Lecture 12: Functions V: Optional Arguments

Allowing for Optional Arguments in Functions

• It is often useful to allow for optional arguments in functions.
  • For example, the plot function can be used as follows:
    >> plot(X, Y);     % Plots Y values versus X values.
    >> plot(Y);        % Assumes X ranges from 1 up to
                      % the length of array y.
  • Similarly, the zeros function can be used in two ways:
    >> a = zeros(3, 4);    % An array with 3 rows and
                       % 4 columns of zeros.
    >> a = zeros(3);       % Assumes both the rows and
                       % columns are 3.
• To write functions with optional arguments, built-in functions are
  provided to examine the arguments...

Checking function arguments

• There are a number of functions that you can use to check arguments provided to a function:
  
  nargin
  - returns the number of arguments that were supplied to the function.
  - This is a function that takes no arguments.

  nargout
  - returns the number of output arguments that the function result is
    being assigned to in the calling program. This is a function that takes
    no arguments.

  msg = nargchk(minArgNo, maxArgNo, actualArgNo)
  - returns an error message msg if a function is called with too few or
    too many arguments (i.e., if actualArgNo is outside [minArgNo, maxArgNo]).
  - If the number of arguments is within the range an empty string is returned.

Checking function arguments (cont.)

  error(msg)
  - displays the specified error message msg and causes the function to
    exit. Used for “fatal” errors where there is no point (or way of)
    continuing. If msg is an empty string no action is taken.

  warning(msg)
  - displays the specified warning message msg, but allows the function
    to continue execution. Used for non-fatal errors.

  isnumeric(V)
  - returns True (1) if V is a numeric array, False (0) otherwise.

  ischar(S)
  - returns True (1) if S is a character array, False (0) otherwise.
Checking for integers

• Matlab does not provide a built-in function to check whether a value is an integer or not.
• We can use the following code to check if a value is an integer:

```matlab
if round(V) ~= V
    error('Argument is not an integer.);
end
```

Checking if an array matches another

• When Matlab performs relational checks on arrays, it does so point-wise.
• For example:
  ```matlab
  >> [3 4 5 6] == [3 1 5 4]
  ans =
     1     0     1     0
  • If the array elements all match the result of the equality test will be an array of all ones.

The `all` function

• There is a function called `all` that checks if an array contains all ones.
  ```matlab
  all(V)
  • If V is a vector, `all` returns True if all the elements of V are nonzero. If V is a matrix, `all` operates on the columns of V, returning a vector.
• For example:
  ```matlab
  >> all([3 4 5 6] == [3 1 5 4])
  ans =
     0
  ```

An example

Let's make the input to `trianglearea"bullet proof":`

```matlab
function area = trianglearea(w, h)
    % TRIANGLEAREA: A function to find the area of a triangle.
    % Usage: area = trianglearea(width, height)
    % Arguments: width - The width of the triangle.
    % height - The height of the triangle.
    % Returns: area - The area of the triangle.
    % Author: Lecturers
    % Date: August 2009

    area = 0.5 * w * h;
end
```
An example (cont.)

```matlab
% Check for legal number of input arguments.
msg = nargchk(2, 2, nargin);
error(msg);
% If msg is not empty, this will print
% out msg and stop the function.
% If msg is empty, this command is ignored.
if ~isnumeric(w) | ~isnumeric(h)
  % Print an error message and return.
  error('Arguments must be numeric.');
end
if ~all(size(w) == [1, 1]) | ~all(size(h) == [1, 1])
  % Print an error message and return.
  error('Arguments must be scalar.');
end
if w < 0 | h < 0
  % Print an error message and return.
  error('Arguments must be non-negative.');
end
if w == 0 | h == 0
  % Print a warning message, but do not stop
  % execution.
  warning('width or height is zero.');
end
% If we get to this line, we can safely assume
% the input is valid.
area = w*h/2;
end % function trianglearea
```

Checks that Matlab does automatically

- Matlab performs a number of checks automatically:
- The dimensions of matrices match.
- For example:
  ```matlab
  >> [3, 4, 5, 6] == [3, 1, 5]
  ??? Error using ==> ==
  Matrix dimensions must agree.
  ```
- Note however, that Matlab typically allows operations between a matrix and a scalar:
- For example:
  ```matlab
  >> [3, 4, 5, 6] == 3
  ans =
  1  0  0  0
  ```
Other languages’ treatment for un-initialised variables

- Note: C and C++ make no check whether a variable has been initialised or not. If a variable has not been initialised it will simply contain whatever random bit pattern was left over from the last variable that used that bit of memory.

- Java ensures that all variables are initialised with a value of 0 and also warns you if you use a variable that has not been explicitly initialised within your code.

Back to optional arguments...

- Example - the zeros function might start something like this:
  
  ```matlab
  function m = zeros(rows, cols)
  % Generate an error if the number of arguments
  % is not 1 or 2.
  msg = narginchk(1, 2, nargin);
  error(msg);
  if nargin == 1  % Only rows was defined - cols
    cols = rows; % Define cols by assigning it
    % the value of rows.
  end
  % Now proceed with the function as normal.
  ```

Default values for omitted arguments

- Using the value returned by nargin, you can fill in default values for all the missing arguments.

- It only makes sense to allow for optional arguments when there is an obvious default value for arguments that are omitted. Otherwise an error should be generated.