JavaScript Program Style Rules

(Last update 5/5/17)

The following style rules are mandatory for all programs submitted. Breaking a style rule will cause your program to bounce, and you will have to redo the code according to correct style. Because in-class examples were written according to earlier versions of this style sheet, some of them may differ from the rules here. Be sure your submitted code follows this style sheet when theres a difference.

1. Identifier Names

A. Use meaningful variable names. X, jj, and bobo don't mean as much as timeLeft, biggest, and whereNext.

B. Avoid single letter variable names, except as loop counters in for-loops when there's no reasonable alternative.

C. If you decide to abbreviate a commonly used term in variable or type names, use exactly the same abbreviation everywhere. (For instance, if the term "category" appeared often in variable or type names, you might abbreviate it *everywhere* to "ctg".)

D. In multiple-word names, use camelCase, with initial lowercase for local variables, and simple globals. Use initial caps for globals that are "packages" containing other subobjects and constructors and for constructor functions themselves.

2. Comments (for assignments requiring commenting – check first)

A. Write comments for each method that is not baldly obvious (such as a simple accessor), including a description of what the function does, and what each parameter is for. If the function makes any special assumptions about the contents of the parameters, list these. Write comments for each class in the program, including descriptions of each member datum. Don't comment the obvious; concentrate on explaining the hard parts. Use JavaDoc notation for header comments.

B. Comments must be **clear**. A set of words in comment markers is not automatically a comment. Comments must be grammatical and correctly spelled.

C. Put comments that exceed one or two lines in function headers, not in the code itself. You may mark a line with a number thus: // 1 , and discuss it in the function header.

3. Function Design

A. Avoid parameter lists of more than 5. Group data into objects instead. For instance:

void printStats(int total, int count, int mean, int median, float stddev)

These parameters are related; they comprise a set of statistics. Instead of passing 5 items, group them into a single object parameter that can be passed JSON-style, with labels on each parameter.

B. Functions must be at most 50 lines long, not counting blank lines, asserts, comments, or lines with only a single brace on them. Don't cram code together to satisfy this limit. Break up your functions instead.

C. Place all local variable declarations at the top of the function, with a single blank line separating the locals from the code that follows.

4. Indentation, Line Length, Parens and Braces

A. Use 3 space indentation.

B. Indent only one level for each nested if, while, or switch:

```
while (...) {

 statement;

 if (...) {

  statement;

 }

}
```

C. Never let a line exceed 80 columns. If a line must be broken into two, indent the second part one space past the first column of the first part: Further continuation lines, if needed, stay at that same 1-space indent. This rule applies even in modern displays because it permits multiple parallel pages of code on a single screen without the need for horizontal scrolling.

```
Big long line....

 with continuation line below

 and further continuation
```

D. Indent both the then and else blocks of an if-statement, even if they are only one line long. Don't write: if (test) statement; However, it is acceptable to use conditional-and or conditional-or tests, e.g. <if this is true> && <then do this>

E. Use blank lines to break up blocks of code. Your code should fall into groups of about 5-8 lines on average, separated by a blank line or a line with only a brace, and possibly started with a one or two line comment. Always put a blank line after the local declarations in a function. Don't put more than one blank line in a row. A single blank line at a time is enough to break up the flow of code properly.

F. Place the opening brace at the end of the if, while, class, etc. line that it applies to, and place the closing brace on a line by itself, or with other closing punctuation only (e.g. }); or },) in the same column as the beginning of the opening line.

I. Branches of an else-if sequence must be indented at the same level, even though technically each else-if should be indented one level deeper than the preceding one.

J. Do not put more than one statement on a line.

K. If you call an anonymous function directly, parenthesize the anonymous function declaration, e.g. (function() {}) ()

L. Callback functions at the end of a paramter list may start on the line after the initiating call, without indentation, as though the initiating call was a "prefix header" to the function. This avoids excessive nesting without harming readability:

```
fNeedsCallback(prm1, prm2,
function(x) {

}
```

5. Spacing

A. Use spaces to clarify your program and to break up long expressions, but don't overdo it. Put spaces after each comma, and around each keyword (note that "if", "while" and "for" are keywords). Put a space after each semicolon in a for header and before any opening brace '{'. Never put a space *before* a comma or semicolon.

B. Put spaces around operators, except in very large expressions where you may avoid them around the highest-priority operators. Dont have more than three variables or operators in a row without a blank space.

```
Good: epsilon = 2*beta + gamma - delta*pi;

Bad: epsilon=2*gamma-delta*pi;
```

C. *Don't* put space after an opening paren, or before a closing one. Do not put space between a function name and the opening paren for the parameter list.

6. Constants

Use constant variables in your code instead of direct numbers. There must be no actual numbers other than 0, 1 or 2 in your code, unless an exception has been granted by your instructor. For web development, the status codes 200, 400, 401, etc are an exception since they are well known and never change.

7. Code Elegance

A. Do not use the clauses "== true" or "== false" in code. Use boolean variables as direct tests, as in "if (mEmpty)", not in equality checks like "if (mEmpty == true)".

B. Do not write an if-else that can be replaced with a simple boolean expression. For instance:

```
if (val1 < va2 && mark == 0)

  oddTest = true;              VS:    oddTest = val1 < val2 && mark == 0;

else

  oddTest = false;
```