JavaScript Array Iteration

Array iteration methods operate on every array item.

JavaScript Array forEach()

The forEach() method calls a function (a callback function) once for each array element.

Example

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value + "<br>";
}
```

Here myFunction() is our callback function.
What is callback function: a function that we pass as parameter to another function.

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter.

Exercise: Creating an HTML list of names

```
let names = ['Jim', 'John', 'Jack', 'Jones']

let myHTML = '<ol>';

function myFunction(value, index, array) {
  myHTML += ("<li>" + value + "</li>")
}

names.forEach(myFunction)

myHTML += "</ol>"
```

```
"<ol><li>Jim</li><li>John</li><li>Jack</li><li>Jones</li></ol>"
```

JavaScript Array map()

The map() method creates a new array by performing a function on each array element.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

This example multiplies each array value by 2:
Example:

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);


function myFunction(value, index, array) {
  return value * 2;
} /* What we are returning from myFunction() is going to variable 'numbers2'. */
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted.

Exercise: Square root of every number

let arr = [3, 4, 5, 6, 7, 8, 9]

arr2 = arr.map(Math.sqrt)

Array(7) [1.73, 2, 2.23, 2.44, 2.64, 2.82, 3]

/* Map each element of my given array onto a new array by finding the square root of elements */

Flat Array and Nested Array

An example:

Flat array: [1, 2, 3, 4, 5, 6, 7, 8]

Nested array: [[1,2], [3,4], [5,6], [7,8]]

JavaScript Array flatMap()

The flatMap() method first maps all elements of an array and then creates a new array by flattening the array.
Example

```
const myArr = [1, 2, 3, 4, 5, 6];
const newArr = myArr.flatMap((x) => x * 2);
```

Problem

Flatten a given array with and without using flatMap():

```
let arr_nested = [[1,2], [3,4], [5,6], [7,8]]

arr_nested.flatMap((value, index, array) => value)

Array(8) [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

JavaScript Array filter()

The filter() method creates a new array with array elements that pass a test.

This example creates a new array from elements with a value larger than 18:
Example

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Here again, our callback function is myFunction()

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

In the example above, the callback function does not use the index and array parameters, so they can be omitted.

Return values of callback function for forEach(), map() and filter()

Callback function of forEach(): not returning anything

Callback function of map(): returns modified value

Callback function of filter(): returns true or false

JavaScript Array reduce()

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

The reduce() method works from left-to-right in the array. See also reduceRight().

The reduce() method does not reduce the original array.

This example finds the sum of all numbers in an array:
Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);


function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters. It can be rewritten to:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);


function myFunction(total, value) {
  return total + value;
}
```

The reduce() method can accept an initial value:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction, 100);
/* The '100' here is the initial value */


function myFunction(total, value) {
  return total + value;
}
```

Subtraction using reduce() with and without initial value

```
numbers = [175, 50, 25];
total = numbers.reduce(myFunc)

function myFunc(total, num) {
  return total - num;
}

100
/* By subtracting values from indices 1, 2, 3... so on from the value at first index 0. */

numbers = [175, 50, 25];
total = numbers.reduce(myFunc, 1000)

function myFunc(total, num) {
  return total - num;
}

750
```

Problem: Explain the output of the code.

```
const getMax = (a, b) => Math.max(a, b);

// callback is invoked for each element in the array starting at index 0
[1, 100].reduce(getMax, 50); // 100

[50].reduce(getMax, 10); // 50

// If we have an initial value, the first two values become value from index 0 and initial value itself.

// callback is invoked once for element at index 1
[1, 100].reduce(getMax); // 100
[1000, 100].reduce(getMax); // 1000

// Above, initial value is not given, so the first two values become values at indices 0 and 1.

// callback is not invoked
[50].reduce(getMax); // 50
[].reduce(getMax, 1); // 1

[].reduce(getMax); // TypeError
```

Explanation

```
const numbers = [4, 9, 45, 16, 25];
const getMax = (a, b) => Math.max(a, b);
let max = numbers.reduce(getMax);
```

Output:
'max' is 45.

Explanation

Iteration 1:

4 and 9 will be compared. Value returned is maximum of 4 and 9 i.e., 9.

Iteration 2:

9 and 45 will be compared. Value returned is maximum of these two, i.e., 45.

Iteration 3:

45 and 16 will be compared. Value returned is maximum of these two, i.e., 45.

Iteration 4:

45 and 25 will be compared. Value returned is maximum of these, i.e., 45.

One more example

```
const getMax = (a, b) => Math.max(a, b);

// callback is invoked for each element in the array starting at index 0
[1, 100].reduce(getMax, 50); // 100
```

Here:

Iteration 1:

1 and 50 will compared. Value returned is maximum of 1 and 50, i.e., 50

Iteration 2:

50 and 100 will be compared. Value returned is maximum of these two, i.e., 100.

How we can check if ‘a’ has the returned value from each iteration?

```
function getMax (a, b) {
        console.log(a);
        return Math.max(a, b);
} /* To track what is coming in ‘a’ */
```

JavaScript Array reduceRight()

The reduceRight() method runs a function on each array element to produce (reduce it to) a single value.

The reduceRight() works from right-to-left in the array. See also reduce().

The reduceRight() method does not reduce the original array.

This example finds the sum of all numbers in an array:

Example

```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduceRight(myFunction);


function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

The example above does not use the index and array parameters.

JavaScript Array every()

The every() method checks if all array values pass a test.

This example checks if all array values are larger than 18:
Example

```
const numbers = [45, 4, 9, 16, 25];
let allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

When a callback function uses the first parameter only (value), the other parameters can be omitted

JavaScript Array some()

The some() method checks if some array values pass a test.

This example checks if some array values are larger than 18:
Example

```
const numbers = [45, 4, 9, 16, 25];
let someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

JavaScript Array indexOf()

The indexOf() method searches an array for an element value and returns its position.

Note: The first item has position 0, the second item has position 1, and so on.
Example

Search an array for the item "Apple":
```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.indexOf("Apple") + 1;
```
Syntax
```
array.indexOf(item, start)
```
item Required. The item to search for.
start Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

Array.indexOf() returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

JavaScript Array lastIndexOf()

Array.lastIndexOf() is the same as Array.indexOf(), but returns the position of the last occurrence of the specified element.

Example

Search an array for the item "Apple":

```
const fruits = ["Apple", "Orange", "Apple", "Mango"];
let position = fruits.lastIndexOf("Apple") + 1;
```

Syntax

```
array.lastIndexOf(item, start)
```

item Required. The item to search for

start Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning

JavaScript Array find()

The find() method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

Example

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

JavaScript Array findIndex()

The findIndex() method returns the index of the first array element that passes a test function.

This example finds the index of the first element that is larger than 18:
Example:

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

JavaScript Array.from()

The Array.from() method returns an Array object from any object with a length property or any iterable object.
Example

Create an Array from a String:
Array.from("ABCDEFG");

JavaScript Array Keys()

The Array.keys() method returns an Array Iterator object with the keys of an array.
Example

Create an Array Iterator object, containing the keys of the array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = fruits.keys();

for (let x of keys) {
  text += x + "<br>";
}

/* x contains the indices */
```

JavaScript Array entries()

Create an Array Iterator, and then iterate over the key/value pairs:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x;
}
```

The entries() method returns an Array Iterator object with key/value pairs:

```
[0, "Banana"]
[1, "Orange"]
[2, "Apple"]
[3, "Mango"]
```

The entries() method does not change the original array.

JavaScript Array includes()

ECMAScript 2016 introduced Array.includes() to arrays. This allows us to check if an element is present in an array (including NaN, unlike indexOf).
Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.includes("Mango"); // is true
```

Array.includes() allows to check for NaN values. Unlike Array.indexOf().

Methods filter(), every() and some()

With respect to our examples with numbers and comparison with 18:

filter(): returns numbers greater than 18

every(): returns true if all numbers are greater than 18

some(): returns true if any of the numbers is greater than 18

find(): callback function returns true if a condition is met.
The find() method returns the value of the first array element that passes a test function.

findIndex(): callback function returns true if a condition is met.
The findIndex() method returns the index of the first array element that passes a test function.

JavaScript Array Spread (...)

JavaScript Array Spread (...)

The ... operator expands an iterable (like an array) into more elements:
Example

```
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
```