SHARIF UNIVERSITY OF TECHNOLOGY
PHYSICS DEPARTMENT

Problem Set 1

Student name: *Ali Fayyaz - 98100967*

Course: *Computational Physcis - (Spring 2023)*
Due date: *February 17, 2023*

Exercise 2.1

Create the Koch Snowflake fractals.

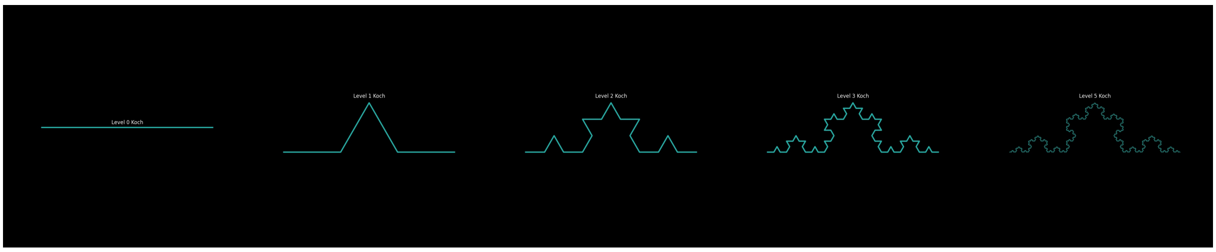
Answer. The approach taken to solve the problem was to utilize linear transformations on points on a plot. Taking a line and iteratively applying the correct linear transformations on that line, the Koch Snowflake fractal can be generated. More specifically, the following two functions were declared; the first one applies linear transformations, the second iteratively applies the first on a list of points:

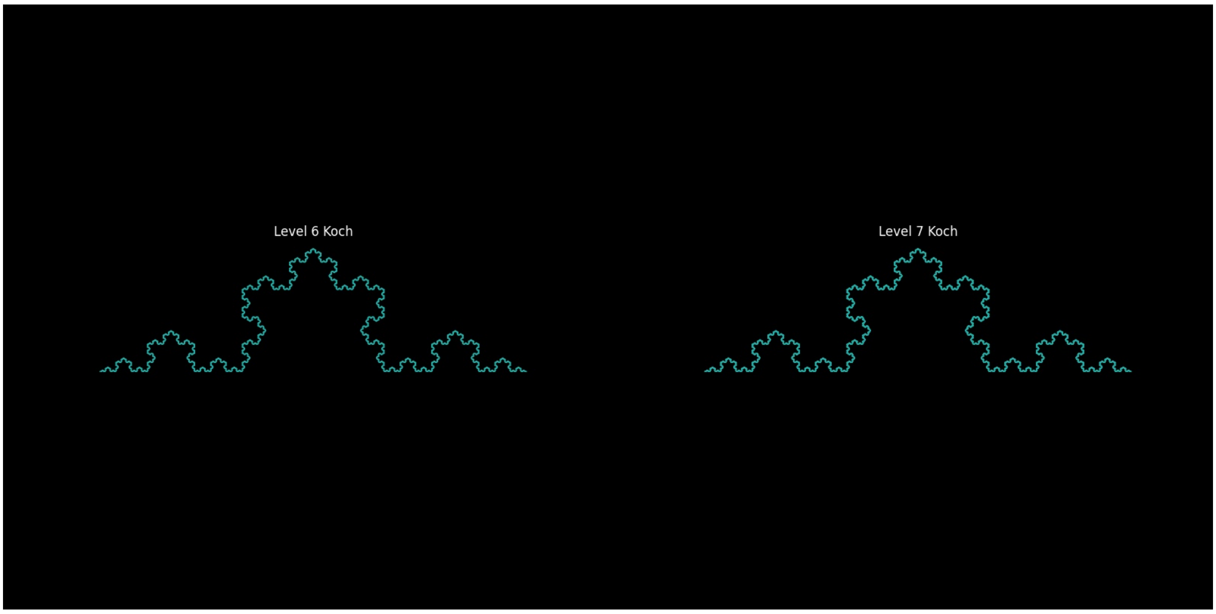
- **koch_grow():**

  The coordinates of two points on an x-y plane are taken, as the start and end points of a line, and the coordinates of three Koch points in-between are generated, using basic linear transformations on the said start and end points. Then the x components and the y components of all the 5 points are put into two lists and returned.

- **draw_koch():**

  This function takes a number as the desired level of the Koch fractal to be generated, repatedly applies **koch_grow()** on two pre-defined points on the x-y plane, updates a list of points untill the desired level of the fractal is reached, and finally plots the points. The algorithm is not recursive, rather a list is extended in each iteration.





Exercise 2.2

Generate the Heighway Dragon fractal. See if you can use two colors to illustrate the fact that the lines never cross each other.

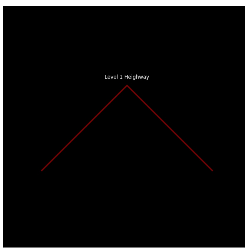
Answer. The approach taken to generate this fractal was quite identical to **Exercise 2.1** and the only difference was the adjustments made on the linear transformations to comply to the new shapes.

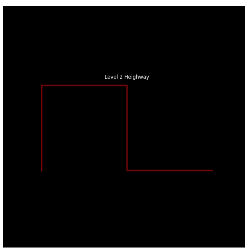
- **heighway_grow():**

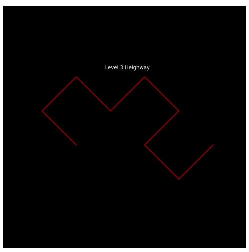
  The coordinates of two points on an x-y plane are taken, as the start and end points of a line, and the coordinates of the corresponding Heighway middle point is generated, using basic linear transformations on the said start and end points. Then the x components and the y components of all the 3 points are put into two lists and returned.

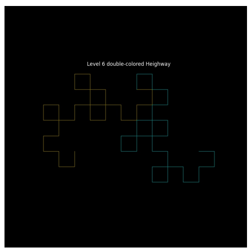
- **draw_heighway():**

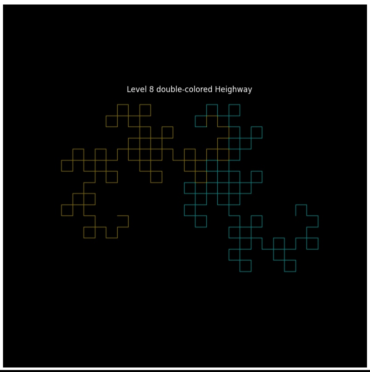
  This function takes a number as the desired level of the Heighway fractal to be generated, repatedly applies **Heighway_grow()** on two pre-defined points on the x-y plane, updates a list of points untill the desired level of the fractal is reached, and finally plots the points. By default, a double-colored shape will be plotted, though the user can turn this off at will. The algorithm is not recursive, rather a list is extended in each iteration.

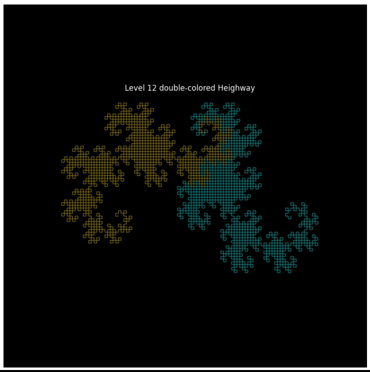


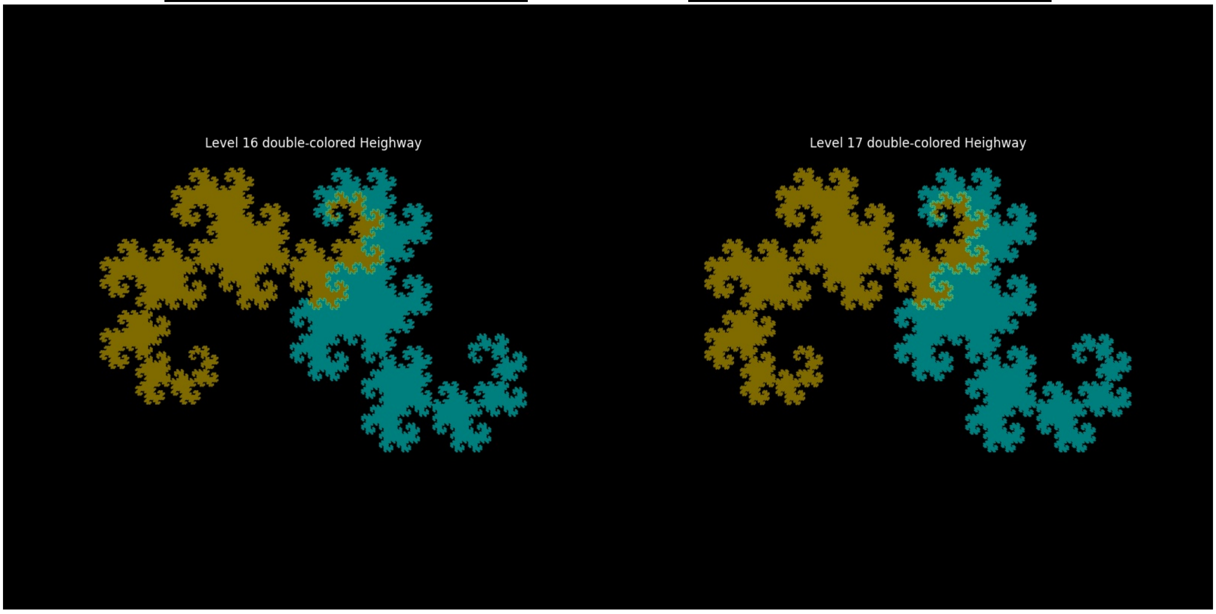












Exercise 2.3

Start with plotting a triangle. Generate the Sierpiński fractal using self-similar transformations on this triangle.

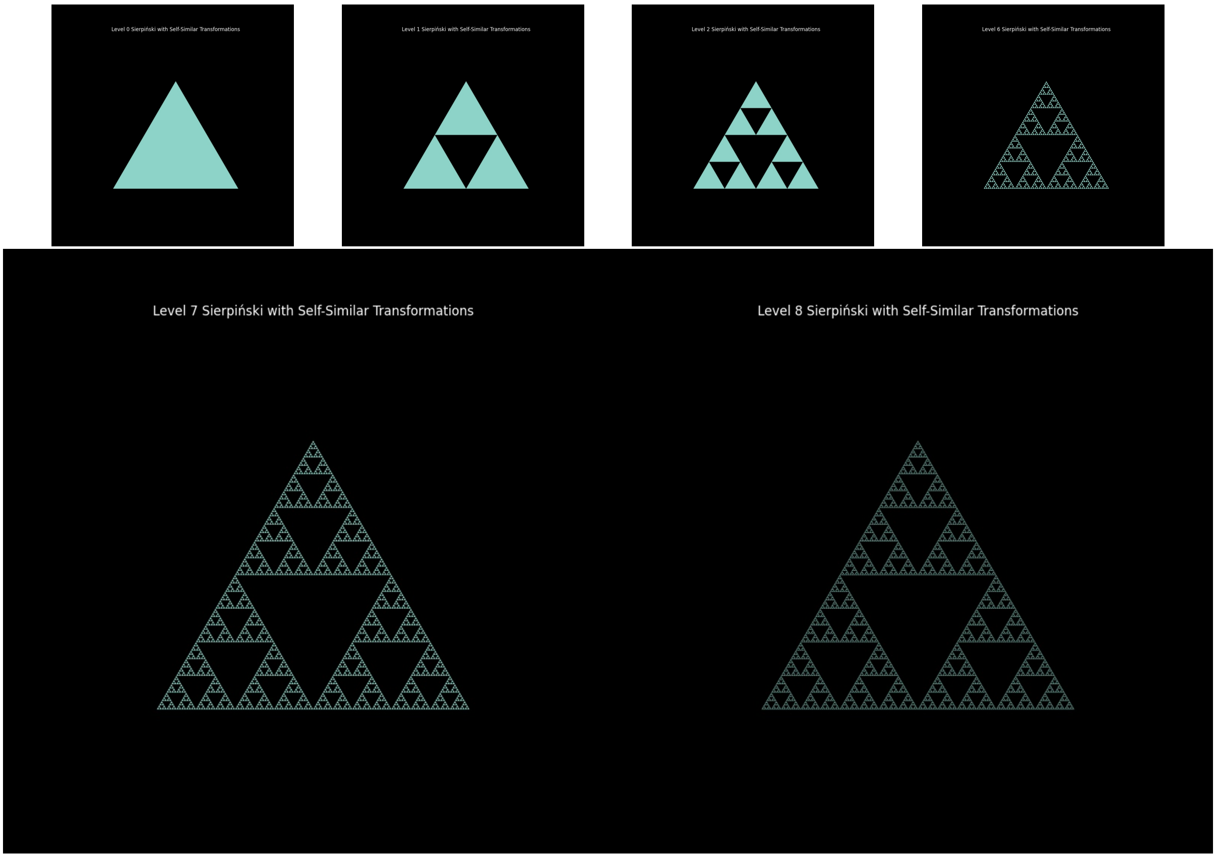
Answer. A *matplotlib.patches.Polygon* object is used to create a first triangle. Three scaling functions corresponding to the ones described in section 2.1 of the textbook are defined:

- **f1()**, **f2()** and **f3()**:

  Each takes a triangular shaped, Polygon object, and scales it to half; the resultant triangle is then pushed to the lower left corner, lower right corner and the upper corner, respectivley.

- **recursive_deterministic_sierpinski():**

  This function takes a number for the desired level of the Sierpiński triangle, and as the name suggests, recursively applies the 3 functions explained above on a triangular Polygon object, and finally draws the resultant shape using the *matplotlib.axes.Axes.add_artist()* method.



Exercise 2.4

Use the Khayyam-Pascal triangle and generate the Sierpiński fractal by coloring the odd numbers green and the even numbers red.

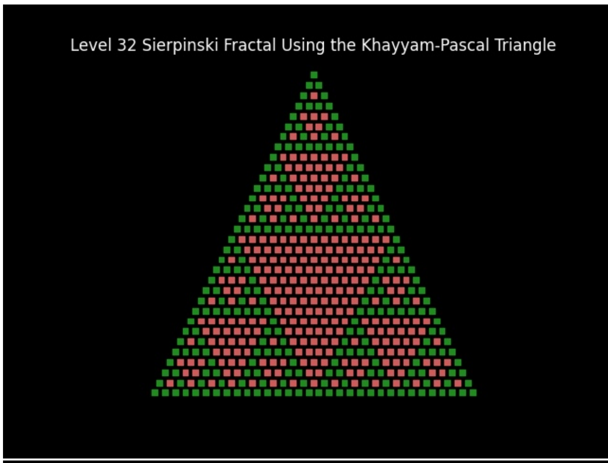
Answer. An algorithm was adapted from *stackoverflow.com* for generating a list of lists, each containing a row of the Khayyam-Pascal numbers. The two following functions were then defined:

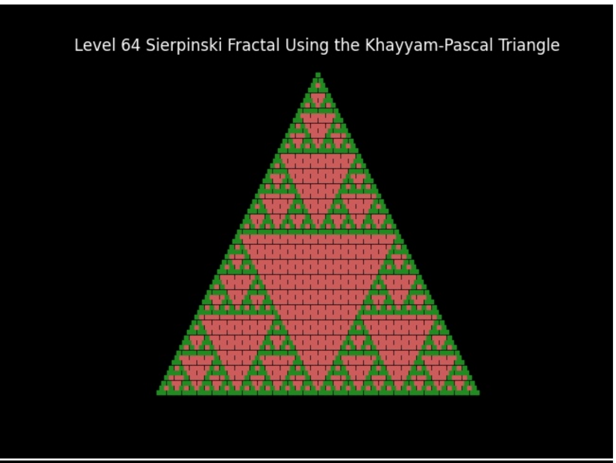
- **KP_data_generator()**:

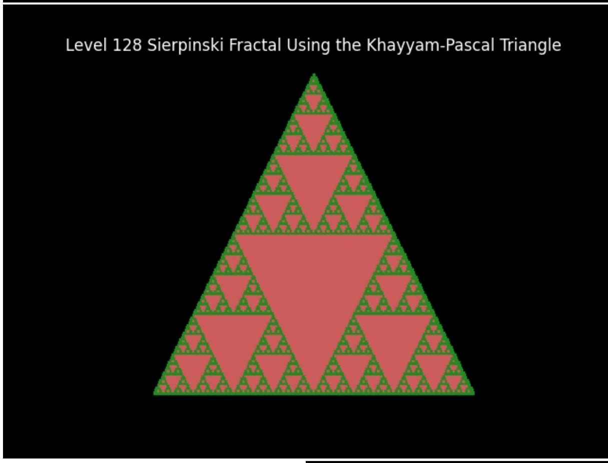
  Basically returns a list of lists, each of which contains the numbers of the Khayyam-Pascal triangle for the corresponding row. The function was adapted from stackoverflow.com.

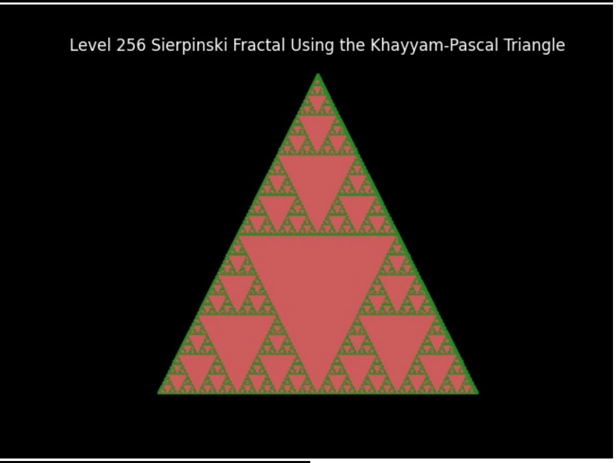
- **draw_sierpinski()**:

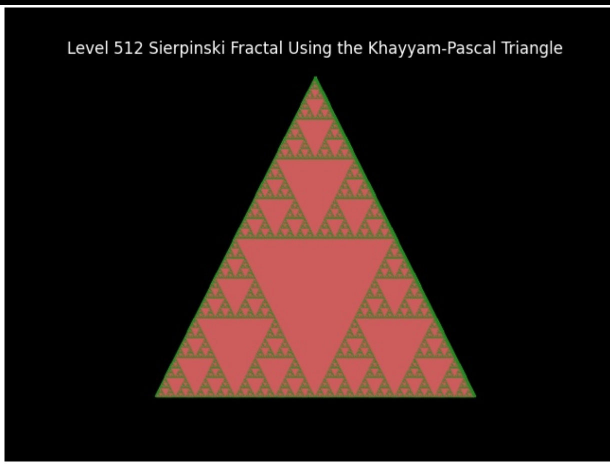
  This function takes the desired number of rows, generates the data with **KP_data_generator()** and then uses a for-loop to plot the data, row at a time, starting from the last row up. Using boolean indexing, each row is seperated to even and odd numbers, and color-coded using *numpy.where()*. In each iteration of the for-loop, *matplotlib.pyplot.scatter()* is called to add one row to the plot. The user can also specify the colors of the resulting figure. Notice that in order to have a full triangle, the number of rows has to be a power of 2.


Level 32 Sierpinski Fractal Using the Khayyam-Pascal Triangle


Level 64 Sierpinski Fractal Using the Khayyam-Pascal Triangle


Level 128 Sierpinski Fractal Using the Khayyam-Pascal Triangle


Level 256 Sierpinski Fractal Using the Khayyam-Pascal Triangle


Level 512 Sierpinski Fractal Using the Khayyam-Pascal Triangle