JavaScript Style Guide

Introduction

The objectives of employing this Style Guide are:

1. To encourage clean, consistent, and conscientious coding techniques
2. To introduce students to the concept of a coding Style Guide, which they may have to use in industry

Clean, Consistent, and Conscientious Coding

While this class focuses primarily on using the JavaScript language, a specific course outcome is for students to, "write clean, consistent code."

At some point, you will be working with other coders, or you will inherit someone else's code. Reading other people's code can be challenging from the simple fact that styles of thinking and mindsets are different. Inconsistent, sloppy coding techniques and lack of comments can only further compound this issue.

Style Guides

Like in an English composition class, some organizations and open source projects employ the use of style guides. These style guides are used for a number of reasons, including outlining requirements for writing secure code, and outline requirements for writing consistent code, among others.

This course presents a simple style guide, intended to set a basic set of requirements you will be expected to adhere to when writing code for this class. This style guide is based off the Google JavaScript Style Guide, so the requirements, while simple, are not arbitrary: they come from a real world entity.

Attribution

This guide has been adapted from the Google JavaScript Style Guide (https://google.github.io/styleguide/jsguide.html). This guide is used and modified under the Creative Commons 3.0 license (https://creativecommons.org/licenses/by/3.0/). Additional Google Style Guides are available at https://google.github.io/styleguide/

1. Comments

1.1 Comment Styles

1.1.1 Block comments and single line comments may be used as needed and to the student's preference.

1.2 Opening Comment

1.2.1 Each JavaScript file will include the Standard Opening Comment.

The **Standard Opening Comment** is a *multiline* comment, placed at the very top of each JavaScript file. The contents of the **Standard Opening Comment** will inclde:

```
/*
    Student Name
    File Name
    Course Number
    Instructor Last Name
    File Creation Date
*/
```

The **Standard Opening Comment** must contain information relevant to the specific JavaScript file in which it is contained (e.g., if the JavaScript file is named sample.js, the *File Name* should be `sample.js`).

Here is an example

```
/*
    Billy Bob Jones
    sample.js
    INFO 2124
    Brown
    10/10/2021
*/
```

1.3 Commenting Code

In addition to the **Standard Opening Comment** the following is the minimum expectation for commenting code:

1.3.1 Method and function comments

In methods and named functions, parameters and returntypes must be documented.

The following information should be included:

```
/*
    Function:       Function Name
    Description:    Description of what function does
    Parameters:
                    param1 {type} parameter description
                    additional params...
    Returns:        {type} - description of what this returns or "none" if
                    returns nothing.

*/

function mySampleFunction(param) {
        //body
}
```

The body of the function must contain comments explaining what the body of the function is doing.

For example:

```
/*
    Function:       capitalizeEachWord
    Description:    This function capitalizes the first character in each word
                    in the passed string.
    Parameters:
                    word {string} This is the string we will work with - it can be a
                    single word or an entire sentence.
    Returns:        {string} - the processed string value.

*/
function capitalizeEachWord(word) {
        //split each of our words using a space as the split delimiter
        //then store the words in an array called 'words'
        const words = word.split(" ");

        //loop through each element of the array . . .
        for(let i=0; i < words.length; i++) {
            //transform the first character of each word to its uppercase
            //value, then add that to the remainder of the string
            //and store that back in the original element...
            words[i] = words[i][0].toUpperCase() + words[i].substr(1);
        }
        //put everything back together and return it
        return words.join(" ");
}
```

2. Formatting

2.1 Braces

2.1.1 Braces are used for all control structures

Braces are required for all control structures (i.e., `if`, `else`, `for`, `do`, `while`, as well as any others), even if the body contains a single statement. The first statement of a non-empty block must begin on its own line.

Incorrect:

```
if(someVeryLongCondition)
    doSomething();

for(let i = 0; i < foo.length; i++) bar(food[i]);
```

Correct:

```
if(someVeryLongCondition) {
    doSomething();
}

for(let i = 0; i < foo.length; i++) bar(food[i]);
```

Braces may be cuddled (on the same line as the condition) or placed on a new line as the student prefers; however, bracing should be **consistent**.

2.2 General Organization

2.2.1 Code shall be cleanly organized and consistent

Code should be organized in a consistent fashion with consistent placement of bracing and consistent indentation. The instructor reserves the right to dock points for messy code.

3. Statements

3.1.1 On statement per line

Each statement is followed by a single line-break (return/enter).

3.1.2 Semicolons are required

Every statement must be terminated with a semicolon. Relying on automatic semicolon insertion is forbidden.

4. Language Features

4.1 Local variable declarations

4.1.1 Use `const` and `let`

Declare all variables with either `const` or `let`. Use `const` by default unless a variable needs to be reassigned. The `var` keyword must not be used.

3. Naming

3.1 Rules common to all identifiers

Identifiers use only ASCII letters and digits.

- You may use all `lowercase` characters for identifiers.
- You may use `camelCase` for identifiers (where the word begins with a lowercase and each subsequent word begins with a capital)
- Do **NOT** use the underscore _ character to separate words (e.g, `my_variable_name`)
- Do **NOT** use all uppercase characters for variable names.
    - Identifiers in `UPPERCASE` are traditionally seen as constants. You are not required to specify constants in full uppercase; however, you are to avoid naming variables fully in uppercase.

Give as descriptive a name as possible, within reason. Do not worry about saving horizontal space as it is far more important to make your code immediately understandable by a new reader. Do not use abbreviations that are ambiguous or unfamiliar to readers outside your project, and do not abbreviate by deleting letters within a word.

Whether you you use all `lowercase` or opt to use `camelCase` notation for identifiers, your style should be consistent throughout your JavaScript.

Correct

```
errorCount          // No abbreviation.
dnsConnectionIndex  // Most people know what "DNS" stands for.
referrerUrl         // Ditto for "URL".
customerId          // "Id" is both ubiquitous and unlikely to be misunderstood.
```

Incorrect

```
n                   // Meaningless.
nErr                // Ambiguous abbreviation.
nCompConns          // Ambiguous abbreviation.
wgcConnections      // Only your group knows what this stands for.
pcReader            // Lots of things can be abbreviated "pc".
cstmrId             // Deletes internal letters.
kSecondsPerDay      // Do not use Hungarian notation.
```