Function

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

AE121: Computational Method

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-06-17

Types of Functions

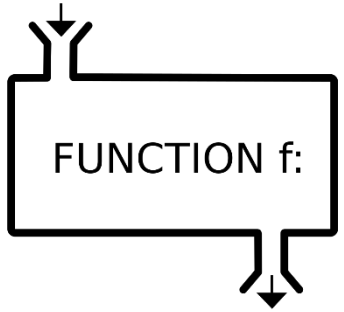
- Categories of functions:
  - functions that calculate and return one value
  - functions that calculate and return more than one value
  - functions that just accomplish a task, such as printing, without returning any values
- They are different in:
  - the way they are called
  - what the function header looks like
- All are stored in code files with the extension .m

Function

A function is a group of statements that together perform a task. In MATLAB, functions are defined in separate files. The name of the file and of the function should be the same.

Built-in Functions: Functions that are frequently used or that can take more time to execute are often implemented as executable files. These functions are called built-ins.

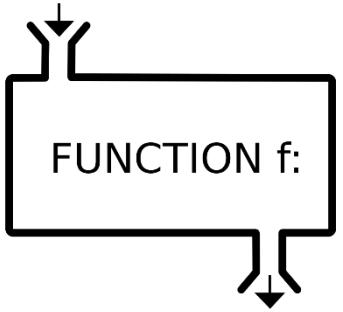
Single input



Single input

```
x = -1
y = abs(-1)
```

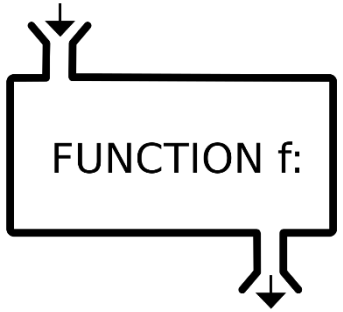
Multiple inputs



Single output

```
vec1 = randi(10,3,3)
```

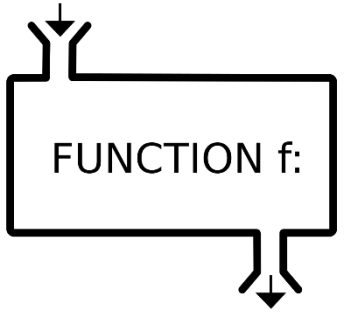
Single/Multiple inputs



Multiple output

```
mat1 = [1 1; 1 1];
[cz, rz] = size(mat1)
```

Single/Multiple inputs



No return value

```
disp('hello')
```

Example: Size Function

size

R2019a

Array size

collapse all in page

Syntax

```
sz = size(A)
szdim = size(A,dim)
[m,n] = size(A)
[sz1,...,szN] = size(A)
```

Description

example

`sz = size(A)` returns a row vector whose elements contain the length of the corresponding dimension of `A`. For example, if `A` is a 3-by-4 matrix, then `size(A)` returns the vector `[3 4]`. The length of `sz` is `ndims(A)`.

If `A` is a table or timetable, then `size(A)` returns a two-element row vector consisting of the number of rows and the number of table variables.

example

`szdim = size(A,dim)` returns the length of dimension `dim`.

example

`[m,n] = size(A)` returns the number of rows and columns when `A` is a matrix.

example

`[sz1,...,szN] = size(A)` returns the length of each dimension of `A` separately.

Generic Function Definition

functionname.m

```
function [output arguments] = functionname (input arguments)
% Comment describing the function


Your script


end
```

Example: Sample Functions

Editor - C:\Users\cmyeum\Dropbox_GitHub\AE121\lecture\function\MyAbs.m

MyAbs.m

```
1  function abs_val = MyAbs(in_val)
2
3      abs_val = sign(in_val) * in_val;
4
5  end
6
7
8
9
```

Command Window

```
>> abs_m3 = MyAbs(-3)

abs_m3 =

     3

>>
```

MyAbs.m | MyAbsSign.m

```
1  function [abs_val, sign_val] = MyAbsSign(in_val)
2
3      sign_val = sign(in_val);
4      abs_val =  sign_val * in_val;
5
6  end
7
8
9
10
```

Command Window

```
>> [abs_m3, sign_m3] = MyAbsSign(-3)

abs_m3 =

     3


sign_m3 =

    -1

>>
```

Example: Sample Functions 2

MyAbs.m | MyDot.m | Untitled4 | sol_problem1_jc.m | sol_problem2_jc.m

```
function dot_ab = MyDot(a, b)

dot_ab = a'*b;

end
```

```
>> a = [1;2;3]; b = [2; 2; 2;];
>> dot_ab_v1 = MyDot(a,b)

dot_ab_v1 =

    12

>> dot_ab_v2 = dot(a,b)

dot_ab_v2 =

    12
```

Example: Tensile Stress

```
num_data = 100;
num_material = 6;
material_char = ['c', 'p', 'w', 's', 'g', 'a'];
strain_data = randi(10000, num_data, 1); %
material_data = zeros(num_data,1);

for ii=1:num_data
    material_data(ii) = material_char(randi(num_material));
end
tensile_stress = zeros(num_data,1);

for ii = 1:num_data
    material_test = material_data(ii); % Proceed through ev
    % corresponding material one at a time using for loop
    strain_test = strain_data(ii);
    switch material_test
        case 'c'
            stress_test = strain_test * 17;
        case 'p'
            stress_test = strain_test * 117;
        case 'w'
            stress_test = strain_test * 9;
        case 's'
            stress_test = strain_test * 180;
        case 'g'
            stress_test = strain_test * 74;
        case 'a'
            stress_test = strain_test * 40;
    end
    tensile_stress(ii) = stress_test;
end
```

```
num_data = 100;
num_material = 6;
material_char = ['c', 'p', 'w', 's', 'g', 'a'];
strain_data = randi(10000, num_data, 1); %
material_data = zeros(num_data,1);
for ii=1:num_data
    material_data(ii) = material_char(randi(num_material));
end
tensile_stress = zeros(num_data,1);

for ii = 1:num_data
    material_test = material_data(ii); % material character
    strain_test = strain_data(ii); % strain data
    tm_test = FindTM(material_test); % find tensile modulus
    tensile_stress(ii) = stress_test * tm_test; % comput tensile stress
end
```

demo_tensile_function.m

```
function modulus = FindTM(material)
% Find tensile modulus

switch material
    case 'c'
        modulus = 17;
    case 'p'
        modulus = 117;
    case 'w'
        modulus = 9;
    case 's'
        modulus = 180;
    case 'g'
        modulus = 74;
    case 'a'
        modulus = 40;
end
```

FindTM.m

Example: Pressure Calculation

(a) 'stn12_data_2D' which contain all the data measured at Station **12**.

(b) 'month_data_2D' which contains all the data collected from days 21 to 50 at Station 3.

(c) 'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

```
num_st = 80; num_day = 50;
data_press_2D = zeros(18,num_day*num_st);
for ii=1:num_st
    press_st = randi([10 30], 18, num_day);
    loc_press = ((ii-1)*num_day+1):(ii*num_day);
    data_press_2D(:,loc_press) = press_st;
end

%(a)
st_num = 12;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
stn12_data_2D = data_press_2D(:,loc_col);

% (b)
st_num = 3;
loc_col = ((st_num-1)*num_day+21):((st_num-1)*num_day+50);
month_data_2D = data_press_2D(:,loc_col);

% (c)
values_greater_25_2D = zeros(80,1);
for ii=1:num_st
    loc_col = ((ii-1)*num_day+1):(ii*num_day);
    st_data_2D = data_press_2D(:,loc_col);
    values_greater_25_2D(ii) = sum(st_data_2D>25, 'all');
end
```

Example: Pressure Calculation (Continue)

(a) 'stn12_data_2D' which contain all the data measured at Station **12**.

(b) 'month_data_2D' which contains all the data collected from **days 21 to 50** at Station **3**.

(c) 'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

Function ExtrStData
input: 'data_press_2D', 'st_id'
output: 'st_data'

1. Read 'data_press_2D'
2. Extract data for station 'st_id'
3. Assign the data to 'st_data'

```
function st_data = ExtrStData(data_press_2D, st_id)

num_day = 50;

st_num = st_id;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
st_data = data_press_2D(:,loc_col);

end
```

Example: Pressure Calculation (Continue)

(a) 'stn12_data_2D' which contain all the data measured at Station **12**.

(b) 'month_data_2D' which contains all the data collected from **days 21 to 50** at Station **3**.

(c) 'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

Function ExtrDayData

input: : 'data_press_2D', 'st_id', 'str_day', 'end_day'

output: 'partial_st_data'

1. Read 'data_press_2D'
2. Extract station data for 'st_id'
3. Extract data collected from 'str_day' to 'end_day'
4. Assign the data to 'partial_st_data'

```
function partial_st_data = ExtrDayData(data_press_2D, st_id, str_day, end_day)

st_data = ExtrStData2(data_press_2D, st_id);
partial_st_data = st_data(:, str_day:end_day);

end

function st_data = ExtrStData2(data_press_2D, st_id)

num_day = 50;

st_num = st_id;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
st_data = data_press_2D(:,loc_col);

end
```

Example: Pressure Calculation (Continue)

(a) 'stn12_data_2D' which contain all the data measured at Station **12**.

(b) 'month_data_2D' which contains all the data collected from **days 21 to 50** at Station **3**.

(c) 'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

Function ExtrDayData

input: : 'data_press_2D', 'st_id', 'str_day', 'end_day'

output: 'partial_st_data'

1. Read 'data_press_2D'
2. Extract station data for 'st_id'
3. Extract data collected from 'str_day' to 'end_day'
4. Assign the data to 'partial_st_data'

```
function partial_st_data = ExtrDayData(data_press_2D, st_id, str_day, end_day)

st_data = ExtrStData(data_press_2D, st_id);
partial_st_data = st_data(:, 21:50);

end
```

Example: Pressure Calculation (Contir

'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

Function CountGrStData
input: : 'data_press_2D', 'thrs'
output: 'num_thrs_data

1. Read 'data_press_2D'
2. Extract station data for station ii
3. Count # of values that are more than 25
4. Assign the value to num_thres_data(ii)

```
function num_thrs_data = CountGrStData(data_press_2D, thrs)
num_st = 80;
num_thrs_data= zeros(num_st,1);

for ii=1:num_st
    st_data = ExtrStData3(data_press_2D, ii);
    logi_mat = st_data>thrs;
    num_thrs_data(ii) = MySum(logi_mat);
end
end

function st_data = ExtrStData3(data_press_2D, st_id)
num_day = 50;
st_num = st_id;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
st_data = data_press_2D(:,loc_col);
end
function num_ones = MySum(logi_matrix)
num_ones = 0;
vec = logi_matrix(:);
for ii=1:numel(vec)
    if vec(ii) == 1
        num_ones = num_ones + 1;
    end
end
end
```

Example: Pressure Calculation (Continue)

'values_greater_25_2D' which contains the number of pressure values greater than **25** during whole fifty days at each station.

Function CountGrStData
input: : ‘data_press_2D’, ‘thrs’
output: ‘num_thrs_data

1. Read ‘data_press_2D’
2. Extract station data for station ii
3. Count # of values that are more than 25
4. Assign the value to num_thres_data(ii)

```
function num_thrs_data = CountGrStData(data_press_2D, thrs)
num_st = 80;
num_thrs_data= zeros(num_st,1);

for ii=1:num_st
    st_data = ExtrStData(data_press_2D, ii);
    num_thrs_data(ii) = sum(st_data>thrs, 'all');
end
```

Example: Pressure Calculation (Comparisons)

```
num_st = 80; num_day = 50;
data_press_2D = zeros(18,num_day*num_st);
for ii=1:num_st
    press_st = randi([10 30], 18, num_day);
    loc_press = ((ii-1)*num_day+1):(ii*num_day);
    data_press_2D(:,loc_press) = press_st;
end

%(a)
st_num = 12;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
stn12_data_2D = data_press_2D(:,loc_col);

% (b)
st_num = 3;
loc_col = ((st_num-1)*num_day+21):((st_num-1)*num_day+50);
month_data_2D = data_press_2D(:,loc_col);

% (c)
values_greater_25_2D = zeros(80,1);
for ii=1:num_st
    loc_col = ((ii-1)*num_day+1):(ii*num_day);
    st_data_2D = data_press_2D(:,loc_col);
    values_greater_25_2D(ii) = sum(st_data_2D>25, 'all');
end
```

```
num_st = 80; num_day = 50;
data_press_2D = zeros(18,num_day*num_st);
for ii=1:num_st
    press_st = randi([10 30], 18, num_day);
    loc_press = ((ii-1)*num_day+1):(ii*num_day);
    data_press_2D(:,loc_press) = press_st;
end

%(a)
stn12_data_2D = ExtrStData(data_press_2D, 12);

% (b)
month_data_2D = ExtrDayData(data_press_2D, 12, 21, 50);

% (c)
values_greater_25_2D = CountGrStData(data_press_2D, 25);
```

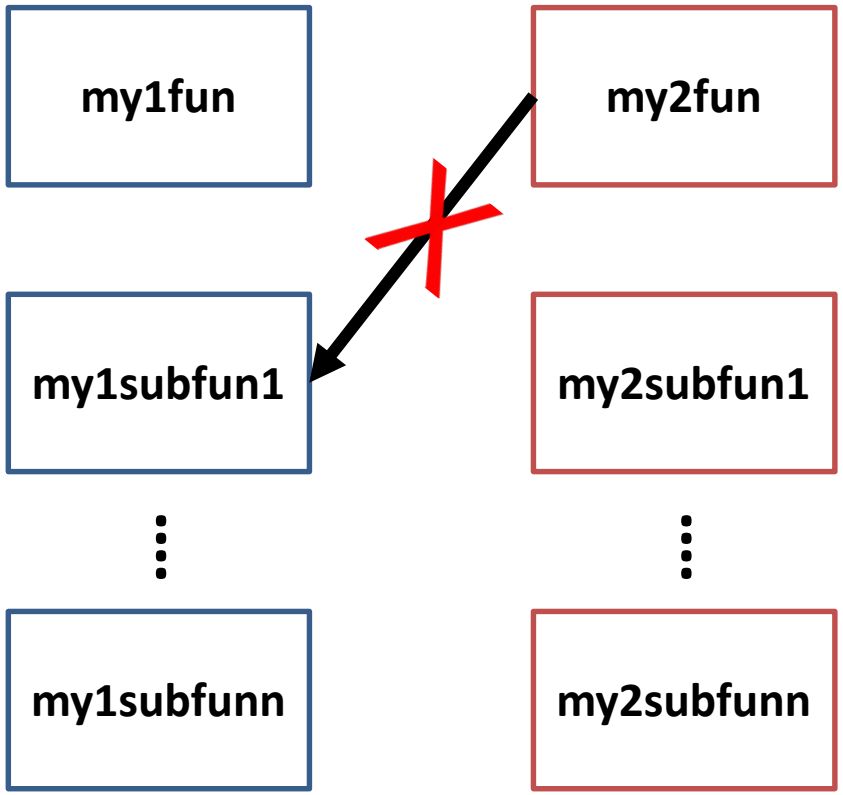
Local Functions

- This topic explains the term *local function*, and shows how to create and use local functions.

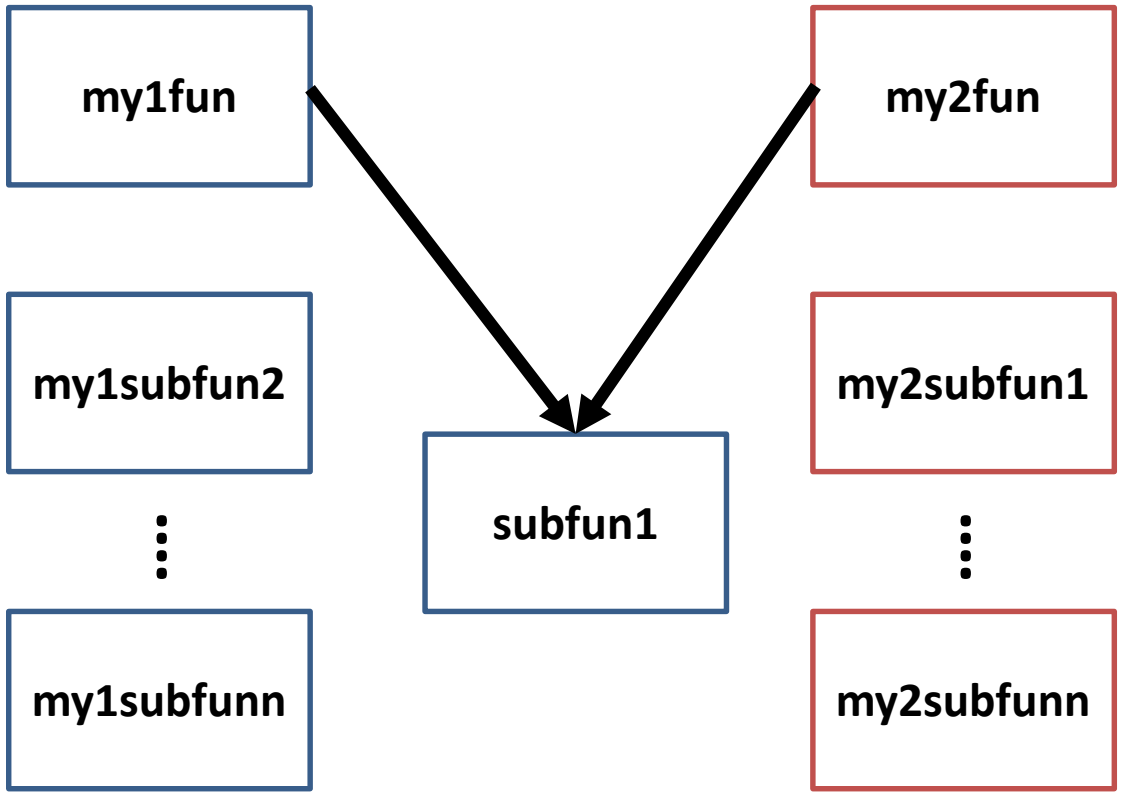
- MATLAB® program files can contain code for more than one function. In a function file, the first function in the file is called the main function. This function is visible to functions in other files, or you can call it from the command line. Additional functions within the file are called local functions, and they can occur in any order after the main function. Local functions are only visible to other functions in the same file. They are equivalent to subroutines in other programming languages, and are sometimes called subfunctions.

https://www.mathworks.com/help/matlab/matlab_prog/local-functions.html

Example: Local Functions



my1fun1.m my2fun.m



my1fun1.m subfun1.m my2fun.m

Add Functions to Scripts

- MATLAB® scripts, including live scripts, can contain code to define functions. These functions are called local functions. Local functions are useful if you want to reuse code within a script. By adding local functions, you can avoid creating and managing separate function files. They are also useful for experimenting with functions, which can be added, modified, and deleted easily as needed. Functions in scripts are supported in R2016b or later.
- Local functions are only visible within the file where they are defined, both to the script code and other local functions within the file. They are not visible to functions in other files, and cannot be called from the command line. They are equivalent to subroutines in other programming languages, and are sometimes called subfunctions.
- To add local functions to a script, first, create the script. Go to the Home tab and select New > Script. For more information about creating scripts, see Create Scripts. You also can Create Live Scripts in the Live Editor.

Example: Add Functions to the Scripts (Pressure Calculations)

```
num_st = 80; num_day = 50;
data_press_2D = zeros(18,num_day*num_st);
for ii=1:num_st
    press_st = randi([10 30], 18, num_day);
    loc_press = ((ii-1)*num_day+1):(ii*num_day);
    data_press_2D(:,loc_press) = press_st;
end

%(a)
stn12_data_2D = ExtrStDataNew(data_press_2D, 12);

% (b)
month_data_2D = ExtrDayDataNew(data_press_2D, 12, 21, 50);

% (c)
values_greater_25_2D = CountGrStDataNew(data_press_2D, 25);


function st_data = ExtrStDataNew(data_press_2D, st_id)

num_day = 50;

st_num = st_id;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
st_data = data_press_2D(:,loc_col);

end
```

Continue

```
function st_data = ExtrStDataNew(data_press_2D, st_id)

num_day = 50;

st_num = st_id;
loc_col = ((st_num-1)*num_day+1):(st_num*num_day);
st_data = data_press_2D(:,loc_col);

end

function partial_st_data = ExtrDayDataNew(data_press_2D, st_id, str_day

st_data = ExtrStDataNew(data_press_2D, st_id);
partial_st_data = st_data(:, 21:50);

end

function num_thrs_data = CountGrStDataNew(data_press_2D, thrs)
num_st = 80;
num_thrs_data= zeros(num_st,1);

for ii=1:num_st
    st_data = ExtrStDataNew(data_press_2D, ii);
    num_thrs_data(ii) = sum(st_data>thrs, 'all');
end
end
```

Example: Main and Local Functions

```
function grade = myfun_score2grade(score)
% This function converts your score into a grade.

if score>90
    grade = 'A';
elseif and(score>=80, score<90)
    grade = 'B';
elseif and(score>=70, score<80)
    grade = 'C';
else
    grade = 'D';
end

end
```

myfun_score2grade.m

```
function score_range = myfun_grade2score(grade)
% This function informs a score range of your grade. The output score_range
% is a 1 x 2 vector including the minimum and maximum scores that can be
% obtained given the grade.

switch grade
    case 'A'
        score_range = [90 100];
    case 'B'
        score_range = [80 70;
    case 'C'
        score_range = [70 80];
    case 'D'
        score_range = [70 0];
    otherwise
        disp('We do not have a such grade.');
end

end
```

myfun_grade2score.m

Example: Main and Local Functions

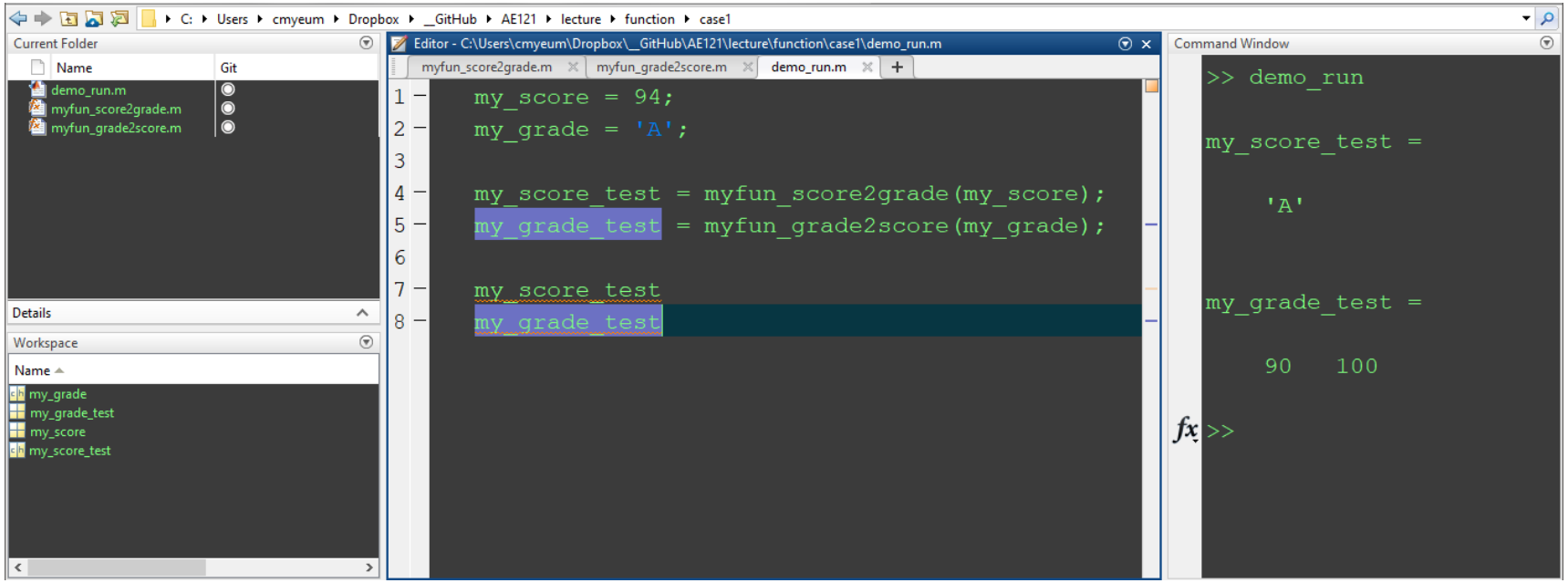
Case1: Each function is stored in a m file (define them as main functions).

myfun_score2grade.m

```
function grade = myfun_score2grade(score)
% This function converts your score into a grade.

if score>90
    grade = 'A';
elseif and(score>=80, score<90)
    grade = 'B';
elseif and(score>=70, score<80)
    grade = 'C';
else
    grade = 'D';
end
```

myfun_grade2score.m

```
function score_range = myfun_grade2score(grade)
% This function informs a score range of your grade. The output score_range
% is a 1 x 2 vector including the minimum and maximum scores that can be
% obtained given the grade.

switch grade
    case 'A'
        score_range = [90 100];
    case 'B'
        score_range = [80 70;
    case 'C'
        score_range = [70 80];
    case 'D'
        score_range = [70 0];
    otherwise
        disp('We do not have a such grade.');
end
```

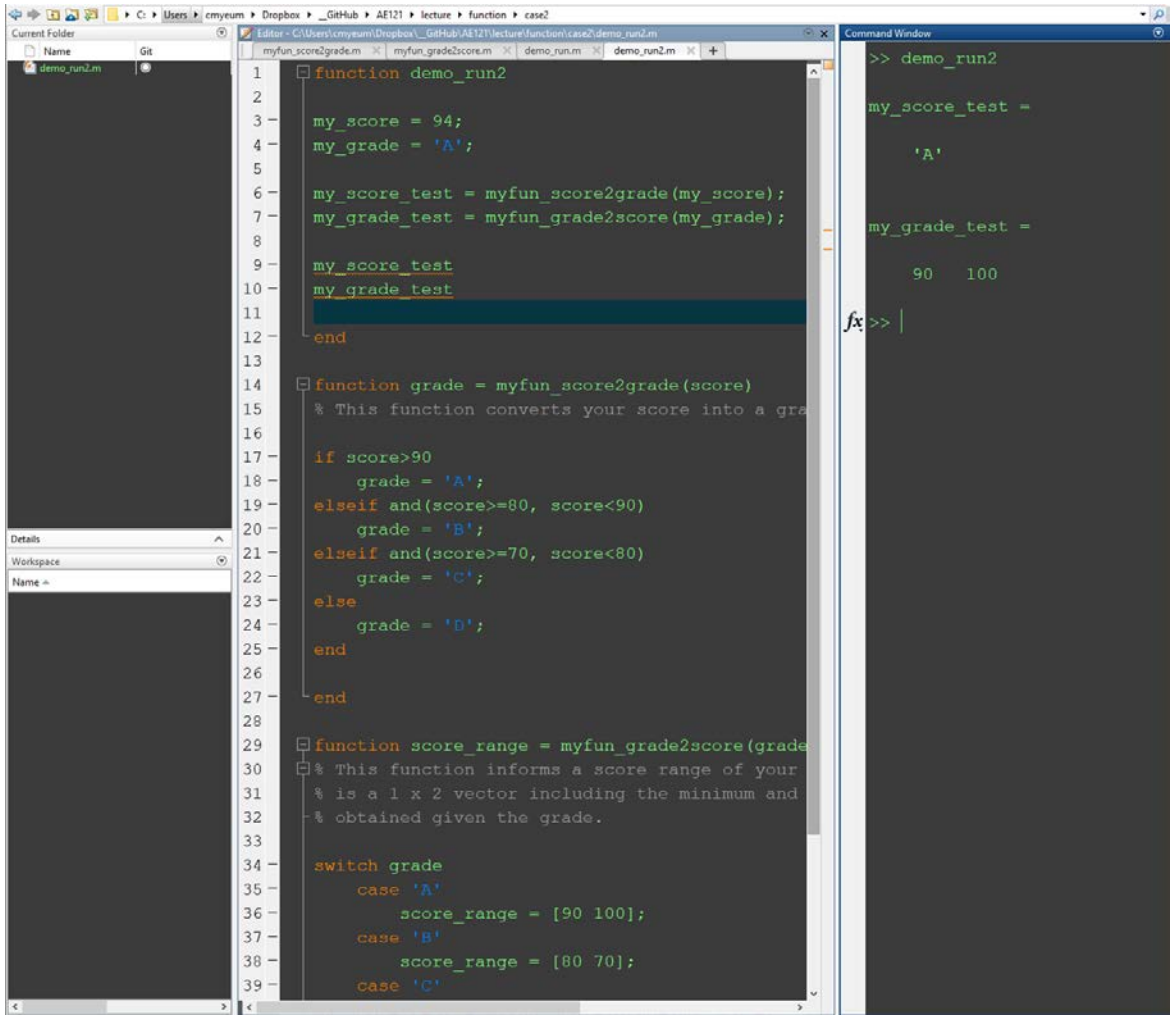


demo_run.m

Any file name is possible !!!

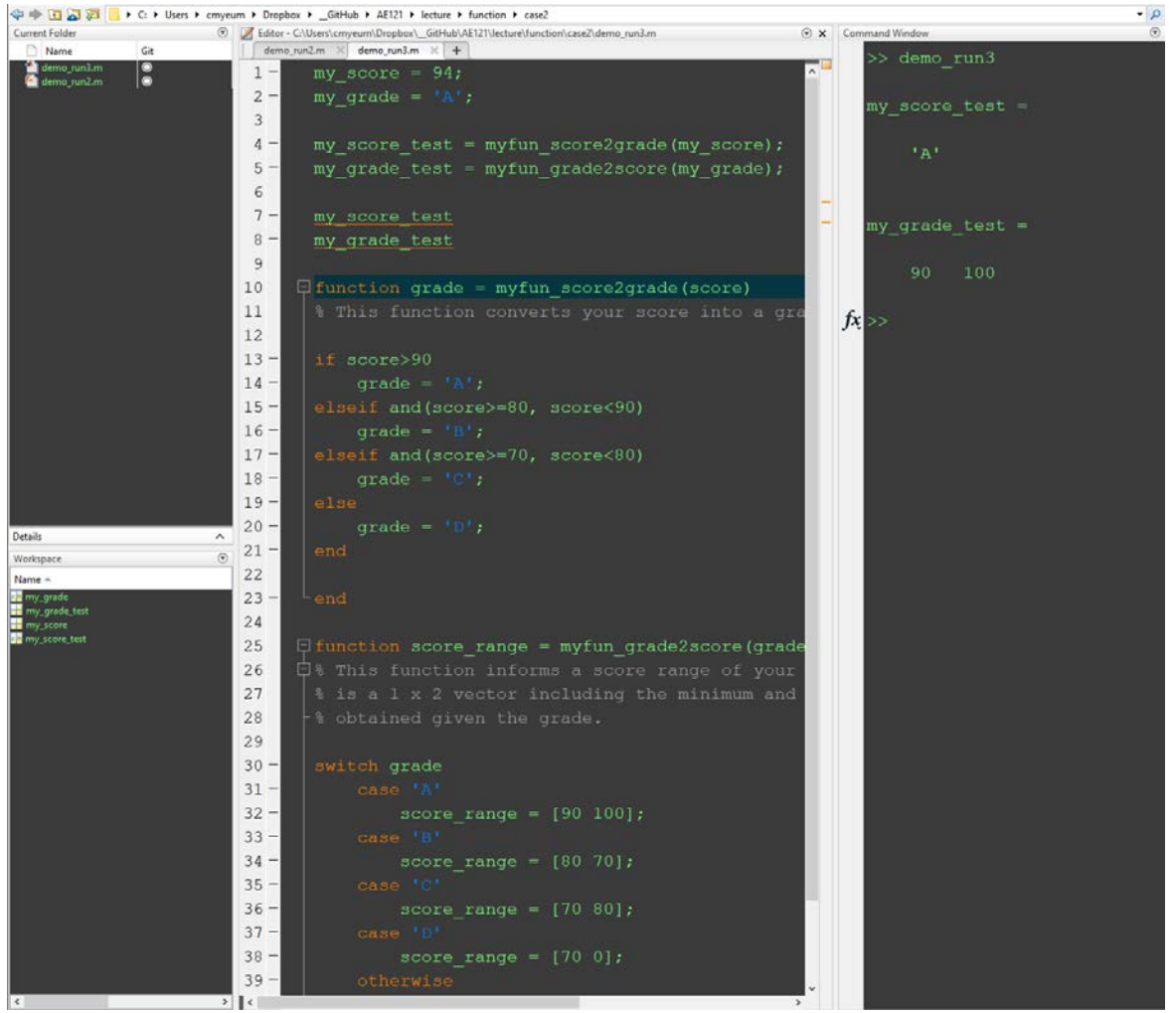
Example: Main and Local Functions

Case2: Functions are stored as local functions.



```
function demo_run2

my_score = 94;
my_grade = 'A';

my_score_test = myfun_score2grade(my_score);
my_grade_test = myfun_grade2score(my_grade);

my_score_test
my_grade_test

end

function grade = myfun_score2grade(score)
% This function converts your score into a gra

if score>90
    grade = 'A';
elseif and(score>=80, score<90)
    grade = 'B';
elseif and(score>=70, score<80)
    grade = 'C';
else
    grade = 'D';
end

end

function score_range = myfun_grade2score(grade
% This function informs a score range of your
% is a 1 x 2 vector including the minimum and
% obtained given the grade.

switch grade
    case 'A'
        score_range = [90 100];
    case 'B'
        score_range = [80 70];
    case 'C'
```

```
>> demo_run2

my_score_test =

    'A'


my_grade_test =

    90   100

>>
```



```
my_score = 94;
my_grade = 'A';

my_score_test = myfun_score2grade(my_score);
my_grade_test = myfun_grade2score(my_grade);

my_score_test
my_grade_test

function grade = myfun_score2grade(score)
% This function converts your score into a gra

if score>90
    grade = 'A';
elseif and(score>=80, score<90)
    grade = 'B';
elseif and(score>=70, score<80)
    grade = 'C';
else
    grade = 'D';
end

end

function score_range = myfun_grade2score(grade
% This function informs a score range of your
% is a 1 x 2 vector including the minimum and
% obtained given the grade.

switch grade
    case 'A'
        score_range = [90 100];
    case 'B'
        score_range = [80 70];
    case 'C'
        score_range = [70 80];
    case 'D'
        score_range = [70 0];
    otherwise
```

```
>> demo_run3

my_score_test =

    'A'


my_grade_test =

    90   100

>>
```

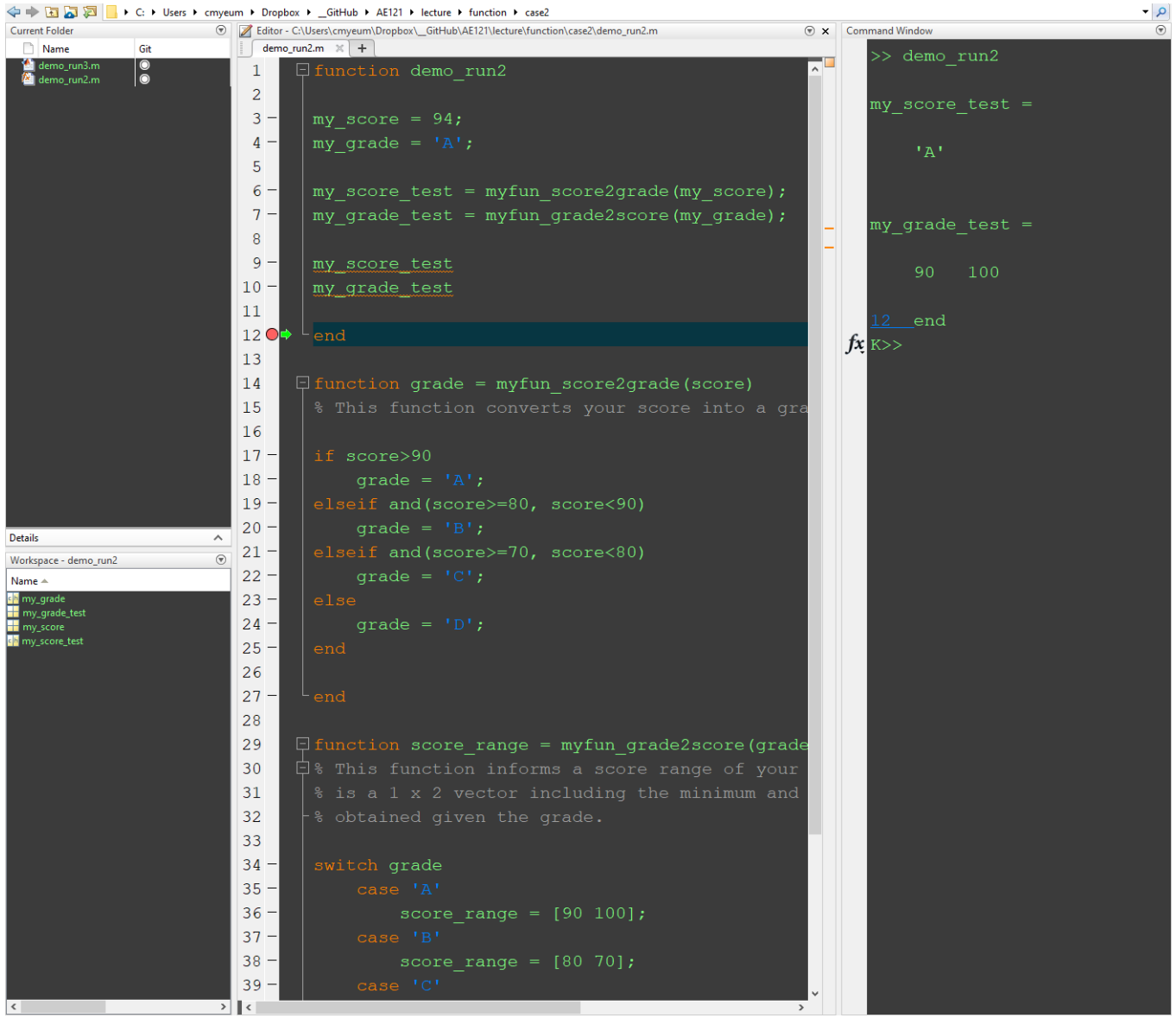
It is possible now!!!!

Variable Scope

- The ***scope*** of any variable is the workspace in which it is valid.
- The workspace created in the Command Window is called the ***base workspace***.
- Scripts also create variables in the base workspace
  - That means that variables created in the Command Window can be used in scripts and vice versa
  - However, that is very poor programming style
- Functions do not use the base workspace. Every function has **its own function workspace.** Each function workspace is separate from the base workspace and all other workspaces to protect the integrity of the data. Even local functions in a common file have their own workspaces. Variables specific to a function workspace are called local variables. Typically, local variables do not remain in memory from one function call to the next.

https://www.mathworks.com/help/matlab/matlab_prog/base-and-function-workspaces.html

How to See Variables in Function Workspace



Example: Variable Scope (Which is a valid code?)

```
in_vec = [1 2 3 4];
n_elem = numel(in_vec);
val1 = myfun_mean(in_vec);
val1

function mean_val = myfun_mean(vec)

mean_val = sum(vec)/n_elem;

end
```

1

```
vec = [1 2 3 4];
n_elem = numel(vec);
mean_val = myfun_mean(vec);
mean_val

function mean_val = myfun_mean(vec)

mean_val = sum(vec)/n_elem;

end
```

2

```
vec = [1 2 3 4];
n_elem = numel(vec);
mean_val = myfun_mean(vec);
mean_val

function mean_val = myfun_mean(vec, n_elem)

mean_val = sum(vec)/n_elem;

end
```

3

```
vec = [1 2 3 4];
n_elem = numel(vec);
mean_val = myfun_mean(vec);
mean_val

function mean_val = myfun_mean(vec)

n_elem = numel(vec);
mean_val = sum(vec)/n_elem;

end
```

4

Notes on Functions

- The function header and function call have to match up:
  - the name has to be the same
  - the number of input arguments must be the same
  - the number of variables in the left-hand side of the assignment should be the same as the number of output arguments
  - if there are no output arguments, the function call is a statement
- Functions that return values do not normally print them, also – that is left to the calling function/script

Calling the function

- Since the function is returning multiple values through the output arguments, the function call should be in an assignment statement with multiple variables in a vector on the left-hand side (the same as the number of output arguments in the function header) in order to capture all of them
- Otherwise, some will be lost.

- For example, if the function header is:

  function [x,y,z] = fnname(a,b)
- This indicates that the function is returning 3 things, so a call to the function might be (assuming a and b are numbers):

  [g,h,t] = fnname(11, 4.3);
- Or using the same names as the output arguments (it doesn't matter since the workspace is not shared):

  [x,y,z] = fnname(11, 4.3);
- This function call would only get the first value returned:

  result = fnname(11, 4.3);

Example: Function Call

```
vec = [1 2 3 4];
n_elem = numel(vec);

[mean_val1, sum_val1] = myfun_mean(vec);
[mean_val2, sum_val2] = myfun_mean(vec, n_elem);

mean_val3 = myfun_mean(vec);
[mean_val4, ~] = myfun_mean(vec);
[~, sum_val5] = myfun_mean(vec);

mean_val1
mean_val2
mean_val3
mean_val4

sum_val1
sum_val2
sum_val5

function [mean_val, sum_val] = myfun_mean(vec, n_elem)

if nargin == 1
    n_elem = numel(vec);
end

sum_val = sum(vec);
mean_val = sum_val/n_elem;

end
```

```
>> example_calling_function

mean_val1 =

    2.5000


mean_val2 =

    2.5000


mean_val3 =

    2.5000


mean_val4 =

    2.5000
```

```
sum_val1 =

    10


sum_val2 =

    10


sum_val5 =

    10
```

Functions that Do Not Return Anything

- A function that does not return anything has no output arguments in the function header, nor does it have the assignment operator
- The statements in the body would typically display or plot information from the input arguments

functionname.m

```
function functionname(input arguments)
% Comment describing the function
  statements here
end
```

Calling a Function with No Output

- Since no value is returned, the call to such a function is a statement
- For example, if this is the function header:

  function fnname(x,y)
- A call to the function might look like this:

  fnname(x,y)
- The following would NOT be a valid call; since the function is not returning anything, there is no value to assign:

  result = fnname(x,y); % Invalid!
- You do not always have to pass input arguments to a function. If you do not, you can have (both in the function header and in the function call) empty (), or you can just leave them out

Example: Functions with No Ouptut

```
clear; clc;

score = 90;
myfun_print_grade(score);

score = 85;
myfun_print_grade(score);

score = 75;
myfun_print_line()
myfun_print_grade(score);
myfun_print_line()

function myfun_print_grade(score)

if score>=90
    disp('Your grade is A.');
elseif and(score>=80, score<90)
    disp('Your grade is B.');
elseif and(score>=70, score<80)
    disp('Your grade is C.');
else
    disp('Your grade is D.');
end

end

function myfun_print_line()
disp('=======================================================');
end
```

```
Your grade is A.
Your grade is B.
=======================================================
Your grade is C.
=======================================================
```

Common Pitfalls

- Not matching up arguments in a function call with the input arguments in a function header.
- Not having enough variables in an assignment statement to store all of the values returned by a function through the output arguments.
- Attempting to call a function that does not return a value from an assignment statement, or from an output statement.

Clear and valid input-output relationship!

Slide Credits and References

- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition