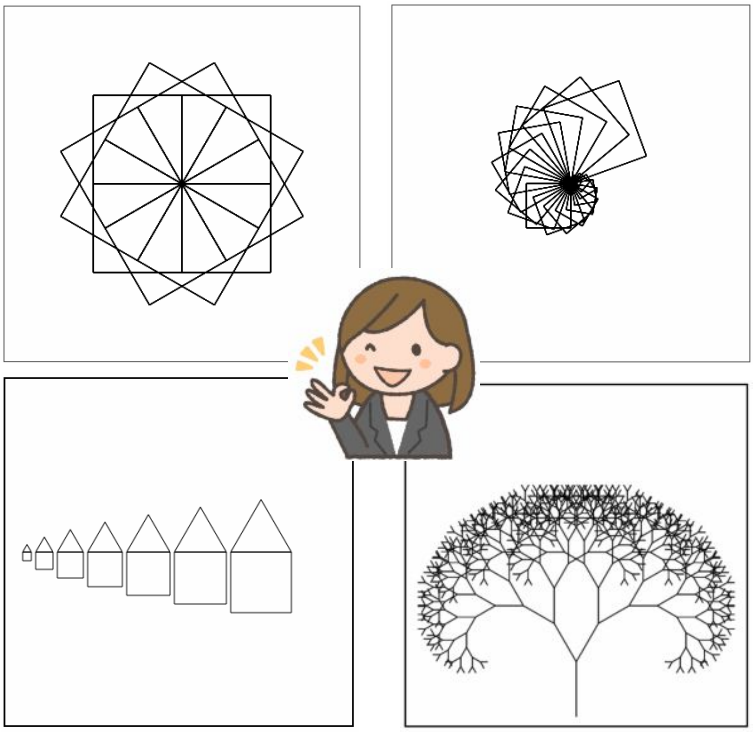
# ななちゃんのIT教室

## タートルグラッフィクスでJavaScript入門の巻

by nara.yasuhiro@gmail.com

プログラミングをまったく知らなかったななちゃんが
グラフィックスプログラムを作れるようになるまでのお話

第 0.6 版　2017 年 5 月 7 日



もくじ

第1回　コンピュータで絵を描こう
第2回　簡単な図を描いてみよう
第3回　指定した回数だけ繰り返す
第4回　回数をコンピュータに判断してもらおう
第5回　まとまりのある絵を部品化しよう
第6回　大きさを変化できる部品
第7回　まとめ
タートルグラフィックスとは
研究課題（1）
研究課題（2）　再帰
研究課題（3）　フラクタル

# 第1回 コンピュータで絵を描こう

なな: これって、朝日新聞の「ののちゃんのDO科学」のパクリ？

先生: パロディって言ってちょうだい。家政婦のミタ（「家政婦は見た」のパロディ）、クレヨンしんちゃんのダズニーランド（「ディズニーランド」のパロディ）みたいなものよ。

フリー素材
http://freeillustration.net

なな: 文部科学省が「2020年からの小学校での『プログラミング教育の必修化』を検討中」と聞いたけど、プログラミングをまったく知らなくて、とても不安なの。

先生: 小学生に教えるくらいだから、とっても簡単よ。米国で小学生にプログラミングを教えるために考案されたタートルグラフィックスを使って勉強しましょう。

なな: 「タートル」って、亀のこと？ 「亀さん、動いてね」と書くの？

先生: それに近いけど、人間がしゃべることばは不正確になりやすいし、コンピュータが理解するのは難しいので、**プログラミング言語**を使って表現するの。その代表選手が「JavaScript」という言語なの。

フリー素材
http://freeillustration.net

なな: ふうん。どうやってコンピュータに伝えるの？

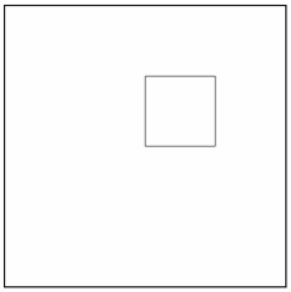
```
fd(100); rt(90);
fd(100); rt(90);
fd(100); rt(90);
fd(100); rt(90);
```

先生: 右のプログラムは正方形を描くプログラムよ。
普通の横書きの文章のように、左上から、右へ、次に2行目、3行目という順に読んでね。
「亀さん、100歩前進（forward 100）、右に90度回転（right 90）ということを4回繰り返してね」という意味なの。

なな: 英語を使うのね。

先生: アメリカの大学で作ったものだし、ベースとして使う JavaScript もアメリカ製だからね。

なな: どんな正方形が描けるの？



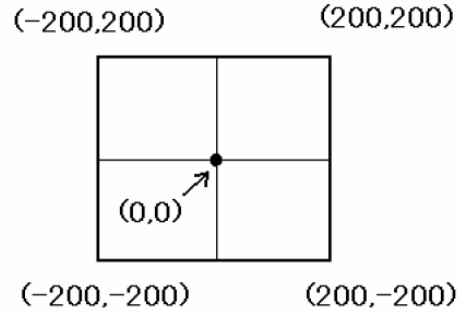
先生: こんな感じ。外側の四角がキャンバスという画面全体で中にある小さい四角が、プログラムで描いた正方形よ。

なな: ふう！ ちょっと疲れそう。

先生: 今回のお話をまとめておきましょうね。タートルに話しかけることばは、ふたつだけ。

| | |
|---|---|
| fd(5) | は前へ (forward) 5歩進むこと、 |
| rt(30) | は右に 30 ° 回転する (right) ことを指令する。 |

画面（キャンバス）の大きさは縦400歩、横400歩で、亀ははじめに中央にいて上を向いているの。




なな: わかったわ！

やってみよう

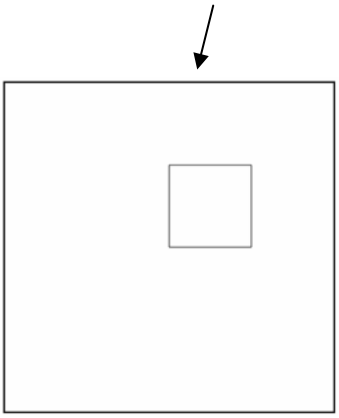
turtle.html

```
<!DOCTYPE html>
<html><body><script>
var canvas = document.createElement('canvas');
document.body.appendChild(canvas);
canvas.width    = canvas.height = 400;
canvas.style.border = "solid 2px";
var ctx = canvas.getContext('2d');
var x = y = a = 0;

function fd(step) {
   ctx.beginPath(); ctx.moveTo(200+x, 200-y);
   x += step * Math.sin(a * Math.PI / 180.0);
   y += step * Math.cos(a * Math.PI / 180.0);
   ctx.lineTo(200+x, 200-y); ctx.stroke();
}
function rt(angle) { a += angle; }

fd(100); rt(90); fd(100); rt(90);
fd(100); rt(90); fd(100); rt(90);
</script></body></html>
```

エディタでこの内容を turtle.html というファイルに書き込む。ファイルのアイコンをダブルクリックすると下記のような画面になる。



この部分以外は理解できなくてかまわない

この部分以外は毎回同じなのでコピーして使う

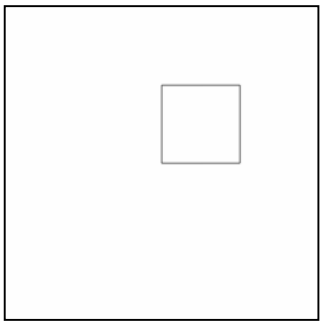
---

sin と cos が、通常と逆になっているが、これは、数学一般に右向きが 0° であるのに対して、タートルグラフィックスでは、上向きが 0° であるためである。sin ( x + 90° ) = cos ( x )、cos ( x - 90° ) = sin ( x )。

# 第2回　簡単な図を描いてみよう

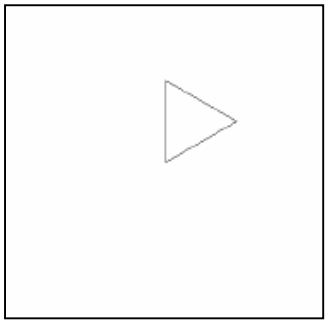
なな：前回、正方形を描く方法を勉強しました。

```
fd(100); rt(90);
fd(100); rt(90);
fd(100); rt(90);
fd(100); rt(90);
```



先生：よく覚えていたわね！　今回は、三角形を描く
プログラムを紹介しましょう。

```
fd(100); rt(120);
fd(100); rt(120);
fd(100); rt(120);
```



なな：ううん。三角形の角度って「60度」じゃなかったっけ。

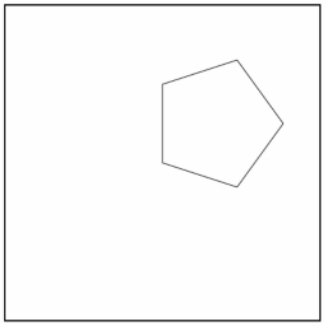
先生：ちょと違うのね。3 回 回転して、1 周、つまり 360度
回転するから、360÷3＝120度になるの。
「三角形の内角の和は 180度」とか、内角、外角なんて
いうより、わかりやすいでしょ？

なな：わかったような、わからないような。

先生：ためしに、五角形を描いてみましょう。

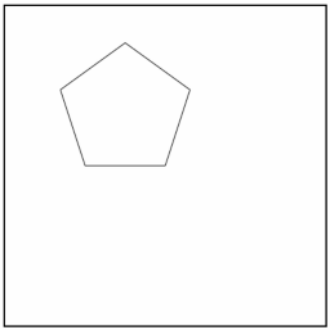
なな：「100歩前進して 右に 72°(= 360°÷ 5) 回転する」
ということを 5 回繰り返せば描ける、5 回右に
72° 曲がって1回転（72×5=360°）といいうことね。

```
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
```



先生：正解！。方向を調整したかったら、

```
rt(-90);
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
fd(100); rt(72);
```



なな：ちょっと左上寄りね。

先生：がまん、がまん。

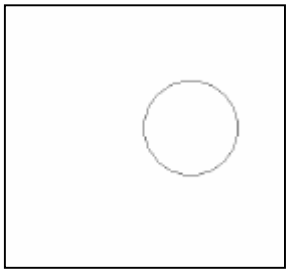
## 第3回 指定した回数だけ繰り返す

先生： 今回は、まず、20角形を描くことを考えましょう。

なな：「fd(100); rt(18);」を 20 回書けば良いのね。

先生： それでは、プログラムがごちゃごちゃするし、画面からはみ出してしまうわ。今回は下記のようなプログラムを勉強しましょう。

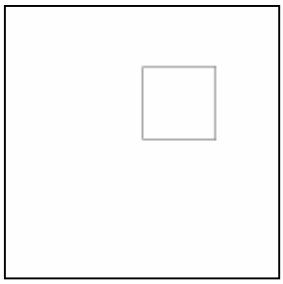
```
for (var i=1; i<=20; i=i+1)  {
      fd(20); rt(18);
}
```



「for (var i=1; i<=20; i=i+1) { ～ }」 というのは、
～ の部分を 20 回実行するという意味なの。

なな：「～」の部分を、別の行にしても良いのね。じゃあ
正方形を描くのは、これでも良いのね、

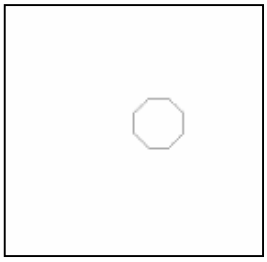
```
for (var i=1; i<=4; i++) {
     fd(100); rt(90);
}
```



先生：細かいことはともかく、今は、やりたい回数を
表現できれば問題ないわ。8 角形で練習してみましょうね。

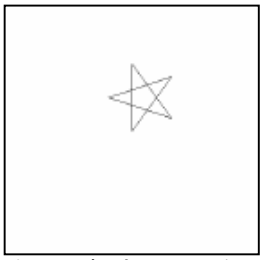
なな： 一辺の長さを 30 くらいにして、1回に45°（360°÷ 8）右回転ということで。できた！

```
for (var i=1; i<=8; i++) {
     fd(30); rt(45);
}
```



先生： 正解！ じゃあ、プレゼント。

```
for (var i=1; i<=5; i++) {
     fd(100); rt(144);
}
```

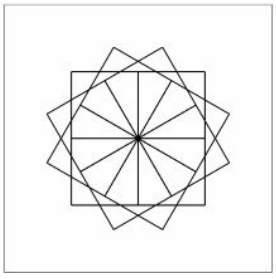


144 × 5 ＝ 720。2回転して原点に戻るの。

なな：すごすぎ！

先生： これもプレゼント。「1辺の長さが100歩の正方形を描いては 30° 右に回転する」ことを12回繰り返すプログラム。

```
for (i=1; i<=12; i++) {
     for (j=1; j<=4; j++) {
          fd(100); rt(90);
     }
     rt(30);
}
```



なな： アンビリーバボ！

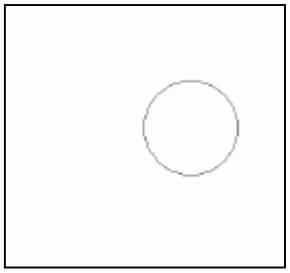
## 第4回 回数をコンピュータに判断してもらおう

先生: 前回は、4 回とか、20 回とか、回数を指定してコンピュータに繰り返してもらったけど、今回は、繰り返し回数をコンピュータに決めてもらう方法を勉強しましょう。

なな:え？

先生: 20歩前進し、18°右回転することを繰り返していると、いつかは出発点に戻り、すでに描いた絵をなぞるようになるわね。何回目の繰り返しで出発点に戻るかを事前に計算して決めることはできるけど、コンピュータ自身（プログラム自身）に判断させることもできるのよ。下のプログラムは、
「20歩前進して 18°右回転することを、タートルが真上を向くまで繰り返してね」という意味なの。

```
for ( ; ; ) {
    fd(20); rt(18);
    if (a % 360 == 0) { break; }
}
```



なな:ふうん。

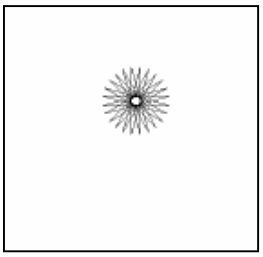
先生: 「for ( ; ; ) { … }」は、「…」を無限に繰り返してねという意味で、「if (a % 360 == 0) { break; }」は、もし、亀さんの方向 （ a ） が、0 か、 360 か、720 とか、つまり、360 で割り切れる角度になったら、繰り返しを中止してねという意味なの。「a % 360」は、a を 360 で割った余り、「== 0」はそれがゼロに等しい。「if」は「もし」という意味なの。

なな:ちょっと自信がないな。

先生: 今は、こういうもんだと割り切って、ちょっと練習してみましょう。ななちゃん、「100歩進み 165°右に回ることをタートルの向きが真上になるまで繰り返す」というプログラムはどうなるかな？

なな: こんな感じ？ わあ、星みたいになった！

```
for ( ; ; ) {
    fd(100); rt(165);
    if (a % 360 == 0) { break; }
}
```



先生: 何回 165 ° 回転したら、真上を向くかは、計算できるかも知れないけど、コンピュータに判断してもらえば済むのね。

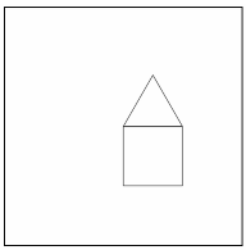
なな: 数学が苦手な私向きだ！

# 第5回 まとまりのある絵を部品化しよう

先生: 前回は、たとえば「ABBBC」というような構成のプログラムの「B」の部分を「繰り返し」で効率良く表現しました。
今回は「ABCBBDB」というような構成のプログラムの「B」の部分を効率良く表現する方法を勉強しましょう。

なな: 「B」が何回も出てくるけど、不規則に出てくるというか、繰り返しとはちょっと違うということね。

先生: 前回までに学んだ知識だけで家(正方形の上に正三角形)を描こうとすると以下のプログラムのようになるわね。



```
1: rt(30);
2: for (var i=1; i<=3; i=i+1) { fd(100); rt(120); }
3: rt(-30); rt(90);
4: for (var i=1; i<=4; i=i+1) { fd(100); rt(90); }
```

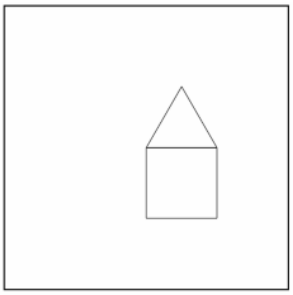
なな: 2 行目で、正三角形を、4行目で正方形を描いているのね。
4 行目で、正方形を描いているのね。1 行目と 3 行目は？

先生: 三角形と正方形を表示する傾きを調整するものよ。
正方形の上にうまく正三角形が乗るようにね。
3行目の rt(-30); は 左に 30°回転するという意味。

このプログラムを他人が書いたプログラムだとして、それを初めて「読む」ことを考えると、分かりにくいわね。
「正三角形と正方形を組み合わせて家を描く」と、すなおに表現できるといいわね。それから、このプログラムをもとにして、いくつかの家からなる複雑な絵を描こうとするとどうなるかしら。

なな: 家を描くプログラム部分を何度も書くことになってうんざりするわね。

先生: 一度作ったプログラムを部品化し、それを用いてさらに高度なプログラムを作れると良いでしょ。下のプログラムは「正三角形を描くプログラム」と「正方形を描くプログラム」を部品化した例です。



```
 10: function triangle() {
 11:     for (var i=1; i<=3; i=i+1) { fd(100); rt(120); }
 12: }
 20: function square() {
 21:    for (var i=1; i<=4; i=i+1) { fd(100); rt(90); }
 22: }
100: rt(30); triangle(); rt(-30);
101: rt(90); square(); rt(-90);
```

10 ～22 行が部品化を行なっている部分で、行100 以降がその部品を利用している部分なの。

このように、正三角形を描く処理や正方形を描く処理を部品化しておくと、triangle(); と書けば正三角形、square(); と書けば正方形が描けるようになるのよ。部品の名前が内容を連想させるようなものにしてあれば行100 以降だけを注意して読めばプログラムの内容が大体理解できでしょ。読み易さの点で良くなったと思う。

部品化している部分を「正三角形を描くメソッド triangle の定義」(行10～12)、「正方形を描くメソッド square の定義」(行20～22) と呼びます。

なな: すごすぎ！

## 第6回 大きさを変化できる部品

先生：さらに正三角形や正方形の一辺の長さまで指定できるようにするには以下のように定義を直すの。

```
10:  function triangle(size) {
11:        for (var i=1; i<=3; i=i+1) { fd(size); rt(120); }
12:   }
20:  function square(size) {
21:        for (var i=1; i<=4; i=i+1) { fd(size); rt(90); }
22:   }
```

このように定義しておくと、triangle(20); と書けば一辺の長さが 20の正三角形、square(50); と書けば 一辺の長さが50の正方形が描けるようになるわ。

この場合、行100 以降、つまりメソッドを利用する部分は以下のように書き換えます。

```
100:  rt(30); triangle(100); rt(-30);
101:  rt(90); square(100); rt(-90);
```

メソッド定義 triangle(int size) の size を仮引数 (かりひきすう) と呼び、
メソッド呼び出し triangle(100); の 100 を実引数 (じつひきすう) と呼びます。
仮引数の名前 (例: size) は自分の好みで決めて良いの。意味を連想しやすいものが良いわね。
メソッド triangle の仮引数の名前とメソッド square の仮引数の名前は同じであっても互いに干渉しなません。もちろん行10 の size と 行11 の size とは同じ名前で統一しなければいけないわ。

さらに、家を描く機能までもメソッドにまとめてしまうと以下のようになります。

triangle、square の定義の後に以下の house の定義を追加する。

```
30:  function house(size) {
31:       rt(30); triangle(size); rt(-30);
32:       rt(90); square(size); rt(-90);
33:  }
```

これに対応して、100行以下 の部分は以下のようになる。

```
100:  house(100);
```

以上をまとめると

```
10:   function triangle(size) {
11:        for (var i=1; i<=3; i=i+1) { fd(size); rt(120); }
12:   }
20:   function square(size) {
21:        for ( var i=1; i<=4; i=i+1) { fd(size); rt(90); }
22:   }
30:   function house(size) {
31:        rt(30); triangle(size); rt(-30);
32:        rt(90); square(size); rt(-90);
33:   }
100:  house(100);
```

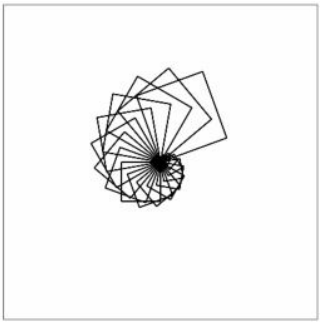
これで、「家」を部品として、いろいろな大きさの家を並べて家並みを描いたり、家と太陽と人間を定義して、複雑な絵を描いたりすることができます。

## 第7回 まとめ

メソッド traiangle、square、house の定義を他人にコピーさせ、「house(size) というメソッドは、現在のタートル位置の右側上下に 一辺 size の家を描く。タートルは元の位置に戻る」と説明すれば利用してもらえる。
「一辺の長さが n の正方形を描いて、20°右回転して、n を 5 増やす」ということを、一辺の長さが 5 から 90 まで繰り返すことにより、貝のようなデザインを描くプログラム。

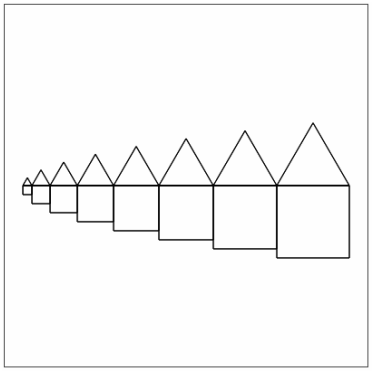
```
function square(size) {
    for (var i=1; i<=4; i=i+1) { fd(size); rt(90); }
}

for (var n=5; n<=90; n+=5) {
     square(n); rt(20);
}
```



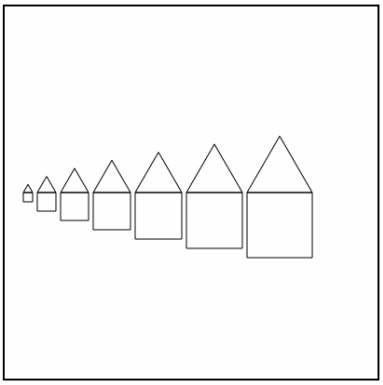
サイズ 10 から 80 までの家を横にならべるプログラム。

```
function triangle(size) {
    for (var i=1; i<=3; i=i+1) { fd(size); rt(120); }
 }
function square(size) {
    for (var i=1; i<=4; i=i+1) { fd(size); rt(90); }
}
function house(size) {
       rt(30); triangle(size); rt(-30);
       rt(90); square(size); rt(-90);
}
rt(-90); fd(180); rt(90);
for (var n=10; n<=80; n+=10) {
    house(n);
    rt(90); fd(n); rt(-90);
}
```



さらに発展させたプログラム。

```
function triangle(size) {
    for (var i=1; i<=3; i=i+1) { fd(size); rt(120); }
}
function square(size) {
    for (var i=1; i<=4; i=i+1) { fd(size); rt(90); }
}
function house(size) {
       rt(30); triangle(size); rt(-30);
       rt(90); square(size); rt(-90);
}
function move(n) {
    x = x + n;
}
move(-180);
for (var n=10; n<=70; n+=10) {
    house(n); move(n+5);
}
```



# タートルグラフィックスとは

南アフリカ出身、米MITに所属した数学者、計算機科学者、発達心理学者、シーモア・パパート（Seymour Papert、1928/3/1-）が、1960年代、コンピュータを導入して小学生の図形能力、論理的思考能力を養うために考案。1967年、LOGO言語として登場し、子供たちへのプログラミング教育に使われている。

数学者パパートは30歳から5年間、ジュネーブの発生認識論センターで、ピアジェと児童心理学を共同研究。。ジャン・ピアジェは発達心理学の大家。「子供は、自己の知識構造の積極的な建設者である」というピアジェの見方に触発された。60年代はじめ、MITでマーヴィン・ミンスキーとAI研究の中で子供の知識獲得の研究。

LOGO言語は、子供の教育用のプログラミング言語と位置づけられている。子供に知識を埋め込むものでなく、子供の概念形成を支援する目的で作られている。8歳から12歳位の子供にも興味を持って使えるように考えられた。タートル（亀）を媒介にコンピュータと対話しながらプログラムを作る。タートルに絵の書き方を教えるという形をとる。

タートルに何をさせればいいかを考える事から始める。指示を出し、思い通りにタートルが動かなければその理由を考えて指示を訂正する。試行錯誤を繰り返しながら子供達は自分の世界をコンピュータ上に作り上げてゆく。このプロセスの中で子供は失敗を恐れなくなり、創造することの意味と方法を学ぶ。バグとデバッキングこそが知識、知能を修得する最強のプロセス。

パパートの考え方は『マインドストーム―子供、コンピューター、そして強力なアイデア』(奥村貴世子訳、未來社、1995)に書かれている。パパートは子供のころから歯車に強い関心を示していた。いろいろな現象を、歯車の動きと関連づけて考えるようになった。そろばんの達人が、頭の中にそろばん珠を思い浮かべながら暗算をするようなもの。

> やがて私は、頭の中で歯車を回転させて、一連の原因と結果を生み出すこともできるようになった。『これがこっちに回るからあれはあっちに回って、だから……』 差動式歯車のようなものは特に気に入っていた。これは、二つの車輪にかかる抵抗の違いによって伝導軸の動きが実に多様に車輪へ分配されるため、単純で直接的な原因作用に従わないところが、格別おもしろく思われたのである。組織というものは、固定的、決定的ではなくても充分法則にかない、完全に理解し得るものだということを発見した時の興奮した気持は今も鮮やかに記憶している。

パパートの心の中にはいつも歯車があった。歯車で世界を考えていた。世界の中に歯車の軋む音を聞き、歯車の具合を直しながら美しい世界を組み立てていった。そんな世界に遊んだり、世界を測ったり、世界を覗いたりする道具。そのようなものがあれば、心の中に豊かな世界を建築していけるはず。パパートはそう考えた。しかし、歯車が万民に向いているとは考えなかった。万民に向いた、歯車以外のものを考えた。

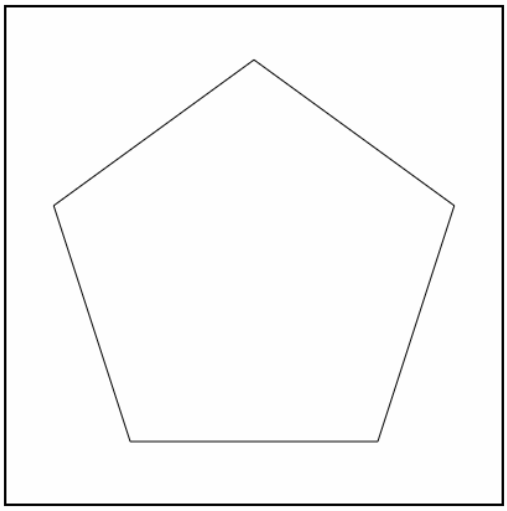
このように学んでいくためには、心の中の世界を建築していくための素材が必要となる。ピアジェはこの素材を準備してやれば子供もは自分で学んでいくと考えた。これが発生認識学の基礎になる。さらにパパートはこの素材こそが文化に準備されている、と考えた。文化と学びは切り離すことができない。

ディスプレイにタートル（かめ）と呼ばれる三角形があらわれる。三角形の頂点が前。前に100進めと指示すれば100進み、50戻れと指示すれば50戻る。右に90度回転と指示すれば右に90度回転する。そしてこのタートルはペンを持っている。ペンを下ろせと指示すればペンを下ろし、ペンを上げろと指示すればペンをあげる。ペンを下ろしているあいだ、タートルを移動させればその軌跡が描かれる。

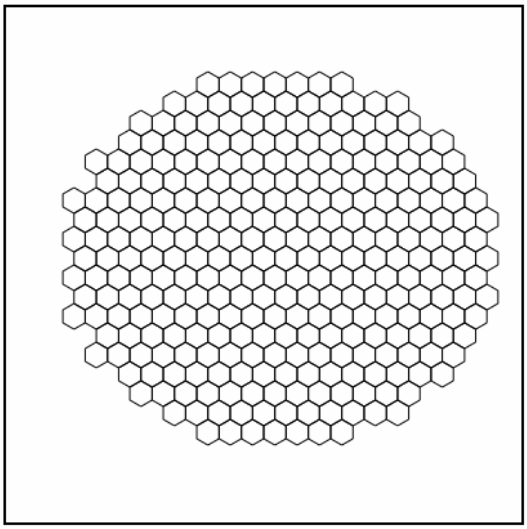
## 研究課題（1）

以下のプログラムを研究し、独自の応用プログラムを考えてみよう。

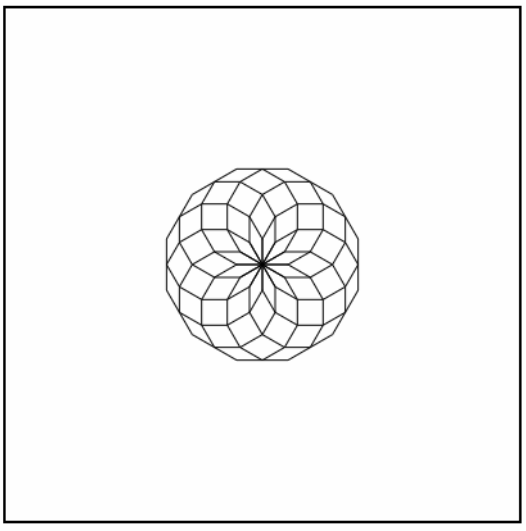
```
x = 100; y = -150; rt(-90);
fd(200); rt(72);
fd(200); rt(72);
fd(200); rt(72);
fd(200); rt(72);
fd(200); rt(72);
```



```
for (yy=-10; yy<10; yy++) {
    for (xx=-10; xx<10; xx++) {
        if (xx*xx+yy*yy >90) continue;
        if (yy % 2 == 0)
            x = (xx * 10 + 5) * Math.sqrt(3);
        else       x = xx * 10 * Math.sqrt(3);
        y = yy * 15;
        for (i=1; i<=6; i++) {
            fd(10); rt(60);
        }
    }
}
```



```
for (var i=0; i<12; i++) {
    for (var j=0; j<12; j++) {
        fd(20); rt(-30);
    }
    rt(-30);
}
```

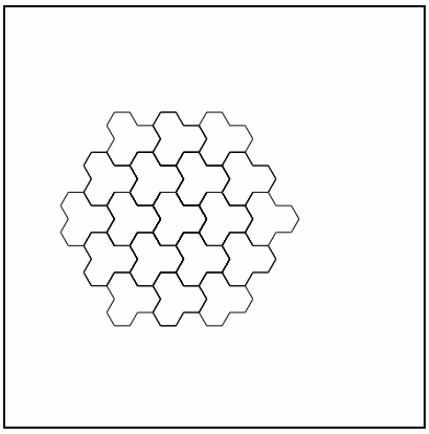


# 研究課題（2） 再帰

関数の中から、自分自身を呼び出すことを再帰という。検索エンジンなどで「再帰」を調べてみよう。

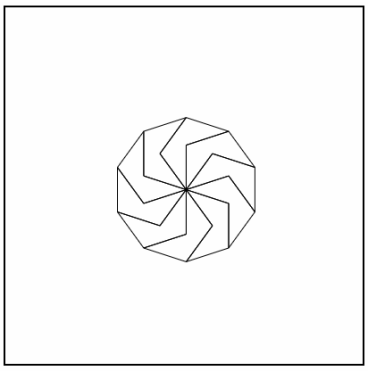
```
function hexagon(s,lv,tp) {
    var ag = (lv % 2 == 0)?60:-60;
    if (lv > 0) {
        hexagon(s,lv-1,tp); rt(ag); hexagon(s,lv-1,-tp); rt(ag);
        hexagon(s,lv-1,tp); rt(ag); hexagon(s,lv-1,-tp); rt(ag);
        hexagon(s,lv-1,tp); rt(ag); hexagon(s,lv-1,-tp); rt(ag);
    }
    rt(tp*30);fd(s/1.7);rt(-tp*60);fd(s/1.7);rt(tp*30);
}

var sz = 25;
x = -sz/1.7; y = -sz/1.7;
hexagon(sz,3,1);
```

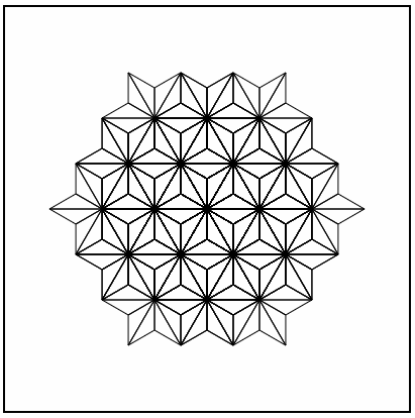


```
y = -70; x = 50; gen(50,9);

function gen(size,lv) {
    fd(size); rt(-72);
    fd(size); rt(144);
    fd(size); rt(72);
    fd(size); rt(-180);
    if(lv > 0) gen(size,lv-1);
    rt(252);
    fd(size); rt(144);
}
```



```
function one(s) {
        fd(s);rt(60); fd(s);
        rt(150); fd(s*Math.sqrt(3));rt(-150);
        fd(s);rt(-60); fd(s);
        rt(-150); fd(s*Math.sqrt(3));rt(150);
}
function ones(s,lv) {
    for(var i=0; i<6; i++ ) { one(s); rt(60); }
    if (lv > 0) {
        for(var i=0; i<6; i++ ) {
            rt(30+60*i); fd( s*Math.sqrt(3)); rt(-30-60*i);
            ones(s,lv-1);
            rt(30+60*i); fd(-s*Math.sqrt(3)); rt(-30-60*i);
}   }   }
ones(30,2);
```



## 研究課題（3） フラクタル

フラクタルは、図形の部分と全体が自己相似になっているものなどをいう。タートルグラフィックスは現在のタートルの位置や角度をもとに命令を実行するという「相対性」が、フラクタルときわめて相性が良い。

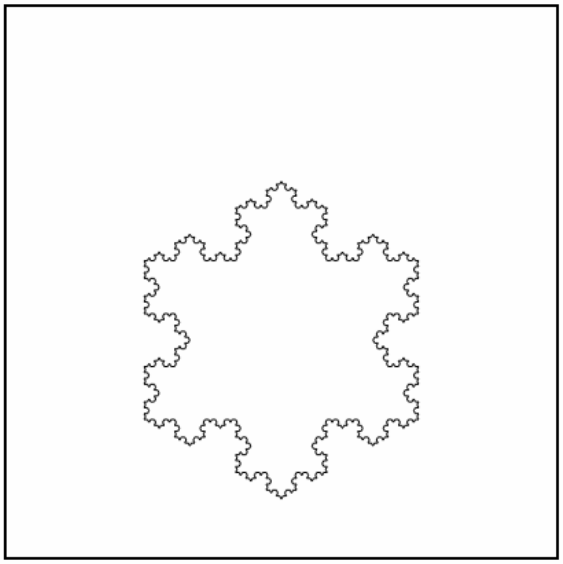
通常のグラフィックスは、通常は絶対座標になる。原点がどこにあるのか、y 座標は上下どちらにすすむとプラスなのかを常に意識する必要がある。部品となる関数を作るにも、意識して相対性を入れないと、固定された位置に図形を描いてしまう。角度がかたむくと、とたんに sin、cos の嵐になってしまう。タートルグラフィックスでは、それらをほとんど意識しないで済む。たとえば、

```
function square(s)  {   for(var i=1;4<=; i++)  { fd(s); rt(90); } }
```

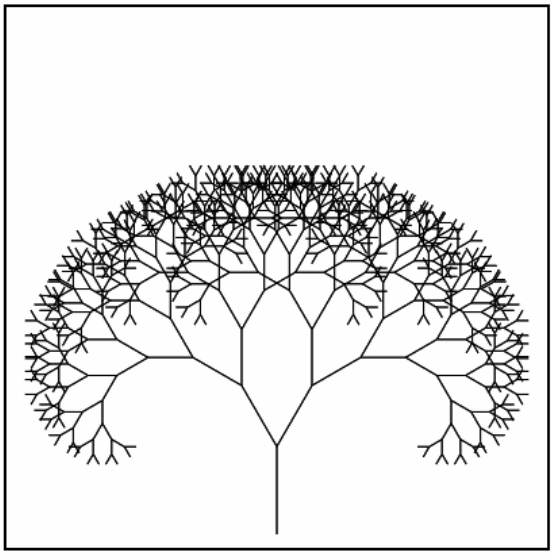
という定義をするだけで、タートルがどこにいても、どんな方向を向いていても、その位置、その方向で正方形を描画してくれる。再帰呼び出しで、s を徐々に小さくしてゆけば、容易に自己相似の図形を描画することができる。おもしろいな、こんな図形を描いてみたいと思ったら、研究してみよう。

```
function koch(n,s) {
    rt(30);
    k(n,s); rt(120); k(n,s); rt(120); k(n,s); rt(120);
    rt(-30);
}
function k(n,s) {
    if (n<=1) { fd(s); return; }
    k(n-1,s/3); rt(-60);
    k(n-1,s/3); rt(120);
    k(n-1,s/3); rt(-60);
    k(n-1,s/3);
}

x = -100; y = -100;
koch(5,200);
```



```
function tree(n,s) {
    if(n>0) {
        fd(s);rt(-30);
        tree(n-1,s*0.8);rt(60);
        tree(n-1,s*0.8);rt(-30); fd(-s);
    }
}
y = -190;
tree(10,65);
```



```
function gen(n,s) {
    g(n,s); rt(90); g(n,s); rt(90);
    g(n,s); rt(90); g(n,s); rt(90);
}
function g(n,s) {
    if (n<=1) { fd(s); }
    else {
        g(n-1,s/3); g(n-1,s/3); rt(90);
        g(n-1,s/3); rt(90); g(n-1,s/3); rt(90);
        g(n-1,s/3); rt(90); g(n-1,s/3); g(n-1,s/3);
} }

x = -100; y = -100;
gen(5,200);
```