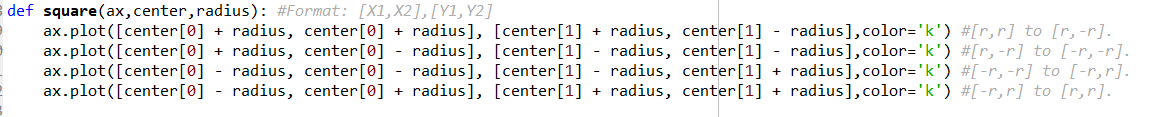
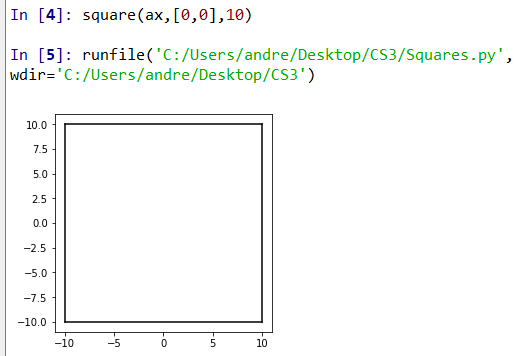
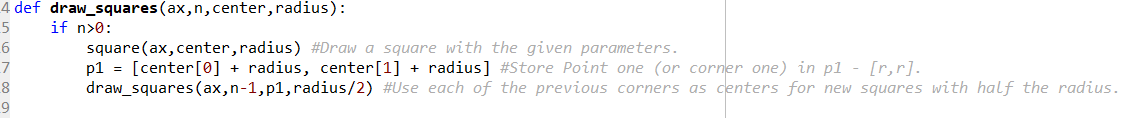
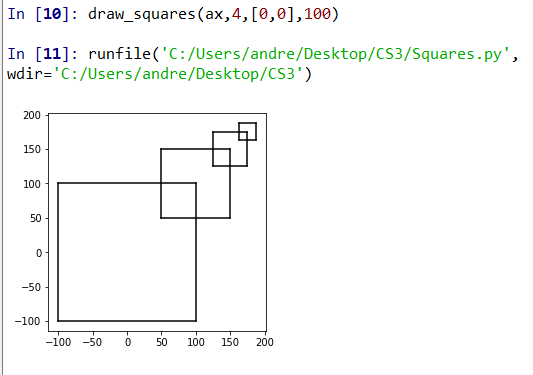
Lab 1 report – Andres Silva - 80588336

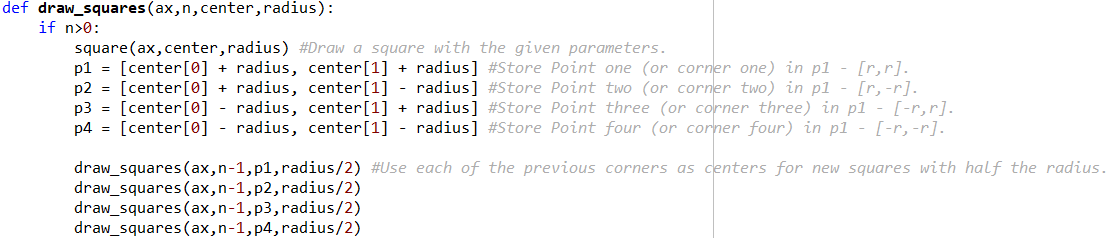
The initial prompt for lab 1 indicated that the task at hand was to draw the specified figures using the model code provided. This code, with small modifications, will yield the first part of the exercise, but it does require extensive changes in order to produce the rest of the outputs. The main problem was to understand the current code, specifically how the plotting of lines was handled. The recursive part of the exercise was also a challenge, but one that was covered in class, helping further to grasp possible solutions. Paying close attention to the model code of the circles made it clear that having a separated method to plot figures with given dimensions could prove useful, but only after understanding the python plot.

Following this mentioned approach, a method to draw a square was the first thing to be implemented. The method should not be hardcoded to draw a specific circle of defined proportions, so it was clear that the method should receive a center and a radius. By adding and subtracting said radius to the center, the 4 points of the corners can be obtained and then lines can be plotted between those points. If the radius is denoted by ‘r’, then the coordinates of the corners consist of [r,r], [r,-r], [-r,r], and [-r,-r].

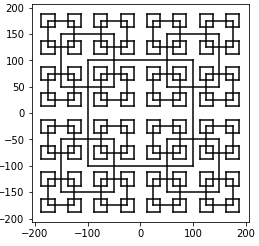
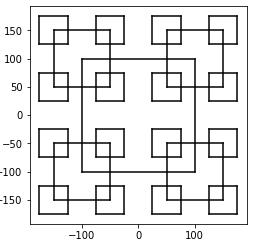
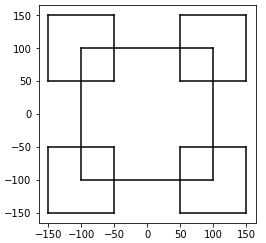
Calling this method with a radius of 10, and a center [0,0] should produce a square with corners of at [10, 10], [10, -10], [-10, 10], and [-10, -10].

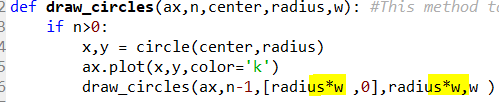
The recursive method should then receive this initial center for the first square and the radius. Each of the smaller squares should be centered at the corners of the bigger square, and should be smaller in size, so the recursive call should take the previous calculated corners as centers and by paying attention to the model figures, it is easy to conclude that the radius of the smaller squares is half of the initial one.

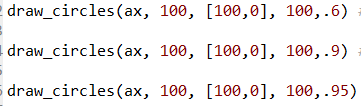
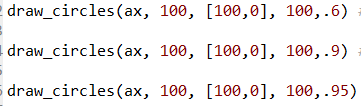
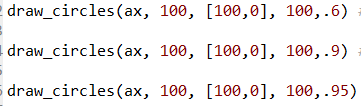
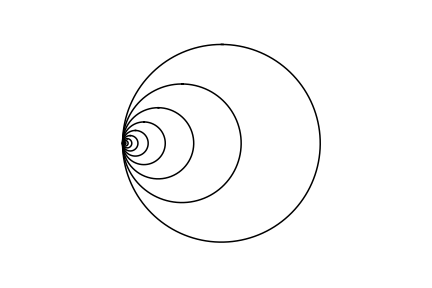
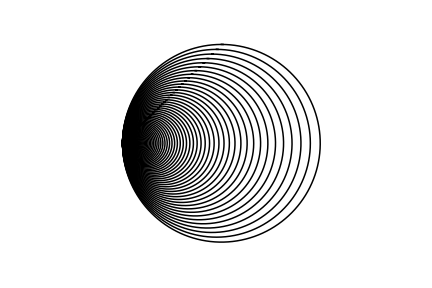
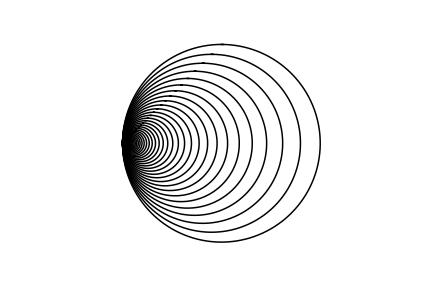
Doing this recursive call with said measurements and a specified level of recursion ‘n’ should output a square in the corner of a bigger square n times.

With this accomplished, it is possible to get the desired figure by having a recursive call for each corner, so each square has 4 squares as shown by *lab1.pdf*

Calling this method with different values on *n*, and same parameters as before shows how each recursive call takes care of each corner.



 The second problem only required to modify the recursive call of the model code. Two things should change with each iteration of the method: The X coordinate of the center and the radius of the circle. We can do this by having a scalar ‘w’ that will multiply them in each recursive call.



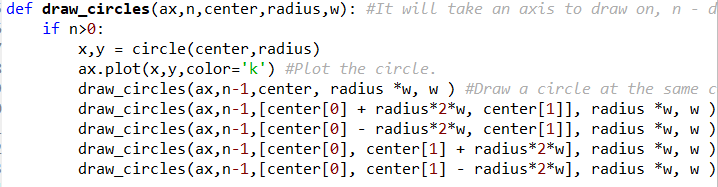
Note in the outputs above that the closer the scalar *w* is to 1, the less the circle shifts and decreases. The correct parameters were found through trial an error to produce figures similar to those indicated in the instructions.

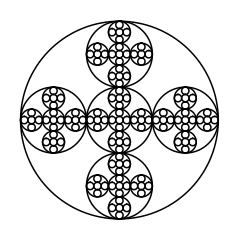
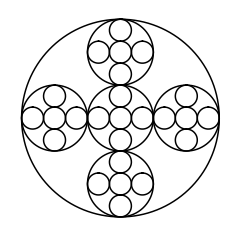
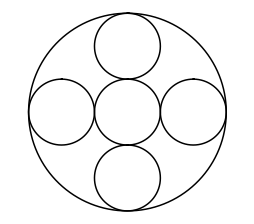
The other circles exercise was similar, but more modifications were used. Once again, the method that draws a circle with a provided center and radius was left intact and with the experience from the squares problem, the idea of having a recursive call for each circle was intended to be implemented.

The proposed solution is also similar to the squares problem: draw a circle at a certain point relative to the center of the first circle with a smaller radius. In this case we need to draw one exactly at the center, one above it, below, to the left, and right. Thus, recursion will be called with a shifted center and a smaller radius (which is reduced by scalar ‘w’). The scalar is 1/3, which when multiplied by 2 and the radius will be added to the center coordinates to find the new centers of each circle. It was chosen 2r\*1/3 because the diameter of each smaller circle will be a third of the current diameter (2r).

The following method with the different parameters below will produce the respective figures.

Note how each recursive call adds or subtracts to the original center in order to move to the previously discussed directions.

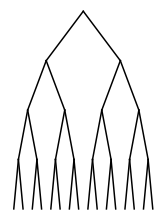
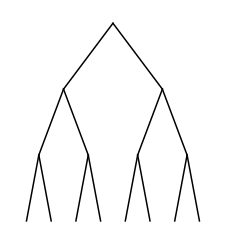
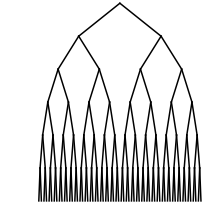




The tree problem was the most difficult to solve because it is not a very common shape whose proportions and points are often analyzed. The approach was similar to the previous problems, this time a method that plots 2 lines between 3 points was created. These points will be referred to as center, point 1 (p1), and point 2 (p2). After paying a close look to the provided images of the desired outputs some things were noted: The height of each ‘level’ of the tree is the same, the width between p1 and p2 decreases by some amount each iteration, the child points of each center are the new centers of each new branch, the Y coordinate of both p1 and p2 decreases, the X coordinate increases or decreases depending on the direction the branch is going to follow (increases for right, decreases for left).

The recursive method then consisted of the following: ‘n’ levels of recursion, a center, segment\_height (a fixed height for each level), and segment\_width (the fixed width of the whole tree). Note how there is no p1 or p2 in the parameters, this is because the points are calculated by the method by adding and subtracting a fraction of segment\_width. The exact fraction by which the width decreases was found by taking an educated guess and trying different numbers (by analyzing the provided figure it was definitive that it was going to be either 2 or 4).

Sample inputs and outputs are shown below.



Conclusion:

In this lab I learned a few important things: Start the assignment as soon as it is posted, asking the Tas for guidance can be very helpful in completing the lab, how in recursive methods it can be convenient to implement multiple recursive calls, instead of the single one we were used to, and that reading the python documentation for new libraries can make all the difference in properly coding.

**Appendix**

**Circles Model Code**

import matplotlib.pyplot as plt

import numpy as np

import math

def circle(center,rad): #Model code without modification.

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): #This method takes the axis to draw on, level of recursion, circle radius, and a factor to scale (reduce) by.

if n>0:

x,y = circle(center,radius)

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

draw\_circles(ax,n-1,[radius\*w ,0],radius\*w,w ) #This recursive call will call itself with n-1 (to advance), an X coordinate shifted to the left by w, a reduced radius, and the same reduction factor.

plt.close("all")

fig, ax = plt.subplots()

draw\_circles(ax, 100, [100,0], 100,.6) #Here we call the "main" method with different values to produce different figures.

#draw\_circles(ax, 100, [100,0], 100,.9) #An scaling factor closer to one will produce circles closer to the original size each iteration.

#draw\_circles(ax, 100, [100,0], 100,.95)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles-3.png')

**Squares Code**

import matplotlib.pyplot as plt

#Here we have a method that will draw a square with a provided center and radius.

#The method calculates the 4 points of the square by substracting or adding the radius to the center.

#The method will then plot straight lines between those corners.

def square(ax,center,radius): #Format: [X1,X2],[Y1,Y2]

ax.plot([center[0] + radius, center[0] + radius], [center[1] + radius, center[1] - radius],color='k') #[r,r] to [r,-r].

ax.plot([center[0] + radius, center[0] - radius], [center[1] - radius, center[1] - radius],color='k') #[r,-r] to [-r,-r].

ax.plot([center[0] - radius, center[0] - radius], [center[1] - radius, center[1] + radius],color='k') #[-r,-r] to [-r,r].

ax.plot([center[0] - radius, center[0] + radius], [center[1] + radius, center[1] + radius],color='k') #[-r,r] to [r,r].

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

if n>0:

square(ax,center,radius) #Draw a square with the given parameters.

p1 = [center[0] + radius, center[1] + radius] #Store Point one (or corner one) in p1 - [r,r].

p2 = [center[0] + radius, center[1] - radius] #Store Point two (or corner two) in p1 - [r,-r].

p3 = [center[0] - radius, center[1] + radius] #Store Point three (or corner three) in p1 - [-r,r].

p4 = [center[0] - radius, center[1] - radius] #Store Point four (or corner four) in p1 - [-r,-r].

draw\_squares(ax,n-1,p1,radius/2) #Use each of the previous corners as centers for new squares with half the radius.

draw\_squares(ax,n-1,p2,radius/2)

draw\_squares(ax,n-1,p3,radius/2)

draw\_squares(ax,n-1,p4,radius/2)

plt.close("all")

fig, ax = plt.subplots()#Call method below this line.

draw\_squares(ax,3,[0,0],100

ax.set\_aspect(1.0) #And before this one.

ax.axis('on')

plt.show()

fig.savefig('squares.png')

**Trees Code**

import matplotlib.pyplot as plt

#This is a method that will draw 2 lines between 3 given points.

#The method will use a center and plot lines between it and two other lower points to make an inverted V.

def lines(ax,center,p1,p2):

ax.plot([center[0],p1[0]],[center[1],p1[1]] ,color='k')#Format: [X1,X2],[Y1,Y2]

ax.plot([center[0],p2[0]],[center[1],p2[1]] ,color='k')

#The following method will take an ax, n times to do recursion, "segment\_height" (which is how tall each level of the tree will be, and a "segment\_width" (which is the defined width of the tree)).

def draw\_trees(ax,n,center,segment\_height,segment\_width):

if n>0:

#The following children points are calculated relative to the center. They will always be segment\_height-tall and segment\_width on the X axis.

p1 =[center[0] - segment\_width/4 ,center[1] - segment\_height] #The two children points are calculated and stored in point 1 (p1) and point 2 (p2).

p2 =[center[0] + segment\_width/4 ,center[1] - segment\_height] #Note how the X coordinate of the point becomes smaller each iteration, and how the Y coordinate decreases by a constant value.

lines(ax,center,p1,p2) #With those points calculated, we call the lines method to simply draw lines between those calculated points.

#Now we have a recursive call for each child point.

draw\_trees(ax,n-1,p1,segment\_height, segment\_width/2) #Each of the previous points now becomes a center, which will have two more children.

draw\_trees(ax,n-1,p2,segment\_height, segment\_width/2) #Note how n will stop recursion, how the center is now a point previously calculated, how the segment\_height stays constant, and the width decreases even more.

plt.close("all")

fig, ax = plt.subplots()#Call method below this line.

draw\_trees(ax,4,[0,0],1000,3000) #Run it with 3,4 for n and 3000 - then run it with n = 6, but last parameter 5000 just for better detail.

#lines(ax,[0,0],[-500,-1000],[500,-1000]).

ax.set\_aspect(1.0) #And before this one .

ax.axis('off')

plt.show()

fig.savefig('trees.png')

**Circles Code**

import matplotlib.pyplot as plt

import numpy as np

import math

def circle(center,rad): #Model code stays unmodified, it just traces a circle with a center and a radius.

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

#The following method will create 5 circles each iteration after the first one.

def draw\_circles(ax,n,center,radius,w): #It will take an axis to draw on, n - depth of recursion, a center, radius, and a scaling factor.

if n>0:

x,y = circle(center,radius)

ax.plot(x,y,color='k') #Plot the circle.

draw\_circles(ax,n-1,center, radius \*w, w ) #Draw a circle at the same center than the previous one (Center circle), but with a third (w) of the radius.

draw\_circles(ax,n-1,[center[0] + radius\*2\*w, center[1]], radius \*w, w ) #Call to draw a circle with a shifted center by 2/3 to the right with a third of the radius.

draw\_circles(ax,n-1,[center[0] - radius\*2\*w, center[1]], radius \*w, w ) #Call to draw a circle with a shifted center by 2/3 to the left with a third of the radius.

draw\_circles(ax,n-1,[center[0], center[1] + radius\*2\*w], radius \*w, w ) #Call to draw a circle with a shifted center by 2/3 up with a third of the radius.

draw\_circles(ax,n-1,[center[0], center[1] - radius\*2\*w], radius \*w, w ) #Call to draw a circle with a shifted center by 2/3 down with a third of the radius.

plt.close("all")

fig, ax = plt.subplots()

#draw\_circles(ax,2,[0,0],1000,1/3)

#draw\_circles(ax,3,[0,0],1000,1/3)

draw\_circles(ax,4,[0,0],1000,1/3)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles3.png')

**Academic Honesty Statement**

“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.”

* Andres Silva.