15-112, Week 4, Graphics Practice

See how many of these problems you and a partner can solve during today's lab.

There are three types of problems today:

- For code tracing, solve the problem on paper with a pencil. When you finish, run the code to check your answer.
- The fill in the blank, check your answer with some teams around you to see how you did.
- For free response, write the requested function, then test it by actually running it (call it from `redrawAll`), and check the picture by eye. Try resizing the window for every problem: if a shape jumps to the wrong place or stops filling the canvas when you resize, you probably hardcoded a pixel value somewhere instead of computing it from `app.width` and `app.height`.

Setup: Installing `cmu_graphics`

You already have Python and Thonny installed, so you don't need the full CMU CS Academy desktop app — just the graphics package itself:

1. Go to **academy.cs.cmu.edu/desktop** and download the zip file for your operating system.
2. Unzip it. Inside you'll find a **`cmu_graphics`** folder (the package itself).
3. Pick (or make) a folder on your computer where you'll keep your lab files this week — for example, a `lab04` folder. Put the **`cmu_graphics` folder** directly *inside* it.
4. Inside the `lab04` folder, create a skeleton file `skel.py` containing the following:

```
from cmu_graphics import *

def onAppStart(app):
    pass

def redrawAll(app):
    pass

runApp()
```

This is your starting template: for each problem below, make a copy of it, rename it (e.g. `drawTicTacToe.py`), and fill in your code.

6. Open one of those files in Thonny and hit Run. If a window pops up, you're set up correctly. (The `cmu_graphics` folder has to live in the same directory as whatever `.py` file you're running, since that's how Python finds it to import.)

Code Tracing Questions

For each of these, don't run the code — predict on paper exactly what gets drawn (shapes, positions, sizes, colors, in the correct stacking order), *then* run it to check yourself.

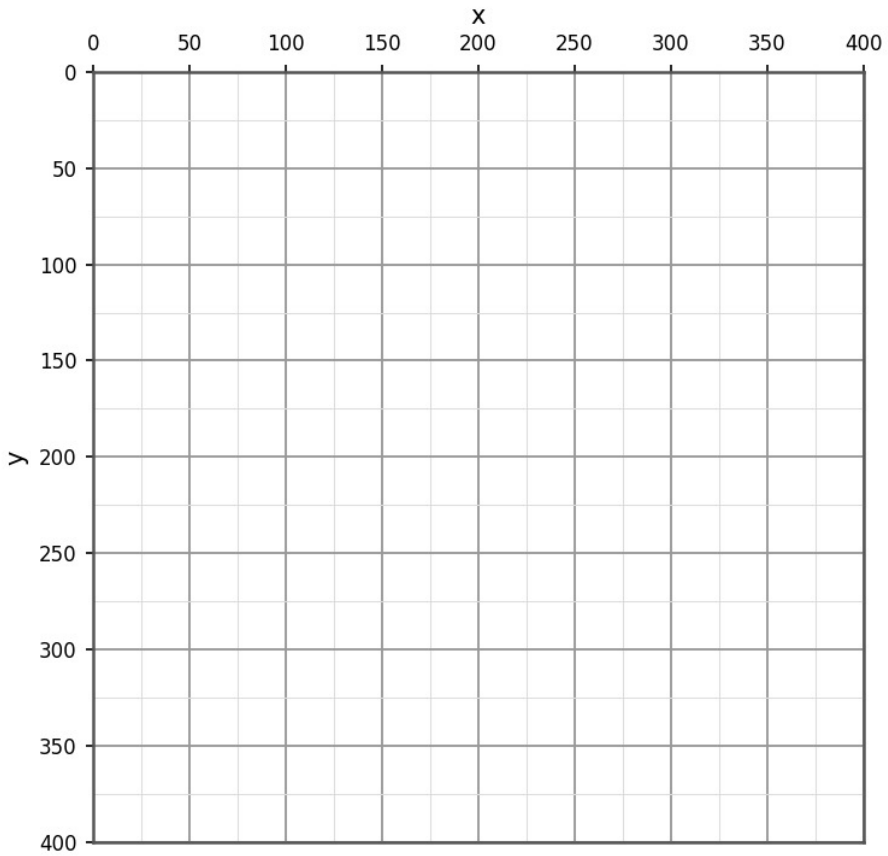
Use a pencil when solving these problems.

Trace #1

Assume the canvas is 400x400.

```
def redrawAll(app):
    step = app.width//4
    for i in range(2):
        for j in range(2):
            color = "white" if (i + j) % 2 == 1 else "black"
            drawCircle(step + 2 * step * i, step + 2 * step * j, step / 2, border="black", fill=colo
    drawRect(step, step, 2 * step, 2 * step, border="black", fill="white")
    drawLine(0, app.height, app.width, 0)
    drawLine(step, step, 2 * step, 2 * step)
```

Sketch your answer on the grid below:

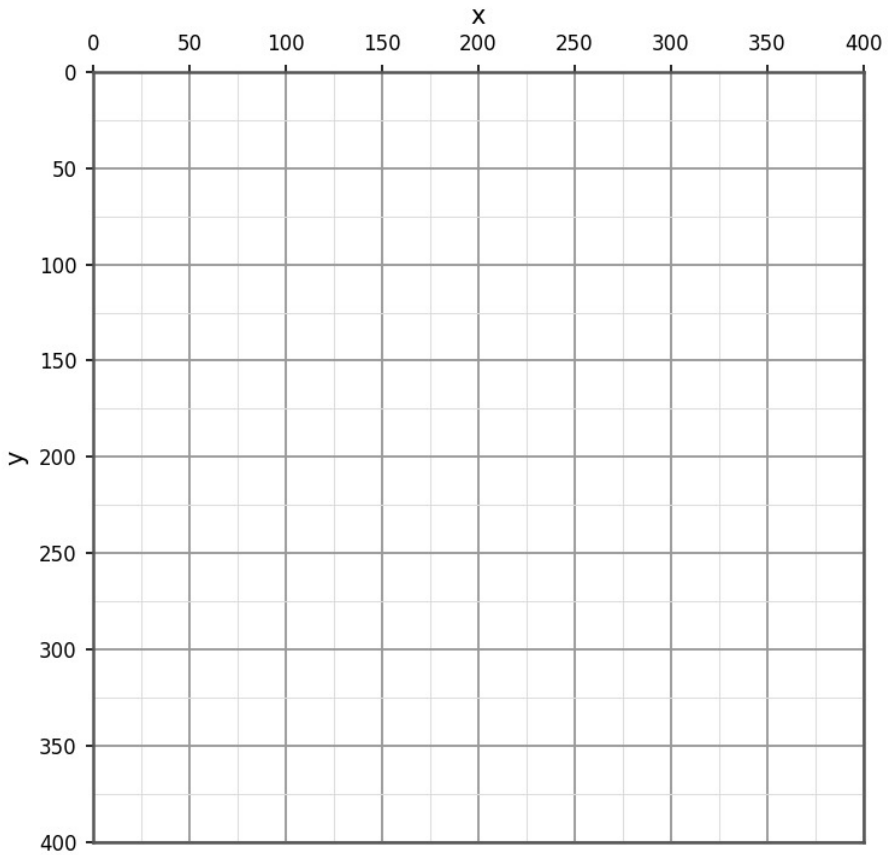


Trace #2

Assume the canvas is 400x400, and this is called as `drawCT1(app, 100, 4)`.

```
def drawCT1(app, m, n):
    d = app.width
    x = 0
    y = 0
    for i in range(n):
        if i%2 == 0:
            color = 'black'
        else:
            color = 'white'
        drawRect(x, y, d, d, fill=color, border='black')
        d-= m
        x += m//2
        y += m//2
    drawLabel("112 Rocks", app.width//2, app.height//2, align='center', fill='black')
```

Sketch your answer on the grid below:

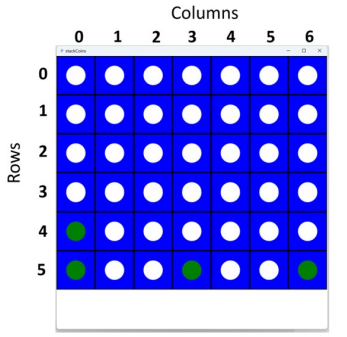


Fill-in-the-Blanks: drawStackCoins

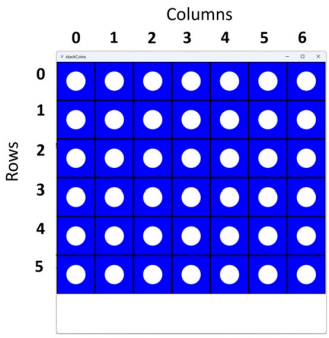
Picture the board from Connect Four: a grid **6 rows tall and 7 columns wide** (42 cells total), where each cell is a blue square with a black border, and each square has a circle centered inside it — green if a coin has been played there, white if the cell is empty. The board should always fill the *entire* canvas, however the window is resized.

Coin positions are passed in as a single comma-separated string, where each position is a two-digit code: the first digit is the row, the second is the column. So `"02,25,31"` means there are coins at row 0 col 2, row 2 col 5, and row 3 col 1 — every other cell is empty.

Consider the following examples (note that the printed row and column numbers are there to help your understanding, they are not present in the actual game):



(a) `drawStackCoins(app, "50,40,53,56")`



(b) `drawStackCoins(app, "")`

Fill in blank lines to complete the function:

```
def drawStackCoins(app, coins):
    rows = 6
    cols = 7

    cellWidth = ______________________

    cellHeight = ______________________

    for row in range(rows):
        for col in range(cols):

            cellLeft = ______________________

            cellTop = ______________________

            drawRect(cellLeft, cellTop, cellWidth, cellHeight, fill='blue', border='black')

            cellCode = ______________________   # e.g. '02' for row 0, col 2

            if cellCode in coins.split(','):
                circleColor = 'green'
            else:

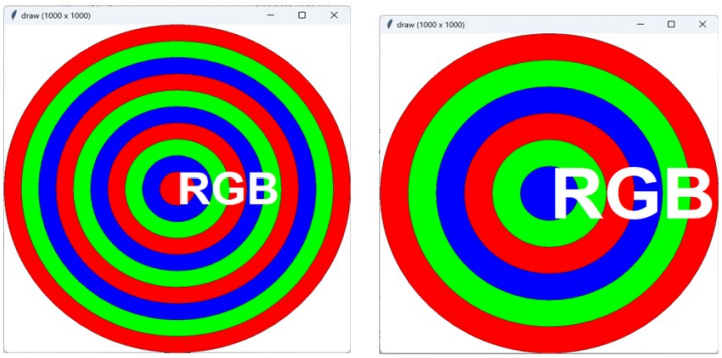
                circleColor = ______________________

            drawCircle(cellLeft + cellWidth/2, cellTop + cellHeight/2,
                       min(cellWidth, cellHeight)/2 - 5,
                       fill=circleColor, border='black')
```

Free Response: drawRGBTarget

Write `drawRGBTarget(app, n)`, which draws `n` concentric (same-center) circles that together fill the canvas, alternating colors **red, green, blue, red, green, blue, ...** from the outside in. A few specifics:

- The circles are centered on the canvas, and the **largest (outermost) circle always spans the full canvas** (its diameter is the smaller of `app.width` and `app.height`) and is **always red**.
- Each successive circle is drawn on top of the previous one, shrinking by an equal amount each time, so the innermost circle is the smallest.
- On top of all the circles, draw the bold label `"RGB"` centered on the canvas, with its font size equal to the diameter of the *smallest* circle.
- It should still look correct after resizing the window (don't hardcode the canvas size).

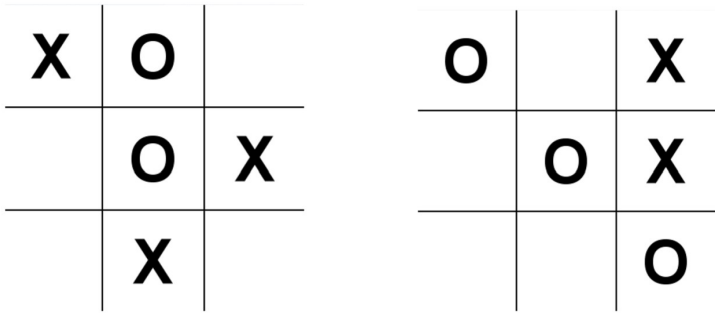


Try it with a few different values of `n` (including something that isn't a multiple of 3) and make sure the color pattern still alternates correctly and nothing looks off-center.

Free Response: drawTicTacToe

Write `drawTicTacToe(app, board)`, which draws a 3x3 tic-tac-toe board spanning the entire canvas. `board` is a 9-character string of `'x'`, `'o'`, and `'_'` characters, one per cell, listed in reading order (left-to-right, top-to-bottom) — so `board[0]` is the top-left cell and `board[8]` is the bottom-right cell.

- Draw the grid lines for the 3x3 board.
- For every cell containing `'x'`, draw an X that fits inside that cell with a bit of padding.
- For every cell containing `'o'`, draw an O that fits inside that cell with a bit of padding.
- Cells containing `'_'` stay empty.
- The whole board should fill the canvas and stay correct after resizing, including on non-square windows.



(a) `drawTicTacToe(app, "xo__ox_x_")` (b) `drawTicTacToe(app, "o_x_ox__o")`

Test it with a string like `'xo__x_o_x'` and check that the marks land in the right cells, especially after you resize the window into something wide and short, or tall and narrow.