情報メディア基盤ユニット用資料（2015 年 6 月 16 日分）

Processing 言語による情報メディア入門

配列

神奈川工科大学情報メディア学科　佐藤尚

配列

次のような上から下へ円が移動するようなプログラムを考えます。このサンプルでは、1 つの円を動かしています。変数 y に円の中心の Y 座標値を保存し、縦方向の移動量を表す変数 v を使って、1) 円の描画位置を計算、2) 下まで到達するとしたら、円を上に移動させる、ついでに中心の X 座標の値も変更、3) 円を描画する、というアルゴリズムでプログラムを作っています。

プログラムの処理内容をアルゴリズム (algorithm) と呼びます。この名称は、現在のイラクのバグダードにおける 9 世紀の数学者アル・フワーリズミー の名前から来ていると言われています。日本語では算法と呼ぶこともあります。現在では、この呼び方をする人は、超少数派だと思いますが。

1 個の円の移動 サンプル 9-1

```
float y; // 円の中心の Y 座標
float x; // 円の中心の X 座標
float v; // 円の縦方向の移動速度
int radius;

void setup(){
  size(300,400);
  smooth();
  radius = 10;
  v = random(1,2); // 移動速度を乱数で決める
  x = random(radius,width-radius); // 出現位置をずらす
  y = -random(radius,2*radius);  // 出現タイミングをずらすため
}

void draw(){
  background(255);

  y = y+v;
  if(y -radius> height){
    x = random(radius,width-radius); // 出現位置をずらす
    y = -random(radius,2*radius);    // 出現タイミングをずらすため
  }
  stroke(255,10,10);
  fill(255,10,10);
  ellipse(width/2,y,2*radius,2*radius);
}
```

ゲームなどでは、沢山の敵キャラ（敵機）が移動してきます。例えば、2 つの円が移動するようなプログラムは、次の様に書くことが出来ます。

2 個の円の移動 サンプル 9-2

```
float y0,y1; // 円の中心の Y 座標
float x0,x1; // 円の中心の X 座標
float v0,v1; // 円の縦方向の移動速度
int radius;

void setup(){
  size(300,400);
  smooth();
  radius = 10;
  v0 = random(1,2); // 移動速度を乱数で決める
  y0 = -random(radius,2*radius);  // 出現タイミングをずらすため
  x0 = random(radius,width-radius); // 出現位置をずらす
  v1 = random(1,2); // 移動速度を乱数で決める
  y1 = -random(radius,2*radius);  // 出現タイミングをずらすため
  x1 = random(radius,width-radius); // 出現位置をずらす
}

void draw(){
  background(255);
  // 中心 (x0,y0) の円の処理
  y0 = y0+v0;
  if(y0 -radius> height){
    x0 =random(radius,width-radius);// 出現位置をずらす
    y0 = -random(radius,2*radius);  // 出現タイミングをずらすため
  }
  stroke(255,10,10);
  fill(255,10,10);
  ellipse(x0,y0,2*radius,2*radius);

  // 中心 (x1,y1) の円の処理
  y1 = y1+v1;
  if(y1 -radius> height){
    x1 = random(radius,width-radius);// 出現位置をずらす
    y1 = -random(radius,2*radius);   // 出現タイミングをずらすため
  }
  stroke(255,10,10);
  fill(255,10,10);
  ellipse(x1,y1,2*radius,2*radius);
}
```

変数 x0 は x1 に、変数 y 0 は y1 に変数 v0 は v1 に変わっているだけです。

サンプル 9-1 とサンプル 9-2 では大きな違いはありません。サンプル 9-2 では、2 つの円を扱う必要があるので、中心が (x0,y0) の円と中心が (x1,y1) の円の 2 つの円を扱ってるので、同じ内容の処理で、変数名の部分が異なっているものが書かれています。

サンプル 9-2 の方針で、もっと沢山の円が移動するプログラムを作るとすると、沢山の変数を用意する必要があります。例えば、10 個の円を表示するように拡張する場合には、変数 x0 〜 x9、y0 〜 y9、v0 〜 v9 が必要になるようの思います。このように沢山の変数

0 から数え始めているので、終わりは 10 ではなく、9 になります。大丈夫ですか？

を扱うプログラムを書くことは可能ですが、変数名の数字の部分だけが異なった処理を 10 回書く必要があります。これはかなり面倒です。

この面倒を避けるためには、変数を変数名と数字のペアで指定できれば、解決できそうです。サンプル 9-2 でも、変数名の x,y,v の部分は共通で、変数名の最後の数字の部分が異なっているだけです。Processing では、このような変数名と数字のペアで変数を指定する方法として、配列と呼ばれるものが準備されています。

プログラムを作成するときの、大きな方針の 1 つは、「同じような処理はまとめて書く」です。優秀なプログラマはものぐさです。それに、コンピュータは同じ繰り返しを飽きずに処理することが得意です。

数学などでも、x_0,x_1 のような添え字を使うことがあります。

配列も変数の一種なので、使う前に宣言が必要となります。また、どんな種類のデータを変数に保存するかというデータ型の情報も必要となります。そのため、次のような方法で配列型変数の宣言を行います。

英語では、配列のことを array と呼びます。
配列に保存されている一つ一つのデータのことを要素 (elment) と呼ぶことがあります。

表 9-1 配列型変数の宣言

宣言方法	宣言例
データ型　[] 配列変数名	float [] x; int [] radius; String [] msg;

表 9-1 の宣言だけでは、何個の分のデータを保存するかということがわからないので、この宣言以外に何個のデータを保存する場所を用意するために、次のような処理を行う必要があります。

この処理を行わないとエラーになります。また、確保した個数以上のデータを使う場合にもエラーとなります。

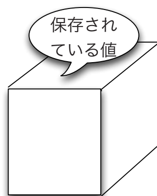
表 9-2 配列型変数のための場所の確保

宣言方法
配列変数名　= new データ型 [確保するデータ数];
x = new float[10];
radius = new int[10];
msg = new String[20];

変数の宣言と場所の確保を同時に行うことが出来ます。

表 9-1 通常の変数と配列型変数のイメージ

変数：一つの名前に、一つのデータを保存



配列型変数：一つの名前に、複数のデータを保存

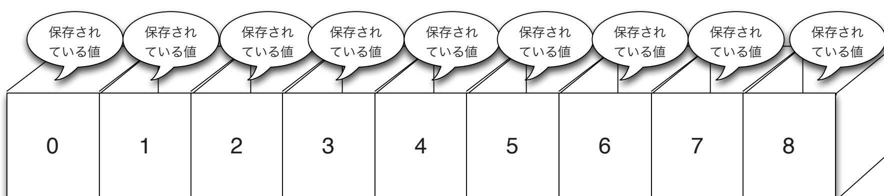


表 9-3 配列型変数の宣言と場所の確保

配列に保存されたデータは、添え字の番号順に一列に並んでいるイメージとなっています。恐らくメモリ内でも一列に並んでいると思います。

宣言方法
データ型 [] 配列変数名　= new データ型 [確保するデータ数];
float[] x = new float[10];
int[] radius = new int[10];
String [] msg = new String[20];

同じデータ型の名前を２箇所に書かないといけないのが、面倒ですよね。もう少し上手く設計して欲しい気がします。

配列の中にデータを保存したり、読み出したするためには、以下のような方法をとります。

表 9-4 配列の要素へのアクセス

配列のアクセス	例
配列変数名 [番号]	x[0] = random(10); y[1] = y[1]+v[1];

これらのサンプルのように、色々な関数を組み合わせることでどんどん複雑なプログラムを作ることが出来るようになります。

つまり、プログラム中で配列の中に保存されているデータを読み出したり、配列の中にデータを保存したりする場合には、必ず [と] の間に数字を指定して指示します。この数字を添え字またはインデックスと呼びます。添え字は配列の中のどのデータを使うのかを指定したり、配列のどの場所にデータを保存するのかを指定します。プログラム中では複数の配列型変数を使用することができます。そのために、どの配列かを区別するために配列変数名を利用します。Processing の配列では添え字の番号は 0 からスタートします。そのため､一番最後の要素の添え字番号は「配列の要素数 -1」となります。例えば、10 個の要素を持つ配列の要素は 0 から 9 までの添え字番号で指定することが出来ます。

配列は数多くのデータを添え字番号で指定することが出来るので、繰り返し処理と相性の良い方法となっています。

以上の説明をもとに、サンプル 9-2 を配列を使ったものに書きかえてみます。

2 個の円の移動（配列版）サンプル 9-3

```
float[] x = new float[2];// 円の中心の X 座標
float[] y = new float[2];;// 円の中心の Y 座標
float[] v = new float[2];;// 円の縦方向の移動速度
int radius;

void setup(){
  size(300,400);
  smooth();

  radius = 10;
  v[0] = random(1,2); // 移動速度を乱数で決める
  y[0] = -random(radius,2*radius);  // 出現タイミングをずらすため
  x[0] = random(radius,width-radius);
  v[1] = random(1,2); // 移動速度を乱数で決める
  y[1] = -random(radius,2*radius);  // 出現タイミングをずらすため
  x[1] = random(radius,width-radius);
}
```

このサンプルでは、配列型変数の宣言と保存場所の確保を同時に行っています。

```
void draw(){
  background(255);

  y[0] = y[0]+v[0];
  if(y[0]-radius > height){
    x[0] = random(radius,width-radius);// 出現位置をずらす
    y[0] = -random(radius,2*radius);; // 出現タイミングをずらすため
  }
  stroke(255,10,10);
  fill(255,10,10);
  ellipse(x[0],y[0],2*radius,2*radius);

  y[1] = y[1]+v[1];
  if(y[1]-radius > height){
    x[1] = random(radius,width-radius);// 出現位置をずらす
    y[1] = -random(radius,2*radius);; // 出現タイミングをずらすため
  }
  stroke(255,10,10);
  fill(255,10,10);
  ellipse(x[1],y[1],2*radius,2*radius);
}
```

サンプル 9-2 とサンプル 9-3 では、大きく変わっていません。サンプル 9-2 で x0 などなっている部分が x[0] などになっているだけです。例えば、「y[0]=y[0]+v[0];」は、配列変数 y の添え字番号 0 番の値と配列変数 v の添え字番号 0 番の値を加えて、その結果を配列変数 y の添え字番号 0 番に保存するという意味です。

サンプル 9-3 の赤字の部分に注目すると、サンプル 9-4 のように for 命令を使った繰り返し処理で書けるように思えます。そこで、for 命令を使って「書きかえたものがサンプル 9-4 です。

2 個の円の移動（for 命令 + 配列版）サンプル 9-4

```
float[] x;// 円の中心の X 座標
float[] y;// 円の中心の Y 座標
float[] v;// 円の縦方向の移動速度
int radius;
void setup(){
  size(300,400);
  smooth();
  radius = 10;
  x = new float[2];
  y = new float[2];
  v = new float[2];
  for(int i=0;i<2;i++){
    v[i] = random(1,2); // 移動速度を乱数で決める
    y[i] = -random(radius,2*radius);  // 出現タイミングをずらすため
    x[i] = random(radius,width-radius);
  }
}
```

配列変数の宣言と、保存する場所の確保を別な位置で行うように変更してみました。

繰り返し処理のカウンタ変数 i を使って、配列の要素に値を保存しています。

```
void draw(){
  background(255);

  for(int i=0;i<2;i++){
    y[i] = y[i]+v[i];
    if(y[i]-radius > height){
      x[i] =random(radius,width-radius);// 出現位置をずらす
      y[i] = -random(radius,2*radius);  // 出現タイミングをずらす
    }
    stroke(255,10,10);
    fill(255,10,10);
    ellipse(x[i],y[i],2*radius,2*radius);
  }
}
```

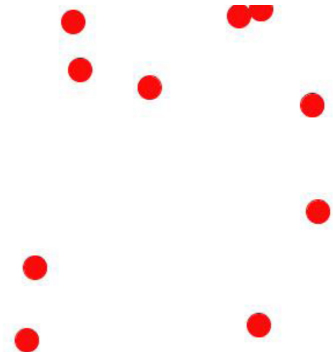
繰り返し処理のカウンタ変数 i を使って、配列の要素にアクセスしています。

サンプル 9-4 のようになると、配列と繰り返し処理の組み合わせが強力なことが見えてきます。例えば、円の数を 10 個に増やしたい場合には、サンプル 9-5 のようになります。

10 個の円の移動（for 命令 + 配列版）サンプル 9-5

```
float[] x;// 円の中心の X 座標
float[] y;// 円の中心の Y 座標
float[] v;// 円の縦方向の移動速度
int radius;
void setup(){
  size(300,400);
  smooth();
  radius = 10;
  x = new float[10];
  y = new float[10];
  v = new float[10];
  for(int i=0;i<10;i++){
    v[i] = random(1,2); // 移動速度を乱数で決める
    y[i] = -random(radius,2*radius);  // 出現タイミングをずらすため
    x[i] = random(radius,width-radius);
  }
}
void draw(){
  background(255);

  for(int i=0;i<10;i++){
    y[i] = y[i]+v[i];
    if(y[i]-radius > height){
      x[i] =random(radius,width-radius);// 出現位置をずらす
      y[i] = -random(radius,2*radius);  // 出現タイミングをずらす
    }
    stroke(255,10,10);
    fill(255,10,10);
    ellipse(x[i],y[i],2*radius,2*radius);
  }
}
```

配列に関わる 2 を 10 に置き換えただけです。



繰り返し回数を配列の要素数より大きな値を指定すると、「y[i]=y[i]+v[i];」の部分で、カウンタ変数 i の値が添え字番号の範囲を超えてしまいます。そうすると、ArrayIndexOutOfBoundsException というエラーが出て、実行が停止します。

サンプル 9-5 を書きかえて、100 個の円を表示するようにするためには、サンプル 9-5 中の赤字の 10 の部分を 100 に書きかえるだけで実現出来ます。配列と繰り返し処理を組み合わせると沢山の物体を表示したりするような処理を簡単に書くことができます。サンプル 9-4 や 9-5 で、for 命令の繰り返し回数を指定している数字は、配列にいくつのデータを保存することが出来るのか（要素数）を指定しています。配列を使ったプログラムでは、要素数を知りたいことが多いので、length というプロパティが用意されています。

length の意味はわかりますか？

表 9-5 配列の要素数の取得

要素数の取得方法	例
配列変数名 .length	for(int i=0;i<x.length;i++){ radius.length; msg.length;

この length を使うと、サンプル 9-5 は次のように書きかえることが出来ます。

10 個の円の移動（length 使用版）サンプル 9-5'

書きかえたのは、赤字の部分のみです。

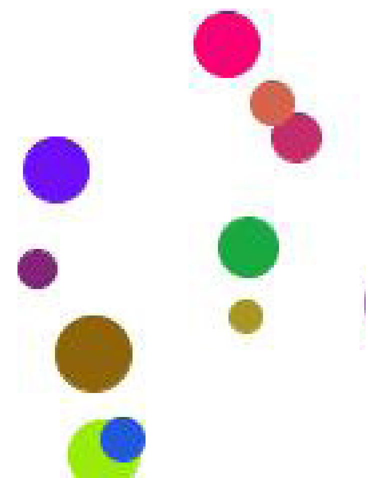
```
float[] x;// 円の中心の X 座標
float[] y;// 円の中心の Y 座標
float[] v;// 円の縦方向の移動速度
int radius;
void setup(){
  size(300,400);
  smooth();
  radius = 10;
  x = new float[10];
  y = new float[10];
  v = new float[10];
  for(int i=0;i<x.length;i++){
    v[i] = random(1,2); // 移動速度を乱数で決める
    y[i] = -random(radius,2*radius);  // 出現タイミングをずらすため
    x[i] = random(radius,width-radius);
  }
}
void draw(){
  background(255);

  for(int i=0;i<y.length;i++){
    y[i] = y[i]+v[i];
    if(y[i]-radius > height){
      x[i] =random(radius,width-radius);// 出現位置をずらす
      y[i] = -random(radius,2*radius);  // 出現タイミングをずらす
    }
    stroke(255,10,10);
    fill(255,10,10);
    ellipse(x[i],y[i],2*radius,2*radius);
  }
}
```

for 命令で繰り返し回数を指定している部分は、配列変数 x,y,v の要素数は全て同じのなので、x.length は、y.length でも v.length でも同じ結果となります。draw 関数の内の for 命令での繰り返しも同じです。

当然、配列は float 型以外でも利用することが出来ます。サンプル 9-5' を書きかえて、color 型の配列変数を使って円の色を変え、int 型の配列変数を使って円の半径を変えてみます。また、表示する円の個数を 50 個に増やしてみます。これを行ったのが、サンプル 9-6 です。色を乱数で変化させるために、colorMode を HSB に変更しています。また、半径の情報を保存している配列 radius には int 型のデータを保存するので、「int(random(5,15);」のように、乱数の値を int 型の値に変更して保存しています。配列名は、変数名と同じように使えますので、x や y のように短い名前ではなく、長い名前を使うことも出来ます。また、円の y 座標の値を決めるために、円の半径の情報を利用しているので、最初に半径の大きさを決めています。

10 個の円の移動（length 使用版）サンプル 9-6

```
float[] x = new float[50];// 円の中心の X 座標
float[] y = new float[50];// 円の中心の Y 座標
float[] v = new float[50];// 円の縦方向の移動速度
color[] cols = new color[50]; // 円の色の情報
int[] radius = new int[50]; // 円の半径の情報

void setup(){
  size(300,400);
  smooth();
  colorMode(HSB,359,99,99);
  for(int i=0;i<x.length;i++){
    radius[i] = int(random(5,15));
    v[i] = random(1,2);
    y[i] = -random(radius[i],2*radius[i]);
    x[i] = random(radius[i],width-radius[i]);
    cols[i] = color(random(360),random(50,100),random(50,100));
  }
}

void draw(){
  background(0,0,99);

  for(int i=0;i<x.length;i++){
    y[i] = y[i]+v[i];
    if(y[i] > height){
      x[i] =random(radius[i],width-radius[i]);
      y[i] = -random(radius[i],2*radius[i]);
    }
    stroke(cols[i]);
    fill(cols[i]);
    ellipse(x[i],y[i],2*radius[i],2*radius[i]);
  }
}
```



配列には、色々な使い方があります。少し複雑な配列の使い方を紹介します。

サンプル 9-7 は、ウインドウ上に表示された円をドラッグするというサンプルです。このサンプルでは、円の中心座標を xBall と yBall という配列に保存しています。同じ添え字番号のデータが同じ円の情報を表しています。つまり、添え字番号で、円を区別していることになります。マウスボタンが押されたときに、マウスの座標と円の中心座標との距離を計算し、その距離が円の半径以下ならば、その円を摑んだと判定しています。円を摑んだと判定したら、その円の添え字番号を int 型変数 pickedID に保存し、boolean 変数 picking に true を代入します。そして、マウスがドラッグされる度に、直前のマウスの位置 (pmouseX と pmouseY) と現在のマウスの位置（mouseX と mouseY）の差がマウスの移動量となるので、摑まれている円の中心座標にマウスの移動量を加えています。マウスボタンが離されたら、円を摑む動作を終了したと判断し、picking の値を false に、pickedID には、あり得ない数字である -1 を代入しています。

picking が true の時には、どれかの円を摑んでおり、どの円かを表す情報は pickedID に保存されています。

この処理は、mousePressed 関数の中に書かれています。

この処理は、mouseDragged 関数の中に書かれています。

この処理は、mouseReleased 関数の中に書かれています。

円を摑んでいる場合、pickedID には、添え字番号が記録さているので、-1 は「あり得ない」値となっています。

円を摑んで移動させる サンプル 9-7

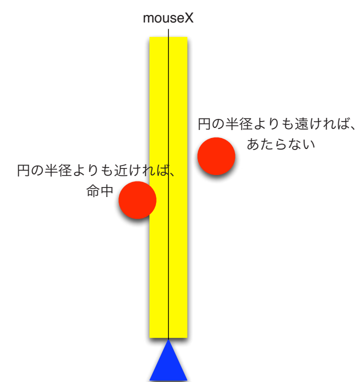
```
int pickedID = -1;
boolean picking=false;
float[] xBall = new float[10];
float[] yBall = new float[10];
color[] cBall = new color[10];
int radius = 10;

void setup(){
  size(400,400);
  colorMode(HSB,359,99,99);
  smooth();
  for(int i=0;i<xBall.length;i++){
    xBall[i] = random(radius,width-radius);
    yBall[i] = random(radius,height-radius);
    cBall[i] = color(random(360),99,99);
  }
}

void mouseDragged(){
  if(picking){
    xBall[pickedID] += (mouseX-pmouseX);
    yBall[pickedID] += (mouseY-pmouseY);
  }
}

void mouseReleased(){
  picking = false;
  pickedID = -1;
}
```

```
void mousePressed(){
  for(int i=0;i<xBall.length;i++){
    if(dist(mouseX,mouseY,xBall[i],yBall[i]) <= radius){
      picking = true;
      pickedID = i;
    }
  }
}
void draw(){
  background(0,0,99);
  for(int i=0;i < xBall.length;i++){
    stroke(cBall[i]);
    fill(cBall[i]);
    ellipse(xBall[i],yBall[i],2*radius,2*radius);
  }
}
```

サンプル 9-8 は、赤い玉を打ち落とすようなサンプルです。mousePressed 関数の中で、ビーム（？）の円への命中判定を行っています。ここでは、簡易的な命中判定を行っています。

ビーム攻撃 サンプル 9-8

```
float[] x;// 円の中心の X 座標
float[] y;// 円の中心の Y 座標
float[] v;// 円の縦方向の移動速度
int radius;
int hit; // 命中回数
PFont font = loadFont("Serif-48.vlw");
// 引数 i で指定された円の位置を初期状態に設定する
void setRandomPosition(int i){
  v[i] = random(1,2); // 移動速度を乱数で決める
  y[i] = -random(radius,2*radius);  // 出現タイミングをずらすため
  x[i] = random(radius,width-radius);
}
// 宇宙船（？）を表示する
void drawShip(){
  stroke(10,10,255);
  fill(10,10,255);
  triangle(mouseX-14,mouseY+20,
           mouseX,mouseY-14,mouseX+14,mouseY+20);
  if(mousePressed){ // マウスボタンが押されていたらビームを描画
    stroke(255,255,10);
    line(mouseX,mouseY-14,mouseX,0);
  }
}
// 得点（今回は単に命中回数）を表示
void displayScore(){
  fill(255);
  textAlign(RIGHT);
  text(hit,width-10,60);
}
```



ビームの命中判定方法

どんなときに、命中判定を誤るかわかりますか？

```
void setup(){
  size(400,400);
  smooth();
  hit = 0;
  radius = 10;
  x = new float[10];
  y = new float[10];
  v = new float[10];
  for(int i=0;i<x.length;i++){
    setRandomPosition(i);
  }
  textFont(font,48);
}

void draw(){
  background(0);

  for(int i=0;i<y.length;i++){
    y[i] = y[i]+v[i];
    if(y[i]-radius > height){
      setRandomPosition(i);
    }
    stroke(255,10,10);
    fill(255,10,10);
    ellipse(x[i],y[i],2*radius,2*radius);
  }
  drawShip();
  displayScore();
}

void mousePressed(){
  for(int i=0;i<x.length;i++){
    // ビームと円との命中判定を行う。
    if(abs(x[i]-mouseX) <= radius && mouseY >= y[i]){
      hit++;
      setRandomPosition(i);
    }
  }
}
```

２つの例では、配列の扱い方に関しては、単純なものを紹介しました。次は、もう少し複雑な配列の操作を伴ったサンプルです。

このサンプル 9-9 では、frameCount という Processing 変数を使用しています。この frameCount 変数は、何回画面を描画したかを保存しています。そのため、描画回数に依存して何かの状況を変化させたいときに、便利な変数です。サンプル 9-9 では、frameCount の値を 360 で割ったときの余りを色相の情報として利用しています。このサンプルでは、100 回分のマウスの位置とその時の色の情報を配列に保存しています。そして、描画が行われる度に、配列に入っている情報

を 1 ずつ移動させています。つまり、

x[99] と y[99] に現在のマウスの位置、cols[99] にその時の色情報
x[98] と y[98] に一つ前のマウスの位置、cols[98] にその時の色情報
x[97] と y[97] に 2 つ前のマウスの位置、cols[97] にその時の色情報
x[96] と y[96] に 3 つ前のマウスの位置、cols[96] にその時の色情報
:
:
:
x[2] と y[2] に 97 つ前のマウスの位置、cols[2] にその時の色情報
x[1] と y[1] に 98 つ前のマウスの位置、cols[1] にその時の色情報
x[0] と y[0] に 99 つ前のマウスの位置、cols[0] にその時の色情報

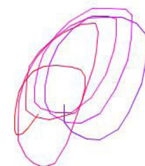
というようにマウスの位置と色の情報を配列に保存しています。そして、過去の情報を一つ前に移動させます。つまり、x[i+1] の値を x[i] に、y[i+1] の値を y[i] に、cols[i+1] の値を cols[i] に代入しています。この処理を i の値を変えながら繰り返し行う必要があるので、for 命令による繰り返し処理で、この処理を実現しています。

new を使って、float 型や int 型の配列変数のための場所を確保した場合には、0 が代入されています。

過去の情報を利用する サンプル 9-9

```
int[] x = new int[100];// マウスの X 座標の値を保存
int[] y = new int[100];// マウスの Y 座標の値を保存
color[] cols = new color[100];// 色の情報を保存

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

void draw(){
  background(0,0,99);
  stroke(255,10,10);
  for(int i=0;i<x.length -1;i++){ // 一つずつ前に移動させる
    x[i] = x[i+1];
    y[i] = y[i+1];
    cols[i] = cols[i+1];
  }
  x[x.length-1] = mouseX; // 最後 (99) に現在の情報を代入する
  y[y.length-1] = mouseY;
  cols[cols.length-1] = color(frameCount % 360,99,99);
  // 配列に保存されている情報を利用して、折れ線を描画する
  for(int i=1;i<x.length;i++){
    stroke(cols[i]);
    line(x[i-1],y[i-1],x[i],y[i]);
  }
}
```

x.length-1 を x.length にすると、エラーになります。なぜだか、わかりますか？



配列の使い方には、1) 配列変数の宣言、2)new を利用して情報を保存する場所を確保、という手順を取ることが一般的です。しかし、配列の要素の記憶する情報が簡単に作れる場合で、その数が少ない場合には、次のよう方法を取ることが出来ます。この場合には、new を利用した場所の確保は必要ありません。

表 9-6 配列の宣言と初期化

宣言と初期化	例
データ型 [] 変数名 = {0 番に保存するデータ , 1 番に保存するデータ , … };	String [] msg = {"Riho", "Tomoyo", "Nene"};

これを利用したサンプル 9-10 を示します。

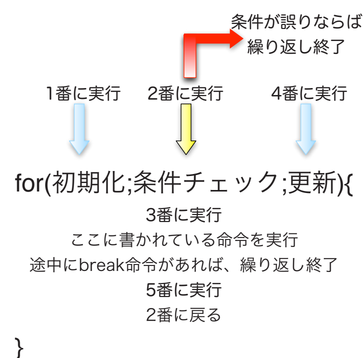
配列の宣言と初期化 サンプル 9-10

```
String [] names = {"Riho",
                   "Tomoyo",
                   "Nene",
                   "Manaka",
                   "Rinko",
                   "Narumi"};
PFont font = loadFont("Serif-48.vlw");

void fadeToWhite(){
  stroke(0,0,99,20);
  fill(0,0,99,20);
  rectMode(CORNER);
  rect(0,0,width,height);
}

void setup(){
  size(400,400);
  colorMode(HSB,359,99,99);
  smooth();
  textFont(font,48);
}

void draw(){
  fadeToWhite();
  // 表示する文字列を選択する
  int idx = int(random(names.length));
  fill(color(random(360),99,99));
  text(names[idx],random(width),random(height));
}
```

ところで、円を掴んで移動させるというサンプル 9-7 では、mousePressed 関数の for 命令の利用した繰り返し部分では、どの円

ちょっと、高度な話題です。

を摑むかが決まれば、最後まで繰り返し処理を実行する必要はありません。繰り返し処理では、途中で繰り返し処理を終了しても良い場合があります。このような機能を実現するために、Processing では break 命令が用意されています。繰り返しの処理の中で break 命令が来ると、一番内側の処理から抜け出します。



break 命令は、Processing をはじめとする C 言語系列のプログラミング言語で利用することが出来ます。

　break 命令を利用して、サンプル 9-7 を書きかえたものが、サンプル 9-11 です。

円を摑んで移動させる (break 版) サンプル 9-11

```
int pickedID = -1;
boolean picking=false;
float[] xBall = new float[10];
float[] yBall = new float[10];
color[] cBall = new color[10];
int radius = 10;

void setup(){
  size(400,400);
  colorMode(HSB,359,99,99);
  smooth();
  for(int i=0;i<xBall.length;i++){
    xBall[i] = random(radius,width-radius);
    yBall[i] = random(radius,height-radius);
    cBall[i] = color(random(360),99,99);
  }
}

void mouseDragged(){
  if(picking){
    xBall[pickedID] += (mouseX-pmouseX);
    yBall[pickedID] += (mouseY-pmouseY);
  }
}

void mouseReleased(){
  picking = false;
  pickedID = -1;
}

void mousePressed(){
  for(int i=0;i<xBall.length;i++){
    if(dist(mouseX,mouseY,xBall[i],yBall[i]) <= radius){
      picking = true;
      pickedID = i;
      break; // もう探す必要がないので、繰り返し処理を終了する
    }
  }
}
```

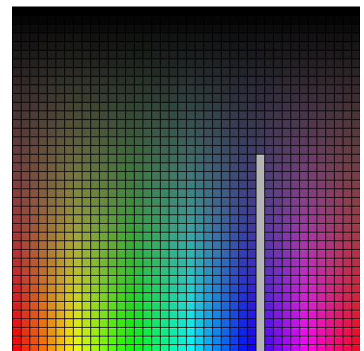
一番内側の繰り返し処理から抜け出します。とはいえ、この例では、「一番内側」というのがピンと来ないですよね。

```
void draw(){
  background(0,0,99);
  for(int i=0;i < xBall.length;i++){
    stroke(cBall[i]);
    fill(cBall[i]);
    ellipse(xBall[i],yBall[i],2*radius,2*radius);
  }
}
```

サンプル 9-12 を見ると break 命令で抜け出す範囲の状況がわかるかも知れません。実行結果とプログラムを見比べて下さい。break 命令が含まれている一番内側の繰り返しを抜けるので、for(int y…){…}の部分から抜け出すだけなので、外側の繰り返し処理 for(int x…){…}の部分は引き続き実行されます。

break の例その 2 サンプル 9-12

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

void draw(){
  background(255);
  stroke(0);
  for(int x=0;x<width;x+=10){
    for(int y=0;y<height;y+=10){
      if((x <= mouseX && mouseX < x+10) &&
         (y <= mouseY && mouseY < y+10)){
        break;
      }
      fill(map(x,0,width-1,0,359),
           map(y,0,height-1,0,99),
           map(y,0,height-1,0,99));
      rect(x,y,10,10);
    }
    // break 命令が実行されると、ここに来る
  }
}
```



この if 命令の条件式はどんな条件を表しているでしょうか？

break 命令は、for 命令による繰り返しだけでなく、while 命令の繰り返し処理からも抜け出ることができます。