

//.map method

**.map()**

Let me explain how it works with a simple example. Say you have received an array containing multiple objects – each one representing a person. The thing you really need in the end, though, is an array containing only the id of each person.

// What you have  
var officers = [  
 { id: 20, name: 'Captain Piett' },  
 { id: 24, name: 'General Veers' },  
 { id: 56, name: 'Admiral Ozzel' },  
 { id: 88, name: 'Commander Jerjerrod' }  
];

// What you need  
[20, 24, 56, 88]

There are multiple ways to achieve this. You might want to do it by creating an empty array, then using .forEach(), .for(...of), or a simple .for() to meet your goal.

Let’s compare!

Using .forEach():

var officersIds = [];

officers.forEach(function (officer) {  
 officersIds.push(officer.id);  
});

Notice how you have to create an empty array beforehand? Let’s see what it looks like when using .map():

var officersIds = officers.map(function (officer) {  
 return officer.id  
});

We can even be more concise with arrow functions (requires ES6 support, Babel or TypeScript)

const officersIds = officers.map(officer => officer.id);

So how does .map() work? Basically is takes 2 arguments, a callback and an optional context (will be considered as this in the callback) which I did not use in the previous example. The callback runs for **each value in the array**and **returns each new value** in the resulting array.

Keep in mind that the resulting array will always be the same length as the original array.

# .reduce()

reduce is a built-in array function

which allows us to transform an array into something else.

It takes a function as an input and this function receives two arguments passed in automatically by

javascript,

the previous value and the current value. The reduce method does not only accept these callback here

which is executed on every element in this array we return here,

it also accepts an initial value,

let's say an empty array.

So the initial value of the reduced value

now of course you want to adjust this reduced value by returning something

and it will then loop through all the elements and simply add them to the initial values step by step.

**.reduce()**

Just like .map(), .reduce() also runs a callback for each element of an array. What’s different here is that reduce passes the result of this callback (the accumulator) from one array element to the other.

The accumulator can be pretty much anything (integer, string, object, etc.) and must be instantiated or passed when calling .reduce().

Time for an example! Say you have an array with these pilots and their respective years of experience:

var pilots = [  
 {  
 id: 10,  
 name: "Poe Dameron",  
 years: 14,  
 },  
 {  
 id: 2,  
 name: "Temmin 'Snap' Wexley",  
 years: 30,  
 },  
 {  
 id: 41,  
 name: "Tallissan Lintra",  
 years: 16,  
 },  
 {  
 id: 99,  
 name: "Ello Asty",  
 years: 22,  
 }  
];

We need to know the total years of experience of all of them. With .reduce(), it’s pretty straightforward:

var totalYears = pilots.reduce(function (accumulator, pilot) {  
 return accumulator + pilot.years;  
}, 0);

Notice that I’ve set the starting value as 0. I could have also used an existing variable if necessary. After running the callback for each element of the array, reduce will return the final value of our accumulator (in our case: 82).

Let’s see how this can be shortened with ES6’s arrow functions:

const totalYears = pilots.reduce((acc, pilot) => acc + pilot.years, 0);

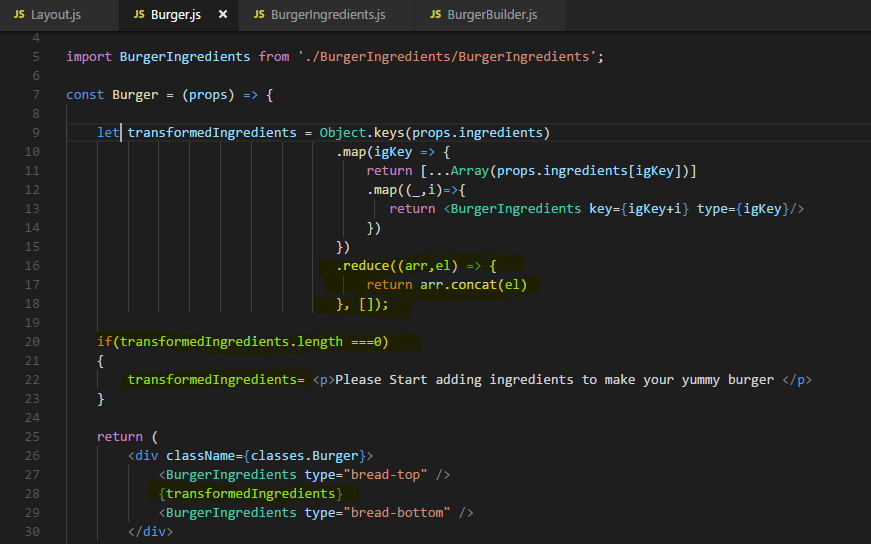
Now let’s say I want to find which pilot is the most experienced one. For that, I can use reduce as well:

var mostExpPilot = pilots.reduce(function (oldest, pilot) {  
 return (oldest.years || 0) > pilot.years ? oldest : pilot;  
}, {});

I named my accumulator oldest. My callback compares the accumulator to each pilot. If a pilot has more years of experience than oldest, then that pilot becomes the new oldest so that’s the one I return.

As you can see, using .reduce() is an easy way to generate a single value or object from an array.

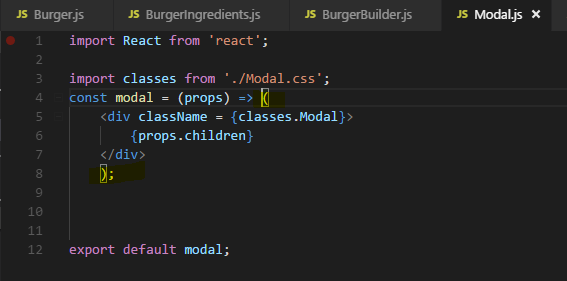
For example –



# Note : -

If you just have to return from arrow function then you don’t need opening and closing curly {} braces and also no return statement in functional component.

Example –



But if you have some logic to write then you need to do this-

Example –

