Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: February 8, 2019

Instructor: Olac Fuentes

Assignment: Lab 1 Recursion

TA: Anindita Nath & Maliheh Zaragan

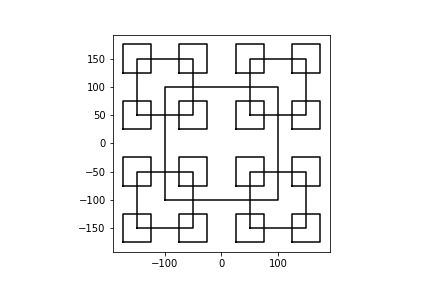
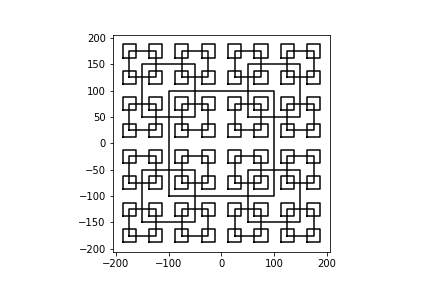
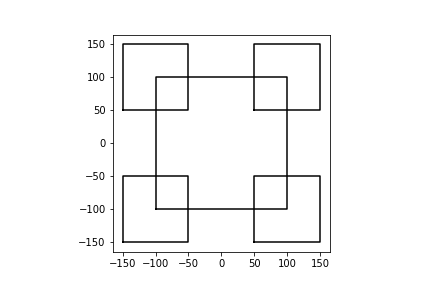
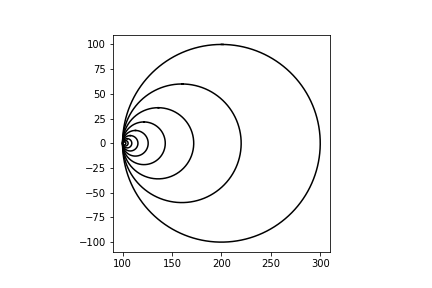
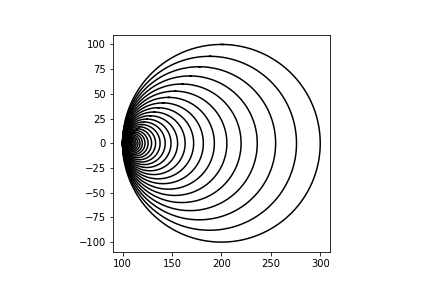
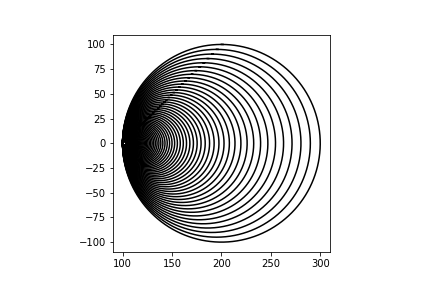
**Introduction:**

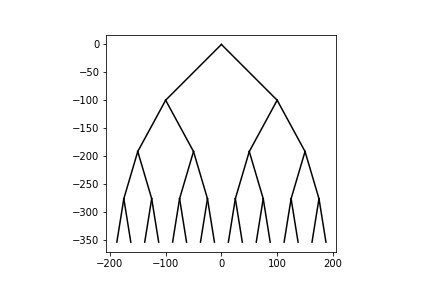
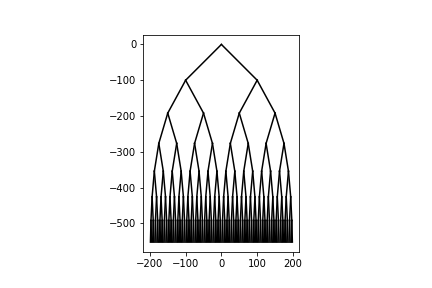
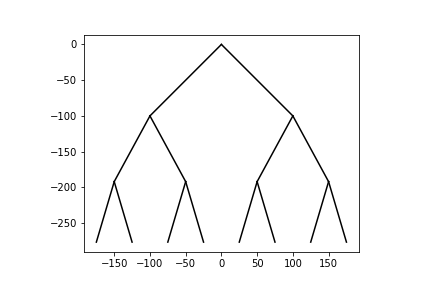
In python, there is a tool you can use called mathplotlib to plot different figures using different methods. The problem in this lab is to plot four specific figures with the limitation that it must be using recursion. The first figure revolves a square with squares on its sides, a circle which calls smaller circles on the left side, a bracket that expands like a tree downwards, and another circle that summons more circles inside of itself. We must also save the images of the variations on this lab to prove that our programs can compile.

**Proposed solution design and implementation:**

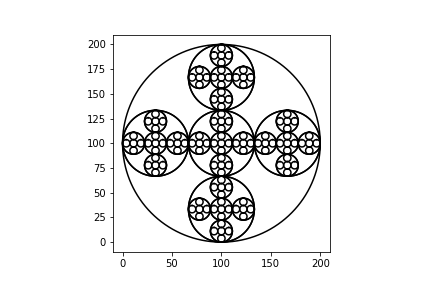
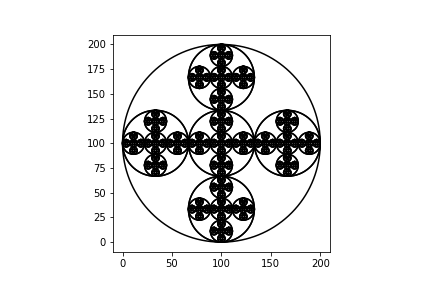
1. Squares.py: For the first part, I attempted to solve the problem by using different methods for each square, however that would be unnecessary as we are using recursion. My code would have to call the method 4 times in order to complete the four squares on each corner. I would have to make four recursion calls based on these four squares. I would have no user interface for this project, but I will have to call on a center and a radius value to help my program identify which square it is making.
2. Circles1.py: For the first part, I attempted to solve the problem by using different methods for each circle, however that would be unnecessary as we are using recursion. My code would have to call the method as necessary times in order to complete the squares the left corner. I would have to make a recursion call to make each necessary circle. I would have no user interface for this project, and I will use the necessary values given by the professor’s code.
3. Brackets.py: For the first part, I attempted to solve the problem by using different methods for each bracket, however that would be unnecessary as we are using recursion. My code would have to call the method 2 times in order to complete the left and right bracket on each corner. I would have to make two recursion calls based on each side. I would have no user interface for this project, but I will have to call on an x and y value to help my program identify which bracket it is making.
4. Circles2.py: For the first part, I attempted to solve the problem by using different methods for each circle, however that would be unnecessary as we are using recursion. My code would have to call the method 5 times in order to complete the five circles on the main one. I would have to make five recursion calls based on these five circles. I would have no user interface for this project, but I will have to call on a center and a radius value to help my program identify which circle it is making.

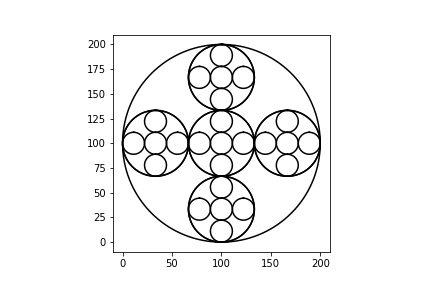
**Experimental results**:

1. Squares.py: For the first part, I had to figure out a way to draw the squares from the corners without altering the main square in the middle. To solve this, I introduced a couple dx and a dy values that would calculate the values of each corner squares. It goes to say that no matter how many times you called it recursively, the values start at the origin, which is the main square in the middle. I would then recursively call this method four times depending on which square it was drawing, with its respective values until it plotted the required squares needed to complete the complex figure. The results below show my results from my experiments.
2. Circles1.py: For the first part, I had to figure out a way to draw the circles from the cleft corner without altering the main circle in the middle. To solve this, I just multiplied the radius by the width every time each recursion call was made. The results below show my results from my experiments.
3. Brackets.py: For the first part, I had to figure out a way to plot the lines while splitting the path way into two without ever touching in the middle. To solve this, I introduced a couple of recursive calls for each part of the two branches. It goes to say that no matter how many times you called it recursively, the values start at the origin, which is the main point in the middle. I would then recursively call this method as many times necessary depending on how many branches it was drawing, with its respective values until it completed the complex figure. The results below show my results from my experiments.



1. Circles2.py: For the first part, I had to figure out a way to draw the five circles from the middle without altering the main circle in the middle. To solve this, I introduced five x and y values that would calculate the values of each respective circle. It goes to say that no matter how many times you called it recursively, the values start at the origin, which is the main circle in the outer rim. I would then recursively call this method five times depending on which circle it was drawing, with its respective values until it plotted the required circles needed to complete the complex figure. The results below show my results from my experiments.



**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including plotting different figures using the mathplotlib library available for Python. I was able to learn to solve different problems by using recursion throughout my lab.

**Appendix :**

**Squares.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: February 8, 2019

Instructor: Olac Fuentes

Assingment: Lab 1 Recursion

TA: Anindita Nath & Maliheh Zaragan

Purpose: To practice using recursion amd to identify the process of ploting figures using

pythons libraries

Part 1 of 4

Nested Squares

'''

#Imports the matplotlibrary and numpy to plot the figures

import matplotlib.pyplot as plt

#Imports the numpy

import numpy as np

#Method that plots the squares to be drawn in the image.

def draw\_squares(ax,n,dx,dy,r):

if n>0:

#Sets a new x and y from the top to be drawn in the figure from the top

x1 = dx-r

y1 = dy+r

#Sets a new x and y from the top to be drawn in the figure from the bottom

x2 = dx+r

y2 = dy-r

#Array that contains the values to be used to plot the squares

p = np.array([[x1, y2],[x1,y1],[x2,y1],[x2, y2],[x1, y2]])

#The call that actually plots the squares in the method

ax.plot(p[:,0],p[:,1],color='k')

#Recursively calls the method again to draw the remaining squares, starting from top left and ending on bottom left

draw\_squares(ax,n-1,x1,y1,r\*.5)

draw\_squares(ax,n-1,x2,y1,r\*.5)

draw\_squares(ax,n-1,x2,y2,r\*.5)

draw\_squares(ax,n-1,x1,y2,r\*.5)

#Closes the window where the figure is drawn

plt.close("all")

fig, ax = plt.subplots()

#Calls the method draw\_squares, and sets the values to draw the figure.

draw\_squares(ax,4,0,0,100)

#Sets the aspect ratio to be shown of the figure at hand

ax.set\_aspect(1.0)

#Turns on or off the measurements of the axis in the figure

ax.axis('on')

#Displays the figure on the terminal

plt.show()

#Saves an image as PNG of the figure you plotted

fig.savefig('squares.png')

'''

For figure a): Change n to 2

For figure b): Change n to 3

For figure c): Change n to 4

'''

**Circles1.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: February 8, 2019

Instructor: Olac Fuentes

Assingment: Lab 1 Recursion

TA: Anindita Nath & Maliheh Zaragan

Purpose: To practice using recursion amd to identify the process of ploting figures using

pythons libraries

Part 2 of 4

Nested Circles #1

'''

#Imports the matplotlibrary and numpy to plot the figures

import matplotlib.pyplot as plt

#Imports the numpy

import numpy as np

#Imports the math module

import math

#Method that caclulates the center and the radius of the specififc circle. Later sends the information back to draw\_circles

def circle(center,rad):

#Calculates the shape of the circle

n = int(4\*rad\*math.pi)

#Returns Return evenly spaced numbers over the specified interval.

t = np.linspace(0,6.3,n)

x = center[0]+rad\*np.sin(t)

y = center[1]+rad\*np.cos(t)

return x,y

#Method that plots the circles to be drawn in the image.

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

#Draws depending on the number of circles needed

if n>0:

#Finds x and y by sending the canter and the radius to the Method Circle, where it will calculate the figure itself.

x,y = circle(center,radius)

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

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

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

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

#Closes the window where the figure is drawn

plt.close("all")

fig, ax = plt.subplots()

#Calls the method draw\_circles, and sets the values to draw the figure

draw\_circles(ax, 80, [100,0], 100,.95)

#Sets the aspect ratio to be shown of the figure at hand

ax.set\_aspect(1.0)

#Turns on or off the measurements of the axis in the figure

ax.axis('on')

#Displays the figure on the terminal

plt.show()

#Saves an image as PNG of the figure you plotted

fig.savefig('circles.png')

'''

For figure a): Change n to 9, and w to .6

For figure b): Change n to 40, and w to .88

For figure c): Change n to 80, and w to .95

'''

**Brackets.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: February 8, 2019

Instructor: Olac Fuentes

Assingment: Lab 1 Recursion

TA: Anindita Nath & Maliheh Zaragan

Purpose: To practice using recursion amd to identify the process of ploting figures using

pythons libraries

Part 3 of 4

Nested Bracket

'''

#Imports the numpy

import numpy as np

#Imports the matplotlibrary and numpy to plot the figures

import matplotlib.pyplot as plt

#Method that plots the tree to be drawn in the image.

def draw\_branch(ax,n,c,x,y):

if c>0:

#Splits the call into two sections, negative and positive x, where it will plot on two different sides of the graph

ax.plot([n[0],n[0]-x],[n[1],n[1]-y], color='k')

#Recursively calls it twice depending on the side of the branch

draw\_branch(ax,[n[0]-x,n[1]-y],c-1,x/2,y\*0.92)

#Splits the call into two sections, negative and positive x, where it will plot on two different sides of the graph

ax.plot([n[0],n[0]+x],[n[1],n[1]-y], color='k')

#Recursively calls it twice depending on the side of the branch

draw\_branch(ax,[n[0]+x,n[1]-y],c-1,x/2,y\*0.92)

#Closes the window where the figure is drawn

plt.close("all")

fig, ax = plt.subplots()

n=np.array([0,0])

#Calls the method draw\_branch, and sets the values to draw the tree

draw\_branch(ax, n , 4, 100, 100)

#Sets the aspect ratio to be shown of the figure at hand

ax.set\_aspect(1.0)

#Turns on or off the measurements of the axis in the figure

ax.axis('on')

#Displays the figure on the terminal

plt.show()

#Saves an image as PNG of the figure you plotted

fig.savefig('bracket.png')

'''

For figure a): Change c to 3

For figure b): Change c to 4

For figure c): Change c to 7

'''

**Circles2.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: February 8, 2019

Instructor: Olac Fuentes

Assingment: Lab 1 Recursion

TA: Anindita Nath & Maliheh Zaragan

Purpose: To practice using recursion amd to identify the process of ploting figures using

pythons libraries

Part 4 of 4

Nested Circles #2

'''

#Imports the matplotlibrary and numpy to plot the figures

import matplotlib.pyplot as plt

#Imports the numpy

import numpy as np

#Imports the math module

import math

#Method that caclulates the center and the radius of the specififc circle. Later sends the information back to draw\_circles

def circle(center,rad):

#Calculates the shape of the circle

n = int(4\*rad\*math.pi)

#Returns Return evenly spaced numbers over the specified interval.

t = np.linspace(0,6.3,n)

x = center[0]+rad\*np.sin(t)

y = center[1]+rad\*np.cos(t)

return x,y

#Method that plots the circles to be drawn in the image.

def draw\_circles\_2(ax,n,center,radius):

#Draws depending on the number of circles needed

if n>0:

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x,y = circle(center,radius)

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x1,y1 = circle(center,radius/3)

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x2,y2 = circle([center[0]+radius-radius/3,center[1]],radius/3)

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x3,y3 = circle([center[0]-radius+radius/3,center[1]],radius/3)

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x4,y4 = circle([center[0],center[1]+radius-radius/3],radius/3)

#Finds x and y by sending the center and the radius to the Method Circle, where it will calculate the figure itself.

x5,y5 = circle([center[0],center[1]-radius+radius/3],radius/3)

#Plots the main circle from the data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

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

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

ax.plot(x1, y1,color='k')

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

ax.plot(x2, y2,color='k')

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

ax.plot(x3, y3,color='k')

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

ax.plot(x4, y4,color='k')

#Plots the circle from data found from the "Circle" method. To give its unique shape, I added the value of the radius to x so that it

ax.plot(x5, y5,color='k')

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

draw\_circles\_2(ax,n-1,center,radius/3)

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

draw\_circles\_2(ax,n-1,[center[0]+radius-radius/3,center[1]],radius/3)

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

draw\_circles\_2(ax,n-1,[center[0]-radius+radius/3,center[1]],radius/3)

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

draw\_circles\_2(ax,n-1,[center[0],center[1]+radius-radius/3],radius/3)

#Recursively calls itself again with a radius times the width. This will call itself over and over until the counter n reaches 0.

draw\_circles\_2(ax,n-1,[center[0],center[1]-radius+radius/3],radius/3)

#Closes the window where the figure is drawn

plt.close("all")

fig, ax = plt.subplots()

#Calls the method draw\_circles2, and sets the values to draw the figure. I removed w or width as it is not necessary in the code to calculate it in this program

draw\_circles\_2(ax, 2, [100,100], 100)

#Sets the aspect ratio to be shown of the figure at hand

ax.set\_aspect(1.0)

#Turns on or off the measurements of the axis in the figure

ax.axis('on')

#Displays the figure on the terminal

plt.show()

#Saves an image as PNG of the figure you plotted

fig.savefig('circles2.png')

'''

For figure a): Change n to 2

For figure b): Change n to 3

For figure c): Change n to 4

'''