Ivan Perez (80602918)
CS2302
Lab 1 Report

Introduction

The lab requires us to make at least 4 recursive methods to create figures. These methods should be able to repeat the same patterns however many times needed. The square figure must have squares made in each corner of each previously recurred square. The circle has the center point shifted so that it seems like its expanding from a left point. The tree has to be able to make 2 branches per each point within the layer. The second circle figure must produce 4 equally sized and placed circles within a previously recurred circle.

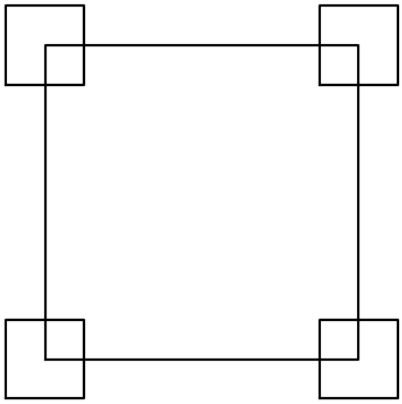
Proposed Solution

- For the square figure I passed the number of layers and the coordinates of the center of each square that needed to be made. I also had the length of a side passed so that I could adjust the coordinates by adding or subtracting the length depending on the square corner.
- The first circle figure just needed tweaking from Mr. Fuentes's original code. The center point needs to be adjusted so that each following circle has their center shifted to the left. Changing the x coordinate by having it shift with the provided decimal
- The tree has to have coordinate points pass to each call and make 2 lines and coordinates with every call. The angular degree made by each 2 lines also needs to be modified depending on the number of levels. Modifying the x coordinate should be able to alter the degree.
- The second circular figure has to pass 5 coordinates recursively per each previous circle to create the five inner circles. Modify the circles center to be placed in every third of the diameter and change the radius by thirds as well will create the figure.

Experimental results

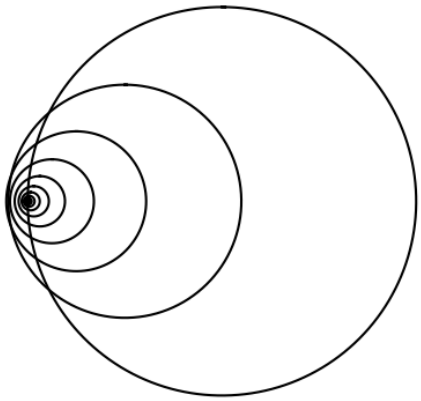
Note: All runtimes were run with the same aspect and file saving code. Any differences between any figures execution is mostly the layers used, any other difference will be mentioned.

Mr. Fuentes's original code had squares made based on the coordinates kept within an array. Keeping this in mind I altered the code to take one coordinate and create the rest of the coordinates within the method based on the length of the previous square. I changed the size of the length to find the correct one (in relation to the code I created the variable rl by dividing length by 2). Below is a result of dividing the length by 4.



- Runtime with 2 layers : .09 seconds - recursive_squares(shape1a,2,[0,0],100)
- Runtime with 4 layers : .25 seconds - recursive_squares(shape1c,4,[0,0],100)

Altering the code already provided to us, we simply had to change the center coordinates from where the circle would be focused on. The center are based on coordinates, the desired figure showed the center should be moved to left on the x axis. Changing the x coordinate to different numbers I found that the circles would move out of the original circle (example shown below). I came to the conclusion that the circles must change their coordinate in a uniformly manner. I decided to multiply by the variable w which moved it along the way it should.

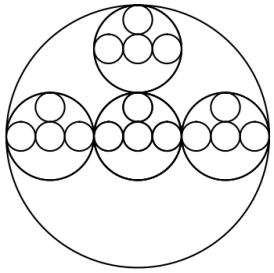


- Runtime with 50 layers and .6 of a change : .15 seconds
- radiating_circles(shape2a, 50, [100,0], 100,.6)
- Runtime with 100 layers and .95 of a change : .28 seconds
- radiating_circles(shape2c, 100, [100,0], 100,.95)

I was not able to completely finish the tree as I could not figure out how to skinny the tree over time in layers. I thought of moving the x coordinates by multiplying them by a variable to "skinny" the point. I was left with the result below (multiply the x coordinates in the q array by .5)

- Runtime with 3 layers : .14 seconds - draw_tree(shape3a,3,[0,0],1)
- Runtime with 6 layers : 4.89 seconds - draw_tree(shape3c,6,[0,0],1)

The penta circles figure was more of an easy figure to solve. Looking at the desired result, each circle was placed in every third of the diameter so the centers and radius of each circle was changed by a third. I then had to put a loop to run the recursive method 5 times to create all the circles needed. Initially I just had the recursive method run four times which gave the figure below. You can accomplish this by having the loop run while h is less than 4



- Runtime with 2 layers : .17 seconds - penta_circles(shape4a, 2, [100,0], 100)
- Runtime with 4 layers : 3.35 seconds - penta_circles(shape4c, 4, [100,0], 100)

Conclusions

This assignment made recursive methods more visual and easier to understand. Recursion is hard to wrap your head around but shows how simple code can be. This assignment was difficult but made me think twice about how I was approaching problems.

Appendix

```
#################################################################
###############
#
# Course: CS2302 (MW - 1:30pm)
# Author: Ivan Perez
# Assignment: Lab 1
# Instructor: Olac Fuentes
# T.A.: Anindita Nath
# Last Modified: Feb 8, 2019
# Purpose: Uses recursive methods to create figures with a
repetitive pattern.
# shapes consists of squares, circles, and binary tree like
patterns. Each
# shape is customizable to display desired number of pattern
layers.
#
#################################################################
###############
import matplotlib.pyplot as plt
import numpy as np
import math

def recursive_squares(sq,n,origin,length):
    if n>0:
        x = origin[0]
        y = origin[1]
        rl = length/2 #revised length

        #creates coordinates of square
        cord = np.array([[x-rl,y-rl],[x+rl,y-rl],[x+rl,y+rl],
                         [x-rl,y+rl],[x-rl,y-rl]])
        sq.plot(cord[:,0],cord[:,1],color='k') #plots points and
lines

        h = 0 #counter
        while h < 4:
            recursive_squares(sq,n-1,cord[h],length/2)
            h += 1

def circle(center,rad): # creates specs of required circles
    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 radiating_circles(circ,n,center,radius,w):
    if n>0:
        x,y = circle(center,radius)
        circ.plot(x,y,color='k')
        #w alters the size and center of plotted circles
        radiating_circles(circ,n-1,[center[0]*w,0],radius*w,w)

def draw_tree(ax,n,p,w):
    if n>0:

        t = ([p[0],p[0]-w], [p[1],p[1]-w],
             [p[0],p[0]+w], [p[1],p[1]-w])

        ax.plot(t[0], t[1], t[2], t[3],color='k')
        x = ([t[0][0], t[1][0]], [t[0][1], t[1][1]],
             [t[2][0], t[3][0]], [t[2][1], t[3][1]])

        g=0
        while g <4:
            draw_tree(ax,n-1, x[g],w)
            g += 1

def penta_circles(ax,n,center,radius):
    if n>0:
        x,y = circle(center,radius)
        ax.plot(x,y,color='k')
        cx = center[0]
        cy = center[1]
        p = (radius*2)/3 #configures radius for smaller circles
        # creates center coordinates for inner circles
        q = np.array([[cx, cy], [cx+p, cy], [cx-p, cy], [cx,
cy+p], [cx, cy-p]])
        h = 0
        while h < 5: # loops recursive call for all the smaller
inner circles
            x,y = circle(q[h],radius*1/3)
            ax.plot(x,y,color='k')
            penta_circles(ax,n-1,q[h],radius*1/3)
            h +=1

        penta_circles(ax,n-1,center,radius*1/3) #general
recursive call

plt.close("all")
```

```
#################################################################
###############
# The code below are almost identical for each figure. The code
consists of
# calling the recursive methods above to make the assignments
figure. It sets
# aspect, removes the axis, and saves each photo under their
respective names


#############   1
#1.a
fig, shape1a = plt.subplots()
recursive_squares(shape1a,2,[0,0],100)
shape1a.set_aspect(1.0)
shape1a.axis('off')
plt.close("all")
fig.savefig('1a.png')

#1.b
fig, shape1b = plt.subplots()
recursive_squares(shape1b,3,[0,0],100)
shape1b.set_aspect(1.0)
shape1b.axis('off')
plt.close("all")
fig.savefig('1b.png')

#1.c
fig, shape1c = plt.subplots()
recursive_squares(shape1c,4,[0,0],100)
shape1c.set_aspect(1.0)
shape1c.axis('off')
plt.close("all")
fig.savefig('1c.png')

#############   2
#2.a
fig, shape2a = plt.subplots()
radiating_circles(shape2a, 50, [100,0], 100,.6)
shape2a.set_aspect(1.0)
shape2a.axis('off')
plt.close("all")
fig.savefig('2a.png')

#2.b
fig, shape2b = plt.subplots()
```

```
radiating_circles(shape2b, 50, [100,0], 100,.9)
shape2b.set_aspect(1.0)
shape2b.axis('off')
plt.close("all")
fig.savefig('2b.png')

#2.c
fig, shape2c = plt.subplots()
radiating_circles(shape2c, 100, [100,0], 100,.95)
shape2c.set_aspect(1.0)
shape2c.axis('off')
plt.close("all")
fig.savefig('2c.png')

##############   3
#3.a
fig, shape3a = plt.subplots()
draw_tree(shape3a,3,[0,0],1)
shape3a.set_aspect(1.0)
shape3a.axis('off')
plt.close("all")
fig.savefig('3a.png')

#3.b
fig, shape3b = plt.subplots()
draw_tree(shape3b,4,[0,0],1)
shape3b.set_aspect(1.0)
shape3b.axis('off')
plt.close("all")
fig.savefig('3b.png')

#3.c
fig, shape3c = plt.subplots()
draw_tree(shape3c,6,[0,0],1)
shape3c.set_aspect(1.0)
shape3c.axis('off')
plt.close("all")
fig.savefig('3c.png')

##############   4
#4.a
fig, shape4a = plt.subplots()
penta_circles(shape4a, 2, [100,0], 100)
shape4a.set_aspect(1.0)
shape4a.axis('off')
plt.close("all")
```

```
fig.savefig('4a.png')

#4.b
fig, shape4b = plt.subplots()
penta_circles(shape4b, 3, [100,0], 100)
shape4b.set_aspect(1.0)
shape4b.axis('off')
plt.close("all")
fig.savefig('4b.png')

#4.c
fig, shape4c = plt.subplots()
penta_circles(shape4c, 4, [100,0], 100)
shape4c.set_aspect(1.0)
shape4c.axis('off')
plt.close("all")
fig.savefig('4c.png')

plt.show()
```