[The following guidelines use the word program but hold true just as well for code pieces, i.e., functions, classes, etc.]

Programs should, optimally, be <u>functional</u>, <u>elegant</u>, <u>efficient</u>, <u>user-friendly</u>, and <u>self-documenting</u>.

Why bother with programming style? Lots of reasons.

- ease in debugging
- to enable other programmers to follow your logic and the way it was carried through in the program. A program is usually used many times, by many different people.
- You may want to use your program again yourself in the future. A poorly written / documented program is hard to understand and does not lend itself to reuse.
- A well-written program is often more efficient and economical than a poorly written one.
- Modifications may be called for in the future. An unintelligible program is a monster to modify and debug.

A. <u>Functional</u> - The program should do what it is called upon to do. It should be error-free and produce output that is correct.

B. <u>Elegant</u> - The program should do it in the best possible way. Avoid convoluted logic and awkward control mechanisms. `break`, `continue`, and `goto` statements are occasionally useful - employ them only when absolutely necessary.

C. <u>Efficient</u> - The program should not be unnecessarily wasteful of computer resources, namely, time and space (memory). Avoid the use of extra variables and/or operations that are not needed for the functionality or readability of the program.

D. <u>User-friendly</u> - The program should make as few assumptions as possible about the capabilities of the user.

   This means giving interactive user screens and printed reports a “natural” style and appearance. As always, we are trying to bring the machine up to the level of the human user (rather than the other way around). We use prompts to let the user know what kind of data is expected and in what format. Whenever possible we provide the user with a menu of options or possible data values to input. We validate the input data and provide helpful error messages (and a chance to enter the data again) if the input data is not in the correct range, format, etc.

E. <u>Self-documenting</u> - The program should contain within the source code as much documentation information as possible.

The program should be structured to highlight the major processes involved. Use descriptive names for variables, constants, functions, classes, objects, etc. Employ a recognized and consistent indenting scheme. Avoid unnecessary variables - too many variables not only waste memory but they also make programs harder to understand. Liberal use of comments: to explain what each variable name stands for; to clarify complicated calculations; to give information regarding the data needed by the program: data format, type, files used, etc. Comment block at the beginning of the program can contain: descriptive name for the program, author's name, date written, date last modified, purpose of program.

Assignment: Redo a programming assignment of your choice to make it as user-friendly and as self-documenting as possible. Submit together with your previous version of the assignment.