Current Skill Array functions

Array

Dealing with an array in JavaScript can be a bit complicated and hard to code. Here is an example to elaborate on that. We want to alert the index of the name Jack in the array people:

```
const people = [{ name: 'Max' }, { name: 'Jack' }, { name: 'Marry' }];
let i = 0;
while (i < people.length && people[i].name != 'Jack') {
 i++;
}
if (i !== people.length) {
 alert('Jack is in ' + i);
} else {
 alert('Cannot find Jack');
}
```

Array functions

It's time for ES6 to come and save the day and provide additional predefined functions to simplify working with arrays.

We are going to cover some of the important ones, such as:

- .find()
- .forEach()
- .filter()
- .map()
- .reduce()

Array function: find

Suppose that we want to find an element in an array, we would consider a **for** loop or a **while** loop to do so.

On the other hand, using ES6 **find()** function will make things very easy.

The find method is a HOC (higher order function), so it takes a function as a parameter. The parameter function defines the criteria for finding the desired element in the array.

The **find()** method will return the value of the first element that satisfies the search criteria.

Here is an example to illustrate that:

```
const people = [{ name: 'Max' }, { name: 'Jack' }, { name: 'Marry' }]
// JavaScript
function findPerson(name) {
 for (let i = 0; i < people.length; i++) {
   let person = people[i]
   if (person.name == name) {
     return person
   }
 }
}
// ES6
function findPerson(name) {
 return people.find(person =>person.name == name)
}
```

Array function for each()

Now, let's look at the **.forEach() array function**.

Suppose we want to perform an action on every element on a given array. The forEach method is our way for using predefined methods.

So what does it really do?

The **.forEach()** method will browse the array and execute the same instructions on every element in the same array.

Let's see an example ! We want to browse the people array and alert every name in it:

```
const people = [{ name: 'Max' }, { name: 'Jack' }, { name: 'Marry' }]
```

```
// JavaScript
function showEachOne(name) {
 for (let i = 0; i < people.length; i++) {
   alert(people[i].name)
 }
}

// ES6
const showEachOne = name => people.forEach(person => alert(person.name));
```

Array function: filter

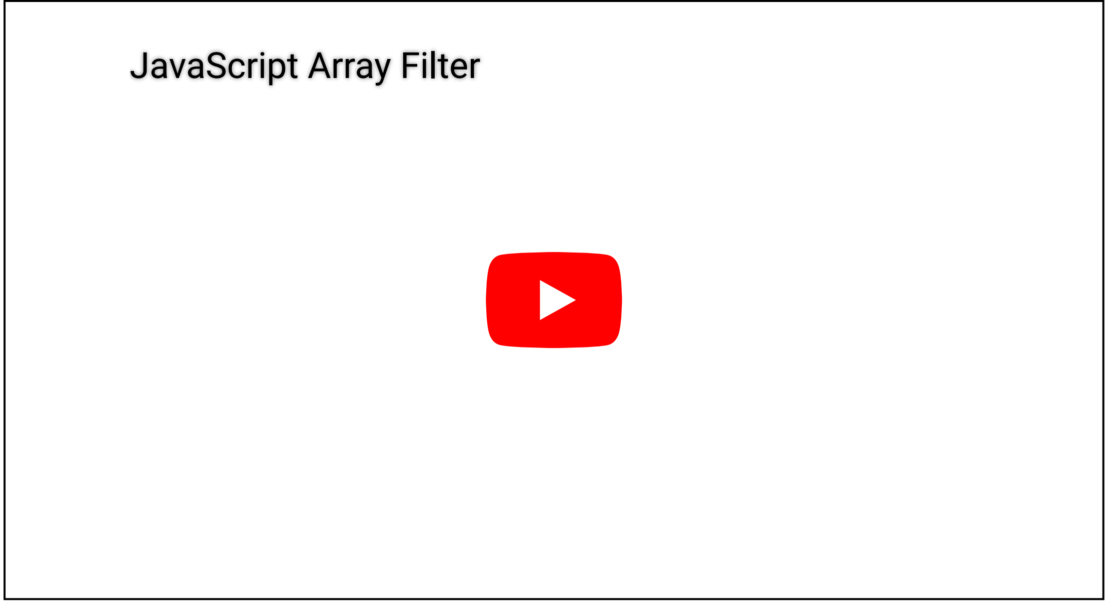
Now let's look at a more interesting **array function**. Imagine we have a cart full of groceries and we want to keep only cheap products. If we consider an array of products where every product is an object (as shown below).

Applying the **filter** method, we will have a ***new array*** that will eliminate all the products that have a price more than 10 dollars.

```
const products = [{name:"Milk",price:15}, {name:"Water", price:9}, {name:"Bread", pr
```

```
// JavaScript
function filterProducts() {
 let cheapProducts = [];
 for (let i = 0; i < products.length; i++) {
   if (products[i].price < 10) cheapProducts.push(products[i]);
 }
 return cheapProducts;
}

// ES6
const filterProducts = () => products.filter(product => product.price < 10);
```



Array function: map

ES6 also provides a function for array transformation using **.map()**.

Let’s suppose we want to add two dollars to the price of each product in the array.

You’re probably wondering by now: what’s the difference between the **.forEach()** and **.map()**. method? Well, it’s very simple. Like we’ve said, the **.forEach** executes the same instruction on every element in a given array and the **.map()** method does the same thing. However, it’s done on a copy of the array, leaving the original data unchanged.

Don’t hesitate to try it yourself !

```
const products = [
 { name: 'Milk', price: 15 },
 { name: 'Water', price: 9 },
 { name: 'Bread', price: 5 },
]

// JavaScript
function changeProducts() {
 for (let i = 0; i < products.length; i++) {
   products[i].price += 2
 }
 return products
}
```

```
// ES6
const changeProducts = () =>
  products.map(product => ({ ...product, price: product.price + 2 }));
console.log(changeProducts());
console.log(products);
```

Array function: reduce

Now, let's have a look at the **reduce()** function.

It is a built-in method that applies a function to each element in the array. It reduces the array to a single value.

The **reduce()** function executes the provided function for each value of an array from left-to-right.

The return value of a function is stored in an accumulator.

```
const data = [5, 10, 15, 20, 25]
const res = data.reduce((total, currentValue) =>  total + currentValue)
console.log(res)// 75
```

Array function recap

Let's recap all the different array functions using emojis:

[🍌, 🍓, 🍍].find(fruit ⇒ fruit.color === "red") // 🍓

[🍌, 🍓, 🍍].forEach(fruit ⇒ putInFridge(fruit)) // All fruits are in the fridge now

[🍊, 🥩,🍎, 🥕, 🍓, 🍐].filter(fruit⇒ isFruit(fruit)) // [🍊,🍎, 🍓, 🍐]

[🐔, 🐮, 🌽, 🥔].map(fruit⇒ isFruit(fruit)) // [🍗,🍔, 🍿, 🍟]

Here's a bonus array function that you can learn more about:

[🕯️, 🍫, 🍭].reduce(putTogether, 🥧) // 🎂