CSC 104 Lab 5, 30 March 2012

This lab is worth 4% of your grade in this course. **You must sign up for a lab time on the course web page**; you are not enrolled automatically. Please sign up well in advance, and please attend your signed-up lab time.

I believe that it is possible, although difficult, to finish this lab during the hour, if you attended and understood the lecture about the Bresenham line algorithm on March 27. But you might want to go through Part A in advance, and maybe the first two points of Part B. The actual deadline for the final lab submission will be announced at the end of the scheduled lab hour.

Note: This program is more difficult than any other Python program you have written this term. Not everyone will get it working fully. Please just do what you can.

This lab can be done individually or in pairs. If working in pairs, please make sure that the student who isn't typing is following everything which is going on, and please switch places at some point.

The problem

Computer displays consist of individual dots we call ''pixels''—picture elements. One common computer graphics task is to draw a line from one point to another, figuring out where each of the intermediate dots should be to draw the smoothest possible line.

Of course, we could just call ''draw_line'' instead of plotting individual points. But someone has to write *that*! In this lab, we examine how such a thing is written.

Part A: Familiarization with my "fatpixel" package

1. I have written a very simple package to plot points using large pixels to make it easy to see what your line-drawing algorithm is doing. Please copy the file /u/ajr/104/lab5/pixeldemo.py to your own directory to experiment with.

2. The first three lines of pixeldemo.py will be the first three lines of all of the Python files you look at today. It is not required for you to understand them, but this point tries to explain them a bit.

The third line of pixeldemo.py says ''import fatpixel'', which allows you to use fatpixel routines such as fatpixel.plot(). But the Python interpreter would not be able to find my fatpixel code if not for the second line, which adds /u/csc104h/winter/pub/lab5 to the list of directories where the Python interpreter looks for packages. But that line refers to ''sys.path'', and to use that you have to do ''import sys'', which is the first line.

3. Please read the rest of pixeldemo.py; try it; and play with it a bit.

4. Note how the drawing window closes when the program exits. That is why it has a raw_input() at the end whose value we ignore—the raw_input() gives you time to see what has been drawn in the window, as follows: The system waits for input, then the return value of raw_input() is the input the user provided (probably just an empty string if you just pressed return like the instructions said); and then we ignore that value.

5. Write a 'for' loop to plot all of the points (10,10), (10,11), (10,12), (10,13), and (10,14). This will make a vertical line. (Try your program before moving on!)

6. Write a 'for' loop to make a horizontal line.

7. Write a 'for' loop to plot all of the points (10,10), (11,11), (12,12), (13,13), and (14,14). Predict what this line will look like before you run your program.

(over)

Part B: Implementing the Bresenham line algorithm (for marks)

1. Please see the file /u/ajr/104/lab5/starter.py. It is similar to pixeldemo.py, but prompts for numbers in a loop. The loop takes care of the problem of the drawing window disappearing when the program exits—the program does not exit. Copy this to the file name "bres.py" in your own directory for your implementation.

2. You will insert code to perform the Bresenham line algorithm where it says "INSERT CODE HERE". For example, see and try /u/ajr/104/lab5/starterdemo.py, which inserts code simply to plot the two end points (but nothing inbetween). (You may have to move the graphics window which comes up so that you can see the terminal window as well. Try typing some coordinates. Press control-C, return, to exit the program.) (You will want to start from starter.py, not starterdemo.py.)

3. In your program you will draw a line from the point (1,1) to the point (x2,y2). The Bresenham line algorithm begins by figuring out how far you have to go in each direction. Assign "dx" to be the difference in x (i.e. x2−x), and assign "dy" to be the difference in y.

4. If dx is greater than dy, we need a loop in which x increases by one each time around the loop, and y increases only sometimes. Inside an "`if dx > dy:`", write this loop in accordance with the Bresenham line algorithm presented in class. This is the main point of today's lab. You can test it with values in which x is greater than y; e.g. plot a line from (1,1) to (15,10). (Whereas plotting a line from (1,1) to (10,15) will go under an "else" we will subsequently write, for the case where dx≤dy.)

5. Your program will not function if you put in non-integer coordinates, or values less than 2 (e.g. if you attempt to plot from (1,1) to (0,0)). Don't worry about this today.

6. Once the above is working, add an "`else:`" and write similar code for the other case. In this code, y will increase by one each time around the loop, and x will sometimes increase by one and sometimes not. (This point is worth much less in grading than point #4.)

7. Before the deadline as announced at the end of the scheduled lab hour, submit your bres.py (in assignment name "lab5") for grading. If you are working with a partner, also submit a file named "partner" which says who the other student is. The "partner" file should contain the CDF account name of the other student only (and no other text).

Part C: Sharing files with your partner

If you worked on this lab in a pair, please copy at least your bres.py file to the other account for future reference, most easily using the "scp" command as described in lab 4.