COMP 120 - Problem Solving Assignment 3

Due: Saturday, March 14 @ 10:00PM

Required Background:

- "Introduction to Python Programming and Data Structures" - Chapter 11, Advanced GUI programming using Tkinter; and Chapter 15, Recursion.

Assignment Overview:

For this assignment you will write two GUI programs: both are based, but are not identical to, programming exercises in your online textbook – problem 1 is based on Exercise 11.17, and problem 2 is based on Exercise 15.31.

Be sure to read this entire document before beginning to work on the assignment.

Initial Setup

Both you and your partner will need to get the starter code for your group using Git.

1. You will need your group number in what follows. Get this from the "PSA Group Numbers" file under the "PSAs" tab on Blackboard.,
2. In VS Code, open the command palette and select the "Git Clone" option.
3. When prompted for the repository URL, enter the following, with X replaced by your group number (e.g. 7 or 12).

   ssh://git@code.sandiego.edu/comp120-sp20-psa3-groupX
4. Choose the "Open Repository" option in the window that pops up in the lower-right corner of the screen. The repository should contain the following files:
   - `moving_circles.py`: You will put all of the code you write for problem 1 in this file.
   - fractal_tree.py: You will put all of the code you write for problem 2 in this file.
5. Each programming team will have it own repository that has been initialized with the same starter code, so when you sync your code, you are sharing with your partners, but with no one else in the class. (I can see your repository also.)

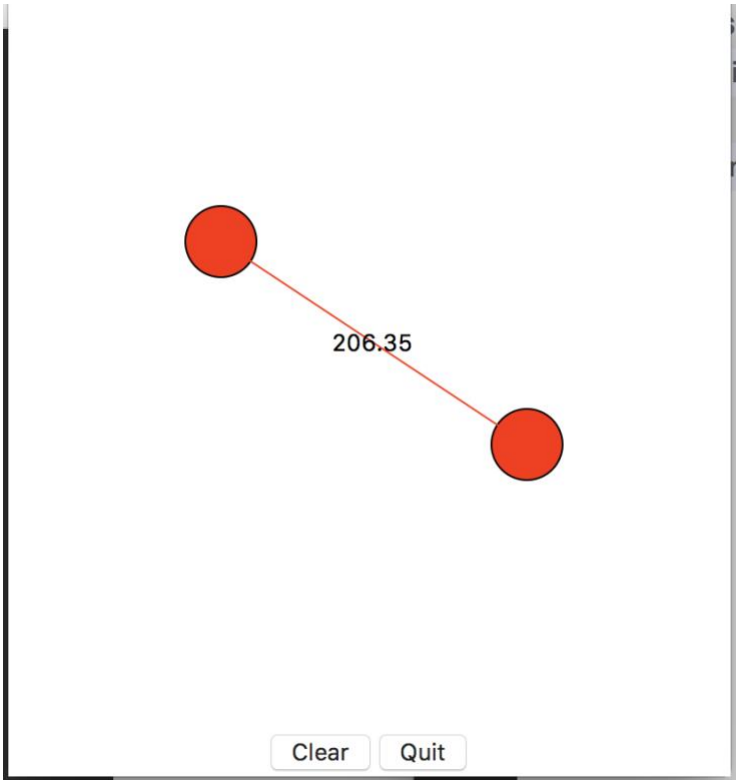
Remember that if you close VS Code, then when you reopen it, you should see your repository. But if you don't see it, then just select File->Open…, and then select the directory containing your repository.

We also recommend that you stage changes, commit those changes, and sync the changes peridically as you are working on the program, and certainly when you are done with a

session with your programming partner. This ensures that you won't lose any of your work in case your computer gets lost or a file gets accidentally deleted.

Problem 1

As said above, problem 1 is based on, but not identical to exercise 11.17 in your textbook. The picture below illustrates what the screen of your program should look like after the user has chosen the centers of the two circles.



Here are the details. Your program must adhere to the following requirements:

1. The screen should have a 400 x 400 canvas on the top of the window, and a button frame at the bottom of the window.
2. The button frame should have a clear button that resets the canvas to the state it was in when the program first starts, and a quit button that terminates the program.
3. When the program starts, the canvas should be blank.
4. When the user clicks in the canvas, create the first circle with center at the mouse click and radius 20.
5. When the user clicks again, create the second circle with center at the mouse click and radius also 20. Also, create the line connecting the circles, and label the line with the distance between the circles. The label should be halfway between the

centers of the circles. The distance can be displayed using the create_text method of the canvas class, self.canvas.create_text((center_x, center_y), text_to_display.

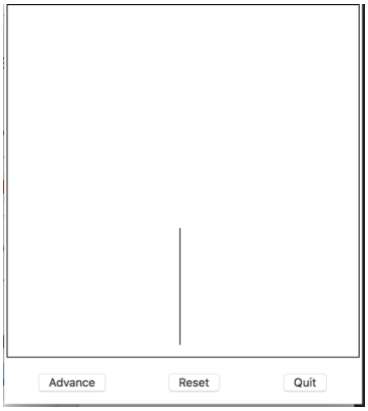
6. Then, the user can drag either circle around the canvas (by pressing the mouse down inside the circle). The program should reposition the circle to be centered at the current mouse location, making sure to move the line and update the distance. Move the line and the distance text by deleting and redrawing them. Move the circles by using the move method of the canvas class, self.canvas.move(circle, del_x, del_y).
7. All of your code must be written in the CircleDistance class.
8. Use descriptive variable names in your program, and use all lowercase letters with underscores separating words. You should appropriately comment your program. It should have a header (this is started for you - be sure to add your names, the date you started it, and a description); each function should have docstring comments.
9. You should also comment blocks of code within your functions, explaining what the code is doing. How much commenting to add is a judgement call - you don't want too much, or too little. If you have a block of code (say up to 10 lines long), put a brief comment before it saying what is about to happen. Then put blank lines between the blocks of code. Don't comment individual lines of code, unless they are doing something that the reader might not see right away.
10. No functions/methods should be longer than 30 lines of code, not counting blank lines and lines with just a comment on them.
11. Put your code in the moving_circles.py file.
12. You are welcome to use any code from programs that we have written in class.

Any questions about how the program should behave should be posted to Piazza (folder #psa3).

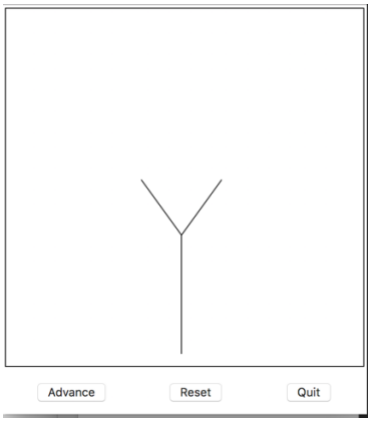
Problem 2

As said above, problem 2 is based on, but not identical to exercise 15.31 in your textbook. It is to write a program that displays a fractal tree.

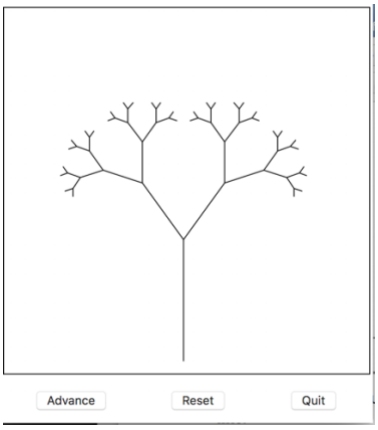
When the program starts, the display should look like (this is 0 levels of recursion):



and after hitting the advance button, the screen should look like (1 level of recursion), it should look like:



and after hitting the button a total of 5 times (5 levels of recursion) the screen should look like:



Here are the details:

1. The screen should have a 400 x 400 canvas on the top of the window, and a button frame at the bottom of the window.

2. Initially, the canvas contains the fractal tree with 0 levels of recursion – just the central branch.
3. The button frame should have an advance button that displays the fractal tree with one additional level of recursion (see the drawings above); a reset button that resets the display to 0 levels of recursion; and a quit button that terminates the program.
4. The length of the initial branch is one third the height of the canvas.
5. The length of each "child" branch is 0.58 times the length of its parent branch. (Define this 0.58 as an instance variable of the FractalTree class, so that it can be easily changed.)
6. The angle made by a child branch relative to the parent branch is pi/5.
7. All of your code must be written in the FractalTree class.
8. The code in fractal_tree.py is the code for the fractal rectangle that we wrote/demonstrated in class. You can reuse quite a bit of it for this project, but you will also have to modify it for the fractal tree.
9. Use descriptive variable names in your program, and use all lowercase letters with underscores separating words. You should appropriately comment your program. It should have a header (this is started for you - be sure to add your names, the date you started it, and a description); each function should have docstring comments.
10. You should also comment blocks of code within your functions, explaining what the code is doing. How much commenting to add is a judgement call - you don't want too much, or too little. If you have a block of code (say up to 10 lines long), put a brief comment before it saying what is about to happen. Then put blank lines between the blocks of code. Don't comment individual lines of code, unless they are doing something that the reader might not see right away.
11. No functions/methods should be longer than 30 lines of code, not counting blank lines and lines with just a comment on them.
12. Comments/hints:
    a. Each time you call draw_fractal you will draw just one branch. Other branches are drawn by recursive calls.
    b. A key decision is what the parameters to draw_fractal should be. Here is a suggestion:
        i. The x coordinate of the base of the branch to be drawn.
        ii. The y coordinate of the base of the branch to be drawn.
        iii. The length of the branch to be drawn.
        iv. The angle that the branch makes (relative to a horizontal line to the right) with positive angles going counterclockwise. (So the initial branch has an angle of +pi/4.)
        v. The remaining levels of recursion to be drawn.

Testing your programs

There are not automated test programs for this assignment, so you should test them on your own. Go down the numbered requirements listed above for each problem, and insure that your program satisfies the requirement. Points will be deducted for each requirement that is not satisfied.

Pair programming requirement

As described in the syllabus, you should write your program using pair programming. Recall that in pair programming, you and your partner work together at one computer, with one of you typing code (the driver), and the other managing (the navigator). To encourage this from you and your partner, when you are the driver for your team, you should be working on your own computer. When you switch roles, the driver should sync her code to the repository, and her partner should then sync onto his computer, and then become the driver. Remember that you should be switching roles every half hour or so.

So when you follow this approach in writing your program, I should see syncs from both of you, with significant differences between the code synced. If I don't see syncs from both of you, there will be a 5 point penalty on your final grade for the assignment.

Submission Instructions

Important: To be safe, you should run your final code on both you and your partner's computers. This will ensure that you are not relying on any special setup on your own computer for the code to work.

To submit your code, you will need to synchronize it using Git. To make sure your changes are saved and synchronized, follow these steps.

1. Open the "Source Control" menu, either by clicking on the 3rd icon on the left (right under the magnifying glass) *or* by going to "View" and "SCM".

2. Your `circle_distance.py` and `fractal_tree.py` files should show up under the "Changes" section. Click on the "+" icon to the right of the name(s) of the file(s) that you changed to "stage" the changes. This should move the file to a section named "Staged Changes."

3. Commit your changes by typing in a descriptive message (e.g. "finished problem 1") into the "Message" textbox (above the Staged Changes area). After entering the message, click the checkmark icon above the message box to perform the commit. This should remove the changed files from the Staged Changes section.

4. Then, Sync your commit(s) to the server by clicking the "…" (to the right of the checkmark from the last step) and select the "Sync" option. This will likely result in a pop-up notifying you that the sync will do a push and a pull. Select "OK" (or "OK and don't ask again") to complete the sync.

   If you run into problems, make sure you are properly connected to USD's eduroam wifi, and try the Sync again. If you are still running into problems, check Piazza and ask a new question there if the answer doesn't already exist.

   To make sure that your changes were synced correctly, have your partner do the final step above (namely "…" and then "Sync"). This should fetch the changes you made. You can then test on their computer to make sure it works exactly the same as on your

computer. If your partner has trouble accessing and/or running the file, it is likely that the grader will also have problems and your grade will be negatively impacted.

You should have one sync with commit message "Finished problem 1", and another sync with message "Finished problem 2".

5. When you have finished both problems, go to Piazza, and post to the `psa3_submit` folder a message saying that you are ready for grading of psa3.

Grading

When I grade the problems, I will go down the requirements listed above, and up to 5 points will be deducted for each item where you have not met the requirements.

Late Penalties

1. If you commit by the due date, no penalty. Else,
2. if you commit by 10PM on Thursday, March 19, 10 points late penalty. Else,
3. if you sync by 10PM on Tuesday, March 24, 20 points total late penalty. Else,
4. no points.

Academic Integrity

Please review the portion of the syllabus that talks about academic integrity with regard to writing programs. In summary, do not share or show your code to any other team in class, and do not turn in any code that you did not write yourself. Don't look at the code from another team, or from any other source. The point of these programs is for you to develop your coding skills.