Scope – Global vs Local Variables:

By default, variables are *locally* defined. This means that variables only exist within an individual program or function. If you change the value of a variable called **x** within a function, the variable **x** in the main program (if it exists) will not be changed.

In many programs you will write for this class, it often makes sense to use the same variable name in the argument list of the function in both the function sub-program and the calling program. For example,

In main.m:

```
x = 1;
fprintf('y(x) is %7.2f \n' , y(x) )
```

In y.m:

```
function [result] = y(x)
result = x^2 + 2*x + 1;
return
endfunction
```

At command line:

```
> main
y(x) is    4.00
```

However, the following form of **y.m** also is valid:

```
function [result] = y(t)
result = t^2 + 2*t + 1;
return
endfunction
```

If we execute the code again, at the command line:

```
> main
y(x) is    4.00
```

The VALUE of **x** in the main program is passed down to **t** in the function **y**. What if we put a variable **x** in the function **y**?

In y.m:

```
function [result] = y(t)
x = 1000;
result = t^2 + 2*t + 1;
return
endfunction
```

In main.m:

```
x = 1;
fprintf('x before function is %7.2f \n' , x)
fprintf('y(x) is %7.2f\n' , y(x) )
fprintf('x after function is %7.2f \n' , x)
fprintf('t after function is %7.2f \n' , t)
```

At command line:

```
> main
x before function is    1.00
y(x) is    4.00
x after function is    1.00
error: `t' undefined near line 6 column
```

The variables **x** and **t** are *locally* defined. The variable **t** is not defined in the main program, so Matlab/Octave doesn't know what to do.

Variables are *locally* defined by default. It is possible to make *global* variables that are defined in both the main program and function(s), but it is a bit dangerous to use *global* variables and we will not be learning about them in this class. If you want to learn about global variables, refer to your textbook or perform a google search.

In main.m

```
clear; clc;
X = 1; Y = 2; Z = 3;
fprintf('main, before funky: X=%i,Y=%i,Z=%i \n' , X,Y,Z)
[A,B] = funky(X,Y,Z)
fprintf('main, after funky: X=%i,Y=%i,Z=%i \n' , X,Y,Z)
```

In funky.m:

```
% The variables in a function only exist while the function is being
% executed. You can 'resurrect' the variables by calling the function again.
function[W,T] = funky(Y,Z,X)
W = X + Y + Z;
T = X - Y - Z;
fprintf('In funky: X=%i,Y=%i,Z=%i \n' , X,Y,Z)
return
endfunction
```

At the command line:

```
> main
main, before funky: X=1,Y=2,Z=3
In funky: X=3,Y=1,Z=2
A =  6
B = 0
main, after funky: X=1,Y=2,Z=3
```

Arrays as input and output:

Arrays may be used as input or output in exactly the same way as variables (remember that variables are treated as 1x1 arrays in Matlab/Octave). Let's make a function that finds the max value from an array without using the **max( )** function.

In main.m:

```
x = [1, 5, -1, 3, 4];
fprintf('The max value is %f \n' , findmax(x) )
fprintf('The max value is %f \n' , findmax(x(3:5) )  )
```

In findmax.m:

```
% Notice that this function will work regardless of the size of the array x.
% x could even be a variable (1x1 array)
function [max] = findmax(x)
max = x(1)
for i=1:numel(x)
  if(x(i) > max)
    max = x(i)
  end
end
endfunction
```

At command line:

```
> main
The max value is 5.0000
The max value is 4.0000
```

ONLINE BONUS EXAMPLE 1:

Here is an example with multiple arguments (input). The variables **x**, **y**, **z** are used in both the main program and in a function, but are completely unrelated.

In main.m:

```
x = 1.1;
y = 1.2;
z = 1.3;
fprintf('In main program, x+y+z = %f \n' , myFunc(x,y,z) )
fprintf('In main program, x,y,z are %4.2f %4.2f %4.2f \n', x,y,z)
```

In myFunc.m:

```
function [output] = myFunc(a,b,c)
x = 0.1;
y = 0.2;
z = 0.3;
output = a + b + c;
fprintf('In myFunc, a+b+c = %f \n' , output)
fprintf('In myFunc, x+y+z = %f \n' , x+y+z)
return
endfunction
```

At command line:

```
> main
In myFunc, a+b+c = 3.600000
In myFunc, x+y+z = 0.600000
In main program, x+y+z = 3.600000
In main program, x,y,z are 1.10 1.20 1.30
```

Notice how **x**, **y**, **z** in the main program are different from **x**, **y**, **z** in the function.

Remove the lines that define **x**, **y**, and **z** in the function,

In main.m:
```
x = 1.1;
y = 1.2;
z = 1.3;
value = myFunc(x,y,z)
```

In myFunc.m:
```
function [output] = myFunc(a,b,c)
output = a + b + c;
fprintf('In myFunc, x,y,z are %f,%f,%f \n' , x,y,z)
return
endfunction
```

When we try to run this code, we get an error that **x** is undefined in **myFunc.m**. The value of **x** in **main.m** is passed to **a** in **myFunc.m**. The function has no idea what **x, y, and z** are.

ONLINE BONUS EXAMPLE 2:

Here is an example with multiple outputs. Often we want to create functions that calculate multiple quantities.

In main.m:
```
a = 1;
b = 2;
[add,subtract,mult,div] = math(a,b)
```

In math.m:
```
function [ a,s,m,d ] = math (c,d)
a = c+d;
s = c-d;
m = c*d;
d = c/d;
endfunction
```

At command line:
```
> main
add =  3
subtract = -1
mult =  2
div =  0.50000
```