Chapter 5

Functions

Outline

5.1 Concepts: Abstraction and Encapsulation

5.2 Black Box View of a Function

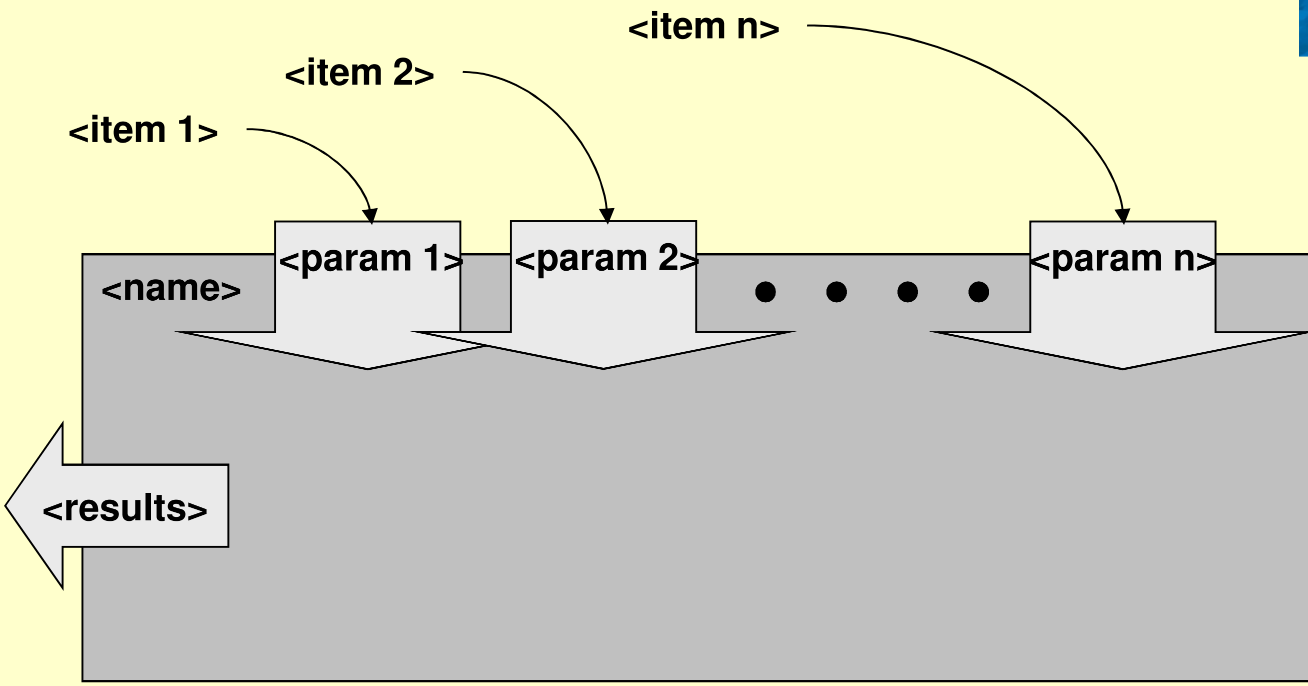
5.3 Matlab Implementation

5.1 Concepts: Abstraction and Encapsulation

When we define user functions, we will take advantage of two concepts:

- **Abstraction** – defining the body of the function as a code block to which we refer by name (the name of the function)
- **Encapsulation** – we also "hide" the code body from the code that invokes (calls) the function, and prevent the function's code from affecting the caller except by returning the results from the function.

5.2 Black Box View of a Function



- Name of function must be *function_name.m*
- Actual (incoming) parameters are copied into the function's workspace and named with the corresponding formal parameter names (<*param 1:n*>)
- Zero or more results can be computed and will be copied back to the caller's workspace.

5.3 Matlab Implementation

The general template for a MATLAB function is:

```
function <return info> <function name> (<parameters>)
      <documentation>
      <code body>
end
```

- The function must be saved in a file named <name>.m
- Helper functions (needed only by this function) may be included in the same file after the main function.
- MATLAB does not require the *end* key word if the function stands alone in its file.

5.3.1 General Template

•All documentation lines are printed to the Command Window when you type the following command:

```
>> help <function_name>
```

```
function volume = cylinder(height, radius)
% function to compute the volume of a cylinder
% volume = cylinder(height, radius)
    base = pi*radius^2
   volume = base*height
end
```

5.3.1 General Template

•If you save the cylinder.m in your Current Directory, and you type help on the function, MATLAB displays the purpose of the function (as defined by you)

```
>> help volume
function to compute the volume of a cylinder
volume = cylinder(height, radius)

>> cylinder(1, 1)
ans = 3.1416
>>
```

5.3.3 Storing and Using Functions

•All user-defined MATLAB functions must be created like scripts in an m-file

You may invoke the function by entering its name and parameters in:

- Command Window
- A script
- Other function definitions.

- If you do not specify an assignment for the results of the function call, it will be assigned to the variable *ans*.

5.3.4 Calling Functions

•When a function is defined, the user provides a list of the names expected by the function, called **formal parameters**.

•When the function is called, the caller must provide the same number of data values expected by the function, called **actual parameters**.

5.3.4 Calling Functions

- Actual parameters could be:

  Constants

  Variables that have been defined

  Result of some mathematical operation(s)

  Results returned from other functions

- Values are assigned to parameters by position in the calling statement and function definition.

- If return variables have been defined for the function, all variables must be assigned at the exit of the function.

5.3.6 Returning Multiple Results

- MATLAB provides the ability to returning more than one result from a function by name.

```
1.function [area, volume]=cylinder(height, radius)
2.    base=pi.*radius.^2;
3.    volume=base.*height;
4.    area=2*pi*radius.*height+2*base;
5.end
```

Line 1: Multiple results to be returned are specified as a "vector" of variable names

Line 2-4: Element-by-element operations

- The names of the variables where value are returned could be any legal name

5.3.8 Variable Scoping

- Variable scoping defines the places within the Command Window, MATLAB system, and m-files to which instructions have access:

  Global scope. When using the Command Window or running a script and you access the value of a variable, the system will reach into the current workspace and then into MATLAB system libraries to find its current value

  Local scope. When you run a function, its local variables <u>are not included in your current work space</u>, and it does not look into your current work space for values of variables

5.3.9 Global Variables

- There are occasions when it is very inefficient to pass large data sets into and out of a function.

- To avoid passing large amounts of data, we can use global variables using the key word *global*. We declare a variable to be global in both the calling space and the called function by placing the following line in the code:

```
global <my_variable>
```

- The function(s) will then be able to access and modify the values in the variable my_variable without having to pass it in and out as a parameter

5.3.9 Global Variables

- The following function fcn1.m can be used to store a value in the global variable *store*:

```
% fcn1.m
function fcn1( val )
    global store
    store = val
end
```

When run it, will produce the following output:

```
>> fcn1(20)
store =
    20
>>
```

5.3.9 Global Variables

- The variable store **does not** show on the Workspace!. To recall the value of store, we can use the following function:

```
% fcn2.m
function y = fcn2()
    global store
    y = store;
end
```

When run it, will produce the following output:

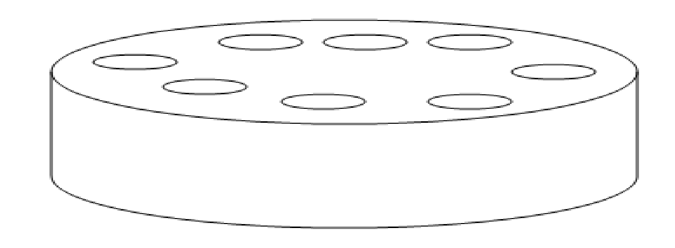
```
>> fcn2()
ans =
   20
>>
```

Let’s Write some Code …

5.4 Engineering Example – Measuring a Solid Object

- Problem:

Consider the disk shown below



- It has a radius R, height h, and eight cylindrical holes each of radius r bored in it.

- During the process of designing this part, we need to know the weight and amount of paint required to finish it: so we need the volume and area of this disk.

5.4 Engineering Example – Measuring a Solid Object

```
clear; clc; format bank;

h=1:5;  % set a range of disk thiknesses
R=35;   % set radius of large disk
r=3;    % set radius of small disks

[Area Vol]=cylinder(h,R) % dimensions of large disk
[area vol]=cylinder(h,r) % dimensions of the hole

% the total volume is the vol. of large disk minus
% dimensons of 8 holes
Vol=Vol-8*vol

% the area to be painted is computed as follows:
% Area of large disk plus the area of 8 disks minus
% the two areas of the top and bottom of the holes
Area=Area+8*(area-2*(2*pi*r.^2))
```

```
function [area,volume]=cylinder(height,radius)
    % function to compute the volume of a cylinder
    % volume = cylinder(height, radius)
    base=pi.*radius.^2;
    volume=base.*height;
    area=2*pi*radius.*height+2*base;
end
```

5.4 Engineering Example – Measuring a Solid Object

```
Vol =

     3622.26      7244.51     10866.77     14489.03     18111.28


Area =

     7615.22      7985.93      8356.64      8727.34      9098.05

>>
```

Homework on Chapter 5 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due in a week