Published in JavaScript in Plain English

Andrew Koenig-Bautista Follow
Apr 6, 2020 · 3 min read · · Listen

How Hoisting Works With 'let' and 'const' in Javascript

A frequent interview concept

Photo by Lance Anderson on Unsplash

The term hoisting is confusing

I believe that one of the first and foremost reasons people struggle to understand *hoisting* is because the term itself is somewhat misleading. The Merriam-Webster definition of the word *hoist* is “an act of raising or lifting”.

This might lead one to assume that hoisting involves written code being physically rearranged somehow. This is not true.

Instead, the term *hoisting* is used as a kind of simile to describe a process that occurs while the JavaScript engine interprets written JavaScript code.

How is JavaScript code interpreted?

called a **Global Execution Context.**

The JavaScript engine interprets the JavaScript written within this Global Execution Context in two separate phases; **compilation** and **execution.**

Compilation

During the compilation phase, JavaScript parses the written code on the lookout for all function or variable declarations. This includes:

```
-let
-const
-class
-var
-function
```

When compiling these keywords, JavaScript creates a unique space in memory for each declared variable it comes across. This process of “lifting” the variable and giving it a space in memory is called hoisting.

Typically, hoisting is described as the moving of *variable* and *function* declarations to the top of their (global or function) scope.

However, the variables **do not** move **at all.**

What actually happens is that during the compilation phase declared variables and functions are stored in memory before the rest of your code is read, thus the illusion of “moving” to the top of their scope.

Execution

After the first phase has finished and all the declared variables have been hoisted, the second phase begins; execution. The interpreter goes back up to the first line of code and works its way down again, this time assigning variables values and processing functions.

declarations in the hoisting process is in their initialization.

During the compilation phase, JavaScript variables declared with `var` and `function` are hoisted and automatically initialized to `undefined`.

```
console.log(name) // undefined
var name = "Andrew";
```

In the above example, JavaScript first runs its compilation phase and looks for variable declarations. It comes across `var name`, hoists that variable and automatically assigns it a value of `undefined`.

Contrastingly, variables declared with `let`, `const`, and `class` are hoisted but remain uninitialized:

```
console.log(name); // Uncaught ReferenceError: name is not defined
let name = "Andrew";
```

These variable declarations only become initialized when they are evaluated during runtime. The time between these variables being declared and being evaluated is referred to as the **temporal dead zone**. If you try to access these variables within this dead zone, you will get the reference error above.

To walk through the second example, JavaScript runs its compilation phase and sees `let name`, hoists that variable, but does not initialize it. Next, in the execution phase, `console.log()` is invoked and passed the argument `name`.

Because the variable has not been initialized, it has not been assigned a value, and thus the reference error is returned stating that `name` is not defined.

JavaScript engine.

It’s not an error to reference `let` and `const` variables in code above their declaration as long as that code is not executed before their declaration.

For example, this code works fine:

```
function greetings() {
  console.log(`Hello, ${name}`);
}

let name = "Hannah";

greetings();
// "Hello Hannah"
```

hoistingSuccess.js hosted with ❤ by GitHub view raw

However, this will result in a reference error:

```
function greetings() {
 console.log(`Hello, ${name}`);
}

greetings();

let name = "Hannah";
// Uncaught ReferenceError: name is not defined
```

hoistingFail.js hosted with ❤ by GitHub view raw

As I hope you understand at this point, this error is generated because `greetings()` was executed before the variable `name` was declared.

A note from JavaScript In Plain English

We are always interested in helping to promote quality content. If you have an article

Thanks to Zack Shapiro