プログラミング言語　第七回

担当：篠沢 佳久
櫻井 彰人

平成29年 5月22日

本日の内容

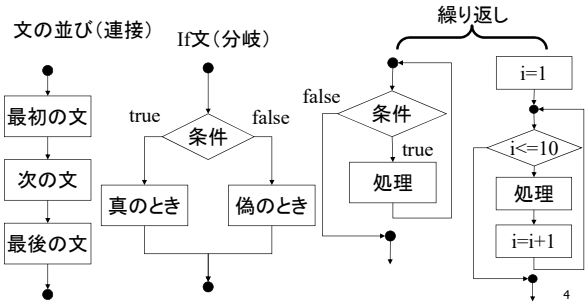
- 繰り返し(3)
- While
- 疑似乱数
- 練習問題

- 他のループ制御
- downto, upto, step, for
  - 参考ですが，閲覧しておいて下さい

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

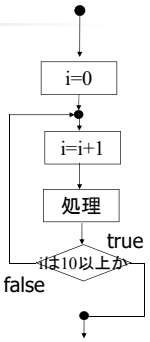
```
loop{
        式
        break 条件式
}
次の式
```

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

10回「こんにちは」を表示するプログラム

```
# coding: Windows-31J
i = 0
loop{
  i = i+1
  print( i , "回目のこんにちは¥n" )
  break  if i == 10
}
```



回数の決まった繰り返し

times
each

7

times①

- 同じ処理をn回繰り返したい
- n.times

```
n.times {
    式
}
```

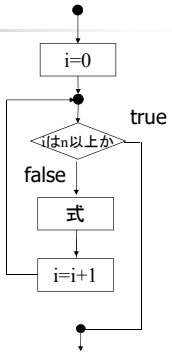
式をn回繰り返す

8

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

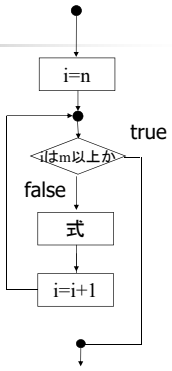


9

each

```
(n..m).times{ |i|
   式
}
```

(m-n+1)回式を繰り返す
iには自動的にnからmが代入される



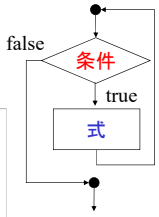
10

whileによる繰り返し

11

whileによる繰り返し

```
while 条件 do
      式
end
```



条件がtrueである場合は式を実行し，false の場合は式を実行しない

すなわち条件がtrueである限りは式を繰り返し実行する

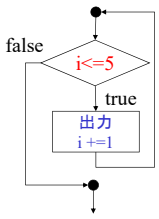
do は省略することもできます

12

while による繰り返し（例①）

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts( i.to_s + "回目です。" )
  i += 1
end
```

i = i+1




```
1回目です。
2回目です。
3回目です。
4回目です。
5回目です。
```

13

while による繰り返し（例①’）

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts( i.to_s + "回目です。" )
  i += 1
end
```

① 初期値 i=1
② 条件判定 -> true
③ puts による出力
④ i に1を加算(i=2)
⑤ 条件判定 -> true
⑥ putsによる出力

```
1回目です。
2回目です。
3回目です。
4回目です。
5回目です。
```

⑦ i に1を加算(i=3)
⑧ 条件判定 -> true
⑨ putsによる出力
⑩ i に1を加算(i=4)
⑪ 条件判定 -> true
⑫ putsによる出力

14

while による繰り返し（例①’）

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts( i.to_s + "回目です。" )
  i += 1
end
```

⑬ i に1を加算(i=5)
⑭ 条件判定 -> true
⑮ putsによる出力
⑯ i に1を加算(i=6)
⑰ 条件判定 -> false
⑱ 終了

```
1回目です。
2回目です。
3回目です。
4回目です。
5回目です。
```

15

前のページと同じプログラム（例①''）

puts の書き方

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts i.to_s + "回目です。"
  i += 1
end
```

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts "#{i}回目です。"
  i += 1
end
```

```
1回目です。
2回目です。
3回目です。
4回目です。
5回目です。
```

16

これまで学んだ方法で書くと①

```
# coding: Windows-31J
5.times{ |i|
    puts( (i+1).to_s + "回目です。" )
}
```

```
# coding: Windows-31J
(1..5).each{ |i|
    puts( i.to_s + "回目です。" )
}
```

17

これまで学んだ方法で書くと②

```
# coding: Windows-31J
i = 1
loop {
  puts( i.to_s + "回目です。" )
  break if i >= 5
  i += 1
}
```

18

while による繰り返し（例②）

```
total = 0
i = 1
while i <= 10 do
        total += i
        i += 1
end
puts( total )
```

i=1 から開始

i<=10 の場合 do 以下を実行

i=11 となった場合，停止

i に1を加算

```
Z:¥Ruby>ruby sample.rb
55
```


while による繰り返し（例②'）

```
total = 0
i = 1
while total <= 100 do
        total += i
        i += 1
end
puts( i , " " , total )
```

i=1 から開始

total<=100 の場合 do 以下を実行

total > 100 となった場合，停止

i に1を加算

```
Z:¥Ruby>ruby sample.rb
15 105
```


while による繰り返しの例

- while による繰り返し（例③~⑥）の出力結果を考えなさい
- 実行する前に結果を予想して下さい

while による繰り返し（例③）

どのような出力結果になるでしょうか

```
i = 10
while i != 0 do
   puts( i )
   i -= 1
end
```

```
i = 10
while i > 0 do
   puts( i )
   i -= 1
end
```

i = i-1

while による繰り返し（例④）

どのような出力結果になるでしょうか

```
i = 0
while i < 10 do
   puts( i * 2 )
   i += 1
end
```

```
i = 0
while i <=10 do
   puts( i * 2 )
   i += 1
end
```


while による繰り返し（例⑤）

どのような出力結果になるでしょうか

```
i = 1000
while i > 0 do
   puts( i )
   i = i / 2
end
```

```
i = 2
while i < 100000 do
   puts( i )
   i = i * 2
end
```

while による繰り返し（例⑥）

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts( i.to_s + "回目です。" )
end
```

実行させるとどうなるでしょうか？

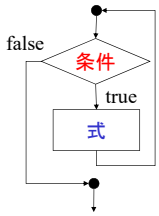
while による繰り返し（例⑥）

```
# coding: Windows-31J
i = 1
while i <= 5 do
  puts( i.to_s + "回目です。" )
end
```

i の値は1から変化しない

そのため条件式は常にtrue であるため，puts が永久に実行される

while による繰り返しにおける注意



条件がtrueである限り式は実行され続ける．そのため，条件がfalseとなり終了するように条件を設定しなければならない

times（each）と while の違い

- 同じである．ただし，コンセプトには結構な違いがある
  - times, each: 各回には，制御変数の値の違いしかない
  - while, loop: 各回には，制御変数以外の変数で違いがある．または，制御変数を自分で作ってあげないといけない

times（each）と while の違い（続）

- While, loop を使うのは
  - ファイルの読み込み（後述）のように，読んでみないとわからない場合
    - 次のテーマ
  - むずかし〜い計算なので，計算してみないとわからない場合
    - Collatz-角谷の予想（練習問題⑤）
    - 素数を求める
    - 次回でてくる配列を用いる計算の中にある
  - 前の行為の結果が，影響を及ぼす場合

例: times と while の違い①

"繰り返し回数"が分っている場合

```
n =10
n.times{ |i|
  puts( i )
}
```

⇔

```
i=0
while i<10 do
        puts( i )
        i = i + 1
end
```

制御変数

制御変数を自分で指定しなければならない

例: times と while の違い②

"繰り返し回数"が分っている場合

制御変数

```
n =10
total = 0
n.times{ |i|
  total += i
}
puts( total )
```

⇔

```
i=0
total = 0
while i<10 do
        total += i
        i = i + 1
end
puts( total )
```

31

例: times と while の違い③

```
i = 1000
while i > 0 do
  puts( i )
  i = i / 2
end
```

```
i = 2
while i < 100000 do
  puts( i )
  i = i * 2
end
```

"停止条件"が分っているが"繰り返し回数"は分らない

→ 変数 i の値はループ内の処理によって変わっていく

→ times では書きづらい

32

例: times と while の違い④

偶数を10個求めるプログラム

```
i = 1
count = 0
while count < 10 do
  if i % 2 == 0 then
      print( i , "¥n" )
      count += 1
  end
  i += 1
end
```

```
Z:¥Ruby>ruby sample.rb
2
4
6
8
10
12
14
16
18
20
```

停止条件は「10個偶数が求まった時」
→実際に何回ループを回せばよいかは分からない

33

例: times と while の違い⑤

偶数を10個求めるプログラム

```
count = 0
while count < 10 do
  x = rand( 100 )
  if x % 2 == 0 then
      print( x , "¥n" )
      count += 1
  end
end
```

rand(n)
0以上n未満の整数を生成

```
Z:¥Ruby>ruby sample.rb
22
60
34
38
88
30
56
62
94
34
```

停止条件は「10個偶数が求まった時」
→実際に何回ループを回せばよいかは分からない

34

例: times と while の違い⑥

偶数を10個求めるプログラム

```
i = 1
count = 0
while count < 10 do
  if i % 2 == 0 then
      print( i , "¥n" )
      count += 1
  end
  i += 1
end
```

```
i = 1
count = 0
while count != 10 do
  if i % 2 == 0 then
      print( i , "¥n" )
      count += 1
  end
  i += 1
end
```

動作は同じであるが，停止条件が一意なため，間違えて無限ループのプログラムとして書いてしまいやすいので注意

35

例: times と while の違い⑥

"繰り返し回数"は予め分からない

```
# coding: Windows-31J
s = 0.0
n =10
n.times{ |i|
  r = rand()
  s  += r
}
a  = s/n
puts( "平均= #{a}")
```

"繰り返し回数" が予めわかっている

```
# coding: Windows-31J
s = 0.0
n = 0
while s<10.0 do
  r = rand()
  s  += r
  n  += 1
end
puts( "#{n} 回目で10を越えた" )
```

rand()
0以上1未満の乱数を生成

"停止条件" が予めわかっている

36

擬似乱数

- 平均、不偏分散、不偏標準偏差を求めてみよう

```
# coding: Windows-31J
s = 0.0
s2= 0.0
n =10
n.times{ |i|
  r = rand()
  s  += r
  s2 += r*r
  puts( "#{r} は #{i} 番目の乱数です." )
}
a  = s/n
v  = (s2 - a*a*n)/(n-1)
sd = Math.sqrt( v )
puts( "平均= #{a}, 分散= #{v}, 標準偏差= #{sd}" )
```

rand()
0以上1未満の乱数を生成

擬似乱数

実行結果

```
Z:¥Ruby>ruby sample.rb
0.9342183253647653 は 0 番目の乱数です.
0.17464151855331933 は 1 番目の乱数です.
0.042984728731218724 は 2 番目の乱数です.
0.783219542598958 は 3 番目の乱数です.
0.9851461205355044 は 4 番目の乱数です.
0.043461396973162536 は 5 番目の乱数です.
0.4956017448990262 は 6 番目の乱数です.
0.9952302150843787 は 7 番目の乱数です.
0.28483068587097293 は 8 番目の乱数です.
0.9527685456573891 は 9 番目の乱数です.
平均= 0.5692102824268696, 分散= 0.1639936761847672, 標準偏差= 0.4049613267767272
```

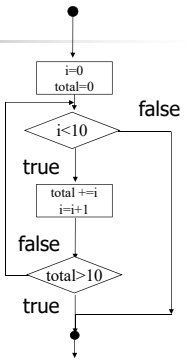

break文①

- 繰り返し(ループ)の中で使用
- そのbreakが所属するループを1つ抜ける

break文②

```
i = 0
total = 0
while i < 10 do
   total += i
  i += 1
   break if total > 10
end
print( i , " " , total )
```

```
Z:¥Ruby>ruby sample.rb
6 15
```



break文③

true となっているため，無限に実行される

```
i=0
loop{
   puts( i )
   i=i+1
   break if i > 10
}
```

```
i=0
while true do
   puts( i )
   i=i+1
   break if i > 10
end
```

条件式をtrueとすることによって無限ループとなる

無限ループ

true となっているため，無限に実行される

```
# coding: Windows-31J
while true do
      puts( "こんにちは" )
end
```

```
Z:¥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
こんにちは
こんにちは
こんにちは
```

Ctrlを押しながらcで停止

break文④

```
i = 1000
while i > 0 do
   puts( i )
   i = i / 2
end
```

二つのプログラムとも同じ動作をします

```
i = 1000
while true do
   puts( i )
   i = i / 2
   break if i <= 0
end
```

```
Z:¥Ruby>ruby sample.rb
1000
500
250
125
62
31
15
7
3
1
```


break文④

```
i = 1000
while i > 0 do
   puts( i )
   i = i / 2
end
```

ループの停止条件
→ i > 0 を満たさない時

```
i = 1000
while true do
   puts( i )
   i = i / 2
   break if i <= 0
end
```

ループの停止条件
→ なし（無限ループ）

break文によってループから抜ける

break文⑤

```
10.times{ |i|
   puts( i )
   break if i > 5
}
```

```
(10..100).each{ |i|
   puts( i )
   break if i > 20
}
```

times，eachにおいてもbreakでループを抜け出ることができる

next

- nextはもっとも内側のループの次の繰り返しにジャンプします
- while であれば，「継続条件」の判定の直前が再開場所です

プログラム例（each と next）

```
# coding: Windows-31J
(0..5).each { |i|
   if ( i==1 || i==4 ) then
      next
   end
   puts( "iは #{i}" )
}
```

「または」です

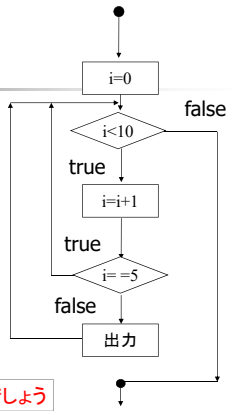
出力結果

```
iは 0
iは 2
iは 3
iは 5
```

i が1または4の場合
→ next によってループの先頭に戻る
→ puts文は実行されない

next②

```
i=0
while i < 10 do
   i += 1
   if i == 5  then
      next
   end
   puts( i )
end
```



出力はどうなるでしょう

next②

```
i=0
while i < 10 do
  i += 1
  if i == 5  then
      next
  end
  puts( i )
end
```

i=5の場合，puts文は実行されない

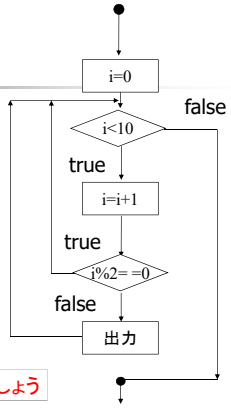
```
Z:¥Ruby>ruby sample.rb
1
2
3
4
6
7
8
9
10
```

49

next③

```
i=0
while i < 10 do
  i += 1
  if i % 2 == 0 then
      next
  end
  puts( i )
end
```

出力はどうなるでしょう



50

next③

```
i=0
while i < 10 do
  i += 1
  if i % 2 == 0 then
      next
  end
  puts( i )
end
```

iが偶数の場合，puts文は実行されない

```
Z:¥Ruby>ruby sample.rb
1
3
5
7
9
```

51

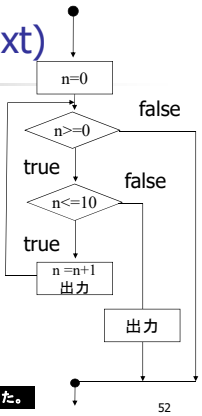
プログラム例(break,next)

ループの先頭であるwhileに戻る

```
#coding: Windows-31J
n = 0
while n>=0 do
  if n<=10 then
    print( n, " ")
    n += 1
    next
  else
    print( "変数 n は10を越えました。" )
    break
  end
end
```

ループwhileを抜ける

```
0 1 2 3 4 5 6 7 8 9 10 変数 n は10を越えました。
```



52

繰り返しのまとめ

times, each
loop, while

53

繰り返しのまとめ①

- 繰り返し回数が分かっている場合
  - times , each
- 繰り返し回数が分かっていない場合
  - loop, while
- ただし，同じ動作をするにしても，いろいろな書き方があります

54

繰り返しのまとめ②

同じ動作をするプログラム

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

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

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

```
i = 0
while i < 10 do
   print( i , "¥n" )
   i += 1
end
```

```
Z:¥Ruby>ruby sample.rb
0
1
2
3
4
5
6
7
8
9
```

55

繰り返しのまとめ③

同じ動作をするプログラム

```
i = 2
while i < 100 do
        print( i , "¥n" )
        i = i*2
end
```

```
i = 1
loop{
        i = i*2
        break if i > 100
        print( i , "¥n" )
}
```

```
i = 1
while true do
        i = i*2
        break if i > 100
        print( i , "¥n" )
end
```

```
Z:¥Ruby>ruby sample.rb
2
4
8
16
32
64
```

56

疑似乱数

疑似乱数のプログラム例

57

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
```

0～1の値（1.0は含まない）

0～9の整数値

0～99999の整数値

58

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

```
111111001001110001101100001 0=> 30
```

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

59

擬似乱数③

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

6の場合，停止

rand(6)によって0から5の整数
rand(6)+1によって1から6の整数を生成

```
Z:¥Ruby>ruby sample.rb
4
1
5
4
5
4
5
6
```

60

擬似乱数④

```
5.times{
  s = rand(21)-10
  print( s , "¥n" )
}
```

rand(21)によって0から20の整数
rand(20)-10によって-10から10の整数を生成

```
Z:¥Ruby>ruby sample.rb
-3
0
-6
-3
9

Z:¥Ruby>ruby sample.rb
4
9
10
-1
2
```

61

擬似乱数⑤

```
5.times{
  s = rand(10)/10.0
  print( s , "¥n" )
}
```

rand(10)によって0から9の整数
rand(10)/10.0によって0.0から0.9まで0.1刻みの小数を生成

```
Z:¥Ruby>ruby sample.rb
0.5
0.0
0.8
0.5
0.2
```

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

62

疑似乱数を用いた例①

```
# coding: Windows-31J
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

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

63

疑似乱数を用いた例②

```
# coding: Windows-31J
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

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

64

疑似乱数を用いた例③

```
# coding: Windows-31J
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

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

65

疑似乱数を用いた例④

```
# coding: Windows-31J
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

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

66

疑似乱数を用いた例⑤

```
# coding: Windows-31J
ans=rand(100)
loop{
        print( "数字を入力して下さい¥n" )
        input = gets.chomp.to_i
        if ans == input then
                print( "正解¥n" )
                break
        elsif input > ans then
                print( "正解はその値よりも小さい¥n" )
        else
                print( "正解はその値よりも大きい¥n" )
        end
}
```

0から99の整数を生成

整数の入力

条件式1

条件式2

条件式3

疑似乱数を用いた例⑤

```
C:¥Ruby>ruby sample.rb
数字を入力して下さい
12
正解はその値よりも大きい
数字を入力して下さい
20
正解はその値よりも大きい
数字を入力して下さい
30
正解はその値よりも大きい
数字を入力して下さい
40
正解はその値よりも大きい
数字を入力して下さい
```

条件式3

```
80
正解はその値よりも小さい
数字を入力して下さい
75
正解はその値よりも大きい
数字を入力して下さい
76
正解はその値よりも大きい
数字を入力して下さい
77
正解はその値よりも大きい
数字を入力して下さい
78
正解はその値よりも大きい
数字を入力して下さい
79
正解
```

条件式2

条件式1

練習問題

練習問題

- 練習問題①から④を行ないなさい.
- （簡単な人は⑤も行ないなさい）
- 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.

練習問題①

- 下記と同じ動作をするプログラムをloopとwhileを用いて書きなさい

```
# coding: Windows-31J

sum = 0
(3..7).each{ |i|
        sum += i*i
}
puts( sum )
```

```
Z:¥Ruby>ruby sample.rb
135
```


練習問題②

整数n（n>1）をキーボードより入力し，nの階乗を印字するプログラムをwhile文を用いて書きなさい

```
Z:¥Ruby>ruby sample.rb
10
3628800
```

練習問題③

- 1以上100以下の整数に対し，それが3の倍数でも10の倍数でもないときに，印字するプログラムを書きなさい．ただし，next を用いてください

```
Z:¥Ruby>ruby sample.rb
1
2
4
5
7
8
11
13
14
16
17
19
22
23
```

73

練習問題④

- サイコロを1000回ふった際，サイコロの目の出現回数を求めるプログラムを書きなさい．
- （60ページ，66ページを参考）

```
Z:¥Ruby>ruby sample.rb
1の出現回数 171
2の出現回数 171
3の出現回数 185
4の出現回数 149
5の出現回数 149
6の出現回数 174
```

74

練習問題⑤

- **Collatz-角谷の予想**
- 自然数nを選び，
  - 奇数ならば，3倍して1をたす。
  - 偶数ならば，2で割る。

これを繰り返すと，どんなnを選んでも，いつかは，1になる

```
3, 10, 5, 16, 8, 4, 2, 1
4, 2, 1
5, 16, 8, 4, 2, 1
6, 3, 10, 5, 16, 8, 4, 2, 1
7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
8, 4, 2, 1
9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16,
8, 4, 2, 1
10, 5, 16, 8, 4, 2, 1
11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

75

練習問題⑤

- 整数nをキーボードから読み込み，前頁の性質を確認するプログラムをwhile文を用いて作成しなさい．

```
Z:¥Ruby>ruby sample.rb
 n > 12
12
6
3
10
5
16
8
4
2
1
```

76

（参考）他のループ制御

downto, upto, step, for

77

他のループ制御

- Ruby では他に、until, for などがある。
  - 拘る人へ: 正確には
    - Ruby の繰返制御構造は、while, until, for である
    - times, upto, downto, step は Integer のメソッド
    - each は Array 等の、Enumerable をインクルードするクラスのメソッド
- ループの中断には、break以外、next, redo, retry がある

78

downto, upto, and step

- 例で学ぼう

```
7.downto(3) { |i| print( i," " ) }
```
7 6 5 4 3

```
3.upto(7) { |i| print( i," " ) }
```
3 4 5 6 7

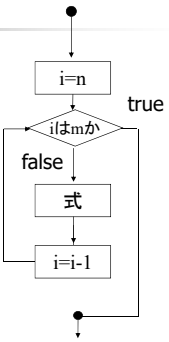
```
2.step(12,3) { |i| print( i," " ) }
```
2 5 8 11

```
12.step(2,-3) { |i| print( i," " ) }
```
12 9 6 3

downto①

```
n.downto(m){ |i|
    式
}
```

n-m+1回式を繰り返す(n>m)
iには自動的にnからmが代入される




```
10.downto(0){ |x|
  printf( "x=%2d x^2=%3d¥n" , x , x**2 )
}
```

%2d
整数を二桁で表示する

x は10から0まで1刻みで変化

```
Z:¥ruby>ruby sample.rb
x=10 x^2=100
x= 9 x^2= 81
x= 8 x^2= 64
x= 7 x^2= 49
x= 6 x^2= 36
x= 5 x^2= 25
x= 4 x^2= 16
x= 3 x^2=  9
x= 2 x^2=  4
x= 1 x^2=  1
x= 0 x^2=  0
```



```
0.downto(-10){ |x|
  printf( "x=%3d x^2=%6d¥n" , x , x**2 )
}
```

x は0から-10まで1刻みで変化

```
Z:¥ruby>ruby sample.rb
x=  0 x^2=     0
x= -1 x^2=    -1
x= -2 x^2=    -8
x= -3 x^2=   -27
x= -4 x^2=   -64
x= -5 x^2=  -125
x= -6 x^2=  -216
x= -7 x^2=  -343
x= -8 x^2=  -512
x= -9 x^2=  -729
x=-10 x^2= -1000
```


downto②

```
total = 0
10.downto(0){ |i|
  total += i
  break if total > 20
}
```

while で書いた場合

```
i = 10
total = 0
while i >= 0 do
  total += i
  break if total > 20
  i -= 1
end
```


downto③

downto と while を利用して書き直してみて下さい

```
total = 1
1.upto(10){ |x|
    total *= x
}
print( total )
```

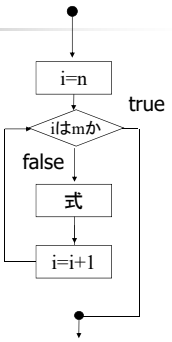
total=
1×2×3×・・・×10

```
Z:¥Ruby>ruby sample.rb
3628800
```

upto①

```
n.upto(m){ |i|
      式
}
```

m-n+1回式を繰り返す（n<m）
iには自動的にnからmが代入される



85

upto②

```
n.upto(m){ |i|
      式
}
```

⟺

```
(n..m).each { |i|
      式
}
```

基本的には each と同じ

86

upto③

```
0.upto(10){ |x|
  print( "x= " , x , "¥n" )
}
```

each で書いた場合

```
(0..10).each{ |x|
  print( "x= " , x , "¥n" )
}
```

```
Z:¥Ruby>ruby sample.rb
x= 0
x= 1
x= 2
x= 3
x= 4
x= 5
x= 6
x= 7
x= 8
x= 9
x= 10
```

87

```
total = 0
0.upto(10){ |x|
  total += x
}
print( total )
```

```
total = 0
10.downto(0){ |x|
  total += x
}
print( total )
```

```
total = 0
11.times{ |x|
  total += x
}
print( total )
```

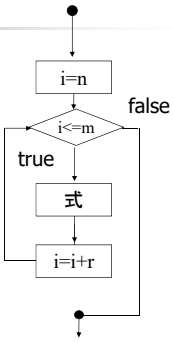
```
total = 0 ; i=0
while i <= 10 do
  total += x
  i += 1
}
print( total )
```

88

step①

```
n.step(m,r){ |i|
      式
}
```

nからmまで刻み幅はrで繰り返す
i には n, n+r, n+2*r ,... が入る



89

step②

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

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

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

```
Z:¥ruby>ruby sample.rb
1
3
5
7
9
```

90

step③

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

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

```
9.step(1,-2){ |x|
  print( x , "¥n" )
}
```

```
Z:¥ruby>ruby sample.rb
9
7
5
3
1
```

91

step④

whileを用いた場合

```
total = 0
1.step(50,2){ |i|
  total += i
}
print( total )
```

```
Z:¥Ruby>ruby sample.rb
625
```

```
total = 0
x = 1
while x <= 50 do
      total += x
      x += 2
end
print( total )
```

92

step⑤

whileを用いた場合

```
total = 0
49.step(1,-2){ |i|
  total += i
}
print( total )
```

```
Z:¥Ruby>ruby sample.rb
625
```

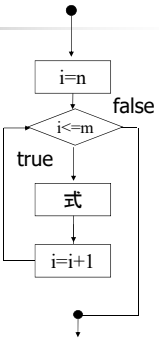
```
total = 0
x = 49
while x >= 1 do
      total += x
      x -= 2
end
print( total )
```

93

forループ①

```
for i in n..m do
  式
end
```

```
(n..m).each{ |i|
  式
}
```



94

forループ②

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

x=1,2,･･･,9,10

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

95

forループ③

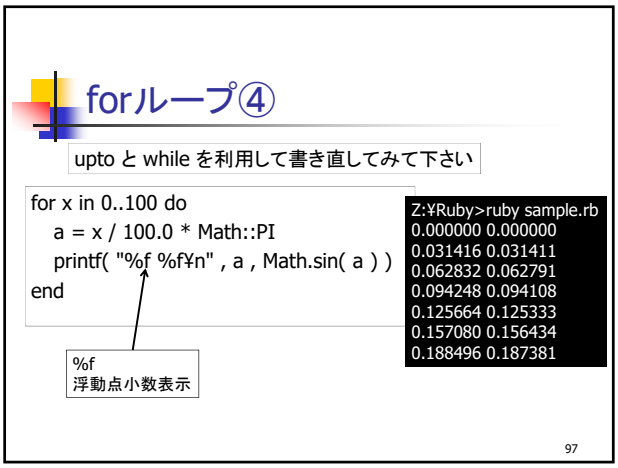
```
for x in -10..10 do
  puts( 2**x )
end
```

x=-10,-9,･･･,9,10

```
total = 0
for x in 1..10 do
  total += x
end
print( total )
```

x=1,2,･･･,9,10

96


forループ④
upto と while を利用して書き直してみて下さい
for x in 0..100 do
a = x / 100.0 * Math::PI
printf("%f %f¥n" , a , Math.sin(a))
end
%f
浮動点小数表示
Z:¥Ruby>ruby sample.rb
0.000000 0.000000
0.031416 0.031411
0.062832 0.062791
0.094248 0.094108
0.125664 0.125333
0.157080 0.156434
0.188496 0.187381
97