A Drip of JavaScript

The `delete` Operator in JavaScript

Originally published in the A Drip of JavaScript newsletter.

The `delete` operator is one of the less frequently used aspects of the JavaScript language. But there are times when you need `delete` and nothing else will do. In this drip, we'll dive into how to use it and how it works.

The purpose of `delete`, as you might imagine, is to delete things. More specifically, it will delete object properties. For example:

```
var multiverse = {
    earth1: "Silver Age",
    earth2: "Golden Age"
};

delete multiverse.earth2;

// Outputs: { earth1: "Silver Age" }
console.log(multiverse);
```

The `delete` operator will not delete ordinary variables.

```
var alex = "Alexander Luthor";

delete alex;
```

```
// Outputs: "Alexander Luthor"
console.log(alex);
```

However, it will delete "global variables," since they are actually properties of the global object (`window` in the browser).

```
// Because var isn't used, this is a property of window
classicFlash = "Jay Garrick";

delete window.classicFlash;

// ReferenceError: classicFlash is not defined
console.log(classicFlash);
```

The `delete` operator also has a return value. If it succeeds in deleting a property, it will return true. If it fails to delete a property because the property is unwritable it will return false, or if in strict mode it will throw an error.

```
var multiverse = {
    earth1: "Silver Age",
    earth2: "Golden Age"
};

var earth2Deleted = delete multiverse.earth2;

// Outputs: true
console.log(earth2Deleted);
```

You are probably wondering under what circumstance you'd want to use `delete`. The answer is whenever you actually want to remove a property from an object.

Sometimes rather than delete a property, JavaScript developers will just give it a value of `null`, like so:

```
var multiverse = {
    earth1: "Silver Age",
    earth2: "Golden Age"
};

multiverse.earth2 = null;
```

While this effectively severs the property from the original value, the property itself still exists on the object, as you can see below:

```
// Outputs: {
//    earth1: "Silver Age",
//    earth2: null
// }
console.log(multiverse.earth2)
```

And some operators like `in` and the `for in` loop will still report the presence of the `null` property. If you are passing around an object that might be inspected using those methods, you probably want to make sure that you really delete any unwanted properties.

Finally, you should keep in mind that `delete` doesn't actually destroy the property's value, just the property itself. For example:

```
var earth3 = "The Crime Syndicate";
multiverse.earth3 = earth3;

delete multiverse.earth3;

// Outputs: "The Crime Syndicate";
console.log(earth3);
```

Here, both `earth3` and `multiverse.earth3` referred to the same value. And as you can see, deleting `multiverse.earth3` doesn't affect `earth3` at all.

That's it for our overview of `delete`.
