プログラミング言語　第九回

担当：篠沢 佳久
櫻井 彰人

平成21年 6月 15日

本日の内容

- 一次元配列の復習
- 多重ループ
- ネスト（入れ子）構造
- 練習問題①～④

ネスト（入れ子）

- ネストする: 入れ子にすること

箱根の十二卵（田中一幸氏作） 左端は実際の鶏卵のLL玉ほど。右端は13番目のヒヨコ

http://dadandmam.whitesnow.jp/sub3-011.htm

配列の復習

一次元配列と繰り返し

配列の宣言

- 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 = []

配列の宣言②

要素が分かっている場合

a=[4, 6, 7, 9, 10]

	a	
0	4	a[0]
1	6	a[1]
2	7	a[2]
3	9	a[3]
4	10	a[4]

配列の宣言②

要素数が分かっている場合

```
a= Array.new(5)

a[0]=4
a[1]=6
a[2]=7
a[3]=9
a[4]=10
```

要素数が決まっていない場合

```
a= []

a[0]=4
a[1]=6
a[2]=7
a[3]=9
a[4]=10
```


配列の要素の参照方法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" )
}
```

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

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

配列名.length
配列の要素数

配列の要素の参照方法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" )
}
```


一次元配列のプログラム例

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5 , 100 , 50 ]
```

配列name

文字列型

	name
0	A
1	B
2	C
3	D
4	E

配列 test

整数型

	test
0	85
1	60
2	5
3	100
4	50

配列の要素への代入

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

name[ 3 ] = "d"
test[ 3 ] = 90

p name
p test
```

```
C:¥Ruby>ruby sample.rb
["A", "B", "C", "d", "E"]
[85, 60, 5, 90, 50]
```


最後の要素への追加

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

name[ name.length ] = "F"
test[ test.length ] = 70

p name
p test
```

```
C:¥Ruby>ruby sample.rb
["A", "B", "C", "D", "E", "F"]
[85, 60, 5, 100, 50, 70]
```

平均点を求める①

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

sum = 0
test.length.times{ |i|
   sum += test[ i ]
}
print( "平均点 --> " , sum / test.length )
```

times を用いた方法

```
C:¥ruby>ruby sample.rb
平均点 --> 60
```


平均点を求める②

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

sum = 0
(0..test.length-1).each{ |i|
   sum += test[ i ]
}
print( "平均点 --> " , sum / test.length )
```

each を用いた方法

```
C:¥ruby>ruby sample.rb
平均点 --> 60
```


平均点を求める③

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

sum = 0
test.each{ |i|
   sum += i
}
print( "平均点 --> " , sum / test.length )
```

each を用いた方法
配列の要素を直接参照

前頁との違いに注意して下さい

```
C:¥ruby>ruby sample.rb
平均点 --> 60
```


平均点未満の名前を出力

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]
sum = 0
test.each{ |i|
   sum += i
}
average = sum / test.length

print( "平均点未満は...¥n" )
test.length.times{ |i|
   if average > test[ i ] then
      print( name[ i ] , ": " , test[ i ] , "点¥n" )
   end
}
```

平均点を求める

```
C:¥ruby>ruby sample.rb
平均点未満は...
C: 5
E: 50
```


最高点とその名前を出力

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

max = 0
test.length.times{ |i|
   if test[ max ] < test[ i ] then
      max = i
   end
}
print( "最高点は" , name[ max ] , " の " , test[ max ] , "点です¥n" )
```

max
最高点の要素番号を格納する

現在の最高点と比較

```
C:¥ruby>ruby sample.rb
最高点はD の 100点です
```


最低点とその名前を出力

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

min = 0
test.length.times{ |i|
   if test[ min ] > test[ i ] then
      min = i
   end
}
print( "最低点は" , name[ min ] , " の " , test[ min ] , "点です¥n" )
```

min
最低点の要素番号を格納する

現在の最低点と比較

```
C:¥ruby>ruby sample.rb
最低点はC の 5点です
```

検索①

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

search = "B"
name.length.times{ |i|
   if name[ i ] == search then
        print( search , " の点は " , test[ i ] , "点です¥n" )
        break
   end
}
```

Bの点数を検索

```
C:¥ruby>ruby sample.rb
B の点は 60点です
```


配列の要素の検索

```
if name[ i ] == search then
```

配列name

	name
0	A
1	B
2	C
3	D
4	E

search = "B"

search と一致した場合，breakにより timesのループから抜け出るため，以降は照合しない

ローカル変数①

```
10.times{
   a = 10
}
print( a , "¥n" )
```

ローカル変数
ブロック*の範囲内でしか利用できない

```
C:¥Ruby>ruby sample.rb
sample.rb:4: undefined local variable or method `a' for main:Object
(NameError)
```

*Rubyには別の意味のブロックもあります

ブロック①

ブロック

```
test.length.times{ |i|
   if average > test[ i ] then
        print( name[ i ] , ": " , test[ i ] , "点¥n" )
   end
}
```

ブロック

ブロック②

ブロック

```
(0..9).each{ |x|
   (0..9).each{ |y|
        z = x*x + y*y
        print( " x = " , x , " y = " , y , ": z= " , z , "¥n" )
   }
}
```

ブロック

ローカル変数②

```
a = 0
10.times{
   a = 10
}
print( a , "¥n" )
```

ローカル変数
ブロックの前で宣言しておく

```
C:¥Ruby>ruby sample.rb
10
```

グローバル変数

```
10.times{
  $a = 10
}
print( $a , "¥n" )
```

グローバル変数
変数名の前に「$」をつける
プログラムのどこからでも参照できる

```
C:¥Ruby>ruby sample.rb
10
```


コマンドライン引数

コマンドライン引数①

- Rubyプログラムにデータを渡すことができます
  - データを整数に限ると次のようにできます

実行例と実行結果例

```
H:¥ruby>ruby sample0302 2 3
 2 + 3 = 5
H:¥ruby>ruby Sample0302 8 2
 8 + 2 = 10
```

引数

コマンドライン引数②

- ruby プログラム 値1 値2 値3
  - 値1, 値2, ・・・ を引数と呼ぶ
- 値1は ARGV[0] に格納される
- 値2は ARGV[1] に格納される
- 値3は ARGV[2]に格納される
- ARGV[0], ARGV[1], ARGV[2] は文字列型配列

コマンドライン引数③

- ruby プログラム 値1 値2 値3 … 値n
- 値1は ARGV[0] に格納される
- 値2は ARGV[1] に格納される
- 値nは ARGV[n]に格納される

コマンドライン引数:プログラム

プログラム例 (sample0302.rb)

```
print( ARGV[0] + " + " + ARGV[1]+ " = " )
print( ARGV[0].to_i + ARGV[1].to_i )
```

文字列型のため型変換(整数)が必要

または

```
print( "#{ARGV[0]} + #{ARGV[1]} = " )
print( ARGV[0].to_i + ARGV[1].to_i )
```

実行例と実行結果例

```
G:¥Ruby>ruby sample0302.rb 3 5
3 + 5 = 8
G:¥Ruby>ruby sample0302.rb 100 200
100 + 200 = 300
```

浮動小数点数にすると

プログラム例 (sample0303.rb)

```
print( ARGV[0] + " + " + ARGV[1]+ " = " )
print( ARGV[0].to_f + ARGV[1].to_f )
```

または

```
print( "#{ARGV[0]} + #{ARGV[1]} = " )
print( ARGV[0].to_f + ARGV[1].to_f )
```

文字列型のため型変換(小数)が必要

実行例と実行結果例

```
G:¥Ruby>ruby sample0303.rb 3 5
3 + 5 = 8.0
G:¥Ruby>ruby sample0303.rb 3.0 5.0
3.0 + 5.0 = 8.0
G:¥Ruby>ruby sample0303.rb 100 200.0
100 + 200.0 = 300.0
```


3個にすると

プログラム例 (sample0304.rb)

```
print( ARGV[0] + " + " + ARGV[1] + " + " + ARGV[2] + " = " )
print( ARGV[0].to_f + ARGV[1].to_f + ARGV[2].to_f )
```

実行例と実行結果例

```
G:¥Ruby>ruby sample0304.rb 1.23 4.56 7.89
1.23 + 4.56 + 7.89 = 13.68
```


検索②

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]

line = ARGV[ 0 ]
search = line.chomp

name.length.times{ |i|
   if name[ i ] == search then
        print( search , " の点は " , test[ i ] , "点です¥n" )
        break
   end
}
```

コマンドライン引数

検索②の実行結果

```
C:¥ruby>ruby sample.rb A
A の点は 85点です

C:¥ruby>ruby sample.rb B
B の点は 60点です

C:¥ruby>ruby sample.rb C
C の点は 5点です

C:¥ruby>ruby sample.rb D
D の点は 100点です

C:¥ruby>ruby sample.rb E
E の点は 50点です
```

ARGV[0] には"A¥n" が代入

↓

line = ARGV[0]

line には "A¥n"

↓

search = line.chomp

search には "A" が代入

二重ループ

一重ループ

$y = x^2$

```
10.times{ |x|
        y = x*x
        print( x , ": " , y , "¥n" )
}
```

```
(0..9).each{ |x|
        y = x*x
        print( x , ": " , y , "¥n" )
}
```

```
C:¥ruby>ruby sample.rb
0: 0
1: 1
2: 4
3: 9
4: 16
5: 25
6: 36
7: 49
8: 64
9: 81
```

二重ループの必要性①

$z = x^2 + y^2$

$0 \le x < 10, 0 \le y < 10$ の範囲で値を求めるには？

x=0 の時，yの値を0から9まで変えて z を求める

```
x = 0
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )
}
```


x=1 の時，yの値を0から9まで変えて z を求める

```
x = 1
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )
}
```

以下同様にx=9 まで同じことを繰り返し z を求める

```
x = 9
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )

}
```


二重ループの必要性②

```
x = 0
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )
}
```

```
x = 1
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )
}
```

```
x = 9
(0..9).each{ |y|
        z = x*x + y*y
        print( x , " " , y , ": " , z , "¥n" )
}
```

xの値も0から9まで一つずつ増やしていけばよい

二重ループ

①のループによって，xは0から9まで変わる

①のループ

```
(0..9).each{ |x|
    (0..9).each{ |y|
        z = x*x + y*y
        print( " x = " , x , " y = " , y , ": z= " , z , "¥n" )
    }
}
```

②のループ

②のループによって，xは0から9まで変わる

二重ループの出力結果①

①のループ中 x=0 として
②のループの処理を行なう

```
C:¥ruby>ruby sample.rb
 x = 0 y = 0: z= 0
 x = 0 y = 1: z= 1
 x = 0 y = 2: z= 4
 x = 0 y = 3: z= 9
 x = 0 y = 4: z= 16
 x = 0 y = 5: z= 25
 x = 0 y = 6: z= 36
 x = 0 y = 7: z= 49
 x = 0 y = 8: z= 64
 x = 0 y = 9: z= 81
```

①のループ中 x=1 として
②のループの処理を行なう

```
 x = 1 y = 0: z= 1
 x = 1 y = 1: z= 2
 x = 1 y = 2: z= 5
 x = 1 y = 3: z= 10
 x = 1 y = 4: z= 17
 x = 1 y = 5: z= 26
 x = 1 y = 6: z= 37
 x = 1 y = 7: z= 50
 x = 1 y = 8: z= 65
 x = 1 y = 9: z= 82
```


二重ループの出力結果②

①のループ中 x=9 として
②のループの処理を行ない終了する

```
 x = 9 y = 0: z= 81
 x = 9 y = 1: z= 82
 x = 9 y = 2: z= 85
 x = 9 y = 3: z= 90
 x = 9 y = 4: z= 97
 x = 9 y = 5: z= 106
 x = 9 y = 6: z= 117
 x = 9 y = 7: z= 130
 x = 9 y = 8: z= 145
 x = 9 y = 9: z= 162
```

二重ループのまとめ①

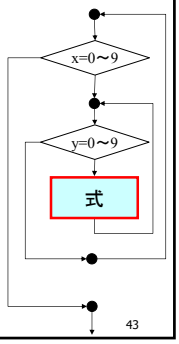
eachを用いた場合

```
(0..9).each{ |x|
   (0..9).each{ |y|
        式
   }
}
```

timesを用いた場合

```
10.times{ |x|
   10.times{ |y|
        式
   }
}
```

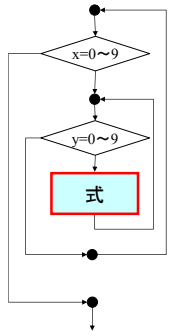
外側と内側の制御変数は異なる名前にする
（この場合は，x と y）



二重ループのまとめ②

while を用いた場合

```
x = 0
while x < 10 do
        while y < 10 do
                式
                y += 1
        end
        x +=1
end
```



二重ループの例

二重ループの例①

九九の表の表示プログラム

```
(1..9).each{ |x|
   (1..9).each{ |y|
        printf( " %d×%d=%2d" , x , y , x * y  )
   }
   print( "¥n" )
}
```



```
x=1
(1..9).each{ |y|
   printf( " %d×%d=%2d" , x , y , x * y  )
}
print( "¥n" )
```

x=1,2,…9と変わっていく

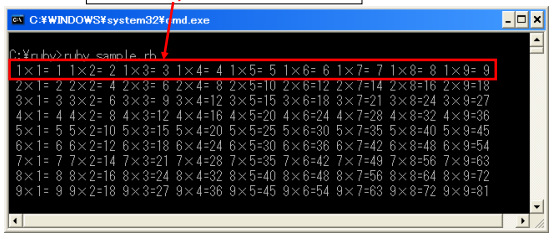
```
x=2
(1..9).each{ |y|
   printf( " %d×%d=%2d" , x , y , x * y  )
}
print( "¥n" )
```

```
x=9
(1..9).each{ |y|
   printf( " %d×%d=%2d" , x , y , x * y  )
}
print( "¥n" )
```


二重ループの例①

前頁の実行画面

xを1とし，yを1から9まで変える



二重ループの例②

対角行列の表示プログラム

```
(1..9).each{ |x|
        (1..9).each{ |y|
                if x == y then
                        print( "1 " )
                else
                        print( "0 " )
                end
        }
        print( "¥n" )
}
```

xとyの値が同じ→"1 "
異なる場合は→"0 "

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


二重ループの例②の出力結果

x=y=1の場合
x=y=2の場合
x=y=3の場合
x=y=4の場合
x=y=5の場合
x=y=6の場合
x=y=7の場合
x=y=8の場合
x=y=9の場合

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


二重ループの例③

```
(1..9).each{ |x|
   (1..9).each{ |y|
      if x == (10-y) then
              print( "1 " )
      else
              print( "0 " )
      end
   }
   print( "¥n" )
}
```

xと(10-y)の値が同じ→"1 "
異なる場合は→"0 "

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


二重ループの例③の出力結果

x=5,y=5
x=6,y=4
x=7,y=3
x=8,y=2
x=9,y=1
x=1,y=9
x=2,y=8
x=3,y=7
x=4,y=6

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


二重ループの例④

```
(1..9).each { |i|
  (1..i).each { |j|
          print( j )
  }
  print ( "¥n" )
}
```

j は 1からｉまで変わる

```
C:¥ruby>ruby sample.rb
1
12
123
1234
12345
123456
1234567
12345678
123456789
```


二重ループの例④の出力結果

```
C:¥ruby>ruby sample.rb
1
12
123
1234
12345
123456
1234567
12345678
123456789
```

i=1の時, j=1
i=1の時, j=1～2
i=1の時, j=1～3
i=1の時, j=1～8
i=1の時, j=1～9

二重ループの例⑤

```
(1..9).each { |i|
  (1..(10-i)).each{ |j|
    print( j )
  }
  print ( "¥n" )
}
```

j は 1から 10-i まで変わる

```
C:¥ruby>ruby sample.rb
123456789
12345678
1234567
123456
12345
1234
123
12
1
```


二重ループの例⑤の出力結果

```
C:¥ruby>ruby sample.rb
123456789
12345678
1234567
123456
12345
1234
123
12
1
```

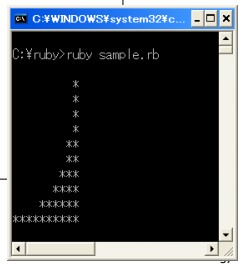
i=1の時, j=1～9
i=1の時, j=1～8
i=1の時, j=1～7
i=1の時, j=1～2
i=1の時, j=1

二重ループの例⑥

```
11.times { |i|
  d = Math.sqrt( 100 - i*i ).to_i
  (1..d).each{
    print( " " )
  }
  ((d+1)..10).each{
    print( "*" )
  }
  print( "¥n" )
}
```

d回は" "(空白)を表示

10-d回は"*"を表示



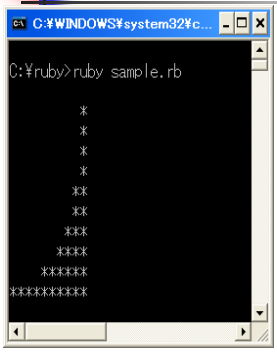
二重ループの例⑥

```
11.times { |i|
  d = Math.sqrt( 100 - i*i ).to_i
  print( d , "¥n" )
}
```

d の値はどう変わっていっているでしょうか

```
C:¥ruby>ruby sample.rb
10
9
9
9
9
8
8
7
6
4
0
```


二重ループの例⑥の出力結果



dの値

```
10
9
9
9
9
8
8
7
6
4
0
```

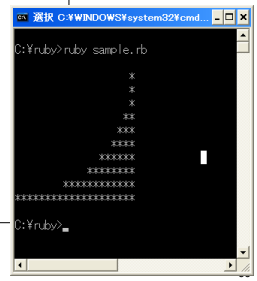
10個 " ", 0個 "*"
9個 " ", 1個 "*"
8個 " ", 2個 "*"
7個 " ", 3個 "*"
6個 " ", 4個 "*"
4個 " ", 6個 "*"
0個 " ", 10個 "*"

二重ループの例⑦
(どうしてこのような出力になるのでしょうか)

```
11.times { |i|
  d = Math.sqrt( 400 - 4*i*i ).to_i
  (1..d).each{
    print( " " )
  }
  ((d+1)..20).each{
    print( "*" )
  }
  print( "¥n" )
}
```

d回は" "(空白)を表示

20-d回は"*"を表示



二重ループの例⑦
（ヒント：dの値はどう変わっていくでしょうか）

```
11.times { |i|
  d = Math.sqrt( 400 - 4*i*i ).to_i
  print( d , "¥n" )
}
```

```
C:¥ruby>ruby sample.rb
20
19
19
19
18
17
16
14
12
8
0
```


金利の計算①

10年後までの計算

```
r = 0.05
n = 10
m = 10000
(1..n).each{ |i|
   x = 10000.0 * ( 1.0 + r ) ** i.to_f
   printf( "%2d 年後は %d¥n" , i , x.to_i )
}
```

利率

元金

1年から10年まで変化

金利の計算①（出力結果）

```
C:¥Ruby>ruby sample.rb
 1 年後は 10500
 2 年後は 11025
 3 年後は 11576
 4 年後は 12155
 5 年後は 12762
 6 年後は 13400
 7 年後は 14071
 8 年後は 14774
 9 年後は 15513
10 年後は 16288
```


金利の計算②

年，利率を変化させた場合

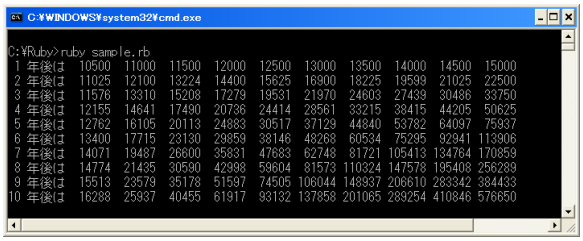
```
n = 10
m = 10000

(1..n).each{ |i|
   printf( "%2d 年後は " , i )
   (1..10).each{ |j|
        r = j * 0.05
        x = 10000.0 * ( 1.0 + r ) ** i.to_f
        printf( "%6d " , x.to_i )
   }
   print( "¥n" )
}
```

1年から10年まで変化

利率を0.05から0.5まで0.5ずつ変化

金利の計算②（出力結果）



配列の初期化①

```
sieve = Array.new(10).fill{ 1 }
```

```
irb(main):004:0> sieve = Array.new(10).fill{ 1 }
=> [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
irb(main):005:0> p sieve
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
=> nil
```

配列名=Array.new(要素数).fill{値}
配列の全要素は「値」となる

配列の初期化②

```
sieve = Array.new(10).fill{ |i| i }
```

```
irb(main):001:0> sieve = Array.new(10).fill{ |i| i }
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

短く書けます

```
sieve = Array.new(10)

sieve.length.times{ |i|
        sieve[ i ] = i
}
```


練習問題

練習問題①～④

練習問題①

- 下記のプログラムにおいて，配列 test の要素値の標準偏差を求めるプログラムを追加しなさい

```
name = [ "A" , "B" , "C" , "D" , "E" ]
test = [ 85 , 60 , 5, 100 , 50 ]
sum = 0
test.each{ |i|
   sum += i
}
average = sum / test.length
```

標準偏差

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - a)^2}{n-1}}$$

練習問題②

- 下記のプログラムにおいて，配列aには1から100の乱数が格納されます．
- j=1～8において，(a[j-1]+a[j]+a[j+1])/3を求めなさい（移動平均）

```
a = []
10.times{ |i|
        a[ i ] = rand(100)+1
        print( a[ i ] , " " )
}
print( "¥n" )
```

```
C:¥Ruby>ruby sample.rb
21 91 34 17 61 22 7 86 93 93
21 91 34 -> 48
91 34 17 -> 47
34 17 61 -> 37
17 61 22 -> 33
61 22 7 -> 30
22 7 86 -> 38
7 86 93 -> 62
86 93 93 > 90
```


練習問題②'

- 前ページの問題を二重ループを用いたプログラムで行ないなさい

練習問題③

- x, y ともに0から10までの整数とする．この場合，

① xとyの和が10となる組み合わせ

② x^2とy^2の和が100となる組み合わせ

を二重ループを用いて求めなさい

練習問題④

- 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

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


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。