情報メディア基盤ユニット用資料（2013 年 5 月 14 日分）

Processing 言語による情報メディア入門

繰り返し処理その 2（while 文）

神奈川工科大学情報メディア学科　佐藤尚

条件指定型繰り返し処理

通常、繰り返し処理には、次の 2 つのパターンがあります。一つは前回説明した回数指定型繰り返し処理です。もう一つは、条件指定型繰り返し処理です。

回数指定型繰り返し処理

条件指定型繰り返し処理

今回は、後者の条件指定型繰り返し処理の説明をします。前回の授業では、サンプル 4-9 の line 関数で X 座標の値を指定している部分を、サンプル 4-10 のように解釈して、for 命令による繰り返し処理を使ったサンプル 4-11 のプログラムを作成しました。

「繰り返し回数」を条件だと思うと、回数指定型繰り返し処理は条件指定型繰り返し処理の特殊な場合と見なすことができます。

line 関数を 11 回実行する サンプル 4-9

```
size(300,200);
background(255);
stroke(0);
line(25,20,25,180);
line(50,20,50,180);
line(75,20,75,180);
line(100,20,100,180);
// 右隣上に続く
```

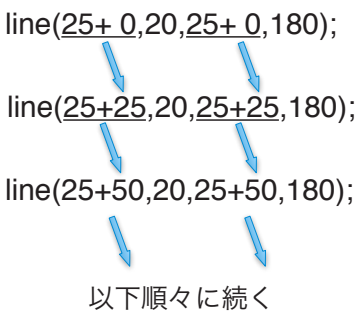
```
line(125,20,125,180);
line(150,20,150,180);
line(175,20,175,180);
line(200,20,200,180);
line(225,20,225,180);
line(250,20,250,180);
line(275,20,275,180);
```

このサンプル 4-9 は、サンプル 5-1 のように書き換えることができます。このままだと、単純な繰り返し処理に書き換えることができません。

line 関数を 11 回実行する サンプル 5-1

```
size(300,200);
background(255);
stroke(0);
line(25+  0,20,25+  0,180);
line(25+ 25,20,25+ 25,180);
line(25+ 50,20,25+ 50,180);
line(25+ 75,20,25+ 75,180);
// 右隣上に続く
```

```
line(25+100,20,25+100,180);
line(25+125,20,25+125,180);
line(25+150,20,25+150,180);
line(25+175,20,25+175,180);
line(25+200,20,25+200,180);
line(25+225,20,25+225,180);
line(25+250,20,25+250,180);
```



このサンプル 5-1 を変数を使うと、次のように書き換えることができます。

変数を使用した line 関数を 11 回実行する サンプル 5-2

```
size(300,200);
background(255);
stroke(0);
int x = 25;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
// 右隣上に続く
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x;
line(x,20,x,180);
x = 25+x
```

このように書き換えると、繰り返しの共通部分がハッキリしてきます。つまり、「line(x,20,x,180);」と「x=x+25;」の 2 行が共通となっています。従って、サンプル 5-3 のように、単純な繰り返し処理を使って、書き換えることが出来ます。

変数と単純な繰り返し命令を利用した書き換え サンプル 5-3

```
size(300,200);
background(255);
stroke(0);
int x = 25;
for(int i=0;i<11;i++){
  line(x,20,x,180);
  x = x+25;
}
```

サンプル 4-9 のプログラムは、単に line 関数を 11 回繰り返し実行したいと言う風にも考えられますが、X 座標の値が 25 から始めて、25 ずつ離れた場所に、X 座標の値が 275 以下の間、繰り返し線分を描くために line 関数を実行するという風にも考えられます。

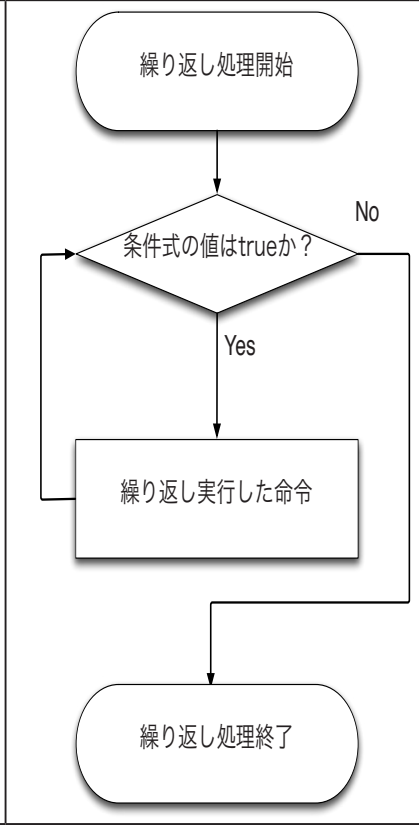
折角、用意した int 型変数 x の出番が少ないのもかわいそうな気がします。

考え方 1	考え方 2
11 本線分を描画するために、line 関数を 11 回実行する。	X 座標の値が 25 から始めて、25 ずつ離れた場所に、X 座標の値が 275 以下の間、線分を描くために line 関数を実行する。

そこで、「X 座標の値が 275 以下の間、繰り返す」のように実行することができる命令があれば、**考え方 2** のようなプログラムを作ることができます。Processing では、この目的のために、while 命令が用意されいます。while 命令は、次に示すように非常に単純な形式になっています。

while 命令の使い方

Processing 言語での記法	処理の流れ

while(条件式){
繰り返し実行したい命令
}
	

while の意味はわかりますよね。

HP が 0 より大きい間、戦闘を繰り返すなどの、「ある条件の間繰り返す」といプログラムの実行の仕方は良く現れるパターンです。

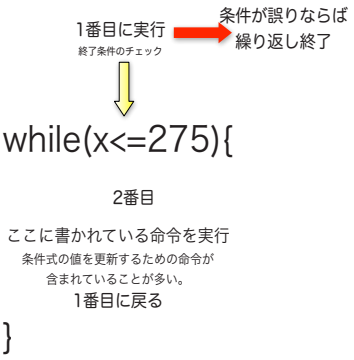
プログラミング言語によっては、ある条件が満たされるようになるまで、繰り返し処理をするという、「〜になるまで」型 (until 型) の繰り返し命令を持っているものを持っていることがあります。

サンプル 5-2 の考え方をベースに、while 命令を使って、サンプル 4-9 を書き換えると次の様になります。

while 命令を利用した繰り返し処理その 1 サンプル 5-4

```
size(300,200);
background(255);
stroke(0);
int x = 25;       // 繰り返し条件の初期値
while(x <= 275){ // 繰り返し条件のチェック
  line(x,20,x,180);  // 繰り返し実行したい命令
  x = x+25;          // 繰り返し条件の更新
}
```

繰り返し条件は、「変数 x の値が 275 以下」なので、while 命令の条件式の部分には、「x <= 275」となっています。



サンプル 5-4 のように、while 命令を使った繰り返し処理では、while 命令を実行する前に、繰り返し条件の初期化と、繰り返し実行したい命令の中に繰り返し条件の更新に関するものが含まれていることが一般的です。

別のサンプルとして、幅 160、高さ 20 の長方形を並べて描くことを考えます。このサンプルでは、次の様な条件となっています。

1. 一番上の長方形の左上の頂点の Y 座標は 10。
2. 長方形の間隔は 5。
3. 長方形の下の部分がウインドウからはみ出ないようにする。

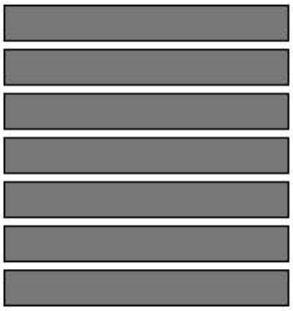
長方形の下の部分がウインドウからはみ出ないようにするためには、長方形の左上のＹ座標と長方形の高さを加えた値がウインドウの高さより小さければ、ウインドウからはみ出ません。つまり、繰り返し条件は、「長方形の左上のＹ座標と長方形の高さを加えた値がウインドウの高さより小さい」間となります。この方針で作成したものがサンプル 5-5 です。

「長方形の左上のＹ座標と長方形の高さを加えた値」とは、長方形の左下のＹ座標の値のことです。

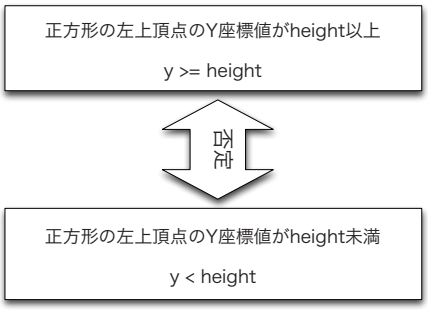
while 命令を利用した繰り返し処理その 2 サンプル 5-5

```
int y;   // 長方形の左上のＹ座標値
int w = 160;  // 長方形の幅
int h = 20;   // 長方形の高さ
int interval = 5;  // 長方形の間隔

size(200,200);
background(255);
fill(120);

y = 10;   // 最初に描く長方形の位置
while((y+h) < height){  // 繰り返し条件のチェック
  rect(20,y,w,h);       // 長方形の描画
  y = y+h+interval;     // 次に長方形を描く場所を求める
}
```

サンプル 5-5 の実行結果



条件指定型繰り返し処理を利用すると、回数指定型繰り返し処理では書くことの難しいプログラムを簡単に書くことができます。次のサンプル 5-6 では、マウスのカーソルの位置から下方向に向けて正方形を書いていきます。正方形の左上頂点のＹ座標値が height 以上の値になったら、正方形の描画を終了します。逆に言うと、正方形の左上頂点のＹ座標値が height 未満の間、正方形の描画を繰り返します。



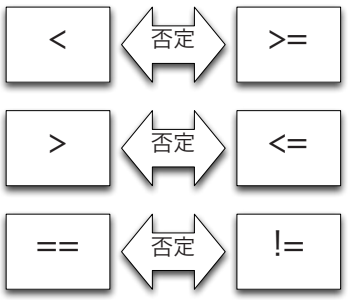
長方形の高さや間隔から何個の長方形を描くことが出来るかは計算で求めることが出来ます。でも、面倒なことはプログラムにさせるのが楽ですよね。

繰り返し処理の停止条件と繰り返し処理の繰り返し条件は、それぞれ否定の関係にあります。「否定」が難しいと「反対」のほうがわかり易いですか？

while 命令を利用した繰り返し処理その 3 サンプル 5-6

```
int y; // 正方形の左上頂点のＹ座標値
int sideLength = 20; // 正方形の一辺の長さ

void setup(){
  size(400,400);
  noFill();
  stroke(0);
  strokeWeight(2);
}
```

良くある否定の関係



```
void draw(){
  background(255);
  y = mouseY;  // マウスカーソルの位置から書き始める
  while(y < height){ // 正方形の左上頂点の Y 座標値が height 未満
    rect(mouseX,y,sideLength,sideLength);
    y = y + sideLength; // 次に正方形を描く位置を計算
  }
}
```

「y = y + sideLength;」ですが、「y += sideLength;」と書かれることのほうが一般的かも知れません。「+=」などの複合代入演算子に関しては、2 回前の授業プリントを参照。

サンプル 5-6 では、Y 座標値だけを変更しました。今度は、X 座標値も変えてみることにします。今回の正方形の左上頂点がウインドウにある間、正方形を描き続けることにします。どんな繰り返し条件式を書けば良いかわかりますか？

while 命令を利用した繰り返し処理その 4 サンプル 5-7

```
int x,y; // 正方形の左上頂点の X 座標値と Y 座標値
int sideLength = 20; // 正方形の一辺の長さ

void setup(){
  size(400,400);
  noFill();
  stroke(0);
  strokeWeight(2);
}

void draw(){
  background(255);
  x = mouseX;// マウスカーソルの位置から書き始める
  y = mouseY;
  while(x < width && y < height){
    rect(x,y,sideLength,sideLength);
    x += sideLength; // 次に正方形を描く位置を計算
    y += sideLength;
  }
}
```

同じデータ型の変数は、「,」で繋ぐことで、複数の変数を一度に宣言することが出来ます。どこかで、書いたかと思いますが。

複合代入演算子を使って書いてみました。

正方形の中を塗りつぶさないのもちょっと寂しいので、塗りつぶす色を変えながら、描画するサンプルをのせておきます。

while 命令を利用した繰り返し処理その 5 サンプル 5-8

```
int x,y; // 正方形の左上頂点の X 座標値と Y 座標値
int sideLength = 20; // 正方形の一辺の長さ
int c;   // 塗りつぶし色を決める変数

void setup(){
  size(400,400);
  noStroke();
}
```

変数 c も int 型変数なので、「int x,y,c;」と書いてもかまいません。通常は、まとまりのあるものを同時に宣言します。変数 x と変数 y は正方形の頂点位置ですが、変数 c は色を表しています。ちょっと、違いますよね。

```
void draw(){
  background(255);
  c = 0;
  x = mouseX;// マウスカーソルの位置から書き始める
  y = mouseY;
  while(x < width && y < height){
    fill(c);
    rect(x,y,sideLength,sideLength);
    c = c+10;
    x += sideLength; // 次に正方形を描く位置を計算
    y += sideLength;
  }
}
```

変数 c の値を draw の中で変えているので、毎回初期化（c=0;）する必要があります。

サンプル 5-5 ～ 5-8 のように、一定の大きさの図形を描くだけであれば、割と簡単に描く必要のある図形の個数を計算できることがあります。そこで、正方形の大きさを 1 割ずつ大きくしながら描画するサンプルを示します。

サンプル 5-6 とは、正方形の描画条件を変えています。

while 命令を利用した繰り返し処理その 6 サンプル 5-9

```
int x,y; // 正方形の左上頂点の X 座標値と Y 座標値
float sideLength; // 正方形の一辺の長さ

void setup(){
  size(400,400);
  stroke(0);
  noFill();
}

void draw(){
  background(255);
  sideLength = 20;
  x = mouseX;// マウスカーソルの位置から書き始める
  y = mouseY;
  while(x < width && y < height){
    rect(x,y,sideLength,sideLength);
    x += sideLength; // 次に正方形を描く位置を計算
    y += sideLength;
    sideLength = 1.1*sideLength;
  }
}
```

辺の長さを 1 割ずつ大きくしながら描画をするので、sideLength を float 型で宣言してます。辺の長さを 1 割長くするために、1.1 倍しているからです。

等比級数の和を求めれば、何とかなる気がします。やっぱり、面倒なことはコンピュータにやらせましょう。

実は、1 割増しなどの単純な変更方法では、がんばると描く必要のある図形の個数を計算できることがあります。そこで､乱数を使って、図形を描く間隔を変えることで、事前に描画する図形の個数を計算できないようなサンプルをのせておきます。このサンプルでは、正方形ではなく、円を描き、色も乱数で変えています。

while 命令を利用した繰り返し処理その 7 サンプル 5-10

```
float diam; // 円の直径
float x,y;  // 円の中心座標
int c;      // 描画色

void setup(){
  size(400,400);
  smooth();
  noStroke();
}

void draw(){
  background(255);
  x = mouseX;
  y = mouseY;
  c = 0;
  diam = 10;
  while(x < width && y < height){
    fill(c);
    ellipse(x,y,diam,diam);
    x = x+random(1,diam);  // X 軸方向の移動は乱数で決定
    y = y+diam/2;          // Y 軸方向には半径分だけ移動
    c = c+int(random(5,10)); // 描画色の決定
    diam = 1.1*diam;         // 直径を 1 割増やす
  }
}
```

乱数を使って、円の位置を変更しているので、変数 x,y は float 型になっています。円の直径も 1 割ずつ増加しているので、変数 diam も float 型になっています。塗りつぶし色も乱数で変えているのです が、int(random(5,10)) で乱数を作っているので、5 以上 10 未満の整数の乱数となっているので、描画色を決めている変数 c は int 型となっています。

もう一つ別な while を使ったサンプルをのせておきます。このサンプルでは、2 段の線分を描いています。ただし、上の段と下の段では、線分を描く間隔が異なっています。この間隔はマウスカーソルの位置 (mouseX の値) で決めています。サンプル 5-11 には、if 命令の部分があります。この部分がないと不都合が起きることがあります。

ここでの不都合とは、描画が終わらないことを指しています。

今まで、キチンと説明をしていませんでしたが、int 型と int 型の割り算は、割り算の結果も int 型になります。例えば、4/2 は 2 となりますが、5/2 は 2.5 ではなく、2 となります。また、1/10 は 0.1 ではなく、0 となります。つまり、mouseX の値が 0 や 1 の時には、mouseX/2 は 0 となります。すると、interval の値は 0 となりますので、常に「x < width」が正しくなるので、描画が終了しないことになります。そこで、interval が 0 の時には、強引に 1 に変更しています。

5.0/2 や 5/2.0 とすれば、float 型と int 型の計算になるので、計算結果は 2.5 となります。

while 命令を利用した繰り返し処理その 8 サンプル 5-11

```
int x,y;  // 線分の描画位置を示す変数
int interval;  // 描画する線分の間隔を表す変数
int len;  // 描画する線分の長さを表す変数
```

```
void setup(){
  size(200,200);
  y = height/2;
  len = height/5;
}

void draw(){
  background(255);
  x = 0;
  interval = mouseX/2;
  if(interval == 0){
    interval = 1;
  }
  while (x < width){    // 上の段の描画
    line(x,y-len,x,y-2*len);
    x = x + interval;
  }
  x = 0;
  while(x < width){    // 下の段の描画
   line(x,y+len,x,y+2*len);
    x = x + 2*interval;
  }
}
```

強力な for 命令

Processing 言語の先祖のようなプログラミング言語に、C 言語があります。C 言語はプログラミング業界に大きな影響を及ぼしています。C 言語以前の多くの言語では、回数指定型の繰り返し処理と条件指定型の繰り返し処理は、明確に異なった命令文を使って、表されていました。C 言語をデザインした人は、回数指定型の繰り返し処理を行う際に指定する必要のあることを、次の 3 つだと考えました。

1. カウンタ変数の**初期化**
2. 繰り返し回数に関する**条件チェック**
3. カウンタ変数の値の**更新**

この 3 つの条件が指定できるように C 言語での for 命令をデザインしました。これの考え方を受け継いで作られたのが Processing 言語の for 命令です。

これをベースに前回説明した for 命令の表記方法を見直してみます。前回は、for 命令の使い方は下のようになっていると説明しました。

C 言語は、1972 年に AT&T ベル研究所のデニス・リッチー (Dennis M. Ritchie,1941-2011) が中心となってデザインしたプログラミング言語です。C 言語と類似の文法が多くのプログラミング言語に取り入れられています。

キチンと調べた訳でないのですが、このように回数指定型の繰り返し処理を考えて、for 命令をデザインしたことが、C 言語の大きな特徴の一つではないかと思います。

for 命令の表記方法（ちょっと退化型？）

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
}

これは、カウンタ変数を 0 から数え始め、カウンタ変数を 1 ずつ増やしながら、カウンタ変数が「繰り返し回数 -1」になるまで、繰り返し実行したい命令を実行するというものです。これには、次のような対応関係があります。

カウンタ変数を 0 から数え始め	＝	int カウンタ名 = 0
カウンタ変数を 1 ずつ増やす	＝	カウンタ名 ++
カウンタ変数が「繰り返し回数 -1」になるまで	＝	カウンタ名 < 繰り返し回数

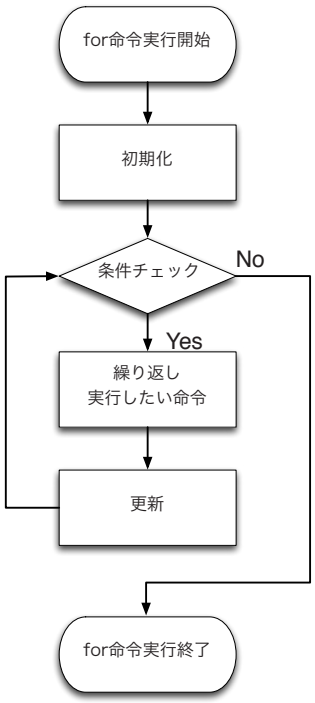
Processing 言語での for 命令は、本来、次の様にデザインされています。

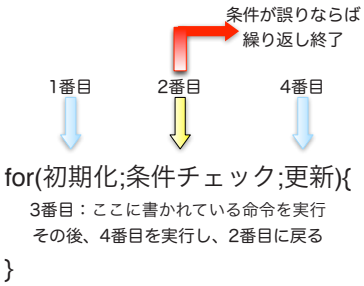
for 命令の本来の使い方（一般型）

Processing 言語での記法

for(初期化 ; 条件チェック ; 更新){
　繰り返し実行したい命令
}

処理の流れ





サンプル 4-9 を次のように書き換えてみます。このように書き換えると、カウンタ変数を 1 で初期化し、カウンタ変数を 1 ずつ増やしながら、カウンタ変数が 11 以下の間、繰り返すということが実現

できれば、良いことがわかります。この方針で、for 命令を使用した作成したプログラムがサンプル 5-13 です。

line 関数を 11 回実行する サンプル 5-12

```
size(300,200);
background(255);
stroke(0);
line( 1*25,20,  1*25,180);
line( 2*25,20,  2*25,180);
line( 3*25,20,  3*25,180);
line( 4*25,20,  4*25,180);
line( 5*25,20,  5*25,180);
line( 6*25,20,  6*25,180);
line( 7*25,20,  7*25,180);
line( 8*25,20,  8*25,180);
line( 9*25,20,  9*25,180);
line(10*25,20, 10*25,180);
line(11*25,20, 11*25,180);
```

カウンタ変数を 1 から始める サンプル 5-13

```
size(300,200);
background(255);
stroke(0);
for(int i=1;i<=11;i++){
  line(i*25,20,i*25,180);
}
```

カウンタ変数 i を 1 から数え始め	=	int i = 1
カウンタ変数 i を 1 ずつ増やす	=	i++
カウンタ変数 i が 11 以下の間	=	i <=11 または i<12

カウンタ変数を 1 から始める サンプル 5-13'

```
size(300,200);
background(255);
stroke(0);
for(int i=1;i<12;i++){
  line(i*25,20,i*25,180);
}
```

サンプル 5-13 では、カウンタ変数を 1 からに変更し、カウンタ変数が 1 ずつ増えるサンプルでした。次は、カウンタ変数を 1 ずつではなく、別な値で変更するサンプルを示します。

前回の講義プリントのサンプル 4-16 では、中心が (0,0)、(40,0)、(80,0)、…、(400,0) と (0,0)、(0,40)、(0,80)…、(200,0) の位置に直径 40 の円を描いています。

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。

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。

整数値の場合には、「11 以下」ということと「12 未満」は同じことなので、左のように 2 通りのやり方があります。

上の注意の意味がわかると、サンプル 5-13 とサンプル 5-13' が同じ動作となることがわかります。

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);
}
```

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。

つまり、X 座標に関しては、カウンタ変数を 0 からはじめて、40 ずつ増加させ 400 以下の間、Y 座標に関しては、カウンタ変数を 0 からはじめて、40 ずつ増加させ、200 以下の間、円を描いています。そこで、サンプル 5-14 のようにすれば、サンプル 4-16 と同じ動作になります。

カウンタ変数を 40 ずつ増やすサンプル 5-14

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

カウンタ変数 x と y を 40 ずつ増やすと言うことを、それぞれ「x+=40」、「y+=40」という複合代入演算子で実現しています。

カウンタ変数の更新は、定数値で決めなくても良いので、while 命令を使って作ったサンプル 5-11 は次のサンプル 5-15 のように書き換えることが出来ます。

変数を利用したカウンタ変数の更新　サンプル 5-15

```
int y;  // 線分の縦位置を決めるための変数
int interval;  // 描画する線分の間隔を表す変数
int len;  // 描画する線分の長さを表す変数

void setup(){
  size(200,200);
  y = height/2;
  len = height/5;
}
```

```
void draw(){
  background(255);
  interval = mouseX/2;
  if(interval == 0){
    interval = 1;
  }
  for(int x=0;x < width; x += interval){// 上の段の描画
    line(x,y-len,x,y-2*len);
  }
  for(int x=0;x < width; x += 2*interval){// 下の段の描画
   line(x,y+len,x,y+2*len);
  }
}
```

また、for 命令の「初期化」、「条件チェック」、「更新」の部分を全て記入する必要はありません。例えば、サンプル 5-10 を次のように for 命令を使って書き換えることが出来ます。この例では、「初期化」と「更新」の部分が省略されています。このサンプル 5-16 のように、「while(条件){ 〜 }」は「for(; 条件 ;){ 〜 }」と全く同じことになります。

for 命令による while 命令の置き換えサンプル 5-16

```
float diam; // 円の直径
float x,y;  // 円の中心座標
int c;      // 描画色
void setup(){
  size(400,400);
  smooth();
  noStroke();
}
void draw(){
  background(255);
  x = mouseX;
  y = mouseY;
  c = 0;
  diam = 10;
  for(;x < width && y < height;){
    fill(c);
    ellipse(x,y,diam,diam);
    x = x+random(1,diam);  // X 軸方向の移動は乱数で決定
    y = y+diam/2;          // Y 軸方向には半径分だけ移動
    c = c+int(random(5,10)); // 描画色の決定
    diam = 1.1*diam;         // 直径を 1 割増やす
  }
}
```

つまり、Processing 言語のような C 言語系のプログラミング言語にとっては、while 命令は盲腸のような存在です。

カウンタ変数を条件チェックでは、どのような条件チェックを行っても良いので、カウンタ変数の値が条件チェックの中に入っていなくてもかまいません。すると、サンプル 5-16 は次のように書き換え

ることも出来ます。ちょっとやり過ぎかも知れませんが。

for 命令による while 命令の置き換えサンプル 5-17

```
float x,y;  // 円の中心座標
int c;      // 描画色
void setup(){
  size(400,400);
  smooth();
  noStroke();
}
void draw(){
  background(255);
  x = mouseX;
  y = mouseY;
  c = 0;
  for(float diam=10;x < width && y < height;diam = 1.1*diam){
    fill(c);
    ellipse(x,y,diam,diam);
    x = x+random(1,diam);  // X 軸方向の移動は乱数で決定
    y = y+diam/2;          // Y 軸方向には半径分だけ移動
    c = c+int(random(5,10)); // 描画色の決定
  }
}
```

キチンと説明はしませんが、サンプル 5-17 はさらに次のように書き換えることが出来ます。ちょっとやり過ぎな気もしますが。

for 命令による while 命令の置き換えサンプル 5-16

```
int c;
void setup(){
  size(400,400);
  smooth();
  noStroke();
}

void draw(){
  background(255);

  c = 0;
  for(float diam=10,x=mouseX,y=mouseY;
      x < width && y < height;
      x += random(1,diam),
      y += diam/2,
      diam *= 1.1,
      c += int(random(5,10))){
    fill(c);
    ellipse(x,y,diam,diam);
  }
}
```

複合代入演算子を使用して「diam *= 1.1」と書いている部分は、「diam = 1.1*diam」書くことも出来ます。

色相、彩度、明度による色指定

色の指定の方法に、色の三原色の組み合わせ (RGB) による方法と色相・彩度・明度 (HSB) による方法があることを説明しました。今まで作ってきたプログラムは RGB により色を指定していました。しかし、HSB による色指定を行うと、赤っぽい色を乱数で出したいなどの場合に便利です。また、色を指定する際には、不透明度（α値）という情報を付加して使用することもあります。Processing では、HSB による色指定を行うことも出来ます。色指定方法を変更するために、colorMode 関数を利用します。colorMode 関数には、沢山の呼び出し方があります。それを以下の表にまとめました。なお、不透明度の値を指定する場合には、明示的に変更しない限り、0 以上 255 以下となっています。

色相：Hue
彩度：Saturation
明度：Brightness

α：ギリシャ文字の小文字の a です。アルファと呼びます。

この不透明度の使い方は、少し後で説明します。

colorMode 関数の使い方

色指定の方法	colorMode 関数の呼び出し方	意味
RGB による色指定	colorMode(RGB)	RGB の各値は 0 以上 255 以下の数値で表します。
	colorMode(RGB, colorMax)	RGB の各値は 0 以上 colorMax 以下の数値で表します。
	colorMode(RGB, rMax, gMax, bMax, alphaMax)	それぞれ、R の値は 0 以上 rMax 以下、G の値は 0 以上 gMax 以下、B の値は 0 以上 bMax 以下の数値で指定します。この場合には、不透明度の値の範囲を明示的に変更していますので、不透明度の値は 0 ～ alphaMax の数値で指定します。
HSB による色指定	colorMode(HSB)	HSB の各値は、0 ～ 255 の数値で指定します。
	colorMode(HSB, huMax, saturationMax, brightnessMax)	それぞれ、hue の値は 0 以上 hueMax 以下、saturation の値は 0 以上 saturationMax 以下 , briightness の値は 0 以上～ brightnessMax 以下の数値で指定します。
	colorMode(HSB, hueMax, saturationMax, brightnessMax, alphaMax)	それぞれ、hue の値は 0 以上 hueMax 以下、saturation の値は 0 以上 saturationMax 以下 , briightness の値は 0 以上～ brightnessMax 以下の数値で指定します。この場合には、不透明度の値の範囲を明示的に変更していますので、不透明度の値は 0 ～ alphaMax の数値で指定します。

基本的に、0 より小さい値を指定した場合には 0、上限値よりも大きな値を指定した場合には上限値となります。

Processing の ColorSelector では、HSB での色指定の際には、hue に関しては 0 以上 359 以下、saturation と brightness に関しては 0

以上 99 以下で表しています。基本的に HSB での色指定を行う場合には、colorMode(HSB,359,99,99) での指定を利用したいと思います。

以下に、HSB により色指定のサンプルを示します。サンプル 5-19 では、色相 (Hue) の値を一定にして、彩度と明度を少し変更したものです。

HSB による色指定その 1 サンプル 5-19

```
size(400,200);
colorMode(HSB,359,99,99);
smooth();

background(190,60,99);
// 上段の円、色相の値は 0
fill(0,99,99);
ellipse(100,50,80,80);
fill(0,60,99);
ellipse(200,50,80,80);
fill(0,30,99);
ellipse(300,50,80,80);
// 下段の円、色相の値は 100
fill(100,99,99);
ellipse(100,150,80,80);
fill(100,60,99);
ellipse(200,150,80,80);
fill(100,30,99);
ellipse(300,150,80,80);
```

サンプル 5-20 では、色相、彩度、明度の値の範囲を、ウインドウの大きさので決めています。このように値の範囲を設定すると、マウスカーソルの位置情報 mouseX と mouseY の値を、直接色指定の情報として利用することが出来ます。

HSB による色指定その 2 サンプル 5-20

```
void setup(){
  size(400,400);
  colorMode(HSB,width-1,height-1,height-1);
  noStroke();
}

void draw(){
  fill(mouseX,mouseY,mouseY);
  rect(mouseX,0,5,height);
}
```

サンプル 5-21 は、HSB による色指定と繰り返し処理を組みあわせたものです。X 軸方向に移動すると色相が変化し、Y 軸方向に移動すると彩度が変化するようになっています。

HSB による色指定その 2 サンプル 5-21

```
size(400,400);
colorMode(HSB,359,99,99);
noStroke();

for(int y = 0;y < 10;y++){
  for(int x = 0;x < 10;x++){
    fill(36*x,9+10*y,99);
    rect(40*x+5,40*y+5,30,30);
  }
}
```

これらのサンプルでもわかるように、HSB による色指定を行うと、何とか色っぽいという色の指定が簡単に行えます。

サンプル 5-22 は、HSB による色指定と乱数を組みあわせたものです。

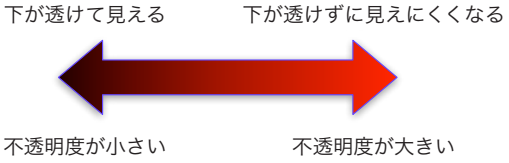
HSB による色指定その 2 サンプル 5-22

```
size(400,400);
colorMode(HSB,359,99,99);
rectMode(CENTER);
background(0,0,99); // 背景を白にする、彩度の値は 0
stroke(0,0,0);      // 枠線を黒にする、明度の値は 0
for(int i=0;i<400;i++){
  float r1 = random(30);
  float r2 = random(50,100);
  fill(r1,r2,r2);
  rect(random(width),random(height),30,30);
}
```

重要：HSB による色指定において、白は彩度の値を 0、黒は明度の値を 0 とします。

不透明度の指定

今までの Processing 言語のサンプルで見たようにように、図形の上に別な図形を描画すると、最初に描かれていた図形は完全に塗りつぶされてしまい、消えてしまいます。このように、最初に描かれた図形を完全に消えてしまうことを防ぐために、不透明度付きの色を利用します。不透明度の値を小さくすると、その下に描かれている図形が透けて見えるようになります。



不透明度のことを α チャンネルと呼ぶこともあります。

不透明度の付きの色の場合には、次のような式で描画色を求めます。簡単のために、不透明度は 0 ～ 255 の数値で表されているものとします。c が

$$c = \frac{\alpha}{255} fgColor + (1 - \frac{\alpha}{255}) bgColor$$

実際に描画される色の値、bgColor が背景色、fgColor が指定描画色です。

このような計算方法を線形補間と呼びます。CG やゲームなどでよく使われる計算方法です。

サンプル 5-2 3 は、不透明度付き色指定を行ったサンプルです。

不透明度付き色指定その 1 サンプル 5-23

```
size(400,200);
smooth();
colorMode(HSB,359,99,99); // 不透明度は 0 ～ 255
background(200,60,99);

fill(0,100,91,255); // 不透明度が 255 なので、下にある図形は隠れる
ellipse(50,height/2,150,150);

fill(0,60,91,127);// 不透明度が 127 なので、下にある図形は少し隠れる
ellipse(150,height/2,150,150);

fill(0,60,91,63);// 不透明度が 127 なので、下にある図形は透ける
ellipse(250,height/2,150,150);

noFill();// 塗りつぶしなし
ellipse(350,height/2,150,150);
```

サンプル 5-24 は、サンプル 5-22 の色指定に不透明度の情報を追加したものです。

不透明度付き色指定その 2 サンプル 5-24

```
size(400,400);
colorMode(HSB,359,99,99);
rectMode(CENTER);
background(0,0,99); // 背景を白にする、彩度の値は 0
stroke(0,0,0);      // 枠線を黒にする、明度の値は 0
for(int i=0;i<400;i++){
  float r1 = random(30);
  float r2 = random(50,100);
  fill(r1,r2,r2,random(100,200));
  rect(random(width),random(height),30,30); // 不透明度を追加
}
```

不透明度の情報を使うと、フェードアウトするような処理を実現出来ます。つまり、下に描かれている図形を完全に消去しないで、不透明度の付きの色で塗りつぶすことで、徐々に消えていくような処理を再現できます。このような方針で作成したものがサンプル 5-25 です。

このサンプルでは、draw 関数の先頭で background 関数を利用した塗りつぶしを行わずに、不透明度の付きの色指定 (不透明度 64 の白) でウインドウ全体を塗りつぶしています。不透明度付きの色でで塗りつぶしているので、始めに描かれている画像が完全に消えずに、少し残ります。従って、昔に書かれた円は色が徐々に薄くなっていき、最終的には消えてしまます。これが繰り返し行われるので、残像が残ったような効果が再現出来ます。不透明度の値を大きくすると、残像は直ぐに消えるようになります。逆に、不透明度の値を小さくすると、

残像が長く残るようになります。

不透明度付き色指定その 3 サンプル 5-25

```
void setup(){
  size(400,400);
  colorMode(HSB,359,99,99);
  smooth();
  noStroke();
}

void draw(){
  fill(0,0,99,64); // 不透明度付きの色で塗りつぶす
  rect(0,0,width,height);
  fill(0,99,99);
  ellipse(mouseX,mouseY,30,30);
}
```

不透明度を使うと、色々な面白い効果を得ることが出来ます。

不透明度付き色指定その 4 サンプル 5-26

```
size(400,400);
colorMode(RGB);
smooth();
background(255);
fill(255,10,10,50);
stroke(10,100,255,80);
strokeWeight(100);
ellipse(100,100,400,400);
ellipse(300,100,400,400);
ellipse(100,300,400,400);
ellipse(300,300,400,400);
```

Processing では、色の情報を保存するために、color 型というデータが用意されています。この color 型を利用すると、色のデータを保存しておくことが出来ます。ただし、色の情報は複数の値を指定しないと確定しないので、color 関数を使用して color 型のデータを作り出します。color 関数の引数は、fill 関数や stroke 関数などと同じです。この color 型を利用したサンプルを次に示します。

color 型による色指定サンプル 2 サンプル 5-27

```
size(400,400);
smooth();

color c1 = color(10,100,255);
color c2 = color(255,200,10,128); // 不透明度の情報を利用できる
color c3 = color(255);  // これは白
color c4 = color(0);    // これは黒
```

```
background(c3);  // background でも利用可能
stroke(c4);      // stroke でも利用可能
fill(c1);        // fill でも利用可能
ellipse(150,150,250,250);
fill(c2);
ellipse(250,250,250,250);
```

color 型による色指定サンプル 2 サンプル 5-28

```
color c,c1,c2;

void setup(){
  size(400,400);
  rectMode(CENTER);
  c1 = color(255,10,10);
  c2 = color(10,255,10);
}

void draw(){
  background(255);
  if(mousePressed){
    c = c1;  // 当然、color 型変数への代入も出来ます
  }else{
    c =c2;
  }
  fill(c);
  rect(width/2,height/2,300,300);
}
```