What is an Array?

Originally published in the A Drip of JavaScript newsletter.

For an experienced developer, that question may seem simplistic, but in the case of JavaScript the answer is rather interesting.

Conceptually, an array is just a list of values which can be accessed by using an integer as the "key". The list starts at `0` and goes up from there. If we were to describe that with JavaScript's object notation, it would look like this:

```
fakeArray = {
    0: "value 1",
    1: "value 2"
}
```

I feel like we're missing something. Oh, I know. It's that pesky "length" property.

```
fakeArray = {
    0: "value 1",
    1: "value 2",
    length: 2
}
```

That's looking suspiciously array-like. And, as I've mentioned before, this is precisely how the `arguments` object works. What I haven't mentioned yet is that this is also how real arrays work "under the hood".

That's right, in JavaScript arrays are just a specialized type of object built into the language. It's easy to tell this by looking at the behavior of objects and arrays.

```
fakeArray = {
    0: "value 1",
    1: "value 2",
    length: 2
};

// Outputs "value 1"
console.log(fakeArray[0]);

realArray = ["value 1", "value 2"];
realArray.text = "some text";

// Outputs "some text"
console.log(realArray.text);
```

As you can see, you can access an object's properties in exactly the same way as an array's. And you can give an array additional properties just like any other object.

What about all those special array methods like `indexOf`, `slice`, and `sort`? Turns out, they're just functions attached to the array object. (More specifically, they belong to the Array prototype, but that's getting ahead of ourselves.)

```
realArray = ["value 1", "value 2"];

// Outputs "0"
console.log(realArray.indexOf("value 1"));

realArray.indexOf = function() {
    return "I'll never tell.";
};

// Outputs "I'll never tell."
console.log(realArray.indexOf("value 1"));
```

In fact, given enough time, it is possible to reimplement almost all of the native array functionality purely in terms of manipulating objects.

Here's an example of reimplementing the `push` method:

```
fakeArray = {
    length: 0,
    push: function (val) {
        // Place the new value into the
        // next available integer key
        this[this.length] = val;

        // Update the length property
        this.length = this.length + 1;

        // Return the updated length
        return this.length;
    }
};

fakeArray.push("first value");
fakeArray.push("second value");

// Outputs "first value"
console.log(fakeArray[0]);

// Outputs "second value"
console.log(fakeArray[1]);
```

The one important thing that can't be reimplemented is the convenient array literal syntax for creating arrays. (The square brackets.) But you can always use a constructor instead. In fact, the following two ways of creating an array are exactly equivalent.

```
literalWay = ["value 1"];

constructorWay = new Array("value 1");
```

If you're willing to give up the literal syntax, you can rebuild the concept of arrays in JavaScript entirely from scratch to achieve something like this:

```
myCustomArray = new CustomArray("value 1");
```

And now you know (if you didn't already) how arrays work in JavaScript.

If you have other subjects you'd like to see covered in the future, drop me a suggestion. Thanks for reading!

Josh Clanton
