Following coding standards in JavaScript helps ensure your code is readable, maintainable, and scalable. Here’s a guide on general JavaScript coding standards:

JavaScript Coding Standards

# 1. Use Strict Mode

* Enable strict mode at the beginning of your JavaScript files or functions to catch common coding errors and “unsafe” actions.

javascript

Copy code

'use strict';

# 2. Variable Declarations

* Use let and const instead of var.
  + Use const when the variable will not be reassigned.
  + Use let when you expect the value to change.

javascript

Copy code

const MAX\_COUNT = 10; // Constant variable

let userAge = 25; // Reassignable variable

# 3. Naming Conventions

* Use **camelCase** for variables and function names, **PascalCase** for class names, and **UPPERCASE\_SNAKE\_CASE** for constants.
* Keep names meaningful and descriptive.

javascript

Copy code

let userName = 'John'; // camelCase for variables

function getUserAge() {} // camelCase for functions

class UserAccount {} // PascalCase for classes

const API\_BASE\_URL = 'https://api.example.com'; // UPPERCASE\_SNAKE\_CASE for constants

# 4. Semicolons

* Although JavaScript automatically inserts semicolons, it’s recommended to explicitly use them to avoid unexpected behavior.

javascript

Copy code

let userAge = 25;

console.log(userAge);

# 5. Use Arrow Functions

* Prefer **arrow functions** (=>) for anonymous functions and callbacks, especially when you want to preserve the this context.

javascript

Copy code

const numbers = [1, 2, 3];

const squares = numbers.map(num => num \* num);

# 6. Consistent Spacing and Indentation

* Use **2 or 4 spaces** per indentation level (no tabs). Add a space after keywords like if, for, while, and after commas, but no space before parentheses.

javascript

Copy code

if (condition) {

console.log('Hello');

}

const add = (a, b) => {

return a + b;

};

# 7. Block Scope and Curly Braces

* Always use curly braces {} for block statements, even for single-line if, for, while, etc., to avoid errors.

javascript

Copy code

if (isValid) {

console.log('Valid');

}

for (let i = 0; i < 10; i++) {

console.log(i);

}

# 8. Avoid Global Variables

* Minimize global variables to avoid name conflicts. Encapsulate your code inside functions or modules.

javascript

Copy code

(function() {

let localVariable = 'Scoped to this function';

})();

# 9. Object and Array Literals

* Use shorthand for object properties when the key and value have the same name. Use concise methods for functions within objects.

javascript

Copy code

const userName = 'John';

const user = { userName }; // Shorthand property

const person = {

name: 'John',

greet() { // Concise method

console.log('Hello');

}

};

# 10. Avoid eval()

* Avoid using eval() as it executes code with the privileges of the caller, leading to security vulnerabilities.

javascript

Copy code

// Avoid this

eval("console.log('Hello');");

# 11. Use Template Literals

* Use **template literals** for string concatenation to improve readability.

javascript

Copy code

const userName = 'John';

console.log(`Hello, ${userName}!`); // Use backticks (`) for template literals

# 12. Error Handling

* Use try...catch for error handling and always provide meaningful error messages. Avoid swallowing errors.

javascript

Copy code

try {

// Code that may throw an error

} catch (error) {

console.error('An error occurred:', error.message);

}

# 13. Avoid Callback Hell

* Avoid deeply nested callbacks (callback hell). Use **Promises** or **async/await** to handle asynchronous code more elegantly.

javascript

Copy code

// Promises

fetchData()

.then(response => processData(response))

.catch(error => console.error(error));

// async/await

async function getData() {

try {

const response = await fetchData();

processData(response);

} catch (error) {

console.error(error);

}

}

# 14. Use === and !==

* Use strict equality (===) and strict inequality (!==) to avoid unexpected type coercion.

javascript

Copy code

// Avoid

if (1 == '1') {

// This returns true due to type coercion

}

// Prefer

if (1 === '1') {

// This returns false as it compares both value and type

}

# 15. Avoid Magic Numbers

* Avoid using “magic numbers” (unnamed numerical constants). Assign them to descriptive constants.

javascript

Copy code

// Avoid

if (userAge > 18) {

// Do something

}

// Prefer

const LEGAL\_AGE = 18;

if (userAge > LEGAL\_AGE) {

// Do something

}

# 16. Use Destructuring

* Use **destructuring** for arrays and objects to extract values cleanly.

javascript

Copy code

const user = { name: 'John', age: 30 };

const { name, age } = user; // Destructuring object

const numbers = [1, 2, 3];

const [first, second] = numbers; // Destructuring array

# 17. Use const for Function Expressions

* Declare function expressions using const to prevent them from being overwritten or redeclared.

javascript

Copy code

const greet = function(name) {

console.log(`Hello, ${name}`);

};

# 18. Use Default Parameters

* Provide default values for function parameters when appropriate.

javascript

Copy code

function greet(name = 'Guest') {

console.log(`Hello, ${name}`);

}

# 19. Use ES Modules

* Use **ES6 modules** (import/export) to break down code into reusable, maintainable pieces.

javascript

Copy code

// export.js

export const greet = (name) => console.log(`Hello, ${name}`);

// import.js

import { greet } from './export.js';

# 20. Commenting

* Use comments to explain complex logic or important parts of your code, but avoid over-commenting trivial code.

javascript

Copy code

// Function to add two numbers

function add(a, b) {

return a + b;

}

By following these JavaScript coding standards, you ensure that your code is consistent, readable, and maintainable, making it easier for others (and your future self) to understand and work on your projects.