Good Programming Practices
ReactorLab.net

1) When starting a project, outline the logic and tasks with comments and blank lines between the comments. That way, you can focus on the logic don't have to worry about command syntax. Then add a few lines of code below a comment and test the program to check for an error. An error in a few lines of code is easy to find. Don't write dozens of new lines before you check, since you may then have many new errors to find.

Remember that programming is an iterative process: write some code, test, debug, write some more... Every progammer makes mistakes, adds bugs. Do not expect to write a perfect program, or even an outline, all at once at the start.

Some people like to outline project logic with a flowchart. For examples, see <http://www.rff.com/structured_flowchart.htm> (accessed Aug 2016). You can draw these by hand. Projects in object-oriented languages can use the Unified Modeling Language (UML) to design projects.

I have found informal sketches useful for designing large projects.

2) Use short descriptive variable names. Consider camel notation (myBigIdea) instead of having to type underscores between words. Consider Hungarian notation to help you remember what values a variable contains: add class identifier at start, e.g., iMyInteger, mMyMatrix, fMyFunction.

3) When you first use a variable, add a comment to give units and a description. I encourage accounting for units (kg, m, s, etc.) from the start of an engineering project. This will help you to include all components of your equations.

4) Comment your code so someone else can understand it. Comment your code so you can understand it when you come back to it later. Understanding your old code is not as easy as it sounds.

5) Assign all numeric values to variables in a one section at the top of a code unit so you only have to change values in one place.

6) Right after you start a repeat or decision structure (for, while, if, select) add the end statement, and only then start coding inside the structure. That way you won't forget the end statement when coding a long structure and get an error message.

7) Don't repeat code. Put blocks of code you see repeating into a function. This makes code shorter and easier to read, and you only have to make a change in one place, minimizing work and chance for typos and missed code.

8) Modularize. Break project into main and function units, each no longer than about 1 screen long. This makes it easier to read the code and understand the project logic. (Attaway, 3rd ed, p. 204)

When you are outlining a project, you can start the modularization process by writing "function stubs." A function stub is quick to write, prints something like "enter function fMyFunc01' and returns trial values so that the main program can be run. The stub contains minimal lines, and you go back and add full functionality later. (Attaway, 3rd ed, p. 224)

Consider breaking up long functions into a main function and sub functions, which the main function calls. The sub functions can be in the same disk file as the main function. (Attaway, 3rd ed, p. 207)

9) Minimize or eliminate the use of global variables. Global values can change unexpectedly when adding new program units. Understand "variable scope" and local, global, and persistent variables. (Attaway, 3rd ed, p. 215)

10) Put output, such as plotting, in a code section in Matlab so you can work on the output without having to run the entire program.

11) Learn the debugging features of your programming tool. In Matlab, search the documentation for "debugging" (in Command window, >> doc debugging) and read the top result on debugging and the Examples and How To sections, especially. By adding "breakpoints" to a program and then running it, the program will pause at each breakpoint. During the pause, you can examine variable values in the Workspace window. This will help you understand what the state of the program is at the breakpoint and find errors in logic. (Attaway, 3rd ed, p. 220)

Remember that there are two main types of bugs, which I call syntax and logic errors. Syntax errors are usually found by Matlab and the type of error and location in the code is reported. Logic errors are tougher to find. The program runs with no errors reported by Matlab but you get bad results. Your program logic is bad somewhere.

12) Check your program by entering test input data for which you know the correct results by a method independent of your program. We used this in our loan calculator by checking results against web calculators. We also used this idea when we compared the results of analytical integration of an ordinary differential equation (ODE) against Euler's numerical method. In that case, however, the differences were due to the numerical method being an approximation and not due to program errors.