プログラミング言語　第五回

担当：篠沢 佳久
櫻井 彰人

平成21年 5月18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

Ruby vs. Excel

- 浮動小数点数の計算能力は同じ
- 整数の計算能力は、Ruby が上
  - Ruby なら何桁でも計算できる
  - Excel には「整数計算だけやって！」ということができない欠点がある
- 使いやすさは Excel
  - Excel には、（Excel がもっている）関数を使うのが簡単という長所がある
- 拡張性は Ruby
  - Ruby には、自分の必要な関数が定義できるという長所がある（Excelでも Visual Basic を使えば類似のことはできる）

条件式

論理式（復習）
条件式（if式）

プログラムの構造

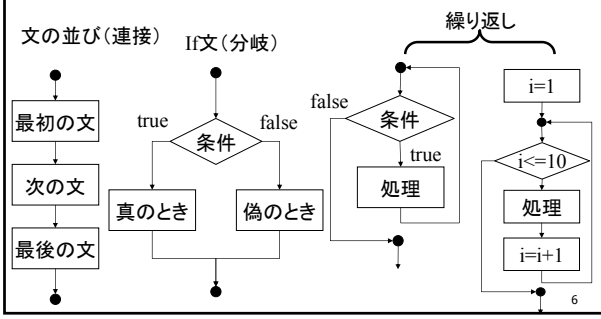
- 基本的には上から順に実行される

```
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

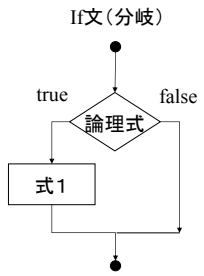
条件式(if式)の種類②

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)
(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

```
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

同じ式

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


if文の構造

```
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

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

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

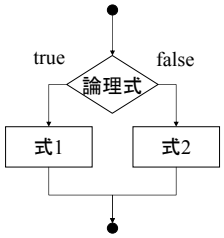
演算子	用途	例	演算結果
==	等	3==2	false
>	大	4 > 2	true
<	小	4 < 2	false
>=	大or等	4>=2	true
<=	小or等	4<=2	false
!=	非等	3 != 2	true

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

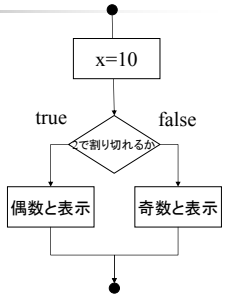
if 論理式 then 式1 else 式2 end




```
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
```

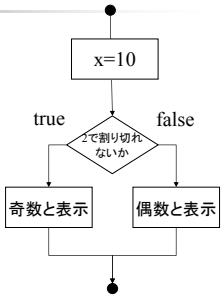

if式②（例）

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



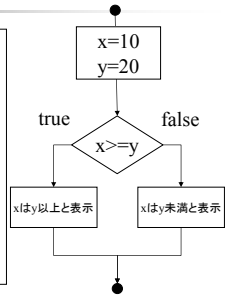
if式②（例）

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



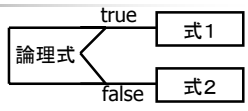
if式②（例）

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

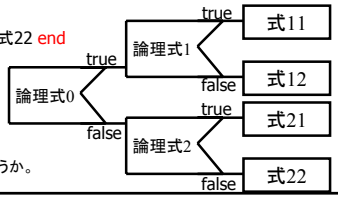


if式③

if 論理式 then 式1 else 式2 end

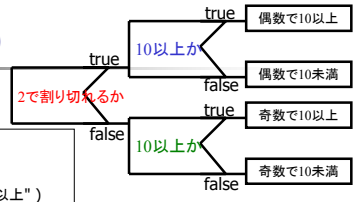



```
if 論理式0 then
  if 論理式1 then 式11 else 式12 end
else
  if 論理式2 then 式21 else 式22 end
 end
```



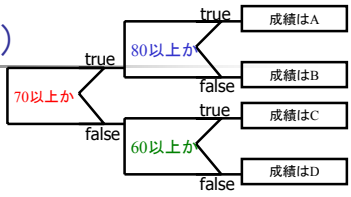
正式にはPAD図といいます
ここでは、分岐図とでもいいましょうか。

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
```

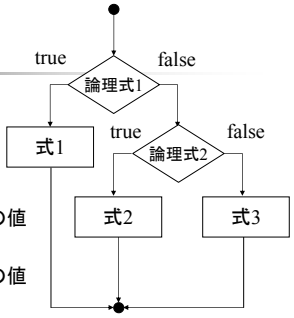
aの値はどうなるでしょうか

- a = -100
- a = -10
- a = 0
- a = 10
- a = 100
- a = 1000


```
y =
if Math.sin(Math::PI/4) >= 0.1
and
( if Math::E == 2.7 then
        2
    else
        1
end ) > 1.5 then
    if Math.exp(0.5) >= 0.3 then
        0.3
    else
        0.5
    end
else
    Math.log(23)
end
```

論理式の中にif式を書くことも可能
式の値は?

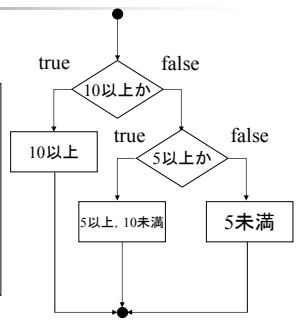
if式④



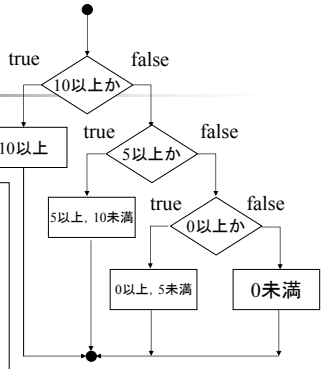

```
if 論理式1 then
   式1  # 論理式1がtrueのときの値
elsif 論理式2 then
   式2  # 論理式2がtrueのときの値
else
   式3  # 論理式1,2がfalseのときの値
end
```


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
```


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

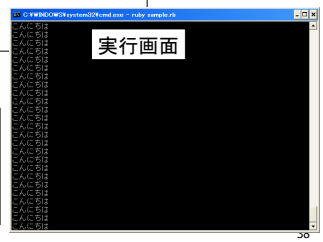
停止するために break を用いる

```
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

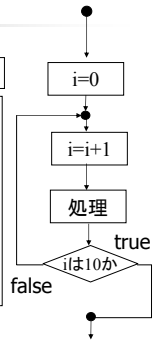
無限の繰り返し

- 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？

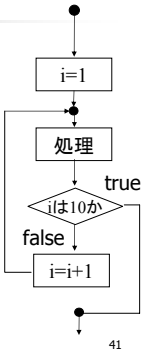
```
i = 0
loop{
  i = i+1
  print( i , "回目のこんにちは¥n" )
  break  if i == 10
}
```



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？

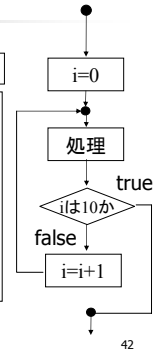
```
i = 1
loop{
  print( i , "回目のこんにちは¥n" )
  break  if i == 10
  i = i+1
}
```



無限の繰り返し④

10のべき乗を表示するプログラム

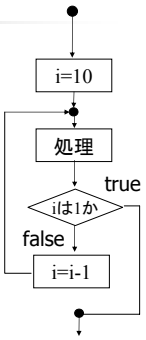
```
i = 0
loop{
  print( 10 ** i , "¥n" )
  break  if i == 10
  i = i+1
}
```



無限の繰り返し④'

10のべき乗を表示するプログラム

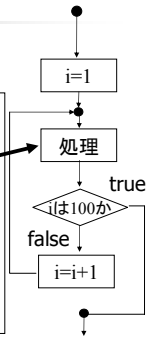
```
i = 10
loop{
   print( 10 ** i , "¥n" )
   break  if i == 1
   i = i-1
}
```



無限の繰り返し⑤

100以下の偶数を表示するプログラム

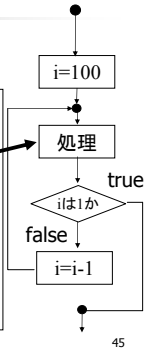
```
i = 1
loop{
   if  i % 2 == 0 then
       print( i , "¥n" )
   end
   break  if i == 100
   i = i+1
}
```



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
}
```

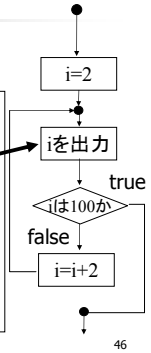


無限の繰り返し⑤''

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
}
```

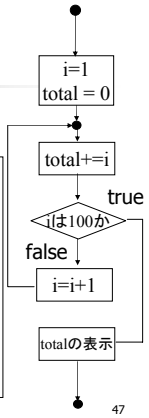
i に2ずつ加算



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

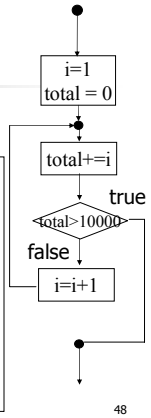
```
i = 1
total = 0
loop{
   total += i
   break  if i == 100
   i = i+1
}
print( "合計は" , total )
```



無限の繰り返し⑥'

どういうプログラムでしょうか

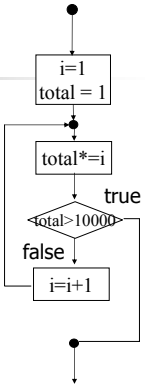
```
i = 1
total = 0
loop{
   total += i
   break  if total > 10000
   i = i+1
}
print( i )
```



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 )
```



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
}
```

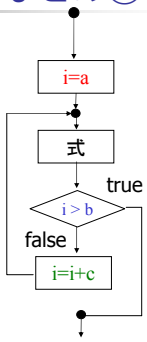
```
i = 0
loop{
  print( "やっほ～ " )
  break if i>=10
  puts( "      Yee-ha! " )
  i = i+1
}
```

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（復習）

「5Enter」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¥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" )
```

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
```


入力回数が分からない場合

- 前頁のプログラムは入力が5回

- 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

=> "¥n"
irb(main):014:0> x = gets

=> "¥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" )
}
```

stopで終了

```
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
```


繰り返し入力できるようにするためには③

```
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

```
G:¥Ruby>ruby -Ks sample0405.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0以外でも終了しますなぜでしょう

練習①

- 下記と同じプログラムをif式④（if～elsif～else）の条件式で書きなさい

```
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" )
```

練習②

- 成績がAならば80点以上，Bならば80点未満70点以上，Cならば70点未満60点以上，Dならば60点未満と出力するプログラムを書きなさい

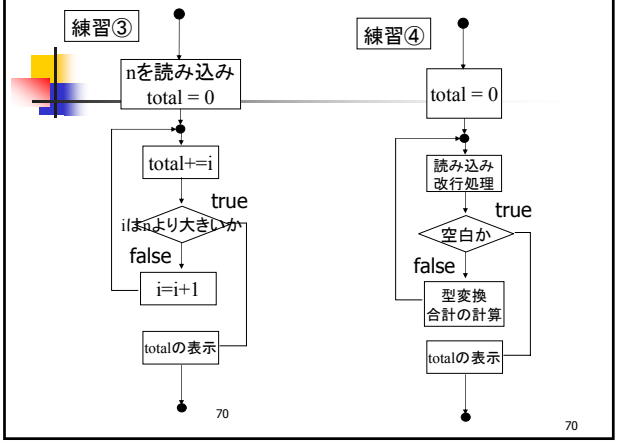
```
grade = "A"
if grade == "A" then
      print( "80点以上" )




end
```


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

練習④

- キーボードから複数個の整数を読み込み，それらの合計値および平均値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
- Enterキーを入力した場合，読み込みを停止し，合計を出力させるものとする

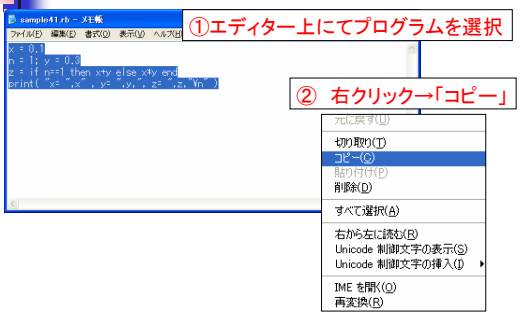


70

本日の練習問題

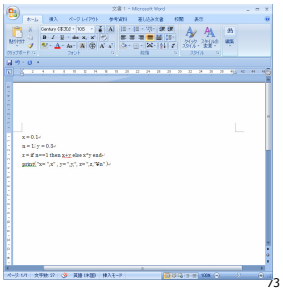
- 練習問題①～④をできる限り試してみなさい
- 簡単な人は自習問題も試してみて下さい
- プログラム（テキストで貼り付けて下さい）と実行結果をワープロに貼り付けてkeio.jp の教育支援システムから提出して下さい（途中であったり，実行できなくてもかまわないので書けた部分まで提出して下さい）
- 時間があったらプログラムの説明も書いて下さい

プログラムと実行結果をMS-Wordへ貼り付ける①



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