CS/INFO 1305 Programming Exercise 2
Due Thursday, July 23, at 1pm

1. Write a *script* to sum the first n terms of the series $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \cdots$. n is a user input value. Name the script `series`.

2. During the previous lab we wrote a *script* to approximate the value of π by simulating dart throws. Convert the script into a *function* `piByDarts` that has one input parameter for the number of darts thrown and returns the value of π estimated in the simulation. Pay attention to the function header and specification (comment).

3. Write a *function* `triangle` to print (in the Command Window) a triangle of asterisks. Each side of the triangle has n asterisks—n is the parameter of the function. This function is supposed to just print a pattern, so there is no value for the function to *return*. Therefore, there should be no output parameter in the function header, as shown below:

```
function  triangle(n)
```

Use nested loops in your function. (Do not call function `printRepeatChar` as we did in class.) Here is example output for $n = 4$:

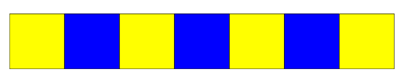
```
*
**
***
****
```

4. Implement the following function:

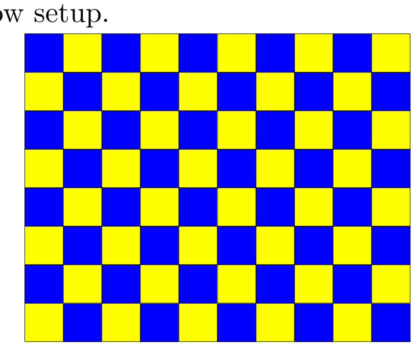
```
function drawRowOfSqrs(n,x,y,s,c1,c2)
% Add to the figure window a row of n adjacent squares.  The lower left
% corner of the first square is at (x,y) and the side length of the square
% is s.  The squares alternate in color, starting with color c1.
```

For example, calling function `drawRowOfSqrs` with the following script will produce the diagram on the right.

```
close all
figure
axis equal off
hold on
drawRowOfSqrs(7,0,0,1,'y','b')
hold off
```



5. Write a script `floorTiles` to draw a 2-color "tile floor" in which adjacent tiles are of different colors. An example of a 10-tile-by-8-tile floor is shown below. Make use of function `drawRowOfSqrs` above! Use the usual figure window setup.



6. Challenge question!! (No need to submit this.) Write a script `floorTiles2` to draw a 2-color "tile floor" in which adjacent tiles are of different colors. This time use *nested loops*. The only user-defined function you can call is `DrawRect`. Enjoy this challenge!

Review

This is a reminder about certain nice properties of *if*-statements and how to cut down on superfluous code. You worked on this in Programming Exercise 1 last week. Suppose you have a *nonnegative* ray angle A in degrees. The following code determines in which quadrant A lies:

```
A = input('Input angle in degrees: ');
A = rem(A, 360);  %Given nonnegative A, result will be in the interval [0,360)

if  A < 90
  quadrant= 1;
elseif  A < 180
  quadrant= 2;
elseif  A < 270
  quadrant= 3;
else
  quadrant= 4;
end

fprintf('Angle %f lies in quadrant %d\n', A, quadrant);
```

Notice that in the second condition, it is **not** necessary to check for `A>=90` in addition to `A<180` because the second condition, in the *elseif* branch, is executed **only if** the first condition evaluates to *false*. That means that by the time the computer gets to the second condition, it already knows that `A` is $\geq$ 90 so writing `A>=90 && A<180` as the second condition would be redundant. Similarly, the concise way to write the third condition is to write only `A<270` as above—it is unnecessary to write the compound condition `A>=180 && A<270`. This is the nice (efficient) feature of "cascading." The same is true for "nesting." If I do not cascade or nest, but instead use independent `if`-statements, then I *must* use compound conditions in some cases, as shown in the fragment below:

```
if  A < 90
  quadrant= 1;
end
if  A >=90 && A < 180
  quadrant= 2;
end
if  A >=180 && A < 270
  quadrant= 3;
end
if  A >=270
  quadrant= 4;
end
```