A Drip of JavaScript

Using JavaScript's 'toString' Method

Originally published in the A Drip of JavaScript newsletter.

Have you ever wondered why if you try to alert a plain old JavaScript object, you get the text `[object Object]`? That's due to JavaScript's built-in `toString` method. Let's take a look at how it works.

Note: In some places, the examples will use `alert` rather than `console.log` in order to bypass the special handling that consoles give to objects.

```javascript
// Alerts: '[object Object]'
alert({});

// Outputs: "function() {}"
console.log(function() {});

// Outputs:
// "Thu Oct 11 1962 00:00:00 GMT-0400 (Eastern Daylight Time)"
// May vary depending on your clock settings
console.log(new Date("October 11, 1962"));
```

What do each of these things have in common? They are each providing a string representation of their values, but their internal values are quite different. For instance, internally JavaScript dates are measured in milliseconds since midnight of January 1, 1970 UTC.

So how are these objects providing that string representation? All three of them have a prototype method called `toString` which converts their internal value into something that makes sense as a string of text.

Let's suppose that we are creating a new type of object and want to give a reasonable string representation to use in messages to a user. Without using `toString`, our code might look like this:

```
function Color (r, g, b) {
    this.r = r;
    this.g = g;
    this.b = b;
    this.text = "rgb(" + r + ", " + g + ", " + b + ")";
}

var red = new Color(255, 0, 0);

// Outputs: "rgb(255, 0, 0)"
alert(red.text);
```

But there are a couple of problems with this. First of all, the `text` property will get out of sync with the actual values if they are changed. Second, we have to remember to explicitly use `red.text` instead of just using the value of `red`.

Here's how that code looks after it is rewritten to use `toString` instead:

```
function Color (r, g, b) {
    this.r = r;
    this.g = g;
    this.b = b;
}

Color.prototype.toString = function () {
    return "rgb(" + this.r + ", " +
                    this.g + ", " +
                    this.b + ")";
```

```
};

var red = new Color(255, 0, 0);

// Alerts: "rgb(255, 0, 0)"
alert(red);
```

By moving the string representation into `toString` on the prototype we get two benefits. The string representation will always stay up to date with the actual values. And any time that we want to deal with the string value, `toString` will be implicitly called by the JavaScript interpreter. For instance:

```
var message = "Red is: " + red;

// Outputs: "Red is: rgb(255, 0, 0)"
console.log(message);
```

While you may not need to use `toString` every day, it's a good tool to have in your utility belt.
