**Introduction**

The purpose to the lab was to help understand and practice recursion. The lab was given with two codes: one to draw squares and another to draw circles. It was expected that four different types of graphs could be drawn with the aid of those two codes. The four graphs to be drawn consisted of squares with one square at each corner, circles shrinking and shifting left to meet the radius of the previously drawn circle, binary trees, and circles with five circles within themselves.

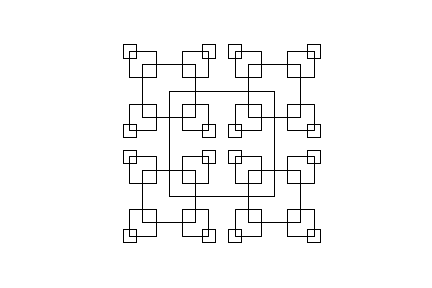
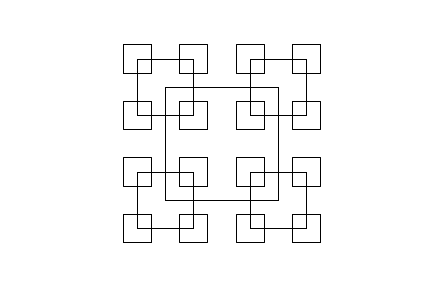
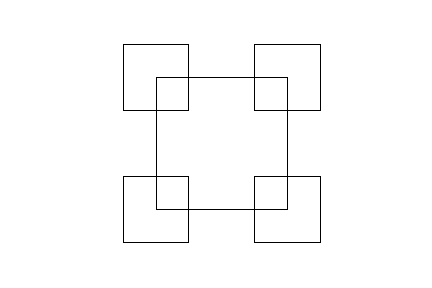
**Proposed Solution Design and Implementation**

The attempts made were originally based on trial and error, until it became apparent that simple Algebra was the major factor in producing each graph. The way the lab was completed was by separating each task at hand and tackling them one at a time.

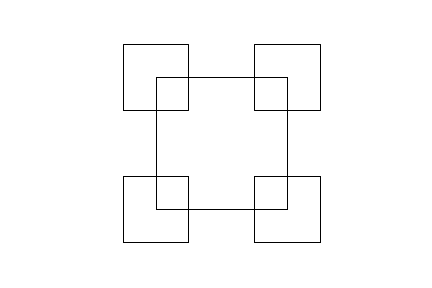
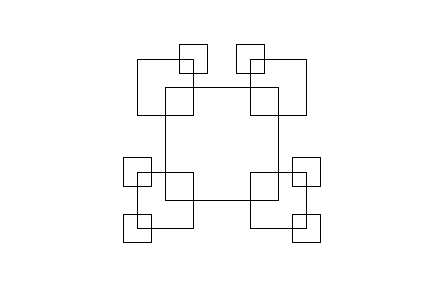
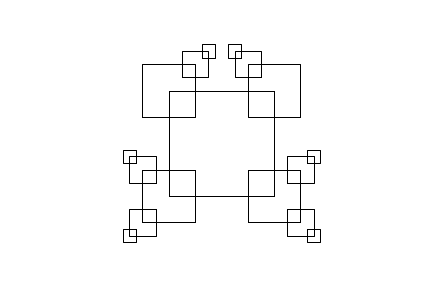
1. **Squares:**
   1. The first task entailed the shifting of each subsequent square to at least one corner, in this case it started at the bottom left corner of each subsequent square.
   2. Once that had been completed, it was soon realized that two corners could be drawn at once by adding a constant variable (z) to shift the squares left and right, up and down, this produces a sequence of squares on the top right and bottom left corners of each subsequent square. A second line of code was used to produce the top left and bottom right squares of each square.
   3. The next step was to find a way to draw another set of squares at each corner, this was done by making four types of coordinates: (+x, +y), (+x,-y), (-x,+y), (-x,-y)effectively reflecting another set of squares upon the x-axis and y-axis of those subsequent squares. The addition of a multiple of ¼th was used to shrink each square to half the size of the previous squares of ½.
   4. The final step was the creation of recursive calls. Each call would multiply each square by a constant w, which would allow for each square’s center to be at the corner of the previous’ corner. Then negative coordinates were introduced here as well to reflect each miniature square onto each of the corners, as were negative z, positive z, and z subtracting from itself added here to shift each square into its proper corner.
2. **Spiraling Circles:**
   1. This task was the easiest to complete, the given code already had the drawing of the shrinking circles. The only task to complete was the shifting of each subsequent circle to the left of the previously drawn circle.
   2. Having algebra in mind, it was understood that the main function that plots the circle would need a left-shift. This is accomplished by adding an arbitrary number to the x-coordinate. In this case, it was fairly simple to realize that adding the radius to the x-coordinate while plotting would shift each circle to the left as needed.
   3. The final task was the recursive call, figuring out how to shrink each succeeding circle. The recursive call requires that radius is multiplied by some number w, where w is less than 1, thus producing a smaller circle with each recursive call.
3. **Binary Trees:**
   1. The first obstacle faced here was figuring out how to contort each following branch into angles that near 90 degrees upward with each recursive call. The addition of x\*dx to the plotting line effectively solved this problem.
   2. Using two lines of the same code, it is possible to plot a left and right branch just by making one line’s x-coordinate negative, this mirrors the left and right branches.
   3. The next step was adding a k variable, where k is a number multiplying by four each recursive call, this was made to shift the following branches left and right to construct the tree. The reason k is multiplying by four each call is because when running the program, the tree would bend with each subsequent call, but it would bend into itself. The first call produced two branches, the second call got a little distorted, and each sequential branch molded into the center of the tree, obviously a shift was needed to shift the branches into their corresponding place. The addition of +dx was added to the x-coordinate of the recursive call. Once there were four +dx, the program worked for a few branches, here it was realized that a constant multiple was needed, thus k\*4 was added, as well as x+k\*dx to the parameters of the recursive call.
   4. It should be noted that each branch moves half the distance of the previous one, which is why dx/2 is used in the recursive call. Each branch moves down by one with each call, which is why dy-1 is used in the recursive call.
4. **Circles:**
   1. This was the most difficult challenge to complete for this lab. This task is similar to the squares one, except instead of corners the production of circles must be within the previously drawn circle.
   2. The first objective was to draw circles minimizing in size each recursive call, this is done by adding radius to x in the plot.
   3. Calling radius/3 in the parameters of the recursive call enables the drawing of three circles along the diameter of the previously drawn circle (main circle).
   4. The final step made was the use of addition to shift the shapes up, down, left, and right, after this, the steps were repeated to create circles on each of the sides (upper, lower, left, and right).

**Results**

**Squares:**

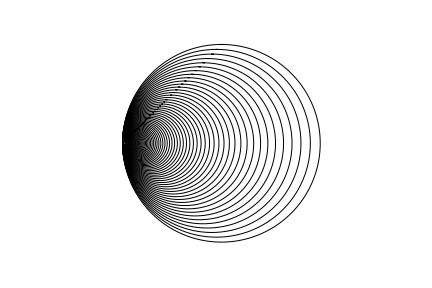
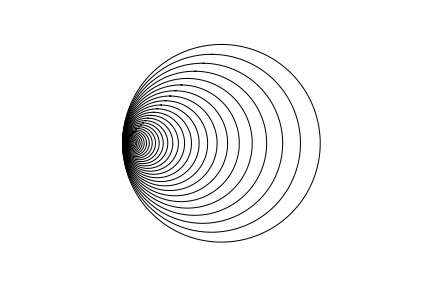
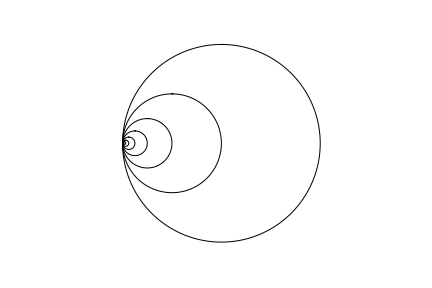


These three pictures represent the final drawings produced by the source code.

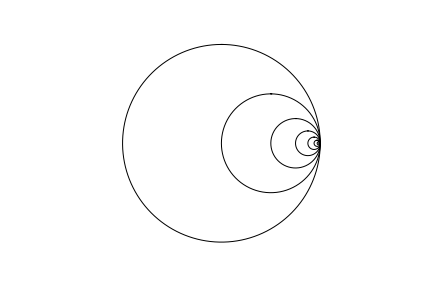
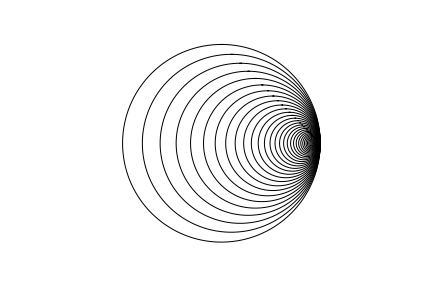
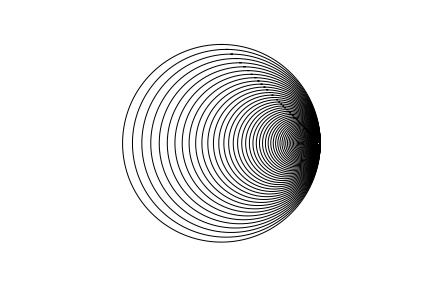
  

These three pictures represent what happens when two recursive calls are removed from the source code: both draw\_squares(ax,n-1,-p\*w,w,-z) and draw\_squares(ax,n-1,-p\*w,w,z-z). These lines are needed to shift the squares about the drawing as well as reflect some squares to complete the picture.

**Spiraling Circles:**

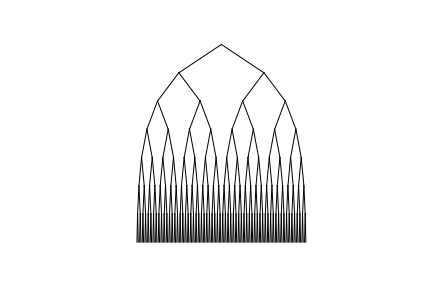
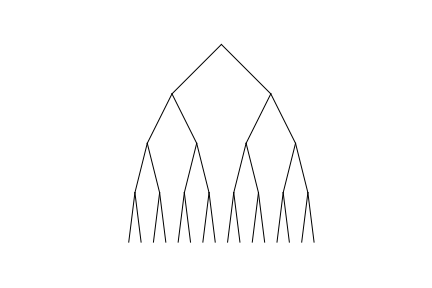
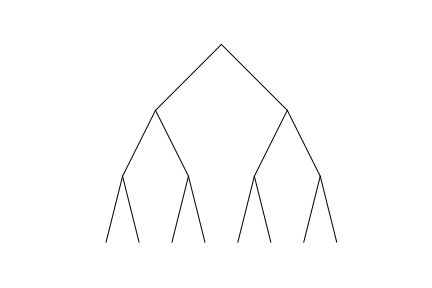


These three pictures represent the final drawings produced by the source code. Function draw\_SpiralCircles has x+radius in its line for plotting.

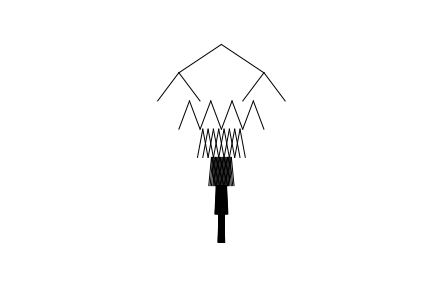
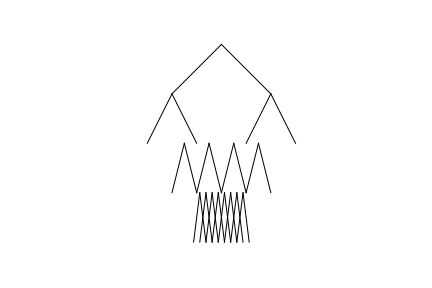
  

These three pictures represent what happens if the function draw\_SpiralCircles has a plotting line with x-radius. This was found by mistake when attempts were made to try and complete the draw\_circles function. Understanding that x-radius created this lead to the solution of x+radius to complete this function.

**Trees:**

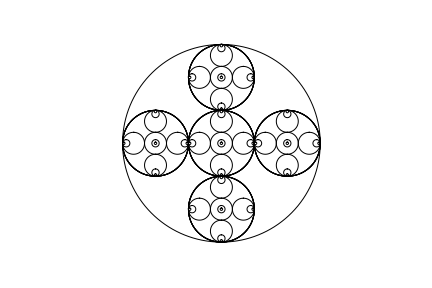
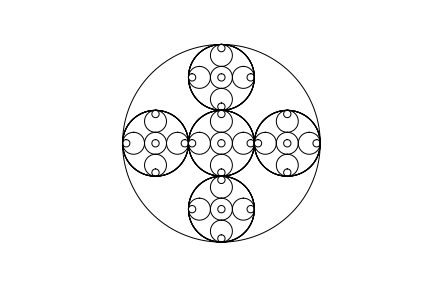
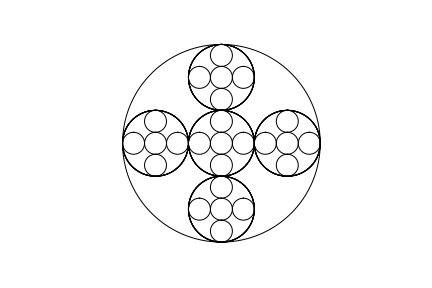


These three pictures represent the final drawings produced by the source code.

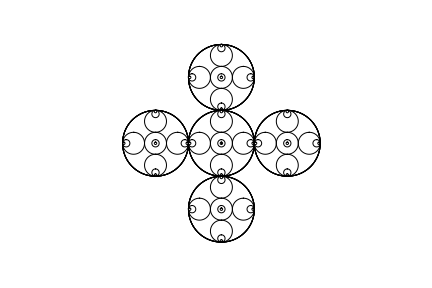
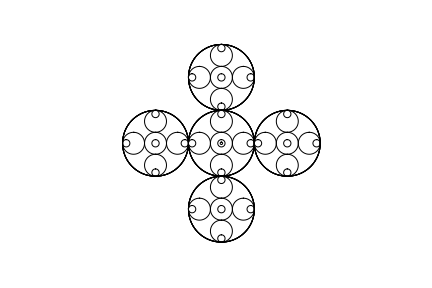
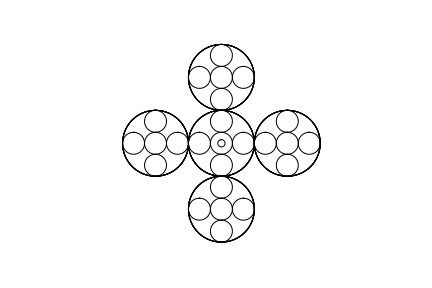
 

These three pictures depict the first drawings produced by the source code, which was done without the use of a k constant multiple. This demonstrates why the use of k\*4 was used in the parameters; without the use of a k constant, the branches shift into each other more and more after every recursive call.

**Circles:**



These three pictures represent the final drawings produced by the source code.



These three pictures are produced when ax.plot(x,y,linewidth=1,color='k') is used in place of ax.plot(x\*3,y\*3,linewidth=1,color='k'). The multiple of three was used to fix this mistake.

**Conclusions**

The main gain from these exercises were a deeper understanding of recursion, as well as an understanding of how mathematics plays a major role in Computer Science. The exercises emphasize the need to understand recursion, without that understanding, the completion of this objective is impossible. The code given really helped in understanding how each call would alter the shapes of the proceeding figures, which gave insight on how to manipulate the figures needed for completion. The amount of algebra needed for this lab only calls for a basic understanding, yet without critical thinking, as well as common sense and patience, the task at hand becomes a very difficult one of guessing. Having finished this lab, it is understood that Computer Science and Mathematics are deeply intertwined, it is also understood that with enough knowledge and understanding in both, any task is accomplishable. The final learned idea from this lab is the fact that regardless of one hundred percent correct-completion or less, the main goal of this exercise was to stimulate and motivate students to complete the job at hand whether in familiar territory (the idea of recursion) or in a whole new environment (the language of Python).

**Appendix: Source Code**

"""

Course: CS 2302

Author: Jacob Montenegro

Lab: 1

Instructor: Dr. Olac Fuentes

T.A.: Anindita Nath

Date of Last Modification: 2/8/2019

Program's Purpose: Practice using recursion to produce figures.

"""

import numpy as np

import matplotlib.pyplot as plt

import math

#draw\_squares recursively draws squares on every corner of a given square, the depth being n, the number of times to recursively call the method

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

if n>0:

#These two lines of code give the Original Square four corners.

ax.plot(p[:,0]/2-z,p[:,1]/2-z,linewidth=1,color='k')

ax.plot(-p[:,0]/2+z,p[:,1]/2-z,linewidth=1,color='k')

#The if statement ensures that the recursive call doesn't produce more squares than necessary.

if n>1:

#The next four lines of code produce miniture squares for the previously produces squares at each corner.

#They are dividing by 4 since that is half the size of the previously created squares.

ax.plot(-p[:,0]/4,-p[:,1]/4+z,linewidth=1,color='k')

ax.plot(p[:,0]/4,-p[:,1]/4+z,linewidth=1,color='k')

ax.plot(-p[:,0]/4+z,-p[:,1]/4,linewidth=1,color='k')

ax.plot(p[:,0]/4-z,-p[:,1]/4,linewidth=1,color='k')

#These recursive calls ensure that z alternates between negative ten and positive ten, z (10) is used to shift the squares left, right, up, and down.

draw\_squares(ax,n-1,p\*w,w,z)

draw\_squares(ax,n-1,-p\*w,w,-z)

draw\_squares(ax,n-1,-p\*w,w,z-z)

#The circle method/function was given by Dr. Fuentes, it creates the circle.

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

#This method produces circles shifting to the left along the perimeter of every previous circle.

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

if n>0:

x,y = circle(center,radius)

#I used x+radius here to shift every circle left, adding to x within a function shifts a given graph left.

ax.plot(x+radius,y,linewidth=1,color='k')

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

#This method draws a binary tree, it has an x,y, change in x (dx) and cahnge in y (dy), as well as a k.

def draw\_trees(ax,n,x,y,dx,dy,k):

if n>0:

#The first line prints a branch, the second line prints a reflection of the first line.

ax.plot(x\*dx,y,linewidth=1,color='k')

ax.plot(-x\*dx,y,linewidth=1,color='k')

#k is added since I noticed that the drawing kept shifting inward by multiples of 2, 4, 16, etc.

#I realized that in order to shift left and right, I would need to continuously add and subtract dx by multiples of 4, which is why I am using k.

draw\_trees(ax,n-1,x+k\*dx,y-dy,dx/2,dy,k\*4)

draw\_trees(ax,n-1,x-k\*dx,y-dy,dx/2,dy,k\*4)

#draw\_circles produces a circle with 5 circles within it, recursively.

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

if n>0:

x,y = circle(center,radius)

#x and why are multiplying by 3 to ensure the first circle engulfs the next five.

ax.plot(x\*3,y\*3,linewidth=1,color='k')

#This if statement prevents any extra circles from being drawn.

if n>1:

ax.plot(x+radius+100,y,linewidth=1,color='k')

ax.plot(x-radius+300,y,linewidth=1,color='k')

ax.plot(x+200,y-radius+100,linewidth=1,color='k')

ax.plot(x+200,y+radius-100,linewidth=1,color='k')

miniCircles(x,y,radius)

#The circles fit along the diameter of the engulfing circle, thats why radius is divided by 3 at each recursive call.

draw\_circles(ax,n-1,center,radius/3,w)

def miniCircles(x,y,radius):

#Produces right circle

ax.plot(x+200,y,linewidth=1,color='k')

#Produces left circle

ax.plot(x-200,y,linewidth=1,color='k')

ax.plot(x+radius-300,y,linewidth=1,color='k')

ax.plot(x-radius-100,y,linewidth=1,color='k')

ax.plot(x-200,y-radius+100,linewidth=1,color='k')

ax.plot(x-200,y+radius-100,linewidth=1,color='k')

#Produces center circle

ax.plot(x+radius-100,y,linewidth=1,color='k')

ax.plot(x-radius+100,y,linewidth=1,color='k')

ax.plot(x,y-radius+100,linewidth=1,color='k')

ax.plot(x,y+radius-100,linewidth=1,color='k')

#Produces top circle

ax.plot(x,y+200,linewidth=1,color='k')

ax.plot(x+radius-100,y+200,linewidth=1,color='k')

ax.plot(x-radius+100,y+200,linewidth=1,color='k')

ax.plot(x,y-radius+300,linewidth=1,color='k')

ax.plot(x,y+radius+100,linewidth=1,color='k')

#Produces bottom circle

ax.plot(x,y-200,linewidth=1,color='k')

ax.plot(x+radius-100,y-200,linewidth=1,color='k')

ax.plot(x-radius+100,y-200,linewidth=1,color='k')

ax.plot(x,y-radius-100,linewidth=1,color='k')

ax.plot(x,y+radius-300,linewidth=1,color='k')

"""Square Section"""

#Square 1

plt.close("all")

orig\_size = 10

p = np.array([[30,30],[orig\_size,30],[orig\_size,orig\_size],[30,orig\_size],[30,30]])

fig, ax = plt.subplots()

draw\_squares(ax,2,p,.5,10)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares.png')

#Square 2

plt.close("all")

orig\_size = 10

p = np.array([[30,30],[orig\_size,30],[orig\_size,orig\_size],[30,orig\_size],[30,30]])

fig, ax = plt.subplots()

draw\_squares(ax,3,p,.5,10)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares1.png')

#Square 3

plt.close("all")

orig\_size = 10

p = np.array([[30,30],[orig\_size,30],[orig\_size,orig\_size],[30,orig\_size],[30,30]])

fig, ax = plt.subplots()

draw\_squares(ax,4,p,.5,10)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares2.png')

"""Spiral Circle Section"""

#Spiraling Circle 1

plt.close("all")

fig, ax = plt.subplots()

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

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')

#Spiraling Circle 2

plt.close("all")

fig, ax = plt.subplots()

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

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles1.png')

#Spiraling Circle 3

plt.close("all")

fig, ax = plt.subplots()

draw\_Spiralcircles(ax, 85, [50,0], 100,.95)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles2.png')

"""Tree Section"""

#Tree 1

plt.close("all")

orig\_size = 10

p = np.array([[0,1],[1,0]])

fig, ax = plt.subplots()

draw\_trees(ax,3,p[0],p[1],1,1,2 )

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('tree.png')

#Tree 2

plt.close("all")

orig\_size = 10

p = np.array([[0,1],[1,0]])

fig, ax = plt.subplots()

draw\_trees(ax,4,p[0],p[1],1,1,2)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('tree1.png')

#Tree 3

plt.close("all")

orig\_size = 10

p = np.array([[0,1],[1,0]])

fig, ax = plt.subplots()

draw\_trees(ax,7,p[0],p[1],1.5,1,1.34)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('tree2.png')

"""Circle Section"""

#Circle 1

plt.close("all")

fig, ax = plt.subplots()

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

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circlesOfCircles.png')

#Circle 2

plt.close("all")

fig, ax = plt.subplots()

draw\_circles(ax, 4, [0,0], 100,.5)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circlesOfCircles1.png')

#Circle 3

plt.close("all")

fig, ax = plt.subplots()

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

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circlesOfCircles2.png')