CS3 Report Lab One

Daniela Flores

Intro

–Lab one consisted of writing various methods to draw figures which we had to use the two methods provided to us by the Instructor as a starting guide. We had to write these methods recursively, that is, with no loops. The first figure consisted of drawing circles within circles, with the centers at the side of the radius. The second figure we had to draw was a square with a square in each of its four vertices. The third figure we had to draw was a circle with five smaller circles inside. The fourth and last figure we had to draw was an upside-down tree.

Purposed Solution Design and Implementation

–I wrote the methods in a way that each method took care of one of the four drawings we had to implement. For the figures with squares, the parameters that were sent to the square's method were a variable which contained the number of times the method would call itself, the center of the first square, and the length from the center of the square to the edge of the square. With this information the method would then draw the first square given the center and the length. After the square was drawn the length of the square was cut in half to make the next set of squares smaller. Given the four vertices of the square already drawn, four variables then store those points locations(x,y). These four variables that store the vertices of the square are then used to call the method recursively again, with the only varying factor is the center will now be one of the four vertices of the square. The method will keep drawing squares in the vertices of the squares already drawn till the variable that keeps track of the number of iterations is zero. I then called the same method three times, with the variable of the number of set of squares varying.

For the method to draw circles one of the methods provided to us by the instructor was of great help as one only had to change a few things from the method. The circles being drawn had to have its center on the touching its left edge. Adding the radius to the center changed the center of the circle from the middle to the left. The method was then only called once until the variable that kept track of the number of circles wanted to be drawn reached zero. I then called this method three times, the only parameter that changed was the amount of times the method would call itself.

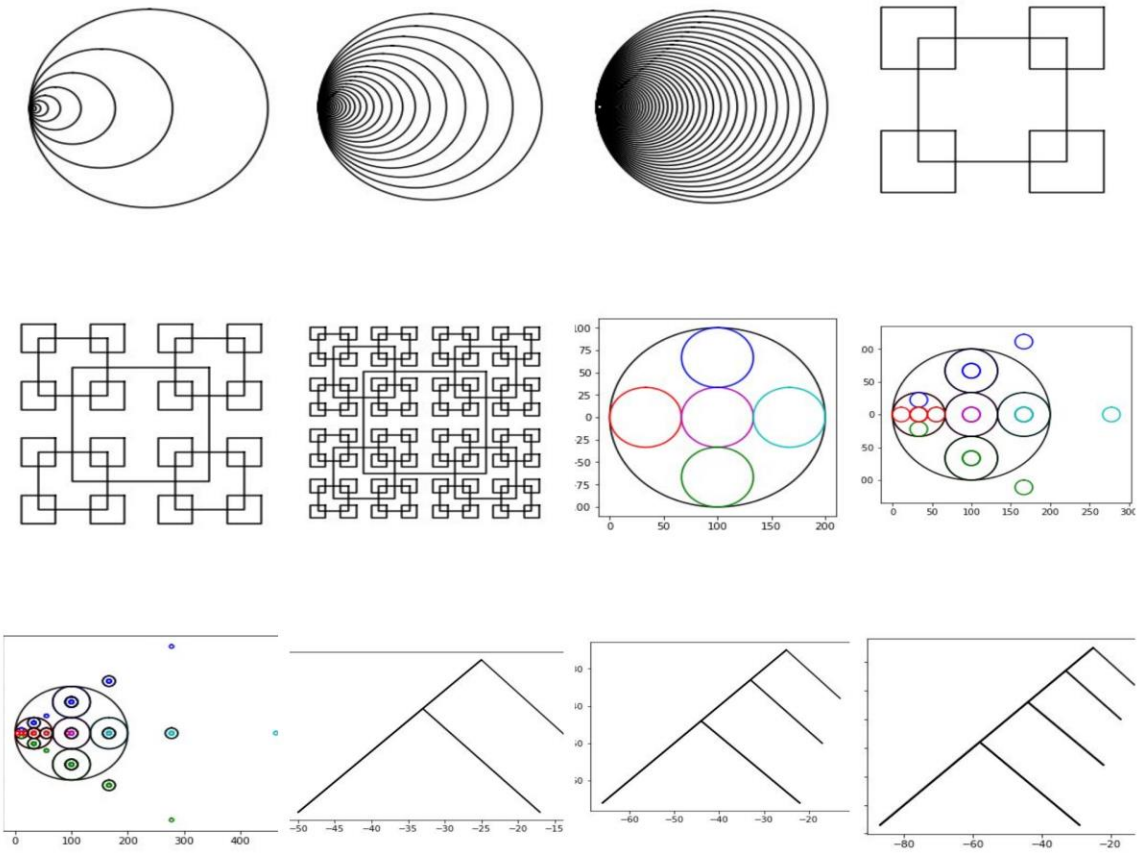
The third figure we had to draw was a bit more challenging as we had to draw five circles inside a bigger circle. In the method to draw this figure its parameters where a variable to keep track of the amount of times the recursion will take place, the center and radius of a circle. When this method is called, a circle is drawn with the given center and radius. After that the radius is then cut to a third. The center of the leftmost circle is calculated by creating its new center a third of the original center to the center e.g. (x= 1/3*center.) the circle in the middle had the same center as the bigger square, but the radius was a third of the biggest circle. The rightmost circle’s center is calculated by adding 2/3 of the original center to the

center. The circle on top's center is calculated by multiplying the centers x value by two thirds and adding that to the y value. The center of the circle at the bottom is calculated by multiplying the x value to by negative two thirds and adding it to the Y value. The method was then called five times, one for each center of the five smaller circles. This method was then called three times, the only parameter that changed was the amount of times the method would call itself.

The last figure we had to draw was a tree. The 'root' of the tree at the very top, with two branches emerging from it and at each tip of the branches, two smaller set of branches emerged from it till the number of times the method would call itself recursively reaches zero. The parameters of the method that draws this figure was the coordinate(x,y) of the root and the desired length each branch would be. The method would first draw the first set of branches from the center. Afterwards it will calculate the coordinates of the tip of the branches that'll come from the tip of the parent branch. The method would then call itself two times; one with the coordinates of the tip of the right branch and the other with the left's branch coordinate and pass them as the new 'center.' for every iteration the length was divided by a third to keep the branches well aligned. I called this method three times, the only variable that changed was the one that stored the number of recursive iterations the method would do on itself.

Experimental Results:

When I first started working on the lab, I had to manipulate or 'dissect' the two methods provided to us by Fuentes as I needed to understand what every variable did and how certain calculations were used to draw figures. I experimented with both methods by changing the values of variables, also the parameters of said methods and by commenting out certain lines of code and running it to see what role those lines played in the method. Aside from that experimentation I also experimented with the methods I wrote after figuring out the calculations needed to draw the figures. I experimented with my methods my increasing and decreasing the variable that stored the number of times the method would call itself recursively. By doing this, I was able to find the exact parameters needed to draw figures that matched the ones in the lab instructions. The following images is what my program output after running my program:



Conclusions:

What I learned from this project is the syntax needed to draw figures within a cartesian plane using python. I also learned why the **import matplotlib.pyplot as plt** library, which was used to draw the figures, and the **import numpy as np**, which was used to turn lists into arrays and why **import math**, which was used in this program to calculate the dimensions of the circles, are useful libraries. I also learned how the same method can be calling itself multiple times with different parameters. Working in this lab also helped me better understand the syntax of python as I just started working with this language.

Apprendix(source code):

```
#Course:CS2302

#aAuthor:Daniela Flores

#Lab1

#Instructor:Olac Fuentes
```

```
#T.A: Dita Nath
#date of last Mod: 2/8/19
#purpose of program: in this program I had to write recursive methods that'll draw various shapes.
import matplotlib.pyplot as plt
import numpy as np
import math
#the next two methods draw circles within the previous circle, the center is the radious
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)
        #here the radius is added to the center
        ax.plot(x+radius,y,color='k')
        draw_circles(ax,n-1,center,radius*w,w)
#call where 10 circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 10, [100,0], 100,.6)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles.png')
```

```
#call where 40 circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 40, [100,0], 100,.87)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles.png')

#call where 62 circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 62, [100,0], 100,.94)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles.png')
#############################################################################
#the following methods draw squares in the vertices of the larger square
def draw_squares(ax,n,p,r,c):
    if n>0:
        #i1 = [[c[0]+radius],2,3,0,1]
        newCor = np.array([ [c[0]+r,c[1]+r], [c[0]-r,c[1]+r],[c[0]-r,c[1]-r],[c[0]+r,c[1]-r],[c[0]+r,c[1]+r]])
        #                   0+25 , 0+25  //  0-25 , 0+25  // 0-25 , 0-25 // 0+25 , 0-25//   0+25, 0+25
        #                     (25,25)      (-25,25)       (-25,-25)       (25,-25)        (25,25)
        ax.plot(newCor[:,0],newCor[:,1],color = 'k')
        #with respect to center:
        newR = r//2
        newCenter =  [c[0]+r,c[1]+r]
```

```
        newCenter2 = [c[0]-r,c[1]+r]
        newCenter3 = [c[0]-r,c[1]-r]
        newCenter4 = [c[0]+r,c[1]-r]

        draw_squares(ax,n-1,newCor,newR,newCenter)
        draw_squares(ax,n-1,newCor,newR,newCenter2)
        draw_squares(ax,n-1,newCor,newR,newCenter3)
        draw_squares(ax,n-1,newCor,newR,newCenter4)

#call where 2 sets of circles are drawn
plt.close("all")
radius = 40
center = [0,0]
p = np.array([[-radius,-radius],[-radius,radius],[radius,radius],[radius,-radius],[-radius,-radius]])
fig, ax = plt.subplots()
draw_squares(ax,2,p,radius,center)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('squares.png')
#call where 3 sets of squares are drawn
plt.close("all")
radius = 40
center = [0,0]
p = np.array([[-radius,-radius],[-radius,radius],[radius,radius],[radius,-radius],[-radius,-radius]])
fig, ax = plt.subplots()
draw_squares(ax,3,p,radius,center)
ax.set_aspect(1.0)
ax.axis('off')
```

```
plt.show()
fig.savefig('squares.png')
#call where 4 sets of squares are drawn
plt.close("all")
radius = 40
center = [0,0]
p = np.array([[-radius,-radius],[-radius,radius],[radius,radius],[radius,-radius],[-radius,-radius]])
fig, ax = plt.subplots()
draw_squares(ax,4,p,radius,center)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('squares.png')
################################################################################
#the following methods draw 5 smaller circles inside the bigger circle
def circle2(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_circles2(ax,n,center,Radius):
    if n>0:
        x,y = circle2(center,Radius)
        ax.plot(x,y,color='k')
        Radius = Radius/3
        newCenterL =np.array( [((1/3)*center[0]),0])
```

```
        x,y = circle2(newCenterL,Radius)
        ax.plot(x,y,color='r')

        newCenterM = np.array([center[0],0])

        x,y = circle2(newCenterM,Radius)
        ax.plot(x,y,color='m')
        newCenterR = np.array( [(((2/3)*center[0])+center[0]),0])

        x,y = circle2(newCenterR,Radius)
        ax.plot(x,y,color='c')
        newCenterUp =np.array( [center[0],((2/3)*center[0])])

        x,y = circle2(newCenterUp,Radius)
        ax.plot(x,y,color='b')
        newCenterDown =np.array( [center[0],((-2/3)*center[0])])
        x,y = circle2(newCenterDown,Radius)
        ax.plot(x,y,color='g')

        draw_circles2(ax,n-1,newCenterL,Radius)
        draw_circles2(ax,n-1,newCenterM,Radius)
        draw_circles2(ax,n-1,newCenterR,Radius)
        draw_circles2(ax,n-1,newCenterUp,Radius)
        draw_circles2(ax,n-1,newCenterDown,Radius)
#here 2 sets of circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 1, np.array([100,0]), 100)
ax.set_aspect(1.0)
```

```
ax.axis('on')
plt.show()
fig.savefig('circles.png')
#here 3 sets of circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 2, np.array([100,0]), 100)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('circles.png')
#4 sets of circles are drawn
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 3, np.array([100,0]), 100)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('circles.png')
########################################################################
#the following methods draw an upside down tree
def draw_triangle(ax,n,p,len,center):
    leftsubL = [-len,-len]
    rightsubL =[len,-len]
    treeAr = np.array([leftsubL,center,rightsubL])
    if n>0:


        LL = ((leftsubL[0]+leftsubL[0]//2),(leftsubL[0]+leftsubL[0]//2))
```

```
        newRightC = [(leftsubL[0]+leftsubL[1]//-2),(leftsubL[0]+leftsubL[0]//2)]
        new = np.array([leftsubL,LL,leftsubL,newRightC])

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

        newLeftR = [(rightsubL[0]-rightsubL[1]//-2),(rightsubL[0]-rightsubL[0]//-2)]

        newRightR = [(rightsubL[0]+rightsubL[0]//2),(rightsubL[0]+rightsubL[1]//2)]
        new2 = np.array([rightsubL,newLeftR,rightsubL,newRightR])
        # ax.plot(new2[:,0],new2[:,1],color = 'k')
        newLen = len+(len//3)



        draw_triangle(ax,n-1,treeAr,newLen,leftsubL)
        draw_triangle(ax,n-1,treeAr,newLen,rightsubL)

#here 2 branches are drawn
plt.close("all")
length = 25
center = [0,0]
p = np.array([[-length,-length],[center[0],center[1]],[length,-length]])
fig, ax = plt.subplots()
draw_triangle(ax,2,p,length,center)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('squares.png')
```

```
#here three branches are drawn
plt.close("all")
length = 25
center = [0,0]
fig, ax = plt.subplots()
draw_triangle(ax,3,p,length,center)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('squares.png')
#here 4 sets of branches are drawn
plt.close("all")
length = 25
center = [0,0]
fig, ax = plt.subplots()
draw_triangle(ax,4,p,length,center)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('squares.png')
```

Honesty signature:

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 inappropiate assistance to any student in the class.

X