Student Copy
AIEP Send-off Scratch 2 Session 8

RECURSION & ITERATED FUNCTION SYSTEM

In the qualifying phase of your AIEP Scratch 2 training, you were introduced to the concept of recursion and how it can be used to generate one of the earliest described fractals: the Koch snowflake. In this handout, we will be building on this knowledge to learn more advanced techniques of art creation in Scratch.
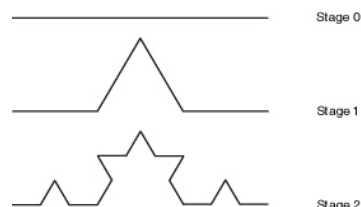

In particular, we will be exploring how recursion can be applied to create animations (instead of just still 2D drawings). We will also be dabbling in an interesting technique for constructing art out of randomness: iterated function systems.

RECURSION

To recap, recursion is defining a task in terms of simpler versions of itself. In this regard, a recursive procedure is a procedure (referred to as a *My Block* in Scratch) that calls itself. In order to help us develop a firmer grasp of this idea, let us take a look at some examples:

- Your great-grandparent can be recursively defined as the parent of the parent of your parent.

- Prime factorization is a recursive procedure. We start by getting two factors of the given number, then proceed to prime factorize these factors. Extending this idea, trees (as in *factor trees* and *binary search trees*) themselves are classified as recursive data structures; a tree, after all, is composed of subtrees.

- Binary search is another recursive algorithm. If the *key* (the value that we are searching for) comes before the middle item, we perform binary search on the left half of the list. Otherwise, we perform binary search on the right half.
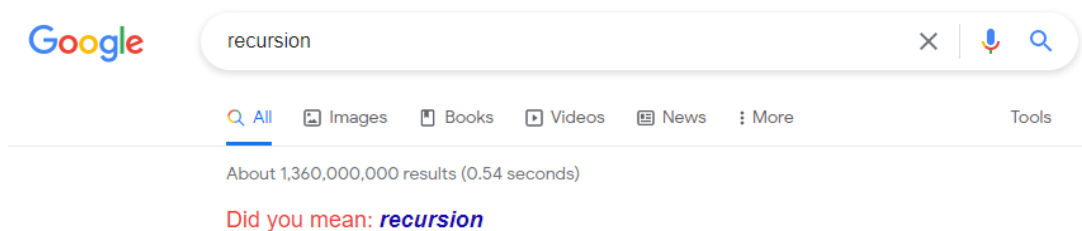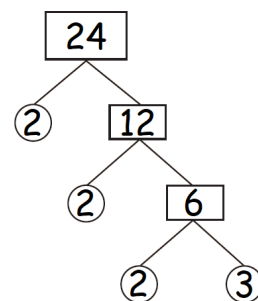
A recursive procedure has two important components:

- Base Case: This tells the procedure when to stop. In binary search, we hit the base case once we find the key (that is, the key and the middle item are identical).

> ⚠ WARNING: Forgetting to include the base case will lead to a stack overflow, potentially crashing our machine as a result of overloading its memory.

- Recursive Call: The procedure should call a simpler version of itself.

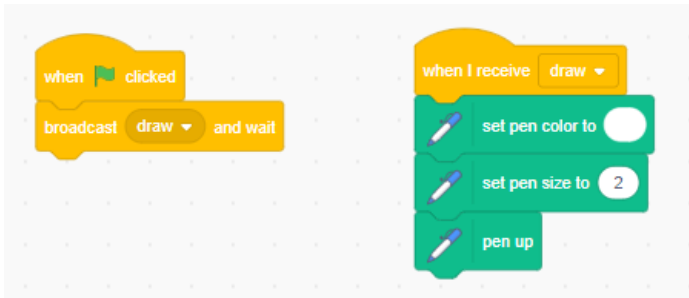*Prepared by Mark Edward M. Gonzales*

Walkthrough SPINNING SQUARES[1]

This walkthrough focuses on employing recursion to create an animated fractal that consists of spinning squares. Here is a video of the final output.

A. Preliminaries

1. Set up the blocks related to the Pen Tool.

2. Create two variables: `direction of large square` and `direction of small square`, and initialize both of them to 0. The first variable is related to the spinning of the large (central) square whereas the second variable is related to the spinning of the four smaller squares (only one variable is needed since they move in sync).

   Anchor the sprite to the origin and point it in the direction of the large square.
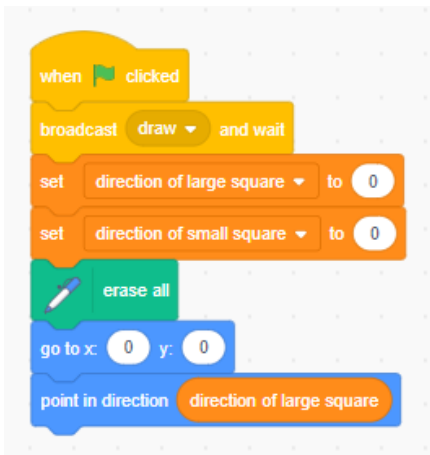
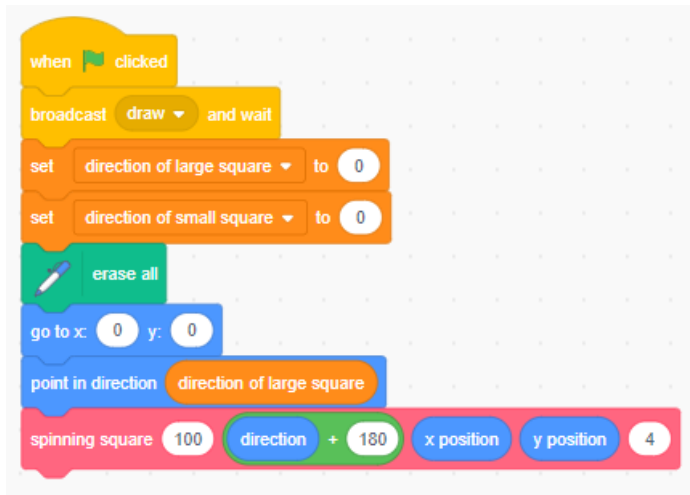3. Create the main My Block: `spinning square`. It has five *parameters* (inputs):

Parameter	Description
size	Size of the large square
direction	Direction in which the large square is facing
x	*x*-coordinate of the sprite
y	*y*-coordinate of the sprite
complexity	Level of detail of the fractal

---

[1] Adapted from https://scratch.mit.edu/projects/8709200/editor/

Asian MathSci League, Inc (AMSLI)

Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
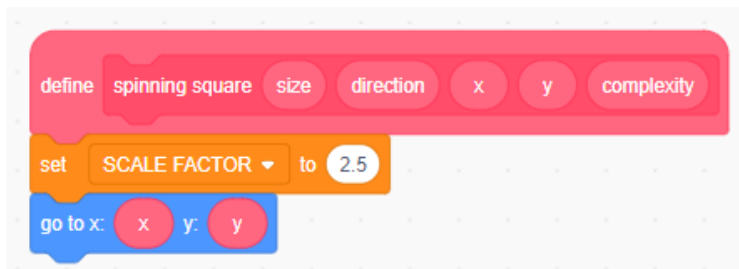Contact Nos: +632-9254526   +63906-1186067

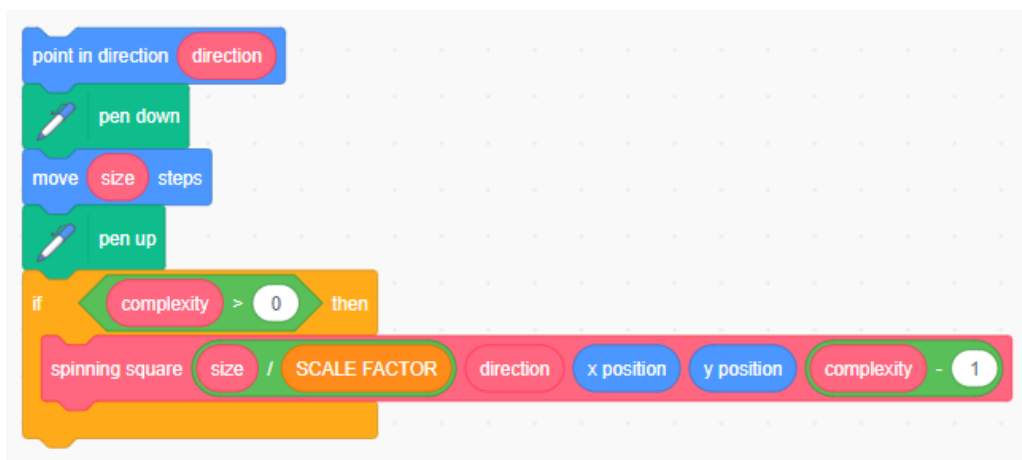4. Call the `spinning square` My Block. Feel free to experiment with the input values.

B. Building the Fractal

The blocks in this section are meant to be chained in sequence.

1. Initialize the constant `SCALE FACTOR`, the ratio of the size of the large square to the size of one of the smaller squares. Feel free to experiment with its value.

2. Create the first side of the fractal. Take note of how the recursive procedure is structured:

   - The entire recursive call is enclosed inside an `if` block, implying the **base case**: the recursive procedure stops when the complexity (level of detail) drops to 0.

   - Recall that recursion is defining a task in terms of **simpler** versions of itself. To this end, the complexity (level of detail) is reduced by 1 with each recursive call.

   - The smaller squares are constructed by dividing the size of the large square by the scale factor.

*Prepared by Mark Edward M. Gonzales*

Running our code thus far should show the following output:
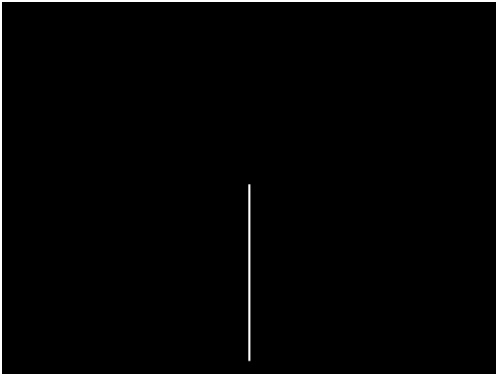
3. Create the second side of the fractal by replacing all the instances of `direction` in the previous code with `direction + 90`. Note that, from this step onwards, we are adjusting the direction by 90°, as we are trying to build a square-based fractal.
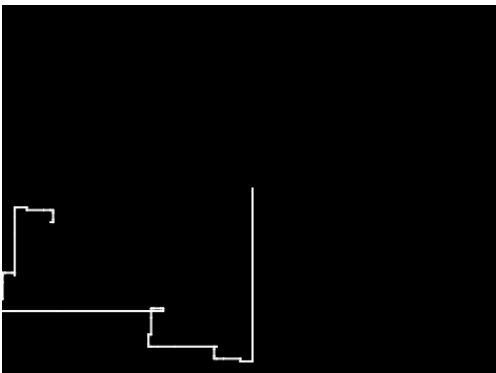
4. Create the third side of the fractal by replacing all the instances of `direction` in the previous code with `direction + 180`.
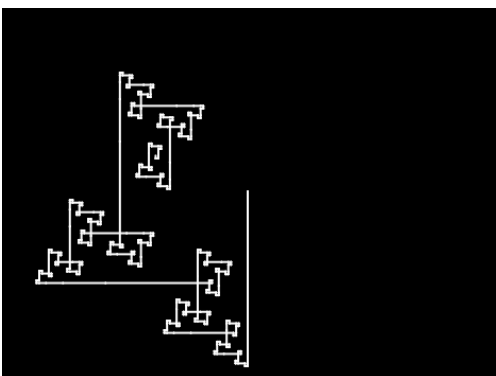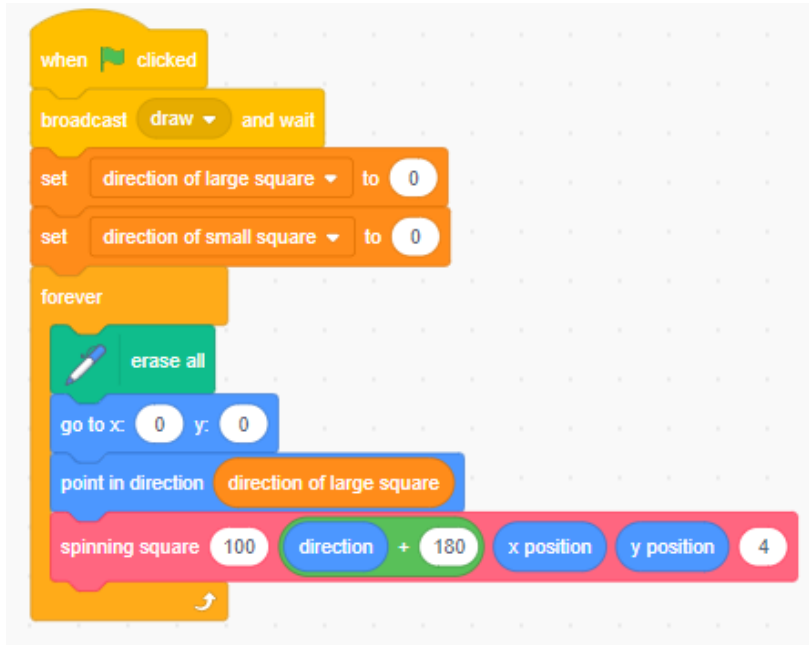

5. Create the last side of the fractal by replacing all the instances of `direction` in the previous code with `direction + 270`.
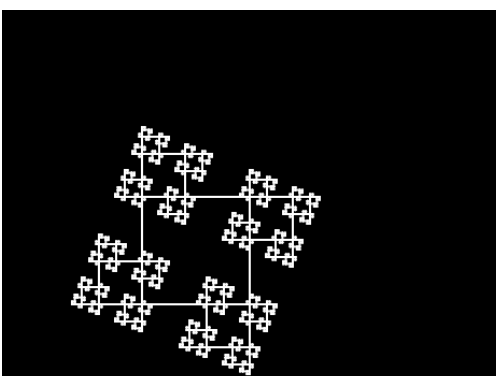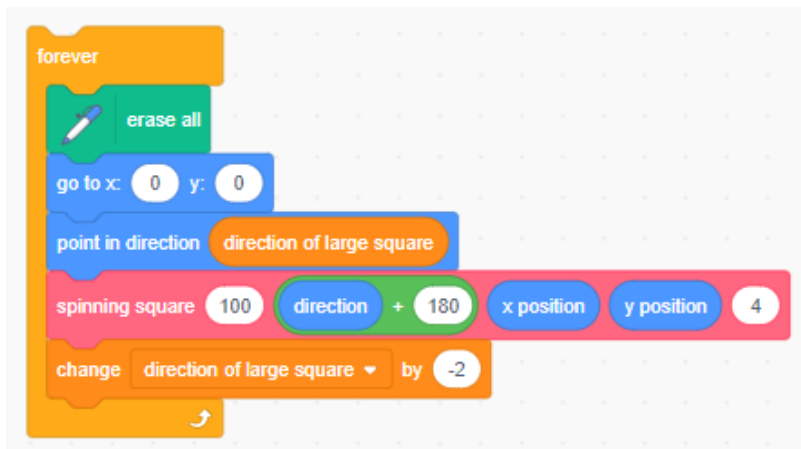
C. Animating the Large Square

1. Now that we have a static 2D version of our fractal, our next goal is to animate it. Check what happens if we enclose some of the blocks in **Step 4 of A. Preliminaries** inside a `forever` loop.

2. The fractal should now be "flickering," signaling that some animation is happening. However, our aim is a spinning motion. To this end, adjust the direction of the large square every time the loop iterates. Note that it is better to adjust by only a small value to keep the animation fluid. (A negative value indicates that the direction of the spin is counterclockwise.)

3. The fractal (specifically the large square) should now be spinning around the stage, albeit still "flickering." This behavior is caused by the screen refreshing every time the `spinning square` My Block is called. In order to prevent this, edit this My Block and check "Run without screen refresh."

💡 **CHECKPOINT:** At the end of this step, your output should be similar to this video.

Prepared by Mark Edward M. Gonzales

D. Animating the Smaller Squares

1. To animate the smaller squares, adjust their direction every time the loop iterates. For artistic effect, it is better to adjust by a positive value so that the squares will spin in a clockwise direction (opposite to the spinning of the large square).
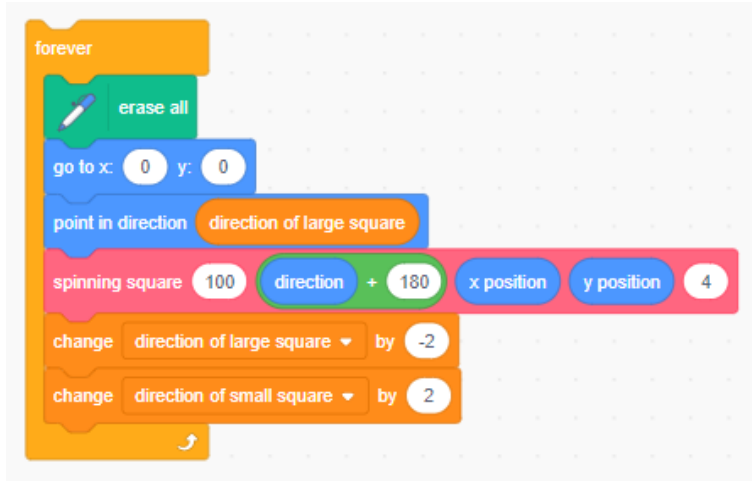

2. Modify the second parameter of each recursive call to take into account the direction of the smaller squares.

> 💡 CHECKPOINT: At the end of this step, your output should be similar to this video.

E. Finishing Touches

1. We are almost finished! The only remaining flaw in our output thus far is that the large square appears to be "disjointed" at times, as seen in the snapshot below:
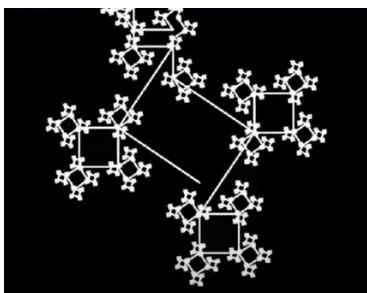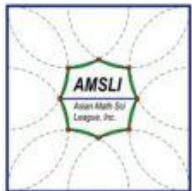

2. Since this is caused by a mispositioning of the sprite during the spinning motion of the smaller squares, a quick fix is to simply force it to go to the correct coordinates by appending this instruction at the end of the `spinning square` My Block:

> 💡 FINAL OUTPUT: At the end of this step, your output should be similar to this video.
>
> Try remixing the program so that the colors change every time the squares spin.

Prepared by Mark Edward M. Gonzales

ITERATED FUNCTION SYSTEM

Aside from recursion, another method to create fractals is via **iterated function systems**, which were introduced by the mathematician John Hutchinson in 1981. In order to help us understand the gist of this concept, we break it down into its constituent words:

- **Iterated**: To iterate means to repeat.
- **Function**: A function can be thought of as an equation that assigns an input *(x)* to a corresponding output *(y)*.
- **System**: A system is a group. In this regard, a system of functions can be viewed as a group of equations.
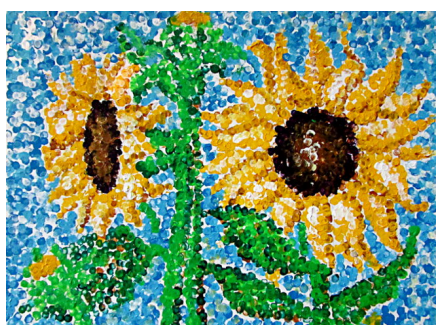
To generate fractals, iterated function systems (or simply **IFS**) usually incorporate **randomness** in an algorithmic technique aptly named the **chaos game**: An equation (function) is randomly selected from a group (system) of equations, and its output is plotted as a point on the *xy*-plane. This process is (iteratively) repeated — the more we repeat, the more detailed our figure will be!

This method of constructing art out of randomness is slightly reminiscent of the Pointillist movement back in the late 1880s, where paintings (such as the one above) are made from hundreds, thousands, or even millions of colored dots.

Walkthrough HEXAGONAL FLOWERS[2]

To demonstrate how IFS can be used to generate fractals, let us create a hexagonal flower.

1. Create two lists called `x` and `y`, and insert the following values:

| x	300	300	0	–300	–300	0
y	–180	180	360	180	–180	–360

2. Inside a loop that repeats 10,000 times (feel free to adjust the number of repetitions):

   a. Pick a random number from 1 to the length of either list (inclusive). Call this random number `random`.

   b. Move the sprite to these coordinates:
      - ***x*-coordinate** = (current *x*-position + item `random` of list x) ÷ 3
      - ***y*-coordinate** = (current *y*-position + item `random` of list y) ÷ 3

   c. Set up the blocks related to the Pen Tool so that the sprite draws a point at the location where it is currently positioned.

---

[2] Adopted from https://scratch.mit.edu/projects/73798570/editor/

3. Turn on Turbo Mode, and run the program. Your output should be similar to this figure:
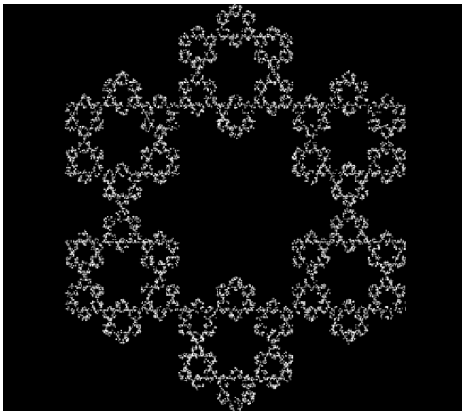
> 💡 **SUGGESTED ACTIVITY:** Try remixing the program so that the color of each point is randomly chosen.

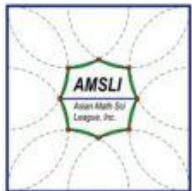## Walkthrough VICSEK FRACTAL[3]

Named after the Hungarian scientist Tamás Vicsek, the **Vicsek fractal** (also known as the **box fractal**) is a fascinating fractal that has applications to the construction of antennas for cellular phones. Pay attention to the general pattern for drawing fractals via IFS.

1. Create two lists called x and y, and insert the following values:

x	300	−300	0	0	0
y	0	0	300	−300	0

2. Inside a loop that repeats 10,000 times (feel free to adjust the number of repetitions):

    a. Pick a random number from 1 to the length of either list (inclusive). Call this random number `random`.

---
[3] Adapted from https://scratch.mit.edu/projects/120691702/editor/

Prepared by Mark Edward M. Gonzales

   b. Move the sprite to these coordinates:

- **x-coordinate** = (current x-position + item `random` of list `x`) ÷ 3

- **y-coordinate** = (current y-position + item `random` of list `y`) ÷ 3

   c. Set up the blocks related to the Pen Tool so that the sprite draws a point at the location where it is currently positioned.

Note that the code is the same as that for Hexagonal Flowers. They differ only in the contents of the lists x and y.

3. Turn on Turbo Mode, and run the program. Your output should be similar to this figure:
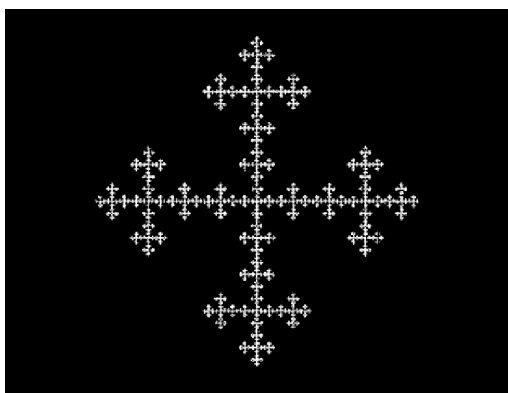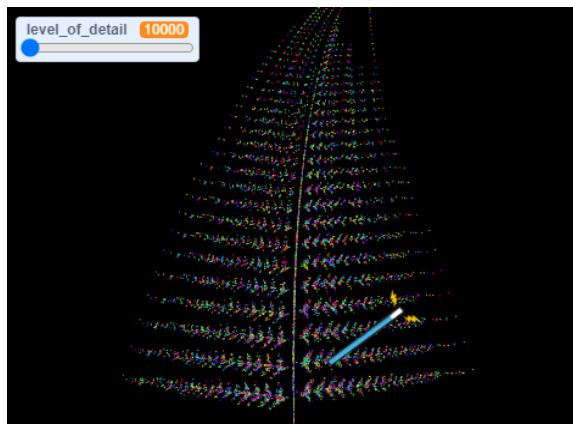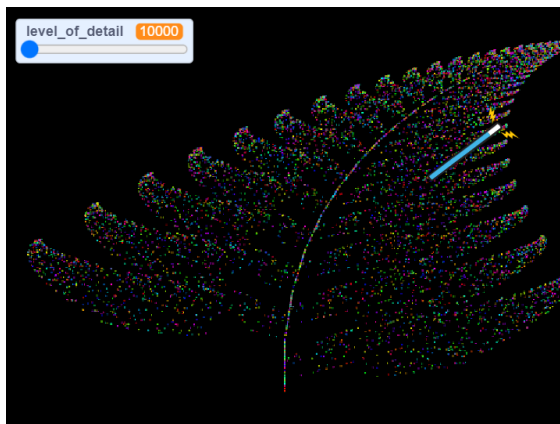

> 💡 **SUGGESTED ACTIVITY:** Try experimenting with the values stored in lists x and y. What shapes or fractals can you generate?

Challenge BARNSLEY FERN

In his 1988 book *Fractals Everywhere*, Michael Barnsley explored how fractals can be used to model various things in nature, such as the structures of plants. His investigations led to the discovery of a set of equations that can be used to draw a fern that resembles the black spleenwort.

This challenge focuses on using IFS to draw this fern — eponymously referred to as the **Barnsley fern** — and a mutant variety that resembles the *Thelypteridaceae* fern.

-- return 0; --