←

`map()`

The `map()` method iterates over an array, applying a callback function to each element and returning a new array of modified elements.

```
// Syntax
myArray.map(callbackFn)
```

For example:

```
let scores = [45, 71, 65, 80, 47];

let newScores = scores.map((score) => score + 5);

console.log(newScores); // Output: [50, 76, 70, 85, 52]
```

`forEach()`

The `forEach()` method loops through all array elements and calls a function (callback function) for each element. The callback function has access to the current element, index, and the entire array on every loop.

Type (alt + O)

```
// Syntax
myArray.forEach(callbackFn)
```

For example:

```
let staff = [
    { name: "John Doe", salary: 120 },
    { name: "Jane Doe", salary: 350 },
    { name: "Karl Don", salary: 710 }
];

let totalSalary = 0;

staff.forEach((staffPerson) => {
    totalSalary += staffPerson.salary;
});

console.log(totalSalary); // Output: 1180
```

Note: `forEach()` doesn't execute the function for empty elements.

`reduce()`

The `reduce()` method applies a reducer function to each element of an array and returns a single output. The reducer function iterates through all elements in the array and returns the result of the previous element's calculation.

Type (alt + O)

```
/ Syntax
myArray.reduce(callbackFn, initialValue)
```

For example:

```
let staff = [
    { name: "John Doe", salary: 120 },
    { name: "Jane Doe", salary: 350 },
    { name: "Karl Don", salary: 710 }
];

const totalSalary = staff.reduce((total, staffPerson) => {
    total += staffPerson.salary;
    return total;
}, 0);

console.log(totalSalary); // Output: 1180
```

Note: `reduce()` is a complex array method.

every()[00:00]

The `every()` method executes a function for each element of an array in the form of an evaluation. It returns true if all the array elements pass the evaluation and returns false (and stops execution) if at least one element fails the evaluation. This method uses a callback function as an argument which is evaluated for each element.

Type (alt + O)

```
// Syntax
myArray.every(callbackFn)
```

Suppose you have an array of users' ages, and you want to check if all the users are below 50 years old:

```
let usersAge = [22, 56, 75, 33, 22, 80, 12, 43, 23];

console.log(usersAge.every((age) => age < 50)); // false
```

You can also decide to create the function and then pass it in:

```
let usersAge = [22, 56, 75, 33, 22, 80, 12, 43, 23];

function checkAge(age) {
    return age < 50;
}

console.log(usersAge.every(checkAge)); // false
```

Note: This method does not execute the function for empty elements.

some()

The `some()` method is very similar to the `every()` method, but this time returns true and stops execution if it passes the evaluation for one of the array elements. It will return false if all the elements fail the evaluation.

Type (alt + O)

```
// Syntax
myArray.some(callbackFn)
```

In the callback function, you have access to each element, the index, and the original array itself:

```
// Normal function
myArray.some(function(element, index, array){ /* ... */ })

// Arrow function
myArray.some((element, index, array) => { /* ... */ })
```

In our array of users' ages, suppose you want to check if at least one is above 50 years:

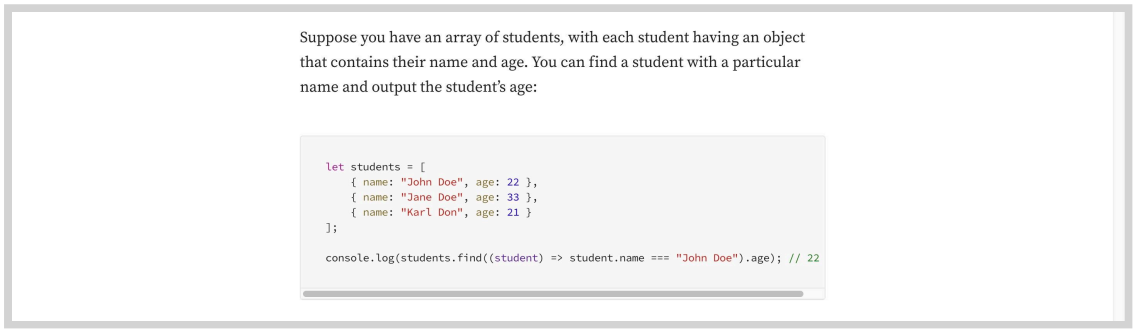
```
let usersAge = [22, 56, 75, 33, 22, 80, 12, 43, 23];

console.log(usersAge.some((age) => age > 50)); // true
```

Note: This method does not execute the function for empty elements.

find()

The `find` method is also similar to the `some()` method but doesn't return a boolean—it will instead return the first element that passes a test. Like the `some()` and `every()` method, it also executes the callback function for every element and will return undefined if no element passes the test. Its syntax is also similar to both methods.

`myArray.find(callbackFn)`



`filter()`

The `filter()` method traverses the entire array, selecting elements that meet a specified condition, and returns them in a new array.

`myArray.filter(callbackFn)`

In the callback function, you have access to each element, its index, and the original array itself. For example:

```
let students = [
    { name: "John Doe", age: 22 },
    { name: "Jane Doe", age: 33 },
    { name: "Karl Don", age: 21 }
];

console.log(students.filter((student) => student.age < 30));
```

This will output:

```
[
    { "name": "John Doe", "age": 22 },
    { "name": "Karl Don", "age": 21 }
]
```

Note: `filter()` doesn't execute the function for empty elements.

concat()

Suppose you have two or more arrays that you want to join together. You can use the `concat()` method. This method joins the arrays or adds an element to an array and then returns a new array containing the joined arrays or new element(s).

```
myArray1.concat(myArray2, myArray3, ..., myArrayX)
// or
array1.concat(element1, ..., elementX)
```

Suppose you have two arrays of European and African country names:

```
let EuroCountries = ["Germany", "Poland", "Sweden"];
let AfricanCountries = ["Ghana", "Nigeria"];
```

You can then combine these arrays with the `concat()` method and assign the new array to a new variable, thereby leaving our original arrays still intact:

```
let countries = EuroCountries.concat(AfricanCountries);

console.log(countries); // ['Germany', 'Poland', 'Sweden', 'Ghana', 'Nigeria']
```

map() T.C -> O(n)

forEach() T.C -> O(n)

reduce() T.C -> O(n)

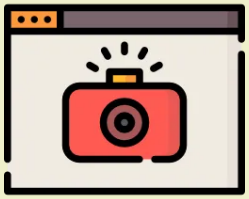
every() T.C -> O(n) (worst-case)

some() T.C -> O(n) (worst-case)

find() T.C -> O(n) (worst-case)

filter() T.C -> O(n)

concat() T.C -> O(n + m) (n, m = lengths of the arrays being joined)



Now you can use Askify in any websites

See How