Lab #1

Data Structures
Laura Berrout
ID# 80607326

Introduction

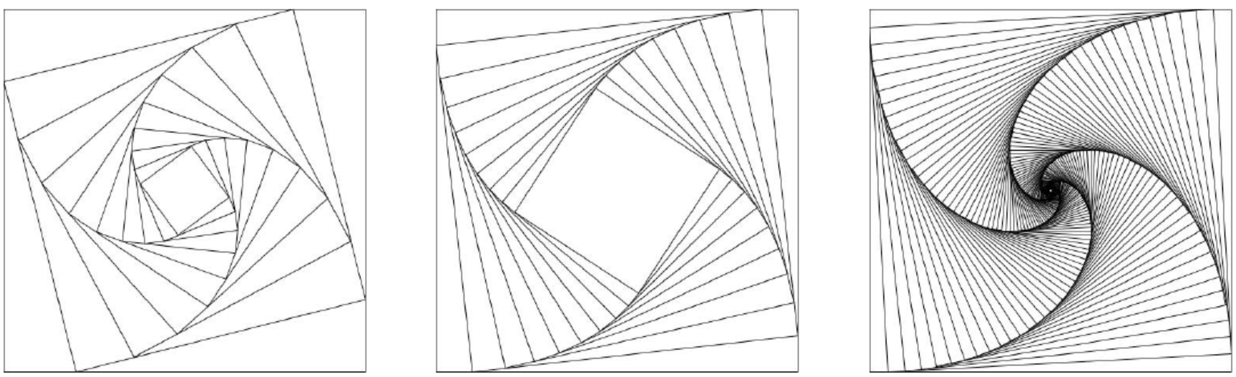
Using recursion, draw interesting figures. The programs draw squares.py and draw circles.py to generate the following figures:

For this lab you will practice using recursion to draw interesting figures. The programs draw squares.py and draw circles.py posted in the class webpage can be used to generate the figures.

Proposed solution design and implementation & Experimental results

For each figure a set of proposed solutions and experimentations are describe in the next tables:

Figure 1:



Proposed Solutions and Experiments

The Proposed Solution and experiments for each are describe in the next table:

Experiments		**Proposed Solution**	**Square a)**
#	**Changes**	**Description**	**Results**
1	Test code	No modifications	Observe how the code behaves
2	n =10	Draw 10 squares	Draw 10 squares intead of 15 but still is not close to the desired figure (a)
3	w = 0.8	Change squares directions	Still in the opposite direction
4	w = 0.2	Change squares directions	Very close to the desired figure (a)
Experiments		**Proposed Solution**	**Square b)**
#	**Changes**	**Description**	**Results**
1	w = 0.1	Closer distances	The distance between the squares vertex decresed, obtained a very close look to the desired figure (b)
Experiments		**Proposed Solution**	**Square c)**
#	**Changes**	**Description**	**Results**
1	w = 0.5	Closer distances	Observe how the code behaves

2	n = 20	Increase the number of squares	Number of squares increased but it's not the desired figure
3	n = 50	Increase the number of squares	Number of squares increased but it's not the desired figure
4	n = 100	Increase the number of squares	Looks more saturated than the desired figure

The resulting Figures are as follows:

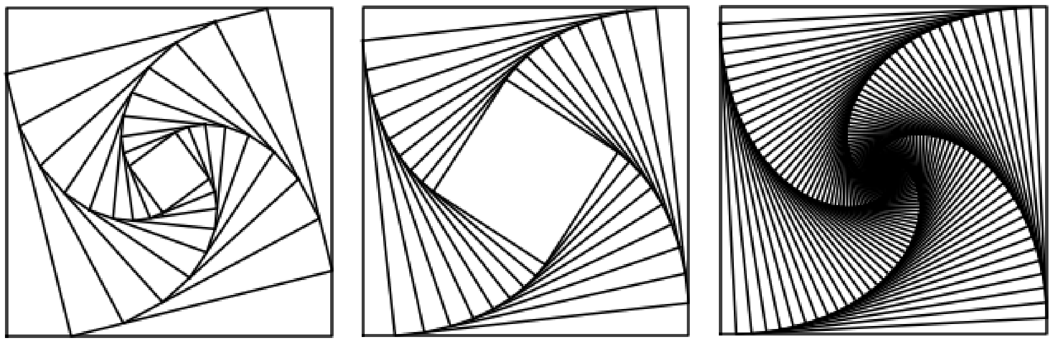
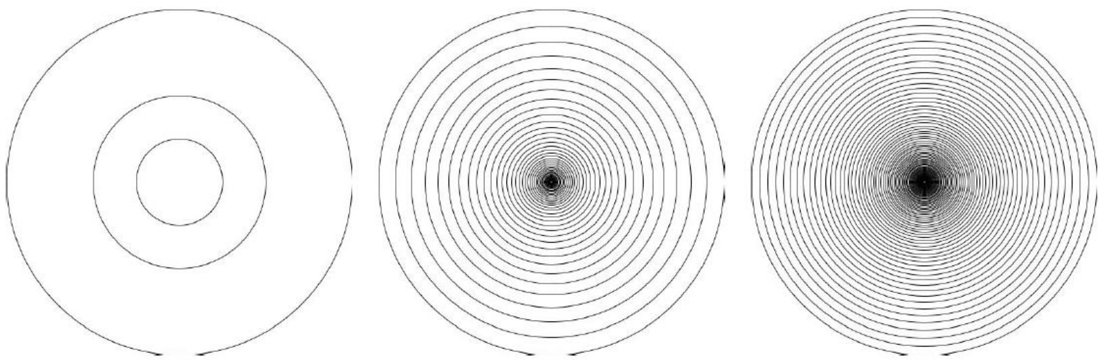


Figure 2:



a) b) c)

Proposed Solutions and Experiments

Experiments		Proposed Solution	Circle a)
#	**Changes**	**Description**	**Results**
1	Test code	No modifications	Observe how the code behaves
2	n = 5	Observe how the code behaves	Draw 5 circles but are not at the center
3	w = 0.5	Observe how the code behaves	Draw 5 circles and is closer to the desired figure but with a very small circle at the center
4	n = 3	Get the desired circles	Very close to the desired figure
Experiments		**Proposed Solution**	**Circle b)**
#	**Changes**	**Description**	**Results**
1	n = 50	Observe how the code behaves	Looks more saturated at the center but the circles have the same distance as the circle a)
2	w = 0.1	Observe how the code behaves	The distance between the circles increased
3	w = 0.9	Observe how the code behaves	The distance between the circles decreased
Experiments		**Proposed Solution**	**Circle c)**
#	**Changes**	**Description**	**Results**

1	w = 0.95	Get the desired distance between circles	Got the distance, but there are still missing circles at the center
2	n = 100	Increase the number of circles	Got the desired figure

The resulting figures are as follows:

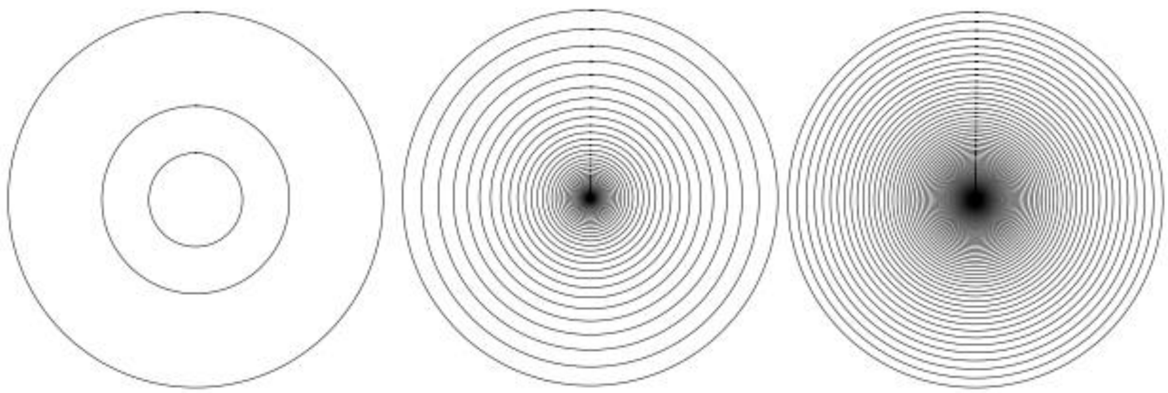
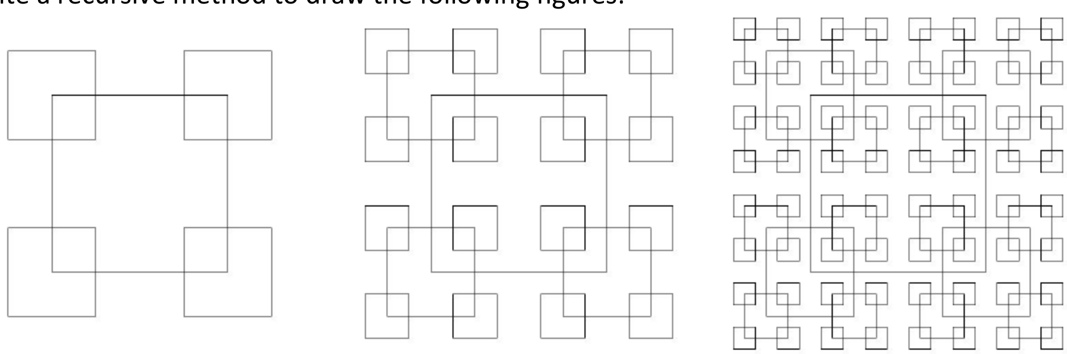


Figure 3:

1. Write a recursive method to draw the following figures:



a) b) c)

Proposed Solutions and Experiments:

Experiments		Proposed Solution	Corner squares a, b and c)
#	Changes	Description	Results
1	q = p[i1]*(1-w)	Change q in the recursion method	Got smaller squares in the same origin
2	w = 0.5	Chage to get a half the size of the square	Still has the same origin as the original square
3	q = p+p[i1]*(-w)	Modify the center of the square	Got squares tilted but remained the same size
4	q = p*w - 200	Modify the center of the square	Got a square in the lower left corner, but it only applies to this corner. If n is increased, the squares only apper on the corner of the next square
5	q = p*w + 600	Modify the center of the square	Got a square in the upper rigt corner. Happens the same cause as before.
6	q = p*w + [-200,600]	Modify the center of the square	Got the upper left square

7	q = p*w + [600,-200]	Modify the center of the square	Got the lower rigth square
8	def draw_corners(ax,n,p,w): draw_squaresur(ax,n,p,w) draw_squaresll(ax,n,p,w) draw_squaresul(ax,n,p,w) draw_squareslr(ax,n,p,w)	Add the recursion method of each corner to a method and add n = n-1	Get the square with squares at its corners. Missing to implemented to modify the size and work to more squares, taking the corners as centers.
	def draw_corners(ax,n,p,w,o)	Modify the recursive method to get the size of the square too	
	o1 = o*0.25 o2 = o*0.75	Add size to each of the methods so it works with every size	
	n = 3		The smaller squares stay at different coordinates
		A recursive call for draw_corners method	Still draws the same square with smaller squares on the cornes at different distances.
	q = p*w + [-o1,o2] q = p*w + (o2) q = p*w - (o1) q = p*w + [o2,-o1]	Modify the equations to work no matter the size	Draws a square with the same size and position
	draw_corners(ax,n,q,w,o)	after each equation, add a recursive call so draws a new square in that position	By modifying n to 2, 3, and 4, the code gives the three desired figures

The resulting figures:

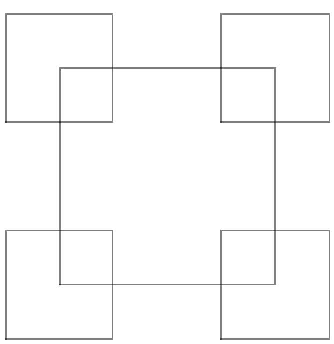
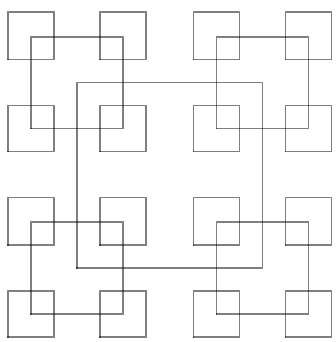
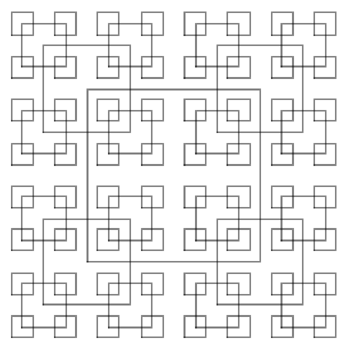
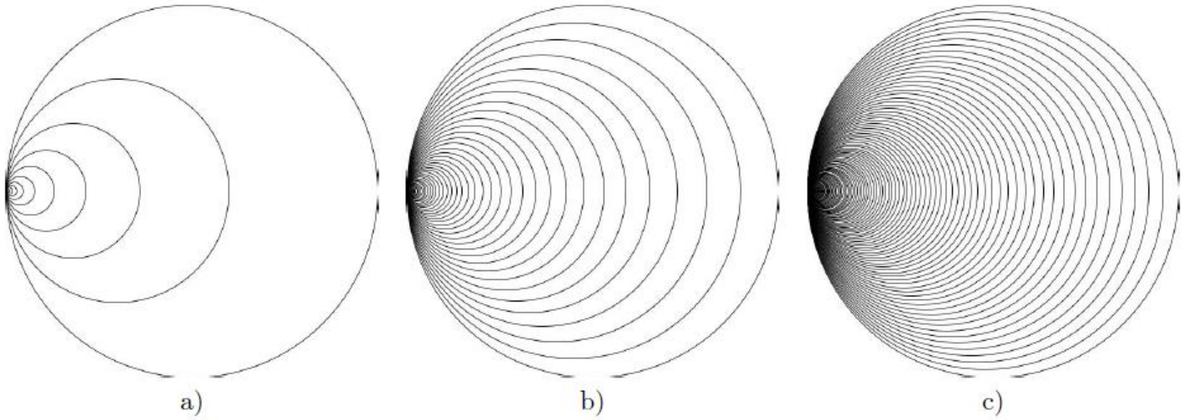




Figure 4:



Proposed Solutions and Experiments:

Experiments		Proposed Solution	Tangent Circles
#	**Changes**	**Description**	**Results**
1	x=(center[0]+rad)+rad*np.sin(t)	Modify the equation to move the center in the x axis	Move the circle
2	draw_circles(ax, 9, [100,0], 100,.60)	Modify to get the correct number of circles and the distance	Got the desired figure a
3	draw_circles(ax, 50, [100,0], 100,.90)	Modify to get the correct number of circles and the distance	Got the desired figure b and c

Resulting figures:

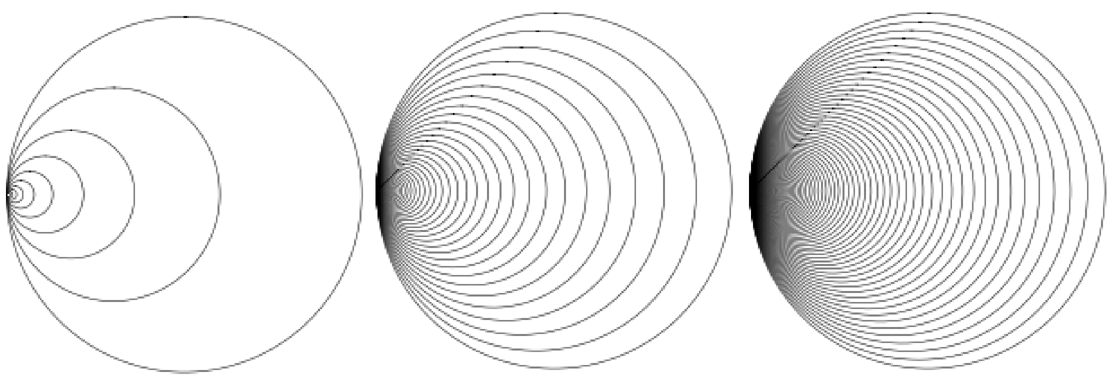
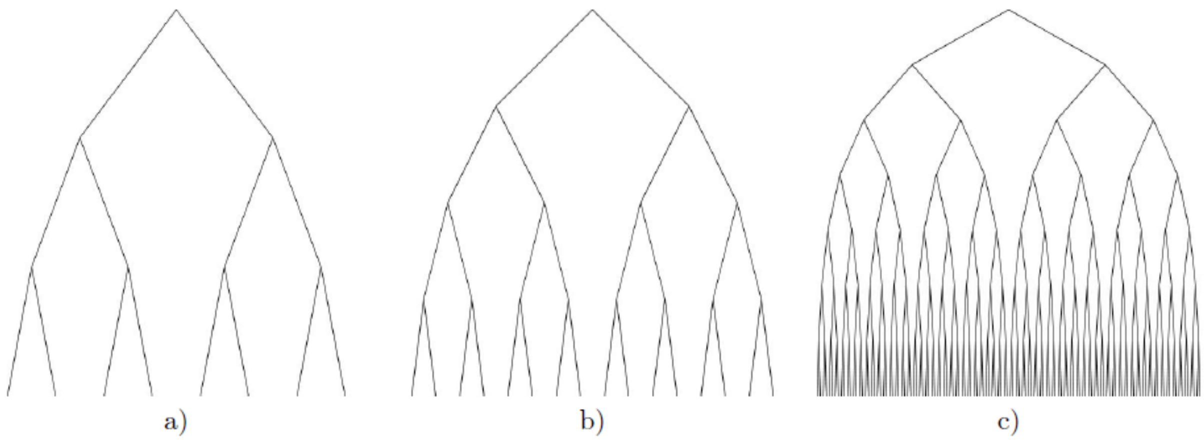


Figure 5:



Proposed Solutions and Experiments:

Results:

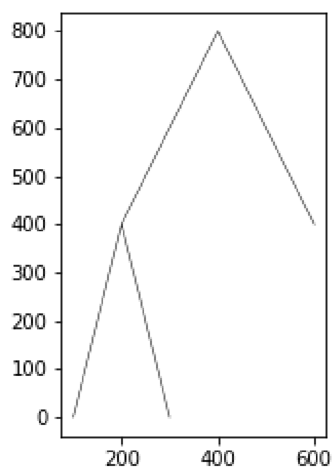
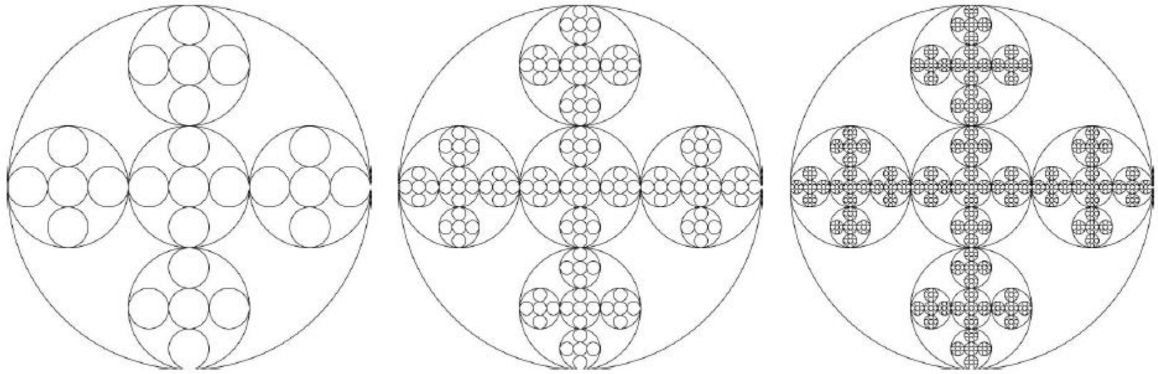


Figure 6:

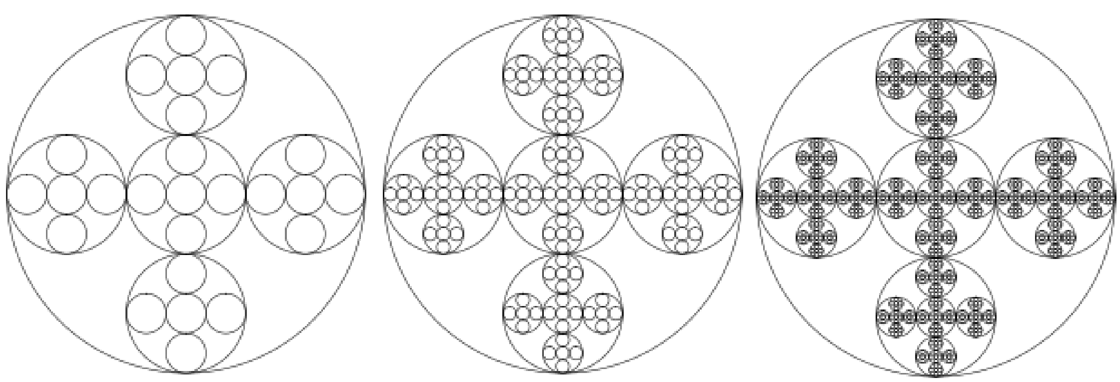


a) b) c)

Proposed Solutions and Experiments:

Experiments		Proposed Solution	Circles inside of circles
#	Changes	Description	Results
1	n = 3	Take original code to draw circles n now is for levels	Instead of the number of circles, RM will draw
2	draw_circles(ax,n,center,radius,w) #draws circle at the center	Call RM with each new center	Draws a circle with the specific center and radius
3	radius = radius/3	Divide radius by 3	Got the radius for the new circles
4	center1=[center[0]+(2*radius),center[1]] center2=[center[0]-(2*radius),center[1]] center3=[center[0],center[1]+(2*radius)] center4=[center[0],center[1]-(2*radius)]	Create 4 different centers for each circle	For each circle a new set of centers are generated for each inside circle
5	def draw_circles(ax,n,center,radius,w):	Create a new RM to draw the inside circles	Gives the circles inside with n been the number of circles drawn

Results:



Conclusions

- Basic programming in python was learned since I still do not have a basic understanding of this.
- The recursive methods in python have a lot of similarity with java.
- The language is easier to handle according to the manipulation of variables.

Appendix

Source Code:

```
"""
Course: Data Structures CS2302
Author: Laura Berrout
Assignment: Lab #1
Instructor: Dr. Olac Fuentes
T.A.:
Date of last modification: 02/08/2019
Purpose: Use recursion to draw interesting figures
"""

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

#Recursion method to draw squares
def draw_squares(ax,n,p,w):
  if n>0:
    i1 = [1,2,3,0,1]
    q = p*w + p[i1]*(1-w)
    ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)
    draw_squares(ax,n-1,q,w)

#Figure Square a
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,10,p,.2)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Square_A.png')

#Figure Square b
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,10,p,.1)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Square_B.png')

#Figure Square c
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,100,p,.05)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Square_C.png')

#Recursion method to draw 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:
    x,y = circle(center,radius)
    ax.plot(x,y,color='k',linewidth=0.5)
    draw_circles(ax,n-1,center,radius*w,w)

#Figure Circle A
plt.close("all")
fig, ax = plt.subplots()
```

```
draw_circles(ax, 3, [100,0], 100,.5)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Circle_A.png')

#Figure Circle B
plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 50, [100,0], 100,.9)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Circle_B.png')

#Figure Circle B
plt.close("all")
fig, ax = plt.subplots()
draw_circles(ax, 100, [100,0], 100,.95)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_Circle_C.png')

#Recursion method to draw squares with more
squares at the corners
# calls the method and then calls but taking the
coordinates of the new square
def draw_corners(ax,n,p,w,o): #for re-call of each
square -> n
  if n>0:
    draw_squaresur(ax,1,p,w,o)
    n = n-1
    o1 = o*0.25
    o2 = o*0.75
    q = p*w + [-o1,o2] #upper left corner
    draw_corners(ax,n,q,w,o)

    q = p*w + (o2) #upper right corner
    draw_corners(ax,n,q,w,o)

    q = p*w - (o1) #lower left corner
    draw_corners(ax,n,q,w,o)

    q = p*w + [o2,-o1] #lower right corner
    draw_corners(ax,n,q,w,o)


def draw_squaresur(ax,n,p,w,o):
  if n>0:
    ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)
    draw_squaresur(ax,n-1,p,w,o)

def draw_squaresll(ax,n,p,w,o):
  if n>0:
    ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)
    draw_squaresll(ax,n-1,p,w,o)

def draw_squaresul(ax,n,p,w,o):
  if n>0:
    ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)
    draw_squaresul(ax,n-1,p,w,o)

def draw_squareslr(ax,n,p,w,o):
  if n>0:
    ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)
    draw_squareslr(ax,n-1,p,w,o)



#Figure Square_Corner a
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_corners(ax,2,p,0.5,orig_size)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_CornerSquares_A.png')


#Figure Square_Corner a
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_corners(ax,3,p,0.5,orig_size)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_CornerSquares_B.png')
```

```
#Figure Square_Corner a
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_corners(ax,4,p,0.5,orig_size) #does not need
to modify w
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_CornerSquares_C.png')


#Method to draw circle
def circle2(center,rad):
    n = int(4*rad*math.pi)
    t = np.linspace(0,6.3,n)
    x = (center[0]+rad)+rad*np.sin(t)
    y = center[1]+rad*np.cos(t)
    return x,y

#Recursion method to draw tangent circles
def draw_circles2(ax,n,center,radius,w):
    if n>0:
        x,y = circle2(center,radius)
        ax.plot(x,y,color='k',linewidth=0.5)
        draw_circles2(ax,n-1,center,radius*w,w)

#Figure Circle A
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 10, [100,0], 100,.60)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_TangentCircle_A.png')

#Figure Circle B
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 50, [100,0], 100,.90)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_TangentCircle_B.png')

#Figure Circle C
plt.close("all")
fig, ax = plt.subplots()
draw_circles2(ax, 100, [100,0], 100,.95)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_TangentCircle_C.png')

#Recursive method to draw a tree, n is the number
of levels
def draw_invtree(ax,p,n,size):
    if n>0:
        size1 = [size[0]/n,size[1]/n]

        p = np.array([[size[0]/4,(n-
1)*(size[1]/n)],[size[0]/2,size[1]],[(3*size[0])/4,(n-
1)*(size[1]/n)]])
        print("new size: ", size1)
        draw_lines(ax,p,size1)
        n=n-1
        draw_invtree(ax,p,n,size1) #draws the left side

def draw_lines(ax,p,size): #left side
        print("points coordindates: ")
        print(p)
        ax.plot(p[:,0],p[:,1],color='k',linewidth=0.5)

#Figure Square a
plt.close("all")
orig_size = 800
size = [orig_size,orig_size]
print("size:",size)
n = 2  #number of levels
p = np.array([[size[0]/4,size[1]-
(size[1]/n)],[size[0]/2,size[1]],[(3*size[0])/4,size[1]-
(size[1]/n)]])
fig, ax = plt.subplots()
draw_invtree(ax,p,n,size)
ax.set_aspect(1.0)
#ax.axis('off')
plt.show()
fig.savefig('Fig_Invtree_A.png')

#Recursion method to draw circles
def draw_circles(ax,n,center,radius,w):
    if n>0:
        circle3(center,radius)

        n=n-1
```

```
    w = radius/3
    radius = radius/3
    center1 = [center[0]+(2*radius),center[1]]
    center2 = [center[0]-(2*radius),center[1]]
    center3 = [center[0],center[1]+(2*radius)]
    center4 = [center[0],center[1]-(2*radius)]

    draw_circles(ax,n,center,radius,w) #draws circle
at the center
    draw_circles(ax,n,center1,radius,w) #draws
right circle
    draw_circles(ax,n,center2,radius,w) #drawa left
circle
    draw_circles(ax,n,center3,radius,w) #drawa
upper circle
    draw_circles(ax,n,center4,radius,w) #drawa
down circle

def circle3(center,rad):
    n = int(4*rad*math.pi)
    t = np.linspace(0,6.3,n)
    x1 = (center[0])+rad*np.sin(t)
    y1 = center[1]+rad*np.cos(t)
    ax.plot(x1,y1,color='k',linewidth=0.5)
    return x1,y1


#Figure Circle A
plt.close("all")
fig, ax = plt.subplots()
radius = 100
n = 3
w = radius/3
draw_circles(ax, n, [100,0], radius, w)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_InsideCircle_A.png')

plt.close("all")
fig, ax = plt.subplots()
radius = 100
n = 4
w = radius/3
draw_circles(ax, n, [100,0], radius, w)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_InsideCircle_B.png')

plt.close("all")
fig, ax = plt.subplots()
radius = 100
n = 5
w = radius/3
draw_circles(ax, n, [100,0], radius, w)
ax.set_aspect(1.0)
ax.axis('off')
plt.show()
fig.savefig('Fig_InsideCircle_C.png')
```

"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."