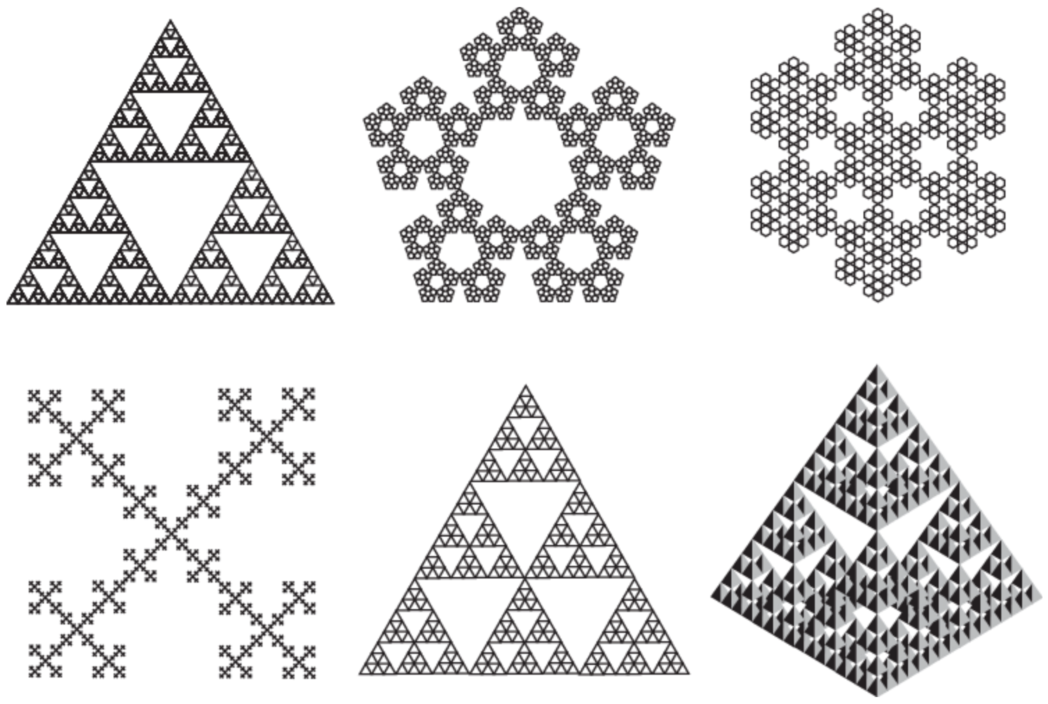
JEo-MC

A Geometric Programming Language

Composed by:

Jeremy Lu - jl5921, **Emma Schwartz** - es3962, **Melody Hsu** - mh4133, **Connie Zhou** - cz2382

JEo-MC (pronounced as jee-oh-mik) is a Python-inspired curve-drawing language that produces illustrations of fractals. The image shown below displays fractals that can be generated using our language.



Source: https://www.researchgate.net/publication/1756308_Martingale_dimensions_for_fractals

Use Cases for JEo-MC

JEo-MC can be used in a modular fashion to construct complex geometric structures including fractals, fundamentally based on user-defined curves. Programs written using JEo-MC are typically scripts that consist of a main recursive function, which controls the drawing portion of the base curve. Also, the program replicates the curve at user-specified orientations and scales. The output of these programs will be SVG (scalable vector graphics) files. In addition, programs written in JEo-MC may be used to write simulations of natural phenomena that can be modeled using fractals, such as various organic biological structures or microbial growth patterns, or physical phenomena like the formation of frost or precipitation of crystalline compounds.

Language Specifications

JEo-MC is intended to be a functional programming language, therefore we anticipate that programs written in JEo-MC will not have any classes or other structures common in object-oriented languages. The functions should be pure, and the data be immutable. Programs will generally be single-purpose, which will typically be to generate some graphical fractal image according to user-specified parameters. For code organization and syntax, we are aiming to follow some of Python's conventions. Notable features included are indentation importance and dynamic typing.

Basic operations:
We plan for JEo-MC to support primitive types such as int and floats. The primitive types will be used mainly for defining the equations of each curve we are drawing. In addition, basic arithmetic operators such as +, -, *, / will be supported as well.

Drawing:
For the actual drawing, we are planning to link to a drawing library (such as Turtle in Python) which will enable us to create visual representations of the curves. Turtle works by converting directional and magnitude inputs in the coordinate plane to trace the curve. After Turtle generates an interactive image, we save and return it at the end of the program.

Recursion:
Due to its highly repetitive nature, we want to use recursion in our language to draw fractals. Since this will be used often, implementing optimizations such as tail recursion may also be considered.

Source Code for Triangular Fractal Creation

First building a normal triangle of the base scale is the logic here. Next, we draw an inverted triangle. The inverted triangle will split the base triangle in 3s. Thereafter, recursion is used to fill each three of those spaces with inverted triangles, and so on until max iteration is reached.

```
max_iterations = 10

def build_triangular_fractal(position, scale, current_iter):
      if current_iteration <  max_iterations:
            if current_iteration == 1:
                  draw_triangle(position, false, scale)
                  # after initial upright triangle, we want a single inverted
      triangle next
                  # with quarter scale and the same centerpoint
                  build_triangular_fractal(position, scale * 0.25, current_iter
      +1)
            else:
                  draw_triangle(position, true, scale)

                  top = find_top_point(position, scale)
                  left = find_left_point(position, scale)
                  right = find_left_point(position, scale)
                  build_triangular_fractal(top, scale * 0.5, current_iter + 1)
                  build_triangular_fractal(left, scale * 0.5, current_iter + 1)
                  build_triangular_fractal(right, scale * 0.5, current_iter + 1)


# Draws a triangle of given scale with target position as the midpoint. Inversion
decides if triangle is inverted
def draw_triangle(position, inversion, scale):
      # triangle generator

# Finds the next position for the inverted triangle we will need to create at each
position
def find_top_point(position, scale):
      # finds the top point of the triangle
def find_left_point(position, scale):
      # finds the left point of the triangle
def find_right_point(position, scale):
      # finds right point of the triangle
```