CS4 Coding Requirements and Tips

January 2015

Requirements

Below is a summary of the coding requirements for your CS4 programming assignments.

1. Comments
   - Begin each script with a comment that states the script’s file name. Follow this with line(s) that provide a succinct description of what the script does.
   - In a function, create comments following the function header that give the function’s specification concisely, including parameter information. These comments must be formatted so that the MATLAB help system can make use of them. Use `doc` to view “Add Help for Your Program,” or visit Add Help for Your Program.
   - Code should be sufficiently, but not excessively, commented. Define constants and important variables with comments and write a concise comment for each individual block of code.

2. General style
   - Use meaningful variable names; they do not need to be long.
   - Save important constants in a variable. Eg, use `badpi = 3.14` instead of simply using the the number 3.14 in your program.
   - Use proper indentation. Note that Smart Indent (highlight code, then right-click and select “Smart Indent”) is a quick, albeit not foolproof, way to format your scripts.
   - Line lengths should not be excessively long—a length of 80 columns is reasonable.
   - Use semicolons as needed and never at “mid-statement,” such as at the end of these lines: if, elseif, else, for, and while. Note the distinction between the end of a line and the end of a statement. Here is an example with an ill-placed semicolon:

```
if x < y; % Semicolon is ill-placed at the end of a line--the if statement isn’t complete here
    x = x*2; % Semicolon is correctly placed at the end of a statement
end
```
3. Clarity & Efficiency

- Write for clarity, but your code should also be reasonably efficient.
- Optimally efficient code makes the minimum number of calculations (or uses the least amount of time) needed to complete a task correctly. Optimally efficient code is not required; reasonably efficient code makes “close” to the minimum number of calculations, but sacrifices the use of “tricky” expressions, and use of excessive numbers of intermediate variables, for the sake of clarity and conciseness. Reasonably efficient code is required.
- There should be no superfluous code (e.g., an empty or unreachable if branch or a useless loop).

Additional information and tips on meeting these guidelines is included below. Read carefully!

Scripts

Scripts are saved as .m files. They are a set of instructions for MATLAB to execute in sequence. In many of your homework assignments, you’ll be asked to write a script that accomplishes a specific task. In each script file, add a comment at the top with the name of the script and a brief description of what the script does. For example:

```
% printEven
% Prints all even numbers between 0 and 10

for i = 0:2:10
    fprintf(’%d’, i)
end
```

Comments and General Style

Comments are very important. Along with having clear code, they allow other people and yourself to easily use and maintain code. Such code is considered to be well-documented.

1. Comments

- Code should be interspersed with useful comments (“inline comments”). These comments might define constants and important variables.
• Big chunks of code should be broken up into sections with comments describing the purpose of each section.
• All this being said, don’t go overboard with comments! Having good variable names and clear coding style should make your code almost “self-documenting.” This is good, because code itself is the most accurate indicator of what your program actually does. An outdated or inaccurate comment is worse than no comment at all. Do not clutter your code with obvious or irrelevant information, such as “This line of code assigns 5 to the variable a.”
• Good comments are concise.

2. Variable names

• In the same spirit of readability, names of variables should be meaningful and correspond to what they represent. For example, initialization of a variable containing a person’s height in inches could look like height_inches = 64;, or even h = 64; % height (in). Simply using x = 64; is not as clear.

3. Indentation

• Use proper indentation. Properly indenting code prevents errors and makes code easy to read. Matlab’s smart indent feature (control-i) is very helpful for doing this.
• In general, code within a block (e.g. if-else, for-loop, while-loop) should be indented. If another block is nested within the first, it should receive one more level of indentation.
  ```matlab
  % myMult
  % Print the multiplication facts
  for i = 1:12
    fprintf(’%d Multiplication facts:’, i)
    for j = 1:12
      fprintf(’%d x %d = %d\n’, i, j, i*j)
    end
  end
  end
  ```
• Keep line length manageable (i.e., about 80 columns wide). Use ... to break up longer lines of code.

4. Index variables

• Do not alter the value of for-loop index variables within the body of a for-loop. Really!
Testing & Debugging

Testing is an important way to make sure your code is correct. Testing is not simply running your function once and saying to yourself, “It works.” Be sure to try your program using all the interesting cases (i.e. cases where it might have difficulty), not just a typical case. Check your work by comparing results to a case where you can work out the expected result in some other way.

While you are writing or debugging code, it can be very helpful to see the values associated with with various statements in your code. Matlab will print out the value associated with statements that do not end in a semicolon ;. 

Omitting semicolons can be very handy. However, you should suppress all this extra information by adding the semicolons back to your code before you turn in your assignments. Printed output that is part of your finished assignment should be explicitly generated using disp or fprintf.

Additionally, Matlab has a built-in Debugging Tool. In any script or function, you are able to click the tick mark next to the line number to set a breakpoint, at which Matlab will pause and give you access to all the variables as they are stored at that point through the Command Window. Once you click the tick mark, it will turn into a dark orange dot to indicate that there is a breakpoint there. This is a straightforward way to see what exactly is happening at key points within your code.

One last note about testing: when you think you are ready to submit your code, it’s a good idea to first clear the workspace (by typing clear all in the Command Window) and then run all the code for one final check. If any changes need to be made, you should repeat this process to make sure that what you are handing in is exactly what you expect it to be.

Before you submit your assignment...

Below is a list of common issues compiled by past and current CS4 TAs, who have had extensive experience working with Matlab. Please use this section as a checklist to review for your work before you turn it in, or before you ask for help with debugging!

- Run all of your code. Matlab will print errors in your Command Window with a description of the problem, as well as the specific line of code in which that error occurred. It seems basic, but please pay attention to these errors. Sometimes, all you need to debug your code is in these printouts. You’d be surprised how many people ask us for help, then have an epiphany about how to fix it as soon as the TA points out the error message.

- Do you have all the necessary end statements?
• Should you be using a dot operator (e.g. should you be using * or .* for multiplication?)

• Do you have any unsuppressed lines of code? Suppress unnecessary output with semicolons.

• Are you efficiently making use of other functions that you’ve already written to avoid redundant code? If you have a function elsewhere in the same directory, you can call it in another function instead of having identical code in two different functions.

• Did you define the output variable within your function?

• Did you name your variables properly? Variable names cannot contain any spaces. Also, be careful with differentiating between the lower-case letter ‘l’ and the number ‘1’, as they look identical in many versions of Matlab.

• Did you index into your vectors or matrices appropriately? The first element of a vector is indexed as 1 in Matlab (several other programming languages use 0).

• Are all of the variables in the script defined before they’re used? If you haven’t defined a value for a variable and ask Matlab to use it in a calculation, Matlab will object.

• Does your function name match the file name? (If your function header reads function out = myFunction(in), the .m file containing this function should be saved as myFunction.m.)

• Use clear all and clc to clear all variable assignments made during your current work session and clear your Command Window in Matlab. After running these commands, execute your scripts again. Sometimes, your scripts will have only worked before because of a variable that you had defined outside the context of the script.

• If you’re using a while loop, could it be producing an infinite loop?

• Are you differentiating between = and ==? The former is for assignment; the latter is to check whether two things are equivalent.

• Are all of the files you need to submit in the correct directory?

Happy programming!