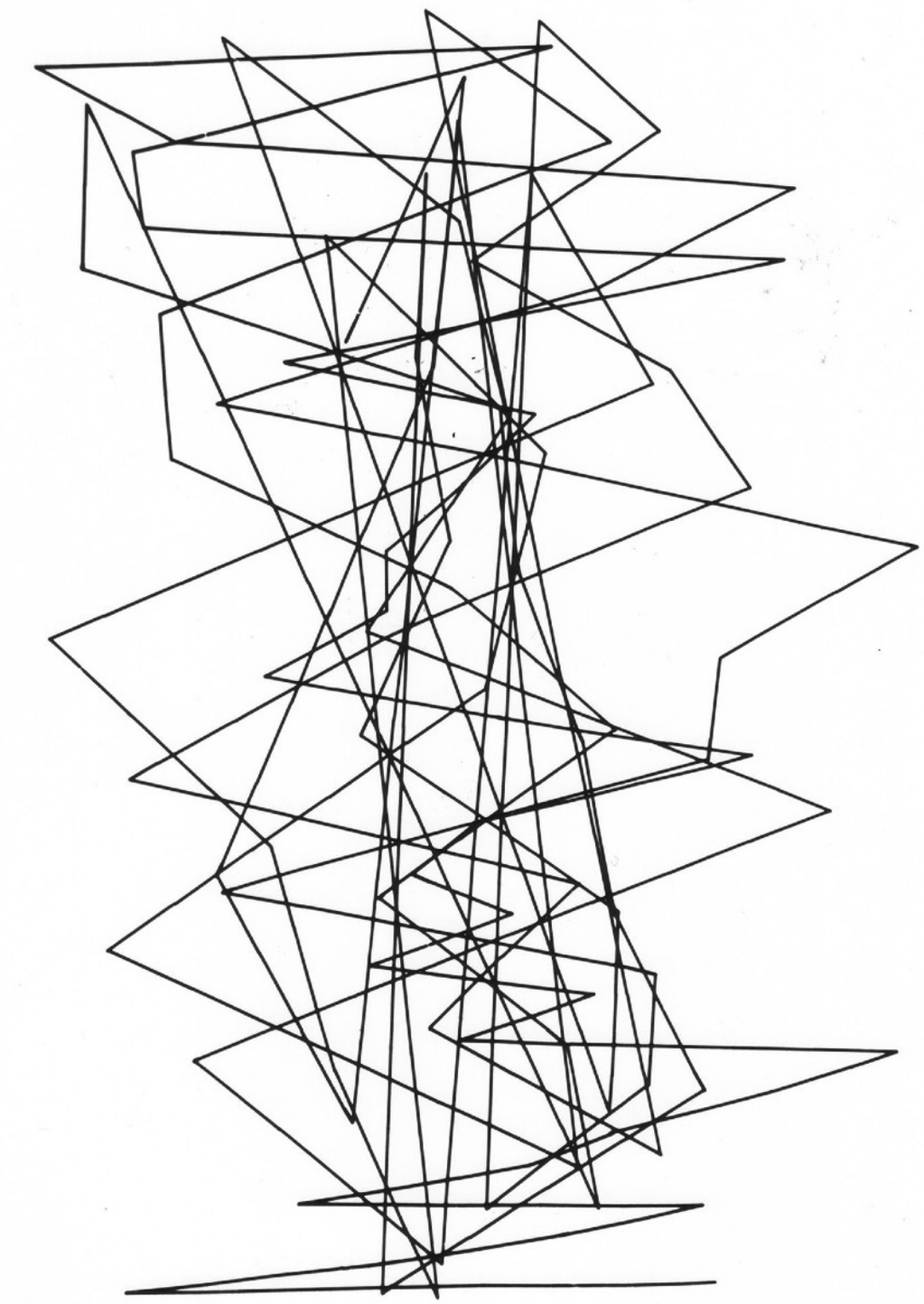
GETTING GENERATIVE WITH P5.JS

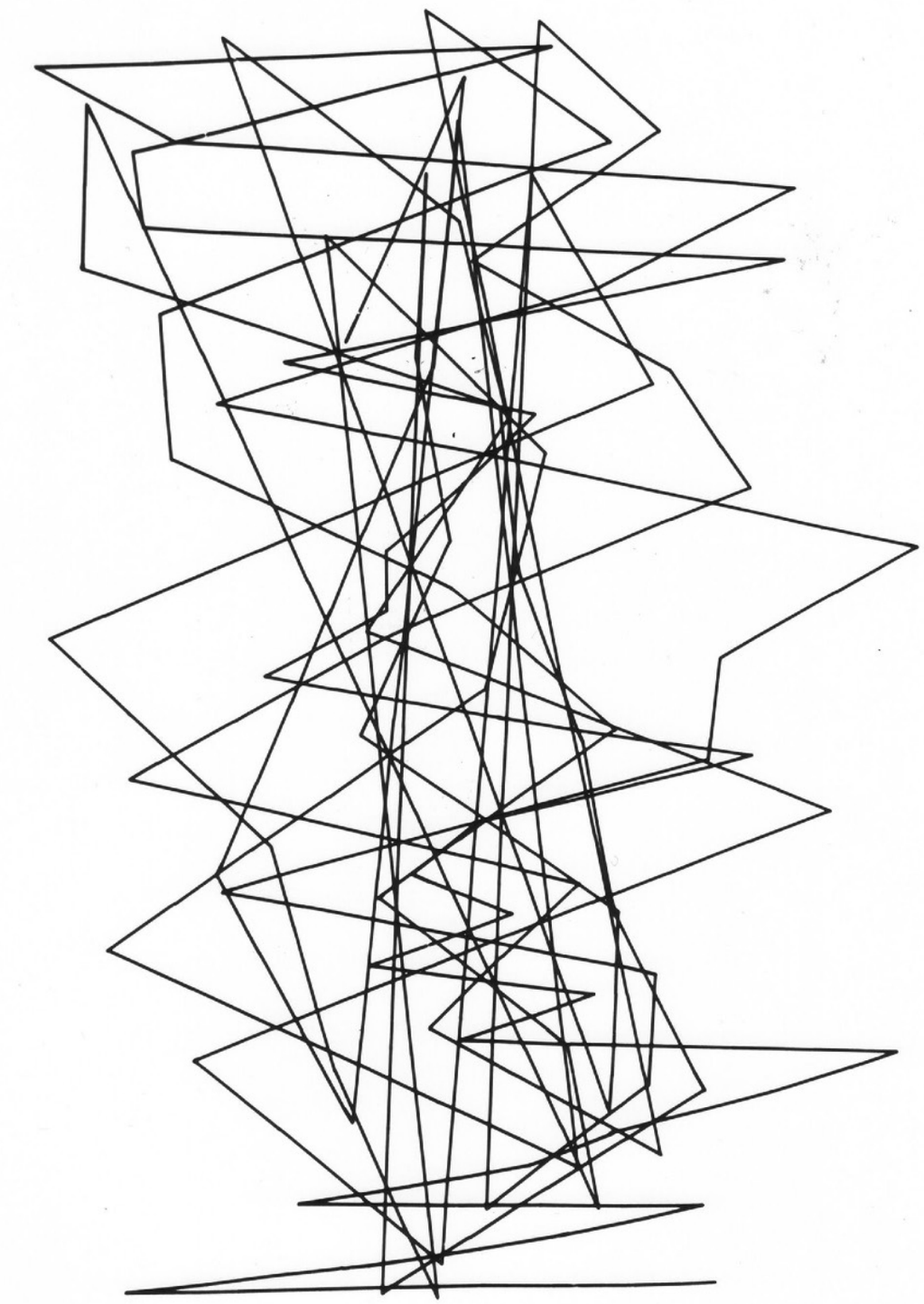
DANIEL HOWE
SHEUNG WAN, HK
FEBRUARY 16, 2019



GAUSSIAN-QUADRATIC (1963)
BY A. MICHAEL NOLL

GETTING GENERATIVE WITH P5.JS

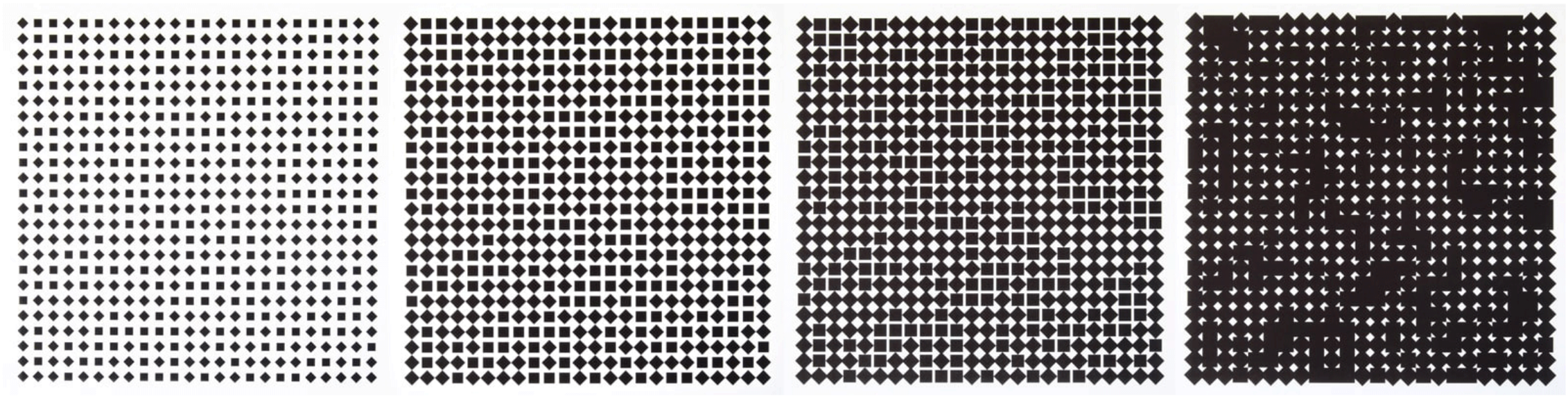
DANIEL HOWE
SHEUNG WAN, HK
FEBRUARY 16, 2019



GAUSSIAN-QUADRATIC (1963)
BY A. MICHAEL NOLL

SLIDES & CODE

https://github.com/dhowe/GetGen



VERA MOLNAR, CARRÉS EN
2 POSITIONS 1-4 , 2011-13

dhowe / GetGen

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

A Workshop for Processing Community Day @ HK 2019

Edit

Manage topics

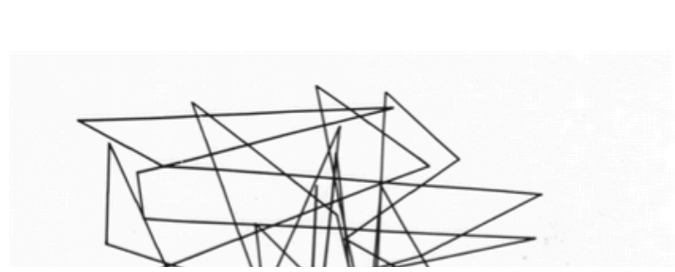
19 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

dhowe updated — Latest commit 49854aa 27 seconds ago

README.md	Update README.md	23 hours ago
basic-tree.js	initial	a day ago
fractal-tree1.js	initial	a day ago
fractal-tree2.js	initial	a day ago
fractal-tree3.js	initial	a day ago
getgen.png	initial	23 hours ago
mapped-tree.js	initial	a day ago
slides.pdf	updated	26 seconds ago

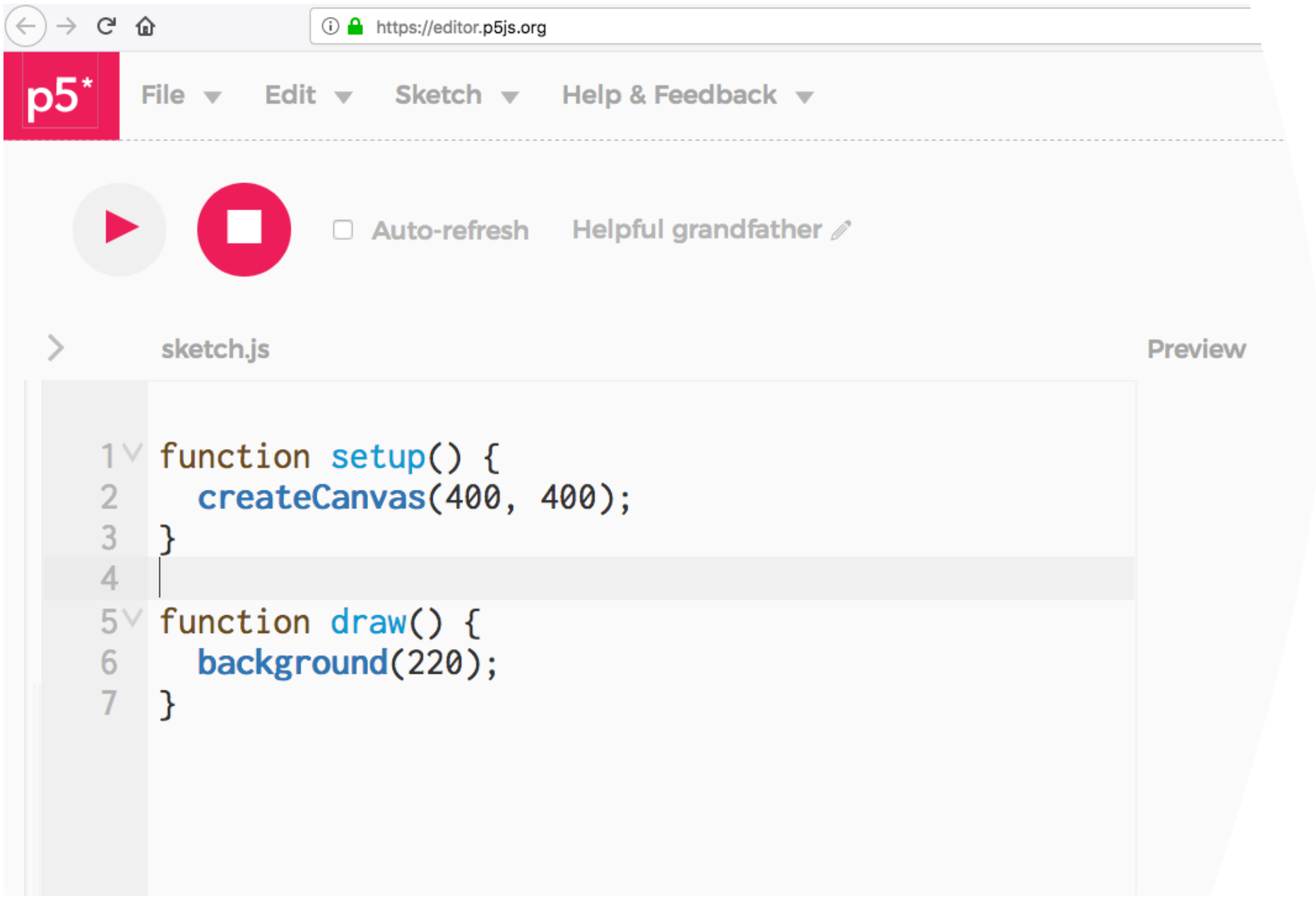
README.md



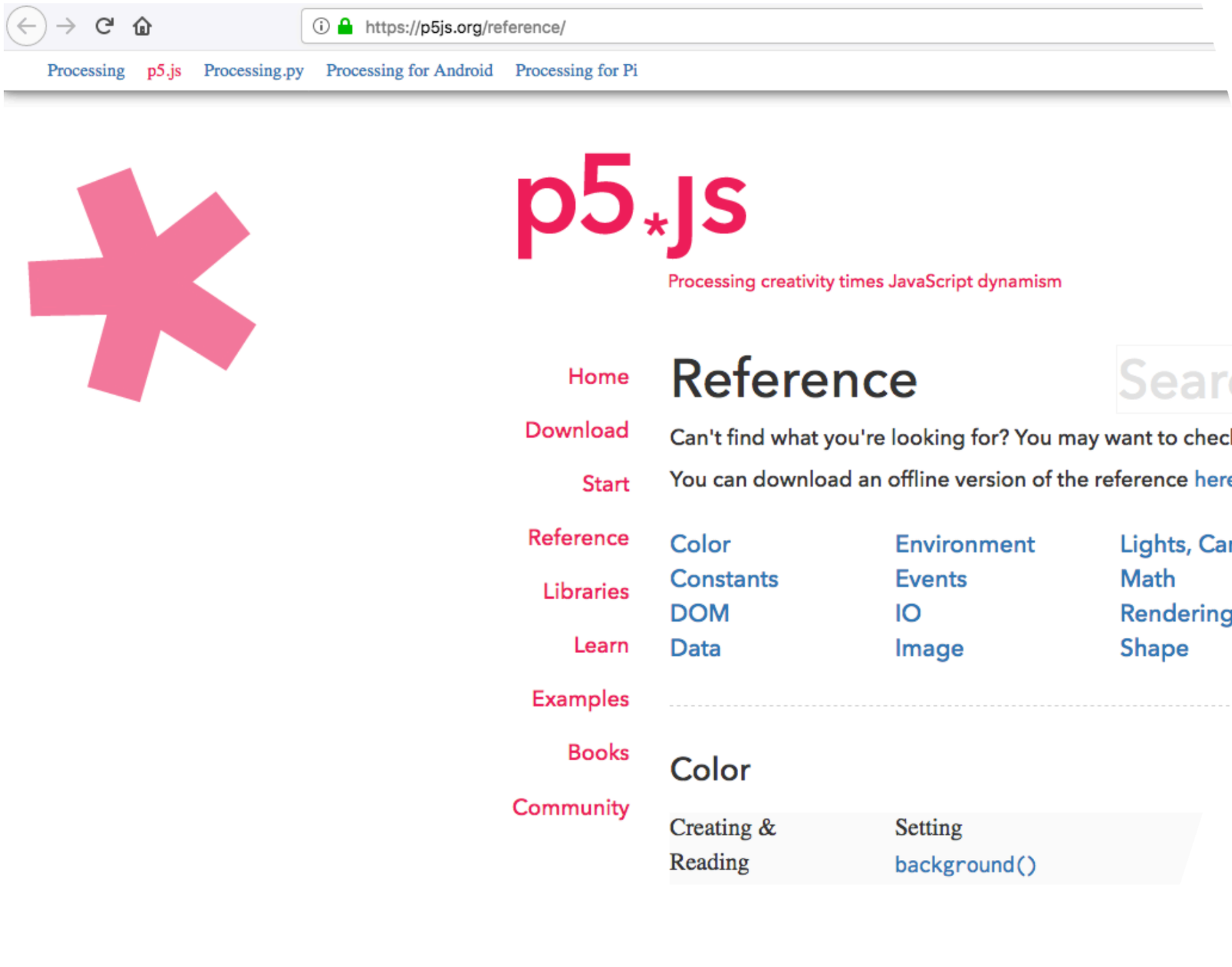
https://github.com/dhowe/GetGen

RECURSION
FRACTALS
AFFINE TRANSFORMS
RANDOM
NOISE

TOOLS: P5.JS EDITOR



TOOLS: REFERENCE



[WHEN] THE ARTIST USES A SYSTEM, SUCH AS A SET OF NATURAL LANGUAGE RULES, A COMPUTER PROGRAM, A MACHINE, OR OTHER PROCEDURAL INVENTION, WHICH IS SET INTO MOTION WITH SOME DEGREE OF AUTONOMY CONTRIBUTING TO OR RESULTING IN A COMPLETED WORK OF ART

GENERATIVE ART

PHILIP GALANTER, 2003

RECURSION

IMAGE BY JAN MISSET
FOR DROSTE CACAA, 1904

re·cur·sion /ri-ˈkər-zhən/

n. When an entity is defined by one or more references to itself.

RECURSION

re·cur·sion /ri-ˈkər-zhən/

n. See recursion.

RECURSION

URBAN DICTIONARY, 2019

EXAMPLE: FACTORIAL

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

How can we define the factorial() function ?

A FACTORIAL FUNCTION

code editor

```
function setup() {
  createCanvas(400, 400);

  console.log(fact(1));  // 1
  console.log(fact(2));  // 2
  console.log(fact(3));  // 6
  console.log(fact(4));  // 24
  console.log(fact(5));  // 120
  console.log(fact(10)); // 3628800
}

function fact(n) {

  return 0;
}
```

A NON-RECURSIVE SOLUTION

```
function setup() {
  createCanvas(400, 400);

  console.log(fact(1));  // 1
  console.log(fact(2));  // 2
  console.log(fact(3));  // 6
  console.log(fact(4));  // 24
  console.log(fact(5));  // 120
  console.log(fact(10)); // 3628800
}

function fact(n) {
  let total = 1;
  for (let i = 1; i <= n; i++) {
    total = total * i;
  }
  return total;
}
```

Console

```
1
2
6
24
120
3628800
```

EXAMPLE: FACTORIAL

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

Can we write a recursive version of factorial() ?

EXAMPLE: FACTORIAL

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

Can we write a factorial() recursively ?

EXAMPLE: FACTORIAL

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

Can we write a factorial() recursively ?

EXAMPLE: FACTORIAL

$$5! =$$

$$= 5 * 4 * 3 * 2 * 1$$

$$= 5 * (4 * 3 * 2 * 1)$$

$$= 5 * 4!$$

EXAMPLE: FACTORIAL

$$5! =$$

$$= 5 * 4 * 3 * 2 * 1$$

$$= 5 * (4 * 3 * 2 * 1)$$

$$= 5 * 4!$$

EXAMPLE: FACTORIAL

1! = 1

2! = 2 * 1 = 2

3! = 3 * 2 * 1 = 6

4! = 4 * 3 * 2 * 1 = 24

5! = 5 * 4 * 3 * 2 * 1 = 120

Can we write a recursive version of factorial() ?

EXAMPLE: FACTORIAL

For any number **n** (greater than 0):

factorial(n) = n * factorial(n-1)

factorial(1) = 1

EXAMPLE: FACTORIAL

```
function fact(n) {

  if (n == 1) return 1;

  return n * fact(n - 1);
}
```

LET'S DOUBLE CHECK

```
1  function setup() {
2    createCanvas(400, 400);
3
4    console.log(fact(1));  // 1
5    console.log(fact(2));  // 2
6    console.log(fact(3));  // 6
7    console.log(fact(4));  // 24
8    console.log(fact(5));  // 120
9    console.log(fact(10)); // 3628800
10 }
11
12 function fact(n) {
13
14   if (n == 1) return 1;
15
16   return n * fact(n - 1);
17 }
18
```

Console

```
1
2
6
24
120
3628800
```

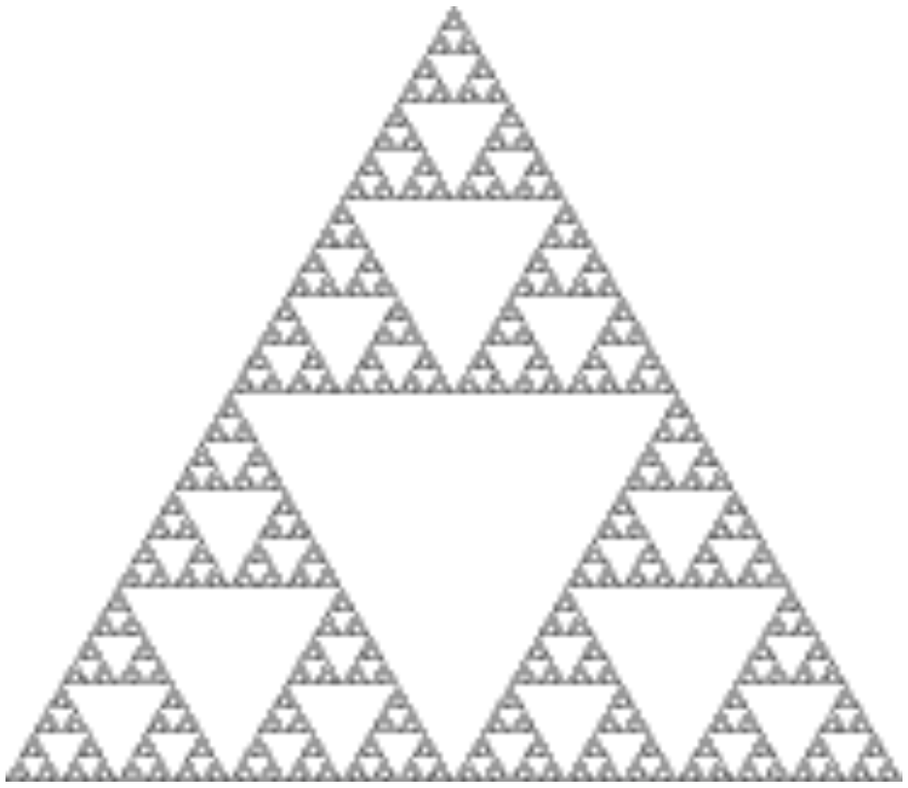
frak·tal /ri-ˈkər-zhən/

n. A fractal is a repeating patterns that is self-similar across different scales. Fractals are created by repeating a simple process over and over in a feedback loop. Driven by recursion, fractals patterns are often familiar, since nature is full of fractals.

FRACTAL

ADAPTED FROM
FRACTALFOUNDATION.ORG/
RESOURCES/WHAT-ARE-FRACTALS/

FRACTALS



the Sierpinski triangle

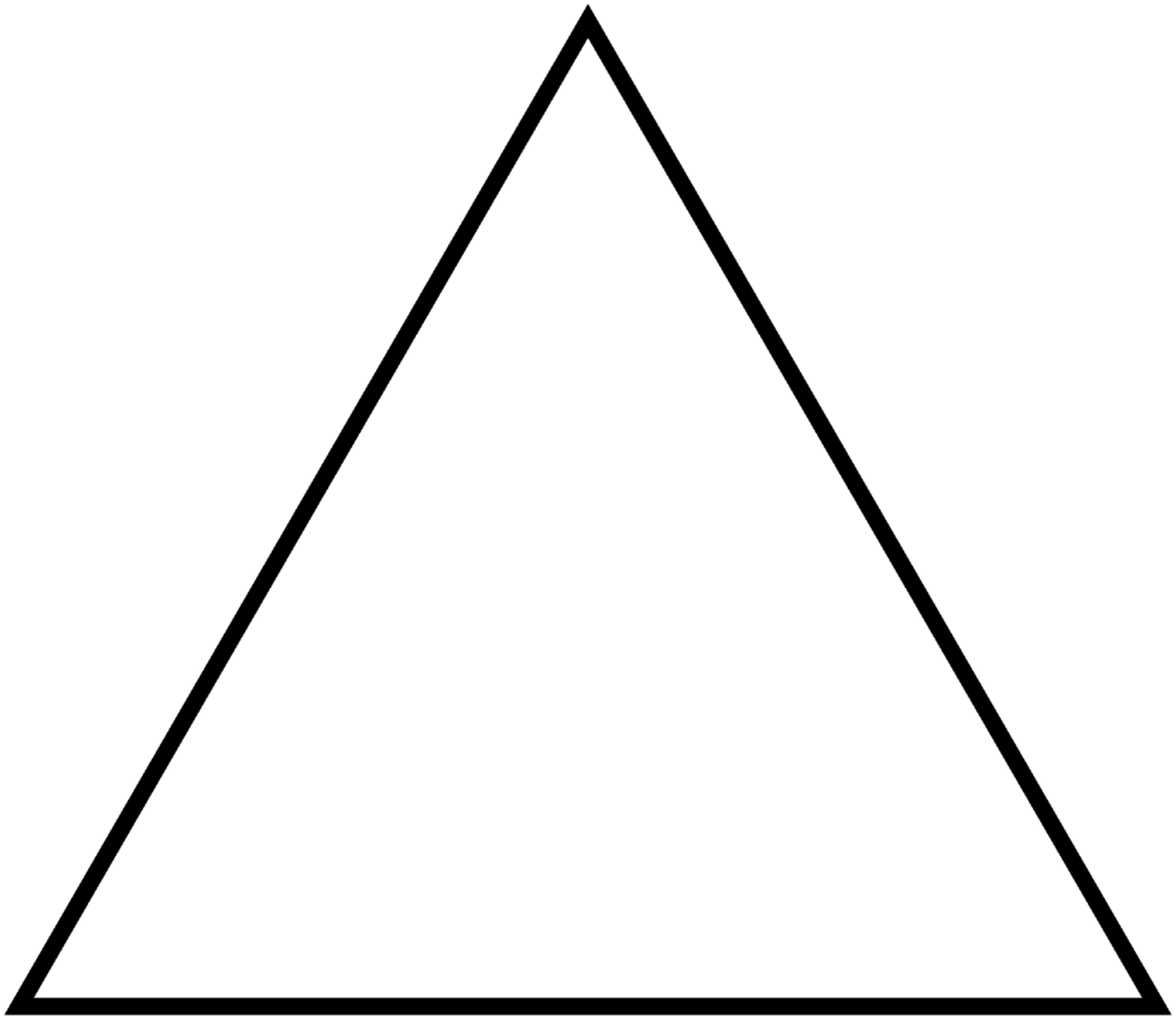
SIERPINSKI TRIANGLE

Procedure:

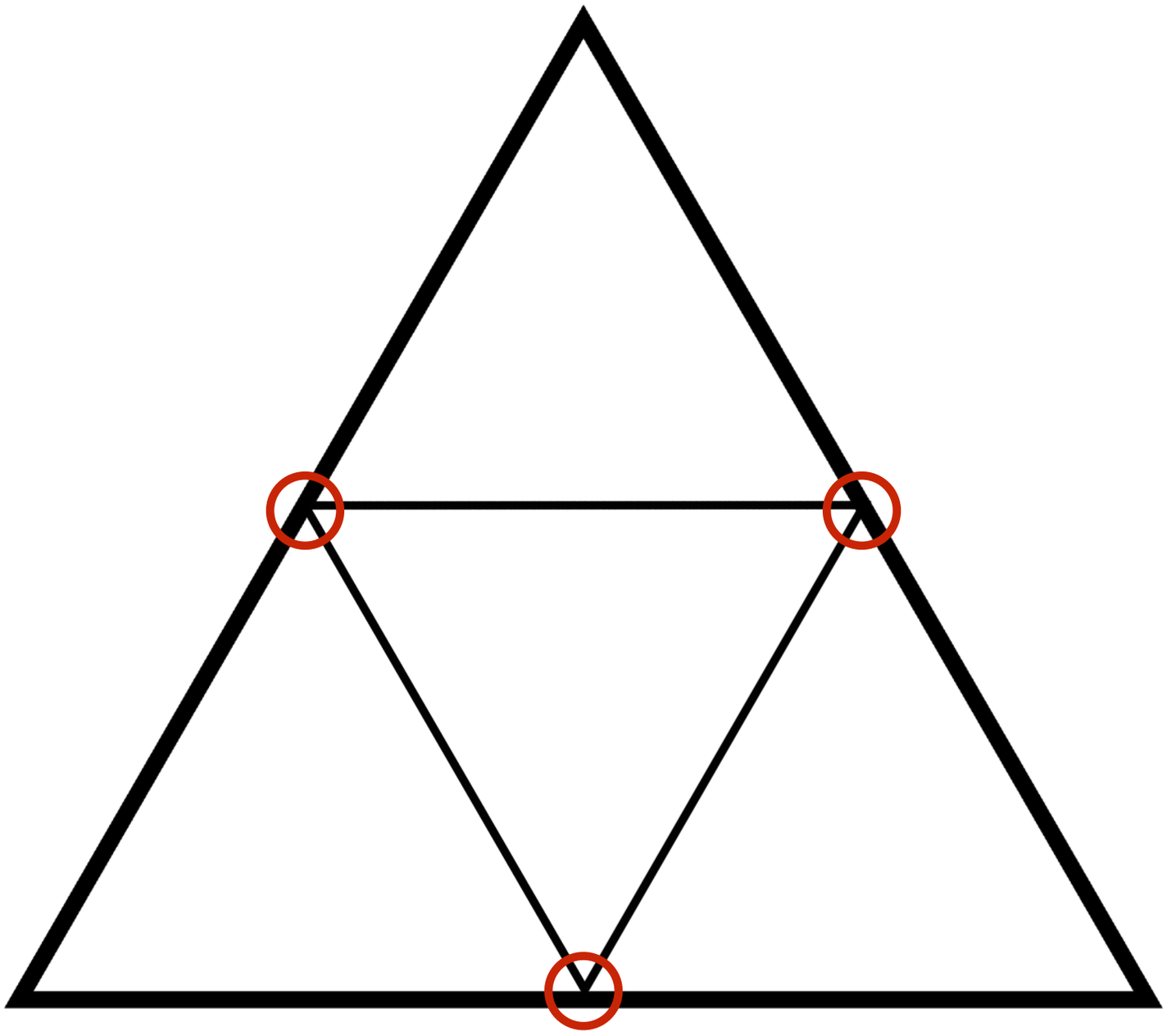
1. Start with an equilateral triangle

2. Make a new triangle from the 3 midpoints of the sides of the triangle.

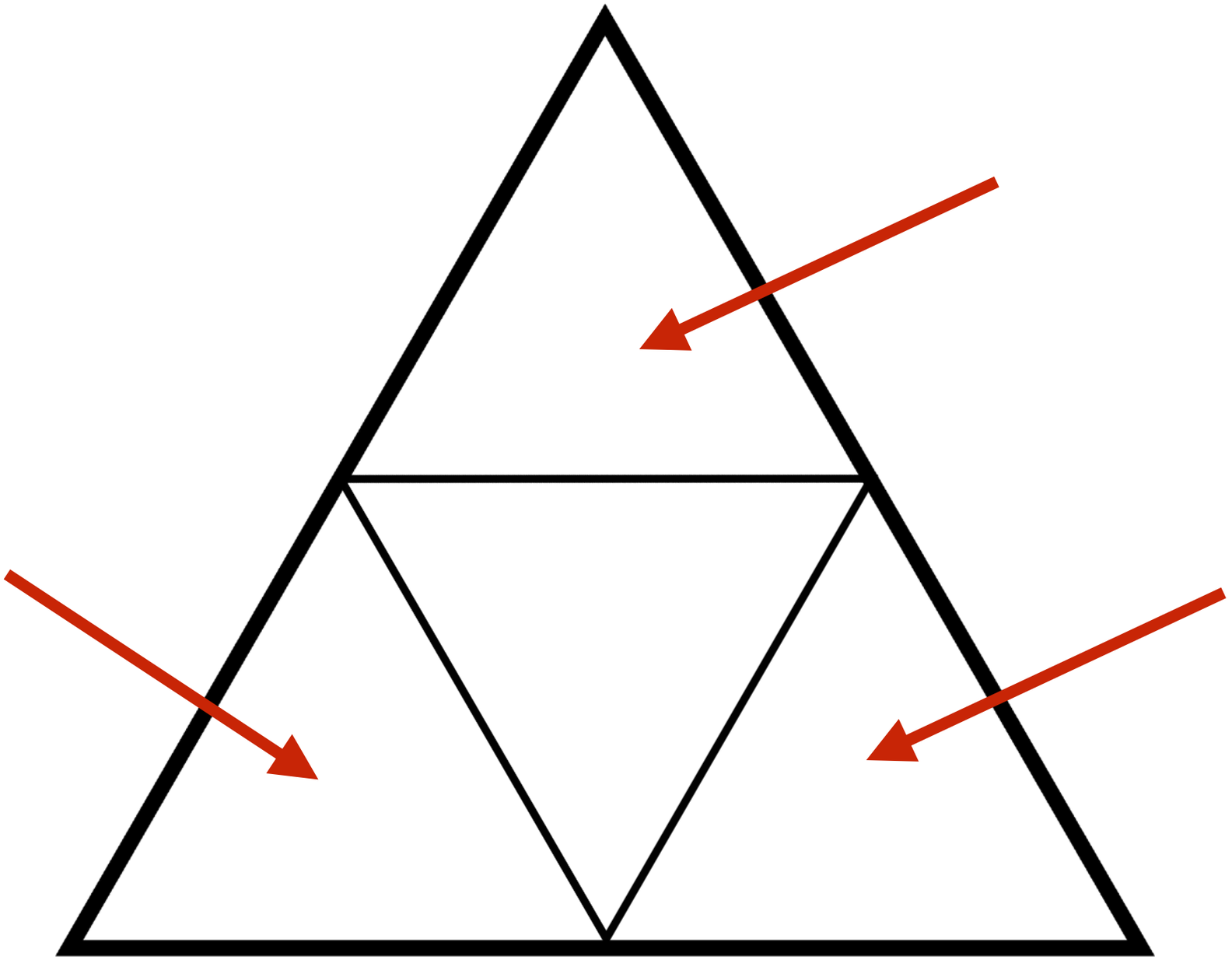
3. Repeat for each of the three newly created triangles



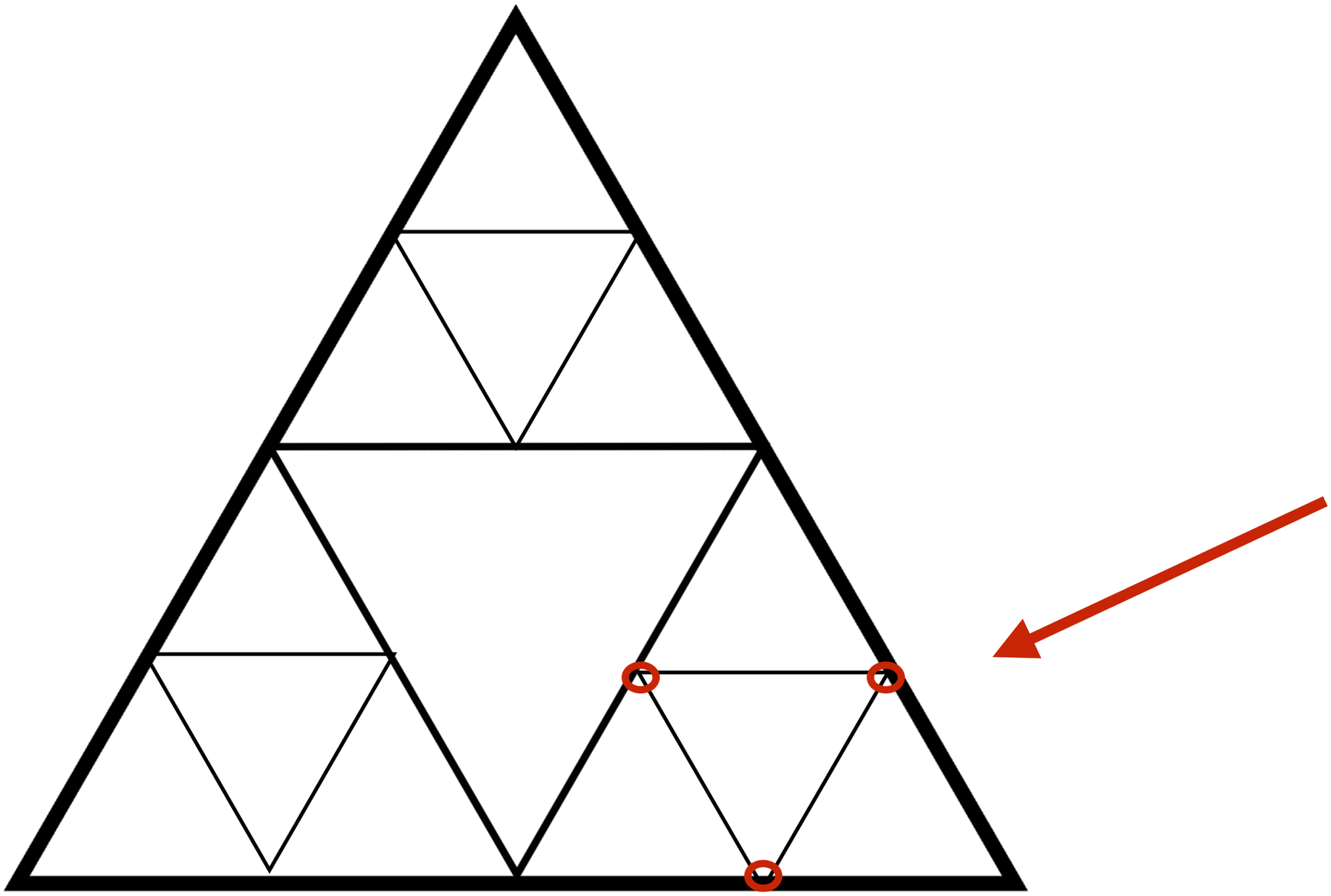
1. Start with an equilateral triangle



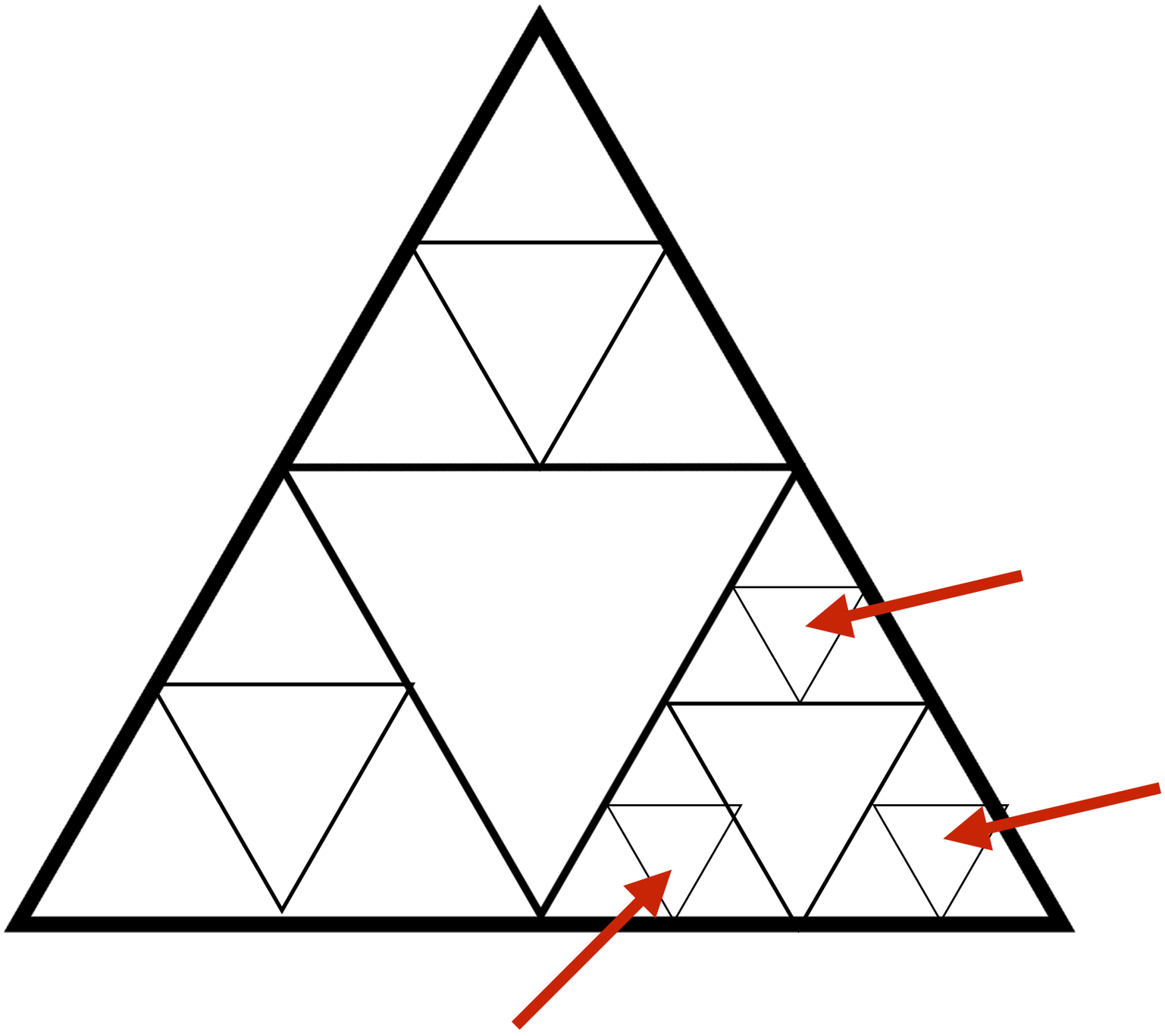
2. Make a new triangle from the three midpoints of the sides of the triangle



3. Repeat for each of the three newly created triangles

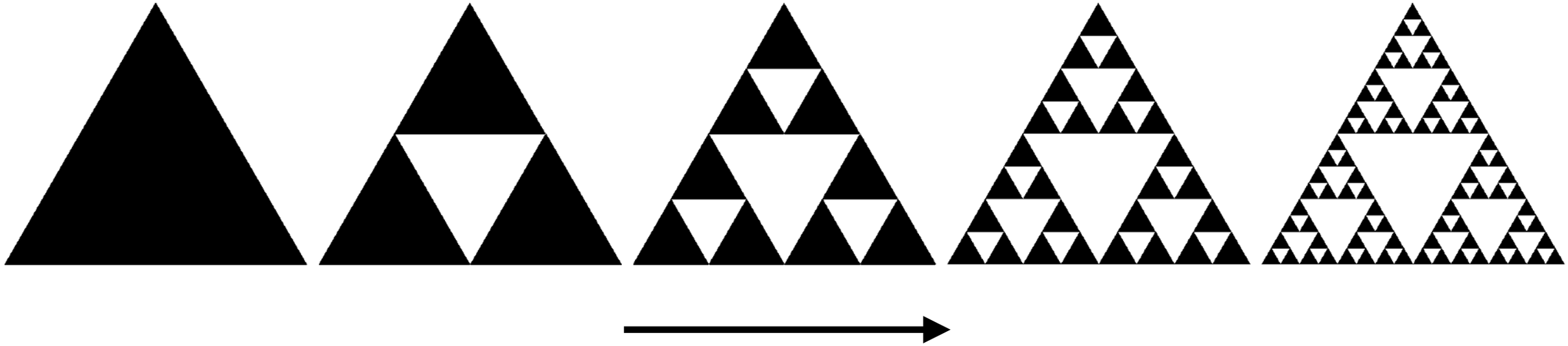


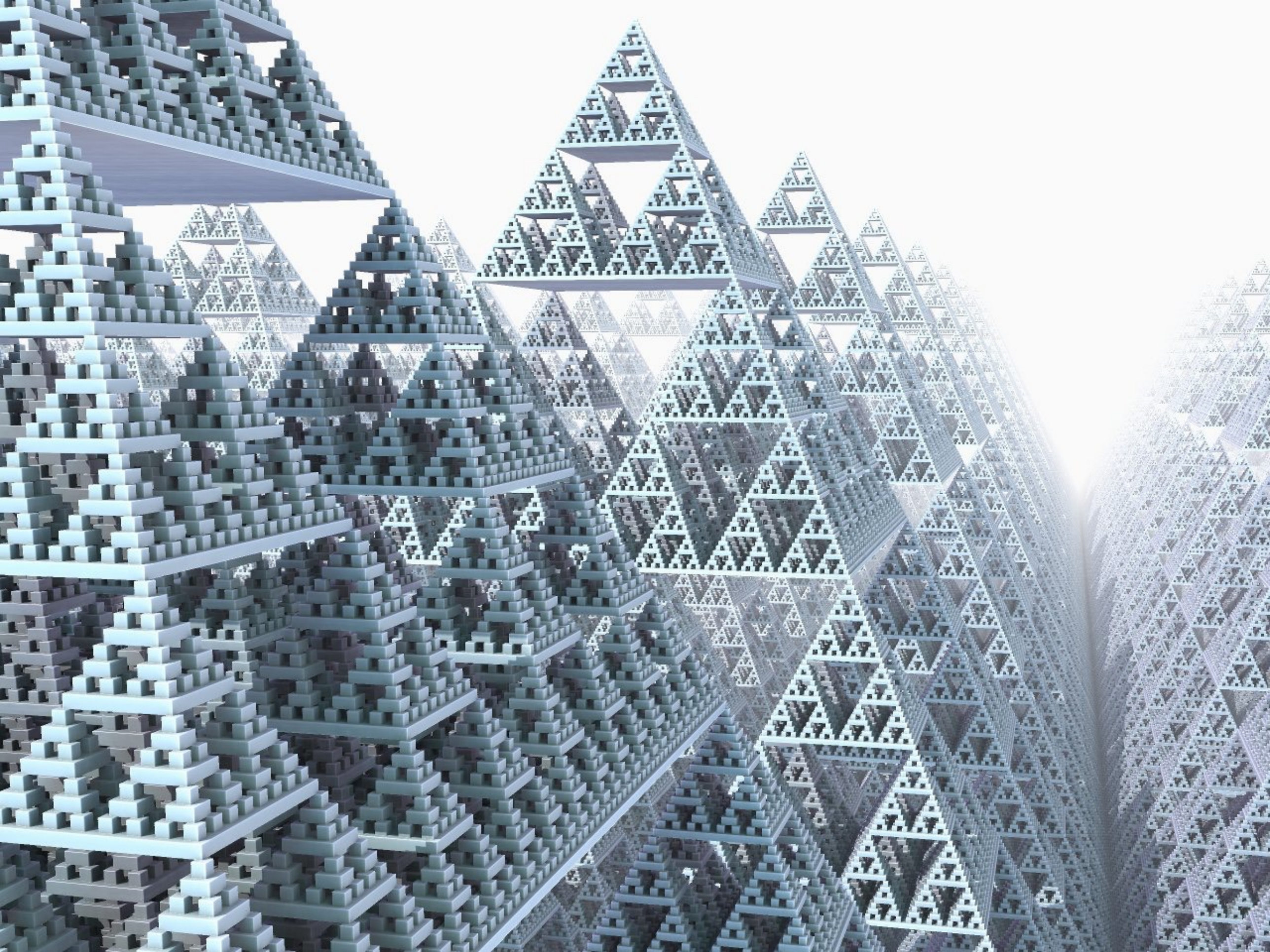
3. Repeat for each of the three newly created triangles



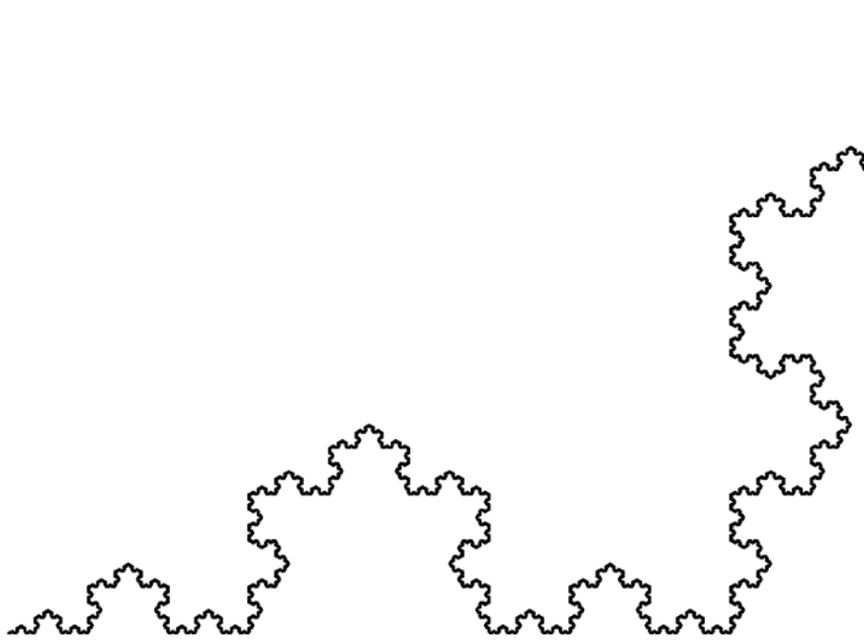
And repeat... and repeat

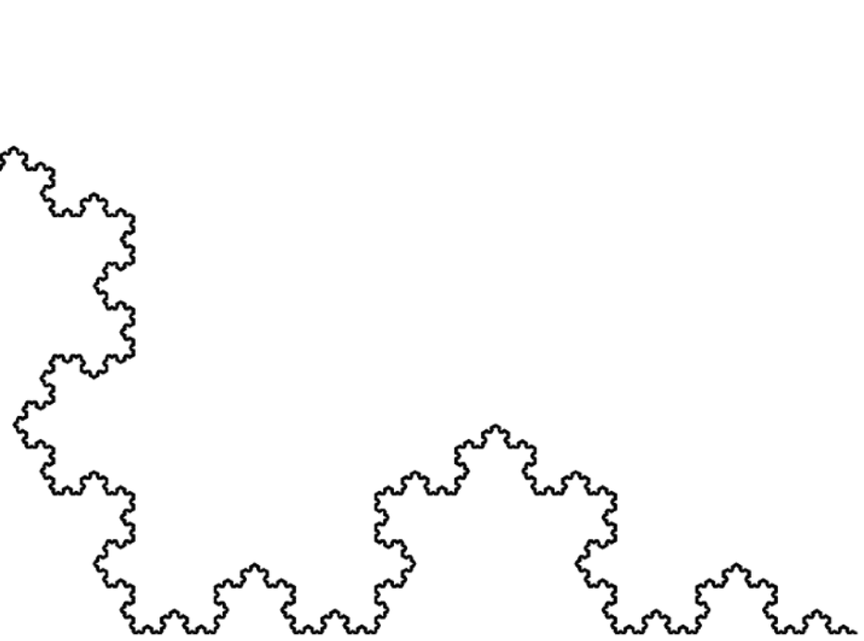
SIERPINSKI TRIANGLE



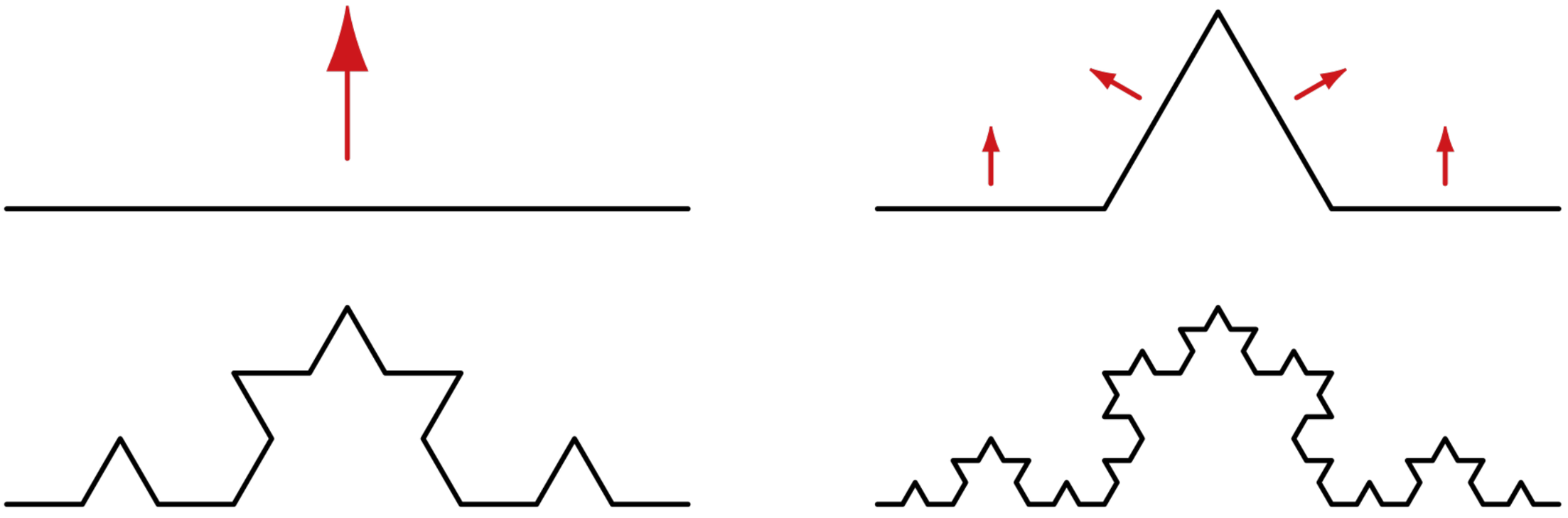


KOCH CURVE

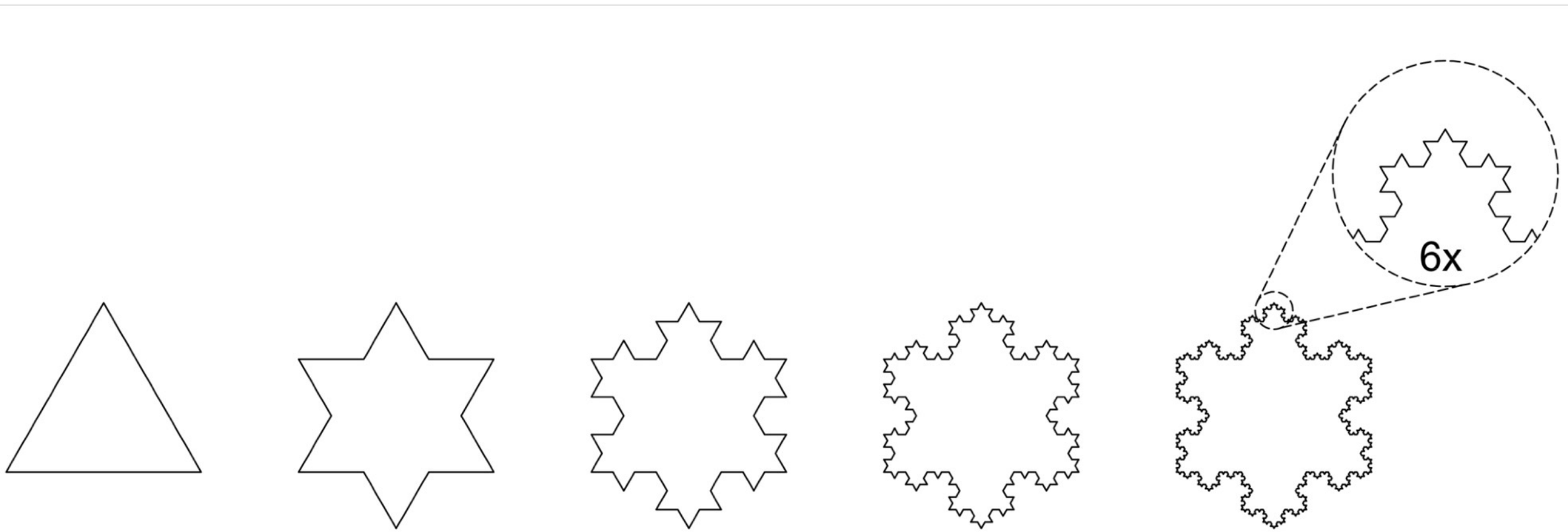




KOCH CURVE

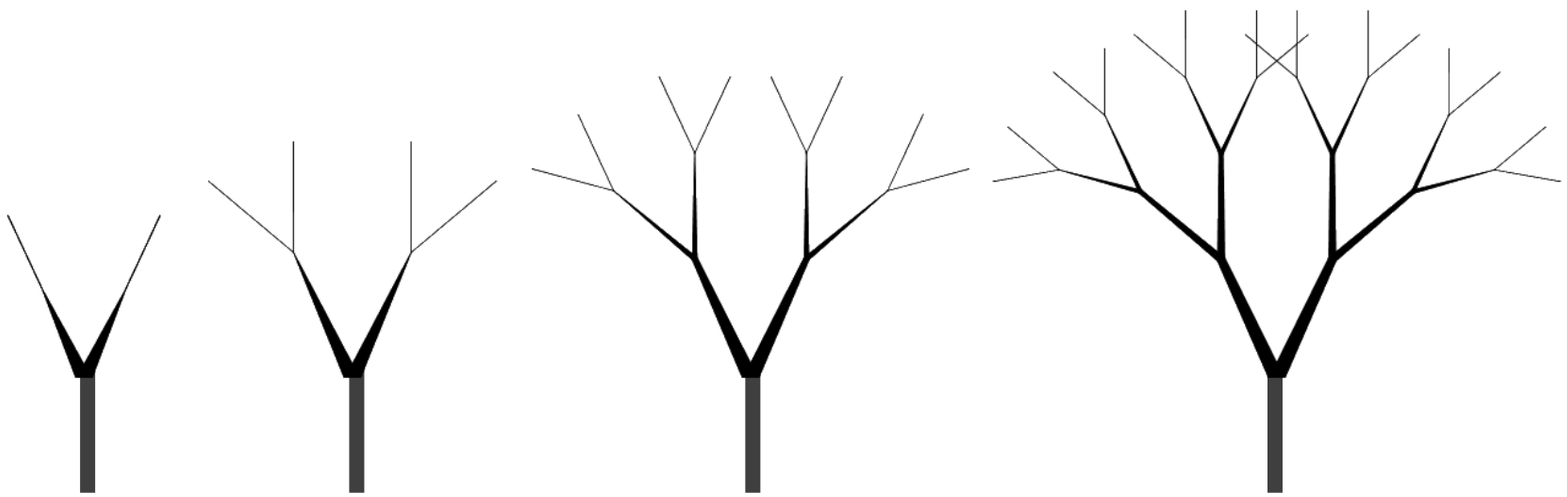


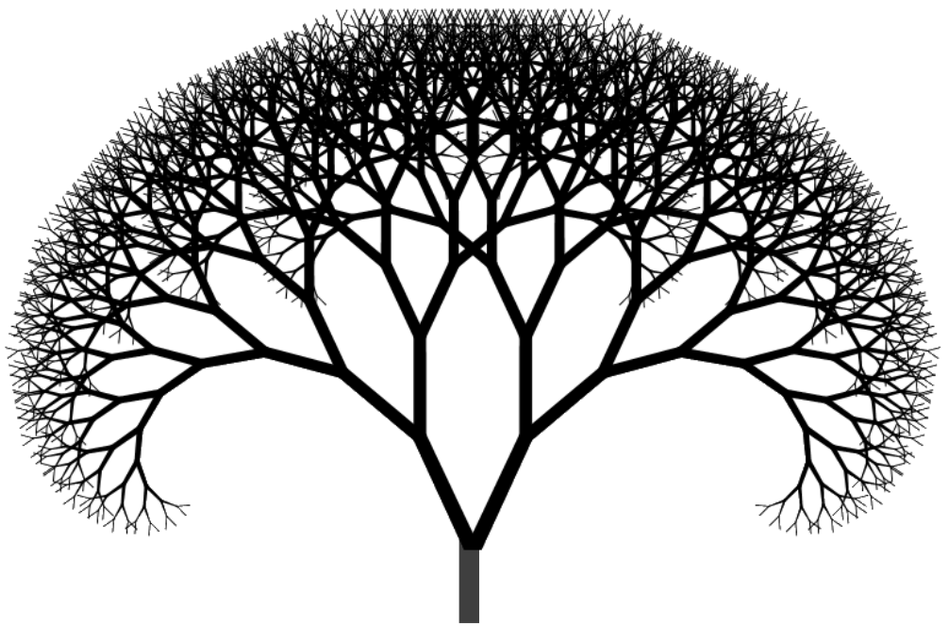
KOCH SNOWFLAKE



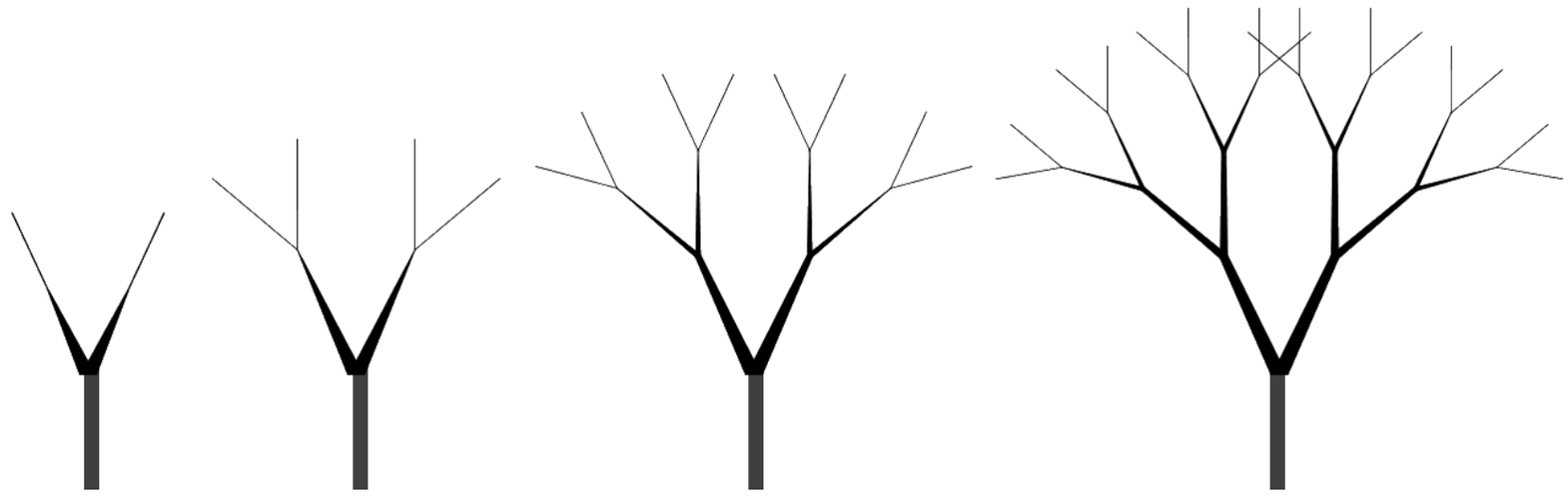
Koch Snowflake fractal - progression of scales

FRACTAL TREE



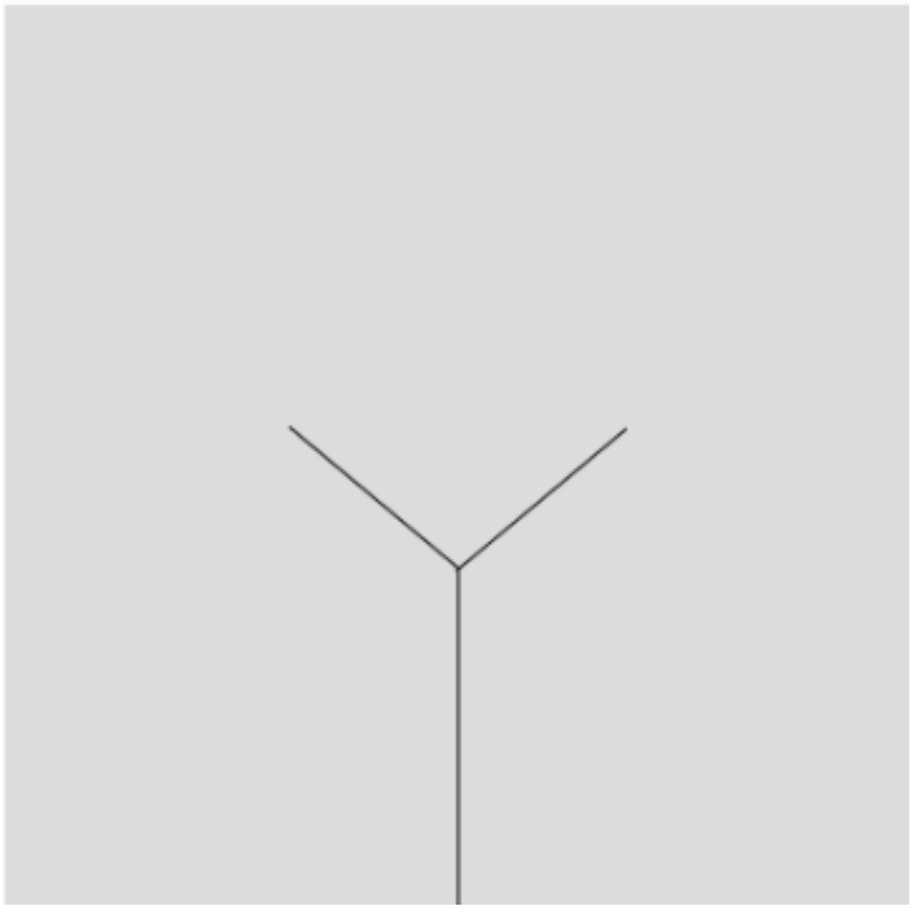


EXAMPLE: FRACTAL TREE



Draw a branch. From its end, rotate right and draw a shorter branch. Then rotate left and do the same. Repeat for the two new branches...

FRACTAL TREE



we can start like this....

A DIFFERENT WAY TO DRAW SHAPES

There are two basic ways of drawing shapes in p5.js:

1) Position the shapes where we want them

2) Move the paper & draw shapes at the origin (0,0)

translate()

rotate()

scale()

AFFINE TRANSFORMATIONS

Functions that (essentially) move the 'graph paper':

translate() → move left/right or up/down

rotate() → rotate around the y-axis (2D)

scale() → make bigger or smaller

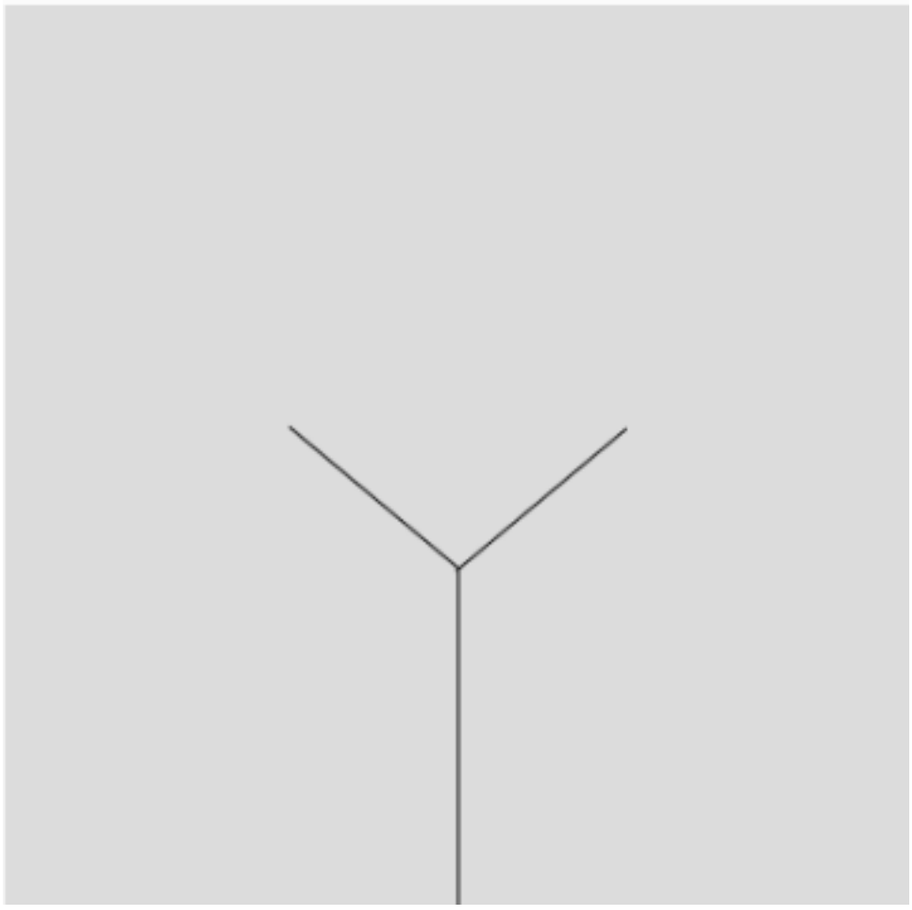
AFFINE TRANSFORMATIONS

These two are often used as well:

`push()` → 'save' the state of the coordinate system, so that we can return to it later

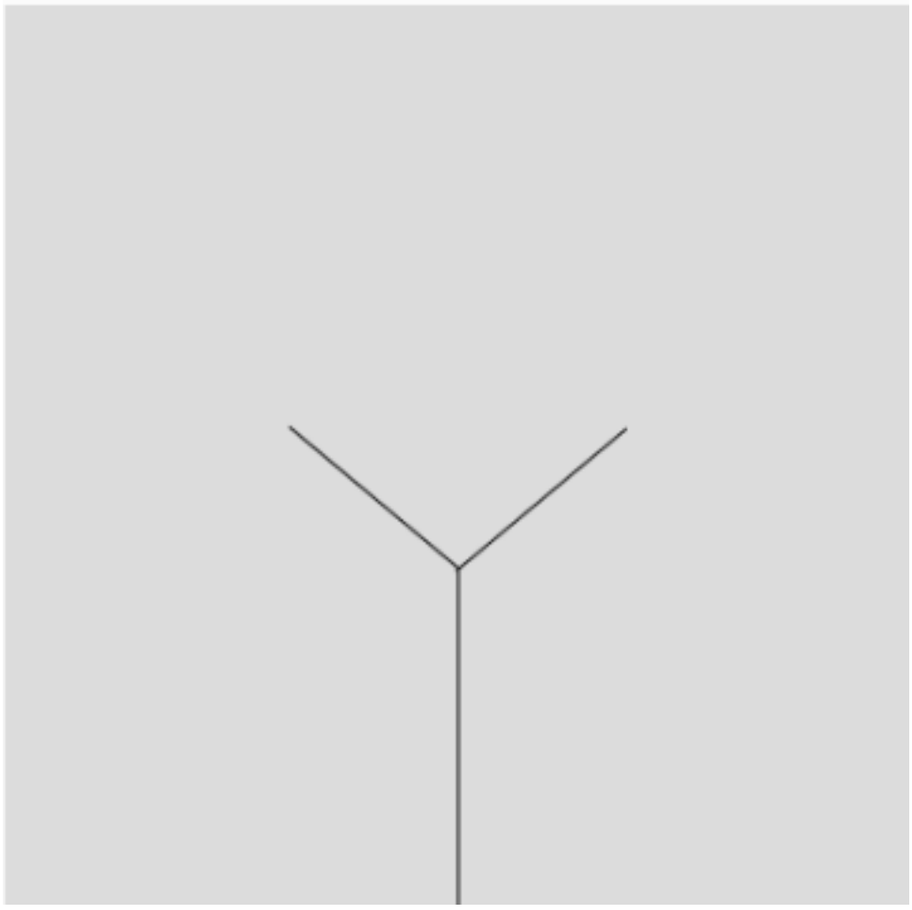
`pop()` → go back to the last 'saved' state for the coordinate system

FRACTAL TREE



translate(), rotate(), line()

Draw a branch. From its end, rotate right and draw a shorter branch. Then rotate left and do the same. Repeat for the two new branches...



translate(), rotate(), line()

FRACTAL TREE

1. Draw a line of a given length
2. From its end, rotate right and draw a shorter line
3. From its end, rotate left and draw a shorter line
4. Repeat for the two new lines...

FRACTAL TREE

A (recursive) branch function:

1. Draw a line of a given length
2. From its end, rotate right & **draw a shorter branch**
3. From its end, rotate left & **draw a shorter branch**
4. Repeat ...

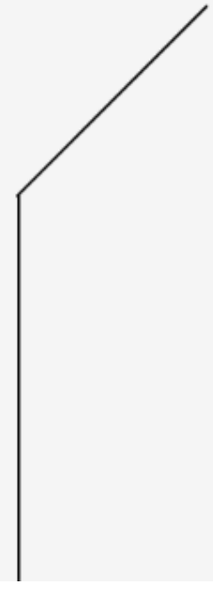
FRACTAL TREE

Auto-refresh Mute sign

sketch.js

Preview

```
function setup() {
  createCanvas(400, 400);
  background(245);

  translate(width/2, height); // move to bottom/center
  branch(150);                // call branch()
}

function branch(len) {

  line(0,0,0,-len);         // draw a line
  translate(0, -len);       // move to the end of it

  rotate(radians(45));      // rotate 45 degrees clockwise
  line(0,0,0,-len * 0.67); // draw a shorter line
}
```



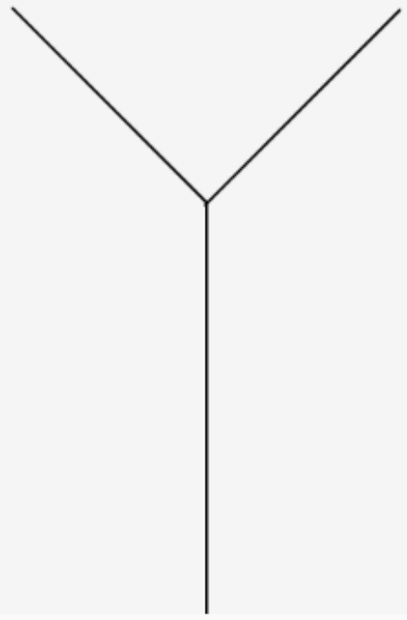
FRACTAL TREE

Auto-refresh Mute sign

sketch.js

Preview

```
function setup() {
  createCanvas(400, 400);
  background(245);

  translate(width/2, height); // move to bottom/center
  branch(150);                // then call branch
}

function branch(len) {

  line(0,0,0,-len);         // draw a line
  translate(0, -len);       // move to the end of it

  rotate(radians(45));      // rotate 45 degrees clockwise
  line(0,0,0,-len * 0.67); // draw a shorter line

  rotate(radians(-90));     // rotate 45 deg anticlockwise
  line(0,0,0,-len * 0.67); // draw a shorter line
}
```



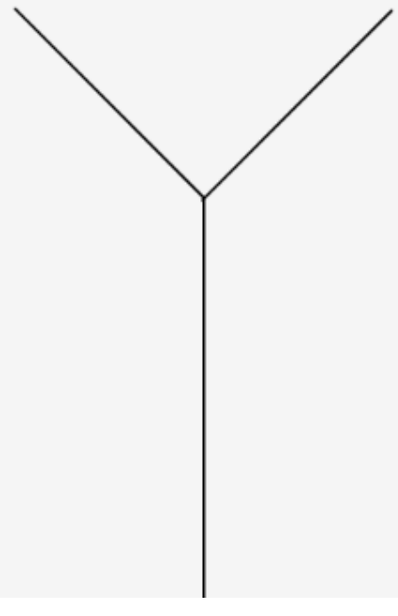
FRACTAL TREE

Auto-refresh Mute sign

sketch.js

Preview

```
function branch(len) {
  line(0,0,0,-len);          // draw a line
  translate(0, -len);        // move to the end of it

  push();
  rotate(radians(45));       // rotate 45 degrees clockwise
  line(0,0,0,-len * 0.67); // draw a shorter line
  pop();

  push();
  rotate(radians(-45));      // rotate 45 deg anticlockwise
  line(0,0,0,-len * 0.67); // draw a shorter line
  pop();
}
```

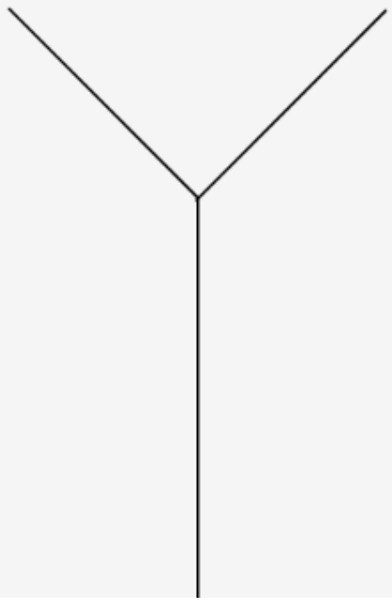


FRACTAL TREE

Auto-refresh Mute sign

sketch.js

```
function branch(len) {

  line(0,0,0,-len);          // draw a line
  translate(0, -len);        // move to the end of it

  push();
  rotate(radians(45));       // rotate 45 degrees clockwise
  branch(len * 0.67);        // draw a shorter *branch*
  pop();

  push();
  rotate(radians(-45));      // rotate 45 deg anticlockwise
  branch(len * 0.67);        // draw a shorter *branch*
  pop();
}
```

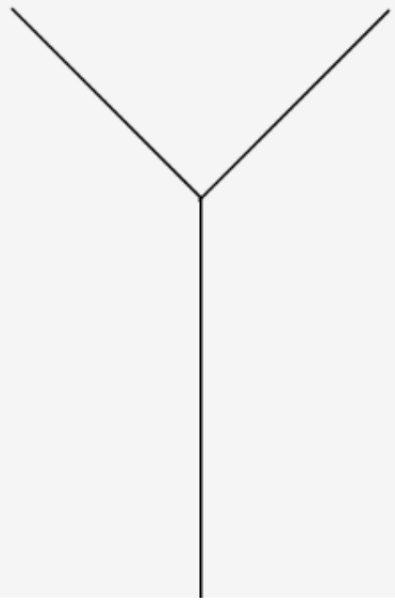


EXIT CONDITION ?

Auto-refresh Mute sign

sketch.js

```
function branch(len) {

  line(0,0,0,-len);          // draw a line
  translate(0, -len);        // move to the end of it

  push();
  rotate(radians(45));       // rotate 45 degrees clockwise
  branch(len * 0.67);        // draw a shorter *branch*
  pop();

  push();
  rotate(radians(-45));      // rotate 45 deg anticlockwise
  branch(len * 0.67);        // draw a shorter *branch*
  pop();
}
```



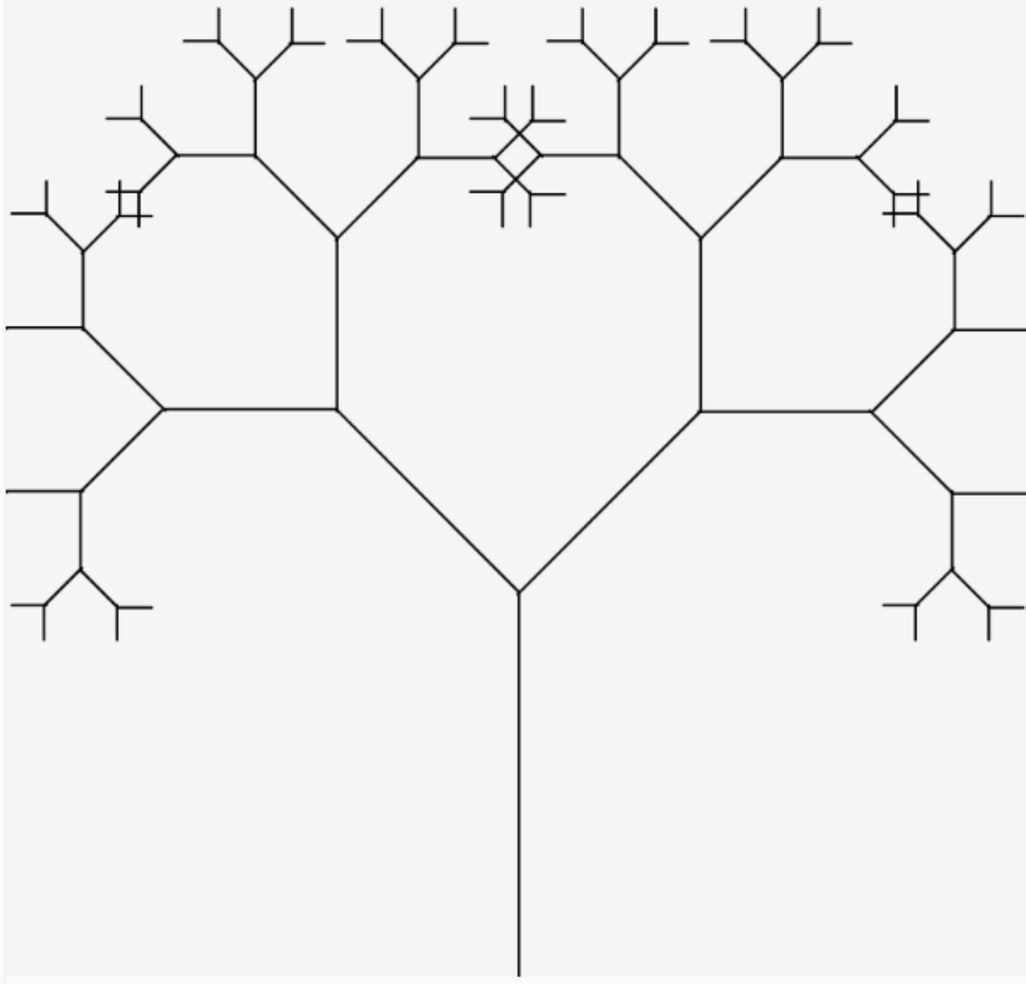
FRACTAL TREE

Auto-refresh Mute sign

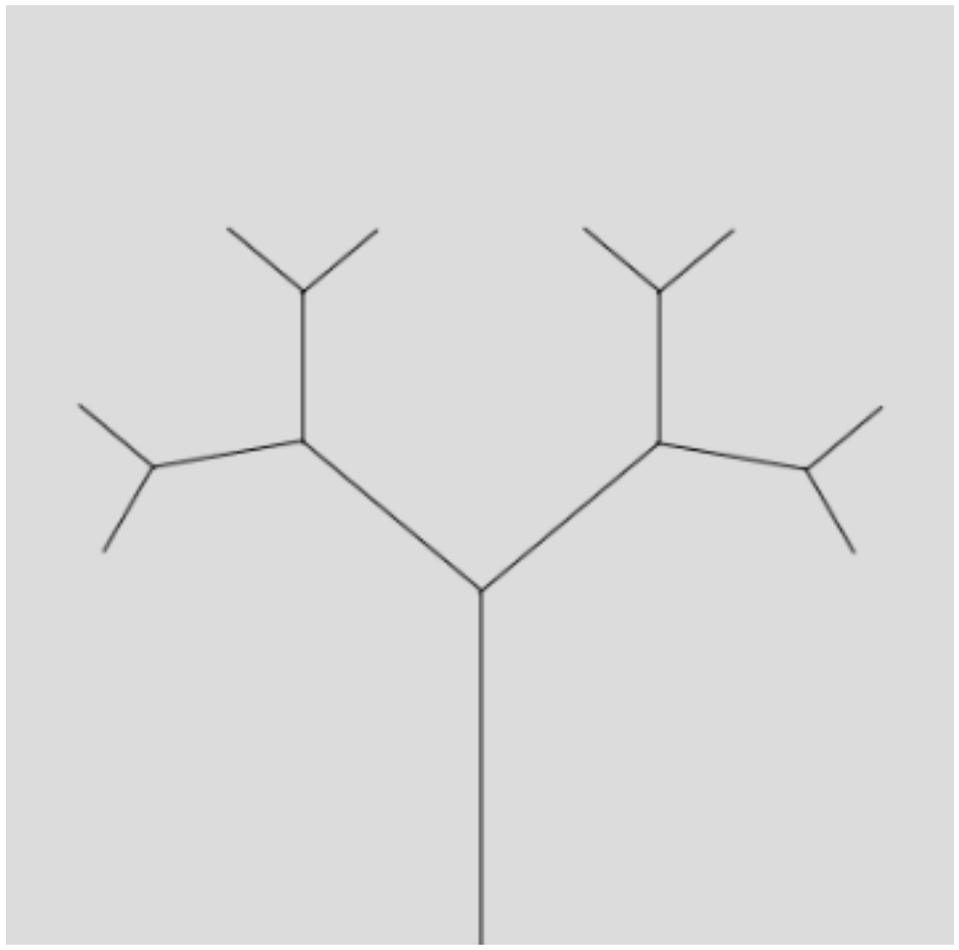
sketch.js

```
function branch(len) {

  if (len > 12)  {

    line(0,0,0,-len);          // draw a line
    translate(0, -len);        // move to the end of it

    push();
    rotate(radians(45));       // rotate 45 degrees clockwise
    branch(len * 0.67);        // draw a shorter *branch*
    pop();

    push();
    rotate(radians(-45));      // rotate 45 deg anticlockwise
    branch(len * 0.67);        // draw a shorter *branch*
    pop();
  }
  else {
    // do nothing
  }
}

```

Preview

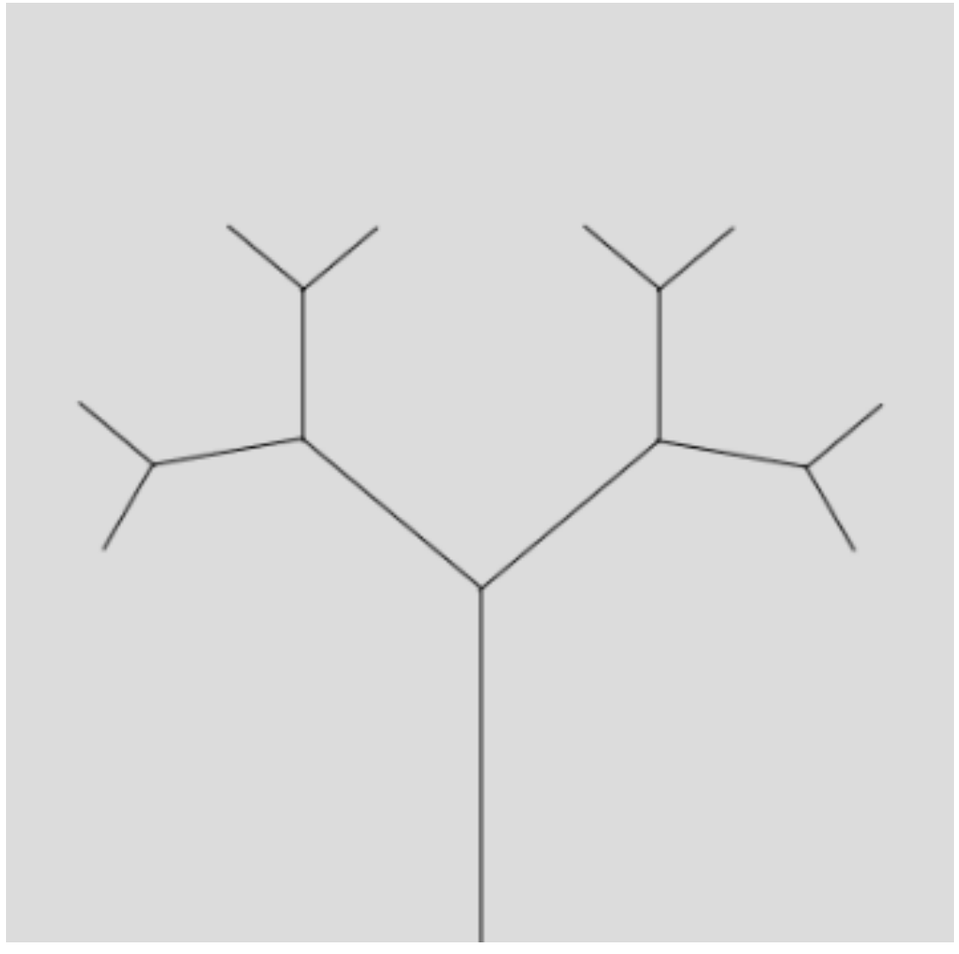


FRACTAL TREE



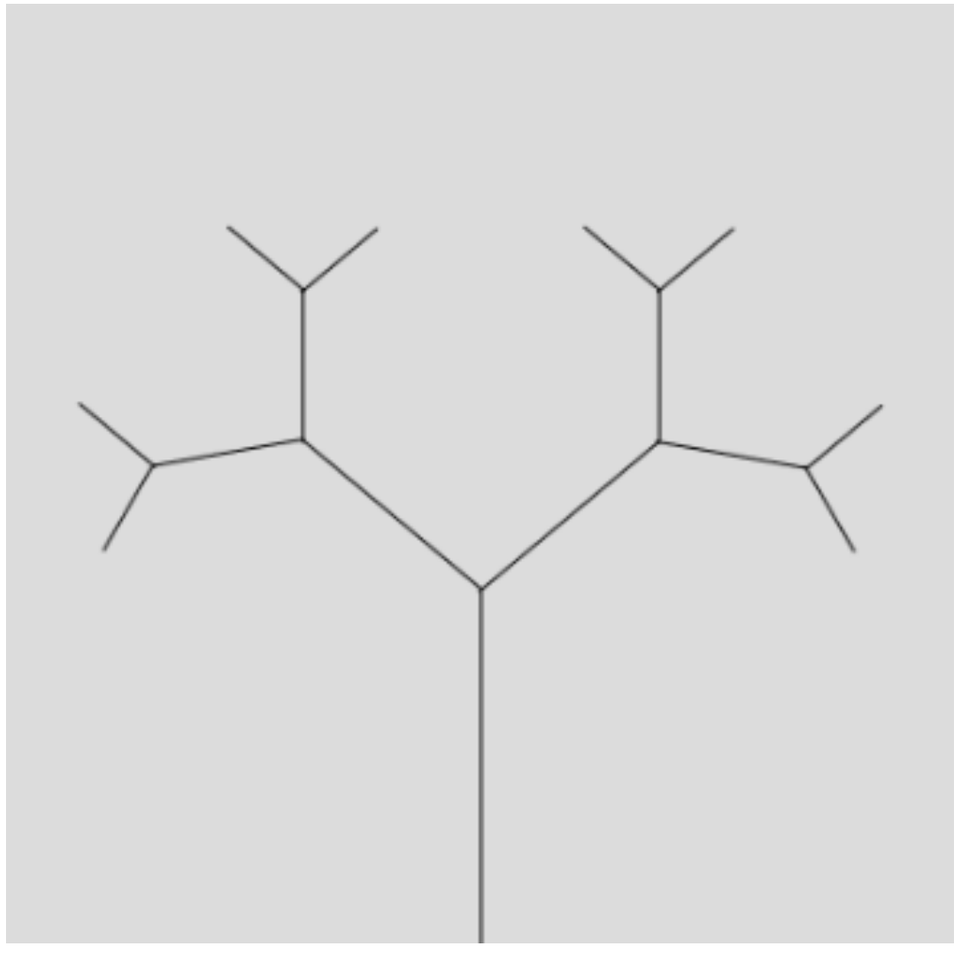
now lets experiment with our variables: angle & lengthScale

EXPLORING A PARAMETER SPACE



we now have two variables: angle and lenScale - how can we *quickly* explore all their possible combinations?

MAPPING THE MOUSE POSITION



we can map() the angle to mouseX
and the lenScale to mouseY

THE MAP() FUNCTION

Description

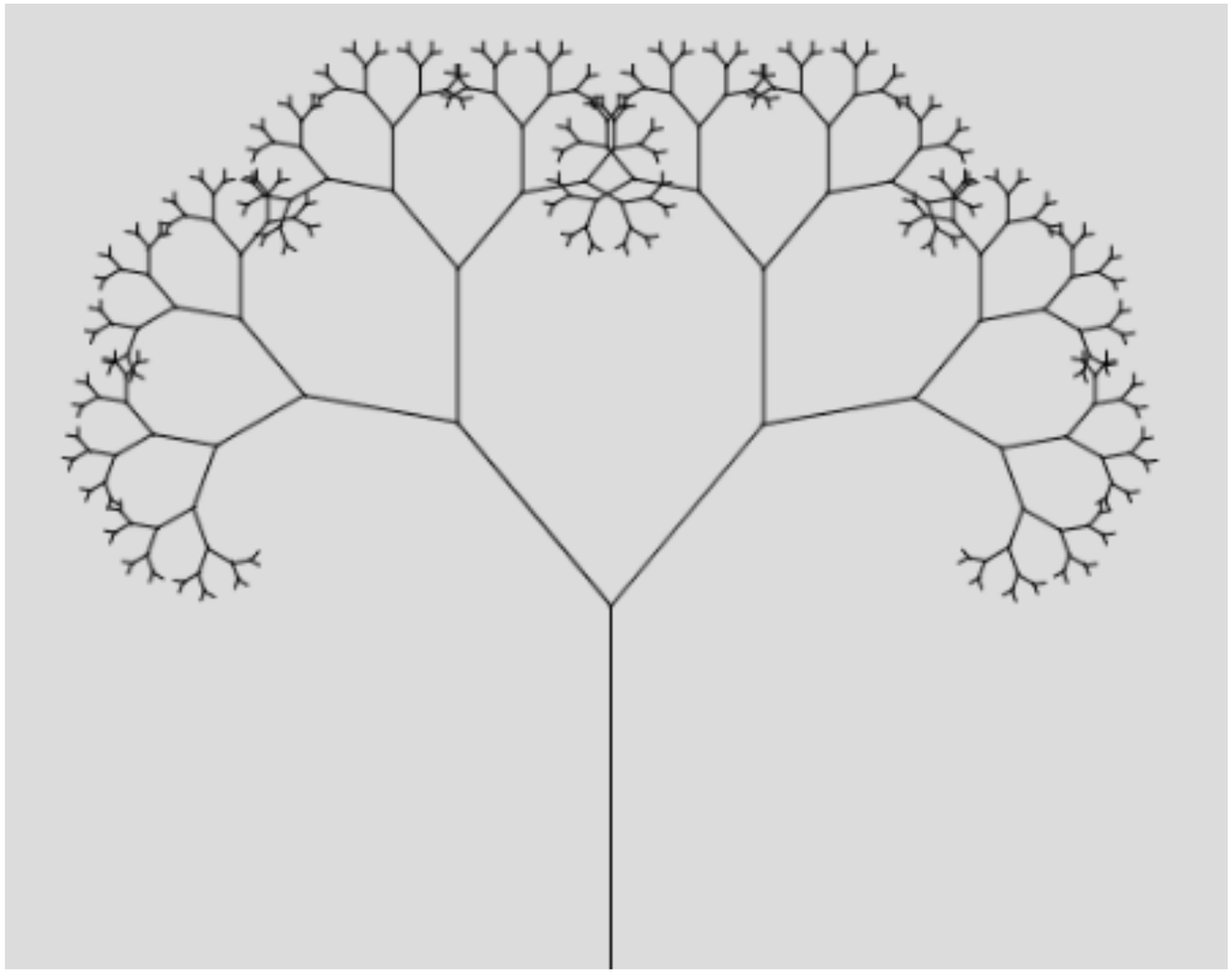
Re-maps a number from one range to another.

```
map(value, start1, stop1, start2, stop2);
```

Parameters

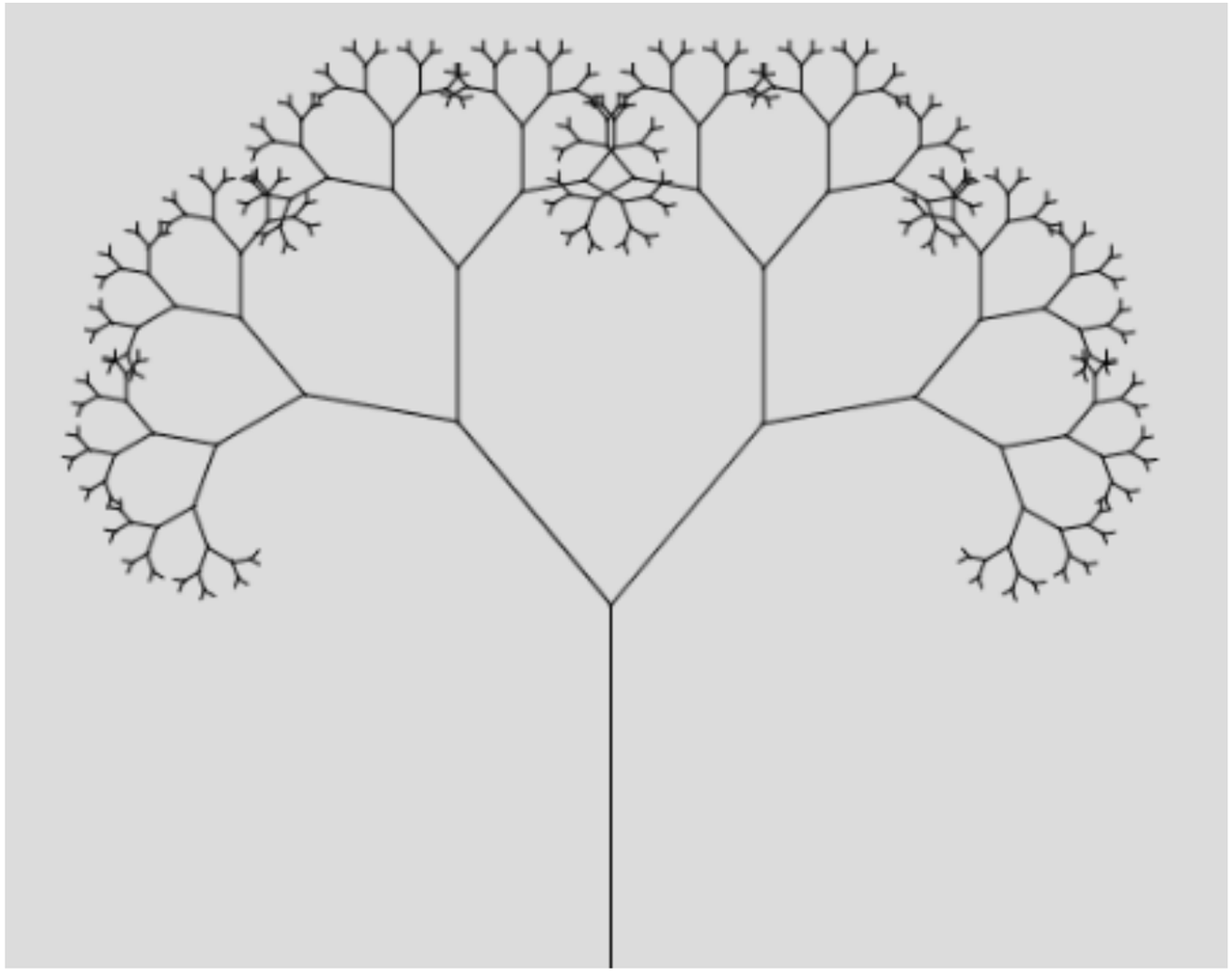
`value`	Number: the incoming value to be converted
`start1`	Number: lower bound of the value's current range
`stop1`	Number: upper bound of the value's current range
`start2`	Number: lower bound of the value's target range
`stop2`	Number: upper bound of the value's target range

FRACTAL TREE



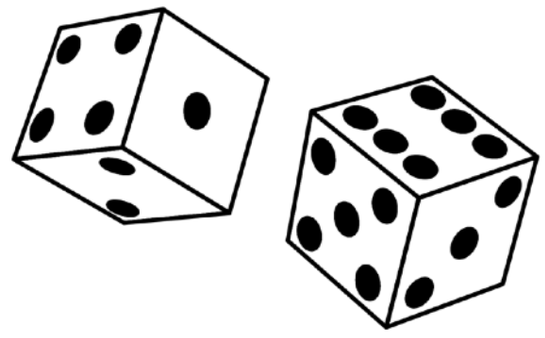
whats the problem with this tree?

FRACTAL TREE



lets use random() variation to make it look more natural...

THE RANDOM() FUNCTION



Description

Return a random floating-point number.

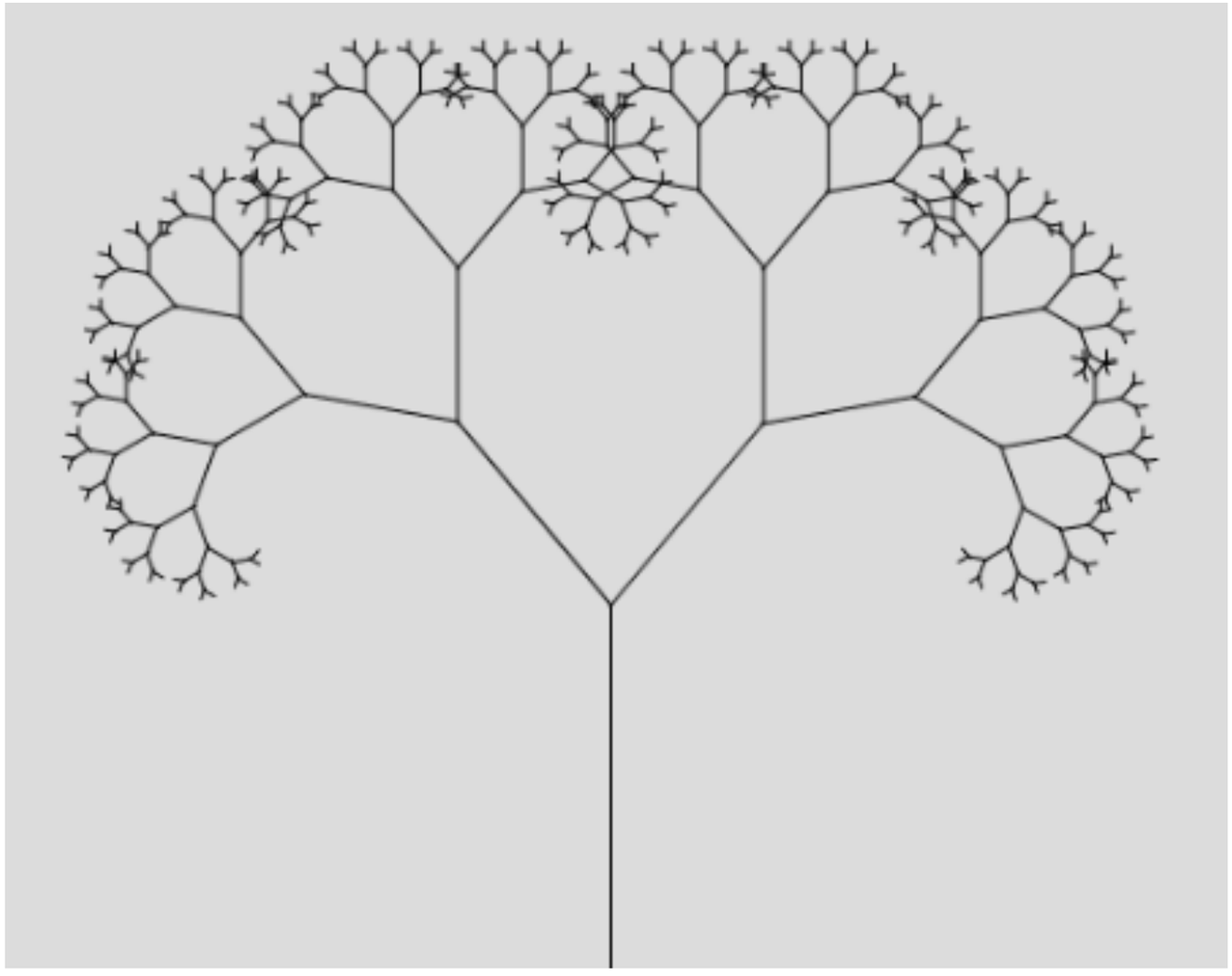
Takes either 0, 1 or 2 arguments.

If no argument is given, returns a random number from 0 up to (but not including) 1.

If one argument is given and it is a number, returns a random number from 0 up to (but not including) the number.

If two arguments are given, returns a random number from the first argument up to (but not including) the second argument.

FRACTAL TREE



now lets use noise() to give it some natural motion...

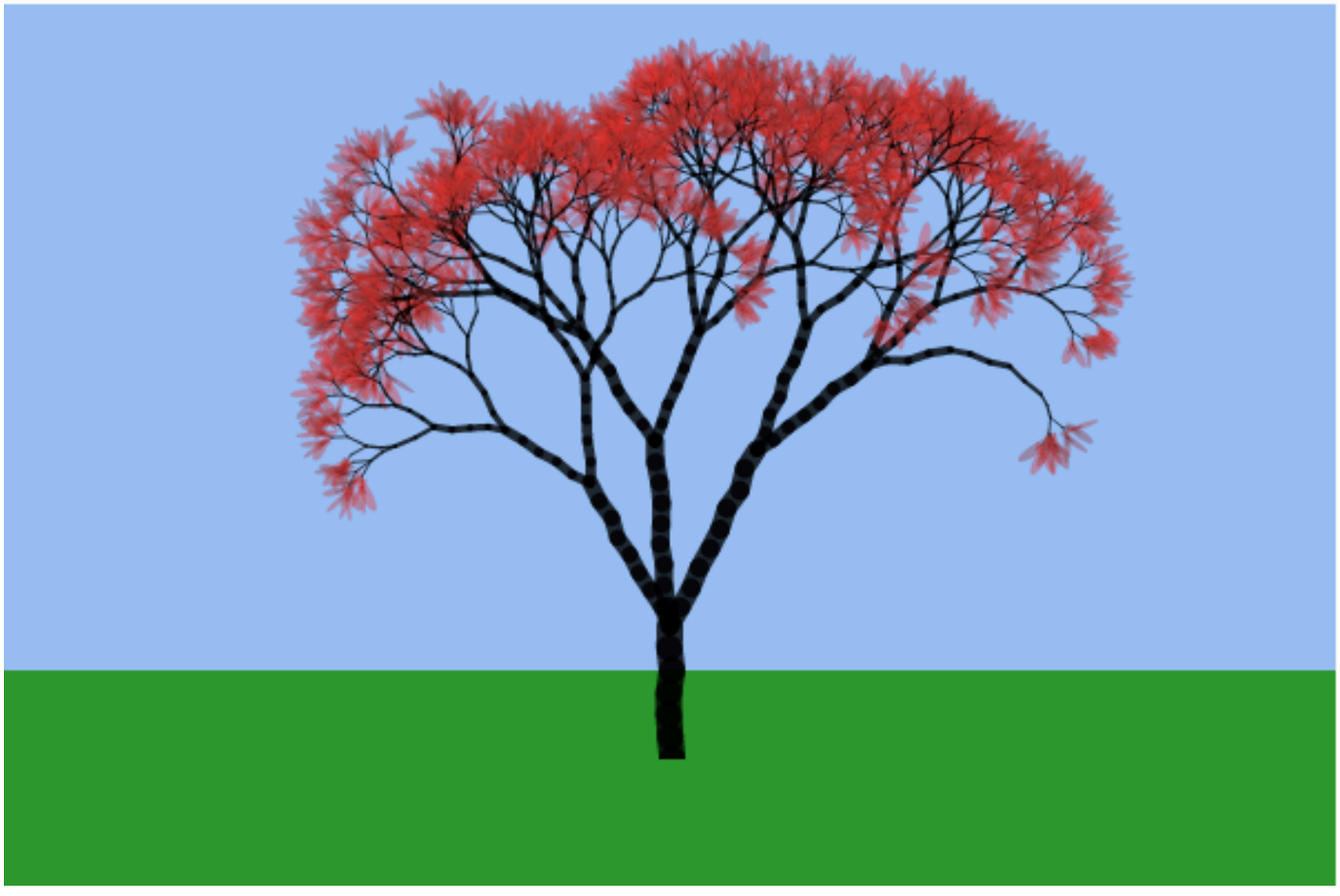
THE NOISE() FUNCTION

Description

Perlin noise is a random sequence generator producing a more natural ordered, harmonic succession of numbers compared to the standard random() function.

```
// 1d noise, 'value' should increase slowly
output = noise(value);
```

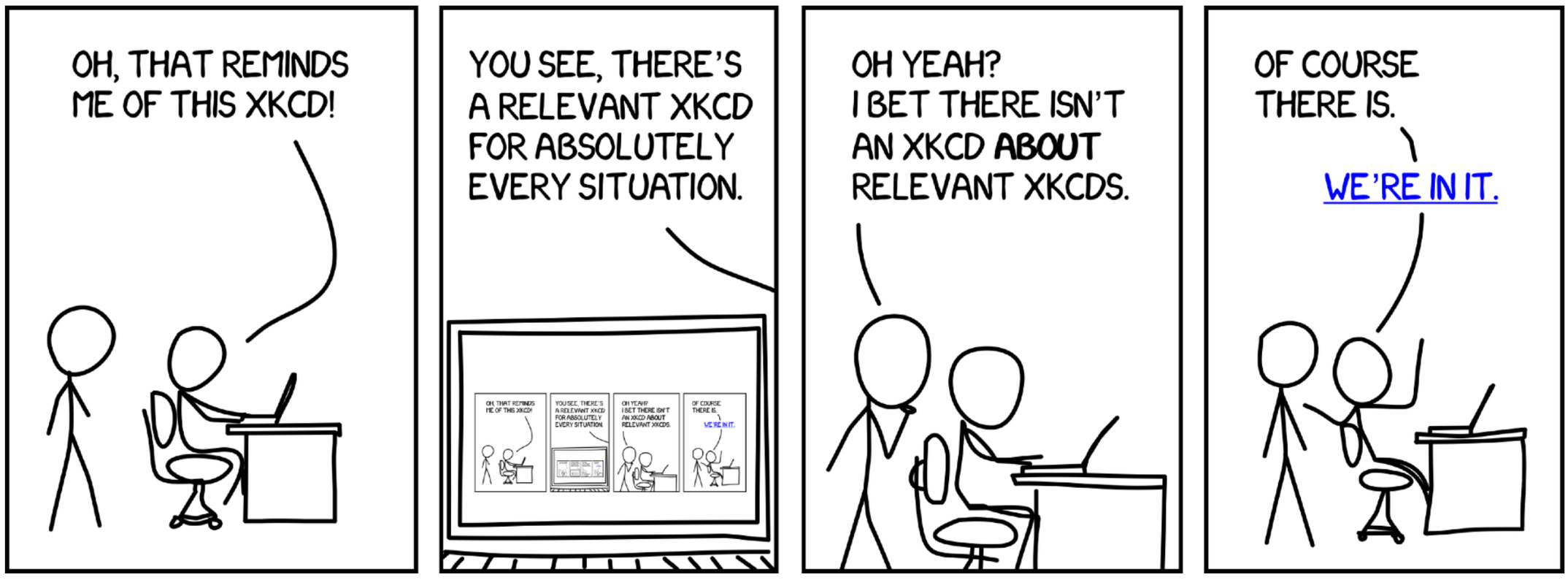
FRACTAL TREE



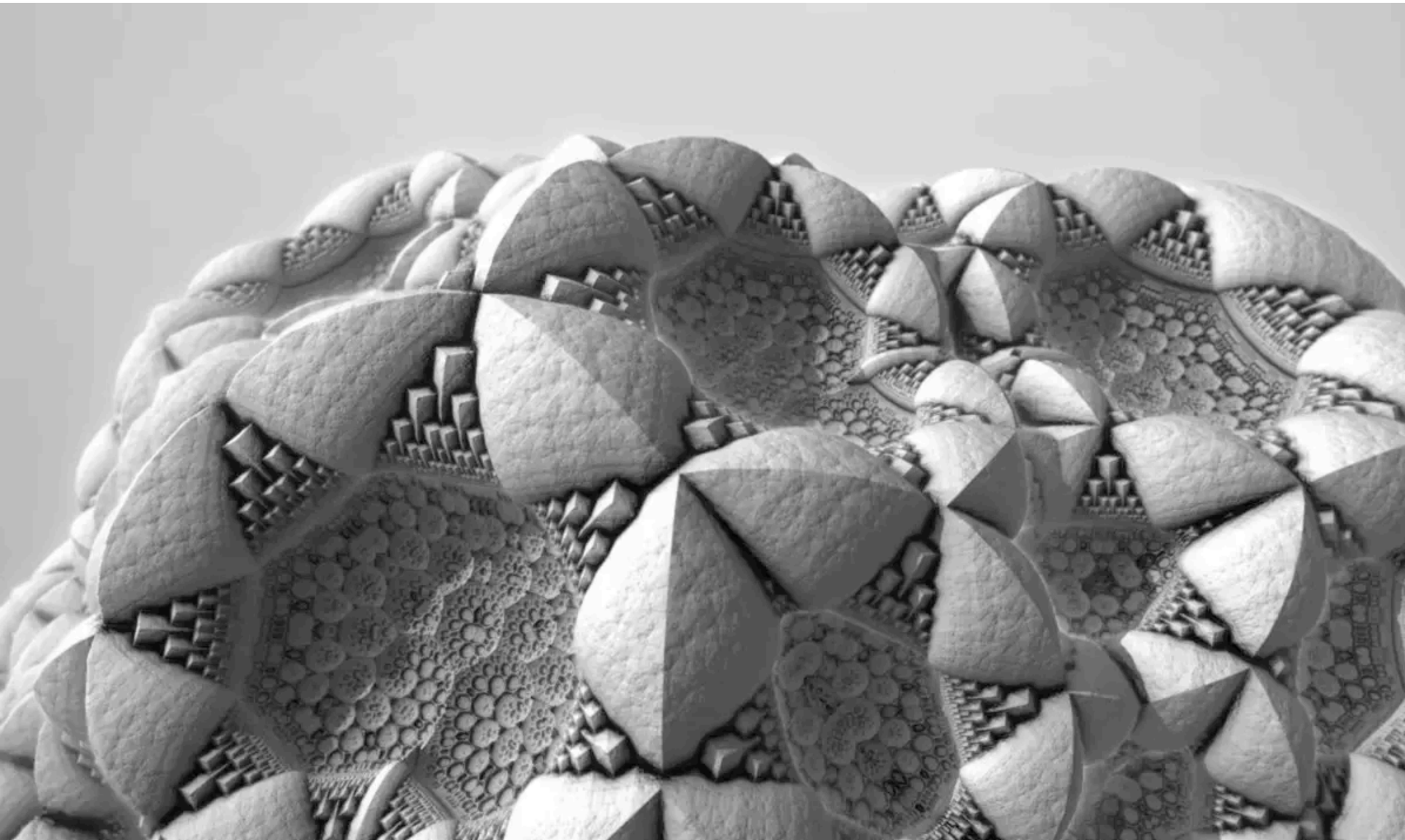
RECURSION
FRACTALS
AFFINE TRANSFORMS
RANDOM
NOISE

re·cur·sion /ri-ˈkər-zhən/

n. When an entity is defined by one or more references to itself.



FRACTALS



AFFINE TRANSFORMATIONS

Functions that (essentially) move the 'graph paper':

translate() → move left/right or up/down

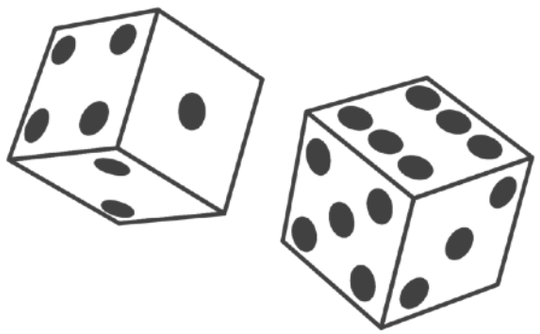
rotate() → rotate around the y-axis (2D)

scale() → make bigger or smaller

push() → 'save' state of the coordinate system

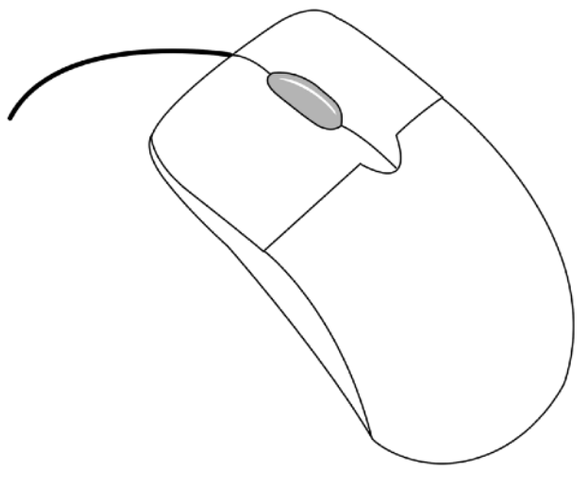
pop() → go back to the last 'saved' state

MAPPING (RANDOM) VARIABLES

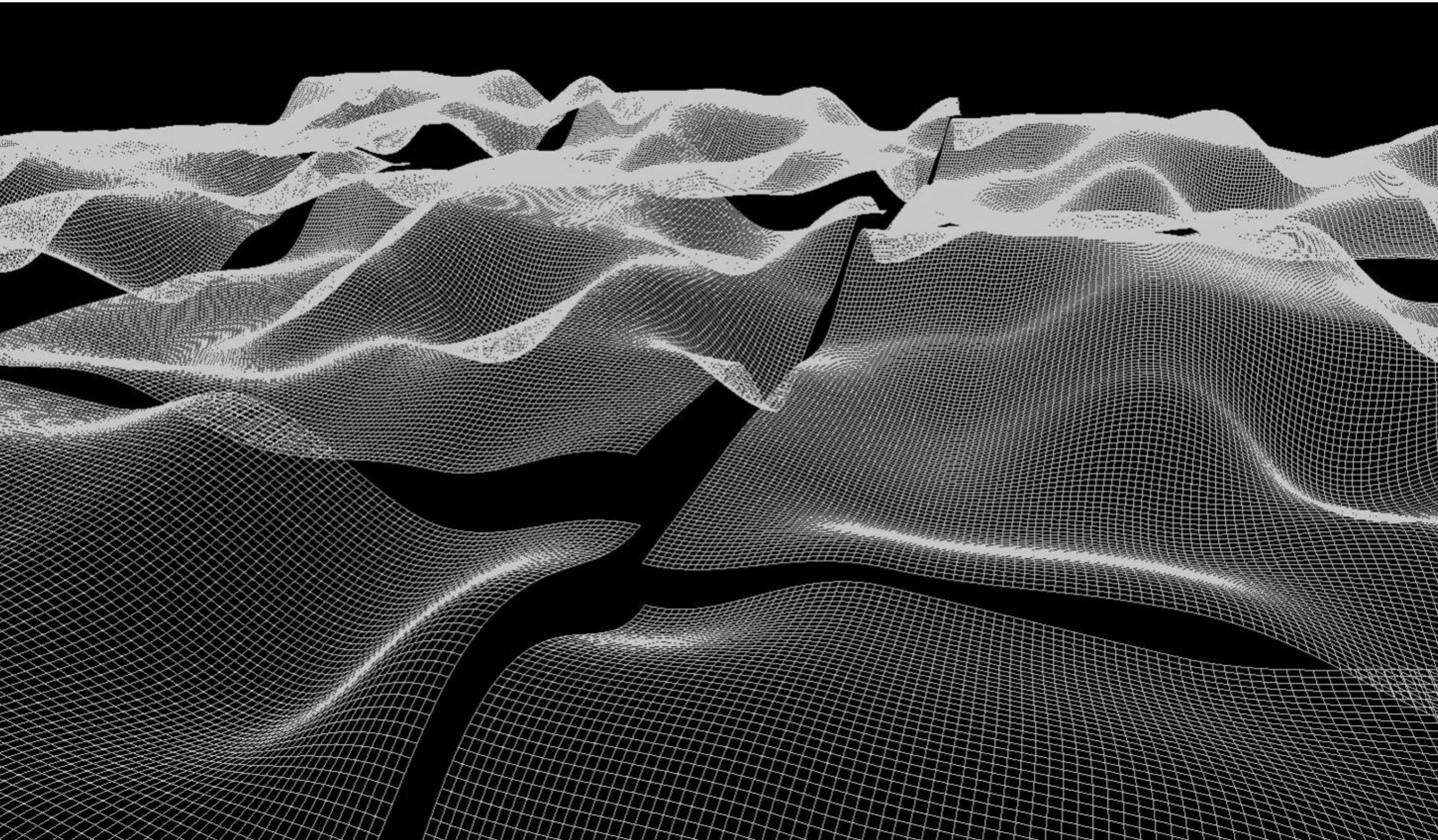


Maps a number from one range to another.

```
map(value, start1, stop1, start2, stop2);
```



PERLIN NOISE



END

DANIEL C. HOWE
email: daniel@rednoise.org
web: https://rednoise.org/daniel
twitter/mastodon: @danielchowe