USER DEFINED FUNCTIONS

Functions in MATLAB

- The MATLAB programming language is built around functions. A *function* is a piece of computer code that accepts an input argument from the user and provides output to the program.
- Functions allow us to program efficiently, enabling us to avoid rewriting the computer code for calculations that are performed frequently.

- For example most computer programs contain a function that calculates the sine of a number.
- In MATLAB, *sin* is the function name used to call up a series of commands that perform the necessary calculations.
- The user needs to provide an angle, and MATLAB returns a result.
- It isn't necessary for the programmer to know how MATLAB calculates the value of *sin(x)*.

User Defined Functions

- We may wish to define our own functions – those are used commonly in our programming.
- User-defined functions are stored as *m-files* and can be accessed by MATLAB, if they are in the current folder or on MATLAB's search path.

- Both built-in MATLAB functions and user-defined MATLAB functions have the same structure. Each consists of a name, user-provided input, and calculated output.
- For example, the function *cos(x),* is named *cos*, takes the user input inside the parentheses (in this case, *x*), and calculates a result.
- The user does not see the calculations performed, but just accepts the answer.

Structure of a Function

- User-defined functions work the same way. Imagine that you have created a function called my_function.
- Using `my_function(x)` in a program or from the command window will return a result, as long as `x` is defined and the logic in the function definition works.

- User-defined functions are created in m-files.
- Each must start with a function definition line that contains:
  - The word function,
  - A variable that defines the function output,
  - A function name,
  - A variable used for the input argument.

Structure of a Function, ctd.

- A function is a box,
- It hides the code and its workspace and communicates with the “world” using the input and output variables

Functions in m-file Structure

- A very simple MATLAB function that calculates the value of a particular polynomial:

```
function output = mypoly(x)
%This function calculates
%the value of a third-order polynomial
output = 3*x.^3 + 5*x.^2 - 2*x +1;
```

- The function name is `mypoly`, the input argument is `x`, and the output variable is named `output` .
- Before this function can be used, it **must be saved into the current folder**. The file name ***must be the same*** as the function name in order for MATLAB to find it.

Functions in m-file Structure, ctd.

- All of the MATLAB naming conventions we learned for naming variables apply to naming user-defined functions.

  In particular,
- The function name must start with a letter.
- It can consist of letters, numbers, and the underscore.
- Reserved names cannot be used.
- Any length is allowed, although long names are not good programming practice.

- Once the m-file has been saved, the function is available for use from the command window, from a script m-file, or from another function.
- You cannot execute a function m-file directly from the m-file itself.
- This makes sense, since the input parameters have not been defined until you call the function from the command window or a script m-file.

Functions in m-file Structure, ctd.

- Consider the poly function just created. If, in the command window, we type `mypoly(4)` then MATLAB responds with

```
ans =
265
```

- If we set `a` equal to 4 and use `a` as the input argument, we get the same result:

```
a = 4;
mypoly(a)
ans =
265
```

Example

- A Matlab function `sumOfNumbers.m`

```
function result = sumOfNumbers(a,b)

result = a+b;
```

The input variable(s)

output variable(s)

- We can use [ ]: if there are more than one output variables ([out1 out2]).

Example, ctd.

- Assume we wrote the function:

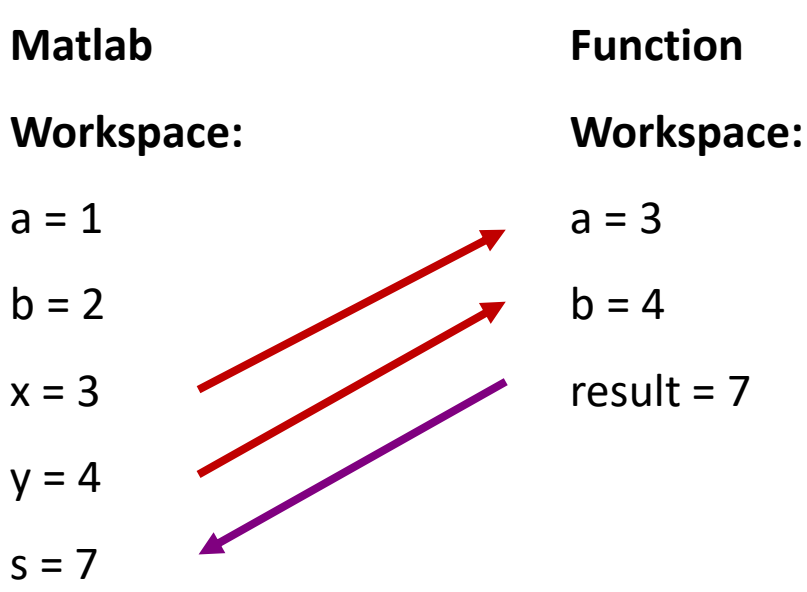
```
function result = sumOfNumbers(a,b)
result = a+b;
```

and in the workspace we run:

```
a = 1;
b = 2;
x = 3;
y = 4;
s = sumOfNumbers(x, y)
```

What is the output?

```
s = 7
```



Example-2

- Write and test a function **degToRad** that changes degrees to radians and another function **radToDeg** that changes radians to degrees. The functions should be able to accept both scalar and matrix input.

1. State the Problem

Create and test two functions, **degToRad** and **radToDeg** , to change degrees to radians and radians to degrees

2. Describe the Input and Output

Input	A vector of degree values
	A vector of radian values
Output	A table converting degrees to radians
	A table converting radians to degrees

3. Develop a Hand Example

*degrees = radians * 180/pi*

*radians = degrees * pi/180*

Example-2, ctd.

4. Develop a MATLAB Solution

```
function y = degToRad(x)
y = x*pi/180;
```

```
function deg = radToDeg(rad)
deg = rad*180/pi;
```

5. Test the solution

```
x = 0:45:180;
in_rad = degToRad(x)
in_rad =
   0    0.7854    1.5708    2.3562    3.1416
radToDeg(in_rad)
ans =
     0    45    90   135   180
```

Comments & Help for Functions

In a MATLAB function, the comments on the line immediately following the very first line serve a special role. These lines are returned when the help function is queried from the command window.

Consider, for example, the following function:

```
function results = myfunc(x)
%This function converts seconds to minutes
results = x./60;
```

- Querying the help function from the command window

```
help myfunc
```

- returns

```
This function converts seconds to minutes
```

Multi Input and Output Functions

```
function [x,v,a] = motion(t,x0)
% This function calculates the lateral position (x),
% velocity (v), and acceleration (a) of a mass attached
% to a spring for a given value of time (t) and initial
% position (x0) assuming the velocity is initially 0.
x = x0.*cos(t);
v = -x0.*sin(t);
a = -x0.*cos(t);
```

We can use the function to find values of position, velocity, and acceleration of the mass at specified times:

```
[x,v,a] = motion(3,1)
x =
   -0.9900
v =
   -0.1411
a =
    0.9900
```

Multi Input and Output Func's, ctd.

- If we call the motion function without specifying all three outputs, only the first output will be returned:

```
motion(3,1)
ans =
   -0.9900
```

- Using a vector of time values from 0 to 3 in the motion function returns three row vectors of answers:

```
time=0:1:3;
[x,v,a] = motion(time,1)
x =
    1.0000    0.5403   -0.4161   -0.9900
v =
         0   -0.8415   -0.9093   -0.1411
a =
   -1.0000   -0.5403    0.4161    0.9900
```

Some Useful Commands for Func's

Command	Description
nargin	Number of function input arguments
nargout	Number of function output arguments
nargchk()	Validate number of input arguments
error()	Display message and abort function
warning()	Display descriptive warning message

Examples:

```
nargin('motion‘)
ans =
     2
nargout('motion')
ans =
     3
```

Local Variables

- The variables used in function m-files are known as *local variables* . The only way a function can communicate with the workspace is through input arguments and the output it returns.
- Any variables defined within the function exist only for the function to use. For example, consider the *g* function:

```
function output = g(x,y)
% This function multiplies x and y together
% x and y must be the same size matrices
a = x .*y;
output = a;
```

- The variables *a, x, y,* and *output* are local variables. They can be used for additional calculations inside the *g* function, but they are not stored in the workspace.

Global Variables

- Unlike local variables, global variables are available to all parts of a computer program. In general, *it is a bad idea* to define global variables. However, MATLAB protects users from unintentionally using a global variable by requiring that it be identified both in the command-window environment (or in a script m-file) and in the function that will use it. Consider the distance function:

```
function result = distance(t)
%This function calculates the distance a falling
% object travels due to gravity
global G
result = 1/2*G*t.^2;
```

- The global command alerts the function to look in the workspace for the value of `G`. `G` must also have been defined in the command window (or script m-file) as a global variable:

```
global G
G = 9.8;
```

- This approach allows you to change the value of `G` without needing to redefine the distance function or providing the value of `G` as an input argument to the distance function.

Example-3

- Factorial calculation of a scalar in a function:

```
function y = fact(x)
if length(x)~=1 || x < 0
    error('You entered an invalid number');
end
f = 1;
if x>0
    for i=1:x
        f = f * i;
    end
else
    f = 1;
end
y=f;
```

Example-4

- Calculation the roots of a quadratic polynomial:

```
function [x1 x2] = findroots(a,b,c)

% This function computes the roots of a
% quadratic polynomial whose coefficients
% a,b and c are given (ax^2+bx+c=0).

delta = b^2-4*a*c;
x1 = (-b+sqrt(delta))/(2*a);
x2 = (-b-sqrt(delta))/(2*a);
```

Subfunctions

- A function M-file may contain the code for more than one function.
- The first function in a file is the *primary function,* and is the one invoked with the M-file name.
- Additional functions in the file are called *subfunctions,* and are visible only to the primary function and to other subfunctions.
- Each *subfunction* begins with its own function definition line.
- Subfunctions follow each other in any order *after* the primary function.

Subfunctions - Example

```
function [x2 x3] = xsc(x)
% Calculation of the square and the cube of the entered
scalar
x2 = sq(x);
x3 = cube(x);

function x2 = sq(x)
x2 = x*x;

function y = cube(a)
y = sq(a)*a;
```

```
>> [sq cu] = xsc(5)
sq =
    25
cu =
   125
```