W1005
Intro to CS and Programming in MATLAB

Functions

Fall 2014

Instructor: Ilia Vovsha

http://www.cs.columbia.edu/~vovsha/w1005

Outline

- Functions
- Recursion
- Errors

Functions

- Three Types: *M-file, anonymous, inline*
- The body of a function is just a set of statements (no different from a script)
- 1st line (function-declaration line) defines the function (name & input/output variables)
- Function name and file name should be identical

Functions (rules)

- The keyword is function. There are multiple ways of defining (*M-file*) functions:
  - function < name > (<var>)
  - function < name > (<var1>, <var2>)
  - function < out_var > = <name> (<var>)
  - function [<out_var1>, <out_var2>] = <name> (<var1>,<var2>)
- You can use the keyword end to terminate a function, but this is not required
- The first set of contiguous comment lines after declaration line are the 'help text'

Functions (rules)

- Identified input and output variables are *local* to the function
- Can't pass data back through input variables
- Can make calls with fewer input/output arguments than specified.
- Can't make calls with more
- Function terminates at the last statement unless a 'return' statement is encountered

Functions (example)

```
function  wakeup(N)
if  N  >  10
      pause(N);
      disp('Alarm');
      beep;
end
```

```
function  s = total(A,B)
if  length(A) == length(B)
       s = A + B;
else
       s = A(1) + B(1);
end
```

Functions (arguments)

- It is possible to have a 'variable' number of (input or output) arguments:
  - function < name > (<var>, varargin)
  - function [<out_var1>, varargout] = <name> (<var1>,<var2>)
- varargin, varargout are cell arrays (we discuss these later)
- You can check the number of declared input (output) arguments for the function using nargin, nargout:
  - Example: nargin('mean')

Recursion

- *Recursive function*: function that calls itself
- After last function call: "collect" return values or *unwind the recursion*
- Template:
  - Identify *stopping case* and return value
  - If stopping case is not reached, make call *(recursive call)* to function itself with different arguments
- Examples: Fibonacci numbers, factorial function

Recursion Example (factorial)

- Count the number of ways in which n objects can be permuted (arranged)
- The *factorial* (n!) is defined for a positive integer n as n! = n*(n-1)…2*1
- Special case: 0! = 1 (one way to arrange zero objects)
- Template:
  - If n = 0 or n = 1: Return 1
  - Recursive call: Return n * (n-1)!

Factorial (code)

```
function  v = recfact(n)
if  n  <=  1
     v = 1;
else
     v = n*recfact(n-1);
end
```

Exercise (In Class)

- Functions & matrix manipulation
- Suppose M is a {0,1} matrix that represents a directed graph. If M(i,j) = 1, then there exists a directed edge from i to j. Write a function that takes M as input, and (for every edge) prints the word 'edge' followed by the vertices of the edge. The function should return the total number of edges in the graph in the variable T.

Errors

- General error types:
  - *Syntax*: violation of grammar rules, during compilation
  - *Run-time:* during execution of program
  - Logic: faulty algorithm
- Error messages:
  - Compiler-generated (syntax errors)
  - Self-error-checking (logic errors)

Error Checking

- To abort operation use *error* function
- To report a warning and then continue operation use *warning* function
- Warnings can be turned on and off
- Terminate execution before reaching end of file: use keyword *return*

Factorial (code + error checking)

```
function  v = recfact(n)
if  n  < 0
     error('n must be positive');
elseif n  > 20
     warning('too big %d', n);
end

if  n  <=  1
     v = 1;
else
     v = n*recfact(n-1);
end
```

Try-Catch Blocks

- User-controlled trapping of errors
- Execute (try) some block of code, if no errors, skip to end statement, otherwise execute code after catch statement (catch error)

```
try
    % code block
catch
    % code block
end
```