Question 1: (15 points)

(a) What is the output from executing the following script? If the program doesn’t terminate or if there will be an error during execution, write the word “error” instead of the output.

```matlab
M = [ 12 16 9 5 7 1; 
    13 17 80 1 5 7; 
    68 66 13 7 1 5];
for r = 1:3
    c = 1;
    d = 6;
    while c<3
        M(r,c) = M(r,d);
        c = c + 1;
        d = d - 1;
    end
end
disp(M)
```

Output:
```
1 7 9 5 7 1
7 5 80 1 5 7
5 1 13 7 1 5
```
First column gets the last column; second column gets the second last column.

(b) Implement the following function as specified:

```matlab
function s = sumMatrix(M)
   % s is the sum of all the values in columns 2,4,6,... of matrix M.
   % M is of type double and has more than three columns.
   % DO NOT USE any built-in functions other than size.
   
   % Example solution:
   [nr,nc]= size(M);
   s= 0;
   for c= 2:2:nc
       for r= 1:nr
           s= s + M(r,c);
       end
   end
```

% Example solution:
Question 2: (10 points)

W is a 1-d cell array of strings. Each cell in W stores one word and all the words in W are different. Write a code fragment below to randomly select and print two different words from cell array W; each word in W should be equally likely to be selected.

```matlab
% Assume W, 1-d cell array of strings as described above, is given.
% W has a length greater than 3.
% ONLY these built-in functions are allowed: length, rand, ceil, floor, strcmp, disp

% Example solution 1: Re-generate 2nd random index until different from first
n = length(W);
idx = ceil(rand(1,2)*n);
while idx(1)==idx(2)
    idx(2) = ceil(rand*n);
end
disp( W{idx(1)} )
disp( W{idx(2)} )

% Example solution 2: Generate 1 index, then reduce the generation space
% from which to generate the second index. This solution solves the problem
% by generating only two random numbers!

n = length(W);
idx = ceil(rand*n);
disp( W{idx} )
idxVec = [1:idx-1 idx+1:n]; % remaining indices
k = ceil(rand*(n-1)); % k is in [1 .. n-1]
disp( W{idxVec(k)} )
```
Question 3: (15 points)

Implement the following function as specified:

function d = darkestInLowerTriangle(A)
    \[ A \] is a 3-d uint8 array of image data. The number of rows in \( A \) is the
    \[ \] same as the number of columns.
    \[ \] The gray value of a pixel is the arithmetic mean of the red, green, and
    \[ \] blue values of a pixel.
    \[ \] \( d \) is the darkest gray value of all the pixels in the lower left triangular
    \[ \] part of the image, including the main diagonal. \( d \) is a uint8 scalar.
    \[ \] DO NOT USE any built-in functions other than size and uint8.

\% Example non-vectorized solution using DOUBLY nested loops:
[\( nr, nc, np \)] = size(A);
d= 255;  % OK to start with type double here since the code below changes
    \% the type to uint8 (since \( A \) and therefore gray has the type uint8)
for r= 1:nr
    for c= 1:r
        gray= \( \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \frac{A(r,c,3)}{3} \);  \% type uint8
        if gray < d
            d= gray;
        end
    end
end

\% Example non-vectorized solution using TRIPLY nested loops:
[\( nr, nc, np \)] = size(A);
d= 255;
for r= 1:nr
    for c= 1:r
        gray= 0;  \% must initialize for each pixel
        for p= 1:np
            gray= gray + \( \frac{A(r,c,p)}{3} \);
        end
        if gray < d
            d= gray;
        end
    end
end

\% Example vectorized solution:
[\( nr, nc, np \)] = size(A);
d= 255;
grayA= \( \frac{A(:,:,1)}{3} + \frac{A(:,:,2)}{3} + \frac{A(:,:,3)}{3} \);
for r= 1:nr
    for c= 1:r
        if grayA(r,c) < d
            d= grayA(r,c);
        end
    end
end
Question 4: (25 points)

Complete the following function as specified. For full credit, do not use vectorized code. Note the example at the bottom of the page.

```matlab
function B = rearrange(A,n)
% A is a 3-d uint8 array of image data and n is a positive integer.
% A is to be divided into n blocks of equal size horizontally. Assume that the number
% of columns in A is divisible by n.
% In every other block beginning with the first block, mirror the pixels row-wise, i.e.,
% flip that block upside down. Do not change the order of the columns and layers.
% B is A rearranged as specified above. B has the same size and type as A.
B = A;
[nr, nc, np] = size(A);
colsInBlk = nc/n;  % number of columns of pixels in a block
% Write your code below. For full credit DO NOT USE vectorized code.
% Example solution 1: Loop over the blocks to be mirrored; calc and loop
% over 1st and last index of each blk w/o built-in functions;
for b = 1:2:n
% Mirror block b
firstCol= (b-1)*colsInBlk + 1;
lastCol= firstCol + colsInBlk - 1;
for p = 1:np
for c = firstCol:lastCol
for r = 1:nr
B(r,c,p) = A(nr-r+1, c, p);
end
end
end
% NOTE: Here's how to calculate the first index of each block using linspace
% firstInd = linspace(1, nc+1, n+1);
% then the c loop above goes from firstInd(b) to firstInd(b+1)-1
% Example solution 2: Use the column loop with a block counter and odd/even check
% to determine whether to mirror
b = 1;
for c = 1:nc
if rem(b,2)==1
for p = 1:np
for r = 1:nr
B(r,c,p) = A(nr-r+1, c, p);
end
end
end
if rem(c,colsInBlk)==0  % This method is not so efficient, since this and the
  % previous if are done at every single col index.
  b = b + 1;
end
end
```

Example: Suppose array A is 5 × 6 × 3 and is to be divided into 3 blocks. Below left is one layer of the original array values in A; on the right is that layer with blocks 1 and 3 flipped row-wise in B. Decompose the problem! Can you flip just one block? How do you flip multiple blocks?
function v = getIndices(str, sep)
  % str and sep are each a string and str is longer than sep. sep is the
  % separator string, i.e., the delimiting string.
  % v is the vector of the indices where the separator begins in str.
  % Therefore the length of v is the number of times that sep occurs in str.
  % Examples: If str is 'Hi!?Ann!?Bob' and sep is '!' then v is [3 8].
  % If str is 'Hi!Ann!Bob' and sep is '?' then v is [3 7].
  % If str is 'Hi!Ann!Bob' and sep is '!' then v is [].
  % Assume that the characters in sep are used only as the delimiter and not
  % in the separated substrings. Assume that separators are always correctly
  % placed--never incomplete and never side-by-side not separating anything.
  % DO NOT USE any built-in functions other than length and strcmp.

  % Example solution with two versions of the necessary loop
  v= []; n= length(sep);

  % Simple for-loop version
  for k= 1:length(str)-n+1
      if strcmp(str(k:k+n-1), sep)
          v= [v k];
      end
  end

  % Slightly more efficient loop, but the difference is small
  k= 1;
  while k<=length(str)-n+1
      if strcmp(str(k:k+n-1), sep)
          v= [v k];
          k= k + n;
      else
          k= k + 1;
      end
  end
Question 5b: (25 points)

Assume that function getIndices of Question 5a has been correctly implemented; make effective use of it in implementing function aveScores below. Note the example at the bottom of the page.

function CA = aveScores(M)
% M is a 2-d array of characters. Each row of M stores the scores of one student:
% a netID followed by one or more scores and these data items are separated by
% commas. There may be trailing spaces in a row of M.
% CA is an n-by-2 cell array where n is the number of students whose record includes
% at least two scores. In each row of CA, the first cell stores the netID of a
% student who has at least two scores and the second cell stores the average score
% of that student. If no student has at least two scores then CA is an empty cell
% array.
% ONLY these built-in functions are allowed: length, size, str2double, sum, mean
% Recall that str2double can handle leading and trailing spaces, e.g.,
% str2double('87 ') returns the type double scalar 87.

% EXAMPLE SOLUTION:
CA= {}; 

[nr,nc]= size(M);

f= 0; % number of students with 2 scores found so far

for r= 1:nr
    commas= getIndices(M(r,:), ',');

    nCommas= length(commas);

    if nCommas>=2
        f= f + 1;
        CA{f,1}= M(r,1:commas(1)-1); % netID string

        % Calculate average score
        commas= [commas nc+1]; % Append vector with an extra index for the
                        % last score that is not followed by a comma
        v= [];
        for k= 1:nCommas
            v= [v str2double( M(r, commas(k)+1 : commas(k+1)-1) )];
        end
        CA{f,2}= mean(v); % or sum(v)/nCommas
    end
end

Example: Suppose M is
[‘vaf34,80,100,90’;...
 ‘aaj91,100’;...
 ‘rt2253,75,95 ’]
Then aveScores(M) should return a 2 × 2 cell array CA:
• In row 1 column 1 is ‘vaf34’ and in row 1 column 2 is the type double scalar 90.
• In row 2 column 1 is ‘rt2253’ and in row 2 column 2 is the type double scalar 85.

Hint: Decompose the problem! For each row of M you need to solve several subproblems: • find the locations of the commas, • convert each score substring to a number, • calculate the average. Organize the subtasks first and then work on them one at a time.