Samuel Chong

2/12/19

CS 2302

Lab 1

MW 10:30-11:50

Olac Fuentes

Introduction:

The problem that I tried to solve for this lab was trying to create figures and make sub-figures by calling the method with recursion. The first figure was a square with sub-squares in each corner. The second figure was a circle and the sub-circles must be coming from the left of the circle, not the center. The third one was a binary tree. Lastly, the last figure was a circle that had 5 sub-circles inside of it, and inside the small circles, they had even smaller circles inside of them.

Proposed solution design and implementation:

To try and figure out how to solve these problems, I started playing around with the codes provided by the professor (circle and square). After doing some tests, I noticed that in the circle code, if you add radius to x or y, the sub-circles moved positions. So, when I added radius to x in the line where it draws the figure, I accidentally managed to solve a problem the lab required us to do. That was one figure I had completed. The hard part was trying to get the correct number of circles each figure needed to get the same figure as the image. So, I just kept guessing until getting correct. The code provided by the professor was the key in order to make the figures

After doing the circles I started to do the squares figure because of the code provided by the professor. After learning how the code work, I noticed the line “p = np.array” is where you plug in the coordinates in order to create a complete square. If you were missing a coordinate, then it would not create a complete circle or even it could create a triangle instead. In order to create a square, we needed different coordinates so it would make the square, so I subtracted and added the radius in different order to create the square figure, the first and last had to be the same or it would not create a square. After having the coordinates, I made a recursive call but noticed it only made one, so I realized that I need 4 recursive calls, each of them different by, again, adding and subtracting the radius. The professor told us the sub-squares depended on r so that is where I got the idea that I needed to add and subtract the radius.

For the binary tree it was a hard one because it was the first code, we had to make without an example provided by the professor. I knew it was somehow related to the square one because the binary tree is just drawing straight lines, same as the square. So, I then noticed we needed to use coordinates to draw the lines, but I kept getting straight lines and not tilted like the binary tree. I couldn’t find the answer to it until the professor told us in class, we needed to subtract y in the coordinates in order it goes down one level, and to make the lines go left and right we needed to add and subtract x

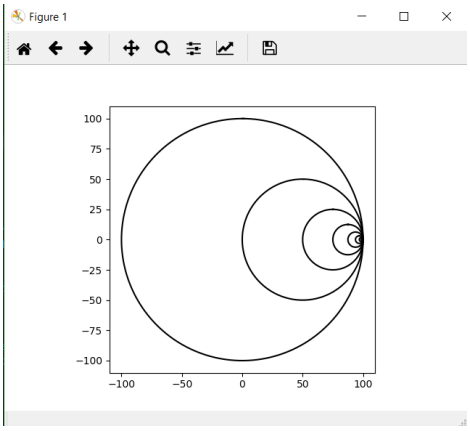
to the coordinates. The same applied to the binary tree as the squares, it needed more then one recursion because it only kept making one line instead of the both sides.

The 5 circles was toughest of all. I could find a way solve it. I approached the same as the other circle figure. I started added and subtracting the radius to x and y, and noticed the circles were moving into place, but when I had for example x+radius in one line and x-radius in the next line, it would create very strange figures. Instead of making sub-circles inside the circle, it would make circles the size of the original but next to the original circle. I was not able to find a solution in time, and I also needed help, so I went to the Tas office hours for help and she helped me with my problem. I needed to use coordinates when calling the circle method. My circles were not the right size as well and in the office hours I learned the radius needed to be divided by 3 instead of 2. When I kept working on my problems, I noticed my inner circles where not in the right place, there was no space for the center circle, so I knew something with the radius was off. I started moving the division and so on and I was able to position them correctly by multiplying by 2 the radius divided by 3, that got the circles in order.

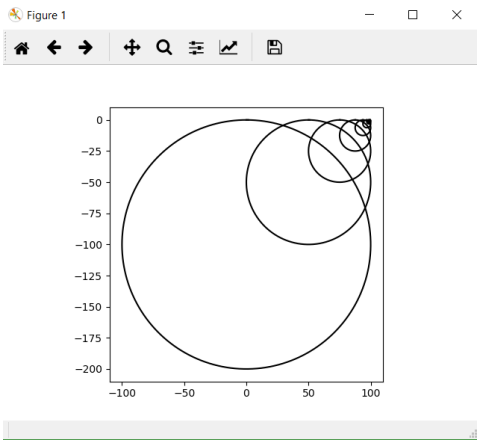
Experiments:

Circle experiments

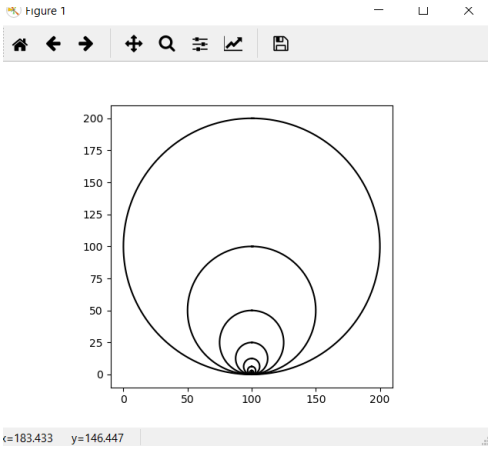
When subtracting radius to x



When subtracting radius to x and y

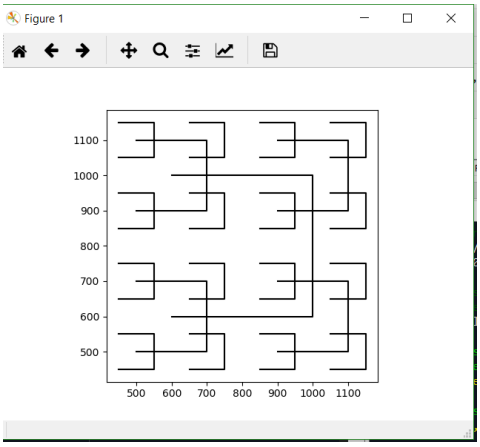


When adding radius to y

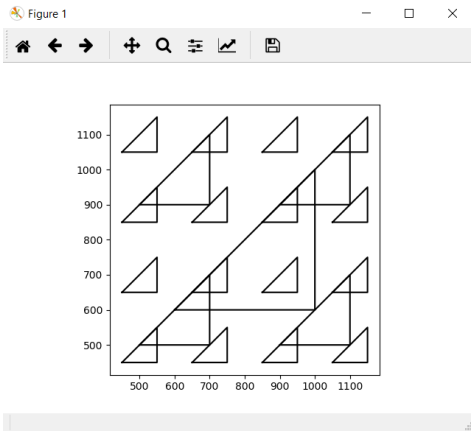


Square experiments

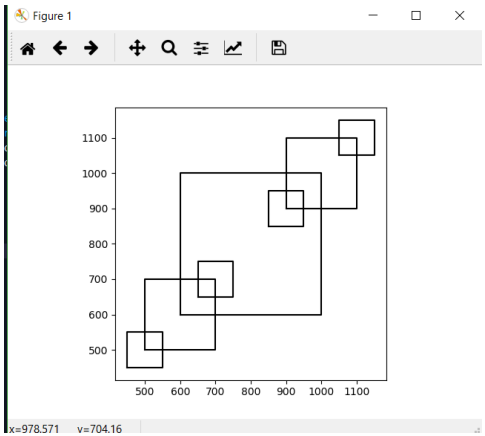
Not having the 5th coordinate that completes the complete square



Missing one coordinate that is not the first or last will cause a triangle

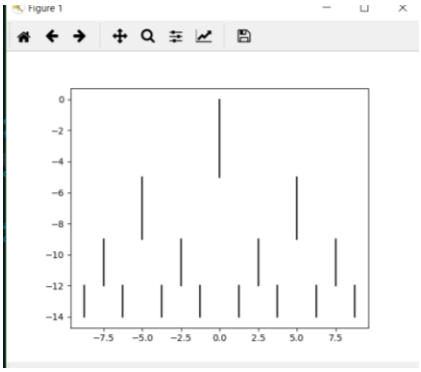


When only having 2 recursive calls instead of 4

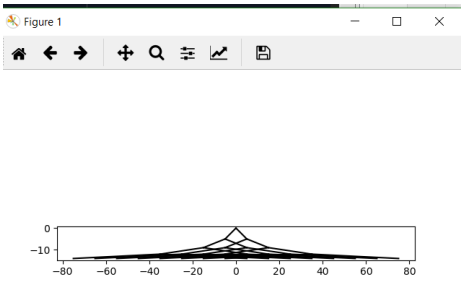


Binary tree experiments

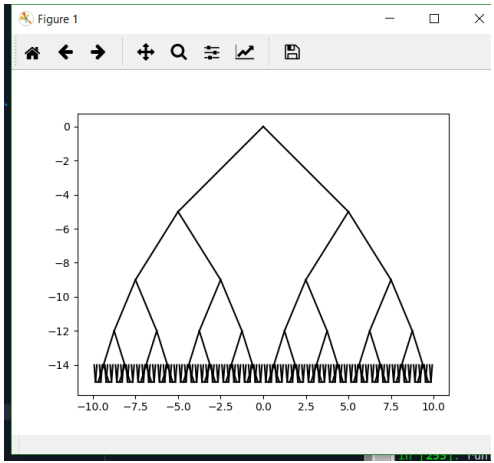
When I didn’t add or subtract x to the coordinates



When making the recursion call, I multiplied x times 2 instead of diving it

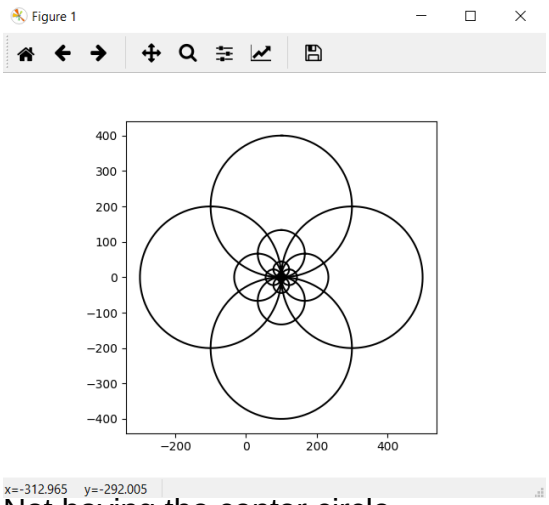


When I call the method 7 times and x and y are 5

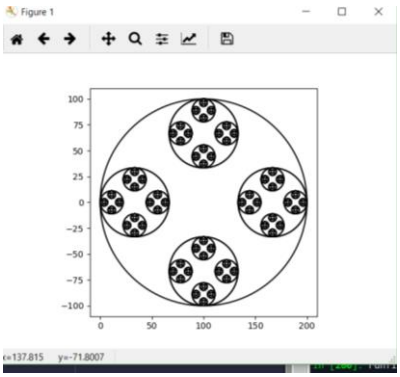


5 Circles experiments

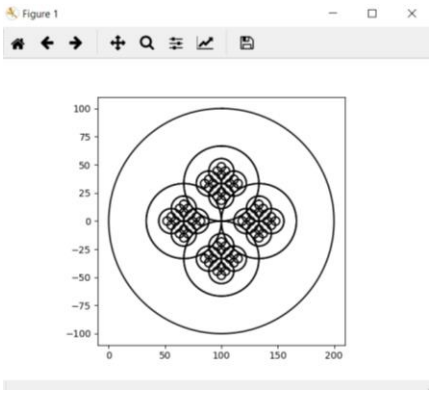
When adding and subtracting radius to x and y instead of being a coordinate



Not having the center circle



Not having the center circle and not multiplying times 2 the radius divided by 3



Running times for squares

```
The running time for 1a squares is:
0.01003885269165039

In [282]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 1b squares is:
0.03490734100341797

In [283]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 1c squares is:
0.10388422012329102
```

Running times for circles

```
The running time for 2a circles is:
0.022580623626708984

In [5]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 2b circles is:
0.07879185676574707

In [6]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 2c circles is:
0.1403360366821289

In [7]:
```

Running time for binary trees

```
The running time for 3a binary tree is:
0.027927875518798828

In [6]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 3b binary tree is:
0.05504775047302246

In [7]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 3c binary tree is:
0.4148902893066406
```

Running time for 5 circles

```
The running time for 4a 5circles is:
0.04089641571044922

In [24]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 4b 5circles is:
0.0479588508605957

In [25]: runfile('C:/Users/samya/Documents/CS 2302/lab1.py', wdir='C:/Users/samya/
Documents/CS 2302')
The running time for 4c 5circles is:
2.0534861087799072
```

Conclusion:

In this lab I was able to learn that it is possible to create figures using Python. Some figures are harder to make than the others and sometimes the figure does not come out correct simply because of the parameters. We always have to be careful where we add and subtract and most importantly how important the parameters are and its variables. Although it took me a long time to figure out how to create each figure I liked this kind of lab, I had never worked with creating figures and it is a new skill I learned thanks to this lab.

Appendix:

```
import numpy as np
import math
import matplotlib.pyplot as plt
"""
@author Samuel Chong
CS2302
Olac Fuentes
Lab 1
"""

def draw_squares(ax,n,coord,r):
    if n>0:
        #the function for p is to draw the complete square, if one coordinate
        #is missing then the square is incomplete or it draws a triangle
        p = np.array(((coord[0] - r,coord[1] - r),(coord[0] + r,coord[1] - r),
                      (coord[0] + r,coord[1] + r), (coord[0] - r,coord[1] + r),
                      (coord[0] - r,coord[1] - r)))
        ax.plot(p[:,0],p[:,1],color='k')
        #this recursion call creates the bottom left square
        draw_squares(ax,n-1,(coord[0] - r,coord[1] - r), r/2)
        #this recursion call creates the bottom right square
```

```
        draw_squares(ax,n-1,(coord[0] + r,coord[1] - r), r/2)
        #this recursion call creates the top left square
        draw_squares(ax,n-1,(coord[0] - r,coord[1] + r), r/2)
        #this recursion call creates the top right square
        draw_squares(ax,n-1,(coord[0] + r,coord[1] + r), r/2)

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)
        #in order for the circle to be moved to the left you need to
        #add radius to x
        ax.plot(x+radius,y,color='k')
        draw_circles(ax,n-1,center,radius*w,w)



def draw_tree(ax,n,orig,x,y):
    if n>0:
        #to draw a line to the left then we must subtract x to our original 'x'
        #coordinate and subtract y to out  original 'y' in order for the line
        #to go down or else it would be a straight line
        ax.plot((orig[0],orig[0]-x), (orig[1],orig[1]-y),color='k')
        #same applies for the right line but in this case you need to add x
        #so the line inclines the other way
```

```
        ax.plot((orig[0],orig[0]+x), (orig[1],orig[1]-y),color='k')
        #call the method twice so there is two children per level
        #divide x by 2 so when adding or subtracting x to coordinate it moves
        #left and right equally, subtract 1 to y so it goes down one level
        draw_tree(ax,n-1,((orig[0]-x,orig[1]-y)), x/2, y-1)
        draw_tree(ax,n-1,((orig[0]+x,orig[1]-y)), x/2, y-1)

def draw_in_circles(ax,n,center,rad):
    if n>0:

        #creates the original circle
        x,y = circle(center,rad)
        ax.plot(x,y,color='k')

        #creates center circle
        x,y = circle((center[0],center[1]),rad/3)
        ax.plot(x,y,color='k')
        draw_in_circles(ax,n-1,(center[0],center[1]),rad/3)

        #creates the left circle
        x,y = circle((center[0]-(rad/3)*2,center[1]),rad/3)
        ax.plot(x,y,color='k')
        draw_in_circles(ax,n-1,(center[0]-(rad/3)*2,center[1]),rad/3)

        #creates the right circle
        x,y = circle((center[0]+(rad/3)*2,center[1]),rad/3)
        ax.plot(x,y,color='k')
        draw_in_circles(ax,n-1,(center[0]+(rad/3)*2,center[1]),rad/3)

        #creates bottom circle
```

```
        x,y = circle((center[0],center[1]-(rad/3)*2),rad/3)
        ax.plot(x,y,color='k')
        draw_in_circles(ax,n-1,(center[0],center[1]-(rad/3)*2),rad/3)

        #creates top circle
        x,y = circle((center[0],center[1]+(rad/3)*2),rad/3)
        ax.plot(x,y,color='k')
        draw_in_circles(ax,n-1,(center[0],center[1]+(rad/3)*2),rad/3)



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

#coordinates for the original square
coord = [800,800]
#1a uses only 2 so it only creates 2 squares
#draw_squares(ax,2,coord,200)

#1b uses only 3 so it only creates 3 squares
#draw_squares(ax,3,coord,200)

#1c uses only 4 so it only creates 4 squares
#draw_squares(ax,4,coord,200)



#2a calls the method 5 times but the circle keeps getting smaller because
#the radius is been multiplied by 'w'
#draw_circles(ax, 10, [100,0], 100,.5)

#2b calls the method 45 times
```

```
#draw_circles(ax, 45, [100,0], 100,.9)

#2c calls the method 100 times
#draw_circles(ax, 100, [100,0], 100,.95)



#initial point for binary tree
origin = [0,0]
#3a calls the method 3 times so there are 3 levels
#draw_tree(ax,3,origin,5,5)

#3b calls the method 4 times so there are 4 levels
#draw_tree(ax,4,origin,5,5)

#3c calls the method 7 times so there are 7 levels
#draw_tree(ax,7,origin,8,8)



#4a calls the method 2 times
draw_in_circles(ax, 2, [100,0], 100)



#4b calls the method 3 times
#draw_in_circles(ax, 3, [100,0], 100)

#4c calls the method times
#draw_in_circles(ax, 4, [100,0], 100)

ax.set_aspect(1.0)
ax.axis('on')
```

```
plt.show()
fig.savefig('lab1.png')
```