Documentation and Style Guidelines

Lab assignments will be graded in at least the following areas:

- **Correct results**: the program should work, and be seen to work. This involves not only producing correct output, but also reacting well to user error, incorrect input, etc.. The code should be *robust*.

- **Coding style**: it is not uncommon for industry to impose a *way* of writing software that is uniform across all programmers. For the purpose of this class, we have the following:

    - The use of variable names that is descriptive of the use of the variable. Variable names should begin with a lower case letter. For instance, use the variable name `hoursWorked` to keep track of the number of hours a person has worked, instead of using `x`.

    - Symbolic constants should be named descriptively, and be in all upper case letters (i.e. **`var`** `PI=3.141592`) There should be no "magic numbers" in your code. Magic numbers are undocumented constants hardcoded into the program. The exceptions are –1, 0, 1, and sometimes 2. Any other constants should be named.

    - Do not write code beyond the 80th character column.

    - Indent the body of blocks (**`while`** loops, **`if`** statements, **`for`** loops, etc.) consistently using 4 spaces (not tabs). You may want to adjust your IDE to do this each time.

    - Code one statement per line.

    - Follow good structured and/or object oriented design and programming techniques (more on this throughout the class)

    - Write standard, portable code. Only the default libraries and the libraries provided with the textbook will be available. Therefore, only they may be used in your programs.

- **Documentation**: good programs require more explanation than is available in the code listing itself. It is required that you place comments throughout your code that explain the workings of your algorithms, the use of variables, any special or tricky points of code, and any modifications of global variables. When in doubt, follow the golden rule of documentation: *Document as you would be documented unto.* If all your code were stripped out, a reader should still be able to follow your thought process as you were programming. Remember, documentation tells another programmer what you intended, not how you are doing it.

    - A comment box at the beginning of the program that has the following format. The format is very specific, because it will allow "automatic" programmer's API web page generation.

```
/**
 * One-line description of code in this file.
 * Longer several line description of the file
 * contents, summarizing the goal of the
 * assignment and what the class does.
 * @author YourFirstName YourLastName
 * @version 1.0 (or whatever version)
 */
```

    - Each function or method should perform one task, and no function should reasonably exceed 50 lines (this is a guideline, not a rule). Most functions will be less than 10 lines. Function names should begin with a lower case letter, and be descriptive of their purpose. Each function should be documented with a comment block similar to the following:

```
/**
 * Brief description of function.
 * Long, perhaps several line description of the function
 * contents.  It should describe what the function does,
 * not how it does it.
 * @param paramName1 parameter 1 description
 * @param paramName2 parameter 2 description
 * @return return value description
 */
```

    - Inline comments starting with // should be used to document what is intended in any tricky or otherwise complex code. For example:

```
Array.prototype.applyFunc = function (func) {
    // apply the function to every element of the array
    for (var i in this) {
        // replace the value with the new value returned
        // from the function.
        this[i] = func(this[i]);
    }
}
```