Tyler Salas

1:30 – 2:50 Class

Dr. Fuentes

80596221

Lab 1 Report

Lab one, the lab assigned this week, assigned to create nested figures using the matplotlib.pyplot function in Python. The methods used to create the nested figures were required to be recursive, made with code Dr. Fuentes shared with us online. The main idea was to simply create methods in Python that would take some number of recursive calls and create nested or extensions of the original figure.

The basic idea to solve the four problems presented in the lab was too take the

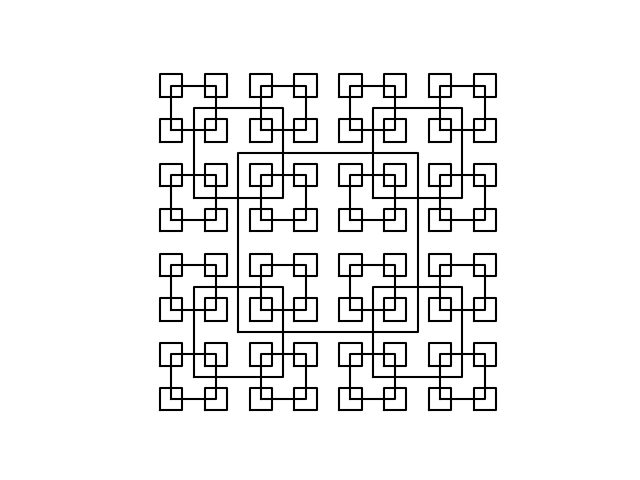
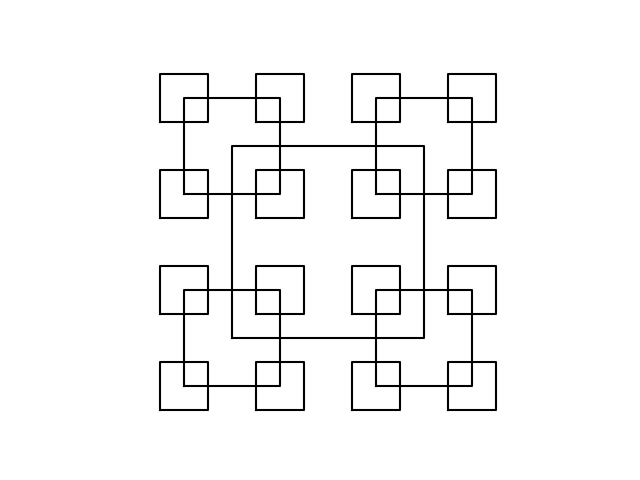
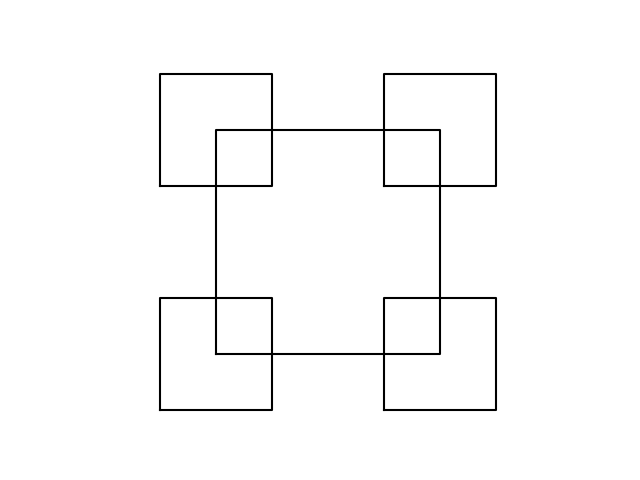
coordinates of the original (parent) figure and relocate or modify the coordinates with the recursive methods to end up with the new (child) figure.

* 1. In problem number one for the lab, we had to take a square and create child figures of the square at all four of its corners, a quarter the size of the original. The approach I took to this one was to take the coordinates of the original square and put the amount of movement needed to get the new coordinate into a list the same dimensions as the parent figure. (I.e. (The bottom left corner coordinates) [0,0] + [-250, -250]) Then I added those two lists into a new list variable to hold the coordinates of the new child figure and applying it to the recursive call. (Four calls, one for each corner) This created the nested squares method and figures.

Run-Times (Seconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 Rec Calls | 3 Rec Calls | 4 Rec Calls |
| Run-Time 1 | 0.07468867301940918 | 0.0978398323059082 | 0.1908550262451172 |
| Run-Time 2 | 0.07471895217895508 | 0.10051536560058594 | 0.24205374717712402 |
| Run-Time 3 | 0.07428431510925293 | 0.10624027252197266 | 0.19224143028259277 |

2 Calls 3 Calls 5 Calls



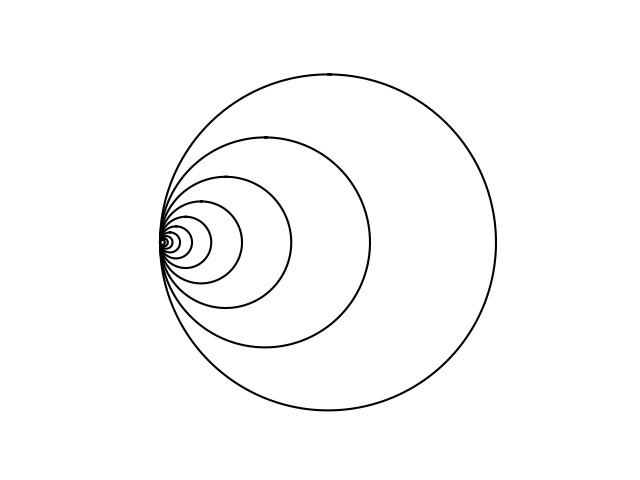
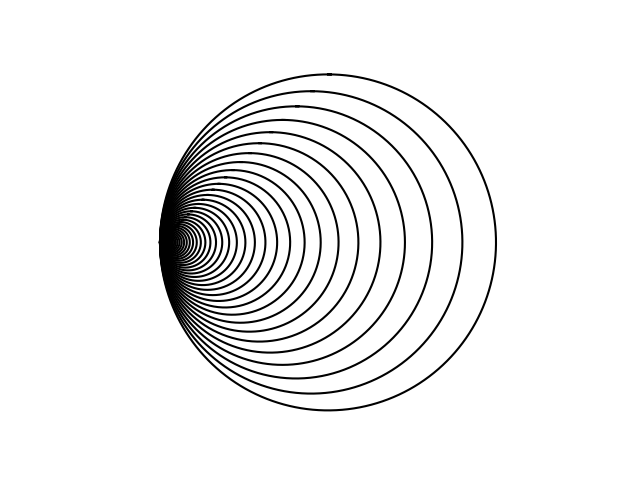
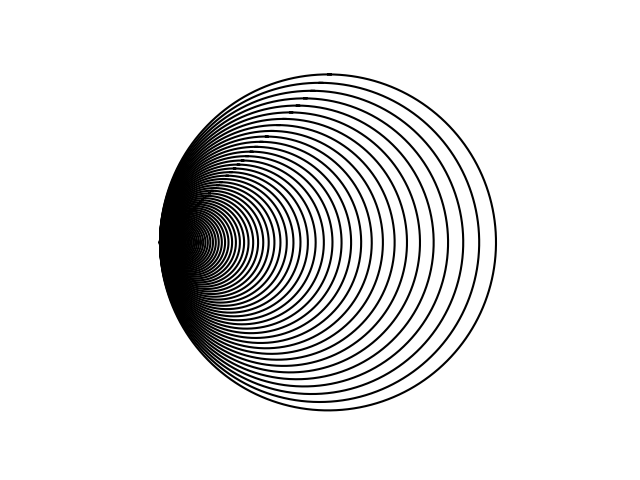
To test the calls all you must do is simply modify the call method variable n to equal the number of recursive calls you desire.

* 1. In problem number two for the lab, we had to take a circle and create child figures of the circle that would be nested inside the original and keep getting smaller. What I did is move the center of each child figure over by adding the radius to the x coordinate, with this the figure is placed to the left side of the original parent figure. To get the different degrees of recurrence you simply up the recursive calls to the original method (n in the method) and to get the distance between the circles smaller, you up the size of the circle's recurrence method ratio. (w in the method)

Run –Times (Seconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10 Rec Calls | 50 Rec Calls | 100 Rec Calls |
| Run-Time 1 | 0.09189915657043457 | 0.16995000839233398 | 0.2827012538909912 |
| Run-Time 2 | 0.08677554130554199 | 0.1690678596496582 | 0.24609088897705078 |
| Run-Time 3 | 0.09036111831665039 | 0.19613385200500488 | 0.2459731101989746 |

2 Calls 3 Calls 5 Calls

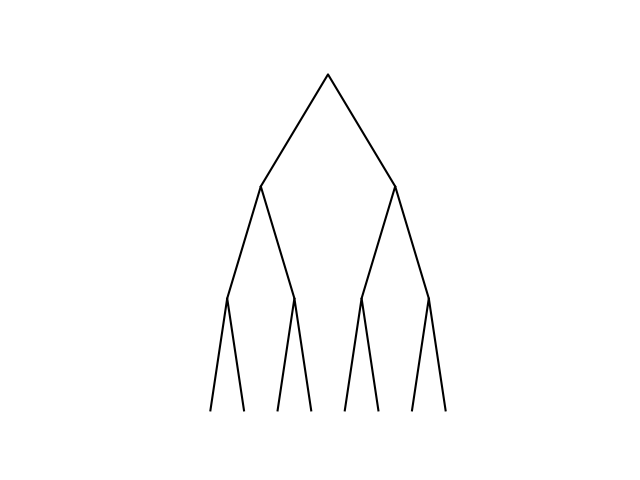
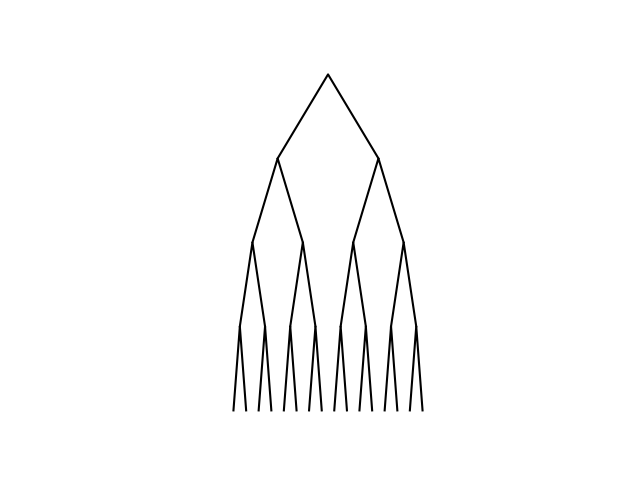
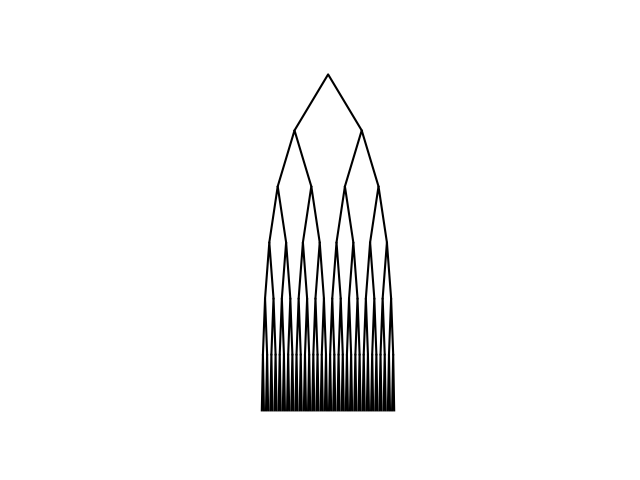
To test the calls all you must do is change the n variable in the method to receive more recursive circles and to change the distance, simply change the w variable.

* 1. In problem number three of the lab, we had to create child figures at the tips of the parent binary tree that would grow smaller in distance between the left and right sides each time. For this one I took the same approach as in problem one by putting the amount of movement needed to get the new coordinates of the child figures into a list and adding it together with the original to create a new variable with the correct coordinates. I called the method twice each time (One for each branch) and made two different lists to move the children figures in the correct place

Run –Times (Seconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3 Rec Calls | 4 Rec Calls | 6 Rec Calls |
| Run-Time 1 | 0.08572530746459961 | 0.09673857688903809 | 0.16156864166259766 |
| Run-Time 2 | 0.0787661075592041 | 0.08876204490661621 | 0.15558505058288574 |
| Run-Time 3 | 0.0747992992401123 | 0.1326456069946289 | 0.15957427024841309 |

2 Calls 3 Calls 5 Calls

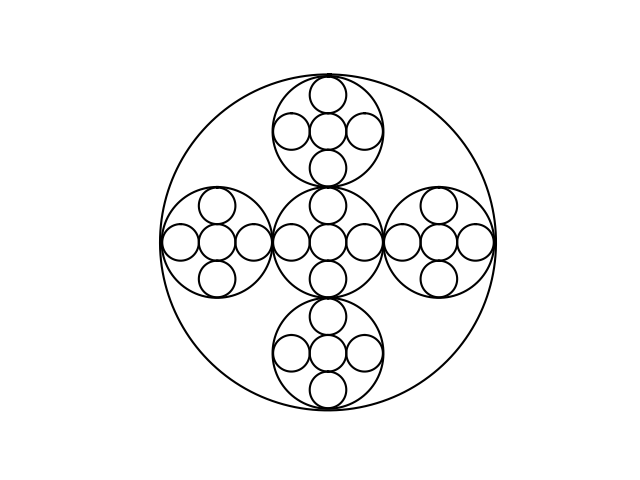
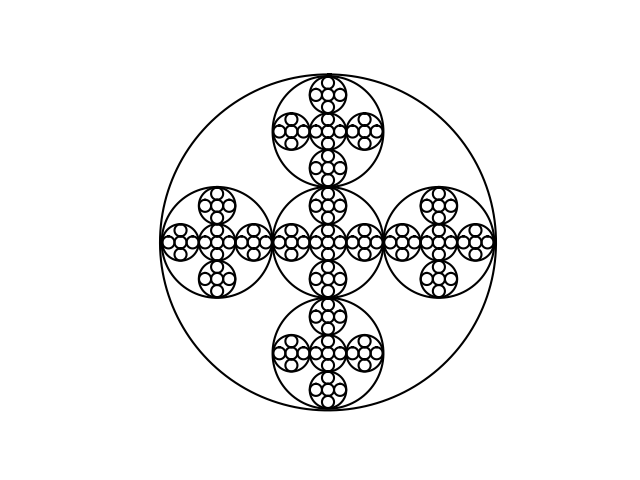
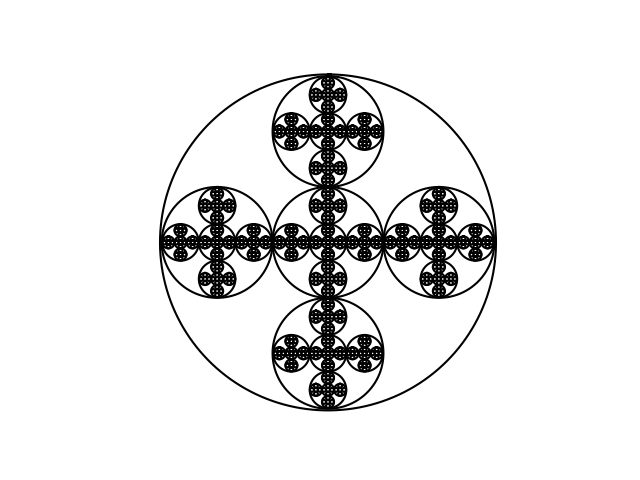
To replicate the figures tested all that is required is to change the n variable in the method to get the number of tree levels you’d like.

* 1. In problem number four of the lab, we had to create recursive circles in a cross that would lie inside of the parent circle. The main idea was to simply move the center of the circle while also downing the size of the child circle to be a third the size of its parent. To move the center, I added or subtracted the radius to the x coordinate to move it left or right and the same thing to the y to move it up and down, while also multiplying the dimensions by .33. I called the method five different times inside the method, one for each point and center of the cross. This created the necessary figures.

Run –Times (Seconds)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 3 Rec Calls | 4 Rec Calls | 5 Rec Calls |
| Run-Time 1 | 0.12865757942199707 | 0.3640251159667969 | 1.4082365036010742 |
| Run-Time 2 | 0.1306006908416748 | 0.3989717960357666 | 1.4262025356292725 |
| Run-Time 3 | 0.16060590744018555 | 0.3430492877960205 | 1.4630889892578125 |

2 Calls 3 Calls 5 Calls

Judging from the run times this method was significantly slower than the rest, due to the amount of work and additional recursive calls done in the method. To replicate this simply change the variable n to the number of recursive calls you’d like to make.

The completion of this project really opened my understanding to the principles of recursion and how they can relate to creating graphics in Python. Previously I had never used imports or programs that dealt with graphs and the use of them to create figures, so the lab taught me a good amount about that as well. Overall the lab was definitely challenging but the outcome and level of understanding brought by it is very worth.