Nancy Hernandez

Olac Fuentez

Lab 1 Report

**Introduction**

The objective of this lab was to generate specific patters by drawing them using matplotlib. To accomplish this, I used the code provided in draw\_squares.py and draw\_circles.py and modified the code to make the four different patterns.

**Proposed Solution**

I attempted to solve the first part by using the function available in the matplotlib called Rectangle. I thought about how I needed to change the origin, or the center, for every new rectangle being placed on each of the corners of the last rectangle. For me to achieve this I thought about creating two if statements. One that would just draw the first rectangle if the number of repetitions was equal to one, and the other if statement would be for when the number of repetitions was more than one. Both if statements should contain a line of code that draws the first, and biggest, rectangle. Now, for the second if statement, I thought about calling the method itself four different times because there are four different corners in each rectangle. Finally, after finishing the method, you must call the method one more time. This will create the final figure.

For the second part I used the code provided to create the circle shape. I then thought about making a new method that would have a recursive call of the previous method given, so that I could actually draw the figures. Since I know that the radius is already decreasing, the key thing in solving this specific figure is to change the center instead. But for the figure to be drawn correctly I should call this same method recursively inside itself. Outside of this method I also called the same method but this time I called it with the dimensions I thought would recreate the figure just like the picture provided in the lab instructions.

In the third part I thought that I would create two different methods, one that drew the first two initial lines and the other that would draw the rest of the line patters. In the first method you need to define the length as the three points being connected with two lines. This will create the shape for the rest of the other lines following. Now, for the second method I thought about using an if statement that allows the rest of the recursion to take place. Inside this if statement I began by calling the first method so that it draws the initial lines. I then made the new origins the bottom two points so that the pattern begins from there. After this method I called the method again recursively. This should be done so that the code recreates the figure as close as possible. I was only able to create a similar figure as the part a of the third figure. As it is, I also couldn’t make it look exactly like the picture given. I had a lot of trouble figuring out how to solve this part of the lab.

Finally, for the last part, like the second part I used the code provided once again to create the initial circle shape. I then thought about creating another method that uses an if statement that makes sure that when the number of repetitions is more than zero, to draw the other circles that make up the figure. I felt that it would be easier to draw the center most circle first so that is what I did, and I did this by calling the first method but multiplying the radius by a third. Next, I created four different variables that define the four different circles around the first initial circle. I multiplied the radius in all four of these variables by two thirds. And according to the position of the circle I added or subtracted each x and y coordinate of the center. After this I thought about calling this same method four different times but substituting the center for the previous four different circles that are being put around the initial circle. Once finished with the method all there should be left to do is call the method recursively to create the final figure.

**Pseudo Code**

*Pseudo Code for Part 1:*

0: if numTimes == 1:

1: #Draws the initial rectangle

2: currAx.add\_patch(Rectangle((x, y), length, length, alpha = 1, Fill = None))

3:

4: elif numTimes > 1:

5: currAx.add\_patch(Rectangle((x, y), length, length, alpha = 1, Fill = None))

6:

7: #top left

8: draw\_squares(x - length / 4, y + (length - length / 4), length \* .5, numTimes - 1)

9: #top right

10: draw\_squares(x + (length - length / 4), y + (length - length / 4), length \* .5, numTimes - 1)

11: #bottom right corner

12: draw\_squares(x + (length-length / 4), y - length / 4, length \* .5, numTimes - 1)

13: #bottom left

14: draw\_squares(x - length / 4, y - length / 4, length \* .5, numTimes - 1)

*Pseudo Code for Part 2:*

0: def circle(center, rad):

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

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

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

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

5: return x, y

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

8: if n > 0:

9: x, y = circle(center, radius)

10:

11: #Creates plot to draw in

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

14: #Decreases the center

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

17: #Recursive call

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

*Pseudo Code for Part 3:*

0: def draw\_first\_lines(ax, p, length):

1: rad = length / 2

3: #connects top point to bottom two points

4: length = np.array([[p[0] - rad, p[1] - (2 \* rad)], [p[0], p[1]], [p[0] + rad, p[1] - (2 \* rad)]])

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

7: def draw\_lines(ax, p, numPeaks, length):

8: rad = length / 2

9: if numPeaks >= 2:

11: #draws biggest peak(top peak)

12: draw\_first\_lines(ax, p, length)

13: #makes bottom points the new origins

14: leftPoint = [p[0] - rad, p[1] - (rad \* 2)]

15: rightPoint = [p[0] + rad, p[1] - (rad \* 2)]

17: #Recursive calls

18: draw\_lines(ax, leftPoint, numPeaks - 1, length \* .4)

19: draw\_lines(ax, rightPoint, numPeaks - 1, length \* .4)

*Pseudo Code for Part 4:*

0: def circle(center, rad):

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

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

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

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

5: return x, y

7: def draw\_mult\_circles(ax, numTimes, center, radius):

8: if numTimes > 0:

9: x, y = circle(center, radius)

10: #Creates plot to draw in

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

13: #Draws center circle

14: draw\_mult\_circles(ax, numTimes - 1, center, radius \* (1 / 3))

16: #draws circle on left side of center

17: radLeft = [center[0] - (radius \* (2 / 3)), center[1]]

18: #draws circle Above center

19: radUp = [center[0], (radius \* (2 / 3)) + center[1]]

20: #draws circle on Right side of center

21: radRight = [(radius \* (2 / 3)) + center[0], center[1]]

22: #draws circle Below center

23: radDown = [center[0], center[1] - (radius \* (2 / 3))]

25: #Recursive calls that create figure

25: draw\_mult\_circles(ax, numTimes - 1, radLeft, radius \* (1 / 3))

25: draw\_mult\_circles(ax, numTimes - 1, radUp, radius \* (1 / 3))

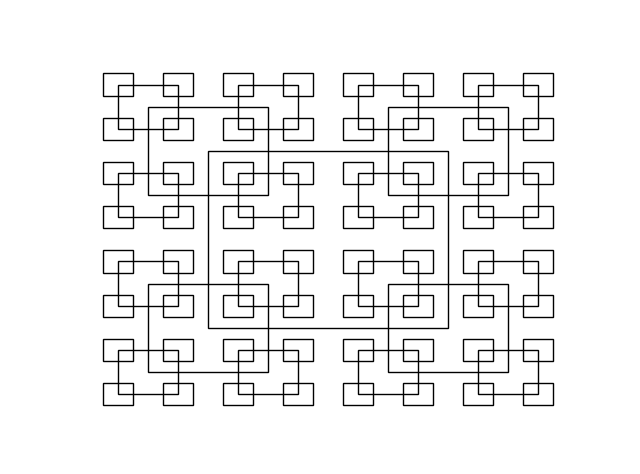
25: draw\_mult\_circles(ax, numTimes - 1, radRight, radius \* (1 / 3))

25: draw\_mult\_circles(ax, numTimes - 1, radDown, radius \* (1 / 3))

**Setup**

To complete this lab I used an HP Pavilion x360 Convertible with a 2.71 GHz Intel® Core(TM) i5 processor.

**Results**

**A close up of a keyboard

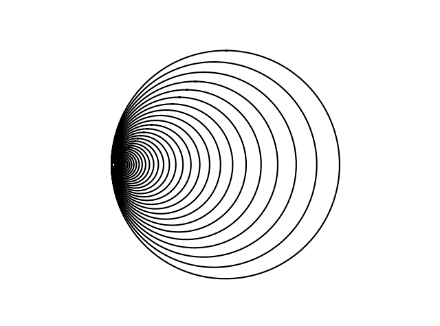
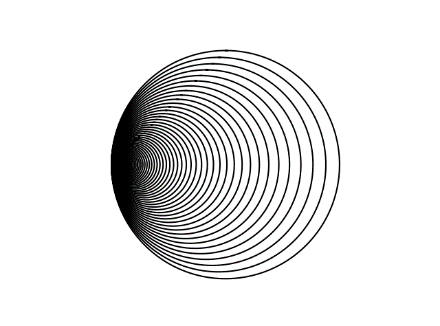
Description automatically generatedA close up of a logo

Description automatically generated1** (a), (b), (c)

I could not figure out how to create a square and apply it to this code, so I researched on googled how to create a rectangle using a function matplotlib provides, as a rectangle is the closest thing to a square.

I tested the program for these figures by modifying the values entered into the recursive call. Which in this case, changed the size and the number of repetitions.

**A close up of a mans face

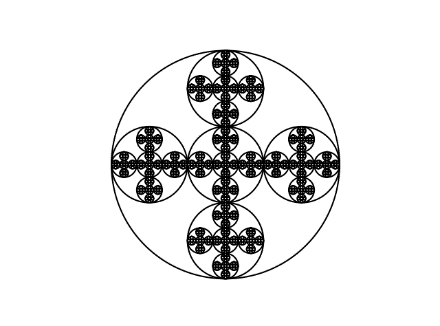
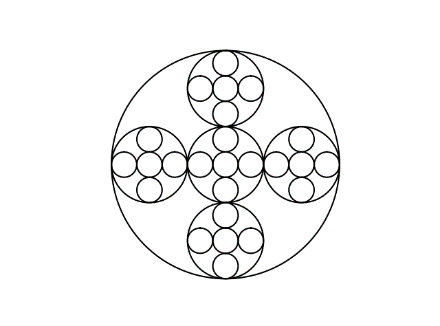
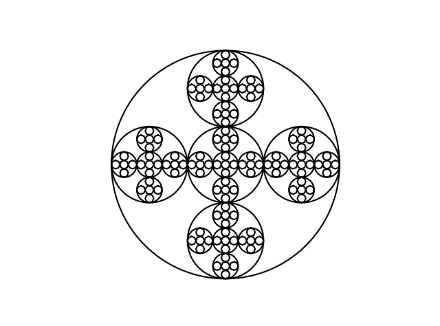
Description automatically generated2** (a), (b), (c)

For this part I tested the program by changing the side in which the center would progressively move. Also by manipulating the number of circles being drawn.

**3** (a), (b), (c)

A close up of a logo

Description automatically generatedMy program for this specific part of the lab was not showing proper results for the last two figures (b) and (c).

**4** (a), (b), (c)

In order to test this section of the lab I increased or decreased the number I wanted the pattern to repeat. As well as modifying the radius of the circles.

**Conclusion**

From this project I learned how to use matplotlib to create figures that have unique patterns. I also learned how recursion works when it comes to creating unique figures. Aside from drawing and using recursion, I also became more acquainted with using python as this is my first time really writing code in this language. It was interesting noticing the big and small difference between python and java.

**Appendix**

**//Draw Squares**

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

p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]])

fig, ax = plt.subplots()

draw\_squares(ax,15,p,.8)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares.png')

**//Draw Circles**

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()

draw\_circles(ax, 50, [100,0], 100,.9)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')