プログラミング言語　第七回

担当：篠沢 佳久
櫻井 彰人

平成21年 6月1日

本日の内容

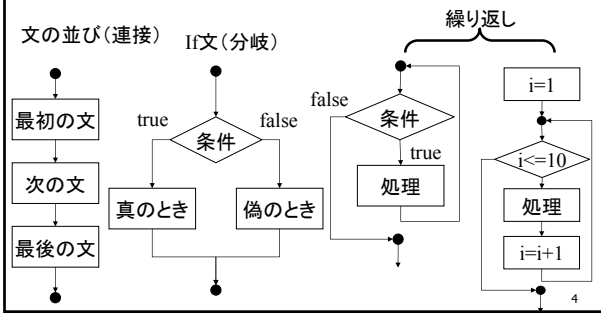
- 繰り返し(3)
- While

- 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

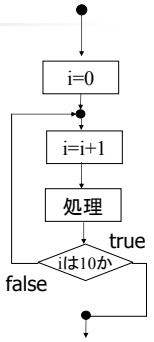
```
loop{
        式
        break 条件式
}
次の式
```

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

```
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

times①

- 同じ処理をn回繰り返したい
- n.times

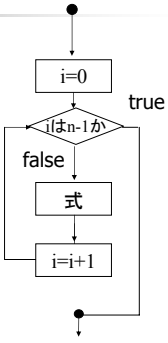
```
n.times {
    式
}
```

式をn回繰り返す

times②

```
n.times{ |i|
  式
}
```

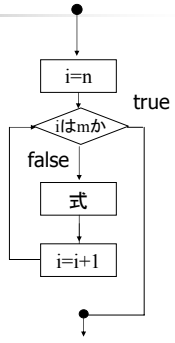
n回式を繰り返す
iには自動的に0からn-1が代入される



each

```
(n..m).each{ |i|
  式
}
```

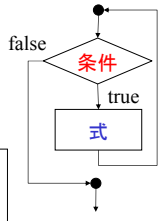
m-n+1回式を繰り返す
iには自動的にnからmが代入される



whileによる繰り返し

whileによる繰り返し

```
while 条件 do
      式
end
```

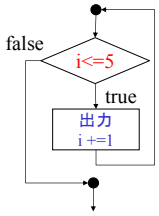


条件がtrueである場合は式を実行し，false の場合は式を実行しない

すなわち条件がtrueである限りは式を繰り返し実行する

while による繰り返し（例①）

```
i = 1
while i <= 5 do
  puts( i.to_s + "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。
```


while による繰り返し（例①'）

```
i = 1
while i <= 5 do
  puts( i.to_s + "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。
```

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

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

while による繰り返し（例①'）

```
i = 1
while i <= 5 do
  puts( i.to_s + "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。
```

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=5)
⑰ 条件判定 -> false
⑱ 終了

前のページと同じプログラム（例②）

puts の書き方

```
i = 1
while i <= 5 do
  puts i.to_s + "回目です。"
  i += 1
end
```

```
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。
```


これまで学んだ方法で書くと①

```
5.times{ |i|
    puts( (i+1).to_s + "回目です。" )
}
```

```
(1..5).each{ |i|
    puts( i.to_s + "回目です。" )
}
```


これまで学んだ方法で書くと②

```
i = 1
loop {
  puts( i.to_s + "回目です。" )
  break if i == 5
  i += 1
}
```

練習①

- 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
```


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）

```
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？

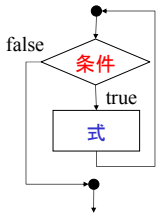
while による繰り返し（例⑥）

```
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
    - 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 )
}
```

⇔

制御変数

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

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

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

⇔

制御変数

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


例: times と while の違い③

```
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

times では書きづらい

例: times と while の違い④

- わざわざ作った例ですが

```
s = 0.0
n =10
n.times{ |i|
  r = rand()
  s  += r
}
a  = s/n
puts( "平均= #{a}")
```

"回数" が予めわかっている

```
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

"継続条件" が予めわかっている

擬似乱数の復習

- 平均、不偏分散、標準偏差を求めてみよう

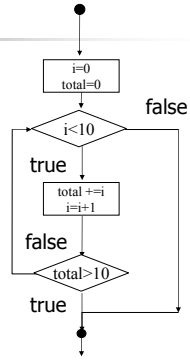
```
s = 0.0
s2= 0.0
n =10
n.times{ |i|
  r = rand()
  s  += r
  s2 += r*r
  puts( "#{r} は #{i} 番目の乱数です." )
}
a  = s/n
v  = (s2 - a*a*n)/(n-1)
sd = Math.sqrt( v )
puts( "平均= #{a}, 分散= #{v}, 標準偏差= #{sd}" )
```


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
```



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

break文④

```
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）

```
(0..5).each { |i|
  if ( i==1 || i==4 ) then
     next
  end
  puts( "iは #{i}" )
}
```

「または」です

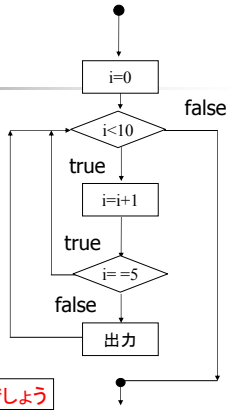
出力結果

```
iは 0
iは 2
iは 3
iは 5
```


next②

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

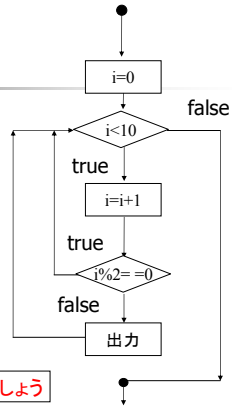
出力はどうなるでしょう



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



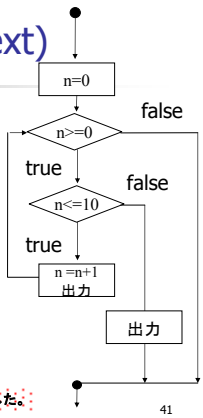
プログラム例(break,next)

```
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
```

ループの先頭であるwhileに戻る

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。
```



他のループ制御

downto, upto, step, for

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

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

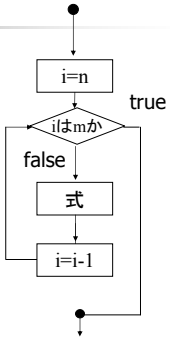
```
2.step(12,3) { |i| print( i," " ) }
```
2 5 8 11

```
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 )
}
```

x は10から0まで1刻みで変化

```
C:¥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

```
C:¥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

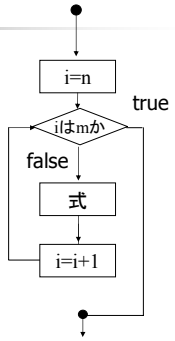
downto と while を利用して書き直してみて下さい

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


upto①

```
n.upto(m){ |i|
      式
}
```

m-n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



upto②

```
n.upto(m){ |i|
      式
}
```

⇔

```
(n..m).each { |i|
      式
}
```

基本的には each と同じ

upto③

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

each で書いた場合

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



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

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

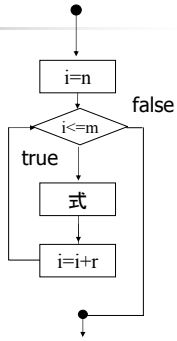
```
total = 0
11.times{ |x|
  total += x
}
print( total )
```

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


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



step②

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

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

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

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


step③

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

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

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

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


step④

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

whileを用いた場合

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


step⑤

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

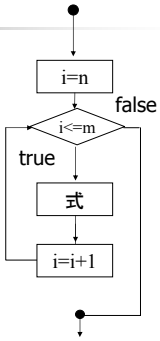
whileを用いた場合

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


forループ①

```
for i in n..m do
  式
end
```

```
(n..m).each{ |i|
  式
}
```



forループ②

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

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

forループ③

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

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


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
```


練習問題

練習問題

- 練習問題②から⑤を行ないなさい。
- 簡単な人は，web 上から自習問題をダウンロードして試してみてください。（回答とは別に）自分で考えて下さい。
- 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。

練習②

① while を用いて1以上10以下の整数の合計値を求めるプログラムを書きなさい

② while を用いて2^n（n=0,1,...10）を求めるプログラムを書きなさい

練習③

- 整数を読み込み，それをnに代入し、nが2で何回割ることができるかどうかを調べ、その結果を印字するプログラムを書きなさい。スライド22の左のプログラムに基づいてください。

練習問題④

- 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

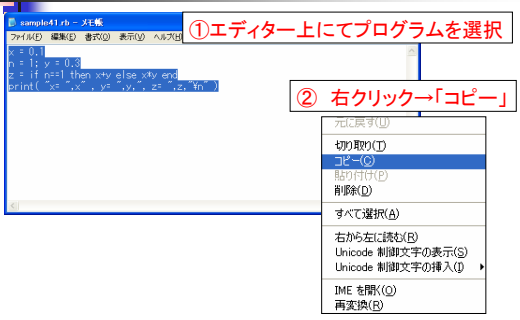
```
C:¥Ruby>ruby sample.rb
1
2
4
5
7
8
11
13
14
16
17
19
22
23
```


練習問題⑤

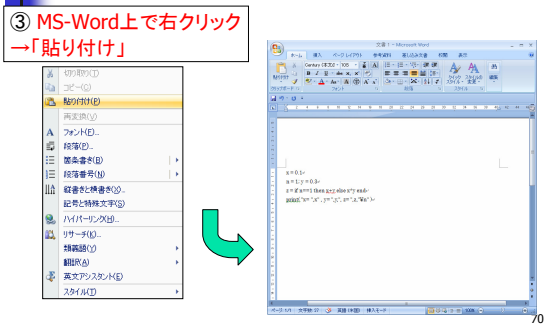
- サイコロを1000回ふった際，サイコロの目の出現回数と期待値を求めるプログラムを書きなさい．

```
C:¥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
期待値 3.453
```


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



プログラムと実行結果をMS-Word へ貼り付ける②