CS 2302

Lab 1 Report

Author: David Ayala

ID: 80617838

Instructor: Olac Fuentes

TA: Anindita Nath

**Introduction**

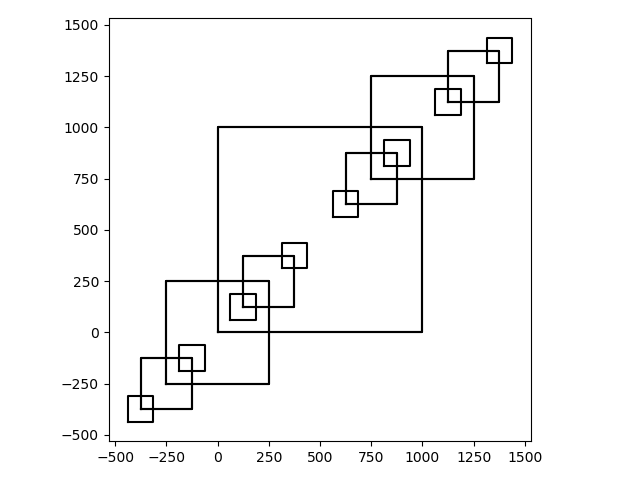
The purpose of this lab was to recreate the figure that were given in the lab 1 handout. This was to see if our ability to understand how recursion works and to see if we had the ability to program recursion. In order to do this, I would have to import packages like matplotlib, numpy and the math packages to be able to draw the recursive figures.

**Proposed Solution & Design Implementation**

When I was first starting to figure out how I would do this lab, I messed with variables of the two codes that Professor Fuentes provided to us on his website (draw\_circle method, draw\_squares method). When I was manipulating the variables, it allowed me to see how the recursive figures were changed. I also changed the coordinates it also changed where and how the recursive figure was manipulated. So, I spent a few days trying configurations of the variables and it led me to see the solution to one of the recursive figures. although I was not able to completely solve the other figures this way, it did allow me to think of other possible approaches to the problems.

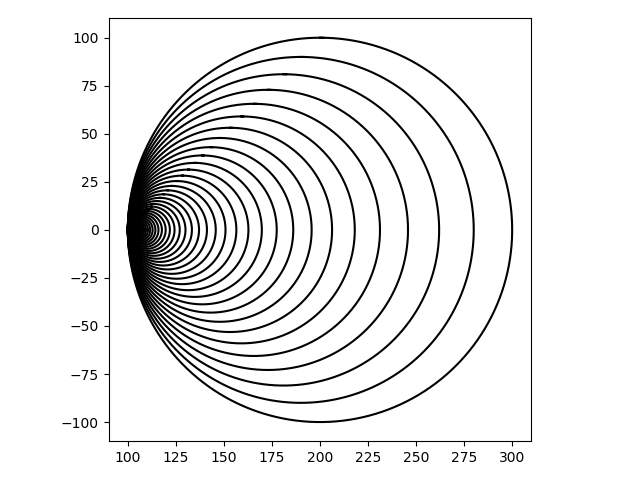
**The draw\_squares method:**

For this method I iterated the “draw\_squares” code that Professor Fuentes provided on his website. Then I simply placed the call “draw\_squares” twice and then I manipulated the variables for the array in order to produce the figure below. Although I managed to recreate the figure partially, I was not able to get the inverse of the figure below to recreate the complete figure. I tried to get the Running time of the program and was not able to get so I did not include it.



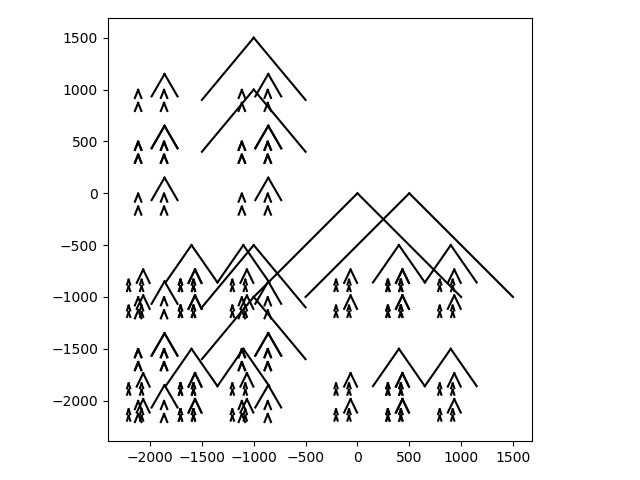
**The draw\_circles method:**

For this method I iterated the “draw\_circles” code that Professor Fuentes provided on his website. Then I simply manipulated the variables for the array in order to make the origin to the far right by adding the radius to the variable to the x\_axis. I then changed the number of times that the method is called in order to produce the figure below. Although I managed to recreate the figure partially, I was not able to get the inverse of the figure below to recreate the complete figure. I tried to get the Running time of the program and was not able to get so I did not include it.



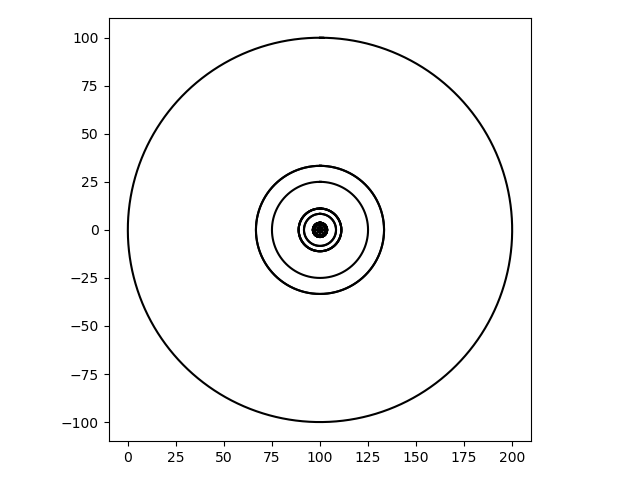
**The draw\_trees\_and\_mountians method:**

For this method I used a combination of the strategies/ ideas that I used in the previous methods. That means that I simply manipulated the variables for the array by adding values to it in order to not get traditional figures like squares or circles. I then changed the number of times that the method is called in order to produce the figure below. Although I managed to create a figure that somewhat resembles the start of a tree and goes out of control, I’m not sure how to fix this method in order to make the actual figure that was asked for. I tried to get the Running time of the program and was not able to get so I did not include it.



**The draw\_circles method:**

For this method I iterated the “draw\_circles” code that Professor Fuentes provided on his website. Then I tried to apply what I had accomplished in “draw\_squares” that way I could produce the smaller circles at different points, but I was not able to manipulate the variables for the array in order to make that happen. I was not able to recreate the figure that was asked for. I tried to get the Running time of the program and was not able to get so I did not include it.

****

**Conclusions:**

While the lab was straight forward, I was not able to fully accomplish the required tasks, I believe that this was due to the fact that I have a larger background in java than in python. While many say that they similar to each other, it is my experience that I will need to explore python further so that I can effectively program in python

**Appendix**

Methods are documented in the following format:

import numpy as np  
import matplotlib.pyplot as plt  
import math  
  
#figure one  
#this method will attempt to draw multiple squares recusively  
def draw\_squares(axis,count,points,w):  
 axis.plot(points[:, 0], points[:, 1], color='k')  
 if count > 0:  
 temp = points\*w  
 axis.plot(points[:,0],points[:,1],color='k')  
 draw\_squares(axis, count-1, temp+750, w)  
 draw\_squares(axis, count-1, temp-250, w)  
  
#plt.close("all")  
size = 1000  
  
points = np.array([[0, 0], [0, size], [size, size], [size,0],[0,0]])  
fig, axis = plt.subplots()  
draw\_squares(axis,3,points,.5)  
axis.set\_aspect(1.0)  
axis.axis('on')  
plt.show()  
fig.savefig('square.png')  
  
#figure two  
def circle(center, radi):  
 x\_axis = center[0] + radi  
 y\_axis = center[1] + radi  
 return x\_axis, y\_axis  
#this method will attempt to draw multiple circles recusively  
def draw\_circles(axis, n, center, radius, w):  
 if n > 0:  
 x\_axis, y\_axis = circle(center, radius)  
 axis.plot(x\_axis+radius, y\_axis, color='k')  
 draw\_circles(axis, n - 1, center, radius \* w, w)  
  
  
#plt.close("all")  
fig, axis = plt.subplots()  
draw\_circles(axis, 50, [100, 0], 100, .9)  
axis.set\_aspect(1.0)  
axis.axis('on')  
plt.show()  
fig.savefig('circle.png')  
  
#figure three  
#this method will attempt to draw multiple binary trees recusively  
def draw\_trees\_and\_mountians(axis,n,count,x\_axis,y\_axis):  
 if count>0:  
 axis.plot([n[1],n[1]-x\_axis],[n[0],n[0]-y\_axis], color='k')  
 axis.plot([n[1],n[1]+x\_axis],[n[0],n[0]-y\_axis], color='k')  
 if count>0:  
 draw\_trees\_and\_mountians(axis,[n[1]-x\_axis,n[0]-y\_axis],count-1,x\_axis/2,y\_axis\*.6)  
 draw\_trees\_and\_mountians(axis,[n[1]+x\_axis,n[0]-y\_axis],count-1,x\_axis/2,y\_axis\*.6)  
  
#plt.close("all")  
fig, axis = plt.subplots()  
n = np.array([[0,0],[0,500],[500,500],[500,0],[0,0]])  
draw\_trees\_and\_mountians(axis,n,7,1000 ,1000)  
axis.set\_aspect(1.0)  
axis.axis('on')  
plt.show()  
fig.savefig('tree.png')  
  
  
#figure four  
def circle(center, rad):  
 n = int(4 \* rad \* math.pi)  
 t = np.linspace(0, 6.3, n)  
 x\_axis = center[0] + rad \* np.sin(t)  
 y\_axis = center[1] + rad \* np.cos(t)  
 return x\_axis, y\_axis  
  
#this method will attempt to draw multiple squares recusively  
def draw\_circles2(axis, n, center, radius, w):  
 x\_axis, y\_axis = circle(center, radius)  
 axis.plot(x\_axis, y\_axis, color='k')  
 if n > 0:  
 x\_axis, y\_axis = circle(center, radius\*1/4)  
 axis.plot(x\_axis, y\_axis, color='k')  
 draw\_circles2(axis, n - 1, center, radius \* 1/3, w\*1/3)  
 draw\_circles2(axis, n - 1, center\*5, radius \* 1/3, w\*1/3)  
  
plt.close("all")  
fig, axis = plt.subplots()  
draw\_circles(axis, 5, [100, 0], 100, 1)  
axis.set\_aspect(1.0)  
axis.axis('on')  
plt.show()  
fig.savefig('circles.png')