sitepoint premium

Handout

JavaScript: Next Steps
Lesson 5 - Step 4
M. David Green

What Is Reduce

As long as we introduced `map`, we may as well talk about `reduce`. A lot of people talk about them together. They are two different ways of interacting with data in an array and coming up with a result.

What `reduce` does is come up with a combined result by performing an operation on each item in the array and adding that value to an accumulator. That continues to pass each time that it iterates through the loop. That accumulator is the value that gets returned by `reduce`.

```
var numbers = [1, 2, 3, 5, 8, 13];

var addThem = function(first, second) {
  return first + second;
};

var total = numbers.reduce(addThem);
console.log(total); //32
```

`numbers` contains a Fibonacci sequence there. What happens with `reduce` is it took our function, `addThem`, and each time that a number came up it added it and used it to add the next value to whatever was in the accumulator from the previous iteration. So first time it added `1 + 2` and what was in the accumulator was 3. Then it added 3 and what was in the accumulator was 6 and so on.

Another convenient thing about `reduce` is that you can set an initial value for the accumulator. All you have to do is pass that initial value as a second argument when you call `reduce`.

```
var numbers = [1, 2, 3, 5, 8, 13];

var addThem = function(first, second) {
  return first + second;
};

var total = numbers.reduce(addThem, 20);
console.log(total); //52
```

This way we simply start with 20.

Tracking the Index

`reduce` also gives you access to the index of the item in the original array. You can even get the original source array. Access these by adding additional arguments to the end of the function that you pass in to your `reduce` method.

```
var numbers = [1, 2, 3, 5, 8, 13];

var addThem = function(first, second, count, source) {
  console.log(count);
  return first + second;
};

var total = numbers.reduce(addThem);
console.log(total); //32
// 0
// 1
// 2
// 3
// 4
// 5
// 32
```

Now our function takes `count` and `source`.

So take this new knowledge and consider using `forEach`, `map` and `reduce` the next time you have an array that you want to iterate over. You could reduce your dependence on extraneous variables and also make your code cleaner, more maintainable, and easier to understand.