プログラミングA　（山内クラス）　第８回

――― 今日の目標 ―――

教科書第７章「配列」の7.2～7.5を学ぶ。　次の問に答えられるようにする。

― 配列の値の初期化を書けるようになる。

― 配列の値の最大値・最小値を求めるなど、繰り返し（forループ）を使った配列要素ごとの処理をできるようになる。

――

１．　配列の初期化　（教科書 p 190）

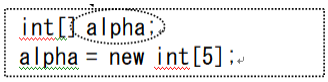
配列の内容に初期値を与えるときに、宣言をするのと同時に、下記のように要素の初期値を与えることができる。

`int[] hairetsu = { 1, 3, 5, 7, 9 };`

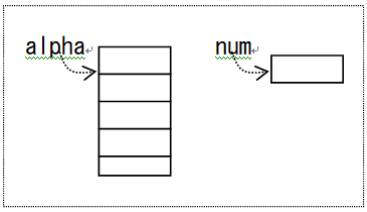
初期値の個数（何個書くか）が配列の大きさを決めることに注意。上記の例だと `hairetsu = new int[5];` に相当

２．　「配列変数」の考え方、　配列の参照、　配列のコピー　（教科書７．５節）

配列変数　とは　配列に付ける「名前」のこと



この「名前」は、　配列全体への「参照」　（＝「名札」のようなもの）　⇒　図



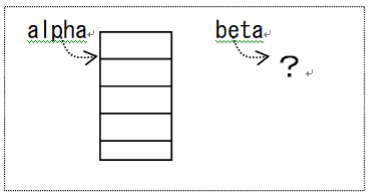
`int alpha[];　alpha = new int[5];`　　のようになる。これは、`int num;　num = 3;` と同じ枠組み・原理。

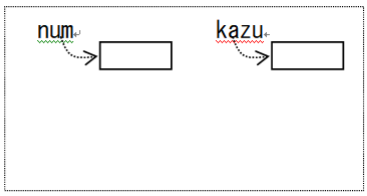
配列から配列へ代入するとどうなるか？

```
int alpha[];    alpha = new int[5];                    int num;    num = 3;

int beta[];                                            int kazu;

beta = alpha;   alphaをbetaに代入                      kazu = num;    numをkazuに代入
```





配列をコピーしたい（同じ内容の配列の実体を、別に作りたい）ときはどうするか？

_____

_____

教科書p 197 の囲み記事　「配列変数の特徴」

第３章で学んだint型などの変数　（　基本型の変数　）　が、値を格納するハコ　「そのもの」　を表すのに対して、

配列変数は値を格納するハコが　「メモリ上のどこに存在しているか」　という、配列の「場所」を表す。

このような種類の変数のことを、基本型の変数に対して、「参照型の変数」　と呼ぶ。

参照型の変数には、配列変数の他に、第８章以降で学ぶ、クラス型、インターフェイス型などがある。

（山内の脱線）　この違いは、他のプログラム言語でも共通に存在する概念。

配列のように、中身が１つでない変数は、どうしても、

- 名前　～～　集まり全体を代表する変数名　　と
- なかみの１つ１つ（のあつまったもの）　　と　　を区別して扱いたくなる。（特にコピーするとき）
- 名前だけをコピーする（なかみはコピーしない）コピーを　「浅い」　コピー　（Shallow Copy）、
- なかみもコピーする（別に新しく、データを全部コピーする）コピーを　「深い」　コピー（Deep Copy）と呼ぶ

「浅い」コピーのことを、「参照渡し」と呼ぶこともある。

- C言語：　名前のコピーは浅いコピーで、深いコピーが必要なときは要素ごとにコピーするかmemcpyを使う
- JavaScript言語：　名前のコピーは浅いコピーで、深いコピーが必要なときはたとえば空の配列とconcatする
- Python言語：　名前のコピーは浅いコピーで、深いコピーが必要なときは全要素を指定して代入かcopyメソッド

３．　配列の長さ（大きさ）を知る　（　教科書７．６　p 198　）

配列変数　alpha があったとき、その長さを知るには、`alpha.length`とすればよい。

```
int alpha[] = new int[5];
for (j=0; j<alpha.length; j++)  {
    println(alpha[j]);
}
```

４．　最大値・最小値　　（教科書には載っていない、オリジナルネタ）

図のようなデータがあるとき、その中の最大値　（または最小値）　を求めよ

a[0]	3
a[1]	5
a[2]	4
a[3]	2

人間なら　（４つぐらいならば）　一目で分かる。　コンピュータは、2つの数の大小を比べることしかできない。 どうするか？

（仕方がないので）要素を1つずつ見てゆくことにする。

はじめに、変数maxの値を、うんと小さい値（あり得ないぐらい小さい値）にセットしておく。

次に、配列aの1つ1つの要素（全部）と比較し、

もし今までわかっているmaxの値よりa[i]の値が大きければ、maxの値をa[i]の値に置き換える

（大きくなければ）　何もしない

全部比較し終わると、変数maxの中にはaの全部の要素より大きいか等しい要素が残る。

自分がコンピュータになったつもりで、試してみよう。

まず、max を a[0] にセットする　～～　maxの初期値としてa[0]を使う

maxとa[1] とを比較する　　～～～　maxは3、a[1]は5、max<a[1]なので、maxの値を5に置き換える

_____

_____

aの全部の要素と比較し終わった。　maxに残っている値は（　　）だった。　これは最大値である。

［演習１］

①　実際にJCPadでプログラムを書いて、試してみよう

②　最小値も求めてみよう

```
int a[] = { 3, 5, 4, 2 };
int max = ..... ;
for (int i=0; i<4; i++)  {
      .....
      .....
      .....
      .....
}
```

５．　配列を使って、数で遊ぶ　（教科書にはありません）

＜乱数列の発生＞

乱数列とは、ランダムな値を並べた数列。　値の範囲をたとえば0～1の小数とすると、イメージとしては

[0.1454829416146144, 0.2912268034112202, 0.7780804560637977, 0.3857684655923168, ..., 0.8637267708712996]

のようなもの。

プログラム言語では、乱数を1つずつ発生して乱数列を作る仕組（関数、メソッド）が用意されている。 Javaの場合、`Math.random()`を使うと乱数列が発生できる。

```
class random0 {
    public static void main(String[] args) {
        for (int i=0; i<10; i++) {
            System.out.println(Math.random());
        }
    }
}
```

⇒　図のプログラムをJCPadで試してみよ　⇒　　範囲0～1の乱数（小数）が表示される。

では、乱数が、「本当にランダムか」を疑ってみよう。　区間を0.1ずつに切って、発生した乱数がどの区間に落ちるかを数えてみよう。　いろいろな方法が思いつく。　たとえば、`if`文を使って、得られた乱数`x`が0～0.1に入るか、0.1～0.2に入るか、…　0.9～1に入るか、とチェックしておき、

もし　0～0.1に入ればcount[0]を１つ増やし、

0.1～0.2に入ればcount[1]を１つ増やし、

。。。　。。。

0.9～1に入ればcount[9]を１つ増やし、

のように考えて、1000個の乱数についてどの範囲に落ちるかを数えてみることができる。

```
for (int i=0; i<10; i++){
    count[i] = 0;
}
for (int i=0; i<10; i++) {
    x = Math.random();
    if ((0.0<=x) and (x<=0.1)) {
        count[0] = count[0] + 1;
    } else if ( ... ... ) {
        count[1] = count[1] + 1;
    } else if
    ... ... ...
    }
}
```

でも、if文を10個も書くのはいやだ、となると、少々面倒なトリックを考える。

得られた乱数は0～1の範囲だが、これを10倍しておいて、0.0～10.0の範囲にしておく。さらに、この10倍した数を無理やり`int`型に変換すると、整数部分だけが取り出せる。（つまり0～9の範囲の整数が得られる。）

```
for (int i=0; i<10; i++) {
    int j = (int)(Math.random()*10.0);
    count[j] = count[j] + 1;
}
```

小数点以下は切り捨てなので、10になるのは元の乱数がちょうど10.0の時に限るが、これは確率的にほとんど起こらない。

このトリックで、0～9のランダムな整数が発生できたことになる。

これをカウントするのは、if文を使わなくても、count配列の添え字に直接与えてしまえばよい。　得られている整数の乱数をxとすれば、配列countに対してcount[x] を1増やすようにすれば、値xが3ならcount[3]が１増やされることになる。

［演習2］　上記の原理を使って乱数の発生頻度を数えるプログラムを作れ。　区間は0～1を0.1ずつに区切るものとし、1000個の乱数を発生させて、カウントした上で、カウント結果を表示せよ。　得られたカウント数について考察せよ。（いくつになるはずか？　どうしたらなるか？）

［おまけの演習3］　今度は、生の乱数ではなく、乱数をN個集めて平均値を取り、そのN個の平均値をxとして、その平均値をとるプロセスをM回繰り返して出現回数を数えてみよ。（全体にはNxM回乱数を発生させることになる。）

Nの値が1だとどうか？　10だとどうか？　100だとどうか？

M=1000個の平均値の出現頻度の分布を出し、得られた数字について考察せよ。

```
for (int i=0; i<M; i++){
    double s = 0.0;
    for (int k=0; k<N; k++) {
        s = s + Math.random();
    }
    int j = (int)(s/N*10.0);
    count[j] = count[j] + 1;
}
```

［追加の課題４］

右のような２次元の座標で、範囲　０≦ｘ≦１、　０≦ｙ≦１　を考える。

この正方形の中に、ランダムに点を落とす。

それには、乱数を２回発生させ　（つまり　Math.random()　を２回呼び出す）

それぞれを　ｘ　と　ｙ　に代入する。

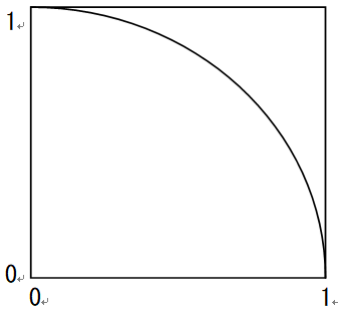
このとき、点は一様な密度で、正方形の中に分布しているはずである。

そこで、点の座標　（ｘ，ｙ）　によって、点を２種類に区分する。　右図のような　１／４円を描き、それの内側であるか、外側であるかによって、区別する。　区別は、　x^2　＋　y^2　≦　１　であるかどうかによって判定すればよい。
１／４円の内側に落ちた点の数を数える。

もし、点が正方形の中に一様に落ちるのであれば、その全部の点の個数の中で１／４円の内側に落ちた点の個数の割合は、面積の割合になるはずである。　つまり正方形の面積が１×１＝１であるから、

ｒ　＝　（円の内側の点の個数）／（全部の点の個数）

は１／４円の面積になるはずである。　４　×　ｒ　を計算すれば、半径１の円の面積、つまりπになるはずである。



これを、プログラムによって実際に計算してみよう。

計算の精度は、ランダムに生成する点　（ｘ，　ｙ）の個数に依存する。　１万点、１０万点、１００万点、のようにだんだんに増やしてみて、結果を並べてみよう。