Rutgers University
School of Engineering

Fall 2011

14:440:127 - Introduction to Computers for Engineers

Sophocles J. Orfanidis
ECE Department
orfanidi@ece.rutgers.edu

week 6

Weekly Topics

Week 1 - Basics – variables, arrays, matrices, plotting (ch. 2 & 3)
Week 2 - Basics – operators, functions, program flow (ch. 2 & 3)
Week 3 - Matrices (ch. 4)
Week 4 - Plotting – 2D and 3D plots (ch. 5)
Week 5 - User-defined functions (ch. 6)
→ Week 6 - Input-output processing (ch. 7)
Week 7 - Program flow control & relational operators (ch. 8)
Week 8 - Matrix algebra – solving linear equations (ch. 9)
Week 9 - Structures & cell arrays (ch. 10)
Week 10 - Symbolic math (ch. 11)
Week 11 - Numerical methods – data fitting (ch. 12)
Week 12 – Selected topics

Textbook: H. Moore, *MATLAB for Engineers*, 2nd ed., Prentice Hall, 2009

Input – Output Processing

input and output functions, **`input, disp`**

saving and loading files and variables, **`save, load`**

formatted screen output, **`fprintf, sprintf`**

file input and output

opening, reading, writing, and saving files

`fopen, fclose, frewind`

`fprintf, fscanf, fgetl, textscan`

reading and writing excel files

reading, writing, playing audio files

image files

MATLAB has a large number of file processing functions for a variety of tasks:

1. file opening, loading, saving
2. text file processing
3. low-level file I/O
4. reading, writing spreadsheets
5. audio and video file processing
6. image & standard graphics files
7. specialized scientific data formats
8. file compression and internet file access
9. XML files

Useful I/O Functions:

```
>> help iofun
>> doc iofun
```

input, disp, num2str

load, save

fprintf, sprintf

fopen, fclose, frewind, fread, fwrite

fscanf, textscan, fgetl, importdata

xlsread, xlswrite

sound, wavread, wavwrite, wavplay, wavfinfo, wavrecord, audioplayer, audiorecorder, auidodevinfo

imread, imwrite, image, imfinfo, im2java

zip, unzip, tar, untar, gzip, gunzip

input/output functions: **disp**, **input**

```
>> x = 10; disp('the value of x is:'); disp(x);
the value of x is:
    10

>> x = input('enter x: ')      % numerical input
enter x: 100                   % 100 entered by user
x =
   100

>> y = input('enter string: ', 's');  % string input
enter string: abcd efg
>> y = input('enter string: ')
enter string: 'abcd efg'
y =
abcd efg
```

prompt string in single quotes

string entered with no quotes
string entered in quotes

```
>> doc disp
>> doc input
```

```
prompt = 'enter a 2x2 matrix A = ';
A = input(prompt)
enter a 2x2 matrix A = [1 2; 3 4]
A =
     1     2
     3     4
```

brackets are required

```
N=3; M=2;
prompt = ['enter a', ...
num2str(N),'x',num2str(M),' matrix A = '];
A = input(prompt)
enter a 3x2 matrix A = [1 2; 3 4; 5 6]
A =
     1     2
     3     4
     5     6
```

using **num2str**

```
>> doc num2str
>> doc int2str
```

saving & loading variables: **save, load**

```
Y = [1 2 3 4; 5 6 7 8];

save('test.dat', 'Y', '-ascii');  % text file
save test.dat Y –ascii;           % equivalent

save test.dat Y;    % binary file test.dat
save Y;             % creates binary file Y.mat
```

```
>> type test.dat


 1.0000000e+000  2.0000000e+000  3.0000000e+000  4.0000000e+000
 5.0000000e+000  6.0000000e+000  7.0000000e+000  8.0000000e+000
```

```
>> doc save
>> doc load
```

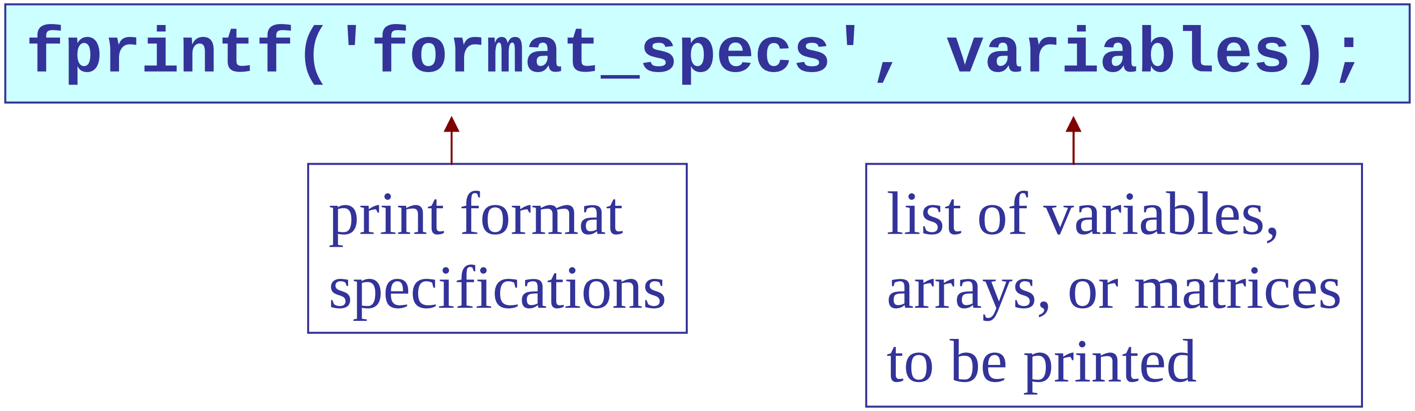
saving & loading variables: **save, load**

```
X = load('test.dat')    % read contents into X
X =
     1     2     3     4
     5     6     7     8
```

```
load test.dat    % creates new variable 'test'
test =
     1     2     3     4
     5     6     7     8
```

the file being loaded must be in the current working directory, or in MATLAB’s path (set/add path from File > Set Path in MATLAB desktop)

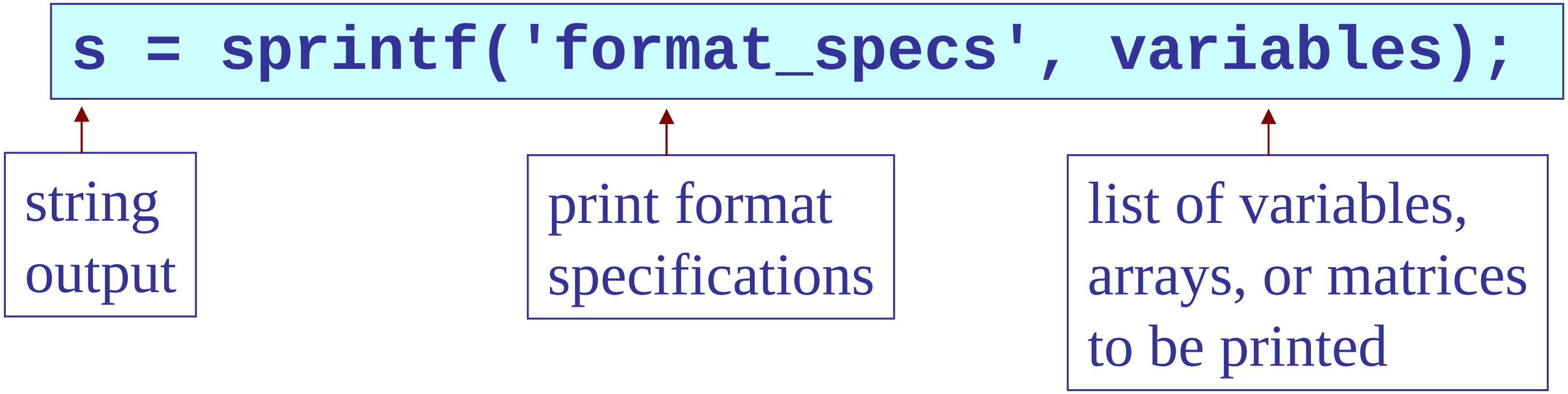
screen output with **fprintf, sprintf**



```
fprintf('format_specs', variables);
```

print format specifications

list of variables, arrays, or matrices to be printed



```
s = sprintf('format_specs', variables);
```

string output

print format specifications

list of variables, arrays, or matrices to be printed

```
>> doc fprintf
>> doc sprintf
>> fprintf('%10.6f\n',  100*pi)
>> fprintf('% 10.6f\n', 100*pi)
>> fprintf('%-10.6f\n', 100*pi)
>> fprintf('%+10.6f\n', 100*pi)
>> fprintf('%10.0f\n',  100*pi)
>> fprintf('%#10.0f\n', 100*pi)
>> fprintf('%010.0f\n', 100*pi)
```

```
314.159265
 314.159265
314.159265
+314.159265
       314
      314.
0000000314
```

```
%10.6f         % width 10, 6 decimal places
% 10.6f        % leave space before field
%-10.6f        % left-justify field
%+10.6f        % print + or – signs
%10.0f         % no decimals
%#10.0f        % print decimal point
%010.0f        % pad with zeros
```

flag

field width
& precision

conversion character:

d, i	integer format
f	fixed-point format
e, E, g	exponential format
c, s	character or string
x	hexadecimal format

there must be as many format specifiers as variables to be printed on each line

```
>> x = 5;
>> fprintf('x = %3.2f x^2 = %3.2f\n', x, x^2);

x = 5.00, x^2 = 25.00
```

printed one column at a time

```
>> x = [5 10 15];
>> fprintf('x = %5.2f, x^2 = %6.2f\n',[x; x.^2]);

x =  5.00, x^2 =  25.00
x = 10.00, x^2 = 100.00
x = 15.00, x^2 = 225.00
```

increase field width to align decimal points

```
>> [x; x.^2]

ans =
      5      10      15
     25     100     225
```

```
>> x = [5; 10; 15];
>> fprintf('x = %5.2f, x^2 = %6.2f\n',[x, x.^2]');

x =  5.00, x^2 =  25.00
x = 10.00, x^2 = 100.00
x = 15.00, x^2 = 225.00
```

printed column-wise

```
>> [x; x.^2]'

ans =
     5    10    15
    25   100   225
```

```
>> [x,x.^2]

ans =
     5    25
    10   100
    15   225
```

```
a = [1; -2; 3; 4;];
b = [10; 20; -30; 40];
c = [100; 200; 300; -400];
```

```
>> [a, b, c]

ans =
     1    10   100
    -2    20   200
     3   -30   300
     4    40  -400
```

need at least **%6.3f** to align first column

```
fprintf('%9.3f %9.3f %9.3f\n', [a, b, c]');
```

```
    1.000    10.000   100.000
   -2.000    20.000   200.000
    3.000   -30.000   300.000
    4.000    40.000  -400.000
```

vectorized version

loop version

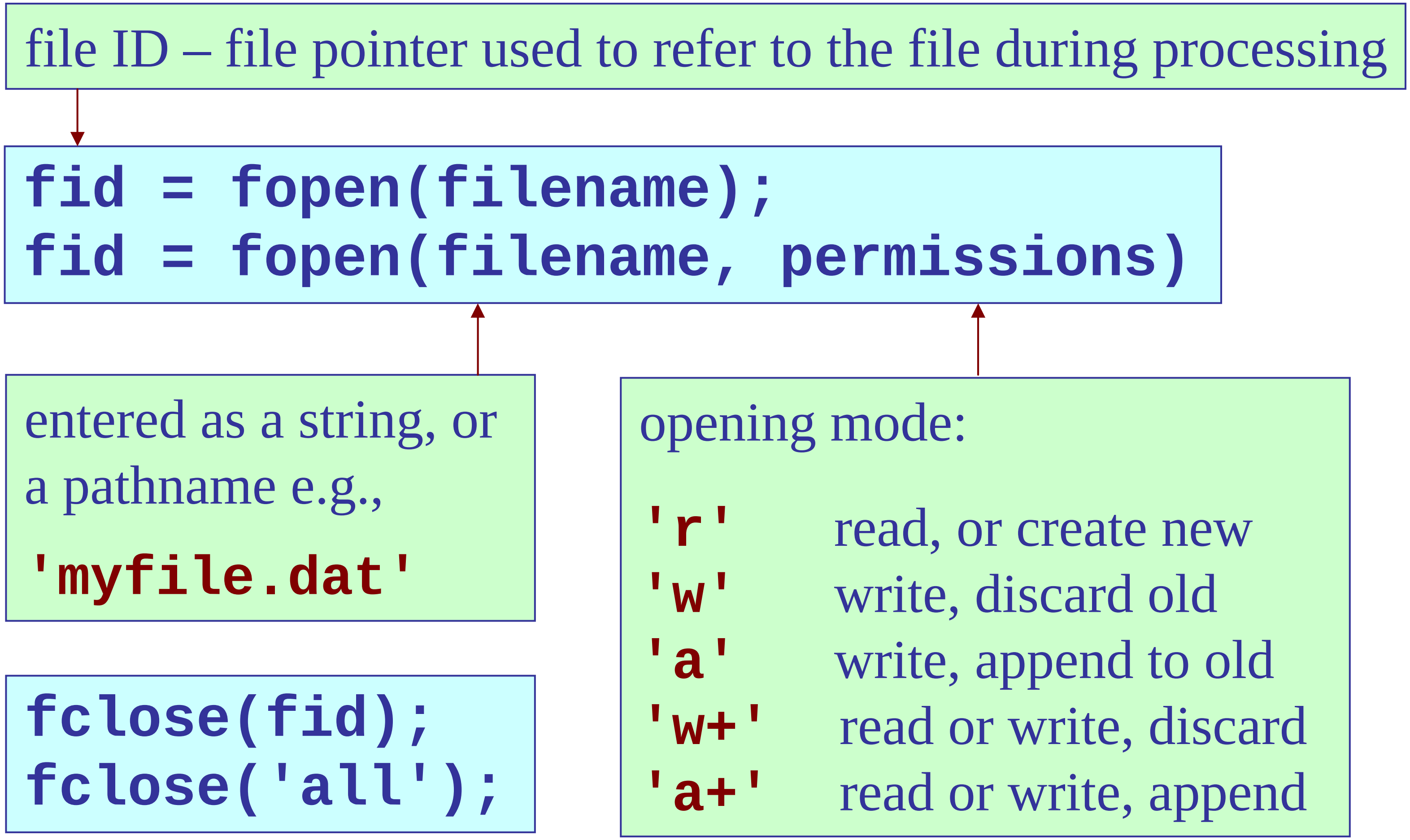
```
for i=1:4,
  fprintf('%9.3f %9.3f %9.3f\n', a(i),b(i),c(i));
end
```

sprintf examples

```
>> x = 5;
>> s = sprintf('x = %3.2f x^2 = %3.2f\n', x, x^2)

s =

x = 5.00, x^2 = 25.00



>> x = [5 10 15];
>> s = sprintf('x=%5.2f, x^2=%6.2f\n',[x; x.^2])

s =

   x= 5.00, x^2=  25.00
   x=10.00, x^2= 100.00
   x=15.00, x^2= 225.00
```

sprintf is useful for producing labels and titles in plots

File input and output – reading and writing files with **fopen, fclose, frewind, fscanf, textscan, fgetl, fprintf**



file ID – file pointer used to refer to the file during processing

```
fid = fopen(filename);
fid = fopen(filename, permissions)
```

entered as a string, or a pathname e.g.,

'myfile.dat'

opening mode:

'r'	read, or create new
'w'	write, discard old
'a'	write, append to old
'w+'	read or write, discard
'a+'	read or write, append

```
fclose(fid);
fclose('all');
```

writing into file with **fprintf**

```
fid = fopen(filename,'w');
fprintf(fid, 'format_specs', variables);
fclose(fid);
```

```
x = [5 10 15];
fp = fopen('test.dat','w');
fprintf(fp, 'x = %5.2f, x^2 = %6.2f\n',[x; x.^2]);
fclose(fp);
```

the file **test.dat** now contains the lines:

```
x =  5.00, x^2 =  25.00
x = 10.00, x^2 = 100.00
x = 15.00, x^2 = 225.00
```

reading data from text file with **fscanf**

```
fid = fopen(filename,'r');
A = fscanf(fid, 'format_specs');
fclose(fid);
```

```
>> fp = fopen('test.dat','r');
>> A = fscanf(fp, 'x = %f, x^2 = %f\n')
>> fclose(fp);
```

A is returned as a column vector

```
A =

     5
    25
    10
   100
    15
   225
```

1st line in file

2nd line

3d line

```
>> reshape(A,2,3)
ans =
     5    10    15
    25   100   225
```

alternative methods of using **fscanf**

```
fp = fopen('test.dat','r');
A = fscanf(fp, 'x = %f, x^2 = %f\n', [2,inf])
fclose(fp);
```

read 2 rows, and indeterminate number of columns

```
A =
     5     10     15
    25    100    225
```

spaces are optional

```
fp = fopen('test.dat','r');
A = fscanf(fp,'%*s %*s %f %*s %*s %*s %f',[2,inf])
fclose(fp);
```

skip over **%*s** fields, read only **%f**

reading data from text file with `textscan`

```
fp = fopen('test.dat','r');
A = textscan(fp, '%*s %*s %f %*s %*s %*s %f')
```

skip over **%*s** fields, read only **%f**, returned in numerical cell arrays

```
frewind(fp);
B = textscan(fp, '%s %s %f %s %s %s %f')
fclose(fp);
```

return all **%s** fields in cell arrays of strings and the **%f** fields in numerical cell arrays

A more complex example: read a file of student names and grades, sort them, save them in a sorted file, re-calculate grades with new weights, sort them, and save them in another file

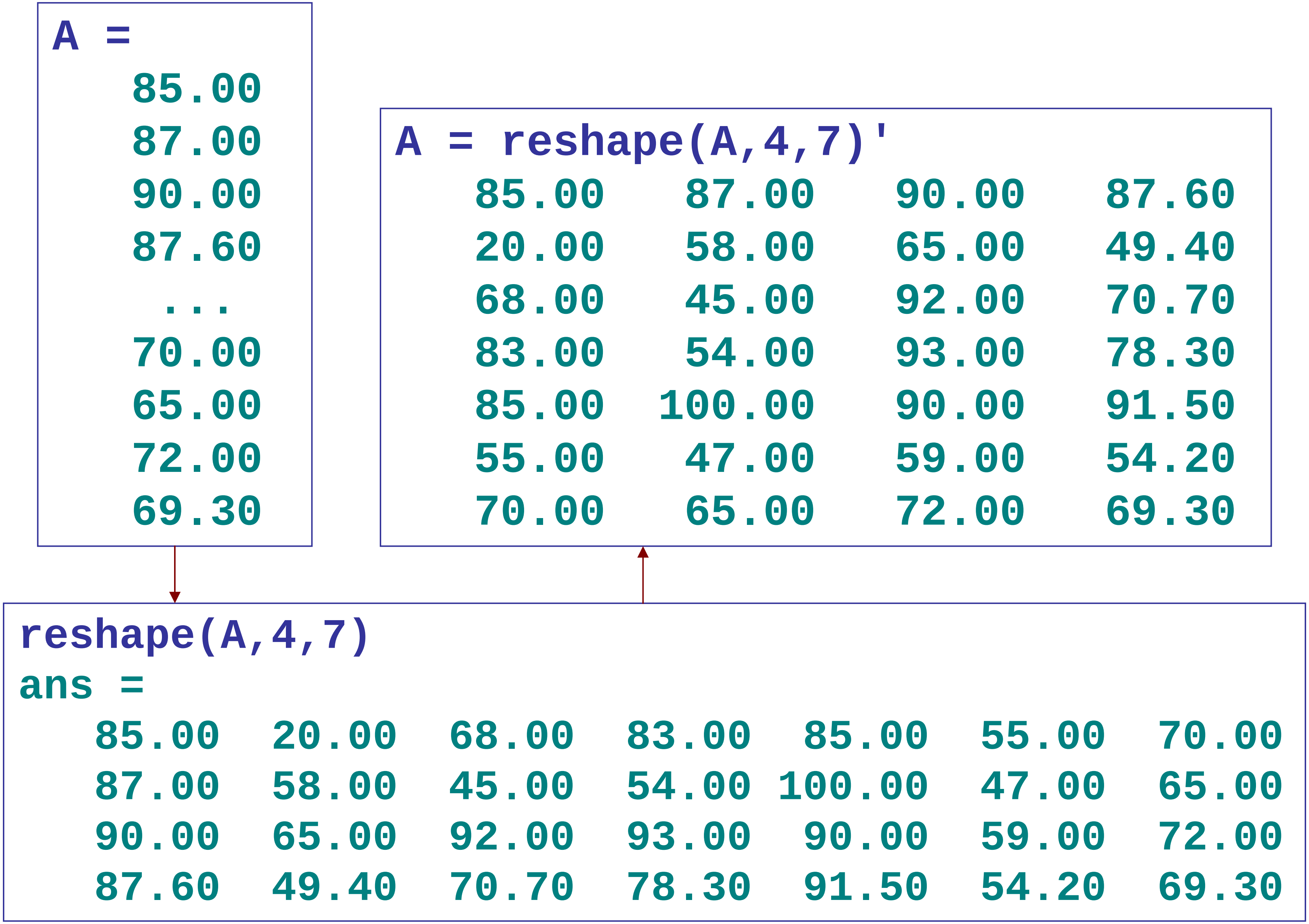
The file **`grades1.dat`** contains the following lines:

```
 Name          E1    E2   E2    AVE    G
-----------------------------------------
Apple,A.       85    87   90   87.60   B+
Exxon,E.       20    58   65   49.40   F
Facebook,F.    68    45   92   70.70   C+
Google,G.      83    54   93   78.30   B
Ibm,I.         85   100   90   91.50   A
Microsoft,M.   55    47   59   54.20   D
Twitter,T.     70    65   72   69.30   C
```

The first two header lines can be skipped over with the help of the **fgetl** (get line) command. For the rest of the file, the first & last columns are strings of unequal length and will be extracted with **textscan** into cell arrays, the numerical columns will be extracted with **fscanf**, and saved in a **7x4** matrix for further processing.

```
 Name          E1    E2   E2    AVE    G
-----------------------------------------
Apple,A.       85    87   90   87.60    B+
Exxon,E.       20    58   65   49.40    F
Facebook,F.    68    45   92   70.70    C+
Google,G.      83    54   93   78.30    B
Ibm,I.         85   100   90   91.50    A
Microsoft,M.   55    47   59   54.20    D
Twitter,T.     70    65   72   69.30    C
```

```
fclose('all');           % close any open files

fp = fopen('grades1.dat');    % open data file

fgetl(fp); fgetl(fp);   % skip two header lines

A = fscanf(fp,'%*s %f %f %f %f %*s');
% read only the %f columns of numbers
% skipping over the %*s fields
% A is returned as a column vector, in which
% every four numbers come from a row of data

A = reshape(A,4,7)';
% reshape A into same shape as the data file
% note the transposition operation

frewind(fp);
% rewind file to its beginning without closing
```



```
A =
     85.00
     87.00
     90.00
     87.60
      ...
     70.00
     65.00
     72.00
     69.30
```

```
reshape(A,4,7)
ans =
   85.00   20.00   68.00   83.00   85.00   55.00   70.00
   87.00   58.00   45.00   54.00  100.00   47.00   65.00
   90.00   65.00   92.00   93.00   90.00   59.00   72.00
   87.60   49.40   70.70   78.30   91.50   54.20   69.30
```

```
A = reshape(A,4,7)'
     85.00    87.00    90.00    87.60
     20.00    58.00    65.00    49.40
     68.00    45.00    92.00    70.70
     83.00    54.00    93.00    78.30
     85.00   100.00    90.00    91.50
     55.00    47.00    59.00    54.20
     70.00    65.00    72.00    69.30
```

```
A =
   85.00   87.00   90.00   87.60
   20.00   58.00   65.00   49.40
   68.00   45.00   92.00   70.70
   83.00   54.00   93.00   78.30
   85.00  100.00   90.00   91.50
   55.00   47.00   59.00   54.20
   70.00   65.00   72.00   69.30
```

```
fgetl(fp); fgetl(fp);   % skip header lines

C = textscan(fp,'%s %*f %*f %*f %*f %s');
% read text %s strings ignoring %f data
% C is 7x2 cell array of strings

N = C{:,1}; G = C{:,2};
% cell arrays of names and letter grades

fclose(fp);             % close file grades1.dat
```

```
>> N                          >> G

ans =                          ans =

    'Apple,A.'                     'B+'
    'Exxon,E.'                     'F'
    'Facebook,F.'                  'C+'
    'Google,G.'                    'B'
    'Ibm,I.'                       'A'
    'Microsoft,M.'                 'D'
    'Twitter,T.'                   'C'
```

```
[av,i] = sort(A(:,4),'descend');
% sort by AVE in descending order
% i = sorting order

As = A(i,:);     % sort A, N, G according to i
Ns = N(i);       % N(i) defines new cell array
Gs = G(i);       % N{i} represents the contents
```

```
fp = fopen('grades2.dat','w');
% create new file for sorted grades

fprintf(fp,'  Name     E1   E2   E2   AVE   G\n');
fprintf(fp,'---------------------------------\n');

for i=1:length(G),
 fprintf(fp, '%-12s %3.0f %3.0f %3.0f %3.2f %-3s\n',...
         Ns{i}, As(i,:), Gs{i});
end
```

cell array

i-th row needs four **%f** fields

i-th entry of cell array

```
fclose(fp);   % close sorted file
```

```
>> type grades2.dat

  Name         E1    E2   E2    AVE    G
-----------------------------------------
Ibm,I.         85   100   90   91.50   A
Apple,A.       85    87   90   87.60   B+
Google,G.      83    54   93   78.30   B
Facebook,F.    68    45   92   70.70   C+
Twitter,T.     70    65   72   69.30   C
Microsoft,M.   55    47   59   54.20   D
Exxon,E.       20    58   65   49.40   F

>> type grades1.dat

  Name         E1    E2   E2    AVE    G
-----------------------------------------
Apple,A.       85    87   90   87.60   B+
Exxon,E.       20    58   65   49.40   F
Facebook,F.    68    45   92   70.70   C+
Google,G.      83    54   93   78.30   B
Ibm,I.         85   100   90   91.50   A
Microsoft,M.   55    47   59   54.20   D
Twitter,T.     70    65   72   69.30   C
```

sorting order

```
i =

     5
     1
     4
     3
     7
     6
     2
```

matrix-vector multiplication

```
w = [1; 1; 1]/3;       % define new weights
AV2 = A(:,1:3)*w;      % compute new weighted average

[AV2, i] = sort(AV2, 'descend');     % sort them

As = A(i,:); Ns = N(i); Gs = G(i);   % sorted grades

fp = fopen('grades3.dat','w');       % open new file

fprintf(fp,'  Name  E1  E2  E2  AVE  G  AV2 G2\n');
fprintf(fp,'----------------------------------\n');

for i=1:length(G),
  G2 = grade(AV2(i));       % map to letter grade
  fprintf(fp, '%-12s  %3.0f  %3.0f  %3.0f  %3.2f ...
%-3s  %3.2f  %-3s\n', Ns{i},As(i,:),Gs{i},AV2(i),G2);
end

fclose(fp);
```

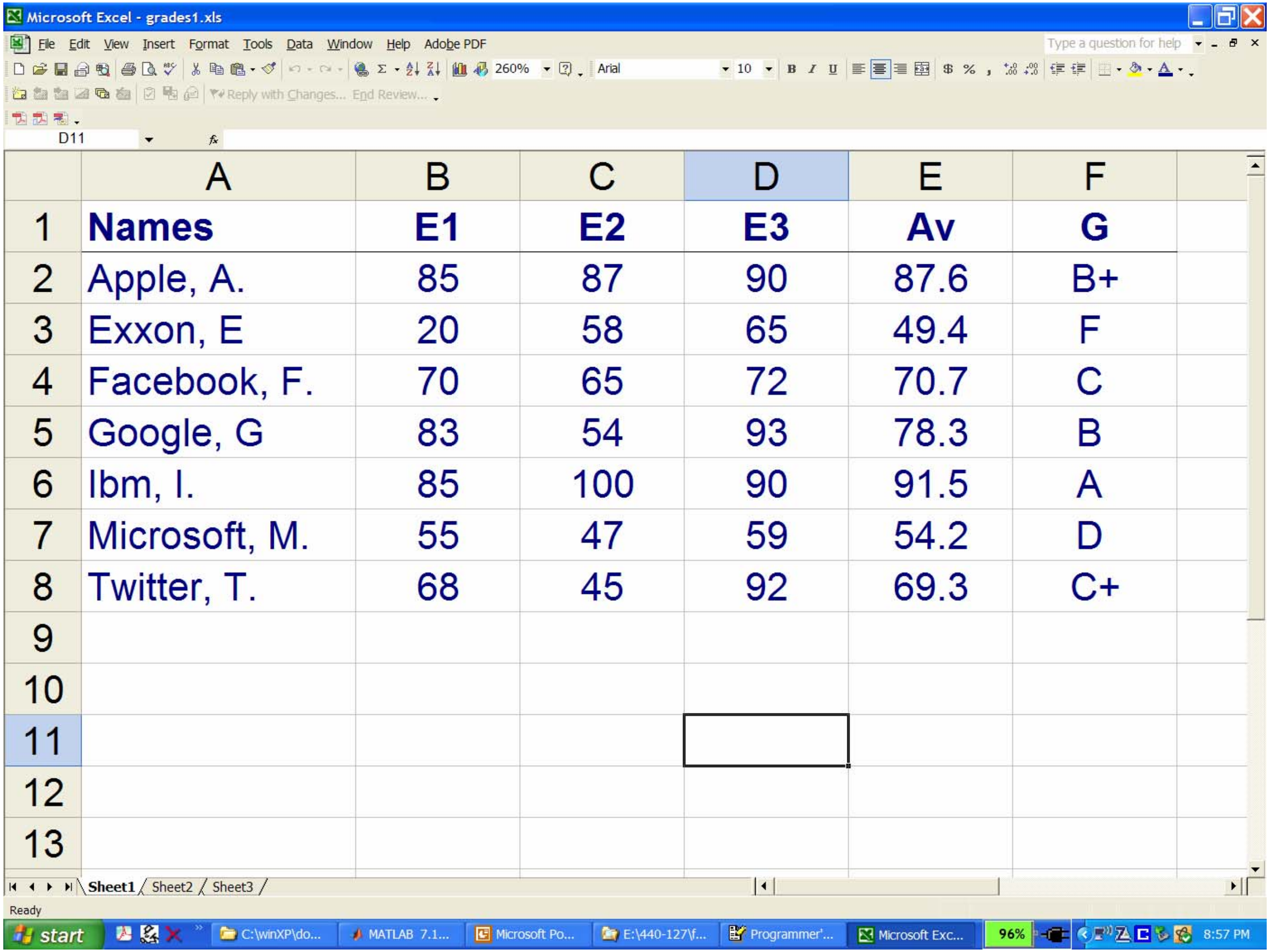
M-file, **grade.m**

```
function G = grade(g)    % letter grade

   if g >= 90,
      G = 'A ';
   elseif g >= 85,
      G = 'B+';
   elseif g >= 75,
      G = 'B ';
   elseif g >= 70,
      G = 'C+';
   elseif g >= 60,
      G = 'C ';
   elseif g >= 50,
      G = 'D ';
   else
      G = 'F ';
   end
```

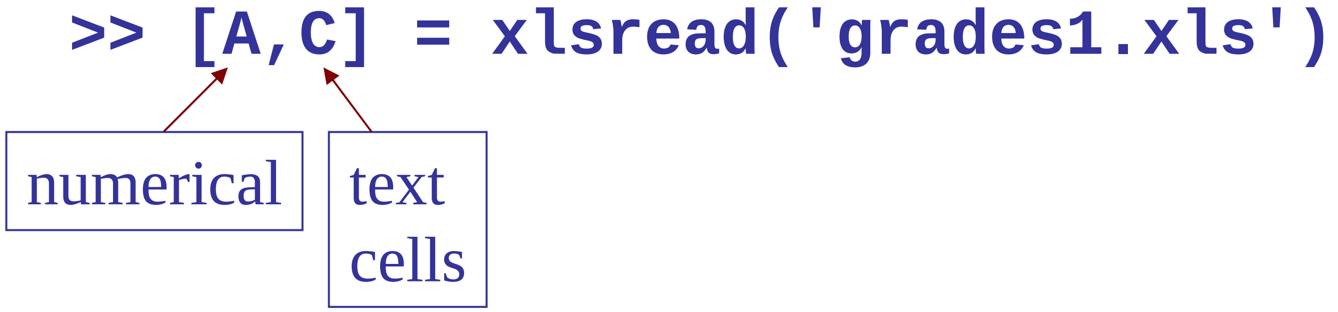
```
>> type grades3.dat

  Name        E1   E2  E2   AVE   G    AV2    G2
------------------------------------------------
Ibm,I.        85  100  90  91.50  A   91.67   A
Apple,A.      85   87  90  87.60  B+  87.33   B+
Google,G.     83   54  93  78.30  B   76.67   B
Twitter,T.    70   65  72  69.30  C   69.00   C
Facebook,F.   68   45  92  70.70  C+  68.33   C
Microsoft,M.  55   47  59  54.20  D   53.67   D
Exxon,E.      20   58  65  49.40  F   47.67   F
```

The zip file **`grades1.zip`** contains the complete source code

reading & writing excel files, `xlsread, xlswrite`



	A	B	C	D	E	F
1	**Names**	**E1**	**E2**	**E3**	**Av**	**G**
2	Apple, A.	85	87	90	87.6	B+
3	Exxon, E	20	58	65	49.4	F
4	Facebook, F.	70	65	72	70.7	C
5	Google, G	83	54	93	78.3	B
6	Ibm, I.	85	100	90	91.5	A
7	Microsoft, M.	55	47	59	54.2	D
8	Twitter, T.	68	45	92	69.3	C+
9						
10						
11						
12						
13						



```
>> [A,C] = xlsread('grades1.xls')
```

numerical

text cells

```
A =

   85.0000   87.0000   90.0000   87.6000
   20.0000   58.0000   65.0000   49.4000
   70.0000   65.0000   72.0000   70.7000
   83.0000   54.0000   93.0000   78.3000
   85.0000  100.0000   90.0000   91.5000
   55.0000   47.0000   59.0000   54.2000
   68.0000   45.0000   92.0000   69.3000
```

```
C =

    'Names'          'E1'  'E2'  'E3'  'Av'  'G' 
    'Apple, A.'      ''    ''    ''    ''    'B+'
    'Exxon, E'       ''    ''    ''    ''    'F '
    'Facebook, F.'   ''    ''    ''    ''    'C '
    'Google, G'      ''    ''    ''    ''    'B '
    'Ibm, I.'        ''    ''    ''    ''    'A '
    'Microsoft, M.'  ''    ''    ''    ''    'D '
    'Twitter, T.'    ''    ''    ''    ''    'C+'
```

```
>> xlswrite('grades2.xls',A);
```

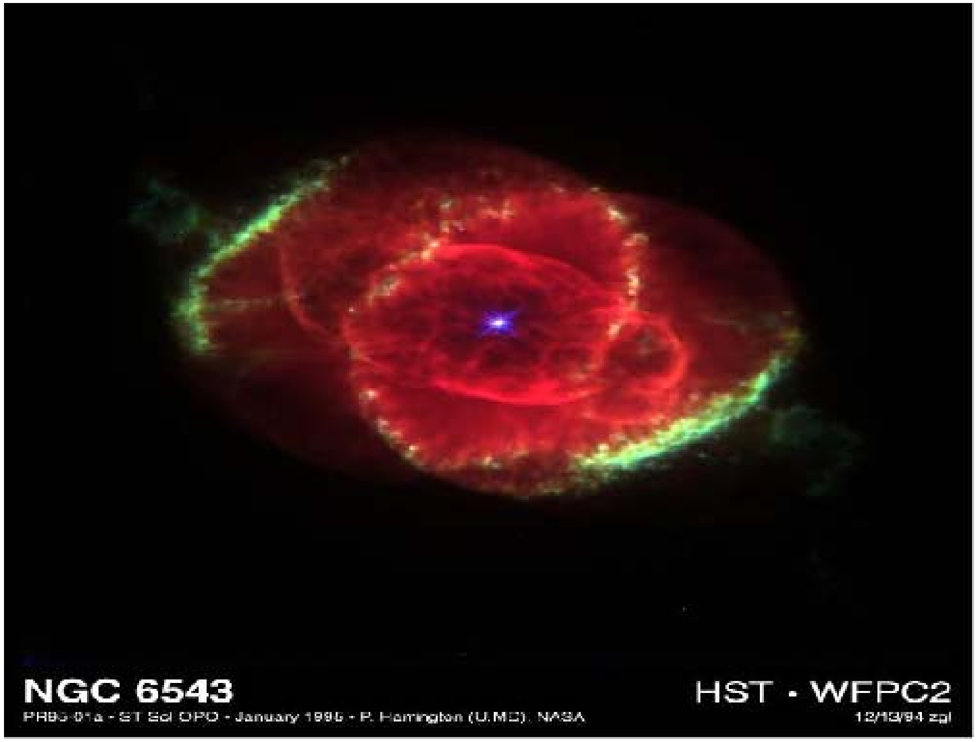
Image Files

```
A = imread(filename, fmt);
[A, map] = imread(filename, fmt);

imwrite(A,filename,fmt);

imfinfo(filename);


fmt: 'jpg', 'jp2', 'png', 'tiff', 'png',
     'gif', 'bmp',  and other
```

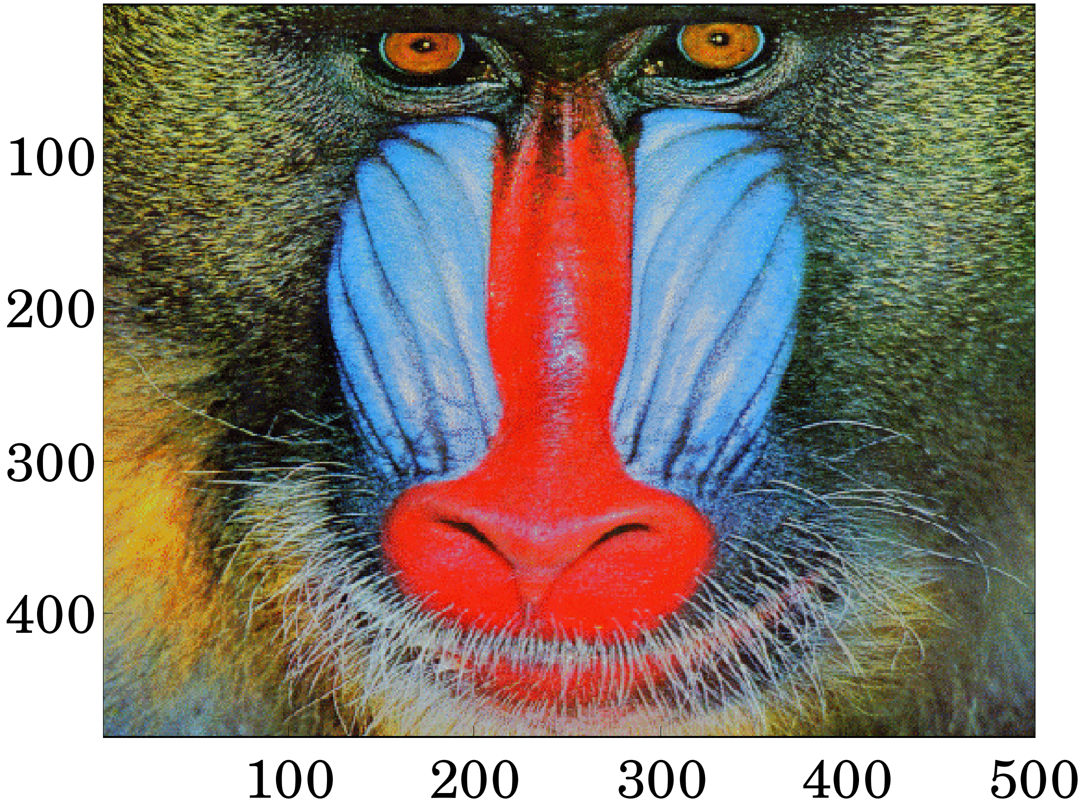
these functions have additional input/output options

```
y = imread('ngc6543a.jpg', 'jpg');
image(y);
title('NGC 6543 Nebula'); axis off;
```

NGC 6543 Nebula


NGC 6543
PRC95-01a · ST ScI OPO · January 1995 · P. Harrington (U.MD), NASA
HST · WFPC2
12/13/94 zgl

```
load mandrill;      % MATLAB demo image
image(X);           % X,map are part of the
colormap(map);      % saved mandrill.mat file
```



```
s1 = 'http://upload.wikimedia.org/';
s2 = 'wikipedia/commons/d/de/';
s3 = 'St_Louis_night_expblend.jpg';
filename = [s1,s2,s3];

y = imread(filename,'jpg');
image(y); axis off;
```

```
'http://upload.wikimedia.org/wikipedia/commons/d/de/St_Louis_night_expblend.jpg'
```

Reading, Writing, Recording, Playing Audio Files

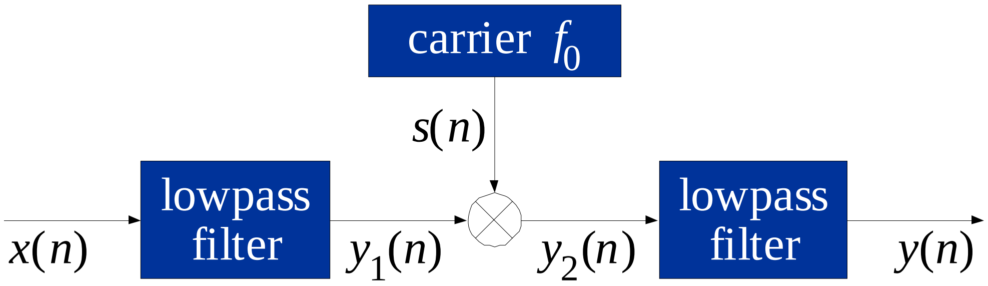
```
[y,fs] = wavread(filename);

wavwrite(y,fs,filename);

y = wavrecord(n,fs);        % n samples
y = wavrecord(N*fs,fs);     % N seconds

sound(y,fs);
wavplay(y,fs);

% typical, fs = 8000, 11025, 22050, 44100
```

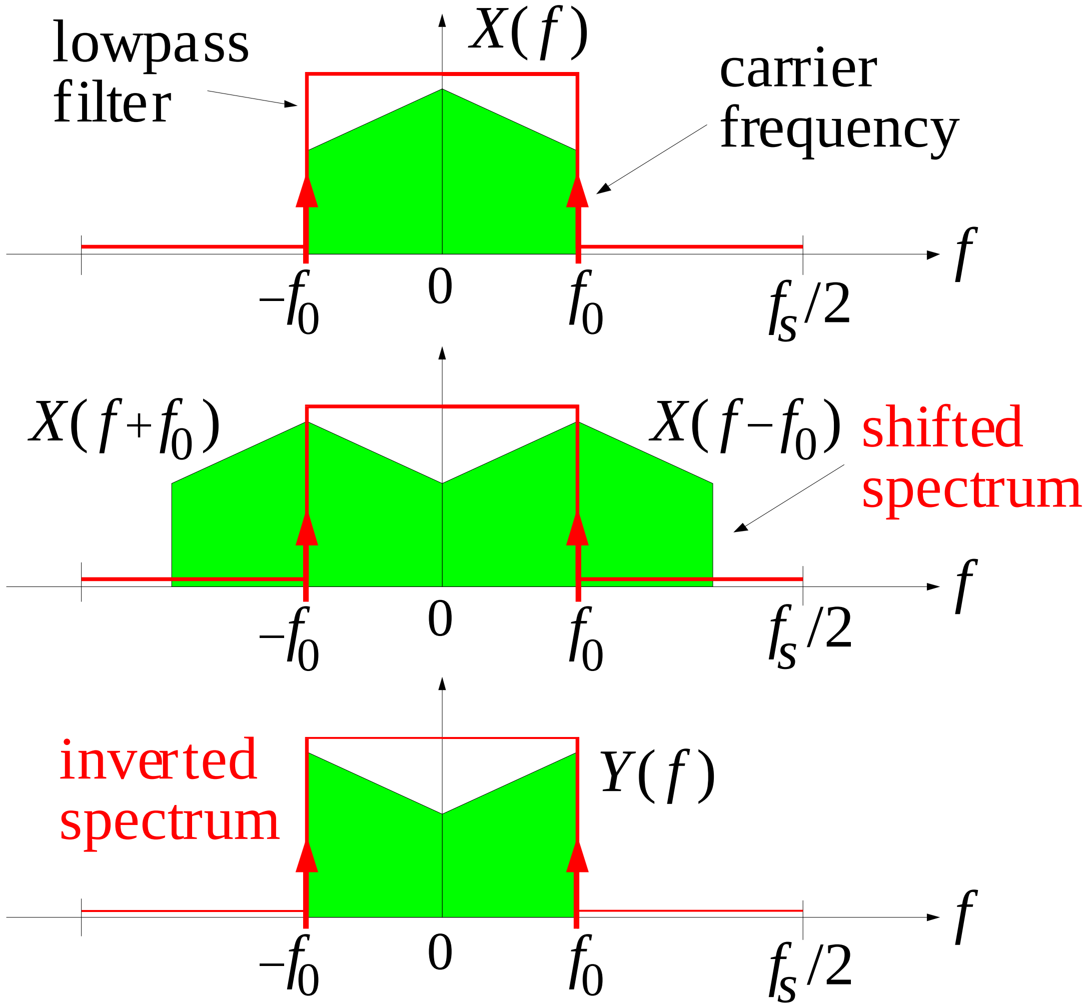
these functions have additional input/output options, see week-2 lecture notes for some examples

Voice Scrambler Example

- reads & plays a wave file
- scrambles it by frequency inversion implemented by lowpass filtering and AM modulation
- plays the scrambled version
- unscrambles it & plays it back

Voice Scrambler




lowpass filter

carrier frequency

shifted spectrum

inverted spectrum

```
% scrambler.m - scrambler example

clear all

fs = 16000; f0 = 3300; w0 = 2*pi*f0/fs;   % filter's cutoff
M = 100; n = 0:M;                         % filter order M=100
w = 0.54 - 0.46*cos(2*pi*n/M);            % Hamming window
h = w .* sinc(w0/pi*(n-M/2)) * w0/pi;     % design filter

[x,fs] = wavread('JB.wav');        % read wave file
sound(x,fs);                       % here, fs=16000

t = (0:length(x)-1)';              % here, length(x)=71472
s = 2*cos(w0*t);                   % sinusoidal carrier

y = filter(h,1,x) .* s;            % scramble by AM modulation
y = filter(h,1,y);                 % and lowpass filtering

pause; sound(y,fs);                % play scrambled file

y = filter(h,1,y) .* s;            % unscramble
y = filter(h,1,y);

pause; sound(y,fs);                % play unscrambled file
```

Record and scramble/unscramble your own voice

Connect a mike at the microphone input of your PC, execute the following MATLAB commands to record your voice for 5 seconds at a sampling rate of 16000 samples/sec, and save the recording in a wavefile **`'test.wav',`** then edit the program **`scrambler.m`** to read this wave file, and run it.

```
fs = 16000;

y = wavrecord(5*fs, fs);

wavwrite(y,fs,'test.wav');
```

`5*fs =` number of samples in 5 sec