Sergio Gramer – 88521512
Lab #1: (Recursively Draw Figures) Report

In the first lab I was tasked to draw a sequence of images after having practiced with drawing some basic squares and circles. The lab instructed to recursively draw sequences of images that I had never seen before. We had two .py programs that gave us a basic introduction into the material and allowed us to play with numbers to see how the recursion happened.

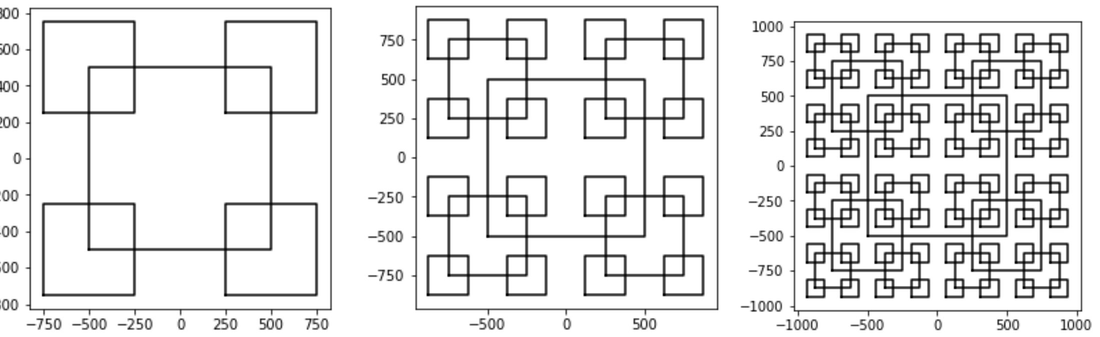
My initial proposed solution was using the code given to us modify it so that I could achieve the images. My first approach to all of the images was modifying the function that made changes to the image. P and Q were my variables used to manipulate the numbers to arrive at variations of the squares, circles and triangles. I first attempted to modify the X and Y values which got me results but only up until a certain point. I was able to get the first set of images (squares) to draw out until the third set of squares which was giving me a hard time. I spent a lot of time modifying and trying to come up with solutions before I started doing the circles. I ended up completing the circles image before the squares because I was able to come up with a different approach during my attempts at solving the circles. I noticed that what I was doing wasn't in fact "true" recursion because it just mathematically modified the values that had already begun. The only parameter stopping it from continuing was the N value which was used to count how many squares I wanted. I formulated that I had to go in levels of squares in order to achieve true recursion. What I did was insert a second recursive function, and then a third and then a fourth to manipulate the numbers and draw out the squares that I needed. I saw that every thing that needed to have "levels" of images had to have more than one recursive call. I achieved this with all of them except the first circle which only needed 1 recursive call to execute successfully.

To achieve this first image I:
Defined my function that would call recursively each square

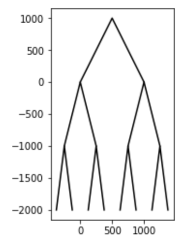
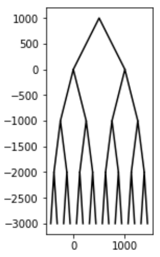
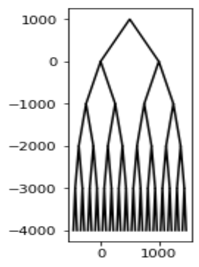
```
def square_level(ax,n,p,w):
   if n > 0:
      draw_square(ax,p) #will ensure a square is drawn given parameters.
      q = p*w
      square_level(ax,n-1,q + [-x , -y],w) #These parameters ensure the recursion
      square_level(ax,n-1,q + [x , y],w)  #call repeats itself enough times to
      square_level(ax,n-1,q + [x, -y],w)  #make all of the squares while modifying its
      square_level(ax,n-1,q + [-x, y],w)  #x & y values
```

and also defined a function that would draw my square separately:

```
#this function will draw the square
def draw_square(ax,p):
      ax.plot(p[:,0],p[:,1],color='k') ## key operation to plot
```



I used almost the same approach for each of the sequences:

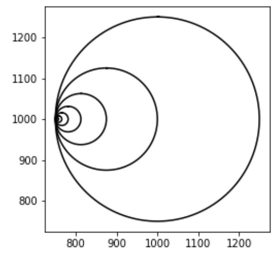
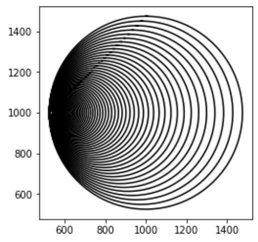
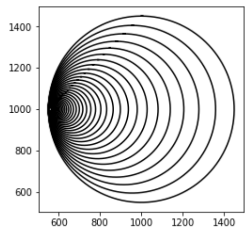




This sequence actually took me some time to notice that only the height was being modified and not the width when modifying P or Q:

```
q = (p * [w, 1])

def triangle_level(ax,n,p,w) :
    if n > 0 :
        draw_triangle(ax, p)
        q = (p * [w, 1])
        triangle_level(ax, n-1, q-[(orig_size*w)*w,orig_size],w) #will draw triangles on left hand side
        triangle_level(ax, n-1, q+[orig_size - (orig_size * (w/2)), -orig_size], w) #will draw triangles on
```

Using 2 recursive calls I was able to get all of the triangles that I needed. I noticed that: If I need 4 squares in my first recursive call I would need 4 recursive calls. If I needed 2 triangles on the second level then I would need just 2.

For the recursive circles method, I only used one recursive call which got me the results I desired.

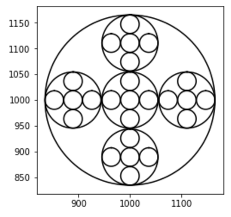




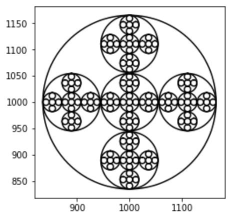
For this method:

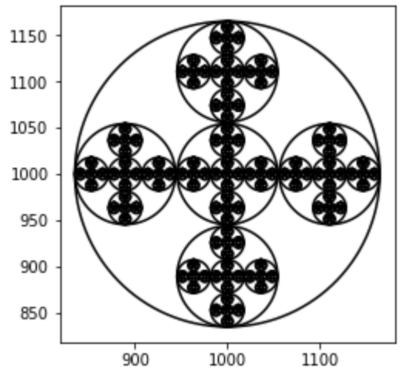
```
def draw_circle(ax, cx, cy, radius) :
        x,y = circle(cx, cy, radius)
        ax.plot(x,y,color='k')

def circle_level(ax, n, cx, cy, radius, w) :
    if n > 0 :
        draw_circle(ax, cx, cy, radius*w)
        circle_level(ax, n-1, cx-(radius-(radius*w))*w, cy, (radius*w), w)
```

It was specially difficult seeing what would get my circles to align properly. I was trying to figure out why if R1^2 –(R1*w)2 wouldn't give me the distance of the circles that needed to be shifted.
I figured subtracted the diameter of the bigger circle from the diameter of the smaller circle would give me the distance needed to travel left to make them align.

The last recursive images I drew: (not in that order)







←(bad image quality)

Some code I used for this image:

```
def draw_circle(ax, cx, cy, radius) :
      x,y = circle(cx, cy, radius)
      ax.plot(x,y,color='k')

def circle_level(ax, n, cx, cy, radius, w) :
   if n > 0 :
      draw_circle(ax, cx, cy, radius*w)
      circle_level(ax, n-1, cx, cy, (radius*w), w)#will draw center circle
      circle_level(ax, n-1, cx-(radius-(radius*w))*w, cy, (radius*w), w) #will draw circles to left
      circle_level(ax, n-1, cx+(radius-(radius*w))*w, cy, (radius*w), w) #will draw circles to right
      circle_level(ax, n-1, cx, cy-(radius-(radius*w))*w, (radius*w), w) #will draw circle down
      circle_level(ax, n-1, cx, cy+(radius-(radius*w))*w, (radius*w), w) #will draw circle up


   elif n == 0 :
      return
```

From this lab, I learned that recursion can be called in many different ways. Depending on what output you are looking for you might need to add more than one recursive call to make things work. I also learned that these images are far more than triangles. It is software doing math and outputting expressions that we input. If your math is off, or are expressions are poorly written the computer can interpret it as something completely different than what we were expecting.

APPENDIX

recursive_squares1.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  6 11:12:01 2019

@author: SergioGramer
"""
import numpy as np
import matplotlib.pyplot as plt

#this function will draw the square
def draw_square(ax,p):
       ax.plot(p[:,0],p[:,1],color='k') ## key operation to plot

#this function will determine which "level" the square is on
#n deteremines the "level"
def square_level(ax,n,p,w):
    if n > 0:
       draw_square(ax,p) #will ensure a square is drawn given parameters.
       q = p*w
       square_level(ax,n-1,q + [-x , -y],w) #These parameters ensure the recursion
       square_level(ax,n-1,q + [x , y],w)  #call repeats itself enough times to
       square_level(ax,n-1,q + [x, -y],w)  #make all of the squares while modifying its
       square_level(ax,n-1,q + [-x, y],w)  #x & y values
    else :
       return


plt.close("all")
orig_size = 1000 #to simplify numbers i chose 1000 arbitrarily
x = orig_size / 2 #will control the x axis point
y = orig_size / 2 #will control y could have been the same var X but for readability
p = np.array([[-x,-y],[-x,y],[x,y],[x,-y],[-x,-y]])
fig, ax = plt.subplots()
square_level(ax,4,p,.5) #changing n will control variables.
ax.set_aspect(1.0)
#ax.axis('off')
plt.show()
fig.savefig('squares.png')
```

recursivecircles.py

```
"""
Created on Thurs Feb  7 11:43:29 2019

@author: SergioGramer
"""

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

def circle(cx, cy, rad) :
    n = int(4*rad*math.pi)
    t = np.linspace(0,6.3,n)
    x = (cx+rad*np.sin(t))
    y = (cy+rad*np.cos(t))
    return x,y

def draw_circle(ax, cx, cy, radius) :
        x,y = circle(cx, cy, radius)
        ax.plot(x,y,color='k')

def circle_level(ax, n, cx, cy, radius, w) :
    if n > 0 :
        draw_circle(ax, cx, cy, radius*w)
        circle_level(ax, n-1, cx-(radius-(radius*w))*w, cy, (radius*w), w)

    elif n == 0 :
        return

orig_size = 1000
fig, ax = plt.subplots()
plt.close("all")
fig, ax = plt.subplots()
circle_level(ax, 10, orig_size, orig_size, 500, .5)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('circles.png')
```

Recursivetrianglespy.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb  8 20:13:04 2019

@author: SergioGramer
"""

import numpy as np
import matplotlib.pyplot as plt

def draw_triangle(ax,p) :
        ax.plot(p[:,0],p[:,1],color='k') ## key operation to plot

def triangle_level(ax,n,p,w) :
    if n > 0 :
        draw_triangle(ax, p)
        q = (p * [w, 1])
        triangle_level(ax, n-1, q-[(orig_size*w)*w,orig_size],w) #will draw triangles on left hand side
        triangle_level(ax, n-1, q+[orig_size - (orig_size * (w/2)), -orig_size], w) #will draw triangles on right hand

plt.close("all")
orig_size = 1000
p = np.array([[orig_size,0], [500,orig_size], [0,0]])
fig, ax = plt.subplots()
triangle_level(ax,5,p,.5)
ax.set_aspect(1.0)
#ax.axis('off')
plt.show()
fig.savefig('triangle.png')
```

Recursivecirclescross.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb  8 21:05:02 2019

@author: SergioGramer
"""

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

def circle(cx, cy, rad) :
    n = int(4*rad*math.pi)
    t = np.linspace(0,6.3,n)
    x = (cx+rad*np.sin(t))
    y = (cy+rad*np.cos(t))
    return x,y

def draw_circle(ax, cx, cy, radius) :
        x,y = circle(cx, cy, radius)
        ax.plot(x,y,color='k')

def circle_level(ax, n, cx, cy, radius, w) :
    if n > 0 :
        draw_circle(ax, cx, cy, radius*w)
        circle_level(ax, n-1, cx, cy, (radius*w), w)#will draw center circle
        circle_level(ax, n-1, cx-(radius-(radius*w))*w, cy, (radius*w), w) #will draw circles to left
        circle_level(ax, n-1, cx+(radius-(radius*w))*w, cy, (radius*w), w) #will draw circles to right
        circle_level(ax, n-1, cx, cy-(radius-(radius*w))*w, (radius*w), w) #will draw circle down
        circle_level(ax, n-1, cx, cy+(radius-(radius*w))*w, (radius*w), w) #will draw circle up


    elif n == 0 :
        return

orig_size = 1000
fig, ax = plt.subplots()
plt.close("all")
fig, ax = plt.subplots()
circle_level(ax, 5, orig_size, orig_size, 500, .33)
ax.set_aspect(1.0)
ax.axis('on')
plt.show()
fig.savefig('circles.png')
```

ACADEMIC CERTIFICATION

I “Sergio Gramer” 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.

Signed: 02/08/2018

Sergio Gramer

[illegible]

____