This lecture covers the following topics:

- Matlab scripts (programs)
- Input-output: The `input` and `disp` commands
- The `fprintf` command
- User-defined functions

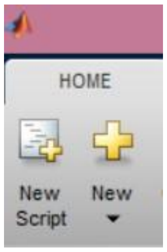
Matlab Scripts (Programs)

A script can be as simple as a series of Matlab commands that is saved in a file. Matlab gives the extension `.m` to the script file name. Therefore, scripts are also known as *M-files*.

Scripts are actually programs that you may want to execute in response to new input. A script is executed by typing its name (without the `.m`) at the" >>" prompt and pressing the return key.

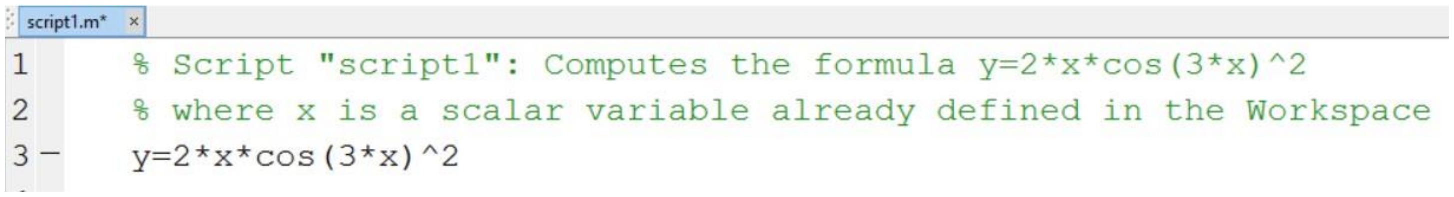
It should be noted that scripts can access any defined variable in the Matlab Workspace. Also, any variable that is modified or defined inside a script affects the Workspace variable directly. These features are important distinction between *scripts* and *functions*, as you will see later in this lecture.

To create a script, just click on the icon "New Script"

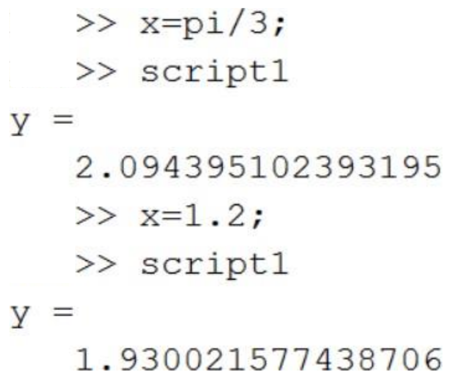


This will open the Editor Window. After creating your script, you may select the "save as" option and give a name to your script (e.g., `script1`). The script will show inside the Current Folder Window.

Example of a simple script:

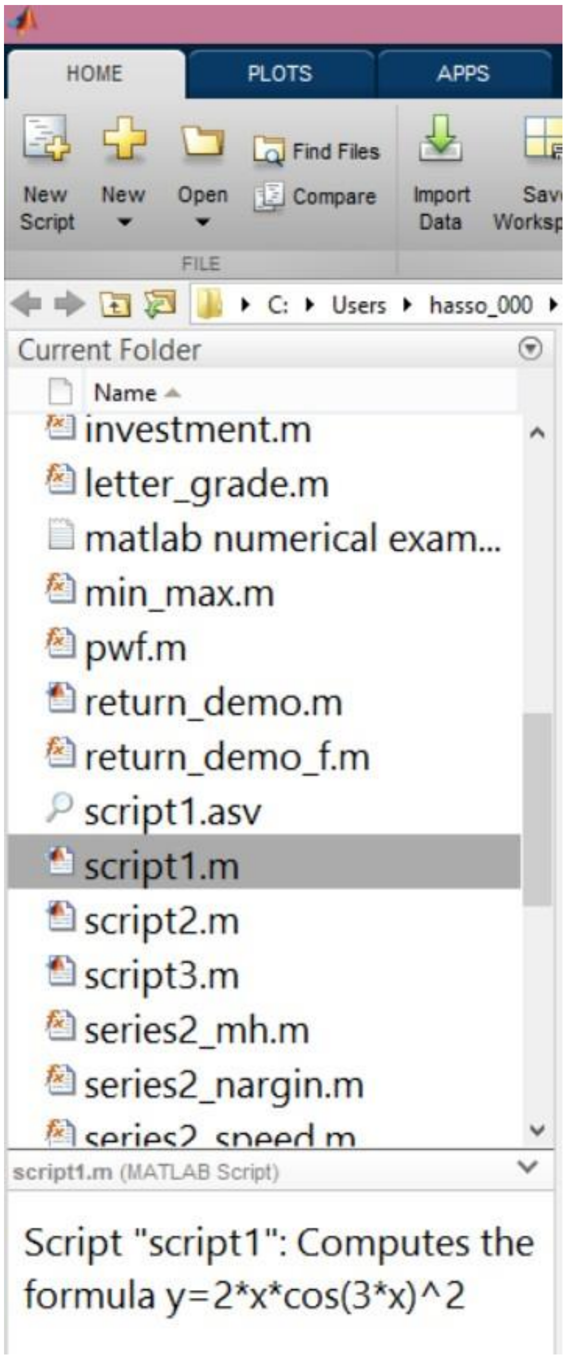


```
% Script "script1": Computes the formula y=2*x*cos(3*x)^2
% where x is a scalar variable already defined in the Workspace
y=2*x*cos(3*x)^2
```



```
   >> x=pi/3;
   >> script1
y =
   2.094395102393195
   >> x=1.2;
   >> script1
y =
   1.930021577438706
```

Left clicking once on a script file (located inside the Current Folder Window) will display in the File Information Window (at the lower left corner) the first comment line encountered in the script:



Script "script1": Computes the formula y=2*x*cos(3*x)^2

Input-Output: The input and disp Commands

The input **Command**. This Matlab command prompts the user for values (inputs) directly from the Command Window. Its syntax is

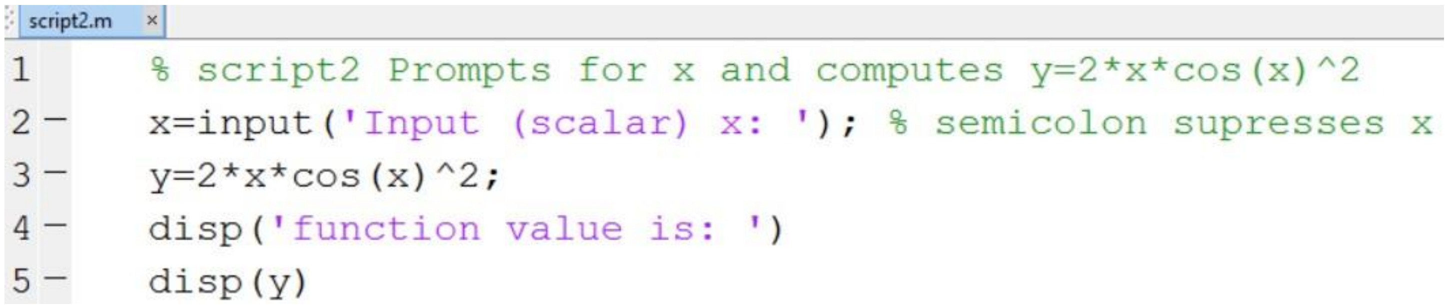
```
x = input('prompt string');
```

The disp **Command.** This Matlab command provides a simple way to display a value or a message (but not both) on a single line. The syntax is

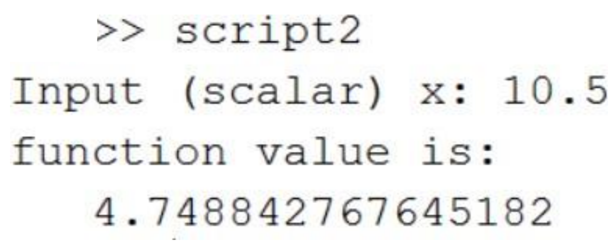
disp(value) or disp('text message')

The use of the input and disp commands are illustrated in the following examples.

Example: Write an interactive script for evaluating the scalar function $y(x) = 2x[\cos(x)]^2$. The script should prompt the user for the input x.

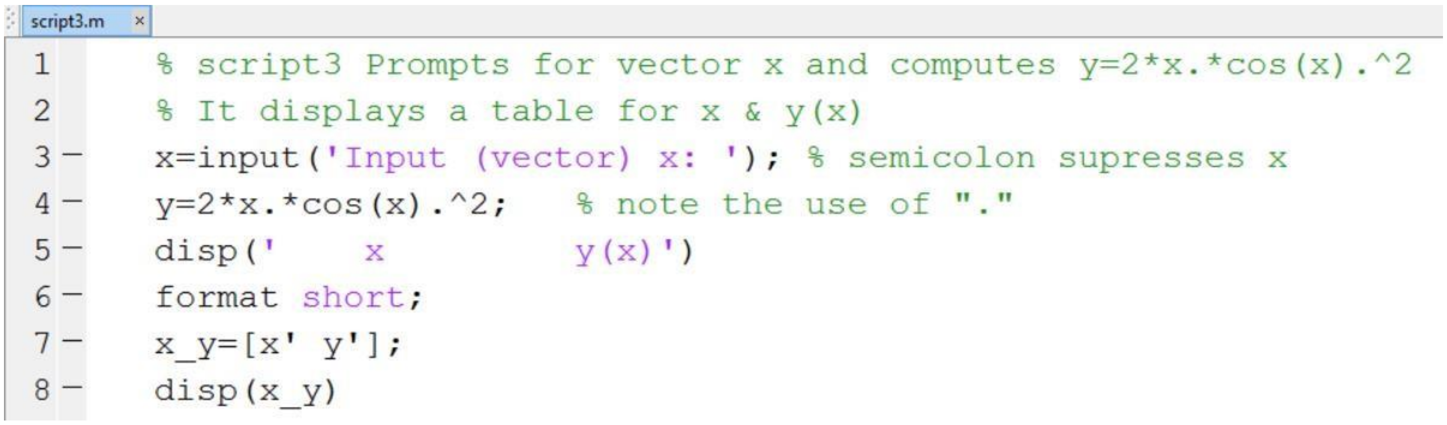

script2.m

```
% script2 Prompts for x and computes y=2*x*cos(x)^2
x=input('Input (scalar) x: '); % semicolon supresses x
y=2*x*cos(x)^2;
disp('function value is: ')
disp(y)
```



```
  >> script2
Input (scalar) x: 10.5
function value is:
   4.748842767645182
```

Example: Here is a script that prompts for a vector x and generates a table for x and its corresponding y(x) values.



```
% script3 Prompts for vector x and computes y=2*x.*cos(x).^2
% It displays a table for x & y(x)
x=input('Input (vector) x: '); % semicolon supresses x
y=2*x.*cos(x).^2;    % note the use of "."
disp('    x          y(x)')
format short;
x_y=[x' y'];
disp(x_y)
```

```
   >> script3
Input (vector) x: [1 2 3 4 5]
    x          y(x)
    1.0000    0.5839
    2.0000    0.6927
    3.0000    5.8805
    4.0000    3.4180
    5.0000    0.8046
```

Note: When the argument of disp is a matrix, the values are displayed one row after the other starting with the first row.

There is a workaround that allows disp to display text and a numerical value on the same line. This can be accomplished by converting the numerical value to a string using the num2string instruction as can be seen from the following example (note the use of the square brackets).

```
   >> x=pi;
   >> disp(['Value is ',num2str(x),' in format short'])
Value is 3.1416 in format short
```

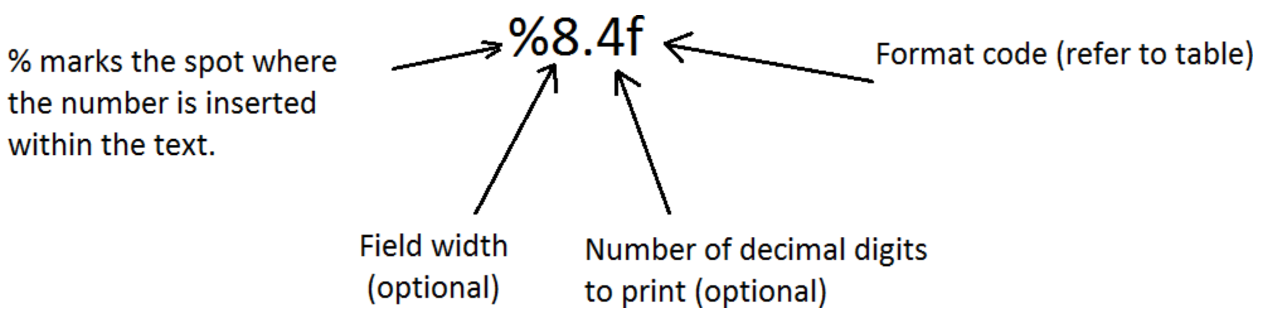
The fprintf Command

The fprintf command gives the user full control over displaying formatted messages and data on the same line. The basic syntax is:

fprintf('format', x, ...)

where format is a string specifying how the value of the variable x is to be displayed. A simple example would be to display a single value along with a message. Here is an example:

```
   >> var=pi;
   >> fprintf('The sum of the series is %8.4f radians \n', var)
The sum of the series is   3.1416 radians
```


%8.4f
% marks the spot where the number is inserted within the text.
Format code (refer to table)
Field width (optional)
Number of decimal digits to print (optional)

Adding a "-" on the right hand side of % (e.g., %-8.4) is a *flag* that will left justify the printed number. A "0" flag pads zeros to the left of the displayed number, if the number is shorter than the field. Try it.

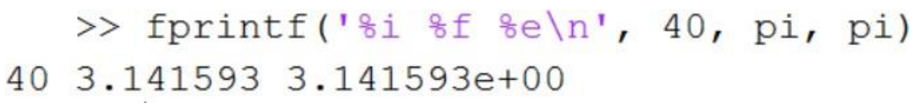
```
EDU>> fprintf('The sum of the series is %-8.4f \n', var)
The sum of the series is 3.1416
```

The control code \n starts a new line.

In the above example, % is a *format code* and \ is a *control code*. Their use is described in the following table.

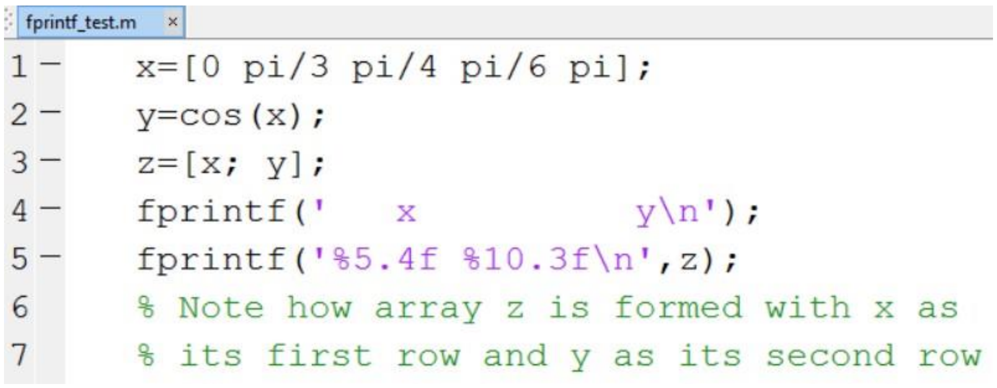
Format Code	**Description**
%i	Integer format
%e	Scientific format with lowercase e
%E	Scientific format with uppercase E
%f	Decimal format
%g	The more compact of %e or %f
Control Code	**Description**
\n	Start new line
\t	Tab

More examples of the use of the fprintf command:

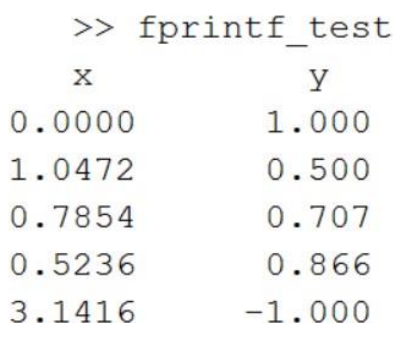


```
   >> fprintf('%i %f %e\n', 40, pi, pi)
40 3.141593 3.141593e+00
```

Here is a script that generates a table of cosine values:



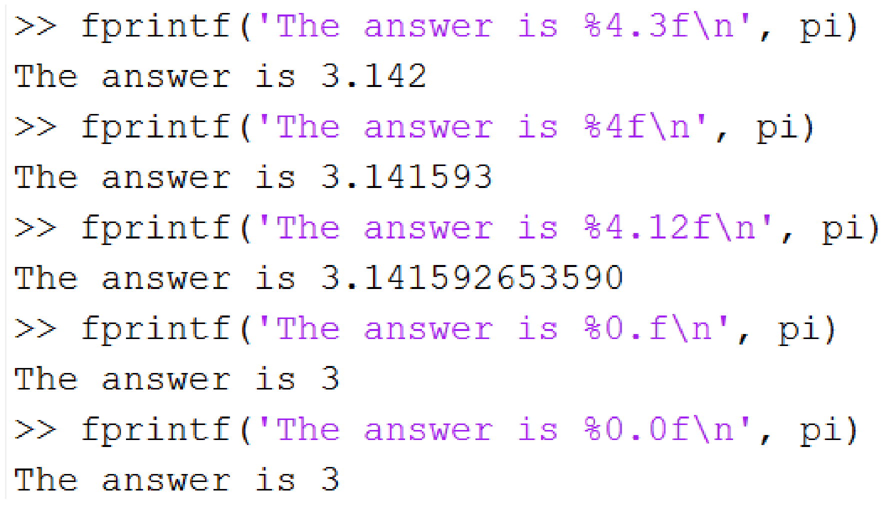
```
x=[0 pi/3 pi/4 pi/6 pi];
y=cos(x);
z=[x; y];
fprintf('    x            y\n');
fprintf('%5.4f %10.3f\n',z);
% Note how array z is formed with x as
% its first row and y as its second row
```



```
   >> fprintf_test
   x           y
0.0000       1.000
1.0472       0.500
0.7854       0.707
0.5236       0.866
3.1416      -1.000
```

If the field width is specified to be smaller than the number of spaces that a number requires, then it is ignored and the full number is displayed. Also, if the number of digits to the right of the decimal point is not specified, then the default is 6 digits.

The following are examples:

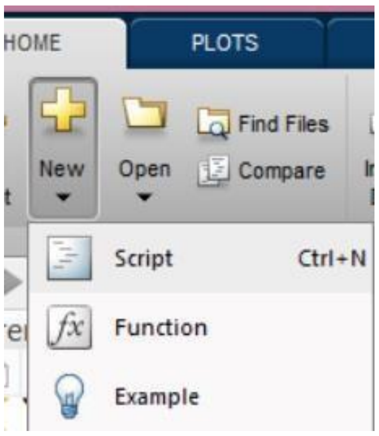


```
>> fprintf('The answer is %4.3f\n', pi)
The answer is 3.142
>> fprintf('The answer is %4f\n', pi)
The answer is 3.141593
>> fprintf('The answer is %4.12f\n', pi)
The answer is 3.141592653590
>> fprintf('The answer is %0.f\n', pi)
The answer is 3
>> fprintf('The answer is %0.0f\n', pi)
The answer is 3
```

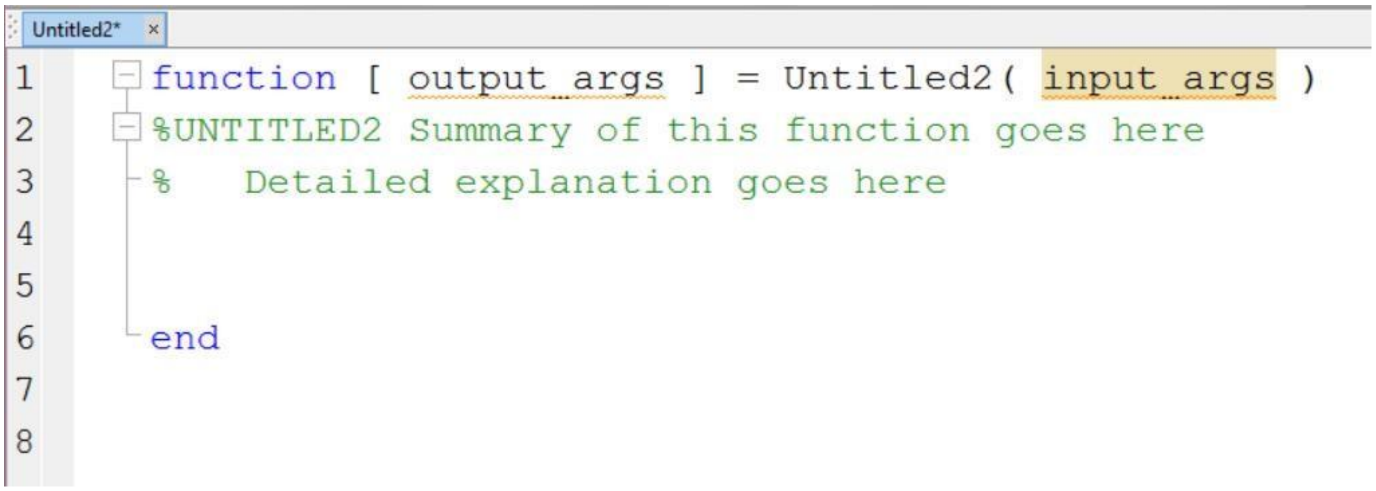
The best way to understand how the fprintf command works is to experiment with it.

User-Defined Functions

Function files are M-files that start with the word function. A convenient way to start a user-defined function is to click on “New” from the Home menu and select “Function”.



Doing that will open the Editor Window with the following function template:



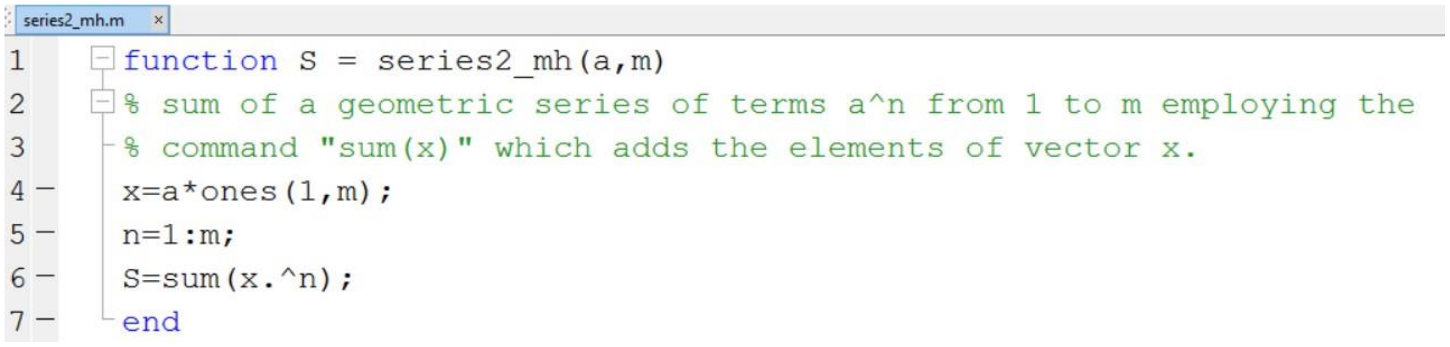
```
function [ output_args ] = Untitled2( input_args )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


end
```

Unlike scripts, user defined Matlab functions have no access to Workspace variables. There are two ways to send variables to functions: (1) As “input arguments” when the function is called and (2) by defining the variables as “global” at the “>>” prompt in the Command Window (or in the script that calls the function).

Refer to the last slide to learn more about global variables.

The following is a function called series2_mh. It accepts the inputs a and m and generates a scalar output S which is the sum of the geometric series:

$$S = \sum_{n=1}^{m} a^n$$


series2_mh.m

```
function S = series2_mh(a,m)
% sum of a geometric series of terms a^n from 1 to m employing the
% command "sum(x)" which adds the elements of vector x.
x=a*ones(1,m);
n=1:m;
S=sum(x.^n);
end
```

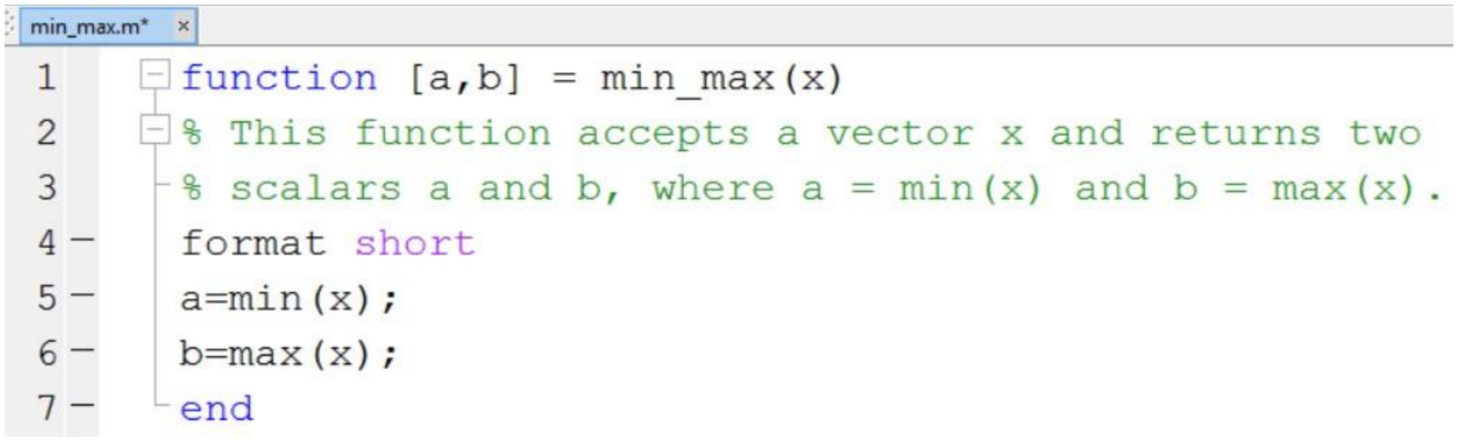
So, if we want to evaluate the series sum for a = 0.5 and m = 10 we would call the above function by typing at the >> prompt: series2_mh(0.5, 10)

```
    >> clear
    >> series2_mh(0.5,10)
ans =
    0.999023437500000
    >> z=series2_mh(0.5,10)
z =
    0.999023437500000
```

It should be emphasized here that, unlike scripts, the variables defined inside a function are only available inside the function and are completely independent ofthose variables defined in the Workspace, as can be seen from the following:

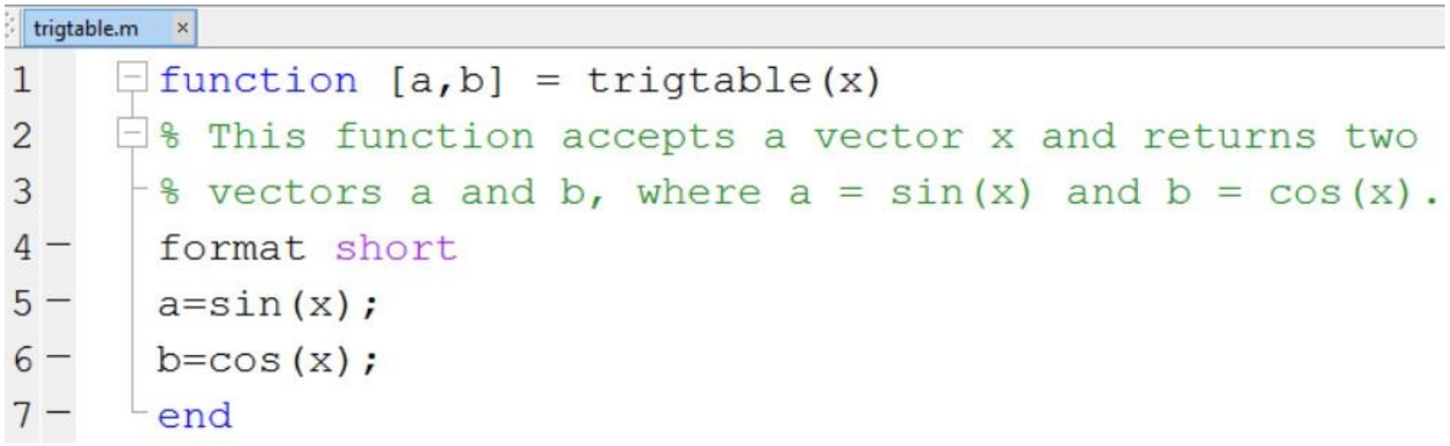
```
    >> S
Undefined function or variable 'S'.
    >> x
Undefined function or variable 'x'.
    >> n
Undefined function or variable 'n'.
```

A function can be written to return multiple outputs. The following min_max function computes the maximum and minimum values in a vector x:


```
min_max.m*
function [a,b] = min_max(x)
% This function accepts a vector x and returns two
% scalars a and b, where a = min(x) and b = max(x).
format short
a=min(x);
b=max(x);
end
```

```
   >> x=[1 2 -3 4 -15 30 2 5];
   >> [m,s]=min_max(x)
m =
   -15
s =
    30
```

Here is a function (trigtable) that computes the sines and cosines of the values in a vector:


```
trigtable.m
function [a,b] = trigtable(x)
% This function accepts a vector x and returns two
% vectors a and b, where a = sin(x) and b = cos(x).
format short
a=sin(x);
b=cos(x);
end
```

```
EDU>> [x,y]=trigtable([0 pi/6 pi/4 pi/3 pi/2]);
EDU>> x
x =
         0    0.5000    0.7071    0.8660    1.0000
EDU>> y
y =
    1.0000    0.8660    0.7071    0.5000    0.0000
```