Joaquin Hidalgo

CS 2302 MW @1:30pm

Lab 01

**Recursion**

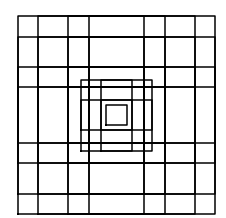
**Introduction:** The problem we are trying to solve is to use Python and understand recursion to help use draw unique shapes. For example, if you were to a square at each corner of the first square and continue to do that until you specified the desired depth, you would need to find the relation from the first square to the next squares which will lead you to your desired unique shape if designed properly.

**Proposed solution design and implementation:** The first steps to solve this problem was to learn as much as Python as possible because I am unfamiliar with the language. After learning as much Python as I could, the next step was how to apply the Python I learned to make unique shapes. Python matplotlib was the desired way to plot our shapes. Matplotlib can take in a matrix of x,y coordinates, the amount of x,y coordinates is the length of the matrix which matplotlib connects the first set of coordinates to the next and so on until there are no more. X-coordinates being the first column of the matrix and the y-coordinates being the second column of the matrix.

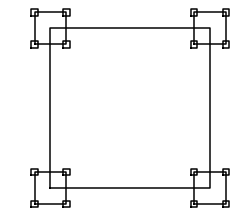
When seeing the desired shape, you have to make notice on how you are going to the draw the first shape. For most of the problems, you just need a center of the shape and how big want it to be. With just that information, I am allowed to create a square or a circle with just knowing the center and the radius or length of the side of the shape with simply geometry. I would change the center or length of shape when calling the recursion method by a ratio of some sort within the recursion method that way, the next shape is different from the first base case shape. User will define these variables with numbers before running the recursion method with great recognizable variable names that can be changed with just 1 line versus having to go through the whole code and adjust every time that particular variable is used. We will need a center point, radius or side of length, and depth of the amount of recursion. This will be my user interface to allow easy adjustments if needed. The inputs will require a radius or side length to be positive. The center shouldn’t matter if the algorithm is designed to be flexible. The output to this problem are .png images that are exported to the project files.

**Experimental results:** I would notice if the next shape is bigger or smaller from the first shape, or has the same center point as before and analyze it. I know to adjust the radius or side length of the shape by a ratio of less than 1 if the next shape is smaller, and save it as the variable, ‘w’. If the next shape was bigger, then I would know to increase the radius or length of the side by a ratio greater than 1 called, ‘w’. Also, if the center was different from the first shape, when calling the recursion method, we would analyze and adjust as needed so when the recursion is called the next time, it would have a new center. To move shapes left or right, you adjust the x-coordinates negatively or positively. If you wanted to the next shape up or down, you would change the y-coordinates negatively or positively respectively. This recursion call will happen until you have reached the desired amount of shapes specified by the user which is known as the depth of recursion and the variable being used will be called, ‘n’.

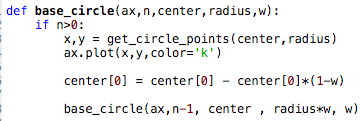
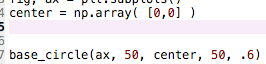
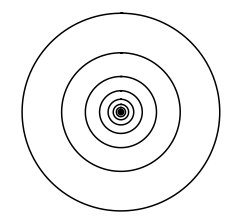
Problem 1: In the recursion call, I multiply by a ratio greater than 1 (side\*2.5) thus making each recursion shape in each corner bigger by that amount in each corner. Here we have 4 recursive calls, a square has 4 edges thus it needs 4 recursive calls. Inside each recursive call decreases ‘n’ by one thus the time is: T(n) = 4T(n-1) +1

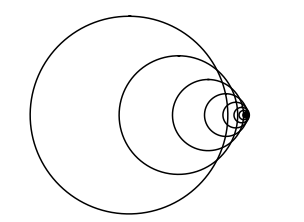
Problem 1: In the recursion call, I multiply by a ratio less than 1 (side\*.2) thus making each recursion shape in each corner smaller than previous shape. We are now getting closer to result, although the image does not look like the shape specified.

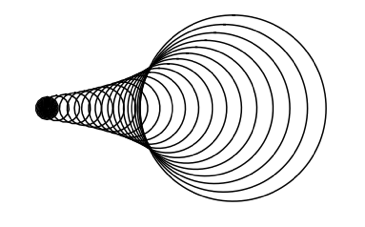
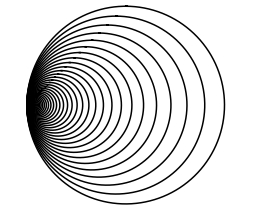
Problem 2: I learned to have your center at 0,0 is a constraint that will sometimes not move the center of the next shape. Even though I am subtracting (center[0] = center[0] – center[0]\*(1-w)) with the center at 0,0. I will not be allowed to move the center when starting at 0,0. Final solution cannot matter where the center is and should work anywhere. Here we only have 1 recursive call within the recursive call with will have a running time of: T(n) = 1T(n-1) +1



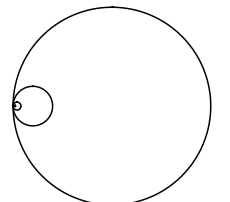
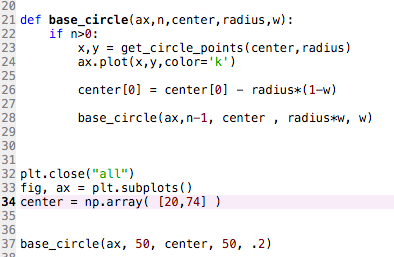
Problem 2: Here I thought I would just would add the base x-coordinate (center[0] ) by half of the radius. I learned that I need to subtract previous center by something else other than just ½ of the radius.



Problem 2: Here I changed the radius. The radius on the picture to the left has a radius of 10. On the other hand, the radius on the right is, 10000. Radius has a directly correlation to the center of the next circle made before due to this line of code. More circles mean longer runtime

Problem 2: Here I used .2 as the variable ‘w’. ‘W’ will be a big factor of that spacing from the first circle to the next. ‘W’ will need to be used accordingly to fit desired shape. 1 recursive call which is because each circle only needs 1 recursive call to execute the shape and will have a running time of: T(n) = 1T(n-1) +1

**Conclusion:** I learned how to use Python a little bit more. I learned how to upload onto GitHub. The main thing is to analyze how the first recursion call is related to the next, adjust to those specifications in the recursion call that way every next shape is closer to how you want it to be. Recursion preforms 1 task at a time, then it goes onto the next. It keeps doing this until a certain point which is called our base case. I learned to recognize a pattern, if the pattern continues to repeat itself, I can use recursion to tackle the problem.

**Problem #1**

import numpy as np

import matplotlib.pyplot as plt

def base\_sqaure(ax,n,side, c):

#Base case

if n>0:

#Create matrix 5x2 of square plots

p = np.array([ [c[0] - side//2, c[1] - side//2 ], [c[0] - side//2, c[1] + side//2],[c[0] + side//2, c[1] + side//2],[c[0] + side//2, c[1] - side//2],[c[0] - side//2, c[1] - side//2] ])

#plot the matrix

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

#Everything below this line is performed after the recursion call..

#p[0] = Bottom left square center point

base\_sqaure(ax,n-1,side\*.5,p[0])

#Top left square

base\_sqaure(ax,n-1,side\*.5,p[1])

#Top right square

base\_sqaure(ax,n-1,side\*.5,p[2])

#Bottom right square

base\_sqaure(ax,n-1,side\*.5,p[3])

plt.close("all")

fig, ax = plt.subplots()

#Paramters

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

orig\_side\_length = 800 # Length of the side

center = [0,0] #Center coordinates of the 1st square

depth = 2 # Depth of recursion squares

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# The Recursive call

base\_sqaure(ax,depth,orig\_side\_length, center)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('square\_1a.png')

**Problem #2**

import matplotlib.pyplot as plt

import numpy as np

import math

#This method returns all x,y coordinates that generates a circle with sin&cos

def get\_circle\_points(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 base\_circle(ax,n,center,radius,w):

#base case

if n>0:

#get x,y coordinates of circle

x,y = get\_circle\_points(center,radius)

#draw the circle

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

# Generate new x,y coordinates of the NEXT circle

center[0] = center[0] - radius\*(1-w)

#Generate new radius of the NEXT circle

radius = radius\*w

# recurrsion call with new center and radius decreasing by ratio 'w'

base\_circle(ax,n-1, center , radius, w)

plt.close("all")

fig, ax = plt.subplots()

#Paramters

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x\_coord = 12

y\_coord = 26

radius\_length = 100

ratio = .6 #Best if between 0 and 1

depth = 10 #Number of circles

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

center\_point = np.array( [x\_coord, y\_coord] )

#The recurrsion call

base\_circle(ax, depth, center\_point, radius\_length, ratio)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circle\_1a.png')