**User-Defined functions**

We will categorize functions as follows:

* 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

Thus, although many functions calculate and return values, some do not. Instead, some functions just accomplish a task. There are differences between these three types of functions, including the format of the function headers and also the way in which the functions are called. Regardless of what kind of function it is, all functions must be defined, and all function definitions consist of the ***header*** and the ***body***. Also, the function must be called for it to be utilized. Although functions can be stored in script code files, for now we will concentrate on functions that are stored in their own code files with an extension of .m.

In general, any function in MATLAB consists of the following:

1. The function header (the first line); this has:
   1. the reserved word **function**
   2. if the function ***returns*** values, the name(s) of the output argument(s), followed by the assignment operator (=)
   3. the name of the function (important: this should be the same as the name of the file in which this function is stored to avoid confusion)
   4. the input arguments in parentheses, if there are any (separated by commas if there is more than one).
2. A comment that describes what the function does (this is printed if **help** is used).
3. The body of the function, which includes all statements, including putting values in all output arguments if there are any.
4. **end** at the end of the function.

### User-defined functions that return a single value

The general form of a ***function definition*** for a function that calculates and returns one value looks like this:

functionname.m

function outputargument = functionname(input arguments)

% Comment describing the function

Statements here; these must include putting a value in the output argument

end % of the function

For example, the following is a function called *calcarea* that calculates and returns the area of a circle; it is stored in a file called *calcarea.m*.

calcarea.m

function area = calcarea(rad)

% calcarea calculates the area of a circle

% Format of call: calcarea(radius)

% Returns the area

area = pi \* rad \* rad;

end

A radius of a circle is passed to the function to the input argument *rad*; the function calculates the area of this circle and stores it in the output argument *area*.

In the function header, we have the reserved word **function**, then the output argument *area* followed by the assignment operator =, then the name of the function (the same as the name of the file), and then the input argument *rad*, which is the radius. As there is an output argument in the function header, somewhere in the body of the function we must put a value in this output argument. This is how a value is returned from the function. In this case, the function is simple and all we have to do is assign to the output argument *area* the value of the built-in constant **pi** multiplied by the square of the input argument *rad*.

The function can be displayed in the Command Window using the **type** command.

>> type calcarea

function area = calcarea(rad)

% calcarea calculates the area of a circle

% Format of call: calcarea(radius)

% Returns the area

area = pi \* rad \* rad;

end

The following is an example of a call to this function in which the value returned is stored in the default variable *ans*:

*>> calcarea(4)*

ans =

50.2655

**Functions that return more than one value**

The general form of a function definition for a function that calculates and returns more than one value looks like this:

functionname.m

function [output arguments] = functionname(input arguments)

% Comment describing the function

% Format of function call

Statements here; these must include putting values in all of the output arguments listed in the header

end

In the vector of output arguments, the output argument names are by convention separated by commas.

Choosing New, then Function brings up a template in the Editor that can then be filled in. If this is not desired, it may be easier to start with New Script.

For example, here is a function that calculates two values, both the area and the circumference of a circle; this is stored in a file called *areacirc.m*:

areacirc.m

function [area, circum] = areacirc(rad)

% areacirc returns the area and

% the circumference of a circle

% Format: areacirc(radius)

area = pi \* rad .\* rad;

circum = 2 \* pi \* rad;

end

As this function is calculating two values, there are two output arguments in the function header (*area* and *circum*), which are placed in square brackets [ ]. Therefore, somewhere in the body of the function, values have to be stored in both.

As the function is returning two values, it is important to capture and store these values in separate variables when the function is called. In this case, the first value returned, the area of the circle, is stored in a variable *a* and the second value returned is stored in a variable *c*:

*>> [a, c] = areacirc(4)*

a =

50.2655

c =

25.1327

If this is not done, only the first value returned is retained - in this case, the area:

*>> disp(areacirc(4))*

50.2655

The **help** function shows the comment listed under the function header:

*>> help areacirc*

This function calculates the area and

the circumference of a circle

Format: areacirc(radius)

**Functions that do not return**

The general form of a function definition for a function that does not return any values looks like this:

functionname.m

function functionname(input arguments)

% Comment describing the function

Statements here

end

Note what is missing in the function header: there are no output arguments and no assignment operator.

For example, the following function just prints the two arguments, numbers, passed to it in a sentence format:

printem.m

function printem(a,b)

% printem prints two numbers in a sentence format

% Format: printem(num1, num2)

fprintf('The first number is %.1f and the second is %.1f\n',a,b)

end

As this function performs no calculations, there are no output arguments in the function header and no assignment operator (=). An example of a call to the *printem* function is:

*>> printem(3.3, 2)*

The first number is 3.3 and the second is 2.0

Note that as the function does not return a value, it cannot be called from an assignment statement. Any attempt to do this would result in an error, such as the following:

>> x = printem(3, 5) % Error!!

Error using printem

Too many output arguments.

We can therefore think of the call to a function that does not return values as a statement by itself, in that the function call cannot be imbedded in another statement such as an assignment statement or a statement that prints.