The **return** command:

When a **return** command is executed in a function, the function is terminated. The output is sent to the calling program immediately. If you omit the **return** command, the function will terminate at **endfunction**.

In main.m:

```
x = 5;
disp( y(x) );
```

In y.m:

```
function [w] = y(x)
if(x > 10)
  w = x + 2;
  fprintf('hello\n')
  return
else
  w = x + 4;
  fprintf('bye\n')
  return
end
w = 1000;
disp('hibye\n')
endfunction
```

At command line:

```
> main
bye
 9
```

Notice that the last two commands in the function did not get executed. As soon as **return** is executed, the function is terminated.

Here is an example of arrays as input and output. Make a function that calculates A .* B, where A and B are arrays, without using the .* operator. The function should check if the dimensions of A and B are the same.

In main.m:

```
A = [1 3 5 ; 3 3 3];
B = [6 6 5 ; 3 2 1];
C = [1 1 ; 4 5 ; 8 9];
disp( MULT(A,B) )
disp('')
disp( MULT(A,C) )
disp('')
disp( MULT(A',C) )
disp('')
disp( MULT(A(1:2,1:2) , C(1:2,1:2) ) )
```

In MULT.m:

```
function[out] = MULT(X,Y)
% if X and Y are the same size...
if( size(X) == size(Y) )
  [row, col] = size(X);
% Multiply each pair of elements
  for i=1:row
    for j=1:col
      out(i,j) = X(i,j) * Y(i,j);
    end
  end
% if X and Y are not the same size...
else
  out = 'Error! ';
end
endfunction
```

At command line:

```
> main
   6   18   25
   9    6    3

Error!

    1    3
   12   15
   40   27

    1    3
   12   15
```

Functions within functions:

You can call functions within functions as long as they are in the same directory. We want to calculate the value of y = 2*<x> + x^2 for every element in an array **x**, where <x> is the average value of **x**, without using the **mean( )** function.

In main.m:

```
x = [2,4,5];
y = myFunc(x);
fprintf('The values of y are: \n')
fprintf('%f \n' , y)
```

In myFunc.m:

```
function [y] = myFunc(x)
avgx = averagex(x);
for i=1:numel(x),
  y(i) = 2*avgx + x(i)^2;
end
return
endfunction
```

In average.m:

```
function [ avgvalue ] = averagex(x)
sum = 0;
for i=1:numel(x),
  sum = sum + x(i);
end
avgvalue = sum / numel(x);
endfunction
```

At command line:

```
> main
The values of y are:
11.333333
23.333333
32.333333
```

Help menu for functions:

If you type **help function_name** at the command line, you can get information about a particular function.

```
> help mean
`mean' is a function from the file c:\Octave\3.2.2_gcc-
4.3.0\share\octave\3.2.2\m\statistics\base\mean.m

 -- Function File:  mean (X, DIM, OPT)
    If X is a vector, compute the mean of the elements of X

        mean (x) = SUM_i x(i) / N
    If X is a matrix, compute the mean for each column and return them
    in a row vector.
… etc.
```

When you make your own function, you can also give information for the user by putting commentary JUST AFTER the function heading. For example, make the following changes to **averagex.m**.

```
function [ avgvalue ] = averagex(x)
% This function calculates the average of the elements in the array x
% mean(x) should do the same thing.
sum = 0;
for i=1:numel(x),
   sum = sum + x(i);
end
avgvalue = sum / numel(x);
endfunction
```

At command line:

```
> help averagex
`averagex' is a function from the file C:\Users\Paul\averagex.m
 This function calculates the average of the elements in the array x
 mean(x) should do the same thing.
```

You can also put sub-functions within the same m-file:

In main.m:

```
x = [2,4,5];
y = myFunc(x);
fprintf('The values of y are: \n')
fprintf('%f \n' , y)
```

In myFunc.m:

```
function [y] = myFunc(x)
avgx = averagex(x);
for i=1:numel(x),
  y(i) = 2*avgx + x(i)^2;
end
endfunction

function [ avgvalue ] = averagex(x)
sum = 0;
for i=1:numel(x),
  sum = sum + x(i);
end
avgvalue = sum / numel(x);
endfunction
```

Functions with no input or output:

Sometimes you may want to make functions that require no input or output. For example if you want to make a function that returns a constant (like pi).

No input example:

In main.m:

```
%%% THE PARENTHESES ARE OPTIONAL IF THERE IS NO INPUT
A = constval();
fprintf('%f',2*A)
```

In constval.m:

```
function [ans] = constval()
ans = 1.23456;
return
endfunction
```

At command line:

```
> main
2.469120
```

No input or output example:

In main.m:

```
score = -5;
if(score<0)
  printwarning()
end
```

In printwarning.m

```
function [ ] = printwarning ()
disp('Warning: Score is negative')
endfunction
```

At command line:

```
> main
Warning: Score is negative
```

Here is an example with functions within functions within functions. If multiple functions are on the same line, they are executed left to right.

In the "main" program:

```
a = [0 3 9 6 0];
b = [1 1 2 3];
c = [3 4 3];
% func1 is calculated first, then func2
w = func1( a(2:4) , a(3:5) ) + func2( b(3) , c(3) );
disp(w)
```

In the m-file containing the function `func1`:

```
function [ out ] = func1 (x, y)
fprintf('In func1 %i  %i\n' , x(1) , y(1) )
out = x(2) + func2( x(2) , y(2) );
endfunction
```

In the m-file containing the function `func2`:

```
function [ out ] = func2 (w, y)
fprintf('In func2 %i  %i\n' , w , y )
out = func3(w) + func3(y);
endfunction
```

In the m-file containing the function `func3`:

```
function [ out ] = func3 (x)
fprintf('In func3 %i\n' , x)
out = x(1)/2;
endfunction
```

At command line:

```
> main
In func1 3  9
In func2 9  6
In func3 9
In func3 6
In func2 2  3
In func3 2
In func3 3
 19
```