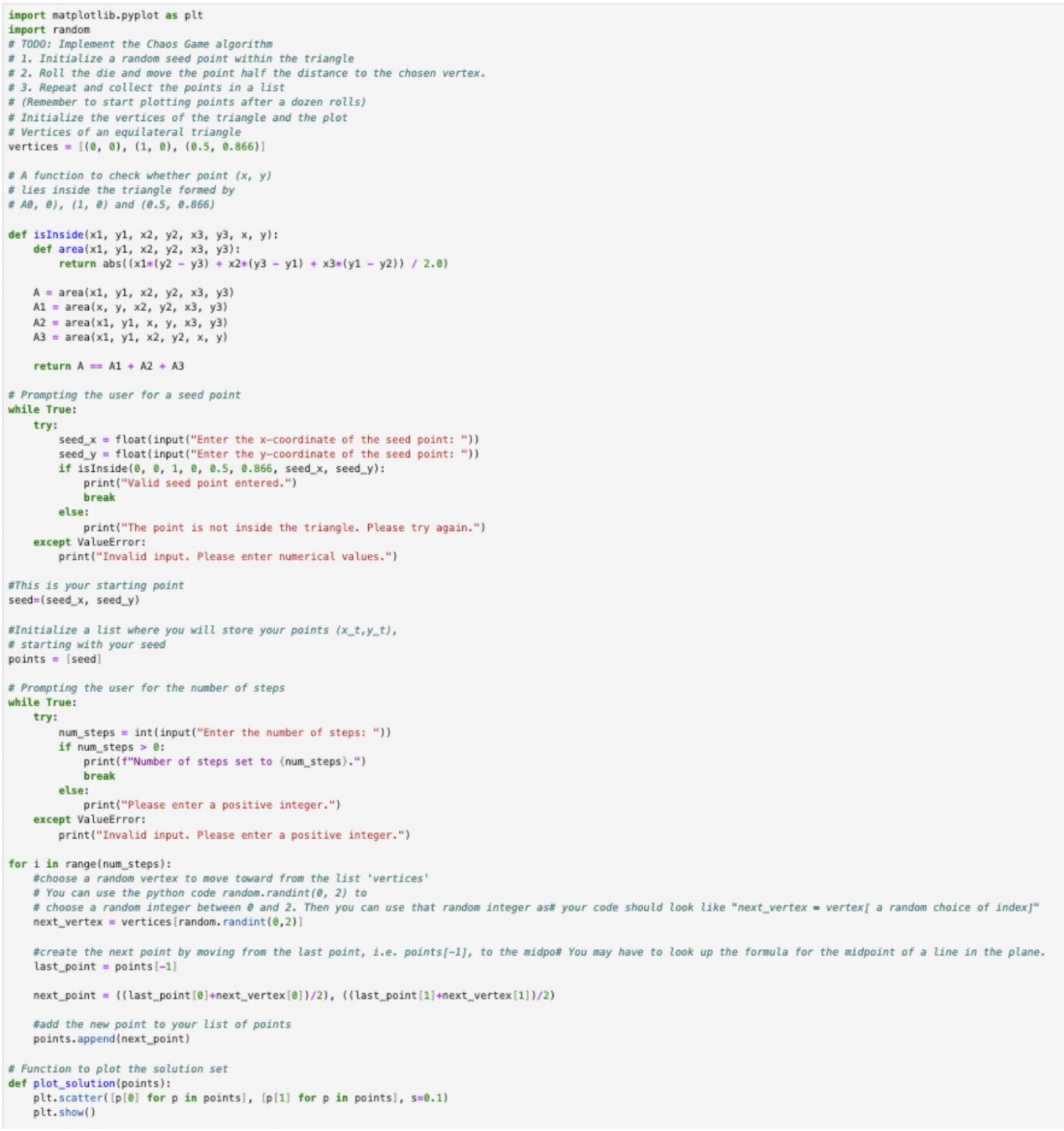
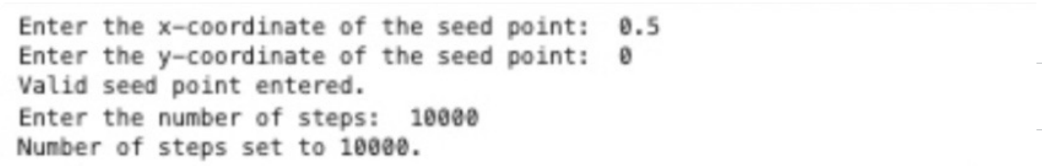
4)

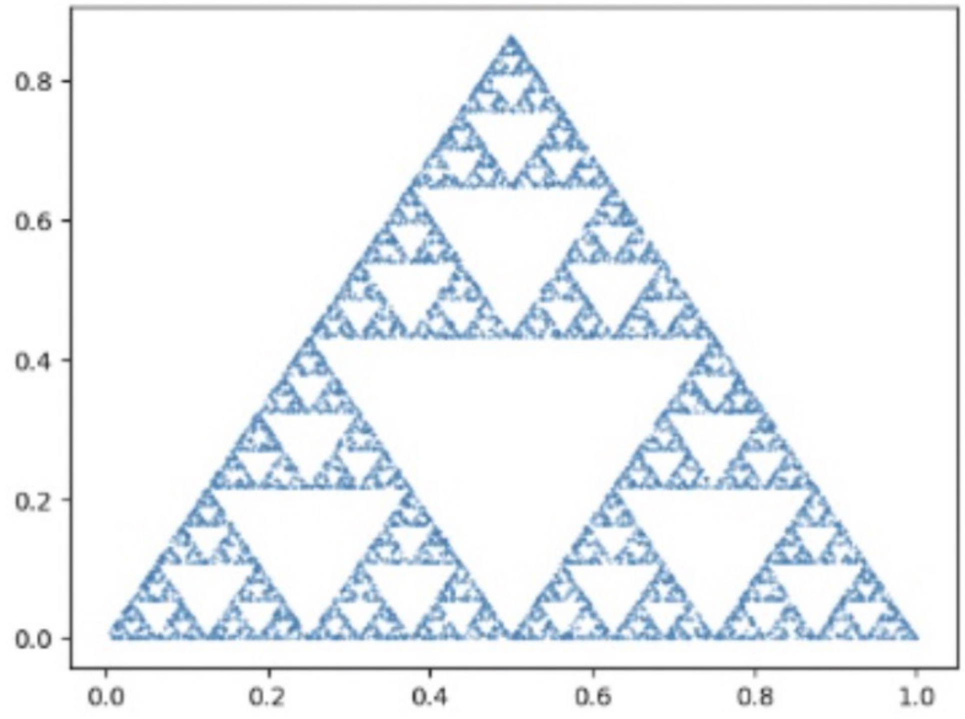


```python
import matplotlib.pyplot as plt
import random
# TODO: Implement the Chaos Game algorithm
# 1. Initialize a random seed point within the triangle
# 2. Roll the die and move the point half the distance to the chosen vertex.
# 3. Repeat and collect the points in a list
# (Remember to start plotting points after a dozen rolls)
# Initialize the vertices of the triangle and the plot
# Vertices of an equilateral triangle
vertices = [(0, 0), (1, 0), (0.5, 0.866)]

# A function to check whether point (x, y)
# lies inside the triangle formed by
# A0, 0), (1, 0) and (0.5, 0.866)

def isInside(x1, y1, x2, y2, x3, y3, x, y):
    def area(x1, y1, x2, y2, x3, y3):
        return abs((x1*(y2 - y3) + x2*(y3 - y1) + x3*(y1 - y2)) / 2.0)

    A = area(x1, y1, x2, y2, x3, y3)
    A1 = area(x, y, x2, y2, x3, y3)
    A2 = area(x1, y1, x, y, x3, y3)
    A3 = area(x1, y1, x2, y2, x, y)

    return A == A1 + A2 + A3

# Prompting the user for a seed point
while True:
    try:
        seed_x = float(input("Enter the x-coordinate of the seed point: "))
        seed_y = float(input("Enter the y-coordinate of the seed point: "))
        if isInside(0, 0, 1, 0, 0.5, 0.866, seed_x, seed_y):
            print("Valid seed point entered.")
            break
        else:
            print("The point is not inside the triangle. Please try again.")
    except ValueError:
        print("Invalid input. Please enter numerical values.")

#This is your starting point
seed=(seed_x, seed_y)

#Initialize a list where you will store your points (x_t,y_t),
# starting with your seed
points = [seed]

# Prompting the user for the number of steps
while True:
    try:
        num_steps = int(input("Enter the number of steps: "))
        if num_steps > 0:
            print(f"Number of steps set to {num_steps}.")
            break
        else:
            print("Please enter a positive integer.")
    except ValueError:
        print("Invalid input. Please enter a positive integer.")

for i in range(num_steps):
    #choose a random vertex to move toward from the list 'vertices'
    # You can use the python code random.randint(0, 2) to
    # choose a random integer between 0 and 2. Then you can use that random integer as# your code should look like "next_vertex = vertex[ a random choice of index]"
    next_vertex = vertices[random.randint(0,2)]

    #create the next point by moving from the last point, i.e. points[-1], to the midpo# You may have to look up the formula for the midpoint of a line in the plane.
    last_point = points[-1]

    next_point = ((last_point[0]+next_vertex[0])/2), ((last_point[1]+next_vertex[1])/2)

    #add the new point to your list of points
    points.append(next_point)

# Function to plot the solution set
def plot_solution(points):
    plt.scatter([p[0] for p in points], [p[1] for p in points], s=0.1)
    plt.show()
```

Result From code above

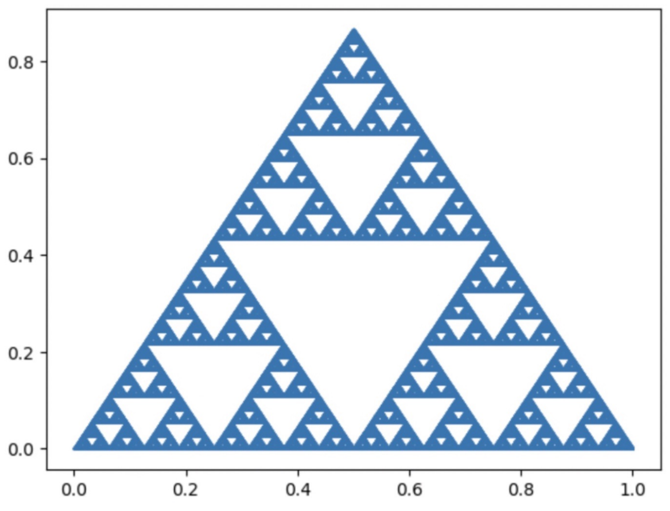


```
Enter the x-coordinate of the seed point:  0.5
Enter the y-coordinate of the seed point:  0
Valid seed point entered.
Enter the number of steps:  10000
Number of steps set to 10000.
```

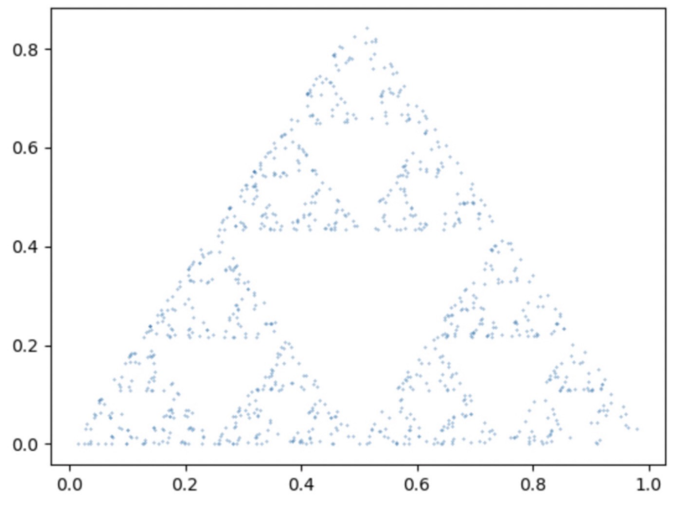


This code shows the number of steps as 10000. The player in this game would select a point at random of the triangle and then roll a dice to select a vertex to move towards. For such code the dice is replaced by generating a random integer. When the number of steps is larger than what we currently recorded for we will have a clearer image and if the number of steps is lower the image will be blurrier.

n is greater at 1,000,000



n is less at 1000



In my view, the fusion of my code with Greg's results in a solution that is remarkably simple and elegant, showcasing the implementation of fractals and geometric patterns. The awe-inspiring patterns generated by randomly selecting a set of values underscore the significance of acquiring coding skills and the myriad possibilities that accompany such proficiency. It's worth noting that the generation of patterns isn't exclusive to this particular code; any algorithm consistently applying the same equation to a given set of values over time will yield a pattern unique to the specific code employed.