Coding Framework

Andres Patrignani and Tyson Ochsner
Soil Physics, Plant and Soil Sciences, Oklahoma State University

The eight steps presented below will guide you through the process of building error-free, effective, and well-documented codes that are targeted to specific needs using appropriate programming languages.

1. **Define objective of the code.**
   Briefly describe the purpose of the code. What need does your idea satisfy?
   The objective of a code can be as simple as analyzing a set of data or as advanced as a complex simulation.

2. **Sketch the process (Prototyping)**
   Informally, draw out in a piece of paper how you foresee the code will work. Include the main inputs, local variables, and outputs, and how they connect with each other. This step will help you defining variables, main processes, and the direction of the process in the code. This can become awkward, so again keep it simple, and keep in mind the main relationships.

3. **Write a Pseudo-code**
   Literally write a few sentences on how you expect the code to work. Describe if inputs will be retrieved from a database or from the user, if you see your code as a webpage, as a simple script, or as a function.

4. **Choose a proper programming language.**
   Based on the outcomes of the previous steps you should choose the programming language that best suits your code. For instance, if you are planning to develop a webpage, PHP is a possible language since it was developed for this purpose. On the other hand, a simple script that analyzes data from a scientific experiment can be coded using Matlab. If you want to program a driver or a router, then C or C++ might be more appropriate options.

5. **Compile a list of variables and functions that may be necessary (Brainstorming).**
   Try to make a tentative list of possible functions. Determine what is going to be defined as a coefficient and what is going to be set variable. You can start by having only few variables, this will help you write an error-free code and maintain the focus on the main inputs and outputs. Then, when you have certainty that the code is working as expected, you can change some coefficients to variables for added flexibility.

6. **Draft the code.**
   This can be done first on paper and then in the Matlab editor, or straightforward in the Matlab editor.
   In this step it is important that you have good programming habits. Always include your name, date, purpose of the code, and a brief description of inputs and outputs (units, string/numeric, mandatory/optional). Make sure you include meaningful comments in almost each line.

7. **Error debugging**
   Finding the errors in your code is critical. At the very end we are building a code on which we can rely and use as many times as we want. At this time you have to keep in mind that syntax errors will prevent the code from running from beginning to end. For instance, syntax errors can be a missing parenthesis, a missing dot, an extra comma, misspelling a function name, a missing or extra "end" statement, etc. Also, mathematical operations between vectors and matrices of different size sometimes cannot be performed. Using the built-in Matlab debugger can be helpful, but also meticulous examination of the code is required. In Matlab programming terms you can debug your code using the following reasoning:

   ```
   Run code
   if number of errors >0
      Find error
      Fix error
      Run code
   end
   ```

8. **Polishing the code**
   This step involves improving the time efficiency of the code by re-writing some lines of code in a more time-efficient way, by replacing functions with similar but more efficient (or newer) functions, removing unnecessary code from loops, using logical indexing, replacing loops by element-wise or array operations, by pre-allocating memory, etc. Also, pay special attention to parts of the code that need comments, and be sure to have a well-documented help section at the beginning of the code.