Variable and Function Hoisting in JavaScript

Originally published in the A Drip of JavaScript newsletter.

One of the trickier aspects of JavaScript for new JavaScript developers is the fact that variables and functions are "hoisted." Rather than being available after their declaration, they might actually be available beforehand. How does that work? Let's take a look at variable hoisting first.

```
// ReferenceError: noSuchVariable is not defined
console.log(noSuchVariable);
```

This is more or less what one would expect. An error is thrown when you try to access the value of a variable that doesn't exist. But what about this case?

```
// Outputs: undefined
console.log(declaredLater);

var declaredLater = "Now it's defined!";

// Outputs: "Now it's defined!"
console.log(declaredLater);
```

What is going on here? It turns out that JavaScript treats variables which will be declared later on in a function differently than variables that are not declared at all. Basically, the JavaScript interpreter "looks ahead" to find all the variable declarations

and "hoists" them to the top of the function. Which means that the example above is equivalent to this:

```
var declaredLater;

// Outputs: undefined
console.log(declaredLater);

declaredLater = "Now it's defined!";

// Outputs: "Now it's defined!"
console.log(declaredLater);
```

One case where this is particularly likely to bite new JavaScript developers is when reusing variable names between an inner and outer scope. For example:

```
var name = "Baggins";

(function () {
    // Outputs: "Original name was undefined"
    console.log("Original name was " + name);

    var name = "Underhill";

    // Outputs: "New name is Underhill"
    console.log("New name is " + name);
})();
```

In cases like this, the developer probably expected `name` to retain its value from the outer scope until the point that `name` was declared in the inner scope. But due to hoisting, `name` is `undefined` instead.

Because of this behavior JavaScript linters and style guides often recommend putting all variable declarations at the top of the function so that you won't be caught by surprise.

So that covers variable hoisting, but what about function hoisting? Despite both being called "hoisting," the behavior is actually quite different. Unlike variables, a function declaration doesn't just hoist the function's name. It also hoists the actual function definition.

```
// Outputs: "Yes!"
isItHoisted();

function isItHoisted() {
    console.log("Yes!");
}
```

As you can see, the JavaScript interpreter allows you to use the function before the point at which it was declared in the source code. This is useful because it allows you to express your high-level logic at the beginning of your source code rather than the end, communicating your intentions more clearly.

```
travelToMountDoom();
destroyTheRing();

function travelToMountDoom() { /* Traveling */ }
function destroyTheRing() { /* Destruction */ }
```

However, **function definition hoisting only occurs for function declarations**, not function expressions. For example:

```
// Outputs: "Definition hoisted!"
definitionHoisted();

// TypeError: undefined is not a function
definitionNotHoisted();

function definitionHoisted() {
    console.log("Definition hoisted!");
}
```

```
var definitionNotHoisted = function () {
    console.log("Definition not hoisted!");
};
```

Here we see the interaction of two different types of hoisting. Our variable `definitionNotHoisted` has its declaration hoisted (thus it is `undefined`), but not its function definition (thus the `TypeError`.)

You might be wondering what happens if you use a named function expression.

```
// ReferenceError: funcName is not defined
funcName();

// TypeError: undefined is not a function
varName();

var varName = function funcName() {
    console.log("Definition not hoisted!");
};
```

As you can see, the function's name doesn't get hoisted if it is part of a function expression.

And that is how variable and function hoisting works in JavaScript.

Thanks for reading!

Joshua Clanton
