**Let**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The let statement defines a block-scoped local variable, meaning it's only defined within its currently-enclosed block. This allows us to safely avoid that variable scope issue we covered in the previous movie. In the exercise files for this movie, we still have the same scope problem. The right box is the same blue as the title and that's not what we want. This is the scope problem. Now I can use let to solve my scoping issue. And I do that by declaring a new variable inside the function scope using let. Save this, go back in the browser. And now we have the correct colors. Looking at this, you may be wondering, How can I do this? I'm effectively creating two variables with the same name. We have var color and then we have let color down here. The reason is we're now working inside a local scope of just this function, and inside this function, we can now declare all new variables using let and they will only exist inside that function. They also ignore the other things that are happening outside. To prove my point, we can also change the global variable here to a let and everything will still work. So now, we're effectively working with two lets: One that sits in global scope and one other one that sits in local scope and only exists in here. This example shows you how important local scope is. When we're working in local scope, we're effectively treating this function here as its own separate code base with its own variables, separate from everything else, and that makes it so powerful. Now, let's see what happens if I try to call a locally-scoped let from outside its local scope. To do that, I need to change the name of this variable here. So I'll call it titleColor. I'll change it down here too, so that everything works. And then I will console log out the titleColor value from inside its scope. So we'll say... Let's see, inside, comma and then we get the proper output. Everything still works in the browser and I can open my console and you'll see down here we now have the output inside blue. So everything is working as expected. But what happens if I now copy this and then paste it outside the function scope? Before we preview it, I want you to make a guess. What happens when we try to call a locally-scoped variable outside its scope? Save it, go back in the browser. And two things happen. First of all, the box up here on the right-hand side is now green. That's the default color that was supplied before JavaScript intervened. Secondly, in the console, it tells us: Uncaught ReferenceError: titleColor is not defined at script.js line 21. Script.js line 21 is what we just inserted. So that means titleColor is not defined here. That's because title color is block-scoped to the heading color function and does not exist anywhere else. So when we're trying to call it outside, nothing happens. Well, that's not true. What happens is JavaScript stops rendering at this point because there's an error, and that's why we're seeing the green color in the right-hand box. These two declarations down at the bottom here never take place because JavaScript stalls at this line since there's an error. This may look like a problem, but this is actually the reason why we have let and we have local scope. By using a block-scope variable, we very quickly see if we're using the variable in a place we shouldn't be using it because JavaScript crashes and gives us an error. It also gives us complete control over when a variable exists and what we can use it for, which brings up an important question: When should you use a var? And when should you use a let? Here's a good rule of thumb: For most situations, when you want to use a changeable or mutable variable, you should use a let. That includes when you want to use it in global scope, because when you declare a let in global scope, it will apply everywhere except where you re-declare it. The var is only really useful if you want a mutable variable with global scope all the time, but that situation is quite rare and is a special case. So the default is, when you want a changeable or mutable variable use a let.

Give fee