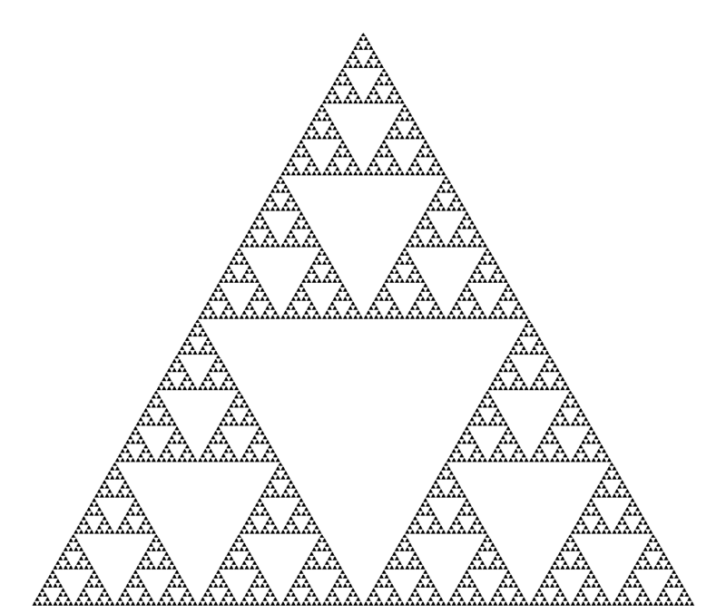
知能情報システム論

第2回　フラクタルとLシステム

[復習]フラクタルとは

フラクタル幾何学（1975年）
→「自己相似性」をもつ図形
一部を拡大しても全体と変わらない

動機：
複雑な形や構造がもつ秩序や規則を調べたい
（海岸線、樹木、山、雲、株価）

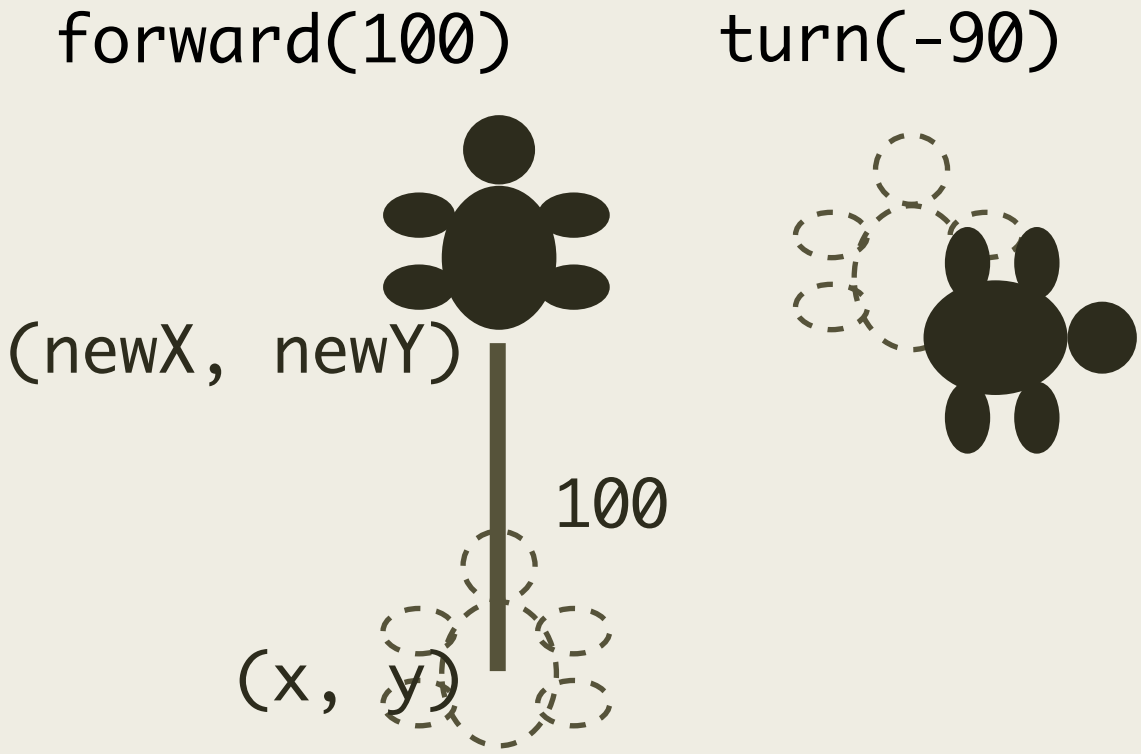


マンデルブロ(1924-2010)

[復習]フラクタル図形をかく

コッホ曲線をプログラムでかく

曲線の各点を計算して線を結ぶのは大変
→タートルグラフィックス(Turtle Graphics)を用いる

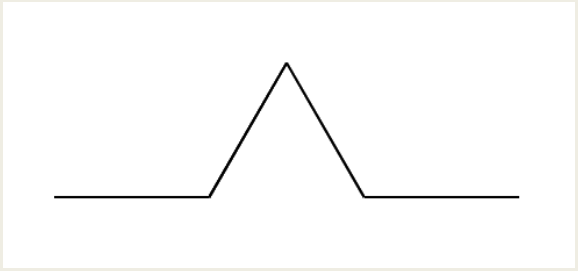


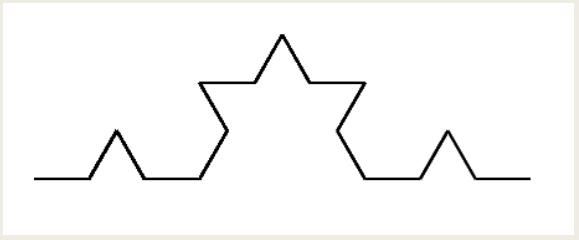

```
forward(100);
turn(60);
forward(100);
turn(-120);
forward(100);
turn(60);
forward(100);
```

[復習]数学的なフラクタル図形をかく

- 再帰的な関数を使うとコッホ曲線がかける。
- 「chinou02.html」を使う。
- 「ベース」と「モチーフ」は前回解説済み。
- 32行目。ボタンを押すとstart 関数が発動。レベル数を取得して関数 go を呼び出す。
- 38行目。関数 go でフラクタルを書いている。

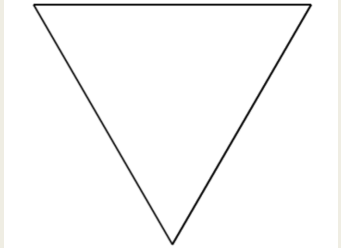
```
31      //ベース
32      function start(e){
33          let level = Number(e.id[0])
34          go(size, level);
35      }
36
37      //モチーフ
38      function go(size, level){
39          if(level == 0){
40              turtle.forward(size);
41          } else {
42              go(size/3, level - 1)
43              turtle.turn(60)
44              go(size/3, level - 1)
45              turtle.turn(-120);
46              go(size/3, level - 1)
47              turtle.turn(60);
48              go(size/3, level - 1)
49          }
50      }
```

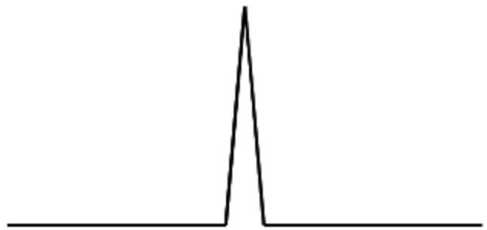


[復習]ベースとモチーフ

例題.
「chinou02.html」内のプログラムのベース部とモチーフ部をそれぞれ以下のように入力し、コッホ雪片を書いてみよう。次に、あなたが考えたモチーフでフラクタル図形を作成してみよう。



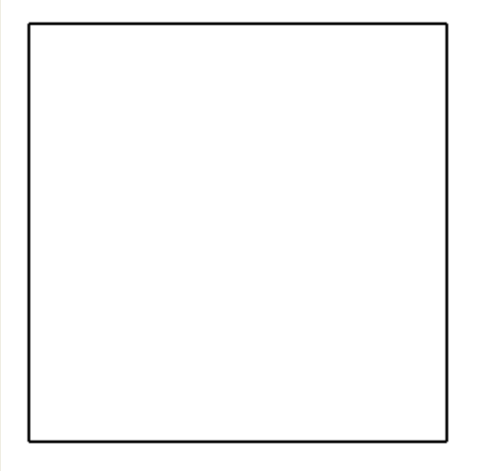
```
go(size, level);
turtle.turn(-120);
go(size, level);
turtle.turn(-120);
go(size, level);
```



モチーフのアイディア

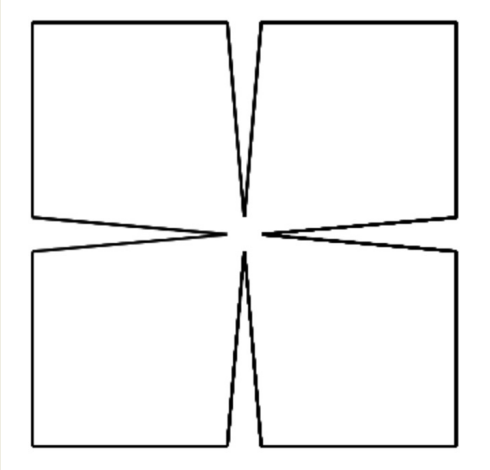
```
function go(size, level){
    if (level == 0){
        turtle.forward(size);
        return;
    } else {
        go(size/3, lebel - 1);
        turtle.turn(60);
        go(size/3, lebel - 1);
        turtle.turn(-120);
        go(size/3, lebel - 1);
        turtle.turn(60);
        go(size/3, lebel - 1);
    }
}
```

[復習]ベースも変える

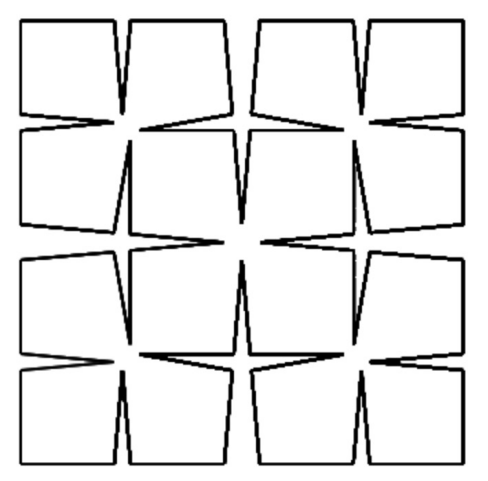


ベース
（Level = 0）

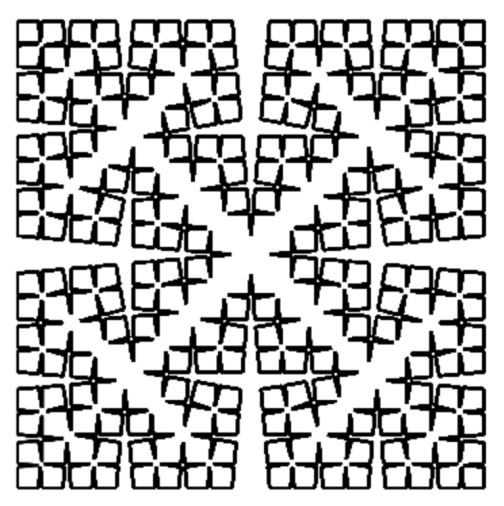
モチーフ



Level = 1

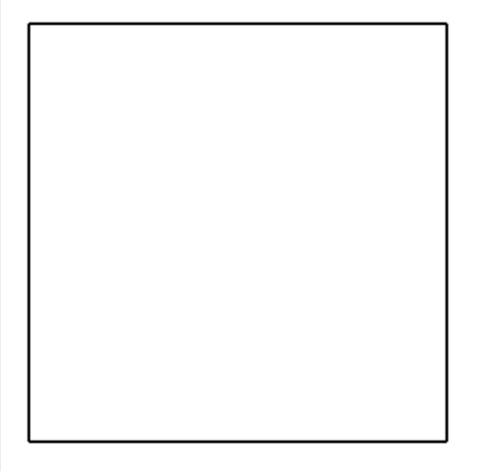


Level = 2



Level = 4

[復習]ベースとモチーフを両方考えよう



```
for(let i=0; i<4; i++){
  go(size, level);
  turn(90);
}
```

繰り返し回数と角度を調整すれば好きな正多角形をベースにできる

```
function go(size, level){
    if (level == 0){
        turtle.forward(size);
        return;
    } else {
        go(size/2.1, lebel - 1);
        turtle.turn(85);
        go(size/2.1, lebel - 1);
        turtle.turn(-170);
        go(size/2.1, lebel - 1);
        turtle.turn(85);
        go(size/2.1, lebel - 1);
    }
}
```

```
function go(size, level){
    if (level == 0){
        turtle.forward(size);
        return;
    } else {
        go(size/2.1, lebel - 1);
        turtle.turn(85);
        go(size/2.1, lebel - 1);
        turtle.turn(-170);
        go(size/2.1, lebel - 1);
        turtle.turn(85);
        go(size/2.1, lebel - 1);
    }
}
```

←モチーフの入力
すらめんどくさい

フラクタルの他の作り方は？

↓

L-system を使った方法
(形式文法を使った方法)

形式文法

■ 「オートマトンと形式言語」で形式文法について習った（履修してない人はここで覚える）

文法 G = <N, Σ, P, S> とは次の4要素からなる

N: 非終端記号の有限集合
Σ: 終端記号の有限集合
P: 書き換え規則の有限集合
S: 開始記号

例1. G_1 = <N, Σ, P, S> を次で定める。
N = {<文>, <主語>, <動詞>}, Σ = {cats, sleep, eat},
P = { <文> → <主語><動詞>, <主語> → cats,
<動詞> → sleep, <動詞> → eat},
S = <文>

例1. G_1 = <N, Σ, P, S> を次で定める。

N = {<文>, <主語>, <動詞>}, Σ = {cats, sleep, eat},

P = { <文> → <主語><動詞>, <主語> → cats,

<動詞> → sleep, <動詞> → eat},

S = <文>

このとき、 cats sleep という記号列は

<文> ⇒ <主語><動詞> ⇒ cats sleep

という書き換えによって導出できる。

例2. G_2 = <N, Σ, P, S> を次で定める。

N = {S, T}, Σ = {a, b},

P = { S → aTb, T → aTb, T → ab}

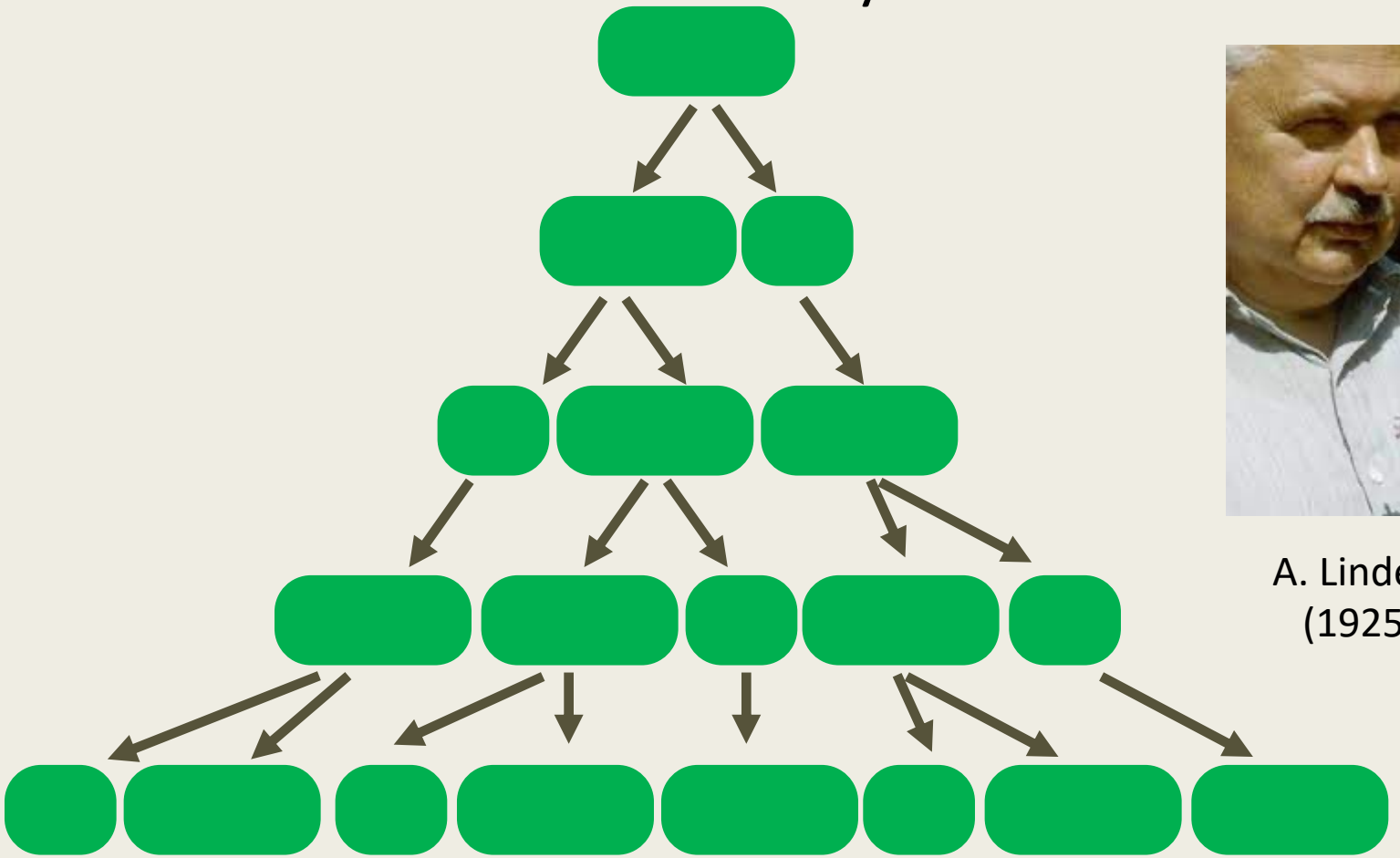
この文法が生成する言語 L は

$L = \{a^n b^n \mid n \geqq 2\}$

である。

L-systems

- L-system は形式文法の一種。
- 生物の発生（細胞分裂）の過程などが記述・表現できるアルゴリズム。
- 「L」は 提唱者の Lindenmayer にちなむ。



A. Lindenmayer
(1925-1989)

L-systems

- Lシステムとは、大雑把に言えば 非終端記号 と 終端記号 を区別しない形式文法（文献にも依るので注意）。

定義.
Lシステム G = <N, P, S> は次の3要素からなる：
N: 記号の有限集合
P: 書き換え規則の有限集合
S: 開始記号

- すべての記号 $x \in N$ がただひとつの書き換え規則をもつとき Lシステム は D0L-システム (deterministic context-free L-system) と呼ばれる。

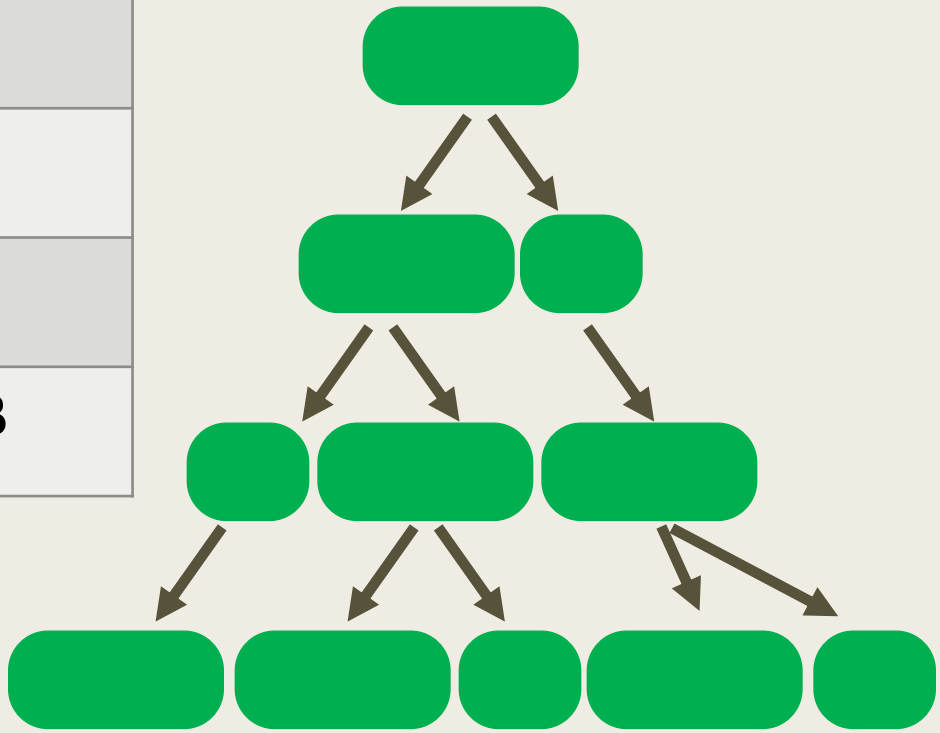
例3. G_3 = <N, P, S> を次で定める。

N = {A, B}, S = A, P = {A → AB, B → A}

Lシステムでは、各記号に対して毎回書き換え規則が適用される。

時刻（Pの適用回数）	記号列
n = 0	A
n = 1	AB
n = 2	ABA
n = 3	ABAAB

G_3は deterministic なので時刻に対して記号列がただひとつ定まる。



例4. (カントール集合)

G_4 = <N, P, S> を次で定める。

N = {A, B}, S = A, P = {A → ABA, B → BBB}

時刻 （Pの適用回数）	記号列
n = 0	A
n = 1	ABA
n = 2	ABABBBABA
n = 3	ABABBBABABBBBBBBBBABABBBABA

- Aを線分、Bを抜き取られた線分と対応させるとカントール集合が得られる。

■ Aを線分、Bを抜き取られた線分と対応させるとカントール集合が得られる。

時刻 （Pの適用回数）	記号列	カントール集合
n = 0	A	――――――――――
n = 1	ABA	―――　　―――
n = 2	ABABBBABA	―　―　　―　―

■ このように記号に操作を対応づけることでフラクタル図形を作ることができる。

L-system を用いたフラクタル

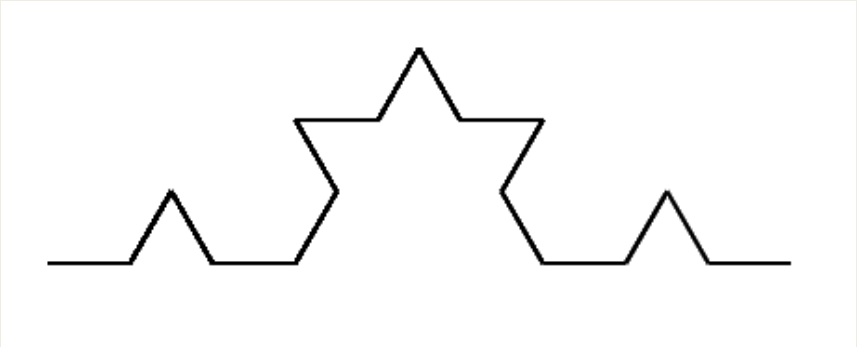
例5. (コッホ曲線)

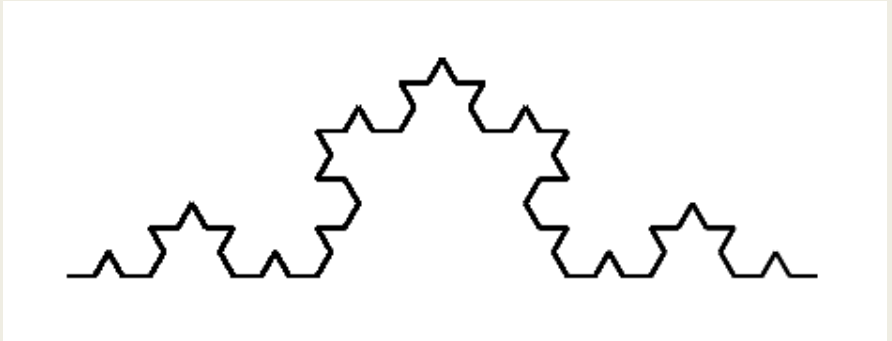
G_5 = <N, P, S> を次で定める。

N = {F, +, -}, S = F, P = {F → F+F--F+F}

各記号に対する操作の対応は次の通り

- F はタートルによる直線の描画
- + はタートルを反時計回りに 60° 回転
- - はタートルを反時計回りに -60° 回転
  （時計回りに 60° 回転）



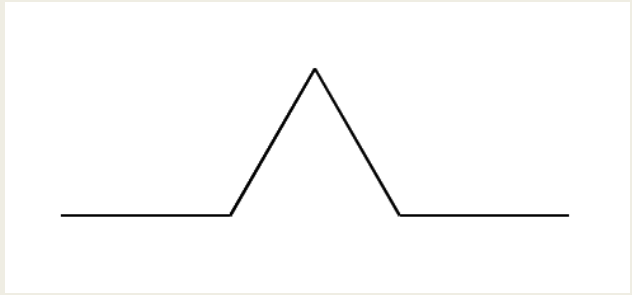


L-system の実装

例5. (コッホ曲線)

G_5 = <N, P, S> を次で定める。

N = {F, +, -}, S = F, P = {F → F+F--F+F}



フラクタルを描画する前に、まずは L-system （文字列の書き換えプログラム）のみ実装してみよう。以下 JavaScript のコードを紹介。

```
//非終端記号の集合（配列）の定義
let N = ["F", "+", "-"];
//開始記号の定義
let S = "F";
//書き換え規則の集合の定義
let P = [["F", "F+F--F+F"]];
```

- Nは実際は定義しなくてもよい。
- P は書き換え規則が複数個あるものを想定するため、長さ2の配列の配列

L-system の実装

書き換えを1ステップのみ行う関数 rewrite を実装。

```
//非終端記号の集合（配列）の定義
let N = ["F", "+", "-"];
//開始記号の定義
let S = "F";
//書き換え規則の集合の定義
let P = [["F", "F+F--F+F"]];
```

```
61    //書き換えを1ステップだけ行う関数
62    function rewrite(string){
63        tmp = "";
64        for(let i=0; i<string.length; i++){
65            let flag = 0;
66            for(let j=0; j < P.length; j++){
67                if(string[i] == P[j][0]){
68                    flag = 1;
69                    tmp += P[j][1];
70                }
71            }
72            if(flag == 0){
73                tmp += string[i];
74            }
75        }
76        return tmp;
77    }
```

- 入力された文字列を1文字目から順に見て行って、Fに出会ったら F+F--F+F に書き換える。
- 簡単のため j = 0 と思ってよい。（2個目の for 文は実際は for文と思わなくてよい。）

[練習]L-system の実装(1)

例題.
N, S, P を右のように定義したとき、
rewrite(S) の結果を答えよ。

```
//非終端記号の集合（配列）の定義
let N = ["F", "a", "b"];
//開始記号の定義
let S = "aFb";
//書き換え規則の集合の定義
let P = [["F", "aF"]];
```

```
61    //書き換えを1ステップだけ行う関数
62    function rewrite(string){
63        tmp = "";
64        for(let i=0; i<string.length; i++){
65            let flag = 0;
66            for(let j=0; j < P.length; j++){
67                if(string[i] == P[j][0]){
68                    flag = 1;
69                    tmp += P[j][1];
70                }
71            }
72            if(flag == 0){
73                tmp += string[i];
74            }
75        }
76        return tmp;
77    }
```

- 63行目 tmp = “”
- i に関する for文がスタート
- i = 0 → S[i] = “a”
- “a” は if の条件を満たさない
- tmp = “a”
- i = 1 → S[i] = “F”
- “F”は if の条件を満たす
- tmp = “aaF”
- i = 2 → S[i] = “b”
- tmp = “aaFb”

- rewrite(S) の結果は “aaFb”
- rewrite(rewrite(S)) の結果は “aaaFb”

[練習]L-system の実装(2)

例題. N, S, P を右のように定義したとき、rewrite(S) の結果を答えよ。

```
//非終端記号の集合（配列）の定義
let N = ["F", "G", "a", "b"];
//開始記号の定義
let S = "bFGb";
//書き換え規則の集合の定義
let P = [["F", "FG"], ["G", "abG"]];
```

```
61  //書き換えを1ステップだけ行う関数
62  function rewrite(string){
63      tmp = "";
64      for(let i=0; i<string.length; i++){
65          let flag = 0;
66          for(let j=0; j < P.length; j++){
67              if(string[i] == P[j][0]){
68                  flag = 1;
69                  tmp += P[j][1];
70              }
71          }
72          if(flag == 0){
73              tmp += string[i];
74          }
75      }
76      return tmp;
77  }
```

- 63行目 tmp = “”
- i に関する for文がスタート
- i = 0 → S[i] = “b”
- “b” は if の条件を満たさない
- tmp = “b”
- i = 1 → S[i] = “F”
- “F”は j=0 の if の条件を満たす
- tmp = “bFG”
- i = 2 → S[i] = “G”
- “G”は j=1 の if の条件を満たす
- tmp = “bFGabG”
- i = 3 → S[i] = “b”
- “b”は if の条件を満たさない
- tmp = “bFGabGb”

各記号に対する操作を定義

各記号に対して以下の操作を定義する。

- F はタートルによる直線の描画
- + はタートルを反時計回りに 60° 回転
- - はタートルを反時計回りに -60° 回転
  （時計回りに 60° 回転）

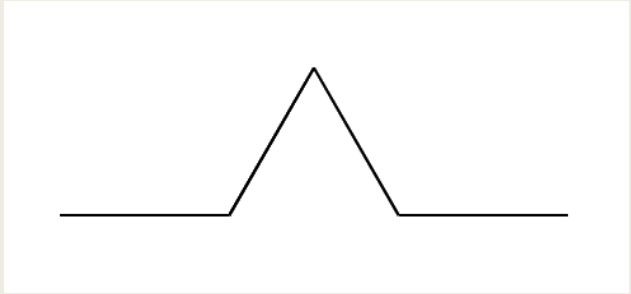
switch 文を使って定義する。

各記号に対する操作を定義

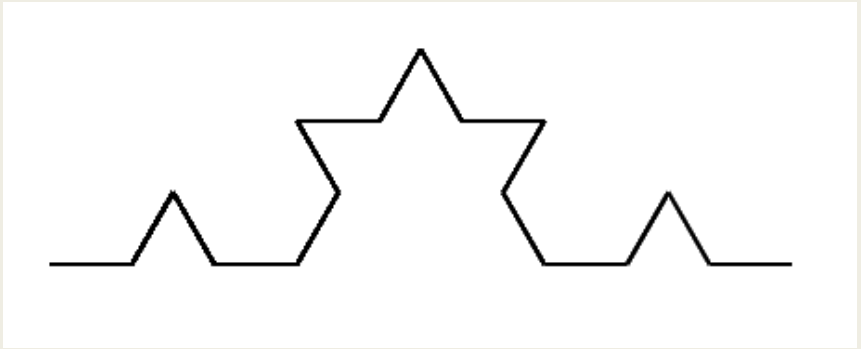
31, 34~37, 39~41行目は無視してよい（JavaScript的な事情）

```
30      function prepare(e){
31          let level = Number(e.id[0])
32          //文字列を生成する
33          let generated_string = S;
34          if(level == 0){
35              document.getElementById("generated_string_id").textContent = generated_string;
36          } else {
37              for(let i=0; i<level; i++){
38                  generated_string = rewrite(generated_string);
39              }
40              document.getElementById("generated_string_id").textContent = generated_string;
41          }
42          //生成した文字列に対してタートルを動かす
43          for(let i=0; i<generated_string.length; i++){
44              console.log(generated_string[i]);
45              switch(generated_string[i]){
46                  case "F":
47                      turtle.forward(size/(3**level));
48                      console.log("F");
49                      break;
50                  case "+":
51                      turtle.turn(60);
52                      console.log("+");
53                      break;
54                  case "-":
55                      turtle.turn(-60);
56                      console.log("-");
57                      break;
58              }
59          }
60      }
```

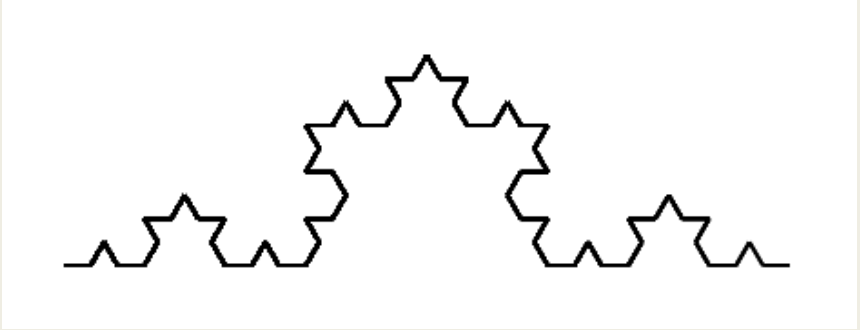
実行結果



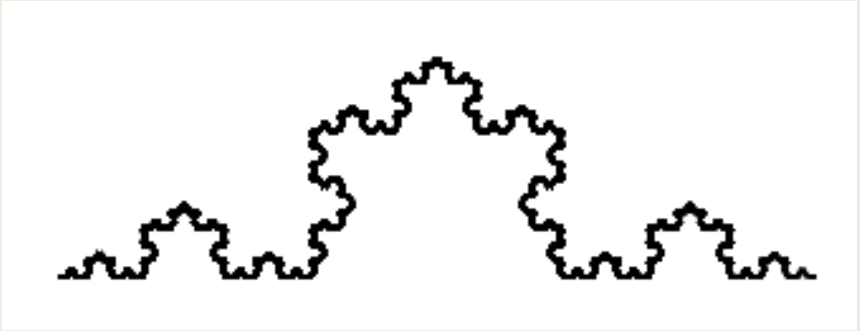
時刻 1（1回の書換）



時刻 2（2回の書換）

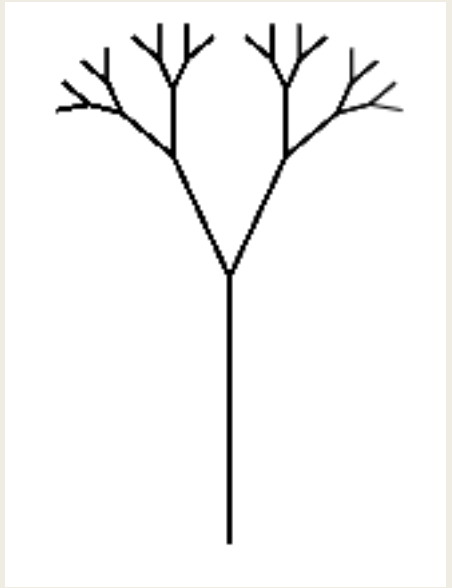


時刻 3



時刻 4

その他の例（木）



例6.

G_6 = <N, P, S> を次で定める。

N = {F, G, +, -, [,], |, s, t}, S = F,

P = {F →s|[+F][-F]t}

各記号に対して以下の操作を定義する。

- F は設定長だけタートルを進め直線を描く
- G はタートルを進めるだけ（線は描かない）
- + はタートルを反時計回りに 25° 回転
- - はタートルを反時計回りに -25° 回転
- [はタートルの位置と方向をスタックに保存
-] はタートルの位置と方向をスタックから取り出す
- | は書き換えの回数に応じた長さだけタートル進め、直線を描く
- s, t は書き換え回数を読み取るためのメモ

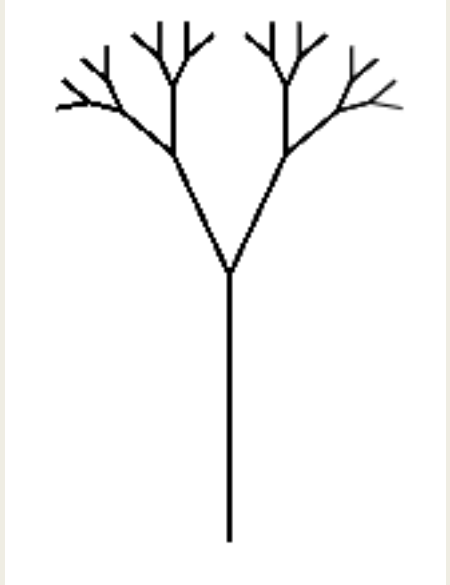
記号 s, t について

例6.

G_6 = <N, P, S> を次で定める。

N = {F, G, +, -, [,], |, s, t}, S = F,

P = {F →s|[+F][-F]t}



1回目の書き換えで生成される文字列は

s|[+F][-F]t

2回目の書き換えで生成される文字列は

s|[+s|[+F][-F]t][-s|[+F][-F]t]t

赤線の|が何回目の書き換えで登場する|なのかをどう判別したら良いだろうか？

- sに出会ったらカウンターを1増やす
- tに出会ったらカウンターを1減らす

実装

- 座標や角度を保存するスタック、ステップ数を保存するカウンターを用意しておく。

```
8       let x = 500;
9       let y = 900;
10      let size = 200; //基本の長さ
11      let angle = -Math.PI/2; //x軸の正の方向を0として、今どの角度を見ているかを表す変数
12      let stack_x = [];
13      let stack_y = [];
14      let stack_angle = [];
15      let step_counter = 0;
```

- 各記号に対応する操作を定義

```
58              switch(generated_string[i]){
59                  case "F":
60                      turtle.forward(size*(0.5**level));
61                      break;
62                  case "G":
63                      turtle.forward_only(size);
64                      break;
65                  case "+":
66                      turtle.turn(25);
67                      break;
68                  case "-":
69                      turtle.turn(-25);
70                      break;
```

```
71                  case "[":
72                      stack_x.push(x);
73                      stack_y.push(y);
74                      stack_angle.push(angle);
75                      break;
76                  case "]":
77                      x = stack_x.pop();
78                      y = stack_y.pop();
79                      angle = stack_angle.pop();
80                      break;
81                  case "|":
82                      turtle.forward(size*(0.5**step_counter));
83                      break;
84                  case "s":
85                      step_counter++;
86                      break;
87                  case "t":
88                      step_counter--;
89                      break;
90              }
```

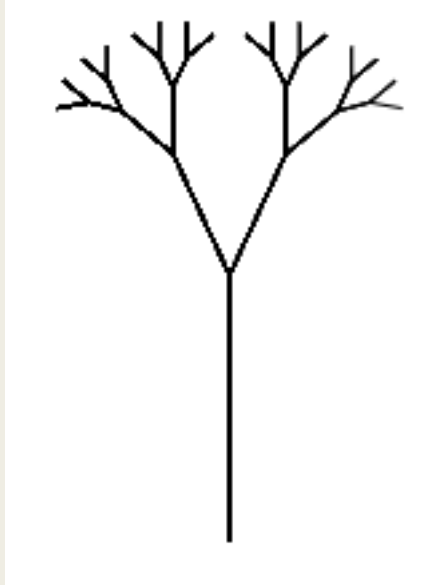
実装

例6.

G_6 = <N, P, S> を次で定める。

N = {F, G, +, -, [,], |, s, t}, S = F,

P = {F →s|[+F][-F]t}



■ N, S, P を定義

```
111     //非終端記号の集合（配列）の定義。実際はこれは不要。
112     let N = ["F", "G", "+", "−", "[", "]", "|"];
113     //開始記号の定義
114     let S = "F";
115     //書き換え規則の集合の定義
116     let P = [["F", "s|[+F][-F]t"]];
```

練習.

「chinou03.html」内のプログラムの P の部分を上の例6のように変更し、右上のような絵が出力されることを確かめよう。
以降、必要に応じて初期位置やキャンバスのサイズを変更せよ。

[演習]いろいろな例を打ち込んでみよう

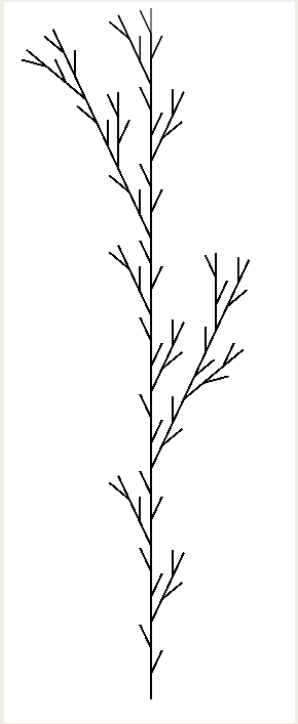
例7.

G_7 = <N, P, S> を
次で定める。

N = {F, G, +, -, [,], |, s, t},

S = F,

P = {F →sF[-F]F[+F]Ft}



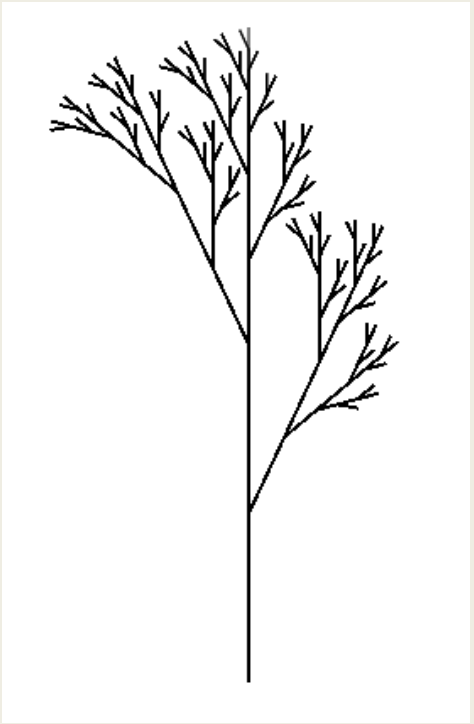
例8.

G_8 = <N, P, S> を
次で定める。

N = {F, G, +, -, [,], |, s, t},

S = F,

P = {F →s|[-F]|[+F]Ft}



[演習]いろいろな例を打ち込んでみよう

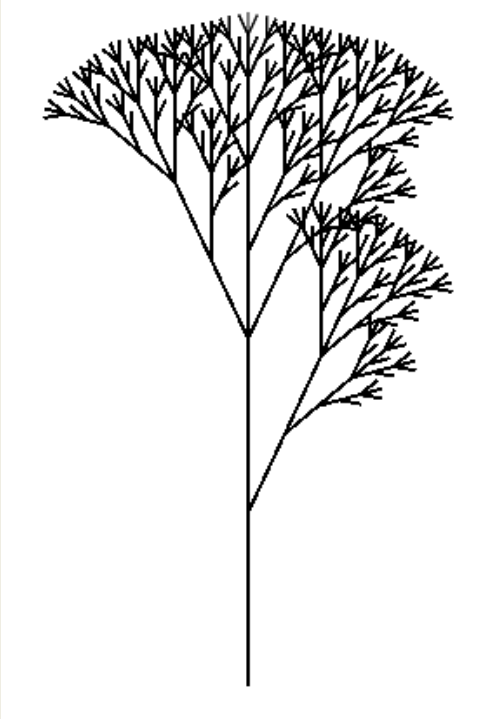
例9.

G_9 = <N, P, S> を
次で定める。

N = {F, G, +, -, [,], |, s, t},

S = F,

P = {F → s|[-F]|[+F][-F]Ft}



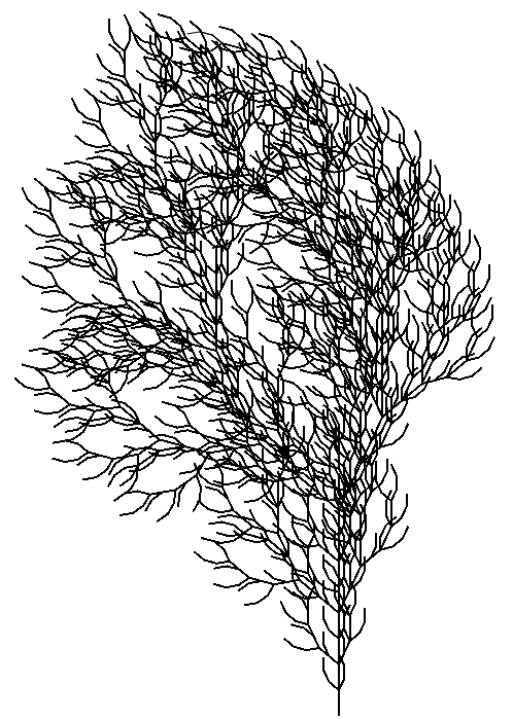
例10.

G_{10} = <N, P, S>

N = {F, G, +, -, [,], |, s, t},

S = F,

P = {F →sFF+[+F-F-F]-[-F+F+F]t}



例11.

G_{11} = <N, P, S> を次で定める。

N = {F, G, +, -, [,], |, s, t},

S = F,

P = {F → s|[++++++F][-------F]-|[++++F][------F]-|[+++F][-----F]-|Ft}

ただし、角度を8°とする。

演習問題.

角度や、書き換え規則の集合Pをいじって遊んでみよう。

演習問題.

第1回の授業で作ったフラクタル図形を L-system でも生成してみよう。

演習問題.

「L-system フラクタル」で検索して色々調べてみよう。