Coding Standard

What is Coding Standard?

A coding standard is a set of rules and guidelines that developers in a project follow when writing code.

---

1. Variables:

Use camelCase for variable and function names. Also, use verbs for function names to clearly define actions.

Example:

```
// Good Practice
let userName = 'JohnDoe';
function getUserData() {
  // function body
}

// Bad Practice
let UserName = 'JohnDoe'; // Starts with uppercase
function get_user_data() {
  // Uses underscores
}
```

2. Constants:

Use UPPERCASE with underscores for constants.

Example:

```
// Good Practice
const MAX_LIMIT = 100;
const API_URL = 'https://api.example.com';

// Bad Practice
const MaxLimit = 100; // Not all uppercase
const apiUrl = 'https://api.example.com'; // Not all uppercase
```

---

3. Classes:

Use PascalCase for class names.

Example:

```
// Good Practice
class UserProfile {
  // class body
}

// Bad Practice
class userProfile {
  // Starts with lowercase
}
```

4. Private Members:

Prefix private variables or methods with an underscore (_).

Example:

```
// Good Practice
class User {
  constructor() {
    this._password = 'secret';
  }

  _validatePassword() {
    // method body
  }
}

// Bad Practice
class User {
  constructor() {
    this.password = 'secret'; // Missing underscore
  }

  validatePassword() {
    // Missing underscore
  }
}
```

---

5. Booleans (camelCase):

Start boolean variables with is, has, or should for clarity.

Example:

```
// Good Practice
let isActive = true;

// Bad Practice
let a = true; // Does not start with is or has, and has no clarity
```

Layout Conventions:

1. Indentation:

Use 2 spaces for indentation.

Example:

```
// Good Practice
function fetchData() {
  if (true) {
    console.log('Indented with 2 spaces');
  }
}

// Bad Practice
function fetchData() {
  if (true) {
      console.log('Incorrect indentation'); // Indented with 4 sp
  }
}
```

---

2. Line Length:

Limit lines to 80-100 characters.

Example:

```
// Good Practice
const userDataGood = fetchUserData(userId, includeAddress, includePreferences);

// Bad Practice
const userDataBad = fetchUserData(userId, includeAddress, includePreferences, additionalParam, anotherParam);
```

3. Semicolons:
Always end statements with semicolons.

Example:

```
// Good Practice
let totalGood = 0;
function calculate() {
  // function body
}

// Bad Practice
let totalBad = 0
function calculate() {
  // Missing semicolons
}
```

---

4. Curly Braces for Loops, Functions, and Conditions:
Place the opening brace on the same line as the loop, function, or condition. Closing braces should be on a new line after the block. Always use braces even for single-line statements.

Example:

```
// Good Practice
for (let i = 0; i < 5; i++) {
  console.log(i);
}

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

if (isValid) {
  proceed();
} else {
  halt();
}

// Bad Practice
function sayHello(name)
{
  console.log(`Hello, ${name}`);
}

// Missing braces
if (isValid) proceed();
```

5. Spaces:
Place spaces around operators and after commas.

Example:

```
// Good Practice
let sumGood = a + b;
function multiply(x, y) {
  return x * y;
}

// Bad Practice
let sumBad=a+b; // No spaces around operators
function multiply(x,y){
  return x*y;
}
```

---

6. Newlines:
Enforce a newline between code blocks and at the end of files.

Example:

```
// Good Practice
function init() {
  // initialization code
}

function start() {
  // start code
}

// Bad Practice
function init() {
  // initialization code
}
function start() {
  // Missing newline between functions
}
```

Comments:

1. Single Line Comments:
Use for short, descriptive notes within the code. Place above the code if the comment applies to the next line.

Example:

```
// Good Practice
// Initialize the app
initApp();

// Bad Practice
//Initialize the app (No space after //)
initApp();
```

---

2. Multi-line Comments:
Use for detailed explanations or documentation of complex logic.

Example:

```
// Good Practice
/*
 * This function processes user input and returns the result.
 * It handles various edge cases and ensures data integrity.
 */
function processInput(input) {
  // function body
}

// Bad Practice
/*
This comment doesn't follow the correct formatting.

Missing asterisk alignment.
*/
function processInput(input) {
  // function body
}
```

References:

https://standardjs.com/rules.html
https://developer.wordpress.org/coding-standards/wordpress-coding-standards/javascript/
https://github.com/rwaldron/idiomatic.js
https://github.com/airbnb/javascript?tab=readme-ov-file
https://www.jscripters.com/jquery-disadvantages-and-advantages/
https://www.franciscomoretti.com/blog/the-pros-and-cons-of-using-eslint#pros-of-using-eslit