情報メディア基盤ユニット用資料（2014 年 7 月 1 日分）

Processing 言語による情報メディア入門

配列（その 2）

神奈川工科大学情報メディア学科　佐藤尚

最大値を見つける

A,B,C,D の 4 つの金の塊があります。この中で一番重たいものを見つけるという問題を考えてみます。ここで使えるのは、天秤ばかりだとします。つまり、2 つのうち、重い方（もしくは軽い方）を見つけることだけが出来るものとします。

この問題は次のようにすると、解くことができます。まず、” 一番重いかも ” という名前をつけた箱を用意します。

1. この箱の中に、A をしまいます。
2. そして、” 一番重いかも ” という箱に入っている金塊と B の重さを比較します。そこで、” 一番重いかも ” の方が、B より重ければ、そのままにします。もし、B の方が重ければ、" 一番重いかも " に入っている金塊を取り出し、B をその中に入れます。
3. 次に、” 一番重いかも ” という箱に入っている金塊と C の重さを比較します。そこで、” 一番重いかも ” の方が、C より重ければ、そのままにします。もし、C の方が重ければ、" 一番重いかも " に入っている金塊を取り出し、C をその中に入れます。
4. 次に、” 一番重いかも ” という箱に入っている金塊と D の重さを比較します。そこで、” 一番重いかも ” の方が、D より重ければ、そのままにします。もし、D の方が重ければ、" 一番重いかも " に入っている金塊を取り出し、D をその中に入れます。

この時に、” 一番重いかも ” という箱に入っている金塊が一番重い金塊となります。これを Proceesing の命令風に書くと次のようになります。

Processing での大きさの比較は、基本的に、大きいか、同じか、小さいかどうかしかわかりません。つもり、天秤ばかりを使って重さを比較しているのと同じ状況になっています。

” 一番重いかも ” の箱の重さは、0 とします。それが嘘くさければ、金塊も同じ箱にないっているとして、比較すれば、箱の重さは無関係となります。

4 個の金塊のなから一番重いもものを見つける 疑似コード 11-1

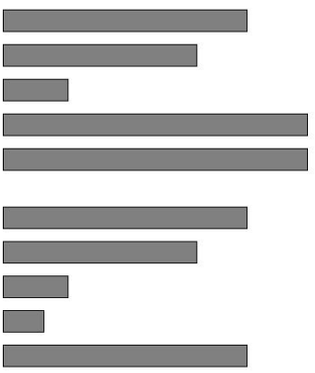
```
一番重いかも = A;
if( 一番重いかも < B){// 一番重いかも > B の時には、何もしなくて良い
  一番重いかも = B;
}
if( 一番重いかも < C){// 一番重いかも > C の時には、何もしなくて良い
  一番重いかも = C;
}
if( 一番重いかも < D){// 一番重いかも > D の時には、何もしなくて良い
  一番重いかも = D;
}
```

処理の内容を、プログラミング言語風に書いたものを、疑似コードと呼ぶことがあります。

この方法を利用して作ったものがサンプル 11-1 です。このサンプ

ルでは、変数 a,b,c の値は乱数で決定し、変数 d の値は mouseX の値を指定します。上から下に横幅がそれぞれ、a、b、c、d の長方形を描き、一番下には、a,b,c,d の中で一番大きな値が横幅になるように長方形を描いています。上の疑似コードの中で、”一番重いかも ” といものは、tentativeMax という変数で表しています。

tentative の意味はわかりますか？



4 つの中から最大値を求める サンプル 11-1

```
float a,b,c,d;

void setup(){
  size(400,200);
  a = random(width);
  b = random(width);
  c = random(width);
}
void draw(){
  background(255);
  stroke(0);
  fill(128);
  d = mouseX;
  float tentativeMax = a;
  if(tentativeMax < b){
    tentativeMax = b;
  }
  if(tentativeMax < c){
    tentativeMax = c;
  }
  if(tentativeMax < d){
    tentativeMax = d;
  }
  rect(0,10,a,25);
  rect(0,50,b,25);
  rect(0,90,c,25);
  rect(0,130,d,25);
  rect(0,170,tentativeMax,25);
}
```

この 4 つ最大値を見つけるという処理を関数として書きかえたものがサンプル 11-2 です。関数名は myMax4 として、float 型の引数 x0,x1,x2,x3 をとる関数としました。

4 つの中から最大値を求める（関数版その 1）サンプル 11-2

```
float a,b,c,d;
void setup(){
  size(400,200);
  a = random(width);
  b = random(width);
  c = random(width);
}
```

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？

```
void draw(){
  background(255);
  stroke(0);
  fill(128);
  d = mouseX;
  float m4 = myMax4(a,b,c,d);
  rect(0,10,a,25);
  rect(0,50,b,25);
  rect(0,90,c,25);
  rect(0,130,d,25);
  rect(0,170,m4,25);
}
float myMax4(float x0,float x1,float x2,float x3){
  float tentativeMax = x0;
  if(tentativeMax < x1){
    tentativeMax = x1;
  }
  if(tentativeMax < x2){
    tentativeMax = x2;
  }
  if(tentativeMax < x3){
    tentativeMax = x3;
  }
// 一番大きな値が tentativeMax に入っているので、この値を返します。
  return tentativeMax;
}
```

関数の引数として与えらえれているので、変数 a ～ d が変数 x0 ～ x3 に変わっています。

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。

ところで、変数を x0 ～ x3 などと表してみると、配列を使ってサンプルが書きかえられそうな気がしてきます。サンプル 11-2 の myMax4 関数の中をジッと見てみると、

```
if(tentativeMax < x 数字 ){
  tentativeMax = x 数字 ;
}
```

ということを繰り返していることに気がつきます。この観察に基づいてサンプル 11-1 を書きかえたものがサンプル 11-3 です。

4 つの中から最大値を求める（配列版その 1）サンプル 11-3

```
float[] x = new float[4];

void setup(){
  size(400,200);
  x[0] = random(width);
  x[1] = random(width);
  x[2] = random(width);
}
```

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。

```
void draw(){
  background(255);
  stroke(0);
  fill(128);
  x[3] = mouseX;
  float tentativeMax = x[0];
  for(int i=1;i<4;i++){
    if(tentativeMax < x[i]){
      tentativeMax = x[i];
    }
  }
  for(int i=0;i<4;i++){
    rect(0,40*i+10,x[i],25);
  }
  rect(0,170,tentativeMax,25);//170 = 40*4+10
}
```

x[1] と tentativeMax の比較から始めたいので、「i=1」となっています。後は、いつもの for 命令を利用した繰り返し処理です。

配列を使うことで、長方形を描く部分も、繰り返し処理を使って書いてみました。この方が、シンプルですよね。

配列を関数の引数にする

配列型も Processing にとっては、単に一つのデータ型です。つまり、Processing にとっては、単純な int 型や String 型などと配列型の扱い方を変える必要はありません。つまり、関数の引数などにも配列型の変数を使用することが出来ます。このことを利用して作成したものがサンプル 11-5 です。

配列変数を引数として渡す場合（仮引数）には、単に変数名だけを書けば OK です。関数を定義する側の引数（実引数）では、配列型であることを明示する必要があります。

つまり、データ型の部分が、「float[] 変数名」などようにします。

4 つの中から最大値を求める（配列版その 2）サンプル 11-5

```
float[] x = new float[4];

void setup(){
  size(400,200);
  x[0] = random(width);
  x[1] = random(width);
  x[2] = random(width);
}

void draw(){
  background(255);
  stroke(0);
  fill(128);
  x[3] = mouseX;
  float m4 = myMax4(x);
  for(int i=0;i<4;i++){
    rect(0,40*i+10,x[i],25);
  }
  rect(0,170,m4,25);
}
```

```
float myMax4(float[] x){
  float tentativeMax = x[0];
  for(int i=1;i<4;i++){
    if(tentativeMax < x[i]){
      tentativeMax = x[i];
    }
  }
  return tentativeMax;
}
```

ところで、サンプル 11-5 の 4 という数字は、配列の要素数を表しています。そこで、サンプル 11-6 は次の様に書きかえることが出来ます。つまり、4 の部分は x.length に書きかえることができます。この書き換えは、些細な書きかえに見えるかもしれませんが、実は非常に大きな書き換えとなっています。このように書きかえるとmyMax4 関数は 4 つの中から最大値を求めるだけでなく、引数としてわたされた配列 x に含まれいている全ての要素の中の最大値を求める関数として、機能するようになります。

4 つの中から最大値を求める（配列版その 3）サンプル 11-6

```
float[] x = new float[4];

void setup(){
  size(400,200);
  x[0] = random(width);
  x[1] = random(width);
  x[2] = random(width);
}

void draw(){
  background(255);
  stroke(0);
  fill(128);
  x[3] = mouseX;
  float m4 = myMax4(x);
  for(int i=0;i<x.length;i++){
    rect(0,40*i+10,x[i],25);
  }
  rect(0,170,m4,25);
}
float myMax4(float[] x){
  float tentativeMax = x[0];
  for(int i=1;i<x.length;i++){
    if(tentativeMax < x[i]){
      tentativeMax = x[i];
    }
  }
  return tentativeMax;
}
```

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。

プログラムの中では、配列の中の最大値や最小値を求めるという処理を行うことがよくあります。そのため、Processing では、配列の中の最大値や最小値を求める関数 max と min が用意されています。

表 10-1 max 関数と min 関数

関数名	意味
max(x)	配列変数 x の要素の最大値を返す関数。x は int 型や float 型の配列です。
min(x)	配列変数 x の要素の最小値を返す関数。x は int 型や float 型の配列です。

引数に配列を取ることが出来るのと同様に、関数の戻り値として配列を利用することができます。サンプル 11-6 の配列を確保し、乱数の値を入れている部分を関数として書きかえたものがサンプル 11-7 です。このサンプルでは、関数 initX の中で、配列を確保しているので、配列変数 x を宣言している場所で、確保する必要がなくなります。

4 つの中から最大値を求める（配列版その 4）サンプル 11-7

```
float[] x;

float[] initX(){
  float[] y = new float[4];
  y[0] = random(width);
  y[1] = random(width);
  y[2] = random(width);
  return y;
}

void setup(){
  size(400,200);
  x = initX();
}

void draw(){
  background(255);
  stroke(0);
  fill(128);
  x[3] = mouseX;
  float m4 = myMax4(x);
  for(int i=0;i<x.length;i++){
    rect(0,40*i+10,x[i],25);
  }
  rect(0,170,m4,25);
}
```

配列を関数の戻り値とする場合は、単に戻り値としたい配列変数を return 命令に書けば OK です。

関数 initX 内で確保 (new) された配列変数 y が戻り値として渡され、それが配列変数 x にコピーされるので、配列変数 x に対して、明示的に new をする必要はありません。

```
float myMax4(float[] x){
  float tentativeMax = x[0];
  for(int i=1;i<x.length;i++){
    if(tentativeMax < x[i]){
      tentativeMax = x[i];
    }
  }
  return tentativeMax;
}
```

このサンプルはサンプル 11-8 のように書きかえることもできます。このサンプル中の関数 initX は配列型の引数 xx をとっています。関数の引数として配列を利用するには、注意をする点があります。それは、関数内で配列変数の要素の値を変更すると、関数を呼び出す時点で仮引数として指定している配列の値も変わってしまう点です。つまり、関数内で配列型引数の値を変更する場合には、注意が必要です。

元の配列の要素の値も変更されることを利用して、プログラムを作る場合もあります。

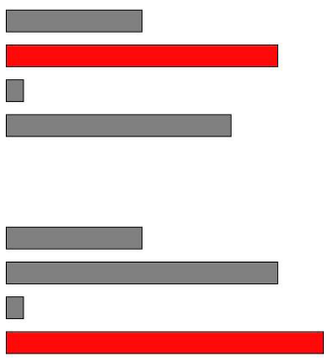
4 つの中から最大値を求める（配列版その 5）サンプル 11-8

```
float[] x = new float[4];
void initX(float[] xx){
  xx[0] = random(width);
  xx[1] = random(width);
  xx[2] = random(width);
}
void setup(){
  size(400,200);
  initX(x);
}
void draw(){
  background(255);
  stroke(0);
  fill(128);
  x[3] = mouseX;
  float m4 = myMax4(x);
  for(int i=0;i<x.length;i++){
    rect(0,40*i+10,x[i],25);
  }
  rect(0,170,m4,25);
}
float myMax4(float[] x){
  float tentativeMax = x[0];
  for(int i=1;i<x.length;i++){
    if(tentativeMax < x[i]){
      tentativeMax = x[i];
    }
  }
  return tentativeMax;
}
```

配列引数 xx の値を書きかえると、元の配列 x の値も書きかえられてしまいます。このサンプルでは、このことを利用して作成しています。

少し別な例題を考えてみます。単に最大値や最小値の値を求めるだけでなく、どの添え字の番号で最大値や最小値になっているかを知りたい場合があります。そこで、このようなことを利用したものがサンプル 11-9 です。このサンプルでは、最大値となっている値の長方形を赤色で塗りつぶします。このサンプル内の findMaxPos 関数は、引数として渡された配列の中で、最大値となっている値の添え字の番号を返す関数です。この関数の中では、直接最大値の候補となる値を保存するのではなく、添え字の値がわかれば、その配列の要素の値がわかることを利用して、最大値の候補となる値が入っている添え字の番号を変数 tentativePos に保存しながら、最大値を求めています。そして、この関数は見つけた最大値の値を返すのではなく、最大値が入っている添え字の番号を関数の戻り値としています。

複数の場所に最大値となる値が入っている場合には、添え字の番号が最も小さいものが戻り値となります。



4 つの中から最大値の入っている添え字を求める サンプル 11-9

```
float[] x = new float[4];
void setup(){
  size(400,200);
  x[0] = random(width);
  x[1] = random(width);
  x[2] = random(width);
}
void draw(){
  background(255);
  stroke(0);
  x[3] = mouseX;
  int maxPos = findMaxPos(x);
  for(int i=0;i<x.length;i++){
    fill(128);
    if(i == maxPos){
      fill(255,10,10);
    }
    rect(0,40*i+10,x[i],25);
  }
}
int findMaxPos(float[] x){
  int tentativePos = 0;
  for(int i=1;i<x.length;i++){
    if(x[tentativePos] < x[i]){
      tentativePos = i;
    }
  }
  return tentativePos;
}
```

多次元配列

今までの配列は、1 つの添え字の番号で要素にアクセスすることが出来ました。用途によっては、複数の添え字の番号で配列の要素にアクセス出来ると便利なこともあります。Processing では、このような用途のために、多次元配列という機能をもっています。2 次元配列の場合には 2 つの添え字で要素へのアクセスができますし、5 次元配列の場合には 5 つの添え字で要素へのアクセスができます。通常は多次元配列としては、2 次元配列を利用する場合が多いように思います。ここでは、2 次元配列について、説明を行います。

たとえば、オセロやマインスイーパーのように盤面の形で情報を保存する必要がある場合などです。

3 次元以上の多次元配列を同じような形で利用できます。

[] の数が配列の次元数を表しています。つまり、[] のペアが 2 つあるので、2 次元配列となっています。

表 10-2 2 次元配列型変数の宣言

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

また、場所の確保は次の様になります。

表 10-3 配列型変数の宣言と場所の確保

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 1][確保するデータ数 2];
float[][] x = new float[10][10];
int[][] radius = new int[10][3];
String [] msg = new String[20][2];

また、2 次元配列の要素へのアクセスは次のようになります。

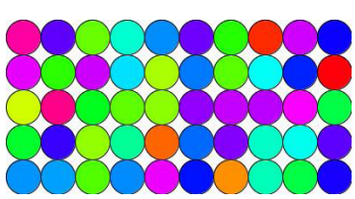
表 10-4 配列の要素へのアクセス

配列のアクセス	例
配列変数名 [番号 1][番号 2]	x[0][1] = random(10); y[1][10] = y[1][0]+v[1][2];

2 次元配列を利用したプログラムがサンプル 11-10 です。

2 次元配列のサンプル（その 1）サンプル 11-10

```
color[][] cols; // 2 次元配列の宣言
void setup(){
  size(400,200);
  colorMode(HSB,359,99,99);
  smooth();
  cols = new color[5][10];//2 次元配列ための場所の確保
  for(int y=0;y<5;y++){
    for(int x=0;x<10;x++){
      cols[y][x] = color(random(360),99,99);
    }
  }
}
```



```
void draw(){
  background(0,0,99);
  stroke(0,0,0);
  for(int y=0;y<5;y++){
    for(int x=0;x<10;x++){
      fill(cols[y][x]);
      ellipse(40*x+20,40*y+20,40,40);
    }
  }
}
```

通常の配列 (1 次元配列) の場合には、length を使って配列の要素数を取り出すことができました。2 次元配列の場合はどのようになるでしょうか？サンプル 11-10 を length を利用して書きかえたものがサンプル 11-11 です。「cols.length」のようにすると、多次元配列の最初の数字で指定できる数の個数が取り出せます。また、「cols[0].length」のようにすると、多次元配列の 2 番目の数字で指定できる数の個数が取り出せます。サンプル 11-11 の場合には、cols.length の値は 5 となり、cols[0].length や cols[1].length の値は 10 となります。

2 次元配列のサンプル（その 2）サンプル 11-11

```
color[][] cols;

void setup(){
  size(400,200);
  colorMode(HSB,359,99,99);
  smooth();
  cols = new color[5][10];
  for(int y=0;y<cols.length;y++){
    for(int x=0;x<cols[0].length;x++){
      cols[y][x] = color(random(360),99,99);
    }
  }
}

void draw(){
  background(0,0,99);
  stroke(0,0,0);
  for(int y=0;y<cols.length;y++){
    for(int x=0;x<cols[0].length;x++){
      fill(cols[y][x]);
      ellipse(40*x+20,40*y+20,40,40);
    }
  }
}
```

[] の数が配列の次元数を表しています。つまり、[] のペアが 2 つあるので、2 次元配列となっています。

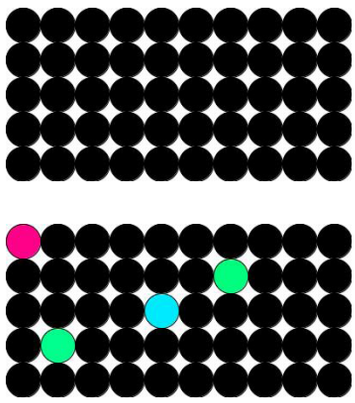
Processing で は、cols[0].length と cols[1].length の 値が異なるような多次元配列を使用することが出来ます。このような多次元配列のことをジャグ配列 (jagged array) と呼びます。

多次元配列の場合も、要素の値を指定しての配列の宣言を行うことができます。これを行ったものが、サンプル 11-12 です。このサ

ンプルでは、マウスをクリックすると、その場所の円の色が cols 変数に保存されているものになるというものです。どの場所の円がクリックされたかを boolean 型 2 次元配列 flipped に保存します。最初の状態では、クリックされた円が無い状態なので、flipped 配列の要素の値は、全て false になっています。

2 次元配列のサンプル（その 3）サンプル 11-12

```
color[][] cols;
// 2 次元配列の宣言と初期化を同時に行う
boolean[][] flipped = {{false,false,false,false,false,
                        false,false,false,false,false},
                       {false,false,false,false,false,
                        false,false,false,false,false},
                       {false,false,false,false,false,
                        false,false,false,false,false},
                       {false,false,false,false,false,
                        false,false,false,false,false},
                       {false,false,false,false,false,
                        false,false,false,false,false}};
void setup(){
  size(400,200);
  colorMode(HSB,359,99,99);
  smooth();
  cols = new color[5][10];
  for(int y=0;y<cols.length;y++){
    for(int x=0;x<cols[0].length;x++){
      cols[y][x] = color(random(360),99,99);
    }
  }
}
void draw(){
  background(0,0,99);
  stroke(0,0,0);
  for(int y=0;y<cols.length;y++){
    for(int x=0;x<cols[0].length;x++){
      if(flipped[y][x]){
        fill(cols[y][x]);
      }else{
        fill(0,0,0);
      }
      ellipse(40*x+20,40*y+20,40,40);
    }
  }
}
void mouseClicked(){
  flipped[mouseY/40][mouseX/40] = true;
}
```



2 次元配列の宣言と初期化を同時に行う場合には、サンプル 11-12

の flipped 配列の宣言と初期化のように行います。

表 10-5 配列型変数の宣言と初期化

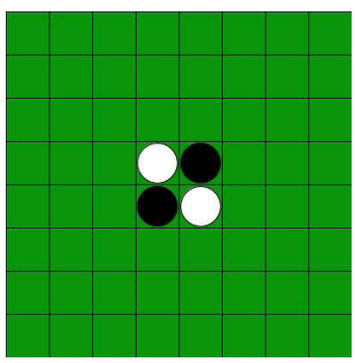
宣言方法

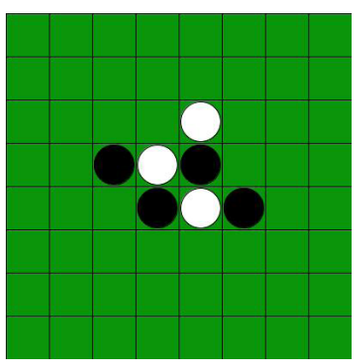
データ型 [][] 配列変数名 =
{{[0][0] に入れるデータ ,[0][1] に入れるデータ ,…},
{[1][0] に入れるデータ ,[1][1] に入れるデータ ,…},
{[2][0] に入れるデータ ,[2][1] に入れるデータ ,…},
…
};
int[][] radius = new int[10][3];
String [] msg = new String[20][2];

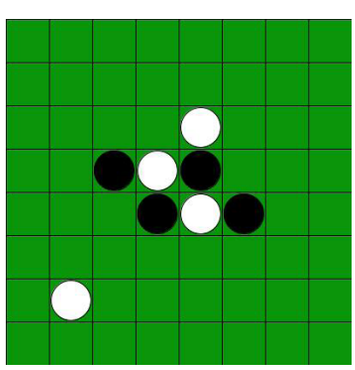
2 次元配列のサンプル（その 2）サンプル 11-13

```
int EMPTY = 0;
int BLACK = 1;
int WHITE = 2;
int[][] board = new int[8][8];
boolean turn = true;

void setupBoard(int[][] board){
  for(int i=0;i<board.length;i++){
    for(int j=0;j<board.length;j++){
      board[j][i] = EMPTY;
    }
  }
  board[3][3] = WHITE;
  board[4][4] = WHITE;
  board[4][3] = BLACK;
  board[3][4] = BLACK;
}
void displayBoard(int[][] board){
  background(10,150,10);
  rectMode(CENTER);
  stroke(0);
  for(int i=0;i<board.length;i++){
    for(int j=0;j<board.length;j++){
      noFill();
      rect(25+50*i,25+50*j,50,50);
      if(board[j][i] == BLACK){
        fill(0);
      }else if(board[j][i] == WHITE){
        fill(255);
      }
      if(board[j][i] != EMPTY){
        ellipse(25+50*i,25+50*j,46,46);
      }
    }
  }
}
```







```
void setup(){
  size(400,400);
  smooth();
  setupBoard(board);
}

void draw(){
  displayBoard(board);
}

void mouseClicked(){
  int x = mouseX/50;
  int y = mouseY/50;
  if(board[y][x] == EMPTY){
    if(turn){
      board[mouseY/50][mouseX/50] = BLACK;
    }else{
      board[mouseY/50][mouseX/50] = WHITE;
    }
    turn = !turn;
  }
}
```