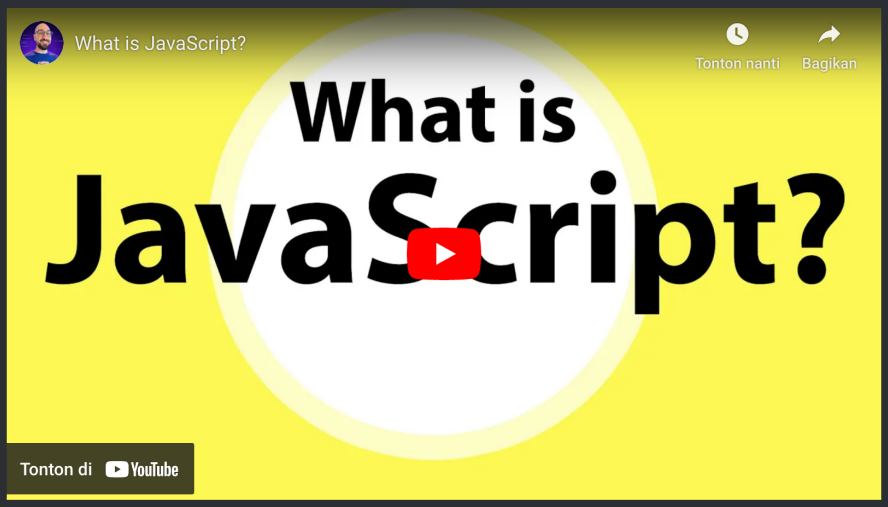
JavaScript Introduction

Writing code

At its most basic level, "code" is just a bunch of words and numbers in a file. That's all it is! As you start working with JavaScript, you'll want to keep this fact in mind—it can help guide your learning process.

In a way, the only tool you need to write JavaScript is a text editor (or a CodePen, as you already know). But there are several rules you have to learn if you want your code to work properly. Before you dive deeper, you'll need to learn more about comments. Comments don't *do* anything when the code runs, but they help coders make notes about what the code does.

Watch this amazing video by Mosh where he explains some basic questions on what JavaScript is.



Basic JavaScript Syntax

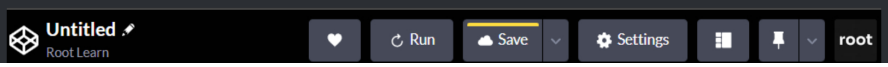
> The **syntax** of a programming language is the **set of rules** that needs to be respected by programmers (*who write the code*) to be successfully interpreted by machines (*that execute that code*)

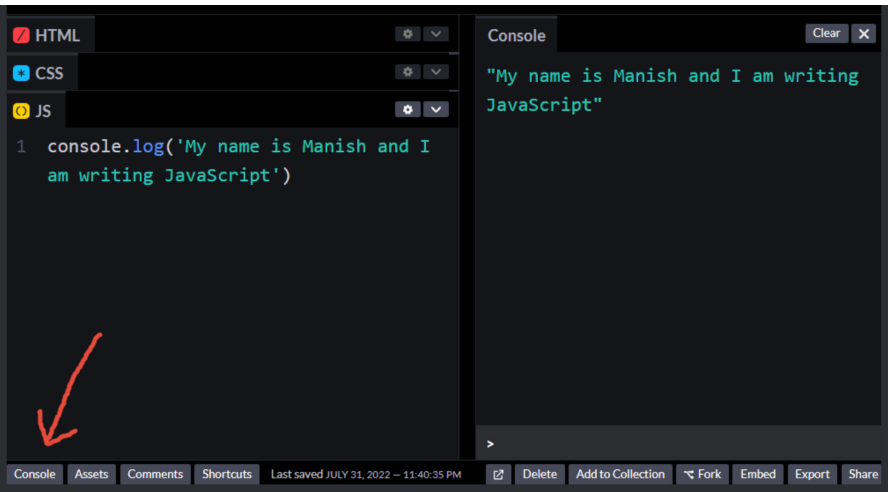
JavaScript's syntax is *loosely* based on C or Java. This means quite a few curly braces {} and parentheses ().

> Although the semicolon is not mandatory, **parentheses and curly braces are mandatory** and will cause an error if they're left out. Be mindful of starting/ending these in the right place.

Output

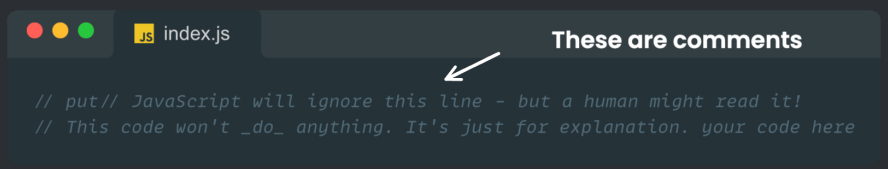
We will use a command called **console.log** to output results and messages in the console. You will get used to writing a lot of *console.log()* and it will become your veeeerrryyyy good friend over time. We will keep using CodePen, as we did earlier in this module but now we will be mainly working in its *JavaScript* section and look for results in its *console*.





Adding comments

Comments allow you to share additional information about your code and communicate with other coders. Take a look at the example below.

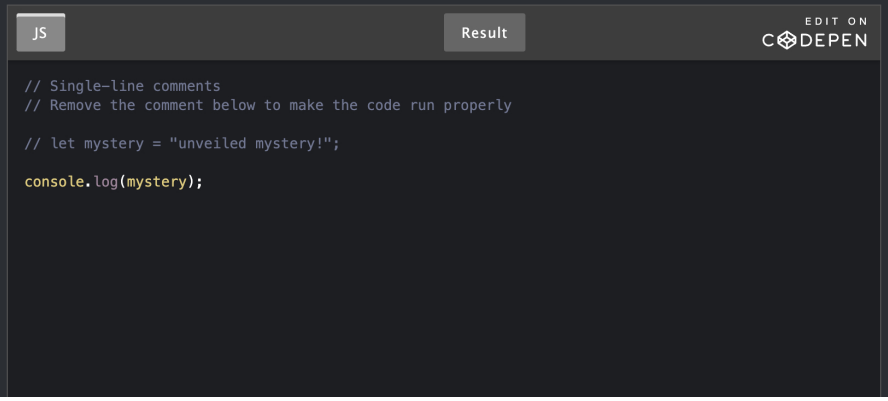


You can type whatever you want in a comment. Coders find comments extremely useful for describing what code is doing, and they're especially important when you collaborate with others on complex coding projects. In those cases, you'll likely want to provide descriptions in the code to help other developers—and future you!—understand what is happening in your code. Learning how to write brief, helpful comments is an essential skill for any developer.

In JavaScript, there are two types of comments: *single-line* (also called *one-line*) comments and *multi-line* comments. Here's an example of each type.
Use the following challenges to practice writing comments.

Demo: Removing one-line comments

First, try running the code below to see what happens. Look at the first line of the error message: **ReferenceError: mystery is not defined**. What do you think this means? Remove the two comment slashes // on line 4 to create and assign the **mystery** variable. What does that do?



Resources

Demo: Adding comments

Now, add single-line and multi-line comments to make the code work properly. What happens this time?

JS

Result

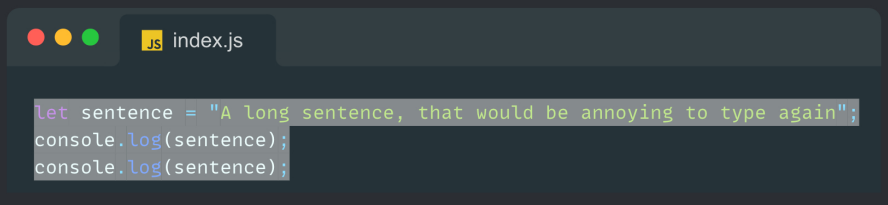
EDIT ON CODEPEN

```
// Adding Comments
// Add a single-line and a multi-line comment
where they should be, respectively.
// If the code runs without an error, you're on
the right spot! If you need to see the
solution:
// https://codepen.io/manish-
poduval/pen/LYdeOLV?editors=0010

I should be a single-line comment

console.log("hello");
console.log("world");

I should be a multi-line comment.
The function below will print out a
message for the user.

console.log("I like to code and add comments
");
```

Resources 1× 0.5× 0.25× Rerun

Storing information inside variables

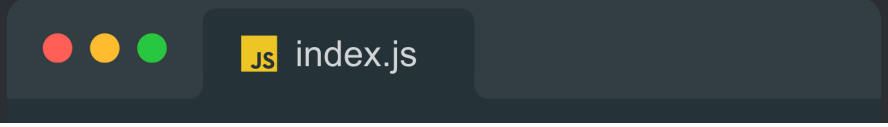
Variables allow you to store and track various items and values in your program. In many programs, values can get quite complex. Fortunately, you can declare and define a variable once and then use that variable later in your code, without having to rewrite the value each time.



This is a much simpler and more efficient process than retyping that sentence multiple times throughout a program.

Grammar and syntax

Just like human languages in linguistics, every programming language in computer science has specific grammar and syntax rules. JavaScript is no different. For example, JavaScript is case-sensitive.



```
// declare two variables
let coding = "fun";
```

In the above example, there are two distinct variables: **coding** and **Coding**. At first blush, these may seem like identical variables. But because JS is case-sensitive, the computer that reads this code will treat these two variables as completely separate items.

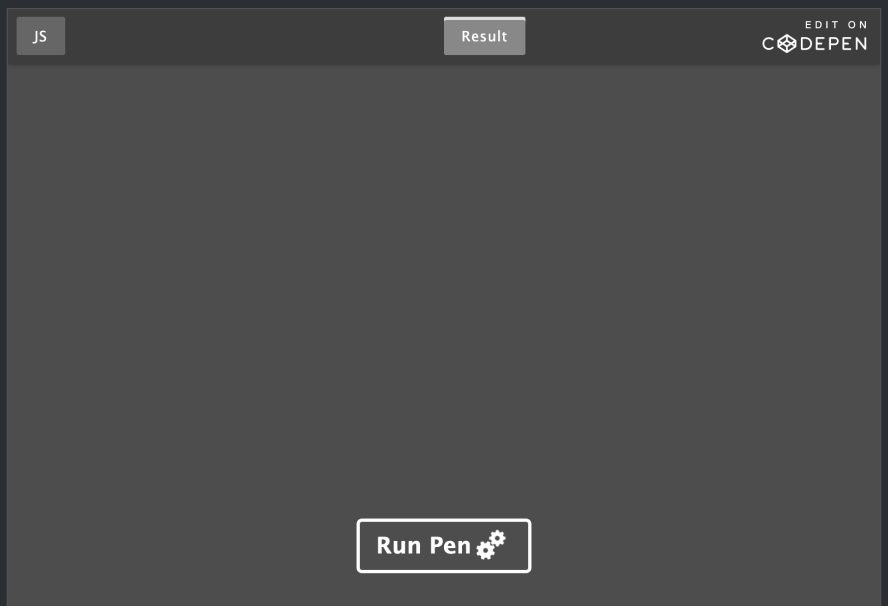
> **Try it out!** You'll recognize the code editor below as a CodePen. Click on "Edit On CodePen" and try it out yourself.

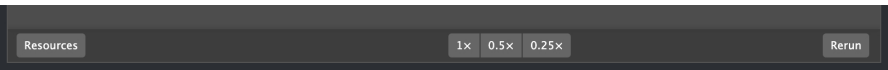
Demo: Declare variables

In the code editor below, declare two variables and print them using the **console.log()** function. (If you want, you can start by copy-and-pasting the code block above into the editor below.)

Demo: Overwriting variables

You can also *overwrite variables* in JavaScript. Simply put, this means that you can change the values of variables as needed. In fact, you can assign new or different values to your variables even if they've already been defined. To get a sense of what this looks like, watch what happens in the code below.





As you can see, when the **=** was used for a second time, **coding** changed from one value to another. So the second time the code ran **console.log**, it produced a different result!

As a developer, you'll need to understand how and when code executes. Because variables can have different values at different places in the code, as you saw above, it's essential that you know when your variables have changed or updated. In fact, a lot of the code you'll write will make modifications to variables and send those modified variables to other parts of the system. It's important to recognize these relationships, and to ensure that you're using the correct variables, and therefore the correct values, when you need them. This is how you'll produce the results that you want.

For example, if you were displaying the above message to a user of your program, you'd need to decide whether you want the user to think that coding is **weird** or **cool and fun**. You need to grab the right variable at the right time to send the right message!

Best practices

When it comes to creating variables, there are a couple of best practices to keep in mind. It's worth noting that if you don't follow these recommendations, your code won't break. But they are best practices for a reason, and you *should* follow them.

Using these recommendations, you can help other JavaScript developers understand your code and make your code much easier to maintain. Below are the most important stylistic guidelines and standards to incorporate into your coding practice.

camelCasing

Every programming language has established patterns that allow developers to understand other developers' work. In JavaScript, one such pattern is to name variables using *camelCase*.

```
thisIsCamelCasing;
alsoThisIsCamelCasing;
andThisIsToo;
camelCasingIsFun;
youWillCamelCaseToo;
```

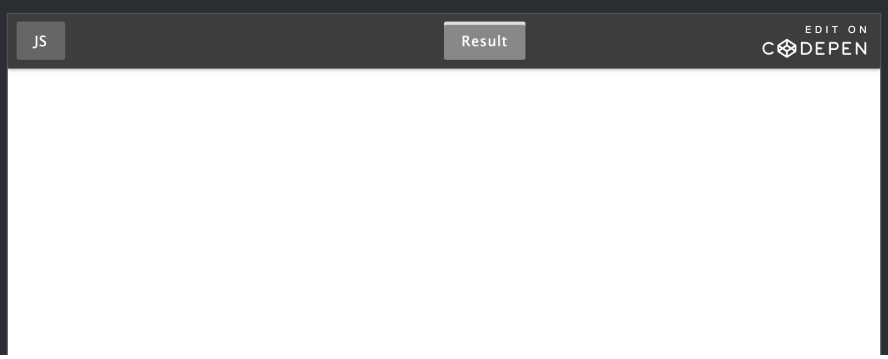
All the examples above are camelCased. A camelCased variable name starts with a lowercase letter for the first word, but every word after the first word begins with a capital letter. There are no spaces between words. As you might have guessed, the name comes from the camel, which has one or two humps in its torso. The capitalized first letters are like humps in a camel's back.

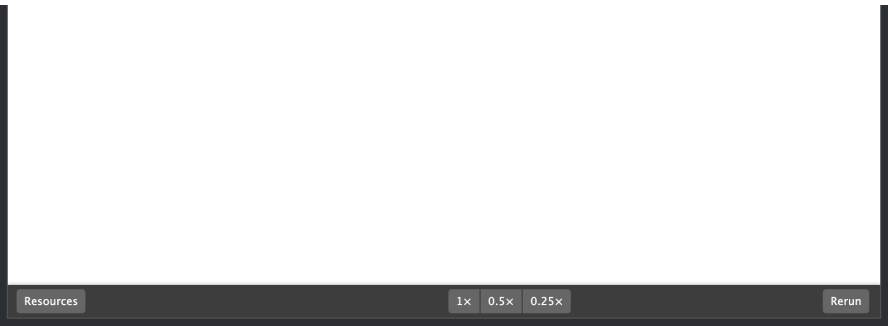
```
thisisnotcamelcasebecauseeveryletterislowercase;
```

But why camelCase? Well, consider the example above. Without camelCase, the text is much harder to read. Because you can't add spaces between words in JavaScript, you'll rely on camelCase to distinguish between words. Of course, you won't *always* use camelCase, but you'll learn about those situations later. For now, use camelCase to get a feel for the style you'll commonly use.

Demo: Camelize

In the code editor below, you'll see a custom piece of code that's called a *function*. This specific function will take any set of words separated by spaces and change them into a camelCased, space-free variable name. Play around with the input in the camelize function to see what camelCasing looks like.





Writing meaningful variable names

It's very important to name variables clearly and deliberately. Variable names should accurately reflect what those variables represent.

Why is this important? Well, as the codebase grows and more developers collaborate on projects, creating meaningful variable names becomes a necessity. If every variable has a name like **x** and **y** and **foo**, you and your fellow developers won't know what those variables actually stand for. But if your variables have more specific names, like **age**, **location**, and **phoneNumber**, it'll be easier to read and understand your variables. And you'll have a much better sense of what your code is doing.

For example, consider the **camelize** function above. It could've been named **camelCaser**, **camelCasingFunction**, or **makeItCamelCased**. There are many good (and bad!) ways to name variables and functions. As you train for your future career, practice naming your variables with clarity and specificity.

Complete lesson →