Back to Week 1

JS | Arrays - Map, Reduce & Filter

LESSON

Learning Goals

After this lesson you will be able to:

- Understand advanced array methods such as `filter`, `reduce`, and `map`
- Distinguish between `forEach` and `map` and when to use each of them
- Know how `reduce` and `filter` works
- Implement `filter`, `map` and `reduce` in real examples

Introduction

We've seen some basic array methods up until this point. One of them is `forEach` - although we use it very often, sometimes we want to do more than iterate through the array.

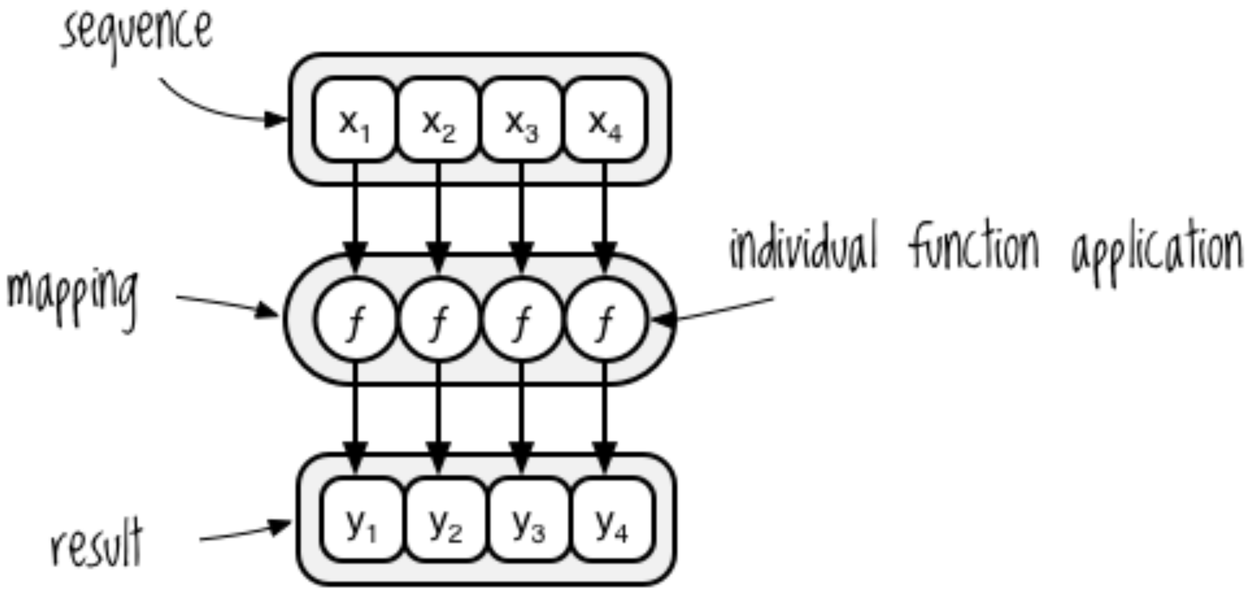
> One thing to keep in mind about most of the functions we'll cover in this lesson is that **we can easily use `forEach`** to get **the same results** but we will end up with **longer/less readable and less efficient code**.

That's why let's introduce these new concepts.

> To start, keep in mind that `.map()`, `.filter()` and `.reduce()` methods **DON'T modify the original array, that is - they don't mutate the original array but rather create a new array**.

Now let's see what that actually means.

.map()



.map() is very similar to `forEach`, except for one **important** distinction:

- the `.forEach()` method doesn't actually return anything (undefined). It simply calls a provided function on each element in your array. Eventually if you want to make some changes in the array you're iterating over, you'll end up changing the array itself (the original array will be mutated).
- the `.map()` method will also call a provided function on every element in the array. The difference is that `.map()` utilizes return values and actually returns a new array of the same size.

In short, we use .map() to *transform* each element into something else.

`forEach` **DOES NOT WORK**

```
const array = [1, 2, 3];

const newArray = array.forEach(function(number){
  return number * 2;
})

console.log(newArray); // <== undefined
```

Explain this code

`map`

- ES5 (good old `function`):

```
const array = [1, 2, 3];

var newArray = array.map(function(number){
  return number * 2;
})

console.log(newArray); // [ 2, 4, 6 ]
```

Explain this code

The `newArray`:

- has the same number of elements as the original array: it's actually a copy
- BUT where each new element is mapped by the callback function to its copied `value`*2

- ES6 `one-line arrow function`:

```
const array = [1, 2, 3];

const newArray = array.map(number => number * 2);

console.log(newArray); // [ 2, 4, 6 ]
```

Explain this code

`forEach` (equivalence)

```
const array = [1, 2, 3];

const newArray = []; // we have to create a newArray
array.forEach(function(numcopy){
  numcopy *= 2;           // double the passed copy (it's a copy)
  newArray.push(numcopy); // and push it to the newArray
});

console.log(newArray); // <== [ 2, 4, 6 ]
```

Explain this code

NB : the real difference is we have to create the `newArray` ourselves (because `forEach` does not return anything other than `undefined`)

Ok, `.map` create a new array for me.
But on the contrary with `.map`, you have to `return` a value out, or else your new array will be filled with a bunch of `undefined`s:

```
const newArray = [1,2,3].map(function (el) {
  el * 2; // Oops, I missed `return` (#britney #diditagain)
});

console.log(newArray); // [undefined, undefined, undefined]
```

Explain this code

More Examples

```
const foods = ["pizza", "sandwiches", "ice cream"];

// ES5:
var capsFoods = foods.map(function(food){
  return food.toUpperCase();
});

console.log(capsFoods);
// [ 'PIZZA', 'SANDWICHES', 'ICE CREAM' ]

// ES6:
const capsFoods = foods.map(food => food.toUpperCase());

console.log(capsFoods);
// [ 'PIZZA', 'SANDWICHES', 'ICE CREAM' ]

```

Explain this code

Exercise

Given an array of cities, return an array with the first letter of each city's name capitalized. You can use ES5 or ES6 syntax, whichever you feel more comfortable with at this point.

Starter Code

```
// array of cities:
var cities = ["miami", "barcelona", "madrid", "amsterdam", "berlin", "sao paulo", "lisbon", "mexi
co city", "paris"];
```

Explain this code

More practice

Imagine you are a Math teacher and you have to grade our students based on their performance on two projects (40% of final grade) and their final exam (60% of final grade). We got the info for each student in an object that looks like this:

```
{
  name: "Student Name",
  firstProject: 80,
  secondProject: 75,
  finalExam: 90
}
```

Explain this code

So the whole class is represented as an array of objects (each object contains the data about that student), and we need to get a *new* array of objects, but this time the objects will contain only student´s name, and their final grade. Let´s do it.

Here is the data:

```
const students = [
    {
      name: "Tony Parker",
      firstProject: 80,
      secondProject: 75,
      finalExam: 90
    },
    {
      name: "Marc Barchini",
      firstProject: 84,
      secondProject: 65,
      finalExam: 65
    },
    {
      name: "Claudia Lopez",
      firstProject: 45,
      secondProject: 95,
      finalExam: 99
    },
    {
      name: "Alvaro Briattore",
      firstProject: 82,
      secondProject: 92,
      finalExam: 70
    },
    {
      name: "Isabel Ortega",
```

```
      firstProject: 90,
      secondProject: 32,
      finalExam: 85
    },
    {
      name: "Francisco Martinez",
      firstProject: 90,
      secondProject: 55,
      finalExam: 78
    },
    {
      name: "Jorge Carrillo",
      firstProject: 83,
      secondProject: 77,
      finalExam: 90
    },
    {
      name: "Miguel López",
      firstProject: 80,
      secondProject: 75,
      finalExam: 75
    },
    {
      name: "Carolina Perez",
      firstProject: 85,
      secondProject: 72,
      finalExam: 67
    },
    {
      name: "Ruben Pardo",
      firstProject: 89,
      secondProject: 72,
      finalExam: 65
    }
]
```

Explain this code

Go ahead to Repl.it and practice `.map()` method. Try to solve it by yourself first, and then look at our solution.

```
const finalGrades = students.map(theStudent => {
  const projectsAvg = (theStudent.firstProject + theStudent.secondProject)/2;
  const finalGrade  = theStudent.finalExam * 0.6 + projectsAvg * 0.4;
  return {
    name: theStudent.name,
    finalGrade: Math.round(finalGrade)
  }
})


console.log(finalGrades);
// [
//   { name: 'Tony Parker', finalGrade: 85 },
//   { name: 'Marc Barchini', finalGrade: 69 },
//   { name: 'Claudia Lopez', finalGrade: 87 },
//   { name: 'Alvaro Briattore', finalGrade: 77 },
//   { name: 'Isabel Ortega', finalGrade: 75 },
//   { name: 'Francisco Martinez', finalGrade: 76 },
//   { name: 'Jorge Carrillo', finalGrade: 86 },
//   { name: 'Miguel López', finalGrade: 76 },
//   { name: 'Carolina Perez', finalGrade: 72 },
//   { name: 'Ruben Pardo', finalGrade: 71 }
```

```
23  //  ]
```

Explain this code

.reduce()

> `.reduce()` is a method that **turns a list of values into one value**.

A reduction in cooking is when you take many ingredients and cook them down until they become one delicious dish.

For instance, when you cook tomato sauce, you use carrots, celery, onion, garlic, and tomatoes. Eventually, after hours of cooking, they become a delicious tomato sauce.

Let's take a look at `.reduce()` in JavaScript.

```
1  array.reduce(function(accumulator, currentValue) {
2    return accumulator + currentValue;
3  });
```

Explain this code

> 💡 Keep in mind that `accumulator` and `currentValue` are placeholders, can be anything

How does this work?

- **accumulator** is an accumulated value of each call. In the first round, it's assumed it's the first value from the array unless we state differently (which we will see how).
- **currentValue** is the current element in each iteration that will be added to the accumulator.

In ES6, the syntax goes to:

```
1  array.reduce((accumulator, currentValue) => accumulator + currentValue)
```

Explain this code

NB : Reduce can accept two additional parameters, so if you want to learn more about that you can check it here.

Examples

Sum

Let's see how all this looks through some examples. In the first one, we are accumulating the sum of numbers from an array.

```js
const numbers = [2, 4, 6, 8];

var total = numbers.reduce(function (accumulator, currentValue) {
  console.log("accumulator is: ", accumulator, "and current value is: ", currentValue);
  return accumulator + currentValue;
});

console.log("total is: ", total);

// accumulator is:  2 and current value is:  4
// accumulator is:  6 and current value is:  6
// accumulator is:  12 and current value is:  8
// total is:  20
```

Explain this code

The execution of the `.reduce()` example above, step by step would be:

call	accumulator	currentValue	return
first	**2**	**4**	6
second	6	6	12
third	12	8	**20**

sum with `initialValue`

Let's see how we can add the sum we get as a reduced value from some array to already existing value. Assume you're given the following array and you have to reduce it and add that number to `23` (numberYouGet + 23).

```js
const numbers = [12, 9, 1, 8];

const total = numbers.reduce((accumulator, currentValue) => {
  console.log("accumulator is: ", accumulator, "and current value is: ", currentValue);
  return accumulator + currentValue;
}, 23); // <= notice the 23!!!

console.log("total is: ", total);

// accumulator is:  23 and current value is:  12
// accumulator is:  35 and current value is:  9
// accumulator is:  44 and current value is:  1
// accumulator is:  45 and current value is:  8
// total is:  53
```

Explain this code

So we see that the initial value of the accumulator is no longer the first element from the array (which is 12), but instead that is `23`. Here we see that we can pass the second argument to reduce, and that value will become the initial value.

call	accumulator	currentValue	return
first	**23**	**12**	35
second	35	9	44
third	44	1	45
fourth	45	8	**53**

product

One more example: Calculate **product** of all elements in the following array.

```
const numbers = [2, 4, 6, 8];

const total = numbers.reduce((accumulator, current) => accumulator * current);

console.log(total); // <== 384
```

Explain this code

concat

`.reduce()` works for iterating over arrays, regardless of the data type that's inside the arrays. We saw examples with numbers, and here is one with the strings:

```
const words = ["This", "is", "one", "big", "string"];

// ES5:
var bigString = words.reduce(function(sum, word){
  return sum + word;
});

// ES6:
const bigString = words.reduce((sum, word) => sum + word);

console.log(bigString); // <== Thisisonebigstring
```

Explain this code

Objects

Sometimes we need to reduce while we're using objects. This process can get a bit tricky, but our way out is going to be setting an **initialValue**.

Correct

```
const people = [
  { name: "Candice", age: 25 },
  { name: "Tammy",   age: 30 },
  { name: "Allen",   age: 49 },
  { name: "Nettie",  age: 21 },
  { name: "Stuart",  age: 17 }
];

const ages = people.reduce(function(sum, person){
  return sum + person.age;
}, 0); // initialValue to 0

console.log(ages); // <== 142
```

Explain this code

Instead of starting with the first two objects, we start with the number 0. Our iteration table looks like the following:

call	accumulator	currentValue	return
first	**0**	**25**	25
second	25	30	55
third	55	49	104
fourth	104	21	125
fifth	125	17	**142**

Using `forEach` to total

```
var numbers = [1, 2, 3, 4];
var total = 0;

numbers.forEach(function(number){
  total += number;
})

console.log(total);
// 10
```

Explain this code

Exercises

Avg calories

Given a menu of foods and their calories, calculate the **average** number of calories for the entire list. Starter code:

```
const menu = [
  { name: "Carrots", calories: 150 },
  { name: "Steak", calories: 350 },
  { name: "Broccoli", calories: 120 },
  { name: "Chicken", calories: 250 },
  { name: "Pizza", calories: 520 }
];

// your code:


console.log(averageCalories); // 278
```

Explain this code

amazon

amazonbasics

AmazonBasics Apple Certified Lightning to USB Cable - 6 Feet (1.8 Meters), White

45,477 customer reviews | 721 answered questions

#1 Best Seller in Cell Phones & Accessories

Price: $7.99 prime

FREE Shipping on orders over $25—or get **FREE Two-Day Shipping** with Amazon Prime

In Stock.

Want it tomorrow, Aug. 17? Order within **6 hrs 47 mins** and choose **One-Day Shipping** at checkout. Details

Ships from and sold by Amazon.com in easy-to-open packaging.

Style: **1-Pack**

1-Pack | 2-Pack

Size: **6 Feet**

4 Inches | 3 Feet | **6 Feet** | 10 Feet

Color: **White**

Configuration: **Stand-alone cable**

Stand-alone cable | 6 Port USB Charger Bundle

- Apple MFi certified charging and syncing cable for your Apple devices
- Apple MFi certification ensures complete charge and sync compatibility with iPhone 7 Plus / 7 / 6s Plus / 6s / 6 Plus / 6 / 5s / 5c / 5 / iPad Pro / iPad Air / Air 2 / iPad mini / mini 2 / mini 4 / iPad 4th gen / iPod Touch 5th gen / iPod nano 7th gen and Beats Pill+

We are developing our super e-commerce website; each product has some user comments and rating stored in an array called “Reviews”. Each review is an object, so we have the following structure:

```
const product = {
  name: "AmazonBasics Apple Certified Lightning to USB Cable",
  price: 7.99,
  manufacturer: "Amazon",
  reviews:
  [
    {  user: "Pavel Nedved",
      comments: "It was really useful, strongly recommended",
      rate: 4
    },
    {  user: "Alvaro Trezeguet",
      comments: "It lasted 2 days",
      rate: 1
    },
    {  user: "David Recoba",
      comments: "Awesome",
      rate: 5
    },
    {  user: "Jose Romero",
      comments: "Good value for money",
      rate: 4
```

```
    },
    {  user: "Antonio Cano",
       comments: "It broked really fast",
       rate: 2
    },
  ]
}
```

Explain this code

We have to show the product on our website, but we do not want to show all the reviews in a first view, we only need the average rate so that other customers can see in a fast way every product rate.

In this case, reduce can help to get the average fast. Go ahead an try to write the code, and then check our solution:

```
const totalReviews = product.reviews.reduce(function (acc, review) {
  return acc + review.rate;
}, 0); // starts acc to 0
const averageRate = totalReviews/product.reviews.length;

console.log(`averageRate rate: ${averageRate}`); // <== averageRate rate:  3.2
```

Explain this code

.filter()

The `.filter()` method iterates through an array and creates a new array with all elements that pass the condition we set.

Looking into ES5 syntax, we see that `.filter()` takes in a callback function. If that callback function returns `true`, then the item will be in the new array. If it returns `false`, then that item will not be in the new array.

```
var numbers = [1, 2, 3, 4, 5, 6];

var evenNumbers = numbers.filter(function(number){
  return number % 2 === 0;
});

console.log(evenNumbers); // <== [ 2, 4, 6 ]
```

Explain this code

The same but in ES6 syntax would be:

```
const numbers = [1, 2, 3, 4, 5, 6];

const evenNumbers = numbers.filter(number => number % 2 === 0);

console.log(evenNumbers); // <== [ 2, 4, 6 ]
```

Explain this code

Exercises

drinking age

Given the following array, filter the ones who are eligible to go to bars in the USA. 😆

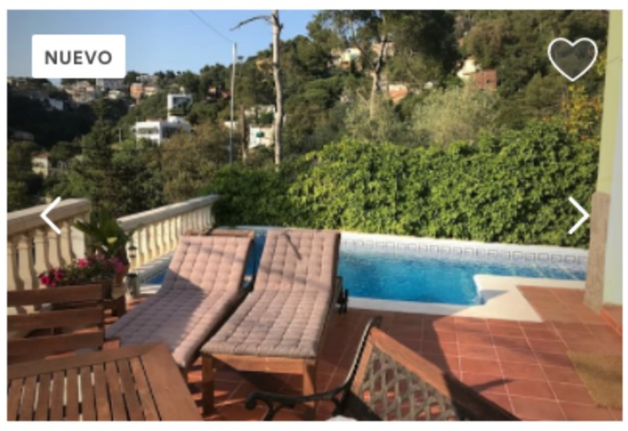
```
var people = [
  { name: "Candice", age: 25 },
  { name: "Tammy", age: 30 },
  { name: "Allen", age: 20 },
  { name: "Nettie", age: 21 },
  { name: "Stuart", age: 17 },
  { name: "Bill", age: 19 }
];

// your code...

console.log(ofDrinkingAge);
// [
//   { name: 'Candice', age: 25 },
//   { name: 'Tammy', age: 30 },
//   { name: 'Nettie', age: 21 }
// ]
```

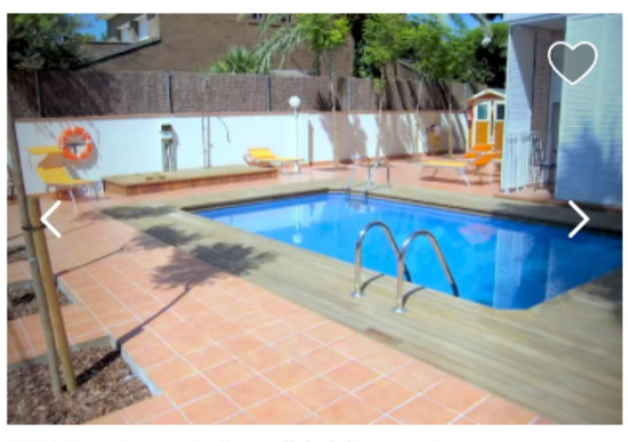
Explain this code

Swimming pool



78€ Suite doble muy Tranquila a 20' Centro de BCN
Habitación privada · 1 cama
2 evaluaciones

80€ Apartamento privado en urbanización
Casa entera · 4 camas
★★★★★ 56



119€ Apartamento Castelldefels.max.3 pax
Apartamento entero · 2 camas
★★★★½ 36

95€ CASA CON VISTAS A MAR MONTAÑA CIUDAD ...
Habitación privada · 1 cama
★★★★★ 7

We are working for Airbnb as developers, and we need to add a filter feature when someone is looking for a home. Imagine somebody starts their search for places in Barcelona, so we bring all the rooms available there.

But it´s summer, and the customers who are searching for rooms want the place to have a pool. Our incredible platform will include a filter feature, so we have to work on it.

Giving the following arrays of objects, let's filter just the one with a pool.

```
const places = [
 {
  title: "Awesome Suite 20' away from la Rambla",
  price: 200,
  type: "Private Room",
  pool: true,
  garage: false
 },
 {
  title: "Private apartment",
  price: 190,
  type: "Entire Place",
  pool: true,
  garage: true
 },
 {
  title: "Apartment with awesome views",
  price: 400,
  type: "Entire Place",
  pool: false,
  garage: false
 },
 {
  title: "Apartment in la Rambla",
  price: 150,
  type: "Private Room",
  pool: false,
  garage: true
 },
 {
  title: "Comfortable place in Barcelona´s center",
  price: 390,
  type: "Entire place",
  pool: true,
  garage: true
 },
 {
  title: "Room near Sagrada Familia",
  price: 170,
  type: "Private Room",
  pool: false,
  garage: false
 },
 {
  title: "Great house next to Camp Nou",
  price: 140,
  type: "Entire place",
  pool: true,
  garage: true
 },
 {
  title: "New apartment with 2 beds",
  price: 2000,
  type: "Entire place",
  pool: false,
  garage: true
 },
 {
  title: "Awesome Suite",
  price: 230,
  type: "Private Room",
```

```
    pool: false,
    garage: false
  },
  {
    title: "Apartment 10' from la Rambla",
    price: 930,
    type: "Entire place",
    pool: true,
    garage: true
  }
]
```

Explain this code

So, go ahead, try to write the code to filter the places and only get the ones with pool available. You can check ours after giving a shot.

```
// your code...

console.log(poolFilter);
// [
//   { title: 'Awesome Suite 20\' away from la Rambla',
//     price: 200,
//     type: 'Private Room',
//     pool: true,
//     garage: false },
//   { title: 'Private apartment',
//     price: 190,
//     type: 'Entire Place',
//     pool: true,
//     garage: true },
//   { title: 'Comfortable place in Barcelona´s center',
//     price: 390,
//     type: 'Entire place',
//     pool: true,
//     garage: true },
//   { title: 'Great house next to Camp Nou',
//     price: 140,
//     type: 'Entire place',
//     pool: true,
//     garage: true },
//   { title: 'Apartment 10\' from la Rambla',
//     price: 930,
//     type: 'Entire place',
//     pool: true,
//     garage: true }
// ]
```

Explain this code

odd numbers

Given an array of numbers, filter out the ones that are *not* even, and are greater than 42.

```
const numbers = [1, 60, 112, 123, 100, 99, 73, 45];

// your code...

console.log(result);
// [ 123, 99, 73, 45 ]
```

Explain this code

Summary

In this lesson, we learned how to iterate over arrays, and perform different tasks such as mapping, reducing or filtering.

Manipulating arrays to get the data we want is one of the most common tasks in programming so don't hesitate to spend some extra time on these topics since knowing them will make you better developer for sure.

Keep in mind that these methods don't mutate the original array.

Extra Resources

- MDN forEach
- MDN Map
- MDN Reduce
- MDN Filter
- Dan Martensen - Map, Reduce and Filter

Mark as completed

PREVIOUS
← JS | Functions Advanced

NEXT
JS | Arrays - Sort & Reverse →