CS/INFO 1305 Programming Exercise 1
Due Monday, July 23, at 6pm.
Submit **either** Level 1 *or* Level 2. Do **not** use these commands: `break`, `continue`, `switch`

1 Level 1

1.1 The Earth, an oblate spheroid (a *what*?)

→ a sphere flattened at the poles

The surface area of an oblate spheroid such as the Earth is given by $A = 4\pi r_1 r_2$ where r_1 is the equatorial radius and r_2 is the polar radius. Write a program that calculates and displays the difference between $4\pi r_1 r_2$ and $4\pi((r_1 + r_2)/2)^2$ for Earth data $r_1 = 3963$, $r_2 = 3957$. Call the program `oblateArea`.

1.2 Function evaluation

Write three different programs (scripts) to determine in which quadrant a user-input value of A degrees belong. Assume that the user may enter any non-negative number. For example, 725° is the same, and should be treated, as 5°. (Hint: the function `rem` might be useful.) To avoid ambiguity, we use the following convention:

$$\text{Quadrant is } \begin{cases} 1 & \text{if } \ 0 \le A < 90 \\ 2 & \text{if } \ 90 \le A < 180 \\ 3 & \text{if } \ 180 \le A < 270 \\ 4 & \text{if } \ 270 \le A < 360 \end{cases}$$

Print the result. In the first script use four *separate* `if` statements (4 separate `if-end` constructs) and call the program `angle1.m`. In the second script, use a *single* `if-elseif-else-end` construction for the evaluation and call it `angle2.m`. In the third script, use *nesting* without using the `elseif` clause and call it `angle3.m`. Pay close attention to the differences among the three programs.

1.3 Golden rectangles

The *golden ratio* $\phi = (1 + \sqrt{5})/2$ is one of the most interesting numbers in all of mathematics. For example, the ancient Greeks regarded an L-by-W rectangle with $L/W = \phi$ or $W/L = \phi$ as the most aesthetically appealing rectangle.

Write a script `goldenRect` that randomly generates the length and width of a rectangle. The length and width should be in the range of 1 to 9. If the ratio of the sides (L/W or W/L) is within $\phi \pm 0.2$, draw the rectangle in red. If the ratio is really "unappealing," i.e., greater than 3, do not draw the rectangle. Otherwise draw the rectangle in yellow. Use the given function `DrawRect` (see Lab 1). Use the following statements to set up your figure window before drawing the rectangle (similar to `drawDemo` in Lab 1):

```
close all       % close all previous figure windows
figure          % open a figure window
axis equal off  % use equal scaling in the x- and y-axes; hide the axes
```

How to generate a random value? The statement `v= rand` assigns to variable `v` a random number in the range of 0 to 1. So how do you get a random number within a different range? First, the statement `v= rand` gets you a real number in the range of 0 to 1. Next, scale (think multiply) and shift (think add) the value `v` to get the range you want.

1.4 Stars, disks and rectangles

Write a script `myPoster` that uses the given functions `DrawRect`, `DrawDisk`, and `DrawStar`, to create a figure of your choice! The only requirement is that the script must involve the *meaningful* use of a `for`-loop. It is not necessary to use all three draw functions but you should use at least one. Have fun!

2 Level 2

2.1 My calendar

Write a script `myCal` that prints a one-month calendar. Your script should solicit input for the number of days in the month and the starting day-of-the-week. The output from an example run of the script is shown below (user input is shown in *italics*):

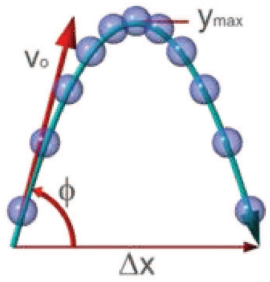
```
Number of days:  31
Starting day-of-the-week (1=Mon, 7=Sun):  2

 Su Mo Tu We Th Fr Sa
        1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30 31
```

For the user's convenience when inputing the starting day-of-the-week, Monday is day 1, Tuesday is day 2, ..., Sunday is day 7. However, for printing the calendar you must use Sunday as the first day of the week. The output dates should line up neatly as shown above, but the format doesn't have to be exactly the same.

Hints: (1) Use a `for`-loop to count from 1 to n where n is the number of days. (2) You need to set an appropriate "test" to determine when to begin a new line.

2.2 Trajectory of a golf ball



In your physics course you might have studied the motion of a projectile. Now you will write a simulation to plot the trajectory of a golf ball in flight, subject to air drag (resistance). Without air drag, one expects the trajectory to be a parabola due to Earth's gravity, with equal time for ascending and descending, as shown in the diagram on the right. What is the effect of air drag? You will find out!

We consider the golf ball to be a unit mass and air resistance to act in the opposite direction of motion. Further we assume that air resistance is proportional to the square of the velocity of the projectile. With these assumptions, the relevant equations are

$$\begin{aligned} v_x &= dx/dt \\ v_y &= dy/dt \\ dv_x/dt &= -k \cdot v_x \sqrt{v_x^2 + v_y^2} \\ dv_y/dt &= -k \cdot v_y \sqrt{v_x^2 + v_y^2} - g \end{aligned}$$

where v_x and v_y are the components of the velocity in the x- and y-directions, k is the coefficient of air drag, and g is the gravitational constant. The golf ball is initially at $x = 0$ and $y = 0$ and is launched with some initial velocity at an angle ϕ measured from the x-axis.

Download the file `golfBall.m` from the *Projects* page. Read and run it. Since the simulation code is missing the output includes only a figure window with axis labels and "dummy values" printed to the Command Window. You will complete the simulation and replace the dummy values with the actual values calculated in the simulation.

Here are the details of our simulation:

- The constants, parameter values, and initial conditions are given in the provided code. Develop your simulation with the given values but you can (and will) experiment with different values later.
- The simulation begins with time $t = 0$ and position $x = 0$ and $y = 0$ and ends when the golf ball lands (y gets back to, or passes, zero) or the maximum allowed simulation time has been reached, whichever happens first.
- Given the initial velocity v and launch angle ϕ, the initial velocities in the x- and y-directions are $v_x = v \cos\phi$ and $v_y = v \sin\phi$

- We use *differencing* to represent the (continuous) derivatives. For example, instead of $\frac{dx}{dt}$ we consider

$$\frac{x_{\text{new}} - x_{\text{current}}}{\triangle t}$$

where $\triangle t$ is a discrete time step. With this discrete representation, at each time step, i.e., each step of the simulation, we can compute the new velocities and positions as follows:

$$\begin{aligned} v_{x\text{new}} &= v_x - \triangle t \cdot k \cdot v_x \sqrt{v_x^2 + v_y^2} \\ v_{y\text{new}} &= v_y - \triangle t \cdot (k \cdot v_y \sqrt{v_x^2 + v_y^2} + g) \\ x_{\text{new}} &= x + v_x \cdot \triangle t \\ y_{\text{new}} &= y + v_y \cdot \triangle t \end{aligned}$$

- Starting at $t = 0$ and after each time step, plot the location of the golf ball. Recall that the command `plot(a,b,'ro')` draws a marker (red circle) at position (a,b). Choose any marker or color you like. Add a pause of 0.01 seconds (`pause(.01)`) after the plot command so that the plotting plays like a movie when the simulation is executed.

  Run the code you have so far to make sure that the simulation works! At this point you should have removed all the dummy values except perhaps those for variables `ascendTime` and `descendTime`. Does the trajectory make sense (a curve that opens down, starting and ending at around $y = 0$)? Do the displayed values in the Command Window look correct?

- Add more code to the simulation to compute the durations of ascent and descent. Do this using code based on the simulation here—do not look up equations from textbooks! Make sure all the dummy variable values are removed.

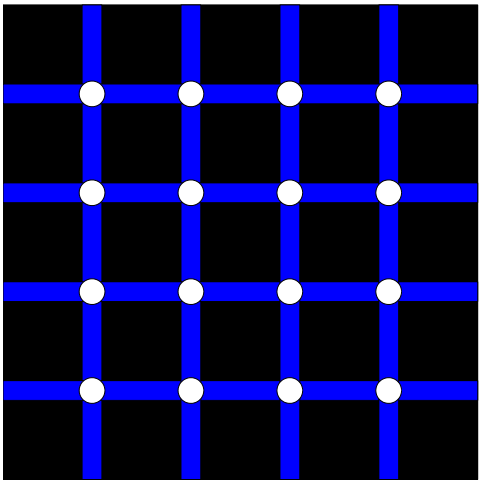
- Finally, modify the parameter values `phi`, `v`, `k`, and `maxTime` to see how the trajectory changes. Once you change the parameter values, the complete trajectory may not be shown since the axes are set to 120m wide and 100m tall and `maxTime` may be "too short," but you can use the values displayed in the Command Window to deduce the trajectory. Add a comment at the end of the script to answer these questions:

  1. Which of these launch angle results in the longest horizontal range, $\pi/3, \pi/4, \pi/5, \pi/6$ (without changing the other parameter values)?
  2. How does the flight time change with the launch angle? Answer in one sentence.
  3. What is the shape of the trajectory when $k = 0$?

  Answer the above questions simply by *running the simulation* with different parameter values. Do not write more code.

Before submitting your file `golfBall.m` on CMS, change all the parameter values back to the original values given: `k=0.02`, `maxTime=10`, `phi=pi/4`, and `v=100`.

2.3 The "Scintillating Grid"



Are your eyes playing tricks on you? Are some of those white disks in the diagram *flickering*? The diagram on the left is an optical illusion called the "Scintillating Grid." If you focus on a disk at a particular intersection, the neighboring disks appear to flicker. Cool! Now stop staring at it ...

Write a script `scinGrid` that solicits n, the number of squares along each side, and draws the "Scintillating Grid." Assume that n is a positive integer value greater than 1. The example on the left has $n = 5$. Approximate the proportion that you see in the diagram and experiment with different colors to find your favorite scheme (that shows off the illusion).

Parameterize your code, i.e., identify the main properties (values) that define the diagram and name them as variables—parameters. Furthermore, choose *one* property to be the "root" and make the other properties dependent on it. For example, let's say that I see two important properties (there are more in this problem): gap width and square length. Instead of "hard coding" the variable values as `g=1; s=4;` I should make one variable dependent on the other, e.g., `g=1; s=4*g;` . Parameterization is an important concept in design and computational engineering. Choosing parameters wisely allows you to easily experiment with different values to "tune" your model and keeps your code easily maintainable.

Use the provided functions `DrawRect` and `DrawDisk`. Download the files from the course website and put them in the same folder as your script `scinGrid`.

Graphics note: Use the typical figure window setup that you've seen in previous graphics examples. Recall that `axis equal off` gives equal scaling in the horizontal and vertical axes and hides the axes labels.