MATLAB lesson 6: m-files
Exercise sheet

Dr. Gerard Capes*

1 Scripts and functions

Based on the lesson

1. Basic script
   (a) Clear all variables in your workspace
   (b) Create two variables `a = 2` and `b = 3`
   (c) Write a script which multiplies these variables together and saves the result in a variable, `c`
   (d) Save your script using a suitable name. Check that this name is not already in use by another m-file or MATLAB function.
   (e) Run your script, and note that the variable `c` remains in the current workspace after the script has finished
2. Convert script to function one part at a time
   (a) Modify the (one-line) script from question 1 by adding the `function` keyword, then the name of the function on line 1 (give your function a suitable name). On line 3 add the `end` keyword
   (b) Run your function. Why doesn't it work? (Look at the error message in the command window for clues. Refer back to the lesson on function m-files and think about variable scope.)
   (c) Modify the function so that it takes two input arguments
   (d) Confirm that your function runs, using the variables `a` and `b` in as input arguments
   (e) `Clear` the variable `c` from your workspace, then rerun the function — why is the calculated variable `c` not in your workspace?
   (f) Finish the function by adding an output argument

*Questions and feedback can be directed to gerard.capes@manchester.ac.uk

(g) Re-run your function with `c` as the output argument and `a` and `b` as the input arguments. Confirm that the output variable `c` is now in your workspace.

(h) Re-run the function, this time specifying a variable `d` to contain the output and use 5 and 6 as the input variables

(i) Add an `H1` line to your function so that it works with MATLAB's `help` and `lookfor` functions. Check that it does.

Also requires some use of MATLAB's help

3. Structure arrays, assertions and nested functions

(a) Check there are some .m files in your current working directory. There should at least be the .m file from question 1. If you have only one or two .m files, create some more .m files using a text editor. The content doesn't matter; just make sure you have at least 5 files.

(b) Read the help entry for the `dir` command, and try some of the examples given. Then write a script that creates a structure array, which contains information about all the files in your current directory with a .m file extension. Save your script with a suitable file name. Double click on the structure you have created and look at the contents.

(c) Modify your script so that the file name and file size are printed to the screen if the file size is greater than 150 bytes. If the file is smaller than 150 bytes output text to indicate the file is very small.

(d) Convert your script into a function which takes a string as an input argument to determine which file types are searched for. Give your function a descriptive name and call it from the command window, testing the results for different file types. Make sure your function works with `help` and `lookfor`.

(e) Call your list-files function using a file extension which won't match anything in your working directory (e.g. *.invalidextension). It should produce no output, but also no error message.

(f) Add an `assertion` to your list-files function which halts execution if no files are matched, and gives a useful error message.

(g) Write a function which converts an input in bytes to an output in KB. Make sure your function works with `help` and `lookfor`.

(h) Call this bytes-to-kilobytes function from within your list-files function from question 3f and output the combined size of all .m files greater than 150 bytes.

(i) Modify your list-files function to take a second input argument which is the threshold in bytes. Run your function for different combinations of file extension and size threshold.

(j) If you haven't already done so, write a comment above each line of code to explain what it does

2 Optimisation

Based on the lesson

1. Download this script and save it in your current working directory. Run it in MATLAB and note that it takes a *very* long time to run. Don't wait for it to finish - click in the command window, then press `ctrl + c` to abort execution.

2. Now look at the script in the MATLAB editor and hover your mouse cursor over the orange coloured marks in the MATLAB editor (on the right edge, beyond the scroll bar). Read the suggestions for improving efficiency.

Requires use of MATLAB's help

3. Make the suggested modifications (adding semi-colons, pre-allocating arrays) one at a time. Re-run the script and note how long it takes after making each modification. When you have made all of the suggested edits, it should be *much* quicker.

4. Use the profiler to examine which lines in the script take the most time.

5. Investigate further improvements to the efficiency of this script – hint: read the following sections of the MATLAB help:

   (a) MATLAB/Advanced Software Development/Performance and memory/Code performance/Concepts/Techniques for improving performance

   (b) MATLAB/Advanced Software Development/Performance and memory/Code performance/Concepts/Vectorization

   Decide how you will check that you don't introduce bugs when making changes to this code.

6. Test that you haven't introduced bugs into the optimised version of the code