K 会 2012 年度夏期講習　情報講座

中級編 Part1　Processing の基本

目次

1-5. プログラムの構造（お話）

プログラムは次の要素から成り立っている。

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++;
    }
```

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))==1){
        Enemy e = (Enemy)en.get(i);
        e.hp--;
        tp.sakujo(j);
      }
    }
  }
}
```

1-6. データ型と演算子（お話）

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
```

1-7. 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");
```

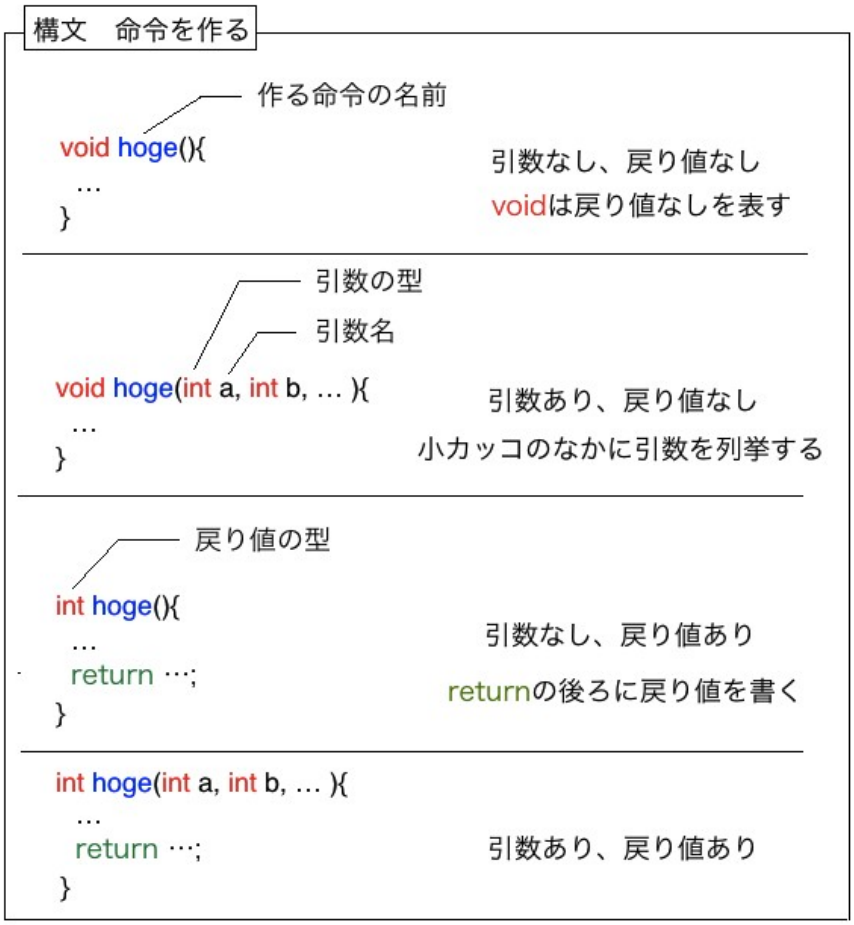
・引数も戻り値もある命令

```
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



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、数字を組み合わせて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 などで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 命令の中でこれを呼び出している。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。

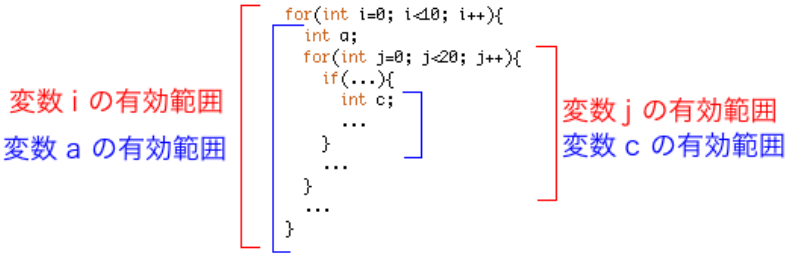
（４）命令の中で他の命令を呼び出す
自作命令の中で、他の自作命令を呼び出すこともちろん可能である。

```
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るだろうか？

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、０００, ０１００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。