プログラミング言語　第十二回

担当：篠沢 佳久
櫻井 彰人

平成29年 7月3日

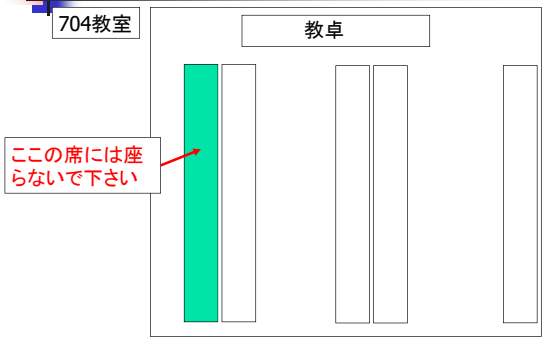
本日の内容

- 二次元配列（2）
  - 二重ループでの処理
  - 成績表の処理
  - エラトステネスの篩
  - ソーティング
- 練習問題

7/17の最終課題について

- 必ず出席して下さい（出席免除者も必ず出席して下さい）
- 海の日ですが，講義日です
- 講義資料などweb 上のリソースは使ってもらって結構です
  - 他人との通信は禁止です
- ITCのPCで課題作成して下さい
  - 自分のノートPCで作成することは禁止です
- これまでのレポート課題，練習問題，自習問題でよく復習しておいて下さい

7/17の最終課題について



二次元配列（復習）

二次元配列
要素の参照，代入

二次元配列の宣言①

3×3の行列 a

$$a = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

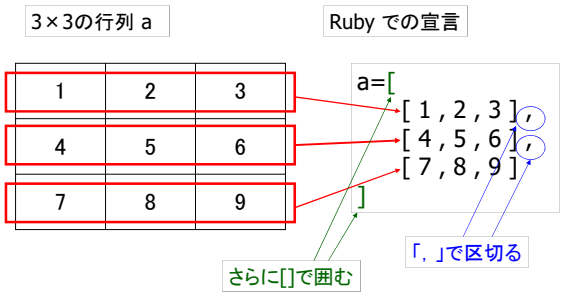
表の場合

1	2	3
4	5	6
7	8	9

Ruby での宣言

```
a=[
    [1,2,3],
    [4,5,6],
    [7,8,9]
]
```

二次元配列の宣言②



7

二次元配列の宣言③

```
a=[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]

p a
```

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

一次元配列

二次元配列は一次元配列の要素を一次元配列としているとみなせる

8

二次元配列の要素の参照①

```
a=[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

9

二次元配列の要素の参照②

```
a=[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

a[0][0]	a[0][1]	a[0][2]	a[0]
a[1][0]	a[1][1]	a[1][2]	a[1]
a[2][0]	a[2][1]	a[2][2]	a[2]

一次元配列

10

二次元配列の要素の参照③

要素数

a.length

1	2	3
4	5	6
7	8	9

a[0]
a[1]
a[2]

```
a=[
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

a[0].length
a[1].length
a[2].length

11

二次元配列の宣言

- 要素の値が分かっている場合
- 要素数のみ分かっている場合
- 要素の値，要素数も分からない場合

12

二次元配列の宣言①

- 要素の値が分かっている場合

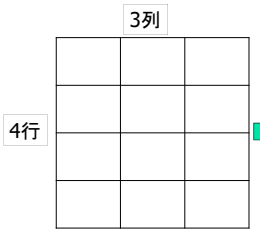
1	2	3
4	5	6
7	8	9
10	11	12

```
a=[
   [ 1 , 2 , 3 ] ,
   [ 4 , 5 , 6 ] ,
   [ 7 , 8 , 9 ] ,
   [ 10 , 11 , 12 ]
]
```

13

二次元配列の宣言②

- 要素数のみ分かっている場合




```
a = Array.new( 4 )

a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )
a[ 2 ] = Array.new( 3 )
a[ 3 ] = Array.new( 3 )
```

14

二次元配列の要素への代入

```
a = Array.new( 4 )

a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )
a[ 2 ] = Array.new( 3 )
a[ 3 ] = Array.new( 3 )

a[ 0 ][ 0 ] = 1
a[ 0 ][ 1 ] = 2
a[ 0 ][ 2 ] = 3
a[ 1 ][ 0 ] = 4
a[ 1 ][ 1 ] = 5
a[ 1 ][ 2 ] = 6
```

```
a[ 2 ][ 0 ] = 7
a[ 2 ][ 1 ] = 8
a[ 2 ][ 2 ] = 9

a[ 3 ][ 0 ] = 10
a[ 3 ][ 1 ] = 11
a[ 3 ][ 2 ] = 12

p a
```

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

15

二次元配列の宣言③

aは配列と宣言

```
a = []

a[ 0 ] = []
a[ 0 ][ 0 ] = 1
a[ 0 ][ 1 ] = 2
a[ 0 ][ 2 ] = 3

a[ 1 ] = []
a[ 1 ][ 0 ] = 4
a[ 1 ][ 1 ] = 5
a[ 1 ][ 2 ] = 6
```

```
a[ 2 ] = []
a[ 2 ][ 0 ] = 7
a[ 2 ][ 1 ] = 8
a[ 2 ][ 2 ] = 9

a[ 3 ] = []
a[ 3 ][ 0 ] = 10
a[ 3 ][ 1 ] = 11
a[ 3 ][ 2 ] = 12

p a
```

a[0], a[1], a[2] , a[3]が配列であることを宣言

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

16

二次元配列と繰り返し（復習）

二重ループ中での要素の参照

17

二次元配列の要素の参照①

要素番号（インデックス）を用いての参照

```
a=[
   [ 1 , 2 , 3 ] ,
   [ 4 , 5 , 6 ] ,
   [ 7 , 8 , 9 ] ,
   [ 10 , 11 , 12 ]
]

4.times{ |i|
   3.times{ |j|
     print( " a[ " , i , " ][ " , j , " ] = " ,
          a[ i ][ j ] , "¥n" )
   }
}
```

i=0, j=0～2

i=1, j=0～2

i=2, j=0～2

i=3, j=0～2

```
Z:¥Ruby>ruby sample.rb
a[ 0 ][ 0 ] = 1
a[ 0 ][ 1 ] = 2
a[ 0 ][ 2 ] = 3
a[ 1 ][ 0 ] = 4
a[ 1 ][ 1 ] = 5
a[ 1 ][ 2 ] = 6
a[ 2 ][ 0 ] = 7
a[ 2 ][ 1 ] = 8
a[ 2 ][ 2 ] = 9
a[ 3 ][ 0 ] = 10
a[ 3 ][ 1 ] = 11
a[ 3 ][ 2 ] = 12
```

i j

18

二次元配列の要素の参照②

```
a=[
   [ 1 , 2 , 3 ],
   [ 4 , 5 , 6 ],
   [ 7 , 8 , 9 ],
   [ 10 , 11 , 12 ]
]
a.length.times{ |i|
   a[ i ].length.times{ |j|
     print( " a[ " , i , " ][ " , j , " ] = " ,
         a[ i ][ j ] , "¥n" )
   }
}
```

a.lengthの値は4

```
Z:¥Ruby>ruby sample.rb
 a[ 0 ][ 0 ] = 1
 a[ 0 ][ 1 ] = 2
 a[ 0 ][ 2 ] = 3
 a[ 1 ][ 0 ] = 4
 a[ 1 ][ 1 ] = 5
 a[ 1 ][ 2 ] = 6
 a[ 2 ][ 0 ] = 7
 a[ 2 ][ 1 ] = 8
 a[ 2 ][ 2 ] = 9
 a[ 3 ][ 0 ] = 10
 a[ 3 ][ 1 ] = 11
 a[ 3 ][ 2 ] = 12
```

a[0].length, a[1].length, a[2].length, a[3].length は全て3

19

二次元配列の要素の参照③

each を用いての参照

```
a=[
   [ 1 , 2 , 3 ],
   [ 4 , 5 , 6 ],
   [ 7 , 8 , 9 ],
   [ 10 , 11 , 12 ]
]

(0..a.length-1).each{ |i|
   (0..a[ i ].length-1).each{ |j|
     print( " a[ " , i , " ][ " , j , " ] = " ,
         a[ i ][ j ] , "¥n" )
   }
}
```

```
Z:¥Ruby>ruby sample.rb
 a[ 0 ][ 0 ] = 1
 a[ 0 ][ 1 ] = 2
 a[ 0 ][ 2 ] = 3
 a[ 1 ][ 0 ] = 4
 a[ 1 ][ 1 ] = 5
 a[ 1 ][ 2 ] = 6
 a[ 2 ][ 0 ] = 7
 a[ 2 ][ 1 ] = 8
 a[ 2 ][ 2 ] = 9
 a[ 3 ][ 0 ] = 10
 a[ 3 ][ 1 ] = 11
 a[ 3 ][ 2 ] = 12
```

20

二次元配列の要素の参照④

```
a=[
   [ 1 , 2 , 3 ],
   [ 4 , 5 , 6 ],
   [ 7 , 8 , 9 ],
   [ 10 , 11 , 12 ]
]
a.each{ |i|
   p  i
   i.each{ |j|
      print( j , "¥n" )
   }
}
```

```
a=[
   [ 1 , 2 , 3 ],
   [ 4 , 5 , 6 ],
   [ 7 , 8 , 9 ],
   [ 10 , 11 , 12 ]
]
a.each{ |i|
   p  i
   i.length.times{ |j|
      print( i[ j ] , "¥n" )
   }
}
```

i にはa[0],a[1],a[2],a[3]が代入されます

jには配列の値が代入されます

jには0,1,2が代入されます

21

二次元配列の要素への代入①

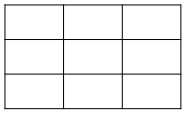
```
a=[]

a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )
a[ 2 ] = Array.new( 3 )

count = 1
3.times{ |i|
   3.times{ |j|
     a[ i ][ j ] = count
     count += 1
   }
}

p a
```

3×3の要素を持つ二次元配列を宣言



代入式

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

22

二次元配列の要素への代入①"

二次元配列の宣言の方法

```
a=[]

count = 1
3.times{ |i|
   a[ i ] = []
   3.times{ |j|
     a[ i ][ j ] = count
     count += 1
   }
}

p a
```

要素数を指定しない

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

23

二次元配列の要素への代入②

```
a=[]

4.times{ |i|
   a[ i ] = []
   4.times{ |j|
     if i == j then
       a[ i ][ j ] = 1
     else
       a[ i ][ j ] = 0
     end
   }
}

a.length.times{ |i|
   a[ i ].length.times{ |j|
     print( a[ i ][ j ] , " " )
   }
   print( "¥n" )
}
```

i と j が同じ場合は 1
それ以外は 0

```
Z:¥Ruby>ruby sample.rb
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

24

二次元配列の要素への代入③

```
a=[]

4.times{ |i|
   a[ i ] = []
   (i+1).times{ |j|
    if i == j then
       a[ i ][ j ] = 1
    else
       a[ i ][ j ] = 0
    end
   }
}

a.length.times{ |i|
   a[ i ].length.times{ |j|
     print( a[ i ][ j ] , " " )
   }
   print( "¥n" )
}
```

要素数が異なってもよい

```
Z:¥Ruby>ruby sample.rb
1
0 1
0 0 1
0 0 0 1
```

25

二次元配列の要素への代入③'

```
a=[]

4.times{ |i|
   a[ i ] = []
   (i+1).times{ |j|
    if i == j then
       a[ i ][ j ] = 1
    else
       a[ i ][ j ] = 0
    end
   }
}

a.length.times{ |i|
   a[ i ].length.times{ |j|
     print( a[ i ][ j ] , " " )
   }
   print( "¥n" )
}
```

i = 0 の場合，1.times
→ 1回のみのループ
→ a[0] は要素数1個

i = 1 の場合，2.times
→ 2回のみのループ
→ a[1] は要素数2個

i = 2 の場合，3.times
→ 3回のみのループ
→ a[2] は要素数3個

i = 3 の場合，4.times
→ 4回のみのループ
→ a[3] は要素数3個

26

二次元配列の要素への代入④

```
a=[]
4.times{ |i|
   a[ i ] = []
   (0..3-i).each{ |j|
   if i+j == 3 then
         a[ i ][ j ] = 1
   else
         a[ i ][ j ] = 0
   end
   }
}

a.length.times{ |i|
   a[ i ].length.times{ |j|
     print( a[ i ][ j ] , " " )
   }
   print( "¥n" )
}
```

どうして配列の要素がこのようになるでしょうか？

```
Z:¥Ruby>ruby sample.rb
0 0 0 1
0 0 1
0 1
1
```

27

成績表の処理

二次元の表の計算

28

2次元表で平均値を求める①

- 4人の平均点を求める
  - 出席番号1番：(70+60+83) / 3 = 71

出席番号	国語	数学	英語
1	70	60	83
2	43	49	76
3	59	79	43
4	67	74	83

  - データは、学生ごとに纏めて記憶するのが自然であろう

```
irb(main):008:0> p = [[1,70,60,83],[2,43,49,76],
irb(main):009:1*      [3,59,79,43],[4,67,74,83]]
=> [[1, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79, 43], [4, 67, 74, 83]]
```

29

2次元表で平均値を求める②

出席番号	国語	数学	英語
1	70	60	83
2	43	49	76
3	59	79	43
4	67	74	83

⟺

```
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]
```

p[0][0]	p[0][1]	p[0][2]	p[0][3]
p[1][0]	p[1][1]	p[1][2]	p[1][3]
p[2][0]	p[2][1]	p[2][2]	p[2][3]
p[3][0]	p[3][1]	p[3][2]	p[3][3]

出席番号 i の平均点
(p[i][1]+p[i][2]+p[i][3]) / 3

30

各人の平均点を求めるプログラム①

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

(0..3).each{ |i|
  sum = 0
  (1..3).each{ |j|
    sum += p[i][j]
  }
  ave = sum / 3
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 です
出席番号 2 の人の平均点は 56 です
出席番号 3 の人の平均点は 60 です
出席番号 4 の人の平均点は 74 です
```


各人の平均点を求めるプログラム①

p[0][0]	p[0][1]	p[0][2]	p[0][3]	p[0]
p[1][0]	p[1][1]	p[1][2]	p[1][3]	p[1]
p[2][0]	p[2][1]	p[2][2]	p[2][3]	p[2]
p[3][0]	p[3][1]	p[3][2]	p[3][3]	p[3]

```
(0..3).each{ |i|
  sum = 0
  (1..3).each{ |j|
    sum += p[i][j]
  }
  ave = sum / 3
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

p[0], p[1], p[2], p[3]の順番に計算

p[i][1]+p[i][2]+p[i][3]

各人の平均点を求めるプログラム①'

変更点はどこでしょうか

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

(0..3).each{ |i|
  sum = 0.0
  (1..3).each{ |j|
    sum += p[i][j]
  }
  ave = sum / 3
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71.0 です
出席番号 2 の人の平均点は 56.0 です
出席番号 3 の人の平均点は 60.3333333333333 です
出席番号 4 の人の平均点は 74.6666666666667 です
```


各人の平均点を求めるプログラム②

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

(0..3).each{ |i|
  sum = 0
  (1..3).each{ |j|
    sum += p[i][j]
  }
  ave = sum / 3
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

このプログラムの場合，科目数や学生が追加された場合，修正しなければならない

各人の平均点を求めるプログラム②'

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += p[i][j]
  }
  ave = sum / ( p[i].length-1 )
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 です
出席番号 2 の人の平均点は 56 です
出席番号 3 の人の平均点は 60 です
出席番号 4 の人の平均点は 74 です
```


各人の平均点を求めるプログラム②'

```
p = [
    [1,70,60,83,65],
    [2,43,49,76,70],
    [3,59,79,43,28],
    [4,67,74,83,81],
    [5,91,80,95,100]
]

(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += p[i][j]
  }
  ave = sum / ( p[i].length-1 )
  puts( "出席番号 #{i+1} の人の平均点は #{ave} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 69 です
出席番号 2 の人の平均点は 59 です
出席番号 3 の人の平均点は 52 です
出席番号 4 の人の平均点は 76 です
出席番号 5 の人の平均点は 91 です
```

各人の平均点を求めるプログラム③

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83],
]

p.each{ |a|
   sum = 0
   (1..a.length-1).each{ |i|
   sum += a[ i ]
    }
    ave = sum / ( a.length-1 )
    puts( "出席番号 #{a[0]} の人の平均点は #{ave} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 です
出席番号 2 の人の平均点は 56 です
出席番号 3 の人の平均点は 60 です
出席番号 4 の人の平均点は 74 です
```


各人の平均点を求めるプログラム③

```
p.each{ |a|
   sum = 0
   (1..a.length-1).each{ |i|
      sum += a[ i ]
    }
    ave = sum / ( a.length-1 )
    puts( "出席番号 #{a[0]} の人の平
    均点は #{ave} です" )
}
```

a には配列
[1,70,60,83]
[2,43,49,76]
[3,59,79,43]
[4,67,74,83]
が順番に代入される

a.length の値は4

a=[1,70,60,83] の場合
sum = a[1] + a[2] + a[3]

a[0] は出席番号

各人の平均点を求めるプログラム③'

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83],
]

p.each{ |a|
   sum = 0
   count = 0
   a.each{ |i|
         if count > 0 then
                  sum += i
         end
      count += 1
    }
    ave = sum / ( a.length-1 )
    puts( "出席番号 #{a[0]} の人の平均点は #{ave} です" )
}
```

また，別の書き方ですが…

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 です
出席番号 2 の人の平均点は 56 です
出席番号 3 の人の平均点は 60 です
出席番号 4 の人の平均点は 74 です
```


各人の平均点を求めるプログラム③'

```
p.each{ |a|
   sum = 0
   count = 0
   a.each{ |i|
         if count > 0 then
                  sum += i
         end
      count += 1
    }
    ave = sum / ( a.length-1 )
    puts( "出席番号 #{a[0]} の人の平均点は #{ave} です" )
}
```

a には配列
[1,70,60,83]
[2,43,49,76]
[3,59,79,43]
[4,67,74,83]
が順番に代入される

iにはa[0],a[1],a[2],a[3]の要素が代入される

各人の平均点を求めるプログラム④

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

ave =[]
(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += p[i][j]
  }
  ave[i] = sum / ( p[i].length-1 )
}

(0..p.length-1).each{ |i|
  puts( "出席番号 #{p[i][0]} の人の平均点は #{ave[i]} です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 です
出席番号 2 の人の平均点は 56 です
出席番号 3 の人の平均点は 60 です
出席番号 4 の人の平均点は 74 です
```

一次元配列 ave に平均点を格納

プログラムの書き方

- 結果を求めるまでは一通りではない
- いろいろな書き方があります

標準偏差を求めるプログラム

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

ave =[]
(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += p[i][j]
  }
  ave[i] = sum / ( p[i].length-1 )
}
```

平均を求める


```
sd = []

(0..p.length-1).each{ |i|
  puts( "出席番号 #{p[i][0]} の人の平均点は #{ave[i]} , 標準
    偏差は #{sd[i]}です" )
}
```

各学生ごとに標準偏差を求め，配列sdに格納する
（練習問題②）

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 , 標準偏差は 11.5325625946708です
出席番号 2 の人の平均点は 56 , 標準偏差は 17.5783958312469です
出席番号 3 の人の平均点は 60 , 標準偏差は 18.0277563773199です
出席番号 4 の人の平均点は 74 , 標準偏差は 8.06225774829855です
```


エラトステネスの篩

素数を見つける方法（アルゴリズム）

素数の判定

```
# coding:Windows-31J
n=gets.chomp.to_i
(2..n-1).each{ |x|
    if n % x == 0 then
        print( "この数字は素数ではありません¥n" )
        break
    end
}
```

整数n
2からn-1で割り切れたら素数ではない

素数でない数字を入力した場合

```
Z:¥Ruby>ruby sample.rb
153
この数字は素数ではありません
```

素数を入力した場合

```
Z:¥Ruby>ruby sample.rb
37
```


素数の場合→「素数です」と出力するには？

```
#coding: Windows-31J
n=gets.chomp.to_i
p = 1
(2..n-1).each{ |x|
    if n % x == 0 then
        p = 0
        break
    end
}
if p == 0 then
    print( "この数字は素数ではありません¥n" )
else
    print( "この数字は素数です¥n" )
end
```

p=1としておく（重要！）

素数でないと判明した場合p=0とする

```
Z:¥Ruby>ruby sample.rb
179
この数字は素数です
```

p が 0の場合→素数
p が1の場合→素数でない

再: 素数を印字するプログラム

```
# coding: Windows-31J
# 素数は、2～「自分-1」では割り切れない整数
# 2から10までの素数を印字しよう
(2..10).each{ |n|                  # 素数候補
  p = 1                            # まだ素数候補
  (2..n-1).each{ |x|               # これで調べる
    (p=0; break) if n%x == 0 # 割り切れたら素数ではない
  }
  puts "#{n} は素数" if p==1  # 素数候補のままなら素数！
}
```

```
2 は素数
3 は素数
5 は素数
7 は素数
=> 2..10
```

しかし、効率が悪い。余分な割り算をたくさん行っている。
エラトステネスの篩にしたい。
篩はどうすれば表現できるか？

エラトステネスの篩

- 自然数nまでの素数を見つける方法（アルゴリズム）
- 計算機によって，問題を効率的に解く手順を「アルゴリズム」と呼びます
- 次ページから1から100までの素数を求める手順を示します

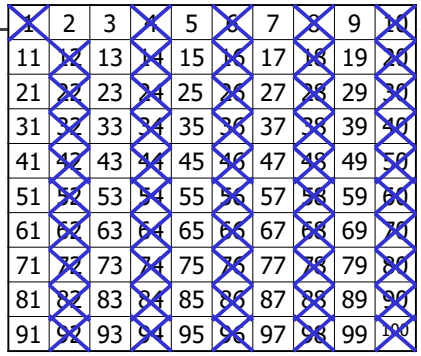
49

1から100までの配列を用意（配列名はsieveとします）

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

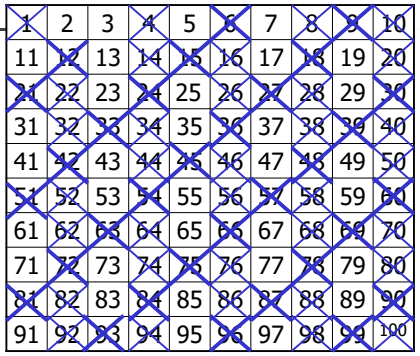
50

1を削除（0とする）　要素 2×2から2要素ごとに削除



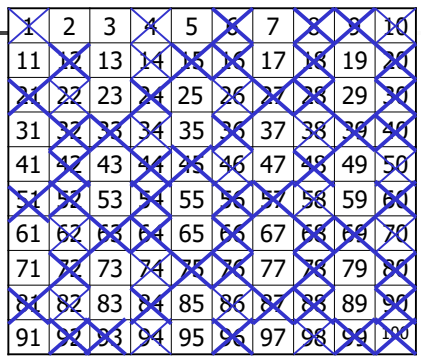
51

要素 3×2から3要素ごとに削除



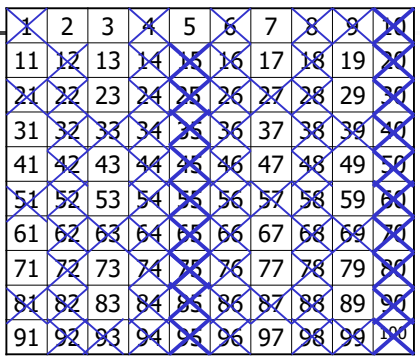
52

（無駄ですが…）要素 4×2から4要素ごとに削除



53

要素 5×2から5要素ごとに削除



54

（無駄ですが…）要素 6×2から6要素ごとに削除

~~1~~	2	3	4	5	~~6~~	7	8	9	~~10~~
11	~~12~~	13	14	15	~~16~~	17	18	19	~~20~~
21	22	23	24	25	~~26~~	27	28	29	~~30~~
31	32	33	34	35	~~36~~	37	38	39	~~40~~
41	42	43	44	45	~~46~~	47	48	49	~~50~~
51	52	53	54	55	~~56~~	57	58	59	~~60~~
61	62	63	64	65	~~66~~	67	68	69	~~70~~
71	72	73	74	75	~~76~~	77	78	79	~~80~~
81	82	83	84	85	~~86~~	87	88	89	~~90~~
91	92	93	94	95	~~96~~	97	98	99	~~100~~

55

要素 7×2から7要素ごとに削除

~~1~~	2	3	4	5	6	~~7~~	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	~~27~~	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	~~57~~	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	~~77~~	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

56

8,9,10の場合も同様に　要素 11×2から11要素ごとに削除

~~1~~	2	3	4	5	6	7	8	9	10
11	~~12~~	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	~~100~~

どこまで調べればよいでしょう？

57

削除されていない数が素数

~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

58

エラトステネスの篩のまとめ

- 基本方針
  - 配列 sieve の第 i 要素が
    - 0以外 なら素数候補（いつ「素数候補」から「素数」になるか？）
    - 0 なら素数でないこと確定

  とする

59

エラトステネスの篩のまとめ

手順

- 配列 sieve を 1 で埋める
  - 最初は、すべてが素数候補
- 配列の第2*2要素から2要素ごとに第100要素まで 0 を代入
- 配列の第3*2要素から3要素ごとに第100要素まで 0 を代入
- 配列の第4*2要素から4要素ごとに第100要素まで 0 を代入　# これは無駄だよね
- 配列の第5*2要素から5要素ごとに第100要素まで 0 を代入
- 配列の第100*2要素から100要素ごとに第100要素まで 0 を代入　# ???

60

エラトステネスの篩: プログラム例①

```
sieve = Array.new(101)
sieve.length.times { |i|
  sieve[i] = 1
}

sieve[0]=0
sieve[1]=0
(2..sieve.length-1).each { |i|
  (i*2).step(sieve.length-1, i) { |j|
      sieve[j]=0
  }
}

sieve.length.times { |i|
      print( "#{i} " ) if sieve[i] != 0
}
```

101個の配列を用意
sieve[0] は利用しない

初期設定
配列の全要素に1を代入

iは2から100まで変わる

jは2*iから100まで，iごとに変わる

素数でないことをチェック

要素が0でなければ素数と判定

エラトステネスの篩: プログラム例②

```
sieve = Array.new(101)
sieve.length.times { |i|
  sieve[i] = i # ちょっと工夫
}
sieve[0]=0
sieve[1]=0
(2..sieve.length-1).each { |i|
  (i*2).step(sieve.length-1, i) { |j|
      sieve[j]=0
  }
}
sieve.each { |p|
      print( "#{p} " ) if p!=0
}
```

初期設定
0ではなく要素番号を代入

エラトステネスの篩: 実行結果

100までの素数

```
Z:¥ruby>ruby sample.rb
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
71 73 79 83 89 97
```

1000までの素数

```
Z:¥Ruby>ruby sample.rb
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127
37 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263
71 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419
31 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577
93 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739
51 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911
29 937 941 947 953 967 971 977 983 991 997
```

この方法は0を無駄に代入している回数が多いです
改良すべき点はどこでしょうか？

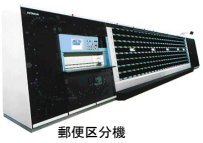
ソーティング

配列の要素の並び替え
バブルソート

ソート: 前書きその１

- 配列要素をある順序に並べ直すこと
  - 事務処理で、最も基本かつ重要な計算。過去さまざまな方法が工夫された
  - とことで、ソートの元の意味は仕分けることであるが、英語でもコンピュータ界では、今では、この意味に使う
  - なぜか？
- ちなみに、現在のソーターの例
  - 「もの」を流しながら、随時、仕分けをしていく

http://www.hokusho.co.jp/



郵便区分機

http://www.hitachi-omron-ts.co.jp/products/gazou/004.html

ソート: 前書きその２

- 配列要素をある順序に並べ直すこと
  - ソートの元の意味は仕分けることであるが、英語でもコンピュータ界では、今では、この意味に使う
  - なぜか？

This is the first horizontal card sorter, introduced by IBM in 1925 to operate at almost twice the speeds of the older Type 70 vertical sorter. This machine uses a direct magnetically operated control for the chute blades which replaced a much more complex mechanical device in the older machine. The Type 80 grouped all cards of similar classification (such as "sales by products") and at the same time arranged such classifications in numerical sequence. With 10,200 units on rental at the close of 1943, the Type 80 had the largest inventory for any machine at that time.

http://www-03.ibm.com/ibm/history/exhibits/attic3/attic3_136.html

The original Hollerith electric tabulating system did not have an adequate method for sorting cards. This became a problem in the 1900 agricultural census, so Herman Hollerith (1860-1929) developed an automatic sorter. The first one was a tabletop model with the bins arranged horizontally. Later, when his system was gaining favor commercially, Hollerith redesigned the sorter into a sturdier, vertical machine that would not take up too much space in small railroad offices. This 070 Vertical Sorting Machine of 1908 could operate at a rate of 250-270 cards a minute.

http://www-03.ibm.com/ibm/history/exhibits/attic3/attic3_070.html

ソーティングとは

- ソーティングとは
  - データを昇順（小さいものから大きいものへの順）に，もしくは，降順（大きいものから小さいものへの順）に並べ替えること

5, 3, 6, 2, 5, 4

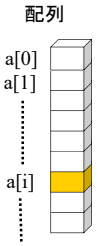
↓ ソートする(昇順)

2, 3, 4, 5, 5, 6

ソーティングの基本操作

- 本日扱うソーティングの対象となるデータ列は，一つの配列に入っているものとする．

・ ソーティングの基本操作
  - 配列要素間の
    - 比較操作と
    - 交換（移動）操作



ソーティングの例

配列a

	a
0	5
1	3
2	6
3	2
4	4
5	5

ソート

配列a

	a
0	2
1	3
2	4
3	5
4	5
5	6

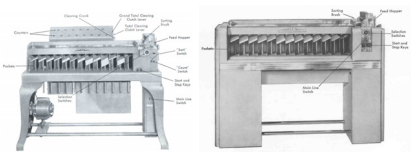
ソーティングのアルゴリズム

- ソーティングには，いろいろなアルゴリズムが知られている．

①馬鹿ソート，②選択ソート
③バブルソート，④シェーカーソート
⑤挿入ソート，⑥シェルソート
⑦クイックソート，⑧マージソート
⑨基数（radix）ソート など

ソート: radix sort

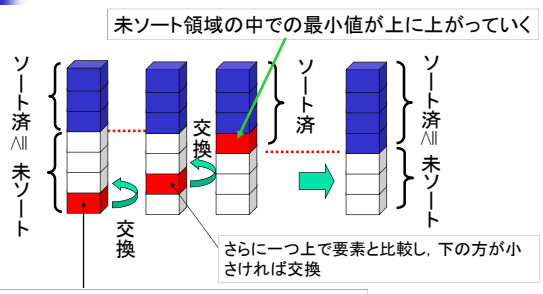
- 実は、区分機を使うと、昇順なり降順に並べることができる
  - 「できる」だけではなく、実際にそうしていた
- たとえば、10進3桁のID番号が振られているカードがあったとしよう
  - 一の位で、0,1,,,9に区分けする
    - 0～9の順に重ねると、下一桁では、0～9の順に並んでいる
  - その順序を崩さずに、十の位で、0,1,,,9に区分けする
    - 0 ～9の順に重ねると、下二桁では、00～99の順に並んでいる
  - その順序を崩さずに、百の位で、0,1,,,9に区分けする
    - 0 ～9の順に重ねると、下三桁では、000～999の順に並んでいる



ソート: bubble sort

- コンピュータでは、radix sort は、まず、使わない
- bubble sort も遅いので殆ど使わない。しかし、
  - わかり易いので教育用に
  - プログラムが短いので、ちょこっと使う時には便利
  - bubble とは「泡」
  - 配列の中を（縦に置く）、より小さいものが泡のように上に上がっていく

バブルソートのアルゴリズム①



73

バブルソートの具体例

次ページから下記の配列に対してバブルソートを行なう例を示します

60	2	11	82	29	21	24	98	51	24

74

i=0

60	2	11	82	29	21	24	98	51	24
60	2	11	82	29	21	24	98	24	51
60	2	11	82	29	21	24	24	98	51
60	2	11	82	29	21	24	24	98	51
60	2	11	82	29	21	24	24	98	51
60	2	11	82	21	29	24	24	98	51
60	2	11	21	82	29	24	24	98	51
60	2	11	21	82	29	24	24	98	51
60	2	11	21	82	29	24	24	98	51
確定 → 2	60	11	21	82	29	24	24	98	51

配列の最後から一つ上の要素と比較

小さければ交換
大きい場合，何もしない

i の位置まで行なう

i=1

2	60	11	21	82	29	24	24	98	51
2	60	11	21	82	29	24	24	51	98
2	60	11	21	82	29	24	24	51	98
2	60	11	21	82	29	24	24	51	98
2	60	11	21	82	24	29	24	51	98
2	60	11	21	24	82	29	24	51	98
2	60	11	21	24	82	29	24	51	98
2	11	60	21	24	82	29	24	51	98
確定 → 2	11	60	21	24	82	29	24	51	98

配列の最後から一つ上の要素と比較

i の位置まで行なう

76

i=2

2	11	60	21	24	82	29	24	51	98
2	11	60	21	24	82	29	24	51	98
2	11	60	21	24	82	29	24	51	98
2	11	60	21	24	82	24	29	51	98
2	11	60	21	24	24	82	29	51	98
2	11	60	21	24	24	82	29	51	98
2	11	21	60	24	24	82	29	51	98
2	11	21	60	24	24	82	29	51	98
確定									

配列の最後から一つ上の要素と比較

i の位置まで行なう

77

i=3

2	11	21	60	24	24	82	29	51	98
2	11	21	60	24	24	82	29	51	98
2	11	21	60	24	24	82	29	51	98
2	11	21	60	24	24	29	82	51	98
2	11	21	60	24	24	29	82	51	98
2	11	21	60	24	24	29	82	51	98
2	11	21	24	60	24	29	82	51	98
確定									

配列の最後から一つ上の要素と比較

i の位置まで行なう

以下，同様...

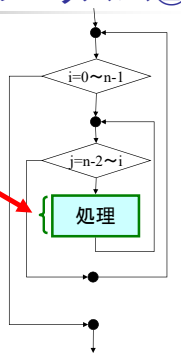
78

バブルソートのアルゴリズム②

配列の一番下から，iまで調べていく

```
(0..a.length-1).each{ |i|
  (a.length-2).downto(i) { |j|
    if a[ j ]>a[ j+1 ] then
      a[ j ] と a[ j+1 ] を交換
    end
  }
}
```

下の要素と比較して，下の要素が小さければ交換する

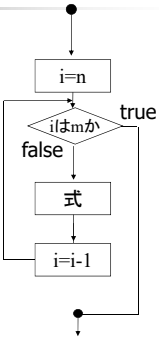


79

downto①

```
n.downto(m){ |i|
      式
}
```

n-m回式を繰り返す（n>m）
iには自動的にnからmが代入される



80

```
10.downto(0){ |x|
  printf( "x=%2d x^2=%3d¥n" , x , x**2 )
}
```

x は10から0まで1刻みで変化

```
Z:¥ruby>ruby sample.rb
x=10 x^2=100
x= 9 x^2= 81
x= 8 x^2= 64
x= 7 x^2= 49
x= 6 x^2= 36
x= 5 x^2= 25
x= 4 x^2= 16
x= 3 x^2=  9
x= 2 x^2=  4
x= 1 x^2=  1
x= 0 x^2=  0
```

81

補足: 入れ替え

- 変数値の入替え
  - 変数 a と変数 b に入っている値を入替えたい。どうすればいいか？
    - 当然，a=b; b=a ではだめです．どうして？

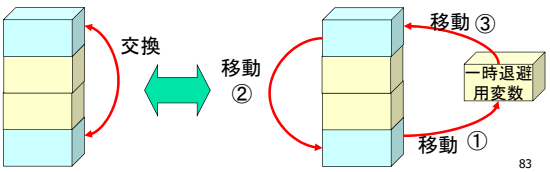
```
irb(main):007:0> a=1; b=10; puts( "a=#{a}, b=#{b}" )
a=1, b=10
=> nil
irb(main):008:0> a=b; b=a; puts( "a=#{a}, b=#{b}" )
a=10, b=10
=> nil
```

82

補足: 入れ替え その2

- 入替えには、作業領域があればよい

```
irb(main):009:0> a=1; b=10; puts( "a=#{a}, b=#{b}" )
a=1, b=10
=> nil
irb(main):010:0> w=a; a=b; b=w; puts( "a=#{a}, b=#{b}" )
a=10, b=1
=> nil
```



83

補足: 入れ替え その3

- 配列要素に対しても同様

```
irb(main):001:0> x=[ 3 , 5 ]
=> [3, 5]
irb(main):002:0> work=x[0]
=> 3
irb(main):003:0> x[0]=x[1]
=> 5
irb(main):004:0> x[1]=work
=> 3
irb(main):005:0> p x
[5, 3]
=> nil
```

84

bubble sort のプログラム①

```
a=[ 4,1,8,2,6,5 ]
p a
(0..a.length-1).each{ |i|
  (a.length-2).downto(i) { |j|
    if a[j]>a[j+1] then
      w = a[j]
      a[j]=a[j+1]
      a[j+1]=w
    end
  }
}
p a
```

a[j]とa[i]を交換

```
Z:¥Ruby>ruby sample.rb
[4, 1, 8, 2, 6, 5]
[1, 2, 4, 5, 6, 8]
```

bubble sort のプログラム②

```
a=[ 4,1,8,2,6,5 ]
p a
(0..a.length-1).each{ |i|
  (a.length-2).downto(i) { |j|
    if a[j]<a[j+1] then
      w = a[j]
      a[j]=a[j+1]
      a[j+1]=w
    end
  }
}
p a
```

逆順にソート
前頁のプログラムとどこが違うでしょうか

```
Z:¥Ruby>ruby sample.rb
[4, 1, 8, 2, 6, 5]
[8, 6, 5, 4, 2, 1]
```

86

bubble sort のプログラム③

```
a=Array.new(10)
a.length.times{ |i|
   a[ i ] = rand( 10 )
}
p a
(0..a.length-1).each{ |i|
  (a.length-2).downto(i) { |j|
    if a[j]>a[j+1] then
      w = a[j]
      a[j]=a[j+1]
      a[j+1]=w
    end
  }
}
p a
```

乱数で整数列を生成
→ a[0]～a[9]に格納

```
Z:¥Ruby>ruby sample.rb
[5, 0, 5, 3, 2, 7, 9, 4, 0, 7]
[0, 0, 2, 3, 4, 5, 5, 7, 7, 9]
```

87

練習問題

練習問題①から④まで（頑張れる人は⑤も行なって下さい）

88

練習問題①

- スライド「2次元表で平均値を求める」（27ページ）において，各科目ごとの平均点を求めるプログラムを二重ループを用いて書きなさい

```
Z:¥Ruby>ruby sample.rb
国語 の平均点 59
算数 の平均点 65
英語 の平均点 71
```

89

練習問題①（ヒント）

p[0][0]	p[0][1]	p[0][2]	p[0][3]
p[1][0]	p[1][1]	p[1][2]	p[1][3]
p[2][0]	p[2][1]	p[2][2]	p[2][3]
p[3][0]	p[3][1]	p[3][2]	p[3][3]

国語の平均点
(p[0][1]+p[1][1]+p[2][1]+p[3][2]) / 4

数学の平均点
(p[0][2]+p[1][2]+p[2][2]+p[3][2]) / 4

英語の平均点
(p[0][3]+p[1][3]+p[2][3]+p[3][3]) / 4

90

練習問題②
各学生の点数の標準偏差を求め，配列sdに格納しなさい

```
# coding: Windows-31J
p = [
    [1,70,60,83],
    [2,43,49,76],
    [3,59,79,43],
    [4,67,74,83]
]

ave =[]
(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += p[i][j]
  }
  ave[i] = sum / ( p[i].length-1 )
}
```

平均を求める


```
sd = []
```

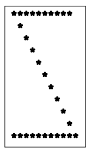
各学生ごとに標準偏差を求め，配列sdに格納する（練習問題②）

```
(0..p.length-1).each{ |i|
  puts( "出席番号 #{p[i][0]} の人の平均点は #{ave[i]} , 標準偏差は #{sd[i]}です" )
}
```

```
Z:¥Ruby>ruby sample.rb
出席番号 1 の人の平均点は 71 , 標準偏差は 11.5325625946708です
出席番号 2 の人の平均点は 56 , 標準偏差は 17.5783958312469です
出席番号 3 の人の平均点は 60 , 標準偏差は 18.0277563773199です
出席番号 4 の人の平均点は 74 , 標準偏差は 8.06225774829855です
```

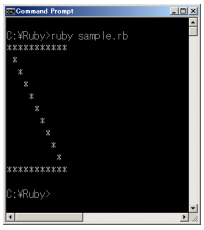

練習問題③

- 縦11文字横11文字（いずれも半角の文字数）の空間に，Z 字を裏返した字を，半角のアスタリスク（＊）を用いて印字してください．最上行と最下行は，11文字がアスタリスクになります．
  ただし，プログラム中でアスタリスクが出現するのは一回だけにして下さい．
  ヒント: 2重のループにすればできます



練習問題③

- 印字結果



- アスタリスクが1個のみでできなければ，複数個記述してもかまいません

練習問題④

- 下記の式において，zが最小となる時の，xとyおよびzの値を求めるプログラムを書きなさい．x,yとも-2から2の範囲とし，刻み幅は0.1とします．

$$z = 2x^2 - 4y^2 + 3xy + 6$$
$$x = -2.0, -1.9, -1.8, \cdots, 1.8, 1.9, 2.0$$
$$y = -2.0, -1.9, -1.8, \cdots, 1.8, 1.9, 2.0$$

```
Z:¥Ruby>sample.rb
xが    yが    の時，最小値は
```


練習問題⑤

- 配列 a 内に整数が記憶されているとする
- この内容を、バブルソートの一行を書き換えることによって，偶数と奇数にわけ，配列 a に入れ直すことを実現しなさい
- ただし，配列 a の前半に偶数、後半に奇数とし，偶数同士の順序，奇数同士の順序は保存せよ

配列a

```
a = [3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3]
```

```
[4, 2, 6, 8, 2, 3, 1, 1, 5, 9, 5, 3, 5, 9, 7, 9, 3, 3]
```

練習問題⑤

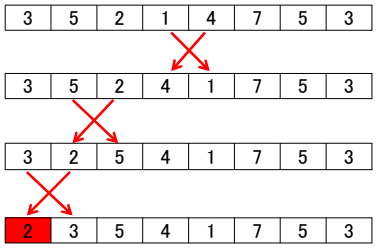
```
a = [3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3]
p a
(0..a.length-1).each{ |i|
  (a.length-2).downto(i) { |j|
    if a[j]>a[j+1] then
      w = a[j]
       a[j]=a[j+1]
      a[j+1]=w
    end
  }
}
p a
```

どこか一行変更してみて下さい

97

練習問題⑤(ヒント)

- 配列 a の前半を偶数、後半を奇数と習い変えたいということは…



98

練習問題

- 練習問題①から⑤を(できるだけ)(頑張って)行ないなさい
- プログラムと実行結果をワープロに貼り付けて，keio.jp から提出して下さい

99