Recursion is the repeated application of a recursive procedure. In programing it usually used as a technique that involves the use of a procedure, subroutine, function, or algorithm that calls itself in a step. It will contain a termination condition called the base case so that the consecutive repetitions are executed up to the critical step where the condition is met. In lab one we had to use recursion to draw interesting figures. I will be using pycharm as my integrated drive electronics (IDE) to program in the python3 language. I will also be using the libraries numpy, math, and matplotlib and two programs draw\_squares.py and draw\_circles.py provided by our teacher Olac Fuentes.

For the first set of figures I had to create a function that would draw squares and recursively draw squares on every corner of all previous squares. The way I started to try and solve this problem was by first looking at the example images provided in the lab. The first thing I notice was that every square that was drawn in the corners where smaller. I estimated that the squares where about a ¼ of the original size of the first squares diameter. I also notice that every new square used the corners as the center of the new squares. Now knowing this I decided to just modify the points every time I called my method by a modifier ‘r’ that would be the original size of the diameter by .25 as a parameter. I would also pass the first set of points for the original square as a double array by using numpy.array function as the parameter ‘p’. Once inside the function I would plot the first shape by using the plot function on p and then proceed to modify my previous points of p into a new numpy double array. I created two variables that would hold the x component and y component of my first corner of p and use them to plot my new square by adding or subtracting r my modifier. After that if I still had loops of recursion left I would append my array q into a list that I passed as a parameter as ‘li’. This would hold all the points for those squares so that I can draw squares on the corners. After that I would then plot my new set of point by using the plot function. After all of the first recursion squares are drawn, I would then start to pop my list of arrays in to a temporary variable and passed them in to my function as the p parameter. All of this would work fine. It would draw the number of squares on all corners but there was a problem in my function. Every time i would call my function for the new set of squares they would become smaller. I figure that i had to change where I modify my r again but after trial and failure I was not able to find a proper solution. In the end my function creates the new set of squares but every time i pop my new set of points they become smaller and smaller.

For the second set of figures I have to create a function that would draw circles and slowly move the the center of the cericale to the left in the x direction and make the size of the circle smaller. For this one it was a lot more easier to make because the draw\_circles.py had everything I needed. In the draw\_circles.py there are two functions. The first function is called circle and it takes a parameter called center and rad. Center is the coordinates of the center of the circle and rad is the radius the circle will have. The next function is called Draw\_circles and it takes ax which is the shape is stored, n which is the number of recursion calls, center which is the original center of the first circle, radius which is the original radius of the first circle and ‘w’ which is the modifier for the radius. Inside the draw\_scircles function is where you call the previous function circle and return all the x coordinates and the y coordinates that will be used to draw the circles. After the xs and ys are returned they are plotted using the .plot function and then we call the draw\_circles function until all recursion calls are made. So the solution to the problem was simple all I had to do was modify the x coordinate of the center to move to the left by adding the radius to the x direction when I called the circle function and every time the function draw\_circles was called inside the function i would modify the radius by w to make the radius smaller. After that the program works just the same as before but now the center of the circle is moved to the left by adding the radius.

The third set of shapes was the hardest for me and I was not able to solve. The third set of shapes was on the shape of a binary tree. I tried doing the same solution in the first set of shapes where I would pass a origin that would represent as the root of the tree and a modifier to the x and the y of the center. After that in the method I would used the origin to create an double array that would contain 3 set of coordinates and modify the two others coordinates that would make the left side and the right side of the tree i would then store all of them in a array where i would call them latter. The problem came for when I would use the recursion call. I would use the function plot to plot all the x set of coordinates to the y set of coordinates, but when that shape was printed to the screen it would no look like a tree because all the points where connected to all previous points.

The last set of shapes was not as hard but I still found difficulties. For this set of figures I had to draw circle that would then draw smaller circles in the center, left of the center, right of the center, top of the center, and bottom of the center. This would continue until all the recursion calls where finish. My approach to this problem was to create a function call draw that would take ax the shape, n the number of recursion calls, radious the original radius of the first circle, and the modifier ‘w’. Inside the function I would create the center of all the circles and then call the draw\_circles using all the new center as the value for the parameter center. The problem was that only three of the circles was draw and that the center of this new circles would all be on the ring of the first bigger circle. I tried to modify the points by changing the difference of the radius multiple times but would end up with something completely different than to the solution of the problem.

In the end I learned how well your recursion call must be and how you have to have a well developed base case. If your program does not have a recursion call that does not work correctly you will end up with something different from what you want. For the base case you need something that will actually terminate the recursion call, otherwise you will crash your computer multiple times. Overall I feel like I’ve become better at using recursion functions.

#CS2302

#Made by Anthony Herrea

#Lab 1

#Olac Fuentes

#Anindita Nath

#2/8/19

#The purpose of this program is to understand recursion better

import numpy as np

import matplotlib.pyplot as plt

import math

def draw\_squares(ax,n,p,w): #This is the basic draw square method provided to us in the lab

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)

def draw\_squares2(ax,p,n,li,r,loop): #this method will create part one of the lab

if n < len(p): #It will create squerse of all the corners of the starting square and all the ones after that

ax.plot(p[:, 0], p[:, 1], color='k') #it takes a shape, a double array that contains all points, number of repetitions, a list, a modifire r, and the number of loops

xx = p[n,0] #what it does is that after ploting the first square it stores all corners x and y to be use as the next senter of the square.

yy = p[n,1]

q = np.array([[xx-r,yy-r],[xx+r,yy-r],[xx+r,yy+r],[xx-r,yy+r],[xx-r,yy-r]])

if loop > 0:

li.append(q)

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

draw\_squares2(ax,p,n+1,li,r,loop)

if len(li)>0:

x = li.pop(0)

draw\_squares2(ax, x,0, li, r\*.25,loop-1)

def circle(center, rad): #this method creates all the points for a cirlce by using the center and the 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

def draw\_circles2(ax, n, center, radius,w): #This is a method that will creat part 3 of the lab

if n > 0: #it will create circles that run along the x axis

x, y = circle([center[0]+radius,center[1]], radius) #it takes a shape, number of repetitions, first center, first radius , and a modifire w

ax.plot(x, y, color='k',linewidth=1)

draw\_circles2(ax, n - 1,center, radius\*w,w)

def draw\_circles(ax, n, center, radius, w): #This is the basic method to draw a circle and change the radius

if n > 0:

x, y = circle(center, radius)

ax.plot(x, y, color='k',linewidth=1)

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

def draw(ax,n,center,radius,w): #This method will create part 4 of the lab

if n > 0: #it creates the centers for all the circles

left = [center[0]-radius,center[1]] #Then it calls the method draw\_circles that will plot them.

right = [center[0]+radius,center[1]] # it takes a shape,number of repetition,the first origin, first radius, and a modifier w

top = [center[0],center[1]+radius]

bottom = top = [center[0],center[1]-radius]

mid = top = [center[0],center[1]]

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

draw\_circles(ax, n-1, left, radius\*w, w)

draw\_circles(ax, n-1, right, radius\*w, w)

draw\_circles(ax, n-1, top, radius\*w, w)

draw\_circles(ax, n-1, bottom, radius\*w, w)

draw\_circles(ax, n-1, mid, radius\*w, w)

#This shape will be part 1 A

list = []

plt.close("all")

orig\_size = 1000

r = orig\_size\*.25

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

fig, ax = plt.subplots()

draw\_squares2(ax,p,0,list,r,0)

ax.set\_aspect(1.0)

# ax.axis('off')

plt.show()

fig.savefig('squares1.png')

#This shape will be part 1 B and C

list = []

plt.close("all")

orig\_size = 1000

r = orig\_size\*.25

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

fig, ax = plt.subplots()

draw\_squares2(ax,p,0,list,r,3)

ax.set\_aspect(1.0)

# ax.axis('off')

plt.show()

fig.savefig('squares2.png')

#this shape will be part 2 A

plt.close("all")

fig, ax = plt.subplots()

draw\_circles2(ax, 7, [100, 0], 50, .5)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles1.png')

#this shape will be part 2 B

plt.close("all")

fig, ax = plt.subplots()

draw\_circles2(ax, 100, [100, 0], 50, .9)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles2.png')

#this shape will be part 2 C

plt.close("all")

fig, ax = plt.subplots()

draw\_circles2(ax, 100, [100, 0], 50, .95)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles3.png')

#This part will be part 4 A-C

plt.close("all")

fig, ax = plt.subplots()

draw(ax, 100, [100, 0], 50, 1/3)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles4.png')