JS Array Methods 2

Learning Objectives

- [ ] Understanding advanced array methods
  - [ ] `includes`
  - [ ] `find` and `findIndex`
  - [ ] `sort` and `reverse`
    - [ ] know how to use `slice()` to make a copy
  - [ ] `some` and `every`
  - [ ] `reduce`

`includes`

Use `array.includes()` to check whether the array contains the specified value. If it does, `true` is returned, otherwise `false`.

```js
const colors = ["hotpink", "aquamarine", "granite"];

colors.includes("aquamarine"); // true
colors.includes("nemo"); // false
```

`find` and `findIndex`

Use `find()` to receive **the first element** of the array that satisfies the provided testing function. Otherwise, it returns `undefined`.

```js
const colors = ["hotpink", "aquamarine", "granite", "grey"];

colors.find((color) => color.startsWith("g")); // 'granite'
colors.find((color) => color.startsWith("b")); // undefined
```

Use `findIndex()` to receive the index **of the first element** of the array that satisfies the provided testing function. If there is no such element, `-1` is returned.

```js
const colors = ["hotpink", "aquamarine", "granite", "grey"];

colors.findIndex((color) => color.startsWith("g")); // 2
colors.findIndex((color) => color.startsWith("b")); // -1
```

`sort` and `reverse`

Use `sort()` to sort the elements of an array. You need to provide a callback function in order to tell how the array is sorted.

Sorting Numbers

```
const numbers = [4, 42, 23, 1];

numbers.sort((a, b) => a - b); // [1, 4, 23, 42]
numbers.sort((a, b) => b - a); // [42, 23, 4, 1]
```

The sorted order is based on the return value of `a - b` / `b - a`:

Return value of `a - b`	sort order
> 0	sort `a` after `b`
< 0	sort `a` before `b`
=== 0	keep original order of `a` and `b`

> 💡 `sort()` converts the elements into strings, then compares their sequences of UTF-16 Code units values. This is why `array.sort()` without a callback is mostly useless.

Sorting Strings

In order to sort strings, you need to tell the `sort()` method two things inside of the callback function:

- lowercase both strings before comparing them (uppercase works as well)
- using if-statements, be explicit about the return values dependent on the result of the comparison (`nameA < nameB` and `nameA > nameB`)

```
const strings = ["Xbox", "PlayStation", "GameBoy"];

strings.sort((a, b) => {
  const nameA = a.toLowerCase();
  const nameB = b.toLowerCase();
  if (nameA < nameB) {
    return -1;
  }
  if (nameA > nameB) {
    return 1;
  }
  return 0;
});

console.log(strings); // ['GameBoy', 'PlayStation', 'Xbox']
```

> 💡 In UTF-16, the upper- and lowercase version of the same letter do not have the same value. An uppercase 'H' has the UTF-16 decimal value of 72, while the lowercase 'h' has a value of 104.
>
> For example, an uppercase 'W' (87) and a lowercase 'd' (100) are sorted behind the uppercase 'H' (72), but before the lowercase 'h' (104); the result would look like ['H', 'W', 'd', 'h']. This is why it's necessary to upper- or lowercase all letters before sorting them.

`reverse`

In order to reverse an array, simply use `array.reverse()`. This can be combined with `sort()` as well:

```
const numbers = [4, 42, 23, 1];

const reversedNumbers = numbers.reverse(); // [1, 23, 42, 4]
```

`slice`

It's important to note that some array methods, as `sort()`, do not create a new array, but mutate the original one.

```
const numbers = [4, 42, 23, 1];

console.log(numbers); // [4, 42, 23, 1]

const sortedNumbers = numbers.sort((a, b) => a - b);

console.log(sortedNumbers); // [1, 4, 23, 42]
console.log(numbers); // [1, 4, 23, 42]

// What happens if sortedNumbers is reversed?

const reversedSortedNumbers = sortedNumbers.reverse();

console.log(reversedSortedNumbers); // [42, 23, 4, 1]
console.log(sortedNumbers); // [42, 23, 4, 1]
console.log(numbers); // [42, 23, 4, 1]
```

This behaviour will often cause errors. To prevent it, just make a copy of the original array using `slice()`.

```
const numbers = [4, 42, 23, 1];

console.log(numbers); // [4, 42, 23, 1]

const sortedNumbers = numbers.slice().sort((a, b) => a - b);

console.log(sortedNumbers); // [1, 4, 23, 42]
console.log(numbers); // [4, 42, 23, 1]
```

`some` and `every`

Use `some()` to test whether **at least one element** in the array passes the provided test.

```js
const colors = ["hotpink", "aquamarine", "granite"];

colors.some((color) => color.startsWith("g")); // true
colors.some((color) => color.startsWith("i")); // false
```

In order to check if **all elements** pass the test, use `every()`.

```js
const colors = ["hotpink", "aquamarine", "granite"];

colors.every((color) => color.length > 5); // true
colors.every((color) => color.length < 3); // false
```

`reduce`

`Array.reduce()` is an array method to reduce a list of values into a single value.

It has the following core features:

- starting from the beginning, it executes the callback function on each element of the array,
- the return value of each calculation is passed to the next calculation (i.e. it becomes the new starting value for the next iteration through the array)
- the final result is a single value.

It's main use case is to calculate the sum of an array of numbers.

```js
const numbers = [4, 42, 23, 1];

const sum = numbers.reduce((a, b) => a + b);

console.log(sum); // 70
```

> ❗ If you find yourself doing anything more complex than this with reduce (like reducing an array to an object, etc.) you should try to find another solution to your problem. Complex reduce functions are very hard to read and thus error prone.
>
> Example of reducing an array to an object without `reduce()`:

```js
const myArray = [
  { foo: 1, bar: "hi" },
  { foo: 4, bar: "hey" },
  { foo: 2, bar: "ho" },
];
const myObject = {};
myArray.forEach((element) => {
  myObject[element.bar] = element.foo;
});

console.log(myObject); // {hi: 1, hey: 4, ho: 2}
```

Resources

- Searching Arrays (javascript.info)
- sort (javascript.info)
- reduce (javascript.info)