TABLE OF CONTENTS

1 Introduction 2
2 Source File Basics 2
    2.1 File Names
3 Source File Implementation 2
    3.1 Imports
4 Formatting 2
    4.1 Braces
    4.2 Block Indentations
    4.3 Statements
    4.4 Column Limit
    4.5 Line Wrapping
    4.6 Whitespace
    4.7 Comments
5 Language Features 3
    5.1 Local Variable Declarations
    5.2 Array Literals
    5.3 Object Literals
    5.4 Functions
    5.5 String Literals
6 Naming 4
    6.1 Rules Common for All Identifiers
    6.2 Rules by Identifier Types
7 Commit Convention 4
    7.1 Commit Message
    7.2 When to Commit

NOTE: Based on usage of Prettier and ESLint

1 Introduction

This is the document that records all of our coding conventions for the Team 4 Project in CSE 210. We will automate our coding practices with Prettier and ESLint. Any rule that is not described in this document is up to the discretion of the developer.

2 Source File Basics

2.1 File Names

All file names must be all lowercase and have dashes “-” but no punctuation. Javascript files must have extension “.js”, HTML files must have extensions “.html”, and CSS files must have extension “.css”.

3 Source File Implementation

3.1 Imports

Imports are placed at the very top of the respective file and cannot be line wrapped. Additionally, it must include all file extensions (ex: “./foo.js”, not “./foo”).

4 Formatting

4.1 Braces

In JS files, braces should be used for all control structures (e.g. if, else, for, do, while) even if the body only has one statement.

4.2 Block Indentations

Every time when opening a new block, the indentation increases by 1 tab space (VSCode). This indentation style should apply to both implementation and comment lines. This should be automated with Prettier.

4.3 Statements

Only one statement per line followed by a line break. Every statement should be terminated by semicolon (;).

4.4 Column Limit

The column limit should be 80 characters. Any overflow should be line-wrapped. This should be automated with Prettier.

4.5 Line Wrapping

How each line is wrapped is up to the developer's discretion.

4.6 Whitespace

Vertical whitespace should be placed to group corresponding statements within a code body.
Vertical whitespace should be placed between each function within the file.

Horizontal whitespace:
No whitespace should be placed between words and parenthesis.

4.7 Comments

Comments should either be in the form // or /* … */. If in the latter form, any subsequent lines of the comment should start with * and be on the same indentation as the previous line.

** Do not use JSDoc for implementation comments. It should be documentation only. **

5 Language Features

5.1 Local Variable Declarations

Should only be using “const” and “let” for variable declarations *NOT* “var”. Additionally, only one variable declaration per line. Variables should be initialized as close to their declarations as possible.

5.2 Array Literals

Do not use the “Array” constructor. Use the literal “[..]” instead.

5.3 Object Literals

Do not use the Object constructor. Use the literal “{...}” instead.
Keys should either be all quoted or all unquoted. Do not mix them.

5.4 Functions

All function parameters and return types should be documented with JSDocs. We have an extension that generates documentation for us.

5.5 String Literals

All string literals within JS and CSS should be single quotes (‘). HTML should only have double quotes around strings (“).

6 Naming

6.1 Rules Common for All Identifiers

All Identifiers should be named with only ASCII characters. Names should be as descriptive as possible, and concise where necessary.

Only time single character variables are allowed is if the scope of the variable is small.

6.2 Rules by Identifier Types

Package, Method, Parameters, and Local Variable names should be written in lower camel case.

Class names should be written in upper camel case.

Global constant variables should be written in all caps with snake case

7 Commit Convention

7.1 Commit Message

Mainly follow: https://www.conventionalcommits.org/en/v1.0.0/
Subject line should be in all lower case. Subject line cannot end with a period. At the very beginning of the subject line, include the type of commit it is:

1) fix: patches a bug
2) feat: adds a feature
3) test, docs, style, perf: up to dev's discretion

Whenever each commit adds a breaking change to any previous commits, must use the 'BREAKING FEATURE' footer or include '!' after the commit type and before the ':'
e.g. feat!: there is a breaking change here

The subject line following the commit type must be in the active voice.
e.g. feat: adding feature 1 (INCORRECT)
feat: add feature 1 (CORRECT)

Further styling of the commit body and footer (outside of BREAKING CHANGE) is up to the discretion of the developer. The commit body should be detailed enough to provide a descriptive explanation for changes, but not too detailed to be verbose.

7.2 When to Commit

Commit whenever there is a notable change in the code. Commit often.