

プログラミング言語 第九回

担当: 篠沢 佳久
櫻井 彰人

平成29年 6月12日

1

本日の内容

- 配列
 - 宣言
 - 代入
 - 要素の参照方法
- 練習問題①～⑥

2

配列とは

配列とは
配列の宣言

3

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

- 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

4

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
- Ruby の場合は、ちょっと、違う
- 「列」だと思ってください。
 - 値の列、場所の列

博識の方へ:
Ruby の配列は、CやJavaの配列とは大きく異なります。
(Lisp を源流とする)リスト構造 (の発展型)と考えるとよい。

5

こんな具合です。

```
定数:  
["Perl", "Python", "Ruby", "Scheme"]  
変数への代入:  
names = ["Perl", "Python", "Ruby", "Scheme"]  
印字  
print["Perl", "Python", "Ruby", "Scheme"]
```

```
irb(main):016:0> print ["Perl", "Python", "Ruby", "Scheme"]  
["Perl", "Python", "Ruby", "Scheme"]=> nil  
irb(main):017:0> puts ["Perl", "Python", "Ruby", "Scheme"]  
Perl  
Python  
Ruby  
Scheme  
=> nil  
irb(main):018:0> p ["Perl", "Python", "Ruby", "Scheme"]  
["Perl", "python", "Ruby", "Scheme"]  
=> nil
```

6

配列の要素①

names = ["Perl", "Python", "Ruby", "Scheme"]

イメージ的には表計算のセル

要素番号 (インデックス)	配列名 names
0	"Perl"
1	"Python"
2	"Ruby"
3	"Scheme"

要素番号は0から始まる

names[0]
names[1]
names[2]
names[3]

7

(参考)表計算ソフトのセル

A1="Perl"
A2="Python"
A3="Ruby"
A4="Scheme"

8

配列の要素②

```

irb(main):004:0> names = ["Perl", "Python", "Ruby", "Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):005:0> p names[ 0 ]
"Perl"
=> "Perl"
irb(main):006:0> p names[ 1 ]
"Python"
=> "Python"
irb(main):007:0> p names[ 2 ]
"Ruby"
=> "Ruby"
irb(main):008:0> p names[ 3 ]
"Scheme"
=> "Scheme"
irb(main):009:0> p names[ 4 ]
nil
=> nil
    
```

names[4]には値が
代入されていない
→nil となる

9

配列の宣言

- names = ["Perl", "Python", "Ruby", "Scheme"]
- 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

10

配列の要素③

- a = [0, 2, 4, 6, 8]
- 配列名[要素番号]
- 配列の要素数
- 配列名.length

この配列のa.lengthの値は5

	a
0	0
1	2
2	4
3	6
4	8

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

```

irb(main):001:0> names = ["Perl", "Python", "Ruby", "Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):002:0> names.length
=> 4
    
```

11

配列の要素への代入①

- 配列名[要素番号] = 値
- names = ["Perl", "Python", "Ruby", "Scheme"]
- names[0] = "C"
- names[3] = "Java"

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

→

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

12

配列の要素への代入②

```

irb(main):014:0> names = ["Perl", "Python", "Ruby",
"Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):015:0> names[ 0 ] = "C"
=> "C"
irb(main):016:0> names[ 3 ] = "Java"
=> "Java"
irb(main):017:0> p names
["C", "Python", "Ruby", "Java"]
=> ["C", "Python", "Ruby", "Java"]
    
```

13

試してみよう①

```

irb(main):001:0> a = [ 1,2,3,4,5]
=> [1, 2, 3, 4, 5]
irb(main):002:0> p( a )<
[1, 2, 3, 4, 5]
=> [1, 2, 3, 4, 5]
irb(main):003:0> print( a )<
[1, 2, 3, 4, 5]> nil
irb(main):004:0> puts( a )<
1
2
3
4
5
=> nil
    
```

表示の違いに注目!

p(配列名)
print(配列名)
puts(配列名)
()はなくてもよい

14

試してみよう②

```

irb(main):001:0> a = [ 1,2,3,4,5]
=> [1, 2, 3, 4, 5]
irb(main):002:0> p( a )
[1, 2, 3, 4, 5]
irb(main):003:0> print( a )
[1, 2, 3, 4, 5]> nil
irb(main):004:0> puts( a )
1
2
3
4
5
=> nil
irb(main):005:0> a[0]
=> 1
irb(main):006:0> a[1]
=> 2
    
```

a[5], a[6]には値が
代入されていない
→nilとなる

```

irb(main):007:0> a.length
=> 5
irb(main):008:0> a[a.length-1]
=> 5
irb(main):009:0> a[a.length]
=> nil
irb(main):010:0> a[a.length+1]
=> nil
irb(main):011:0> a.length
=> 5
irb(main):012:0> a[7] = 77
=> 77
irb(main):013:0> p( a )
[1, 2, 3, 4, 5, nil, nil, 77]
=> [1, 2, 3, 4, 5, nil, nil, 77]
    
```

注目!

15

Ruby の配列は柔軟

- すでに存在する要素に代入できる
 - これは当たり前
- まだ「ない要素」に代入すると、配列を拡張！して(当該要素を作って)くれる

```

irb(main):006:0> abc = ["a","b","c"]
=> ["a", "b", "c"]
irb(main):007:0> abc[3] = "d"
=> "d"
irb(main):008:0> abc
=> ["a", "b", "c", "d"]
irb(main):009:0> abc[10] = "k"
=> "k"
irb(main):010:0> abc
=> ["a", "b", "c", "d", nil, nil, nil, nil, nil, nil, "k"]
irb(main):011:0>
    
```

注目!

16

Ruby の配列は柔軟

abc = ["a","b","c"] abc[3] = "d" abc[10] = "k"

abc	
0	a
1	b
2	c

→

abc	
0	a
1	b
2	c
3	d

→

abc	
0	a
1	b
2	c
3	d
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
10	k

abc[4]からabc[9]の値はnilとなる

17

各要素に代入する(失敗編)

```

irb(main):062:0> primes[0]=2
NameError: undefined local variable or method `primes' for main:Object
from (irb):62
from C:/Program Files/Ruby-1.9.3/bin/irb.bat:19:in `<main>'
irb(main):063:0>
    
```

注目!

あれ？

いろいろややこしい事情があるので。
Ruby では (Ruby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
Ruby 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
Ruby 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！)

18

配列だということを教える①

- もちろん、教える相手は Ruby

配列名 = Array.new(要素数)
要素数分の配列を用意する

```
abc = Array.new(5)
```

```
irb(main):073:0> abc = Array.new(5)
=> [nil, nil, nil, nil, nil]
irb(main):074:0>
```

注目!

```
irb(main):075:0> abc[3]=333
=> 333
irb(main):076:0> abc
=> [nil, nil, nil, 333, nil]
irb(main):077:0>
```

注目!

19

配列だということを教える②

```
abc = Array.new(5)
```

	abc
0	nil
1	nil
2	nil
3	nil
4	nil

配列名abc
要素数5個を用意する

abc[0]からabc[4]まで
値は入っていない(nil)

20

配列だということを教える③

- 要素数も分からない場合は?

- 配列名 = [] と宣言する

21

配列だということを教える③

配列名 = []

```
x = []
x[0] = 3
x[1] = 5
```

xが配列であることを宣言

```
irb(main):008:0> x=[]
=> []
irb(main):009:0> x[0] = 3
=> 3
irb(main):010:0> x[1] = 5
=> 5
irb(main):011:0> p x
=> [3, 5]
```

注目!

22

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]
- 要素数のみが決まっている場合
 - 配列名 = Array.new(要素数)
- 要素数が決まっていない場合
 - 配列名 = []

23

配列の宣言のまとめ②

	a
0	3
1	4
2	1

要素数のみが決まっている場合
a = Array.new(3)
a[0] = 3
a[1] = 4
a[2] = 1

要素が分かっている場合
a = [3, 4, 1]

要素数が決まっていない場合
a = []
a[0] = 3
a[1] = 4
a[2] = 1

24

試してみよう2

```

irb(main):001:0> a[3]
NameError: undefined local variable or method 'a' for main:Object
from (irb):1
from C:/Program Files/Ruby-1.9.3/bin/irb.bat:19:in `main'
irb(main):002:0> x = a[2]
NameError: undefined local variable or method 'a' for main:Object
from (irb):2
from C:/Program Files/Ruby-1.9.3/bin/irb.bat:19:in `main'
irb(main):003:0> a = Array.new(5)
=> [nil, nil, nil, nil, nil]
irb(main):004:0> a[10]
=> nil
irb(main):005:0> a
=> [nil, nil, nil, nil, nil]

```

配列aは宣言されていない

配列名はa
要素数は5個と宣言

25

その他の参照方法

- 配列名[n..m]
 - n番目からm番目の要素を参照する
- 配列名[n,length]
 - n番目からlength個の要素を参照する

26

その他の参照方法(例①)

```

irb(main):001:0>
x=["A","B","C","D","E"]
=> ["A","B","C","D","E"]
irb(main):002:0> x[2..4]
=> ["C","D","E"]
irb(main):003:0> x[1..3]
=> ["B","C","D"]
irb(main):006:0> x[3..10]
=> ["D","E"]

```

配列	x
0	A
1	B
2	C
3	D
4	E

要素番号は4までしか存在しない

27

その他の参照方法(例②)

```

irb(main):001:0>
x=["A","B","C","D","E"]
=> ["A","B","C","D","E"]
irb(main):007:0> x[2,2]
=> ["C","D"]
irb(main):008:0> x[0,3]
=> ["A","B","C"]

```

配列	x
0	A
1	B
2	C
3	D
4	E

28

配列と繰り返し

配列の要素の参照方法

29

配列の要素の参照方法①

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

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

names[0]~names[3]まで
順番に参照するには？

30

配列の要素の参照方法①'

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

繰り返しを用いて記述する

31

配列の要素の参照方法②

- 配列の要素の参照方法には

- 要素番号を用いて要素の値を取り出す方法
- 要素の値を直接取り出す方法

- があります

32

要素番号を用いて一つずつ取り出す

```
a=[1,3,5,7,9]
5.times{ |i|
  print( a[ i ] , "\n" )
}
```

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

a.length=5

```
Z:~Ruby>ruby sample.rb
1
3
5
7
9
```

33

要素番号を用いて一つずつ取り出す

```
names=["Perl", "Python", "Ruby", "Scheme"]
4.times { |i| print( "#{i} 番目は #{ names[i] }\n" ) }
```

注目!

```
irb(main):033:0> names=["Perl", "Python", "Ruby", "Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):034:0> 4.times { |i| print( "#{i} 番目は #{ names[i] }\n" ) }
0 番目は Perl
1 番目は Python
2 番目は Ruby
3 番目は Scheme
=> 4
irb(main):035:0>
```

iには0,1,2,3と代入される
names[0], names[1], names[2], names[3]と参照される

要素番号を用いて一つずつ取り出す(続)

```
names=["Perl", "Python", "Ruby", "Scheme"]
names.length.times { |i| print( "#{i} 番目は #{ names[i] }\n" ) }
```

注目!

```
irb(main):035:0> names=["Perl", "Python", "Ruby", "Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):036:0> names.length.times { |i| print( "#{i} 番目は #{ names[i] }\n" ) }
0 番目は Perl
1 番目は Python
2 番目は Ruby
3 番目は Scheme
=> 4
```

names.length=4

35

要素番号を用いて一つずつ取り出す(続々)

```
a=[1,3,5,7,9]
(0..4).each{ |i|
  print( a[ i ] , "\n" )
}
```

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]
(2..4).each{ |i|
  print( a[ i ] , "\n" )
}
```

iには2,3,4が代入される
a[2],a[3],a[4]と参照される

36

要素番号を用いて一つずつ取り出す (続々)

```
a=[1,3,5,7,9]
(0..4).each{ |i|
  print( a[ i ] , "\n" )
}
```

```
Z:¥Ruby>ruby sample.rb
1
3
5
7
9
```

```
a=[1,3,5,7,9]
(2..4).each{ |i|
  print( a[ i ] , "\n" )
}
```

```
Z:¥Ruby>ruby sample.rb
5
7
9
```

37

要素番号を用いて一つずつ取り出す (続々)

a.length=5

```
a=[1,3,5,7,9]
(0..a.length-1).each{ |i|
  print( i , " " , a[ i ] , "\n" )
}
```

```
Z:¥Ruby>ruby sample.rb
0 1
1 3
2 5
3 7
4 9
```

```
a=[1,3,5,7,9]
(0..a.length).each{ |i|
  print( i , " " , a[ i ] , "\n" )
}
```

```
Z:¥Ruby>ruby sample.rb
0 1
1 3
2 5
3 7
4 9
5
a[5]の値はnil
→表示されない
```

38

要素番号を用いて一つずつ取り出す①

times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

```
配列の長さ.times{ |i|
  配列[ i ]の処理
}
```

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

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

39

要素番号を用いて一つずつ取り出す②

each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

```
(0..配列の長さ-1).each{ |i|
  配列[ i ]の処理
}
```

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

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

40

要素を直接一つずつ取り出す

```
[値1,値2,...,値n].each{ |i|
  print( i , "\n" )
}
```

i に値1,値2,...,値nが代入される

```
[1,3,5,7,9].each{ |i|
  print( i , "\n" )
}
```

i に1,3,5,7,9が代入される

41

要素を直接一つずつ取り出す

```
[1,3,5,7,9].each{ |i|
  print( i , "\n" )
}
```

i には1,3,5,7,9と代入される

```
a=[1,3,5,7,9]
a.each{ |i|
  print( i , "\n" )
}
```

```
Z:¥Ruby>ruby sample.rb
1
3
5
7
9
```

42

要素を直接一つずつ取り出す

```
["Perl", "Python", "Ruby", "Scheme"].each { |lang|  
  print( "I like ", lang, "\n" )  
}
```

注目!

```
irb(main):028:0> ["Perl", "Python", "Ruby", "Scheme"].each  
irb(main):029:1* { |lang| print( "I like ", lang, "\n" )  
irb(main):030:1> }  
I like Perl  
I like Python  
I like Ruby  
I like Scheme  
=> ["Perl", "Python", "Ruby", "Scheme"]
```

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

43

要素を直接一つずつ取り出す(続)

前ページと同じ

```
names=["Perl", "Python", "Ruby", "Scheme"]  
names.each { |lang| print( "I like #{lang}\n" ) }
```

```
irb(main):030:0> names=["Perl", "Python", "Ruby", "Scheme"]  
=> ["Perl", "Python", "Ruby", "Scheme"]  
irb(main):031:0> names.each { |lang| print( "I like #{lang}\n" ) }  
I like Perl  
I like Python  
I like Ruby  
I like Scheme  
=> ["Perl", "Python", "Ruby", "Scheme"]
```

44

要素を直接一つずつ取り出す(続)

配列名=[値1,値2,...,値n]

```
配列名.each { |i|  
  print( i, "\n" )  
}
```

i に値1,値2,...,値nが代入される

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

```
a.each { |i|  
  print( i, "\n" )  
}
```

i に1,3,5,7,9が代入される

45

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]

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

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

46

まとめ②

- 要素を直接参照したい場合

```
a=[1,3,5,7,9]  
a.each { |i|  
  print( i, "\n" )  
}
```

```
[1,3,5,7,9].each { |i|  
  print( i, "\n" )  
}
```

47

試してみよう 3

```
irb(main):001:0> a = [11,12,13,14,15]  
=> [11, 12, 13, 14, 15]  
irb(main):002:0> a.length  
=> 5  
irb(main):003:0> a.each { |x| print( "#{x} " ) }  
11 12 13 14 15 => [11, 12, 13, 14, 15]  
irb(main):004:0> (0..a.length-1).each { |i| puts( "#{i}:  
#{a[i]} " ) }  
0: 11  
1: 12  
2: 13  
3: 14  
4: 15  
=>0..4
```

a.length-1 であることに注意

48

試してみよう 3

前ページの続き

```

irb(main):005:0> (3..a.length+1).each{ |i| puts( "#{i}:
#{a[i]} " ) }
3: 14
4: 15
5: a[ a.length ]
6: a[ a.length+1 ]
=> 3..6

```

a.length 以上の要素は存在しない

```

irb(main):006:0> n=88; [2,3,5,7,11].each{ |p| puts(
"#{n} is divisible by #{p}" ) if n%p==0 }
88 is divisible by 2
88 is divisible by 11
=> [2, 3, 5, 7, 11]

```

変数pには2,3,5,7,11が代入される

49

試してみよう 4

注目!

```

names=["Perl", "Python", "Ruby", "Scheme"]
names[0] = "Ada"
names.length.times { |i|
  print( "#{i} 番目は #{ names[i] }¥n" )
}

irb(main):059:0> names=["Perl", "Python", "Ruby", "Scheme"]
=> ["Perl", "Python", "Ruby", "Scheme"]
irb(main):060:0> names[0] = "Ada"
=> "Ada"
irb(main):061:0> names.length.times { |i|
  print( "#{i} 番目は #{ names[i] }¥n" )
}
0 番目は Ada
1 番目は Python
2 番目は Ruby
3 番目は Scheme
=> 4

```

試してみよう 5

```

irb(main):001:0> a = ["a","b","c"]
=> ["a", "b", "c"]
irb(main):002:0> a[0] = "x"
=> "x"
irb(main):003:0> a
=> ["x", "b", "c"]
irb(main):004:0> a[7]="h"
=> "h"
irb(main):005:0> a
=> ["x", "b", "c", nil, nil, nil, nil, "h"]

```

a[3]からa[6]にはnilが代入

51

試してみよう 5

	a
0	"x"
1	"b"
2	"c"
3	nil
4	nil
5	nil
6	nil
7	"h"

a=["x", "b", "c"]

a[7]="h"

a[3]からa[6]は nil
(値がない)

52

試してみよう 5

配列aの要素は変化しない

```

irb(main):005:0> a= ["x", "b", "c"]
irb(main):006:0> a[7] = "h"
irb(main):007:0> a.each{ |x| x="0" }
=> ["x", "b", "c", nil, nil, nil, nil, "h"]
irb(main):008:0> a
=> ["x", "b", "c", nil, nil, nil, nil, "h"]
irb(main):009:0> (0..a.length-1).each{ |i| a[i]="0" }
=> 0..7
irb(main):010:0> a
=> ["0", "0", "0", "0", "0", "0", "0", "0"]

```

53

試してみよう 5

```

# coding: Windows-31j
a = [ 1, 2, 3, 4, 5 ]
a.each{ |x|
  print( " 代入前 ", x, "¥n" )
  x="0"
  print( " 代入後 ", x, "¥n" )
}
p 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

```

Z:¥ruby>ruby sample.rb
代入前 1
代入後 0
代入前 2
代入後 0
代入前 3
代入後 0
代入前 4
代入後 0
代入前 5
代入後 0
[1, 2, 3, 4, 5]

```

54

試してみよう 5

```
# coding: Windows-31J
a = [ 2, 3, 5, 8, 4 ]
(0..a.length-1).each{|i|
  print( "代入前", a[ i ], "%n" )
  a[i]="0"
  print( "代入後", a[ i ], "%n" )
}
p a
```

配列の要素を直接変更している

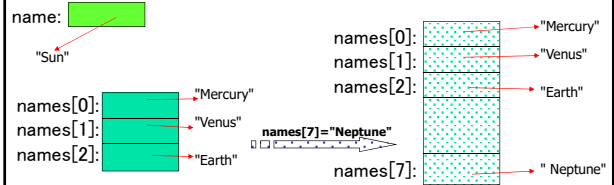
Z:¥ruby>ruby sample.rb

```
代入前2
代入後0
代入前3
代入後0
代入前5
代入後0
代入前8
代入後0
代入前4
代入後0
["0", "0", "0", "0", "0"]
```

55

変数と配列の裏事情

ユーザ(皆さんです)が、変数や配列を使いたいというと、コンピュータは「場所」を確保する必要あり



配列の大きさが変わると、Ruby は配列を作り直している。古い場所はそのまま捨て置かれる。すなわち、無駄が発生している。ユーザには見えないが。

56

配列要素には何が代入できるか

- 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！
- しかも、混在！できる
 - CやJavaでは「混在」はできない

```
irb(main):011:0> abc = ["a","b","c"]
=> ["a", "b", "c"]
irb(main):012:0> abc[1] = 111
=> 111
irb(main):013:0> abc
=> ["a", 111, "c"]
irb(main):014:0> abc[3] = 3.33
=> 3.33
irb(main):015:0> abc
=> ["a", 111, "c", 3.33]
irb(main):016:0> abc[4] = [4,5,6]
=> [4, 5, 6]
irb(main):017:0> abc
=> ["a", 111, "c", 3.33, [4, 5, 6]]
```

注目!

57

試してみよう 6

前のスライドと同じです

```
irb(main):011:0> abc = ["a","b","c"]
=> ["a", "b", "c"]
irb(main):012:0> abc[1] = 111
=> 111
irb(main):013:0> abc
=> ["a", 111, "c"]
irb(main):014:0> abc[3] = 3.33
=> 3.33
irb(main):015:0> abc
=> ["a", 111, "c", 3.33]
irb(main):016:0> abc[4] = [4,5,6]
=> [4, 5, 6]
irb(main):017:0> abc
=> ["a", 111, "c", 3.33, [4, 5, 6]]
```

58

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

```
# coding: Windows-31J
names = ["Perl", "Python", "Ruby", "Java"]
names.each_with_index{|x,i|
  print( i, "番目の要素:", x, "%n" )
}
```

iには要素番号(0~3)が代入される

Z:¥Ruby>ruby sample.rb
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

59

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]

配列名.each_with_index{|x,i|

print(i, " ", x, "%n")

}

xに値1,値2,...,値nが代入される

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

i=0,1,2,...,n-1が代入される

a.each_with_index{|x,i|

print(i, " ", x, "%n")

}

i=0,1,2,3,4

xに1,3,5,7,9が代入される

60

配列の要素の参照例

61

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]
sum = 0
a.length.times{|i|
  sum += a[i]
}
print("sum = ", sum)
```

iには0,1,2,3,4が代入

```
a=[4,2,1,6,7]
sum = 0
a.each{|i|
  sum += i
}
print("sum = ", sum, "\n")
```

iには4,2,1,6,7が代入

```
Z:~Ruby>ruby sample.rb
sum = 20
```

62

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 < a.length do
  sum += a[i]
  i += 1
end
print("sum = ", sum)
```

```
a=[4,2,1,6,7]
i = 0
sum = 0
loop{
  sum += a[i]
  i += 1
  break if i == a.length
}
print("sum = ", sum)
```

while, loop を用いても同じ動作ができます

63

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]
a.length.times{|i|
  if i % 2 != 0 then
    print(a[i], "\n")
  end
}
```

```
Z:~Ruby>ruby sample.rb
2
6
```

```
a=[4,2,1,6,7]
a.each{|i|
  if i % 2 != 0 then
    print(i, "\n")
  end
}
```

```
Z:~Ruby>ruby sample.rb
1
7
```

64

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]
a.length.times{|i|
  if i % 2 != 0 then
    print(a[i], "\n")
  end
}
```

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]
a.each{|i|
  if i % 2 != 0 then
    print(i, "\n")
  end
}
```

iには4,2,1,6,7が代入される
表示されるのは1,7

65

配列の要素の参照例④

x=[] 配列xを宣言

```
10.times{|i|
  x[i] = rand(100)
}
```

10個の乱数を生成
配列xに格納

```
sum = 0
x.length.times{|i|
  sum += x[i]
}
p x
print("sum = ", sum)
```

合計値の計算

```
Z:~Ruby>ruby sample.rb
[26, 14, 15, 79, 64, 50, 76, 79, 33, 48]
sum = 484
```

66

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]
a.length.times { |i|
  print(a[ a.length-1-i ], "\n" )
}
```

a[4],a[3],a[2],a[1],a[0]
の順に出力される

```
Z:~Ruby>ruby sample.rb
7
6
1
2
4
```

67

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
a.length.times { |i|
  x[i] = a[i]
}
p x  x[i]にa[i]を代入
```

```
Z:~Ruby>ruby sample.rb
[4, 2, 1, 6, 7]
```

```
a=[4,2,1,6,7]
x=[] 配列xを宣言
a.length.times { |i|
  x[i] = a[i]*a[i]
}
p x  x[i]にa[i]*a[i]を代入
```

```
Z:~Ruby>ruby sample.rb
[16, 4, 1, 36, 49]
```

68

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=Array.new(a.length)
a.length.times { |i|
  x[i] = a[i]
}
p x  x[i]にa[i]を代入
```

配列xの要素数が分かっている場合

```
a=[4,2,1,6,7]
x=Array.new(a.length)
a.length.times { |i|
  x[i] = a[i]*a[i]
}
p x  x[i]にa[i]*a[i]を代入
```

69

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!

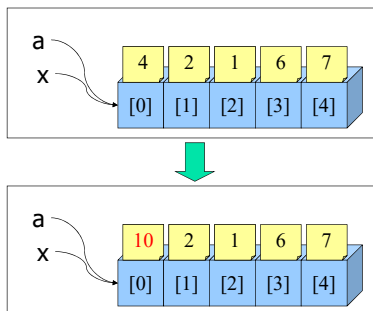
```
a=[4,2,1,6,7]
x=a 配列xにコピー?
p a
p x
a[0]=10 配列aだけ変更
p a
p x  しかし、xも変っている!
```

```
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]
```

70

注意: 配列の要素の参照例⑥

```
a=[4,2,1,6,7]
x=a
p a, x
a[0]=10
p a, x
```



71

配列の要素の参照例⑦

配列の最後の要素に値を追加

```
a=[4,2,1,6,7]
a[ a.length ] = 8
p a
```

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

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
4	2	1	6	7	8

72

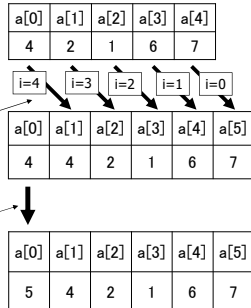
配列の要素の参照例⑧

配列の先頭の要素に値を追加

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

```
n=a.length
n.times{ |i|
  a[ n-i ] = a[ n-i-1 ]
}
```

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



73

配列の要素の参照例⑨

```
a=[]
```

```
n=10
n.times{ |i|
  a[ i ] = rand( 10 )
}
```

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

実行結果

```
Z:\ruby>ruby sample.rb
3 ***
3 ***
5 *****
2 **
3 ***
4 ****
2 **
1 *
7 *****
5 *****
```

*を a[i]個出力

74

配列の要素の参照例⑩

```
a=[1,2,3]
b=[4,5,6]
x=[]
```

```
a.length.times{ |i|
  [ a[i], b[i] ]
}
```

```
p a
p b
p x
```

二つの配列の要素の和の計算

練習問題③の回答

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

75

練習問題

配列に関する練習①～⑥
(簡単な人は⑥も行なって下さい)

76

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書きなさい

```
Z:\ruby>ruby sample.rb
[5, 4, 2, 7, 6]
max --> 7
min --> 2
```

77

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、times を一回用いて各要素に値を格納するプログラムを書きなさい。

78

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

参考ですが...

```
irb(main):003:0> a=[4,3,6,9,1]
=> [4, 3, 6, 9, 1]
irb(main):004:0> b=[1,9,5,2,3]
=> [1, 9, 5, 2, 3]
irb(main):005:0> a+b
=> [4, 3, 6, 9, 1, 1, 9, 5, 2, 3]
```

「+」演算子は二つの配列を結合

79

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)

```
Z:\Ruby> ruby sample.rb
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

80

練習問題⑤

- 配列 $x=[3,4,9,6,2]$ の要素を `times` メソッドを用いて、逆順に並び変えるプログラムを作成しなさい
- 配列 x の要素を直接入れ換えること

```
irb(main):001:0> x=[3,5,6,7]
=> [3, 5, 6, 7]
irb(main):002:0> x.reverse
=> [7, 6, 5, 3]
```

reverse というのがありますが使ってはいけません

81

補足: 入れ替え

- 変数値の入れ替え
 - 変数 a と変数 b に入っている値を入れ替えたい。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
```

- 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
```

83

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。

```
# coding: Windows-31J
a=[]
20.times{ |i|
  a[ i ] = rand( 100 )
}
p a
```

```
Z:\Ruby> sample.rb
[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。

```
Z:~Ruby>ruby sample.rb
[74, 11, 0, 15, 10, 28, 10, 98, 51, 29, 96, 13, 21, 18, 41, 4, 32, 60, 56, 5]
最長の部分列の長さは 3
4<32<60

Z:~Ruby>ruby sample.rb
[34, 61, 78, 98, 94, 37, 80, 9, 34, 45, 82, 43, 96, 44, 19, 56, 23, 19, 16, 0]
最長の部分列の長さは 4
34<61<78<98
9<34<45<82
```

34<61<78<98

9<34<45<82

練習問題

- 練習問題①から⑥を行ないなさい。
- 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。