JS Fundamentals Lesson 4

Mohamed Emary

September 7, 2023

1 Loops & Arrays in JavaScript

1.1 Arrays and Array Methods in JavaScript

Here In this lesson we will learn about loops and arrays in JavaScript. We will learn some of the most common array methods and how to use them. We will also learn about the different types of loops in JavaScript and how to use them.

In JavaScript, arrays use numbered indexes. In JavaScript, objects use named indexes. Arrays are a **special kind of objects, with numbered indexes.**

```
let friends = ["mohamed", "ahmed", "emary", ["omar", "ahmed"]];
console.log(friends);
console.table(friends); // appears in console
console.log(Array.isArray(friends));
```

```
friends = [0, 1, 2, 3, 4];
console.log(friends);
console.log(friends.length);
friends[9] = 9;
console.log(friends);
console.log(friends.length);
```

```
friends = [0, 1, 2, 3, 4];
friends.length = 3;
console.log(friends);
```

`shift()` method

Removes the first element from an array and returns it. If the array is empty, undefined is returned and the array is not modified.

`unshift(item)` Inserts new elements at the start of an array, and returns the new length of the array.

`push(item)` Appends new elements to the end of an array, and returns the new length of the array.

`pop()` method

Removes the last element from an array and returns it. If the array is empty, undefined is returned and the array is not modified.

`indexOf(searchElement, startIndex)` The `startIndex` is **Optional**

`indexof` method returns the index of the element if it exists in the array, otherwise it returns -1

`lastIndexOf(searchElement, startIndex)` The `startIndex` is **Optional**

Returns the index of the last occurrence of a specified value in an array, or -1 if it is not present.

`startIndex` The array index at which to begin searching backward. If fromIndex is omitted, the search starts at the last index in the array. `startIndex` takes negative values, counting backwards from the last index in the array. If the calculated index is less than 0, -1 is returned, i.e. the array will not be searched.

`includes(searchElement, startIndex)` The `startIndex` is **Optional**

Determines whether an array includes a certain element, returning true or false as appropriate.

`sort(compareFn)` The `compareFn` is **Optional**

Sorts an array in place. This method mutates the array and returns a reference to the same array. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of `UTF-16` code units values.

`reverse()`

Reverses the elements in an array in place. This method mutates the array and returns a reference to the same array. `reverse` is the opposite of `sort`.

`slice(start, end)` Both `start` & `end` are **Optional**

`slice` Function returns a **copy** (Means that it doesn't affect the original array) of a section of an array. For both `start` and `end`, a negative index can be used to indicate an offset from the `end` of the array. For example, -2 refers to the second to last element of the array.

Since both `start` & `end` are Optional if `start` is omitted it starts from index 0. If `end` is omitted it goes to the end of the array.

Notice that the `end` index is not included so if you want to include it you have to add 1 to it (like python). Documentation Says:

> *`@param end` — The end index of the specified portion of the array. This is exclusive of the element at the index 'end'. If end is undefined, then the slice extends to the end of the array.*

```
let arr = [7, 2, 51, 23, 12, 34, 5];
// remember the last index is not included
console.log(arr.slice(_, 4));
console.log(arr.slice(3));
console.log(arr.slice(-3)); // last 3 elements 12, 34, 5
console.log(arr.slice(2, -2)); // 51, 23, 12
console.log(arr.slice(-4, -2)); // 23, 12
```

`splice(start, deleteCount, item1, item2, ...)` All parameters are **Optional** except `start`

Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.

`@param start` — The zero-based location in the array from which to start removing elements.

`@param deleteCount` — The number of elements to remove.

`@param items` — Elements to insert into the array in place of the deleted elements.

`@returns` — **An array** containing the elements that were deleted.

`concat(...items)` All parameters are **Optional**

Combines two or more arrays. This method **returns a new array** without modifying any existing arrays.

`@param items` — Additional arrays and/or items to add to the end of the array.

`join(separator)` The `separator` is **Optional**

Adds all the elements of an array **into a string**, separated by the specified separator string.

`@param separator` — A string used to separate one element of the array from the next in the resulting string. If omitted, the array elements are separated with a comma.

`flat(depth)` The `depth` is **Optional**

Creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

`@param depth` — The depth level specifying how deep a nested array structure should be flattened. Defaults to 1.

```
let arr = [1, 2, [3, 4, [5, 6]]];
console.log(arr.flat());  // [1, 2, 3, 4, [5, 6]]
console.log(arr.flat(2)); // [1, 2, 3, 4, 5, 6]
console.log(arr.flat(Infinity));  // [1, 2, 3, 4, 5, 6]
```

1.2 Loops in JavaScript

What are the different types of loops in js other than the following:

1. The `for` loop
2. The `while` loop
3. The `do while` loop
4. The `for...in` loop
5. The `for...of` loop

The first loop type is the `for` loop.

Its syntax is as follows:

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

`let i = 0` is the **initialization part**, `i < 10` is the **update part**, and `i++` is the **increment part**. All of these parts are **optional** but if omit any of them you have to put a semicolon **;** in its place.

```
let i = 0;
for (;;) {
  console.log(i);
  i++;
  if (i === 10) break;
}
```

The second loop type is the `while` loop.

Its syntax is as follows:

```
let i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

```
let i = 0;
while (true) {
  console.log(i);
  i++;
  if (i === 10) break;
}
```

The third loop type is the `do while` loop.

Its syntax is as follows:

```
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 10);
```

```
let i = 0;
do {
  console.log(i);
  i++;
  if (i === 10) break;
} while (true);
```

You can control the loop using the `break` and `continue` statements. You can also use loop label to control the loop.

```
outer: for (let i = 0; i < 10; i++) {
  inner: for (let j = 0; j < 10; j++) {
    if (i === 5 && j === 5) break outer;  // break the outer loop (`outer` label)
    console.log(i, j);
  }
}
```

```
outer: for (let i = 0; i < 3; i++) {
  console.log("Outer loop iteration:", i);
  inner: for (let j = 0; j < 3; j++) {
    console.log("Inner loop iteration:", j);
    if (i === 1 && j === 1) continue outer;
  }
}
```

The fourth loop type is the `for...in` loop.

`for in` loop iterates over the enumerable properties of an object, in arbitrary order. For each distinct property, statements can be executed.

Its syntax is as follows:

```
let obj = {
  name: "Mohamed",
  age: 21,
  address: "Egypt",
  friends: ["Ahmed", "Omar", "Emary"],
};
for (let key in obj) {
  console.log(key, obj[key]);
}
```

The fifth loop type is the `for...of` loop.

`for of` loop iterates over iterable objects such as arrays, strings, etc. For each distinct property, statements can be executed.

Its syntax is as follows:

```
let arr = ["Mohamed", "Ahmed", "Emary"];
for (let value of arr) {
  console.log(value);
}
```

The difference between `for...in` and `for...of` is that `for...in` loops iterate over the keys or property names of an object. `for...of` loops iterate over the values of an iterable object like an array.

---

2 Notes

Adding elements with high indexes can create undefined "holes" in an array:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";  // Creates undefined "holes" in fruits
```

---

3 Notes on Exercises

```
  if (!Number.isInteger(min) || !Number.isInteger(max)) return "ERROR";
```

In this one my solution is better but I didn't know about `Math.round()` function.

```
  return Math.round((fahrenheit - 32) * (5 / 9) * 10) / 10;
  return Math.round(((celsius * 9) / 5 + 32) * 10) / 10;


  const C = (F - 32) / 1.8;
  return +C.toFixed(1);

  let F = C * 1.8 + 32;
  return +F.toFixed(1);
```

```
// we have 2 solutions here, an easier one and a more advanced one.
// The easiest way to get an array of the rest of the arguments that are passed to a
↪  function
// is using the rest operator. If this is unfamiliar to you look it up!
const removeFromArray = function (array, ...args) {
  // create a new empty array
  const newArray = [];
  // use forEach to go through the array
  array.forEach((item) => {
    // push every element into the new array
    // UNLESS it is included in the function arguments
    // so we create a new array with every item, except those that should be removed
    if (!args.includes(item)) {
      newArray.push(item);
    }
  });
  // and return that array
  return newArray;
};

// A simpler, but more advanced way to do it is to use the 'filter' function,
// which basically does what we did with the forEach above.

// var removeFromArray = function(array, ...args) {
//   return array.filter(val => !args.includes(val))
// }
//
```