USER-DEFINED FUNCTIONS/SUB-PROGRAMS:

In many programs, a task needs to be completed more than once. For example, you may need to calculate the factorial of a number many times in a program.

Additionally, if a program is very long it sometimes makes sense to break the program into many parts. For example, in air quality modeling there are sub-programs for handling advection, diffusion, chemistry, emissions, deposition, etc.

You are already familiar with the built-in functions such as **cos(x)**, **sqrt(x)**, and **mean(x)**. Often you will want to write your own functions to accomplish tasks that your program needs to do often.

Note: Technically, it is possible to never use functions in your Matlab/Octave code. However, your code may become unnecessarily complicated and lengthy if you do not use them.

You *call* a function in the main program or another function. A list of **arguments** (input) is sent to the function and the function returns some **output**.

With **cos(x)**, the argument **x** (an array) is passed down to the function **cos( )**. The answer (an array) is returned as output

User-defined functions are created in separate M-Files. These M-Files must have the following syntax:

```
function  [output_variables] = function_name(input variables, also called arguments)
commands
return
endfunction
```

(Note: If you only have 1 output variable, which often is the case, you may leave off the brackets.)

Let's create a function $y(x) = x^2 + 2x + 1$ using techniques we have learned in the past.

In main.m:

```
x = input('Enter the value of x: ');
y = x^2 + 2*x + 1;
fprintf('The value of y(x) is: %f \n', y)
```

After execution: (assume the value of 3 is entered by the user)

```
Enter the value of x: 3
The value of y(x) is: 16.000000
```

We can do the same task with a function. The function M-File name must be the same as the name of the function (**function_name**). For example,

In the M-File y.m:

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

There is one argument (input) called **x** and one output variable called **output**. You can call the function **y(x)** in any other M-File as long as function M-file and M-file that calls the function are both in the same directory. For example, in the main M-File (let's call it "main.m"),

In main.m:

```
x = input('Enter the value of x: ');
fprintf('The value of y(x) is: %f \n', y(x))
fprintf('The value of y(x) is: %f \n', y(1))
```

At command line:

```
> main
Enter the value of x: 3
The value of y(x) is: 16.000000
The value of y(x) is:  4.000000
```

IMPORTANT: Both main.m and y.m must be in the same directory!

Each time you call a function, the VALUE of the argument in the calling program (main.m, in this case) is passed down to the argument in the function. In the above example,

- In the first fprintf( ) the VALUE of x (which is 3) is passed down to the variable **x** in the function **y**. This value is used to calculate **result** (which is 16) in the function **y**, which is the output for the function. Once the function is completed, VALUE of the result is passed back to the calling program and displayed to the screen.

- In the second fprintf( ) the VALUE of the argument (which is 1) is passed down to the variable **x** in the function **y**. This value is used to calculate **result** (which is 4) in the function **y**, which is the output for the function. Once the function is completed, VALUE of the result is passed back to the calling program and displayed to the screen.

You can call up multiple functions within a program. Let's create another function z(x) = x-1. First, we need to create another M-file called **z.m**.

In the M-File z.m:

```
function [result] = z(x)
result = x-1;
return
endfunction
```

Now we can use this function whenever we want in the main program as long as **z.m** is in the same directory as **main.m**

In main.m:

```
x = input('Enter the value of x: ');
fprintf('The value of y(x) is: %f \n', y(x) )
fprintf('The value of z(x) is: %f \n', z(x) )
fprintf('The value of y(x) * z(x) is: %f \n', z(x)*y(x) )
x = 2;                                                    % The value of x has changed
fprintf('The value of y(x) * z(x) is: %f \n', z(x)*y(x) )
```

After execution:

```
Enter the value of x: 3
The value of y(x) is: 16.000000
The value of z(x) is: 2.000000
The value of y(x) * z(x) is: 32.000000
The value of y(x) * z(x) is: 9.000000
```

You can call functions as many times as you like, which is the entire point of creating functions in the first place.

Functions with multiple input:

Thus far, we have only examined functions with a single input (or argument). We can pass down many arguments to functions. The surface area of a cylinder is a function of both the height and radius. Let's make a function to calculate the surface area of a cylinder called **SAC**.

In SA_cyl.m:

```
function  [SA] = SAC(h,r)
SA = 2*pi*r^2 + 2*pi*r*h;
return
endfunction
```

In main.m:

```
r1 = 2;
h1 = 1;
fprintf('The surface area of cyclinder 1 is: %7.2f \n' , SAC(h1,r1) )
r2 = 1;
h2 = 1;
fprintf('The surface area of cyclinder 2 is: %7.2e \n ' , SAC(h2,r2) )
```

At command line:

```
> main
The surface area of cyclinder 1 is:   37.70
The surface area of cyclinder 2 is: 1.26e+001
```

Notice that the arguments are labeled **h1**, **r1**, **h2**, and **r2**, not the names of the arguments in the function (**h** and **r**). Remember that we are only sending down the VALUE of the arguments **h1**, **r1**, **h2**, and **r2** to the function.

Functions with multiple output:

Thus far, we have only made functions with a single output. But often we want functions to calculate multiple quantities. Here is a currency converter.

In main.m:

```
dollars = input('Enter number of dollars: ');
yenperdollar = 90;
eurosperdollar = 0.70;
[yen, euros] = convert(dollars, yenperdollar, eurosperdollar);
fprintf('This is %6.1f yen \n', yen);
fprintf('This is %6.1f euros \n', euros);
```

In convert.m:

```
function [yen, euros] = convert (dollars, yenperdollar, eurosperdollar)
yen = dollars * yenperdollar;
euros = dollars * europerdollar;
return
endfunction
```

At command line:

```
> main
Enter number of dollars: 50
This is 4500.0 yen
This is   35.0 euro
```