A Drip of JavaScript

Default Parameters in JavaScript

Originally published in the A Drip of JavaScript newsletter.

Some languages — like Ruby, CoffeeScript, and upcoming versions of JavaScript — have the ability to declare default parameters when defining a function. It works like this:

```
function myFunc(param1, param2 = "second string") {
    console.log(param1, param2);
}

// Outputs: "first string" and "second string"
myFunc("first string");

// Outputs: "first string" and "second string version 2"
myFunc("first string", "second string version 2");
```

Unfortunately, this construct isn't available in current versions of JavaScript. So what can we do to achieve this behavior with our current set of tools?

The simplest solution looks like this:

```
function myFunc(param1, param2) {
    if (param2 === undefined) {
        param2 = "second string";
    }
    console.log(param1, param2);
}

// Outputs: "first string" and "second string version 2"
myFunc("first string", "second string version 2");
```

This relies on the fact that a parameter omitted at call time is always `undefined`. And it's a good solution if you have only one parameter to deal with. But what if you have several?

Well if you have more than a few parameters, you should probably be passing in an object parameter, as that has the advantage of explicitly naming everything. And if you're passing in an object parameter, it makes sense to declare your defaults the same way.

```
function myFunc(paramObject) {
    var defaultParams = {
        param1: "first string",
        param2: "second string",
        param3: "third string"
    };

    var finalParams = defaultParams;

    // We iterate over each property of the paramObject
    for (var key in paramObject) {
        // If the current property wasn't inherited, proceed
        if (paramObject.hasOwnProperty(key)) {
            // If the current property is defined,
            // add it to finalParams
            if (paramObject[key] !== undefined) {
                finalParams[key] = paramObject[key];
            }
        }
    }
    console.log(finalParams.param1,
                finalParams.param2,
                finalParams.param3);
}
```

That's a little unwieldy, so if you're using this pattern a lot, it makes sense to extract the iteration and filtering logic into its own function. Fortunately, the clever folks who write jQuery and Underscore have done just that with their respective `extend` methods.

Here's an updated version which uses Underscore's extend to achieve the same result.

```
function myFunc(paramObject) {
    var defaultParams = {
        param1: "first string",
        param2: "second string",
        param3: "third string"
    };

    var finalParams = _.extend(defaultParams, paramObject);

    console.log(finalParams.param1,
                finalParams.param2,
                finalParams.param3);
}

// Outputs:
// "My own string" and "second string" and "third string"
myFunc({param1: "My own string"});
```

And that's how you can get default parameters in current versions of JavaScript.