CSC 110, Spring 2018
Programming Assignment #4: Doodle (20 points)

Due: Tuesday, February 13, 2018, 7:00 PM

Based on an assignment from the University of Washington by Stuart Reges and Marty Stepp

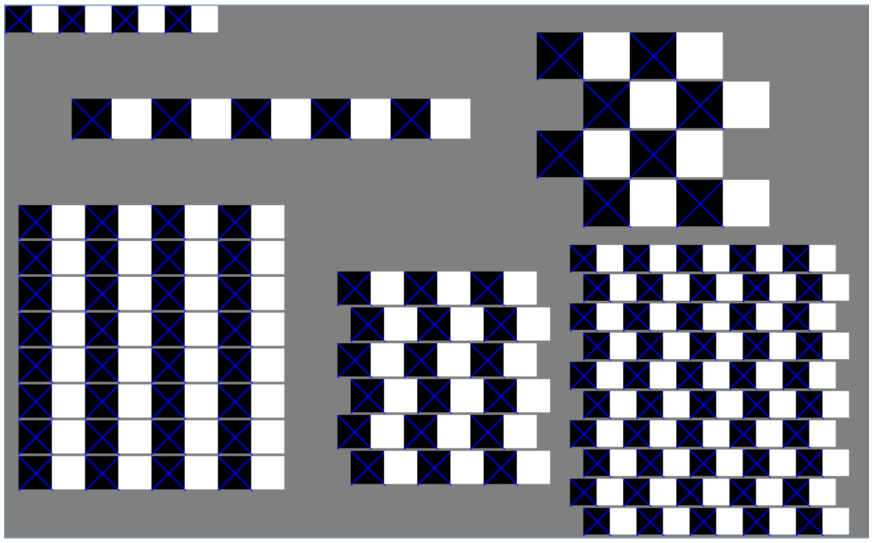
Program Description:

This assignment covers parameters and graphics. Turn in Python files named `doodle.py` and `cafewall.py`.

To run this assignment, you must download the file `DrawingPanel.py` from the Homework section of the class web page and save it in the same folder as your code. Do not turn in `DrawingPanel.py`.

Part A: `Doodle` (2 points):

For the first part of this assignment, turn in a file `doodle.py` that draws an **animated** figure using the `DrawingPanel` provided in class. You may draw any figure you like that is at least 100 x 100 pixels, contains at least three shapes, uses at least two distinct colors, is your own work, is not highly similar to your figure for Part B, and animates. Your program also should not have any infinite loops and should not read any user input. Your score for Part A will be based solely on external correctness as just defined; it will not be graded on internal correctness.



Part B: `CafeWall` (18 points)

For the second part of this assignment, turn in a file named `cafewall.py` that draws several instances of the Café Wall illusion. In this optical illusion, the rows appear not to be straight even though they are. Your program should exactly reproduce the image at left.

This image has several levels of structure. Black and white squares are used to form rows and rows are combined to form grids. The output has two free-standing rows and four grids.

The overall drawing panel is size **650 x 400**. Its background is gray and the squares are black and white and have borders of width 0. A blue X covers each black square.

The six figures on the grid should have the following properties:

Description	(x, y) position	Number of pairs	Size of each box	2nd row offset
upper-left	(0, 0)	4	20	N/A
mid-left	(50, 70)	5	30	N/A
lower left	(10, 150)	4	25	0
lower middle	(250, 200)	3	25	10
lower right	(425, 180)	5	20	10
upper right	(400, 20)	2	35	35

If there is no visible difference to the naked eye between the expected output and your output, it is considered correct. (If your figure looks the same but has "thicker" black lines, you may be re-drawing the same shapes multiple times.)

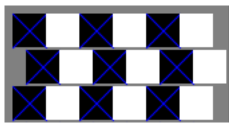
Implementation Guidelines for Part B:

To receive full credit on Part B, you are required to have two particular functions described below. These functions use a great deal of parameter passing and perform the program's complex numeric computations.

1. Function to draw a row

Your first function should draw a single row. Notice that we specify each row in terms of how many pairs of black/white boxes it has. Your function doesn't have to be able to produce a row that has a different number of black versus white boxes. The boxes are specified using a single size measure because each should be a square. Different rows have different sizes, positions, and so on. Therefore, your function should accept several parameters so that it is possible to call it many times to draw the many different rows on the screen.

2. Function to draw a grid



Once you have completed the function that produces one row, write another function that produces a grid of rows. Grids are composed of a series of pairs of rows where the second row is offset a certain distance in the x direction relative to the first.

Each grid is a square, which is why a single value (number of pairs) indicates the number of rows and columns. A single box size is again used because each box should be a square. The offset indicates how far the second row should be shifted to the right in each pair. The figure in the lower-left has no offset at all. The grid in the upper-right is offset by the size of one of the boxes, which is why it has a checkerboard appearance. The other two grids have an offset that is in between, which is why they produce the optical illusion.

Each pair of lines in the grid should be separated by a certain distance, revealing the gray background underneath. This is referred to as the “mortar” that would appear between layers of brick if you were to build a wall with this pattern. The mortar is essential to the illusion. Your program should use **2** pixels of separation for the mortar, but you should introduce a program constant that would make it easy to change this value to something else.

Place the following statement at the top of your Python files, so that your code can use graphics:

```
from DrawingPanel import *    # so that I can use Graphics
```

Development Strategy (How to Get Started):

This program does not require as many lines of code as past ones. However, the numeric computations and parameters are not simple. You might be overwhelmed with the amount of detail you have to handle all at once. As famous computer scientist Brian Kernighan once said, "Controlling complexity is the essence of computer programming." To make things easier, begin with a smaller piece of the problem.

Write your code in stages, repeatedly making small improvements. Start by having your first function draw only the upper-left row, then generalize it by **adding one parameter at a time**. For example, add parameters to change the x/y position. Test the parameter by passing different values. Once it works, move on to the next.

Style Guidelines:

For this assignment you are limited to the language features discussed in lectures 1 - 11.

We require at least the two functions named previously. You may use additional functions if you like. You may receive a deduction if your functions accept too many parameters or unnecessary parameters. An "unnecessary" parameter in this case is one whose value is redundant with another's or could be computed using others' values.

Give meaningful names to functions, variables, and parameters. Follow Python’s naming standards as specified in lecture. Limit the lengths of your lines to fewer than 80 characters. Include meaningful comment headers at the top of your program and at the start of each function.