Mini-Matlab Lesson 5: Functions and Loops

Writing Functions and Scripts

Contents

Relational and logical operators

```
clear;
close all;
help relop

% Logical data value is either 0 (for false) or 1 (for true)

p = 1;

p == 0
p ~= 0
p > 0
p < 0
isinteger(p)
```

```
 Relational operators.
  < > Relational operators.
      The six relational operators are <, <=, >, >=, ==, and ~=.
      A < B does element by element comparisons between A and B
      and returns a matrix of the same size with elements set to logical
      1 (TRUE) where the relation is true and elements set to logical 0
      (FALSE) where it is not.  A and B must have the same dimensions
      (or one can be a scalar).

  &   Element-wise Logical AND.
      A & B is a matrix whose elements are logical 1 (TRUE) where both A
      and B have non-zero elements, and logical 0 (FALSE) where either has
      a zero element.  A and B must have the same dimensions (or one can
      be a scalar).

  &&  Short-Circuit Logical AND.
      A && B is a scalar value that is the logical AND of scalar A and B.
      This is a "short-circuit" operation in that MATLAB evaluates B only
      if the result is not fully determined by A.  For example, if A equals
      0, then the entire expression evaluates to logical 0 (FALSE), regard-
      less of the value of B.  Under these circumstances, there is no need
      to evaluate B because the result is already known.

  |   Element-wise Logical OR.
```

```
      A | B is a matrix whose elements are logical 1 (TRUE) where either
      A or B has a non-zero element, and logical 0 (FALSE) where both have
      zero elements.  A and B must have the same dimensions (or one can
      be a scalar).

  ||  Short-Circuit Logical OR.
      A || B is a scalar value that is the logical OR of scalar A and B.
      This is a "short-circuit" operation in that MATLAB evaluates B only
      if the result is not fully determined by A.  For example, if A equals
      1, then the entire expression evaluates to logical 1 (TRUE), regard-
      less of the value of B.  Under these circumstances, there is no need
      to evaluate B because the result is already known.

  ~   Logical complement (NOT).
      ~A is a matrix whose elements are logical 1 (TRUE) where A has zero
      elements, and logical 0 (FALSE) where A has non-zero elements.

  xor Exclusive OR.
      xor(A,B) is logical 1 (TRUE) where either A or B, but not both, is
      non-zero.  See XOR.


ans =

     0


ans =

     1


ans =

     1


ans =

     0


ans =

     0
```

IF loops

Here is an example of an **IF** loop. Note that elseif and else statements are not required. Also note the **END** at the end.

```
p = 0;
if p < 0
    disp('p is less than zero');
elseif (p >= 0) & (p < 2)
```

```
    disp('p is between 0 and 2')
else
    disp('p is greater than or equal to 2');
end
```

```
p is between 0 and 2
```

FOR loops

FOR loops use a counter. It is good practice not to use 'i' for the counter, because Matlab also uses 'i' for imaginary numbers.

The **break** command breaks you out of the innermost loop

```
p = 0;
for j = 1:10
    p = p + 1;
    if p == 8
        break;
    end
end
p
```

```
p =

     8
```

WHILE loops

These loops keep on looping until a certain condition is met.

If you ever get stuck or want to kill a computation, press **CTRL-C**

```
p = 0;
while (p <= 5)
    p = p + 1;
    disp(['Current value of p = ', num2str(p)]);
end
```

```
Current value of p = 1
Current value of p = 2
Current value of p = 3
Current value of p = 4
Current value of p = 5
Current value of p = 6
```

Scripts and functions

Matlab functions and scripts are plain text files. Matlab will treat any file that ends in .m as either a function or a script.

A **script** is a list of commands to be run in some order. Placing these commands in a file that

ends in .m allows you to "run" the script by typing its name at the command line. The filename can be anything.

A **function** is capable of taking particular variables (called arguments) and doing something specific to "return" some particular type of result. The syntax for a function is

Defining and using functions

A function is typically defined in the following way:

function [return values] = functionname(arguments)

You **must** save the .m file as functionname.m and place it in the same directory as where you are calling the function. Alternatively, you can place it in one of the directories that Matlab searches. If you program for long enough, you will want to build your own directory of common functions, and then use the **addpath** command

One very important comment to make is about **variable scope**. A variable's **scope** is basically where (in the program/script) it can be accessed.

"MATLAB stores variables in a part of memory called a workspace. The base workspace holds variables created during your interactive MATLAB session and also any variables created by running scripts. *Functions do not use the base workspace. Every function has its own function workspace*.

To output the current workspace variables, use **whos**.

```
% Go and look at mysum.m

dat = 0;
disp(['Here, the value of dat = ', num2str(dat)]);
S1 = mysum(1, 2)

dat = 1;
disp(['Here, the value of dat = ', num2str(dat)]);
S2 = mysum(1)
```

```
Here, the value of dat = 0
Here, the value of dat = 99
  Name      Size            Bytes  Class     Attributes

  S         1x1                 8  double
  dat       1x1                 8  double
  x         1x1                 8  double
  y         1x1                 8  double


S1 =

     3

Here, the value of dat = 1
Here, the value of dat = 99
  Name      Size            Bytes  Class     Attributes

  S         1x1                 8  double
  dat       1x1                 8  double
  x         1x1                 8  double
```

```
  y         1x1                 8  double
```

```
S2 =

     1
```

Anonymous functions

Not only can you define functions in .m files, but you can define **in-line** functions. These are useful for mainly two purposes: (i) quick function definitions, or (ii) passing parameters. We already talked about (i) in the last few lessons.

Sometimes you want or need to call a function with only one input, but the function actually requires several parameters. This is **very helpful** and often necessary. A prototypical example is when MATLAB asks you to provide a function of N variables, but in order to define the function, you need to insert some parameters.

```
% Go and look at mysum2.m
x = [1 1 1];
mysum2(x, 2)

% Can we call the function with only one input?
func = @(x)mysum2(x, 2);
func(x)
```

```
ans =

     6


ans =

     6
```

Published with MATLAB® 7.12