Coding Standard

1. Formatting

- Use 4 spaces for indenting your code.
- Never mix tabs and spaces.
- No whitespace at the end of line or on blank lines.
- Use UNIX-style newlines (\n).
- Limit your lines to 80 characters.
- Use single quotes for strings (except in JSON).

```
1   var name = 'LAMIN';
```

- Opening braces go on the same line as the statement.

```
3   if (true) {
4       console.log('winning');
5   }
```

- Declare one variable per statement.

```
7   var keys = ['Samia', 'Maliha'];
8   var values = [23, 42];
```

2. Naming Conventions

2.1. Variables:

- Use lowerCamelCase.
- Names must be descriptive, not single letters or unclear short forms.

```
11   var userName = 'Samia';
12   var totalUsers = 100;
```

2.2. Constants:

- Use ALL_UPPERCASE_WITH_UNDERSCORES.
- Define at module level when possible.

```
15   const MAX_USERS = 500;
16   const API_KEY = '12345-ABCDE';
```

2.3. Functions:

- Use lowerCamelCase for function names.
- Name should represent the action.

```
function getUserById(id) {
    return users[id];
}
```

2.4. Classes:

- Use UpperCamelCase for class names.

```
class UserProfile {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}
```

2.5. Files & Modules:

- File names should be lowercase with dashes.
- Keep them short and meaningful.

```
user-controller.js
db-connection.js
```

3. Variables:

- Use trailing commas in arrays and objects.

```
var numbers = [1, 2, 3,];
var user = { name: 'Samia', age: 22, };
```

- Keep short declarations in a single line.
- For non-trivial conditions, assign to a descriptive variable.

```
var isValidUser = (age > 18 && isLoggedIn);
if (isValidUser) {
    console.log('Access granted');
}
```

4. Conditionals:

- Always use triple equality (===) for comparisons.
- Keep conditions simple and clear.

```
48  if (count === 0) {
49      console.log('No items');
50  }
```

5. Comments

- Use // → for single-line comments, /* ... */ → for general multi-line notes and /** ... */ → for documenting functions, parameters, and return values.
- Write comments to explain purpose, logic, or complex code.

```
53  // Verify if the user session is still active
54
55  /*
56     This is a normal multi-line comment.
57     It is usually used inside code to explain logic.
58  */
59
60  /**
61   * Get user details by ID
62   * @param {number} id - User ID
63   * @returns {Object} user data
64   */
```

6. Miscellaneous

- Place all require statements at the top of the file to show dependencies clearly.
- Write require statements in alphabetical order for easy readability.
- Always use const for imported modules instead of var.
- Add documentation for each class and method, including parameters and return values.

```
69  const fs = require('fs');
70  const http = require('http');
```

7. Semicolons

- Always end statements with a semicolon (;) to avoid unexpected errors.

8. Arrays and Function Calls

8.1. Arrays

- Short arrays can be written on a single line.
- Long arrays should have each element on its own line, ending with a comma.
- Always use array literals [] instead of the Array constructor.

```
// Preferred for long arrays/objects
var obj = {
    ready: 9,
    when: 4,
    'you are': 15,
};

// Acceptable for small arrays/objects
var arr = [ 9, 4, 15 ];
```

```
const visitedCities = [];
```

8.2. Function Calls & Spacing

- Always include spaces around elements and arguments for readability.

```
foo( 'string', object );

foo( options, object[ property ] );
```

9. Multi-line Statements

- When a statement is too long to fit on one line, line breaks must occur after an operator.

```
var sum = '<p>The sum of ' + a + ' and ' + b + ' plus ' + c +
    ' is ' + ( a + b + c ) + '</p>';
```

10. Indentation and Line Breaks

- Use tabs for indentation.
- Indent the contents of functions or closures for readability.
- Proper indentation and line breaks make complex code easier to read.

```
function greet(name) {  // top level
    if (name) {          // indented inside function
        console.log('Hello, ' + name);  // further indented
    }
}
```

11. Control Structures

- Add one space between the keyword and ((e.g., if (condition)).
- Always use curly braces {}, even for single-line blocks, for clarity and safety.
- Applies to if, for, while, switch, etc.

```
if (condition1 || condition2) {
    action1();
} else if (condition3 && condition4) {
    action2();
} else {
    defaultAction();
}
```

12. Error Handling

- Always handle errors with try...catch (for async/await) or error-first callbacks.
- Never ignore promise errors.

```
try {
    const data = await getUserData();
} catch (error) {
    console.error('Error fetching user data:', error.message);
    res.end('Internal Server Error');
}
```

13. Security Practices

- Never hardcode sensitive information (passwords, API keys).
- Sanitize user inputs to prevent injection attacks.

14. Testing

- Write unit tests for important functions.
- Ensure code works correctly before merging.

15. Version Control Practices

- Write meaningful commit messages.
- Keep commits small and focused on one task.

Reference:

1. https://developer.wordpress.org/coding-standards/wordpress-coding-standards/javascript/
2. https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Code_style_guide/JavaScript
3. https://www.drupal.org/docs/develop/standards/javascript-coding-standards/javascript-coding-standards
4. https://www.w3schools.com/jS/js_conventions.asp
5. https://github.com/airbnb/javascript