Jon Munoz

Lab 1

CS2302

Professor: Olac Fuentes

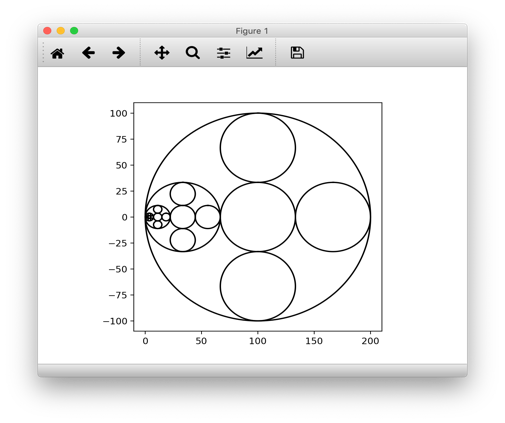
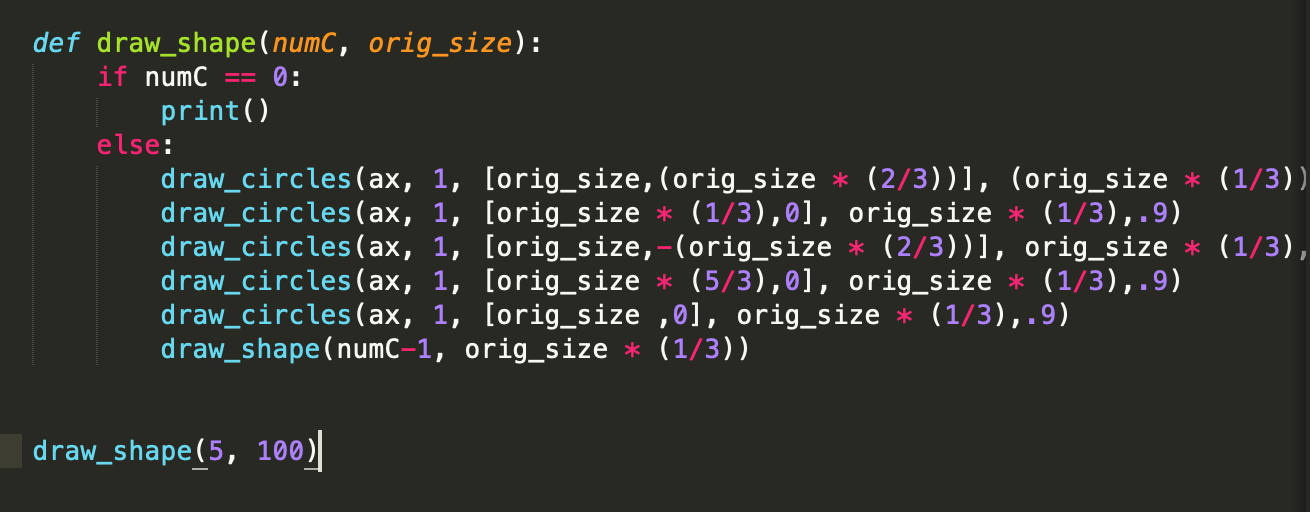
**Introduction**

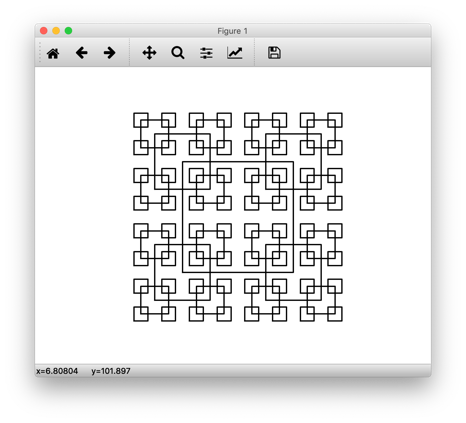
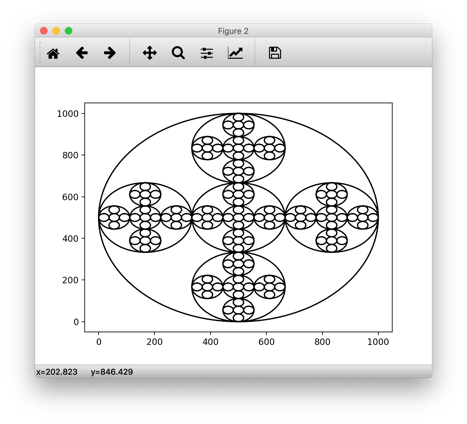
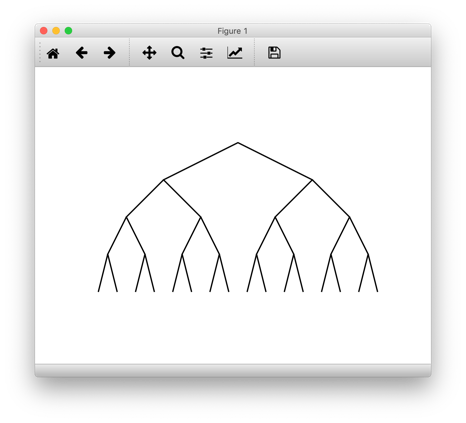
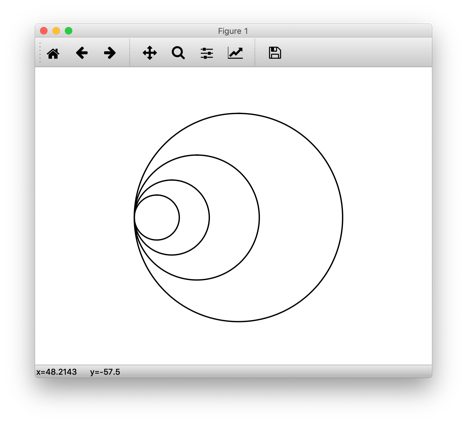
For this lab we were assigned the problem of making four unique shapes using recursion. The shapes were variations on making smaller shapes based on the original and moving them to different spots. For shape one we had to make squares in the corners of each square that we made. For shape two we had to make a smaller inside of each circle that we drew. For shape three we had to make a circle that had five smaller circles inside and we had to continue this for each circle that we drew. For the final shape we had to make a binary tree shape that got progressively smaller as the levels of recursion increased.

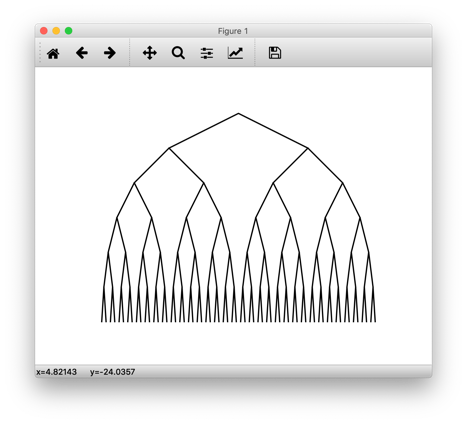
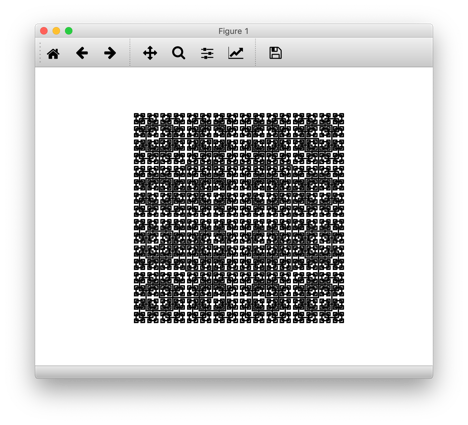
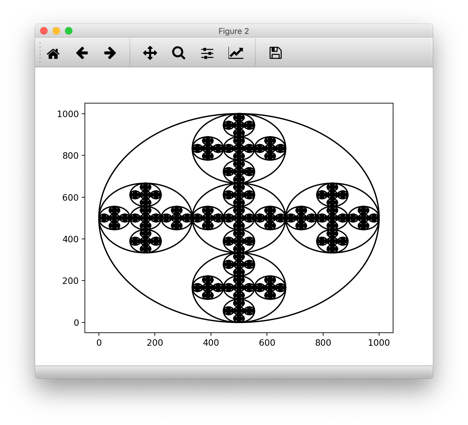
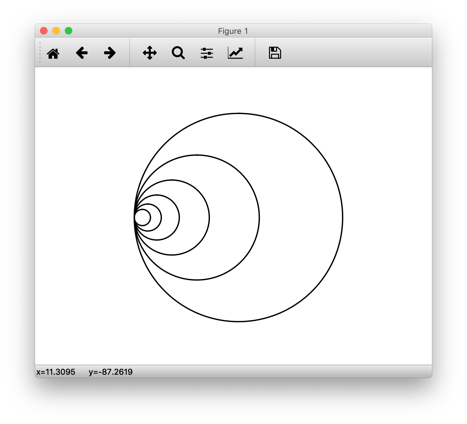
**Solution and Design**

When I first started each shape, I had the approach of having the base original shape (big square/circle/tree) made outside of my method and have the method make the smaller shapes. So for the square problem I had four calls to the draw\_squares method along with my recursive one within my own method. I did this, or tried to do this, for each one of the problems and it would yield the results that I wanted for the first level of recursion but not anything after. I then took the approach of making the base figure inside my method and then having multiple recursive calls following that. I soon learned that this would not work so I changed my approach to where I tried having the amount of recursive calls to my method be less than the amount of calls that I had to the method that drew each shape.

**Experimental Results**

For example, my original approach for the circle with five circles inside was as follows and resulted in the shown shape:

This approach ultimately did not work so I decided to try the other way that I described, and it ended up resulting in the desired outcome. Each shape had different specifications as to how much I needed to shrink each subsequent shape (for example 1/3 or 1/2) and where I moved those shapes to. When I tried this new approach I got the following shapes which were what I was originally trying to get:

As I increased the number of times that I wanted to have the method run the run time increased and the complexity of the shapes also increased. For example each shape above was with 4 recursive calls (n = 4) so the run times would be O(nlog13), O(nlog15), O(nlog16), O(nlog12) respectivley. If we increase the value of n to 6 then the shapes change and so does the run time:

**Conclusion**

At the end of this assignment I learned how to better utilize and understand recursion. My original appraoch was not correctly using recursion which is why my shapes were coming out wrong but once I sat down and put thought into it then I was able to get the outcomes that I wanted

**Appendix**

**Square Code**

import numpy as np

import matplotlib.pyplot as plt

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

if n>0:

i1 = [1,2,3,0,1]

q = p\*w + p[i1]\*(1-w)

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

draw\_squares(ax,n-1,q,w)

plt.close("all")

orig\_size = 800

fig, ax = plt.subplots()

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares.png')

#method for square shape

#levelRec controls the level of recursion, x and y control the midpoint of each square

#halfW is the distance between the midpoint and a side of the square

def draw\_shape(levelRec, x, y, halfW):

if levelRec == 0:#base case for when levelRec hits 0

print()

else:

# p = ........ sets the points for the square

p = np.array([[x-halfW,y-halfW],[x-halfW,y+halfW],[x+halfW,y+halfW],[x+halfW,y-halfW],[x-halfW,y-halfW]])

draw\_squares(ax, 1, p,.5)#draw out the square with the oints from p

draw\_shape(levelRec - 1, x-halfW, y-halfW, halfW/2)#bottom left square recursive call, while decreasing halfW by 2 to make square smaller

draw\_shape(levelRec - 1, x-halfW, y+halfW, halfW/2)#top left square recursive call, while decreasing halfW by 2 to make square smaller

draw\_shape(levelRec - 1, x+halfW, y+halfW, halfW/2)#top right square recursive call, while decreasing halfW by 2 to make square smaller

draw\_shape(levelRec - 1, x+halfW, y-halfW, halfW/2)#bpttom right square recursive call, while decreasing halfW by 2 to make square smaller

#method call

draw\_shape(6, 100, 100, 50)

**Circle Code**

import matplotlib.pyplot as plt

import numpy as np

import math

def circle(center,rad):

n = int(4\*rad\*math.pi)

t = np.linspace(0,6.3,n)

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

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

return x,y

def draw\_circles(ax,n,center,radius,w):

if n>0:

x,y = circle(center,radius)

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

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

plt.close("all")

fig, ax = plt.subplots()

fig, ax2 = plt.subplots()#second plane for second shape

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')

#method to draw shape number 2

#counter controls my number of recursive calls, xcent is the center point of my circles

#scale controls the rate at which the circles shrink

def draw\_shape1(counter, xcent, scale):

if counter == 0:#base case for when counter hits 0

print()

else:

draw\_circles(ax, 1, [xcent, 0], xcent, .9)#draws the circle for each call to the method(including recursive calls)

draw\_shape1(counter-1, (xcent\*scale), scale)#recursively calls the method while shrinking the circle in each call

#method call for shape1

draw\_shape1(6, 100, .6)

#method to draw shape number 4

#levelRec controls my number of recursive calls, x and y is the center point of my circles

#scale controls the rate at which the circles shrink

def draw\_shape2(levelRec, x,y, rad):

if levelRec == 0:#base case for when levelRec hits 0

print()

else:

draw\_circles(ax2, 1, [x,y], rad, .9)#draws the circle

draw\_shape2(levelRec - 1, x, y, rad\*(1/3)) #middle circle recursive call

draw\_shape2(levelRec - 1, x, y-((2/3)\*rad), rad\*(1/3)) #bottom cricle recursive call

draw\_shape2(levelRec - 1, x, y+((2/3)\*rad), rad\*(1/3)) #top circle recursive call

draw\_shape2(levelRec - 1, x-((2/3)\*rad), y, rad\*(1/3)) #left circle recursive call

draw\_shape2(levelRec - 1, x+((2/3)\*rad), y, rad\*(1/3)) #right circle recursive call

#method call for shape2

draw\_shape2(6, 500,500, 500)

**Tree Code**

import numpy as np

import matplotlib.pyplot as plt

def draw\_triangle(ax,n,p,w):

if n>0:

i1 = [1,0,1]

q = p\*w + p[i1]\*(1-w)

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

draw\_triangle(ax,n-1,q,w)

plt.close("all")

orig\_size = 1000

fig, ax = plt.subplots()

ax.set\_aspect(.5)#set aspect ratio to .5 to make the figure not seem so small

ax.axis('off')

plt.show()

fig.savefig('tree.png')

#my method that draws the tree where each leaf gets smaller

#counter controls my levels of recursion, x and y are my coordinates, height I use to control the height of my tree

#and rad is the "radius" of my triangles, since my original point is the middle bottom of my triangle I use rad to control the

#distance that my bottom two x points are from the center

def draw\_tree(counter, x, y, height, rad):

if counter == 0:#base case for when my counter reaches 0

print()

else:

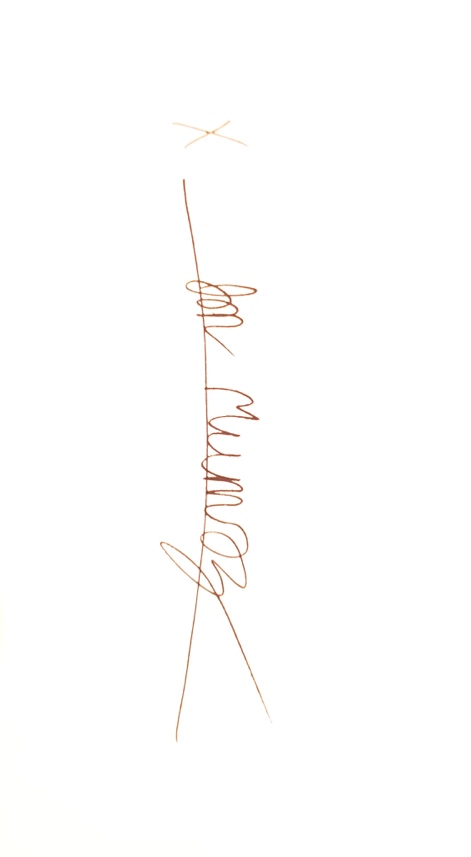
p = np.array([[x-rad,y],[x, y+height],[x+rad,y]])#this line sets the points for my triangle

draw\_triangle(ax, 1, p,.9)#this line draws the triangle

draw\_tree(counter - 1, x-rad, y-height, height, rad/2)#recursive call that sets the next triangle to the bottom left and makes it smaller

draw\_tree(counter - 1, x+rad, y-height, height, rad/2)#recursive call that sets the next triangle to the bottom right and makes it smaller

draw\_tree(6, 10, 10, 10, 10)#method call



“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provide inappropriate assistance to any student in the class.”