
CS1371

Introduction to Computing for Engineers

Scripts and Functions

1

Matlab Scripts and Functions

Objectives

- Matlab scripts
- Basics of a Matlab function
- Workspaces in Matlab
- Using functions in program design (functional programming)

Topics

- Scripts
- How to design a function
- Function workspace
- Encapsulation
- Using functions
- Examples

2

Scripts and Functions

- Matlab's Command prompt is where you can enter commands to be executed immediately...
 - *Just like a calculator*
 - *You can see what you've done but it must be re-entered at the command prompt to be recalculated*
 - *Only the results (variables) are retained in the Matlab workspace (diary on will log commands and results to a diary file; diary off disables this)*

```
>> length=5.5;
>> radius=2.25;
>> volume=pi.*radius^2.*length

volume =
    87.4737
```

- *What if you want to enter different values for radius or length?*

3

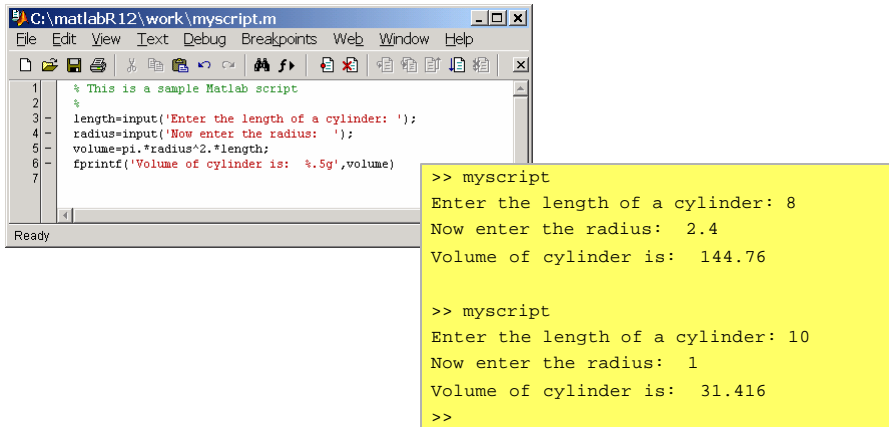
Matlab Scripts

- Matlab **scripts** are the solution to this problem!
 - *You can create a "script" that can be repeatedly executed*
 - *This is the basic Matlab "program"*
- Scripts are simply text files containing Matlab statements
 - *You can use any text editor but the built-in editor indents and uses color to highlight the language syntax*
 - *Script files always have the ".m" extension, e.g., m-files*
- When a script (m-file) is executed, it is simply read sequentially and each line is presented to the Matlab command prompt just like it was typed by hand
 - *Speed and repeatability are key features*
 - *Matlab control & loop statements (e.g., if, for, while...) can be executed in this way*

4

Example Script in an m-file

- Use File/New/M-file to start Matlab editor
- Save file with .m extension in directory in Matlab's path
- Type m-file name at prompt to execute



```
C:\matlabR12\work\myscript.m
File Edit View Text Debug Breakpoints Web Window Help
1 % This is a sample Matlab script
2 %
3 length=input('Enter the length of a cylinder: ');
4 radius=input('Now enter the radius: ');
5 volume=pi.*radius^2.*length;
6 fprintf('Volume of cylinder is: %.5g',volume)
7

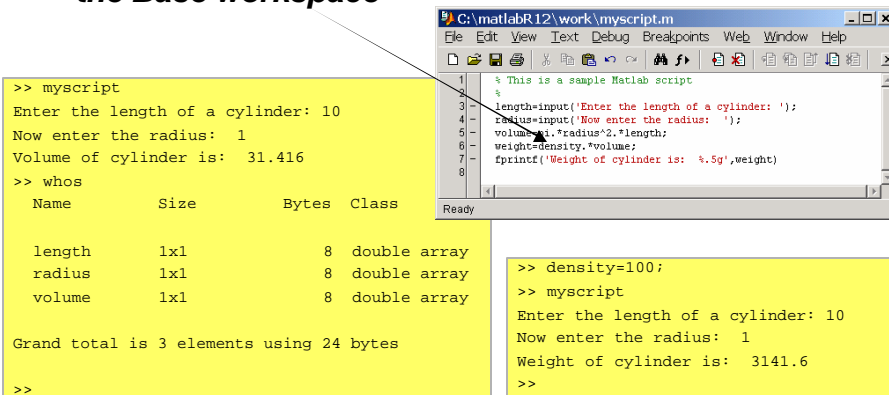
>> myscript
Enter the length of a cylinder: 8
Now enter the radius: 2.4
Volume of cylinder is: 144.76

>> myscript
Enter the length of a cylinder: 10
Now enter the radius: 1
Volume of cylinder is: 31.416
>>
```

5

Script Workspace

- When commands are executed, the results are left in the Matlab **BASE workspace**.
 - *whos* will list all the current variables in the workspace
 - Scripts can make use of variables already defined in the Base workspace



```
C:\matlabR12\work\myscript.m
File Edit View Text Debug Breakpoints Web Window Help
1 % This is a sample Matlab script
2 %
3 length=input('Enter the length of a cylinder: ');
4 radius=input('Now enter the radius: ');
5 volume=pi.*radius^2.*length;
6 weight=density.*volume;
7 fprintf('Weight of cylinder is: %.5g',weight)
8

>> myscript
Enter the length of a cylinder: 10
Now enter the radius: 1
Volume of cylinder is: 31.416
>> whos
Name          Size          Bytes  Class
-----
length        1x1            8  double array
radius        1x1            8  double array
volume        1x1            8  double array
Grand total is 3 elements using 24 bytes
>>

>> density=100;
>> myscript
Enter the length of a cylinder: 10
Now enter the radius: 1
Weight of cylinder is: 3141.6
>>
```

6

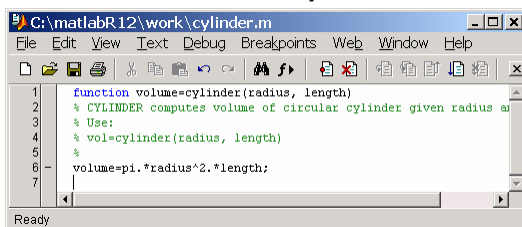
More Scripts...

- First, try out statements individually in Command window...
- Use scripts to collect together statements that can be used to solve more complicated problems.
- Scripts can call other scripts
 - Can “chain” together individual small programs
 - Each script can be tested and debugged separately
- PROBLEMS:
 - All scripts share same Base workspace
 - Variables can be confused and mis-used (may accidentally overwrite a previously defined variable)
 - Can’t localize results (create individual workspaces)
 - In other words, we can’t **encapsulate** variables and constants inside a script where they are **hidden** from other code.

7

Matlab Functions (m-functions)

- M-functions are like “functions” in algebra
 - **Algebra**: function is a rule that assigns a value based on specified values of the function arguments
 - **Matlab**: function is a program module that computes a returned variable based on specified values of the function arguments



```
C:\matlabR12\work\cylinder.m
File Edit View Text Debug Breakpoints Web Window Help
function volume=cylinder(radius, length)
% CYLINDER computes volume of circular cylinder given radius and length
% Use:
% vol=cylinder(radius, length)
%
volume=pi.*radius^2.*length;
```

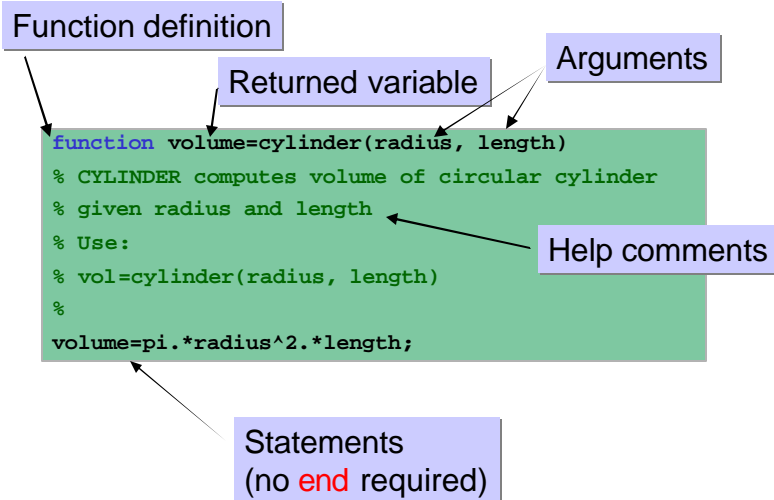
```
>> cylinder(1,10)
```

```
ans =
```

```
31.4159
```

8

M-function Structure



NOTE: function names are NOT case sensitive in Windows

9

Using M-File Editor to Create a Function

- From Command menu choose File/New/M-file
 - new edit window will appear with "Untitled"
 - enter code for your new function
 - when saving, make sure file name is same as function name (the **file name** is what Matlab uses to identify your m-functions)

```
C:\matlabR12\work\cylinder.m*
File Edit View Text Debug Breakpoints Web Window Help
[Icons] Stack: BASE
1 function volume=cylinder(radius, length)
2 % CYLINDER computes volume of circular cylinder
3 % given radius and length
4 % Use:
5 % vol=cylinder(radius, length)
6 %
7 volume=pi.*radius^2.*length;
```

On Your Own:

Figure out what all the options in the **File, Edit, View & Text** menus do...

10

Using Comments in M-functions

- Comments immediately following function statement will be listed by the Matlab [Help](#) function
 - *Place comments that define what your function does*
 - *Include comments to illustrate how to use function*
 - *Include version, date and author*

```
>> help cylinder

CYLINDER computes volume of circular cylinder given radius and length
Use:
vol=cylinder(radius, length)

>>
```

11

M-function Workspace

- Each time an m-function is executed, a new workspace is created just for that instance
 - *All variables except arguments and returned variable are defined only in the [function workspace](#)*
 - *Function variables are not defined in Base workspace, nor are Base variables defined in function workspace*
 - *Function workspace is destroyed when function completes*
- Function variables are “hidden” from world (and from accidental misuse or alteration)

```
>> cylinder(1,10)
ans =
    31.4159

>> whos
  Name      Size      Bytes  Class
  ans       1x1         8      double array
Grand total is 1 elements using 8 bytes
```

12

M-function Workspace-cont'd

- You can even define other M-functions inside the first M-function!
 - These are called “*sub-functions*”
 - Each has its own local function workspace independent of any other function workspace
 - These workspaces are created and destroyed with each function invocation (nothing persists between invocations)
 - Sub-functions are available only inside the primary function and can be used in other sub-functions
 - **Convention:** begin sub-functions with `local_myfun(...)`
- **On your own:** find out about Matlab’s “*private*” functions

13

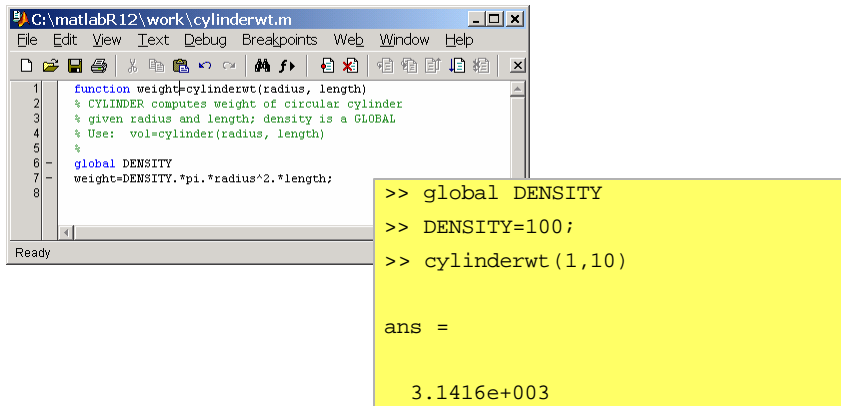
global Variables

- Sometimes it is awkward to provide ALL variables to function in arguments
 - “Fixed” variables are a good case
 - These are often called “parameters”
- Consider a new variable “density” in our examples
 - Suppose it doesn’t change for all of our examples
 - Why have to enter it as a third argument each time?
- Solution: `global` variables
 - Declaring a variable as `global` in a workspace means that it may be accessed in another workspace if it is also declared as a `global` variable in that workspace too.
 - Convention suggests using **UPPER CASE** to delineate all `global` variables
 - Don’t use `global` variables unless all else fails... they can cause a myriad of problems and are considered dangerous!

14

Example of the `global` Statement

- `DENSITY` is shared in both the Base and the function workspaces
- Any function that includes `DENSITY` in its global statement will also share this variable



```
C:\matlabR12\work\cylinderwt.m
File Edit View Text Debug Breakpoints Web Window Help
1 function weight=cylinderwt(radius, length)
2 % CYLINDER computes weight of circular cylinder
3 % given radius and length; density is a GLOBAL
4 % Use: vol=cylinder(radius, length)
5 %
6 global DENSITY
7 weight=DENSITY.*pi.*radius^2.*length;
8

Ready

>> global DENSITY
>> DENSITY=100;
>> cylinderwt(1,10)

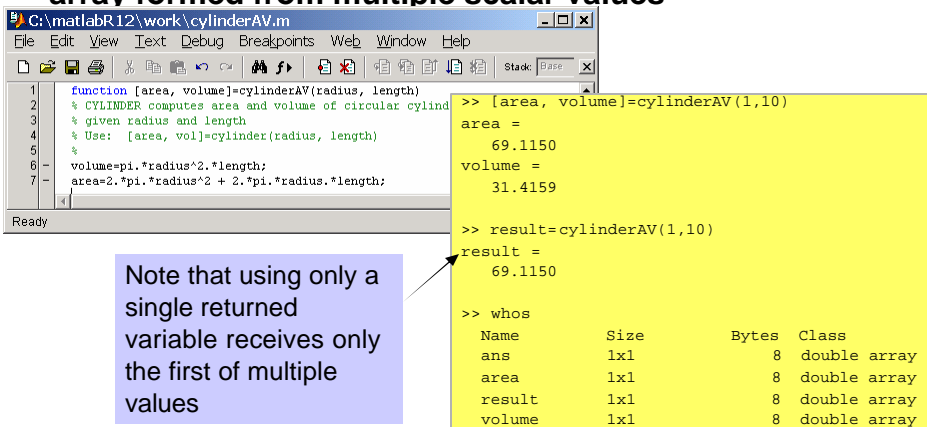
ans =

3.1416e+003
```

15

M-functions Can Return Multiple Values

- The returned variable from an M-function can be a scalar or an array (even a cell array)
- M-functions can use the `[]` constructor to return an array formed from multiple scalar values



```
C:\matlabR12\work\cylinderAV.m
File Edit View Text Debug Breakpoints Web Window Help
1 function [area, volume]=cylinderAV(radius, length)
2 % CYLINDER computes area and volume of circular cylinder
3 % given radius and length
4 % Use: [area, vol]=cylinder(radius, length)
5 %
6 volume=pi.*radius^2.*length;
7 area=2.*pi.*radius^2 + 2.*pi.*radius.*length;

Ready

>> [area, volume]=cylinderAV(1,10)
area =
69.1150
volume =
31.4159

>> result=cylinderAV(1,10)
result =
69.1150

>> whos
Name      Size      Bytes  Class
ans       1x1        8  double array
area      1x1        8  double array
result    1x1        8  double array
volume    1x1        8  double array
```

Note that using only a single returned variable receives only the first of multiple values

16

M-functions are the Core of Matlab

- Matlab scripts must be processed line by line by the command processor: **TIME CONSUMING, SLOW!**
- M-functions are compiled into “**p-code**” the first time they are invoked and this p-code is then executed
 - *This is MUCH faster than command interpretation*
 - *Editing an M-function will discard the old p-code*
 - *Repeated function executions are much faster this way!*
- You can force-compile an M-function into p-code and save the p-code in a “p-file”
 - *`pcode myfunction` creates `myfunction.p`*
 - *The p-code can be distributed just like an M-function*
 - *Users cannot easily decode your function*
- **Learn to make use of M-functions!**

17

More M-function Details

- If you encounter an error condition in a function:
 - *Executing the `error('...')` function will exit the function and terminate your script in the command window*
 - *A less traumatic solution is the `warning('...')` function which is similar but continues execution in the function*
- M-functions can be invoked using fewer arguments than specified in the function definition
 - *Use the `nargin` function to determine the actual number of input arguments supplied in a given invocation of the function*
 - *Your function must then predefine those arguments that are not supplied by the invocation*
 - *The `nargout` function is similar for output (return) arguments*

18

Example M-function

```
function tic
%TIC Start a stopwatch timer.
% The sequence of commands
%   TIC, operation, TOC
% prints the number of seconds required for the operation.
%
% See also TOC, CLOCK, ETIME, CPUTIME.
% Copyright 1984-2000 The MathWorks, Inc.
% $Revision: 5.8 $ $Date: 2000/06/01 16:09:47 $
% TIC simply stores CLOCK in a global variable.
global TICTOC
TICTOC = clock;
```

```
function t = toc
%TOC Read the stopwatch timer.
% TOC, by itself, prints the elapsed
% time (in seconds) since TIC was used.
% t = TOC; saves the elapsed time in t,
% instead of printing it out.
%
% See also TIC, ETIME, CLOCK, CPUTIME.
% Copyright 1984-2000 The MathWorks, Inc.
% $Revision: 5.9 $ $Date: 2000/06/01 16:09:47 $
% TOC uses ETIME and the value of CLOCK
% saved by TIC.
global TICTOC
if isempty(TICTOC)
    error('You must call TIC before calling TOC.');
```

```
end
if nargin < 1
    elapsed_time = etime(clock,TICTOC)
else
    t = etime(clock,TICTOC);
end
```

19

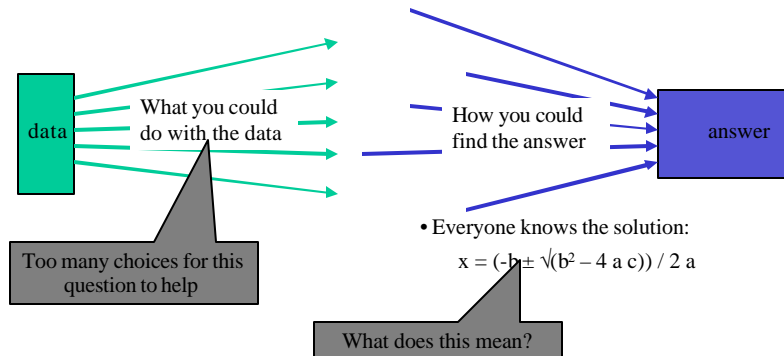
How are M-Functions Located?

- Sometimes it is important to know how Matlab will look for an m-function, especially if two have the same name...
- Suppose Matlab encounters a name, `velocity`
 - check to see if `velocity` is a variable in current workspace?
 - is `velocity` a built-in Matlab function?
 - is `velocity` a sub-function in the current function?
 - is `velocity.p` (and then `velocity.m`) a private function?
 - is `velocity.p` (and then `velocity.m`) in the current directory?
 - is `velocity.p` (and then `velocity.m`) in the Matlab path, searching from the current directory down?
- The first instance is used and any others are ignored
 - This can cause problems if you don't understand...
 - It can also let you replace functions with different versions
 - It is not quite like "overloading" in object-oriented languages...

20

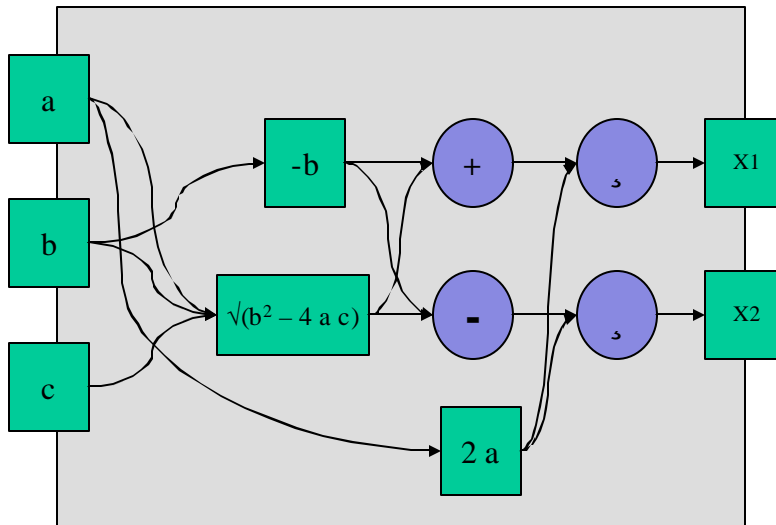
Problem Solving

- Finding the Roots of a Quadratic Equation
- Given coefficients a, b and c of the equation:
$$a x^2 + b x + c = 0$$
- Find the two values of x satisfying these equations



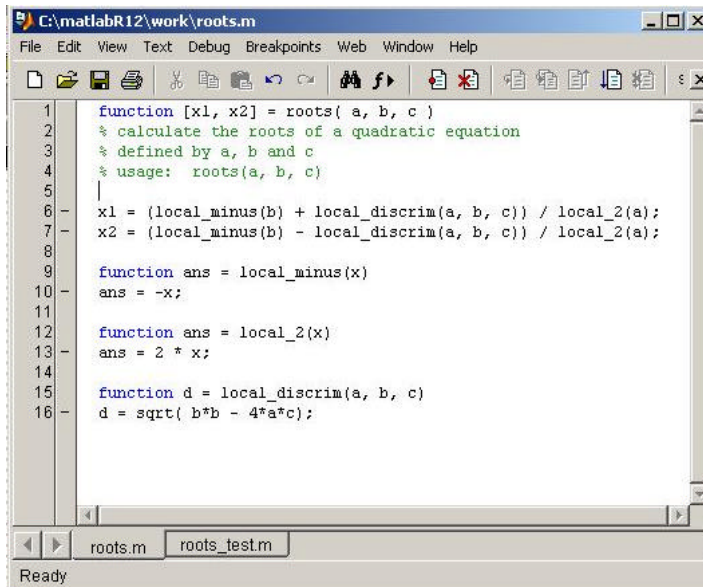
21

Solution Diagram



22

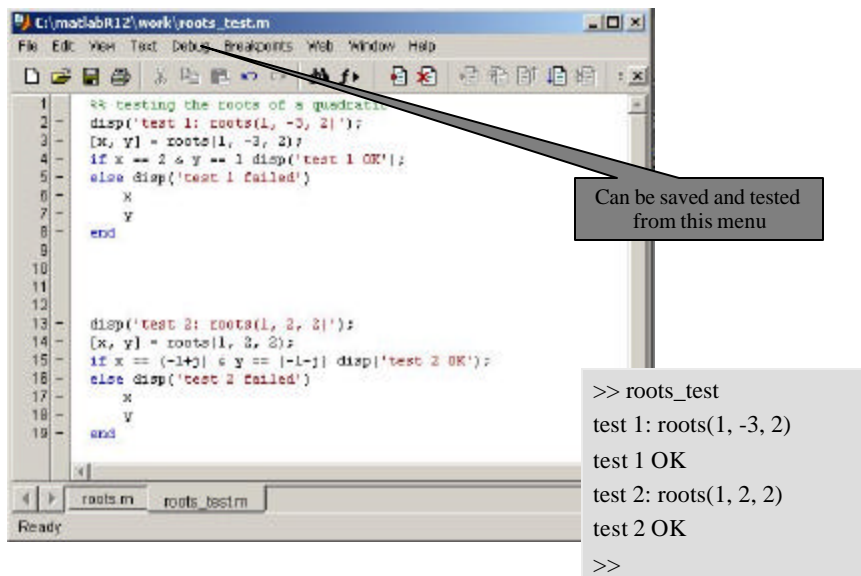
Code the Functions



```
1 function [x1, x2] = roots( a, b, c )
2 % calculate the roots of a quadratic equation
3 % defined by a, b and c
4 % usage: roots(a, b, c)
5 |
6 - x1 = (local_minus(b) + local_discrim(a, b, c)) / local_2(a);
7 - x2 = (local_minus(b) - local_discrim(a, b, c)) / local_2(a);
8
9 function ans = local_minus(x)
10 - ans = -x;
11
12 function ans = local_2(x)
13 - ans = 2 * x;
14
15 function d = local_discrim(a, b, c)
16 - d = sqrt( b*b - 4*a*c);
```

23

Write a Test Script



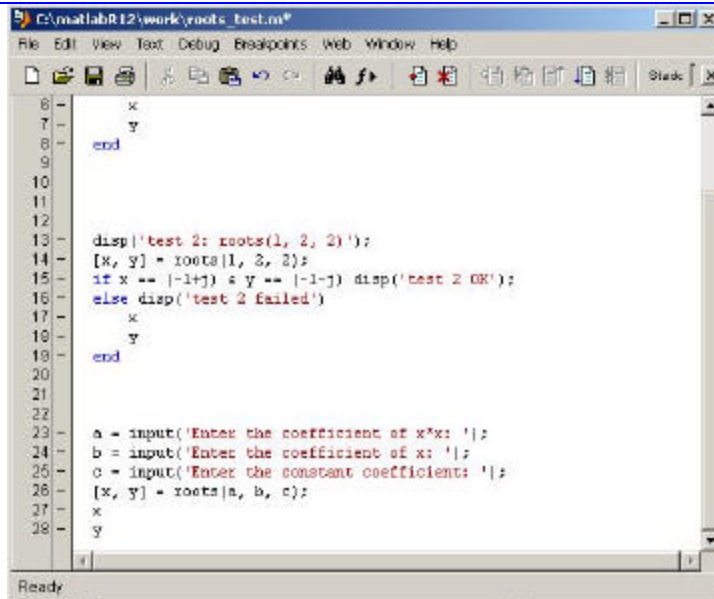
```
1 %% testing the roots of a quadratic
2 disp('test 1: roots(1, -3, 2)');
3 [x, y] = roots(1, -3, 2);
4 if x == 2 & y == 1 disp('test 1 OK');
5 else disp('test 1 failed')
6     x
7     y
8 end
9
10
11
12
13 disp('test 2: roots(1, 2, 2)');
14 [x, y] = roots(1, 2, 2);
15 if x == (-1+1j) & y == (-1-1j) disp('test 2 OK');
16 else disp('test 2 failed')
17     x
18     y
19 end
```

Can be saved and tested from this menu

```
>> roots_test
test 1: roots(1, -3, 2)
test 1 OK
test 2: roots(1, 2, 2)
test 2 OK
>>
```

24

Extend the Tests as Necessary



```
6 - x
7 - y
8 - end
9
10
11
12
13 - disp('test 2: roots(1, 2, 2)');
14 - [x, y] = roots(1, 2, 2);
15 - if x == |-1+j| & y == |-1-j| disp('test 2 OK');
16 - else disp('test 2 failed')
17 - x
18 - y
19 - end
20
21
22
23 - a = input('Enter the coefficient of x^2: ');
24 - b = input('Enter the coefficient of x: ');
25 - c = input('Enter the constant coefficient: ');
26 - [x, y] = roots(a, b, c);
27 - x
28 - y
```

25

Understanding Functions

- **Abstract Data Types**
- **Abstraction**
- **Illustration – roots of a quadratic**

26

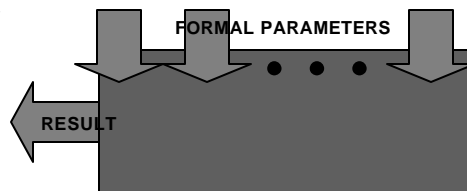
Abstract Data Types

- Abstract Data Types (ADTs) allow us to think about logical constructs without being concerned about the implementation.
- Usually expressed in a language-independent way:
 - *Simple English sentences*
 - *Graphical pictures*

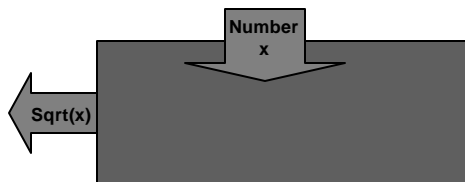
27

Example: Function definition

- In words:
 - *A function is a modular component which accepts zero or more parameters and returns a single construct*
 - *e.g the sqrt function takes a number and returns its square root.*
- Graphically:



- e.g sqrt:



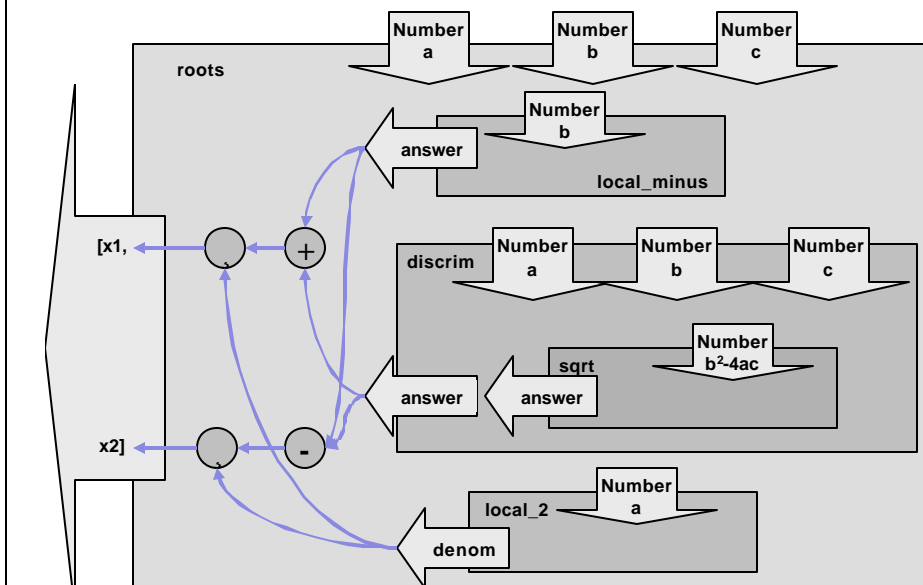
28

A note about data types

- Untyped languages does not require you to specify the type of the data provided to a function
- However, to program the function correctly, it is necessary to be aware of the type of the data provided to the function in the parameter(s) if any.
- Similarly, in order to use the function effectively, one must specify the type of data returned by it.

29

Abstraction Example



30

Questions?