Question 1. (15 points)

(a) What will be printed when the following code is run? Be clear about which things appear on the same line as others. If the program would not terminate, or if an error would occur during execution, write the word "error" in the output box.

```
for a = 4:-2:-1
    b = 1;
    while b <= a - 1
        if a*b > b
            fprintf('%d ', b)
        end
        b = b + b;
    end
    fprintf('%d\n', a)
end
```

Solution:

```
1 2 4
1 2
0
```

(b) What will be printed when the following script is executed?

Script	*Function* (in `tek.m`)
`x= 1;`	`function [h,p] = tek(y,x)`
`y= 3;`	
`[p, x]= tek(x, y);`	`p= y - x;`
`fprintf('x is %d\n', x);`	`h= x + y;`
`fprintf('y is %d\n', y);`	`x= 2*y;`
`fprintf('p is %d\n', p);`	`fprintf('x is %d\n', x)`

Solution:

```
x is 2
x is -2
y is 3
p is 4
```

Question 2. (17 points)

(a) Fill in the loop condition and loop body below so that the code snippet simulates rolling two 20-sided dice and counts the number of rolls it takes until their sum is an even number greater than or equal to 30. The dice are fair with faces numbered 1–20. Do not write any additional code before the loop.

```
roll= 0;  % Sum of last pair of dice rolled (0 if none rolled)
count= 0;  % Number of rolls so far
```

Solution:

```
while roll < 30 || rem(roll, 2) ~= 0
    roll= ceil(20*rand()) + ceil(20*rand());
    count= count + 1;
end
```

```
fprintf('Rolling an even number >= 30 required %d throws\n', ...
        count);
```

(b) Complete the following function as specified, making effective use of a `for`-loop. You may write code before and after the loop if necessary. You may assume that `v` contains at least two elements.

```
function p = biggestProd(v)
% Return the largest product (by magnitude) of two adjacent
%   elements in vector v.  If two such products have the same
%   magnitude but different sign, return the first one.
% Example: If v is [6, -2, -8, 3, 8], then p is -24 (the product
%   of -8 and 3).
```

Solution:

```
p= v(1)*v(2);

for k = 2 : 1 : length(v)-1

    q= v(k)*v(k+1);
    if abs(q) > abs(p)
        p= q;
    end

end
```

Question 3. (22 points)

(a) If you own stock in a company, they may pay you a "dividend" each quarter of the year. The dividend is earned for each "share" of stock that you own at the end of that quarter. Between payouts, you may buy additional shares, and the dividend amount may change based on the company's fortunes.

Implement the following function as specified to keep track of income earned from dividends. For full credit, avoid redundant computations.

```
function [shares, income, total] = stockIncome(shares, buys, div)
% Compute income from stock dividends while acquiring shares.
% `buys(k)` is the number of shares bought in the kth quarter, and
% `div(k)` is the dividend (in dollars) paid _per share_ at the end of
% the kth quarter.  `income(k)` is the income (from dividends) earned at
% the end of the kth quarter from all shares acquired up to that point,
% and `total` is total income earned over all quarters represented by
% the inputs.  When called, `shares` is the number of shares already
% owned before the first quarter, and upon return, `shares` gives the
% number of shares owned after the last quarter.
```

Solution:

```
total= 0;
for k = 1:length(buys)
    shares= shares + buys(k);
    income(k)= shares*div(k);
    total= total + income(k);
end
```

(b) Back in 2018 you decided to start buying stock in PolarPrintersRUs (prior to then you did not own any shares). You kept your financial records organized in separate vectors for each year. **Complete the following script** to compute how much dividend income you earned since the start of your investment through last year and print it to the command window. Your solution should make multiple calls to function `stockIncome()` from part (a) (assume it was implemented correctly); *do not use vector concatenation.*

```
% 2018 shares bought and dividends
buys_2018= [1, 1, 2, 3];  divs_2018= [10, 11, 12, 14];
% 2019 shares bought and dividends
buys_2019= [5, 5, 10, 15];  divs_2019= [14, 16, 18, 22];
% 2020 shares bought (or sold) and dividends
buys_2020= [20, -30, 2, 4];  divs_2020= [0, 2, 3, 5];
```

Solution:

```
shares= 0;
total= 0;
[shares, inc, t]= stockIncome(shares, buys_2018, divs_2018);
total= total + t;
[shares, inc, t]= stockIncome(shares, buys_2019, divs_2019);
total= total + t;
[shares, inc, t]= stockIncome(shares, buys_2020, divs_2020);
total= total + t;
fprintf('Total income: %d\n', total)
```

Question 4. (24 points)

(a) In the game of NumPong, a random real number between 0 and some *upper bound* (initially 100) is generated each round. If that number is below a certain *threshold*, a point is scored, but the threshold is then cut in half for the next round. If the number is ~~above~~ not below the threshold, then the upper bound is cut in half (no point is scored). The game ends after scoring enough points (the *win condition*), and the score for the game is equal to the number of rounds played.

For example, suppose the initial threshold is 20 and the win condition is 2. You generate a random number between 0 and 100; suppose it is 62.8. This is above the threshold, so the upper bound is cut in half to 50. Next, you generate a random number between 0 and 50; suppose it is 8.3. This is below the threshold, so you score a point, but now the threshold is cut in half to 10. You generate another random number between 0 and 50; it is 2.7. You score your second point, ending the game after 3 rounds (earning a game score of 3).

Write a function `numPong()` to simulate playing this game. Your function must accept two input parameters—the initial threshold and the win condition, in that order—and must return the number of rounds played. You do not need to write the function's specification comment for this exam.

Solution:

```
function rounds = numPong(thresh, win)
bound= 100;
points= 0;
rounds= 0;
while points < win
    r= bound*rand();
    if r < thresh
        points= points + 1;
        thresh= thresh/2;
    else
        bound= bound/2;
    end
    rounds= rounds + 1;
end
```

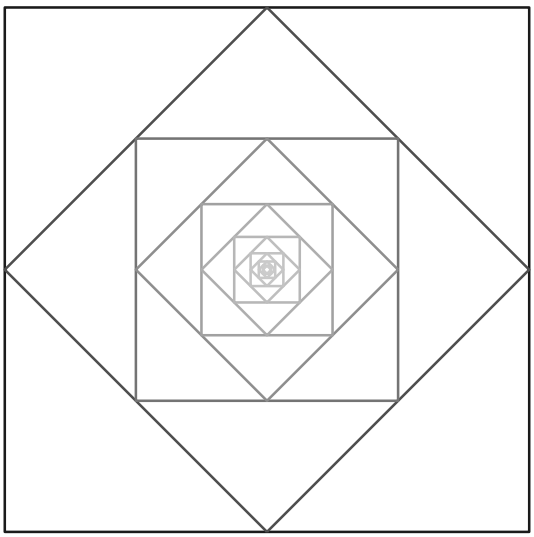
(b) A NumPong match consists of 12 games organized into 3 series. Each series has a different initial threshold: 10, 20, 30. Within each series 4 games are played with win conditions of 2, 4, 6, and 8.

Write a script to simulate a full match of NumPong and to print the scores from all of the games in a table with each series in its own row. Scores from games with the same win condition should line up neatly in columns; you may assume that scores will be no higher than 99. Call your `numPong()` function from part (a) to simulate each game. Do not print row or column headers.

Solution:

```
for t = 10:10:30
    for w = 2:2:8
        fprintf('%3d', numPong(t, w))
    end
    fprintf('\n')
end
```

Question 5. (22 points)



Complete a script to plot nested squares as shown in the figure. The outermost square should have sides of length 100 with its bottom-left corner at the origin $(0, 0)$. Each subsequent square should have its vertices at the midpoints of the sides of the previous one. Stop when the next square to be drawn would have a side length less than 1.

As the squares get smaller, the color used to draw them should get lighter. Specifically, their color should be *linearly interpolated*, based on the length of a side, between `darkGray` (for a side length of 100) and `lightGray` (for a side length of 1).

To draw the squares, make effective use of the following function (do not implement it yourself):

```
function drawPoly(xs, ys, color)
% Draw a polygon with vertices at (xs(k),ys(k)) and RGB color `color`.
% `xs` and `ys` are vectors whose length matches the number of vertices in
% the polygon, and `color` is a vector of length 3.
```

Reminder: the distance between two points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. We've provided the graphics setup for you; write the rest of the script in the space below:

```
axis equal off;    hold on
darkGray= [0.1 0.1 0.1];    lightGray= [0.8 0.8 0.8];
```

Solution:

```
xs= [0, 100, 100, 0];
ys= [0, 0, 100, 100];
s= sqrt((xs(2) - xs(1))^2 + (ys(2) - ys(1))^2);
while s >= 1
    f= (s - 1)/99;  % Relative distance from light endpoint
    c= f*darkGray + (1 - f)*lightGray;
    drawPoly(xs, ys, c);
    x1= xs(1);
    y1= ys(1);
    for k = 1:3
        xs(k)= (xs(k) + xs(k+1))/2;
        ys(k)= (ys(k) + ys(k+1))/2;
    end
    xs(4)= (xs(4) + x1)/2;
    ys(4)= (ys(4) + y1)/2;
    s= sqrt((xs(2) - xs(1))^2 + (ys(2) - ys(1))^2);
end
```

```
hold off
```