Good Programming Guidelines

These guidelines are intended to provide students with an introduction to good programming practice. These guidelines assume that you have already written a program that is correct. They are intended for programs that are relatively small; larger programs require more complex guidelines. These are intended to be helpful in letting the instructor, other readers, and you better understand the code you have written.

Good programming practice should include the following considerations:

Good program structure: This means code that is inherently easy to read and understand. Go to statements are never used, except when required for VBA error trapping. Well-structured programs have a control flow that can be easily seen by someone reading the code. Well-structured code does not have many levels of nesting within a single function. Instead, the overall code is broken up into several functions, each of which performs a small, distinct task. Some programmers believe that each function should have only one point where values are returned to the calling function. Others believe that more than one return point may be okay, if justified by the logic of the function. For example, a function that does an iterative calculation until convergence is achieved or some maximum number of iterations is exceeded may have two return points: one for the converged result, the second for a result that has not converged. (However, it is also possible to write such a function with only a single return.)

Effective use of comments: These statements describe what the code is supposed to do so that another person can follow the code. It also makes it easy for the author to review the code at a later date. Each function should start with an introductory set of comments giving the programmers name and date and a statement of the purpose of the function. It is especially important to describe all the arguments in a function or sub, what ones the user must supply, what ones are returned, and any other information such as the data types and expected values for each argument. An overall description of the entire program is also helpful. Avoid comments, which are obvious from reading the code. Comment statements can be used to define the purpose of each variable. (In this course, comments are judged not only on their technical content but also on the quality of the writing, including spelling and grammar.)

Well-formatted output: Your output should be easy to follow and understand. Although it is possible to use complicated output statements to develop elaborate output, it is also possible to use simple output (e. g. msgbox “x = “ &, x). This technique provides results that can be understood although they may not be perfectly formatted. Labeling of results so that they can be understood is more important that perfection in the output formats.

Meaningful variable names: Variable names should be chosen which make the meaning of the variables clear. For mathematical software it is usually convenient to use the symbols that are used in equations for variables. Thus Ohm's Law might be written as e = i * r. This is just as meaningful as writing voltage = current * resistance. Sometimes Greek letters are spelled out. For example, the common symbol for density is the Greek letter ρ. Computer programs can use density or rho as variable names for density.

Long variable names are sometimes separated by an underscore such as `average_grade`; an alternative is to capitalize the start of the new word as in `averageGrade` or `AverageGrade`. Symbolic constants are often written in all capital letters such as PI or MAXIMUM_ELEMENTS.

There are a variety of naming conventions that may be used to define variables. This usually involves a prefix that indicates the scope of the variable or the variable type. Such guidelines are used for the development of complex programs where it is useful to have such information and where professional programmers are familiar with the notational conventions.

Portability: Always use standard language concepts; do not use special options for a particular platform. This assures that programs written for one machine can be used on another machine with a minimum of conversion work. This should not be a problem for the VBA or MTALAB programs in this course.

Documentation: This includes a discussion of how the overall code and the various functions work. It also includes a list of all the variables and their meaning. The intention of program documentation is different from that of user instructions. The latter are intended to allow another person to use your routine as a black box. Program documentation should allow another person to be able to understand how your program works and to modify it if necessary. For VBA codes that require each variable to be assumed a data type, using a single Dim statement for each variable, followed by a comment describing the variable is a useful technique.

Neat Appearance of the code: This includes the following: indenting structures such as loops and if blocks, using blank lines to separate comments and different portions of the code, using symbols such as rows of asterisks or =equal signs to separate portions of the code, and using white space in statements. In general any process that makes the code easier to read and understand should be used. In the example below, the block on the right is easier to read than the block on the left, even though both blocks accomplish the same task. For additional examples look at the appearance of the codes provided in the various laboratory exercises you have been assigned.

The two sets of statements below do exactly the same task. Which of these is easier for you to read and understand?

<table>
<tr>
<td><pre>If x=3 then z=y+w/x : y=x*x/cos(z) : Else z=y-3*x : End If</pre></td>
<td><pre>If x = 3 then
        z = y + w / x
        y = x * x / cos(z)
Else
        z = y - 3 * x;
End If</pre></td>
</tr>
</table>

Generality: This means a program, which can handle as general a problem as possible with no modifications to the routine. Always use variable names to represent quantities, which may change in some future application of the program. Use symbolic constants, especially for array dimensions. Writing general subs and functions allows code for basic calculations to be reused in later programs.

The items in the above list are given in roughly the order of their importance. The major idea is that the code should be easy to understand when it is read by someone not familiar with the code. This includes the code's author who is viewing the code after several months.

Students enter this course with a wide variety of backgrounds in programming experience. Students who have no previous experience in programming will be working hard just to complete the assignments. Those of you with some programming experience should take more time to see if you can follow all of the guidelines presented here.

All students should be concerned about simple items like meaningful variable names, the use of white space and indenting portions of the code to show its structure. These items will make your code easier for you to understand and to debug.