Programming JS

Margit Tennosaar

Sometimes my code is like this......
Don't know, what it does.
But i am scared to delete.

classList object

Every HTML element has a classList object.

It provides methods to add, remove, and toggle CSS classes on an element.

- **element.classList.contains(className)** returns true when the element has the class and false otherwise.
- **element.classList.toggle(className)** will add it when it's not already present and remove it otherwise.
- **element.classList.replace(oldClassName, newClassName)** will replace the oldClassName with the newClassName.
- **element.classList.add()** can be used to add multiple classes at the same time.
- **element.classList.remove()** can be used to remove multiple classes at the same time.

```
const element = document.querySelector('#first-item');

<ul id="shopping-list">
  <li id="first-item>Carrots</li>
  <li>Avocado</li>
</ul>

element.classList.add('highlighted');

<ul id="shopping-list">
  <li id="first-item" class="highlighted">Carrots</li>
  <li>Avocado</li>
</ul>

element.classList.remove('highlighted’);

element.classList.toggle('responsive');

<ul id="shopping-list">
  <li id="first-item" class=”responsive">Carrots</li>
  <li>Avocado</li>
</ul>
```

Notes

You can only access .classList on a single element. If you had a NodeList (from document.querySelectorAll), you need to loop through it with forEach.

To be able to manage classes (in this case, add a class), you have to start by accessing classList. A common mistake is to try and call element.add() or element.addClass(), but these methods do not exist.

All the methods under classList expect a class name. So you should prefix the class name without a dot. element.classList.add(".highlighted)" would not work as expected here.

Modern JS

Modern JS

- JavaScript = EcmaScript
- Vanilla JS
- ECMAScript (ES) is the standard that defines JavaScript
- New versions add better syntax, performance, and features

ES5 (2009)

Known as the "old JavaScript", it's widely supported but lacks many features introduced in later versions.

ES6/ES2015

- Let and const keywords for block-scoped variable declarations.
- Arrow functions for concise function syntax.
- Template literals for easier string interpolation.
- Classes for object-oriented programming.
- Promises for managing asynchronous operations.
- Destructuring Assignment

Some new features

Async/Await: Write asynchronous code in a synchronous-like manner. (ES8)

Rest/Spread Properties: Spread properties for object literals. (ES9)

Optional Chaining (?.): Access deeply nested object properties without having to check for nullability. (ES11)

Destructuring

Destructuring is a feature in JavaScript that allows you to unpack values from arrays or properties from objects into distinct variables. It simplifies the process of accessing values and can make your code more readable and concise. Destructuring is widely used for its readability and the concise syntax it offers, making it easier to work with data structures in JavaScript.

```
const [first, second] = ["apple", "banana", "cherry"];
console.log(first); // "apple"
console.log(second); // "banana"
```

```
const user = { name: "John", age: 30 };
const { name, age } = user;
console.log(name, age); // "John", 30
```

```
const user = { details: { username: "sam_doe", email: "sam@example.com" } };
const { details: { username, email } } = user;
console.log(username, email); // "sam_doe", "sam@example.com"
```

```
const { name, role = "guest" } = { name: "Jane" };
console.log(name, role); // "Jane", "guest"
```

```
const { name: userName, age: userAge } = { name: "Tom", age: 25 };
console.log(userName, userAge); // "Tom", 25
```

Spread and Rest

The spread and rest operators, introduced in ECMAScript 2015 (ES6), are denoted by three dots .... Despite their similar syntax, they serve different purposes depending on the context in which they are used.

The ... operator is used to expand (spread) or collect (rest) values.

Spread

```
// Expands an array into individual arguments.
const numbers = [1, 2, 3];
console.log(Math.max(...numbers)); // Outputs: 3
```

```
// cloning arrays
const original = ['a', 'b', 'c'];
const copy = [...original]; // ['a', 'b', 'c']
```

```
// concatenating arrays
const first = [1, 2];
const second = [3, 4];
const combined = [...first, ...second]; // [1, 2, 3, 4]
```

```
// working with objects
const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 }; // { a: 1, b: 2, c: 3 }
```

Rest

```
// Collects function arguments
function concatenate(separator, ...args) {
  return args.join(separator);
}
console.log(concatenate('.', 'a', 'b', 'c')); // "a.b.c"
```

```
// Helps functions accept an indefinite number of arguments as an array.
function sumAll(...numbers) {
  return numbers.reduce((accumulator, current) => accumulator + current, 0);
}

console.log(sumAll(1, 2, 3)); // Outputs: 6
console.log(sumAll(10, 20, 30, 40, 50)); // Outputs: 150
```

Rest

```
// Allows for extracting multiple elements or properties into a single array or object.

const [first, ...rest] = [10, 20, 30, 40];
console.log(first); // 10
console.log(rest); // [20, 30, 40]


const { x, ...y } = { x: 1, y: 2, z: 3 };
console.log(x); // 1
console.log(y); // { y: 2, z: 3 }
```

Chaining

Chaining is a powerful concept in JavaScript, particularly with arrays, allowing you to combine multiple methods in a single statement. This approach is efficient and concise, enabling you to perform complex operations with minimal code.

```
const users = [
  { id: 1, name: 'Sam Doe' },
  { id: 2, name: 'Alex Blue' },
];

const csv = users.map((user) => user.name).join(', ');
console.log(csv); // "Sam Doe, Alex Blue"
```

```
const users = [
  { name: "Alice", active: true },
  { name: "Bob", active: false }
];
const activeUserNames = users.filter(user => user.active).map(user => user.name);
console.log(activeUserNames); // ["Alice"]
```

Optional chaining

Optional chaining (?.) simplifies accessing nested properties.

It prevents errors by returning undefined if a reference is nullish (null or undefined), rather than throwing an error.

```
const user = {
  details: { name: { firstName: 'Sam' } },
};

console.log(user.details?.name?.firstName); // "Sam”

const data = { temperatures: [-3, 14, 4] };
const firstValue = data.temperatures?.[0]; // -3

const person = { name: 'Sam' };
const upperCasedName = person.name?.toUpperCase(); // "SAM"
```

Safe data handling

```
const customer = { orders: [1001, 1002] };
console.log(customer?.orders?.[0]); // 1001



const user = {};
console.log(user.logout?.()); // No error, just `undefined`
```

Summary

- Destructuring → Extract values from arrays/objects easily
- Spread & Rest Operators → Expand/collect values dynamically
- Method Chaining → Efficient, readable array operations
- Optional Chaining (?.) → Prevents errors with missing data