Basic JavaScript Part 6: Automatic Semicolon Insertion

January 12th, 2011

Here are the links to the previous installments:

1. Functions
2. Objects
3. Prototypes
4. Enforcing New on Constructor Functions
5. Hoisting

For this post, I'm going to discuss a feature of JavaScript that can easily get you in trouble. Semicolons in JavaScript are not mandatory. This means that if you forget to place a semicolon after some statement, JavaScript isn't going to complain about it. But most JavaScript engines make up for this lack of consistency by automatically adding a semicolon for you at runtime. This is were you can get in trouble and leave you wondering what is going on. Let's take a look at the following code snippet:

```
function shoeFactory() {
    return
    {
        shoeSize: 48
    };
}

var shoe = shoeFactory();
console.log(shoe);
```

You might expect that calling this *shoeFactory* function is going to return an object with a*shoeSize* property set to the number 48. Wrong! When executing this code, the *shoeFactory*method actually returns *undefined* instead of a new object.

What really happens at runtime is this:

```
function shoeFactory() {
    return;     // Automatic semicolon insertion (returns undefined)
    {    // Unreachable code block
        shoeSize: 48
    };
}
```

The way to get this code to behave properly in this case is by placing the opening curly brace of the object literal at the end of the line that contains the *return* statement.

```
function shoeFactory() {
    return {
        shoeSize: 48
    };
}
```

The placement of semicolons might come off as a matter of preference. But the way you outline braces in JavaScript can be very important for your JavaScript code to behave properly. It's highly recommended to not rely on the automatic semicolon insertion mechanism and to always try to put semicolons on the right place in the code. Also trying to outline curly braces as shown above can save you some headaches in case a semicolon is forgotten somewhere in the code.

That's it for this one. Until next time.