


Basic JavaScript Part 12: Function Hoisting

 March 24th, 2011

Here are the links to the previous installments:

1. [Functions](#)
2. [Objects](#)
3. [Prototypes](#)
4. [Enforcing New on Constructor Functions](#)
5. [Hoisting](#)
6. [Automatic Semicolon Insertion](#)
7. [Static Properties and Methods](#)
8. [Namespaces](#)
9. [Reusing Methods of Other Objects](#)
10. [The Module Pattern](#)
11. [Functional Initialization](#)

In a [previous post](#) I already discussed the phenomenon of hoisting in JavaScript. In that post I showed the effects of variable hoisting and why it's important to declare all variables at the top of a function body. For this post I want to briefly focus on function hoisting. Let's start off with an example to illustrate this concept.

```
functionExpression();           // undefined
functionDeclaration();          // "Function declaration called."
```

```
var functionExpression = function() {
  console.log('Function expression called.');
```

```
};
```

```
functionExpression();           // "Function expression called."
functionDeclaration();          // "Function declaration called."
```

```
function functionDeclaration() {
  console.log('Function declaration called.');
```

```
}
```

```
functionExpression();           // "Function expression called."
functionDeclaration();          // "Function declaration called."
```

In order to understand what's going on here, we first need to understand the distinction between a function expression and a function declaration. As its name implies, a function expression defines a function as part of an expression (in this case assigning it to a variable). These kind of functions can either be anonymous or they can have a name.

```
//
// Anonymous function expression
//
var functionExpression = function() {
  console.log('Function expression called.');
```

```
};
```

On the other hand, a function declaration is always defined as a named function without being part of any expression.

So, for the example shown earlier, the function expression can only be called after it has been defined

while the function declaration can be executed both before and after its definition. Let's look at how JavaScript actually interprets this code in order to explain why it behaves that way.

```
var functionExpression,           // undefined
    functionDeclaration = function() {
        console.log('Function declaration called.');
```



```
};

functionExpression();             // Still undefined
functionDeclaration();           // "Function declaration called."

// The assignment expression is still left at the original location
// although the variable declaration has been moved to the top.
functionExpression = function() {
    console.log('Function expression called.');
```



```
};

functionExpression();             // "Function expression called."
functionDeclaration();           // "Function declaration called."

// Here we originally defined our function declaration
// which has been completely moved to the top.

functionExpression();           // "Function expression called."
functionDeclaration();          // "Function declaration called."
```

JavaScript turns our function declaration into a function expression and hoists it to the top. Here we see the same thing happening to our function expression as I explained in the previous post on variable hoisting. This also explains why the first call of our function expression results in an error being thrown because the variable is undefined.

So basically, JavaScript applies different rules when it comes to function hoisting depending on whether you have a function expression or a function declaration. A function declaration is fully hoisted while a function expression follows the same rules as variable hoisting. It definitely took me a while to wrap my head around this.

Until next time.



Jan Van Ryswyck