**Draw squares**

For the file draw squares, I was required to create a python program that will draw squares in a certain figure with recursion only. In this lab I am prohibited the use of python turtle to draw since the purpose of this lab is to enhance my skills in recursion with python. I was provided a program that draws the squares recursively but now I had to either change or expand on the program to be able to create the figure below.



In order to be able to start I first had to understand how the original program work which required me to ask various questions to my Instructional Assistant (IA) which I understood better now and by using the information provided by the class. When I began to program, I knew that I needed to create my new squares on each corner of the original square, however the only way to do that was to find the middle point of each dimension of the original square and make the new squares be created on that middle point that was found.

**Proposed solution and my Approaches**

The first way I began to approach was to see if there was a way that I could modify the existing code that I was given to create the new squares on each corner of the original square, but it proved to not be enough since the most I could do was draw one new square in the corner and the rest would not be drawn on the remaining corners. I had many difficulties doing it that way since my computer or the python console would crash sometimes if I made a mistake with my code, but it would only happen about 4 times due to various unsuccessful runs.

The second approach I had was to create an entire new method in order to create brand new squares. The method I created is called **new\_squares**, and it takes the parameters of **ax, n, x, y, and w**. These parameters are needed because x and y will always be declared as 0 initially in order to perform its function to find the middle point for each side of the square. The purpose of n is the number of squares you wish to have which will be also used as the amount of times we recursively traverse through the method. The purpose of w is to store the initial midpoint of the figure which is 500 since we can assume that it will always be half of the figure and ax is used to be able to plot the squares.

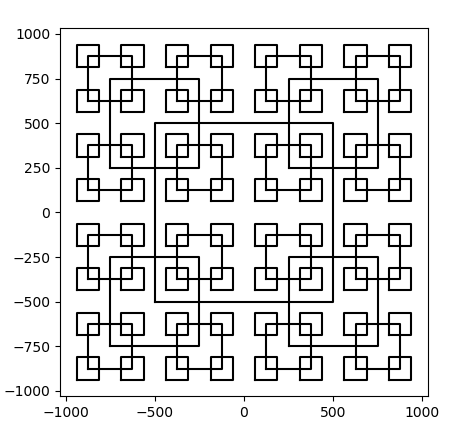
My proposed solution was my second approach to the code, because the only way I could be able to have the squares be drawn in each corner of the first original square, is to find the midpoints of each dimension in order words I needed the dimensions of the left side, right side, top, and bottom of the original square. The reason I need the out points is to be able to plot the new square, but those points must be required to be in a list in order to connect each point.

**Tracing**

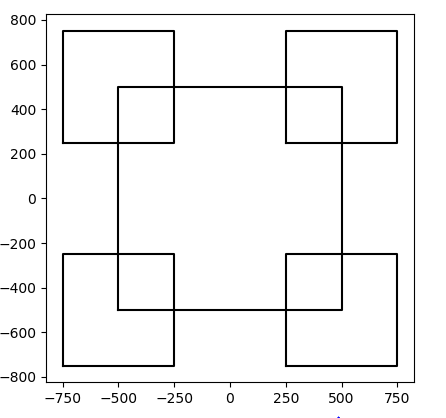
Now when I begin to trace the method, I will call the method **new\_squares** and pass these variables new\_squares(ax,4,0,0,midpoint) which essentially say that 4 is the number of squares we want to draw, x and y are 0, and the midpoint is equal to 500. Now when we go through the method, we see the base case which **if n>0** then we go though the line of code that is inside the if statement and if our n is less than 0 then we wouldn’t go through the method, but in this case, we actually go through the method since our **n = 4** and is greater than 0.

Inside the method we find the variables **left, right, top, and bottom** which these are the variables that we will store the points where we draw our new squares. Tracing the code, we see that left is equal to x minus w **(left = x – w),** but in this instance our x is 0 and w is 500, so it would look like 0-500 = -500 which we now found the new left point that draws the new square. If you follow the next equations you will find that right is 500, top is 500, and bottom is -500 which the points that the program found will be used to create the square. The next line of code declares that **p = np.array([[left,bottom],[left,top],[right,top],[right,bottom],[left,bottom]])**; what this does is that we create a list called p that holds the coordinates for our square and a way to see it is that the left connects to the bottom, left connects to top, right connects to top, right connects to bottom, and left connects to bottom to close the square. Once the program knows how to squares should be connected then we proceed to ax.plot(p[:, 0],p[:,1],color='k'). What this does is the only way that we can actually draw/plot our squares

Now since we drew our square first square then its time to recursively go through the method till our base case doesn’t let us proceed any more, however when we call our method recursively we actually call it 4 times with different parameters in our x and y variables how I previously mentioned, new\_squares(ax,n-1,left,bottom,w/2) will recursively call the method with the new dimensions and the new midpoint and it will do the same for the next 3 recursive call methods.

Here is my output if n = 4 or if you want to draw 4 squares

Here is my output for n =2



Here is my output for n =3

