

## リピーターコース Part1 Processing の基本

### 目次

1. Processing の基本
2. 文法について
3. 命令づくり

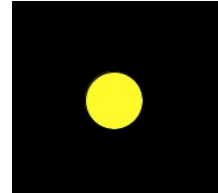
## 1-1. Processing の基本

初級編のテキストを参考にしながら次の問題をプログラミングしていこう。

(1) 星空 (図形描画、色、for 文、乱数)

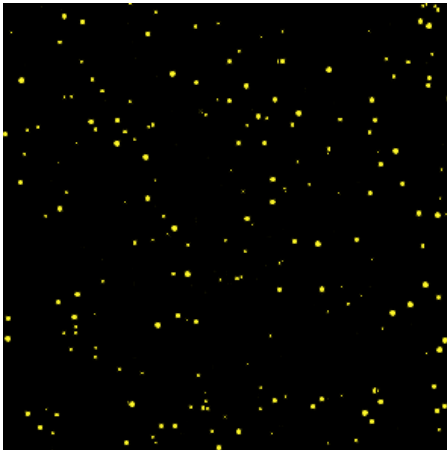
(i) 星をひとつ描く

- ・ 画面サイズを 300\*300 にする (size 命令)
- ・ 黒い背景を塗る (background 命令)
- ・ 星をひとつ描く (fill 命令と ellipse 命令で黄色の円を描く)



(ii) ランダムにたくさんの星を描く

- ・ 星がランダムな位置、ランダムな半径で描かれるようにする
- ・ for 文で数百個の星を描くようにする



## K 会 2012 年度冬季講習 情報講座

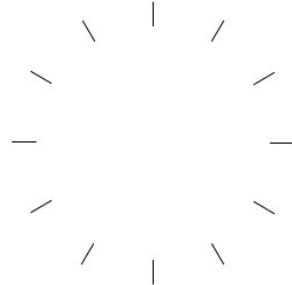
### (2) 時計 (変数、アニメーション、三角関数)

#### (i) 文字盤を描く

- ・ for 文と line 命令で、30 度間隔の 12 本の直線を描く。「30 度」は processing では  $2\pi/360$  倍されて、「 $30 \times 2\pi/360 = \pi/6$ 」という数 (ラジアン) で表される。

直線を描く位置は三角関数で決定する。

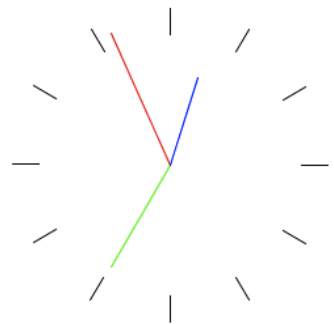
(補足資料を参考にせよ)



#### (ii) 針を描く

- ・ 時刻を表す変数 s を作成せよ。s は 0:00 から何秒経過したかを表す変数とする。
- このとき、分は  $s/60$ , 時間は  $s/60/60$  となる。
- ・  $(s/60/60)$  時  $(s/60)$  分  $(s)$  秒の位置に正しく 3 本の針が描かれるようにせよ。

右は  $s=45926$  のとき (12 時 34 分 56 秒)



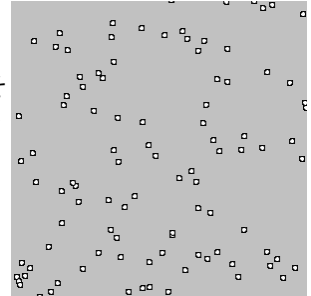
#### (iii) アニメーションさせる

- ・ アニメーションさせよう。今書いたプログラムに setup ブロックと draw ブロックを付け、各命令を適切な位置に配置する (画面サイズを決める size 命令は setup ブロックの中、直線の描画などは draw ブロックの中、変数 s の作製はブロックの外側)
- ・ draw ブロックが一回実行される度に 1 秒進むようにする。
- ・ frameRate 命令 (1 秒に何回 draw を実行するか) を setup ブロック内に書き、現実の 1 秒の間に時計が 10 秒進むようにせよ。

### (3) 雪が降る (アニメーション、変数、配列、マウス操作)

#### (i) 配列の用意

- ・ 雪玉の  $x$  座標と  $y$  座標を表す配列 `float[] x, y` を用意する。
- ・ 雪玉の  $x$  方向速度、 $y$  方向速度を表す配列 `float[] vx, vy` を用意する。
- ・ `setup` ブロックの中で各配列の確保 (`new` 文) を実行し、そのあとで、`for` 文を使って  $x, y, vx, vy$  の各要素  $x[i]$  などにランダムな数 ( $x, y$  なら画面内になるように、 $vx, vy$  は  $-1$  から  $1$  の間くらい) を代入する。
- ・ `draw` ブロックの中で、`for` 文を使って、各雪玉  $i$  の位置  $x[i], y[i]$  に `ellipse` で円を描くようにする。



#### (ii) 雪玉を動かす

- ・ `draw` が一回実行される度に、 $x[i]$  が  $vx[i]$  だけ、 $y[i]$  が  $vy[i]$  だけ加算されるようにする。これで雪玉  $i$  は  $(vx[i], vy[i])$  方向に直線運動するようになる。
- ・ `draw` 内で雪玉を描く前に `background(255,255,255)` を実行して、画面全体を真っ白に塗りつぶしておく (そうしないと前に書いたものが残ったままになる)。
- ・ `if` 文を使って、端にきた玉が反対側の端から出てくるようにする。例えば右端に着いた玉は、 $x[i]$  が画面の横幅分だけ引かれて左端から出てくるようにする。

#### (iii) マウス操作

- ・ マウスの方向に風が吹くようにする。マウスが画面の右側にあるときは雪玉も右側に流れるようにする。具体的には、 $x[i]$  や  $y[i]$  に  $vx[i], vy[i]$  を加算するところで、マウスの位置に応じた値をさらに加算するようにする。マウスの位置は  $(mouseX, mouseY)$  である。

#### (iv) 見た目を良くする

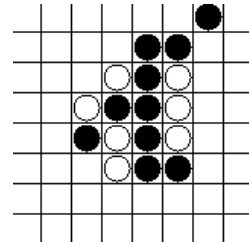
- ・ 雪玉を描くところで、`ellipse` を何度か実行し、白色が外側ほど薄く、内側ほど濃くなるように雪玉を描くと、右図のように綺麗になる。下：拡大図



### (4) オセロ (if 文、マウス操作、二次元配列)

#### (i) 盤面の用意

- ・ int 型の二次元配列 `m[][]` を用意する。
- ・ `m[i][j]` が 0 なら *i* 行 *j* 列には何も置かれていない、1 なら黒、2 なら白丸を描くようにしよう。



#### (ii) マウス操作

- ・ マウスがクリックされたとき (`mousePressed()`)、そのマスにオセロ駒を置けるようにする。
- ・ 今、白と黒どちらのターンかを表す変数 `turn` を用意し、その色の玉が置かれるようにする。交互に手番がまわるようにする。

#### (iii) ゲームの実装 (難)

- ・ 白に挟まれた黒玉、黒に挟まれた白玉がひっくり返るようにする。例えば今おいたのが黒玉で、その右側がひっくり返るか調べるとき、
  - 今おいた黒玉の右側を、白玉でなくなるまで右に進んでいく
  - 盤面の外に出たらなら、ひっくり返す玉はない
  - 空白のマスにでたら、ひっくり返す玉はない
  - 黒玉にでたら、今通ってきた白玉を全部黒にひっくり返すのように処理する。これを白黒両方について、縦横斜め 8 方向について行う。
- ・ 何もひっくり返らないようなマスには置けないようにする。
- ・ パスができるようにする (何もひっくり返せないときのため)

## 1-2. 文法について

### (1) プログラムの構造

#### 構文 プログラムの構造

定数: 100 0.12 -3 "hello"  
 変数: x hensu a[3]  
 命令: `println("hello")` `sin(1.0)`  
 演算子: + - \* / % < == || =

式: 値を持つもの。上の要素の組み合わせでできる。

`1 + 2` `sin(x)*cos(x)` `a==1&&b==1`

文: セミコロン (;) で区切られる 1 単位。式を含む。命令の実行、代入文、変数定義、return 文など。

`int a = sin(x)*cos(x);` `rect(random(0, 100), random(0, 100), w, h);`

ブロック: 中括弧 {} に囲まれた部分。複数の文から成る。if, for, 命令定義, class 定義など。

```
if(a==1){
    a = 2;
    b++;
}
```

プログラムは次の要素から成り立っている。

コメントを使い、インデントに注意するとプログラムが読みやすくなる。

#### 構文 コメント

// ... 一行コメント

/\*  
 ...  
 \*/ 複数行コメント

... の部分に書いたことはプログラムに含まれない  
 メモや、一時的にその部分を実行させないようにしたいときに使う

#### インデント

内側のブロックの中身は、先頭に空白を入れて字下げをする

Tab キー あるいは 空白 2 個

```
void draw(){
    background(255, 255, 255);
    er.idou();
    te.idou();
    tp.idou();
    pl.idou();
    for(int i=0; i<en.n; i++){
        for(int j=tp.n-1; j>=0; j--){
            if(isCollided(en.get(i), tp.get(j))){
                Enemy e = (Enemy)en.get(i);
                e.hp--;
                tp.sakujo(j);
            }
        }
    }
}
```

## (2) データ型と演算子

プログラム中で扱えるデータの種類には次のようなものがある。

### データ型

#### 基本型

**boolean** : true, falseのみ  
**int** : -2147483648 ~ 2147483647 の整数  
**float** : 小数 (有効数字6桁ほど)  
**double** : 小数 (有効数字15桁ほど)  
**char** : 文字 ('a' 'A' '1'など) シングルコーテーション「'」でくる

#### コンポジット

**String** : 文字列 ("Hello"など) ダブルコーテーション「"」でくる  
配列 : **int**[], **float**[] など。その型の変数の集まり  
オブジェクト : **class**ブロックで定義される、データと命令を組み合わせた型

データに対して計算を行うには演算子を使う。Int,int→int というのは、int のデータ同士を足し算すると結果も int になるという意味である。

### 演算子

計算: int,int→int / int,float→float / float,float→float  
+ - \* / %  
比較: int,int / int,float / float,float → boolean  
== != < <= > >=  
論理: boolean,boolean → boolean  
&& ||  
代入: int→int / int→float / float→float  
= += -= \*= /= %=  
加減: int / float  
++ --  
文字列の連結: String,String / String,int / String,float → String  
+

型同士の変換を行うときは、次の方法を使う。

### 型の変換 (キャスト)

#### (型名)(変換元)

例: float型の変数xをint型に変換 (切り捨て) してyに代入  
y = (int)x;  
例: int型の変数a, bの割り算を小数として計算  
(float)a/b

### 1-3. 命令づくり

ここでは自分で命令を作る方法を学ぶ。

その前に、今まで使ってきた命令について考えてみる。

- ・ 命令とは

構文 命令

呼び出し方

戻り値のないとき：一行で書く  
`命令名(引数1, 引数2, ... );`

戻り値のあるとき：式の中や他の命令の引数として使う  
`x = 命令名(引数1, 引数2, ... );`

例

- ・ 引数も戻り値もない命令  
`noStroke();`  
`noFill();`  
`println();`
- ・ 引数があって戻り値のない命令  
`frameRate(30);`  
`size(300, 300);`  
`rect(0, 100, 200, 300);`  
`println("hello world");`
- ・ 引数も戻り値もある命令  
`x = random(0, 1);`  
`x = sin(PI/2) + cos(PI/2);`

命令名の後にカッコが書かれているものが命令である。

命令によっては、「引数」によって挙動を変えることができる。例えば、rect 命令なら、引数で四角形を描く位置を変えられる。また、random や sin, cos といった命令は「戻り値」を持ち、数として使うことができる。



・ 命令の作り方

構文 命令を作る

```
void hoge(){
  ...
}
```

作る命令の名前

引数なし、戻り値なし  
voidは戻り値なしを表す

---

```
void hoge(int a, int b, ... ){
  ...
}
```

引数の型  
引数名

引数あり、戻り値なし  
小カッコのなかに引数を列挙する

---

```
int hoge(){
  ...
  return ...;
}
```

戻り値の型

引数なし、戻り値あり  
returnの後ろに戻り値を書く

---

```
int hoge(int a, int b, ... ){
  ...
  return ...;
}
```

引数あり、戻り値あり

命令の名前は、上の説明では hoge とつけているが、英語で始まっていて、英字、数字を組み合わせで作られた名前であり、すでに使われている名前（println, if, int など）でなければなんでも良い。

命令を作るときは、引数と戻り値に気をつけないといけない。上の例では引数も戻り値も int 型だが、float や配列などでも良い。ただし、基本型とコンポジットでは扱いが違う。これについては後でまた話す。

ところで setup, draw, mousePressed などと同じ書き方をしていた。実はこれらは特別な命令名で、この名前の命令は setup ならアニメーションの最初に一回実行、などと機能が決まっている。

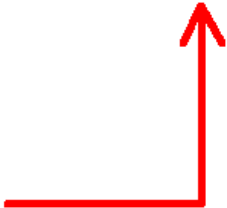
### (1) 引数も戻り値もない命令

次のプログラムを書こう。

```
void random_circle(){
    ellipse(random(0, width), random(0, height), 10, 10);
}

void setup(){
    size(100, 100);
    frameRate(30);
}

void draw(){
    random_circle();
}
```



このプログラムは random\_circle という名前の命令を作って、draw 命令の中でこれ呼び出している。命令の中では、2 行めに書かれているように画面内のランダムな位置に円を描く。

このプログラムを改良して、ランダムな位置に三角形を描く命令を作って、それを draw 命令の中で呼び出せ。

### (2) 引数のある命令

次のプログラムを書こう。

```
void num_hello(int n){
    for(int i=0; i<n; i++){
        println("Hello world! " + i);
    }
}

void setup(){
    size(100, 100);
    noLoop();
}

void draw(){
    num_hello(5);
}
```

9 行目の noLoop(); は、アニメーションしない、という指令である。Draw 命令は setup 命令に続いて 1 回だけ実行されるようになる。

num\_hello 命令は、引数として一つの整数をとる。draw 命令の中で、引数として 5 を入れているので、num\_hello 命令の中で、引数変数 n には 5 が入った状態で num\_hello が呼び出される。

二つの引数 x, y をとり、その位置に三角形を描く命令を作れ。

### (3) 引数と戻り値のある命令

次のプログラムを書こう。

```
int mx(int a, int b){
    if(a>b){
        return a;
    }else{
        return b;
    }
}

void setup(){
    size(100, 100);
    noLoop();
}

void draw(){
    println(mx(3, 5));
}
```

戻り値があるときは、return の後ろに返す数を書く。1 を返すなら return 1;変数 a を返すなら return a;と書く。戻り値の型は、命令定義の最初に書いた型（上の mx なら、int mx( ... と書いているので int 型）と一致していないといけない。

return 文が書かれると、その時点でその命令は終了される。ちなみに、戻り値のない命令では return;と書くと、そこで命令を終了させることができる。

この命令 mx は、2 つの引数 a と b をとって、a が b より大きかったら a を、そうでなかったら b を戻り値として返す。だから、この命令の結果は引数として与えた 2 つの数の大きい方となる。

これを参考に、2 つの引数 a, b をとって、a の b 乗を返すような命令を作れ。

### (4) 命令の中で他の命令を呼び出す

自作命令の中で、他の自作命令を呼び出すことももちろん可能である。

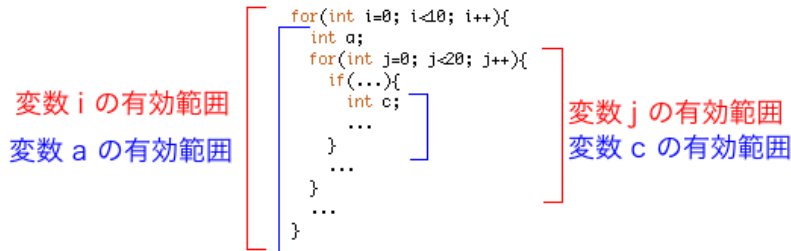
```
int mx3(int a, int b, int c){
    return mx(a, mx(b, c));
}
```

この命令を上記の mx を作ったプログラムに追加してみよ。これは 3 つの引数の中で一番大きい数を返すプログラムになる。なぜそうなるのか考えて、確かめてみよ。

理解したら、4 つの引数を取って、そのなかから一番大きい数を返す命令を作れ。

## (5) グローバル変数の操作

変数には有効範囲があって、ブロック（中括弧）の中で作った変数は、そのブロック内ではしか使えないという決まりがある。



逆に、すべてのブロックの外側（setup とか draw とかその他の命令の外側）で作った変数は、全ての命令で使うことができる。これを「グローバル変数」という。

右のプログラムを書こう。

このプログラムは、マウスをクリックした位置に円を追加していくプログラムだ。グローバル変数に  $n$  と配列  $x, y$  があり、これらには次のような役割を担わせている。

$n$  : 円の個数

$x[i], y[i]$  :  $i$  番目の円の位置

マウスが押されると、自作命令 `add_circle` が呼び出される。この命令では、グローバル変数を操作して、配列  $x, y$  の末尾に新しい円の位置（その時点でのマウスの位置）を記録して、円の個数を 1 加算する。draw 命令では、 $x, y$  に記録された  $n$  個の円を実際に描画している。

これに新たな自作命令 `idou` を付け加えて、そこには円を動かす（ $x[i], y[i]$  に適当な値を足す）文を書こう。これを draw の中で呼び出すことで、 $n$  個の円が動きまわるアニメーションが描ける。

```
int n;
int[] x, y;

void add_circle(){
    x[n] = mouseX;
    y[n] = mouseY;
    n++;
}

void setup(){
    size(100, 100);
    frameRate(30);

    n = 0;
    x = new int[1000];
    y = new int[1000];
}

void draw(){
    background(255, 255, 255);
    for(int i=0; i<n; i++){
        ellipse(x[i], y[i], 10, 10);
    }
}

void mousePressed(){
    add_circle();
}
```

## (6) 変数を渡した時の挙動

実際に書かなくてもいいが、右のプログラムを実行すると何が起ころうか？

println(a);が2つあるので、2回変数 a の中身が出力されるが、この2つの間には自作命令 hoge が挟まっている。

自作命令 hoge の中では引数 a に 10 を代入しているので、一個目の println(a)では 0, 二回目では 10 が出力されそうだが、実はどちらも 0 が出力される。

```
void hoge(int a){
    a = 10;
}

void setup(){
    noLoop();
}

void draw(){
    int a = 0;
    println(a);
    hoge(a);
    println(a);
}
```

つまり、draw 命令の中の変数 a と、命令 hoge の引数変数 a は別物である。命令 hoge が呼び出されると、引数変数 a が新たに作られ、draw の中の変数 a が引数変数 a にコピーされる。

## (7) 配列渡し

次のプログラムを書こう。これは配列を引数として渡す方法を示している。

```
void hoge(int[] a){
    a[1] = 10;
}

void setup(){
    noLoop();
}

void draw(){
    int[] a = {0, 0, 0};
    println(a[0] + " " + a[1] + " " + a[2]);
    hoge(a);
    println(a[0] + " " + a[1] + " " + a[2]);
}
```

次の点に注目しよう。

- ・ int[] a = {0, 0, 0};というのは、次の略記である。  
int[] a = new int[3];  
a[0]=0; a[1]=0; a[2]=0;
- ・ 自作命令の引数としては int[] a と書けば良い。
- ・ 呼び出すときは配列名を引数として渡す (hoge(a)) 。
- ・ これを実行すると、0 0 0, 0 10 0 となる。(6) では変数を渡すとコピーが作られると書いたが、配列を渡すときはコピーは作られない。だから a[1]は hoge によって 10 にされてしまう。混乱しやすいので注意しよう。

