Back to Week 1

JS | Data Types in JavaScript - Arrays

LESSON

Learning Goals

After this lesson, you will be able to:

- Understand what arrays are and why they are useful
- Access an element using an array's index
- Add elements using **unshift** and **push** methods
- Remove elements using **splice**, **shift** and **pop** methods
- Iterate over an array with a for loop
- Iterate over an array with the **forEach** method
- Use `.split()` method to turn the string into the array of its elements

Why Arrays?

Arrays are data structures that allow us to group a collection of elements together in one variable.

We can later **access each of the elements individually** using an **index**, which represents the position of each element in the structure of an array, or we can pass them around grouped as the array.

For example, if you have a class of several people and want to keep all their names saved in variables, instead of having one for each you can create one array. Let's take a look at how that works.

Arrays operations

Declaration

An array can be declared empty:

```
const arrayNames = [];
```

Explain this code

Or you can declare it with some elements already in it:

```
const arrayNames = ['Pedro', 'Jake', 'Joan', 'Mike'];
```

Explain this code

The elements of the array don’t have to be all the same type; we can mix strings, numbers or any other type of data we want.

```
const arrayNames = ['Pedro', 2, true];
```

Explain this code

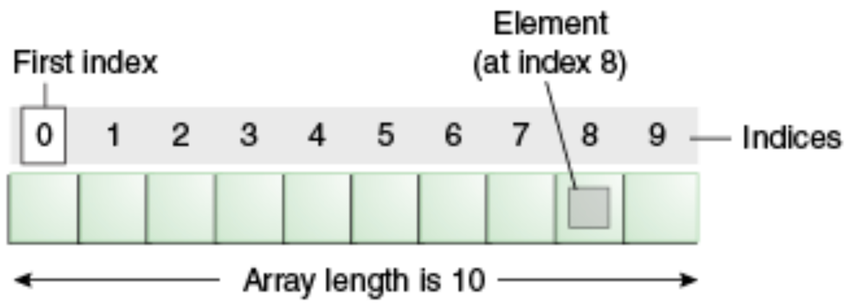
Accessing Elements

We can access individual elements in the array by their position in the array. The position is named *index*.

Index	`0`	`1`	`2`	`3`
Value	`"Pedro"`	`"Jake"`	`"Joan"`	`"Mike"`

⚠️ **Zero-Indexed:** Notice how the index of the first element is not `1` as we may think, but `0`!

The **length of the array** is the number of elements the array is storing at a particular point in time. So if an array has `10` elements, the first index will be `0` and the last one `9` *(length - 1)*



Let’s try to print *Pedro* in the console:

```
const arrayNames = ['Pedro', 'Jake', 'Joan', 'Mike'];
console.log(arrayNames[0]); // <== Pedro
```

Explain this code

Simple as that, you can access the elements inside an array by referencing their position in the array, but remember every array starts at *0*, not at *1*! This position is called **index** commonly, so the index 0 of *arrayNames* returns/has “Pedro” as its value.

```
console.log(arrayNames[1]); // <== Jake
console.log(arrayNames[2]); // <== Joan
console.log(arrayNames[3]); // <== Mike
console.log(arrayNames[99]); // <== undefined
```

Explain this code

⚠️ If we try to access an index that does not exist, it will return **undefined**.

Adding Elements

We already know how to create arrays with initial values inside them, but what if we don't know at the moment of creation which values will be stored there? Don't worry, we can easily add them later with **push** method.

Now, arrayNames has a fourth element, and if we try again to get the fourth position (index 3), it gives us *Rachel*.

An important thing to keep in mind is that, **using `.push()` method, the new value is stored at the end of the array**, not at the beginning or in a random position.

If you wanted to **add an element at the beginning of the array**, use **unshift** instead of *push*:

```
const arrayNames = ['Pedro', 'Jake', 'Joan'];
arrayNames.unshift('Rachel');
console.log(arrayNames[0]); // <== Rachel
```

Explain this code

Removing Elements

What if we want to delete an element from an array? The **splice** method allows us to do it!!

> **array.splice(start, deleteCount)**
>
> **Start:** Index at which to start changing the array
> **deleteCount:** number of old array elements to remove

Let's see it in action!

JS

Result

EDIT ON CODEPEN

```
var arrayNames = ["Pedro", "Jake", "Joan"];

console.log(arrayNames[0]);
arrayNames.splice(0,1);
console.log(arrayNames[0]);
```

Resources

1× 0.5× 0.25×

Rerun

So, in this example, we are starting at index `0` (first element of the array) and we want to delete 1 element. Finally, if we access the first element again, it gives us *Jake*, which was the second element originally.

Try it yourself:

```
arrayNames.splice(0, 2);
arrayNames.splice(1, 1);
arrayNames.splice(2, 1);
```

Explain this code

But there are other ways to delete elements. For instance, the **pop** method allows us to delete the last value, while **shift** deletes the first one.

Summary of Methods

Method	Action
`push`	Adds an element at the end
`unshift`	Adds an element at the beginning
`shift`	Removes the first element
`pop`	Removes the last element
`splice`	Removes elements from anywhere in the array

Iterating over each element in an array

Let's say we want to add up all the numbers in an array, or we want to say hello to each of the names in an array. For things like this, it is very useful to be able to iterate over several/all elements in the array with one piece of code. So how do we do that?

You already know what a loop is, for this example, we are going to use the `for` loop. For instance, imagine we want to print all the names inside `arrayNames`. Maybe the first thing you think is:

```
console.log(arrayNames[0]);
console.log(arrayNames[1]);
console.log(arrayNames[2]);
```

Explain this code

But what if we have hundreds of names, or even thousands? This is how we can accomplish something like this with a `for` loop:

```
const arrayNames = ['Pedro', 'Jake', 'Joan'];
for (let i = 0; i < arrayNames.length; i++) {
  console.log(arrayNames[i]);
}
```

Explain this code

Now it doesn't matter how many elements are in the array, this loop is going to print them all in just three lines of code.

`forEach` method

Iterating over an array with `for` is cool, but since iterating over arrays is something we are going to be doing very often, JavaScript provides us with a much cleaner way of expressing the same idea. Welcome your new friend, the **forEach** method!

JS | Result | EDIT ON CODEPEN

```
var arrayNames = ["Pedro", "Jake", "Joan"];

arrayNames.forEach(function(name){
  console.log(name);
})
```

Resources | 1× | 0.5× | 0.25× | Rerun

> **`forEach`** method iterates through all the elements of an array, and **FOR EACH** element in the array, it will call another function, passing in it each element, one by one.

`forEach` receives only one parameter: a function. This function doesn't need a name (it's anonymous), but it can have zero, one, two or three parameters.
In case you are familiar with arrow functions syntax, you know we can shorten the previous syntax a bit:

```
const arrayNames = ['Pedro', 'Jake', 'Joan'];
arrayNames.forEach(name => console.log(name));

// console:
// Pedro
// Jake
// Joan
```

Explain this code

If this syntax doesn't make sense at this moment, it is completely ok. Very soon you will know what arrow functions are and how to write them.

No parameters

Will just call the function as many times as elements are in the array:

```
// ES5:
['a', 'b'].forEach(function () {
  console.log('hi!');
});

// ES6:
['a', 'b'].forEach(() => console.log('hi!'));

// => hi!
// => hi!
```

Explain this code

First parameter: Element

If we pass one parameter, it will be equal to each element on every iteration.

```
// ES5:
//                        placeholder, can be anything (naming has to make sense though)
//                            |
[1, 2, 3, 4].forEach(function (element) {
  console.log(element * 2); // we can access each element inside!
});

// ES6:
[1, 2, 3, 4].forEach(element => console.log(element * 2));

// console:
// => 2
// => 4
// => 6
// => 8
```

Explain this code

Second parameter: Index

```
const raceResults = ['Helen', 'John', 'Peter', 'Merry'];
raceResults.forEach(function (elem, index) {
  console.log(elem + ' finished the race in ' + (index + 1) + ' position!');
});

// => Helen finished the race in 1 position!
// => John finished the race in 2 position!
// => Peter finished the race in 3 position!
// => Merry finished the race in 4 position!
```

Explain this code

📝 Practice time

Write the function above using arrow function approach.

Your turn

Can you guess the output of this?

```
function printStars(howMany) {
  console.log('*'.repeat(howMany));
}

[1, 2, 3, 4, 5].forEach(function (num) {
  printStars(num);
});
```

Explain this code

String `.split()` method

It might sound like a mistake to introduce a string related method in the array lesson, but we have a good reasoning behind it.

> The **split** method allows us to separate a string into pieces and will **return** all the pieces as **elements of a new array**.

So, imagine we have a loooong string and we want to count the number of words inside:

JS

```
let phrase = "This is long enough for a string to
count it words";

let words = phrase.split(" ");

console.log(words);
console.log(words[0]);
console.log(words.length);
```

Result

EDIT ON CODEPEN

Resources 1× 0.5× 0.25× Rerun

Step one would be to take the string and put each word separately into an array. How? Applying the *split* method to a string, we will need to set the **separator**, in this case a space (" "), which is the character where the split is going to cut the string and add the rest of the characters before it, and then do it again every time it finds that character.

Summary

Arrays are data structures that allow us to store a collection of elements, doesn't matter the type. We can manipulate arrays, getting, changing, adding or deleting their elements.

We also have several ways to go through all of their values, like loops such us *for*, or methods like *forEach*.

Extra Resources

- MDN Array
- JavaScript Arrays - tips, tricks and examples
- .split()
- Does it mutate

Mark as completed

PREVIOUS
← JS | Functions Intro

NEXT
LAB | JavaScript Functions and Arrays →