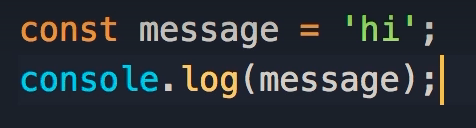
# Section 8 - Local vs Global Scope

## Scope

In this section, we will be covering a very important concept called scope.



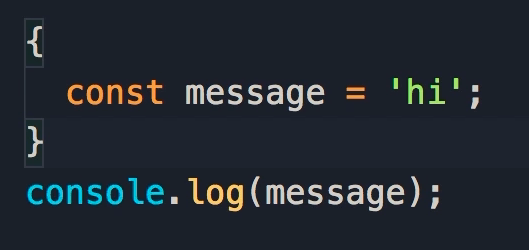
Here we define a constant called message and set it to ‘hi’.



When we log message, we get ‘hi’ in our console

## Code Block

What would happen if we place our constant in a code block?



We get this error:

Uncaught ReferenceError: message is not defined



This is all about scoping.

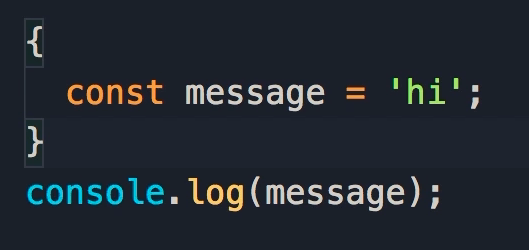
## What is Scope?

Scope of a variable or a constant, determines where that variable or constant is accessible.

When we declare variables or constants with let or const, their scope is limited to the block in which they are defined.

## Scoping

That means the message variable is accessible within the code block, but not outside it.

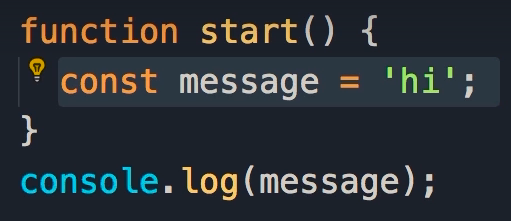




So why would our message constant be in a code block?

## Scope of Functions

The code block could be part of a function.

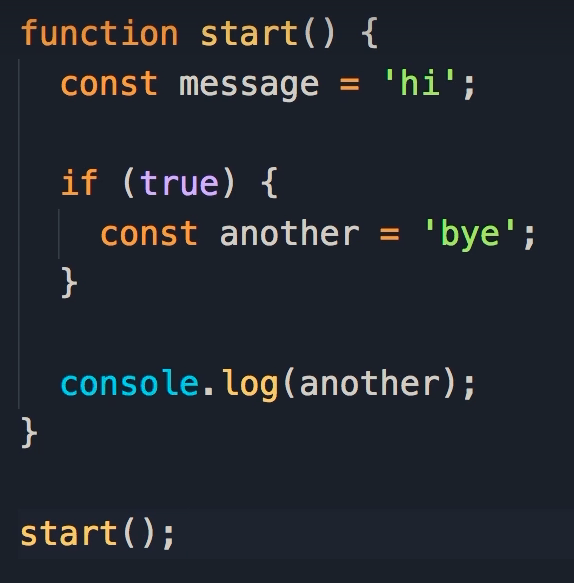


Here we have a function called start.

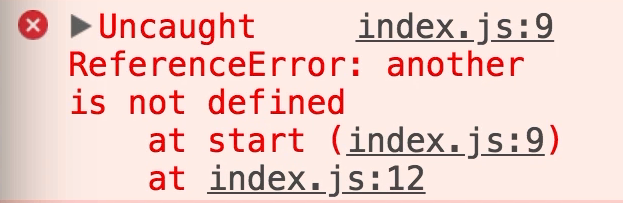
Because the scope of this constant is limited to the block in which it's defined, we cannot access this constant outside of the start function.

## Scope of If Statements

The same is true when we declare a variable or constant in an if block.



Here we set up an if statement, within which we have declared a constant called another.



This constant is only accessible in this block, if I go out of this block, and try to log it on the console, we're going to get an error.

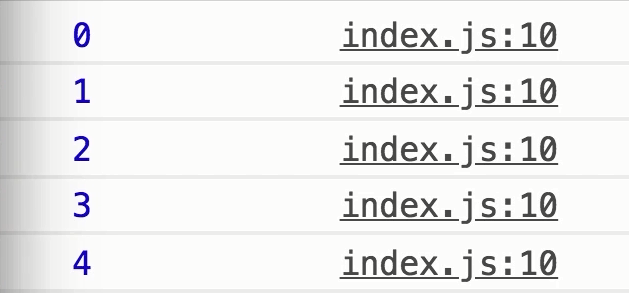
## Scope of Loops

We have the same concept with loops.



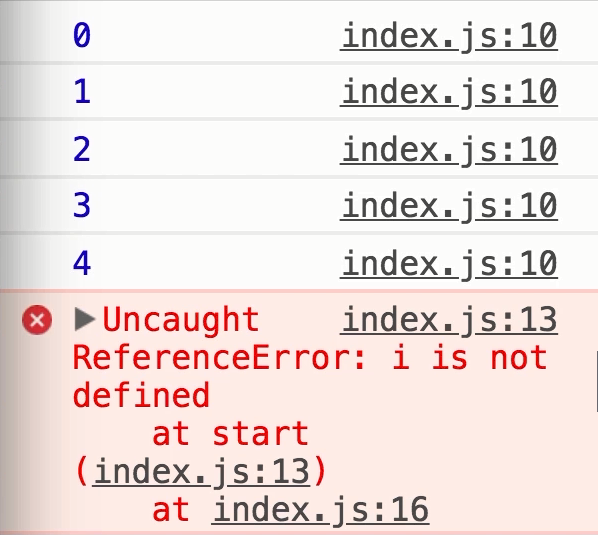
Here we define a loop to run 5 times.

Note that i is now a variable only available in this block.



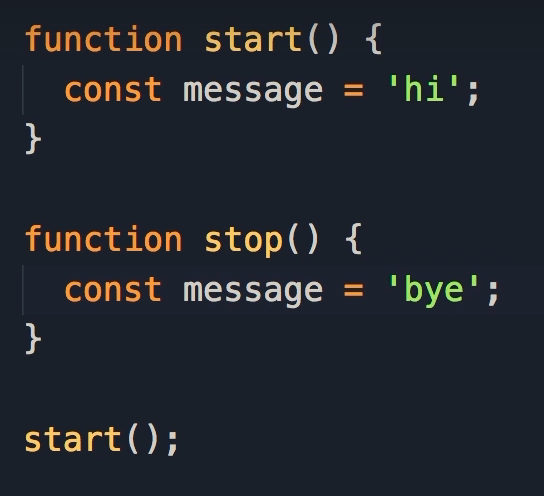
When we log i, we get the numbers 0-4 in our console.

If we log i outside of the loop, we will get a reference error.



## Scope of separate Functions

Here we have 2 functions: start() and stop().



Each of them have a constant called message, with a different value assigned.

This is perfectly valid JavaScript code.

Because the scope of the constant is limited to the block in which it's defined.

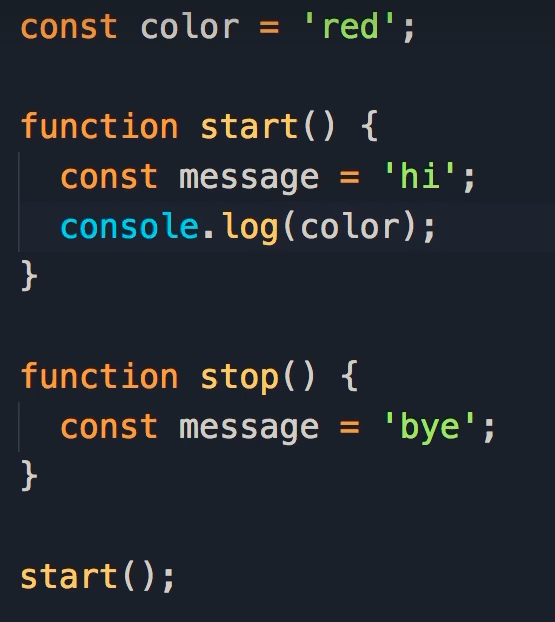
So we can have two variables or constants with the same name but different functions.

That's perfectly valid.

## Global Scope

What if we define a variable or constant outside of a function?

Here we have a constant called color and set it to red.



Since this variable is not surrounded by any code blocks, it has global scope.

Global means this constant is accessible everywhere, globally.

In our start() function, we log the color constant.



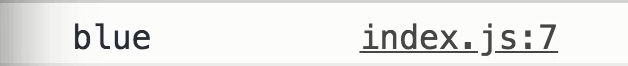
In our console we see we have access to the color constant, so this constant is accessible to all functions we have in our program.

## Global vs Local Scope

Here we declared another constant with the same name within our function, and assigned it the value of ‘blue’



When we log color within the function, blue is displayed on the console.



Therefore, local variables and constants take precedence over global variables and constants.

In general, you should avoid defining global variables or constants.

This is considered bad practice.

## Metaphor

Imagine that a constant/variable is a toothbrush and each function is a person.

You don't want to share a toothbrush with multiple people.

Each person should have their own toothbrush.

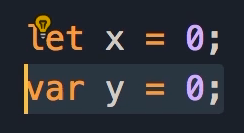
You should avoid defining global variables or constants because they're accessible globally, and each function can accidentally change their value, and this will lead to all kinds of bugs and issues in our programs.

# Section 9 - Let vs Var

## Declaring Variables

So far we have been declaring variables using the let keyword.

At the beginning of course, it was briefly mentioned that there is another way to declare variables, using the var keyword.

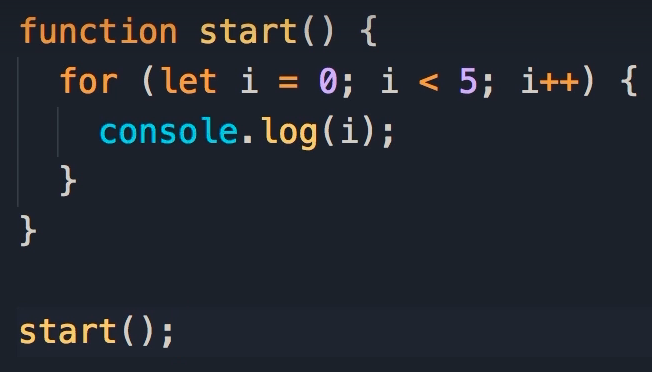


We see this in a lot of JavaScript code online.

In this sectionwe're going to look at the var keyword, problems with the var keyword, and why you should avoid it.

## Let vs Var

Here we have defined a function called start, which contains a loop.



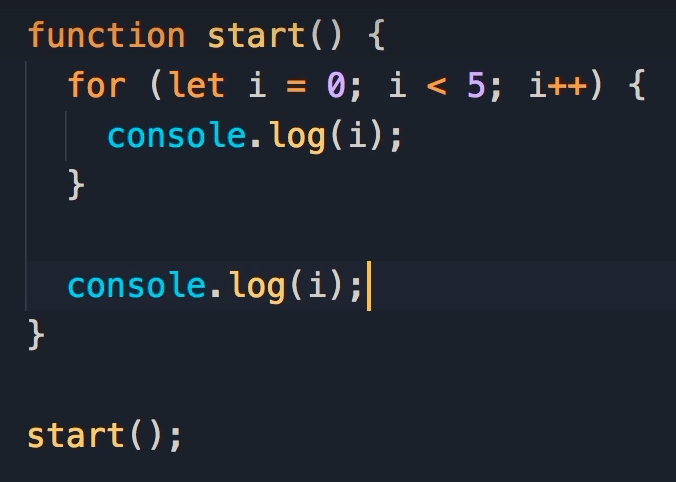
We initiate the loop using:

let i = 0;

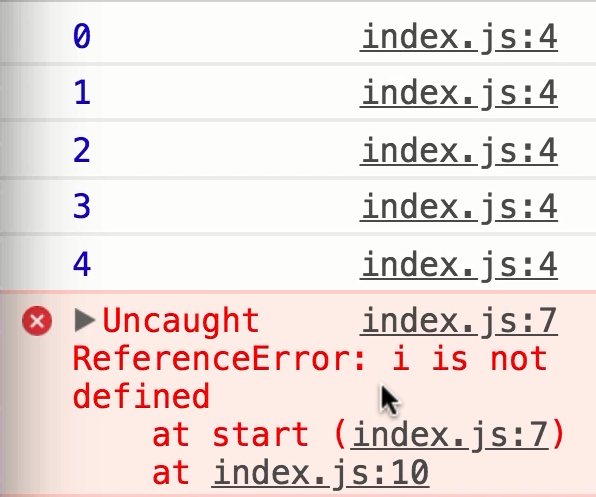


When we call the start function, the numbers 0-4 are logged on the console.

In the last section, you learnt that i is only accessible within the code block.

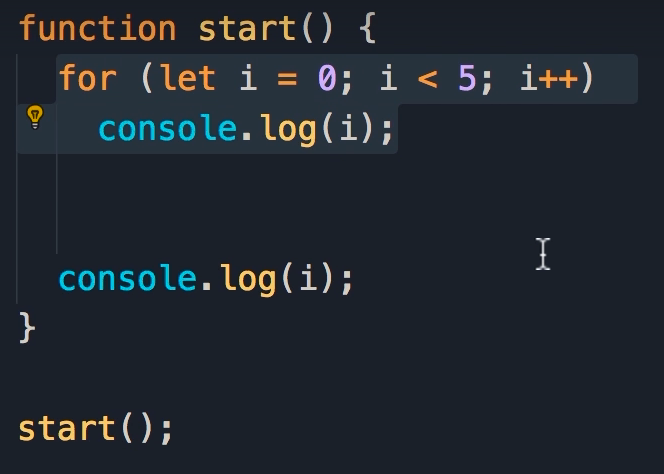


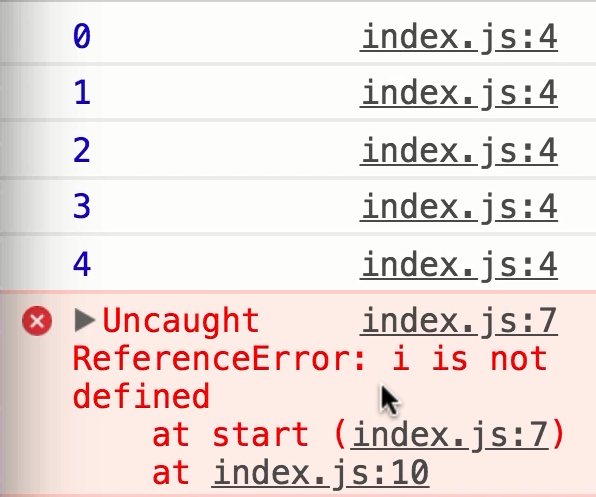
If we try to log i outside of the code block, we get an error.



## Code Blocks

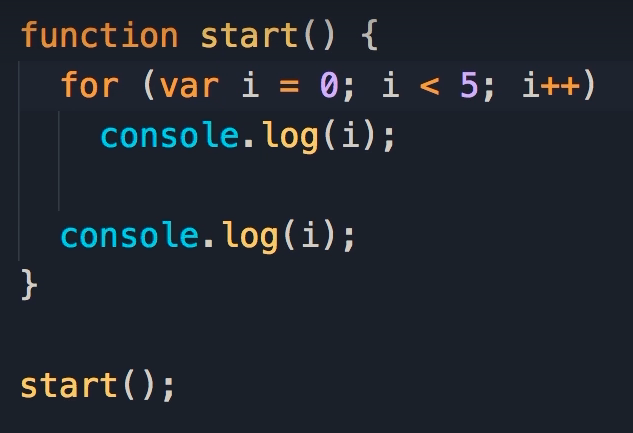
\* Note: We still have the same concept if we remove the curly braces from the for loop, because here we have a single statement, so the scope of i is still this block here.



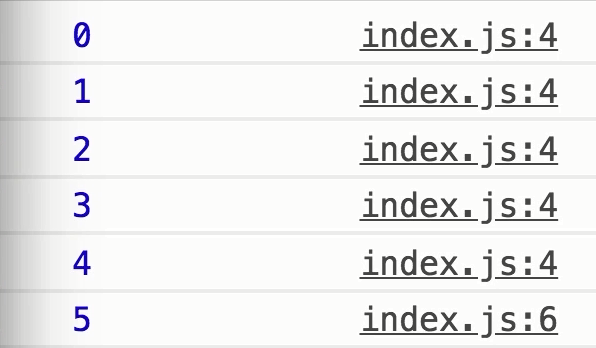


## Var vs Let

When we replace let with var in our loop, the error disappears.



This shows that i is accessible outside of the code block.



In the last iteration, i becomes 5.

Because 5 is not less than 5, the loop terminates, then underneath we display the current value of i.

## The Var Keyword

This is the issue with the var keyword.

When you declare a variable with var, it's scope is not limited to the block in which it's defined.

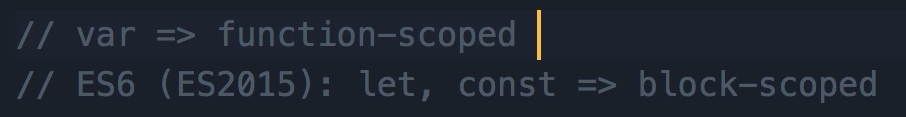
It's limited to the function in which it's defined, and this is not how most or all programming languages work.

It's one of those weird things in JavaScript that we have had for a long time.

## Var vs Let vs Const

Before ES6, var was the only way to declare variables and constants.

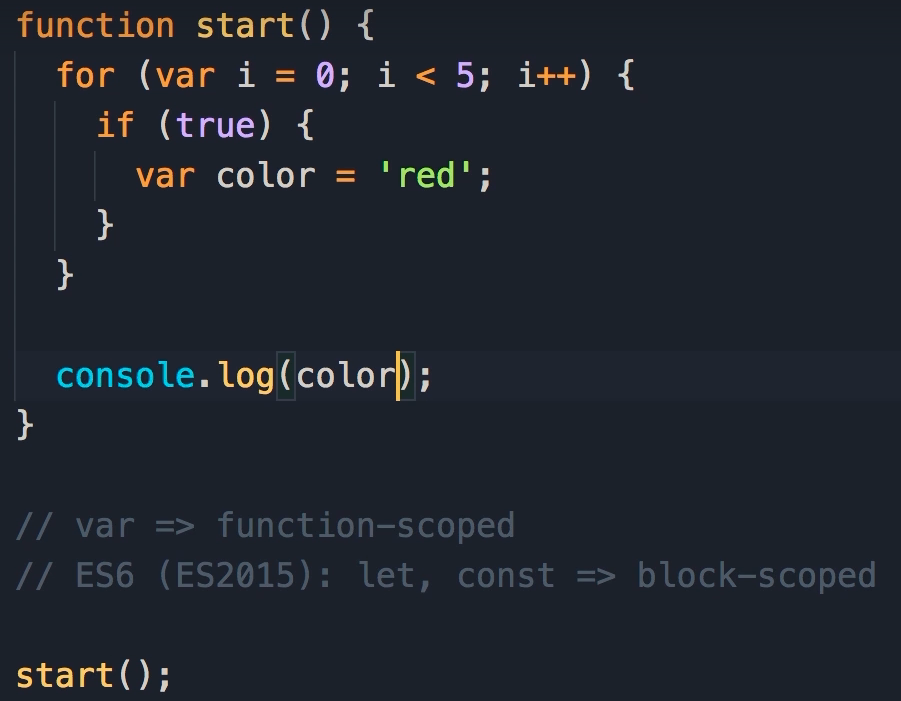
Starting from ES6, also called ES2015, we have two new keywords let and const, to define different variables and constants.



These two keywords create block scope variables, but var creates function scoped variables.

## Let vs Var

Inside the for loop, we have added an if statement.



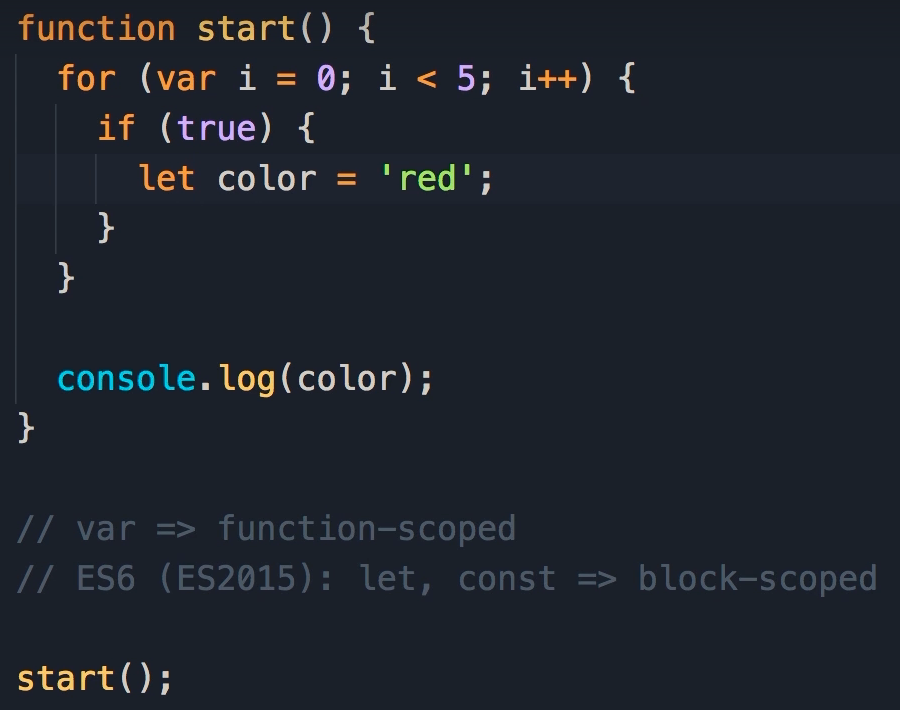
Inside the if block, we declare a variable using var, and set it to red.

Technically in almost all programming languages out there, this variable should only be accessible in this block.

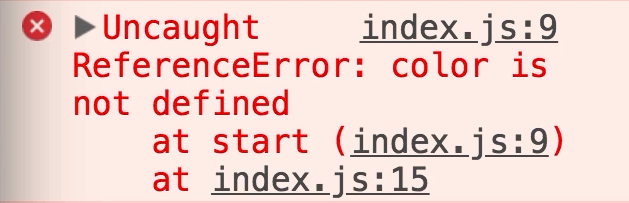
But because we have used var, color is accessible anywhere in this function.



So, at the end of this function, you can log color, and red will be displayed in the console.



If we replace var with let, we get an error because color is not accessible here.



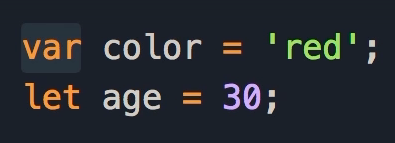
That’s how most programming languages work.

This is the first problem with var.

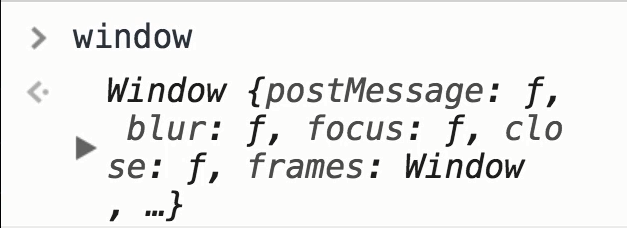
The second issue is with global variables.

## Global Variable

Here we have 2 variables declared with var and const respectively.



When we use var outside of a function, it creates a global variable and attaches that global variable to the window object in the browser.



In browsers we have a window object, which is a complex object with lots of properties and methods.

When you want to build frontend applications you're going to work with this window object a lot.

What matters here is the var keyword attaches the color variable to the window object.