FUNCTIONS IN MATLAB PROGRAMMING (LECTURE 2)
WHAT WE HAVE LEARNED

Variables
Operations
Conditions
Loops
WHAT WE WILL LEARN TODAY

Function
FUNCTION IN PROGRAMMING

Functions
- “Self contained” modules of codes that accomplish a specific task
- Encapsulate a task
- Program is a collection of functions
Your program

```matlab
... ...
if a > 0
    a = a + 1;
end ...
if res > 0
    disp(num2str(res))
end ...
```

Function block

```matlab
disp('it is a positive num')
x = k + 1
y = k*2
z = k^2
...
...
More than 1000 lines, but the same work
```
SEQUENCE OF OPERATIONS

Your program

... 1
... 3
if a > 0
   
a = a + 1;
end

... 5
if res > 0
   
disp(num2str(res))
end

Function block

disp('it is a positive num')
x = k+1
y = k*2
z = k^2
...
...
More than 1000 lines, but the same work
In MATLAB, each sub-task is coded as a separate `function` for

- Unit testing
- Reusability
- Isolation from unintended side effects

**Function**

- A special type of M-file
- Running in its own independent workspace
- Input and output
MATLAB FUNCTION

Function format

function [outarg1, outarg2, ...] = fname(inarg1, inarg2, ...)
% H1 comment line
% Other comment lines
...
(Executable code)
...
(return)
(end)

Function definition

function [r, theta] = rect2polar (x, y)

Function calling

[r, theta] = rect2polar (x, y)
function distance = dist2(x1, y1, x2, y2)
% DIST2 Calculate the distance between two points
% Function DIST2 calculates the distance between
% two points (x1,y1) and (x2,y2) in a Cartesian
% coordinate system.
%
% Calling sequence:
% distance = dist2(x1, y1, x2, y2)
%
% Define variables:
% x1    -- x-position of point 1
% y1    -- y-position of point 1
% x2    -- x-position of point 2
% y2    -- y-position of point 2
% distance -- Distance between points
%
% Record of revisions:
% Date    Programmer Description of change
% ===    =======                  ==================
% 02/01/07 S. J. Chapman Original code
%
% Calculate distance.
distance = sqrt((x2-x1).^2 + (y2-y1).^2);
end % function distance
FUNCTION CALL EXAMPLE

% Script file: test_dist2.m
%
% Purpose:
% This program tests function dist2.
%
% Record of revisions:
% Date        Programmer    Description of change
% ----        02/01/07      S. J. Chapman    Original code
%
% Define variables:
% ax   -- x-position of point a
% ay   -- y-position of point a
% bx   -- x-position of point b
% by   -- y-position of point b
% result  -- Distance between the points
%
% Get input data.
disp('Calculates the distance between two points:');
ax = input('Enter x value of point a: ');
ay = input('Enter y value of point a: ');
bx = input('Enter x value of point b: ');
by = input('Enter y value of point b: ');

% Evaluate function
result = dist2 (ax, ay, bx, by);
%
% Write out result.
fprintf('The distance between points a and b is %f\n',result);
QUIZ

In the last lecture, you have written down the program that displays prime numbers among the values between 1 and 10000.

Your code has the following structure

```plaintext
while condition
    % check that the given number is a prime number or not
    % if it is a prime number, display the number
end

Make a function for checking the prime number

while var <= 10000
    res = check_prime(var) %return Boolean data
    if res
        disp(var)
    end
end
```
FUNCTION ARGUMENTS

Variables for input and output

Data in variables are passed from the caller to the callee (function)

```matlab
% Get input data.
disp('Calculate the distance between two points:');
ax = input('Enter x value of point a: ');
ay = input('Enter y value of point a: ');
bx = input('Enter x value of point b: ');
by = input('Enter y value of point b: ');

% Evaluate function
result = dist2(ax, ay, bx, by);

% Write out result.
fprintf('The distance between points a and b is %f\n',result);
```

Data in \(\text{ax}, \text{ay}, \text{bx},\) and \(\text{by}\) are passed to the function \textit{dist2}
PASS BY VALUE

Copy of the actual arguments and pass them to the function

**sample.m file**

```matlab
function out = sample(a, b, c)
    fprintf('In sample: a = %f, b = %f %f\n',a,b);
a = b(1) + 2*a;
b = a .* b;
out = a + b(1);
    fprintf('In sample: a = %f, b = %f %f\n',a,b);
```

**test_sample.m file**

```matlab
a = 2; b = [6 4];
fprintf('Before sample: a = %f, b = %f %f\n',a,b);
out = sample(a,b);
fprintf('After sample: a = %f, b = %f %f\n',a,b);
fprintf('After sample: out = %f\n',out);
```
MEMORY SPACE

Base workspace (test_sample)

\[
\begin{align*}
\text{a} &= 2 \\
\text{b} &= [6, 4] \\
\text{out} &=
\end{align*}
\]
MEMORY SPACE

When `sample` is invoked:
* Variables are created
* Arguments (a, b) are copied

Base workspace (test_sample)

```
a = 2
b = [6, 4]
out
```

function workspace (sample)

```
a = 2
b = [6, 4]
c
out
```
MEMORY SPACE

Base workspace (test_sample)

\[ a = 2 \]
\[ b = [6, 4] \]
\[ \text{out} \]

Function workspace (sample)

\[ a = 10 \]
\[ b = [60, 40] \]
\[ c \]
\[ \text{out} = 70 \]
MEMORY SPACE

Base workspace (test_sample)

\[a = 2\]
\[b = [6, 4]\]
\[\text{out} = 70\]

When `sample` is finished

function workspace (sample)

\[a = 10\]
\[b = [60, 40]\]
\[c\]
\[\text{out} = 70\]

* Value in `out` is copied
GLOBAL MEMORY

A special type of memory that can be accessed from any workspace

Used to share data between functions

global X Y Z

Example code

- average.m

```matlab
function avg = average(nums)
    global TOTAL;
    avg = sum(nums)/TOTAL;
end
```

- test_average.m

```matlab
global TOTAL
TOTAL = 10
n = [34, 45, 25, 45, 33, 19, 40, 34, 38, 42]
av = average(n)
fprintf('average value : %fn", av);
```
GLOBAL MEMORY

Base workspace (test_average)

\[ n = [34, \ldots] \]
\[ av \]

Global memory

TOTAL = 10
GLOBAL MEMORY

Base workspace (test_average)

- \( n = [34, ..] \)
- \( \text{av} \)

When \( \text{average} \) is invoked

Global memory

TOTAL = 10

function workspace (average)

- \( \text{nums} = [34, ..] \)
- \( \text{avg} \)
GLOBAL MEMORY

Base workspace (test_average)

\[ n = [34, ..] \]
\[ av = 35.5 \]

function workspace (average)

\[ \text{nums} = [34, ..] \]
\[ \text{avg} \]

When \text{average} is finished

Global memory

TOTAL = 10

* Value in \text{avg} is copied
PERSISTENT MEMORY

A special type of memory that can be accessed only from within the function but is preserved unchanged between calls to the function.

Used to preserve some local information within a function between calls to the function.

`persistent x y z`
function [ave, std] = runstats(x)
% Declare persistent values
persistent n % Number of input values
persistent sum_x % Running sum of values
persistent sum_x2 % Running sum of values squared

% First reset running sums
[ave std] = runstats('reset');

nvals = input('Enter number of values in data set: ');
for ii=1:nvals
    string = ['Enter value ' int2str(ii) ':  ' ];
    x = input(string);
    [ave std] = runstats(x);
end

fprintf('Average = %8.4f; std dev = %8.4f
', ave, std)
MEMORY SPACE

Base workspace (test_runstats)

ave
std

When runstats is firstly invoked

function workspace (runstats)

x = 'reset'
ave = 0
std = 0
msg = ""

Persistent memory

n = 0
sum_x = 0
sum_x2 = 0
MEMORY SPACE

Base workspace (test_runstats)

ave = 0
std = 0

When runstats is finished

function workspace (runstats)

x = 'reset'
ave = 0
std = 0
msg = ""

Persistent memory

n = 0
sum_x = 0
sum_x2 = 0
MEMORY SPACE

Base workspace (test_runstats)

ave = 0
std = 0
Neval = 5
ii = 1

function workspace (runstats)

When runstats is secondly invoked

x = 'Enter value 1:'
ave = 1
std = 0
msg = ""

Persistent memory

n = 1
sum_x = 3
sum_x2 = 9
MEMORY SPACE

Base workspace (test_runstats)

ave = 1
std = 0

When runstats is finished

function workspace (runstats)

x = 'reset'
ave = 0
std = 0
msg = ""

Persistent memory

n = 0
sum_x = 0
sum_x2 = 0
QUIZ

Write a program that takes 10 numbers from a user and compute their 1) **maximum**, 2) **minimum**, 3) **average**, 4) **standard deviation**.

Requirements)

1. **Do not use MATLAB built-in functions** for getting maximum, minimum, average, and standard deviation. Instead, write your own functions for each results. That is, your program has four your own functions.

2. Your program takes numbers from users during the **runtime**.
SPECIAL FUNCTIONS FOR OPTIONAL ARGUMENTS

\textbf{nargin}\n\begin{itemize}
  \item returns the number of actual input arguments that were used to call the function
\end{itemize}

\textbf{nargout}\n\begin{itemize}
  \item returns the number of actual output arguments that were used to call the function
\end{itemize}

\textbf{nargchk}\n\begin{itemize}
  \item returns a standard error message if a function is called with too few or too many arguments
  \begin{verbatim}
  message = nargchk(min_args, max_args, num_args)
  \end{verbatim}
\end{itemize}

\textbf{error}\n\begin{itemize}
  \item displays error message and abort the function producing the error.
  \item This function is used if the argument errors are fatal
\end{itemize}

\textbf{warning}\n\begin{itemize}
  \item displays warning message and continue function execution.
  \item This function is used if the argument errors are not fatal and execution can continue
\end{itemize}

\textbf{inputname}\n\begin{itemize}
  \item returns the actual name of the variable that corresponds to a particular argument number
\end{itemize}
function [mag, angle] = polar_value(x,y)
% Check for a legal number of input arguments.
msg = nargchk(1,2,nargin);
error(msg);
% If the y argument is missing, set it to 0.
if nargin < 2
    y = 0;
end
% Check for (0,0) input arguments, and print out
% a warning message.
if x == 0 & y == 0
    msg = 'Both x and y are zero: angle is meaningless!';
    warning(msg);
end
% Now calculate the magnitude.
mag = sqrt(x.^2 + y.^2);
% If the second output argument is present, calculate
% angle in degrees.
if nargin == 2
    angle = atan2(y,x) * 180/pi;
end
FUNCTION FUNCTIONS

Function whose input arguments include the name of other functions

```matlab
» fzero('cos',[0 pi])
ans =
  1.5708
```

The keys to the operation of function functions

- **eval** evaluates a character string as though it had been typed in command window
  ```matlab
  » x = eval('sin(pi/4)')
x =
  0.7071
  ```

- **feval** evaluates a named function defined by an M-file at a specified input value
  ```matlab
  » x = feval('sin',pi/4)
x =
  0.7071
  ```
function quickplot(fun,xlim)
msg = nargchk(2,2,nargin);
error(msg);

% Check the second argument to see if it has two
% elements. Note that this double test allows the
% argument to be either a row or a column vector.
if ( size(xlim,1) == 1 & size(xlim,2) == 2 ) % ...
( size(xlim,1) == 2 & size(xlim,2) == 1 )
% Ok--continue processing.
n_steps = 100;
step_size = (xlim(2) - xlim(1)) / n_steps;
x = xlim(1):step_size:xlim(2);
y = feval(fun,x);
plot(x,y);
title(['\bf Plot of function ' fun ']');
xlabel('\bfx');
ylabel(['\bf fun ' (x)]);
else
% Else wrong number of elements in xlim.
error('Incorrect number of elements in xlim.');
end
FUNCTION SCOPE

Function scope of ordinary functions

- Current working directory, basically
- Directory on the MATLAB search path
More than one function in a file

- Top function is a normal or primary function, while the ones below it are subfunctions.
- The primary function should have the same name as the file it appears in.
- Scope of a subfunction is restricted to the other functions within the same file.

```matlab
function [avg, med] = mystats(u)
    n = length(u);
    avg = mean(u, n);
    med = median(u, n);

function a = mean(v, n)
    a = sum(v)/n;

function m = median(v, n)
    w = sort(v);
    if rem(n, 2) == 1
        m = w((n+1)/2);
    else
        m = (w(n/2) + w(n/2+1))/2;
    end
```
PRIVATE FUNCTIONS

Functions that are Resided in subdirectories with the special name ‘private’

Visible only to
- Other functions in the private directory
- Functions in the parent directory
Functions that are **defined entirely within the body of another function**, called the host function

Visible only to
- The host function in which they are embedded
- Other nested functions embedded at the same level

Nested function must be terminated with ‘end’
MATLAB FUNCTION SEARCH

Searching sequence of a function
- Nested function in that function?
- Subfunction in the same file?
- Private function?
- Function in the current directory?
- Function in the standard search path?

Hiding functions
- Hide functions as subfunctions or private functions
- Prevent accidental use, conflict with other public functions with the same name
function distance = dist2 (x1, y1, x2, y2)

%DIST2 Calculate the distance between two points
% Function DIST2 calculates the distance between
% two points (x1,y1) and (x2,y2) in a Cartesian
% coordinate system.
%
% Calling sequence:
% distance = dist2(x1, y1, x2, y2)

% Define variables:
% x1 -- x-position of point 1
% y1 -- y-position of point 1
% x2 -- x-position of point 2
% y2 -- y-position of point 2
% distance -- Distance between points

% Record of revisions:
% Date Programmer Description of change
% ===== ======== ==================
% 02/01/07 S. J. Chapman Original code

% Calculate distance.
distance = sqrt((x2-x1).^2 + (y2-y1).^2);
end % function distance
function distance = dist2 (x1, y1, x2, y2)

%DIST2 Calculate the distance between two points
% Function DIST2 calculates the distance between
% two points (x1,y1) and (x2,y2) in a Cartesian
% coordinate system.
%
% Calling sequence:
% distance = dist2(x1, y1, x2, y2)

% Define variables:
% x1 -- x-position of point 1
% y1 -- y-position of point 1
% x2 -- x-position of point 2
% y2 -- y-position of point 2
% distance -- Distance between points

% Record of revisions:
% Date Programmer Description of change
% 02/01/07 S. J. Chapman Original code

% Calculate distance.
distance = sqrt((x2-x1).^2 + (y2-y1).^2);
end % function distance

Help command displays from H1 line until the first blank or the first executable statement

» help dist2
DIST2 Calculate the distance between two points
Function DIST2 calculates the distance between two points (x1,y1) and (x2,y2) in a Cartesian coordinate system.

Calling sequence:
res = dist2(x1, y1, x2, y2)