



Declarative Amsterdam 23

Talking about my
Generations

our

Lindenmayer

A close-up photograph of a green, textured surface, possibly a leaf or a piece of fabric, with the name 'Lindenmayer' overlaid in a purple font. The texture consists of numerous small, rounded, green protrusions that create a bumpy, organic appearance. The lighting is soft, highlighting the individual bumps and creating a sense of depth. The background is dark, making the green texture stand out.

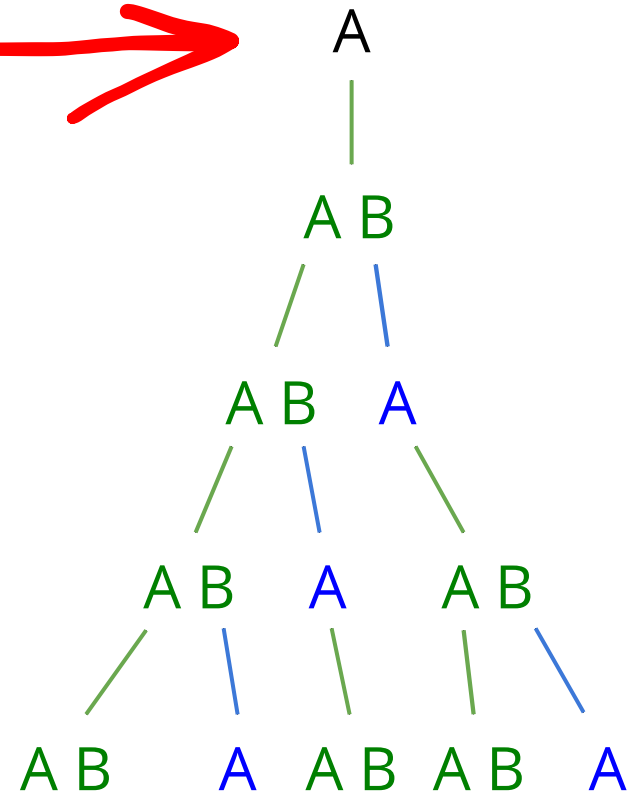
start with the symbol A



per iteration

replace any A with A and B

replace any B with A



1

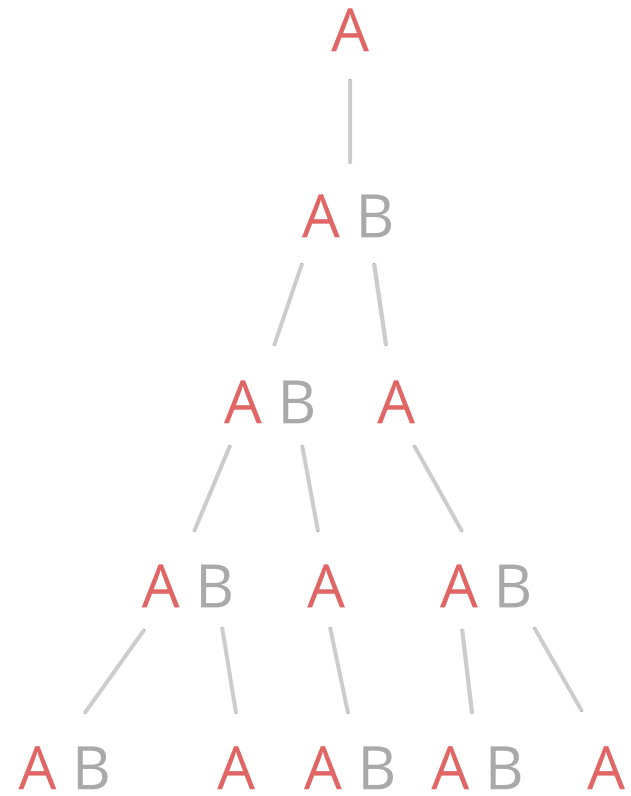
1

2

3

5

?



·L· Systems · 

Alphabet **A** **B**

Axiom **A**

Production Rules **A** \rightarrow **A** **B**
B \rightarrow **A**

XML Representation In Linsy

```
<system>  
  <axiom>A</axiom>  
  <grammar>  
    <variable match="A">AB</variable>  
    <variable match="B">A</variable>  
  </grammar>  
</system>
```


XML Representation In Linsy

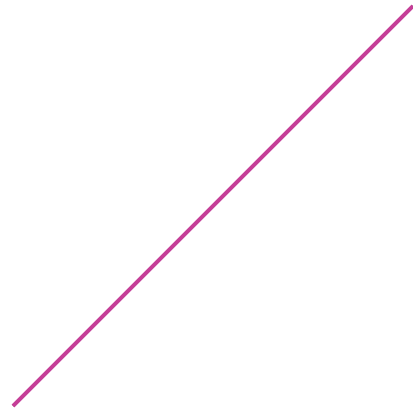
```
<system>
  <axiom>0</axiom>
  <grammar>
    <variable match="0">1[0]0</variable>
    <variable match="1">11</variable>
    <terminal match="[" />
    <terminal match="]" />
  </grammar>
</system>
```

Production Rules

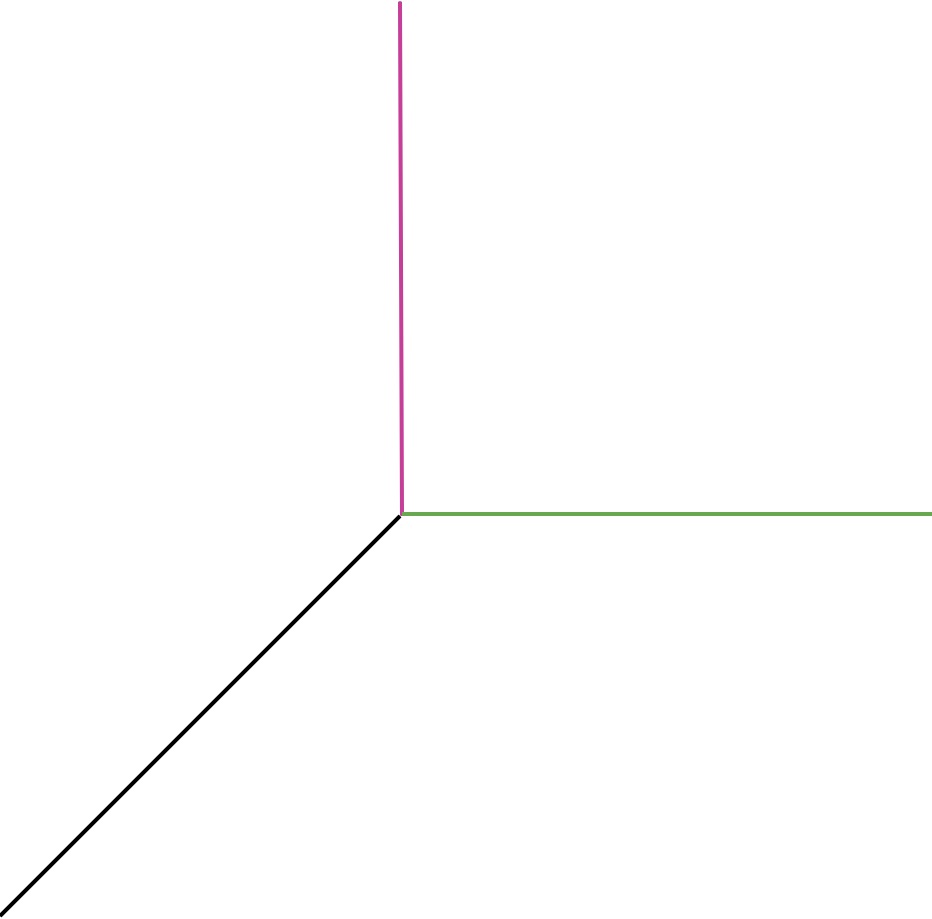
Binary Tree Production Rules

- 1 1 = draw line forward
- 2 0 = draw line forward
- 3 [= push *state* and turn right (branch)
- 4] = pop *state* and turn left (end branch)

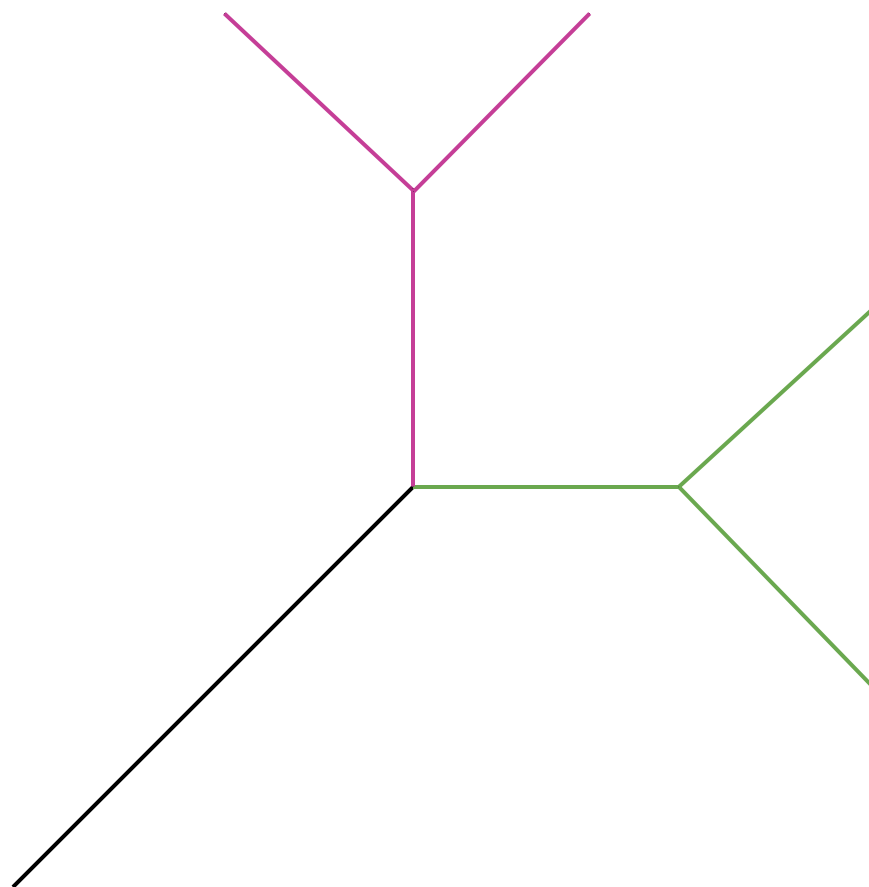
0



1 [0] 0



11[1[0]0]1[0]0



Render

```
1 <render viewBox="-120 0 240 240">
2     <state x="0" y="240" orientation="90"
3         angle="45" velocity="16" />
4 </render>
```


Inkscape

```
1 A-F = draw line forward
2 G-L = move forward
3 +   = turn right
4 -   = turn left
5 |   = turn around
6 [   = push state (branch)
7 ]   = pop state (end branch)
```

```
1 <system>
2   <axiom>B</axiom>
3   <grammar>
4     <variable match="A">AA</symbol>
5     <variable match="B">A[+B]-B</symbol>
6     <terminal match="+" />
7     <terminal match="-" />
8     <terminal match="[" />
9     <terminal match="]" />
10  </grammar>
11  <render />
12 </system>
```

```
1 <system iterations="3">
2   <axiom>B</axiom>
3   <grammar>
4     <variable match="A">AA</symbol>
5     <variable match="B">A[+B]-B</symbol>
6     <terminal match="+" />
7     <terminal match="-" />
8     <terminal match="[" />
9     <terminal match="]" />
10  </grammar>
11  <render viewBox="-120 0 240 240">
12    <state x="0" y="240" angle="45" velocity="16" orientation="90" />
13  </render>
14 </system>
```


paulbourke.net/fractals/lsys/

- # Increment the line width (by line width increment)
- ! Decrement the line width (by line width increment)
- @ Draw a dot (with line width radius)
- { Open a polygon
- } Close a polygon (and fill it with fill colour)
- > Multiply the line length (by the line length scale factor)
- < Divide the line length (by the line length scale factor)
- & Swap the meaning of + and -
- (Decrement turning angle (by turning angle increment)
-) Increment turning angle (by turning angle increment)

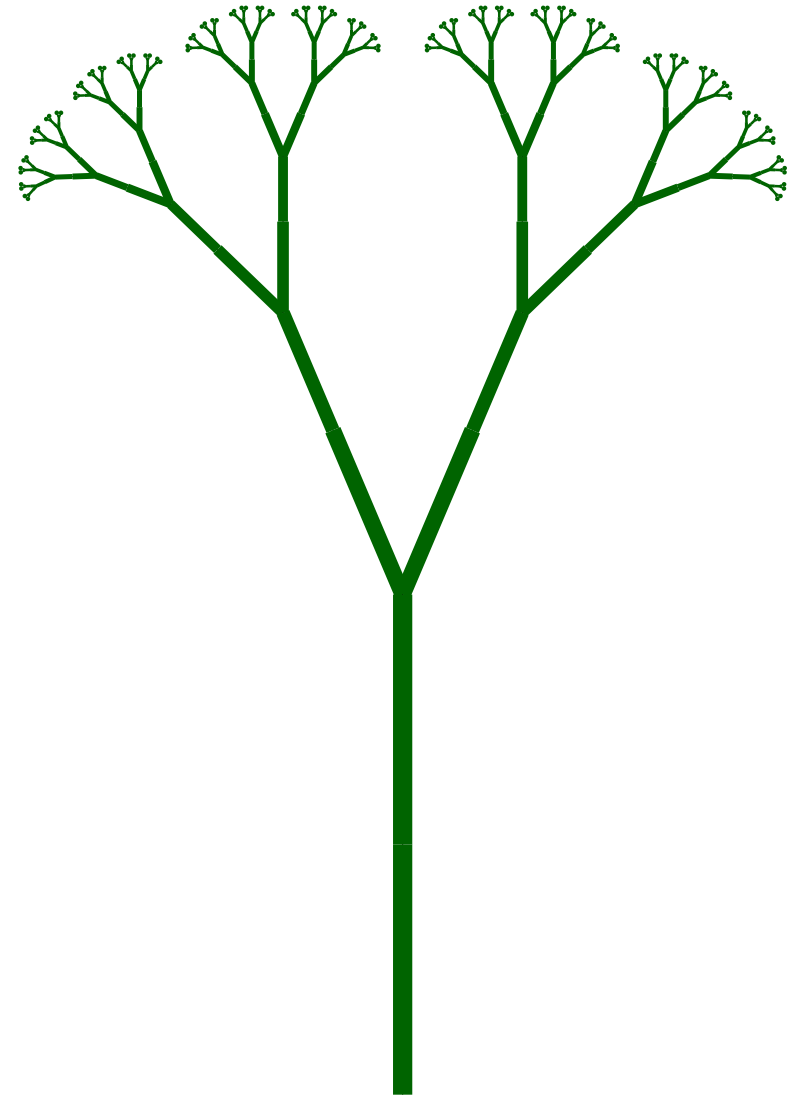
axiom: A@

alphabet: A @ - + [] < >

rules:

A → <A>

@ → <A> [+A>@]-A>@



```
1 <system iterations="7">
2   <axiom>A@</axiom>
3   <grammar>
4     <variable symbol="A">&lt;A&gt;</variable>
5     <variable symbol="@">&lt;A&gt;[+A&gt;@]-A&gt;@</variable>
6     <terminal symbol="&lt;" />
7     <terminal symbol="&gt;" />
8     <terminal symbol="[" />
9     <terminal symbol="]" />
10    <terminal symbol="+" />
11    <terminal symbol="-" />
12  </grammar>
13  <render viewBox="0 0 800 800">
14    <state x="400" y="800"
15      velocity="16"
16      angle="23" orientation="90"
17      acceleration="1.4"
18      color="darkgreen"
19    />
20  </render>
21 </system>
```

```
1 declare function render:symbol (  
2     $state as map(*),  
3     $next-symbol as xs:string  
4 ) as map(*) {  
5     switch($next-symbol)  
6         case "A" case "B" case "C" case "D" case "E" case "F"  
7             return render:line($state)  
8         case "G" case "H" case "I" case "J" case "K" case "L"  
9             return render:move($state)  
10        case "@" return render:circle($state)  
11        case "-" return render:turn-left($state)  
12        case "+" return render:turn-right($state)  
13        case "[" return render:push-stack($state)  
14        case "]" return render:pop-stack($state)  
15        case "<" return render:increase-velocity($state)  
16        case ">" return render:decrease-velocity($state)  
17        default return error()  
18 };
```

stochastic

```
axiom: X
angle: 25
alphabet: F X [ ] + -
rules:
  X → F+[[X]-X]-F[-FX]+X (75%)
  X → F-F[-FX]+[[X]-X]+X (25%)
  F → FF (95%)
  F → FFF (5%)
```



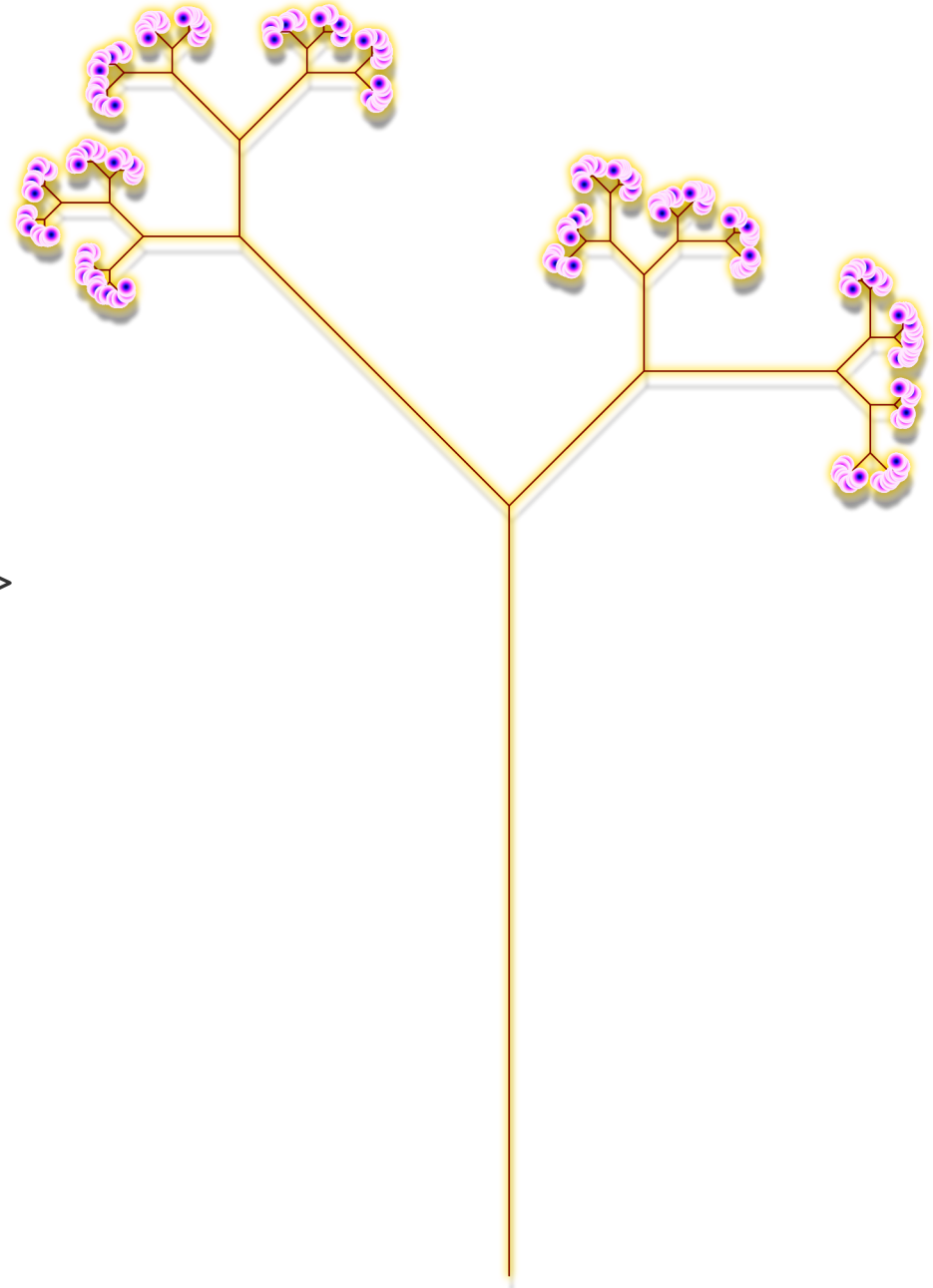
```
1 <system iterations="4">
2   <axiom>0</axiom>
3   <grammar type="stochastic">
4     <variable match="1">11</variable>
5     <variable match="0">
6       <option>1[0]0</option>
7       <option>11[0]0</option>
8       <option>0</option>
9     </variable>
10    <terminal match="[" />
11    <terminal match="]" />
12  </grammar>
13 </system>
```

```
let $random := 0.3456789
```

```
1 [0.33333333333333333333, ("1", "[", "0", "]", "0")],  
2 [0.66666666666666666667, ("1", "1", "[", "0", "]", "0")],  
3 [1.0, "0"]
```

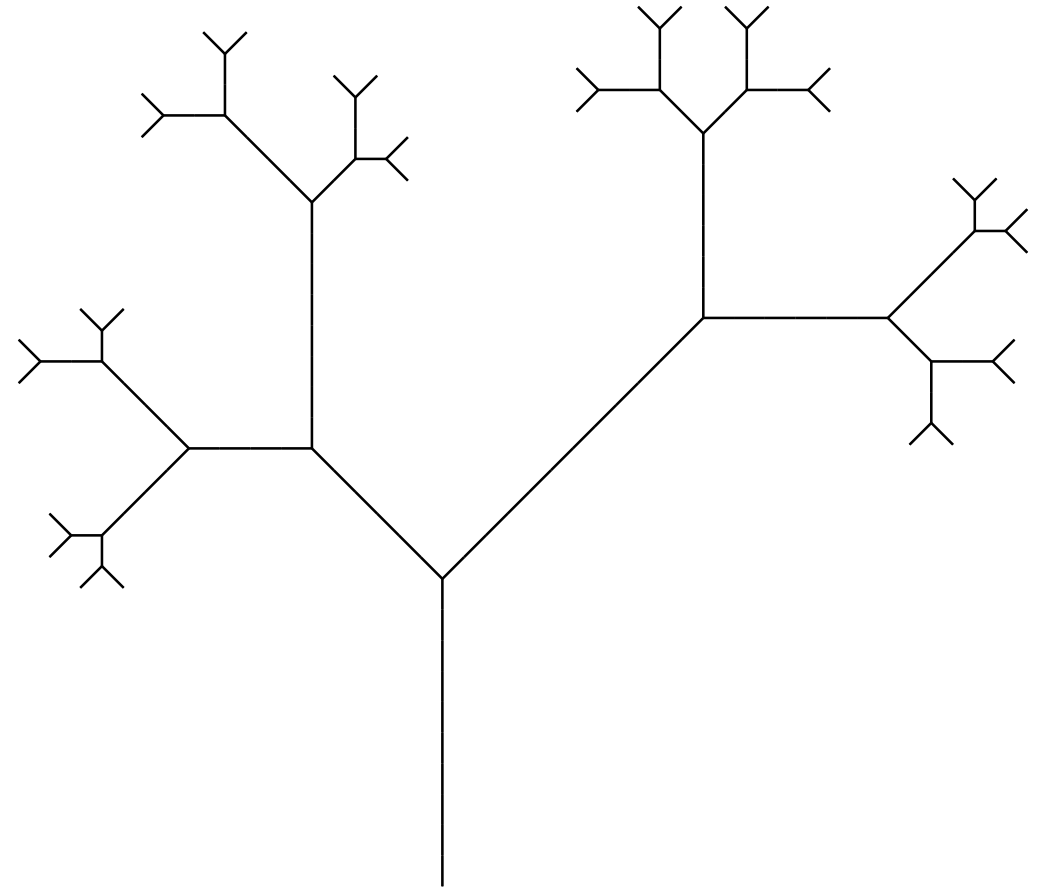
But weight...

```
<variable match="0">  
  <option weight="10">1[0]0</option>  
  <option weight="10">11[0]0</option>  
  <option weight="1">0</option>  
</variable>
```



Growing Seeds

```
1 <system iterations="5" seed="23">  
2   <axiom>A</axiom>  
3   <grammar type="stochastic">  
4     ...  
5   </grammar>  
6   <render ... />  
7 </system>
```

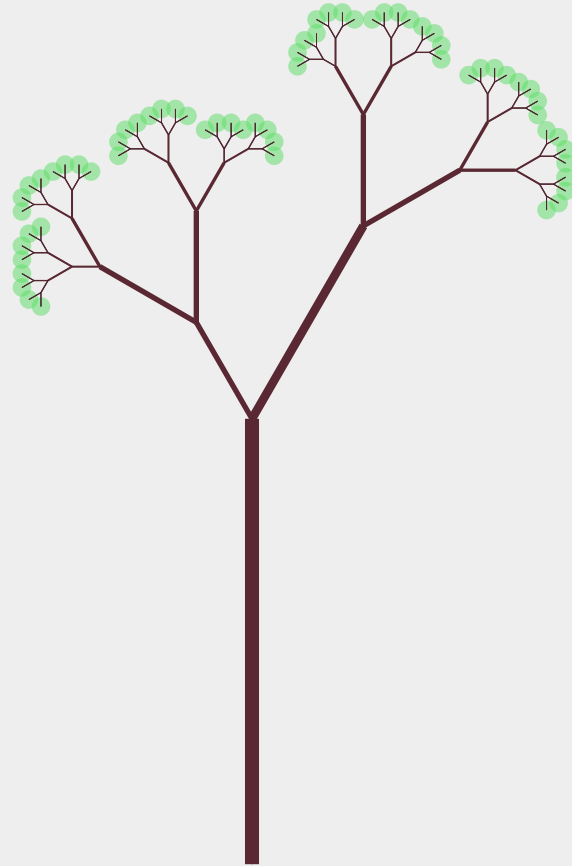


A small, light-colored toy car with black wheels is lying on its side on a grey asphalt surface. The car is positioned in the center of the frame, and its body is partially obscured by the text.

Performance & Stack Overflows

```
1 <grammar>
2   <variable match="0">1[0]0</variable>
3
4   <variable match="1">2</variable>
5   <variable match="2">3</variable>
6   <variable match="3">4</variable>
7   ...
8   <variable match="9">a</variable>
9   <variable match="a">b</variable>
10  <variable match="b">c</variable>
11  <variable match="d">e</variable>
12  <variable match="e">f</variable>
13
14  <terminal match="[" />
15  <terminal match="]" />
16 </grammar>
```

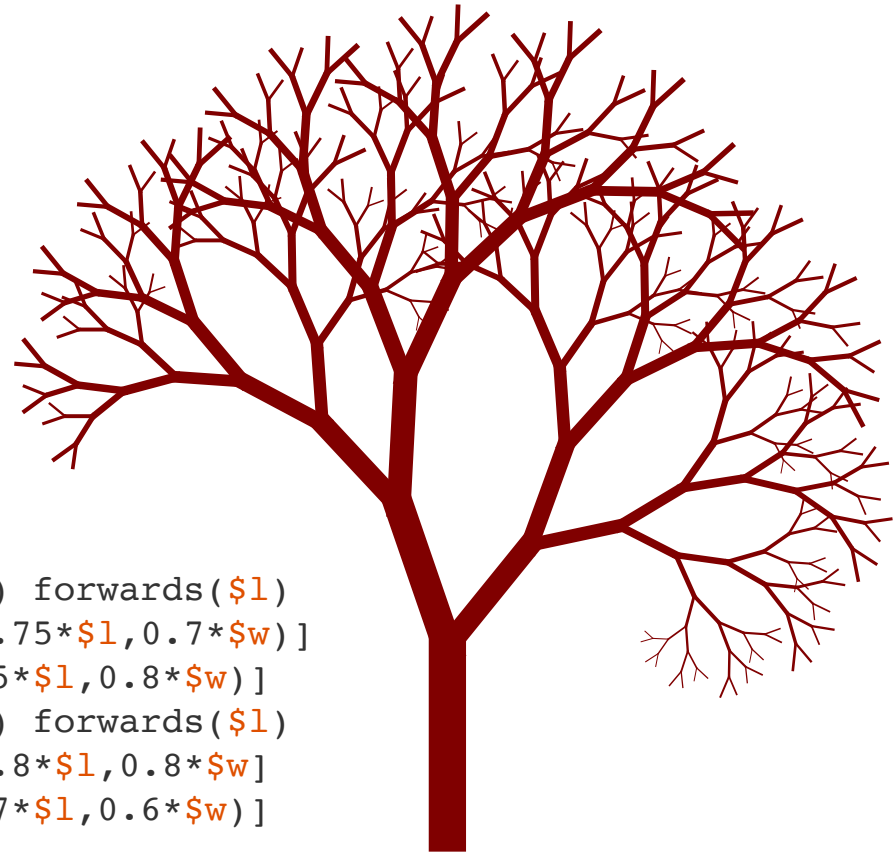
a = 30



```
6442210]0]10]0]210]0]10]0]3210]0]10]0]210]0]10]0]43210]0]10]0]210]0]10]0]3110]0]  
10]0]210]0]10]0]543110]0]10]0]210]0]10]0]3210]0]10]0]210]0]10]0]43210]0]10]0]210  
]0]10]0]3210]0]10]0]210]0]10]0
```

```
1 declare function local:draw ($state as map(*), $next-symbol as xs:string) {
2     switch($next-symbol)
3         case "0" return $state => local:line(0) => local:leaf()
4         case "1" return $state => local:line(0)
5         case "2" return $state => local:line(1)
6         case "3" return $state => local:line(2)
7         case "4" return $state => local:line(3)
8
9         case "f" return $state => local:line(14)
10        case "[" return $state => render:pop-stack() => render:turn-right()
11        case "]" return $state => render:push-stack() => render:turn-left()
12        default return error()
13 };
```

parametric



```
axiom: B(200,40)
A($l,$w) = width($w) forwards($l)
  [rotate(-24) B(0.75*$l,0.7*$w)]
  [rotate(22)A(0.85*$l,0.8*$w)]
B($l,$w) = width($w) forwards($l)
  [rotate(-19) A(0.8*$l,0.8*$w)]
  [rotate(39) B(0.7*$l,0.6*$w)]
```

Evaluation in generation

$A(\$n) = \text{rotate}(\$a), [\text{move}(\$n, \$s) \text{shape}(\$s)] A(\$n+1)$

"A(0)"

"rotate(136), [move(0,12)shape(12)]A(1)"

"rotate(136), [move(0,12)shape(12)]rotate(136), [move(1,12)shape(12)]A(2)"

Evaluation in interpretation

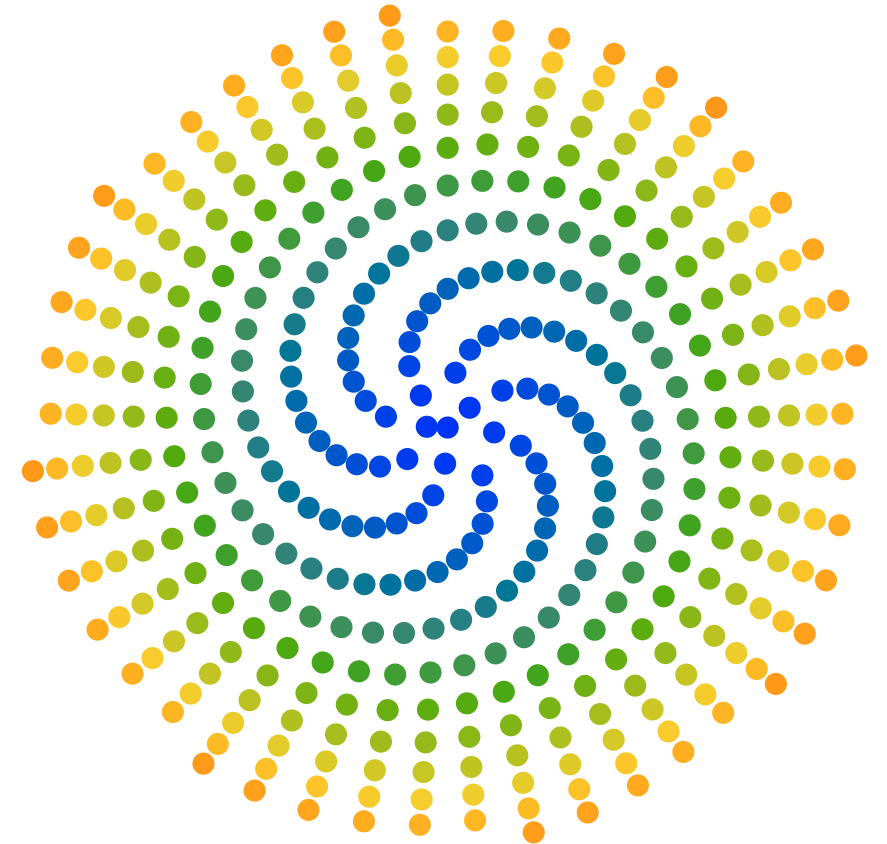
"rotate(136), [move(0,12)shape(12)]rotate(136), [move(1,12)shape(12)]A(2)"

Interpretation functions for each functional form

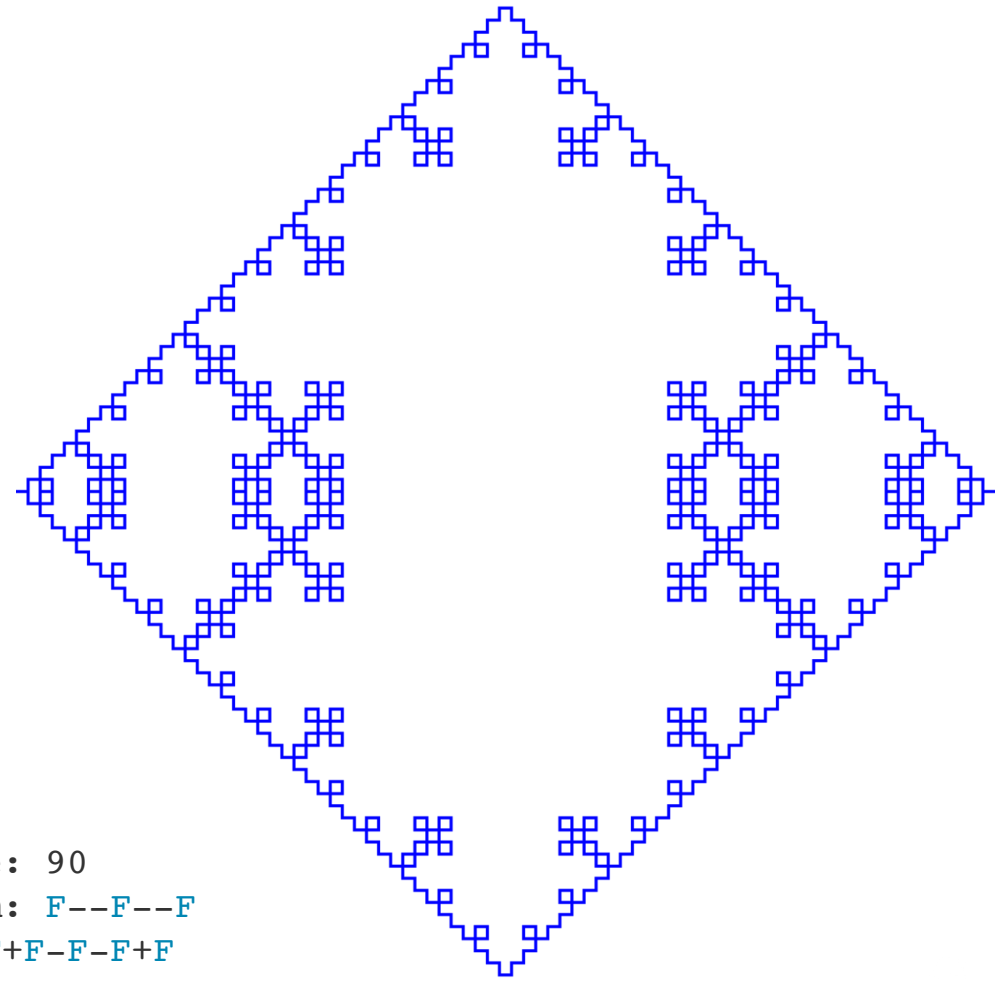
A central point at the bottom of the diagram has seven arrows pointing upwards to different parts of the code string above. The arrows point to the opening quote, the first 'rotate', the first '[' bracket, the first 'shape', the second 'rotate', the second '[' bracket, and the closing quote.

Putting it all together

```
1 let $axiom := "A(0)"
2 let $rules := map {
3   "A($n)": "rotate($a), [move($n,$s)shape($s)]A($n+1)"
4 }
5 let $parms := map {"a": 136, "s": 12}
6 let $interpreters := map {
7   "move#2": function($stack, $n, $s) {
8     $stack=>sys:move(2 * $s * math:sqrt($n))
9   },
10  "shape#1": function($stack, $s) {
11    stack=>sys:circle($s, sys:current($stack)("cix"))
12  }
13 }
```



invisible

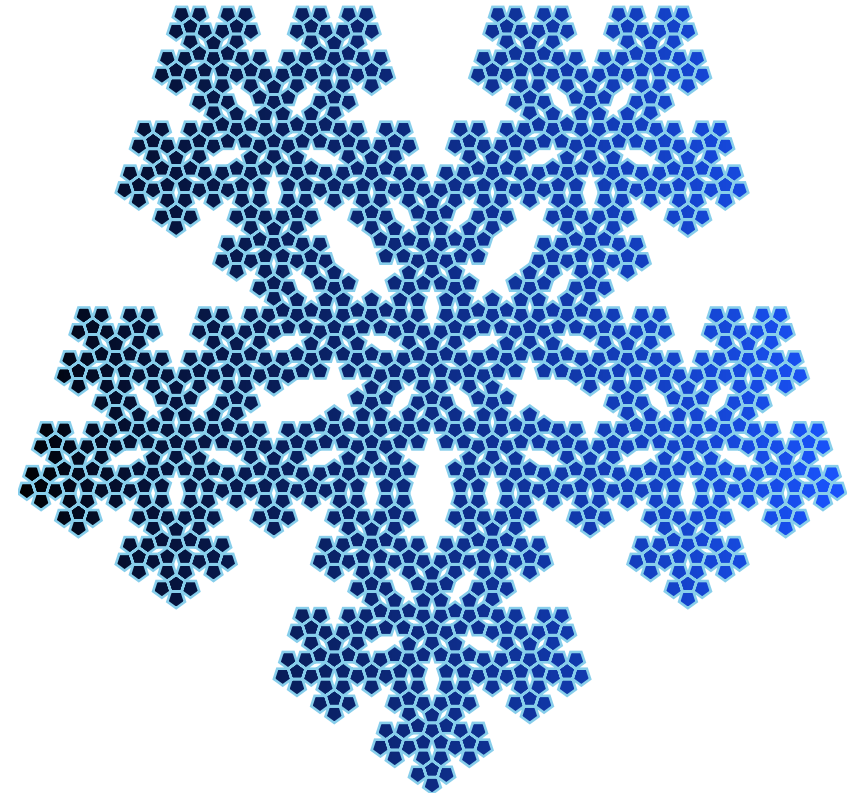


```
angle: 90  
axiom: F--F--F  
F = F+F-F-F+F
```

Parsing output

```
1 root = (path | branch)+ .
2 path = (forward | turn | -extra)+ .
3 branch = -"[" , (path | branch)+ , -"]" .
4 forward = -"F" .
5 turn = direction .
6 @direction = left | right | reverse .
7 left = -"-" , +"left" .
8 right = -"+" , +"right" .
9 reverse = -"|" , +"reverse" .
10 extra = -~["F"; "["; ";" ]"; "+"; "-" ] .
```

CoffeeSacks choose-alternative:
greedy choice (longest leading match)

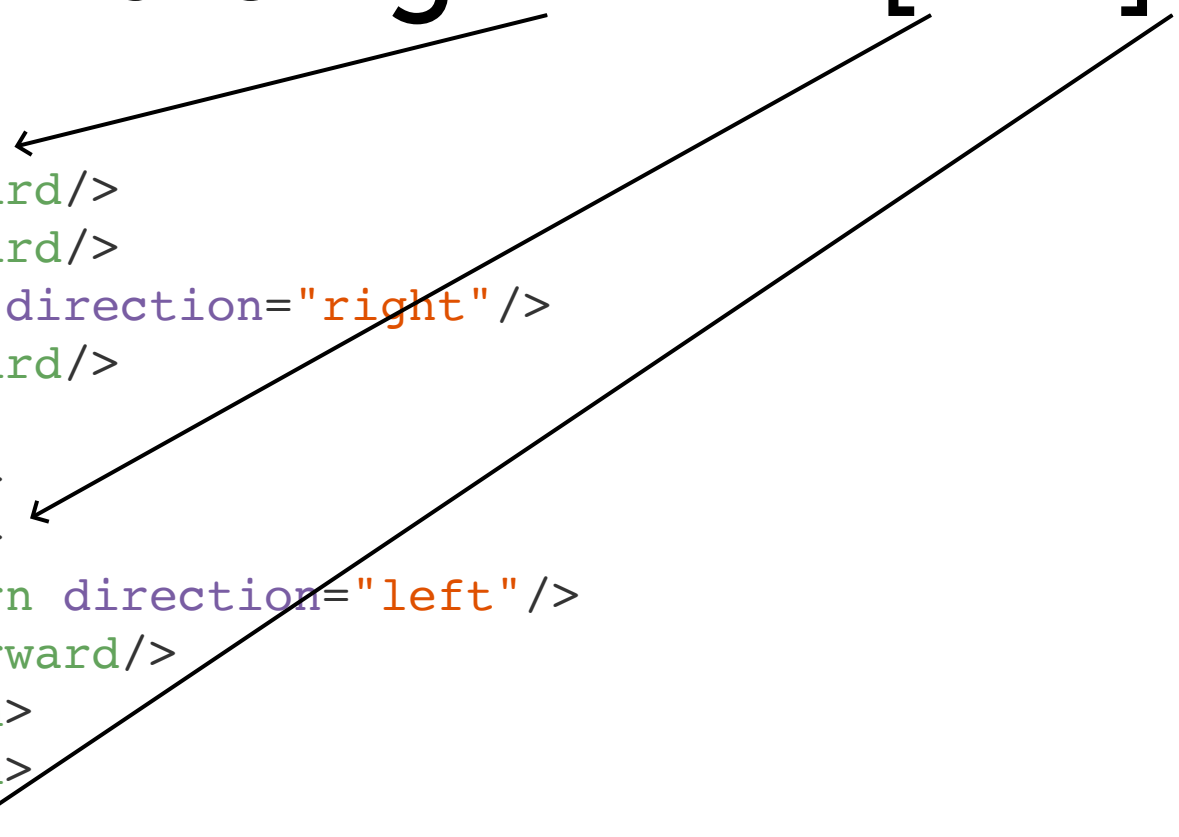


angle: 36
axiom: F++F++F++F++F
F = F++F++F|F-F++F

Handwritten mark

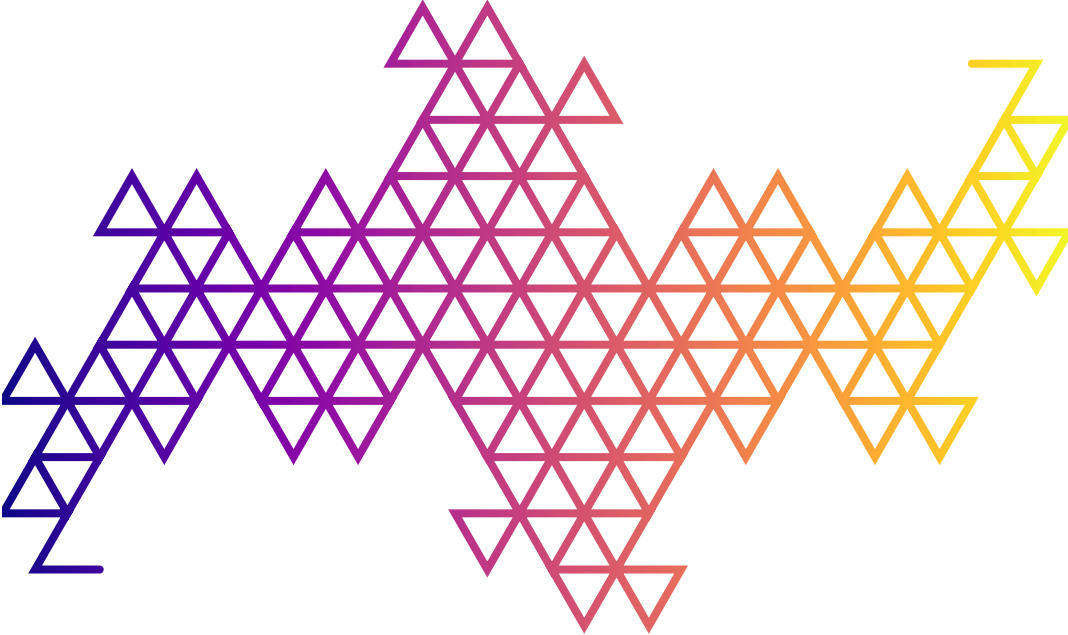
Parsing "FF+XF[X-F]+FF"

```
1 <root>
2   <path>
3     <forward/>
4     <forward/>
5     <turn direction="right" />
6     <forward/>
7   </path>
8   <branch>
9     <path>
10      <turn direction="left" />
11      <forward/>
12    </path>
13  </branch>
14  <path>
15    <turn direction="right" />
16    <forward/>
17    <forward/>
18  </path>
19 </root>
```



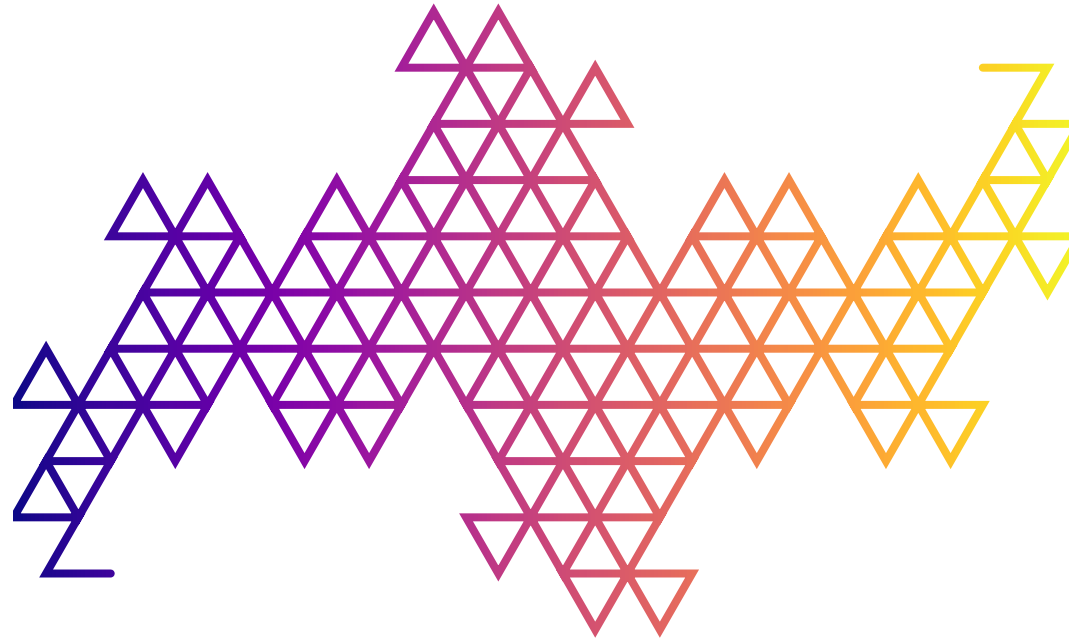
Generation

```
1 { Terdragon curve: axiom: F; F = F+F-F }  
2 { F=forward; -=turn 120 left; +=turn 120 right }  
3  
4 root = (-F | "+" | "-")+ .  
5 F = -"F", +"F+F-F" .
```



Generation

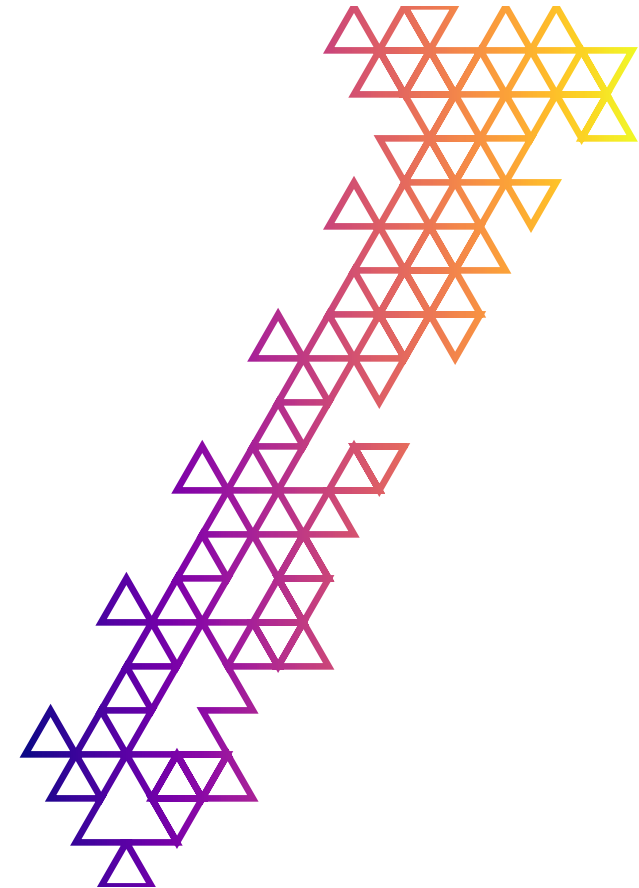
- 1 $F \Rightarrow \langle \text{root} \rangle F+F-F \langle / \text{root} \rangle$
- 2 $F+F-F \Rightarrow \langle \text{root} \rangle F+F-F+F+F-F-F+F-F \langle / \text{root} \rangle$
- 3 $F+F-F+F+F-F-F+F-F$



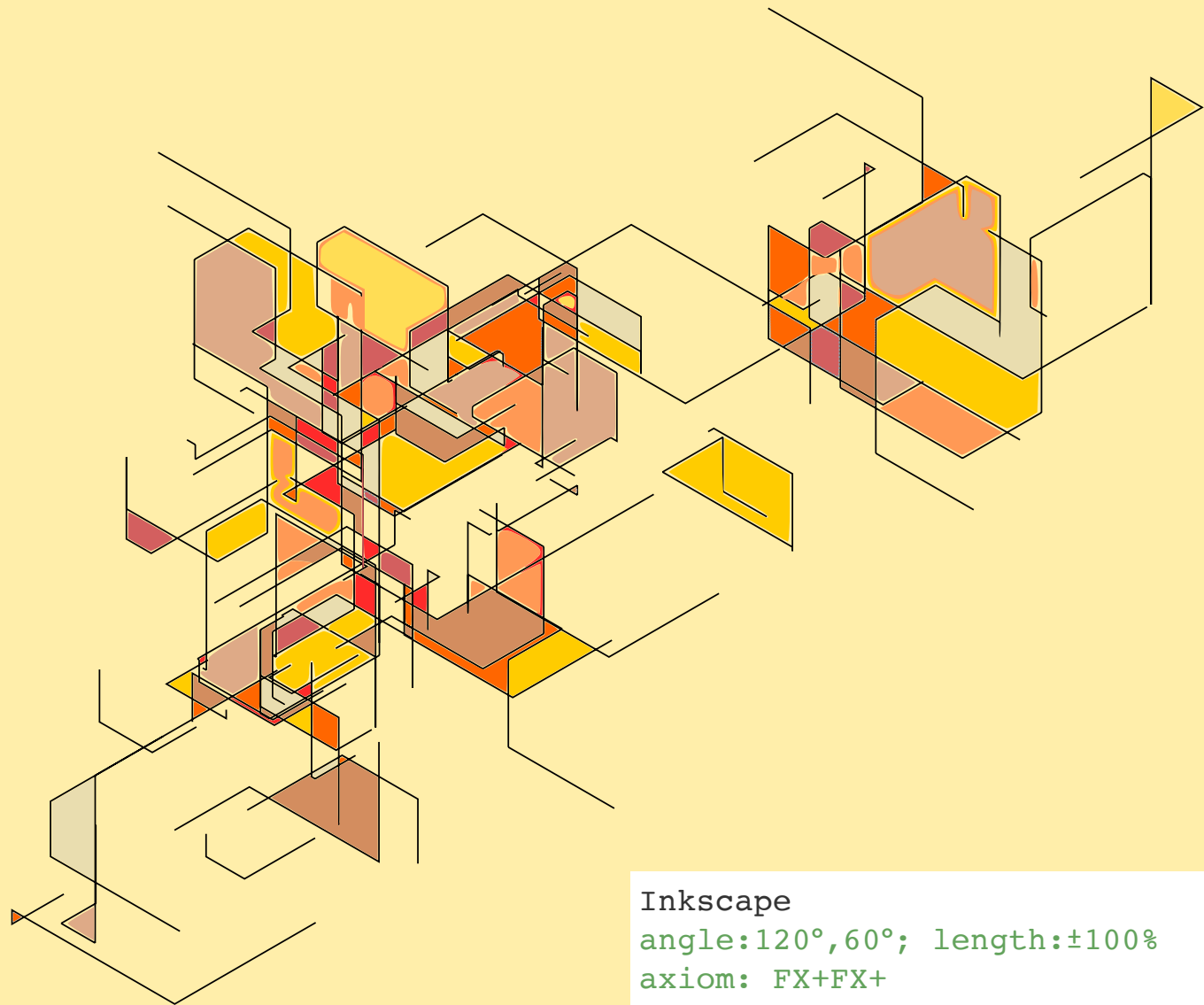
Stochastic Generation

```
1 { Stochastic Terdragon curve: axiom: F; F = F+F-F (80%), F = F-F+F (20%) }
2 { F=forward; -=turn 120 left; +=turn 120 right }
3
4 root = (-F | "+" | "-")+ .
5 F = -"F", +"F+F-F" | -"F", +"F-F+F".
```

Use CoffeeSacks choose-alternative:
weighted random choice



visual

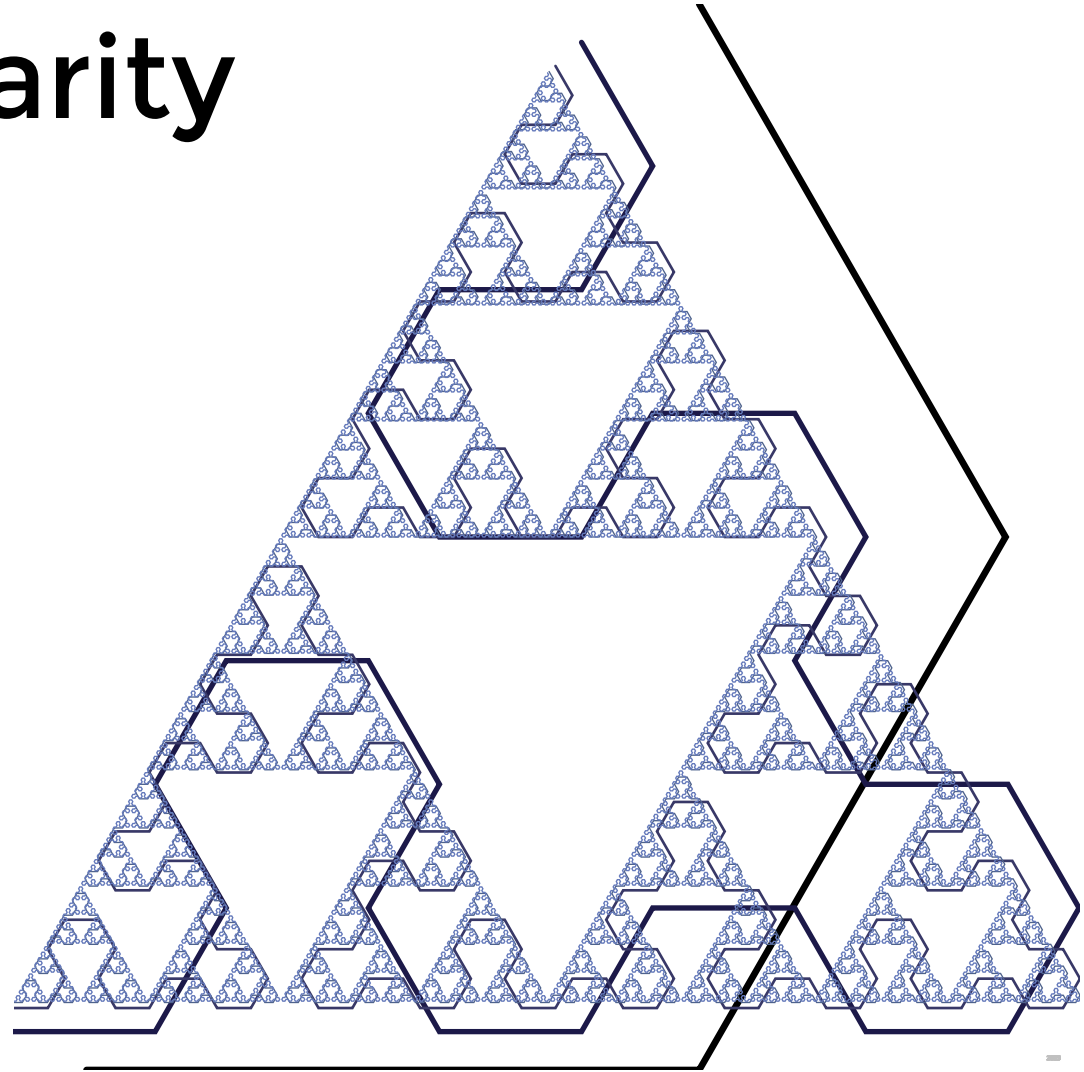


Inkscape
angle:120°,60°; length:±100%
axiom: FX+FX+
X = FX+FY
Y = FX-FY

Recursive self-similarity

Fractal

```
1 let $axiom := "A"  
2 let $rules := map {  
3   "A": "B-A-B",  
4   "B": "A+B+A"  
5 }
```



Recursive self-similarity

Fractal

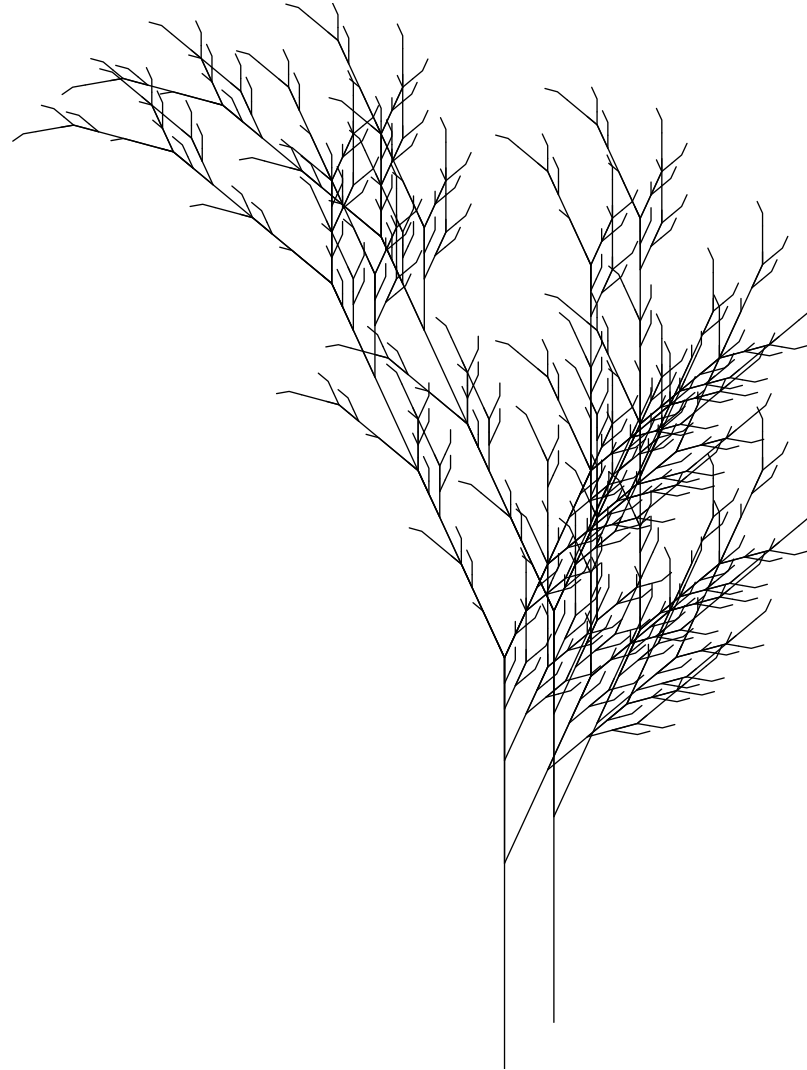
```
1 let $axiom := "A"  
2 let $rules := map {  
3   "A": "B-A-B",  
4   "B": "A+B+A"  
5 }
```



Recursive self-similarity

Organic growth

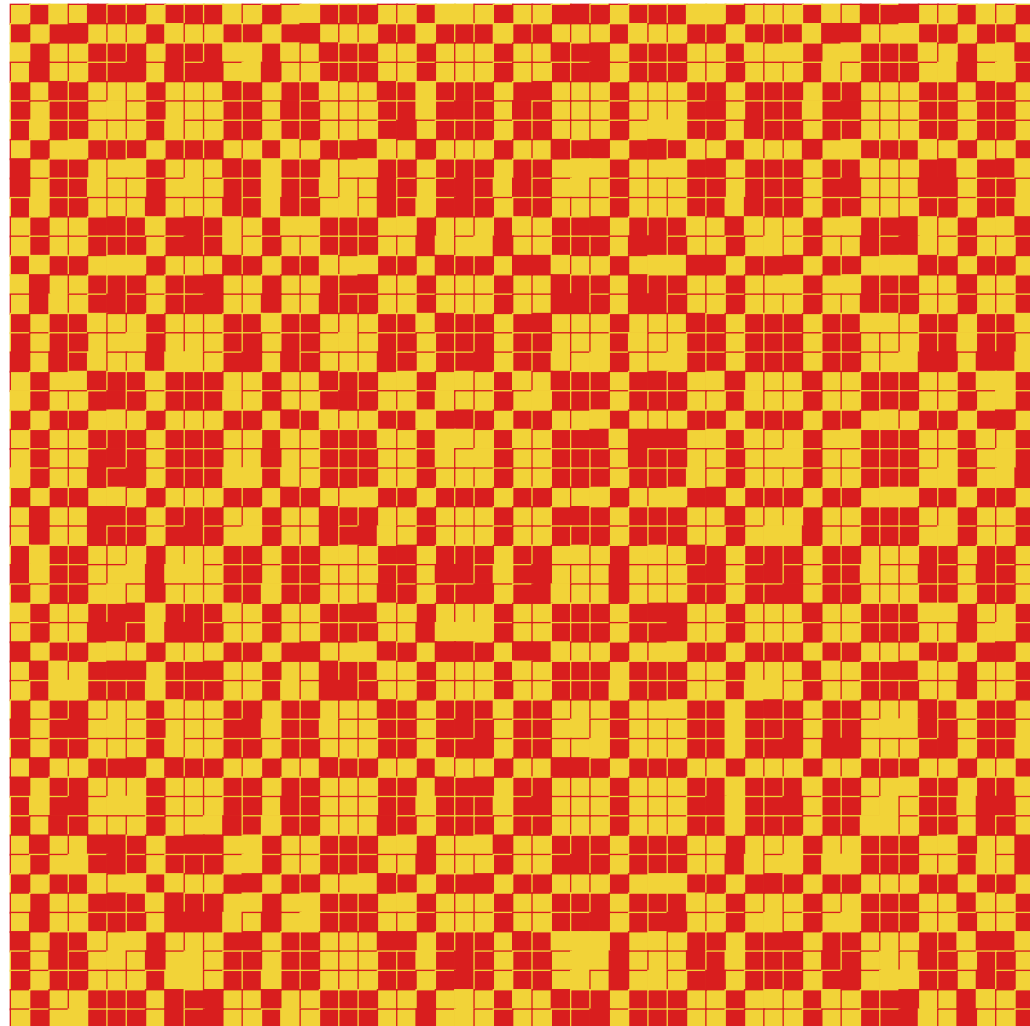
```
1  let $axiom := "X"  
2  let $rules := map {  
3      "X": "F+[ [X]-X ]-F[-FX]+X",  
4      "F": "FF"  
5  }
```



Pattern-generation

Structured, rule-based

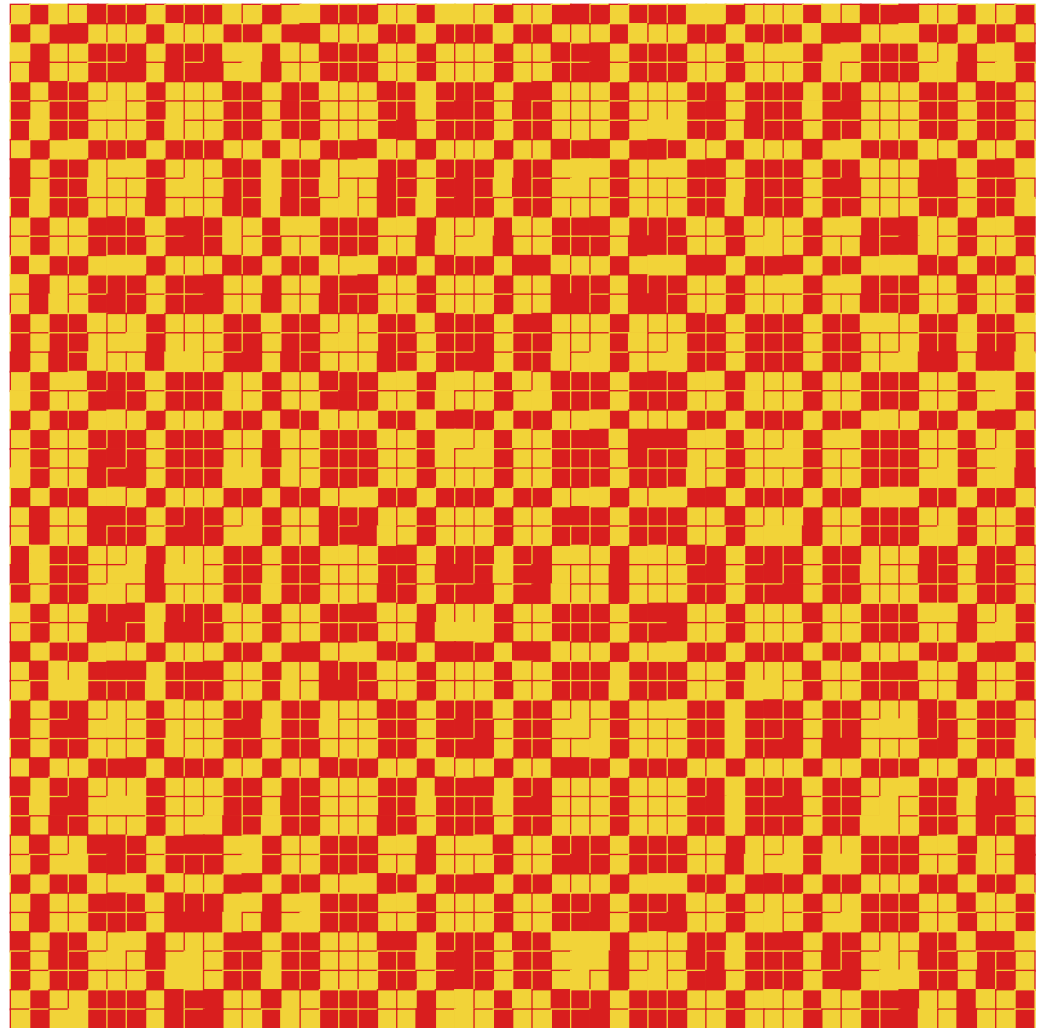
```
let $axiom := this:rbinary(0,31)
let $rules := map {
  "0": "01",
  "1": "10"
}
```



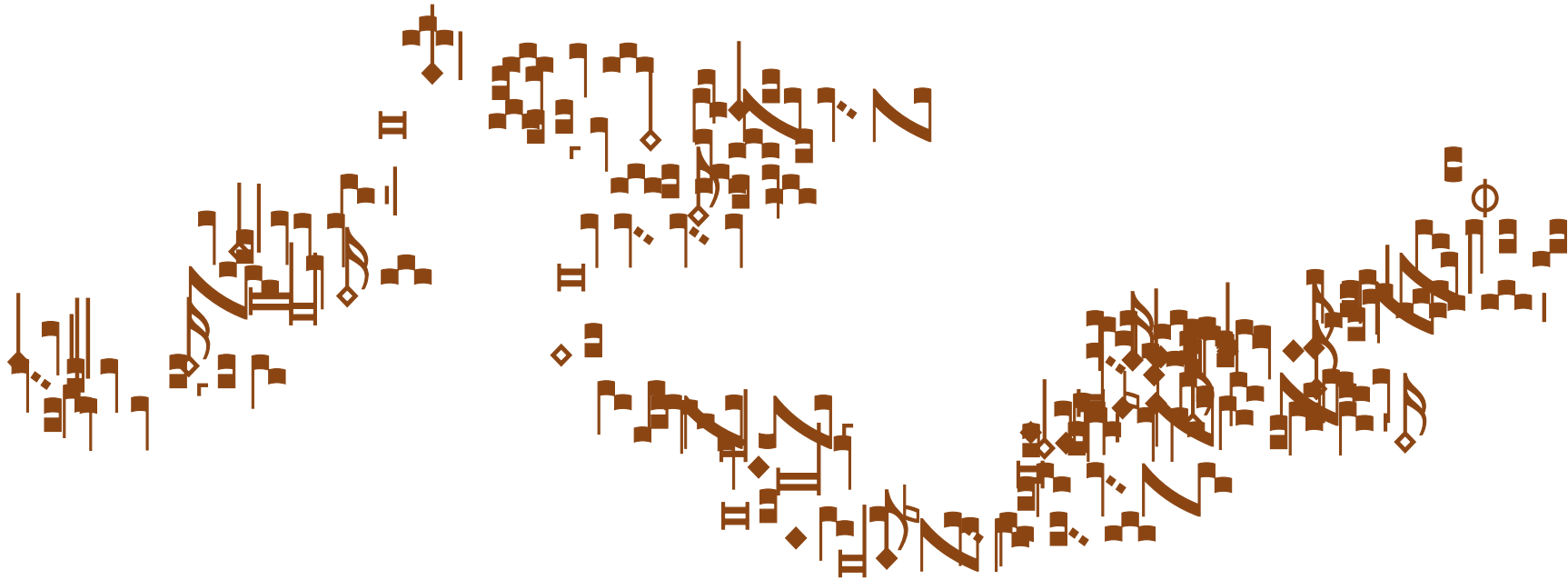
Pattern-generation

Create an integer sequence

```
1 011
2 011010
3 011010010101
4 011010011001011001100110
```



musical



angle: 90

axiom: ?F

F = F+<GF-GF>

linguistic

This idiosyncratic Svengali tacitly despised the contractable safe-deposit box.

axiom: R

R = SAVO (40%)

R = SVBO (30%)

R = SAV (7%)

R = SVB (3%)

R = SAVPO (7%)

R = SVBPO (3%)

R = SAVOCR (7%)

R = SVBOCR (3%)

S = DJN

N = N (95%)

N = NN (5%)

O = DN (50%)

O = DJN (50%)

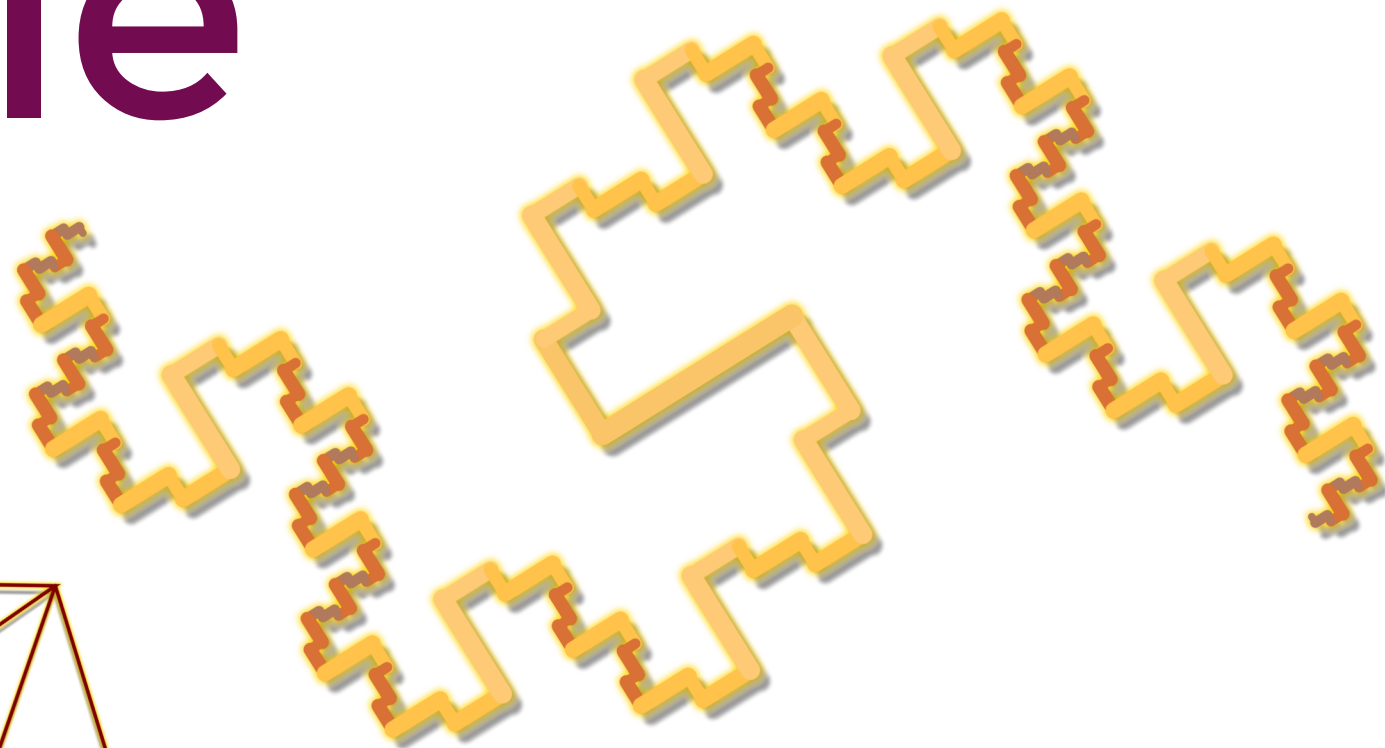
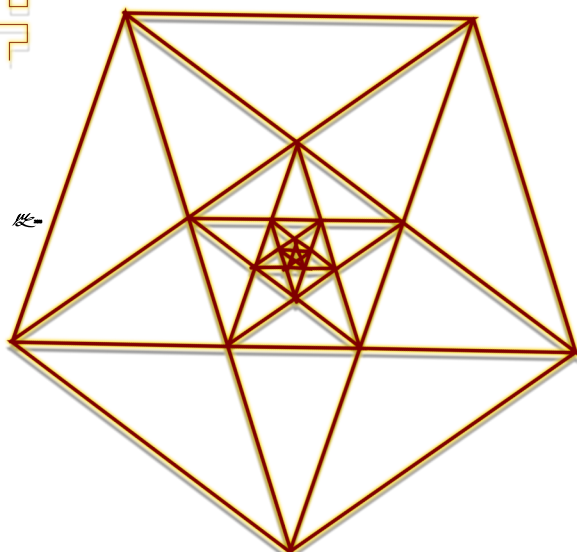
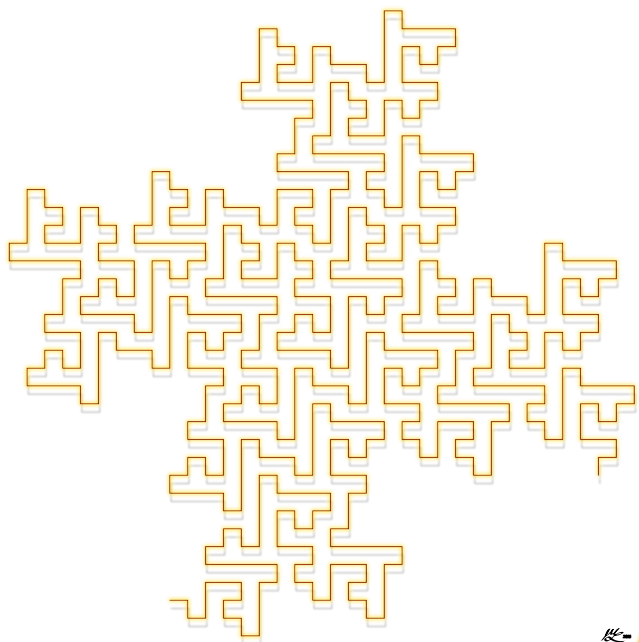
J = J (95%)

J = JJ (5%)

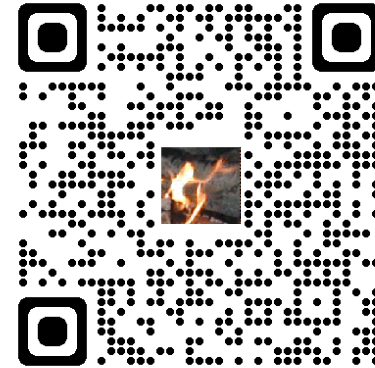
architecture



multiple



Mary's art code: <https://mathling.com/code/>



Juri's library: <https://github.com/line-o/linsy>