プログラミング言語　第六回

担当：篠沢 佳久
櫻井 彰人

平成21年 5月25日

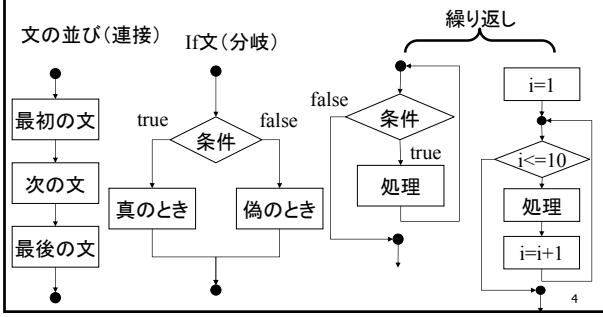
本日の内容

- 繰り返し（2）
- times, each
- 繰り返しについての練習問題
- 第二回レポート課題

前回の復習

制御構造
繰り返し（1）

制御構造（復習）



無限の繰り返し

```
loop{
        式
}
```

式が永久に実行される
停止するために break を用いる

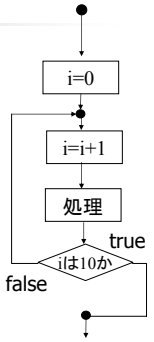
```
loop{
        式
        break 条件式
}
次の式
```

条件式を満たした場合のみ停止する（loopブロックの次の式を実行する）

無限の繰り返し①

10回「こんにちは」を表示するプログラム

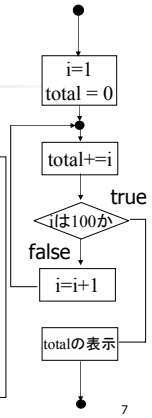
```
i = 0
loop{
   i = i+1
   print( i , "回目のこんにちは¥n" )
   break  if i == 10
}
```



無限の繰り返し②

100以下の整数の和を求めるプログラム

```
i = 1
total = 0
loop{
   total += i
   break  if i == 100
   i = i+1
}
print( "合計は" , total )
```



回数の決まった繰り返し

times
each

times①

- 同じ処理をn回繰り返したい
- n.times

```
n.times {
    式
}
```

式をn回繰り返す

回数がわかっている繰り返し①

- 10.times

```
10.times {print "やっほ～ "; puts "      Yee-ha! " }
```

```
10.times {
  print( "やっほ～ " )
  puts( "      Yee-ha! " )
}
```

停止条件は書かなくてもよい
n.times で指定されたn回，式を繰り返す

loop{}で書く場合には

```
10.times {
  print( "こんにちは¥n" )
}
```

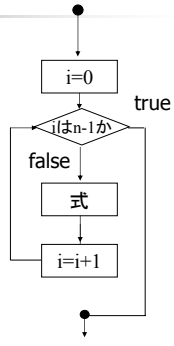
```
i = 0
loop{
  i = i+1
  print( "こんにちは¥n" )
  break  if i == 10
}
```

```
C:¥Ruby>ruby sample.rb
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
こんにちは
```


times②

```
n.times{ |i|
  式
}
```

n回式を繰り返す
iには自動的に0からn-1が代入される



回数がわかっている繰り返し②

```
10.times { |i|
  print( i , "回目のこんにちは¥n" )
}
```

```
C:¥ruby>ruby sample.rb
0回目のこんにちは
1回目のこんにちは
2回目のこんにちは
3回目のこんにちは
4回目のこんにちは
5回目のこんにちは
6回目のこんにちは
7回目のこんにちは
8回目のこんにちは
9回目のこんにちは
```

変数iには0から代入されていく

変数iには9まで代入されていく

回数がわかっている繰り返し②'

0から9までの合計を求めるプログラム

```
total = 0
10.times { |i|
  total += i
}
print( "合計は" , total )
```

変数iは0から9まで1づつ加算

```
C:¥Ruby>ruby sample.rb
合計は45
```


回数がわかっている繰り返し②''

1から10までの合計を求めるには？

```
total = 0
10.times { |i|
  total += i+1
}
print( "合計は" , total )
```

```
C:¥Ruby>ruby sample.rb
合計は55
```


回数がわかっている繰り返し③

10のべき乗を表示するプログラム

```
10.times{ |i|
  print( 10 ** i , "¥n" )
}
```

i は0から9まで1づつ加算

```
C:¥Ruby>ruby sample.rb
1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
```

10^0

10^9

times と loop の関係

10のべき乗を表示するプログラム

```
10.times{ |i|
  print( 10 ** i , "¥n" )
}
```

```
i = 0
loop{
  print( 10 ** i , "¥n" )
  break  if i == 10
  i = i+1
}
```


回数がわかっている繰り返し④

```
10.times {|i| print i; puts "番目"}
10.times {|i| puts( "#{i} 番目" ) }
10.times {|j| puts "*"*j}
```

```
0番目
1番目
2番目
3番目
4番目
5番目
6番目
7番目
8番目
9番目
```

```
*
**
***
****
*****
******
*******
********
*********
```

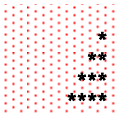
iやj=0,1,2,3,4,5,6,7,8,9 の順で、繰り返し、式の計算をする

回数が分っている繰り返し⑤

- なぜ，このような出力になるでしょうか

```
n=5
n.times {|k| print( " "*(10-k), "*"*k, "¥n" ) }
```

出力結果



k	10-k
0	10
1	9
2	8
3	7
4	6

回数が分っている繰り返し⑥

- どのような出力になるでしょうか

```
n=10
n.times {|k| print( " "*(10-k), "*", " "*(9-k), "¥n" ) }
```


each①

```
n.times{ |i|
  式
}
```

n回式を繰り返す
iには自動的に0からn-1が代入される

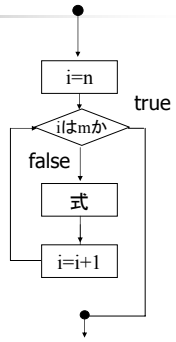
```
(n..m).each{ |i|
  式
}
```

m-n回式を繰り返す
iには自動的にnからmが代入される

each②

```
(n..m).each{ |i|
  式
}
```

i には自動的にnからmが代入される
その結果、式は m-n回繰り返される



変数範囲が決まっている繰り返し①

10のべき乗を表示するプログラム

```
(0..9).each{ |i|
  print( 10 ** i , "¥n" )
}
```

```
10.times{ |i|
  print( 10 ** i , "¥n" )
}
```


変数範囲が決まっている繰り返し②

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

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

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

5 は奇数
6 は偶数
7 は奇数
8 は偶数
9 は奇数
10 は偶数

変数範囲が決まっている繰り返し③

10から100までの合計値を求めるプログラム

```
total=0
(10..100).each { |i|
   total += i
}
print( total )
```

nからmまでの合計値を求めるプログラム

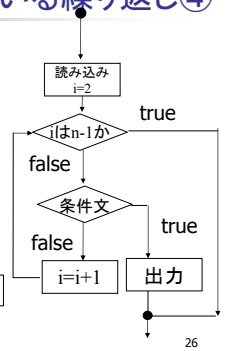
```
n=gets.chomp.to_i
m=gets.chomp.to_i
total=0
(n..m).each { |i|
   total += i
}
print( n , "から" , m , "までの合計は" , total )
```


変数範囲が決まっている繰り返し④

何を調べているプログラムでしょう

```
n=gets.chomp.to_i
(2..n-1).each { |i|
   if n % i == 0 then
      print( "〇〇ではありません" )
      break
   end
}
```

break でループを抜け出ることも可能



変数範囲が決まっている繰り返し⑤

```
(-10..10).each{ |i|
   print( i , " " , i**2 , "¥n" )
}
```

負の場合は？

```
(10..-10).each{ |i|
   print( i , " " , i**2 , "¥n" )
}
```

(n..m).each
n>m の場合は？

刻み幅に小数値を使用したい場合①

0から1まで0.1刻みで二乗の計算を行なう

i は0から10まで1刻みの整数値

```
11.times{ |i|
   print( i / 10.0 , " " , (i/10.0)**2.0 , "¥n" )
}
```

10.0で割る

刻み幅に小数値を使用したい場合②

```
(0..10).each{ |i|
   print( i / 10.0 , " " , (i/10.0)**2.0 , "¥n" )
}
```

```
(0..100).each{ |i|
   x = i.to_f / 100
   print( x, " " ,  x**2.0 , "¥n" )
}
```


擬似乱数①

- コンピュータが計算して作り出す乱数。本当の乱数ではないが、かなり本物に近い。
- ruby には組み込み関数として、rand() がある
  - rand(n) とすると 0以上n未満の整数値が一様ランダムに生成される

```
irb(main):090:0> rand()
=> 0.0422245532019152
irb(main):091:0> rand(10)
=> 7
irb(main):092:0> rand(100000)
=> 16339
```

擬似乱数②

```
30.times {
  if rand() > 0.5 then
    print( "1" )
  else
    print( "0" )
  end
}
```

```
30.times { print rand(2) }
```

実行する度に結果は異なります

```
11111100100111000111011000011O=> 30
```

```
30.times {
  print( if rand() > 0.5 then "1" else "0" end )
}
```


擬似乱数③

```
s = 0
loop{
  s = rand(6)+1
  break if s == 6
  print( s , "¥n" )
}
```

1から6の整数を生成

どういう乱数でしょうか

(rand(10)+1)*100

(rand(10)+1)/10.0

(rand(5)+1)/10.0+0.5

疑似乱数を用いた例①

```
total = 0
5.times{
  x = rand(100)
  print( x , "¥n" )
  total += x
}
print( " 合計は" , total )
```

0から99の整数を生成

```
C:¥Ruby>ruby sample.rb
72
29
88
4
98
 合計は291
```


疑似乱数を用いた例②

```
max = 0
5.times{
  x = rand(100)
  print( x , "¥n" )
  if max < x then
    max = x
  end
}
print( " 最大値は" , max )
```

最大値を求めるプログラム

```
C:¥Ruby>ruby sample.rb
90
88
38
79
68
 最大値は90
```


疑似乱数を用いた例③

```
min = 999
5.times{
  x = rand(100)
  print( x , "¥n" )
  if min > x then
    min = x
  end
}
print( " 最小値は" , min )
```

最小値を求めるプログラム

```
C:¥Ruby>ruby sample.rb
62
66
63
14
77
 最小値は14
```


疑似乱数を用いた例④

```
a = 0
b = 0
10000.times{
  x = rand(2)
  if x == 0 then
    a += 1
  else
    b += 1
  end
}
print( " 0の出現回数 " , a , "回¥n" )
print( " 1の出現回数 " , b , "回¥n" )
```

乱数が0の場合，aに1を加算
乱数が1の場合，bに1を加算

```
C:¥Ruby>ruby sample.rb
 0の出現回数 4927回
 1の出現回数 5073回
```

練習問題

練習①

1から100までの整数のうち偶数のみを印字するプログラムを times を用いて書いて下さい

ヒント1: まず、1から100までの整数を印字するプログラムを書いてみましょう。右の通りです

```
100.times{ |i|
    print( i+1 , "¥n" )
}
```

ヒント2: では、「偶数のみ」にはどう対応しますか？右のようにします。

```
100.times{ |i|
    if   ????    then
        print( i+1 , "¥n" )
    end
}
```


times と loop の関係（練習①）

下記と同様な機能を持つプログラムをtimes を用いて書きなさい

```
i = 1
loop{
   if  i % 2 == 0 then
       print( i , "¥n" )
   end
   break  if i == 100
   i = i+1
}
```

1から100までの整数で偶数を調べる

練習②

1から100までの整数の自乗和を印字するプログラムを times を用いて書いて下さい

ヒント1: まず、1から100までの整数の自乗を印字するプログラムを書いてみましょう。右の通りです

```
100.times{ |i|
    print( (i+1)**2, "¥n" )
}
```

ヒント2: では、「合計」はどう計算しますか？右のようにします。

```
?? = 0
100.times{ |i|
    total = ?? + ???
      # または total += ???
}
print( ????? )
```


times と loop の関係（練習②）

下記と同様な機能を持つプログラムをtimes を用いて書きなさい

```
i = 1
total = 0
loop{
   total += i**2
   break  if i == 100
   i = i+1
}
print( "自乗和は" , total )
```

1から100までの整数の自乗和を求める

練習③

下記と同様な機能を持つプログラムをeach を用いて書きなさい

```
total = 0
10.times { |i|
   total += (i+5)
}
print( total + "¥n" )
```

```
total = 1
10.times { |i|
   total *= (i+1)
}
print( total + "¥n" )
```

練習④

- ーnからnまで0.1n刻みで cos の値を求めるプログラムを each を用いて書きなさい。

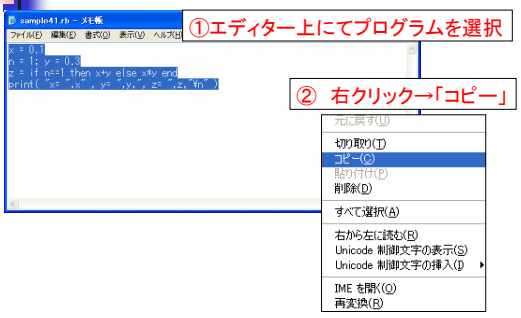
練習問題⑤

- 二つの整数mとn(m>0, n>0)を読み込みmのn乗を求めるプログラム times および loop を用いて書きなさい
- m**n として求めてはいけません
- mをn回かけることによって求めて下さい

練習問題

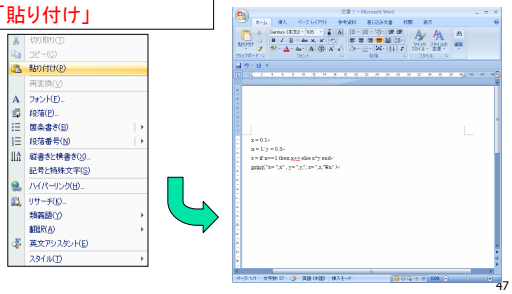
- 練習問題①から⑤をできるだけ行ないなさい。
- プログラムと実行結果をワープロに貼り付けて、keio.jp から提出して下さい。

プログラムと実行結果をMS-Wordへ貼り付ける①

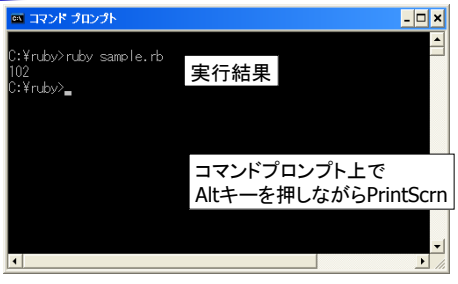


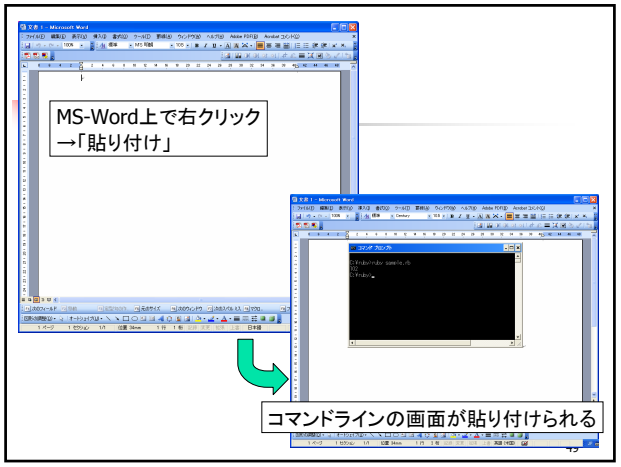
プログラムと実行結果をMS-Wordへ貼り付ける②

③ MS-Word上で右クリック→「貼り付け」



プログラムと実行結果をMS-Wordへ貼り付ける②




MS-Word上で右クリック
→「貼り付け」
コマンドラインの画面が貼り付けられる