JavaScript
TUTORIAL

JavaScript Hoisting

Summary: in this tutorial, you'll learn about JavaScript hoisting and how it works under the hood.

Introduction to the JavaScript hoisting

When the JavaScript engine executes the JavaScript code, it creates the global execution context. The global execution context has two phases:

- Creation
- Execution

During the creation phase, the JavaScript engine moves the variable and function declarations to the top of your code. This is known as hoisting in JavaScript.

Variable hoisting

Variable hoisting means the JavaScript engine moves the variable declarations to the top of the script. For example, the following example declares the `counter` variable and initialize its value to `1`:

```
console.log(counter); // 👉 undefined
var counter = 1;
```

In this example, we reference the `counter` variable before the declaration.

However, the first line of code doesn't cause an error. The reason is that the JavaScript engine moves the variable declaration to the top of the script.

Technically, the code looks like the following in the execution phase:

```
var counter;

console.log(counter); // 👉 undefined
counter = 1;
```

During the creation phase of the global execution context, the JavaScript engine places the variable `counter` in the memory and initializes its value to `undefined`.

The let keyword

The following declares the variable `counter` with the `let` keyword:

```
console.log(counter);
let counter = 1;
```

The JavaScript issues the following error:

```
"ReferenceError: Cannot access 'counter' before initialization
```

The error message explains that the `counter` variable is already in the heap memory. However, it hasn't been initialized.

Behind the scenes, the JavaScript engine hoists the variable declarations that use the `let` keyword. However, it doesn't initialize the `let` variables.

Notice that if you access a variable that doesn't exist, the JavaScript will throw a different error:

```
console.log(alien);
let counter = 1;
```

Here is the error:

```
"ReferenceError: alien is not defined
```

Function hoisting

Like variables, the JavaScript engine also hoists the function declarations. This means that the JavaScript engine also moves the function declarations to the top of the script. For example:

```
let x = 20,
  y = 10;

let result = add(x, y);
console.log(result); // 👉 30

function add(a, b) {
  return a + b;
}
```

Output:

```
30
```

In this example, we called the `add()` function before defining it. The above code is equivalent to the following:

```
function add(a, b){
    return a + b;
}

let x = 20,
    y = 10;

let result = add(x,y);
console.log(result); // 👉 30
```

During the creation phase of the execution context, the JavaScript engine places the `add()` function declaration in the heap memory. To be precise, the JavaScript engine creates an object of the `Function` type and a function reference `add` that refers to the function object.

Function expressions

The following example changes the `add` from a regular function to a function expression:

```
let x = 20,
    y = 10;

let result = add(x,y); // ❌ Uncaught ReferenceError: add is not defined
console.log(result);

let add = function(x, y) {
    return x + y;
}
```

If you execute the code, the following error will occur:

```
Uncaught ReferenceError: add is not defined
```

During the creation phase of the global execution context, the JavaScript engine creates the `add` variable in the memory and initializes its value to `undefined`.

When executing the following code, the `add` is `undefined`, hence, it isn't a function:

```
let result = add(x,y);
```

The `add` variable is assigned to an anonymous function only during the execution phase of the global execution context.

Arrow functions

The following example changes the `add` function expression to the arrow function:

```
let x = 20,
    y = 10;

let result = add(x,y); // ❌ Uncaught ReferenceError: add is not defined
```

```
console.log(result);

let add = (x, y) => x + y;
```

The code also issues the same error as the function expression example because arrow functions are syntactic sugar for defining function expressions.

```
Uncaught ReferenceError: add is not defined
```

Similar to the functions expressions, arrow functions are not hoisted.

Summary

- JavaScript hoisting occurs during the creation phase of the execution context that moves the variable and function declarations to the top of the script.
- The JavaScript engine hoists the variables declared using the `let` keyword, but it doesn't initialize them as the variables declared with the `var` keyword.
- The JavaScript engine doesn't hoist the function expressions and arrow functions.