情報メディア基盤ユニット用資料（2014 年 5 月 13 日分）

Processing 言語による情報メディア入門

繰り返し処理その 1（for 文、繰り返し回数指定型）

神奈川工科大学情報メディア学科　佐藤尚

乱数

ゲームなどを作成する際には、敵キャラの出現位置をデタラメに決めたいことがあります。これを実現するためには、敵キャラの出現位置を決める座標値をデタラメに設定すれば可能です。ゲームなどを作る際には、デタラメな値が必要となることがあります。これを実現するために、乱数という仕組みが Processing 言語などのプログラミング言語では用意されています。Processing 言語の場合には、random 関数を使用します。この関数は呼び出される度に、デタラメな数値を返します。

真面目な科学技術計算や金融関係の計算でも、乱数は利用されます。そのため、乱数を作り出す方法について、沢山の方法が知られています。

random の意味はわかりますか？

乱数を返す random 関数

使い方	意味
random(high)	0 以上 high 未満の乱数を返す
random(low,high)	low 以上 high 未満の乱数を返す

サンプル 4-1 は random 関数を使って、円を描く場所を決めているので、実行する度に異なった場所に円が描かれます。

乱数を使ったサンプル 4-1

```
size(400,200);
smooth();
// デタラメな場所に円を描く
ellipse(random(width),random(height),20,20);
```

random(1) とすると、0 以上 1 未満のデタラメな数が返されます。このため、random 関数では float 型の値が返されます。つまり、次のプログラムはエラーとなってしまいます。

このような値を戻り値 (return value) と呼びます。

乱数を使ったサンプル 4-2

```
size(400,200);
smooth();
// デタラメな場所に円を描く
int x = random(width);
int y = random(height);
ellipse(x,y,20,20);
```

エラーメッセージは「cannot convert float to int」となっているはずです。これは、float 型の値を int 型の値には変更出来ない (cannot convert) という意味です。

そこで、整数の乱数値 (int 型の乱数値) が必要となる場合には、「int(random(10))」などとします。この「int(…)」は、強制的に整数値 (int 型) の値に変更する関数です。

乱数を使ったサンプル 4-3

```
size(400,200);
smooth();
// デタラメな場所に円を描く
int x = int(random(width)); //random(width) の値を int 型に変換
int y = int(random(height));//random(height) の値を int 型に変換
ellipse(x,y,20,20);
```

このように強制的にデータ型を変える関数としては、以下のようなものがよく使われます。

データ型を変える関数

使い方	意味
int(value)	値 value を強制的に int 型の値に変更する
float(value)	値 value を強制的に float 型の値に変更する
str(value)	値 value を強制的に String 型に変更する

強制的にデータの型を変更することを明示的型変換（キャスト変換）と呼びます。一方、データ格納領域がより広い型への変換は自動的に行われます。これを、暗黙的な型変換と呼んでいます。char < int <float の順にデータ格納領域が広くなっているので、char 型の値を int 型や float 型の変数に代入するや int 型の値を float 型の変数に代入するなどの処理においては、暗黙の型変換が行われます。

最後に、もう一つ乱数を利用したサンプルを挙げておきます。

乱数を使ったサンプル 4-4

```
void setup(){
  size(400,200); // 400X200 のウインドウを表示
  smooth();      // アンチエリアシングをして図形を描画
  background(0); // 背景を黒で塗りつぶす
}

void draw(){
  fill(random(100,256));  // 塗りつぶし色を乱数で設定
   ellipse(random(width),random(height),20,20);  // 直径 20 の円を乱数で決めた場所に表示
}
```

関数

今までの説明の中では、random や ellipse などを命令文や関数と呼んできました。ellipse 命令を使って楕円を描くためには、コンピュータ内部では沢山の処理が行われています。多くのプログラミング言語では、まとまった処理に名前をつける機能が用意されています。Processing 言語では、このような仕組みのことを関数 (function) もしくはメソッド (method) と呼んでいます。random や ellipse のように Processing 言語がどのような処理を行うべきかを最初から知っているものは、組み込み関数 (built-in function) や組み込みメソッド (built-in method) と呼ばれます。Processing には、組み込み関数が用意されています。

関数やメソッドには、関数名やメソッド名と呼ばれる名前がついています。関数名やメソッド名として使える名前の付け方は、変数名の付け方と同じです。

関数とメソッドの違いは別な機会に紹介します。今まで出てきた「命令文」は、関数と呼ばれるものです。

ですから正確には、random 関数や elipse 関数と呼ぶべきものです。

関数名や変数名のことを識別子 (identifier) と呼びます。

正確には、この引数を実引数と呼びます。ということは、仮引数 (parameter) と呼ばれるものもあります。別の機会に説明します。

256 などのように数値を表した表現をリテラル (literal) または数値リテラルと呼びます。

関数が知っている処理内容を実行させることを、関数を呼び出す (call) と言うことがあります。関数を呼び出す場合には、何らかの付加的な情報をつけて呼び出す場合と、付加的な情報をつけることなく呼び出す場合の 2 通りがあります。この付加的な情報を引数 (argument) と呼びます。関数を呼び出す場合には、次のように記述します。

通常、引数の部分には値が置かれます。値 (value) とは、数値、文字列、true や false などのことです。簡単に言ってしまうと、Processing のプログラム中で扱う全てのデータが値です。また、式 (expression) は計算することで値を得ることの出来るものです。簡単に言うと、引数には数値、式、変数名などが使われます。「random(10)」なども値となります。

数学で出てくるような式と if 文などで使用される条件式も式です。

関数呼び出し

引数の数	関数の呼び出し方	例
引数が無い場合	関数名 ()	smooth()
引数が 1 つの場合	関数名 (引数 1)	random(width/2)
引数が 2 つの場合	関数名 (引数 1, 引数 2)	size(400,400)
引数が 3 つの場合	関数名 (引数 1, 引数 2, 引数 3)	fill(10,20,30)
全てまとめて書くと	関数名 ([引数 1[, 引数 2…[, 引数 3…]])	

「全てのまとめて書くと」の [と] は実際に書くわけではありません。カギ括弧 [ ] は省略可能を示す、コンピュータ業界では、一般的な書き方です。

関数を利用しなくても、原理的にはプログラムを作成することが出来ます。それでは、なぜ関数を利用するのでしょうか？大雑把に言うと、2 つの理由（可読性の向上、再利用）があります。

長い行数のプログラムを作ると、全体の処理の流れを理解することが難しくなっていきます。しかし、処理の塊が一定の意味をもった処理を行っていることがあります。そこで、この意味をもった処理の塊の部分を取り出して、名前をつけます。これが関数です。

関数を使ってプログラムを作ると言うことは、部品を組み合わせて、ものを作ることに似ています。料理などでも、以前は自分で野菜を切ったりして下準備をするのが当たり前でしたが、最近ではカット野菜を利用することが多くなってきています。また、プログラムの中で、同じような処理が繰り返し行われていることがあります。他の人が作成した関数を利用したり、自分で処理の塊に名前をつけて関数を作ったりして、その関数を呼び出すことで、プログラムを作って行くことができます。また、ある処理の塊に名前をつけているので、ある不都合が起きたときには、まずはその不具合を起こしそうな関数を調べるというデバッグ作業を開始することが出来ます。CD プレイやで再生が上手く行かないときには、CD が汚れていないか、ピックアップが汚れていないかなど、チェックをして行くことができます。もし、CD と CD プレイーが一体のものとして作られていたら、こんなことは出来ません。

ガルパンで、大洗女子学園のチームのメンバの名前を全て覚えることは難しいです。でも、乗っている戦車のチーム毎に把握すれば、何とかなりますよね。指揮をしているときに、いちいちメンバの名前を呼んでいられません。ウサギさんチームのメンバ全員の名前を呼んでいられません。車長の澤さんに命令を出せが、彼女が他のウサギさんチームのメンバに必要な指示を出してくれます。

回数指定型繰り返し処理（その 1）

Processing 言語をはじめとして、多くのプログラミング言語では、命令の実行に関しては、以下の 3 つものがあります。

1. 逐次処理
2. 条件分岐処理
3. 繰り返し処理

前回の授業では条件分岐処理を紹介しました。今回は繰り返し処理を紹介します。繰り返し処理は少ないプログラムの記述量で沢山の命令を実行させようとするものです。

次のサンプルプログラムは乱数を使ってデタラメな場所に 10 個の円を表示するものです。ちょっと長いので 3 列に分けてプログラムを書いてあります。

円を 10 個描く サンプル 4-5

```
float x,y;
size(400,200);
smooth();
background(150);
fill(255);

x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);
// 隣列上に続く
```

```
x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);
// 隣列上に続く
```

```
x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);

x = random(width);
y = random(height);
ellipse(x,y,20,20);
```

random 関数を使っているので、実行する度に描かれる画像が変化します。

このサンプル 4-5 では 10 個の円を表示するために、表示する円の中心を決める命令（x = random(width); と y=random(height);）と円を表示する命令 (ellipse(x,y,20,20);) の組を 10 回実行しています。このように同じ命令を何回も実行したい場合を考えます。繰り返し回数が少なければ、単純に命令を書いていけば、プログラムを作ることが出来ます。しかし、その回数が多ければ、この方法でプログラムを作ることは困難になります。例えば、このプログラムの円の表示個数を 100 個や 1000 個にする場合を考えれば、想像できると思います。

繰り返し処理のことをループ (loop) 処理と呼ぶこともあります。

そこで、同じ命令を何度も繰り返し実行した場合に使われるのが、繰り返し処理です。多くのプログラミング言語では、繰り返し処理を行うために、2 つの方法が用意されています。それは、

1. 繰り返し回数指定型
2. 繰り返し条件指定型

「繰り返し回数」が「繰り返し条件」と見なせば、、繰り返し回数指定型と繰り返し条件指定型は同じだと考えることも出来ます。

です。一般的は、回数指定型繰り返し処理には for 命令、条件指定型繰り返し処理には while 命令を使用します。

サンプル 4-6 はサンプル 4-5 を for 命令を利用するように書き換えた者です。

for 命令を利用した円を 10 個描く サンプル 4-6

```
float x,y; // 変数 x,y は円の中心座標を表す float 型の変数

size(400,200); // 400X200 のウインドウを表示
smooth();      // アンチエリアシングをして図形を描画
background(150); // 背景を灰色で塗りつぶす
fill(255);     // 図形の塗りつぶし色を白色に設定

for(int i=0;i<10;i++){// カウンタ変数 i を 0 ～ 9 で変化させながら
  x = random(width); // 円を表示する x 座標を乱数で決定
  y = random(height);// 円を表示する y 座標を乱数で決定
  ellipse(x,y,20,20);// (x,y) を中心として直径 20 の円を描画
}
```

サンプル 4-6 の先頭部分が、「float x,y;」となっています。これは、同時に複数の変数を宣言する方法です。同じデータ型の変数を複数宣言する場合に便利な方法です。つまり、変数宣言の形式は以下のようになります。

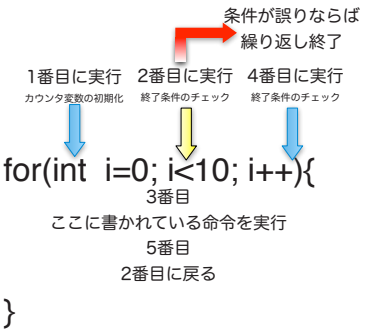
変数宣言の形式
データ型　変数名 ;
データ型　変数名 1, 変数名 2,…;
データ型　変数名 1[, 変数名 2[,…]]

ここからが、本題です。サンプル 4-6 の「for(int i=0;i<10;i++){」から次の「}」の部分が繰り返し処理を行う部分になっています。「for(int i=0;i<10;i++){」部分の 10 によって繰り返し回数を指定してます。繰り返し処理を行う場合には、何回目の繰り返しかを知りたい場合があるので、繰り返し回数を記憶させるための変数を用意します。この変数のことをカウンタ変数と呼ぶことがあります。サンプル 4-6 では、カウンタ変数として i を使っています。「int i=0」の部分でカウンタ変数を指定します。カウンタ変数の変数名には特に制約はありません。

for 命令の括弧 () 内は、”;” で区切られた 3 つ部分から構成されています。1 つ目は、「int i=0」の部分で、カウンタ変数の宣言とカウンタ変数の値を 0 に設定しています。2 つ目は、「i<10」の部分で、繰り返し回数のチェックをしている部分です。この場合には、10 回繰り返し処理を行いたいので、「i<10」となっています。3 つ目は、「i++」の部分で、カウンタ変数の値を 1 増やしています。

for 命令の使い方は、まとめると次の様になります。

ここで述べる、繰り返し処理の表し方は、Processing 言語だけでなく、C 言語系の言語ではほぼ共通の書き方（構文）になっています。
for 命令が非常に強力なので、条件指定型繰り返し処理も for 命令で書かれる場合が多くあります。



「for(int i=0;…){ ～ }」となってい部分の、中括弧 { と } で作っているブロックの部分が繰り返し実行されます。

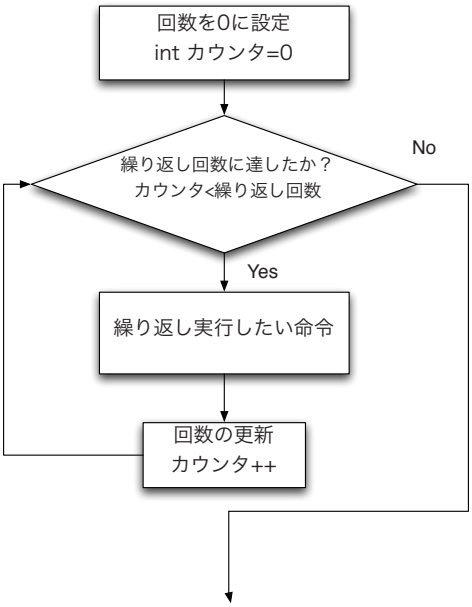
カウンタ変数も普通の変数と代わりないので、カウンタ変数名としては、普通の変数名として利用できるものであれば、何でも OK です。
通常は、i,j,k などのシンプルな名称を利用することが多いようです。この習慣は、FORTRAN 言語と呼ばれる、非常に古いプログラミング言語からの影響だと思います。初期の FORTRAN では、I ～ N で始まる変数名の変数には整数型変数と見なすルールがありました。

for 命令のシンプルな使い方

Processing 言語での記法

```
for(int カウンタ名 =0; カウンタ名 < 繰り返し回数 ; カウンタ名 ++){
  繰り返し実行したい命令
}
```

処理の流れ



カウンタ変数の初期値を 0 としているので、繰り返し回数を指定している部分では、比較に「<」を使用しています。

サンプル 4-6 の繰り返し回数を 100 回に変更したものがサンプル 4-7 です。このように簡単に繰り返し回数を変更することができます。

カウンタ変数の値を更新 (1 増やす) するために、増分演算子 (++) を使用しています。増分演算子に関しては、前回授業資料を参考にして下さい。

for 命令を利用した円を 100 個描く サンプル 4-7

```
float x,y; // 変数 x,y は円の中心座標を表す float 型の変数

size(400,200); // 400X200 のウインドウを表示
smooth();       // アンチエリアシングをして図形を描画
background(150); // 背景を灰色で塗りつぶす
fill(255);     // 図形の塗りつぶし色を白色に設定

for(int i=0;i<100;i++){// カウンタ変数 i を 0 ～ 99 で変化させながら
  x = random(width); // 円を表示する x 座標を乱数で決定
  y = random(height);// 円を表示する y 座標を乱数で決定
  ellipse(x,y,20,20);// (x,y) を中心として直径 20 の円を描画
}
```

変更部分は赤色になっています。

サンプル 4-8 は、カウンタ変数の値を表示するものです。

カウンタ変数の値を表示する サンプル 4-8

```
// カウンタ変数は loop
for(int loop=0;loop < 20;loop++){//loop を 0 ～ 19 まで変化させながら
  println(loop); // 変数 loop の値を表示
}
```

19 は「繰り返し回数 -1」です。0 から繰り返し回数を数えているので、「繰り返し回数 -1」回まで繰り返せば、ちょうど繰り返し回数だけ「{ ～ }」の部分を実行できます。

このサンプル 4-8 を実行すればわかるように、カウンタ変数の値は、0 から始まって 19 まで 1 ずつ増えながら、「{ ～ }」の部分を実行していきます。

繰り返し処理でのカウンタ変数の利用

単純に同じ処理を繰り返すだけの繰り返し処理では、あまり利用する機会がありません。例えば、サンプル 4-9 では line 関数を 11 回実行しています。しかし、引数の値が異なっているため、全く同じではありません。従って、単純な繰り返し処理では扱うことが出来ません。

line 関数を 11 回実行する サンプル 4-9

```
size(300,200);          //300X200 のウインドウを表示
background(255);        // 背景を白色で塗りつぶす
stroke(0);              // 線分の描画色を黒色に設定

line(25,20,25,180);     // 線分を描画
line(50,20,50,180);     // 線分を描画
line(75,20,75,180);     // 線分を描画
line(100,20,100,180);   // 線分を描画
line(125,20,125,180);   // 線分を描画
line(150,20,150,180);   // 線分を描画
line(175,20,175,180);   // 線分を描画
line(200,20,200,180);   // 線分を描画
line(225,20,225,180);   // 線分を描画
line(250,20,250,180);   // 線分を描画
line(275,20,275,180);   // 線分を描画
```

このサンプル 4-9 では、同じ命令を繰り返し実行している訳でないので、for を利用した繰り返し処理に書き換えることが出来ないように見えます。そこで、このプログラムにおける line による線分を描画する位置を、次のプログラムのように書き換えてみます。一見するとトリッキーな書き換えのように見えます。しかし、描画する線分の両端の x 座標の値を指定している引数の値は、25、50、75…のように、25 からから始まり、ちょうど 25 ずつ増加しています。これは、中学生の時に学習した一次関数となっています。そこで、一次関数の式y=ax+bを思い出してもらえば、この書き換えがトリッキーな書き換えでないことが理解できると思います。

line 関数を 11 回実行する サンプル 4-10

```
size(300,200);     //300X200 のウインドウを表示
background(255);   // 背景を白色で塗りつぶす
stroke(0);         // 線分の描画色を黒色に設定
```

```
line( 0*25+25,20,  0*25+25,180);  // 線分を描画
line( 1*25+25,20,  1*25+25,180);  // 線分を描画
line( 2*25+25,20,  2*25+25,180);  // 線分を描画
line( 3*25+25,20,  3*25+25,180);  // 線分を描画
line( 4*25+25,20,  4*25+25,180);  // 線分を描画
line( 5*25+25,20,  5*25+25,180);  // 線分を描画
line( 6*25+25,20,  6*25+25,180);  // 線分を描画
line( 7*25+25,20,  7*25+25,180);  // 線分を描画
line( 8*25+25,20,  8*25+25,180);  // 線分を描画
line( 9*25+25,20,  9*25+25,180);  // 線分を描画
line(10*25+25,20, 10*25+25,180);  // 線分を描画
```

このように変更すると、共通部分の構造が見えてくると思います。赤字の部分は異なっていますが、それ以外の部分は共通になっています。赤字となっている数字の部分は、0 から 1 ずつ増加しています。カウンタ変数の値が 0 から 1,2... と 1 ずつ増加していくのと同じように変化しています。つまり、繰り返し部分を書く場所で、カウンタ変数に記録されている値を利用することで、for を利用した繰り返し処理のプログラムに書き換えることが出来ます。それを行ったものがサンプル 4-11 です。

カウンタ変数の値を利用その 1 サンプル 4-11

```
size(300,200);     //300X200 のウインドウを表示
background(255);   // 背景を白色で塗りつぶす
stroke(0);         // 線分の描画色を黒色に設定
for(int i=0;i<11;i++){ // カウンタ変数 i の値を 0 ～ 10 まで変えながら
  // 2 点 (i*25+25,20),(i*25+25,180) の間に線分を描画する
  line(i*25+25,20, i*25+25,180);
}
```

「i*25+25」の部分は、中学生の時に学習した一次関数の計算となっています。

サンプル 4-11 と同じように、カウンタ変数の値を利用した繰り返し処理のサンプルを示します。このサンプルでは、カウンタ変数の値を利用して、長方形を描く位置を決定しています。

カウンタ変数の値を利用その 2 サンプル 4-12

```
size(200,405);     //200X405 のウインドウを表示
background(255);   // 背景を白色で塗りつぶす
fill(170);         // 背景を白色で塗りつぶす
for(int j=0;j<10;j++){ // カウンタ変数 j の値を 0 ～ 9 まで変えながら
  // 点 (30,10+40*j) を頂点とする横 140、縦 20 の矩形を表示する
  rect(30,10+40*j,140,20);
}
```

「10+40*j」で y 座標の値を決めているので、10,50,90 のように、10 から 40 ずつ増加しながら値が変化しています。

サンプル 4-12 では、for 命令の実行が始まり、

1. 最初にカウンタ変数 j の値が 0 になり (int j=0)、
2. カウンタ変数 j の値（今は 0）は繰り返し回数よりも少ないので、

矩形を描く命令が実行されます、

3. 次にカウンタ変数 j の値が 1 増やされます (j++)、
4. カウンタ変数 j の値（今は 1）は繰り返し回数よりも少ないので、矩形を描く命令が実行されます、
5. 次にカウンタ変数 j の値が 1 増やされます (j++)、
6. **中略**
7. カウンタ変数 j の値（今は 9）は繰り返し回数よりも少ないので、矩形を描く命令が実行されます、
8. 次にカウンタ変数の値が 1 増やされます (j++)、
9. カウンタ変数 j の値（今は 10）は繰り返し回数よりも小さくないので、for 命令による繰り返し処理は終了します。

「繰り返し回数よりも小さくない」とは、「j<10」が false になることを意味しています。

矩形を描く際には、カウンタ変数の値を利用して、矩形の描画位置 (30,10+40*j) を決めています。

サンプル 4-13 では、カウンタ変数を利用して長方形の描画位置を決めるだけでなく、塗りつぶし色の変更 (fill(10+20*j);) もカウンタ変数の値を利用して行っています。

塗りつぶし色は (10,10,10), (30,30,30), (50,50,50)…のように、変化してきます。最後の色は RGB で表すとどんな色になるでしょうか？

カウンタ変数の値を利用その 3 サンプル 4-13

```
size(200,405);    //200X405 のウインドウを表示
background(255);  // 背景を白色で塗りつぶす
for(int j=0;j<10;j++){
  fill(10+20*j);  // 塗りつぶし色を (10+20*j,10+20*j,10+20*j) に変更
  // 点 (30,10+40*j) を頂点とする横 140、縦 20 の矩形を表示する
  rect(30,10+40*j,140,20);
}
```

サンプル 4-14 と 4-15 は、for 命令の繰り返し処理を行う部分で、新たな変数を利用する例（int x=10+8*i;; と int x=20+20*i;）となっています。

カウンタ変数の値を利用その 4 サンプル 4-14

```
size(400,200);    //400X200 のウインドウを表示
smooth();         // アンチエリアシングをかけながら図形を描画
strokeWeight(2); // 線分の太さを 2 に変更
for(int i=0;i<40;i++){//カウンタ変数の値を 0 ～ 39 まで変化させながら、
  int x=10+8*i;       // int 型の変数 x を宣言し、値 10+8*i を代入
  line(x,40,x+60,160);// 2 点 (x,40)、(x+60,160) を結ぶ線分を描画
}
```

変数宣言の仕方は同じです。プログラムの先頭で変数を宣言するとの違いは、変数の有効範囲の違いです。これに関しては、別の機会に説明します。

カウンタ変数の値を利用その 5 サンプル 4-15

```
size(400,200);    //400X200 のウインドウを表示
smooth();         // アンチエリアシングをかけながら図形を描画
strokeWeight(2);  // 線分の太さを 2 に変更
```

ついでながら、プログラム中（除く、コメント文中）に全角の空白があると、「unexpected char :''」のようなエラーとなります。気をつけて下さい。

```
for(int i=0;i<20;i++){//カウンタ変数の値を0～19まで変化させながら、
  int x=20+20*i;    // int 型の変数 x を宣言、値 20+20*i を代入
  // 3 点 (x,0)、(x+x/2,120)、(1.2*x,height) を順番に結ぶ線分を描画
  line(x,0,x+x/2,120);
  line(x+x/2,120,1.2*x,height);
}
```

プログラム中には、複数個の繰り返し処理を書くことができます。サンプル 4-16 は複数個の繰り返し処理を記述したものです。

複数個の繰り返し処理その 1 サンプル 4-16

```
size(400,200);
background(0);
smooth();
fill(255);
for(int x=0;x<11;x++){ // この繰り返しでは、変数 x がカウンタ変数
  ellipse(40*x,0,40,40);
}
for(int y=0;y<6;y++){ // この繰り返しでは、変数 y がカウンタ変数
  ellipse(0,40*y,40,40);
}
```

このサンプルでは、2 箇所の for 命令では異なった変数名のカウンタ変数を使用しています。このサンプル 4-16 のように「独立」した繰り返し処理では、同じ変数名のカウンタ変数を利用することが出来ます。同じカウンタ変数名を使用して、書き換えたものがサンプル 4-17 です。

複数個の繰り返し処理その 2 サンプル 4-17

```
size(400,200);
background(0);
smooth();
fill(255);
for(int x=0;x<11;x++){ // この繰り返しでは、変数 x がカウンタ変数
  ellipse(40*x,0,40,40);
}
for(int x=0;x<6;x++){ // この繰り返しでも、変数 x がカウンタ変数
  ellipse(0,40*x,40,40);
}
```

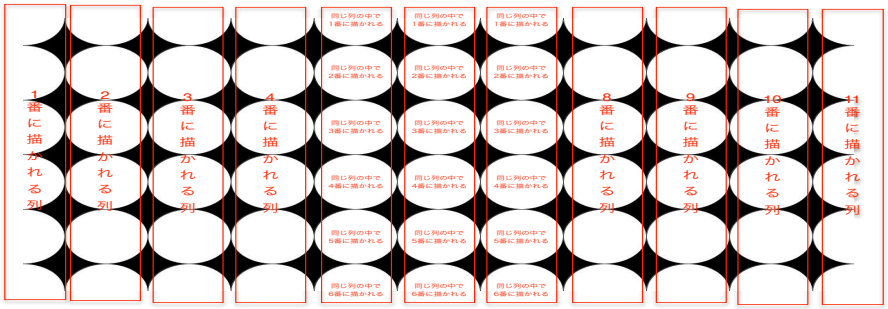
繰り返し処理の入れ子

for 命令の繰り返し処理の中に、また繰り返し処理を入れることが出来ます。for による繰り返し処理を入れ子にする場合には、カウンタ変数名は異なる変数名にする必要があります。

サンプル 4-18 では繰り返し処理を入れ子にしています。このサン

多くの言語学者は、この for 命令の入れ子のように、入れ子になった文が扱えることが人間の言語能力の大きな特徴だと考えています。

プルは円を格子状に配置して表示するプログラムです。カウンタ変数 x の値を利用して表示する円の中心の x 座標の値を決め、カウンタ変数 y の値を利用して表示する円の中心の y 座標の値を決めています。「for(int x=0;x<11;x++)」による繰り返し処理では、カウンタ変数 x の値を 1 ずつ増やしながら、「for(int y=0;y<6;y++)」による繰り返し処理を行っています。「for(int y=0;y<6;y++)」による繰り返し処理では、カウンタ変数の y の値を 1 ずつ増やしながら、円を描く命令(ellipse(40*x,40*y,40,40);) を実行しています。これにより、ellipse 関数は 66 回実行されています。



サンプル 4-18 の実行例

複数個の繰り返し処理その 3 サンプル 4-18

```
size(400,200);
background(0);
smooth();
fill(255);
for(int x=0;x<11;x++){  // 一番外側の繰り返し処理のカウンタ変数は x
  for(int y=0;y<6;y++){ // この繰り返し処理のカウンタ変数 y
    ellipse(40*x,40*y,40,40);
  }
}
```

サンプル 4-19 も繰り返し処理を入れ子にしたものです。何回でも繰り返し処理の入れ子にすることが出来ますが、実際の使用では、サンプル 4-18 や 4-19 のように、2 重の入れ子や 3 重の入れ子が多いように思います。

複数個の繰り返し処理その 4 サンプル 4-19

```
size(400,200);
background(0);
smooth();
fill(255);
stroke(100);
```

for 命令や if 命令などで、処理ブロックをハッキリさせるために、字下げやインデント(indent) と呼ばれることを行います。これは、処理ブロック毎に一律に右方向に移動して、命令を書くことです。インデントを行うことで、プログラムの構造を理解しやすくします。Processing 言語ではインデントをしなくても、エラーとはなりません。

```
for(int j = 0;j < 17;j++){
  int y = 20 + 10*j;
  for(int i = 0;i < 37;i++){
    int x = 20+10*i;
    ellipse(x,y,4,4);
    line(x,y,width/2,height/2);
  }
}
```

Python 言語などでは、インデントを行うことで、処理ブロックを指定します。つまり、適切にインデントを行わないと、エラーとなります。

インデント無しバージョン サンプル 4-19'

```
size(400,200);
background(0);
smooth();
fill(255);
stroke(100);
for(int j = 0;j < 17;j++){
int y = 20 + 10*j;
for(int i = 0;i < 37;i++){
int x = 20+10*i;
ellipse(x,y,4,4);
line(x,y,width/2,height/2);
}
}
```

サンプル 4-19' はサンプル 4-19 をインデントを行わないで書いたものです。プログラムの構造が把握しにくいと思います。

インデントがないと、for による繰り返し処理の範囲がわかりづらくなっていると思います。

繰り返し処理を利用したサンプルをいくつかのせておきます。

複数個の繰り返し処理その 5 サンプル 4-20

```
size(400,400);
background(255);
noStroke();
for(int y = 0;y < 10;y++){
  for(int x = 0; x < 10;x++){
    fill(25*x,25*y,20);
    rect(40*x,40*y,30,30);
  }
}
```

規則的な配置をもった画像を作り出すことが出来ます。また、乱数を使用することで揺らぎをもった画像を作り出すことも出来ます。この辺りが、コンピュータのプログラムを利用して作り出す画像の面白さだと思います。
この辺りの、コンピュータ使った作品に関することは、メディアアートやディジタルデザインで詳しく扱われると思います。

乱数と繰り返し処理の組み合わせ サンプル 4-21

```
size(400,400);
background(255);
for(int k=0;k<100;k++){
  stroke(random(256),random(256),random(256));
  line(random(width),random(height),random(width),random(height));
}
```

カウンタ変数の値を利用その 6 サンプル 4-22

```
size(400,400);
background(255);
stroke(0);
```

サンプル 4-22 を実行すると、曲線が見えてきませんか？このような曲線のことを、包絡線と呼びます。

```
for(int y=0;y < 40;y++){
  int v =10*y; // 何回も同じ値を使うので変数に計算結果を保存
  stroke(255,10,10);
  line(0,v,v,height);
  stroke(10,10,255);
  line(v,0,width,v);
}
```

for 命令の繰り返し処理では、繰り返し処理を行う部分に置いては、カウンタ変数と同じ名前の変数を宣言して、使うことは出来ません。従って、サンプル 4-22 では、int 型の変数名 v という変数を宣言して使っています。

再び四角形の描画

rectMode 関数を使うと、rect 関数を利用して長方形を描くときに、色々な長方形の描画位置指定の方法を選ぶことが出来ます。長方形を描く際の位置指定方法を変更すると、次に rectMode 関数を呼び出して明示的に変更しない限り、位置指定方法は変更されません。

rectMode：rect の座標指定方法を変更する命令

rectMode 関数の呼び出し	rect 命令の引数に与える値の役割	備考
rectMode(CORNER)	rect(左上隅 x 座標 , 左上隅 y 座標 , 幅 , 高さ)	デフォルトの指定方法
rectMode(CENTER)	rect(中心の x 座標 , 中心の y 座標 , 幅 , 高さ)	長方形の中心位置を指定
rectMode(CORNERS)	rect(左上隅 x 座標 , 左上隅 y 座標 , 右下隅 x, 右下隅 y)	長方形の左上と右下の位置を指定

正確に言うと、rectMode(CORNERS) では、長方形の対角線の両端の座標を指定しています。

rectMode(CENTER) での長方形描画　サンプル 4-23

```
void setup(){
  size(400,400);
  rectMode(CENTER);
}
void draw(){
  background(255);
  fill(175);
  rect(mouseX,mouseY,100,60);
  fill(0);
  rect(mouseX-30,mouseY-35,20,10);
  rect(mouseX+30,mouseY-35,20,10);
  rect(mouseX-30,mouseY+35,20,10);
  rect(mouseX+30,mouseY+35,20,10);
}
```

車のつもりなのですが。

rectMode(CENTER) での長方形描画　サンプル 4-24

```
size(400,400);
background(255);
stroke(0);
rectMode(CENTER);
```

```
for(int i=0;i<12;i++){
  rect(width/2,height/2,370-30*i,370-30*i);
}
```

傾きが負の一次関数を利用して、正方形の辺の長さを決めています。

rectMode(CORNERS) での長方形描画　サンプル 4-25

```
void setup(){
  size(400,300);
  rectMode(CORNERS);
  noStroke();
  fill(175);
}
void draw(){
  background(255);
  rect(mouseX,mouseY,width-mouseX,height-mouseY);
}
```

落ち葉拾い：折れ線の描画

以前に配布した資料の「少し複雑な図形を描く」の補足説明です。

多角形を描くために使用される beginShape 関数、endShape 関数、vertex 関数は、noFill 関数と組み合わせて使用することで、折れ線を描くためにも使用できます。endShape 関数の引数に何も値を指定しないで呼び出すと、最初に指定頂点を最後に指定した頂点を結ぶ辺を描画しないで、多角形が描かれます。そこで、noFill 関数を利用して塗りつぶしを行わないような設定にすると、多角形の辺だけを描くようになります。つまり、最初に指定頂点を最後に指定した頂点を結ぶ辺を描画しないで多角形の辺だけを描画するので、折れ線が描くことが出来るようになります。

折れ線の描画方法

1. noFill 関数を呼び出し、塗りつぶしを行わない設定にする
2. beginShape() を実行する
3. vertex 関数で折れ線の始点の位置を指定する
4. vertex 関数で折れ線の途中の点の位置を指定する
5. vertex 関数で折れ線の終点の位置を指定する
6. endShape() を実行する

この方法で折れ線を描画するサンプルを示します。

当然、line 関数を利用して、折れ線を描くことも出来ます。でも、ちょって面倒になります。なぜかわかりますか？

beginShape と endShape による折れ線描画　サンプル 4-26

```
size(400,400);
noFill();  // 塗りつぶしを行わないようにする
stroke(10,10,255);
beginShape(); // 折れ線の頂点位置指定を開始
vertex(0,height/2);  // 始点位置を指定
vertex(width/4,random(height));  // 途中の頂点位置を指定
vertex(width/2,random(height));
vertex(3*width/4,random(height));
vertex(width,height/2);  // 終点位置を指定
endShape(); // 折れ線の頂点位置指定の終了
```

乱数を利用した折れ線描画　サンプル 4-27

```
size(400,400);
noFill();
stroke(255,10,10);
beginShape();         // 折れ線の頂点位置指定の開始
float y = height/2;  // 始点の Y 座標の値は height/2
vertex(0,y);          // 始点位置を指定
for(int x=0;x<40;x++){
  y = y + random(-10,10); // 乱数を使って頂点の Y 座標を変更
  vertex(10*x+10,y);      // 頂点位置を指定
}
endShape();           // 折れ線の頂点位置指定の終了
```

サンプル 4-26 を line 関数を使って折れ線を描画するようにしたものをサンプル 4-28 としてのせておきます。

line 関数による折れ線描画　サンプル 4-28

```
size(400,400);
stroke(10,10,255);
float y0 = height/2;
float y1 = random(height);
line(0,y0,width/4,y1);
y0 = y1;
y1 = random(height);
line(width/4,y0,width/2,y1);
y0 = y1;
y1 = random(height);
line(width/2,y0,3*width/4,y1);
line(3*width/4,y1,width,height/2);
```

一つ前の線分の終点位置を憶えている必要があります。これが、ちょっとプログラムが複雑になる理由だと思います。