JavaScript—Learn to Chain Map, Filter, and Reduce

codeburst.io/javascript-learn-to-chain-map-filter-and-reduce-acd2d0562cd4

This is article #4 in a four part series this week.

- Learn Map— `Array.prototype.map()`
- Learn Filter— `Array.prototype.filter()`
- Learn Reduce— `Array.prototype.reduce()`
- **Chaining Map, Filter, and Reduce**

If you're unfamiliar with `map()`, `filter()`, and/or `reduce()`, be sure to use the links above to check out my articles on them before continuing.

Chaining Map, Filter, & Reduce

Consider the following data:

```
data = [
  {
    name: 'Butters',
    age: 3,
    type: 'dog'
  },
  {
    name: 'Lizzy',
    age: 6,
    type: 'dog'
  },
  {
    name: 'Red',
    age: 1,
    type: 'cat'
  },
  {
    name: 'Joey',
    age: 3,
    type: 'dog'
  },
];
```

As you can see we have an array of objects. Each object represents a pet. The pets have a `name`, an `age`, and a `type`.

The goal of this article is going to be to write some JavaScript that will sum all of the dogs ages in dog years.

Our process might look something like this:

1. Select only the dogs
2. Translate their ages into dog years (multiply them by seven)
3. Sum the results

Here's how we could accomplish this with a `for` loop:

```
function getAges(data) {
  let sum = 0;

  for (var i = 0; i < data.length; i++){
    if (data[i].type === 'dog'){
      let tempAge = data[i].age;
      sum += (tempAge * 7);
    }
  }

  return sum;
}

// getAges(data) = 84
```

We create a variable named `sum` and set it equal to `0`. Then we loop through our array, one object at a time. If our pet is a `dog`, we take the age of that dog, multiply it by seven, and add the resulting value to our `sum`. We repeat this operation for each `dog` in the array. When our loop finishes, we return the `sum`.

And it works! We get `84`. And while the above code isn't wrong, it's also not a perfect solution.

Namely, it's doing a lot of things at once. It's accomplishing all three of our challenges in one function which makes it somewhat difficult to read. The code also isn't very reusable.

To solve this problem, we're going to utilize `map()`, `reduce()`, and `filter()` to accomplish the same goal.

The first thing we need to do with our data is filter out the cats. We can use `filter()` to accomplish this.

Our filter method takes the pet as input and will return `true` on a an animal if its `type` is equal to `dog`:

```
let ages = data.filter((animal) => {
  return animal.type === 'dog';
})
```

Now that we only have dogs, we need to find the ages of the dogs and multiply them by 7. We can do this with the `map()` method. Our map function will simply return the animals age multiplied by 7:

```
.map((animal) => {
  return animal.age *= 7
})
```

Finally, we need to sum the ages of all of our dogs. We can do this with the `reduce()` method. Our reduce function will return the sum of our animals age and the current sum:

```
.reduce((sum, animal) => {
  return sum + animal.age;
});
```

Now that we have all three of our steps completed, we simply chain our actions together. Here's what the code looks like:

```
let ages = data
  .filter((animal) => {
    return animal.type === 'dog';
}).map((animal) => {
    return animal.age * 7
}).reduce((sum, animal) => {
    return sum + animal.age;
});

// ages = 84
```

Awesome! We run our code and again get `84`.

...But our code is still a little convoluted.

To fix this, we'll create three pure functions and use them with our chain.

If you're unfamiliar, a pure function in JavaScript is one that **given the same input, will always return the same output without side effects.** Put simply, pure functions only depend on their input arguments.

First, we'll create a function that checks if an element is a dog. It takes our element as input and returns either `true` or `false`.

```
let isDog = (animal) => {
  return animal.type === 'dog';
}
```

Next, we'll create a function that multiplies the age of an element by seven and returns only the age in dog years:

```
let dogYears = (animal) => {
  return animal.age * 7;
}
```

Finally, we need a function that sums two numbers and returns the result:

```
let sum = (sum, animal) => {
  return sum + animal;
}
```

Now that we have our three functions, we can use them with our `map()`, `filter()`, and `reduce()` chain:

```
let ages = data
  .filter(isDog)
  .map(dogYears)
  .reduce(sum);

// ages = 84
```

It works! Again, we get `84`.

Our final version of the code is very easy to read, understand, and test. Plus our code has been broken out into pure functions that we are able to reuse easily throughout our program as necessary.

Closing Notes:

Thanks for reading! This has been a brief introduction into chaining JavaScript's `map()`, `filter()`, and `reduce()` functions. If you're ready to finally learn Web Development, check out the **The Ultimate Guide to Learning Full Stack Web Development in 6 months**.