Stat 650/750 Programming Practices Checklist

The checklists below are intended to give you some guidance in developing clear, readable, and effective code. Review these items throughout your development process.

The initial guidelines are language independent, a variety of language-specific guidelines and pointers are given in another document.

Basic Principles

- ☐ Someone else reading my code would find it easy to understand.
- ☐ I have decomposed complex tasks into smaller sub-tasks.
- ☐ I have preferred small and focused functions.
- ☐ I have organized my code so that it is doing one task at a time.
- ☐ I have made my code concise without sacrificing clarity.
- ☐ I have avoided needless code repetition.
- ☐ I have used a *consistent* style throughout my code.
- ☐ I have used a style *consistent* with common standards for my programming language.
- ☐ I have preferred to use library routines when they meet my needs.

Names

- ☐ I have used names that are meaningful, concrete, and descriptive.
- ☐ I have avoided using abbreviations unless they are very well known.
- ☐ When it would improve clarity, I have used suffixes to convey additional information or attributes such as units, formats, and conventions (e.g., `start_time_ms` for a time in milliseconds).
- ☐ I have checked my name choices to remove interpretations that are misleading or ambiguous. (Example: boolean variable `read_file` versus `must_read_file` or `have_read_file`.)
- ☐ I have adhered to the naming conventions and idioms of the language/system I am using, and if I have not, I have done so consistently and with good reason.
- ☐ Classes are typically named with descriptive nouns or noun phrases.
- ☐ Functions that primarily perform a procedure are named descriptively with strong verbs and an object (e.g., `addDataRow`).
- ☐ Functions that primarily return a value are named to describe that value (e.g., `penColor`).
- ☐ Constant names are all `UPPER_CASE` with words separated by underscores; this includes constant class data.
- ☐ I have avoided generic names like `tmp`, `retval`, `it`.

- ☐ Loop iterator variables are meaningful (e.g., `for member in members` or `for (count.index in 1:length(counts))`.
- ☐ Generic indices `i`, `j`, `k`, and `l` are integers, if used at all.
- ☐ I have considered the expectations of readers when choosing names.
- ☐ I have freely used long names as needed to be descriptive, but allowing shorter names for variables visible in narrow scopes.
- ☐ My variable names clearly distinguish private members and class data from other forms.

Formatting

- ☐ I have organized my code into "paragraphs" focused around a single idea or task.
- ☐ I have separated these "paragraphs" by a single blank line, with *no* blank lines at the beginning or end of a function.
- ☐ Comments leading a code "paragraph" are not separated from the code by whitespace.
- ☐ I have used single blank lines between entities in the code (paragraphs, functions, classes, comment blocks) and two blank lines between major conceptual sections.
- ☐ While being consistent with the previous items, I have minimized vertical whitespace.

- ☐ Lines have length less than 80 characters unless absolutely necessary.
- ☐ I have arranged line breaks consistently, compactly, meaningfully.
- ☐ I have used column alignment when it helps to clarify common structure (e.g., lining up ='s in a series of assignments).

- ☐ I have avoided the use of tabs, preferring spaces.
- ☐ I have left no trailing whitespace.
- ☐ I have used a consistent indenting (preferably 4 spaces but 2 is fine)
- ☐ I have used spaces around all binary operators including assignments, except when it is helpful to highlight factors in a complicated expression (e.g., `x*y + y*z + z*x`).
- ☐ I have used spaces after commas in function calls (but not before).
- ☐ No spaces surround = for naming parameters in function calls (e.g., `foo(x, y=2, z=0)`).
- ☐ I have avoided unnecessary parentheses in `if`'s, `while`'s, and `return`'s.
- ☐ Function calls have no spaces before and after parentheses.
- ☐ Line breaks follow commas (function calls) or operators (expressions).

Documentation

- ☐ Comments are as concise as possible without sacrificing clarity.
- ☐ High-level comments help clarify the *intent* of the code.
- ☐ Comments do not describe the obvious or trivial or facts that can be inferred directly from the code.
- ☐ Comments highlight tricky or non-obvious features of the code or algorithm, potential issues or problems, todos or fixes, and commentary on performance and function,
- ☐ Longer docstrings and block comments are written in clear, complete, and grammatical sentences.
- ☐ Comments describing major changes have initials and a date.

- ☐ Each file begins with a comment describing its purpose and content, author/maintainer, contact email/url, and a version number. The first line names the file and give a short descriptive phrase.
- ☐ Each function has a docstring or comment that describes what the function does and how to *use* it.
- ☐ Each class begins with a comment describing its purpose, behavior, and use. The first line gives a short descriptive phrase.
- ☐ Discussion of *how* a function/class does its job are placed in comments in the body of the code rather than in the docstring.
- ☐ Docstrings begin with a short, descriptive phrase or sentence on its own line. (This may be all that is needed in simpler cases.)
- ☐ Longer docstrings/comments describe the *contract* fulfilled by the function or class by giving information a caller/user should know, including description of inputs and outputs; pre- and post-conditions and invariants, side effects, likely pitfalls or caveats, and performance.
- ☐ Each function's docstring describes its behavior precisely.
- ☐ Longer function docstrings highlight references to the parameters.

- ☐ Block comments have the opening comment character lined up in the column at the same level as the relevant code, with no blank lines between them.
- ☐ Inline comments trail a code line.
- ☐ All comments and docstrings have lines $<$ 80 characters in length.
- ☐ I have used brief examples in comments to illustrate boundary cases where helpful.
- ☐ I have avoided ambigous pronouns in comments.

Organization

- ☐ My code reads from top to bottom.
- ☐ I have grouped related statements together (cf., “paragraphs” above).
- ☐ My code makes the dependencies among statements salient.
- ☐ My names for routines and parameters help clarify these dependencies.

- ☐ I have tried to make my loops short enough to be read all at once.
- ☐ I have limited nesting to three levels whenever possible.
- ☐ I have made my loops’ termination conditions clear and ensured that the loops will always end.

- ☐ I have avoided using global variables.
- ☐ I have avoided hard-coded numeric constants in the code.

- ☐ I have aggressively identified code that tackles unrelated subproblems and extracted it into its own general-purpose, reusable modules.
- ☐ I have written and run tests of my code’s basic functionality, possibly before writing the actual code.