Lab 1: Drawing Figures Using Recursion

Manuel Ruvalcaba

CS2302 1:30-2:50

1. Introduction

There are many different ways to draw figures in Python. In this lab, we are trying to draw four different types of figures using recursion. One of these being a square in which corner squares would be drawn recursively in each level. Others being a circle in which other circles with smaller radii would be drawn recursively, a tree that would gain two children on every child node in each recursive call, and another circle that would gain five circles inside each circle with recursive calls.

2. Proposed Solution Design and Implementation

To solve the problem with the square, we had a single recursive method. This method, *draw_cornerSquares*, would have parameters for a subplot, *ax*, an integer, *n*, for the number of levels for recursion, an array, *p*, which contains the coordinates for each corner of the square, and a number, *w*, used as a ratio to make the new corner squares smaller. Inside the method, as long as n was greater than zero, we have a variable, *q*, which initializes an array the same length as *p*, to be the new square. Another variable, *k*, which finds the distance between two adjacent corners on the square *p*, and a variable, *d*, which assigns a negative or positive *k* value to positions in the array. Before each of the four recursive calls, array *q* is assigned the value of the sum of array *d* and different points in array *p*. The recursive calls then pass the new array *q* and decrease *n* by one.

To solve the problem with recursively drawing circles of smaller radii inside of a starting circle, we would need a single recursive method. This method, *draw_circles*, would have parameters for a subplot, *ax*, an integer, *n*, for the number of levels for recursion, a variable, *center*, which would receive a point for a center, a variable, *radius*, that would hold a value for the radius, and a variable, *w*, used as a ratio to make the new circle smaller. As long as n is greater than zero, variable *x,y* would create a circle with a specific center and radius by calling a method called *circle*. The circle would then be plotted and make the recursive call with a new center that has the x-coordinate at the position of *radius*w* and the new radius, *radius*w*.

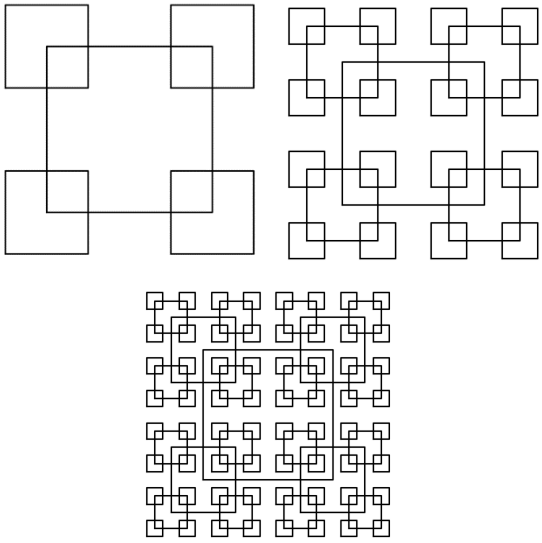
To solve the problem with the tree, we would need a single recursive method. This method, *draw tree*, would have parameters for a subplot, *ax*, an integer, *n*, for the number of levels for recursion, an array, *p*, which contains the coordinates for the parent, left child, and right child. There is also a variable, *w*, which is the ratio to make the new subtrees. Inside the method, I have an array, *q*, with the same length as *p* to hold the points for the new subtree. I have a variable, *dx*, which holds a fourth the distance between the children and a variable, *dy*, which holds the difference in the y-coordinate between the child and parent. There is also an array, *d*, that holds the points to be added to *p*. Before each of the two recursive calls, that will create the left and right children, array, *q*, is assigned the value of the sum of array *d* and the left and right child in array *p*. The recursive calls then pass the new array *q* and decrease *n* by one.

Finally, to solve the problem with recursively drawing five circles in each circle, I used a single recursive method. This method, *draw_circles2*, would have parameters for a subplot, *ax*, an integer, *n*, for the number of levels for recursion, a variable, *center*, which would receive a point for a center, a variable, *radius*, that would hold a value for the radius, and a variable, *w*, used as a ratio to make the new circle smaller. As long as n is greater than zero, variable *x,y* would create a

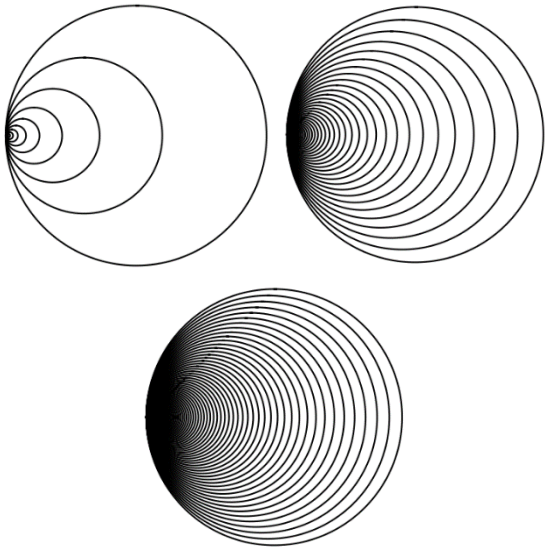
circle with a specific center and radius by calling a method called *circle*. The circle would then be plotted and make five recursive calls that would create the five new circles, one with the same *center* and four with a different *center*, all of which had a new *radius* of *radius***w*.

3. Experimental Results

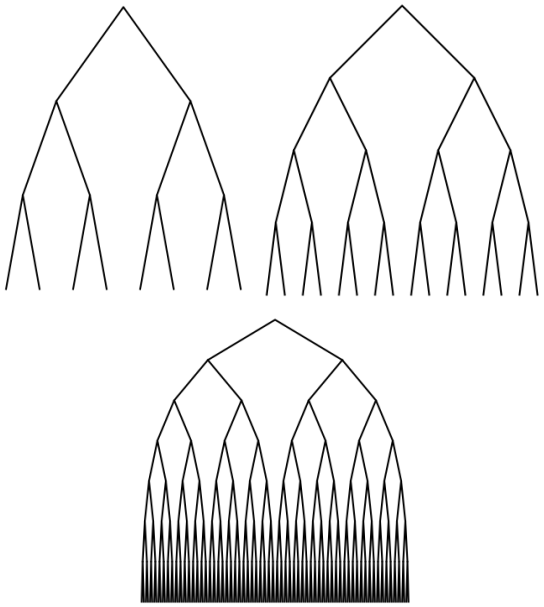
To test the method with the squares, I made three calls to the *draw_cornerSquares* method with a subplot *ax*, different values of *n*, which were 2, 3, and 4, an array, *p*, which contained points for a square, and 1/2 as *w* to draw and save the following three figures, which were my results. I used the *timeit* library to find the runtime of each initial call to the method. The runtimes, in seconds, for the three calls were 0.021, 0.102, and 0.346.



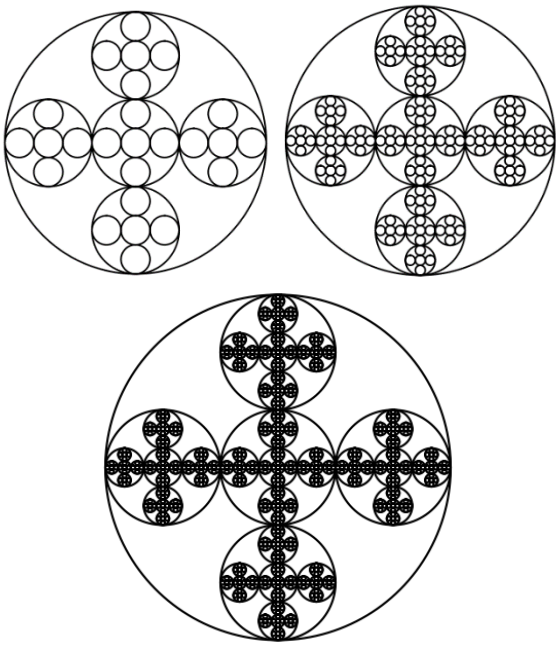
To test the method with the circles of recursively smaller radii, I made three calls to the *draw_circles* method with a subplot *ax*, different values of *n*, which were 25, 50, and 100, a center of [100,0], a radius of 100, and different values of *w*, which were .6, .9, and .95. These parameters were used to draw and save the following three figures, which were my results. I used the *timeit* library to find the runtime of each initial call to the method. The runtimes, in seconds, for the three calls were 0.114, 0.226, and 0.424.



To test the method that would make subtrees on each child in the main tree, I made three calls to the *draw_tree* method with a subplot *ax*, different values of *n*, which were 3, 4, and 7, an array *p* that contained points for a tree, and a w of 1/4. I also modified the aspect ratio of the axis in *ax.set_aspect* to draw and save the following figures, which were my results. I used the *timeit* library to find the runtime of each initial call to the method. The runtimes, in seconds, for the three calls were 0.030, 0.070, and 0.521.



To test the method for drawing the five circles in each circle, I made three calls to the *draw_circles* method. I passed a subplot *ax*, different values of *n*, which were 3, 4, and 5, a center of [0,0], a radius of 100, and a value of 1/3 for *w* to draw and save the following figures, which were my results. I used the *timeit* library to find the runtime of each initial call to the method. The runtimes, in seconds, for the three calls were 0.177, 0.664, and 3.344.



4. Conclusion

With this project, I have learned to use recursion to draw various types of images by using the plotting library in Python. I also learned how to modify points in an array to create the points for new shapes.

5. Appendix

```
"""
Course: CS2302
Author: Manuel Ruvalcaba
Assignment: Lab 1
Instructor: Dr. Olac Fuentes
TA: Anindita Nath, Maliheh Zargaran
Date of last modification: 02/08/19
Purpose of program: The purpose of this program is to draw and save various
shapes through recursion.
"""

import matplotlib.pyplot as plt
import numpy as np
import math

def circle(center,rad): #makes the circle
    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): #plots the points for the circles in the
second part of the lab
    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)

plt.close("all")
fig, ax = plt.subplots()
```

```
draw_circles(ax, 25, [100,0], 100,.6)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles_1.png')

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_2.png')

plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 100, [100,0], 100,.95)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles_3.png')

def draw_circles2(ax,n,center,radius,w): #plots the circles for the fourth part
of the lab
    if n>0:
        x,y = circle(center,radius)
        ax.plot(x,y,color='k')
        draw_circles2(ax,n-1,center,radius*w,w)
        draw_circles2(ax,n-1,[center[0],center[1]-(2*radius*w)],radius*w,w)
        draw_circles2(ax,n-1,[center[0]+(2*radius*w),center[1]],radius*w,w)
        draw_circles2(ax,n-1,[center[0]-(2*radius*w),center[1]],radius*w,w)
        draw_circles2(ax,n-1,[center[0],center[1]+(2*radius*w)],radius*w,w)
```

```
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 3, [0,0], 100, 1/3)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles2_1.png')


plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 4, [0,0], 100, 1/3)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles2_2.png')


plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 5, [0,0], 100, 1/3)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('circles2_3.png')


def draw_cornerSquares(ax,n,p,w): #plots the squares for the first part of the
lab
    if n>0:
        q = np.zeros((5,2)) #array with points for the corners of the corner
        square
        k = (p[3,0]-p[0,0])/4 # 1/4 the distance from one corner to another
        d = np.array([[-k,-k],[-k,k],[k,k],[k,-k],[-k,-k]]) #array to edit new
        points
```

```
        ax.plot(p[:,0],p[:,1],color='k')
        q[:,0] = d[:,0]+p[0,0] #adds distance array to x values in p array
        q[:,1] = d[:,1]+p[0,1] #adds distance array to y values in p array
        draw_cornerSquares(ax,n-1,q,w)
        q[:,0] = d[:,0]+p[1,0]
        q[:,1] = d[:,1]+p[1,1]
        draw_cornerSquares(ax,n-1,q,w)
        q[:,0] = d[:,0]+p[2,0]
        q[:,1] = d[:,1]+p[2,1]
        draw_cornerSquares(ax,n-1,q,w)
        q[:,0] = d[:,0]+p[3,0]
        q[:,1] = d[:,1]+p[3,1]
        draw_cornerSquares(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_cornerSquares(ax,2,p,(1/2))
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('squares.png')

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_cornerSquares(ax,3,p,(1/2))
ax.set_aspect(1.0)
ax.axis('off')
```

```
plt.show()
fig.savefig('squares_2.png')


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_cornerSquares(ax,4,p,(1/2))
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('squares_3.png')


def draw_tree(ax,n,p,w): #plots the trees for the third part of the lab
    if n>0:
        q = np.zeros((3,2)) #array with point for the new tree
        dx = (p[0,0]-p[2,0])/4 #difference in x between children nodes
        dy = (p[1,1]-p[0,1]) #difference in y between children and parent
        d = np.array([[-dx,-dy],[0,0],[dx,-dy]])
        ax.plot(p[:,0],p[:,1],color='k')
        q[:,0] = d[:,0]+p[0,0] #adds distance array to x values in p array in
        left child
        q[:,1] = d[:,1]+p[0,1] #adds distance array to y values in p array in
        left child
        draw_tree(ax,n-1,q,w)
        q[:,0] = d[:,0]+p[2,0] #adds distance array to x values in p array in
        right child
        q[:,1] = d[:,1]+p[2,1] #adds distance array to y values in p array in
        right child
        draw_tree(ax,n-1,q,w)


plt.close("all")
orig_size = 400
```

```
p = np.array([[-orig_size,-orig_size],[0,orig_size],[orig_size,-orig_size]])
fig, ax = plt.subplots()
draw_tree(ax,3,p,3)
ax.set_aspect(.7)
ax.axis('off')
plt.show()
fig.savefig('tree.png')

plt.close("all")
orig_size = 400
p = np.array([[-orig_size,-orig_size],[0,orig_size],[orig_size,-orig_size]])
fig, ax = plt.subplots()
draw_tree(ax,4,p,1/4)
ax.set_aspect(.5)
ax.axis('off')
plt.show()
fig.savefig('tree_2.png')

plt.close("all")
orig_size = 400
p = np.array([[-orig_size,-orig_size],[0,orig_size],[orig_size,-orig_size]])
fig, ax = plt.subplots()
draw_tree(ax,7,p,1/4)
ax.set_aspect(.3)
ax.axis('off')
plt.show()
fig.savefig('tree_3.png')
```