Coding Standard

Authors

Rubayed All Islam

Jannati Tajrimin Mitu

Umma Sumaiya Jahan

1. Naming Conventions

1.1 General:

- **Avoid overly general or wordy names.**
  - **Bad Practice:**

    ```
    var data_structure, my_list, info_map;
    ```

  - **Good Practice:**

    ```
    var userProfile, menuOptions, wordDefinitions;
    ```

- **When using camelCase, capitalize all letters of an abbreviation (e.g., HTTPServer).**

1.2 Variables:

- **Use `camelCase` for variable and function names.**
  - **Example:**

    ```
    let examRoll, studentId;
    let isSubmitted = false;
    ```

1.3 Constants:

- **Use `UPPERCASE_WITH_UNDERSCORES` for constants.**
  - **Example:**

    ```
    const MAX_USERS = 100;
    const API_URL = "https://example.com/api";
    ```

1.4 Classes:

- **Follow PascalCase for class names.**
  - **Example:**

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

- Class properties and methods should use camelCase.

1.5 Functions:

- Use `camelCase` for function names.
    - **Example:**

```
function calculateTotalPrice() {
  // Logic here
}
```

1.6 Booleans:

- Prefix boolean variables or methods with
    - **Example:**

```
let isLoggedIn = false;
function isUserActive(userId) {
  // Logic here
}
```

---

2. Code Layout

2.1 Indentation:

- Use 4 spaces for indentation.
    - **Example:**

```
function calculateTotalPrice(price, taxRate) {
    let tax = price * taxRate;
    let total = price + tax;
    return total;
}

if (total > 100) {
    console.log('High price');
} else {
    console.log('Reasonable price');
}
```

2.2 Maximum Line Length:

- Limit lines to 80 characters.
    - **Example:**

```
const totalIncome = baseIncome
    + investmentIncome
    - expenses;
```

2.3 Blank Lines:

- **Use blank lines to separate sections of code for clarity.**
  - **Example:**

```
// Function 1
function fetchData() {
  // logic
}

// Function 2
function processData() {
  // logic
}
```

2.4 Whitespace in Expressions and Statements:

- **Avoid extraneous whitespace.**
  - **Correct:**

```
const result = (x + y) * z;
```

  - **Wrong:**

```
const result = ( x + y ) * z ;
```

2.5 Use of Semicolons:

- **Always use semicolons** to avoid potential pitfalls due to automatic semicolon insertion (ASI).
  - **Example:**

```
let totalPrice = 100;
let taxRate = 0.05;

function calculateTax(price, taxRate) {
    return price * taxRate;
}
```

---

3. Comments and Documentation

3.1 Comments:

- **Use comments to explain complex logic.**
  - **Single-line comments:**

    ```
    // Increment the count by 1
    count += 1;
    ```

- **Multi-line comments:**

  ```
  /*
   * This is a longer comment that explains
   * multiple steps in the code.
   */
  ```

3.2 Documentation Strings:

- **Write JSDoc-style documentation for public functions and classes.**
  - **Example:**

    ```
    /**
     * Calculates the sum of two numbers.
     * @param {number} a - The first number.
     * @param {number} b - The second number.
     * @returns {number} The sum of `a` and `b`.
     */
    function calculateSum(a, b) {
      return a + b;
    }
    ```

---

4. Declarations and Assignments

4.1 Variable Declarations:

- Use `const` **for constants** and `let` **for mutable variables**. Avoid using `var`.
  - **Example:**

    ```
    const taxRate = 0.05; // Constant value
    let totalPrice = 100; // Mutable value
    ```

4.2 Function Declarations:

- **Use function declarations** where hoisting is required, and **arrow functions** for shorter syntax and `this` binding in callbacks.
  - **Example:**

```
// Function declaration
function greet(name) {
    return `Hello, ${name}`;
}

// Arrow function
const add = (a, b) => a + b;
```

---

5. Object and Array Manipulation

5.1 Destructuring:

- **Use destructuring** for more readable code when working with objects and arrays.
  - **Example:**

```
const user = { name: 'Alice', age: 25 };
const { name, age } = user;

const numbers = [1, 2, 3];
const [first, second, third] = numbers;
```

5.2 Spread Operator:

- **Use the spread operator** for copying and merging arrays/objects.
  - **Example:**

```
const userWithAddress = { ...user, address: '123 Main St' };
const numbersCopy = [...numbers];
```

---

6. Member Access and Modifiers

6.1 Public Members:

- Public properties and methods use standard naming conventions.
  - **Example:**

```
class User {
    constructor(name, age) {
        this.name = name; // public
        this.age = age;    // public
    }

    displayInfo() {
        console.log(`Name: ${this.name}, Age: ${this.age}`);
    }
}
```

6.2 Protected Members (Convention):

- **Prefix with `_` to indicate "protected" members.**
    - **Example:**

```
class Car {
    constructor(brand) {
        this._brand = brand; // "protected" (by convention)
    }
}
```

6.3 Private Members:

- **Use `#` to declare private members.**
    - **Example:**

```
class BankAccount {
    #balance = 0; // private

    deposit(amount) {
        this.#balance += amount;
    }
}
```

---

7. Error Handling

7.1 try/catch:

- **Use try/catch for error handling.**
    - **Example:**

```
try {
    let result = riskyOperation();
} catch (error) {
    console.error('Operation failed', error);
}
```

7.2 Promises:

- **Handle promises using `.then()`/`.catch()` or `async/await`.**

- **Example:**

  ```javascript
  async function fetchData() {
      try {
          let response = await fetch('/api/data');
          let data = await response.json();
          console.log(data);
      } catch (error) {
          console.error('Fetching failed', error);
      }
  }
  ```

---

8. Class Member Order

Member Order:

1. Static properties
2. Static methods
3. Instance properties
4. Constructor
5. Public instance methods
6. Private/Protected methods

Example:

```javascript
class User {
    // 1. Static properties
    static MIN_AGE = 18;

    // 2. Static methods
    static isValidAge(age) {
        return age >= User.MIN_AGE;
    }

    // 3. Instance properties
    _balance; // protected (convention)
    #password; // private

    // 4. Constructor
    constructor(name, age, password) {
        this.name = name;
        this.age = age;
        this.#password = password;
```

```
        this._balance = 0;
    }

    // 5. Public methods
    deposit(amount) {
        this._balance += amount;
    }

    // 6. Private/Protected methods
    #resetPassword(newPassword) {
        this.#password = newPassword;
    }
}
```

9. Asynchronous Patterns

9.1 Handling Multiple Promises:

- Use `Promise.all()` to handle multiple asynchronous operations in parallel.
  - **Example:**

```
const fetchDataFromMultipleSources = async () => {
  try {
    const [data1, data2] = await Promise.all([fetch(url1),
fetch(url2)]);
    const result1 = await data1.json();
    const result2 = await data2.json();
    console.log(result1, result2);
  } catch (error) {
    console.error('Error fetching data', error);
  }
};
```

9.2 Using `Promise.race()`:

- Use `Promise.race()` when you need only the fastest promise to resolve or reject.
  - **Example:**

```
Promise.race([promise1, promise2])
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
```

---

10. Best Practices for Performance

10.1 Optimize Loops:

- **Avoid unnecessary computations inside loops.**
  - **Example:**

```
// Inefficient
for (let i = 0; i < array.length; i++) {
    if (array[i] % 2 === 0) {
        console.log(array[i]);
    }
}

// Efficient
const isEven = num => num % 2 === 0;
for (let item of array) {
    if (isEven(item)) {
        console.log(item);
    }
}
```

10.2 Minimize DOM Manipulations:

- **Batch DOM updates to reduce reflows and repaints.**
  - **Example:**

```
// Inefficient
for (let item of items) {
        document.body.appendChild(createElement(item));
}

// Efficient
const fragment = document.createDocumentFragment();
for (let item of items) {
        fragment.appendChild(createElement(item));
}
document.body.appendChild(fragment);
```

10.3 Minimize DOM Access:

- **Batch DOM reads and writes** to improve performance.
  - **Example:**

```
// Bad:
element.style.width = '100px';
element.style.height = '200px';

// Good:
element.style.cssText = 'width: 100px; height: 200px;';
```

10.4 Event Delegation:

- **Use event delegation to manage event listeners efficiently**, especially for dynamically added elements.
  - **Example:**

```
document.querySelector('#parent').addEventListener('click', (event) =>
{
  if (event.target.matches('.child')) {
    // Handle click on child
  }
});
```

10.5 Throttling and Debouncing:

- **Use throttling or debouncing for performance-critical functions** such as scroll or resize events.
  - **Example using lodash:**

```
window.addEventListener('scroll', _.throttle(() => {
  console.log('Throttled scroll event');
}, 200));

const searchInput = document.getElementById('search');
searchInput.addEventListener('input', _.debounce(() => {
  console.log('Debounced search input');
}, 300));
```

11. Security Best Practices

11.1 Avoid `eval()`:

- **Never use `eval()`** as it can make your code vulnerable to injection attacks.

- **Example:**

```
// Avoid:
eval("var a = 10");

// Safe alternative:
let a = 10;
```

11.2 Escape User Input:

- **Always sanitize and escape user input** to prevent cross-site scripting (XSS) attacks.
  - **Example:**

```
const safeString = userInput.replace(/[<>&'"]/g, function (char) {
  return ({
    '<': '&lt;',
    '>': '&gt;',
    '&': '&amp;',
    "'": '&#39;',
    '"': '&quot;'
  }[char]);
});
```

---

12. Modern ES6+ Features

12.1 Template Literals:

- **Use template literals for building strings dynamically.**
  - **Example:**

```
const name = 'Alice';
const message = `Hello, ${name}!`;
console.log(message); // Output: Hello, Alice!
```

12.2 Default Parameters:

- **Use default parameters** to assign default values to function arguments.
  - **Example:**

```
function greet(name = 'Guest') {
  return `Hello, ${name}`;
}

console.log(greet()); // Output: Hello, Guest
```

12.3 Rest and Spread Operators:

- **Use rest and spread operators** for cleaner array and object manipulation.
  - **Example (Rest):**

    ```
    function sum(...numbers) {
      return numbers.reduce((acc, num) => acc + num, 0);
    }

    console.log(sum(1, 2, 3)); // Output: 6
    ```

  - **Example (Spread):**

    ```
    const arr1 = [1, 2, 3];
    const arr2 = [...arr1, 4, 5];

    const obj1 = { name: 'Alice', age: 25 };
    const obj2 = { ...obj1, job: 'Developer' };
    ```

---

13. References

1. **Airbnb JavaScript Style Guide**
   A widely adopted and comprehensive JavaScript style guide, known for its practical recommendations and best practices.
2. **Google JavaScript Style Guide**
   Google's official JavaScript style guide, focusing on clarity, consistency, and simplicity in JavaScript code.
3. **MDN Web Docs: JavaScript Guide**
   Comprehensive and up-to-date documentation on JavaScript, maintained by Mozilla.
4. **ECMAScript 2022 (ES13)**
   The latest edition of the ECMAScript language specification, which standardizes JavaScript.
5. **JSDoc Documentation**
   Official documentation for JSDoc, a popular tool for generating documentation from JavaScript comments.
6. **Node.js Best Practices**
   A community-driven guide to best practices for Node.js development, including error handling, performance, and code structure.
7. **JavaScript Info**
   A modern tutorial on JavaScript, covering both fundamental and advanced topics in depth.
8. **Prettier**
   A code formatting tool that enforces a consistent style across JavaScript codebases.