# プログラミング言語　第十回

担当：篠沢 佳久
櫻井 彰人

平成23年 7月4日



## 本日の内容

- 二次元配列 (2)
- 二次元配列と繰り返し
- 練習問題①〜④



## 連絡事項①

- 第一回レポート
  - 61005157, 61008583, 61012042
- 第二回レポート
  - 61005551, 61018485
- 提出してもらったファイルが壊れてました．program-lang@ae.keio.ac.jp宛にファイルを送って下さい．



## 連絡事項②

- 来週 (7/11) は講義中に最終問題を行ないます
- これまでの練習問題，レポート，自習問題などで復習しておいて下さい



## 二次元配列 (復習)

二次元配列の宣言
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

3×3の行列 a

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Ruby での宣言

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

「, 」で区切る

さらに[]で囲む



## 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

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

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

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

p a[ 2 ][ 0 ]
p a[ 2 ][ 1 ]
p a[ 2 ][ 2 ]
```

```
C:¥Ruby>ruby sample.rb
1
2
3
4
5
6
7
8
9
```



## 二次元配列の要素の参照③

```
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



## 二次元配列の要素の参照④

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

p a[ 0 ]
p a[ 1 ]
p a[ 2 ]
```

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

a[ 0 ]

a[ 1 ]

a[ 2 ]



## 二次元配列の要素の参照⑤

要素数

a.length

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | a[ 0 ] |
| 4 | 5 | 6 | a[ 1 ] |
| 7 | 8 | 9 | a[ 2 ] |

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

```
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

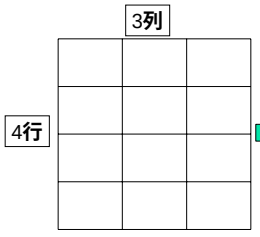
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

```
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




```
a = Array.new( 4 )

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



## 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
```

```
C:¥Ruby>ruby sample.rb
[[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[ 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

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



## 二次元配列と繰り返し

二重ループ中での要素の参照

## 二重ループ（復習）

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

```
C:¥Ruby>ruby sample.rb
0 + 0 = 0
0 + 1 = 1
0 + 2 = 2
1 + 0 = 1
1 + 1 = 2
1 + 2 = 3
2 + 0 = 2
2 + 1 = 3
2 + 2 = 4
3 + 0 = 3
3 + 1 = 4
3 + 2 = 5
```

i = 0 の時, j = 0〜2

i = 1 の時, j = 0〜2

i = 2 の時, j = 0〜2

i = 3 の時, j = 0〜2



```
i=0
3.times{ |j|
        print( i , " + " , j , " = " , i + j , "¥n" )
    }
}
i=1
3.times{ |j|
        print( i , " + " , j , " = " , i + j , "¥n" )
    }
}
i=2
3.times{ |j|
        print( i , " + " , j , " = " , i + j , "¥n" )
    }
}
i=3
3.times{ |j|
        print( i , " + " , j , " = " , i + j , "¥n" )
    }
}
```

## 二次元配列の要素

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

⇔

| | | |
|---|---|---|
| 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] |
| a[3][0] | a[3][1] | a[3][2] |



## 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

```
C:¥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



```
3.times{ |j|
    print( " a[ 0 ][ " , j , " ] = " , a[ 0 ][ j ] , "¥n" )
}
3.times{ |j|
    print( " a[ 1 ][ " , j , " ] = " , a[ 1 ][ j ] , "¥n" )
}
3.times{ |j|
    print( " a[ 2 ][ " , j , " ] = " , a[ 2 ][ j ] , "¥n" )
}
3.times{ |j|
    print( " a[ 3 ][ " , j , " ] = " , a[ 3 ][ j ] , "¥n" )
}
```



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

i=0, j=0〜3

i=1, j=0〜3

i=2, j=0〜3

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

j i

```
4.times{ |j|
    print( " a[ " , j , " ][  0 ] = " , a[ j ][ 0 ] , "¥n" )
}
```

```
4.times{ |j|
    print( " a[ " , j , " ][  1 ] = " , a[ j ][ 1 ] , "¥n" )
}
```

```
4.times{ |j|
    print( " a[ " , j , " ][  2 ] = " , a[ j ][ 2 ] , "¥n" )
}
```



## 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

```
C:¥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



## 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" )
   }
}
```

```
C:¥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
```



## 二次元配列の要素の参照③'

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

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

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

i　j



## step (復習)

```
0.step(10,2){ |x|
   print( x , "¥n" )
}
```

```
C:¥ruby>ruby sample.rb
0
2
4
6
8
10
```

```
10.step(0,-2){ |x|
   print( x , "¥n" )
}
```

```
C:¥ruby>ruby sample.rb
10
8
6
4
2
0
```



## 二次元配列の要素の参照④

要素番号 (インデックス) ではなく直接、二次元配列の要素を参照するには？

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

a.each{ |i|
   p  i
}
```

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

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

変数 i には順番に「配列」
a[ 0 ] , a[ 1 ] , a[ 2 ] , a[ 3 ] が代入される

# 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[ 0 ].each{ |j|
   print( j , "¥n" )
}
a[ 1 ].each{ |j|
   print( j , "¥n" )
}
a[ 2 ].each{ |j|
   print( j , "¥n" )
}
a[ 3 ].each{ |j|
   print( j , "¥n" )
}
```

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

a.each{ |i|
   p  i
   i.each{ |j|
      print( j , "¥n" )
   }
}
```

```
C:¥Ruby>ruby sample.rb
[1, 2, 3]
1
2
3
 i にa[0]が代入された場合
[4, 5, 6]
4
5
6
 i にa[1]が代入された場合
[7, 8, 9]
7
8
9
 i にa[2]が代入された場合
[10, 11, 12]
10
11
12
 i にa[3]が代入された場合
```

jには配列の値が代入されます



# 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[ 0 ].each{ |j|
   print( j , "¥n" )
}
a[ 1 ].each{ |j|
   print( j , "¥n" )
}
a[ 2 ].each{ |j|
   print( j , "¥n" )
}
a[ 3 ].each{ |j|
   print( j , "¥n" )
}
```



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

```
C:¥Ruby>ruby sample.rb
[1, 2, 3]
1
2
3
 i にa[0]が代入された場合
[4, 5, 6]
4
5
6
 i にa[1]が代入された場合
[7, 8, 9]
7
8
9
 i にa[2]が代入された場合
[10, 11, 12]
10
11
12
 i にa[3]が代入された場合
```

jには0,1,2が代入されます



# 二次元配列の要素の参照④''と④'''
# 違いをよく理解して下さい

```
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



# 二次元配列の要素の参照⑤

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

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

配列の要素数が異なってもよい

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

a[ 0 ]

a[ 1 ]

a[ 2 ]

a[ 3 ]

## 二次元配列の要素の参照⑤'

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

a.each{ |i|
    p  i
    p  i.length
}
```

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

a[ 0 ]　a[ 1 ]　a[ 2 ]　a[ 3 ]

**変数 i には a[0], a[1], a[2], a[3] が代入されていく**



## 二次元配列の要素の参照⑤''

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

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

```
C:¥Ruby>ruby sample.rb
1
2
3
4
5
6
7
8
9
10
```

a[ 0 ]　a[ 1 ]　a[ 2 ]　a[ 3 ]

**変数 i には a[0], a[1], a[2], a[3] が代入されていく**



## 二次元配列の要素の参照⑤'''

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

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

```
C:¥Ruby>ruby sample.rb
1
2
3
4
5
6
7
8
9
10
```

a[ 0 ]　a[ 1 ]　a[ 2 ]　a[ 3 ]

**変数 i には a[0], a[1], a[2], a[3] が代入されていく**
**変数 j には a[ i ]の要素が代入されていく**



## 二次元配列の要素への代入①

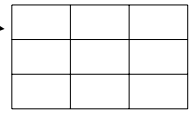
```
a=[]

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

```
C:¥Ruby>ruby sample.rb
[[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=[]

3.times{ |i|
  a[ i ] = Array.new( 3 )
}

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

p a
```

配列の要素数は3

```
a=[]

count = 1
3.times{ |i|
   a[ i ] = Array.new( 3 )
   3.times{ |j|
         a[ i ][ j ] = count
         count += 1
   }
}

p 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
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

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

## 二次元配列の要素への代入②

```
a=[]

(0..2).each{ |i|
   a[ i ] = []
   (0..2).each{ |j|
    a[ i ][ j ] = rand( 10 )
   }
}

p a
```

0から9の乱数を代入

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



## 二次元配列の要素への代入②'

forループ

```
a=[]

for i in (0..2) do
   a[ i ] = []
   for j in (0..2) do
    a[ i ][ j ] = rand( 10 )
   end
end

p a
```

0から9の乱数を代入

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



## forループ（復習）

```
for x in 1..10 do
  puts( 2**x )
end
```

x=1,2,・・・,9,10

```
C:¥ruby>ruby sample.rb
2
4
8
16
32
64
128
256
512
1024
```



## 二次元配列の要素への代入③

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

b=[]
3.times{ |i|
   b[ i ] = []
   3.times{ |j|
     b[ i ][ j ] = a[ j ]
   }
}

p b
```

```
C:¥Ruby>ruby sample.rb
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```



## 二次元配列の要素への代入③'

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

b=[]
3.times{ |i|
   b[ i ] = []
   3.times{ |j|
     b[ i ][ j ] = a[ i ]
   }
}

p b
```

```
C:¥Ruby>ruby sample.rb
[[1, 1, 1], [2, 2, 2], [3, 3, 3]]
```



## 二次元配列の要素への代入④

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

b=[]
3.times{ |i|
   b[ i ] = []
   b[ i ] = a
   p b
}

p b
```

b[0],b[1],b[2]にaを代入（?）

```
C:¥Ruby>ruby sample.rb
[[1, 2, 3]]
[[1, 2, 3], [1, 2, 3]]
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```

## 二次元配列の要素への代入④'

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

b=[]
3.times{ |i|
   b[ i ] = []
   b[ i ] = a
}

p b

a[0]=100
p b
```

b[0],b[1],b[2]にaを代入 (?)

配列aの要素を変えるとbも変わることに注意！

```
C:¥Ruby>ruby sample.rb
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
[[100, 2, 3], [100, 2, 3], [100, 2, 3]]
```



## (復習) 注意: 配列の要素の参照例

「配列のコピー」にはなりません！

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

配列xにコピー?

配列aだけ変更

しかし、x も変っている！

```
C:¥ruby>ruby sample.rb
[4, 2, 1, 6, 7]
[4, 2, 1, 6, 7]
[10, 2, 1, 6, 7]
[10, 2, 1, 6, 7]
```

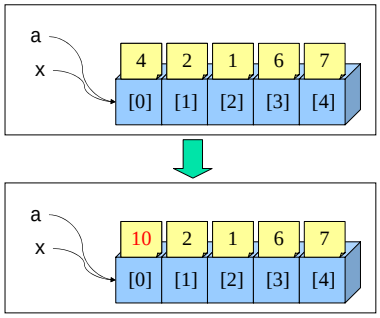


## (復習) 注意': 配列の要素の参照例

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

a[0]=10

p a, x
```






## 二次元配列の要素への代入⑤

```
a=[]
3.times{ |i|
   a[ i ] = []
   3.times{ |j|
  a[ i ][ j ] = gets.chomp.to_i
   }
}

p a
```

キーボード入力

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



## 二次元配列のコピー①

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

b = []
a.length.times{ |i|
   b[ i ] = []
   a[ i ].length.times{ |j|
      b[ i ][ j ] = a[ i ][ j ]
   }
}

p b
```

配列 b の宣言

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



## 二次元配列のコピー①'

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

b = []
a.length.times{ |i|
   b[ i ] = []
   b[ i ] = a[ i ]
}

p b
```

配列 b の宣言

b[i] に a[i] を代入 (?)

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

# 二次元配列のコピー①''

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

b = []
a.length.times{ |i|
   b[ i ] = []
   b[ i ] = a[ i ]
}

p b
a[0][0] = 100
p b
```

配列 b の宣言

b[i] に a[i] を代入 (?)

配列aの要素を変えるとbも変わることに注意！

```
C:¥Ruby>ruby sample.rb
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
[[100, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```



# 二次元配列のコピー②

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

b = []
3.times{ |i|
   b[ i ] = []
   a.length.times{ |j|
      [                ]
   }
}
```

練習問題の答えです…

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

行列の転置

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



# 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

```
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]]
```



# 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]
]

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

p[i][1] i番目の学生の国語の成績
p[i][2] i番目の学生の数学の成績
p[i][3] i番目の学生の英語の成績

```
C:¥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
```

p[i][0] i番目の学生の出席番号

## 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 ] |

p[ 0 ], p[ 1 ], p[ 2 ], p[ 3 ]の順番に計算

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

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]
]

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

```
C:¥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

```
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 #{p[i][0]} 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'

```
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 #{p[i][0]} の人の平均点は #{ave} です" )
}
```

```
C:¥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 #{p[i][0]} の人の平均点は #{ave} です" )
}
```

```
C:¥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

```
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" )
}
```

```
C:¥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'

```
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…

```
C:¥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

a[0]は出席番号なので加算しない



## 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 =[]
(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" )
}
```

```
C:¥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



## 練習問題

練習問題①～④

## 練習問題①

- 配列 a を3×3の二次元配列とする
- 二重ループを用いて下記のような要素を持つ配列にしなさい

```
a=[
    [ 1 , 0 , 0 ] ,
    [ 0 , 1 , 0 ] ,
    [ 0 , 0 , 1 ]
  ]
```

```
C:¥Ruby>ruby sample.rb
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

```
a=[
    [ 0 , 0 , 1 ] ,
    [ 0 , 1 , 0 ] ,
    [ 1 , 0 , 0 ]
  ]
```

```
C:¥Ruby>ruby sample.rb
[[0, 0, 1], [0, 1, 0], [1, 0, 0]]
```



## 練習問題②

- 下記のような出力を行なうプログラムを二重ループを用いて書きなさい

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

9行9列



## 練習問題③

- 二次元配列 a の転置行列を出力するプログラムを二重ループを用いて作成しなさい

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

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



## 練習問題④

- スライド「2次元表で平均値を求める」(56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



## 練習問題④ (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



## 練習問題

- 練習問題①から④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。

# 最終問題 (もう一度)

- 来週 (7/11) は講義中に最終問題を行ないます
- これまでの練習問題，レポート，自習問題などで復習しておいて下さい