A Drip of JavaScript

Testing Array Contents with Array#some

Originally published in the A Drip of JavaScript newsletter.

When working with arrays, it is quite common to perform different actions based on whether or not the array contains a particular item. A fairly straightforward implementation of this pattern can be found below:

```
planets = [
    "mercury",
    "venus",
    "earth",
    "mars",
    "jupiter",
    "saturn",
    "uranus",
    "neptune"
];

// Default to false
var containsPluto = false;

for (var i = 0; i < planets.length && !containsPluto; i++) {
    if (planets[i] === "pluto") {
        containsPluto = true;
    }
}
```

```
// Outputs: false
console.log(containsPluto);
```

Here our for loop is doing the work of iterating over the array, and a simple if statement checks to see if our condition is met yet. This for loop is a little more complicated than most because it is set up to stop iterating as soon as a matching element is found in the array.

While it is not that difficult to understand, manually keeping track of the iteration and the "short circuiting" logic just isn't the problem that we're trying to solve. The fewer such things we have to keep in our head, the better. We could fix this by moving the loop into it's own function (probably a good idea), but fortunately the latest version of the JavaScript standard includes such a function for us.

The function is called `some` and it is available on all arrays. Here is an example of how it would apply to our problem.

```
function isPluto(element) {
    return (element === "pluto");
}

// Outputs: false
console.log(planets.some(isPluto));

dwarfPlanets = [
    "ceres",
    "pluto",
    "haumea",
    "makemake",
    "eris"
];

// Outputs: true
console.log(dwarfPlanets.some(isPluto));
```

All we have to do to use `some` is to pass it a callback function. The callback will be executed once for each element until the callback returns true. At that point the `some` method itself will return true. If the callback never returns true, then `some` will return false.

The great thing about `some`, though is that you don't have to think about that unless you want to. You just have to pass in your callback.

In addition to the array element itself, the callback function has access to two other parameters: the current index, and the entire array. This can be useful in situations where you are comparing the current element against other members of the array. Consider this example:

```
function isLessThanPrev(el, index, arr) {
    // The first element doesn't have a predecessor,
    // so don't evaluate it.
    if (index === 0) {
        return;
    } else {
        return (el < arr[index - 1]);
    }
}

evens = [2, 4, 6, 8];
randoms = [0, 9, 2, 5];

// Outputs: false
console.log(evens.some(isLessThanPrev));

// Outputs: true
console.log(randoms.some(isLessThanPrev));
```

As with other features of ES5 that we've talked about, `some` is only supported in IE9 and higher. If you'd like to use it in older browsers, you will need to use a library like Underscore which has an equivalent method or a compatibility shim which ports `Array#some` back into older browsers.

Soon we'll be taking a look at what it takes to write your own compatibility shims.

Josh Clanton
