30 Javascript Array Methods: A Cheat Sheet for Developers

Silviya S · Follow

6 min read · Aug 12, 2023

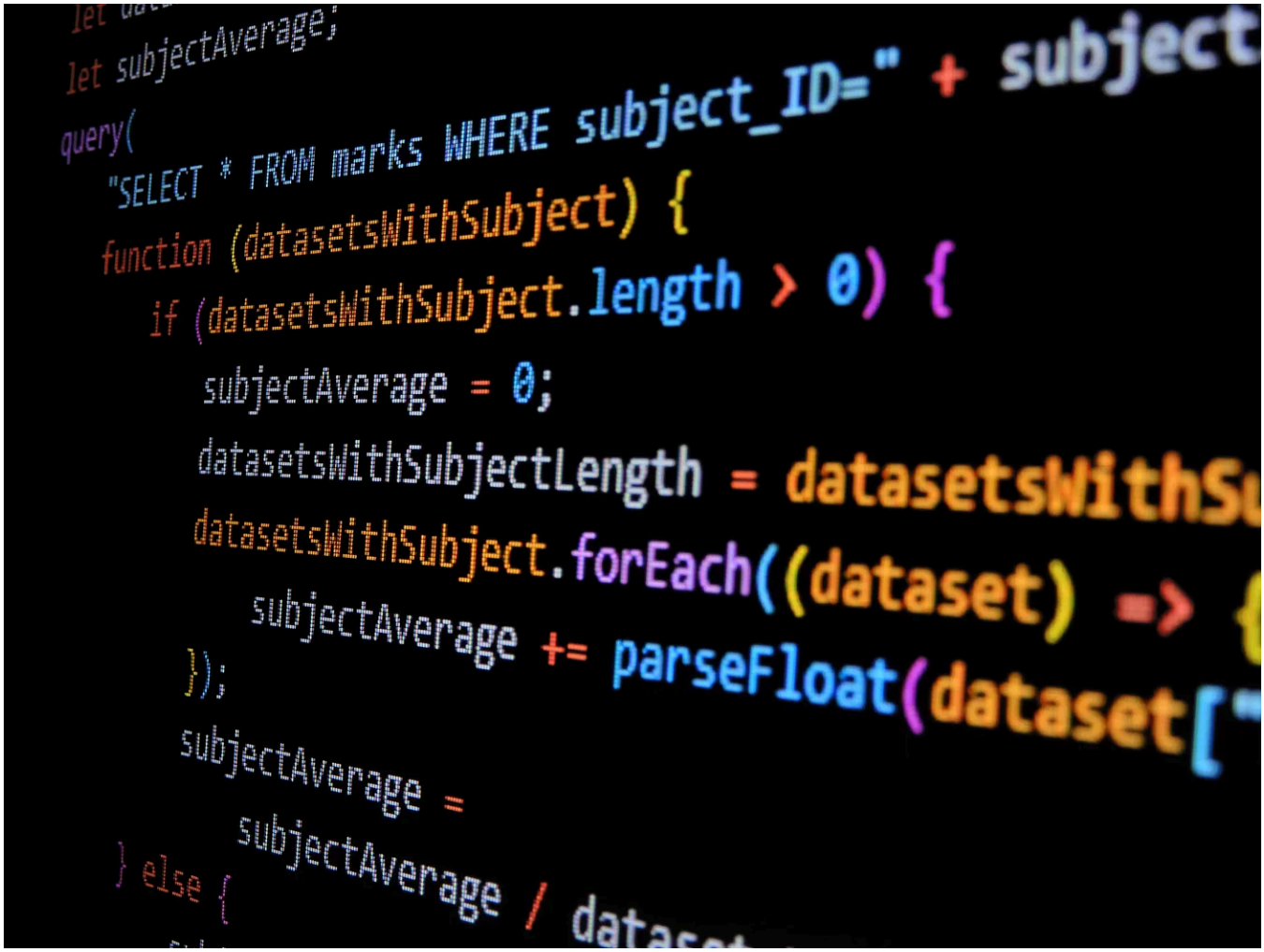

Photo by Gabriel Heinzer on Unsplash

JavaScript, an array is a data structure that contains a list of elements that store multiple values in a single variable. Array methods are functions built-in to

JavaScript that we can apply to our arrays. JavaScript has multiple array methods and each method has unique functionalities.

1. map()

The map method lets you loop on each element inside the array, and manipulate them as per the requirement.

```
const numbers = [2, 4, 6, 8, 10];
const numbersMap = numbers.map(element => element * 2);
console.log(numbersMap); //[4, 8, 12, 16, 20];
```

2. filter()

The filter method lets you create a new array based on conditions that evaluate to `true` from the original array.

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

3. reduce()

The reduce method applies a function against an accumulator and each element in the array to reduce it to a single value.

```
const numbers = [2, 4, 8, 1];
const reducedNumbers = numbers.reduce((accumulator, current) => {
  return accumulator + current;
}); // 15
```

4. reduceRight()

The reduceRight method executes a reducer function that you provide on each element of the array resulting in a single output value(from right to left).

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

```
numbers.reduceRight((accumulator, current) => accumulator + current, 0); // 15
```

5. push()

The push method adds a new element to the end of an array and returns the new length. It modifies the original array.

```
const numbers = [1, 2, 3, 4, 5, 6];
numbers.push(7);
console.log(numbers); // [1, 2, 3, 4, 5, 6, 7]
```

6. pop()

The pop method removes the last element of an array and returns that element. It modifies the original array.

```
const numbers = [1, 2, 3, 4, 5, 6];
numbers.pop(); // 6
console.log(numbers); // [1, 2, 3, 4, 5]
```

7. shift()

The shift method removes the first element of an array and returns the first element of an array. It changes the original array.

```
const numbers = [10, 20, 30, 40, 50];
numbers.shift(); // 10
console.log(numbers); // [20, 30, 40, 50];
```

8. unshift()

The unshift method adds new items to the beginning of an array and returns the new length. It changes the original array.

```
const numbers = [10, 20, 30, 40, 50];
numbers.unshift(1); // 6
console.log(numbers); // [1, 10, 20, 30, 40, 50];
```

9. splice()

The splice method adds/removes items to/from an array, and returns the removed item(s). It changes the original array.

```
const numbers = [1, 2, 3, 4, 5, 6, 7];
numbers.splice(2, 0, 9, 10);
console.log(fruits); // [1, 2, 9, 10, 3, 4, 5, 6, 7]
```

10. slice()

The slice method returns the selected elements in an array, as a new array object. This does not affect the original array.

```
const numbers = [10, 20, 30, 40, 50, 60];
const newNumbers = numbers.slice(1, 3);
console.log(newNumbers); // [20, 30]
```

11. concat()

The concat method is used to join two or more arrays. It does not change the existing arrays but returns a new array, containing the values of the joined array.

```
const arr1 = [1,2,3];
const arr2 = [4,5,6];
const arr3 = arr1.concat(arr2);
console.log(arr3); //[1,2,3,4,5,6]
```

12. some()

This method returns true if at least one element in the array passes the test implemented by the provided function.

```
const numbers = [1, 2, 3, 4, 5];
numbers.some((element) => element === 3); // true
numbers.some((element) => element === 6); // false
```

13. every()

In an array, this method checks every element in the array that passes the condition, returning true or false as appropriate.

```
const numbers = [1, 2, 3, 4, 5];
numbers.every((element) => element > 3); // false
numbers.every((element) => element < 6); // true
```

14. fill()

The fill method fills the specified elements in an array with a static value. We can specify the position of where to *start* and *end* the filling.

```
var numbers = [1,1,1,1,1];
numbers.fill(0, 1, 3);
console.log(numbers); // [1, 0, 0, 1, 1]
```

15. reverse()

The reverse method reverses the order of the elements in an array. It modifies the original array.

```
const numbers = [1, 2, 3, 4, 5];
numbers.reverse(); // [5, 4, 3, 2, 1]
numbers; // [5, 4, 3, 2, 1]
```

16. includes()

The includes method takes an element as an input and returns true or false if the array indeed contains this exact element.

```
const numbers = [1,10,50,100,200];
const numberIsIncluded = numbers.includes(100);
console.log(numberIsIncluded) // true
```

17. at()

This method returns a value at the specified index. It does not modify the original array.

```
const numbers = [10, 20, 30, 40, 50];
numbers.at(4); // 50
```

18. find()

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

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

19. forEach()

The forEach method helps to loop over an array by executing a provided callback function for each element in an array.

```
const numbers = [1, 2, 3];
numbers.forEach((ele) => {
    console.log(ele);
});
//1
//2
//3
```

20. flat()

The flat method returns a new array with all sub-array elements concatenated into it recursively up to the specified depth. It does not modify the original array.

```
const numbers = [1, 2, [3, 4, [5, 6]]];
numbers.flat(Infinity); // [1, 2, 3, 4, 5, 6]
```

21. flatMap()

The flatMap method returns a new array formed by applying a given callback function to each element of the array.

```
const numbers = [1, 2, 3];
numbers.flatMap((element) => [element, element * element]); // [1, 1, 2, 4, 3,
```

22. indexOf()

The indexOf method returns the first index at which a given element can be found in the array, or -1 if it is not present.

```
const list = [1, 2, 3, 4, 5];
list.indexOf(3); // 2
list.indexOf(5); // 4
list.indexOf(7); // -1
```

23. findIndex()

The findIndex method returns the index of the element in an array that passes a condition or -1 if it is not present. It executes the function once for each element present in the array.

```
const numbers = [1,2,3,4];
const indexFound = numbers.findIndex(element => element === 3);
console.log(indexFound); //2
```

24. lastIndexOf()

The lastIndexOf method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backward, starting

at fromIndex.

```
const numbers = [1, 2, 3, 4, 5];
numbers.lastIndexOf(3); // 2
numbers.lastIndexOf(6); // -1
```

25. join()

The join method joins all elements of an array into a string. It does not modify the original array.

```
const numbers = [1, 2, 3, 4, 5];
numbers.join(''); // "12345"
```

26. sort()

The sort method sorts the items of an array. The sort order can be either alphabetic or numeric, and either ascending or descending.

```
const alpabeticalSort = ['Virginia', 'Cary', 'Tennessee', 'Alaska'];
alpabeticalSort.sort(); // ['Alaska', 'Cary', 'Tennessee', 'Virginia']
```

27. toString()

This method converts the elements of a specified array into string form, without affecting the original array.

```
const numbers = [1, 2, 3, 4, 5];
numbers.toString(); //'1,2,3,4,5'
```

28. from()

The from method creates a new array from an array-like or iterable object.

```
const set = new Set([1, 2, 3, 4, 5, 6]);
Array.from(set); // [1, 2, 3, 4, 5 ,6]
```

29. entries()

It creates an iterator object and a loop that iterates over each key/value pair.

```
const numbers = [1, 2, 3];
const array = numbers.entries();

for (let x of array) {
  console.log(x);
}
// [0, 1]
// [1, 2]
// [2, 3]
```

30. isArray()

This method returns true if the given value is an array.

```
Array.isArray([1, 2, 3, 4, 5]); // true
Array.isArray(5); // false
```

Conclusion

To make manipulations in the array, we should use these array methods to make our work easier and it helps in writing clean code.

Reference: MDN

Thank you for reading,
Silviya S