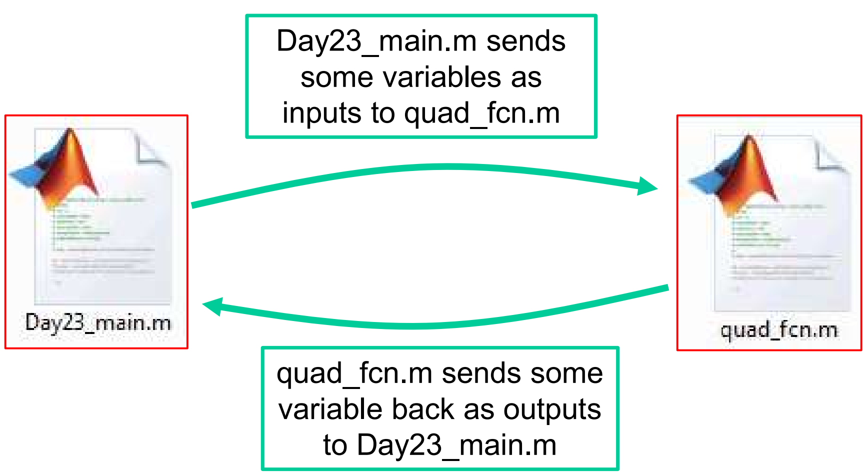
Day 23

- (Concept Question)
- Understanding variables in functions
- Using functions to declutter your programs
- Exercises

Understanding variables in functions

When writing functions for your programs, it is very important to know how variables behave when you pass them to a function.

There are two m-files on the course web designed to illustrate how variables move between a program and a function:



Understanding variables in functions

Day23_main.m is shown below. It calls quad_fcn and gives values to 6 different input values.

Where do those values go?

```
%
%  Test the function quad_fcn
%
clear variables
close all
clc
min_range = -1;
max_range = 2;
delta = 1;
[x,y] = quad_fcn(-3,2,1,min_range,max_range,delta);
```

Day23_main.m

Understanding variables in functions

Those values from Day23_main get assigned to the variables that are shown in the first line of the function:

```
[x,y] = quad_fcn(-3,2,1,min_range,max_range,delta);
```

quad_fcn.m

```
function [x_vec,y_vec] = quad_fcn(a,b,c,x_min,x_max,delta_x)
%
%   Calculates the value of a quadratic
%   y = a*x^2 + b*x + c
%   for x from x_min to x_max, in increments of delta_x.
%   The value of x is stored in the vector x_vec
%   and the value of y is stored in the vector y_vec.
%
row = 0 ;
for x = x_min:delta_x:x_max
    row = row + 1;
    y = a * x^2 + b * x + c;
    x_vec(row,1) = x;
    y_vec(row,1) = y;
end
```

Understanding variables in functions

The function, however, must assign values to the output variables. In this case, `x_vec` and `y_vec` are the output variables. After the function is done, where do the values of the output variables go?


```
function [x_vec,y_vec] = quad_fcn(a,b,c,x_min,x_max,delta_x)
%
%   Calculates the value of a quadratic
%   y = a*x^2 + b*x + c
%   for x from x_min to x_max, in increments of delta_x.
%   The value of x is stored in the vector x_vec
%   and the value of y is stored in the vector y_vec.
%
row = 0 ;
for x = x_min:delta_x:x_max
    row = row + 1;
    y = a * x^2 + b * x + c;
    x_vec(row,1) = x;
    y_vec(row,1) = y;
end
```


Understanding variables in functions

Those values from the function output get assigned to the variables shown in the main program.

```
%
%  Test the function quad_fcn
%
clear variables
close all
clc
min_range = -1;
max_range = 2;
delta = 1;
[x,y] = quad_fcn(-3,2,1,min_range,max_range,delta);
```


```
function [x_vec,y_vec] = quad_fcn(a,b,c,x_min,x_max,delta_x)
```

Understanding variables in functions

This passing of the variables back and forth can be seen clearly when you run the DEBUGGER with a function like this. You can try this on your own as well.

If we set a break point at line 9, run the program and step once to get to line 10, we see the following in the workspace:

```
1     %
2     % Test the function quad_fcn
3     %
4     clear variables
5     close all
6     clc
7     min_range = -1;
8     max_range = 2;
9     delta = 1;
10    [x,y] = quad_fcn(-3,2,1,min_range,max_range,delta);
```

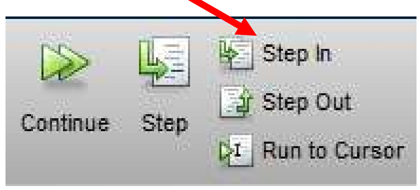
NOTE the values we have in our workspace right now.

Workspace

Name	Value
delta	1
max_range	2
min_range	-1

Understanding variables in functions

When you press the “Step In” button in the Debugger controls, you will “step” into the function.



The Debugger then shows its green arrow in the function itself—see the next slide.

Understanding variables in functions

Here is where the Debugger makes its next stop—now in the function:


```
function [x_vec,y_vec] = quad_fcn(a,b,c,x_min,x_max,delta_x)
%
%   Calculates the value of a quadratic
%   y = a*x^2 + b*x + c
%   for x from x_min to x_max, in increments of delta_x.
%   The value of x is stored in the vector x_vec
%   and the value of y is stored in the vector y_vec.
%
row = 0 ;
for x = x_min:delta_x:x_max
    row = row + 1;
    y = a * x^2 + b * x + c;
    x_vec(row,1) = x;
    y_vec(row,1) = y;
end
```

Now see how all the variables in the function have values.
Also notice that the variables from Day23_main ***do not appear.*** *Their values have been* ***passed to these variables****.*

Workspace

Name	Value
a	-3
b	2
c	1
delta_x	1
x_max	2
x_min	-1

Understanding variables in functions

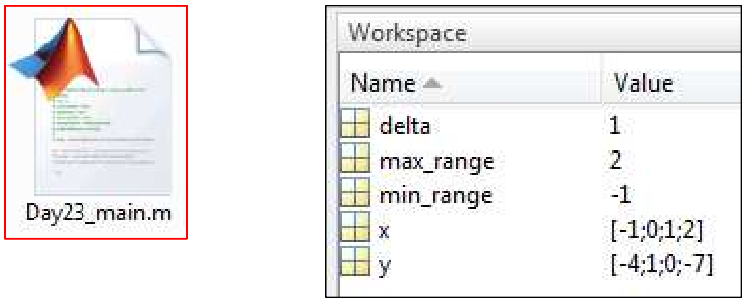
If you continue to press the step button, you will see the function assign values to the `x_vec` and `y_vec` variables—the output variables of the function:

Workspace

Name	Value
a	-3
b	2
c	1
delta_x	1
row	4
x	2
x_max	2
x_min	-1
x_vec	[-1;0;1;2]
y	-7
y_vec	[-4;1;0;-7]

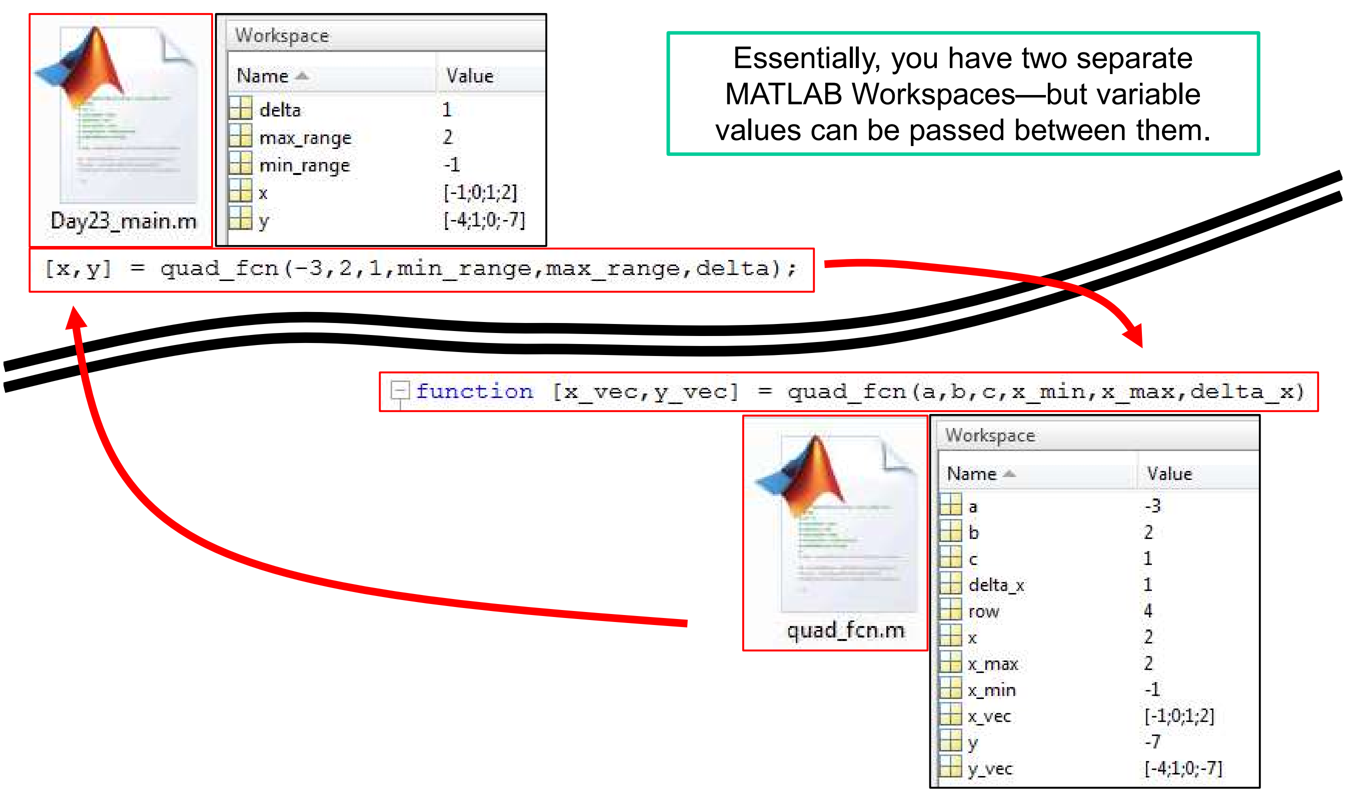
Understanding variables in functions

And when you step back out to the main program, you see that the values of `x_vec` and `y_vec` have been passed to the main program variables, `x` and `y`:



Understanding variables in functions

Essentially, you have two separate MATLAB Workspaces—but variable values can be passed between them.



Using functions to declutter your programs

As an example of using functions to declutter a program, we will consider some graphics tasks.

If you look up the `fill` command in the MATLAB help, you will find that it allows you to draw polygons and fill them in.

You draw a polygon by defining the x and y positions of its vertices. The x and y positions are stored in vectors.

Let's consider how we might write a function to draw a square.

Using functions to declutter your programs

Our function might have the following structure for inputs:

```
function [] = Draw_Square(center_location, edge_length, rgb)
```

where

`center_location` is a vector with the x,y location of the center

`edge_length` is a single value, the edge length of the square

`rgb` is a vector of 3 values for color—[0 0 0] is black, [1 1 1] is white. Type `doc fill` in MATLAB for more details.

Using functions to declutter your programs

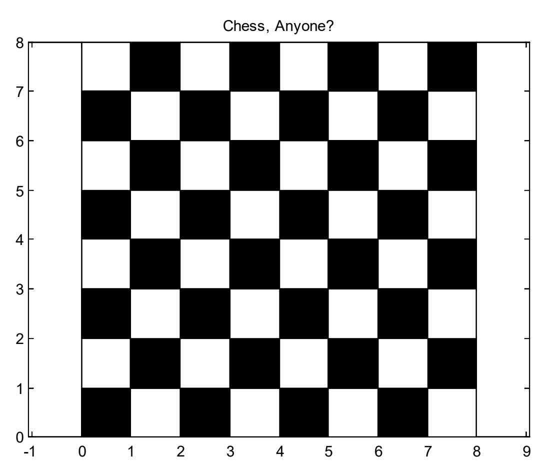
Here is the function:

```
function [] = Draw_Square(center_location, edge_length, rgb)

x_lower_left  = center_location(1) - edge_length / 2.0;
x_lower_right = center_location(1) + edge_length / 2.0;
x_upper_right = center_location(1) + edge_length / 2.0;
x_upper_left  = center_location(1) - edge_length / 2.0;

y_lower_left  = center_location(2) - edge_length / 2.0;
y_lower_right = center_location(2) - edge_length / 2.0;
y_upper_right = center_location(2) + edge_length / 2.0;
y_upper_left  = center_location(2) + edge_length / 2.0;

x_coor = [x_lower_left x_lower_right x_upper_right x_upper_left];
y_coor = [y_lower_left y_lower_right y_upper_right y_upper_left];

fill(x_coor, y_coor, rgb);

axis equal;
```


Exercises

Exercise 1: Write a main program that will create the chess board shown here. Make use of the Draw_Square.m function in your main program.

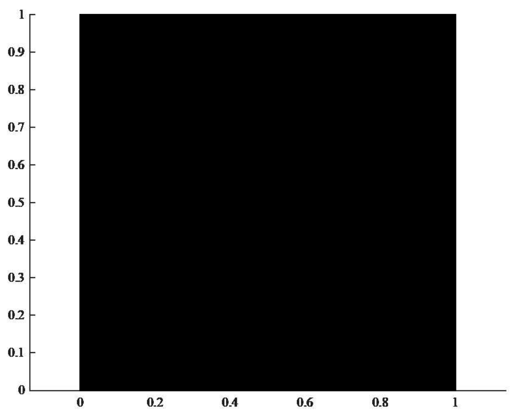
NOTE: The main program would be significantly more cluttered if you had to include all the Draw_Square.m code.



Plotting multiple squares in the same plot

We can plot the 1 × 1 black square in the lower left corner of the chessboard with the following line of code:

```
Draw_Square([0.5 0.5], 1, [0 0 0]);
```

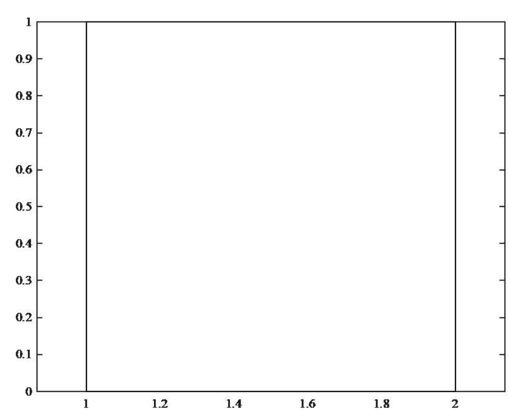


Plotting multiple squares in the same plot

We would expect to then plot the white square on the right by simply adding another `Draw_Square` function call:

```
Draw_Square([0.5 0.5], 1, [0 0 0]);
Draw_Square([1.5 0.5], 1, [1 1 1]);
```

However, the plot of the black square is wiped out and replaced with just the white square:



Plotting multiple squares in the same plot

Use `hold on` to draw both squares in the same plot:

```
hold on
Draw_Square([0.5 0.5], 1, [0 0 0]);
Draw_Square([1.5 0.5], 1, [1 1 1]);
```