プログラミング言語　第五回

担当：篠沢 佳久
櫻井 彰人

平成23年5月23日

本日の内容

- 制御構造
- 条件式
  - 論理式（復習）
  - 条件式（if式）
- 繰り返し（1）
  - 無限の繰り返し

条件式

論理式（復習）
条件式（if式）

プログラムの構造

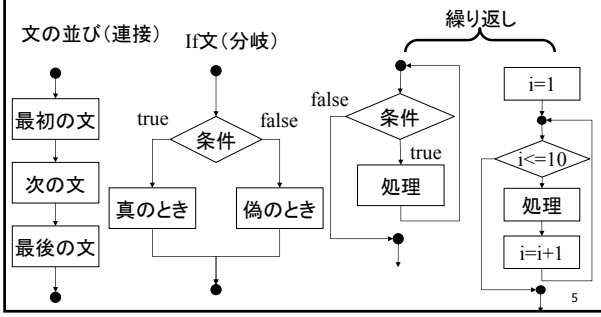
- 基本的には上から順に実行される

```
print( "一番目の数値を入力してください: " )
line = gets.chop
x1 = line.to_f
print( "二番目の数値を入力してください: " )
line = gets.chop
x2 = line.to_f
print( "#{x1} + #{x2} = #{ x1+x2 }" )
```

実行順番

- 実行順番を変えることも必要
  - 制御構造

制御構造



条件式（if式）

プログラムを書いている際には

- ある条件式が成立した場合には，処理Aを行ない，成立しなかった場合には処理Bを行なう

という要求が発生する

条件式(if式)の種類①

① if 論理式 then 式1 end

② if 論理式 then 式1 else 式2 end

③ if 論理式0 then
  if 論理式1 then 式11 else 式12 end
else
  if 論理式2 then 式21 else 式22 end
end

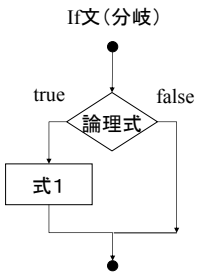
条件式(if式)の種類②

④ if 論理式1 then
  式1 # 論理式1がtrueのときの値
elsif 論理式2 then
  式2 # 論理式2がtrueのときの値
else
  式3 # 論理式1,2がfalseのときの値
end

if式①

if 論理式 then 式1 end

if 論理式 then
  式1 # 論理式がtrueのときの値
end



if式①(例)

```
x = 5
if x < 0  then
  print( x )
end
```

```
x = 0
if x < 0  then
  print( x )
end
```

```
x = -5
if  x < 0  then
  print( x )
end
```

print(x) が実行されるのは？

if式①(例)
(print文が実行されるか考えなさい)

```
x = -5
if x != 0 then
   print( x )
end
```

```
x = -5
y = -10
if x < 0  and y < 0 then
   print( x )
end
```

```
x = -5
if not x == 0 then
   print( x )
end
```

```
x = -5
y = 10
if x < 0  or y < 0 then
   print( x )
end
```


if文の構造①

```
if a > 0 then
  if a < 10 then
     print( a )
  end
end
```

if式中にif式を書くことも可能
「aが0より大きい」かつ
「aが10より小さい」

同じ式

```
if a > 0 and a < 10 then
     print( a )
end
```

if文の構造②
endの対応付けに注意

```
if a > 0 then
  if a < 10 then
    print( a )
  end
end
```

end

end

「if 条件 then」を書いたら、すぐに「end」で閉じること！

if文の構造③

```
if a > 0 then
  if a < 10 then
    x = a*10
  end
  if a >= 10 then
    x = a*100
  end
end
```

if式中にif式を書くことも可能

aが10より小さい

aが10以上

if文の構造④

前のページと同じプログラム

```
if a > 0 then
  if a < 10 then
    x = a*10
  end
end
if a > 0 then
  if a >= 10 then
    x = a*100
  end
end
```


if式①（例）
（aの値を考えなさい）

```
a = ?
if a > 0 then
  if a > 10 then
    a += 1
  end
  a += 1
end
print( a )
```

aの値はどうなるでしょうか

a = -100

a = 0

a = 1

a = 10

a = 100

a += 1 は
a = a + 1 を計算

論理式（復習）

- 論理値（真偽値）を計算する式を論理式と呼ぶことにします。
- 論理式の基本は、数式または文字列式（の値）と数式または文字列式（の値）とを比較演算子を用いて比較する式です。これをand/or/not で組み合わせます

比較演算子

演算子	用途	例	演算結果
==	等	3==2	false
>	大	4 > 2	true
<	小	4 < 2	false
>=	大or等	4>=2	true
<=	小or等	4<=2	false
!=	非等	3 != 2	true

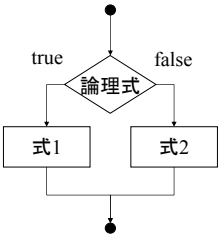
論理演算子

演算子	用途	例	演算結果
!	否定	! (3==2)	true
&&	かつ	2==2 && 4>2	true
||	または	2==3 || 4>2	true
not	否定	not 3==2	true
and	かつ	2==2 and 4>2	true
or	または	2==3 or 4>2	true

if式②

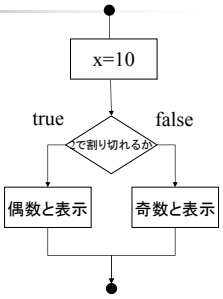
if 論理式 then 式1 else 式2 end

```
if 論理式 then
   式1  # 論理式がtrueのときの値
else
   式2  # 論理式がfalseのときの値
end
```



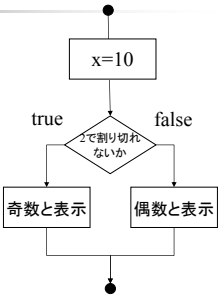
if式②（例）

```
x=10
if  x % 2 == 0 then
    print(  x , "は偶数です" )
else
    print(  x , "は奇数です" )
end
```



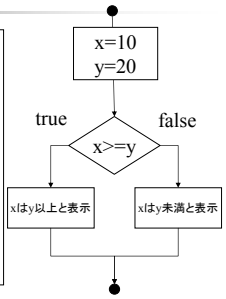
if式②（例）

```
x=10
if  x % 2 != 0 then
    print(  x , "は奇数です" )
else
    print(  x , "は偶数です" )
end
```



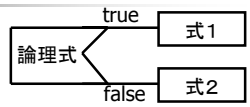
if式②（例）

```
x=10
y=20
if  x >=y then
    print(  x , "は" , y , "以上" )
else
    print(  x , "は" , y , "未満" )
end
```

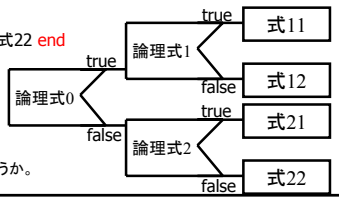


if式③

if 論理式 then 式1 else 式2 end

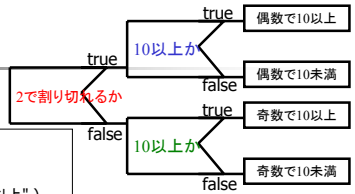



```
if 論理式0 then
  if 論理式1 then 式11 else 式12 end
else
  if 論理式2 then 式21 else 式22 end
 end
```



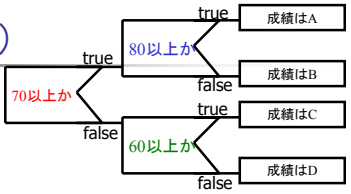
正式にはPAD図といいます
ここでは、分岐図とでもいいましょうか。

if式③（例）




```
x=10
if  x % 2 == 0 then
   if  x >= 10  then
         print(  x , "は偶数で10以上" )
   else
         print(  x , "は偶数で10未満" )
   end
else
   if  x >= 10  then
         print(  x , "は奇数で10以上" )
   else
         print(  x , "は奇数で10未満" )
   end
end
```


if式③（例）




```
score = 75
grade =
    if score >= 70 then
      if score >= 80 then "A" else "B" end
    else
      if score >= 60 then "C" else "D" end
    end
print( "Your score #{score} corresponds to
  #{grade}¥n" )
```


複数の式からなる式

- Ruby では、複数の式を並べたものも式となる。

```
score = 75
grade =
    if score >= 70 then
      if score >= 80 then "A" else "B" end
    else
      if score >= 60 then "C" else "D" end
    end
print( "Your score #{score} corresponds to
  #{grade}¥n" )
```

式

if式③（例）

```
if a > 10 then
   if a > 100 then
        a += 1
   else
        a -= 1
   end
else
   if  a > -10  then
        a += 10
   else
        a -= 10
   end
end
```

aの値はどうなるでしょうか

a = -100

a = -10

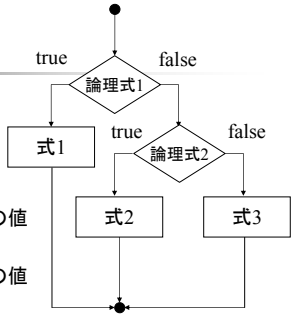
a = 0

a = 10

a = 100

a = 1000

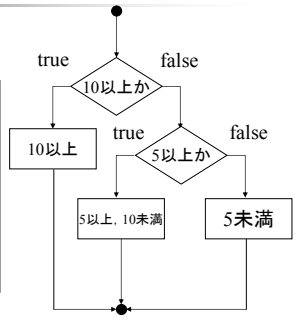
if式④



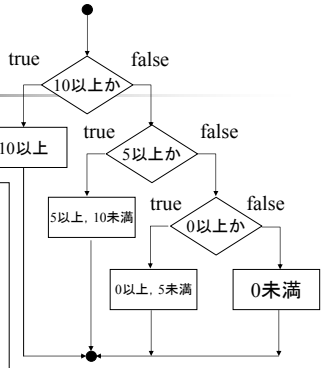

```
if 論理式1 then
   式1  # 論理式1がtrueのときの値
elsif 論理式2 then
   式2  # 論理式2がtrueのときの値
else
   式3  # 論理式1,2がfalseのときの値
end
```


if式④（例）

```
if x >= 10  then
   print( x , "は10以上" )
elsif x >= 5  then
   print( x , "5以上，10未満" )
else
  print( x , "は5未満" )
end
```

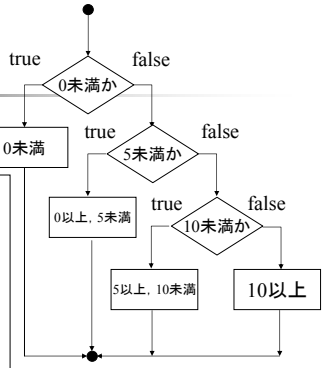


if式④（例）




```
if x >= 10  then
    print( x , "は10以上" )
elsif x >= 5  then
    print( x , "は5以上，10未満" )
elsif x >= 0  then
    print( x , "は0以上，5未満" )
else
    print( x , "は0未満" )
end
```


if式④（例）




```
if x < 0 then
    print( x , "は0未満" )
elsif x < 5  then
    print( x , "は0以上，5未満" )
elsif x < 10 then
    print( x , "は5以上，10未満" )
else
    print( x , "は10以上" )
end
```

前のページと同じプログラムです

読みやすいプログラム

- プログラムは文法的に間違えていなければ動きます
- しかし、読みやすいプログラムを書くことを心がけましょう
- 例えばインデント（字下げ）を揃えると分かりやすいプログラムになります

インデント①

```
x = 5
y = -10
if x+y < 0  then
print( x )
print( y )
end
```

```
x = 5
y = -10
if x+y < 0  then
    print( x )
    print( y )
end
```

x+y < 0 の場合，二つの文が実行されるとすぐに分かる

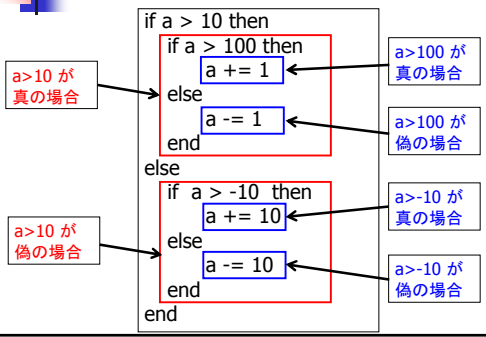
インデント②

左のプログラムと比べて条件式によってどこか実行されるか理解しやすい

```
if a > 10 then
if a > 100 then
a += 1
else
a -= 1
end
else
if  a > -10  then
a += 10
else
a -= 10
end
end
```

```
if a > 10 then
    if a > 100 then
        a += 1
    else
        a -= 1
    end
else
    if  a > -10  then
        a += 10
    else
        a -= 10
    end
end
```


インデント③



インデントのつけ方

スペースもしくはTabキーで揃える

```
if a > 10 then
   if a > 100 then
        a += 1
   else
        a -= 1
   end
else
   if  a > -10  then
        a += 10
   else
        a -= 10
   end
end
```

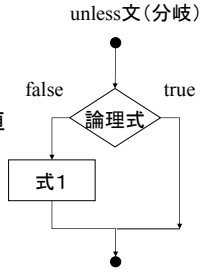

その他の条件式

unless, case

unless文

```
unless 論理式 then 式1 end
```

```
unless 論理式 then
   式1 # 論理式がfalseのときの値
end
```



unless文

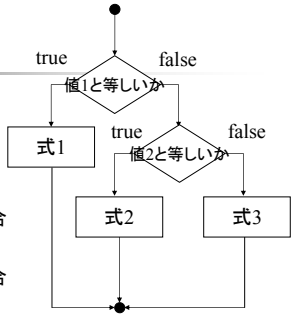
```
x=10
if  x % 2 != 0 then
    print(  x , "は奇数です" )
else
     print(  x , "は偶数です" )
end
```

C:¥Ruby>ruby sample.rb
10は偶数です

```
x=10
unless  x % 2 == 0 then
    print(  x , "は奇数です" )
else
     print(  x , "は偶数です" )
end
```

C:¥Ruby>ruby sample.rb
10は偶数です

case文




```
case  変数
when 値1 then
   式1  #  変数の値が値1の場合
when 値2 then
   式2  #  変数の値が値2の場合
else
   式3  # 変数の値が値1，2以外の場合
end
```


case文①

```
x=10
y=5
opt="+"

case opt
when "+" then
        print( x , "+" , y , "=" , x+y )
when "-" then
        print( x , "-" , y , "=" , x-y )
when "*" then
        print( x , "*" , y , "=" , x*y )
when "/" then
        print( x , "/" , y , "=" , x/y )
else
        print( " x = " , x , " y= " , y )
end
```

opt は "+"
→ このwhen文が実行される

C:¥Ruby>ruby sample.rb
10+5=15

case文①'

```
x=10
y=5
opt="+"

case opt
when "+" then
        print( x , "+" , y , "=" , x+y )
when "-" then
        print( x , "-" , y , "=" , x-y )
when "*" then
        print( x , "*" , y , "=" , x*y )
when "/" then
        print( x , "/" , y , "=" , x/y )
else
        print( " x = " , x , " y= " , y )
end
```

opt は "^"
→該当しないためelse文が実行される

```
C:¥Ruby>ruby sample.rb
x = 10 y= 5
```


case文②

```
opt=3

case opt
when 1,2,3
        puts( "1～3" )
when 4,5,6
        puts( "4～6" )
when 7,8,9
        puts( "7～9" )
else
        puts( "10～" )
end
```

opt は 3
→ このwhen文が実行される

値は複数個書くことができる
(「, 」で区切る)

```
C:¥Ruby>ruby sample.rb
1～3
```


case文②'

```
opt=20

case opt
when 1,2,3
        puts( "1～3" )
when 4,5,6
        puts( "4～6" )
when 7,8,9
        puts( "7～9" )
else
        puts( "10～" )
end
```

opt は 20
→該当しないためelse文が実行される

```
C:¥Ruby>ruby sample.rb
10～
```


無限の繰り返し

```
loop
break
```


繰り返しの必要性①

- 1から10の整数を出力したい
- print("1 2 3 4 5 6 7 8 9 10¥n")

- 1から1000の整数を出力したい
- print("1 2 … 1000¥n")
  - と書ければよいが…

繰り返しの必要性②

プログラムを書いている際には

- 同じ処理をある回数だけ行ないたい
- ある条件が成立するまで、同じ処理を繰り返したい
- 値を変化させながら、同じ処理を繰り返したい

という要求が発生する

繰り返しの必要性③

- 繰り返しを行なう際に考えること
  - 何を繰り返すのか
  - 何回繰り返しを行なうのか
  - どういう条件で繰り返しを停止するのか

繰り返しの必要性④

- 1から1000の整数を出力したい
  - 何を繰り返すのか
    →出力を繰り返す
  - どういう条件で繰り返しを停止するのか
    →1000まで出力したら停止する

無限の繰り返し①

```
loop{
        式
}
```

式が永久に実行される

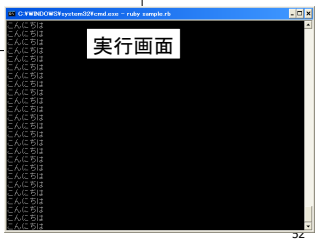
停止するために break を用いる

```
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②

```
loop{
   print( "こんにちは¥n" )
}
```



無限に「こんにちは」と表示される

停止するにはCtrlキーを押しながらc

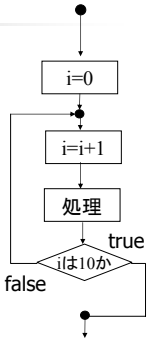
無限の繰り返し

- loop{} の場合，式が無限に繰り返される
- 停止させるためには，break式と条件式で設定しなければならない

無限の繰り返し③

10回で「こんにちは」の表示をやめるには？

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



i が10となったら停止する

無限の繰り返し③

10回で「こんにちは」の表示をやめるには？

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

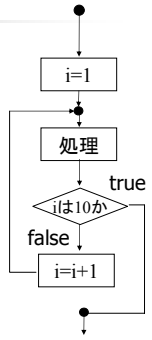
```
C:¥Ruby>ruby sample.rb
1回目のこんにちは   i=1
2回目のこんにちは   i=2
3回目のこんにちは
4回目のこんにちは
5回目のこんにちは
6回目のこんにちは
7回目のこんにちは
8回目のこんにちは
9回目のこんにちは   i=9
10回目のこんにちは  i=10
```

i が10となったら停止する

無限の繰り返し③'

10回で「こんにちは」の表示をやめるには？

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



前のページと同じ動作をします

間違えやすいミス①

10回で「こんにちは」の表示をやめるには？

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

```
C:¥Ruby>ruby sample.rb
sample.rb:2: undefined method `+' for nil:NilClass (NoMethodError)
        from sample.rb:1:in `loop'
        from sample.rb:1
```

変数 i が初期化されていない

間違えやすいミス②

10回で「こんにちは」の表示をやめるには？

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

```
C:¥Ruby>ruby sample.rb
0回目のこんにちは
0回目のこんにちは
0回目のこんにちは
0回目のこんにちは
0回目のこんにちは
0回目のこんにちは
```

変数 i を1ずつ増やしていない
→ iは0のままのためプログラムは終了しない（無限ループ）

間違えやすいミス③

10回で「こんにちは」の表示をやめるには？

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

```
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
10回目のこんにちは
```

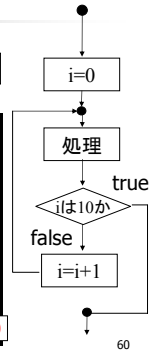
i の初期値を0として開始

無限の繰り返し④

10のべき乗を表示するプログラム

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

```
1           i=0
10          i=1
100         i=2
1000
10000
100000
1000000
10000000
100000000
1000000000  i=9
10000000000 i=10
```

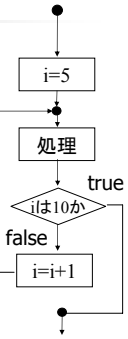


無限の繰り返し④'

10のべき乗を表示するプログラム

```
i = 5
loop{
   print( 10 ** i , "¥n" )
   break  if i == 10
   i = i+1
}
```

```
100000      i=5
1000000     i=6
10000000
100000000
1000000000  i=9
10000000000 i=10
```

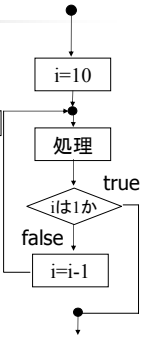


無限の繰り返し④''

10のべき乗を表示するプログラム

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

```
10000000000 i=10
1000000000  i=9
100000000
10000000
1000000
100000
10000
1000
100         i=2
10          i=1
```



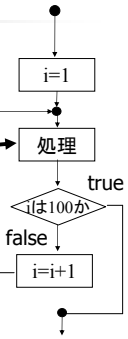
無限の繰り返し⑤

100以下の偶数を表示するプログラム

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

```
2           i=2
4           i=4
6
8
10
12

98
100         i=100
```



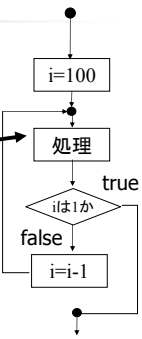
無限の繰り返し⑤'

100以下の偶数を表示するプログラム

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

```
100         i=100
98          i=98
96
94
92

4           i=4
2           i=2
```

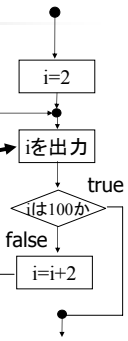


無限の繰り返し⑤''

100以下の偶数を表示するプログラム

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

i に2ずつ加算



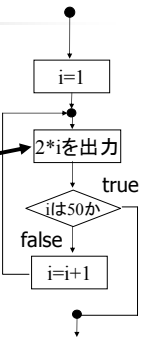
無限の繰り返し⑤'''

100以下の偶数を表示するプログラム

```
i = 1
loop{
   print( 2*i , "¥n" )
   break  if i == 50
   i = i+1
}
```

iの2倍の値を出力

i に1ずつ加算



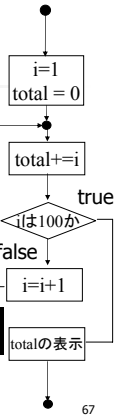
無限の繰り返し⑥

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 )
```

iが100の場合，停止する

```
C:¥Ruby>ruby sample.rb
合計は5050
```

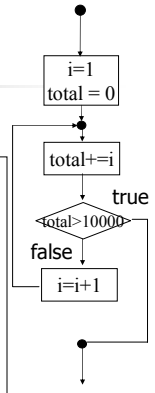


無限の繰り返し⑥'

どういうプログラムでしょうか

```
i = 1
total = 0
loop{
   total += i
   break  if total > 10000
   i = i+1
}
print( i )
```

totalの値が10000を越えたら停止

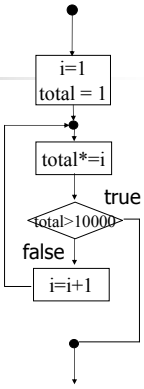


無限の繰り返し⑥''

どういうプログラムでしょうか

```
i = 1
total = 1
loop{
   total *= i
   break  if total > 10000
   i = i+1
}
print( i )
```

totalの値が10000を越えたら停止

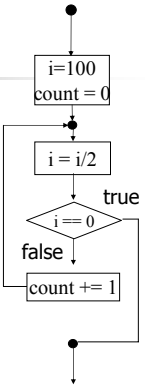


無限の繰り返し⑥'''

どういうプログラムでしょうか

```
i = 100
count = 0
loop{
   i = i / 2
   break  if i == 0
   count = count + 1
}
print( count )
```

iの値が0の場合，停止



無限の繰り返しのまとめ①

- loop{ … }
- 上記「 …」を無限に繰り返す。無限個のコピーを作ると考えてもよい。ただし、いきなり作るのではなく、必要があったら作るのですが。
- しかし、いずれにせよ、無限に作られるのは困る。
- 途中で止めなければ意味がない。
- 途中で止める道具（これも式だが、まったく式らしくない）が break です。

```
i = 0
loop{
  print( "やっほ～ " )
  if i>=10 then break end
  puts( "      Yee-ha! " )
  i = i+1
}
```

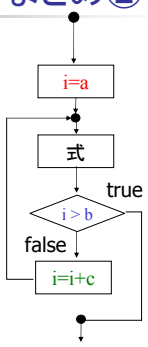
```
i = 0
loop{
  print( "やっほ～ " )
  break if i>=10
  puts( "      Yee-ha! " )
  i = i+1
}
```

これを if 修飾子という
式 if 論理式
が一般形

無限の繰り返しのまとめ②

a から b まで c ずつ加算しながら繰り返し処理を行なう

```
i = a
loop{
      式
      break if  i > b
      i += c
}
```



if修飾子①

```
if i==0 then
   break
end
```

同じ

```
break if i==0
```

```
if a > b then
   print( " aはbよりも大きい" )
end
```

同じ

```
print( " aはbよりも大きい" ) if a>b
```


if修飾子②

```
if a !=b  then
   a = a * 2
   b = b + 5
end
```

同じ

```
a=a*2 ; b = b+ 5  if a != b
```


標準入力と繰り返し

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（chomp）
  - 最後の一文字を削除する（最後の一文字が改行ならば削除する）

- 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

標準入力①（復習）

- getsによる標準入力
- 文字列型で入力される
- 末尾に"¥n"（もしくは"¥r¥n"）が挿入される

⇩

- 整数値（小数値）として利用したい場合
- 末尾の改行を削除
- 文字列型から整数（小数）へ変換する必要がある

標準入力②（復習）

入力

```
irb(main):001:0> x=gets
3.1415
=> "3.1415¥r¥n"
irb(main):002:0> x.chop
=> "3.1415"
irb(main):003:0> x.chop.to_f
=> 3.1415
irb(main):004:0> x.chop.to_i
=> 3
```

chop で最後の一文字（改行）を削除

文字列型を小数に変換

文字列型を整数に変換

標準入力③（復習）

入力

```
irb(main):001:0> x=gets
abcd
=> "abcd¥r¥n"
irb(main):002:0> x.chop
=> "abcd"
```

chop で最後の一文字（改行）を削除

標準入力（復習）

「5↵」と入力

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

```
C:¥Ruby>ruby sample.rb
5
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
```

gets により "5¥r¥n"
chomp により "5"
to_i により整数5に変換される

簡単に書くには...①

```
a = gets.chomp.to_i
b = gets.chomp.to_i
c = gets.chomp.to_i
d = gets.chomp.to_i
e = gets.chomp.to_i
sum = a+b+c+d+e
print( "合計は" , sum , "です¥n" )
```

5個の整数を読み込み，合計を出力

```
C:¥Ruby>ruby sample.rb
43
34
2
1
1
合計は81です
```


簡単に書くには...②

```
sum = 0
i = 0
loop{
        x = gets.chomp.to_i
        sum += x
        i += 1
        break if i == 5
}
print( "合計は" , sum , "です¥n" )
```

変数xに整数を入力

5回入力したら停止

```
C:¥Ruby>ruby sample.rb
4
1
4
6
7
合計は22です
```


入力の繰り返し①

```
max = 0
i = 0
loop{
        x = gets.chomp.to_i
        if x > max then
            max = x
        end
        i += 1
        break if i == 5
}
print( "最大は" , max , "です¥n" )
```

変数xに整数を入力

5回入力したら停止

```
C:¥Ruby>ruby sample.rb
23
12
68
45
23
最大は68です
```


入力の繰り返し②

```
sum = 0
loop{
        x = gets.chomp.to_i
        sum += x
        break if sum > 100
}
print( "合計は" , sum , "です¥n" )
```

変数xに整数を入力

sumの値が100を越えたら停止

```
C:¥Ruby>ruby sample.rb
34
23
17
56
合計は130です
```

入力回数が分からない場合

- 前々頁のプログラムは入力が5回
- 前頁のプログラムは条件式によって停止
- 入力回数が分からない場合はどうすればよいか
  - 特定の文字(Enterなど)を入力した場合のみ入力を終了させるようにする

繰り返し入力できるようにするためには①

```
loop{
  print( "何か文字を入れて下さい(Enterで終了します)¥n" )
  line = gets.chomp
  break if line == ""
  print( line , "¥n" )
}
```

```
C:¥Ruby>ruby sample.rb
何か文字を入れて下さい(Enterで終了します)
24
24
何か文字を入れて下さい(Enterで終了します)
abcd
abcd
何か文字を入れて下さい(Enterで終了します)
sdds
sdds
何か文字を入れて下さい(Enterで終了します)
```

Enterで終了

キーボードからの入力

キーボードでEnterキーを入力した場合

```
loop{
  line = gets.chomp
  break if line == ""
  print( line , "¥n" )
}
```

chomp で改行文字が削除されるため，lineには空白文字が入る

lineには空白文字が入っているため break が実行され停止する

改行の処理(復習)

```
irb(main):013:0> gets

=> "¥r¥n"
irb(main):014:0> x = gets

=> "¥r¥n"
irb(main):015:0> print( x )

=> nil
irb(main):016:0> x.chomp
=> ""
```

Enterキーのみを入力

chomp により改行を削除されるため空白文字となる

繰り返し入力できるようにするためには②

```
loop{
  print( "何か文字を入れて下さい(stopで終了します)¥n" )
  line = gets.chomp
  break if line == "stop"
  print( line , "¥n" )
}
```

```
C:¥Ruby>ruby sample.rb
何か文字を入れて下さい(stopで終了します)
32
32
何か文字を入れて下さい(stopで終了します)
stop
```

stopで終了

繰り返し入力できるようにするためには③

```
sum = 0
loop{
  print( "整数を入れて下さい(Enterで終了します)¥n" )
  line = gets.chomp
  break if line == ""
  print( line , "¥n" )
  sum += line.to_i
}
print( "合計値は" , sum )
```

末尾の改行を削除

空白かどうか

文字型の変数line整数に型変換

```
sum = 0
loop{
  print( "整数を入れて下さい(Enterで終了します)¥n" )
  line = gets.chomp
  break if line == ""
  print( line , "¥n" )
  sum += line.to_i
}
print( "合計値は" , sum )
```

```
C:¥Ruby>ruby sample.rb
整数を入れて下さい(Enterで終了します)
32
32
整数を入れて下さい(Enterで終了します)
23
23
整数を入れて下さい(Enterで終了します)
11
11
整数を入れて下さい(Enterで終了します)

合計値は66
```


繰り返し入力できるようにするためには④

```
loop {
  print( "Enter your score: " )
  line = gets.chomp
    break if line==""
  score = line.to_f
  grade =
    if score >= 70 then
      if score >= 80 then "A" else "B" end
    else
      if score >= 60 then "C" else "D" end
    end
  print( "Your score #{score} corresponds to #{grade}¥n" )
}
```

改行キー(Enter)のみ入力された場合，停止

```
G:¥Ruby>ruby -Ks sample.rb
Enter your score: 90
Your score 90.0 corresponds to A
Enter your score: 40
Your score 40.0 corresponds to D
Enter your score:
```


どこが違うでしょうか

```
loop{
  print( "何か文字を入れて下さい(0で終了します)¥n" )
  line = gets.chomp
  break if line == "0"
  print( line , "¥n" )
}
```

```
loop{
  print( "何か文字を入れて下さい(0で終了します)¥n" )
  line = gets.chomp.to_i
  break if line == 0
  print( line , "¥n" )
}
```

実は英文字でも終了します
なぜでしょう

どこが違うでしょうか

英文字を整数型に型変換すると...

```
irb(main):001:0> "a".to_i
=> 0
irb(main):002:0> "x".to_i
=> 0
irb(main):003:0> "abc".to_i
=> 0
irb(main):004:0> "1".to_i
=> 1
irb(main):005:0> "10".to_i
=> 10
```


練習問題

練習①～③

練習①

- 正の整数a,b,cに対し、これを各辺とする三角形が存在することと下記の条件を満たすことは同値です．

$$\begin{array}{ccccc} a & < & b & + & c \\ b & < & a & + & c \\ c & < & a & + & b \end{array}$$

- そこで，整数a,b,c をキーボードから入力し，三角形が作れるかどうかを調べるプログラムを書きなさい．

練習①（実行結果の例）

```
H:¥Ruby>ruby sankaku.rb
辺の長さ?
3
4
5
三角形は作れます
```

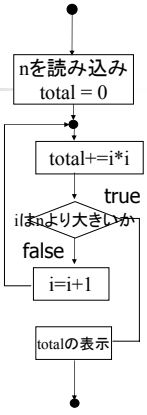
```
H:¥Ruby>ruby sankaku.rb
辺の長さ?
3
6
1
三角形は作れません
```

練習②

- 1から10までの整数について，偶数か奇数かを判定するプログラムを書きなさい
- 「無限の繰り返し⑤」を改良すればよい

練習③

- キーボードから整数n（n>1）を読み込み，1からnまでの自乗和を求めることができるプログラムをloop{}を用いて書きなさい
- 「無限の繰り返し⑥」を改良すればよい

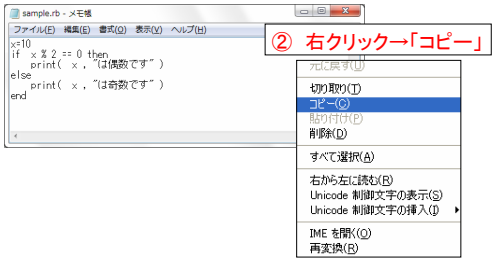


本日の練習問題

- 練習問題①～③を行なって下さい
- 簡単な人は自習問題も試してみて下さい
- プログラム（テキストで貼り付けて下さい）と実行結果をワープロに貼り付けてkeio.jp の教育支援システムから提出して下さい
- 時間があったらプログラムの説明も書いて下さい

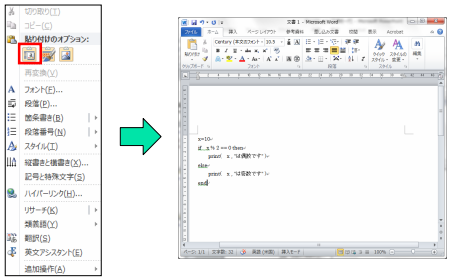
プログラムと実行結果をMS-Word へ貼り付ける①

①エディター上にてプログラムを選択

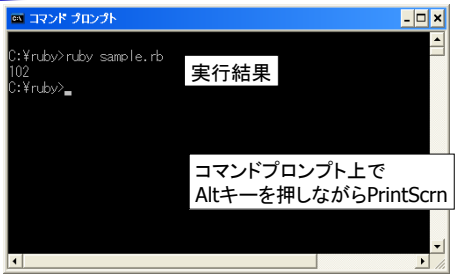


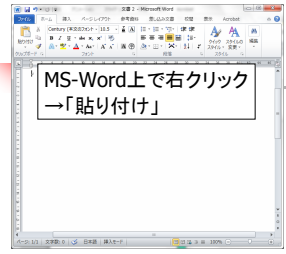
プログラムと実行結果をMS-Word へ貼り付ける②

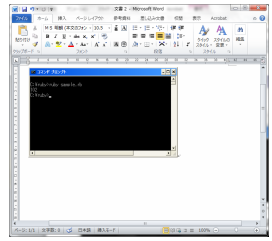
③ MS-Word上で右クリック→「貼り付け」



プログラムと実行結果をMS-Wordへ貼り付ける②



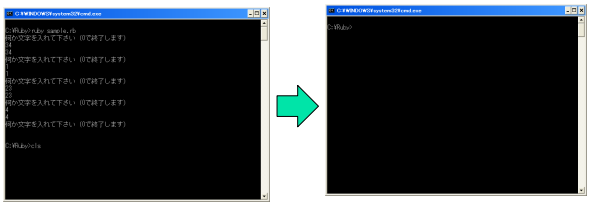




コマンドラインの画面が貼り付けられる

プログラムと実行結果をMS-Wordへ貼り付ける③

コマンドプロンプトの画面をきれいにするには



> cls と入力