

プログラミング基礎 I

第5回 配列

九州産業大学 理工学部

pk@is.kyusan-u.ac.jp

内容

- 配列の宣言、生成
- 配列の初期化
- 配列と繰り返し
- 配列の練習

同じ型の変数を複数扱うには？

(例) テストの点数を管理する

- 5人のテスト結果について、それぞれの点数と全員の平均点を計算・表示、テスト結果から可否を判定したい
 - 整数型の変数 `score0`, `score1`, `score2`, `score3`, `score4` を宣言し、右の表の値で初期化
 - 宣言した変数をそれぞれ、整理番号0の学生の点数、整理番号1の学生の点数、...と考える
 - 5名の点数の平均値を計算して出力し改行
 - まず合計値を計算、合計値を人数で割って平均値を計算する
 - 5名それぞれについて、平均点以上であれば"合格"、そうでなければ"再テスト"と表示

番号	変数名	点数
番号0	score0	77
番号1	score1	12
番号2	score2	45
番号3	score3	76
番号4	score4	38

平均点は49.6点
番号0:77点 合格
番号1:12点 再テスト
番号2:45点 再テスト
番号3:76点 合格
番号4:38点 再テスト

変数宣言

- 0番～4番までの5人分のテスト結果を格納する変数を宣言

```
int score0 = 77;  
int score1 = 12;  
int score2 = 45;  
int score3 = 76;  
int score4 = 38;
```

- 人数が増えたら、score5, score6,...と番号を増やして対応

5人分の平均点数を計算

- 合計点を格納する変数を宣言

```
int sum = 0;  
sum += score0;  
sum += score1;  
sum += score2;  
sum += score3;  
sum += score4;
```

- 平均点を計算（合計点を人数で割る）
 - （整数型/整数型 にならないようにキャストする）

```
double average = (double)sum / 5;
```

型変換（キャスト） 教科書P31,37

- 値（数値や変数に代入されている値）を他の型に変換できる
- 数値型のキャスト
 - (型名)値;
`int x = 5;`
`int y = x / 2;`
`System.out.println( y );` // 出力結果：2
`double z = (double)x / 2;` // 整数型のxをdouble型に変換
`System.out.println( z );` // 出力結果：2.5

小数型 / 整数型 とすると、小数点以下の計算もできるようになる

平均点以上か判定

- もしも、平均点以上であれば"合格"、
そうでなければ"再テスト"と表示

```
System.out.print("番号0:" + score0 + "点 ");  
if(score0 >= average) {  
    System.out.println("合格");  
} else {  
    System.out.println("再テスト");  
}  
System.out.print("番号1:" + score0 + "点 ");  
if(score1 >= average) {  
    System.out.println("合格");  
} else {  
    System.out.println("再テスト");  
}
```

```
int score0 = 77;  
int score1 = 12;  
int score2 = 45;  
int score3 = 76;  
int score4 = 38;
```

```
//平均値を計算して表示
```

```
//5人分の点数の合計
```

```
int sum = 0;
```

```
sum += score0;
```

```
sum += score1;
```

```
sum += score2;
```

```
sum += score3;
```

```
sum += score4;
```

```
//sumを人数(5)で割って平均値を計算
```

```
double average = (double)sum / 5;
```

```
System.out.println("平均点は" + average + "点");
```

```
System.out.print("番号0:" + score0 + "点 ");
```

```
if(score0 >= average) {
```

```
    System.out.println("合格");
```

```
} else {
```

```
    System.out.println("再テスト");
```

```
}
```

```
//同様に、番号4まで表示していく
```

更にたくさんの同じ型の変数を扱うには？

(例) テストの点数を管理する

- 50人分が増えたら・・・

- 50人のテスト結果について、それぞれの点数と全員の平均点を計算・表示、テスト結果から可否を判定したい

1. 無理ではないが、とても大変
何とか出来たとして、500人分と言われたら？

番号1の子生の点数、...と考える

- 50名の点数の平均値を計算して出力し改行

1. 50名それぞれについて、平均点以上であれば"合格"、そうでなければ"再テスト"と表示

番号	変数名	点数
番号0	score0	77
番号1	score1	12
番号2	score2	45
番号3	score3	76
番号4	score4	38
番号5	score5	69
番号6	score6	74
番号7	score7	25
番号8	score8	31
番号49まで続く・・・		
番号9	score9	100

配列：同じ型のデータを番号をつけてまとめて扱う

同種の沢山のデータを扱うときに

変数を並べる（これまでの変数（原始型）でできること）

10	20	1	90
a	b	c	d

…変数が増えると扱いが困難

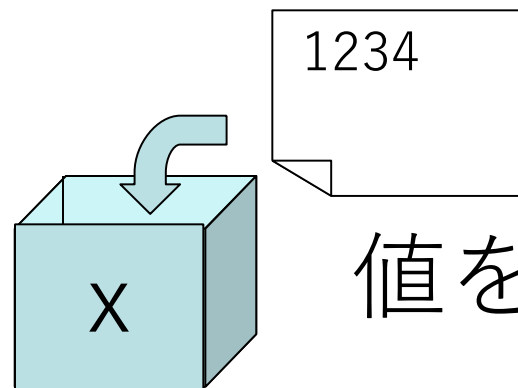
配列：同じ型のデータを番号をつけてまとめて扱う

score	10	20	1	90
	score[0]	score[1]	score[2]	score[3]

scoreという名前の大きな箱を用意して、中の小箱に番号を付けて管理

配列と原始型

原始型の変数

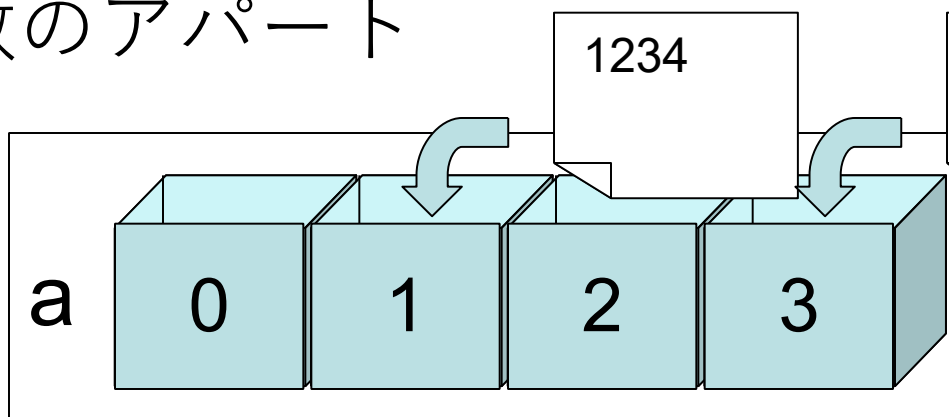


値を格納

$x = 1234;$

変数のアパート

配列



同じ型の複数の値を格納

$a[1] = 1234;$

$a[3] = 987;$

a アパートの
1号室に1234を
3号室に987を

配列の宣言、生成 教科書P76

- 原始型の場合

- 宣言 **型 変数名**;

int **x**; // int型の変数xを宣言



x

- 配列の場合

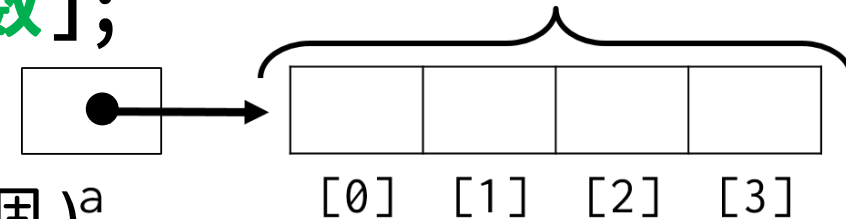
- 宣言 **要素の型 [] 配列名**; 要素:1つ1つの箱

int [] **a**; // int型の配列 a を宣言

- 要素の生成 **配列名** = new **要素の型**[**要素数**];

a = new **int**[**4**]; // 生成(要素 4 個)

b = new **int**[**5000**]; // 生成(要素5,000個)^a



配列の要素へのアクセス

- 原始型の変数へのアクセス

変数名

x に10を代入

x = 10;

x の値を画面に表示

System.out.println(**x**);

10

x

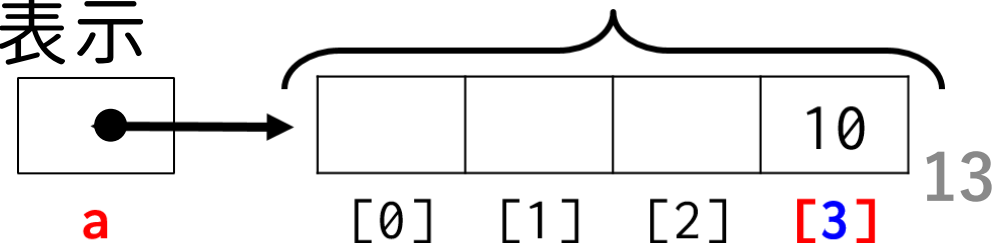
- 要素へのアクセス **配列名**[**添え字**] 添え字：要素の番号(0から)

配列 **a** の 添え字**3**の要素 に10を代入

a[**3**] = 10;

配列 **a** の 添え字**3**の要素 の値を画面に表示

System.out.println(**a**[**3**]);



配列の宣言と生成 要素数

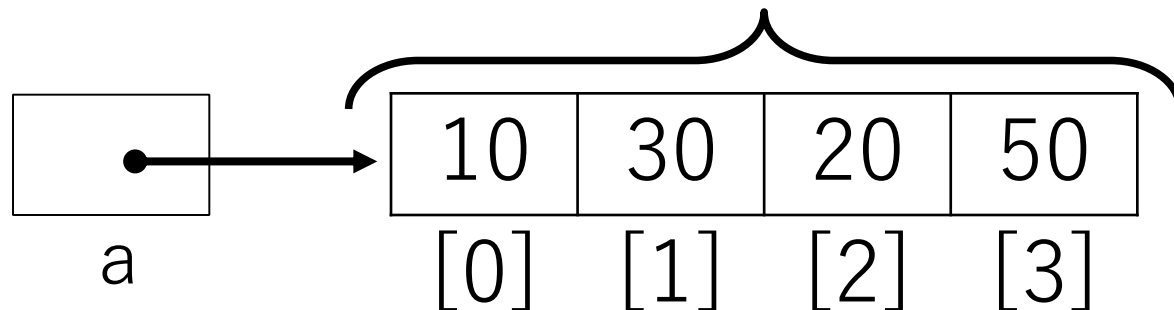
- 宣言と生成 `要素の型 [] 配列名 = new 要素の型[要素数];`

`int [] a;` と `a = new int [4];` は？

`int [] a = new int [4];`

要素の数 **配列名.length**

`a.length` // 配列aの要素の数



配列の初期化 教科書P80

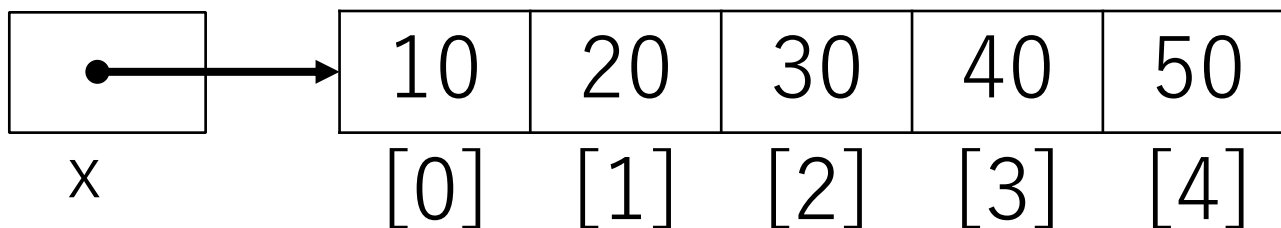
配列の宣言時の初期化

要素の型 [] 配列名 = { 式1, 式2, ... } ;

例 int型の配列 xを宣言：初期値10,20,30,40,50

int [] x = {10, 20, 30, 40, 50};

// 初期化の場合はnewはいらない



穴埋め：配列の宣言

- 配列の宣言

型 [] 配列名;

int [] score; //int型の配列scoreを宣言

String [] names; //String型の配列namesを宣言

boolean [] checks; //boolean型の配列checksを宣言

穴埋め：要素の生成と代入

- 要素の生成と代入

配列名 = new 型[要素数];

score = new **int**[**5**]; //scoreに**int**型の**5**つの要素を生成して代入

names = new **String**[**3**]; //namesに**String**型の**3**つの要素を生成して代入

checks = new boolean[500]; //checksにboolean型の500個の要素を生成して代入

穴埋め：配列の初期化

配列の宣言時の初期化

型 [] 配列名 = { 式1, 式2, ...} ;

```
// int型の配列 xを宣言：初期値 10,20,30,40,50  
int [] x = {10, 20, 30, 40, 50};
```

```
// String型の配列 namesを宣言：初期値 "Aさん", "Bさん", "Cさん"  
String [] names = {"Aさん", "Bさん", "Cさん"};
```

穴埋め：要素へのアクセス

配列の要素へのアクセス：**配列名**[**添え字**]

```
score[0] = 5;    //scoreの0番目の要素に5を代入
score[3] = 15;   //scoreの3番目の要素に15を代入
names[2] = "hanako"; //namesの2番目の要素に"hanako"を代入
score[i] = 5;    //scoreのi番目の要素に5を代入
names[x] = "tarou"; //namesのx番目の要素に"tarou"を代入
score[5] = score[0]; //scoreの5番目の要素にscoreの0番目の要素を代入
```

要素をコンソールに出力

要素を出力：System.out.println(配列名[添え字]); 等

System.out.println(score[0]); //scoreの0番目の要素を表示

System.out.println(names[i]); //namesのi番目の要素を表示

System.out.println(numbers[x]); //numbersのx番目の要素を表示

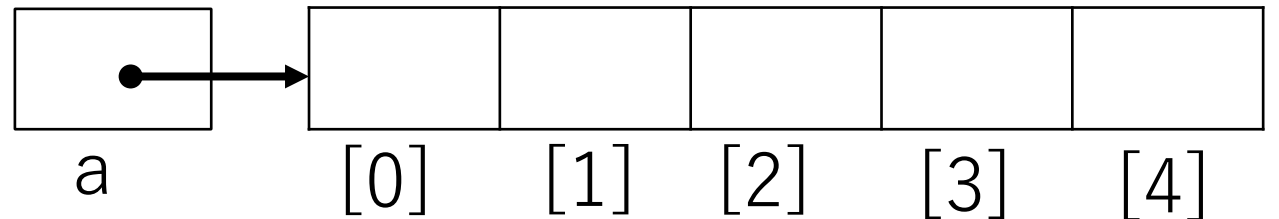
//scoreのi番目の要素を「score[0]:10」の様に表示

System.out.println("score[" + i + "]: " + score[i]);

穴埋め：実行結果の予想

```
int i = 3;  
int[] a;  
a = new int [5];  
a[0] = 5; a[1] = a[0]+ 3; a[2] = 4;  
a[3] = 9; a[4] = a[1]; a[2]++; a[i] = a[1]+3;  
System.out.print(a[0] + " ");  
System.out.print(a[1] + " ");  
System.out.print(a[2] + " ");  
System.out.print(a[3] + " ");  
System.out.print(a[4] + " ");  
System.out.println();
```

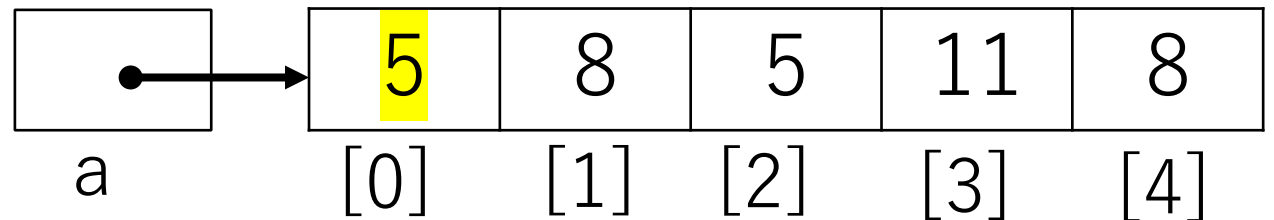
実行結果を予想して図に値を書き入れよう！



穴埋め：実行結果の予想

```
int i = 3;  
int[] a;  
a = new int [5];  
a[0] = 5; a[1] = a[0]+ 3; a[2] = 4;  
a[3] = 9; a[4] = a[1]; a[2]++; a[i] = a[1]+3;  
System.out.print(a[0] +" ");  
System.out.print(a[1] +" ");  
System.out.print(a[2] +" ");  
System.out.print(a[3] +" ");  
System.out.print(a[4] +" ");  
System.out.println();
```

実行結果を予想して図に値を書き入れよう！



(例) テストの点数を配列で管理する

- 最初の問題を配列で書き換え

- 5人のテスト結果について、それぞれの点数と全員の平均点を計算・表示、テスト結果から可否を判定したい

1. 整数型の配列 scores宣言し、右の表の値で初期化

2. 5名の点数の平均値を計算して出力し改行

- まず合計値を計算、合計値を人数で割って平均値を計算する

3. 5名それぞれについて、平均点以上であれば"合格"、そうでなければ"再テスト"と右の結果の様に表示

番号	添え字	点数
番号0	0	77
番号1	1	12
番号2	2	45
番号3	3	76
番号4	4	38

平均点は49.6点
番号0:77点 合格
番号1:12点 再テスト
番号2:45点 再テスト
番号3:76点 合格
番号4:38点 再テスト

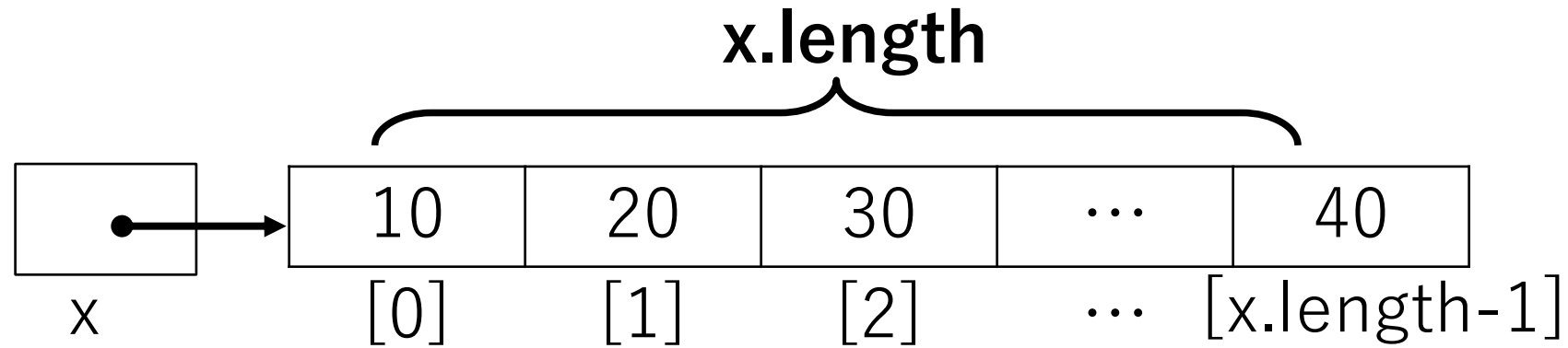
```
public class ScoreArray {  
  
    public static void main(String[] args) {  
        // TODO 自動生成されたメソッド・スタブ  
        int [] scores = {77, 12, 45, 76, 38};  
        int sum = 0;  
        sum += scores[0];  
        sum += scores[1];  
        sum += scores[2];  
    }  
}
```

変数の宣言、初期化は楽になったけど・・・

配列を使っても、同じようなことを**繰り返し**何度も書かなければいけないことに変わりはない？

```
        System.out.println("再テスト");  
    }  
    //番号5まで点数と評価結果を表示  
}  
}
```

配列とfor文 教科書P82



配列 `x` の要素をすべて表示（先頭から長さ分繰り返す）

```
for (int i = 0 ; i < x.length ; i++) {  
    System.out.print( x[i] ); // 添え字は変数 i  
}
```

```
System.out.println();
```

for文を使わないと
`System.out.println(x[0]);`
`System.out.println(x[1]);`
...

穴埋め：配列とfor文(配列の要素をすべて出力)

配列 x の要素をすべて表示（先頭から長さ分繰り返す）

```
for (int i = 0 ; i < x.length; i++ ) {  
    System.out.print( x[i] ); // 添え字は変数 i  
}  
System.out.println();
```

配列 score の要素をすべて表示（先頭から長さ分繰り返す）

```
for (int a = 0; a < score.length; a++ ) {  
    System.out.print( score[a] ); // 添え字は変数 a  
}  
System.out.println();
```

配列の要素の合計

- 配列の要素を合計する考え方：全部足して足した回数で割る

```
int sum2 = 0;
sum2 += scores[0];
sum2 += scores[1];
sum2 += scores[2];
sum2 += scores[3];
sum2 += scores[4];
double average = (double)sum2 / 5;
```

合計値を入れておく変数を宣言

配列の要素を全て合計

要素数で割る

小数型/整数型として、結果の小数点が切り捨てられないようにする
(整数型 / 整数型 の結果は整数型になる)
• (変数の型)値 は値の**キャスト**という

合計の処理は配列の添え字が変わっているだけ

変数を使って簡便に

- 配列の要素を合計する考え方

```
int sum = 0; int i = 0;  
sum += scores[i]; i++;  
sum += scores[i]; i++;  
sum += scores[i]; i++;  
sum += scores[i]; i++;  
sum += scores[i];  
double average = sum / 5.0;
```

合計値を入れておく変数を宣言

配列の要素を全て合計

要素数で割る(割る数を小数型としても良い)

配列の添え字を変数*i*で表すと、
合計する処理はコピー＆ペーストで増やしていける

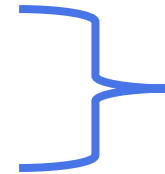
ただし、配列の要素数が変わるとプログラムも修正しないといけない

繰り返しを使って配列の要素を合計

- 配列の要素を合計する考え方

```
int scoreSum = 0
for(int i = 0; i < scores.length; i++){
    scoreSum += scores[i];
}
```

合計値を入れておく変数を宣言



配列の要素を全て合計

```
double average = scoreSum / (double) scores.length;
```

要素数で割る

for文で繰り返せば、
簡単に記述できる

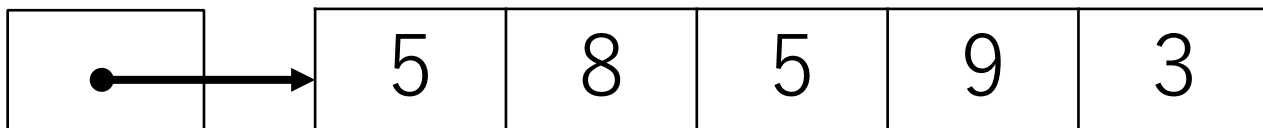
配列の要素数が変わってもプログラムを変更しなくて良い

配列の要素の合計

```
int sum = 0;
for(int i = 0; i < a.length; i++) {
    sum += a[i];
}
System.out.println("合計は"+sum);
```

配列名

変数iの値	配列a[i]の値	sumの値
0	5	5
1	8	13
2	5	18
3	9	27
4	3	30



a



sum

練習問題 2 : PK05p1

(PK05p0を配列と繰り返しで書き換える)

- PK05p0を配列で書き換え
 - 5人のテスト結果について、それぞれの点数と全員の平均点を計算・表示、テスト結果から可否を判定したい
- 1. 整数型の配列 scores宣言し、右の表の値で初期化
- 2. 5名の点数の平均値を計算して出力し改行
 - for文で合計値を計算、合計値を**配列の要素数**で割って平均値を計算する
- 3. 5名それぞれについて、平均点以上であれば"合格"、そうでなければ"再テスト"と右の結果の様に表示
 - 繰り返しで、 i番目の要素についてそれぞれ点数を表示して合格or再テストと表示

番号	添え字	点数
番号0	0	77
番号1	1	12
番号2	2	45
番号3	3	76
番号4	4	38

平均点は49.6点
番号0:77点 合格
番号1:12点 再テスト
番号2:45点 再テスト
番号3:76点 合格
番号4:38点 再テスト

p5のプログラムと比べて短く、見やすいプログラムに

```
int [] scores = {77, 12, 45,76,38};
int ????????? = 0;
for(int i = 0; i < scores.length; i++) {
    scoreSum += scores[i];
}
double scoreAve = (double)???????? / scores.????????;
System.out.println("平均点:" + ????????? + "点");
for(int i = 0; i < ?????????; i++) {
    System.out.println(i);
    if(scores[i] >= ?????????) { //scoresのi番目が平均点以上なら
        System.out.println("番号" + ? + ":" + ??????[i] + "点 合格");
    } else {
        System.out.println("番号" + ? + ":" + ????????? + "点 再テスト");
    }
}
```

要素数が変わっても・・・

P5のプログラムと違い、配列の要素数が変わってもプログラムを変える必要はない

```
int [] scores = {77, 12, 45 ,76 ,38, 88, 44, 66, 74, 20, 75, 98, 10, 5, 68, 31, 45};
int ????????? = 0;
for(int i = 0; i < scores.length; i++) {
    scoreSum += scores[i];
}
double scoreAve = (double)???????? / scores.????????;
System.out.println("平均点:" + ????????? + "点");
for(int i = 0; i < ?????????; i++) {
    System.out.println(i);
    if(scores[i] >= ?????????) { //scoresのi番目が平均点以上なら
        System.out.println("番号" + ? + ":" + ??????[i] + "点 合格");
    } else {
        System.out.println("番号" + ? + ":" + ????????? + "点 再テスト");
    }
}
```

配列の要素数を超えると...

```
int[] b;           // int型の要素をもつ配列bを宣言
```

```
b = new int [3];   // 要素が3個ある配列を生成
```

```
b[0] = 5;          b[1] = 3;
```

```
b[2] = 4;          b[3] = 2;
```

配列の添え字は0から始まるので、
最後の要素の添え字は、要素数-1

```
System.out.println(b[0]);
```

```
System.out.println(b[1]);
```

```
System.out.println(b[2]);
```

```
System.out.println(b[3]);
```

5	3	4
0	1	2

3

そんな要素ないよ！



プログラムとエラーの内容

```
Foo.java
1
2 public class Foo {
3
4     public static void main(String[] args) {
5         // TODO 自動生成されたメソッド・スタブ
6         int[] b;           // int型の要素をもつ配列aを宣言
7         b = new int [3];    // 要素が3個ある配列を生成
8         b[0] = 5;           b[1] = 3;
9         b[2] = 4;           b[3] = 2;
10        System.out.println(b[0]);
11        System.out.println(b[1]);
12        System.out.println(b[2]);
13        System.out.println(b[3]);
14    }
15 }
16
```

問題 @ Javadoc Declaration コンソール

<終了> Foo (1) [Java アプリケーション] /Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/java (2018/10/15 11:05:47)

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException: 3](#)
at Foo.main([Foo.java:9](#))

エラーの内容

```
java.lang.ArrayIndexOutOfBoundsException:3  
    at Foo.main(Foo.java:9)
```

Array: 配列 **Index**: 添え字

OutOfBoundsException: 区域を超える

→ 配列の添え字が要素数を超えた

at Foo.main(Foo.java:9) → **9**行目が怪しい

- エラーが出たら、まずはエラーの行を確認！
- エラーの内容がわからない時は、出たエラーの内容で検索してみると、何が間違いかわかる！

配列中の最大の要素を求める

教科書P85

- 配列中の最大の要素を求める方法
 1. 暫定的な最大要素を格納する変数を用意、配列の先頭の要素を代入（最初の要素を暫定最大要素としておく）
 2. 配列の先頭から順番に、暫定最大要素と比較し、その要素が暫定最大要素よりも大きければ、暫定最大要素を更新
 - ただし、先頭の要素と比較する必要はない
- 配列の最後の要素まで比較すれば、暫定最大値が配列の要素の最大値と分かる

62	84	78	50	91	91
0	1	2	3	4	max

配列中の最大値を求めるプログラム

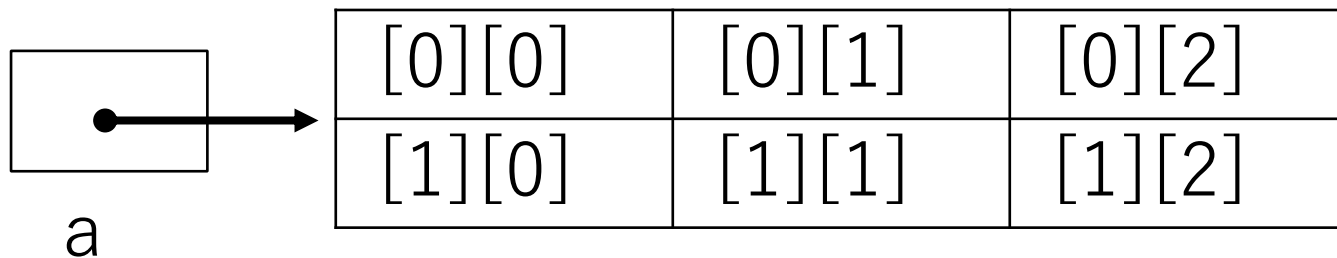
```
public class MaxArray {  
    public static void main(String[] args) {  
        // TODO 自動生成されたメソッド・スタブ  
        int [] tests = {62, 84, 78, 50, 91};  
        int max = tests[0];  
        for(int i = 1; i < tests.length; i++) {  
            if(max < tests[i]) {  
                max = tests[i];  
            }  
        }  
        System.out.println("max:" + max);  
    }  
}
```

多次元配列 教科書P86

- n個のインデックスで要素を指定する配列をn次元配列という
- 2次元以上を多次元配列と呼ぶ

```
int [ ] [ ] a = new int [2][3];
```

配列のイメージ図



多次元配列の長さ

- 2*3 の要素を持つ 2 次元配列

```
String [ ][ ] matrix = {  
    {"0-1", "0-2", "0-3"},  
    {"1-1", "1-2", "1-3"}  
};
```



デバッグモードで見ると分かりやすい

- `System.out.println(matrix.length);`
 の結果は 2
- `System.out.println(matrix[0].length);`
 の結果は 3 になる

多次元配列を繰り返しで処理

- 2次元配列の場合、2重ループで処理

```
for(int y = 0; y < matrix.length; y++) {  
    for(int x = 0; x < matrix[y].length; x++) {  
        System.out.print(matrix[y][x] + " ");  
    }  
    System.out.println();  
}
```

```
for(String[] array : matrix) {  
    for(String s : array) {  
        System.out.print(s + " ");  
    }  
    System.out.println();  
}
```

拡張forの場合

- 配列の要素を先頭から順に処理する際に、
下記のような記述をすることが出来る

```
String [] testArray = {"A", "B", "C", "D"};
```

```
//for文で表示
```

```
for(int i = 0; i < testArray.length; i++) {  
    System.out.print(testArray[i] + ",");  
}
```

```
System.out.println();
```

```
//拡張for文で表示
```

```
for(String s : testArray) {  
    System.out.print(s + ",");  
}
```

s に testArrayの要素が先頭から順番に代入される

同じ型にする

```
for( 型 名前 : 配列名 ) {  
}
```

※注意：拡張for文では配列の要素を書き換えることが出来ない

forと拡張forの使い分け

- 拡張forが利用できるのは、
配列の要素を先頭から末尾まで順に処理する場合で、
配列の要素を書き換ええないときのみ
 - この場合には、拡張forを使う方が望ましい
- 慣れてくるまでは、for文のみを覚えておけば問題ない
 - 使い分けが理解できたら、拡張forも覚えていこう

mainメソッドの引数

- mainメソッドの引数(`String[] args`)は文字列型の配列
- Javaプログラム実行時に引数として値を渡すことができる
- 実行時に引数が文字列として格納される

```
class Foo {  
    public static void main (String[] args) {  
        ...  
    }  
}
```

まとめ

- 配列

- 同じ型のデータを扱いたいときに、データを並べてまとめる
- 要素 ひとつひとつの箱
- 添え字 要素につけられた番号 配列名[添え字] で要素にアクセス
- 配列の宣言 要素の型 [] 配列名;
- 要素の生成 配列名 = new 要素の型[要素数]
- 宣言と要素の生成 要素の型 [] 配列名 = new 要素の型[要素数];
- 配列の初期化 要素の型 [] 配列名 = {要素1,要素2,...};
- 配列と繰り返し for文と配列名.length(要素数)を使う