CS/INFO 1305 Programming Exercise 2
Due Wednesday, July 25, at 10pm
Submit **either** Level 1 or Level 2. For Level 2, problem 2.3 is required; complete ONE of 2.1 and 2.2.

1 Level 1

1. 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).

2. Write a *function* `sumSeries` 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 the function parameter and the function returns the sum.

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 $n = 4$:

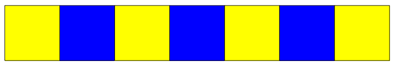
```
*
**
***
****
```

4. Implement the following function:

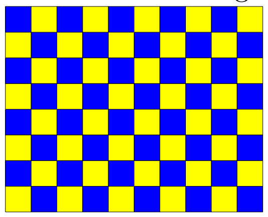
```
function drawRowOfSqrs(n,x,y,s,c1,c2)
% Add to the figure window a row of n adjacent squares.  The lower left
% corner of the first square is at (x,y) and the side length of the square
% is s.  The squares alternate in color, starting with color c1.
% Assume hold is on.
```

To test function `drawRowOfSqrs`, call it with the following script. The diagram on the right should be produced. Try other values for the parameters.

```
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! Solicit the user for the dimensions. Use the usual figure window setup.

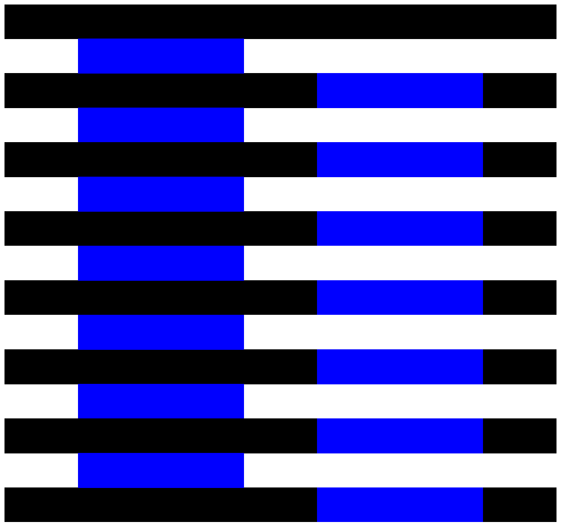


2 Level 2

2.1 The Munker-White Illusion

Go to this website to check out the "Munker-White Illusion":

`http://www.michaelbach.de/ot/lum-white/`



The website has an interactive graphic and gives a short description of the illusion. Use the slider next to the graphic to slide away the "grid rectangles" (black rectangles if you use the default graphic); you will see that the two stacks of colored rectangles (green in the default graphic) actually have the same brightness (luminance)! To the right is a Matlab implementation of the Munker-White Illusion. The two stacks of blue rectangles appear to have different brightness—the illusion is more pronounced if you look at it from a distance—but in fact they were drawn using the same blue color.

You will explore the properties of the M-W Illusion by writing a function. If all we want to do is to draw one illusion, one could just write a script with the dimensions and colors "hard-coded". To faciliate exploration, however, you will write a function whose parameters are the properties that we want to explore. For example, one may want to find out what combinations of color and rectangle dimensions would give a strong illusion and what values would result in a weak or ineffective illusion. Implement the following function as specified:

```
function mwIllusion(n, w, cg, cs, f, a, b)
% Display the Munker-White Illusion in the current figure window.
% The horizontal width of the diagram is w, and the rectangles that span
% the width of the diagram (the grid rectangles) is in color cg, where cg
% is a predefined color name such as 'k', 'b', ..., etc, or an rgb vector.
% Both the height of each rectangle and the vertical space between the
% rectangles are 1.  There are n grid rectangles.  The lower left corner of
% the diagram is at coordinates (a,b).
% There are two "stacks" of n-1 rectangles in color cs, where cs is a
% predefined color name or an rgb vector.  The horizontal width of these
% rectangles is f*w, where f is a fraction less than 0.5.  These two stacks
% are horizontally centered in the diagram, with the same amount of space
% left of the left stack, between the stacks, and right of the right stack.
% Check parameter f:  if f>=.5, set f to .3
```

You will then complete the script `showIllusion` to call function `mwIllusion` three times to produce three different illusions, placed side by side in one figure window. The first two should "show off" the illusion—give a strong illusion of differing brightness in the two stacks of rectangles—while the last illusion should be a weak(er) one. Try different combinations of parameter values!

Specifics and hints

1. Download the m-file `DrawRect.m` from the *Insights* page of the course website. Read the function comments and code. This function draws a rectangle with a black outline. In order to draw the M-W Illusion, you need to modify `DrawRect` to draw a colored rectangle without a black outline. This can be done simply by modifying the last statement in the file, which calls the built-in function `fill`, to be

```
fill(x,y,c,'line','none')
```

Change the function name (and therefore filename) to `DrawRectNoLine`. Make effective use of `DrawRectNoLine` in drawing the illusion!

2. In function `mwIllusion`, use the command `hold on` at the beginning and `hold off` at the end.
3. In the provided (incomplete) script file `showIllusion`, the given code sets up a wide figure window for you to place three M-W Illusions in it side by side.
4. We have been using MATLAB's predefined color names, e.g., 'r' for red. Instead of using a color name, we can specify the exact color vector, e.g., `DrawRect(0,0,1,1,[1 0 0])` will draw a red square because the "rgb vector" [1 0 0] specifies red (full contribution of red, zero green and blue). Take a look at this webpage that shows color swatches and their corresponding rgb values: `http://prideout.net/archive/colors.php` The last column of the table shows the values that should go into the rgb vector.

Think about this . . .

Appreciate this problem as a metaphor. In computational science the sequence of actions

Observe phenomena

↓

Devise a model with parameters to explain/predict the phenomena

↓

Determine the "best" choice for the model parameters using computer simulation

is typical. The problem-solving behind the design of `mwIllusion`—choosing the function parameters—essentially involves the same sequence.

Submit your files `mwIllusion.m` and `showIllusion.m` only. We will use our version of `DrawRectNoLine` when we test your code.

2.2 Hailstones

The Collatz conjecture, named after mathematician Lothar Collatz, states:

> Start with any natural number n. If n is even then divide it by two; otherwise multiply it by three and add one. Repeat the process indefinitely and you will eventually get 1.

The conjecture proposed in 1937 and the sequence of numbers involved, often called the "hailstone sequence," have intrigued many. The conjecture had been tested for all numbers $\leq 191758 \approx 5.4817^{18}$ and 1 was always reached. Every few years there would be rumbling of a successful proof of the conjecture, but in fact the conjecture is still unproven and remains an open question!

Write a script `hailstones` to determine and plot the lengths of the sequences with the starting number n ranging from 2 to 10000. Which value of n gives the longest sequence? (Within the range 2 to 10000 there is only one n that gives the longest sequence.) Here are the first few sequences and their lengths:

n	Sequence	Length
2	2 1	2
3	3 10 5 16 8 4 2 1	8
4	4 2 1	3

Notes on program development:

- For each n, you need the *length* of the sequence. Do not display or try to store the entire sequence. *Hint:* you need a counter.

- Break down the problem! Don't worry about finding the maximum length at first. Also, use **small** n values during initial program development; let n go to 10000 only after successful initial testing.
- Once you are able to calculate (and plot) the lengths, then write more code to determine the maximum length. Do *not* use the built-in function `max`. In fact you don't need it! All you need to do is keep track of the maximum length *so far* as you move from one value of n to the next.

Your script should draw a plot of the sequence length vs the starting value n. To plot a point at position (x,y) using a blue dot as the marker, call the `plot` function like this:

```
plot(x,y, 'b.')
```

Other blue markers include `'b*'`, `'bo'`, `'bx'`, `'bd'`; experiment with them and pick your favorite. You can of course use other colors as well.
The title of your plot should indicate the maximum sequence length and associated starting value n. Use the commands `title` and `sprintf`. Here's an example:

```
maxVal= 1000;
n= 5;
message= sprintf('Start at %d, max sequence length is %d', n, maxVal);
title(message)
```

Use these commands to set up your figure window:

```
close all
figure
hold on
```

Use the command `hold off` at the end (for good programming style). Label the axes.

Submit your file `hailstones.m` in CMS.

2.3 Explorations in interactive graphics

Implement a function `sketchPad` that allows the user to draw rectangles, disks, and stars in the figure window by clicking within the figure window. The basic organization of the program should be as follows:

```
% Display figure window with "buttons" for the three shapes, clearing the figure,
%   and quitting the tool

title('Click your choice below')  % Display instructions on the figure window

[a,b]= ginput(1);  % Accept a click in the figure window.
                   % a,b store the x-,y-coords of the click

while %not quitting

    % Deal with the selected shape (prompt for more input using functions
    % title and ginput, check input and re-prompt as necessary, draw the shape

    % Prompt for next action (draw a shape, clear sketchpad, or quit)
end
```

A "button" is simply a colored/labelled area in the figure that the may click on. Your code will check whether the click was within a "button." The above skeleton is a guideline—you can feel free to deviate from it in order to create a fun tool.

Use the basic figure window setup previously shown and the given functions `DrawRect`, `DrawDisk`, and `DrawStar`. Here's some additional information that will be helpful:

- Command `cla` clears axes area (drawing area) of the figure window
- Use `drawRect` to draw a rectangle to represent the canvas–this fixes the size (range) of the axes area.
- Each button can be as simple as a call to `drawRect` followed by writing in a label using the function `text`:
  `text(x,y, 'buttonname')`
  Hint: Line up the"buttons" across the top or one side of your "canvas" to simplify the code for checking which button has been clicked.
- A function file can contain multiple functions—start each with a function header. The top function is the main function which carries the name of the file; the remaining functions are subfunctions which can be accessed within the file.

Have fun! Submit your single function file `sketchPad.m` to CMS.