**Course:** CS 2302

**Author:** Kimberly Morales

**Assignment:** Lab 1

**Instructor:** Olac Fuentes

**TA(s):** Anindita Nath, Maliheh Zargaran

**Date:** 2/8/2019

# **Introduction**

The problem that I was aiming to solve was to create geometric figures composed of smaller modules of a similar shape by utilizing recursion. In terms of breaking down the larger problem into smaller ones, four large shapes were to be composed by creating smaller modules and then use them recursively to create a larger shape. The first shape was of nested squares that recursively drew themselves at each convex corner of the square while being half the length of the predecessor square. The second shapes were of a concentric circle composed of circles of a radius is proportional to the predecessor circle while also have a center to the rightmost part of the circle. The third shape was of a tree with a constant height. The final shape was of a lotus-like circle with circles being within a larger one.

# **Proposed Solution**

My first approach to the overall problem was to first analyze the shapes that were to be composed and understanding the mathematical relationship between each success level. This required a hands-on approach of me drawing the shape, plotting coordinates on graph paper, and seeing the changes by tracing pseudocode. After having a rough guideline, I went to observe the code of the circle and square programs. By tinkering with the code, I wrote down notes to see how each variable changed the overall shape in each run of the program. With these notes, I then proceeded to focus on each shape separately and proceeded to break down the problems more specifically.

## **Solution Design**

For the nested squares, I realized from early on that the squares of the next level were always half of the length of the predecessor square. Another observation was that each of these successor squares was using the corners of the predecessor square as a center point. I decided that to draw the overall shape, I would need to have both my x and y coordinates to be half in each recursive call. However, there would at least need to be four recursive calls for the four corners of the square. Taking into consideration of a cartesian plane, all four recursive calls needed to be in different planes which mean different sign values for the recursive calls. Next, I needed a constant variable to shift my squares into the convex of the predecessor square since it needed to be pointing out rather than recursively drawing squares within the predecessor square. I broke down this shape into only one method (excluding the draw squares method) since it was simple to understand that the square needed to create smaller squares into each plane of a cartesian plane.

The second shape, the concentric circles, required geometric knowledge of circles and to observe the radius and center relationship. This shape seemed simple since it was merely within one bigger circle with a center shifted all the way to the rightmost part of the circle. The tricky aspect was of how big the ratio of the radius and center was. In the end, I found that 0.9 was a good ratio since it would not drastically change the area of each successive circle since 0.5 and 0.3 would often limit the circles in each recursive call. I broke down this shape into two methods (excluding the circle's method) since I found my original code to be unreadable, so I separated it to make it more readable and manageable. The first method, draw circles, oversaw merely creating a circle with a ratio of 0.9. This first method would be called once in the second method, draw con circles, which it then shifted the center coordinates by shifting the x coordinates by adding the radius and the shifter variable, d. D is just the original radius size for the original circle.

Next shape was the tree, which required a new method since it only needed 2 lines which used three points. I reused parts of the draw squares method and made 0 to be the origin coordinate. The other two coordinates were of similar magnitude but with opposite signs of the x coordinate. Another observation made was that the line length of the tree in the lab 1 instructions was constant and did not change which meant a constant tree height. The only changes being made was that the x coordinates were half of the predecessor's coordinates and the y coordinates being related to the height. Each new smaller tree also reused the endpoints except for the center as a new center for the smaller tree. I only used one method for this shape since I saw a similar pattern like the nested squares where I realized that the coordinates were mirroring each other except only with the x coordinates. This meant that I only needed to use two methods since the only points that I needed to change in each recursive call were the x coordinates.

Finally, the lotus shape was the most intricate but was constructed similarly to the above shapes. This shape required 5 circles within a larger circle which mean that the radius relation to the center was at least a third since the circles were not overlapping. There was also a relationship with the circles with the cartesian plane since the horizontal circles mirrored each other along with the vertical circles. I thought of each smaller shape in arbitrary positions: center, right, left, up, and down. This made it important since it helped me thought of the signs for each coordinate.

For the overall program, I wanted a menu that generated shapes as many times that I wanted along with a specified recursion level. I also made sure to have a try-catch error code snippet in order to prevent exceptions from odd or unusual input. This menu also allowed for an exit button since it allows the user a friendlier option of exiting the program rather than closing the prompt. To organize each generated file of the shape, I put the name of each shape that I used for the menu and assigned it the recursive level that was used to generate it. This allows the program to save multiple images while also specifying the levels it used.

# **Experimental Results**

To test my program required messing with the input of the number of levels for each shape. The results than expected for all shapes to exhibit was 0 to be empty figures and 1 being the original base shape of the larger shapes. I also needed to replicate the shapes from the original lab instructions. This required breaking down the results into each shape individually with the sections A, B, C, being the shapes generated from the lab instructions.

The inputs were of choice and level. The choice variable is the menu number for each shape and is an integer. The level variable is the number of recursive calls and is also an integer.

Base Case:

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = 1 |  |
| Concentric Circles | choice = 2, level = 1 |  |
| Tree | choice = 3, level = 1 |  |
| Lotus | choice = 4, level = 1 |  |

Drawings A:

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = 2  30 seconds |  |
| Concentric Circles | choice = 2, level = 9  30 seconds |  |
| Tree | choice = 3, level = 3  Under a minute |  |
| Lotus | choice = 4, level = 3  Under a minute |  |

Drawings B:

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = 3  Under a minute |  |
| Concentric Circles | choice = 2, level = 20  Under a minute |  |
| Tree | choice = 3, level = 4  Under a minute |  |
| Lotus | choice = 4, level = 4  Under a minute |  |

Drawings C:

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = 4  Under a minute |  |
| Concentric Circles | choice = 2, level = 50  Under a minute |  |
| Tree | choice = 3, level = 8  Under a minute |  |
| Lotus | choice = 4, level = 5  Under a minute |  |

Odd/ Error Input

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = -5 | NO IMAGE |
| Concentric Circles | choice = 2, level = -5 | NO IMAGE |
| Tree | choice = 3, level = -5 | NO IMAGE |
| Lotus | choice = 4, level = -5 | NO IMAGE |

Upper Bound

|  |  |  |
| --- | --- | --- |
| Shape Name | Input | Output |
| Nested Squares | choice = 1, level = 99  Froze | NO IMAGE |
| Concentric Circles | choice = 2, level = 99  Under a minute |  |
| Tree | choice = 3, level = 99  Froze | NO IMAGE |
| Lotus | choice = 4, level = 5  Froze | NO IMAGE |

# **Conclusions**

Overall, I learned a lot about matplotlib, python’s many implicit features, and the nature of paper programming. If I had started this lab without pseudocode done on paper, then I wouldn't have broken down the program into a modular way and make a concise algorithm for each figure. I realized the importance of a good foundation. On matplotlib, while I have used this library before for a research project, I never thought of utilizing it with recursion and being able to create interesting and intricate figures. I also saw the relationship between arrays and graphs since it never occurred to me that a graph is merely a collection of coordinates like an array would hold for a collection of integers. This is like how one would work out and connect the mind and muscle memory, except the connections being made, were of taking a geometric problem and using computer science to solve the problem. I also saw how recursion can occur in the real world and real-life structure. Recursion is like how humans think of solving problems by repeating steps and stacking results until an answer is finally formed.

# **Appendix**

"""

Course: CS 2302 [MW 1:30-2:50]

Author: Kimberly Morales

Assignment: Lab 1

Instructor: Olac Fuentes

TA(s): Anindita Nath , Maliheh Zargaran

Date: 2/8/2019

Date of last modification: 2/8/2019

Purpose of program:

To generate fractal like shapes such as circles, squares,and trees with recursive algorithms.

This means to break down the problem into smaller pieces and build it to a bigger pictures.

For this case, breaking down the overall larger shape and utilize basic geometry to efficently produce these images.

"""

import numpy as np

import matplotlib.pyplot as plt

import math

#############################################################################################################################

#FIGURE 1: Nested Squares

#############################################################################################################################

def draw\_nest\_squares(ax,n,x,y,d):

if n > 0:

#Continously plots the x and y coordinates for each recursive call

ax.plot(x[:]-d,y[:]-d, color='k')

#Four recursive calls to create squares that have half the length of their predecessor and their vertexes

#For x coordinates: Each new square is half of the predecessor and is +- with d

#For y coordinates: Each new square is hald of the predecessor and is +- with d

draw\_nest\_squares(ax,n-1,x/2+d, y/2+d, d) #Upper Right

draw\_nest\_squares(ax,n-1,x/2+d, y/2-d, d) #Lower Right

draw\_nest\_squares(ax,n-1,x/2-d, y/2+d, d) #Upper Left

draw\_nest\_squares(ax,n-1,x/2-d, y/2-d, d) #Lower Left

def deploy\_nest\_squares(level):

orig\_size = 800

fig, ax = plt.subplots()

#Seperates the 2d array into x and y components of size 800 and for the four points of the square(the fifth is for the cycle)

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

px = p[:,0]

py = p[:,1]

draw\_nest\_squares(ax,level,px,py,orig\_size)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('nest-squares' + str(level) + '.png')

plt.close("all")

#############################################################################################################################

#FIGURE 2: Concentric Circles

#############################################################################################################################

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:

#Plots circles based on w ratio and center point

x,y = circle(center,radius)

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

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

def draw\_con\_circles(ax,n, center, radius,d):

if n > 0:

#d: Shifter variable and shares similar value to the radius except that it stays constant in the recursive calls

#Draws one circle from draw\_circles and recursively calls con-circles to shift to the left with d

draw\_circles(ax,1,[radius,0],radius,.9)

draw\_con\_circles(ax,n-1, [radius+d, 0],radius+d,d)

def deploy\_con\_circles(level):

plt.close("all")

fig, ax = plt.subplots()

draw\_con\_circles(ax,level, [100, 0], 100,100)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('con-circles' + str(level) + '.png')

#############################################################################################################################

#FIGURE 3: Tree

#############################################################################################################################

def draw\_trees(ax,n,x,y,d,h):

if n > 0:

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

#d: Increment value that shifts the x values with the value of the line length

#h: Height of the tree, shifts the y coordinates consistently

#Two recursive calls that plots on opposite sides of the x axis with each x coordinate being half of their predecessor

draw\_trees(ax,n-1, (x/2)-d, y-(h),d, h)

draw\_trees(ax,n-1, (x/2)+d, y-(h),d, h)

def deploy\_trees(level):

#line\_length: Same line length from the origin to the end points

line\_length = 50

#p: 2D array composed of three lines with three coordinates. The origin is index 1 with the two endpoints being mirrored

p = np.array([[-line\_length, -line\_length],[0, 0], [line\_length, -line\_length]])

px = p[:,0]

py = p[:,1]

plt.close("all")

fig, ax = plt.subplots()

draw\_trees(ax,level,px,py,line\_length,50)

ax.set\_aspect('auto')

ax.axis('off')

plt.show()

fig.savefig('trees' + str(level) + '.png')

#############################################################################################################################

#FIGURE 4: Lotus

#############################################################################################################################

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

if n>0:

x,y = circle(center,radius)

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

#Recursively calls five circles that are plotted in these positions: Center, right, left, up, down

#For x circles (left,right): Plots the x coordinate that +- to the radius/3

#For y circles (up,down): Similar to x circles except it is done with the y coordinates

draw\_lotus(ax,n-1, center,radius\*w, w) #Center

draw\_lotus(ax,n-1, [center[0]+2\*(radius/3), center[1]],radius\*w, w) #Right

draw\_lotus(ax,n-1, [center[0]-2\*(radius/3), center[1]],radius\*w, w) #Left

draw\_lotus(ax,n-1, [center[0], center[1]+2\*(radius/3)],radius\*w, w) #Up

draw\_lotus(ax,n-1, [center[0], center[1]-2\*(radius/3)],radius\*w, w) #Down

def deploy\_lotus(level):

plt.close("all")

fig, ax = plt.subplots()

draw\_lotus(ax,level,[0, 0],100,1/3)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('lotus-circles' + str(level) + '.png')

#############################################################################################################################

#MAIN INPUT/OUTPUT: Holds menu for different figures to draw

#############################################################################################################################

try:

choice = 0

while choice != 5:

print("Enter the number of which shape you want: \n"

+ "1: Nested Squares\n"

+ "2: Concentric Circles\n"

+ "3: Tree\n"

+ "4: Lotus\n"

+ "5: EXIT\n")

choice = int(input())

if choice != 5:

level = int(input("Enter the number of levels \n"))

if choice == 1:

deploy\_nest\_squares(level)

if choice == 2:

deploy\_con\_circles(level)

if choice == 3:

deploy\_trees(level)

if choice == 4:

deploy\_lotus(level)

except ValueError:

print("ERROR: Invalid input")

# **Academic Honesty**

“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 inappropriate assistance to any student in the class.”

Name: Kimberly Morales

Signature: