Array Improvements

This lesson covers the new methods to create and handle Array in ES6.

WE'LL COVER THE FOLLOWING

- `Array.from()`
- `Array.of()`
- `Array.find()`
- `Array.findIndex()`
- `Array.some()` & `Array.every()`

`Array.from()`

`Array.from()` is the first of many new array methods that ES6 introduced.

It will take something arrayish- meaning something that looks like an array but isn't- and transform it into a real array.

Look at the code in the JavaScript and the HTML tab to see what is happening.

Output

JavaScript

HTML

Apple

Banana

Orange

Console

Clear

```
[
   {},
   {},
   {}
]
```

```
[
   "Apple",
   "Banana",
   "Orange"
]
```

We can also simplify like this:

Output

JavaScript

HTML

CSS (SCSS)

```
document.addEventListener("DOMContentLoaded", function () {
const fruits = Array.from(document.querySelectorAll(".fruits p"));
const fruitNames = fruits.map(fruit => fruit.textContent);

console.log(fruitNames);
// ["Apple", "Banana", "Orange"]
});
```

Console

Clear

Now we transformed **fruits** into a real array, meaning that we can use any sort of method such as `map` on it.

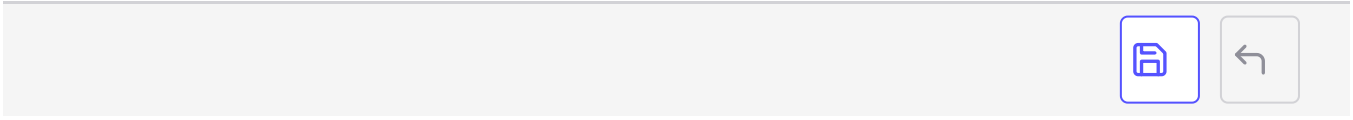
`Array.from()` also takes a second argument, a `map` function so we can write:

Output

JavaScript

HTML

```
document.addEventListener("DOMContentLoaded", function () {
  const fruits = document.querySelectorAll(".fruits p");
const fruitArray = Array.from(fruits, fruit => {
  console.log(fruit);
  // <p> Apple </p>
  // <p> Banana </p>
  // <p> Orange </p>
  return fruit.textContent;
  // we only want to grab the content not the whole tag
});
console.log(fruitArray);
// ["Apple", "Banana", "Orange"]
});
```



Console

Clear

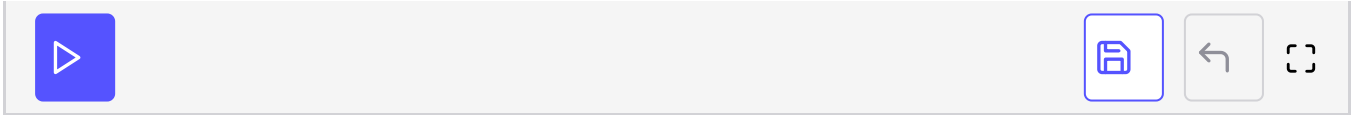
In the example above we passed a `map` function to the `.from()` method to push into our newly formed array. This includes only the `textContent` of the `p` tags and not the whole tag.

`Array.of()`

`Array.of()` will create an array with all the arguments we pass into it.

```
const digits = Array.of(1,2,3,4,5);
console.log(digits);

// Array [ 1, 2, 3, 4, 5];
```



`Array.find()`

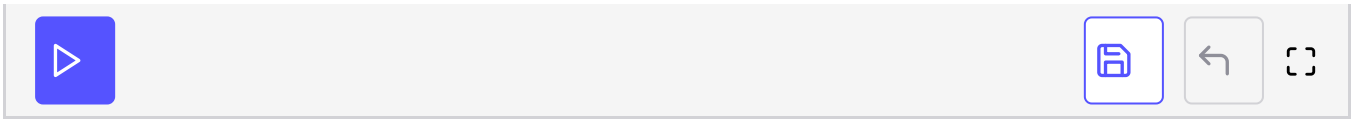
`Array.find()` returns the value of the first element in the array that satisfies

the provided testing function. Otherwise `undefined` is returned.

Let's lookw at a simple example to see how `Array.find()` works.

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

// this will return the first element in the array with a value higher than 3
let found = array.find( e => e > 3 );
console.log(found);
// 4
```



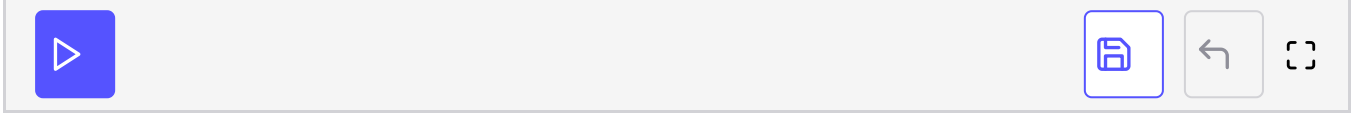
As we mentioned, it will return the **first element** that matches our condition, that's why we only have **4** and not **5**.

`Array.findIndex()`

`Array.findIndex()` will return the *index* of the **first** element that matches our condition.

```
const greetings = ["hello","hi","byebye","goodbye","hi"];

let foundIndex = greetings.findIndex(e => e === "hi");
console.log(foundIndex);
// 1
```



Again, only the index of the **first element** that matches our condition is returned.

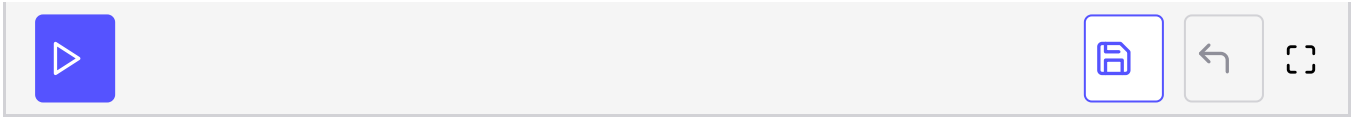
`Array.some()` & `Array.every()`

I'm grouping these two together because their use is self-explanatory: `.some()` will search if there are *some* items matching the condition and stop once it finds the first one. Whereas, `.every()` will check if all items match the given condition or not.

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

let arraySome = array.some( e => e > 2);
console.log(arraySome);
// true

let arrayEvery = array.every(e => e > 2);
console.log(arrayEvery);
// false
```



Simply put, the first condition is true, because there are **some** elements greater than 2, but the second is false because **not every element** is greater than 2.

Get ready for another quiz up next.