Coding Standards

HTML

1. Indentation and Formatting:

- Use a consistent indentation style, such as 2 or 4 spaces per level.

- Use lowercase for all HTML tags and attribute names.

- Use double quotes for attribute values, e.g., `<img src="image.jpg" alt="Description">`.

2. Document Structure:

- Start with a `<!DOCTYPE html>` declaration.

- Use `<html>` which will be the root element, `<head>`, and `<body>` elements to structure your document.

- Include a `<meta charset="UTF-8">` tag for character encoding.

- Include a `<title>` tag within the `<head>` section.

3. Comments:

- Use comments (`<!—Comment here →`) to explain complex sections or to provide context within your code.

4. HTML5 Semantic Elements:

- Utilize HTML5 semantic elements like `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>`, and `<footer>` to improve code readability and SEO.

5. Proper Nesting:

- Ensure that HTML tags are properly nested and closed.

6. Avoid Inline Styles:

- Separate CSS from HTML by using external CSS files or inline `<style>` blocks within the `<head>` section.

7. CSS Classes and Ids:

- Use meaningful class and ID names that describe the purpose of the element, rather than generic names so that it can be easily understandable by other programmers to understand the purpose of the element.

8. Consistent Naming Conventions:

- Maintain a consistent naming convention for classes and Ids (e.g., camelCase, kebab-case, or underscore_case).

9. Accessibility:

- Make sure that your HTML is accessible by using appropriate `alt` attributes for images and providing meaningful text for links and buttons.
- Make sure to use proper heading structures and semantic elements so that it can be used for screen readers properly.

10. Validation:

- Regularly validate your HTML code using tools like the W3C Markup Validation Service.

11. White Space:

- Use white space judiciously to enhance code readability and don't use white space if it is not very necessary.

12. Avoid Deprecated Elements:

- Ensure to avoid using deprecated HTML elements and attributes.

13. Responsive Design:

- Design web pages to be responsive by using media queries and flexible layouts.

CSS

1. Property and Value Formatting:

- Use lowercase for all property names and values (e.g., `font-size`, `color`, `margin`, `auto`).
- Use a single space after the colon.
- Use a semicolon to separate multiple declarations on the same selector.
- Separate property-value pairs with a colon and space (e.g., `margin: 10px;`).
- Use indentation for better readability, typically with two or four spaces.

Example:

```css
/* Good naming convention using BEM */
.header {}
.header__logo {}
.header__nav {}
.header__nav—active {}

/* Avoid overly generic names */
.button {}
```

2. Specificity:

- Avoid using overly specific selectors when possible. Prefer using class and ID selectors over tag selectors.
- Be cautious with the use of `!important`, as it can lead to debugging issues.
- Avoid inline styles in HTML whenever possible.

Example:

```css
/* Prefer class and ID selectors over tag selectors */
.header {}
.header h1 {}

/* Avoid using !important */
.error-message {
   Color: red !important;
}
```

3. Formatting:

- Use 4 spaces for indentation (no tabs).
- Use lowercase for all property names and values (e.g., `font-size`, `color`).
- Separate property-value pairs with a colon and a space (e.g., `margin: 15px;`).
- End each declaration with a semicolon.
- Use one declaration per line for clarity.

Example:

```css
.selector {
   Property1: value1;
   Property2: value2;
}
```

4. Selectors:

- Use meaningful, descriptive class and ID names.
- Avoid overly generic names (e.g., `div` or `span`) unless necessary.
- Prefer class selectors (`.classname`) over ID selectors (`#element`) for reusability.
- Limit the use of complex selectors for performance reasons.

Example:

```css
.button {
  /* ... */
}

#header {
  /* ... */
}
```

5. Comments:

- Use comments to explain complex or non-obvious styling choices.
- Comment your code generously for clarity.
- Use `/* ... */` for comments.

Example:

```css
/* Prefer class selectors over tag selectors */
.button {
  /* ... */
}
```

6. Modularity:

- Organize CSS into separate files for components or functionality.
- Follow a naming convention like BEM (Block Element Modifier) for complex components.
- Keep related styles together within a component.

Example:

```css
/* BEM naming convention */
.header {}
.header__logo {}
.header__nav {}
```

7. Vendor Prefixes:

- Use autoprefixing tools to add vendor prefixes automatically for cross-browser compatibility.
- Minimize manual vendor prefixing.

8. File Organization:

- Keep CSS files organized, possibly with a consistent folder structure.
- Consider using CSS preprocessors like Sass or LESS for improved organization.

Java Script

1. Indentation:
    - Use consistent indentation (usually 2 or 4 spaces) to make your code more readable.
    - Use another separate line for each statement or declaration.

2. Use Semicolons:

   End statements with semicolons to avoid unexpected behavior due to automatic semicolon insertion.

3. Use Single Quotes or Double Quotes Consistently:

   Choose either single quotes or double quotes for string literals and stick with your choice consistently.

4. Variable and Function Naming:

   Use meaningful, descriptive names for variables and functions. Follow naming conventions like camelCase for variables and functions.

5. Constants:

Use uppercase letters and underscores for constant values. For example: const MAX_COUNT = 10;

6. Declare Variables Properly:

Use let or const to declare variables, and avoid using var.

7. Avoid Global Variables:

Minimize the use of global variables to prevent unintended side effects.

8. Arrow Functions:

Use arrow functions (() => {}) when creating simple, anonymous functions.

9. Comments:

Use comments to explain complex logic, algorithms, and code blocks. Follow a consistent commenting style (e.g., JSDoc).

10. Use === and !==:

Prefer strict equality (===) and inequality (!==) operators over loose equality (==) and inequality (!=) operators.

11. Spacing:

Use consistent spacing around operators and after commas for improved readability.

12. Curly Braces:

Use curly braces for all control structures (if statements, loops, functions), even for single-line blocks.

13. Avoid Eval:

Avoid using eval() as it can introduce security vulnerabilities and make code harder to understand.

14. Avoid “With” statement:

Avoid using the with statement, as it can lead to ambiguity and bugs.

15. Error Handling:

Implement proper exception-handling tools like try, catch, and throw for unexpected exceptions.

16. Modularization:

Organize the code into modules and you may use tools like CommonJS for better code structure.

17. Consistent Formatting:

Enforce consistent code formatting across your project using tools like ESLint or Prettier.

18. Use ES6+ Features:

You can use ES6 and later features like destructuring, spread/rest operators, and template literals for variable declarations and template literals.

19. Use Promises or Async/Awai:

Use Promises or Async/Await for asynchronous operations instead of callbacks when applicable.