to write comments in JavaScript:

1. You can now freely express your ideas about any code that you write.
2. You can add comments to any code that already exists.
3. Those comments can be intended for your future self, or for colleagues on your development team.

So, comments are empowering because they facilitate communication with your future self or with your team members, allowing you to ask questions about the code, mark the code as "to do", or as "to improve", or just simply explain what a given piece of code does.

Additionally, you can even comment out some working code in a JavaScript file - to prevent it from running.

Effectively, comments allow you to "switch off" pieces of JavaScript code.

There can be many reasons for that:

1. Trying to understand how a given piece of code works.
2. Testing different solutions to a coding problem - while not having to delete existing code.
3. Debugging - trying to pin-point why your code is broken or behaving unprediticably.

**The semi-colon in JavaScript**

In the English language, the fullstop or period - the **.** character - is used to separate thoughts into sentences.

By clearly separating thoughts with the fullstop, you avoid being misunderstood.

In JavaScript, the semi-colon - the **;** character - has a similar purpose: it is used to clearly delimit parts of the code from some other parts of the code.

**Automatic Semi-Colon Insertion (ASI)**

Interestingly, the browser has a feature known as "Automatic Semi-colon Insertion" - meaning, it does a pretty good job of "filling in the blanks" in case there is a missing semi-colon where there should be one.

Effectively, what that means for developers is that most of the time, it makes no difference if a semi-colon is added or not, since the browser is likely to figure it out anyway.

That's why some developers say that you shouldn't bother with adding semi-colons at all.

However, other developers argue that it's better to use it wherever it's needed - for the sake of clarity.

The truth is that most of the time, you can think of adding semi-colons in JavaScript as optional - and somewhat of a stylistic preference.

**A note on using the console in the developer tools in your browser**

As already mentioned earlier on in this course, one of the reasons why JavaScript is so popular is because it's so approachable.

To get started with JavaScript, all you need is a browser. In this course I'll be using Google Chrome.

Once you've installed the browser and run it, right-click on the currently active web page and click the **Inspect** command on the right-click contextual menu.

This will open the Developer Tools and then you can click on the Console tab to open the console, or alternatively, pressing the **ESC** key will toggle on and off the console regardless of the currently active Developer Tools panel.

You can type any JavaScript command you like into the DevTools console.

**If you need to type multiple lines of code before you run them, make sure to press the SHIFT + ENTER shortcut key to get onto the next line.**

Notice the distinction between pressing the **ENTER** key to run the JavaScript code you've typed, versus pressing the **SHIFT + ENTER** shortcut to move onto the next line of code (rather than running the code you've already typed up).

This is all that you need to get started writing JavaScript code!

In the upcoming lessons, feel free to follow along in either of two ways:

1. Using the VS Code editor and the Code Runner extension as previously described
2. Using the Chrome browser itself, and running the code inside the DevTools console as described in this reading

**Output a greeting into the console**

Now that you know how to get to the Developer Tools' Console tab, let's now use it to run your first piece of real JavaScript code.

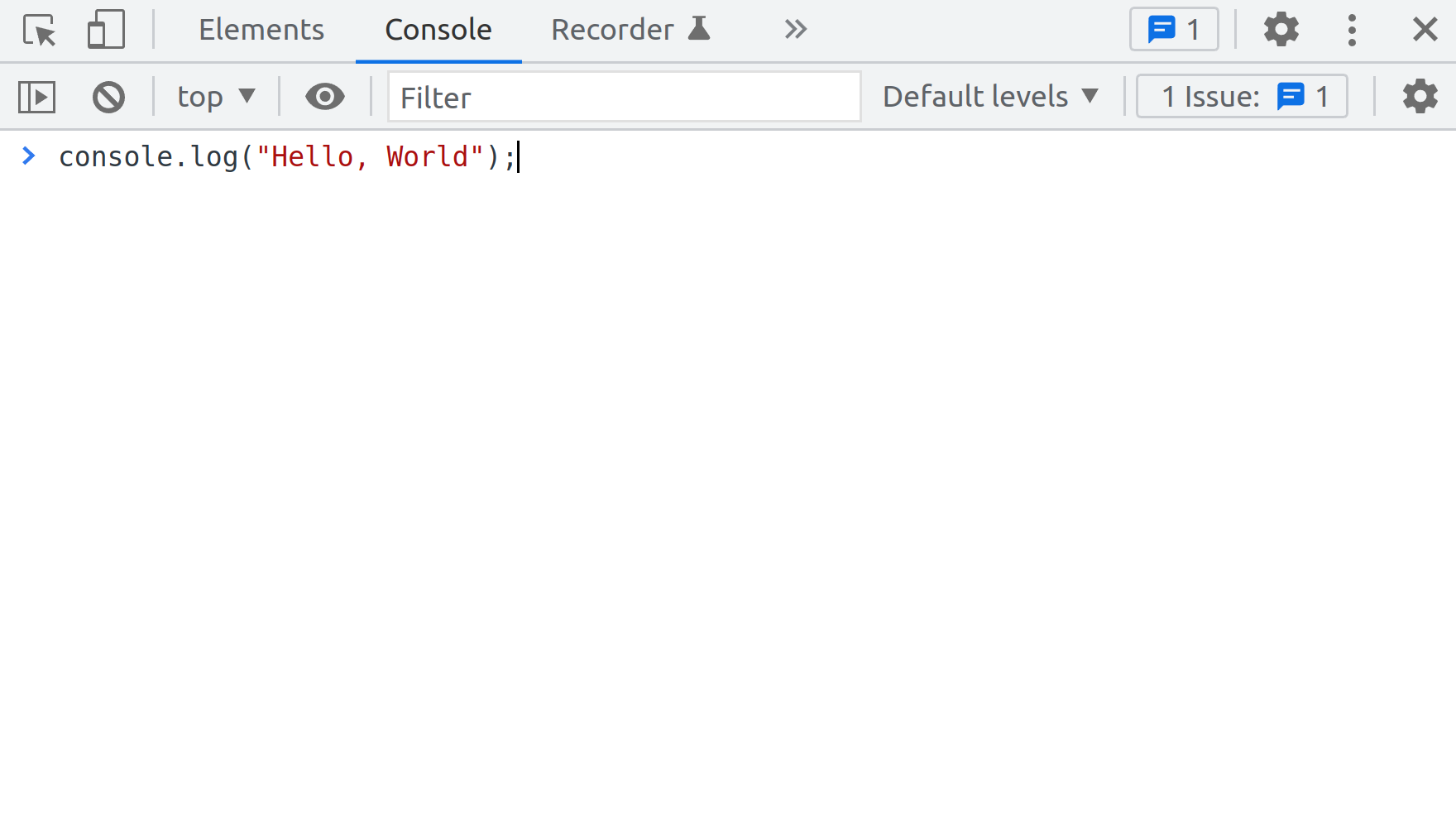
In Chrome, with the Developer Tools open, click into the empty space in the console tab, just to the right of the blue **>** character. You should see a blinking vertical line (also referred to as "the cursor"). The cursor indicates that you can type into the console.

If you type valid JavaScript code, it will be executed, meaning: it will be processed and it might result in some kind of output.

You'll now use the **console.log** function to output the words "Hello, World".

To do so, type the following command into the console:

console.log("Hello, World");



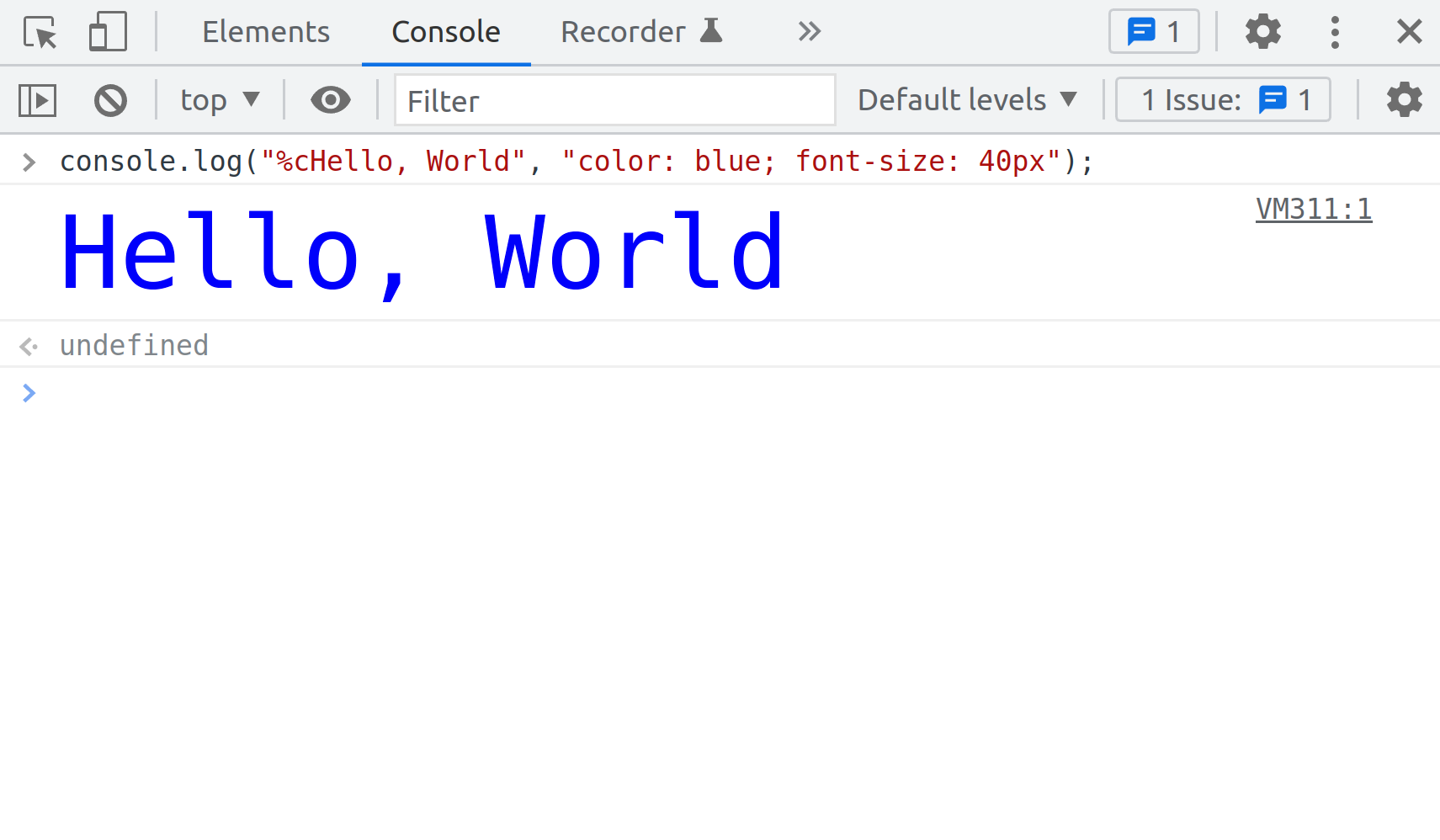
If you've done everything as instructed, the words "Hello, World" should be output in the console.

Here's another, more complex command, to show you that the **console.log** command comes with a number of tricks.

For example, did you know that you can style the output in the console?

In this code snippet, there are a few additions: the font size is different and the color is blue:

console.log("%cHello, World", "color: blue; font-size: 40px");

console log with percentage c authors

If you copy-paste this piece of code, or perhaps, simply type it into the console, once you press the **ENTER** key to run it, you'll get the words "Hello, World" output to the console. This time, however, the color of the letters will be blue, and the font size will be 40px. So, you've just learned a nice trick with the console.

If you add the **%c** right after the **"** character, you can then style the console output by adding the **,** character after the second **"**, and then, inside another pair of **"** and **"** characters, use valid CSS code to style the words you want to output in the console.

The reason for showing you this little trick was to hopefully get you motivated to practice writing various words into the **console.log** command, and to use your CSS skills to change the styling of these words in the console output. That way, you might find it fun to practice this newly acquired skill - and learning and fun always go nicely together.

**Output multiple words into the console**

To output multiple words into the console, you can join them using the **+** character, formally known as the "concatenation operator" when we're joining pieces of text, or the "addition operator", for performing the mathematical operation of adding two numbers.

Here is an example of joining three separate pieces of text: **console.log("Hello " + "there, " + "World")**. The output of this command will be: *Hello there, World*.

Here is an example of outputting three separate pieces of text using the , character instead:

**console.log("Hello ", "there, ", "World")**

The output of this command will still be: *Hello there, World*.