Marco Valerio

CS 2302

Olac Fuentues

Lab Report #1

02/12/2019

Introduction

Lab 1 was intended to test our knowledge of recursion, math plots, and array slicing. We were given the task to draw 4 different figures using the given codes “draw\_circles.py” and “draw\_squares.py”. We had to modify the parameters and the math that determined the plotted shapes in order to get the desired outcome.

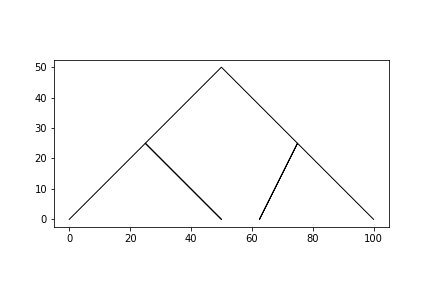
Proposed Solution design and Implementation:

I attempted to solve the square and binary tree designs by modifying the given code for “draw\_sqaures.py”. I was not able to properly print out the squares in the corners, but I was able to shrink down a single square multiple time. I did this by altering the function for q and w (the scale), instead of (q = p\*w + p[i1]-(1-w)) it became (q=p\*w) and I disregarded i1. In doing so, this stopped the squares from shifting and creating a spiral effect. For the binary tree, I modified the plot array with new point to create the origin and two leaves and I called it “t”. In the “binary\_tree” method I altered “t” by dividing by two every call, this output was like a binary tree but a little wider.

For the two circle designs I modified “draw\_circles.py” in attempt to get the desired outcome for the circles. Not much was needed to be changed in order to achieve the circle which get shifted to the left while it shrinks. All I did to alter the original code was add the radius to the x coordinate in ax.plot, so it would shift to the left every call. The second circle design was difficult to create, and I didn’t exactly get the shape correctly. I did multiple alterations to ax.plot changing the x and y coordinates by adding and subtracting the radius. The output was similar to the expected result, but it included extra circles surrounding the main circle.

Experimental Results:

While I was testing each method, I changed the number of times they would call themselves recursively. For the two square method calls and the binary tree call, I changed when testing to see if the first printed shape would output the original shape and with some recursive calls. This meant changing the value of “n” to a small integer between the range of 1-5, and adjusting “w” to get the right alteration for the shapes, making the number smaller (.5) to see any difference.

.  A screenshot of a cell phone

Description automatically generated

Testing the circles was like the squares and binary tree, the value of “n” was altered in a way so I could see if the recursive calls would alter the original shape in the correct way. Everything was practically the same except I changed the radius of the 2nd circle to half of the original which was 100.

A close up of a logo

Description automatically generated A screenshot of a cell phone

Description automatically generated

Conclusions:

In conclusion, I gained experience with math plot and the numpy tools. This also gave me some more practice with recursion in python. This is my first-time using recursion and arrays in python, and I learned quite a lot about the implementation of these data structures and method calls. Switching from java to python is quite difficult in my opinion, I understand the algorithms and logic are the same, but I keep finding myself trying to revert back to java.

Appendix:

Source Code:

import math

import numpy as np

import matplotlib.pyplot as plt

def draw\_squares(ax,n,p,w):

if n>0:

q = p\*w

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

draw\_squares(ax,n-1,q,w/2)

plt.close("all")

orig\_size = 800

q1\_size = 1000

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

fig, ax = plt.subplots()

draw\_squares(ax,6,p,.5)

ax.set\_aspect(1.0)

ax.axis('on')

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\_squares(ax,6,p,.7)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('squares1.png')

def binary\_tree(ax,t,n):

if n>0:

t1 = t/2

ax.plot(t[:,0],t[:,1],Linewidth=1,color='k')

binary\_tree(ax,t1,n-1)

plt.close("all")

orig\_size = 50

t = np.array([[0,0],[orig\_size\*.5,orig\_size\*.5],[orig\_size,0],[orig\_size\*.5,orig\_size\*.5],[orig\_size,orig\_size],[orig\_size\*1.5,orig\_size\*.5],[orig\_size\*1.25,0],[orig\_size\*1.5,orig\_size\*.5],[orig\_size\*2,0]])

fig, ax = plt.subplots()

binary\_tree(ax,t,1)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('binary1.png')

plt.close("all")

orig\_size = 50

t = np.array([[0,0],[orig\_size\*.5,orig\_size\*.5],[orig\_size,0],[orig\_size\*.5,orig\_size\*.5],[orig\_size,orig\_size],[orig\_size\*1.5,orig\_size\*.5],[orig\_size\*1.25,0],[orig\_size\*1.5,orig\_size\*.5],[orig\_size\*2,0]])

fig, ax = plt.subplots()

binary\_tree(ax,t,10)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('binary2.png')

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+radius,y,color='k')

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

plt.close("all")

fig, ax = plt.subplots()

draw\_circles(ax, 5, [100,0], 100,.9)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('circles\_single.png')

plt.close("all")

fig, ax = plt.subplots()

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

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('circles.png')

def circle1(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\_circles1(ax,n,center,radius,w):

if n>0:

x,y = circle1(center,radius/3)

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

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

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

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

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

draw\_circles1(ax,n-1,center,radius\*w,w)

plt.close("all")

fig, ax = plt.subplots()

draw\_circles1(ax, 2, [100,0], 50,.3)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('circles1\_single.png')

plt.close("all")

fig, ax = plt.subplots()

draw\_circles1(ax, 5, [100,0], 50,.3)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('circles1.png')