

Join GitHub today


GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

Branch: master curriculum / 05-advanced-javascript / 01.advanced-arrays.md

Find file Copy path

 riboher Fix typos and add reduce exercises

ddb1c32 5 days ago

1 contributor

366 lines (273 sloc) 11.3 KB

Raw Blame History   

Arrays in depth

So far we've covered how to **loop** over iterable elements in javascript, such as arrays (**Disclaimer:** objects are [also iterable](#) but let's leave that for another time).

We should know by now how to use a `for` loop and calling the `.forEach` function with an array without much struggle.

These forms of iterating over arrays and manipulate them are mostly ok in the majority of cases, and you could easily do pretty much everything you need with those two types of loops, but there's another way which with you can save a lot of code while keeping your codebase clean and [avoid mutating your arrays](#) (more on this later).

Javascript let us do three basic operations with three different methods that we're going to teach you right now. These three functions have all one thing in common, and that is that they are [Higher order functions](#). Don't worry if you don't understand that concept now, just remember that all three work exactly the same: They are **functions** that accept another function as their parameter (or [callback function](#)). Let's see them in action!

Map

According to [MDN](#) documentation, the map function

"creates a new array with the results of calling a provided function on every element in the calling array".

This is pretty much what we were doing with our arrays, right? The main difference is that `map` already returns a new array (which you must **store** somewhere, of course) with the result of **executing our callback function** once for each one of the elements of our array. So let's see it in action:

```
const arr = [1, 2, 3, 4, 5, 6, 7]
const multiplyBy2 = (n) => n * 2

const arrTimes2 = arr.map(num => multiplyBy2(num))

console.log(arrTimes2) // [2, 4, 6, 8, 10, 12, 14]
```

Ok, so let's dive first into that code from above:

1. First, we've **declared** an array with some numbers, from 1 to 7.
- 2.

Then, we've **declared a new function** that **receives** a number and multiplies it by 2, **[returning]** (<https://jaketrent.com/post/javascript-arrow-function-return-rules/> the result.

3. We call the `map` function from our array, executing our previously declared function with the element provided by `map`, as many times as elements has the array.

Note that the following code would do exactly the same:

```
const arr = [1, 2, 3, 4, 5, 6, 7]

const arrTimes2 = arr.map(num => {
  return num * 2
})

console.log(arrTimes2) // [2, 4, 6, 8, 10, 12, 14]
```

Easy, right? Certainly it saves us a lot of code considering how should we have done it before:

```
const arr = [1, 2, 3, 4, 5, 6, 7]
const arrTimes2 = []
const multiplyBy2 = (n) => n * 2

for (let i = 0; i < arr.length; i++) {
  const result = multiplyBy2(arr[i])
  arrTimes2.push(result)
}

console.log(arrTimes2) // [2, 4, 6, 8, 10, 12, 14]
```

Now, let's move to the next function that will let us easily work with our arrays.

Filter

As its own name tells us, filter is used to select certain elements of an array and choose them to be part of a new array, based on a certain condition. Filter function allows us to discard elements that we don't need in our newly created array. According to [MDN](#):

The `filter()` method creates a new array with all elements that pass the test implemented by the provided function.

As `map` does, `filter` also returns a different array than the one we are working with, with the exception that it will contain only the elements that the **returning expression** of our callback function returns `true` for.

Let's see it better with an example:

```
// Let's return those words longer than 6 characters
const words = ['Hello', 'Javascript', 'Web development', 'CSS', 'HTML']

const longWords = words.filter(word => {
  return word.length > 6
})

console.log(longWords) // ['Javascript', 'Web development']
```

As you can see, filter is pretty straightforward. You just need to return an expression that evaluates to `true` or `false`, and all the elements of the array that pass that test, will be in the new array. All those that don't pass it, will be automatically discarded.

Again, this saves us from doing a lot of unnecessary extra steps as we needed to do before:

```
// Let's return those words longer than 6 characters
const words = ['Hello', 'Javascript', 'Web development', 'CSS', 'HTML']
const longWords = []

for (let i = 0; i < words.length; i++) {
  if (words[i].length > 6) {
    longWords.push(words[i])
  }
}

console.log(longWords) // ['Javascript', 'Web development']
```

Now let's see the last methods that will make our lives easier when working with arrays.

Reduce

The `Reduce` function works pretty much the same as the other two. It accepts a function as a parameter exactly the same way as `map` and `filter` do, with the exception that this callback function (called reducer), instead of receiving the current element of the array as first parameter, it takes as first argument the accumulated result (accumulator) of the previously executed iterations. Let's see how [MDN](#) defines this method and then we'll see it with some code:

The `reduce()` method executes a reducer function (that you provide) on each element of the array, resulting in a single output value.

Another thing to notice is that instead of returning a new array, `reduce` always outputs a single value, the result of accumulating the returning value of each iteration. Let's see how it works:

```
// Let's get the sum of all the values in the array
const nums = [124, 22, 874, 173, 5739]

const sumOfNums = nums.reduce((accum, num) => {
  return accum + num
}, 0)

console.log(sumOfNums) // 6932
```

As we said before, you can notice how on each iteration, we receive the previous result of adding the accumulator to the element of the array, resulting in the end with the output of the sum of all the elements in the array.

There's a second thing to take into account with the `reduce` function, and that is that apart from the accumulated value and the reducer function, we can pass a third parameter, consisting on the initial value that our reducer function will accept as accumulator on the first iteration. Let's see it better with an example:

```
// Let's get the multiplication of all the values in the array
const nums = [124, 22, 874, 173, 5739]
const multiplicationOfNums = nums.reduce((accum, num) => {
  // On the first iteration, when num's value is 124
  // accum's value will be 1
  return accum * num
}, 1 /* what if this were a zero? */)
console.log(multiplicationOfNums) // 2367217302384
```

This might seem confusing at first, but everything will be crystal clear once you start practicing with a few exercises!



Exercises

1. Map exercises

- Given the following array, create a second array with the result of getting the power of each number by itself.

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

// Write your code here!

// Expected result
// [1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489, 1000000000]
```

- Given the following array, create a second array that uses each one of the words to form the sentences seen below.

```
const foodList = ['Pizza', 'Ramen', 'Paella', 'Chuletón']

// Write your code here!

//Expected result:
/* [
  'Since I'm Italian, I love eating Pizza',
  'Since I'm Japanese, I love eating Ramen',
  'Since I'm from Valencia, I love eating Paella',
  'Although I don't eat meat, Chuletón is certainly tasty'
]
*/
```

- Given the following array, create a second array that forms sentences using the objects' properties:

```
const mfrStaff = [
  {
    name: 'Andrea',
    role: 'coordinator',
    hobbies: ['sing', 'watch movies']
  },
  {
    name: 'Xavi',
    role: 'teacher',
    hobbies: ['read', 'code']
  },
  {
    name: 'Ricardo',
    role: 'teacher',
    hobbies: ['read', 'garden']
  },
  {
    name: 'Rafa',
    role: 'teacher',
    hobbies: ['travel', 'watch series']
  },
  {
    name: 'Jordi',
    role: 'teacher',
    hobbies: ['travel', 'build robots']
  }
]

// Write your code here!

// Expected result
/*
[
  'Andrea is a coordinator and they like to sing and watch movies',

```

```
    'Xavi is a teacher and they like to read and code',
    'Ricardo is a teacher and they like to read and garden',
    'Rafa is a teacher and they like to travel and watch series',
    'Jordi is a teacher and they like to travel and build robots'
  ]
}
*/
```

2. Filter exercises

- Given the following array, create a second array only with the even numbers

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
// Write your code here!
```

```
// Expected result
```

```
// [2, 4, 6, 8, 10]
```

- Given the following array, create a second array that filters out all the food that has meat and outputs the sentence seen below

```
// Tip: You can chain map and filter to first remove the elements you don't want in your new array and then
```

```
const foodList = [
  {
    name: 'Tempeh',
    isVeggie: true
  },
  {
    name: 'Cheesbacon burger',
    isVeggie: false
  },
  {
    name: 'Beyond meat burger',
    isVeggie: true
  },
  {
    name: 'Chuletón',
    isVeggie: false
  }
]

// Write your code here!

//Expected result:
/* [
  'Nothing like a good tempeh for dinner!',
  'Nothing like a good Beyond meat burger for dinner!'
]
*/
```

- Given the following array, create a second array that **outputs only the name** of the products that cost less than 200€

```
const inventory = [
  {
    name: 'Mobile phone',
    price: 199
  },
  {
    name: 'TV 4K',
    price: 400
  }
]
```

```
    },
    {
      name: 'PS5',
      price: 600
    },
    {
      name: 'Programming course license',
      price: 155
    }
  ]

// Write your code here!

// Expected result
/*
[
  'Mobile phone',
  'Programming course license'
]
*/
```

3. Reduce exercises

- Given the following array, get the multiplication of all the elements of it.

```
const numbersList = [45, 5, 1, 23, 32]
```

```
// Write your code here!
```

```
// Expected output --> 165600
```

- Given the following array, concatenate all the elements in the array using `.reduce()`.

```
const sentenceElements = [
  'My',
  'name',
  'is',
  /* Place your name here */,
  'and',
  'I',
  'love',
  'javascript'
]
```

```
// Write your code here!
```

```
// Expected result --> 'My name is John Doe and I love javascript'
```

- Given the following array, get the total amount of those products that belong to the category of `coding`

```
const booksList = [
  {
    name: 'You don\'t know JS',
    author: 'Kyle Simpson',
    price: 15,
    category: 'code'
  },
  {
    name: 'The handmaids tale',
    author: 'Margaret Atwood',
    price: 18,
    category: 'novel'
  }
]
```

```
    },  
    {  
      name: 'Game of Thrones',  
      author: 'George R. Martin',  
      price: 32,  
      category: 'Fantasy'  
    },  
    {  
      name: 'Eloquent Javascript',  
      author: 'Marijn Haverbeke',  
      price: 40,  
      category: 'code'  
    }  
  ]  
  
  // Write your code here!  
  
  // Expected result --> 55
```