Using the Array Map Method

- Due No Due Date
- Points 1
- Submitting a website url

 (https://github.com/learn-co-curriculum/phase-1-array-map-method-lab)  (https://github.com/learn-co-curriculum/phase-1-array-map-method-lab/issues/new)

Learning Goals

- Review how the `map()` method works
- Demonstrate `map()` with `Array` s
- Demonstrate `map()` with complex data structures

Introduction

As developers, we find ourselves responsible for all sorts of common, but tedious, tasks, such as iterating over arrays. Although a `for` loop will work for these tasks, we can take advantage of a method like `map()` to save ourselves work and to organize and optimize our code, resulting in more readable and understandable functions.

Review How the `map()` Method Works

`Array.prototype.map()` is a method that iterates over an array and applies a function to each element, modifying it in some way. The result is then returned as a *new* array, leaving the original array the same. This is super helpful, because it saves us from having to build out the loop, or create a new array and copy stuff in there. It also leaves the elements in the original array unchanged, which helps protect our code from bugs.

Demonstrate `map()` With `Array` s

As mentioned above, we use `map()` when we want to perform an action on each element in an `Array` , and "gather" the results into a new `Array` . We'll start by looking at how we would build the functionality ourselves, using a `for...of` loop, then show how `map()` can save us work and

improve our code.

We'll also use this as a chance to demonstrate some of the power of functions in JavaScript. We'll write the code **four times**, making it increasingly efficient and *expressive* each time.

Using `for...of` in Place of `.map()`

In this example, we are using a standard bit of iteration code. The code below recreates the functionality of the native `.map()` method using `for...of`. But because `for...of` (and `for` as well) is a *general* function that can be used to do lots of things, another programmer would have to examine the loop's inner workings to determine exactly what the code is doing.

```
const skiSchool = ["aki", "guadalupe", "lei", "aalam"];
const rollCall = [];

for (const student of skiSchool) {
  rollCall.push(student + " the skier");
}

//=> rollCall = ["aki the skier", "guadalupe the skier", "lei the skier", "aalam the skier"];
```

If we use the `.map()` method, on the other hand, we are saying to other programmers: "Expect a new array to come out of this after each element is modified in some way!"

Let's look at a few different ways to implement the native `.map()` method.

`map()` With a Function Declaration

```
function studentRollCall(student) {
  return student + " the skier";
}

const skiSchool = ["aki", "guadalupe", "lei", "aalam"];
```

```
const rollCall = skiSchool.map(studentRollCall);
//=> rollCall = ["aki the skier", "guadalupe the skier", "lei the skier", "aalam the skier"];
```

We use `map()` when we want to transform the elements in an array in some way. To do this, we pass a function *as an argument*; that function (the callback) is what executes our desired transformation. In JavaScript, arguments can be primitive types like `Number` or `String`, but they **can also be work**. Very few other programming languages allow that!

The iterator function `map()` calls the callback for each element in turn, passing the element as an argument, and stores the return value in a new `Array`. When the iterations are complete, it returns that new array.

This code is more *expressive* than the version using `for...of` because as soon as a developer sees that `map()` is being used, they know a lot about what the code is doing.

Note that this code is using a *named* function as the callback. This is perfectly valid, but the `studentRollCall` function isn't doing much work. We may want to streamline our code a bit more by using a function expression ("anonymous function") instead.

`map()` With a Function Expression

```
const skiSchool = ["aki", "guadalupe", "lei", "aalam"];
const rollCall = skiSchool.map(function (student) {
  return student + " the skier";
});
//=> rollCall = ["aki the skier", "guadalupe the skier", "lei the skier", "aalam the skier"];
```

By defining a function expression inline, we're able to tighten up our code without changing its functionality or making it less expressive.

`map()` With an Arrow Function

Thanks to arrow functions, we can shorten up the function even more:

```
// When the parameter list is only one element, we can drop () !
const skiSchool = ["aki", "guadalupe", "lei", "aalam"];
```

```
const rollCall = skiSchool.map((student) => student + " the skier");
//=> rollCall = ["aki the skier", "guadalupe the skier", "lei the skier", "aalam the skier"];
```

The code now fits on one line! We've pared down all that noisy JavaScript code in the `for...of` version by using `map()` along with more efficient JavaScript syntax. This makes our code even more expressive: that single line of code tells us everything we need to know about what the code is doing.

Demonstrate `map()` With Complex Data Structures

Let's use the `map()` function on a trickier data structure — a list of objects. To start things off, we have an array of robots. We want to activate all of them. To activate a robot, we need to mark it as such using the `isActivated` boolean, and also double its number of modes:

```
const robots = [
  { name: "Johnny 5", modes: 5, isActivated: false },
  { name: "C3PO", modes: 3, isActivated: false },
  { name: "Sonny", modes: 2.5, isActivated: false },
  { name: "Baymax", modes: 1.5, isActivated: false },
];

const activatedRobots = robots.map((robot) => {
  return Object.assign({}, robot, {
    modes: robot.modes * 2,
    isActivated: true,
  });
});

console.log(activatedRobots);

/*
 Result:
 [
   { name: 'Johnny 5', modes: 10, isActivated: true },
```

```
    { name: 'C3PO', modes: 6, isActivated: true },
    { name: 'Sonny', modes: 5, isActivated: true },
    { name: 'Baymax', modes: 3, isActivated: true }
  ]
 */
```

We could, of course, accomplish the same thing using a `for` or `for...of` loop, but using the native `map()` function frees us from having to create an empty array, code the looping mechanism, push the modified values into the empty array, and return the modified array at the end. Instead of having to rewrite the iteration code every time we need to modify elements in an array, `map()` allows us to focus all our effort on building the actions we need in our callback function.

Lab: Using `map()` to Generate a New Array

Let's put our newly acquired knowledge of `map()` to use! We just uploaded 10 coding tutorials online, but some of them have inconsistent casing. We want all the titles to be "title case", in other words, the first letter of each word should be capitalized. Create a new array containing the names of the tutorials with proper title case formatting. For example, `'what does the this keyword mean?'` should become `'What Does The This Keyword Mean?'` .

```
const tutorials = [
  "what does the this keyword mean?",
  "What is the Constructor OO pattern?",
  "implementing Blockchain Web API",
  "The Test Driven Development Workflow",
  "What is NaN and how Can we Check for it",
  "What is the difference between stopPropagation and preventDefault?",
  "Immutable State and Pure Functions",
  "what is the difference between == and ===?",
  "what is the difference between event capturing and bubbling?",
  "what is JSONP?",
];
```

Your job is to write the following function:

- `titleCased()` : returns an array with title case tutorial names. Note that this function takes no arguments and should use the global `tutorials` variable as data.

NOTE: This lab is challenging! You will need to iterate through the `tutorials` array, modifying the name of each tutorial. To do this, you will **also** need to access and modify each individual word.

Some questions to consider:

- How can we "iterate" through individual words in a string?
- Can we execute an iteration inside an iteration? How?
- How can we capitalize just the first letter in a word?

A couple of hints:

- Break the task into smaller chunks: using the console or a REPL, start by figuring out how to modify one individual element in the `tutorials` array. Once you've got that working, then figure out how to update the array itself.
- Use Google!!

Remember the workflow:

1. Install the dependencies using `npm install` .
2. Run the tests using `npm test` .
3. Read the errors; vocalize what they're asking you to do.
4. Write code; repeat steps 2 and 3 often until a test passes.
5. Repeat as needed for the remaining tests.

After you have all the tests passing, remember to commit and push your changes up to GitHub, then submit your work to Canvas using CodeGrade.

Conclusion

`map()` takes 2 arguments — a callback and the optional context. The callback is called for each value in the original array and the modified value is added to a new array. Its return value is a new array that is the same length as the original array. Using `map()` saves time while making the code simpler and more expressive.

Resources

- MDN: Array.prototype.map() (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)