JAVA SCRIPT CODING STANDARD RULE DOCUMENT

INTRODUCTION

Coding standards for JavaScript aim to ensure that scripts are consistent, maintainable, and easily readable. They promote best practices and facilitate collaboration among developers. These standards cover areas such as variable naming, code formatting, error handling, and more. Adhering to JavaScript coding standards helps produce high-quality code that is efficient and less prone to bugs.

Table of Contents

1. Variable Names

- Use camel-case for variables and function names:
  - Camel-Case is a naming convention where each word in a name begins with a capital letter except for the first word, which starts with a lowercase letter. For example, `myVariableName`.

- All names should start with a letter:
  - Variable and function names must begin with a letter (a-z, A-Z).

Example:

```
let firstName = "John";
let lastName = "Doe";
function calculateTax() { /* Function in camelCase */ }
```

2. Spaces Around Operators

- Always put spaces around operators (=, +, -, *, /) and after commas:
  - Adding spaces around operators and after commas enhances code readability.

Example:

```
let result = a + b; // Good practice
const myArray = ["Volvo", "Saab", "Fiat"]; // Good practice
```

3. Code Indentation

- Use 2 spaces for code indentation:
  - Consistent indentation makes code more visually appealing and easier to read.

- Avoid using tabs for indentation:
  - Different text editors interpret tabs differently, which can lead to inconsistent formatting.

Example:

```
function toCelsius(fahrenheit) {
  return (5 / 9) * (fahrenheit - 32);
}
```

4. Statement Rules

- End simple statements with a semicolon:
  - Semicolons are used to terminate statements, ensuring proper code execution.

- Place the opening bracket at the end of the first line for complex statements:
  - For functions, loops, conditionals, and objects, place the opening bracket at the end of the first line.

- Do not end complex statements with a semicolon:
  - Complex statements like functions and conditionals do not require a semicolon at the end.

Example:

```
const cars = ["Volvo", "Saab", "Fiat"]; // Semicolon
const person = {
  firstName: "John", // No semicolon
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

5. Object Rules:

- Place the opening bracket on the same line as the object name:
  - This is a common formatting style for object definitions.

- Use a colon plus one space between each property and its value:
  - Colon-space format makes it clear that properties are associated with values.

- Use quotes around string values, not around numeric values:
  - String values should be enclosed in quotes, while numeric values should not.

- Do not add a comma after the last property-value pair:
  - The trailing comma is not necessary and can lead to errors in some JavaScript engines.

- Place the closing bracket on a new line, without leading spaces:
  - This formatting enhances readability.

- Always end an object definition with a semicolon:
  - Semicolons are used to terminate statements.

Example:

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

6. Line Length < 80:

- Avoid lines longer than 80 characters:
  - Long lines of code can be challenging to read and maintain. Limit line lengths for readability.

- Break lines after an operator or a comma when necessary for readability:
  - When a line exceeds the recommended length, break it after an operator or comma for better readability.

Example:

```
document.getElementById("demo").innerHTML =
  "Hello Dolly.";
```

7. Naming Conventions:

- Use consistent naming conventions throughout your codebase:
  - Consistency in naming conventions helps maintain a clean and organized codebase.

- Follow camelCase for variable and function names:
  - CamelCase enhances readability by making it clear where word boundaries are.

- Consider using UPPERCASE for global variables and constants:
  - UPPERCASE identifiers are often used for constants and global variables to distinguish them from local variables.

8. File Extensions:

- Use appropriate file extensions for your JavaScript files (e.g., .js):
  - Proper file extensions help identify the type of content within a file.

9. Use Lower Case File Names:

- Use lowercase file names to avoid case sensitivity issues:
  - Lowercase file names help prevent conflicts between case-sensitive and case-insensitive file systems.

10. Function Declarations

- Use function declarations for named functions.
- Place function declarations at the top of their containing scope.

11. Function Expressions

- Use function expressions for anonymous functions or when functions are assigned to variables.

12. Function Parameters

- Keep the number of function parameters small, preferably under three.
- Use default parameter values when applicable.