CMSC 101 / IS 101Y - Fall 2013 - Processing Coding Standards

In order to make code more readable, most organizations will have written style guidelines for code. These can be either fairly flexible or quite rigid, and are designed to make code easily maintained by large teams of programmers over a long period of time. This document explains that coding standards and expectations that will be used for CMSC 101 / IS 101Y assignments. Part of your grade for each programming assignment will be based on the design and style, including following the standards in this document.

General Design Suggestions

- If the same (or very similar) code appears in multiple places, it should be separated out into a named function to reduce repetition and improve readability.
- Generally speaking, functions should not be longer than a single screen -- functions that are longer than this should be decomposed into several smaller functions.

Use of Whitespace

The use of whitespace (blank lines as well as space) goes a long way to making your program readable.

- Blank lines should be used to separate major parts of a function.
- Use indentation to make code easier to read.
    - For example, after a method you should indent the code underneath it so you can easily identify it from the other code.
    - Handy tip: Under the "Edit" menu, "Auto Format" will reindent all of your code. Keyboard shortcut on Mac: command-T.
- Use spaces around all operators. For example, write **x = y + 5**, NOT **x=y+5**.

Comments

Comments are the programmer's main source of documentation. In Processing, any text that is preceded by "//" will be treated as a comment (i.e., ignored by the Processing interpreter).

<u>File Header Comments</u>

Every file should contain an opening comment describing the contents of the file and other pertinent information. This "file header comment" **MUST** include the following information.

- The file name
- Your name
- The date the file was created
- The class name
- Your UMBC e-mail address
- A description of the contents of the file

This document is adapted from "CMSC 201 - Python Coding Standards." *Computer Science I for Majors*. N.p., 21 Jan. 2010. Web. 28 Aug. 2013. <http://www.csee.umbc.edu/courses/201/spring10/standards.shtml>

For example:

```
// File:    Bubbles.pde
// Author:  Marie desJardins
// Date:    8/26/13
// Class:   CMSC 101
// E-mail:  mariedj@cs.umbc.edu
// Description:
// This program shows colorful bubbles chasing each
// other around the screen.
```

Function Header Comments

EACH FUNCTION must have a header comment that includes the following:

- function name
- a description of what the function does
- a description of the function's inputs
- a description of the function's outputs (if any)
- side effect of the function (if any -- this should be rare)

For example:

```
// findCircleArea(Radius) calculates the area of a circle
// Input:  Radius, the circle's radius
// Output: the circle's area
```

In-Line Comments

- "In-line" comments are used to clarify what your code does, NOT how it does it.
- Well structured code will be broken into logical sections that perform a simple task. Each of these sections of code (typically starting with an 'if' statement, or enclosed by a loop) should be documented.
- Any confusing-looking code should also be commented.
- Do not comment every line of code. Trivial comments (// increment x) are worse than no comments at all.
- An in-line comment should appear *above* the code to which it applies and should be indented to the same level as the code.

For example:

```
// print all odd numbers in array Numbers[]
 for ( i = 0 ; i < Length ; i++ ) {
     // if Numbers[i] is odd, print it; if even, do nothing
     if ( Numbers[i] % 2 == 1 ) {
          println (Numbers[i]);
     }
```

This document is adapted from "CMSC 201 - Python Coding Standards." *Computer Science I for Majors*. N.p., 21 Jan. 2010. Web. 28 Aug. 2013. <http://www.csee.umbc.edu/courses/201/spring10/standards.shtml>

- Avoid endline comments except for variable declarations :
    - OK:
      ```
      int Numbers[];  // list of numbers to check for oddness
      ```
    - not OK:
      ```
      println (Numbers[i]);  // print an odd number]
      ```

Naming Conventions

Using these consistent naming conventions for functions, variables, and constants will help a reader to understand what your code is doing. You are not required to use these particular conventions, but you should be consistent in the naming that you use (e.g., don't sometimes capitalize and sometimes not capitalize variable names).

- Use meaningful variable and function names ! !
    - For example, if your program needs a variable to represent the radius of a circle, call it 'radius', NOT 'r' and NOT 'rad'. The use of single-letter variable names is strongly discouraged (except for counter variables like "i" and "y").
    - The use of obvious, common, meaningful abbreviations is permitted. For example, 'number' can be abbreviated as 'num' as in 'numStudents'.
- Begin variable and function names with lowercase letters and use "camel case"[1] for multi-word names
    - For example: listLength, circleRadius.
- Constant names should be capitalized, again using camel case for multi-word names -- e.g., TaxRate.
- Note on global variables: In future programming courses, you will likely be taught that global variables (i.e., variables defined outside the scope of any function) are not allowed. Although this is generally true, in this course, we won't worry about this requirement. In fact, the arrays in CrashBasics.pde (Processing Assignment 3) are all treated as global variables.

Use of Constants

To improve readability, you should use constants whenever you are dealing with values that don't change while your program is running. Your code shouldn't have any "magic numbers": that is, numbers used in the code rather than being defined as a constant. In general, this means that no numbers should be used in calculations other than 0 or 1. Here's an example of poor style:

```
total = subtotal + subtotal * .06
```

.06 is a magic number. What is it? What does it mean? We don't know just by looking at the code. So, at the very least, this line of code would require a comment. However, if we use a constant, the number's meaning becomes obvious, the code more readable, and no comment is required. Constant

[1] "Camel case" is a term used to refer to multi-word variable names in which the first letter of each word is capitalized, and the other letters are all lowercase.

This document is adapted from "CMSC 201 - Python Coding Standards." *Computer Science I for Majors*. N.p., 21 Jan. 2010. Web. 28 Aug. 2013. <http://www.csee.umbc.edu/courses/201/spring10/standards.shtml>

definitions are always placed near the top of the program so that if their value ever changes, they are easy to locate and to modify. Constant definitions should always be outside of any function, so that they are treated as “global” (available in any of your functions). The “fixed” keyword tells Processing that a definition is a constant, rather than a variable.

```
fixed float TaxRate = .06;          //

      ... (lots of code goes here)

total = subtotal + subtotal * TAX_RATE

      ... (other code goes here)

println ("Maryland’s sales tax is “ + (100 * TaxRate) + “ percent”);
```

This document is adapted from "CMSC 201 - Python Coding Standards." *Computer Science I for Majors*. N.p., 21 Jan. 2010. Web. 28 Aug. 2013.
<http://www.csee.umbc.edu/courses/201/spring10/standards.shtml>