



プログラミング言語 第十二回

担当: 篠沢 佳久
栗原 聡

2019年 7月8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



7/22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)
- 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, 練習問題でよく復習して下さい

7/22の最終課題について

704教室

教卓

この席には座
らないで下さい



二次元配列(復習)

二次元配列
要素の参照, 代入

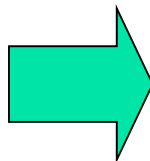
二次元配列の宣言①

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



Python での宣言

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

二次元配列の宣言②

3×3の行列 a

1	2	3
4	5	6
7	8	9

Python での宣言

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

さらに[]で囲む

「,」で区切る

二次元配列の宣言③

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

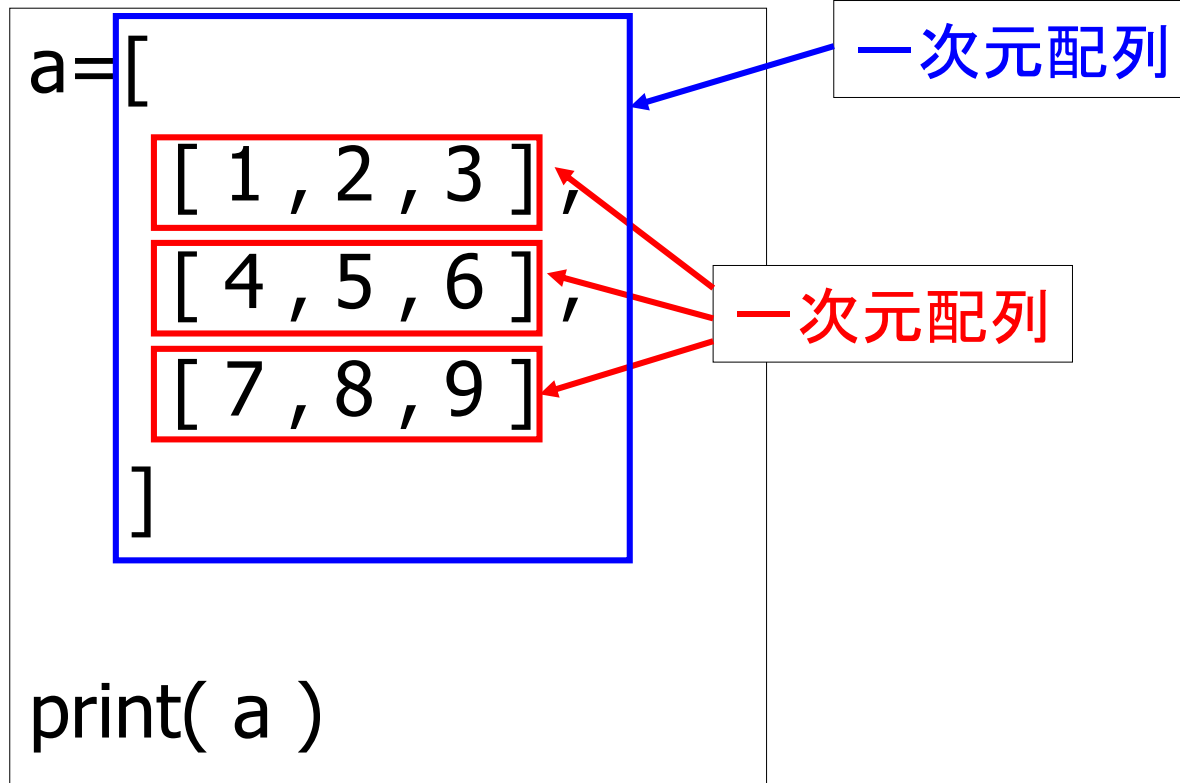
```
print( a )
```

```
>python sample.py  
[[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

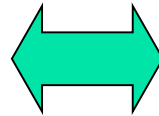
二次元配列の宣言④



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

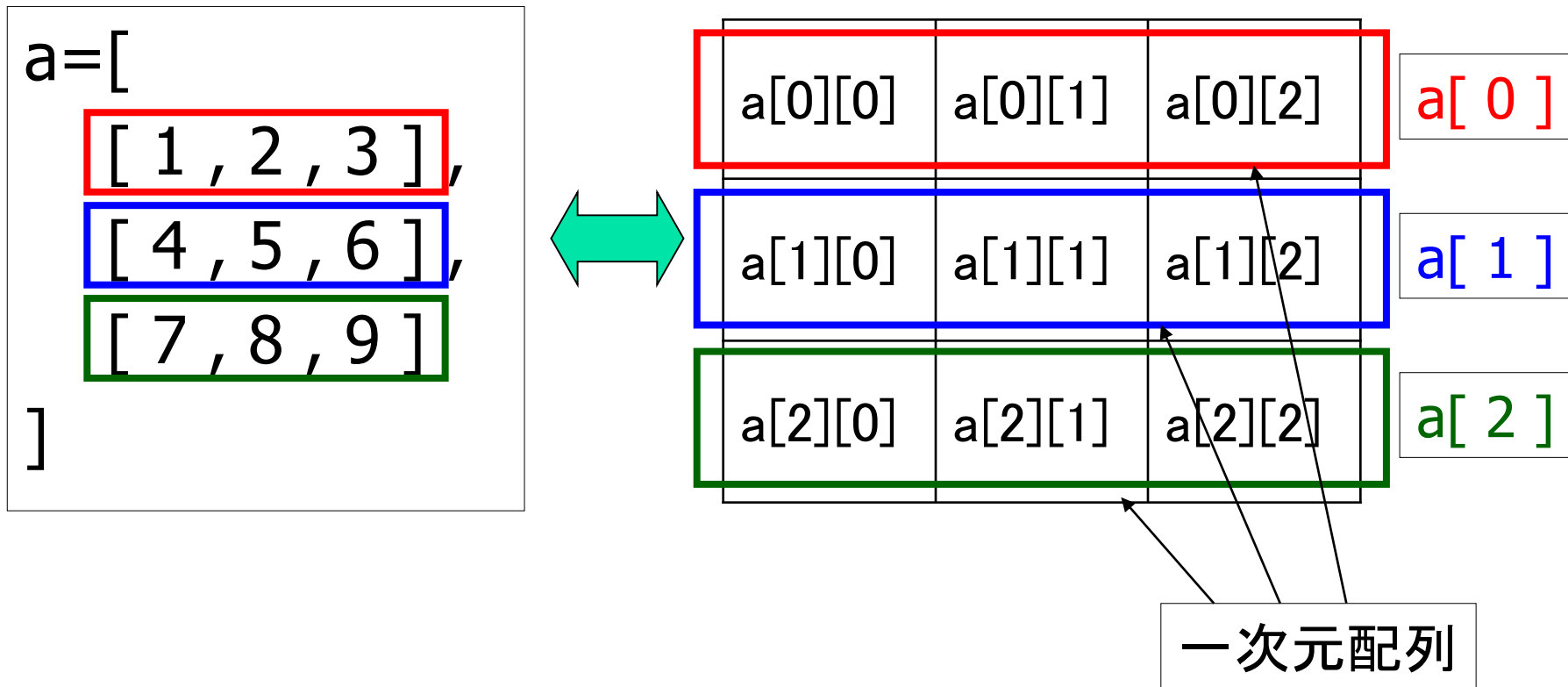
二次元配列の要素の参照①

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



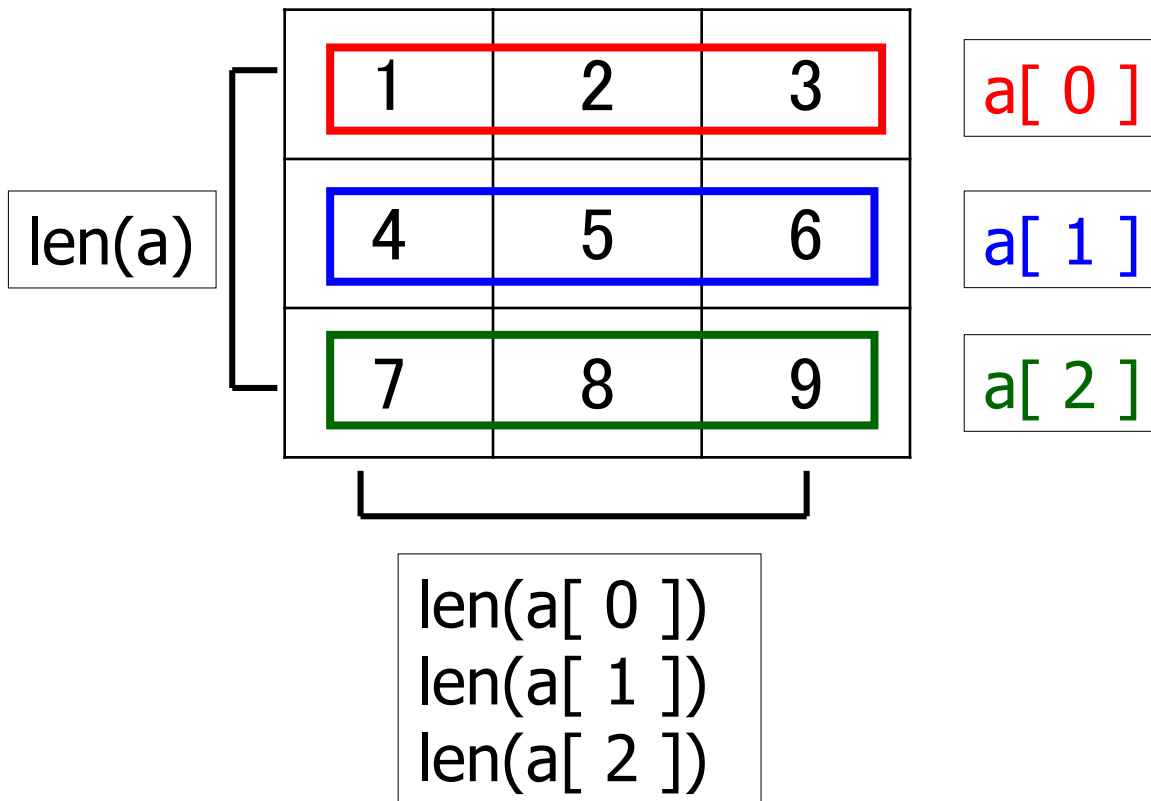
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

二次元配列の要素の参照②



二次元配列の要素の参照③

要素数



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



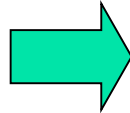
二次元配列の宣言

- 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

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



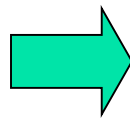
```
a=[  
  [ 1 , 2 , 3 ],  
  [ 4 , 5 , 6 ],  
  [ 7 , 8 , 9 ],  
  [ 10 , 11 , 12 ]  
]
```

二次元配列の宣言②

- 要素数のみ分かっている場合

3列

4行



```
a =[0]*4
```

```
a[ 0 ] = [0]*3
```

```
a[ 1 ] = [0]*3
```

```
a[ 2 ] = [0]*3
```

```
a[ 3 ] = [0]*3
```



二次元配列の要素への代入

```
a =[0]*4  
a[ 0 ] = [0]*3  
a[ 1 ] = [0]*3  
a[ 2 ] = [0]*3  
a[ 3 ] = [0]*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
```

```
print( a )
```

```
>python sample.py
```

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


二次元配列の宣言③

aは配列と宣言

```
a = []
```

```
a.append([])  
a[0].append(1)  
a[0].append(2)  
a[0].append(3)
```

```
a.append([])  
a[1].append(4)  
a[1].append(5)  
a[1].append(6)
```

```
a.append([])  
a[2].append(7)  
a[2].append(8)  
a[2].append(9)
```

```
a.append([])  
a[3].append(10)  
a[3].append(11)  
a[3].append(12)
```

```
print( a )
```

a[0], a[1], a[2],
a[3]を配列として
追加

```
>python sample.py  
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```



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

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]  
]
```

```
for i in range(4):  
    for j in range(3):  
        print( " a[" , i , "][" , j , "] =" , a[ i ][ j ] )
```

$i=0, j=0\sim 2$

$i=1, j=0\sim 2$

$i=2, j=0\sim 2$

$i=3, j=0\sim 2$

>python sample.py

```
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

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 ]  
]
```

len(a)の値は4

```
for i in range(len(a)):  
    for j in range(len(a[i])):  
        print( " a[" , i , "][" , j , "] =" , a[ i ][ j ] )
```

len(a[0]), len(a[1]), len(a[2]), len(a[3])は全て3

```
>python sample.py
```

```
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
```



二次元配列の要素の参照③

whileを用いての参照

```
a=[  
    [ 1 , 2 , 3 ],  
    [ 4 , 5 , 6 ],  
    [ 7 , 8 , 9 ],  
    [ 10 , 11 , 12 ]  
]  
  
i = 0  
while i < len(a):  
    j = 0  
    while j < len(a[i]):  
        print( " a[" , i , "][" , j , "]" = " , a[ i ][ j ] )  
        j+=1  
    i+=1
```

```
>python sample.py
```

```
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
```

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 ]  
]
```

x にはa[0],a[1],a[2],a[3]が代入されます

```
for x in a:  
    print( x )  
    for j in x:  
        print( j )
```

jには配列の値が代入されます

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

```
for x in a:  
    print( x )  
    for j in range(len(x)):  
        print( x[ j ] )
```

jには0,1,2が代入されます

二次元配列の要素への代入①

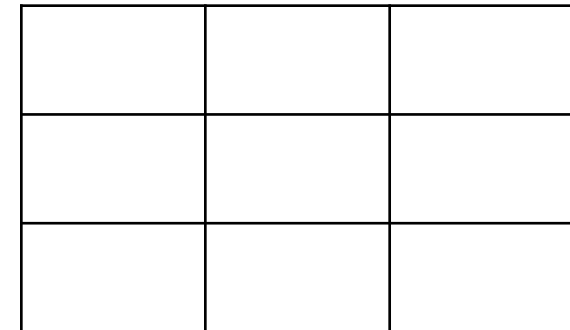
```
a=[0]*3
```

```
a[ 0 ] = [0]*3
```

```
a[ 1 ] = [0]*3
```

```
a[ 2 ] = [0]*3
```

3×3の要素(値は0)を持つ
二次元配列を宣言




```
count = 1
```

```
for i in range(3):
```

```
    for j in range(3):
```

```
        a[i][j] = count
```

```
        count += 1
```

代入式

```
print( a )
```

```
>python sample.py
```

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

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=[0]*3
```

要素数を3個と指定

```
count = 1
```

```
for i in range(3):
```

```
    a[ i ] = [0]*3
```

```
    for j in range(3):
```

```
        a[ i ][ j ] = count
```

```
        count += 1
```

```
print( a )
```

```
a=[]
```

配列の要素を追加

```
count = 1
```

```
for i in range(3):
```

```
    a.append([])
```

```
    for j in range(3):
```

```
        a[ i ].append(count)
```

```
        count += 1
```

```
print( a )
```


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
```

```
for i in range(3):
```

```
    a.append([])
```

```
    a[ i ]=[0]*3
```

```
    for j in range(3):
```

```
        a[ i ][ j ] = count
```

```
        count += 1
```

```
print( a )
```

a[i]を配列として追加

要素数を3個と指定

```
>python sample.py  
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

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
```

```
for i in range(3):
```

```
    a[ i ]=[0]*3
```

```
    for j in range(3):
```

```
        a[ i ][ j ] = count
```

```
        count += 1
```

```
print( a )
```

要素数を3個と指定

→ a[i]を配列と宣言していない

```
>python sample.py
```

```
Traceback (most recent call last):
```

```
File "C:/Users/shino/Desktop/sample.py", line 6, in <module>
```

```
    a[ i ]=[0]*3
```

```
IndexError: list assignment index out of range
```

二次元配列の要素への代入②

```
a= [0]*4

for i in range(4):
    a[ i ] = [0]*4
    for j in range(4):
        if i == j:
            a[ i ][ j ] = 1
        else:
            a[ i ][ j ] = 0

for i in range(4):
    for j in range(4):
        print( a[ i ][ j ] , end=" " )
    print()
```

i と j が同じ場合は 1
それ以外は 0

```
>python sample.py
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

二次元配列の要素への代入③

```
a=[]

for i in range(4):
    a.append([])
    for j in range(i+1):
        if i == j:
            a[ i ].append(1)
        else:
            a[ i ].append(0)

for i in range(4):
    for j in range(i+1):
        print( a[ i ][ j ] , end=" " )
    print()
```

要素数が異なってもよい

```
>python sample.py
1
0 1
0 0 1
0 0 0 1
```

二次元配列の要素への代入③'

```
a=[]
```

```
for i in range(4):
```

```
    a.append([])
```

```
        for j in range(i+1):
```

```
            if i == j:
```

```
                a[ i ].append(1)
```

```
            else:
```

```
                a[ i ].append(0)
```

```
for i in range(4):
```

```
    for j in range(i+1):
```

```
        print( a[ i ][ j ] , end=" " )
```

```
    print()
```

i = 0 の場合

→ for j in range(1)

→ 1回だけのループ

→ a[0] は要素数1個

i = 1 の場合

→ for j in range(2)

→ 2回だけのループ

→ a[1] は要素数2個

i = 2 の場合

→ for j in range(3)

→ 3回だけのループ

→ a[2] は要素数3個

i = 3 の場合

→ for j in range(4)

→ 4回だけのループ

→ a[3] は要素数4個

二次元配列の要素への代入④

```
a=[]

for i in range(4):
    a.append([])
    for j in range(4-i):
        if i+j == 3:
            a[ i ].append(1)
        else:
            a[ i ].append(0)

for i in range(4):
    for j in range(4-i):
        print( a[ i ][ j ] , end=" " )
    print()
```

どうして配列の要素がこのようになるのでしょうか？

```
>python sample.py
0 0 0 1
0 0 1
0 1
1
```



成績表の処理

二次元の表の計算



2次元表で平均値を求める①

- 4人の平均点を求める

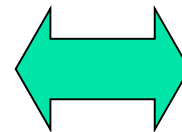
- 出席番号1番: $(70+60+83) / 3 = 71$

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

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



各人の平均点を求めるプログラム①

```
p = [  
    [1,70,60,83],  
    [2,43,49,76],  
    [3,59,79,43],  
    [4,67,74,83]  
]  
  
for i in range(4):  
    sum = 0  
    for j in range(1,4):  
        sum += p[i][j]  
    ave = sum // 3  
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```

```
>python sample.py
```

```
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]

```
for i in range(4):
```

```
    sum = 0
```

```
    for j in range(1,4):
```

```
        sum += p[i][j]
```

```
    ave = sum // 3
```

```
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```

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

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

```
for i in range(4):  
    sum = 0  
    for j in range(1,4):  
        sum += p[i][j]  
    ave = sum / 3  
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```

```
>python sample.py
```

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 60.33333333333333 です

出席番号 4 の人の平均点は 74.66666666666667 です



各人の平均点を求めるプログラム②

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

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

```
for i in range(4):  
    sum = 0  
    for j in range(1,4):  
        sum += p[i][j]  
    ave = sum // 3  
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```



各人の平均点を求めるプログラム②'

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

```
>python sample.py
```

```
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
```

```
for i in range(len(p)):  
    sum = 0  
    for j in range(1,len(p[i])):  
        sum += p[i][j]  
    ave = sum // (len(p[i])-1)  
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```



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]  
]
```

```
for i in range(len(p)):  
    sum = 0  
    for j in range(1,len(p[i])):  
        sum += p[i][j]  
    ave = sum // (len(p[i])-1)  
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave ) )
```

```
>python sample.py
```

```
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

```
p = [  
    [1,70,60,83],  
    [2,43,49,76],  
    [3,59,79,43],  
    [4,67,74,83],  
]  
  
for a in p:  
    sum = 0  
    for i in range(1,len(a)):  
        sum += a[ i ]  
    ave = sum // (len(a)-1)  
    print( "出席番号 {0} の人の平均点は {1} です".format( a[0] , ave ) )
```

```
>python sample.py
```

```
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

```
for a in p:
    sum = 0
    for i in range(1, len(a)):
        sum += a[i]
    ave = sum // (len(a)-1)
    print( "出席番号 {0} の人の平均点
    は {1} です".format( a[0], ave ) )
```

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

len(a) 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'

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

また、別の書き方ですが...

```
for a in p:  
    sum = 0  
    count = 0  
    for i in a:  
        if count > 0:  
            sum += i  
        count+=1  
    ave = sum // (len(a)-1)  
    print( "出席番号 {0} の人の平均点は {1} です".format( a[0] , ave ) )
```

```
>python sample.py
```

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'

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

```
for a in p:
    sum = 0
    count = 0
    for i in a:
        if count > 0:
            sum += i
        count += 1
    ave = sum // (len(a)-1)
    print( "出席番号 {0} の人の平均点は {1} です".format( a[0] , ave ) )
```

iにはa[0],a[1],a[2],a[3]の値が直接
代入される

各人の平均点を求めるプログラム④

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

```
ave=[]
```

```
for i in range(len(p)):
```

```
    sum = 0
```

```
    for j in range(1,len(p[i])):
```

```
        sum += p[i][j]
```

```
    ave.append( sum // (len(p[i])-1) )
```

```
    print( "出席番号 {0} の人の平均点は {1} です".format( p[i][0] , ave[i] ) )
```

```
>python sample.py
```

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

- 結果を求めるまでには一通りではない
- 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

```
import math
```

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

```
ave=[]
```

```
for i in range(len(p)):
```

```
    sum = 0
```

```
    for j in range(1,len(p[i])):
```

```
        sum += p[i][j]
```

```
    ave.append( sum // (len(p[i])-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)

```
for i in range(len(p)):
```

```
    print( "出席番号 {0} の人の平均点は {1},標準偏差は{2}です  
          ".format( p[i][0] , ave[i] , sd[i]) )
```

```
>python sample.py
```

出席番号 1 の人の平均点は 71,標準偏差は9.41629792788369です

出席番号 2 の人の平均点は 56,標準偏差は14.352700094407323です

出席番号 3 の人の平均点は 60,標準偏差は14.730919862656235です

出席番号 4 の人の平均点は 74,標準偏差は6.582805886043833です



エラステネスの篩

素数を見つける方法(アルゴリズム)



素数を判定するプログラム①

```
n=int(input())
```

```
for x in range(2,n):
```

```
    if n % x == 0:
```

```
        print( "この数字は素数ではありません" )
```

```
        break
```

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

```
>python sample.py
```

```
153
```

```
この数字は素数ではありません
```

素数を入力した場合

```
>python sample.py
```

```
37
```

素数の場合→「素数です」と出力するには？

```
n=int(input())
```

p=1としておく(重要！)

```
p=1
```

```
for x in range(2,n):
```

```
    if n % x == 0:
```

```
        p = 0
```

```
        break
```

素数でないと判明した場合p=0とする

```
if p == 0:
```

```
    print( "この数字は素数ではありません" )
```

```
else:
```

```
    print( "この数字は素数です" )
```

```
>python sample.py
```

```
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

素数の場合→「素数です」と出力するには？ (Pythonらしいプログラム)

```
n=int(input())

for x in range(2,n):
    if n % x == 0:
        print( "この数字は素数ではありません" )
        break
else:
    print( "この数字は素数です" )
```

```
>python sample.py
179
この数字は素数です
```

breakでfor文が終了しなかった場合、
else文が実行される



再：素数を印字するプログラム

```
# 素数は、2～「自分-1」では割り切れない整数
# 2から20までの素数を印字しよう
for n in range(2,21):
    p=1
    for x in range(2,n):
        if n % x == 0:
            p = 0
            break
    if p == 1:
        print( n , "は素数" )
```

```
> python sample.py
2 は素数
3 は素数
5 は素数
7 は素数
11 は素数
13 は素数
17 は素数
19 は素数
```

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

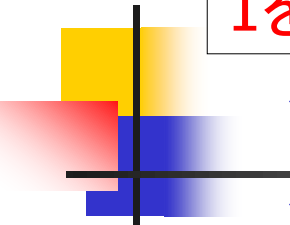


1から100までの配列を用意(配列名はsieveとします)

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

1を削除(0とする)

要素 2×2 から2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

エラトステネスの篩: プログラム例①

```
sieve = [1]*101
```

101個の配列を用意
sieve[0] は利用しない

```
sieve[0]=0
```

```
sieve[1]=0
```

1は素数でない
→ 0とする

初期設定
配列の全要素に1を代入

```
i=2
```

iの値は2

```
for j in range(i*2,len(sieve),i):
```

```
    sieve[j]=0
```

素数でないことをチェック

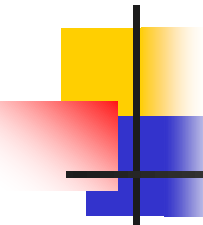
jは2*iから100まで
, iずつ加算される

```
for i in range(len(sieve)):
```

```
    if sieve[i] !=0:
```

```
        print( i , end=" " )
```

要素が0でなければ
素数と判定



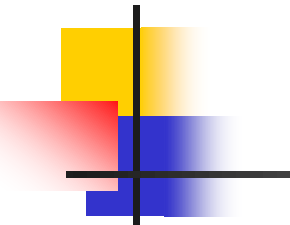
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

2の倍数
を削除

```
> python sample.py
```

```
2 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45
47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87
89 91 93 95 97 99
```

要素 3×2 から3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

エラトステネスの篩: プログラム例②

```
sieve = [1]*101
```

101個の配列を用意
sieve[0] は利用しない

```
sieve[0]=0
```

```
sieve[1]=0
```

1は素数でない
→ 0とする

初期設定
配列の全要素に1を代入

```
for i in range(2,4):
```

iの値は2,3

```
    for j in range(i*2,len(sieve),i):
```

```
        sieve[j]=0
```

素数でないことをチェック

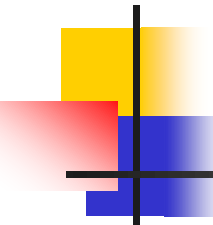
jは2*iから100まで
, iずつ加算される

```
for i in range(len(sieve)):
```

```
    if sieve[i] !=0:
```

```
        print( i , end=" " )
```

要素が0でなければ
素数と判定



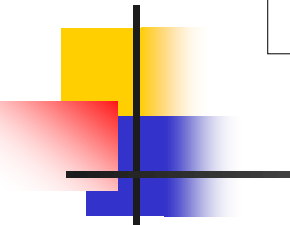
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

2と3の倍
数を削除

```
>python sample.py
```

```
2 3 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53 55
59 61 65 67 71 73 77 79 83 85 89 91 95 97
```

(無駄ですが...)要素 4×2 から4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

エラトステネスの篩: プログラム例③

```
sieve = [1]*101
```

101個の配列を用意
sieve[0] は利用しない

```
sieve[0]=0
```

```
sieve[1]=0
```

1は素数でない
→ 0とする

初期設定
配列の全要素に1を代入

```
for i in range(2,5):
```

iの値は2,3,4

```
    for j in range(i*2,len(sieve),i):
```

```
        sieve[j]=0
```

素数でないことをチェック

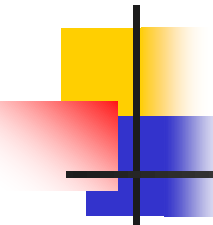
jは2*iから100まで
, iずつ加算される

```
for i in range(len(sieve)):
```

```
    if sieve[i] !=0:
```

```
        print( i , end=" " )
```

要素が0でなければ
素数と判定



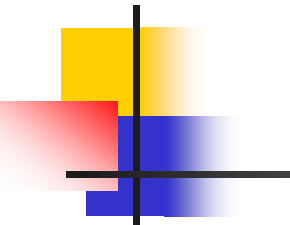
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

2,3,4の倍
数を削除

```
>python sample.py
```

```
2 3 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53 55
59 61 65 67 71 73 77 79 83 85 89 91 95 97
```

要素 5×2 から5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

エラトステネスの篩: プログラム例④

```
sieve = [1]*101
```

101個の配列を用意
sieve[0] は利用しない

```
sieve[0]=0
```

```
sieve[1]=0
```

1は素数でない
→ 0とする

初期設定
配列の全要素に1を代入

```
for i in range(2,6):
```

iの値は2,3,4,5

```
    for j in range(i*2,len(sieve),i):
```

```
        sieve[j]=0
```

素数でないことをチェック

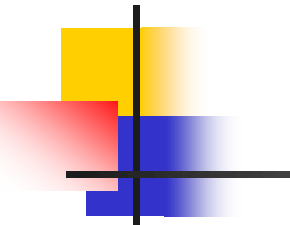
jは2*iから100まで
, iずつ加算される

```
for i in range(len(sieve)):
```

```
    if sieve[i] !=0:
```

```
        print( i , end=" " )
```

要素が0でなければ
素数と判定



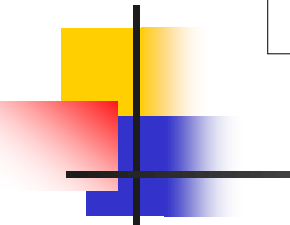
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

2,3,4,5
の倍数
を削除

```
> python sample.py
```

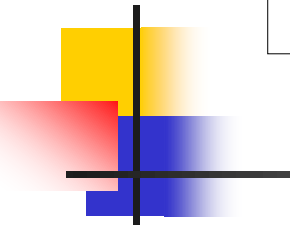
```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 49
53 59 61 67 71 73 77 79 83 89 91 97
```

(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

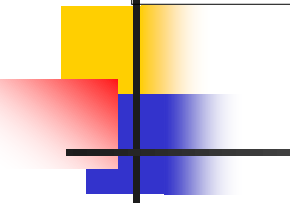
要素 7×2から7要素ごとに削除



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

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？

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



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



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 # ???

エラトステネスの篩①

```
sieve = [1]*101
```

101個の配列を用意
sieve[0] は利用しない

```
sieve[0]=0
```

```
sieve[1]=0
```

初期設定
配列の全要素に1を代入

```
for i in range(2,len(sieve)):  
    for j in range(i*2,len(sieve),i):  
        sieve[j]=0
```

iは2から100まで変わる

素数でないことをチェック

jは2*iから100まで
, iずつ加算される

```
for i in range(len(sieve)):  
    if sieve[i] !=0:  
        print( i , end=" " )
```

要素が0でなければ
素数と判定



エラトステネスの篩②

```
sieve = [1]*101
for i in range(len(sieve)):
    sieve[i]=i
sieve[0]=0
sieve[1]=0

for i in range(2,len(sieve)):
    for j in range(i*2,len(sieve),i):
        sieve[j]=0

for p in sieve:
    if p != 0:
        print( p , end=" " )
```

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

```
>python sample.py
```

```
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

```
>python sample.py
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131  
137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263  
269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409  
419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569  
571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719  
727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881  
883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

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？

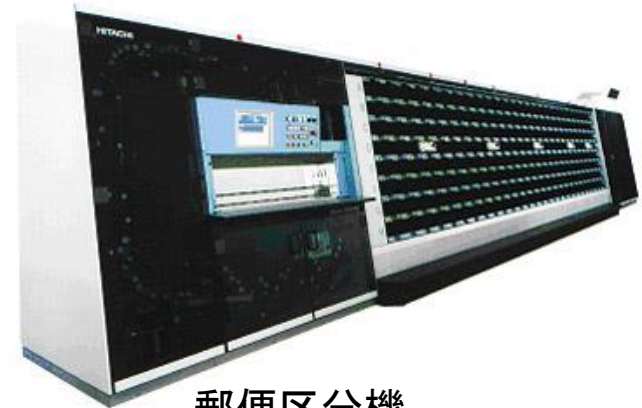


ソーティング

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
 - ところで、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



郵便区分機

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_034.html

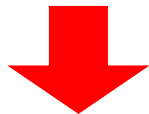


ソーティングとは

- 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



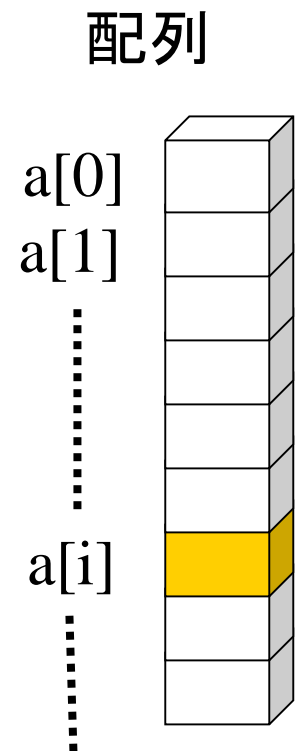
ソートする(昇順)

2, 3, 4, 5, 5, 6

ソートिंगの基本操作

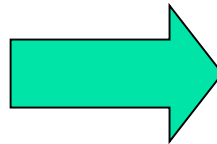
- 本日扱うソートिंगの対象となるデータ列は、一つの配列に入っているものとする.

- ・ ソートिंगの基本操作
 - 配列要素間の
 - 比較操作と
 - 交換(移動)操作



ソートリングの例

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	



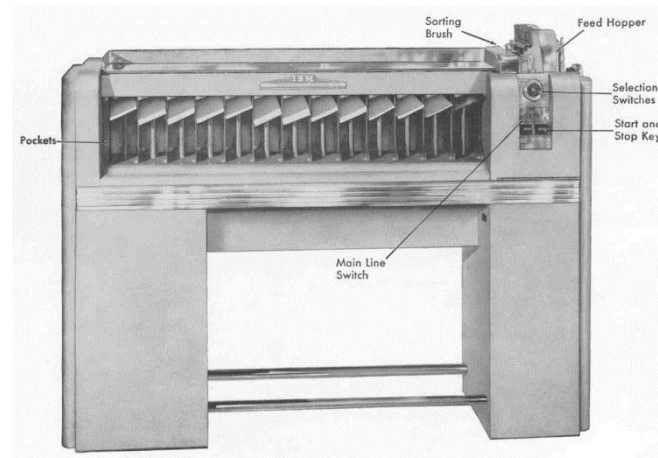
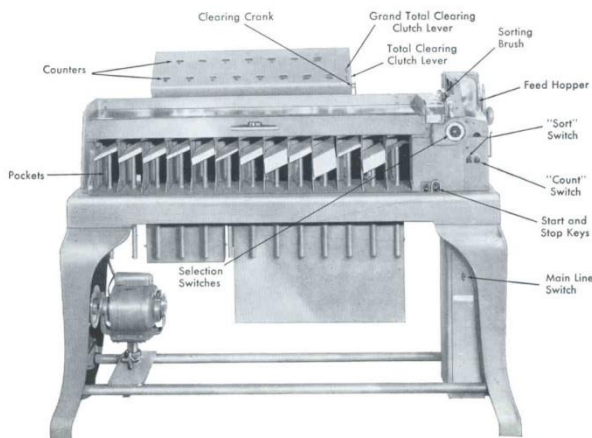
ソーティングのアルゴリズム

- 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

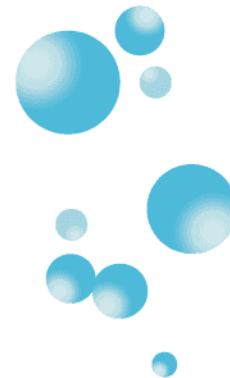
- 実は、区分機を使うと、昇順なり降順に並べることができる
 - 「できる」だけではなく、実際にそうしていた
- たとえば、10進3桁のID番号が振られているカードがあったとしよう
 - 一の位で、0,1,...,9に区分けする
 - 0~9の順に重ねると、下一桁では、0~9の順に並んでいる
 - その順序を崩さずに、十の位で、0,1,...,9に区分けする
 - 0~9の順に重ねると、下二桁では、00~99の順に並んでいる
 - その順序を崩さずに、百の位で、0,1,...,9に区分けする
 - 0~9の順に重ねると、下三桁では、000~999の順に並んでいる





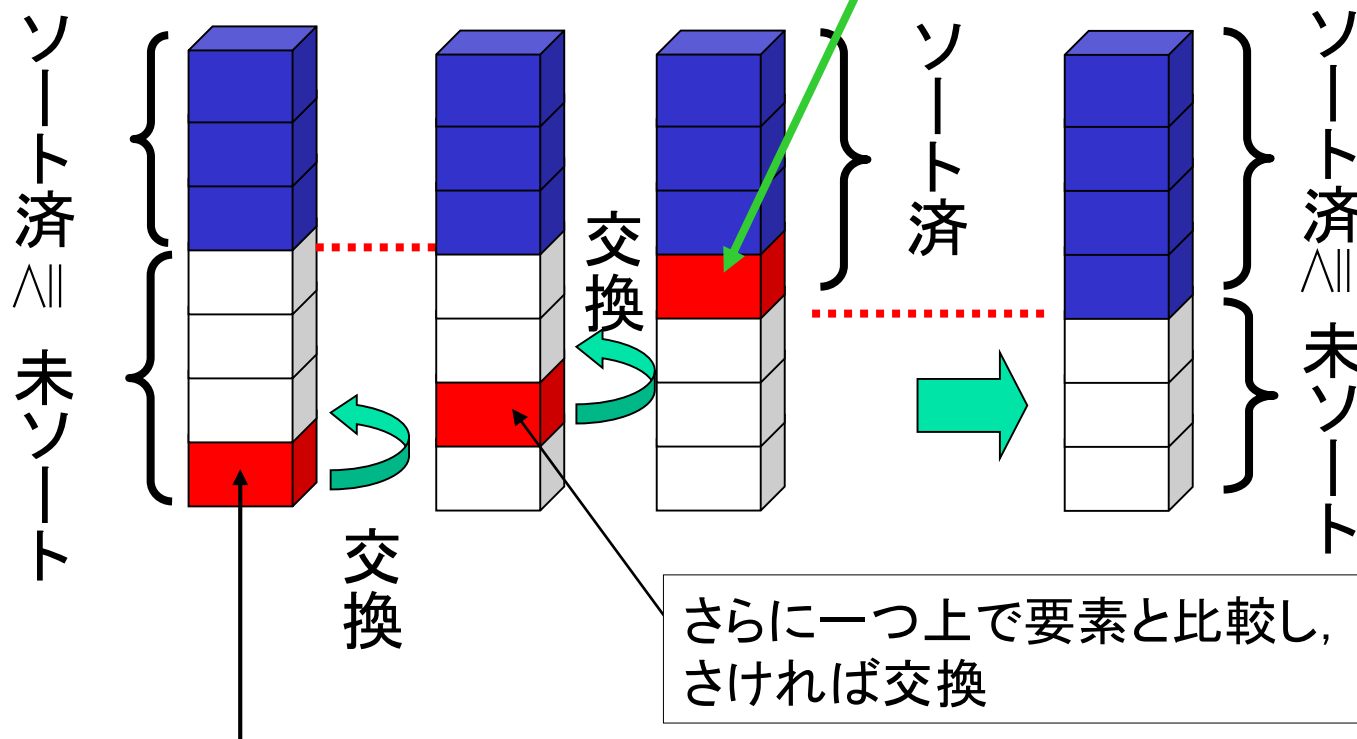
ソート: bubble sort

- 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

未ソート領域の中での最小値が上に上がっていく



配列の最後の要素(一番下)から一つ上の要素と比較
下の方が小さければ交換



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
----	---	----	----	----	----	----	----	----	----

バブルソートのプログラム①

i=0の場合

一番小さい値を確定する

```
p=[60,2,11,82,29,21,24,98,51,24]
```

i = 0

```
print(p)
```

```
for j in range(len(p)-1,i,-1):
```

```
    if p[j] < p[j-1]:  
        work = p[j]  
        p[j] = p[j-1]  
        p[j-1] = work
```

```
    print( p )
```

配列の一番下から隣接間の大小を調べていく

jはlen(p)から1の範囲

隣接する配列同士を比較
→下の要素が小さい場合、交換

バブルソートのプログラム①

実行結果

```
> python sort.py
```

```
[60, 2, 11, 82, 29, 21, 24, 98, 51, 24]
```

51と24を比較
→ 24の方が小さいので交換

```
[60, 2, 11, 82, 29, 21, 24, 98, 24, 51]
```

98と24を比較
→ 24の方が小さいので交換

```
[60, 2, 11, 82, 29, 21, 24, 24, 98, 51]
```

```
[60, 2, 11, 82, 29, 21, 24, 24, 98, 51]
```

29と21を比較
→ 21の方が小さいので交換

```
[60, 2, 11, 82, 29, 21, 24, 24, 98, 51]
```

```
[60, 2, 11, 82, 21, 29, 24, 24, 98, 51]
```

82と21を比較
→ 21の方が小さいので交換

```
[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]
```

60と2を比較
→ 2の方が小さいので交換

```
[2, 60, 11, 21, 82, 29, 24, 24, 98, 51]
```

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

バブルソートのプログラム②

i=1の場合

二番目に小さい値を確定する

配列pはi=0の実行後の状態

```
p=[2, 60, 11, 21, 82, 29, 24, 24, 98, 51]
```

```
i = 1
```

```
print(p)
```

```
for j in range(len(p)-1,i,-1):
```

```
    if p[j] < p[j-1]:  
        work = p[j]  
        p[j] = p[j-1]  
        p[j-1] = work
```

```
print( p )
```

配列の一番下から隣接間の大小を調べていく

jはlen(p)から2の範囲

隣接する配列同士を比較
→下の要素が小さい場合、交換

バブルソートのプログラム②

実行結果

```
> python sort.py
```

```
[2, 60, 11, 21, 82, 29, 24, 24, 98, 51]
```

98と51を比較
→ 24の方が小さいので交換

```
[2, 60, 11, 21, 82, 29, 24, 24, 51, 98]
```

29と24を比較
→ 24の方が小さいので交換

```
[2, 60, 11, 21, 82, 29, 24, 24, 51, 98]
```

```
[2, 60, 11, 21, 82, 29, 24, 24, 51, 98]
```

82と24を比較
→ 24の方が小さいので交換

```
[2, 60, 11, 21, 82, 29, 24, 24, 51, 98]
```

```
[2, 60, 11, 21, 24, 82, 29, 24, 51, 98]
```

```
[2, 60, 11, 21, 24, 82, 29, 24, 51, 98]
```

60と11を比較
→ 11の方が小さいので交換

```
[2, 60, 11, 21, 24, 82, 29, 24, 51, 98]
```

```
[2, 11, 60, 21, 24, 82, 29, 24, 51, 98]
```

確定

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

バブルソートのプログラム③

i=2の場合

三番目に小さい値を確定する

配列pはi=1の実行後の状態

p=[2, 11, 60, 21, 24, 82, 29, 24, 51, 98]

i = 2

print(p)

for j in range(len(p)-1,i,-1):

```
if p[j] < p[j-1]:  
    work = p[j]  
    p[j] = p[j-1]  
    p[j-1] = work
```

print(p)

配列の一番下から隣接間の大小を調べていく

jはlen(p)から3の範囲

隣接する配列同士を比較
→下の要素が小さい場合、交換

バブルソートのプログラム③

実行結果

```
> python sort.py
```

```
[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, 60, 21, 24, 24, 82, 29, 51, 98]
```

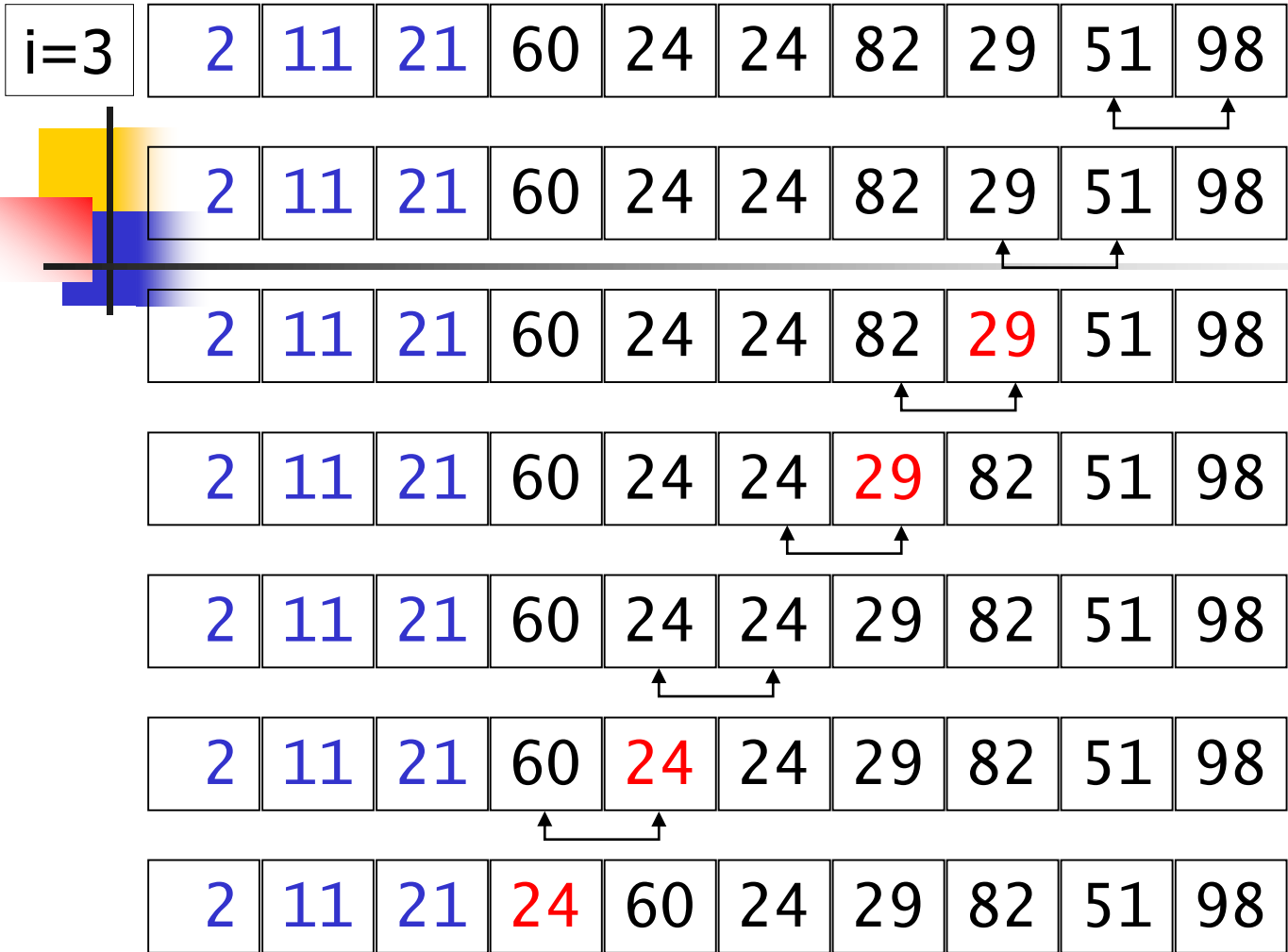
```
[2, 11, 21, 60, 24, 24, 82, 29, 51, 98]
```

29と24を比較
→ 24の方が小さいので交換

82と24を比較
→ 24の方が小さいので交換

60と21を比較
→ 21の方が小さいので交換

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

確定

バブルソートのプログラム④

i=3の場合

四番目に小さい値を確定する

配列pはi=2の実行後の状態

p=[2, 11, 21, 60, 24, 24, 82, 29, 51, 98]

i = 3

print(p)

for j in range(len(p)-1,i,-1):

```
if p[j] < p[j-1]:  
    work = p[j]  
    p[j] = p[j-1]  
    p[j-1] = work
```

print(p)

配列の一番下から隣接間の大小を調べていく

jはlen(p)から4の範囲

隣接する配列同士を比較
→下の要素が小さい場合、交換

バブルソートのプログラム④

実行結果

```
> python sort.py
```

```
[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, 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, 24, 60, 24, 29, 82, 51, 98]
```

82と29を比較
→ 29の方が小さいので交換

60と24を比較
→ 24の方が小さいので交換

確定

以下, 同様...

バブルソートのまとめ①

p=[60,2,11,82,29,21,24,98,51,24]

i = 0

print(p)

for j in range(len(p)-1, i-1):

if p[i] < p[j]:

print(p)

for j in

if p[

print

i = 1

print(p)

for j in range(len(p)-1, i, -1):

if p[j] < p[j-1]:

work = p[j]

p[j] = p[j-1]

p[j-1] = work

print(p)

i = 2

print(p)

for j in range(len(p)-1, i, -1):

if p[j] < p[j-1]:

work = p[j]

p[j] = p[j-1]

p[j-1] = work

print(p)

i = len(p)-1

print(p)

for j in range(len(p)-1, i, -1):

if p[j] < p[j-1]:

work = p[j]

p[j] = p[j-1]

p[j-1] = work

print(p)

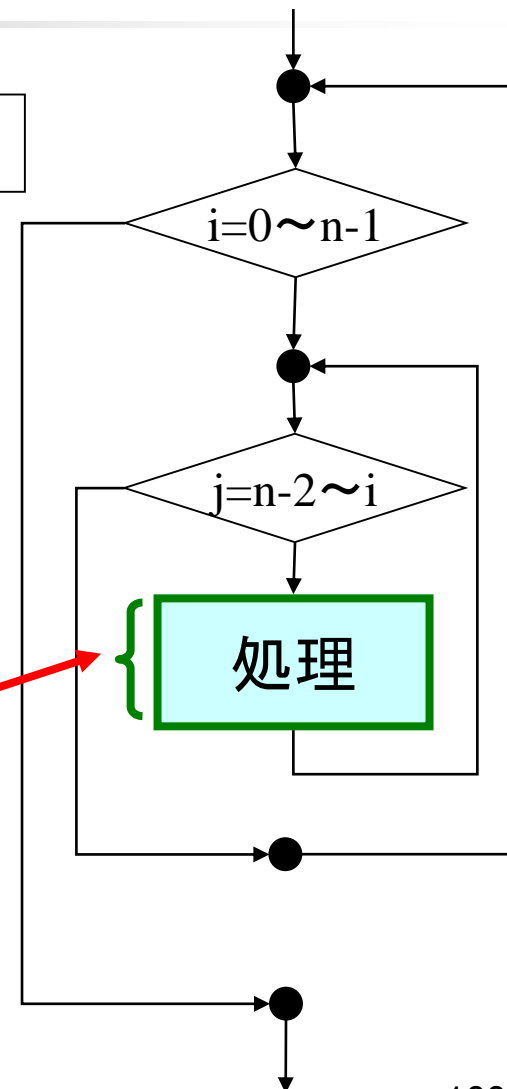
バブルソートのまとめ②

$i=0,1,2,\dots,\text{len}(p)-1$ と繰り返す

```
p=[60,2,11,82,29,21,24,98,51,24]
```

```
for i in range(0,len(p)-1):  
    print(p)  
    for j in range(len(p)-1,i,-1):  
        if p[j] < p[j-1]:  
            work = p[j]  
            p[j] = p[j-1]  
            p[j-1] = work
```

```
print(p)
```





バブルソートの実行結果

```
> python sort.py
```

```
[60, 2, 11, 82, 29, 21, 24, 98, 51, 24]
```

```
[2, 60, 11, 21, 82, 29, 24, 24, 98, 51]
```

```
[2, 11, 60, 21, 24, 82, 29, 24, 51, 98]
```

```
[2, 11, 21, 60, 24, 24, 82, 29, 51, 98]
```

```
[2, 11, 21, 24, 60, 24, 29, 82, 51, 98]
```

```
[2, 11, 21, 24, 24, 60, 29, 51, 82, 98]
```

```
[2, 11, 21, 24, 24, 29, 60, 51, 82, 98]
```

```
[2, 11, 21, 24, 24, 29, 51, 60, 82, 98]
```

```
[2, 11, 21, 24, 24, 29, 51, 60, 82, 98]
```

```
[2, 11, 21, 24, 24, 29, 51, 60, 82, 98]
```

バブルソートの例①

```
p=[60,2,11,82,29,21,24,98,51,24]
print(p)
```

```
for i in range(0,len(p)-1):
    for j in range(len(p)-1,i,-1):
        if p[j] > p[j-1]:
            work = p[j]
            p[j] = p[j-1]
            p[j-1] = work
```

```
print(p)
```

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

```
> python sort.py
[60, 2, 11, 82, 29, 21, 24, 98, 51, 24]
[98, 82, 60, 51, 29, 24, 24, 21, 11, 2]
```

バブルソートの例②

```
import random
```

```
p=[0]*10  
for i in range(len(p)):  
    p[i] = random.randint(0,100)  
print(p)
```

乱数で整数列を生成
→ p[0]～p[9]に格納

```
for i in range(0,len(p)-1):  
    for j in range(len(p)-1,i,-1):  
        if p[j] < p[j-1]:  
            work = p[j]  
            p[j] = p[j-1]  
            p[j-1] = work
```

```
print(p)
```

```
>python sample.py  
[4, 31, 39, 84, 54, 56, 53, 0, 51, 97]  
[0, 4, 31, 39, 51, 53, 54, 56, 84, 97]
```



練習問題

練習問題①から⑤(できるだけ頑張ってください)



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.

```
>python 12-1.py  
国語 の平均点 59  
算数 の平均点 65  
英語 の平均点 71
```



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

$$(p[0][1] + p[1][1] + p[2][1] + p[3][1]) / 4$$

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

$$(p[0][3] + p[1][3] + p[2][3] + p[3][3]) / 4$$

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

```
import math
```

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

```
ave=[]  
for i in range(len(p)):  
    sum = 0  
    for j in range(1,len(p[i])):  
        sum += p[i][j]  
    ave.append( sum // (len(p[i])-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)

```
for i in range(len(p)):
```

```
    print( "出席番号 {0} の人の平均点は {1},標準偏差は{2}です  
          ".format( p[i][0] , ave[i] , sd[i]) )
```

```
>python 12-2.py
```

出席番号 1 の人の平均点は 71,標準偏差は9.41629792788369です

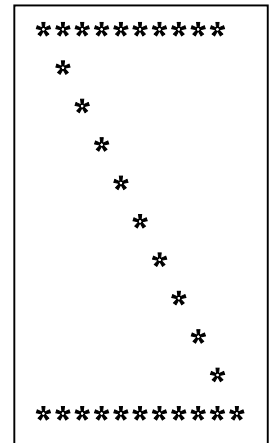
出席番号 2 の人の平均点は 56,標準偏差は14.352700094407323です

出席番号 3 の人の平均点は 60,標準偏差は14.730919862656235です

出席番号 4 の人の平均点は 74,標準偏差は6.582805886043833です

練習問題③

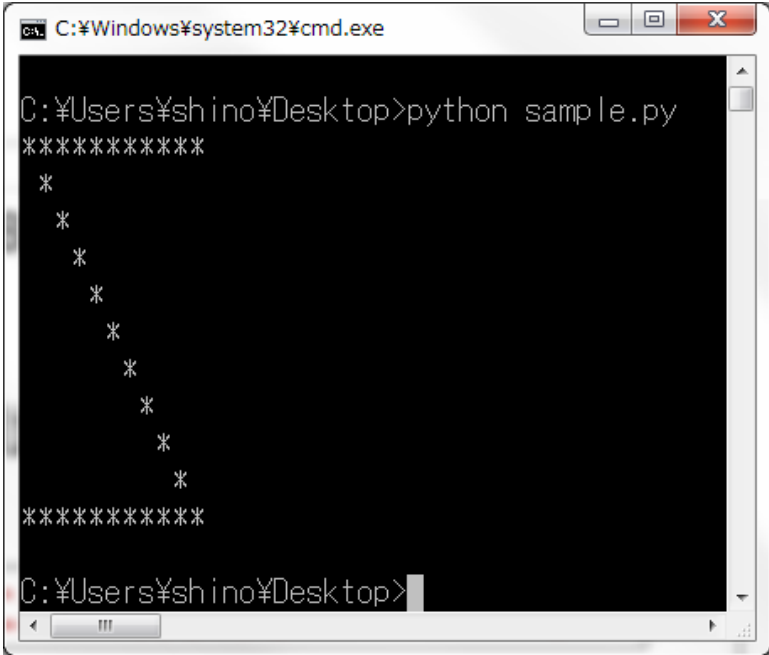
- 縦11文字横11文字(いずれも半角の文字数)の空間に, Z 字を裏返した字を, 半角のアスタリスク(*)を用いて印字してください. 最上行と最下行は, 11文字がアスタリスクになります.
ただし, プログラム中でアスタリスクが出現するのは一回だけにして下さい.
ヒント: 2重のループにすればできます.



```
*****
*
*
*
*
*
*
*
*
*
*
*****
```

練習問題③

- 印字結果



```
C:\¥Windows¥system32¥cmd.exe

C:\¥Users¥shino¥Desktop>python sample.py
*****
*
*
*
*
*
*
*
*
*****

C:\¥Users¥shino¥Desktop>
```

- 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.

$$z = 2x^2 - 4y^2 + 3xy + 6$$

$$x = -2.0, -1.9, -1.8, \dots, 1.8, 1.9, 2.0$$

$$y = -2.0, -1.9, -1.8, \dots, 1.8, 1.9, 2.0$$

```
>python 12-4.py
```

```
xが  yが  の時, 最小値は 
```

練習問題⑤

- 配列p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pに入れ直すことを実現しなさい.
- ただし, 配列p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p

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



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



練習問題⑤

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

```
print(p)
```

```
for i in range(0,len(p)-1):  
    for j in range(len(p)-1,i,-1):  
        if p[j] < p[j-1]:  
            work = p[j]  
            p[j] = p[j-1]  
            p[j-1] = work
```

どこか一行変更してみてください

```
print(p)
```

練習問題⑤(ヒント)

- 配列pの前半を偶数, 後半を奇数と習い変えたいということは...





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