Morteza Kazemi | 97243054
Amir Masoud Shaker | 97243081
11/07/2021

Homework #2

Q1

Input Arguments of cylinder2: (height, radius)
Output Arguments of cylinder2: [area, volume]
Name of a function must start with an alphabetic character to be valid.

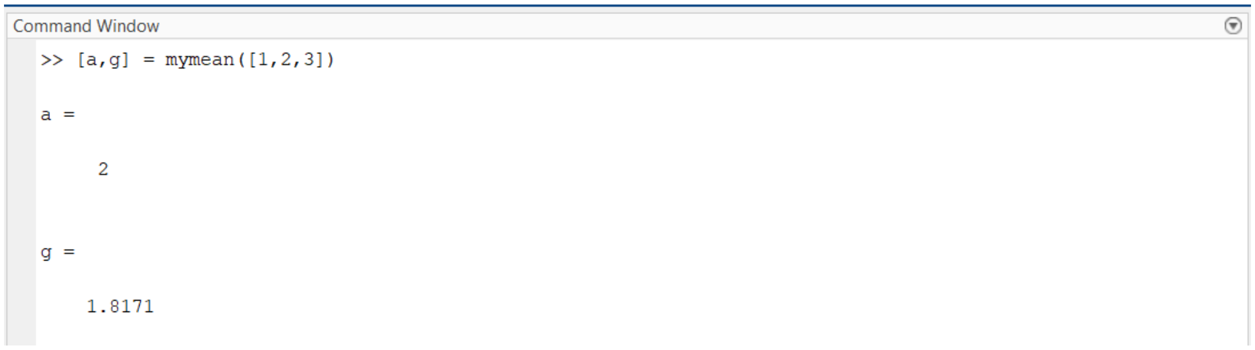
Name	Description
Local functions	In a function file, the first function is called the main function. This function is visible to functions in other files and it can be called from the command line. Additional functions within the file are called local functions, and they can occur in any order after the main function. Local functions are only visible to other functions in the same file.
Nested functions	A nested function is a function that is completely contained within a parent function. The main difference between nested functions and other types of functions is that they can access and modify variables that are defined in their parent functions. Notes: To nest any function in a program file, all functions in that file must use an end statement. A nested function can not be defined inside any control statements such as if/elseif/else, case/switch, try/catch, for, while. A nested function can be called directly by name or using a function handle that you create with the @ operator.

	All of the variables in nested functions or the functions that contain them must be explicitly defined.
Private functions	Private functions are useful to limit the scope of a function. A function can be designated as private by storing it in a subfolder with the name private. So, the function is available only to functions and scripts in the folder immediately above the private subfolder.
Anonymous functions	An anonymous function is a function that is not stored in a program file, but is associated with a variable whose data type is function handle. Anonymous functions can accept multiple inputs and return one output. They can contain only a single executable statement.

The examples are in the following:

Local:

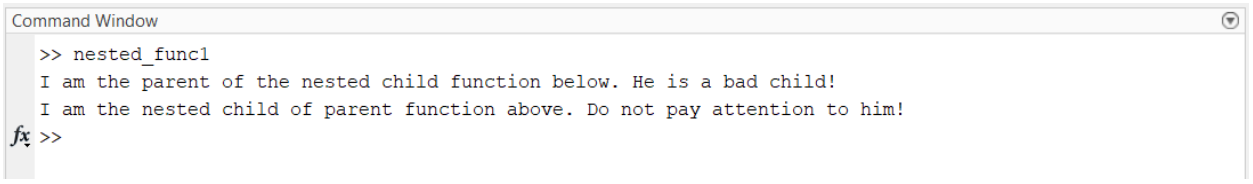
```
function [arithmetic, geometric] = mymean(x)
% The main function of the m file2
n = length(x);
arithmetic = arithmetic_mean(x,n);
geometric = geometric_mean(x,n);
end

function a = arithmetic_mean(x,n)
% arithmetic_mean example of a local function
a = sum(x)/n;
end

function g = geometric_mean(x,n)
% geometric_mean example of a local function
g = nthroot(prod(x),n);
end
```



```
Command Window
>> [a,g] = mymean([1,2,3])

a =

     2


g =

    1.8171
```

Nested1:

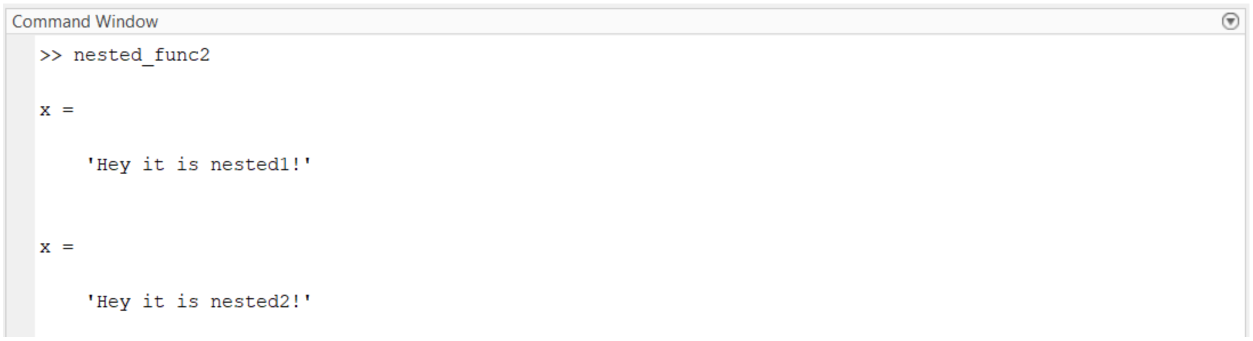
```
function parent
% The main function
disp('I am the parent of the nested child function below. He is a bad child!')
nested_child

   function nested_child
       % The nested function
      disp('I am the nested child of parent function above. Do not pay attention to
him!')
   end

end
```



```
Command Window
>> nested_func1
I am the parent of the nested child function below. He is a bad child!
I am the nested child of parent function above. Do not pay attention to him!
fx >>
```

Nested2:

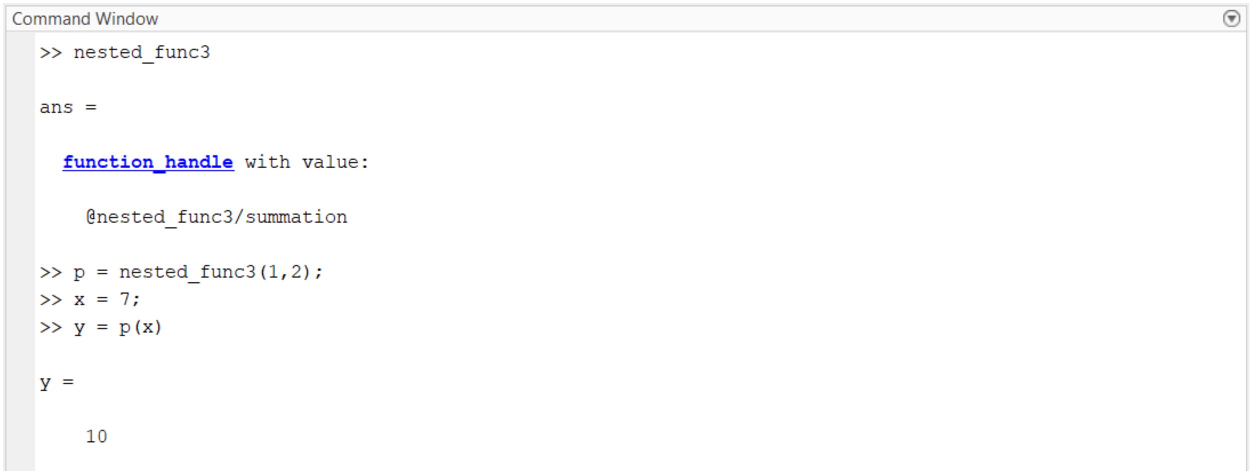
```
function nested_func2
% The main function including two nested functions that share variable x
   nested1
   nested2

   function nested1
       % nested function 1
      x = 'Hey it is nested1!'
   end

   function nested2
       % nested function 2
      x = 'Hey it is nested2!'
   end
end
```



```
>> nested_func2

x =

    'Hey it is nested1!'


x =

    'Hey it is nested2!'
```

Nested3:

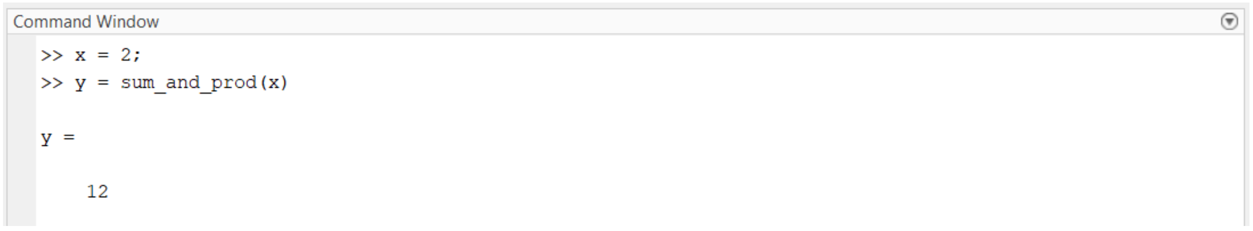
```
function p = nested_func3(a,b)
% The main function that returns a handle to the nested function summation
p = @summation;

   function y = summation(x)
   % The nested function
   y = x + a + b;
   end

end
```



```
>> nested_func3

ans =

  function_handle with value:

    @nested_func3/summation

>> p = nested_func3(1,2);
>> x = 7;
>> y = p(x)

y =

    10
```

Anonymous:

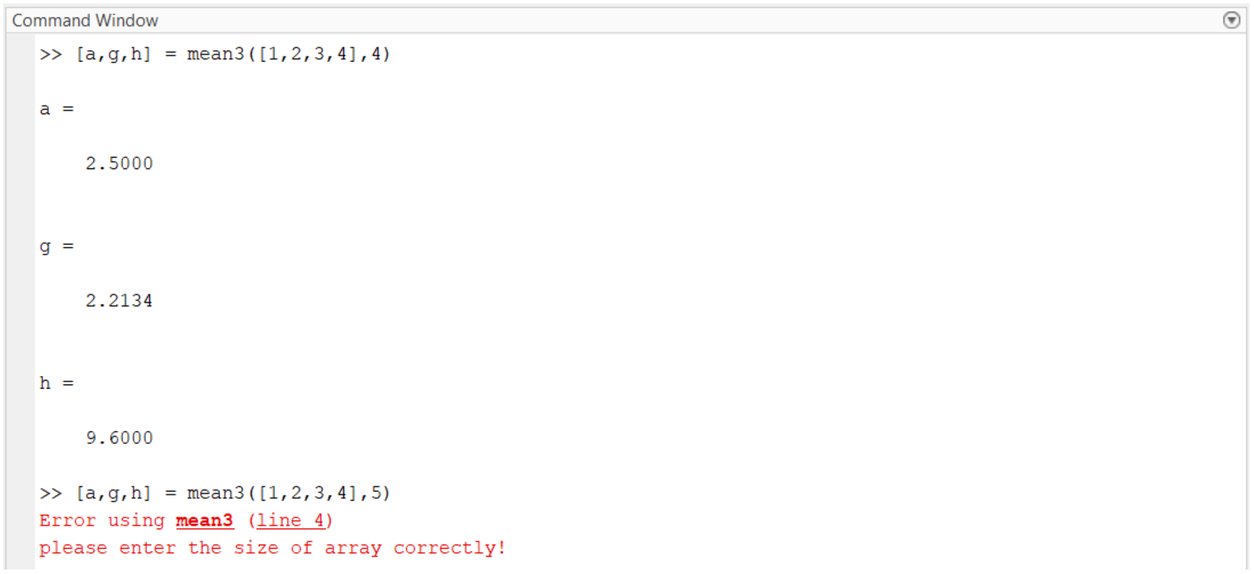
```
a = 1;
b = 2;
c = 3;
sum_and_prod = @(x) x*(a+b+c);
% An anonymous function with variable x which is a function handle
```



```
>> x = 2;
>> y = sum_and_prod(x)

y =

    12
```

Q2

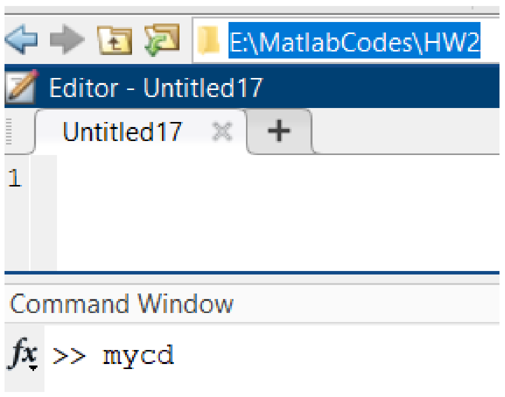
```
function [arithmetic,geometric,harmonic] = mean3(vector,size_of_vector)
% function to compute arithmetic,geometric and harmonic mean of a vector
if size_of_vector ~= size(vector)
    error('please enter the size of array correctly!')
end
% check that user enters the size of array correctly
arithmetic = sum(vector)/size_of_vector;
% Arithmetic Mean = (x1 + x2 + … + xN) / N
geometric = nthroot(prod(vector),size_of_vector);
% Geometric Mean = N-root(x1 * x2 * … * xN)
harmonic = size_of_vector*prod(vector)/sum(vector);
% Harmonic Mean = N / (1/x1 + 1/x2 + … + 1/xN)
```



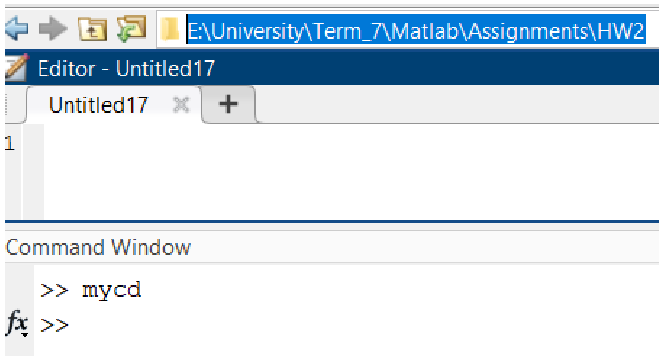
```
>> [a,g,h] = mean3([1,2,3,4],4)

a =

    2.5000


g =

    2.2134


h =

    9.6000

>> [a,g,h] = mean3([1,2,3,4],5)
Error using mean3 (line 4)
please enter the size of array correctly!
```

Syntax	Meaning	Description
1) cd 2) cd newFolder 3) oldFolder = cd(newFolder)	Change current folder	1) This syntax displays the current folder. 2) This syntax changes the current folder to newFolder 3) This syntax returns the existing current folder to oldFolder, and changes the current folder to newFolder.

```
cd 'E:\University\Term_7\Matlab\Assignments\HW2'
% simple changing directory
% Notice how the directory changes on top of the screenshot
```

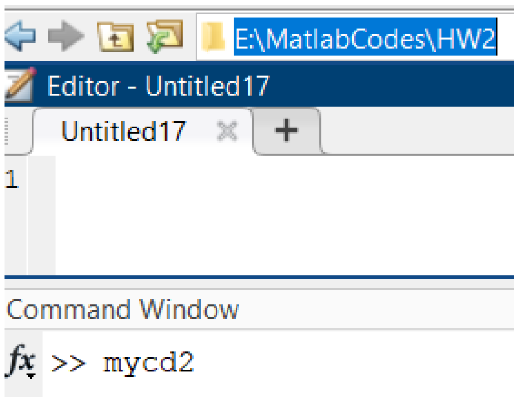


```
>> mycd
```

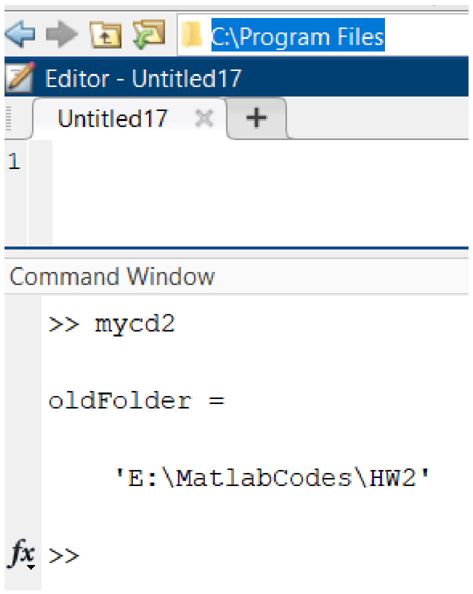


```
>> mycd
>>
```

```
oldFolder = cd('C:\Program Files')
% Change the current folder to C:\Program Files
% saving the folder path before changing it
% Notice how the directory changes on top of the screenshot
```



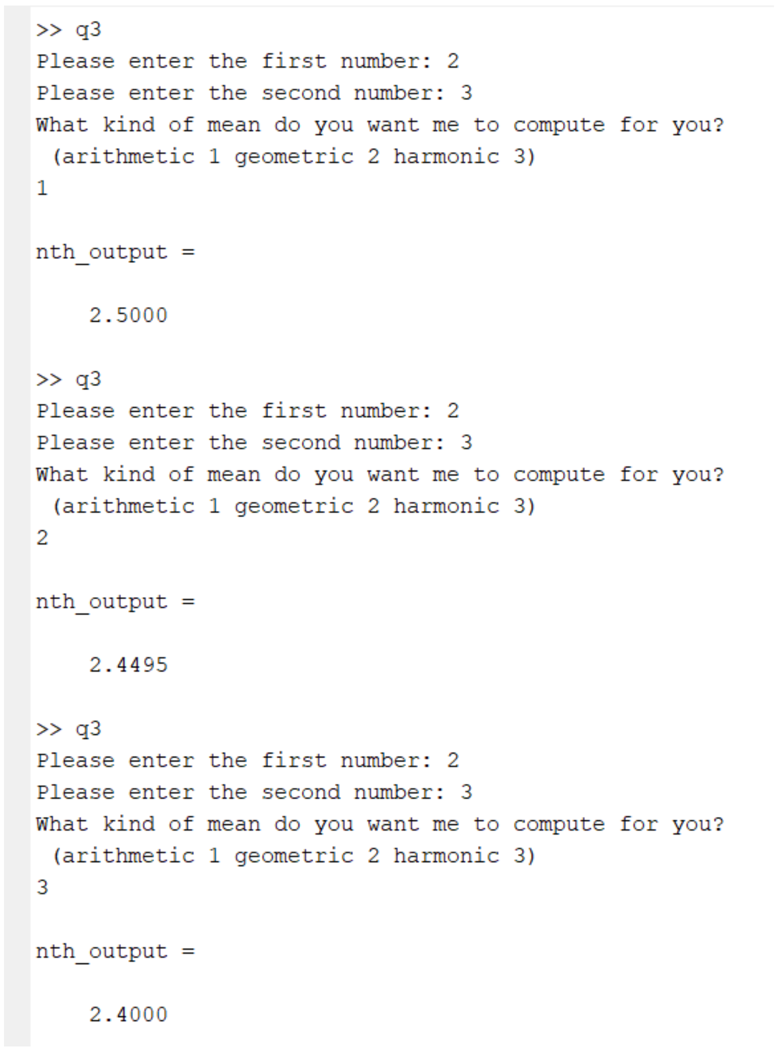
```
>> mycd2
```



```
>> mycd2

oldFolder =

    'E:\MatlabCodes\HW2'

>>
```

Q3

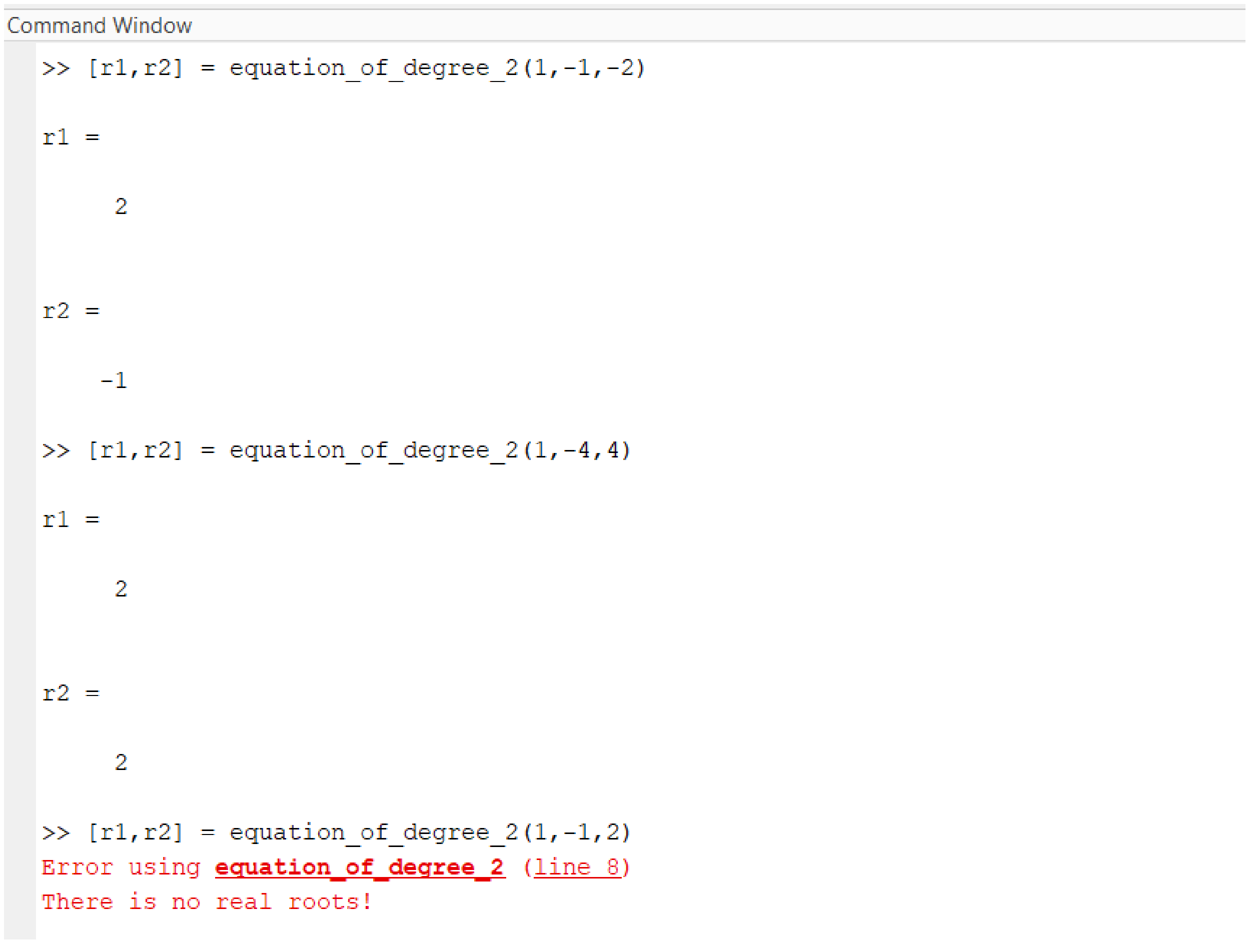
```
prompt1 = 'Please enter the first number: ';
first_number = input(prompt1);
% get the first number

prompt2 = 'Please enter the second number: ';
second_number = input(prompt2);
% get the second number

prompt3 = 'What kind of mean do you want me to compute for you?\n (arithmetic 1
geometric 2 harmonic 3) \n';
n = input(prompt3);
% get the type of mean that should be calculated

out = cell(1,n);
% create a cell with size of n
[out{:}] = mean3([first_number,second_number],2);
% insert the output to the cell
nth_output = out{n}
% get the nth output
```



```
>> q3
Please enter the first number: 2
Please enter the second number: 3
What kind of mean do you want me to compute for you?
 (arithmetic 1 geometric 2 harmonic 3)
1

nth_output =

    2.5000

>> q3
Please enter the first number: 2
Please enter the second number: 3
What kind of mean do you want me to compute for you?
 (arithmetic 1 geometric 2 harmonic 3)
2

nth_output =

    2.4495

>> q3
Please enter the first number: 2
Please enter the second number: 3
What kind of mean do you want me to compute for you?
 (arithmetic 1 geometric 2 harmonic 3)
3

nth_output =

    2.4000
```

Q4

```
function [root1,root2] = equation_of_degree_2(a,b,c)
% function to compute the root(roots) of an equation of degree 2 if
% if it has any roots
delta = b^2 - 4*a*c;
% calculate delta

if delta<0
    error('There is no real roots!')
end
% if the equation doesn't have real roots, send an error
root1 = (-1*b+sqrt(delta))/2*a;
% calculate first root of the equation
root2 = (-1*b-sqrt(delta))/2*a;
% calculate second root of the equation
```



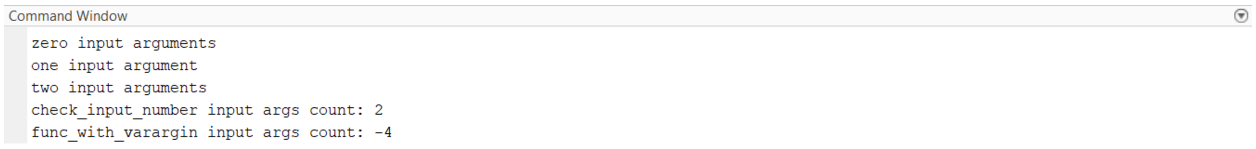
```
>> [r1,r2] = equation_of_degree_2(1,-1,-2)

r1 =

     2


r2 =

    -1

>> [r1,r2] = equation_of_degree_2(1,-4,4)

r1 =

     2


r2 =

     2

>> [r1,r2] = equation_of_degree_2(1,-1,2)
Error using equation_of_degree_2 (line 8)
There is no real roots!
```

Q5

Syntax	Meaning	Description
1) nargin 2) nargin(fun)	Number of function input arguments	1) nargin returns the number of function input arguments given in the call to the currently executing function. Use this syntax in the body of a function only. When using an arguments validation block, the value returned by nargin within a function is the number of positional arguments provided when the function is called. 2) nargin(fun) returns the number of input arguments that appear in the fun function definition. If the function includes varargin in its definition, then nargin returns the negative of the number of inputs. For example, if function myFun declares inputs a, b, and varargin, then nargin('myFun') returns -3. If the input argument refers to a function that uses an arguments validation block, then the returned value is the number of declared positional arguments in the function definition as a non-negative value.

Examples:

```
clc;
clear;

check_input_number();
check_input_number(1);
check_input_number(1, 2);

% formatted printing(%i is for integer)
fprintf('check_input_number input args count: %i\n', nargin('check_input_number')) %
Number of function input arguments

fprintf('func_with_varargin input args count: %i\n', nargin('func_with_varargin')) %
Number of function input arguments
% displays -4. meaning: 4 input args and the 4th one is a varargin

% Function definitions can be in a script! They must appear at the end of the file.
%.....................................................................................
function check_input_number(a,b)
    switch nargin %switch on number of inputs
        case 0
            disp('zero input arguments')
        case 1
            disp('one input argument')
        case 2
            disp('two input arguments')
    end
end
%.....................................................................................
function func_with_varargin(a, b, c, varargin)
end
```

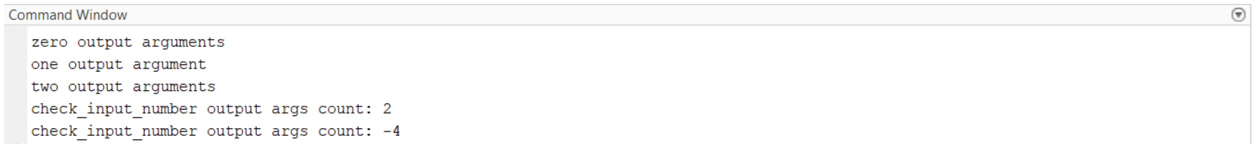

Command Window

```
zero input arguments
one input argument
two input arguments
check_input_number input args count: 2
func_with_varargin input args count: -4
```

Syntax	Meaning	Description
1) nargout 2) nargout(fun)	Number of function output arguments	1) nargout returns the number of function output arguments specified in the call to the currently executing function. Use this syntax in the body of a function only. 2) nargout(fun) returns the number of outputs that appear in the fun function definition. If the function includes varargout in its definition, then nargout returns the negative of the number of outputs. For example, if function myFun declares outputs y, z, and varargout, then nargout('myFun') returns -3.

Examples:

```
clc;
clear;

check_output_number();
a = check_output_number();
[a, b] = check_output_number();

fprintf('check_input_number output args count: %i\n', nargout('check_output_number'))
% Number of function input arguments

fprintf('check_input_number output args count: %i\n', nargout('func_with_varargout'))
% Number of function input arguments
% displays -4. meaning: 4 output args and the 4th one is a varargin

% Function definitions can be in a script! They must appear at the end of the file.
%.....................................................................................
function [a, b] = check_output_number() % acts based on the outputs it is called with
    switch nargout %switch on number of outputs
        case 0
            disp('zero output arguments')
        case 1
            disp('one output argument')
        case 2
            disp('two output arguments')
    end
    a = 3;
    b = 2;
end
%.....................................................................................
function [a, b, c, varargout] = func_with_varargout()
end
```


Command Window

```
zero output arguments
one output argument
two output arguments
check_input_number output args count: 2
check_input_number output args count: -4
```

Q6

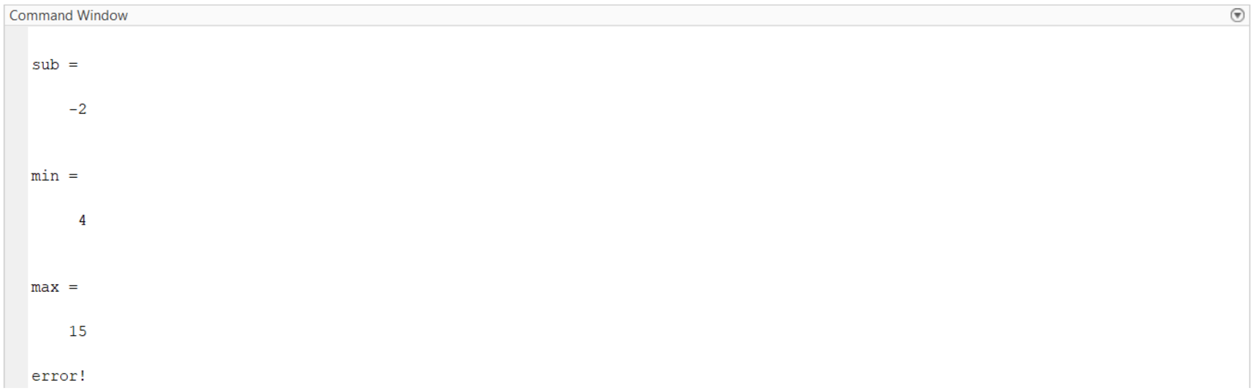
The details of varargin are explained in Q5 and Q8. The function is :

```
function output = q6_function(varargin) %gets variable input arguments in the form of
a cell array
% if gets 2 inputs -> subtraction
% if gets 3 inputs -> min
% if gets 4 inputs -> max
% if gets >4 inputs -> print error

% switch case just like what we have in Java and C!
switch nargin % nargin gives the total number of input arguments (varargin here)
   case 2
      output = varargin{1} - varargin{2}; % subtract cell1 and cell2 of the input cell
array
   case 3
      output = min([varargin{1} varargin{2} varargin{3}]); % put the cells in an array
and find the minimum
   case 4
      output = max([varargin{1} varargin{2} varargin{3} varargin{4}]); % put the cells
in an array and find the maximum

   otherwise % default statement
      disp('error!');
end


end
```

The script written to test the function is as follows:

```
clc;
clear;

sub = q6_function(5, 7)
min = q6_function(5, 8, 4)
max = q6_function(5, 7, 15, 9)
q6_function(5, 7, 8 , 9 , 10) % should display an error
```

The output of running the script in the command window:



```
Command Window

sub =

   -2


min =

    4


max =

   15

error!
```

Q7

Here is the information we need for the commands. Examples of using these commands are also provided in the following.

Syntax	**Meaning**	**Description**
3) tic 4) timerVal = tic	Start stopwatch timer	3) tic works with the toc function to measure elapsed time. The tic function records the current time, and the toc function uses the recorded value to calculate the elapsed time. 4) timerVal = tic stores the current time in timerVal so that you can pass it explicitly to the toc function. Passing this value is useful when there are multiple calls to tic to time different parts of the same code. timerVal is an integer that has meaning only for the toc function.

Notes:

- Consecutive tic calls overwrite the internally recorded starting time.
- The clear(clear command line) function does not reset the starting time recorded by a tic function call!

Syntax	**Meaning**	**Description**
1) toc 2) toc(timerVal) 3) elapsedTime = toc 4) elapsedTime = toc(timerVal)	Read elapsed time from stopwatch	1) toc reads the elapsed time since the stopwatch timer started by the call to the tic function. Matlab reads the internal time at the execution of the toc function and displays the elapsed time since the most recent call to the tic function without an output. The elapsed time is expressed in seconds.

		2) toc(timerVal) displays the elapsed time since the call to the tic function corresponding to timerVal. 3) elapsedTime = toc returns the elapsed time since the most recent call to the tic function. 4) elapsedTime = toc(timerVal) returns the elapsed time since the call to the tic function corresponding to timerVal.

Notes:

- Consecutive toc calls with no input return the elapsed time since the most recent call to tic. This property enables you to take multiple measurements from a single point in time.
- Consecutive calls to the toc function with the same timerVal input return the elapsed time since the tic function call that corresponds to timerVal.

Input Arguments:

Value of the internal timer saved from a previous call to the tic function and used by toc function are specified as a scalar of type uint64.

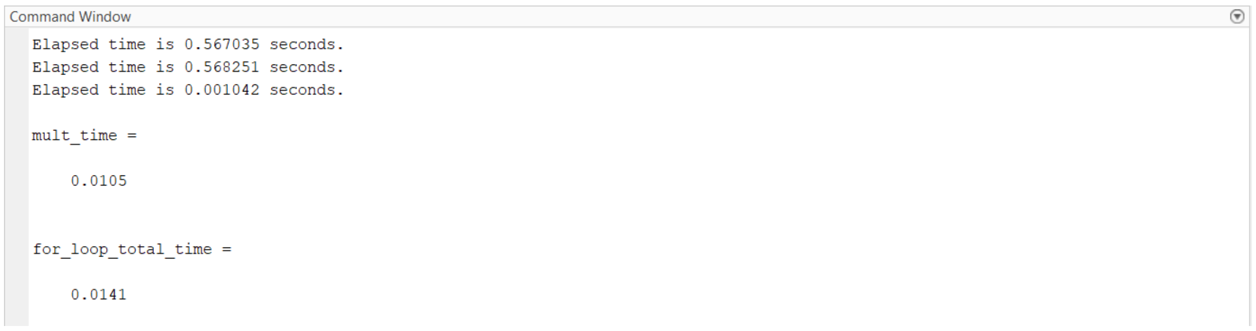
Examples:

```
clc;
clear;

tic % start timer (tic1)
a = (1:10);
b = a.^2;
toc % elapsed time from tic1

secondTic = tic; % tic2 does not reset tic
b = a.^2;

toc % elapsed time from tic1
toc(secondTic) % elapsed time from tic2

t_begin_for_loop = tic; % tic3 does not reset tic
mult_time = 0;
for i = 1:10
    tic          % tic4 resets tic
    c = a.*b;
    elapsed = toc;  % elapsed time from tic4
    mult_time = mult_time + elapsed;
end
mult_time
for_loop_total_time = toc(t_begin_for_loop) % elapsed time from tic3
```

All the details are explained in the code through comments.

The outputs are shown below:

Outputs:



```
Elapsed time is 0.567035 seconds.
Elapsed time is 0.568251 seconds.
Elapsed time is 0.001042 seconds.

mult_time =

    0.0105


for_loop_total_time =

    0.0141
```

Q8

Here is the information we need for the commands. Examples of using these commands are also provided in the following.

Syntax	Meaning	Description
1) varargin	Variable-length input argument list	varargin is an input variable in a function definition statement that enables the function to accept any number of input arguments. When the function executes, varargin is a 1-by-N cell array, where N is the number of inputs that the function receives after the explicitly declared inputs. If the function receives no inputs after the explicitly declared inputs, then varargin is an empty cell array. Note: Specify varargin using lowercase characters, and include it as the last input argument after any explicitly declared inputs.

Syntax	Meaning	Description
1) varargout	Variable-length output argument list	varargout is an output variable in a function definition statement that enables the function to return any number of output arguments. When the function executes, varargout is a 1-by-N cell array, where N is the number of outputs requested after the explicitly declared outputs. Inside of a function, varargout is an uninitialized variable and is not preallocated. Note: Specify varargout using lowercase characters, and include it as the last output argument after any explicitly declared outputs.

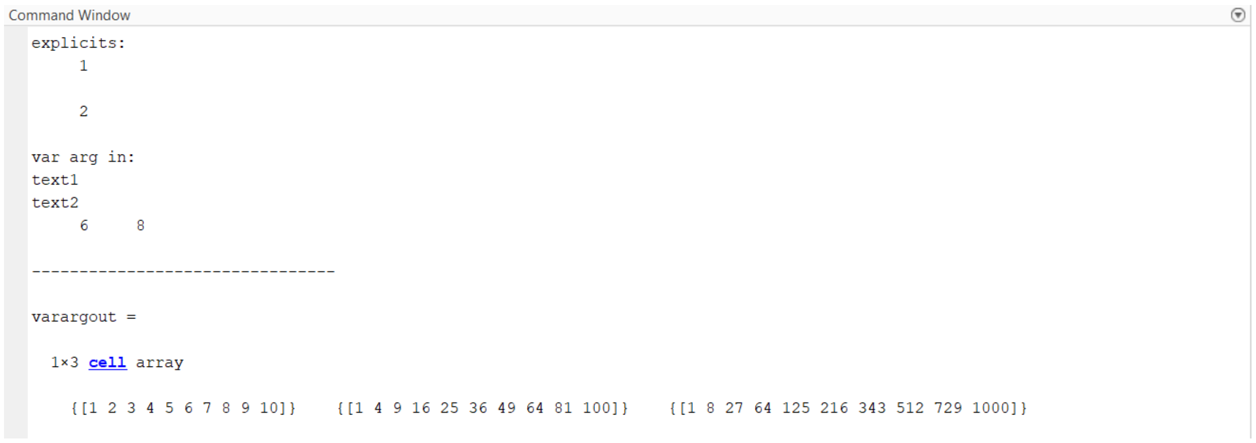
Examples:

```
clc;
clear;


get_variable_num_of_inputs(1, 2, 'text1','text2',[6 8])
disp('--------------------------------')
[seq1, seq2, seq3] = var_power_sequences(1, 3, 5); %put the sequences(cells) in three
variables

% Function definitions can be in a script! They must appear at the end of the file.
%...........................................................................
function get_variable_num_of_inputs(explicit1, explicit2, varargin) % Variable Number
of Function Inputs
    % varargin is a cell array
    disp('explicits: ');
    disp(explicit1);
    disp(explicit2);
    disp('var arg in: ');
    for i = 1:(nargin-2) % nargin gives the total number of input arguments (varagin +
explicit)
       disp(varargin{i});
    end
end
%...........................................................................
function varargout = var_power_sequences(varargin) % variable input and variable
output arguments
    % varargin gives n numbers as powers
    % the output is a varargout containing n power sequences of each of the input
powers

    num_of_seqs = size(varargin, 2); % number of seqences that will be on output = num
of input numbers
    % vargin is 1*N so we want the second dimension as the num of sequences

    sequence = [1:10]; % gives the sequence 1, 2, ..., 10

    for i = 1:num_of_seqs
        varargout{i} = sequence.^i; % each cell is a power sequence of the input
powers
    end
    varargout  % shows that varargout is a 1*3 cell array

end
%...........................................................................
```

All the details are explained in the code through comments.

The outputs are shown below:

Outputs:



```
explicits:
     1

     2

var arg in:
text1
text2
     6     8

--------------------------------

varargout =

  1×3 cell array

    {[1 2 3 4 5 6 7 8 9 10]}    {[1 4 9 16 25 36 49 64 81 100]}    {[1 8 27 64 125 216 343 512 729 1000]}
```