



プログラミング言語 第九回

担当：篠沢 佳久
栗原 聡

2019年 6月17日



本日の内容

- 配列
 - 宣言
 - 代入
 - 要素の参照方法
- 練習問題



配列とは

配列とは
配列の宣言



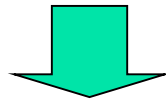
配列の必要性

- $x_1=5, x_2=4, x_3=3, \dots, x_{100}=10$

100個の変数の合計値を求めたい

$$\text{sum} = x_1 + x_2 + x_3 + \dots + x_{100}$$

- 乱数を1,000個生成し、変数に格納し、処理したい



「配列」を利用



今回は配列

- 配列とは、普通、一次元の表、二次元の表、三次元の表、・・・のこと
- Pythonの場合は、ちょっと違う
- 「列」(リストと呼びます)だと思って下さい
 - 値の列, 場所の列



こんな具合です

配列namesの宣言



```
names = ["Perl", "Python", "Ruby", "Java"]  
print( names )
```

```
> python sample.py  
['Perl', 'Python', 'Ruby', 'Java']
```

配列の要素①

```
names = ["Perl", "Python", "Ruby", "Java"]
```

イメージ的には表計算のセル

要素番号は0から始まる

配列名

要素番号
(インデックス)

	names
0	"Perl"
1	"Python"
2	"Ruby"
3	"Java"

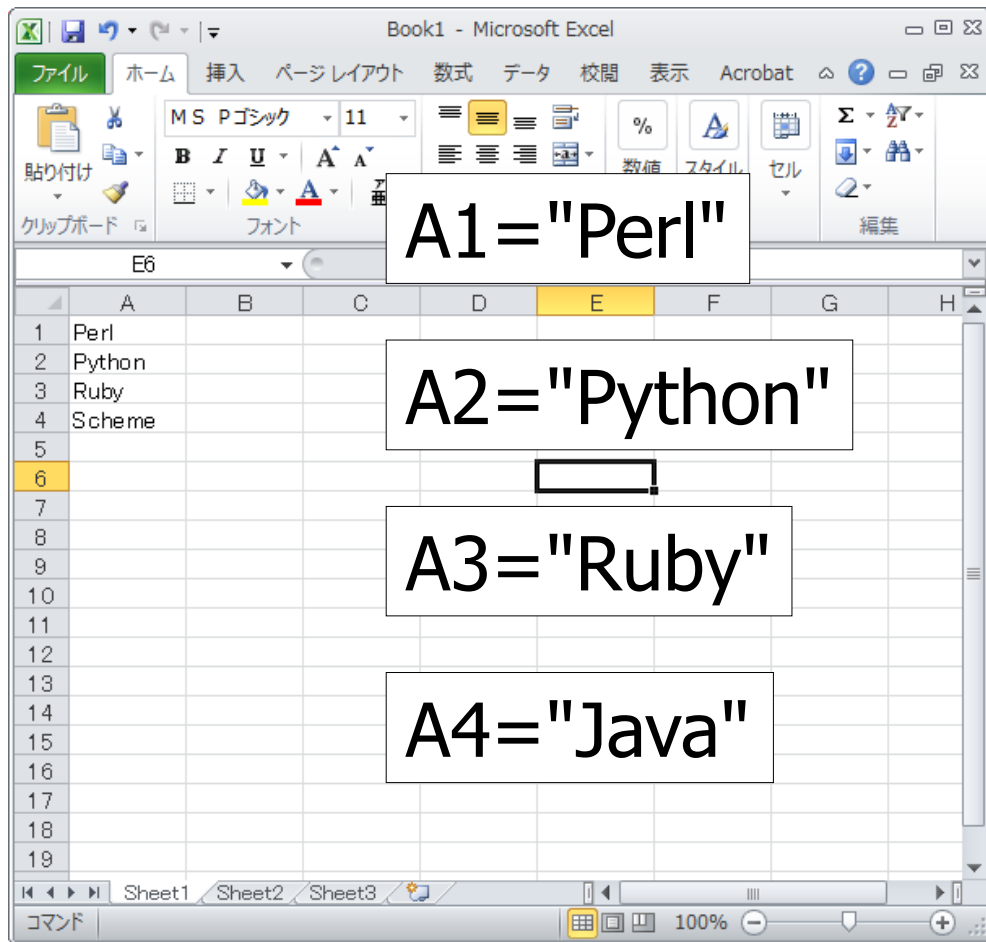
names[0]

names[1]

names[2]

names[3]

(参考) 表計算ソフトのセル



配列の要素②

```
names = ["Perl", "Python", "Ruby", "Java"]  
print( names )  
print( names[0] )  
print( names[1] )  
print( names[2] )  
print( names[3] )
```

```
> python sample.py  
['Perl', 'Python', 'Ruby', 'Java']  
Perl  
Python  
Ruby  
Java
```

names

names[0]

names[1]

names[2]

names[3]



配列の宣言

- `names = ["Perl", "Python", "Ruby", "Java"]`
- `a = [0 , 2 , 4 , 6 , 8]`
- 配列名 = [値1 , 値2 , ... , 値n]

	a
0	0
1	2
2	4
3	6
4	8

配列の要素③

- `a = [0 , 2 , 4 , 6 , 8]`

- 配列の要素

- 配列名[要素番号]

- 配列の要素数

- `len(配列名)`

この配列の`len(a)`の値は5

	a	
0	0	<code>a[0]</code>
1	2	<code>a[1]</code>
2	4	<code>a[2]</code>
3	6	<code>a[3]</code>
4	8	<code>a[4]</code>

```
a=[0,2,4,6,8]  
print( len(a) )
```

```
> python sample.py  
5
```

配列の要素への代入①

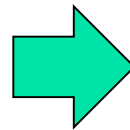
配列名[要素番号] = 値

```
names = ["Perl", "Python", "Ruby", "Java" ]
```

```
names[ 0 ] = "C"
```

```
names[ 3 ] = "Pascal"
```

	names
0	"Perl"
1	"Python"
2	"Ruby"
3	"Java"



	names
0	"C"
1	"Python"
2	"Ruby"
3	"Pascal"



配列の要素への代入②

```
names = ["Perl", "Python", "Ruby", "Java" ]  
print( names )  
names[ 0 ] = "C"  
names[ 3 ] = "Pascal"  
print( names )
```

```
> python sample.py  
['Perl', 'Python', 'Ruby', 'Java']  
['C', 'Python', 'Ruby', 'Pascal']
```

配列の要素への代入③

```
names = ["Perl", "Python", "Ruby", "Java" ]  
print( names )  
names[ 4 ] = "C"  
print( names )
```

配列の末尾に代入(追加)したい

```
> python sample.py  
['Perl', 'Python', 'Ruby', 'Java']  
Traceback (most recent call last):  
  File "C:/Users/shino/Desktop/sample.py", line 5, in <module>  
    names[ 4 ] = "C"  
IndexError: list assignment index out of range
```

配列の末尾に代入することはできない
→ 範囲外の要素を参照した場合エラーとなる

配列の末尾への追加①

```
abc = ["a","b","c"]
```

	abc
0	a
1	b
2	c

```
abc.append("d")
```

	abc
0	a
1	b
2	c
3	d

```
abc.append("e")
```

	abc
0	a
1	b
2	c
3	d
4	e

```
abc = ["a","b","c"]  
print( abc )  
abc.append( "d" )  
print( abc )  
abc.append( "e" )  
print( abc )
```

"d"を追加

"e"を追加

```
> python sample.py  
['a', 'b', 'c']  
['a', 'b', 'c', 'd']  
['a', 'b', 'c', 'd', 'e']
```

配列の末尾への追加②

配列名.**append**(値)

```
x = [ 0 , 1 , 2 , 3 ]
```

```
print( x )
```

```
x.append( 4 )
```

4を追加

```
print( x )
```

```
x.append( 5 )
```

5を追加

```
print( x )
```

```
x.append( 6 )
```

6を追加

```
print( x )
```

```
x.append( 7 )
```

7を追加

```
print( x )
```

```
> python sample.py
```

```
[0, 1, 2, 3]
```

```
[0, 1, 2, 3, 4]
```

```
[0, 1, 2, 3, 4, 5]
```

```
[0, 1, 2, 3, 4, 5, 6]
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```


配列の末尾への追加③

```
x = [ 0 , 1 , 2 , 3 ]  
print( x )  
for i in range( 4,10 ):  
    x.append( i )  
print( x )
```

```
> python sample.py
```

```
[0, 1, 2, 3]
```

```
[0, 1, 2, 3, 4]
```

x.append(4)

```
[0, 1, 2, 3, 4, 5]
```

x.append(5)

```
[0, 1, 2, 3, 4, 5, 6]
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

x.append(9)

配列の途中の要素への追加(挿入)①

```
abc = ["a","b","c"]
```

	abc
0	a
1	b
2	c

```
abc.insert(1,"d")
```

	abc
0	a
1	d
2	b
3	c

```
abc.insert(1,"e")
```

	abc
0	a
1	e
2	d
3	b
4	c

```
abc = ["a","b","c"]  
print( abc )  
abc.insert( 1,"d" )  
print( abc )  
abc.insert( 1,"e" )  
print( abc )
```

```
> python sample.py  
['a', 'b', 'c']  
['a', 'd', 'b', 'c']  
['a', 'e', 'd', 'b', 'c']
```

配列の途中の要素への追加(挿入)②

配列名.**insert**(挿入したい位置, 値)

```
x = [ 0 , 1 , 2 , 3 ]
```

```
print( x )
```

```
x.insert( 1 , 4 )
```

1番に4を挿入

```
print( x )
```

```
x.insert( 1 , 5 )
```

1番に5を挿入

```
print( x )
```

```
x.insert( 1 , 6 )
```

1番に6を挿入

```
print( x )
```

```
x.insert( 1 , 7 )
```

1番に7を挿入

```
print( x )
```

```
> python sample.py  
[0, 1, 2, 3]  
[0, 4, 1, 2, 3]  
[0, 5, 4, 1, 2, 3]  
[0, 6, 5, 4, 1, 2, 3]  
[0, 7, 6, 5, 4, 1, 2, 3]
```

配列の途中の要素への追加(挿入)③

先頭への挿入

```
x = [ 0 , 1 , 2 , 3 ]  
print( x )  
x.insert( 0 , 4 )  
print( x )
```

0番目の位置(先頭)に4を挿入

```
> python sample.py  
[0, 1, 2, 3]  
[4, 0, 1, 2, 3]
```

末尾への挿入(追加)

```
x = [ 0 , 1 , 2 , 3 ]  
print( x )  
x.insert( len(x) , 4 )  
print( x )
```

len(x)=4

```
> python sample.py  
[0, 1, 2, 3]  
[0, 1, 2, 3, 4]
```

4番目の位置に4を挿入

配列の途中の要素への追加(挿入)④

```
x = [ 0 , 1 , 2 , 3 ]  
print( x )  
for i in range( 4,10 ):  
    x.insert( 0,i )  
    print( x )
```

```
> python sample.py
```

```
[0, 1, 2, 3]
```

```
[4, 0, 1, 2, 3]
```

```
[5, 4, 0, 1, 2, 3]
```

```
[6, 5, 4, 0, 1, 2, 3]
```

```
[7, 6, 5, 4, 0, 1, 2, 3]
```

```
[8, 7, 6, 5, 4, 0, 1, 2, 3]
```

```
[9, 8, 7, 6, 5, 4, 0, 1, 2, 3]
```

x.insert(0,4)

x.insert(0,5)

x.insert(0,9)

配列の途中の要素への追加(挿入)⑤

```
x = [ 0 , 1 , 2 , 3 ]  
print( x )  
for i in range( 4,10 ):  
    x.insert( i,i )  
print( x )
```

```
> python sample.py
```

```
[0, 1, 2, 3]
```

```
[0, 1, 2, 3, 4]
```

x.insert(4,4)

```
[0, 1, 2, 3, 4, 5]
```

x.insert(5,5)

```
[0, 1, 2, 3, 4, 5, 6]
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

x.insert(9,9)

各要素に代入する(失敗編)

```
primes[0]=2  
print( primes[0] )
```

注目!

Traceback (most recent call last):

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

```
primes[0]=2
```

NameError: name 'primes' is not defined

あれ？

いろいろややこしい事情があるのです。

Python では(Pythonに限らずどの言語でも), 未定義の変数が使われるとエラー

Python では「使う」以外に現れると, 「これから使うぞ! 」という宣言と考える

Python では, 左辺に現れる以外は, 「使う」ことに相当

従って, 新しい名前を左辺に書くと, 普通は, 「これから使うぞ! 」という宣言になる。

(だから問題は発生しない)

しかし, 配列の要素として現れる(primes[0])と「使う」ことになってしまう

(ないものをいきなり使うことはできない. 使おうとすればエラー!)

配列だということを教える①

- もちろん, 教える相手は Python
- **配列名=[]** と宣言する

primesは配列と宣言	
<pre>primes=[]</pre>	<pre>> python sample.py</pre>
<pre>print(primes)</pre>	<pre>[]</pre>
<pre>primes.append(2)</pre>	<pre>[2]</pre>
<pre>print(primes)</pre>	<pre>2</pre>
<pre>print(primes[0])</pre>	<pre>[2, 4]</pre>
<pre>primes.append(4)</pre>	<pre>4</pre>
<pre>print(primes)</pre>	
<pre>print(primes[1])</pre>	

配列だということを教える②

① `primes = []`

要素を持たない配列

	primes
--	--------

② `primes.append(2)`

	primes
0	2

③ `primes.append(4)`

	primes
0	2
1	4

配列だということを教える③

```
x=[]  
x.append(1)  
x.append(3)  
x.append(5)  
x.append(7)  
x.append(9)  
print( x )  
print( len(x) )
```

xが配列であることを宣言

```
> python sample.py  
[1, 3, 5, 7, 9]  
5
```



```
x=[1,3,5,7,9]
```

配列だということを教える④

`x=[]` ←

xが配列であることを宣言

```
for i in range(10):  
    x.append(i)  
print( x )
```

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

`x=[]` ←

xが配列であることを宣言

```
for i in range(10):  
    x.append(0)  
print( x )
```

`[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

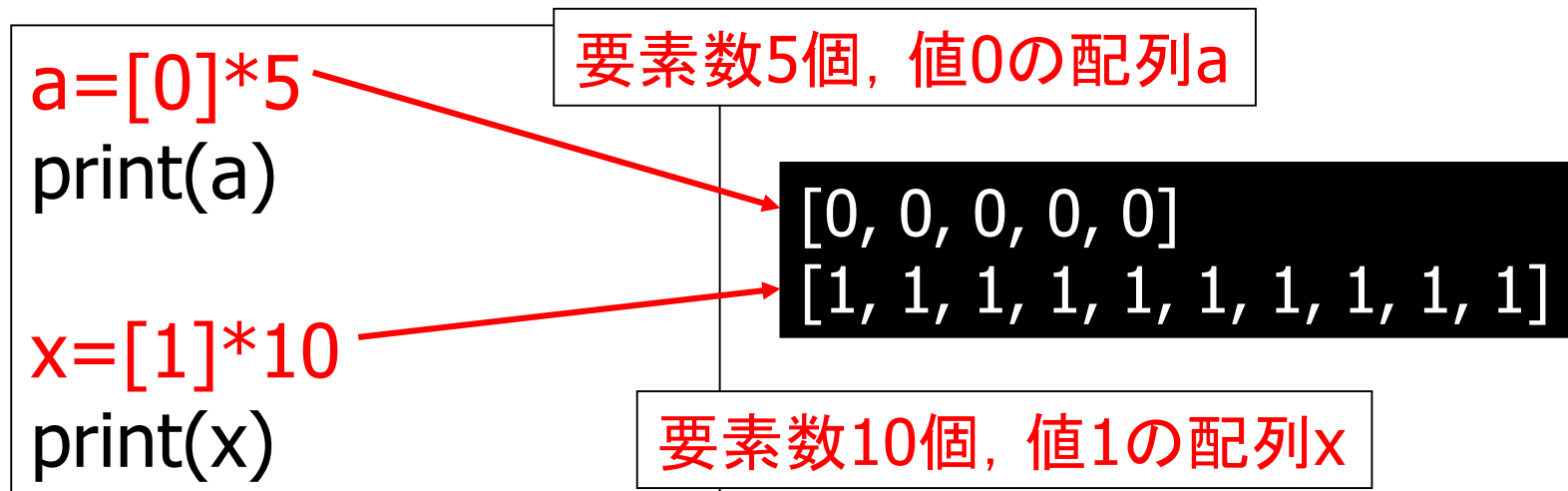


配列の宣言のまとめ①

- 要素が分かっている場合
 - 配列名 = [値1, 値2, ... , 値n]
- 要素が分かっていない場合
 - 配列名 = []
 - appendを用いて要素を追加
- 要素数のみ分かっている場合
 - 配列の初期化(次頁)

配列の宣言のまとめ② (配列の初期化)

- 配列名=[**値**]***n**
 - 要素数**n**個の**値**を持つ配列を作成





配列の宣言のまとめ③

	a
0	3
1	4
2	1

要素が分かっている場合
`a = [3,4,1]`

要素が分かっていない場合
`a=[]`
`a.append(3)`
`a.append(4)`
`a.append(1)`

要素数のみ分かっている場合
`a=[0]*3`
`a[0]=3`
`a[1]=4`
`a[2]=1`



その他の参照方法①

- 配列名[n:m]
 - n番目からm-1番目の要素を参照する
- 配列名[n:]
 - n番目から最後の要素まで参照する
- 配列名[:n]
 - 最初からn-1番目の要素まで参照する



その他の参照方法②

- 配列名[n:m:s]
 - n番目からm-1番目の要素をsごとに参照する

その他の参照方法③

配列	x
0	A
1	B
2	C
3	D
4	E

```
x = ["A","B","C","D","E"]  
print( x[0] )  
print( x[1:4] )  
print( x[2:] )  
print( x[:3] )  
print( x[1:4:2] )
```

A

['B', 'C', 'D']

['C', 'D', 'E']

['A', 'B', 'C']

['B', 'D']

1番目から3番目まで

2番目から最後まで

先頭から2番目まで

1番目から3番目まで2ごと
(1番と3番)



配列の結合

■ 配列1+配列2

```
x = [0,1,2,3,4]
```

```
y = [5,6,7,8,9]
```

```
z = x+y
```

```
print( x )
```

```
print( y )
```

```
print( z )
```

```
[0, 1, 2, 3, 4]
```

```
[5, 6, 7, 8, 9]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



配列と繰り返し

配列の要素の参照方法



配列の要素の参照方法①

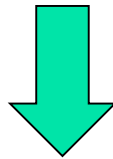
```
names = ["Perl", "Python", "Ruby", "Java"]
```

names[0]～names[3]まで
順番に参照するには？

	names	
0	"Perl"	names[0]
1	"Python"	names[1]
2	"Ruby"	names[2]
3	"Java"	names[3]

配列の要素の参照方法①'

```
names = ["Perl", "Python", "Ruby", "Java"]  
print( names[ 0 ] )  
print( names[ 1 ] )  
print( names[ 2 ] )  
print( names[ 3 ] )
```



繰り返しを用いて記述する



配列の要素の参照方法②

- 配列の要素の参照方法には,
 1. 要素番号を用いて要素の値を取り出す方法
 2. 要素の値を直接取り出す方法
- があります



要素番号を用いて一つずつ取り出す①

```
names = ["Perl", "Python", "Ruby", "Java"]  
for i in range(4):  
    print( names[ i ] )
```

i には0,1,2,3が代入される

names[0],names[1],names
[2],names[3]と参照される

```
>python sample.py  
Perl  
Python  
Ruby  
Java
```

要素番号を用いて一つずつ取り出す②

```
names = ["Perl", "Python", "Ruby", "Java"]  
for i in range(len(names)):  
    print( names[ i ] )
```

len(names)=4

i には0,1,2,3が代入される

names[0],names[1],names[2]
,names[3]と参照される

```
>python sample.py  
Perl  
Python  
Ruby  
Java
```


要素番号を用いて一つずつ取り出す③

```
a=[1,3,5,7,9]
for i in range(5):
    print( a[ i ] )
```

i には0,1,2,3,4が代入される

a[0],a[1],a[2],a[3],a[4]と
参照される

```
a=[1,3,5,7,9]
for i in range(len(a)):
    print( a[ i ] )
```

len(a)=5

```
>python sample.py
1
3
5
7
9
```

要素番号を用いて一つずつ取り出す (続々)

```
a=[1,3,5,7,9]
for i in range( 1,4 ):
    print( a[ i ] )
```

i には1,2,3が代入される
→a[1],a[2],a[3]が参照

```
> python sample.py
3
5
7
```

```
a=[1,3,5,7,9]
for i in range( 0,5,2 ):
    print( a[ i ] )
```

i には0,2,4が代入される
→a[0],a[2],a[4]が参照

```
> python sample.py
1
5
9
```

要素番号を用いて一つずつ取り出す (while文でもできます)

```
a=[1,3,5,7,9]
i=0
while i<len(a):
    print( a[i] )
    i+=1
```

```
a=[1,3,5,7,9]
i=0
while True:
    print( a[i] )
    i+=1
    if i>=len(a):
        break
```

```
>python sample.py
1
3
5
7
9
```



要素番号を用いて一つずつ取り出す

forを用いる場合

	a
0	0
1	2
2	4
3	6
4	8

a[0]

a[1]

a[2]

a[3]

a[4]

この順番に要素を取り出したい

```
for i in range(len(配列)):  
    配列[ i ]の処理
```

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

i は0,1,2,3,4 と代入されるため
a[0], a[1], a[2], a[3], a[4]
となる

要素を直接一つずつ取り出す①

```
for x in 配列:  
    print( x )
```

xに配列[0],配列[1],...が代入される

```
a = [1,3,5,7,9]  
for x in a:  
    print( x )
```

xに1,3,5,7,9が代入される



要素を直接一つずつ取り出す②

```
for x in [値1,値2,...,値n]:  
    print( x )
```

xに値1,値2,...値nが代入される

```
for x in [1,3,5,7,9]:  
    print( x )
```

xに1,3,5,7,9が代入される



要素を直接一つずつ取り出す③

```
a = [1,3,5,7,9]
for x in a:
    print( x )
```

xには1,3,5,7,9と代入される

```
for x in [1,3,5,7,9]:
    print( x )
```

```
>python sample.py
1
3
5
7
9
```



要素を直接一つずつ取り出す④

```
names = ["Perl", "Python", "Ruby", "Java" ]  
for lang in names:  
    print( " I like " , lang )
```

```
> python sample.py  
I like Perl  
I like Python  
I like Ruby  
I like Java
```

lang に "Perl" , "Python" , "Ruby" ,
"java" と代入される



要素を直接一つずつ取り出す⑤

```
for lang in ["Perl", "Python", "Ruby", "Java"]:  
    print( " I like " , lang )
```

```
> python sample.py  
I like Perl  
I like Python  
I like Ruby  
I like Java
```

lang に "Perl" , "Python" , "Ruby" ,
"java" と代入される



まとめ①

- 要素番号で要素の値を参照したい場合

```
a=[1,3,5,7,9]
```

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

```
i=0  
while i<len(a):  
    print( a[i] )  
    i+=1
```



まとめ②

- 要素を直接参照したい場合

```
a=[1,3,5,7,9]  
for x in a:  
    print( x )
```

```
for x in [1,3,5,7,9]:  
    print( x )
```

試してみよう①

```
a =[11,12,13,14,15]
```

```
print( a )
```

```
print( len(a) )
```

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

```
    print( i , a[i] )
```

[11, 12, 13, 14, 15]

5

0 11

1 12

2 13

3 14

4 15

iは0,1,2,3,4

→a[0],a[1],a[2],a[3],a[4]

試してみよう②

```
a = [11, 12, 13, 14, 15]
print( a )
print( len(a) )
for i in range( len(a)+1 ):
    print( i , a[i] )
```

iは0,1,2,3,4,5

→a[0],a[1],a[2],a[3],a[4],a[5]

→a[5]は存在しない

```
[11, 12, 13, 14, 15]
```

```
5
```

```
0 11
```

```
1 12
```

```
2 13
```

```
3 14
```

```
4 15
```

存在しない配列の要素を参照した場合、エラーとなる
→ IndexError: list index out of range

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

```
File "C:\Users\shino\Desktop\sample.py", line 5, in <module>
```

```
    print( i , a[i] )
```

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



試してみよう③

```
a =[11,12,13,14,15]
for i in range(2,len(a)):
    print( i , a[i] )
```

```
2 13
3 14
4 15
```

iは2,3,4
→a[2],a[3],a[4]

```
a =[11,12,13,14,15]
for x in a:
    print( x )
```

```
11
12
13
14
15
```

xは11,12,13,14,15



試してみよう④

```
names = ["Perl", "Python", "Ruby", "Java" ]
```

```
names[ 0 ] = "C++"
```

```
for lang in names:
```

```
    print( " I like " , lang )
```

names[0]に"C++"を代入

```
I like C++  
I like Python  
I like Ruby  
I like Java
```

lang に "C++" , "Python" , "Ruby" ,
"java" と代入される



試してみよう⑤

```
a=[1,2,3,4,5]  
for i in range(len(a)):
```

```
    a[i]=a[i]*10
```

配列の各値を10倍

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



10
20
30
40
50

試してみよう⑥

```
a = [ 1 , 2 , 3 , 4 , 5 ]  
print( a )  
for x in a:  
    print( " 代入前 " , x )  
    x=x*10  
    print( " 代入後 " , x )  
print( a )
```

x にはa[0]～a[4]の値が代入
されるだけで配列の要素を直
接変更するわけではない

```
[1, 2, 3, 4, 5]  
代入前 1  
代入後 10  
代入前 2  
代入後 20  
代入前 3  
代入後 30  
代入前 4  
代入後 40  
代入前 5  
代入後 50  
[1, 2, 3, 4, 5]
```

試してみよう⑦

```
a = [ 2 , 3 , 5 , 8 , 4 ]  
print( a )  
for i in range(len(a)):  
    print( "代入前" , a[ i ] )  
    a[i]=a[i]*10  
    print( "代入後" , a[ i ] )  
print( a )
```

配列の要素を直接変更している

[2, 3, 5, 8, 4]

代入前 2

代入後 20

代入前 3

代入後 30

代入前 5

代入後 50

代入前 8

代入後 80

代入前 4

代入後 40

[20, 30, 50, 80, 40]



配列要素には何が代入できるか①

- 変数に代入できるものなら何でも代入できる
- 整数，浮動小数点数，文字列，**配列**！
 - 配列を混在させた場合は，多次元配列（これについては後ほど）
- しかも，混在！できる

配列要素には何が代入できるか②

```
abc=["a","b","c"]
```

```
print( abc )
```

```
abc[1] = 111
```

整数

```
print( abc )
```

```
abc.append( 3.33 )
```

小数

```
print( abc )
```

```
abc.append( "x" )
```

文字列

```
print( abc )
```

```
abc[4] = [4,5,6]
```

配列

```
print( abc )
```

['a', 'b', 'c']

['a', 111, 'c']

['a', 111, 'c', 3.33]

['a', 111, 'c', 3.33, 'x']

['a', 111, 'c', 3.33, [4, 5, 6]]

配列も混在できる

要素と要素番号を同時に取り出す方法

配列名=[値1,値2,...,値n]

```
for i , x in enumerate(配列名):  
    print( i , x )
```

x に値1,値2,...値nが代入される

a=[1,3,5,7,9]

```
for i , x in enumerate(a):  
    print( i , x )
```

i=0,1,2,...,n-1が代入される

i=0,1,2,3,4

x に1,3,5,7,9が代入される



要素と要素番号を同時に取り出す方法

xには配列xの要素の値が代入される

```
names = ["Perl", "Python", "Ruby", "Java"]  
for i, x in enumerate(names):  
    print(i, "番目の要素: ", x)
```

iには要素番号(0~3)が代入される

```
>python sample.py  
0番目の要素: Perl  
1番目の要素: Python  
2番目の要素: Ruby  
3番目の要素: Java
```

複数の配列の要素を同時に取り出す①

```
number = [1,2,3,4]
lang = ['Python' , 'Ruby' , 'Java' , 'C++' ]

for i , j in zip(number,lang):
    print( i , j )
```

iにはnumberの要素

jにはlangの要素

```
> python sample.py
1 Python
2 Ruby
3 Java
4 C++
```

複数の配列の要素を同時に取り出す②

```
name = [ 'A' , 'B' , 'c' , 'd' ]  
english = [ 50 , 80 , 90 , 70 ]  
math = [ 40 , 100 , 60 , 70 ]
```

```
for i , j , k in zip(name,english,math):  
    print( i , j , k )
```

kにはmathの要素

iにはnameの要素

jにはenglishの要素

```
> python sample.py  
A 50 40  
B 80 100  
c 90 60  
d 70 70
```




配列の要素の参照例



配列の要素の参照例①

配列の要素の合計を求める

```
a=[4,2,1,6,7]
```

iには0,1,2,3,4が代入

```
sum = 0
```

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

```
    sum += a[ i ]
```

```
print( " sum = " , sum )
```

```
a=[4,2,1,6,7]
```

iには4,2,1,6,7が代入

```
sum = 0
```

```
for i in a:
```

```
    sum += i
```

```
print( " sum = " , sum)
```

```
>python sample.py  
sum = 20
```

配列の要素の参照例②

配列の要素の合計を求める

```
a=[4,2,1,6,7]
```

```
i = 0
```

```
sum = 0
```

```
while i < len(a):
```

```
    sum += a[ i ]
```

```
    i += 1
```

```
print( " sum = " , sum )
```

```
a=[4,2,1,6,7]
```

```
i = 0
```

```
sum = 0
```

```
while True:
```

```
    sum += a[ i ]
```

```
    i += 1
```

```
    if i >= len(a):
```

```
        break
```

```
print( " sum = " , sum )
```

whileを用いても同じ動作ができます

配列の要素の参照例②'

配列の要素の合計を求める

```
a=[4,2,1,6,7]
```

```
i = 0
```

```
sum = 0
```

```
while i <= len(a):
```

```
    sum += a[ i ]
```

```
    i += 1
```

```
print( " sum = " , sum )
```

iは0,1,2,3,4,5

→a[0],a[1],a[2],a[3],a[4],a[5]

→a[5]は存在しない

→エラー

IndexError: list index out of range

Traceback (most recent call last):

File "C:¥Users¥shino¥Desktop¥sample.py", line 5, in <module>

sum += a[i]

IndexError: list index out of range



配列の要素の参照例③

どう違うでしょうか

```
a=[4,2,1,6,7]
```

```
for i in range(len(a)):
    if i % 2 != 0:
        print( a[ i ] )
```

```
>python sample.py
2
6
```

```
a=[4,2,1,6,7]
```

```
for x in a:
    if x % 2 != 0:
        print( x )
```

```
>python sample.py
1
7
```



配列の要素の参照例③

どう違うでしょうか

```
a=[4,2,1,6,7]
```

```
for i in range(len(a)):
    if i % 2 != 0:
        print( a[ i ] )
```

iには0,1,2,3,4が代入される
表示されるのはa[1],a[3]

```
a=[4,2,1,6,7]
```

```
for x in a:
    if x % 2 != 0:
        print( x )
```

xには4,2,1,6,7が代入される
表示されるのは1,7

配列の要素の参照例④

```
import random
```

```
x=[]
```

配列xを宣言

10個の乱数を生成
配列xに格納

```
for i in range(10):
```

```
    x.append( random.randint(0,100) )
```

```
sum = 0
```

```
for i in range(10):
```

```
    sum += x[ i ]
```

合計値の計算

```
print( x )
```

```
print( " sum = " , sum )
```

```
>python sample.py
```

```
[26, 14, 15, 79, 64, 50, 76, 79, 33, 48]
```

```
sum = 484
```

配列の要素の参照例⑤

配列の最後の要素から出力

```
a=[4,2,1,6,7]
```

```
for i in range(len(a)):  
    print(a[ len(a)-1-i ])
```

$a[4], a[3], a[2], a[1], a[0]$
の順に出力される

```
>python sample.py  
7  
6  
1  
2  
4
```


配列の要素の参照例⑥

配列のコピー

```
a=[4,2,1,6,7]
```

```
x=[]
```

配列xを宣言

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

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

```
print( x )
```

xにa[i]を追加

```
>python sample.py  
[4, 2, 1, 6, 7]
```

```
a=[4,2,1,6,7]
```

```
x=[]
```

配列xを宣言

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

```
    x.append(a[i]*a[i])
```

```
print( x )
```

xにa[i]*a[i]を追加

```
>python sample.py  
[16, 4, 1, 36, 49]
```



配列の要素の参照例⑥

前のページと同じです

```
a=[4,2,1,6,7]  
x=[0]*len(a)
```

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

x[i]にa[i]を代入

```
a=[4,2,1,6,7]  
x=[0]*len(a)
```

```
for i in range(len(a)):  
    x[i]=a[i]*a[i]  
print( x )
```

x[i]にa[i]*a[i]を代入

配列xの要素数が分かっている場合

配列の初期化②(もう一度)

- 配列名=[**値**]***n**
 - 要素数**n**個の**値**を持つ配列を作成

```
a=[0]*5  
print(a)
```

要素数5個, 値0の配列a

[0, 0, 0, 0, 0]

```
x=[1]*(len(a)+5)  
print(x)
```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

要素数10個, 値1の配列x



配列の初期化③

どちらも同じ配列a,xを作成します

```
a=[]  
for i in range(5):  
    a.append(0)  
print(a)
```

```
x=[]  
for i in range(len(a)+5):  
    x.append(1)  
print(x)
```

```
a=[0]*5  
print(a)
```

```
x=[1]*(len(a)+5)  
print(x)
```

```
[0, 0, 0, 0, 0]  
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

注意: 配列の要素の参照例⑥

「配列のコピー」にはなりません！

```
a=[4,2,1,6,7]
```

```
x=a
```

配列xにコピー？

```
print( a )
```

```
print( x )
```

```
a[0]=10
```

配列aだけ変更

```
print( a )
```

```
print( x )
```

しかし、x も変っている！

```
Z:¥ruby>ruby sample.rb
```

```
[4, 2, 1, 6, 7]
```

```
[4, 2, 1, 6, 7]
```

```
[10, 2, 1, 6, 7]
```

```
[10, 2, 1, 6, 7]
```

注意: 配列の要素の参照例⑥

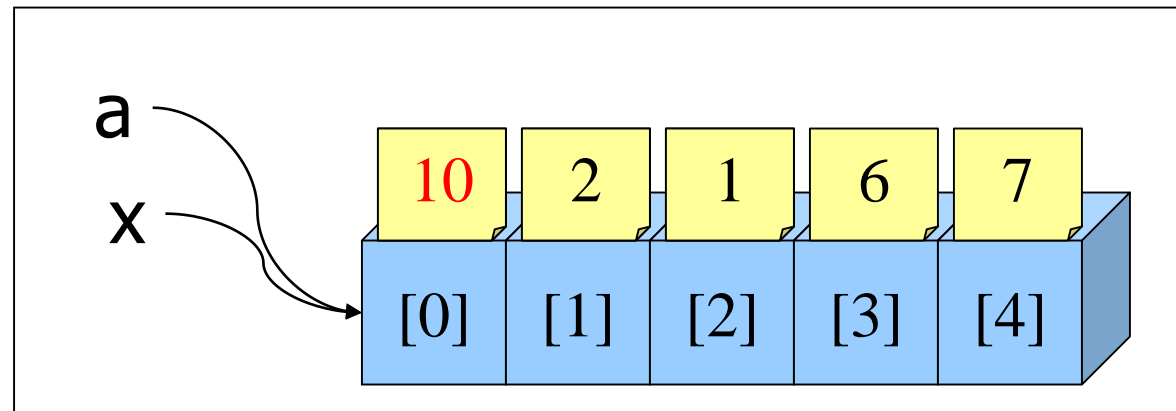
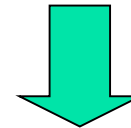
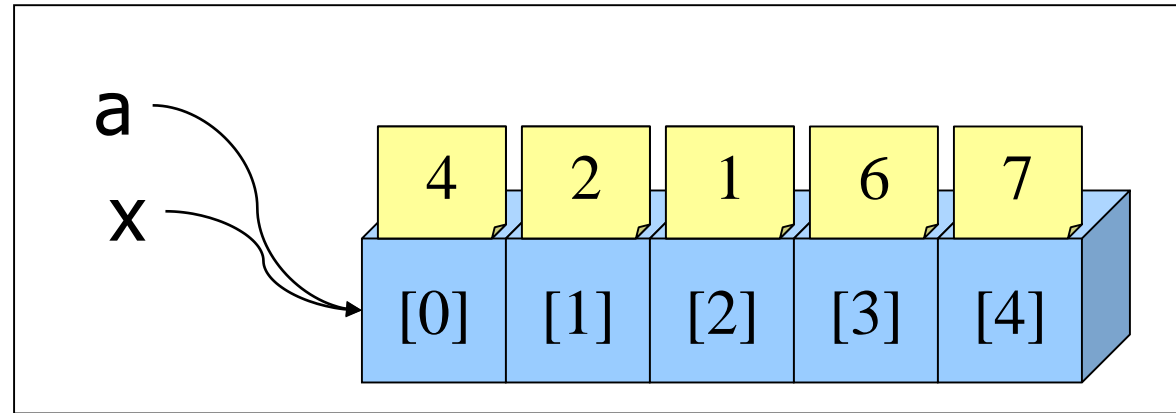
```
a=[4,2,1,6,7]
```

```
x=a
```

```
print( a, x )
```

```
a[0]=10
```

```
print( a, x )
```



配列の要素の参照例⑦

配列の先頭の要素に値を追加
(insertを用いない場合)

```
a=[4,2,1,6,7]
```

```
a.append(0)
```

```
n=len(a)-1
```

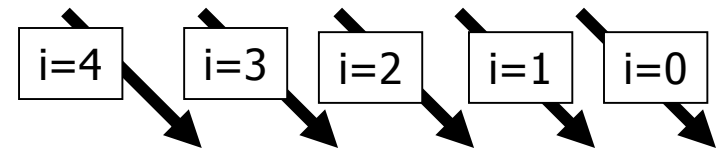
```
for i in range(n):
```

```
    a[ n-i ] = a[ n-i-1 ]
```

```
a[ 0 ] = 5
```

```
print( a )
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
4	2	1	6	7	0



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
4	4	2	1	6	7



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
5	4	2	1	6	7

配列の要素の参照例⑧

実行結果

```
import random
```

```
n=10
```

```
a=[0]*n
```

要素数10個の値0を持つ配列

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

10個の乱数を生成

```
    a[i]=random.randint(1,10)
```

```
    print( a[ i ] , "*" * a[ i ] )
```

* を a[i] 個出力

```
>python sample.py
```

```
3 ***
```

```
3 ***
```

```
5 *****
```

```
2 **
```

```
3 ***
```

```
4 ****
```

```
2 **
```

```
1 *
```

```
7 *****
```

```
5 *****
```




配列の要素の参照例⑨

```
a=[1,2,3]
```

```
b=[4,5,6]
```

```
x=
```

練習問題③の回答

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

```
}
```

```
print( a )
```

```
print( b )
```

```
print( x )
```

二つの配列(ベクトル)の要素
の和の計算

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



練習問題

配列に関する練習①～⑤
(簡単な人は練習⑥も行なって下さい)



練習問題①

- 配列 `a=[5,4,2,7,6]` の要素の中で最小値, 最大値を求めるプログラムをfor文を用いて書きなさい

```
>python 9-1.py  
[5, 4, 2, 7, 6]  
max -> 7  
min -> 2
```



練習問題②

- 配列a, b, cに

a=[1,2,3,4,5]

b=[1,4,9,16,25]

c=[1,8,27,64,125]

という値を設定しなさい.

- ただし, for文を一回用いて各要素に値を格納するプログラム(次頁)を書きなさい.

練習問題②

(どちらかのプログラムのfor文を完成させなさい)

配列の宣言

```
a=[0]*5  
b=[0]*5  
c=[0]*5
```

```
for i in ...
```

```
print( a )  
print( b )  
print( c )
```

```
a=[]  
b=[]  
c=[]
```

```
for i in ...
```

```
print( a )  
print( b )  
print( c )
```

```
> python 9-2.py  
[1, 2, 3, 4, 5]  
[1, 4, 9, 16, 25]  
[1, 8, 27, 64, 125]
```



練習問題③

- 二つの配列 $a=[4,3,6,9,1]$
 $b=[1,9,5,2,3]$ をベクトルとした場合, 二つのベクトルの和を配列 x に, 二つの内積を変数 y に求めるプログラムを書きなさい

```
>python 9-3.py  
[4, 3, 6, 9, 1]  
[1, 9, 5, 2, 3]  
[5, 12, 11, 11, 4]  
82
```

練習問題④

- キーボードから整数を入力し，順番に配列xに格納し，その結果を出力するプログラムを書きなさい。（キーボードからの入力はqを入力することで終了とする）

```
>python 9-4.py
4
5
6
7
12
q
[4, 5, 6, 7, 12]
```

整数を入力

qで終了

配列xを出力



練習問題⑤

- 配列`x=[3,4,9,6,2]`の要素をfor文を用いて、逆順に並び変えるプログラムを作成しなさい
- もう一つ配列を使ってはいけません.
- 配列`x`の要素を直接入れ換えること

```
>python 9-5.py  
[3, 4, 9, 6, 2]  
[2, 6, 9, 4, 3]
```


補足: 入れ替え①

■ 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?

```
a = 30
```

```
b = 75
```

```
print( a, b )
```

```
a=b
```

```
b=a
```

```
print( a, b )
```

30 75

75 75



補足：入れ替え②

- 入替えには，作業領域があればよい

```
a = 30  
b = 75  
print( a, b )  
work=a  
a=b  
b=work  
print( a, b )
```



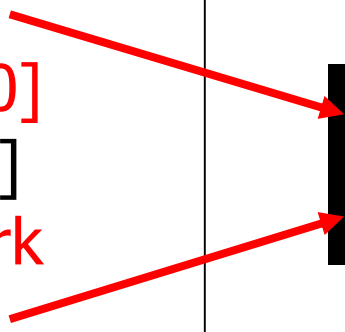
30	75
75	30



補足: 入れ替え③

- 配列要素に対しても同様

```
x = [30,75]  
print( x )  
work=x[0]  
x[0]=x[1]  
x[1]=work  
print( x )
```



```
[30, 75]  
[75, 30]
```



練習問題⑥

- 下記のプログラムによって, 20個の要素を持つ配列aに100未満の整数値を格納します.

```
import random
a=[0]*20
for i in range(20):
    a[i] = random.randint(0,100)
print(a)
```

```
> python 9-6.py
```

```
[17, 8, 87, 76, 64, 67, 9, 67, 41, 31, 59, 30, 65, 79, 79, 74, 22, 60, 80, 11]
```



練習問題⑥

- 配列aの要素において、常に左の要素の値<右の要素の値となっている部分列を求め、その最長の部分列の長さ(要素数)を印字するプログラムを書きなさい。

```
>python 9-6.py
```

```
[93, 70, 18, 65, 10, 46, 84, 50, 68, 31, 40, 86, 33, 65, 85, 90, 12, 9, 1, 95]
```

33<65<85<90

最長部分列の長さは 4

```
>python 9-6.py
```

```
[0, 89, 100, 13, 50, 67, 50, 59, 69, 5, 34, 59, 1, 2, 53, 17, 8, 35, 77, 12]
```

最長部分列の長さは 3



練習問題

- 練習問題①から⑤を行ないなさい。
(簡単な人は練習⑥も行なって下さい)
- 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.