CSCI 232: Data Structures: Project 3

Due Sunday, November 2 at 11:55 pm

Note there are two parts to the exercise in this assignment.

Problem Description

This assignment will require the implementation of a recursive algorithm for drawing a Koch snowflake. The Koch snowflake belongs to a class of self-similar patterns known as fractals. It is a triangular composition of the Koch curve, which is briefly discussed below.

The following section is supplemental to the textbook's section on Visualizing Recursion in Chapter 4. It is highly recommended that you read this section before attempting the assignment.

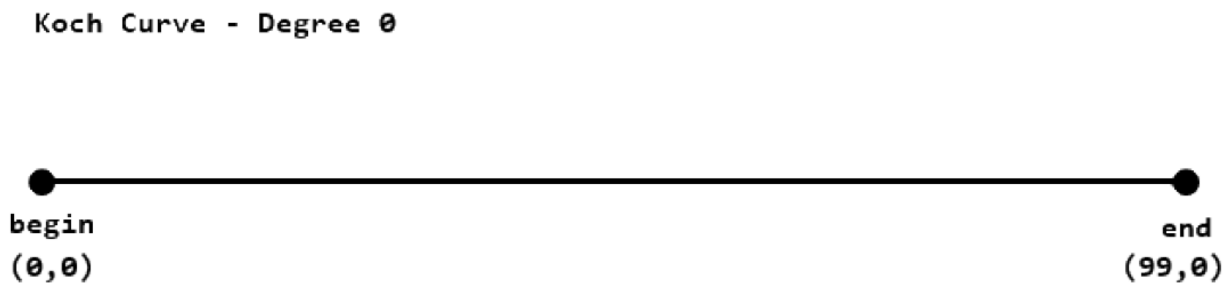
The Koch Curve

The Koch curve is a self-similar pattern that can be calculated using recursion. In this case, self-similar means that a portion of the curve (no matter how small) looks like the entire curve. As an algorithm, its basic steps can be described as:

1. Receive line segment(s) defined by a **begin** and **end** point.
2. Split line segment(s) into thirds by calculating points **a** and **b**.
3. Triangulate a fifth point **c** between **a** and **b**.
4. If possible, repeat the algorithm on new line segments connecting the five points: (**begin**, **a**), (**a**, **c**), (**c**, **b**), (**b**, **end**).

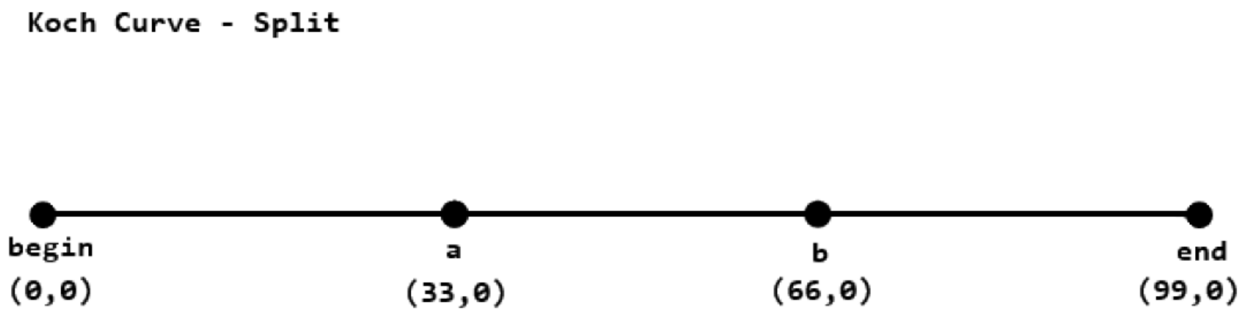
This algorithm takes a line segment and repeatedly splits it into smaller line segments. The algorithm can be recursed on the line segments it created during a previous execution. The ***degree*** of the algorithm is the number of times it has recursed. This algorithm could execute over an infinite number of degrees because it always produces more line segments than it consumes.

At **Degree 0**, a Koch curve is initially defined by a single line segment at any orientation. Consider the following line segment defined by two points:

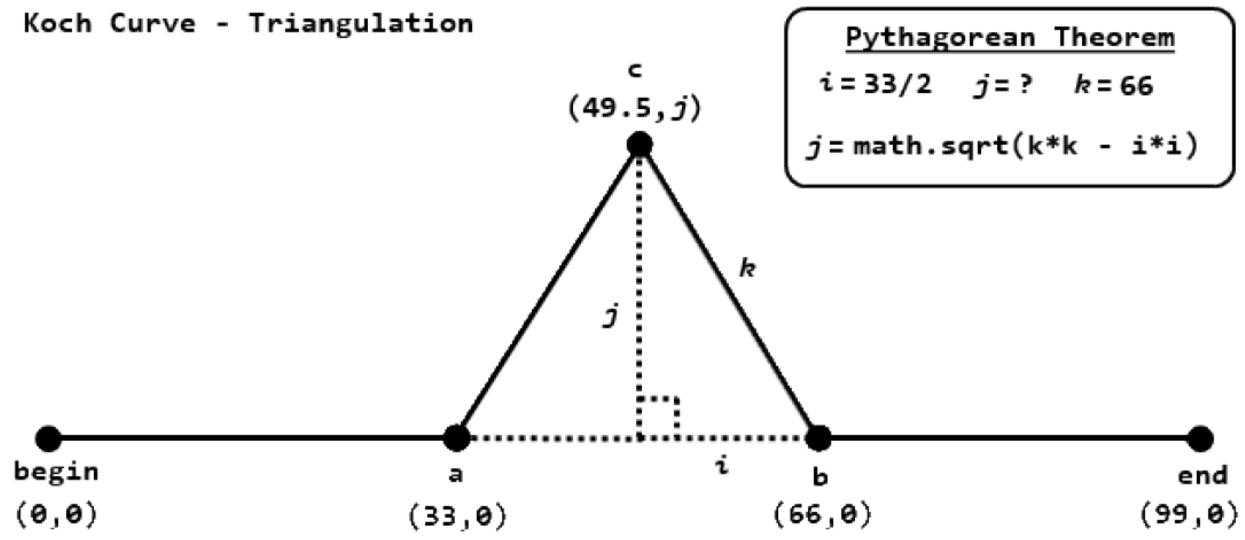


In this example, one point of this line segment has been placed on the origin and another has been placed 99 units away along the x-axis. In this degree, the algorithm does not execute its steps. Instead, it just outputs the line segment it has received.

At **Degree 1**, the line is split into three equal-length line segments. To do this, two new points are found between **begin** and **end**. The first point will be 1/3 the distance between **begin** and **end**. The second point will be 2/3 the distance between **begin** and **end**. The result of this splitting operation is shown below:



Two new points, **a** and **b**, have been calculated between **begin** and **end**. By introducing two new points, the singular line segment has become three line segments. Now, a third point **c** must be triangulated between **a** and **b**.

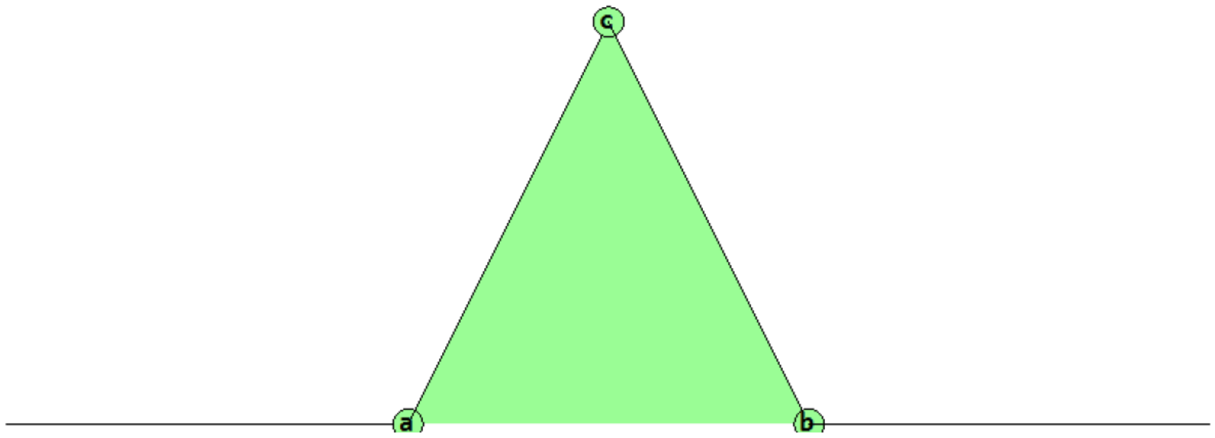


The new point **c** is inserted between points **a** and **b**. This creates new line segments between points **a** and **c** and points **c** and **b**. However, this destroys the line segment between points **a** and **b**. The point **c** is located at a distance ***j*** "above" and a distance ***i*** "between" the destroyed line segment. The first, ***j***, can be calculated via the Pythagorean Theorem. The second, ***i***, is just half the distance between **a** and **b**.

Example Output

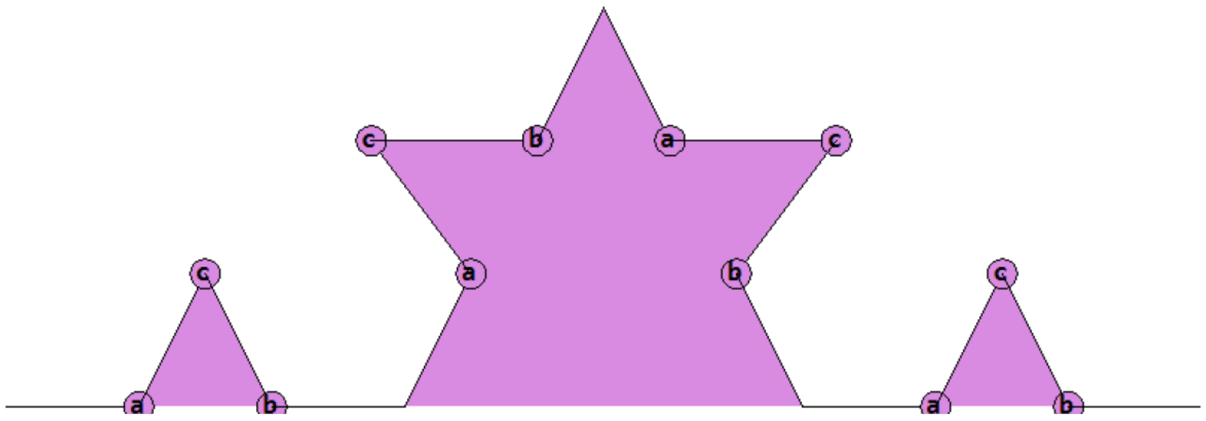
Here are some examples of how the Koch curve "expands" as its ***degree*** increases.

Degree 1:



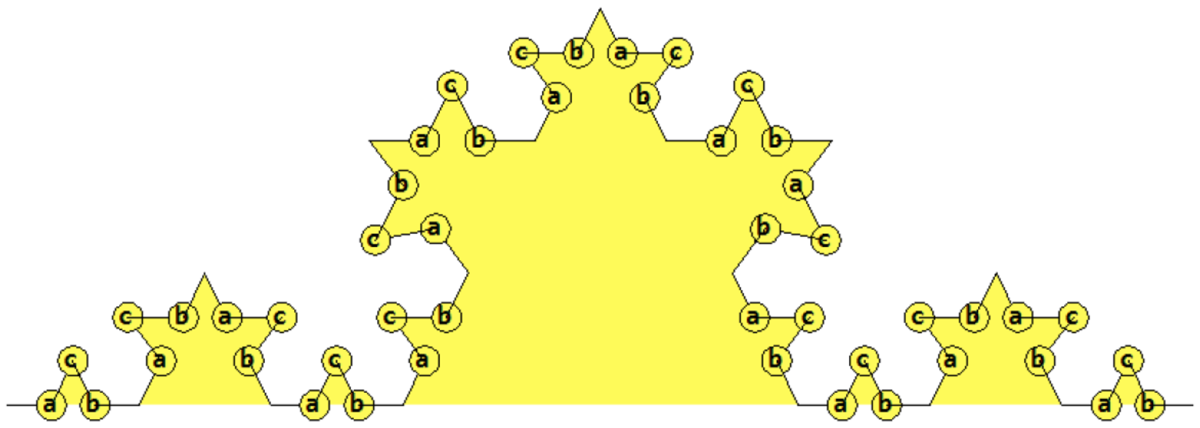
A single line segment has been "expanded" to include three new points: **a**, **b**, and **c**. With the endpoints of these lines, 4 line segments have been defined.

Degree 2:



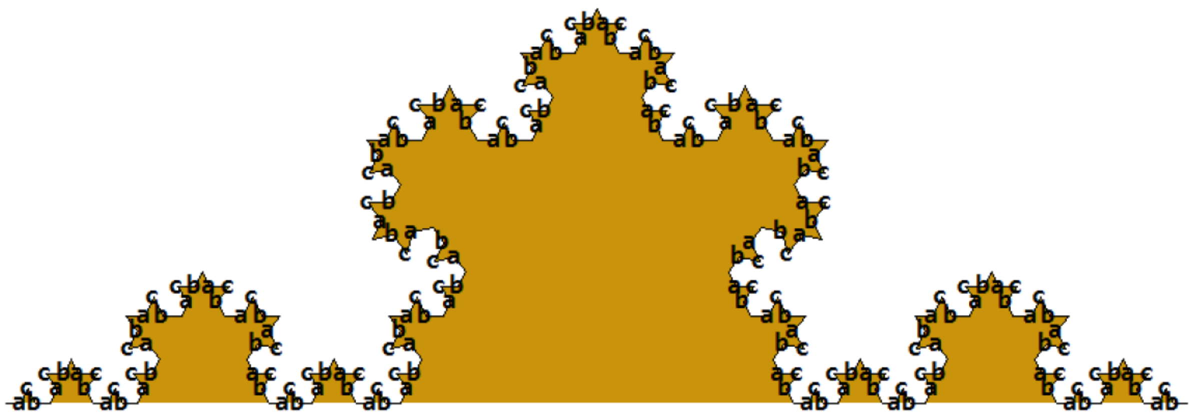
The 4 line segments from the previous iteration of the Koch curve are each "expanded" to include three new points. Including the endpoints, 16 line segments have been defined.

Degree 3:



The 16 line segments from the previous iteration of the Koch curve are each "expanded" to include three new points. Including the endpoints, 64 line segments have been defined.

Degree 4:



The 64 line segments from the previous iteration of the Koch curve are each "expanded" to include three new points. Including the endpoints, 256 line segments have been defined.

In general, the number of lines for any degree of Koch curve can be calculated by:

$$\text{num\_lines}_d = 4 * \text{num\_lines}_{d-1}$$
$$\text{num\_lines}_0 = 1$$

The first line indicates how the number of lines increases upon each degree of the Koch curve. The second line is a base case, where the degree of the Koch curve is zero.

Koch Snowflake – Part A

Two modules have been provided for this assignment. One is named **snowflake** and the other is named **point_helpers**. Each contains a series of function definitions that you must implement. Wherever implementation is required a special comment notation has been applied: #~~~~~~~~~~~~~~~~~~~~~~~~~~.

Open **point_helpers.py** and implement the function definitions for:

split(begin, end, n=2):
Split a single line segment (defined by **begin** and **end**) into **n+2** equal-length line segments. For this function, **n** is the number of equidistant points that should be calculated. The parameters **begin** and **end** are two-element lists. Their first element is an x-coordinate and their second element is a y-coordinate. This function should return **n** new equidistant points between **begin** and **end**.

dist(begin, end):
Find the distance between two points, **begin** and **end**. The distance between two points can be calculated by:

$$\text{distance} = \mathbf{math}.\text{sqrt}((\text{x1} - \text{x2})^2 + (\text{y1} - \text{y2})^2)$$

The **math** module is in the Python Standard Library. If you **import** this module, you can gain access to the `sqrt` function.

pythag(a=None,b=None,c=None)
The parameters **a** and **b** are the lengths of two sides of a right triangle. The parameter **c** is the length of the hypotenuse of a right triangle. At any one time, this function expects only two of its parameters to not equal **None**. The Pythagorean Theorem should be used to calculate the value of the only parameter whose value is equal to **None**.

Two other functions (**diff** and **triangulate**) are defined. The first finds the difference between x-coordinates AND the difference between y-coordinates. The second finds a point "between" and "above" a line segment between two points. You may use the **diff** function when implementing the **split** and **dist** functions.

For the sake of this assignment, all "points" will be represented by a list containing two elements. The first element (at index 0) is a point's x-coordinate. Its second element (at index 1) is a point's y-coordinate. Any sequential pair of these points can be considered the **begin** and **end** points of a line segment.

After implementing the functions in the **point_helpers module**, you will define the operations necessary to calculate the points of a Koch Curve using the materials in the **Problem Description** as a guide. Two recursive functions will be defined to perform these calculations: **koch** and **expand**.

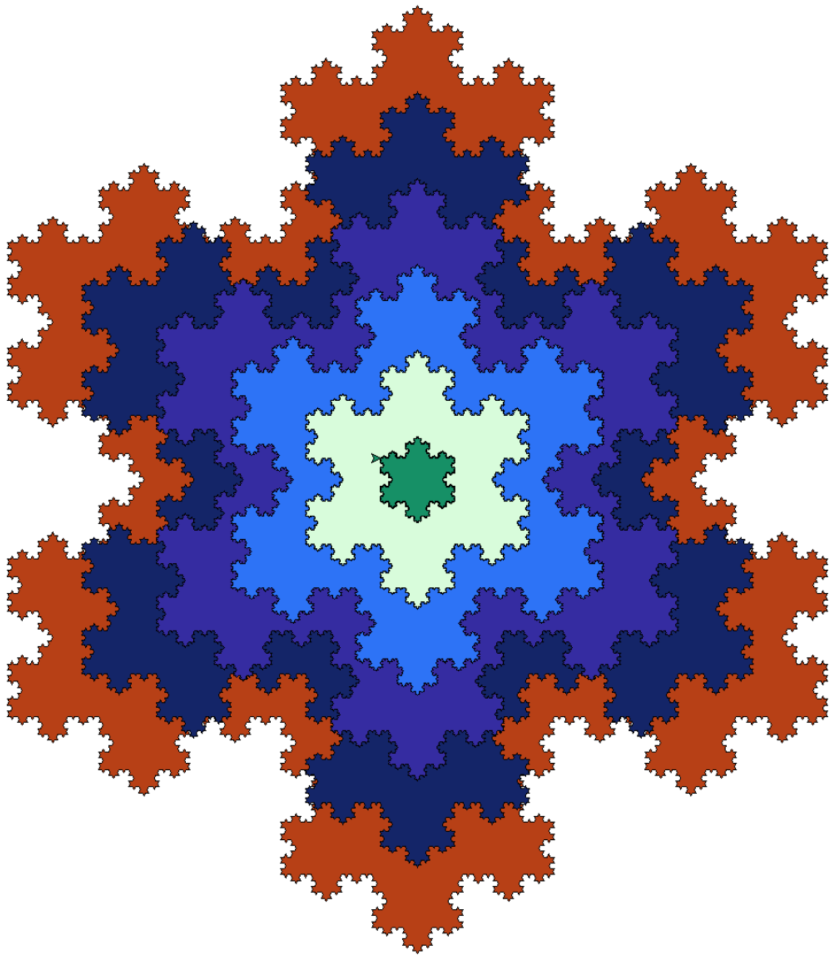
Open **snowflakes.py** and implement the function definitions for:

> **koch(points, degree):**
> The **koch** function accepts a list of **points** and the **degree** to which they should be "expanded." Its base case is when **degree** is equal to zero at which point it returns the **points** it receives. Otherwise, it "expands" the points and recurses on the expanded points by one less degree. See the textbook's implementation of the Sierpinksi Triangle for some intuition on how this can be performed.
>
> **expand(points):**
> The **expand** function accepts a list of **points** (representing a collection of connected line segments). Its base case is when there are only two points to consider. In this case, it splits the line segment into three equal-length line segments by finding points **a** and **b**. Then, it finds a triangulated point **c** between **a** and **b**. Finally, a list containing all five points **[begin,a,c,b,end]** is returned.
>
> Otherwise, it recurses on the first line segment and recurses on the remaining line segments. It stores the result of both recursions and returns a cohesive list of connected line segments.

At the conclusion of this exercise, a single Koch snowflake of (at least) degree 3 can be generated.

Koch Snowflake – Part B

Now that you can create a Koch Snowflake of varying degrees, create and save a new module named **nested_snowflake**. In this module, write an algorithm to draw six nested Koch snowflakes like those shown in the following figure:



Final Deliverable

The **deliverable** will be a compressed zip file containing the following files:

- `snowflake.py` **# Drawing of Koch snowflake.**
- `point_helpers.py` **# Calculations required by 2D points.**
- `nested_snowflake.py` **# Drawing of six nested snowflakes.**