プログラミング言語　第十一回

担当：篠沢 佳久
櫻井 彰人

平成21年 6月29日

本日の内容

- 二次元配列（2）
- 二重ループ（3）
  - 二次元配列と繰り返し
  - 二重ループでの利用

レポート課題①ヒント

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


]
```

二次元配列を利用すれば…

一フレーム目

二フレーム目

三フレーム目

レポート課題②ヒント

- 5/11の練習問題③を参照

最終レポート

- 7/13（月），講義中に行ないます．
- 必ず出席して下さい．
- しっかりとこれまでの復習してきて下さい．

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 ]
]
```

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 ]
]

p a
```

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

一次元配列

二次元配列は一次元配列の要素を一次元配列としているとみなせる

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 ]
]
```

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③

```
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④

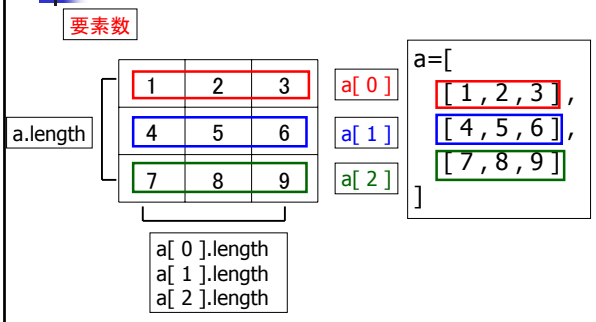
```
a=[
   [ 1 , 2 , 3 ],
   [ 4 , 5 , 6 ],
   [ 7 , 8 , 9 ]
]

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

配列a の要素数

```
C:¥Ruby>ruby sample.rb
3
3
3
3
```

a[0], a[1], a[2] の要素数

二次元配列の要素の参照⑤



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

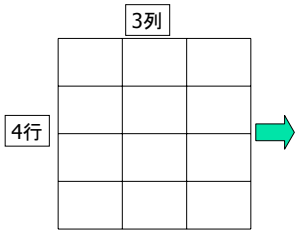
- 要素の値が分かっている場合

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]]
```


二次元配列の宣言④

配列の宣言をしない場合

```
a = []

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
sample.rb:3: undefined method `[]=' for nil:NilClass (NoMethodError)
```


二次元配列と繰り返し①

二重ループ中での要素の参照
for ループ

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

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

i にa[0]が代入された場合
i にa[1]が代入された場合
i にa[2]が代入された場合
i にa[3]が代入された場合

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

i にa[0]が代入された場合
i にa[1]が代入された場合
i にa[2]が代入された場合
i にa[3]が代入された場合

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
```

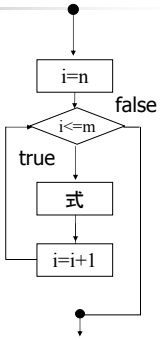
a[0]
a[1]
a[2]
a[3]

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

for と each ループ

```
for i in n..m do
   式
end
```

```
(n..m).each{ |i|
   式
}
```



for と each ループ②

- Ruby には for 構文があります。
  - CやJavaと似ていますが少し違います。
    - 使いやすいので、CやJavaに戻れない、、、

```
(5..10).each {|i|
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
}
```

```
for i in (5..10) do
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
end
```

```
5 は奇数
6 は偶数
7 は奇数
8 は偶数
9 は奇数
10 は偶数
```


配列を使う each と for

- 「範囲式」の代わりに配列が使えます

```
a = [3,1,4,1,5]
a.each {|i|
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
}
```

```
a = [3,1,4,1,5]
for i in a do
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
end
```

```
3 は奇数
1 は奇数
4 は偶数
1 は奇数
5 は奇数
=> [3, 1, 4, 1, 5]
```

```
[3,1,4,1,5].each {|i|
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
}
```

```
for i in [3,1,4,1,5] do
  print( i )
  if i%2==0 then
    print( " は偶数" )
  else
    print( " は奇数" )
  end
  print( "¥n" )
end
```


forループを用いて書くと…①

```
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" )
   }
}
```

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

for i in 0..a.length-1 do
   for j in 0..a[ i ].length-1 do
      print( " a[ " , i , " ][ " , j , " ] =
          " , a[ i ][ j ] , "¥n" )
   end
end
```

forループを用いて書くと…②

```
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 ]
]

for i in a do
   p i
   for j in i do
      print( j , "¥n" )
   end
end
```

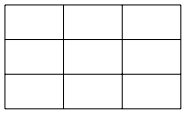

二次元配列と繰り返し②

二重ループ中での要素への代入

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①'''

```
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

```
a=[]

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

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

i と j が同じ場合は 1
それ以外は 0

```
C:¥Ruby>ruby sample.rb
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=[]

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

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

要素数が異なってもよい

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


二次元配列の要素への代入③'

```
a=[]

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

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

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

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

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

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

二次元配列の要素への代入④

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

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

どうして配列の要素がこのようになるでしょうか？

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


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
```

二次元配列の要素への代入⑤

```
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⑥

```
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]]
```


（例題）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 #{i+1} 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

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

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


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 #{i+1} 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 #{i+1} の人の平均点は #{ave} です" )
}
```

このプログラムの場合，科目数や学生が追加された場合，修正しなければならない

各人の平均点を求めるプログラム②'

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

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

```
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 #{i+1} 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②'

for ループを用いた場合

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

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

```
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 = [
    [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

標準偏差を求めるプログラム

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

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

平均を求める


```
sd = []
(0..p.length-1).each{ |i|
  sum = 0
  (1..p[i].length-1).each{ |j|
    sum += ( p[i][j] - ave[i] ) * ( p[i][j] - ave[i] )
  }
  sd[ i ] = Math.sqrt( sum / ( p[ i ].length-2 ) )
}

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

平均との差の二乗和

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

練習問題

練習問題①②③

練習問題①

- 下記は10未満の整数の要素を持つ3×3の二次元配列を作成するプログラムです．要素中，最小値，最大値を求めるプログラムを追加しなさい

```
a = []
3.times{ |i|
   a[ i ] = []
   3.times{ |j|
        a[ i ][ j ] = rand(10)
   }
}
p a
```

```
C:¥Ruby>ruby sample.rb
[[3, 0, 1], [2, 6, 9], [6, 4, 3]]
 min -> 0
 max -> 9
```


練習問題②

- スライド「2次元表で平均値を求める」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②（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③

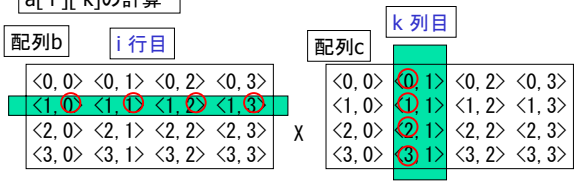
- 二つの正方行列の加算、乗算および一つの正方行列の転置を行うプログラムを二重ループを用いて書きなさい
- 行列要素の値は、プログラム内に書いてあってよい

プログラムの大枠

```
a = [[],[],[]]
b = [[10,20,30],[40,50,60],[70,80,90]]
c = [[100,200,300],[400,500,600],[700,800,900]]

(0..a.length-1).each{ |i|
  (0..a[i].length-1).each{ |j|
          要素ごとの足し算
  }
}
p( a )

# a = b*c
# 印字

# a = bの転置
# 印字
```

念のため: 行列の積

a[i][k]の計算



行列の積のプログラム(ヒント)

三重ループ

```
(0..a.length-1).each{ |i|
  (0..a[i].length-1).each{ |k|
    sum = 0.0
    (0..c.length-1).each{ |j|
       sum に b[i][j] と c[j][k] の積を足す
    end
    a[i][k] に sum を代入
  }
}
```


行列の転置のプログラム(ヒント)

配列b

<0,0>	<0,1>	<0,2>
<1,0>	<1,1>	<1,2>
<2,0>	<2,1>	<2,2>

配列a

<0,0>	<1,0>	<2,0>
<0,1>	<1,1>	<2,1>
<0,2>	<1,2>	<2,2>

練習問題

- 練習問題①②③を行ないなさい。
- 簡単なものは，自習問題を別の方法でプログラミングしてみて下さい。
- 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。