PAOLA TERRAZAS

CS2302

RECURSIVELY DRAWING SQUARES

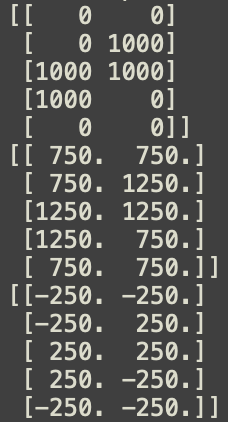
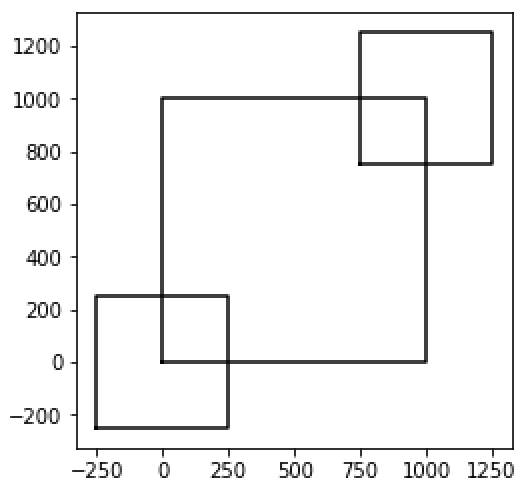
DR. FUENTES

TA - ANINDITA NATH

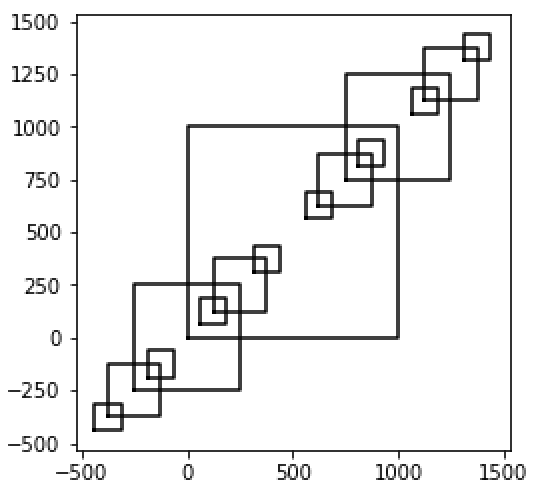
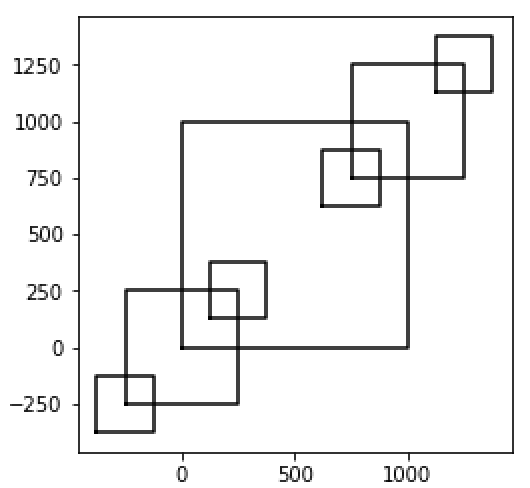
2.11.2019

REPORT

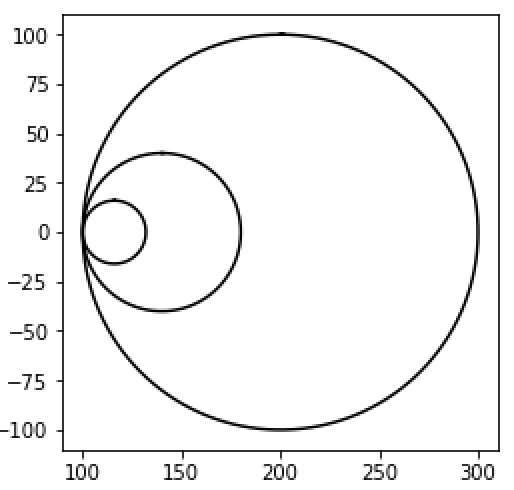
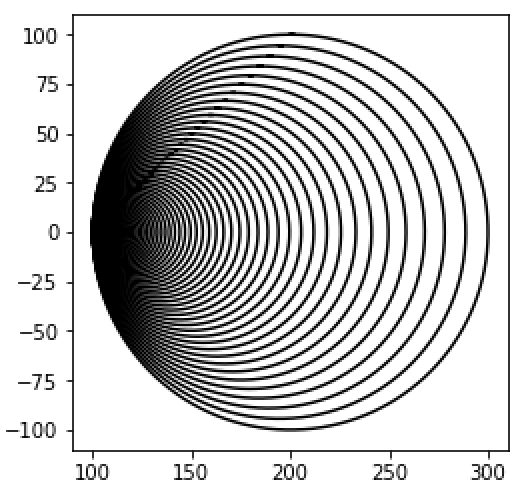
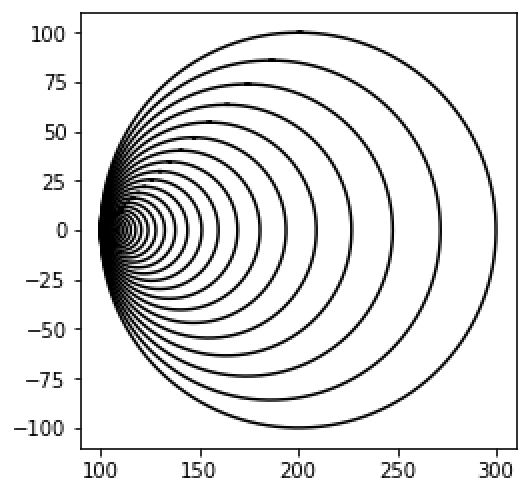
In our first lab, we were assigned to draw figures recursively in python. We had to create figures then recursively call them again on the original figure. For problem one, I began by creating a method “draw\_squares.” This method was going to draw the square and recursively increment the amount of squares. My parameters were “ax, n, p.” ax was used to help me draw the picture using ax.plot. n was the iterator, and p was the initial array that created the square, as follows: [[0,0],[0,original\_size],[original\_size,original\_size],[original\_size,0],[0,0]. The original p creates the square as you can see, the coordinates given create a square up to the “original\_size” that being 1000. Then I created a variable “equation” since it’s the equation that divides the original square by half, giving it the smaller squares needed for this recursive drawing. Then we I called draw\_squares (the recursive part of the problem) I used the parameters ax, n-1 because I’m subtracting one from iterator n, and as for p I substituted it for “equation+750” being the top right square and “equation-250.” Now, the reason that I used these values is because when I wrote down problem for myself, I realized that the top squares are moved up 75% of the original square and the bottom squares are moved down 25%. In conclusion, the plus and the minus only indicate moving up (plus) and moving down (minus). What really helped me through these labs was printing p, which gave me my coordinates and that helped me analyze what I needed to do to move things where I needed them to. Although, for my squares I had a difficult time getting the top left and bottom right squares, I could move the squares around but I was not able to figure out how to move them along the y-axis. I would be able to draw the missing squares easily, with coordinates and in a way “hard-coding” but that would not run recursively. As shown below, printing p is what helped me get the coordinates where they need to be for those two squares by adding 750 to one and subtracting 250 to another. Then, calling the method using draw\_squares(ax,2,p), draw\_squares(ax,3,p) and draw\_squares(ax,4,p).



Shown below are draw\_squares(ax,3,p) and draw\_squares(ax,4,p), working half way.

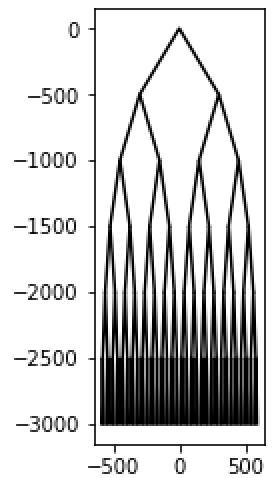
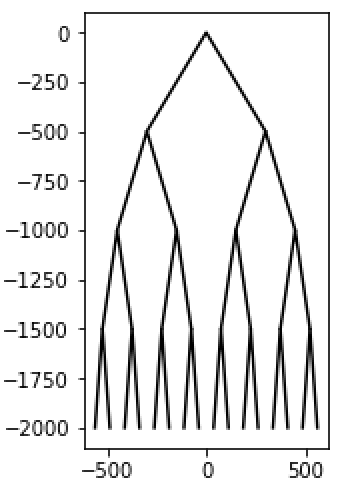
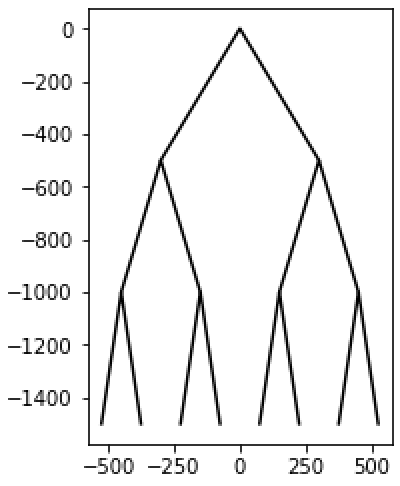


For problem two, I looked over the code Dr. Fuentes first provided for us with the practice circles and commented what each line was doing for my better understanding until I knew exactly what was going on. Then, I analyzed the sample pictures and realized that all that way happening per recursive call was that the new radius of each circle was the radius of the previous circle. Since the previous circle was smaller, the radius was closer to the x-axis. Making the following shapes per recursive call,

This problem was the easiest, all I had to do was add the rad to “x” in the method known as “circle.”

Problem three was one of the most challenging, I had to analyze the sample picture and first realized that although the tree is recursively calling more and more branches, the tree does not grow out but stay in the same bounds. Therefore, I thought about limiting the graph to two coordinates that can not be exceeded. In my case those were [-500,500]. I began by picking my coordinates where my tree begins, then I came up with an algorithm to get the tree to branch down. I had to use temp variables, in my case known as n2 and n1. Which I then add to original array p [-300,-500],[0,0],[300,-500]. The original array p is what draws the first part of the tree that looks like an upside down “V”. Then, to get the distance between p[0] and p[3] I used w1. Then I multiplied w1 by -1 to make it positive. The algorithms I used in n1 and n2 took me a long time but they only came together by messing with the points over and over again. My recursive call is then the same parameters I used in problem one, ax, n-1 but for p I used p1 and p2 being the variables that add my temp variable n1 and n2 to my original p. Resulting in,



As for problem four, I was not able to finish it or quite get it running. I tried for many days but I can not figure out what the recursive call is, how its drawing the circles inside every time, I could not catch the pattern in it. Because that’s what they all have, a pattern that you turn into code and get the recursion running.

This project taught me a lot. Although I was not able to move 2 new squares up and down a y-axis of a graph, I learned the most important concept of this lab, recursion. I understood that you need to be able to use your method for input. That is what recursion is all about. Even when I tried to draw the two missing squares in problem one, I got them to show up, but with hard-coding and that’s when I realized the important lesson of recursion. I also learned how to draw in python, which is really exciting to do.

**Problem 1 – Source Code**

# PAOLA TERRAZAS

# CS2302

# RECURSIVELY DRAWING SQUARES

# DR. FUENTES

# TA - ANINDITA NATH

# 2.10.2019

# NUM 1 SQUARES

import numpy as np

import matplotlib.pyplot as plt

def draw\_squares(ax, n, p):

if n > 0:

#n1 = [[-250, +750], [-250, 250], [-750, 250], [-750, +750], [-250, +750]]

#n2 = [[+750, -250], [+750, -750], [250, -750], [250, -250], [+750, -250]]

#equation1 = p + n1

#equation2 = p + n2

ax.plot(p[:,1], p[:,0], color='k')

equation = p /2

print(p)

# top right

draw\_squares(ax, n-1, equation+750)

# bottom left

draw\_squares(ax, n-1, equation-250)

# top left

#draw\_squares(ax, n-1, equation1)

# bottom right

#draw\_squares(ax, n-1, equation2

plt.close("all")

original\_size = 1000

p = np.array([[0,0],[0,original\_size],[original\_size,original\_size],[original\_size,0],[0,0]])

fig, ax = plt.subplots()

draw\_squares(ax,2,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('squares.png')

plt.close("all")

fig, ax = plt.subplots()

draw\_squares(ax,3,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('squares1.png')

plt.close("all")

fig, ax = plt.subplots()

draw\_squares(ax,4,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('squares2.png')

**Problem 2 – Source Code**

# PAOLA TERRAZAS

# CS2302

# RECURSIVELY DRAWING CIRCLES

# DR. FUENTES

# TA - ANINDITA NATH

# 2.10.2019

# NUM 2 CIRCLES

import matplotlib.pyplot as plt

import numpy as np

import math

def circle(center, rad): #method to find the x and y coordinates of a circle

n = int(4\*rad\*math.pi) #multiplying the radius of the circle with 4 and pi

t = np.linspace(0,6.3,n) #divides the range 0 to 6.3 in n parts and stores array in t

#ploting the x and y coordinates based on the value of t

#adding the rad to x coordinate

x = center[0]+rad\*np.sin(t)+rad

y = center[1]+rad\*np.cos(t)

return x,y

def draw\_circles(ax,n,center,radius,w): #method to draw circles on the screen

if n>0:

x,y = circle(center,radius) #getting the value of x,y from the above method

#plotting (putting a dot on the screen at x and y coordinates of k color)

ax.plot(x,y,color='k')

#calling recursively by increasing the radius value and decreasing n

draw\_circles(ax,n-1,center,radius\*w,w)

#closing the plot method

plt.close("all")

#subploting using the subplot method

fig, ax = plt.subplots()

#drawing circles with the above ax value

draw\_circles(ax, 3, [100,0], 100,.4)

#scaling the circle from data to plot units for x and y

ax.set\_aspect(1.0)

#turning on the axis of the subplot

ax.axis('on')

#showing the plot on the screen

plt.show()

#saving the observed ploting as circles.png image

fig.savefig('circlesa.png')

plt.close("all")

fig, ax1 = plt.subplots()

draw\_circles(ax1, 90, [100,0], 100,.8594537489)

ax1.set\_aspect(1.0)

ax1.axis('on')

plt.show()

fig.savefig('circlesb.png')

plt.close("all")

fig, ax2 = plt.subplots()

draw\_circles(ax2, 120, [100,0], 100,.9435443)

ax2.set\_aspect(1.0)

ax2.axis('on')

plt.show()

fig.savefig('circlesc.png')

**Problem 3 – Source Code**

# PAOLA TERRAZAS

# CS2302

# RECURSIVELY DRAWING TREES

# DR. FUENTES

# TA - ANINDITA NATH

# 2.10.2019

# NUM 3 TREES

import numpy as np

import matplotlib.pyplot as plt

def draw\_tree(ax, n, p):

if n>0:

w1 = p[0,0] - p[2,0]

w1 = w1 \* -1

w = p[0,0] - p[2,0]

w = (w/2)

n1 = [[w/2,-500],[w,-500],[w\*1.5,-500]]

n2 = [[(w/2)+w1,-500],[w+w1,-500],[(w\*1.5)+w1,-500]]

p1 = p + n1

p2 = p + n2

ax.plot(p[:,0],p[:,1],color='k')

draw\_tree(ax,n-1,p1)

draw\_tree(ax,n-1,p2)

plt.close("all")

p = np.array([[-300,-500],[0,0],[300,-500]])

fig, ax = plt.subplots()

draw\_tree(ax,3,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('tree.png')

plt.close("all")

fig, ax = plt.subplots()

draw\_tree(ax,4,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('tree1.png')

plt.close("all")

fig, ax = plt.subplots()

draw\_tree(ax,6,p)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('tree2.png')