

# メディアアート・プログラミング2

東京藝術大学 芸術情報センター開設科目 後期金曜4限 第7週

2023.11.17 松浦知也 ([matsura.tomoya@noc.geidai.ac.jp](mailto:matsura.tomoya@noc.geidai.ac.jp) [teach@matsuuratomoya.com](mailto:teach@matsuuratomoya.com))



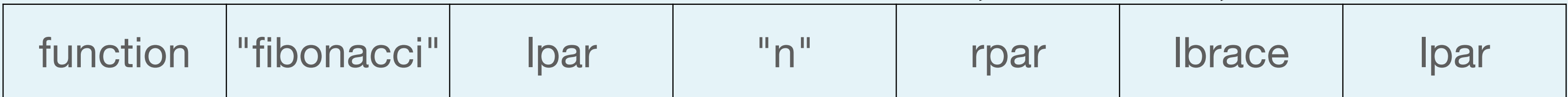
実行可能なテキスト：  
プログラミング言語

# プログラミング言語とは何か

- 見方その1: なんらかのテキストデータをルールに従って別の構造体に変換していくプログラム
  - テキストデータ (バイト列)
  - トークン (単語) 配列
  - 構文木
  - ...
  - マシン語

# プログラミング言語とは何か

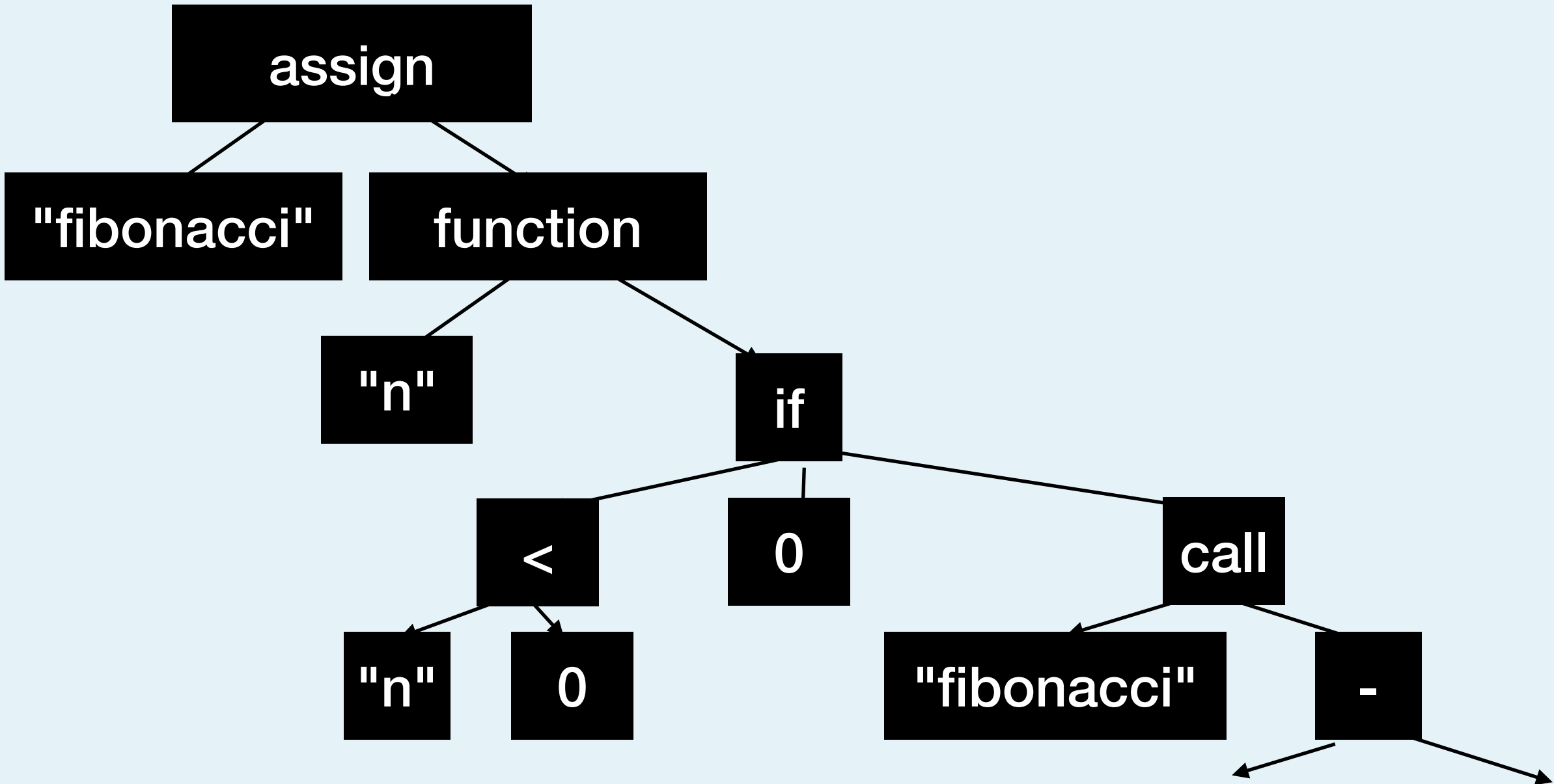
```
function fibonacci(n) {\n (n<0)? 0 : n*fibonacci(n-1) \n}
```

 テキストデータ (バイトの列)

単語 (トークン) の列

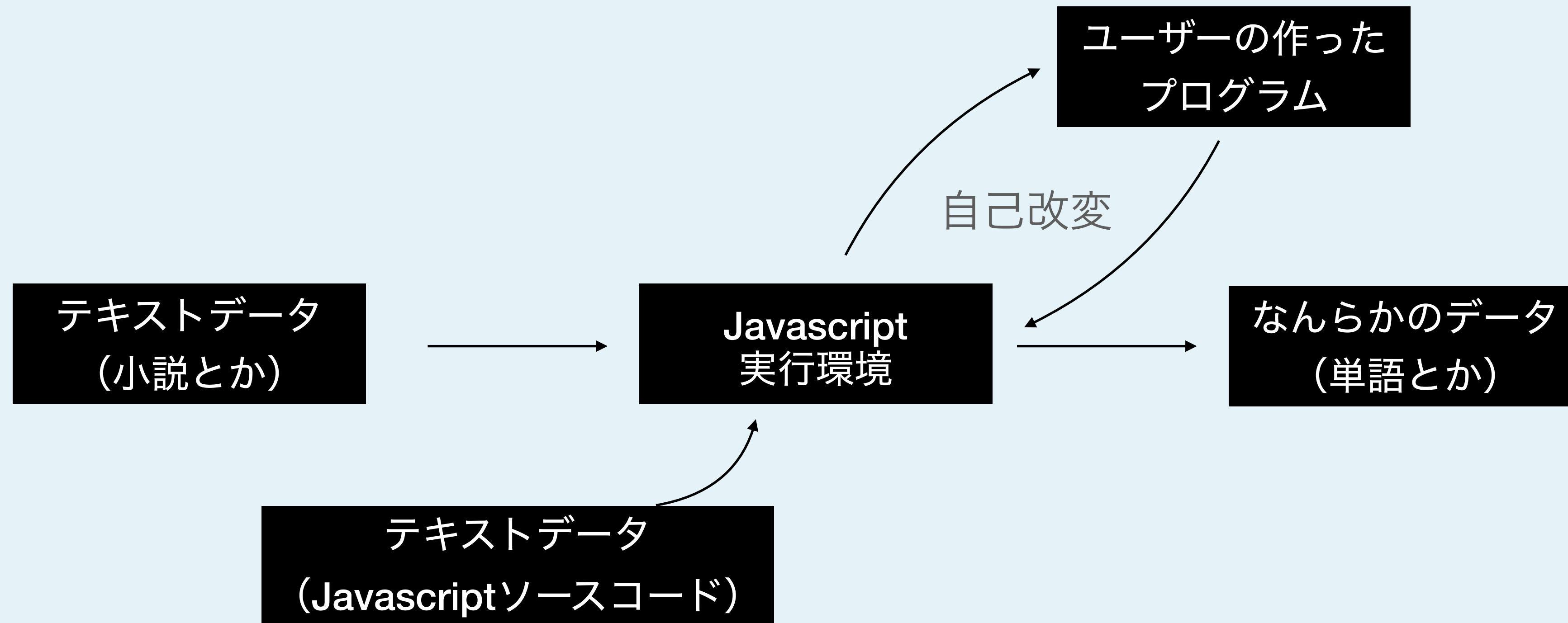


もっと複雑なデータ構造 (木構造とか)






今まではこういう感じで捉えていたが




より正確にはこんな感じ

```
let a = 10  
(a + 2)
```



```
function power(n) {  
    return n*n  
}
```



もう少し抽象的に言うと、プログラミング言語とは"記号の置き換え"を自動化したシステムである

# プログラミング言語作りの一歩目

- プログラミングでテキスト列を操作できるんなら、  
コンピューターを操作する命令を吐き出すこともできるのでは？
- 極論、JavascriptでJavascriptのソースコードを吐き出せたらより複雑なことが  
できるのでは？
- →できます！



# 黒魔術: `eval()`

- 与えられた文字列をJavascriptのソースコードと解釈してその場で実行
  - 変数のスコープとか全部無視するので、まともなプログラミングでは使ってはいけないものとされているが、きちんと使いこなすと強力
- 他にも似たものとして、`process.exec()`というコマンドライン

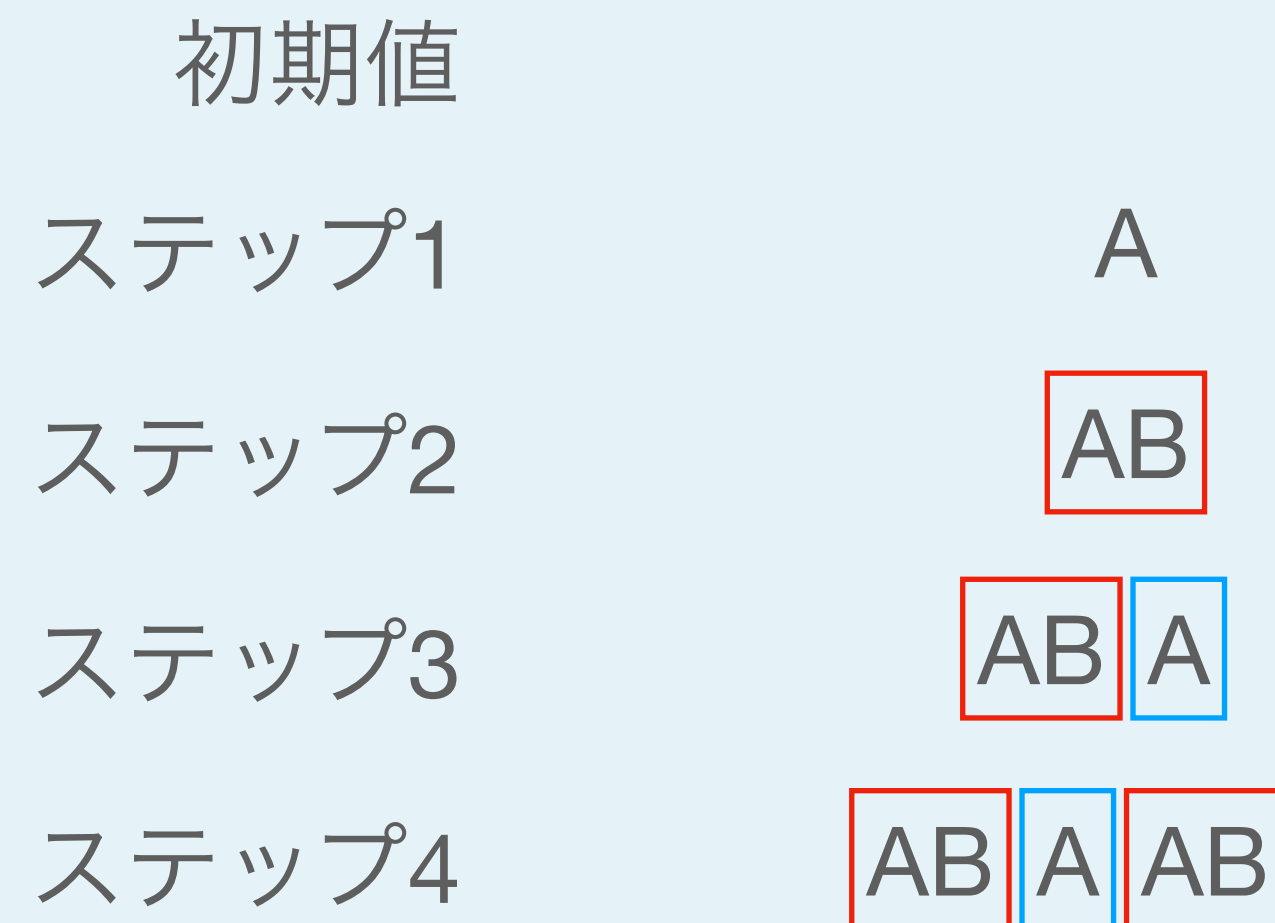
```
> eval("1+2")
3
> let src = "1+2"
undefined
> eval(src)
3
> eval(src+src)
24
> eval(src+" "+src)
6
>
```

例えばここでは、"1+2"という文字列同士の足し算で  
"1+21+2"として結合されてから評価されているので  
24という答えが出てきている

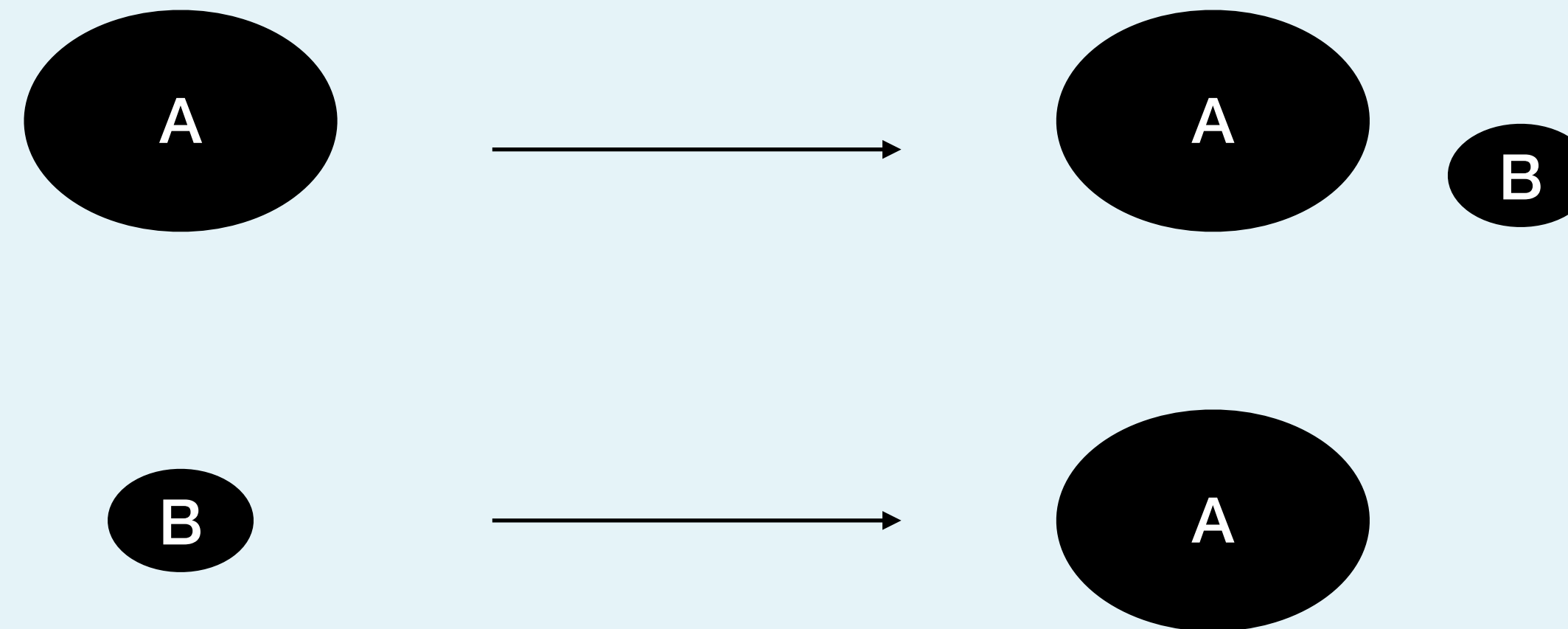
# 項書き換え系 (Term-Rewriting-System)

- ある文字列(ABなど)のそれぞれの文字をルールに従って置き換えて新しい文を作るシステム

ルール：A→AB、B→A



# 項書き換え系 (Term-Rewriting-System)



ちなみに、このルールは

A：大きくなった細胞がAとB：大きな細胞と小さな細胞に分かれ、

Bはやがて成長しAになる

という細胞分裂と増殖の過程をモデル化したもの

起きてる現象を言語  
で記述できるなら、  
それをソースコード  
として実行  
=シミュレーション  
ができる

## 算道

### 論理珠算

#### 論理算盤

#### 整珠

#### 規則

#### 計算

#### 計算例

#### 論理算盤の配布

#### 基本珠

#### 組珠

#### 形

#### 稽古

#### 演算

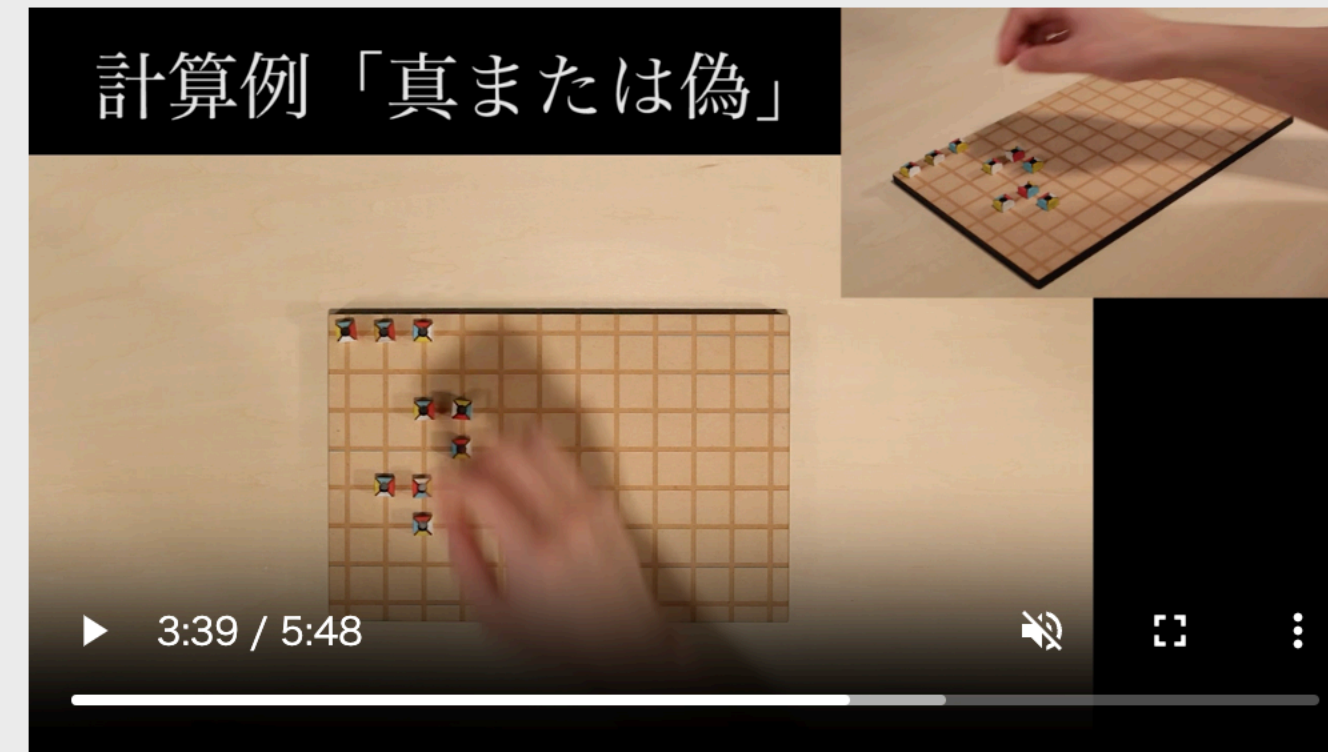
#### 思考

#### 連絡先

## 算道 | Sando

算道とは論理珠算を用いて計算の宇宙を探求する道である。

Sando is the act of investigating the Universe of Calculation using Logical Abacus Calculation.



## 算道の心 | Essence of Sando

記号の発明によって人類は高い精度で情報を伝達出来るようになり、電気技術によってその効力は顕著に向上し、私達の生活を変えた。あらゆる物体は記号に置き換えられ操作され、そして再提示される世の中で、記号はまるで一つの実体のように認識されているだろう。一方で、記号は人間の身体を離れて変換されるようになり、計算そのものは隠蔽されるようになった。そのような時世において情報がどのように操作されるかを知ることは、記号化されていく私達の世界を見直す新たな視点になりうる。

算道の第一義は、論理算盤上の珠という物質性と記号性を兼ね備えた存在により、身体を通じて「計算とは何か」を探求する知的な営みである。しかしながら、現代における算道はただの知的好奇心を超えて、現実を捉えるために必要不可欠な知覚をも与えるだろう。

Through the invention of symbols, humanity became able to relay information with high precision, and through electronic technology, that effect advanced remarkably, changing our lives. In this world where every kind of thing/event has been overwritten by symbols, manipulated, and then re-presented, symbols are likely recognized as one entity. On the other hand, symbols have become such that they are converted separate from the human body, leading to calculation

"算道"(2016) 山本一彰

<https://sando.monophile.net>

# 書いてみよう

```
const src = "A"
let res = src;

const rule_bacteria = (char) => {
  switch (char) {
    case "A": return "AB"
    case "B": return "A"
    default: char
  }
}

const rewrite = (str, rule) => {
  return Array.from(str).map(rule).join("")
}

res = rewrite(res, rule_bacteria);
console.log(res);
res = rewrite(res, rule_bacteria);
console.log(res);
res = rewrite(res, rule_bacteria);
console.log(res);
```

# (補足) Switch-Case文

```
const rule_bacteria = (char) => {  
  if (char=="A"){  
    return "AB"  
  }else if (char == "B"){  
    return "B"  
  }else{  
    char  
  }  
}
```

```
const rule_bacteria = (char) => {  
  switch (char) {  
    case "A": return "AB"  
    case "B": return "A"  
    default: char  
  }  
}
```

ifの条件が複数になる場合はswitch-caseを使うと上手く書けることがある

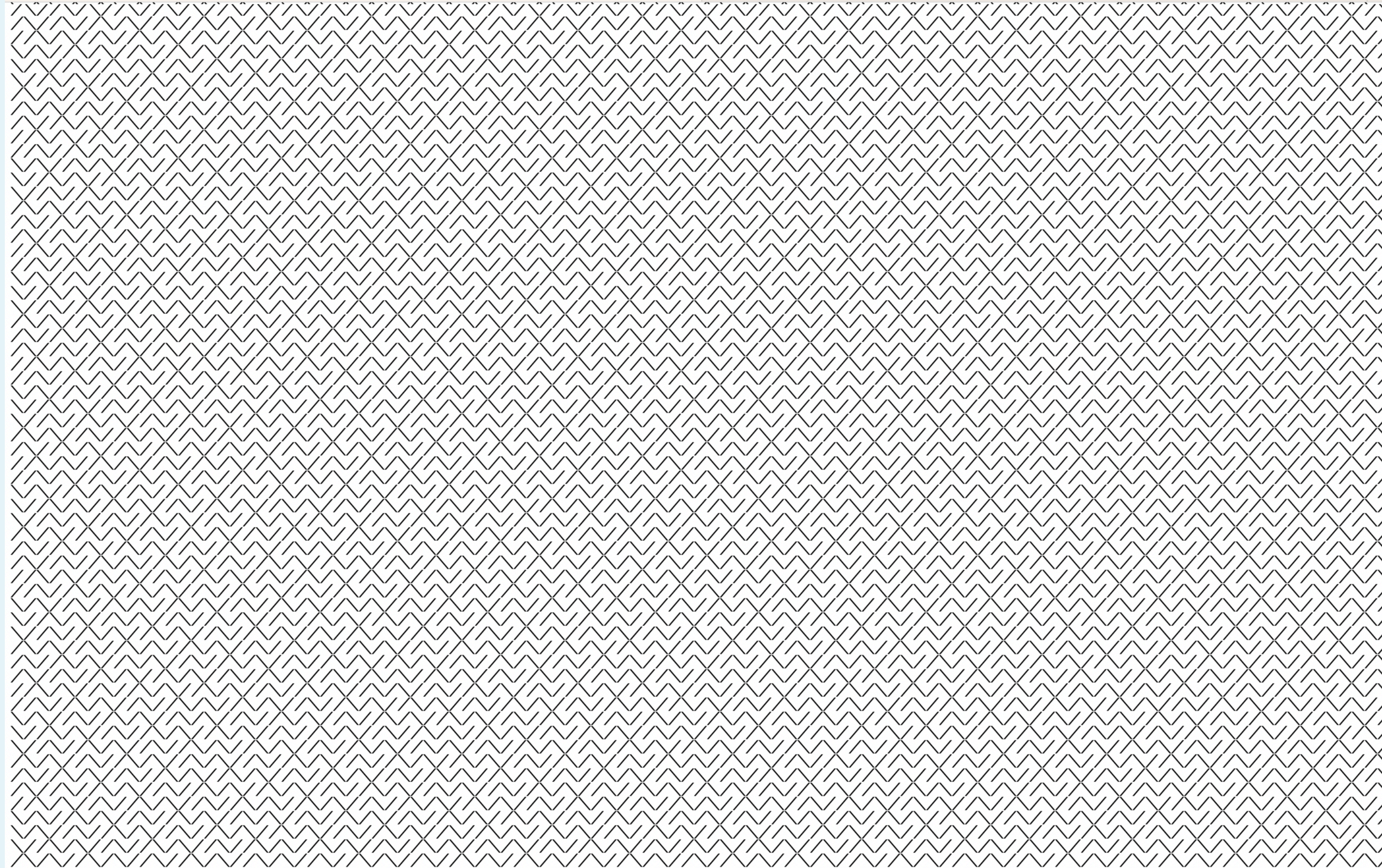
```
tomoya@tomoyanoMacBook-Air 20231117_map2_termrewriting % node rewrite_basic.js
AB
ABA
ABAAB
ABAABABA
ABAABABAABAAB
ABAABABAABAABABAABABA
tomoya@tomoyanoMacBook-Air 20231117_map2_termrewriting %
```

この状態だと、AとBはただの操作した結果でしかない。  
このデータを使って別のことができないか？

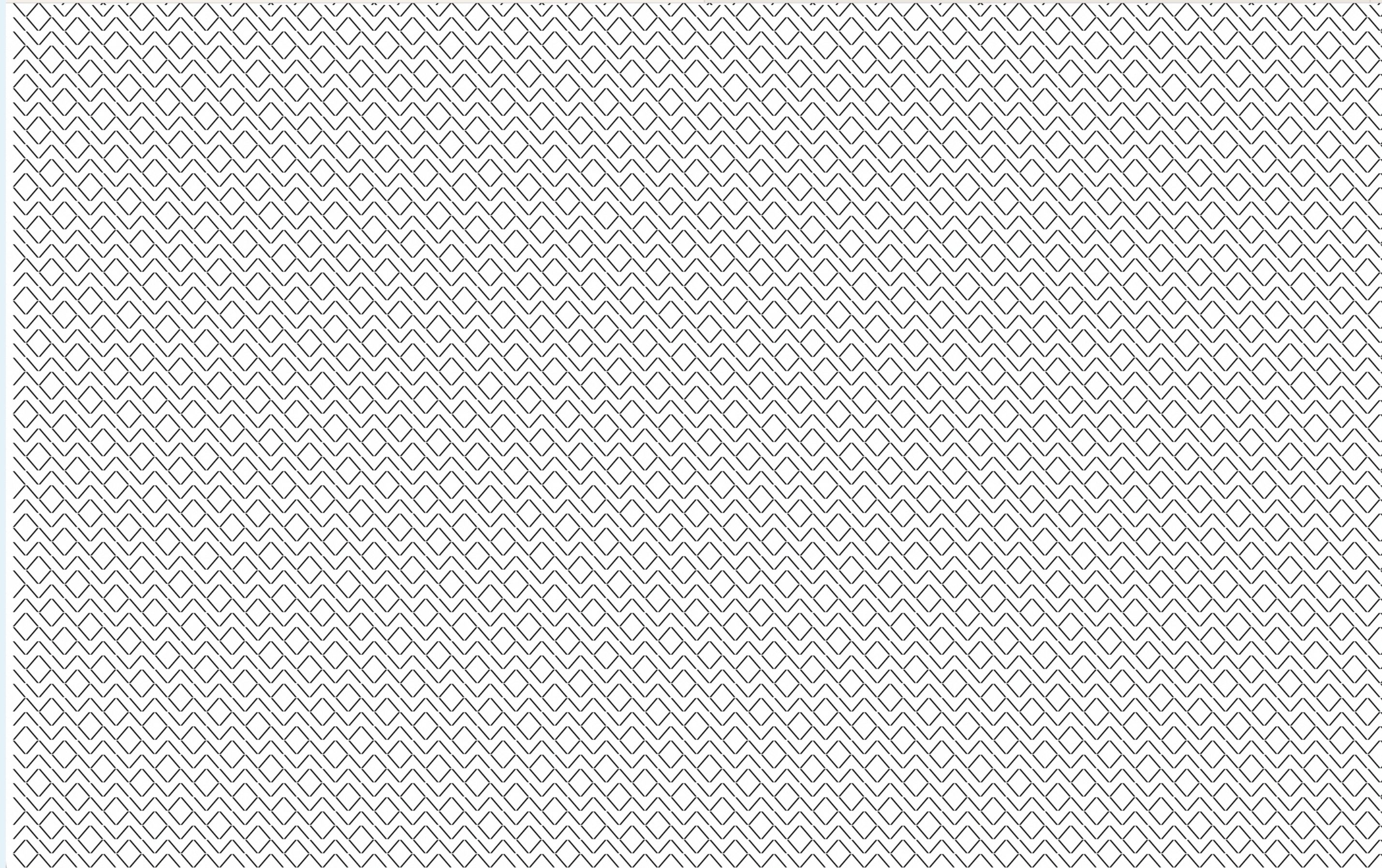


```
const repeat = 25;
const draw10print =(rule)=>{
  let res = src;
  for (let i = 0; i < repeat; i++) {
    res = rewrite(res, rule);
  }
  res = res.replace(/A/g, "/")
  res = res.replace(/B/g, "\")
  console.log(res)
}
draw10print(rule_bacteria)
```

例えば、2回目にやった10 PRINTに対応させてみる

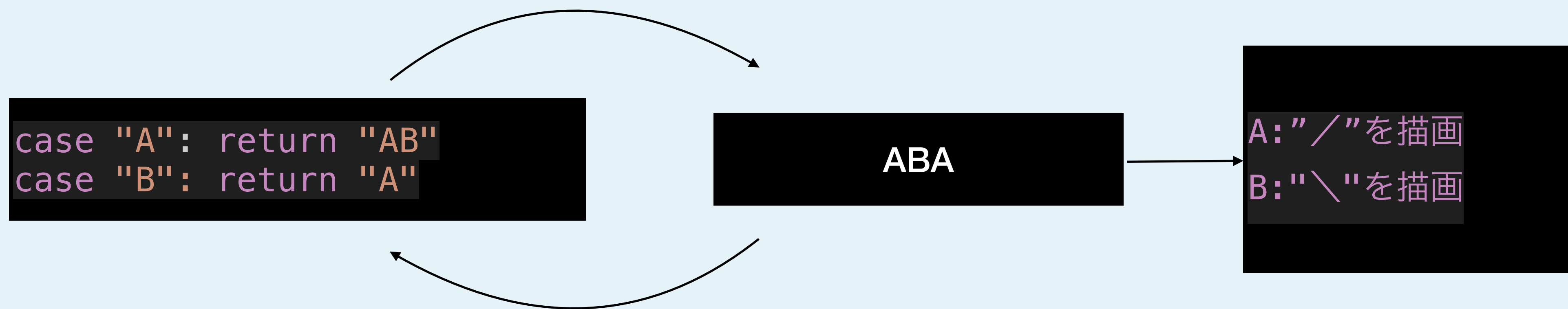


rule\_bacteria( $A \rightarrow AB$ ,  $B \rightarrow A$ , 初期值A)



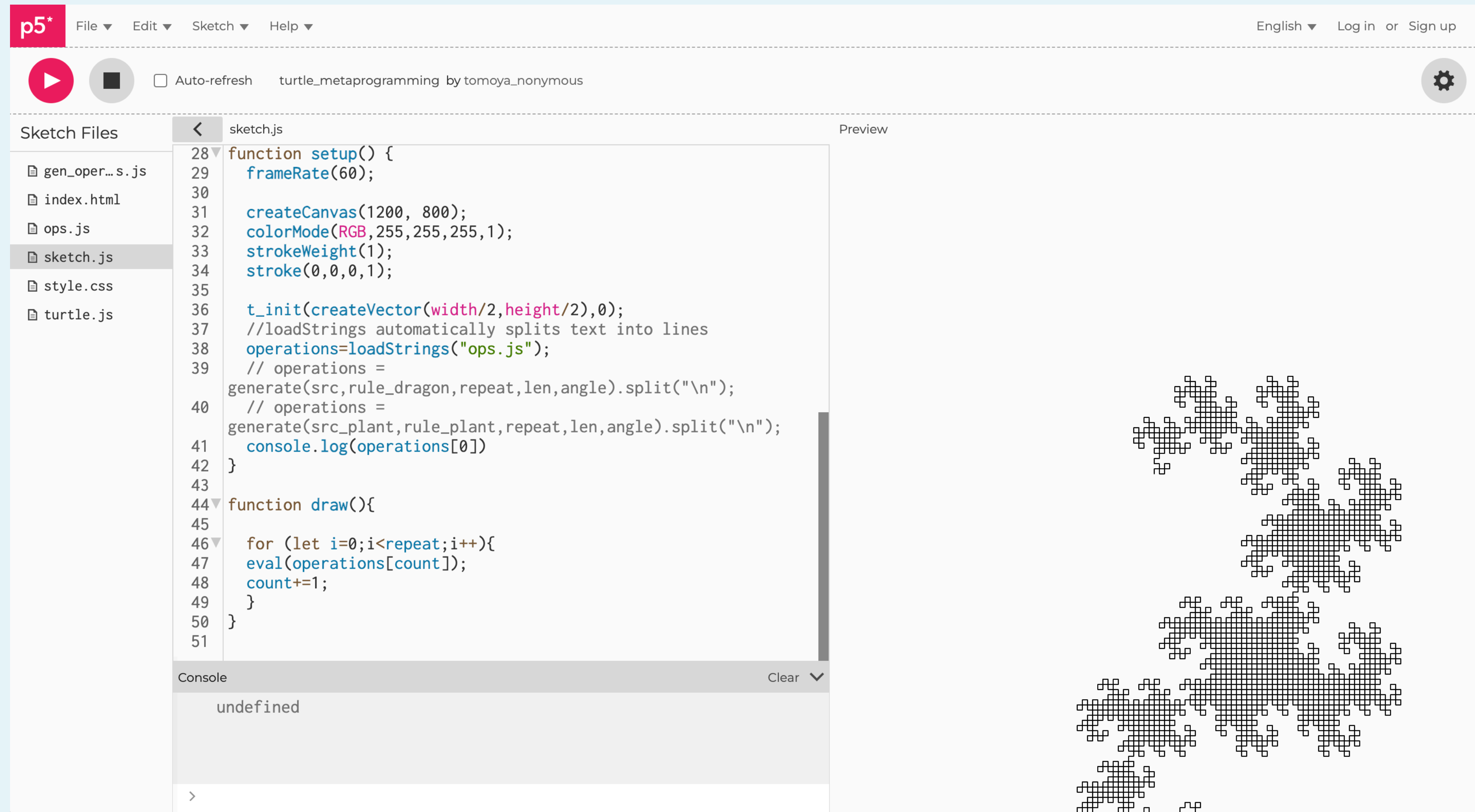
rule\_fibonacci( $A \rightarrow B$ ,  $B \rightarrow AB$ 、初期値AB)

- ここで、AやBというテキストは操作対象でもあり、ソースコードにもなっている



もっと複雑な描画のプログラム方法はあるだろうか

# p5.jsでドラゴンカーブを描いてみよう



The screenshot shows the p5.js editor interface. The top navigation bar includes the p5.js logo, menu items (File, Edit, Sketch, Help), and language options (English, Log in, Sign up). Below the navigation bar, there are controls for auto-refresh and the current sketch name, "turtle\_metaprogramming by tomoya\_nonymous".

The left sidebar shows the "Sketch Files" list, including "gen\_oper...s.js", "index.html", "ops.js", "sketch.js" (selected), "style.css", and "turtle.js".

The main editor area displays the code for "sketch.js":`28 function setup() {  
29 frameRate(60);  
30  
31 createCanvas(1200, 800);  
32 colorMode(RGB, 255, 255, 255, 1);  
33 strokeWeight(1);  
34 stroke(0, 0, 0, 1);  
35  
36 t_init(createVector(width/2, height/2), 0);  
37 //loadStrings automatically splits text into lines  
38 operations=loadStrings("ops.js");  
39 // operations =  
40 generate(src, rule_dragon, repeat, len, angle).split("\n");  
41 // operations =  
42 generate(src_plant, rule_plant, repeat, len, angle).split("\n");  
43 console.log(operations[0])  
44 }  
45  
46 function draw(){  
47 for (let i=0;i<repeat;i++){  
48 eval(operations[count]);  
49 count+=1;  
50 }  
51 }`

The right side of the editor shows a "Preview" window displaying a complex, fractal-like dragon curve drawn with black lines on a white background. The curve is composed of many small, interconnected segments, forming a dense, branching structure.

At the bottom of the editor, the "Console" window shows the output "undefined".

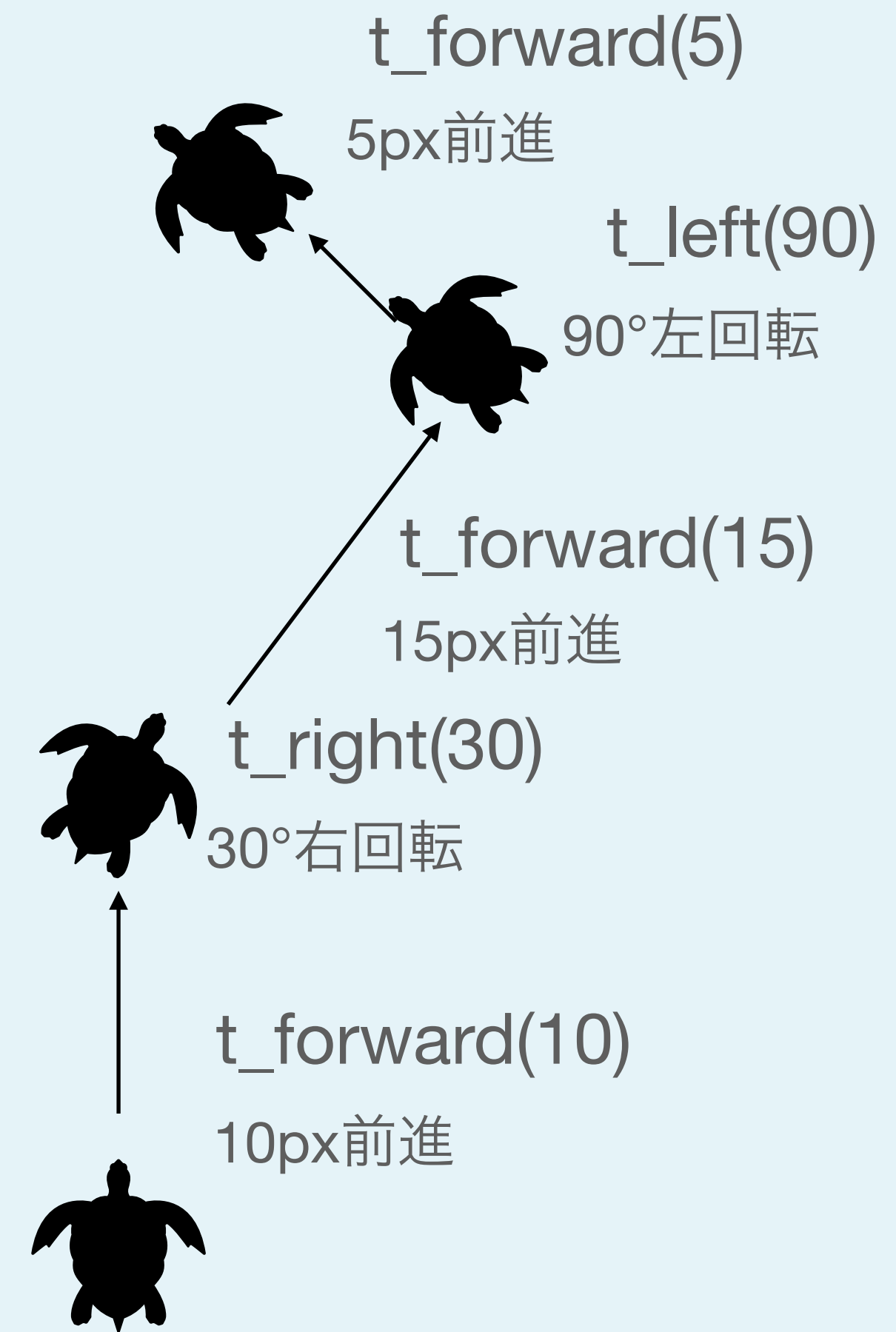
[https://editor.p5js.org/tomoya\\_nonymous/sketches/tF5rNCHJ0](https://editor.p5js.org/tomoya_nonymous/sketches/tF5rNCHJ0)

# タートルグラフィックス

- 前進、右折、左折の命令のみで線を描くプログラム

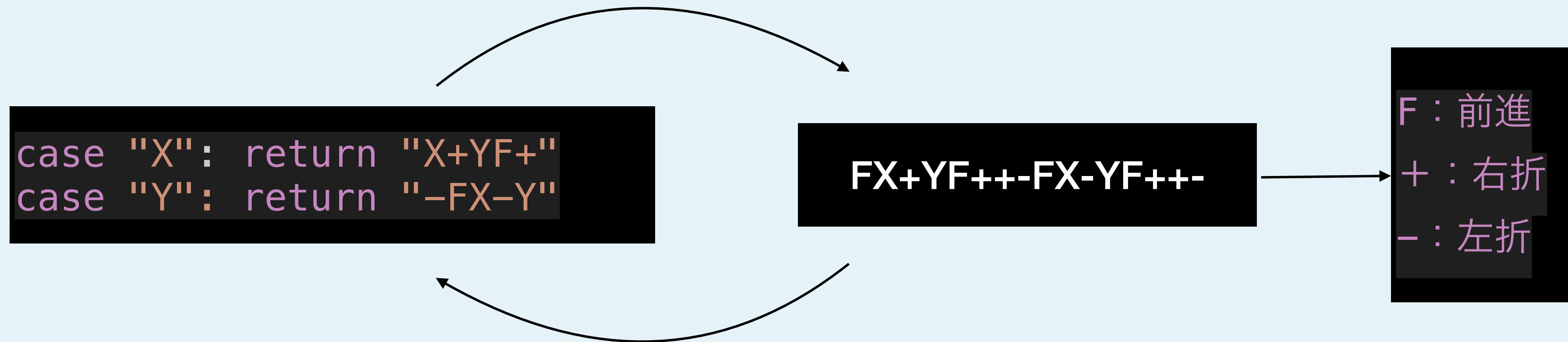
例

```
t_forward(10);  
t_right(30);  
t_forward(15);  
t_left(90);  
t_forward(5);
```



# L-System

- 項書き換え系にタートルグラフィックスの描画命令を加えたもの



置き換えプログラムに  
としてはF,+,-は意味なし

描画プログラムに  
としてはX,Yは意味なし

# L-System

## ドラゴンカーブの置き換えルール

```
const src = "FX";
const rule_dragon = (char)=>{
  switch (char) {
    case "X": return "X+YF+"
    case "Y": return "-FX-Y"
    default: return char
  }
}
```

F、+、-はそのまま何もせず返しているのに注目



# L-System

それぞれの文字をタートルグラフィックス操作関数に置き換える

```
const generate =(src,rule,repetition,len,angle)=>{
  let res = src;
  for (let i = 0; i < repetition; i++) {
    res = rewrite(res, rule);
  }
  res = res.replace(/F/g,`t_forward(${len})\n`)
  res = res.replace(/\+/g,`t_right(${angle})\n`)
  //角度に負の値が入ると-が混ざってエラーになるので注意
  res = res.replace(/\-/g,`t_left(${angle})\n`)
  res = res.replace(/X/g, "")
  res = res.replace(/Y/g, "")
  return res
}
const len = 5;
const angle = 90;
const repeat = 12;
console.log(generate(src,rule_dragon,repeat,len,angle))
```

X,Yは何もせず消しているのに注目

```
t_forward(5)
t_right(90)
t_forward(5)
t_right(90)
t_right(90)
t_left(90)
t_forward(5)
t_left(90)
t_forward(5)
t_right(90)
t_right(90)
t_left(90)
t_forward(5)
t_right(90)
t_forward(5)
t_right(90)
t_left(90)
t_left(90)
t_forward(5)
t_left(90)
t_forward(5)
t_right(90)
t_right(90)
t_left(90)
t_forward(5)
t_right(90)
t_forward(5)
t_right(90)
t_right(90)
t_left(90)
t_forward(5)
```

repeat5回で1万行ぐらいのjsコードが出てくる

The screenshot shows the p5.js editor interface. At the top, there is a menu bar with 'File', 'Edit', 'Sketch', and 'Help'. Below the menu bar, there are icons for play, stop, and a checked 'Auto-refresh' checkbox. The sketch name is 'turtle\_metaprogramming' by 'tomoya\_nonymous'. The 'Sketch Files' panel on the left lists several files: 'gen\_oper...s.js', 'index.html', 'ops.js' (selected), 'sketch.js', 'style.css', and 'turtle.js'. The main editor area shows the code for 'ops.js' with line numbers 1 through 25. The code is as follows:

```
1 t_forward(5)
2 t_right(90)
3 t_forward(5)
4 t_right(90)
5 t_right(90)
6 t_left(90)
7 t_forward(5)
8 t_left(90)
9 t_forward(5)
10 t_right(90)
11 t_right(90)
12 t_left(90)
13 t_forward(5)
14 t_right(90)
15 t_forward(5)
16 t_right(90)
17 t_left(90)
18 t_left(90)
19 t_forward(5)
20 t_left(90)
21 t_forward(5)
22 t_right(90)
23 t_right(90)
24 t_left(90)
25 t_forward(5)
```

At the bottom, the console shows 'undefined'.

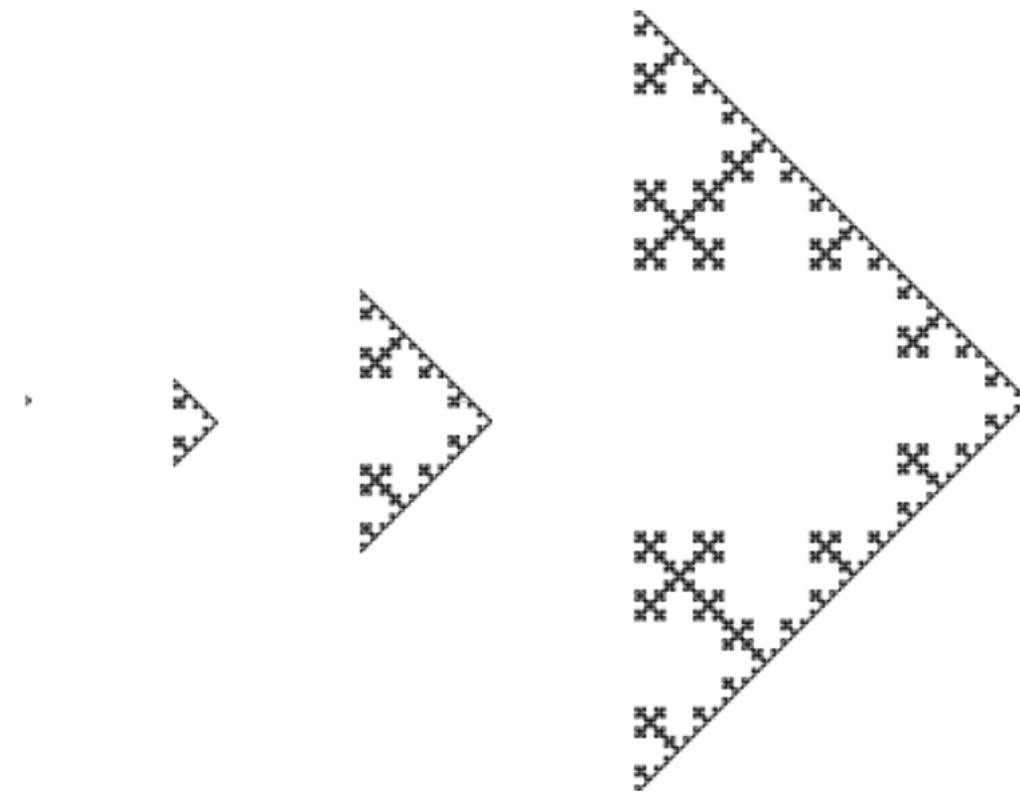
[https://editor.p5js.org/tomoya\\_nonymous/sketches/tF5rNCHJ0](https://editor.p5js.org/tomoya_nonymous/sketches/tF5rNCHJ0)

でops.jsというファイルをペーストして置き換える

- 「何かを操作するテキスト」を機械的に生成するプログラムを作った
- ある意味、詩とは操作対象が人間の情動なプログラムかもしれない
- 楽譜とは操作対象が演奏者なプログラムかもしれない
- 戯曲とは操作対象が演者なプログラムかもしれない

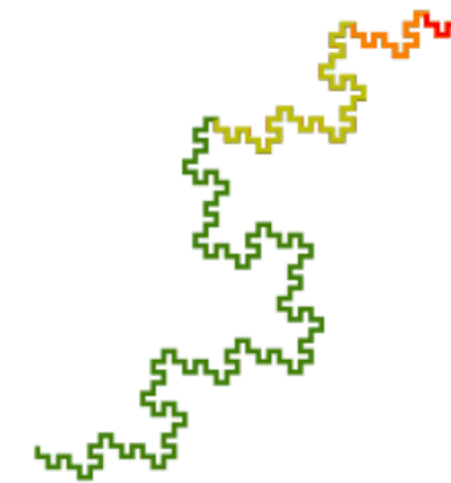
# L-Systemのさまざまなルール

No.01 コッホ曲線 Koch Curve



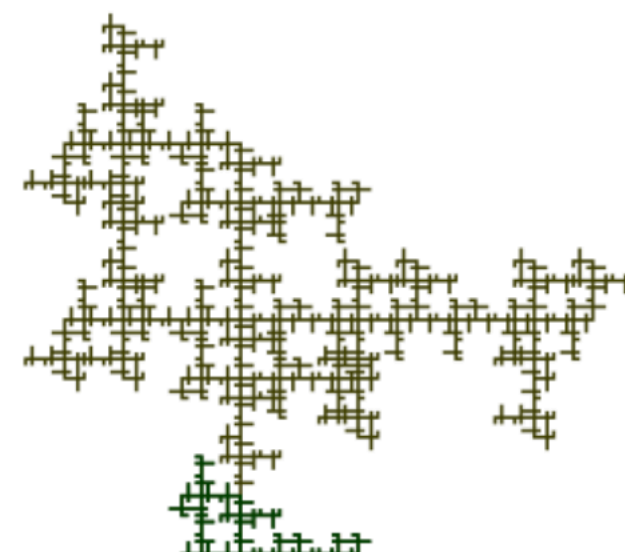
SVG File	01.svg
公理(初期)Axiom	F
ルールRule	F=F+F-F+F
角度(左,右)Angle(L,R)	90,90
ランダム化Randomize	0
標準級数Order	---

No.02 蛇足 Meandering Snake

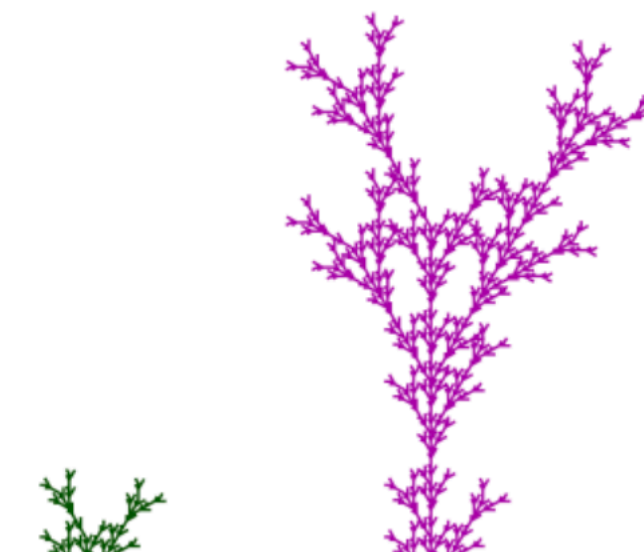


SVG File	02.svg
公理(初期)Axiom	F
ルールRule	F=F++F--F+F
角度(左,右)Angle(L,R)	90,90
ランダム化Randomize	0
標準級数Order	---

No.03 都市探査 Urban Charting



No.04 水草A



# Special Thanks

- 今日の授業はSchool for Poetic Computation Summer in Yamaguchi 2019のRobby Kraftによる授業"Garden Mathematics"を大いに参考にしています！
- Today's class is heavily based on the class "Garden Mathematics" taught by Robby Kraft in School for Poetic Computation Summer in Yamaguchi 2019.
  - <https://www.ycam.jp/events/2019/sfpc/>
  - <https://robykraft.com/cv.html>