Course reader: *Scripts and functions, part I*

- *Scripts* are text files that hold code and comments. *Functions* are the same but are modular and typically not changed much.

- Comments are indicated by % sign, and all text after the % is not run. Use a lot of comments! Comments tell others (and yourself) what the code does, and what the code doesn't (yet) do.

- Scripts should start with your name and a brief description of the main function of the script.

- Functions start with a function line, definition of the function, inputs/outputs, and any other useful information (including your name).

- Soft-coding is much better than hard-coding.
  - *Soft-coding* means all of the settings and parameters are stored as variables and specified at the top of the script. This makes it easy to see the parameters and to change them.
  - *Hard-coded* means parameters are written into the code in numbers. Hard-coded parameters are hard to find/change, and can be forgotten or lead to confusion or errors.

- All parameters and soft-coded data should be at the top of the script so they are easy to find.

- Cells (not to be confused with cell arrays) are like paragraphs of code and are indicated using %%.

- Code in a script can be run one line at a time by highlighting and right-clicking or pressing F9. Entire cells can be run with Ctrl-Enter. The entire script can be run using F5 or *Editor* → *Run*.

- MATLAB can only run functions that are in its *path*. The path is a list of folders MATLAB searches for functions and data. You can view the path from *Home* → *Set Path*.
  - Changes made to the path via the GUI are **persistent** (meaning they remain on all future MATLAB sessions until explicitly changed)!
  - Don't remove folders unless you are sure you know what you are doing. You can really screw up MATLAB by accidentally removing critical folders.
  - Best practice is not to change the permanent path, but instead to add whatever extra paths you need in your script via `addpath(foldername);`

- Spacing within a line of code is usually optional, but highly recommended to increase readability. Consider the following two lines, which do exactly the same thing:

```
4*5+3/4-13^5
4*5 + 3/4 - 13^5
```

- Many unix functions work in the MATLAB command. The most important ones to know are:
  - `cd pathname`: Change the current directory (folder) to pathname.
  - `mkdir tmp`: Make a few folder called tmp.
  - `dir`: List the files/folders in the current folder. A related function is `what`, which lists only the MATLAB files.

- Successful programming and data analysis requires having clean and organized code. As you gain experience with programming, you will develop your own programming style, but there are a few broad-stroke ideas to keep in mind when writing code:
  1. A script should have **one** main purpose, just like a book has one main idea or a scientific report has one main focus. You should explain the main idea of the script in a few sentences at the top of the script.
  2. After this description, include your name or contact information, so people know who to credit (or who to contact).
  3. Following this analogy, each cell of code (indicated by %%) is like a paragraph. It's a single chunk of thought that can be described in a few words or one sentence max.
  4. Use comments!
  5. If there are known bugs, exceptions, or if you have to-do list, note them at the top of the script, not hidden deep inside the middle of the code where you are likely to forget about them.

Exercises

1. Write a function that will generate a matrix of size $m \times n$ containing random numbers (using the `randn` function), then multiply that matrix by another random number (using the `rand` function). The function needs m and n as inputs, and should output the matrix and the multiplying number.
2. Using the function `sort` instead of `min`, write a function that finds the minimum value of an input vector.
3. Adapt the previous function to find the value closest to zero, regardless of its sign.
4. Create a 2D matrix of a size you specify (soft-coded!) containing random integers between 0 and 10. Write code to determine the total number of elements in the matrix. There are many ways to solve this; you can consider using the functions `size` and `prod` or the function `numel`.
5. The *median* is the center of a distribution. Write a function to compute the median of a vector of input numbers. To find the median, you need to sort the data and then take the middle value. If there is an even number of elements, then you can take the smaller of the two (technically it should be the average of the two middle values, but that requires using control functions that we haven't yet discussed). How does the output of your function compare to the MATLAB `median` function?
6. Condense the following code down to one line.

```
N = 14;
randnums = rand(N,2);
rn_size  = size(randnums);
zeromat  = zeros(rn_size(1),1);
```

7. Find and fix the errors on the following lines of code.

```
mat  = ones( randn );
name = [ 'My name ' 4 ' today is ' "Mudd" ];
name = 'Sometimes I like the number ' 7;
asdf = [1:.2:4 3:-2/3:-1]; asdf(.2)
```

8. The following lines of code are correct but you can make them easier to interpret by adding spaces, parentheses, or new lines. Run the code with and without your aesthetic improvements to make sure you haven't changed the behavior.

```
var=linspace(1,19,13).^2*3;mvar=mean(var);disp(['The mean is ' num2str(mvar)]);
result=4-6/7*2^mean(randn(10,1));
```

9. The function `which` tells you the location of a file in the MATLAB path.
(1) Type `which mean` to see where the `mean` function is located. Then open that file using a non-MATLAB text editor like NotePad so you can see that it's just a regular text file (IMPORTANT: do not change or move this file!).
(2) This function can be used to determine whether it's OK to use a variable name. Let's say you want to create a variable called `mmean` but you suspect that it might be a function. Use the `which` function to decide whether the following names are acceptable variable names.

a) `mmean` **b)** `mean2` **c)** `nanmean` **d)** `fftmean` **e)** `meen`

Answers

1. Something like this:

```
function [mat,s] = random_matrix_mult(m,n)
% function [mat,s] = random_matrix_mult(m,n);
%   Given matrix sizes m and n, this function
%   returns a matrix of m-by-n random numbers
%   multiplied by another random number s.
%
%     INPUTS:  m   - number of columns
%              n   - number of rows
%     OUTPUTS: mat - matrix of normal random numbers
%              s   - random number multiplier
%
% function written by mikexcohen.com

% here there should be checks on the inputs
% but we haven't learned about that yet.
s   = rand;
mat = s*randn(m,n);
```

2. The previous answer includes the function line and help text. Below is just the 'meat' of the function.

```
sortvals = sort(inputvar,'ascend');
minval = sortvals(1);
```

3. Just the 'meat':

```
sortvals = sort( abs(inputvar) ,'ascend');
minval = sortvals(1);
```

4. One possible solution:

```
% create matrix
m = 4; % M rows
n = 6; % N columns

% matrix elements by scaling up to 10
% then rounding to get integers
mat = round(rand(m,n)*10);

% total number of elements (various methods)
totalN = m*n;
totalN = prod( size(mat) );
totalN = numel(mat);
```

5. First sort, then find the value of the number in the middle position.

```
N = 22;
var = randn(N,1);
varsort = sort(var);
halfpos = round(N/2);
median_val = varsort(halfpos);
```

6. Condensed:

```
zeromat = zeros(14,1);
```

7. When debugging code, you sometimes have to guess what the intended action is.

```
mat  = ones( 2 );
name = [ 'My name ' num2str(4) ' today is Mudd' ];
name = [ 'Sometimes I like the number ' num2str(7) ];
asdf = [1:.2:4 3:-2/3:-1]; asdf(2)
```

8. These aesthetic adjustments are subjective. I would reformat the first example like this:

```
var  = 3*linspace(1,19,13).^2;
mvar = mean(var);
disp([ 'The mean is ' num2str(mvar) ]);
```

9. .

a) `mmean`: OK **b)** `mean2`: is a function **c)** `nanmean`: is a function
d) `fftmean`: OK **e)** `meen`: OK