Topic: User-defined function, random numbers, review
Reading: CFile Chapter 4, Sec 4.1, 4.2

Example p: what will be printed?

```
% Script file      |    % Function file absolute.m
p= -3;             |    function q = absolute(p)
q= absolute(p);    |    if (p<0)
disp(p)            |       p= -p;
                   |    end
                   |    q= p;
```

User-Defined Function

- *Procedural abstraction*–can easily reuse code
- Functions can be independently tested easily
- Keep *driver* program clean by keeping detail code in separate functions
- Upon invocation, each function has its own memory space that is inaccessible by other functions or the command window space—variables in a function can be "seen" only inside the function
- Arguments are "passed by value"
- Values stored in variables are not preserved between function calls.

Subfunctions

- more than one function in an M-file
- top function is normal
- remaining functions are *subfunctions*, accessible only by top function

Global Memory

- Global memory can be accessed from any workspace
- Global variable must be declared to be global before it is used for the first time in a function.

  global *variable1 variable2 ...*

Note: In Project 2, we used several global variables for convenience in our programs. This indiscriminate use of global variables is *not* good! In general, we want a function to have *local* variables that are visible only within the function—protected from other functions. We used many global variables in Project 2 only because we have not covered data structure yet. In the future we will use better designs!

Persistent Memory

Persistent memory can be accessed from within the function only *and is preserved unchanged between calls to the function.*

persistent *variable1 variable2 ...*

A random example. I mean a random walk example!

Write a function that performs a "random walk." In a random walk, possible moves are left, right, up, or down (in a Cartesian plane). The input parameters are the number of steps, `n`, and the initial coordinates `x0`, `y0`. The function returns the final coordinates `xFinal`, `yFinal`.

```
% (xFinal,yFinal) is the location after n steps of random walk starting from (x0,y0).
% In each step, possible moves are left, right, up, or down.

fprintf('Do a random walk starting from (%f, %f)!\n', x0, y0)

% Perform n steps of random walk
```

```
fprintf('End up at (%f,%f) after %d steps\n', xFinal, yFinal, n)
```

Random number generator `rand`

MATLAB's built-in function **rand** generates a number in the range of 0 to 1 randomly. In other words, function **rand** generates a number from the standard *uniform* distribution: any number in the range of 0 to 1 is *equally likely to occur*. Note that the range is the open interval (0,1).

```
x= rand(1,1);      % one random number in (0,1)
x= 6*rand(1,1);    % one random number in (0,6)
x= 6*rand(1,1)+1;  % one random number in (1,7)
```

Do the expressions below each gives an *integer* in (1..6) with *equal* likelihood?

```
x= round(rand(1,1)*6)          x= ceil(rand(1,1)*6)
```

Review: developing algorithms

Develop an algorithm for calculating the *mode* of a sequence. The mode is the number in the sequence that occurs with maximum frequency. Assume that the sequence is (a) non-negative, (b) entered one by one and terminated by a negative number, and (c) entered in non-decreasing order. E.g., the mode of the sequence 87,92,92,98,98,98,100 is 98. Assume that only scalar variables are allowed.