



# JS ARRAY METHODS COURSE

APRIL 2022

YONATAN BENEZRA

**- DO NOT DISTRIBUTE –**

**- DO NOT COPY – ALL RIGHTS RESERVED TO THE  
AUTHOR –**

In JavaScript, an array is a data structure that contains a list of elements which store multiple values in a single variable.  
The strength of JavaScript arrays lies in the array methods.

Array methods are functions built-in to JavaScript that we can apply to our arrays — each method has a unique function that performs a change or calculation to our array and saves us from writing common functions from scratch.

## JavaScript Array Methods

### 1. map()

This method creates a new array, resulting in calling a provided function on every element in this array.

```
1  const arr = [1, 2, 3, 4, 5, 6];
2  const mapped = arr.map(element => element + 30);
3  console.log(mapped); // [31, 32, 33, 34, 35, 36]
```

#### **When not to use map()**

Since map builds a new array, using it when you aren't using the returned array is an anti-pattern; in those cases, you will be better off using forEach or for...of instead.

You shouldn't be using map if:

- You're not using the array it returns; and/or
- You're not returning a value from the callback.

### 2. filter()

This method creates a new array, consisting only of elements that pass the condition inside the provided function.

```
1  const arr = [1, 2, 3, 4, 5, 6];
2  const filtered = arr.filter(element => element === 2 || element === 4);
3  console.log(filtered); // [2, 4]
```

### 3. sort()

This method is used to arrange/sort the array's elements, either in ascending or descending order.

```

1  const arr = [1, 2, 3, 4, 5, 6];
2  const alphabet = ["f", "a", "c", "v", "z"];
3
4  // sort in descending order
5  descend = arr.sort((a, b) => a > b ? -1 : 1);
6  console.log(descend); // [6, 5, 4, 3, 2, 1]
7
8  // sort in ascending order
9  ascend = alphabet.sort((a, b) => a > b ? 1 : -1);
10 console.log(ascend); // ["a", "c", "f", "v", "z"]

```

#### 4. forEach()

This method helps to loop over an array by executing a provided callback function for each element in the given array.

```

1  const arr = [1, 2, 3];
2  arr.forEach(element => {
3    console.log(element);
4  });
5  // 1
6  // 2
7  // 3

```

#### 5. concat()

This method is used to merge two or more arrays, returning a new array, without effecting the existing arrays.

```

1  const arr1 = ["a", "b", "c"];
2  const arr2 = ["d", "e", "f"];
3
4  console.log(arr1.concat(arr2)); // ["a", "b", "c", "d", "e", "f"]
5  console.log(arr1); // ["a", "b", "c"]
6  console.log(arr2); // ["d", "e", "f"]

```

#### 6. every()

This method checks each element of the array that passes the condition, returning true or false accordingly.

```
1  const arr = [1, 2, 3, 4, 5, 6, 7];
2
3  // all elements are greater than 5
4  const greaterFive = arr.every(num => num > 5);
5  console.log(greaterFive); // false
6
7  // all elements are less than 9
8  const lessnine = arr.every(num => num < 9);
9  console.log(lessnine); // true
```

## 7. some()

This method checks if **at least one** element of the array passes the condition, returning true or false accordingly.

```
1  const arr = [1, 2, 3, 4, 5, 6, 7];
2
3  // at least one element is greater than 5?
4  const greaterNum = arr.some(num => num > 5);
5  console.log(greaterNum); // true
6
7  // at least one element is less than or equal to 0?
8  const lessNum = arr.some(num => num <= 0);
9  console.log(lessNum); // false
```

## 8. includes()

This method checks if an array includes the element that passes the condition, returning true or false accordingly.

```
1  const arr = [1, 2, 3, 4, 5, 6];
2  console.log(arr.includes(2)); // true
3  console.log(arr.includes(7)); // false
```

## 9. join()

This method returns a new string by concatenating each of the array's elements, separating them by the specified separator.

```
1  const arr = ["m", "a", "n", "d", "e", "e", "p"];
2  console.log(arr.join('')); // mandeep
```

### 10. reduce()

This method applies a function towards an accumulator and each element in the array in turn, resulting in a single (reduced) value.

```
1  const arr = [1, 2, 3, 4, 5, 6];
2  const reduced = arr.reduce((total, current) => total + current);
3  console.log(reduced); // 21
```

### 11. find()

This method returns the value of the first element in an array that passes the test of a testing function.

```
1  const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
2  const found = arr.find(element => element > 5);
3  console.log(found); // 6
```

### 12. findIndex()

This method returns the index of the first element in an array that passes the test of a testing function.

```
1  const arr = ["Danny", "Mandeep", "John", "Ruby"];
2  const indexFinder = arr.findIndex(element => element === "Mandeep");
3  console.log(indexFinder); // 1
```

### 13. indexOf()

This method returns the index of the first occurrence of the specified element in the array, or -1 if none is not found.



```
1  const arr = ["Danny", "Mandeep", "John", "Ruby"];
2  const indexFinder = arr.indexOf("Mandeep");
3  console.log(indexFinder); // 1
```

#### 14. fill()

This method fills the elements in an array with a static value and returns the modified array.

```
1  const arr = new Array(3);
2  console.log(arr); // [empty, empty, empty]
3  console.log(arr.fill(10)); // [10, 10, 10]
```

#### 15. slice()

This method returns a new array with specified start to end elements.

```
1  const arr = ["a", "b", "c", "d", "e"];
2  const sliced = arr.slice(2, 4);
3  console.log(sliced); // ["c", "d"]
4  console.log(arr); // ["a", "b", "c", "d", "e"]
```

#### 16. reverse()

This method reverses the placement of each element in the array. The element placed in the last index space will now be first, and the element at index 0 will now be last.

```
1  const arr = [1, 2, 3];
2  arr.reverse();
3  console.log(arr); // [3, 2, 1]
```

#### 17. push()

This method adds one or more elements to the end of an array and returns the new length of the array.

```
1  const fruits = ["Apple", "Peach"];
2  console.log(fruits.push("Banana")); // 3
3  console.log(fruits); // ["Apple", "Peach", "Banana"]
```

### **18. pop()**

This method removes the last element from the end of an array and returns that element.

```
1  const fruits = ["Apple", "Peach"];
2  fruits.pop();
3  console.log(fruits); // ["Apple"]
```

### **19. shift()**

This method removes the first element from an array and returns that element.

```
1  const fruits = ["Apple", "Peach"];
2  fruits.shift();
3  console.log(fruits); // ["Peach"]
```

### **20. unshift()**

This method adds one or more elements to the beginning of an array and returns the new length of the array.

```
1  const fruits = ["Apple", "Peach"];
2  console.log(fruits.unshift("Banana")); // 3
3  console.log(fruits); // ["Banana", "Apple", "Peach"]
```

### **Conclusion**

To make JavaScript array manipulation easier, array methods can be used to simplify tasks that would otherwise be much more complicated, resulting in shorter and cleaner code.