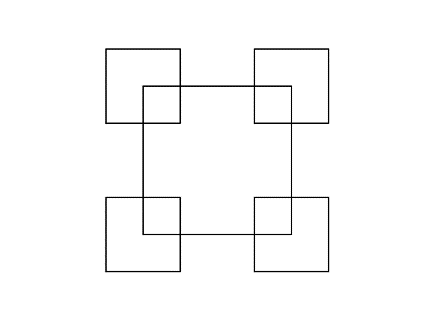
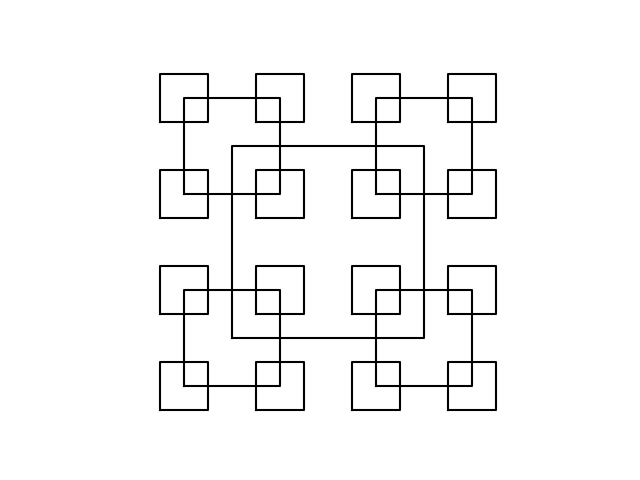
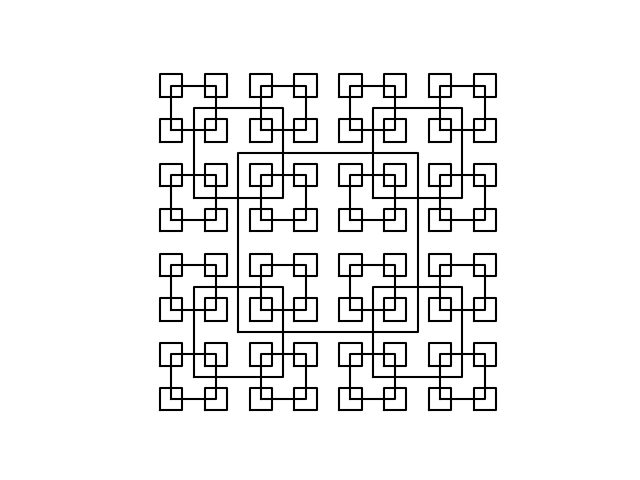
Lab one asked of us to draw fractal esc shapes by only using recursion to draw them.

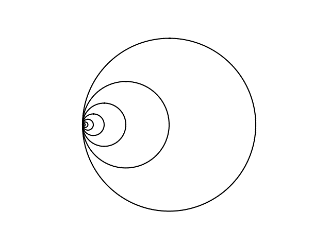
Beginning with the first problem drawing a square that gains a square in each corner with each call I began to solve the problem by modifying the code given to us earlier on how to draw a square. We wouldn’t need certain parameters that were in that code, like w the weighted average, however I would need to keep track of the radius of the square. Aptly I replaced w with a variable named radius. From there I began attempting to draw each corner square individually through a recursive call. The hardest ones to do were the top left and bottom right squares. I was able to draw the top right and bottom left squares by creating a square half the size of the last one and adding - for the top right- and subtracting - for the bottom left - the radius from the entire array containing the vertex points. To get the other two I had to physically draw on paper to understand how to move them concluding at a point to copy what I did before but now subtracting or adding to only the Y axis. The reasoning I had for this was because the next squares were on the same x axis as one of them only translated down or upwards by 2\*radius amount vertically, and as such I translated that into the code creating a separate call for each corner square created.

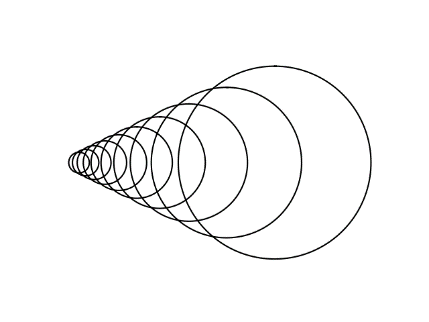
The second problem was circles which edged closer to the left side of the circle while staying inside the original circle’s radius. The first one of the problems was creating a circle half the size of the previous one, which was simple I just modified the original draw\_circles code by moving the center along the x axis half of the previous radius’ distance alongside the condition w, the weight or size the circles reduces by, is .5 or ½. The next problem was similar however the circles edge towards the edge of the circle at a different rate and reduce at a smaller rate as well. Keeping to a similar strategy I only moved the center of the circle along the x-axis this time instead of going half of the radius over I began to move over w distance at a time. This was to keep the ever-shrinking radius of the circle touching the same edge point as the largest circle in the picture as both shrink at the rate of w making the circle smaller and moving over the same distance as it shrinks keeping one tangent point between every circle made. The exact same code was used to draw the C part of the problem the only difference between the calls was w the rate of shrinking and the number of circles needed to be drawing which I found with trial and error which I will discuss later.

The third problem was personally by far the most menacing of them all. I initially broke the problem down by applying similar methods as the square problem from earlier where by knowing where the next points are I would just modify the points I store in an array of numbers that shrink the x axis as I make more calls, however this was a dead end creating odd images which did not nearly look similar to the one’s provided. At this point I turned to the professor who gave me another idea to lean towards and instead of passing an array that has the points, just pass the tip of each branching node and create an array to draw with based off that. So, I began drawing the binary tree at the point 0,0 as it’s easy to modify 0 by adding and subtracting. Dy is a constant depth that the tree goes down by and Dx is the amount in each direction either left or right of the current x point. I would create an array that draws similar to the square function from earlier drawing from an array of x and y points and recursively calling calls that would handle the left and right sides of the tree, and each would reduce the amount moving out on the x axis in half of the previous calls amount which I handled by reducing Dx by 2 each time.

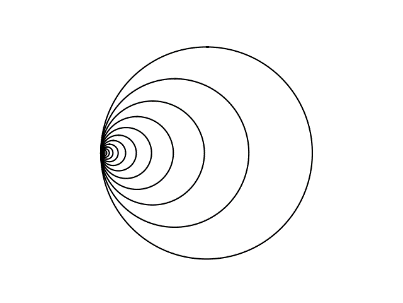
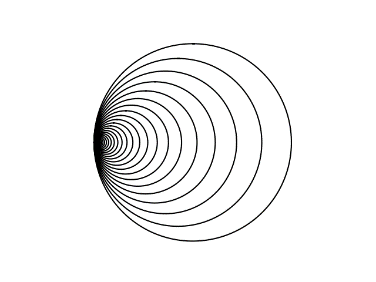
The final problem in this lab was creating a circle with 5 circles 1/3rd the size of the last one. This was a tricky problem, yet it wasn’t too challenging. With my knowledge of the squares problem I applied similar logic by creating the circles and moving them along the x or y axis. This time the circle function needed one less parameter as the circles will always shrink by a pre-determined amount, 1/3rd the size of the last which can be handled by dividing it by 3 each call. Similarly, to the square one I also gave each circle a recursive call of its own, one to hand the center most and one for each one to go around it. This runs somewhat fast as smaller calls but once the number of calls reaches 4 or 5 it can slow down significantly and that is due to the amount of recursive calls done per circle.

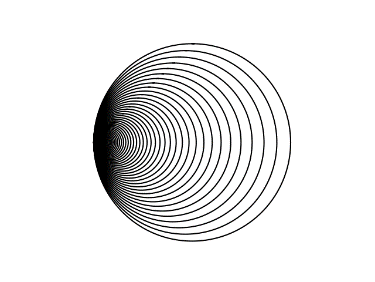
While testing my code for each problem I would vary the sizes of the shapes to ensure I wasn’t creating a size specific solution, say the squares I would vary the size of the squares to ensure the specific size wouldn’t be a variable and the function was robust enough to handle any square. Another thing I would do is individually test each square to ensure each one would be made properly. The squares portion of the program would result in the following images.

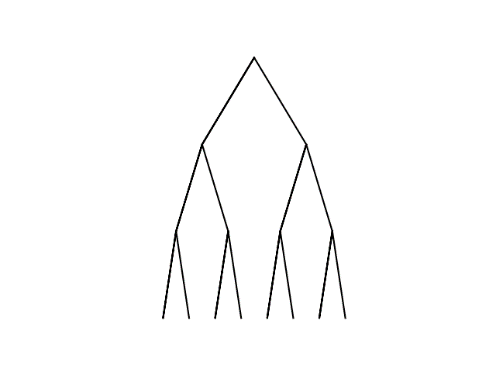
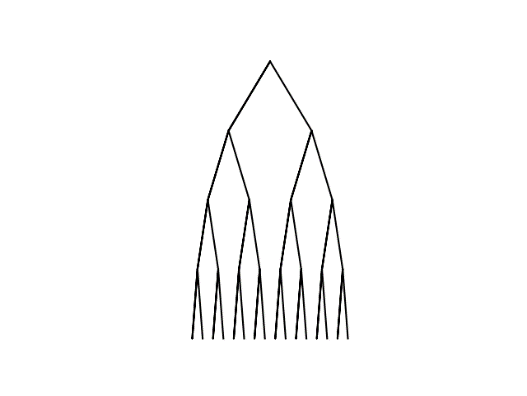
The circle code was more experimental however, I quickly got the hang of the first circle problem with little effort quickly identifying that the circle just moves over and reduces by half the radius each time making it quite quickly and effortless to code. Part a.)

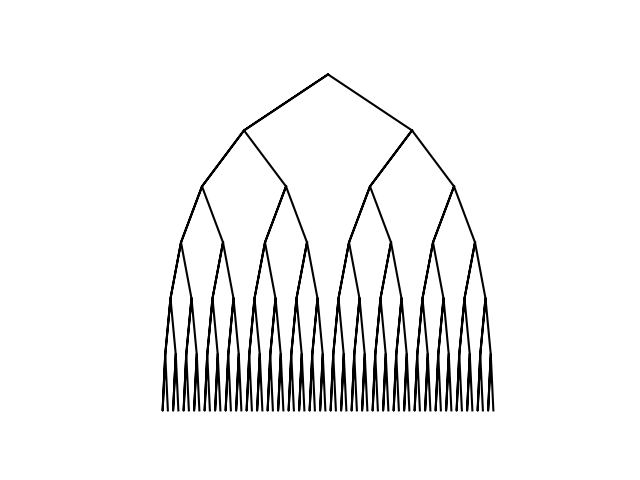
Part b is where it became tricky. I attempted to reuse the code of the first one from part a, but the circle would go out of bounds of the original circle as follows

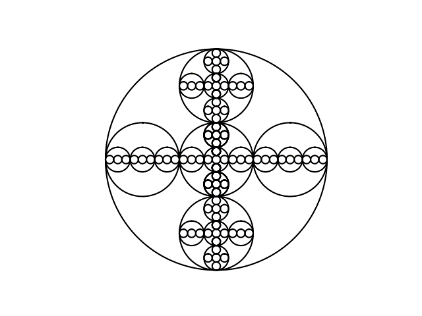
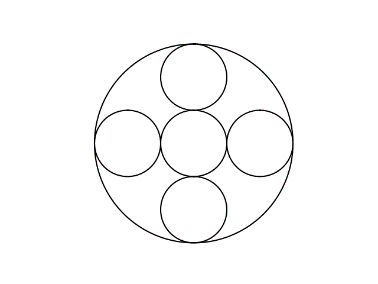
The issue was solved when It was brought to my attention that the next circle’s radius cannot move based on the radius anymore, it must move based on the rate that the circle is being reduced by giving me closer to the correct output. Once I found a suitable and working code for the problem I couldn’t quite tell from the picture at the rate that the circles were going down by in part b. As a solution I began at putting .7 as w as an educated guess, while close it was visibly off of the original figure so I began to increase it by increments of 5 until I settled that .85 was most similar as the value for w in part b giving me the following images

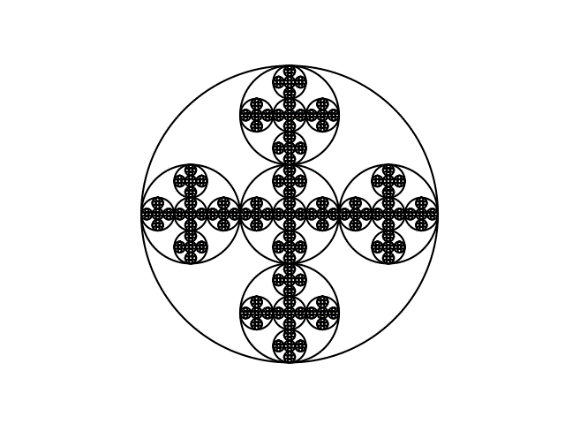
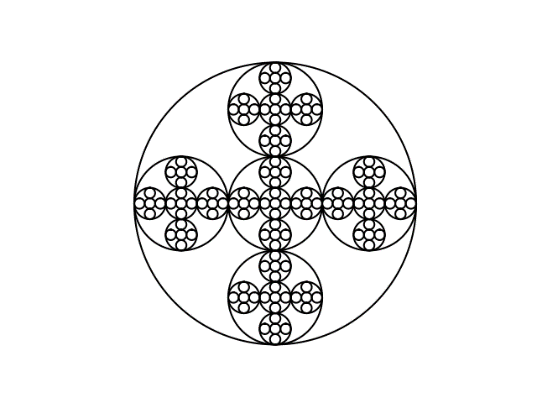
(The left is .7 the right Is .85)

Solving the final part for b was just a matter of increasing the number of circles made by a vast amount and decreasing it at a slower rate, initially guessing .9 and seeing how close I was to the given figure I guess .93 and decided that was fairly close to the original provided and stuck with the following image.

Problem 3 of the original however I cannot show my initial code for before changing my thought process as I entirely scrapped it and cannot show the output of those either as I continued to save over them by keeping the name they were saved as in the code the same. I will say that the images visually looked okay for 2 recursive calls and on the 3rd call it would begin to deform and after visiting professor Fuentes found that I was making a case specific solution for the problem by recursively giving information that only worked for the size I initially kept it at for testing which was a poor decision. Once the professor changed my view coding became simpler and while not initially changing inputs, I got a correct image, and tested it by halving the initial sizes of Dx and Dy to ensure I got a non-case specific output bestowing me this image, part b(the right image) is just part a but with one more call added to the depth of the image.

 Part c was like the circles part b and c where I attempted trial and error till I found a near matching image to the provided one. However I did have a decent starting place knowing that the Dx value or the amount going horizontally was initially wider than the amount the tree went down, Dy. Eventually through a bit of trial and error I began to produce the following image.

The final problem was interesting to code. The first call creating the main circle with 5 inside was simple and I got it working quickly(left image), afterwards I found that I wasn’t producing the vertical circles correctly after the 2nd call which left me puzzled with doubt. I kept getting the following image on the right.

After color coding some of the calls I found that the calls weren’t working as the center I based them on were only based on 0 as the x axis center point, meaning that the circles were being recursively made, they were just drawing over circles that were already made. I found my solution in passing the correct variable being passing part of the 1-D array values being the x value of the call, so instead of saying the center was at 0, center[1] I would declare it to be at center[0], center[1] which solved my issue granting me the correct image being the following image, and part c of this issue was to just add another few calls to the function.

From this entire lab I learned a few things. The first being that I should try to change my point of view myself a bit more if I keep hitting a dead end on a problem, which problem 3 gave me quite a few. The other thing I learned is that I should be testing my code with a sizable range of values to ensure it is a robust working code and not a specific case solution that works for only the values I worked with. The last thing I learned is that recursion can be a bit tricky but over thinking recursion can make the problem itself more complex than it really is, so keeping your thought process straight forward and simple can help, as well as physically drawing and writing the problems out to help better visualized them to simplify them.

Appendix –

import numpy as np

import matplotlib.pyplot as plt

import math

def draw\_squares(ax,n,p,radius):#n is number of shapes drawn, p is the the size of the shape, w is where the shape's next vertexes will apear

if n>0:

q = p//2

TLCorner = q - radius#creates new array where the bottom left corner is

for i in range(len(q)):

TLCorner[i,1] = TLCorner[i,1] + 2\*radius #changes y value to move square to top right corner

BRCorner = q + radius#creates new array for a square in top right corner

for i in range(len(q)):

BRCorner[i, 1] = BRCorner[i, 1] - 2\*radius #changes y value to move square to bottom right corner

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

draw\_squares(ax, n-1, q+radius, radius)#creates top right corner

draw\_squares(ax, n-1, q-radius, radius)#creates bottom left corner

draw\_squares(ax, n-1, TLCorner, radius)#creates top left corner

draw\_squares(ax, n-1, BRCorner, radius)#creates bottom right corner

def draw\_tree(ax, v, Dx, Dy, n):

if n > 0:

Node = np.array([[v[0], v[1]] , [v[0] - Dx,v[1] - Dy], [v[0], v[1]], [v[0] + Dx, v[1] - Dy]])

#^Code above saves current vertex and the end points of the other two branches

#by taking off how man Dx units moved to the left or right and how many Dy units down

ax.plot(Node[:,0],Node[:,1] , color = 'k')#draws tree with branches

vL = np.array([Node[1, 0], Node[1, 1]])#left vertex's corodinates

vR = np.array([Node[3, 0], Node[3, 1]])#right vertex's corodinates

draw\_tree(ax, vL, Dx/2,Dy, n-1)#draws left node with half current Dx value to make smaller branches

draw\_tree(ax, vR, Dx/2, Dy, n-1)#draws right node with half current Dx value to make smaller branches

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[0]-radius/2, 0],radius\*w,w)#cuts each circle in half of current radius

def draw\_circles2(ax,n,center,radius,w):

if n>0:

x,y = circle(center,radius)

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

draw\_circles2(ax,n-1,[center[0]\*w, 0],radius\*w,w)#creates next circle moving the center over by w

def draw\_circles3(ax,n,center,radius):

if n>0:

x,y = circle(center,radius)

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

#the radius is cut into thirds each call to fit 3 circles across in each direction

draw\_circles3(ax,n-1,[center[0], center[1]],radius/3)#next center most circle

draw\_circles3(ax,n-1,[center[0] - 2\*(radius/3), center[1]],radius/3)#left of center call

draw\_circles3(ax,n-1,[center[0] + 2\*(radius/3), center[1]],radius/3)#right of center

#moves center of the left circle over by 2 \* radius/3 of current calls center

#moves the center of the right circle over by 2 \* radius/3 of current calls center

draw\_circles3(ax,n-1,[center[0], center[1] - 2\*(radius/3)],radius/3)#below center

draw\_circles3(ax,n-1,[center[0], center[1] + 2\*(radius/3)],radius/3)#Above center

#moves the center of the top circle up by 2\* radius/3 of current calls center

#moves the center of the bottom circle down by 2\*radius/3 of current calls center

plt.close("all")

ogSize = 500

p = np.array([[-ogSize, -ogSize],[ogSize,-ogSize],[ogSize,ogSize],[-ogSize,ogSize],[-ogSize,- ogSize]])

radius = ogSize

fig, ax = plt.subplots()

draw\_squares(ax,2,p,radius)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squaresA.png')

fig, ax2 = plt.subplots()

draw\_squares(ax2,3,p,radius)

ax2.set\_aspect(1.0)

ax2.axis('off')

plt.show()

fig.savefig('squaresB.png')

fig, ax3 = plt.subplots()

draw\_squares(ax3,4,p,radius)

ax3.set\_aspect(1.0)

ax3.axis('off')

plt.show()

fig.savefig('squaresC.png')

fig, ax4 = plt.subplots()

draw\_circles(ax4, 10, [100,0], 100,.5)

ax4.set\_aspect(1.0)

ax4.axis('off')

plt.show()

fig.savefig('circlesA.png')

fig, ax5 = plt.subplots()

draw\_circles2(ax5, 30,[100, 0], 100, .85)

ax5.set\_aspect(1.0)

ax5.axis('off')

plt.show()

fig.savefig('circlesB.png')

fig, ax6 = plt.subplots()

draw\_circles2(ax6, 100, [100, 0], 100, .93)

ax6.set\_aspect(1.0)

ax6.axis('off')

plt.show()

fig.savefig('circlesC.png')

v = np.array([0, 0])#current vertex

fig, ax7 = plt.subplots()

draw\_tree(ax7, v, 60, 100,3)

ax7.set\_aspect(1.0)

ax7.axis('off')

plt.show()

fig.savefig('TreeA.png')

fig, ax8 = plt.subplots()

draw\_tree(ax8, v, 60, 100,4)

ax8.set\_aspect(1.0)

ax8.axis('off')

plt.show()

fig.savefig('TreeB.png')

fig, ax9 = plt.subplots()

draw\_tree(ax9, v,150, 100,6)

ax9.set\_aspect(1.0)

ax9.axis('off')

plt.show()

fig.savefig('TreeC.png')

fig, ax10 = plt.subplots()

draw\_circles3(ax10, 3, [0, 0], 100)

ax10.set\_aspect(1.0)

ax10.axis('off')

plt.show()

fig.savefig('templarA.png')

fig, ax11 = plt.subplots()

draw\_circles3(ax11, 4, [0, 0], 100)

ax11.set\_aspect(1.0)

ax11.axis('off')

plt.show()

fig.savefig('templarB.png')

fig, ax12 = plt.subplots()

draw\_circles3(ax12, 5, [0, 0], 100)

ax12.set\_aspect(1.0)

ax12.axis('off')

plt.show()

fig.savefig('templarC.png')

“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 provided inappropriate assistance to any student in the class.”

* Seth Abel Flores