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29年 6月26日

本日の内容

- 二次元配列
- 二次元配列と繰り返し
- ファイル操作

- 練習問題①～⑤

二次元配列

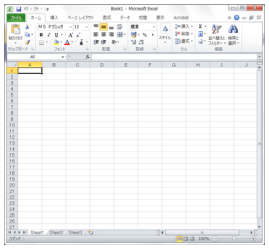
二次元配列の宣言
要素の参照，代入

2次元配列

- 表が使えると、随分便利です。
- スプレッドシートを思い起こしてください
  - スプレッドシートって何ですか？

二次元の配列＝二次元の表（行列）

- 表といえば、二次元かな。
- 表計算ソフトも2次元だしな。



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

7

二次元配列の宣言③

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

p a
```

```
Z:¥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

8

二次元配列の宣言④

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

p a
```

一次元配列

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

9

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]

10

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]
```

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

11

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

12

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 ]
```

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

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

13

二次元配列の要素の参照⑤

```
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

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

a[0], a[1], a[2] の要素数

14

二次元配列の要素の参照⑤'

```
a=[
   [ 1 ],
   [ 4 , 5 ],
   [ 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

```
Z:¥Ruby>ruby sample.rb
3
1
2
3
```

a[0], a[1], a[2] の要素数

15

二次元配列の要素の参照⑥

要素数

a.length

1	2	3
4	5	6
7	8	9

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

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

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

16

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 ]
]
```

17

二次元配列の宣言②

- 要素数のみ分かっている場合

3列

4行

```
a = Array.new( 4 )

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

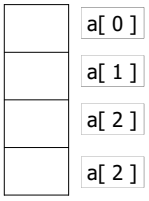
18

二次元配列の宣言②'

要素数4の配列a を作成

```
a = Array.new( 4 )

p a
```




```
Z:¥Ruby>ruby sample.rb
[nil, nil, nil, nil]
```

値が入っていないので「nil」となる

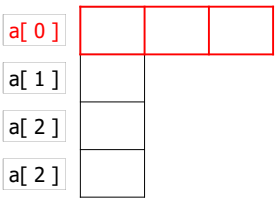
19

二次元配列の宣言②'

```
a = Array.new( 4 )
a[ 0 ] = Array.new( 3 )

p a
```

配列a[0]に要素が3の配列を作成




```
Z:¥Ruby>ruby sample.rb
[[nil, nil, nil], nil, nil, nil]
```

a[0] のみ3個の要素を持つ配列

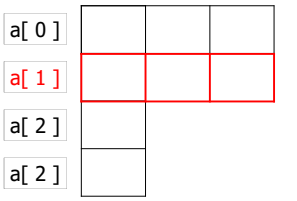
20

二次元配列の宣言②'

```
a = Array.new( 4 )
a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )

p a
```

配列a[1]に要素が3の配列を作成



a[0] , a[1]
3個の要素を持つ配列

```
Z:¥Ruby>ruby sample.rb
[[nil, nil, nil], [nil, nil, nil], nil, nil]
```

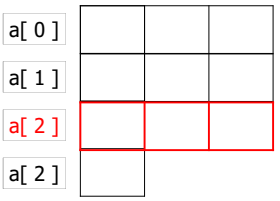
21

二次元配列の宣言②'

```
a = Array.new( 4 )
a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )
a[ 2 ] = Array.new( 3 )

p a
```

配列a[2]に要素が3の配列を作成



a[0] , a[1] , a[2]
3個の要素を持つ配列

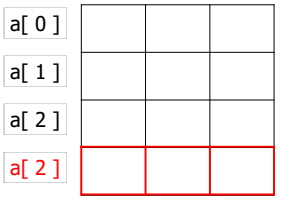
```
Z:¥Ruby>ruby sample.rb
[[nil, nil, nil], [nil, nil, nil], [nil, nil, nil], nil]
```

22

二次元配列の宣言②'

```
a = Array.new( 4 )
a[ 0 ] = Array.new( 3 )
a[ 1 ] = Array.new( 3 )
a[ 2 ] = Array.new( 3 )
a[ 3 ] = Array.new( 3 )
p a
```

配列a[3]に要素が3の配列を作成




```
Z:¥Ruby>ruby sample.rb
[[nil, nil, nil], [nil, nil, nil], [nil, nil, nil], [nil, nil, nil]]
```

23

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
```

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

24

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

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

25

二次元配列の宣言③'

```
a =[]

p a
```

配列a を作成

```
Z:¥Ruby>ruby sample.rb
[]
```

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

p a
```

配列a[0] を作成

```
Z:¥Ruby>ruby sample.rb
[[]]
```

26

二次元配列の宣言③"

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

p a
```

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

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

a[ 1 ] =[]

p a
```

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

27

二次元配列の宣言③'''

```
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

p a
```

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

28

二次元配列の宣言③''''

```
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 ] = []
p a

a[ 2 ][ 0 ] = 7
a[ 2 ][ 1 ] = 8
a[ 2 ][ 2 ] = 9

p a
```

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

29

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
```

```
Z:¥Ruby>ruby sample.rb
sample.rb:3: undefined method `[]=' for nil:NilClass (NoMethodError)
```

30

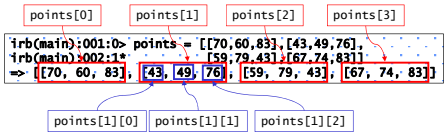
二次元配列の宣言のまとめ

- 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

31

二次元配列のまとめ①

- 一次元配列の一次元配列
  - Ruby では、子供の一次元配列の長さは異なってよい。Java でも同様。Cではダメ。



points.length は 4, points[0].length は 3

32

二次元配列のまとめ②

- 一次元配列の一次元配列だから

```
irb(main):001:0> points = [[1, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79,
43], [4, 67, 74, 83]]
=> [[1, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79, 43], [4, 67, 74, 83]]
irb(main):002:0> points[0]
=> [1, 70, 60, 83]
irb(main):003:0> points[0][0] = 999
=> 999
irb(main):004:0> points
=> [[999, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79, 43], [4, 67, 74, 83]]
```

```
irb(main):004:0> points
=> [[999, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79, 43], [4, 67, 74, 83]]
irb(main):005:0> p = points[0]
=> [999, 70, 60, 83]
irb(main):006:0> p[0] = 99
=> 99
irb(main):007:0> p
=> [99, 70, 60, 83]
irb(main):008:0> points
=> [[99, 70, 60, 83], [2, 43, 49, 76], [3, 59, 79, 43], [4, 67, 74, 83]]
```

p と points[0] が同じものとなる

注意!

p の要素を変えたら points[0] の要素が変わった！

33

二次元配列と繰り返し①

二重ループ中での要素の参照

34

二重ループ（復習）

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

```
Z:¥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

35

```
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

```
Z:¥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

37

```
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" )
}
```

38

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

```
Z:¥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

39

```
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" )
}
```

40

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

```
Z:¥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

41

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" )
  }
}
```

```
Z:¥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
```

42

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
}
```

```
Z:¥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

43

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" )
}
```

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

44

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" )
   }
}
```

```
Z:¥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

45

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" )
}
```

46

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" )
   }
}
```

```
Z:¥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

47

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" )
   }
}
```

jには配列の値が代入されます

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

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

jには0,1,2が代入されます

i にはa[0],a[1],a[2],a[3]が代入されます

48

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

```
Z:¥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
}
```

```
Z:¥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" )
    }
}
```

```
Z:¥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" )
    }
}
```

```
Z:¥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

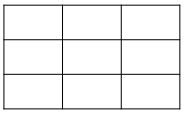
```
a=[]

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

```
Z:¥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

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

55

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

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

56

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
```

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

57

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(?)

```
Z:¥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]]
```

58

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！

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

59

(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！

```
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]
```

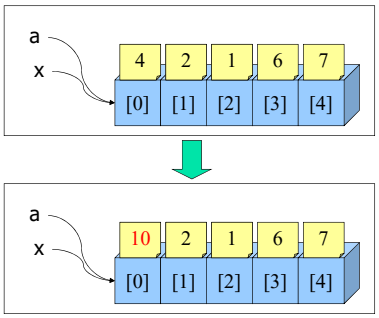
60

(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
```



61

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

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

62

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

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

63

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(?)

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

64

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！

```
Z:¥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]]
```

65

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

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

66

ファイル操作

ファイルからの読み込み，書き込み

キーボードからの入力（復習）

- line = gets.chop
- line = gets.chomp

- gets
  - キーボードから文字列を読み込む
  - この場合，改行文字が文字列の最後に含む
- chop
  - 最後の一文字を削除する
- chomp
  - 最後の一文字が改行の場合のみ改行コードを削除する

- line には読み込まれた文字列が代入される
- 文字列のため，数字に「to_i」「to_f」を用いて数値に変換する

ファイルからの読み込み①

```
while 継続条件 do #継続条件が真の間、処理を繰り返す
                  #継続か否かの判定は、一番最初
   式1
   式2
    .
    .
end

# よくある使い方
第一回目のための準備
while 継続条件 do
   するべき作業
   次回への準備
end
```

一行読み込む

```
# coding: Windows-31J
open( "HumptyDumpty.txt" ) { | f |
  ln = 1; l = f.gets          # 第一回目のための準備
  while !(l.nil?) do
    print( ln, "行目:", l )   # 作業
    ln += 1; l = f.gets       # 次回への準備
  end
}
```

```
1行目:Humpty Dumpty sat on a wall.
2行目:Humpty Dumpty had a great fall.
3行目:All the king's horses and all the king's men
4行目:Couldn't put Humpty together again.
```

http://www.authorama.com/through-the-looking-glass-6.html

ファイルからの読み込み②

読み込みたいファイル名を記述

```
open( "ファイル名") { |f|
    while line = f.gets do

    end
}
```

文字列変数 line にファイルから一行読み込まれる

ファイルの末尾まで読み込まれるとwhile 文から抜け出る

ファイルからの読み込み③

```
open("HumptyDumpty.txt") { |f|
    while line = f.gets  do
        puts( line )
    end
}
```

ファイルから一行づつ読み込む

一行出力

```
Z:¥Ruby>ruby sample.rb
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
```

HumptyDumpty.txt

```
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
```


ファイルからの読み込み④

```
# coding: Windows-31J
sum = 0
open( "file.txt" ) { |f|
  while line = f.gets  do
      x = line.chomp.to_i
      print( x , " " )
      sum += x
  end
  print( "¥n 合計:" , sum )
}
```

file.txt

```
1
2
3
4
5
6
7
8
9
10
```

```
Z:¥Ruby>ruby sample.rb
1 2 3 4 5 6 7 8 9 10
 合計:55
```

ファイルからの読み込み⑤

```
# coding: Windows-31J
sum = 0
x =[]
open( "file1.txt" ) { |f|
   while line = f.gets  do
        x = line.chomp.split(/,/)
        print( x[ 0 ] , " " , x[ 1 ] , "¥n" )
        sum += x[ 1 ].to_i
   end
   print( "¥n 合計:" , sum )
}
```

file1.txt

```
A,1
B,2
C,3
D,4
E,5
F,6
G,7
H,8
I,9
J,10
```

73

ファイルからの読み込み⑤'

```
# coding: Windows-31J
sum = 0
x =[]
open( "file1.txt" ) { |f|
   while line = f.gets  do
        x = line.chomp.split(/,/)
        print( x[ 0 ] , " " , x[ 1 ] , "¥n" )
        sum += x[ 1 ].to_i
   end
   print( "¥n 合計:" , sum )
}
```

「,」で分割し，x[0]とx[1]に代入する

```
Z:¥Ruby>ruby sample.rb
A 1
B 2
C 3
D 4
E 5
F 6
G 7
H 8
I 9
J 10

合計:55
```

74

ファイルへの書き込み①

```
f = open( "text.txt" , "w" )
10.times{ |i|
   f.print( i , "¥n" )
}
f.close
```

```
Z:¥Ruby>ruby sample.rb

Z:¥Ruby>type text.txt
0
1
2
3
4
5
6
7
8
9
```

type
ファイルの内容を見るコマンド

75

ファイルへの書き込み②

```
f = open( "text.txt" , "w" )
10.times{ |i|
   f.print( i , "¥n" )
}
f.close
```

text.txt という名前のファイル（書き込み用）を準備

text.txt に書き込む

ファイルを閉じる（忘れずに！）

open("ファイル名" , "モード")

モード（省略した場合は「r」）
w　書き込み用
r　読み込み用
a　追加書き込み用

76

ファイルへの書き込み③

```
s = open( "text.txt" , "w" )
open("HumptyDumpty.txt") { |f|
   while line = f.gets  do
        s.print( line )
   end
}
s.close
```

f　読み込み用

s　書き込み用

```
Z:¥Ruby>ruby sample.rb

Z:¥Ruby>type text.txt
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
```

77

練習問題

練習問題①～⑤

78

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]
  ]
```

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

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

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

練習問題②

- 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（スライド55ページを参照）

```
a=[
    [0,2,0],
    [0,5,0],
    [0,8,0]
  ]
```

```
Z:¥Ruby>ruby sample.rb
[[0, 2, 0], [0, 5, 0], [0, 8, 0]]
```

```
a=[
    [1,2,0],
    [4,0,6],
    [0,8,9]
  ]
```

```
Z:¥Ruby>ruby sample.rb
[[1, 2, 0], [4, 0, 6], [0, 8, 9]]
```

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作成しなさい

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

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

行列の転置のプログラム（ヒント）

配列a

<0,0>	<0,1>	<0,2>
<1,0>	<1,1>	<1,2>
<2,0>	<2,1>	<2,2>
<3,0>	<3,1>	<3,2>

配列a^t

<0,0>	<1,0>	<2,0>	<3,0>
<0,1>	<1,1>	<2,1>	<3,1>
<0,2>	<1,2>	<2,2>	<3,2>

練習問題④

- 二次元配列 a, b の和の行列cと差の行列dを求めるプログラムを二重ループを用いて書きなさい

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

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

```
Z:¥Ruby>ruby sample.rb
c --->
[[10, 10, 10], [10, 10, 10], [10, 10, 10]]

d --->
[[-8, -6, -4], [-2, 0, 2], [4, 6, 8]]
```

練習問題④

- （さらに頑張れる人へ）二次元配列 a, b の積の行列eを求めるプログラムを三重ループ（二重ループでもよい）を用いて書きなさい

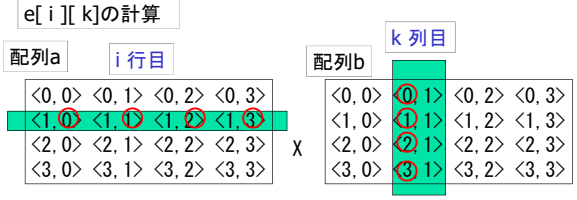
```
a=[
    [1,2,3],
    [4,5,6],
    [7,8,9]
  ]
```

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

```
Z:¥Ruby>ruby sample.rb
e --->
30 24 18
84 69 54
138 114 90
```

念のため: 行列の積

e[i][k]の計算




```
e[ i ][ k ] = a[ i ][ 0 ] * b[ 0 ][ k ] + a[ i ][ 1 ] * b[ 1 ][ k ]
            + a[ i ][ 2 ] * b[ 2 ][ k ] + a[ i ][ 3 ] * b[ 3 ][ k ]
```

85

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..b.length-1).each{ |j|
       sum に a[i][j] と b[j][k] の積を足す
    }
    e[i][k] に sum を代入
  }
}
```

86

練習問題⑤

- HumptyDumpty.txtの内容が最後の行から印字されるようにサンプルプログラムを改良しなさい

```
Z:¥Ruby>ruby sample.rb
Couldn't put Humpty together again.
All the king's horses and all the king's men
Humpty Dumpty had a great fall.
Humpty Dumpty sat on a wall.
```

- HumptyDumpty.txtはプログラムと同じフォルダーに置いて下さい

87

練習問題⑤

```
open("HumptyDumpty.txt") { |f|
    while line = f.gets  do
        puts( line )
    end
}
```

プログラムを実行した場合

```
Z:¥Ruby>ruby sample.rb
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
```

HumptyDumpty.txt

```
Humpty Dumpty sat on a wall.
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
```

88

練習問題⑤(ヒント)

- ファイルから，一行ずつ読み込んだ文字列を配列に格納する
- 読み込み終了後，配列の最後の要素から印字する

89

練習問題

- 練習問題①から⑤を(できるだけ)(頑張って)行ないなさい。
- プログラムと実行結果をワープロに貼り付けて、keio.jp から提出して下さい。

90