Basic JavaScript Part 5: Hoisting

December 24th, 2010

Here are the links to the previous installments:

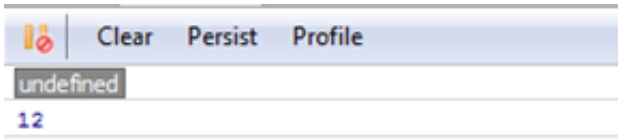
1. Functions
2. Objects
3. Prototypes
4. Enforcing New on Constructor Functions

I just wanted to quickly share a little tidbit that I ran into the other day while I was messing with JavaScript again. Try to guess what the output is going to be when running the following code snippet:

```
var num = 56;
function calculateSomething() {
    console.log(num);
    var num = 12;
    console.log(num);
}

calculateSomething();
```

Without further ado, this is the output shown in the console window:



I must admit that this had me beat the first time I saw this, but it really makes perfect sense.

JavaScript doesn't support block scope but instead it makes use of function scope. Block scope means that variables declared in a block are not visible to code outside that block. We all know that the following C#code doesn't compile for exactly that reason:

```
public void Stuff()
{
    {
        var i = 2;
    }

    // Compiler error: The name 'i' does not exist in the current context.
    Console.WriteLine(i);
}
```

But function scope means that all variables and parameters declared inside a function are visible everywhere within that function, even before the variable has been declared. This is the behavior that we see by running the code snippet shown earlier.

Just as with C#, we can put a variable declaration anywhere in our JavaScript function as we did earlier. In that case, JavaScript will just act as if the variable has been declared at the top of the function. This behavior is called *hoisting*. This means that it is valid to use this variable as long as it has been declared somewhere within the function. Going back to our previous sample, the net result is that JavaScript will interpret the function as something like this:

```
var num = 56;
function calculateSomething() {
    var num;            // undefined
    console.log(num);    // outputs 'undefined'
```

```
    num = 12;            // 12
    console.log(num);    // outputs '12'
}

calculateSomething();
```

Now I know why Douglas Crockford advised in his book JavaScript – The Good Parts to declare all variables at the top of the function body. In order to prevent nasty side-effects like this to happen, I think it's best taking up on that advice.

Till next time.