A Drip of JavaScript

Arbitrary Parameters with the arguments Object

Originally published in the A Drip of JavaScript newsletter.

In last week's issue we discussed default parameters. That gives us some flexibility when working with parameters, but what if we want something that would accept as many arguments as we can throw at it? Say for example that we want to create a function which adds together all the arguments we pass to it. How would we go about that?

```
function addAll () {
    // What do we do here?
}

// Should return 6
addAll(1, 2, 3);

// Should return 10
addAll(1, 2, 3, 4);
```

Fortunately, JavaScript does have an answer, though it is a little quirky. The answer is the `arguments` object.

Though it isn't exactly an array, the `arguments` object **acts** like an array that happens to contain every parameter passed to the function.

```
function myFunc () {
    console.log(arguments[0], arguments[1]);
}

// Outputs "param1" and "param2"
myFunc("param1", "param2");
```

Now that we know about `arguments`, it is easy to make an addition function that will operate on any number of parameters.

```
function addAll () {
    var sum = 0;

    for (var i = 0; i < arguments.length; i++) {
        sum = sum + arguments[i];
    }

    return sum;
}

// Returns 6
addAll(1, 2, 3);

// Returns 10
addAll(1, 2, 3, 4);
```

One problem to watch out for with `arguments` is that it isn't **really** an array. We can see this by running the following:

```
function myFunc () {
    console.log(Array.isArray(arguments));
}

// Will output 'false'
myFunc('param');
```

So it's not an array. Does that make a difference? Unfortunately, yes. It doesn't have any of the normal array methods like `push`, `pop`, `slice`, `indexOf`, or `sort`.

```
function sortArgs () {
    // This won't work!
    sorted = arguments.sort()

    return sorted;
}
```

This can easily bite you, especially if you pass the contents of `arguments` to another function which expects a real array.

The solution is surprisingly easy to use, but a little more difficult to understand.

```
function sortArgs () {
    // Convert arguments object into a real array
    var args = [].slice.call(arguments);

    // Now this will work!
    sorted = args.sort()

    return sorted;
}
```

What's going on here has several steps:

1. We create an empty array.

2. We use the array's `slice` method.

3. We use the `call` method to tell `slice` that it should operate on `arguments` rather than on the empty array.

Invoking `slice` without specifying which index the slice should begin at will return an unsliced **array**. And that's how we end up with exactly what we want: a real array that contains every parameter that was passed to the function.