プログラミング言語　第四回

担当：篠沢　佳久
櫻井　彰人

平成22年 5月10日

本日の内容

- Rubyプログラムの作成と実行方法
  - 第一回の実習の復習
- 標準出力
- 標準入力
- 第一回レポート課題

履修者のみなさんへ

- プログラミングは簡単には身に付きません（一つの言語をマスターすると他の言語を覚えるのは比較的容易です）
- 講義資料には例題を多くつけます
- 講義では全てを説明できませんので，自習のための資料を含めて，復習して下さい

Rubyプログラムの実行

Rubyプログラムの記述と実行

Ruby プログラム①

- Ruby のプログラムは「式」の列（まとまり）である
- 例:
  - 0.3
  - 2 * 3 * 4
  - x = 5 * 6
- しかし、非常に困ったことに、ruby.exe は各式の値を計算するのだが、それを、我々にわかるように表示してはくれない
  - irb は、一行ずつ入力された式の値は表示してくれる。
- 不親切なようだが、必要に迫られてのこと。
  - 大きなプログラムでは、全ての式の値を表示されたら、ごみの山
- そこで、式の値を表示する特別な式が用意された。
  - それが、print や puts

Ruby プログラム②

- irb上で実行
  - 式（一行）で入力
  - 入力する度に実行され，式の値も表示
- Ruby コマンドで実行
  - プログラム（式のまとまり）として入力
  - 一行づつ実行されるが，式の値は出力しない
  - 出力するためには，print，puts で表示させなければならない

式と値（復習）①

- 式は（実行すると）値をもつ
  - 「式」は書かれた文字列
  - 「値」は（大抵の場合）数値
- 例：
  - 2+3 は 5 という値をもつ
  - x –y は (xが5、yが2ならば)3という値をもつ
  - x =4 は 4という値をもつ
  - x==3 は、xが値3を持っていれば、true という値をもつ
  - print(x)は、nilという値をもつ
- irb が出力するものは、式の値である
- 2個以上の式をセミコロンでつなぐと、最後の式の値が、全体の値となる

式と値（復習）②

```
irb(main):002:0> x=3
=> 3
irb(main):003:0> x==3
=> true
irb(main):004:0> x=4
=> 4
irb(main):005:0> x==3
=> false
irb(main):006:0> x=3;y=4
=> 4
irb(main):007:0> x=3;print(x)
3=> nil
```

二つ以上の式の場合，最後の式の値となる

今のところの Ruby プログラムの形

H:¥ruby¥sample41.rb

```
x = 0.1
n = 1; y = 0.3
z = if n==1 then x+y else x*y end
print( "x= ",x,", y= ",y,", z= ",z,"¥n" )
```

irb上で実行させた場合とRubyコマンドで実行させた場合の違いについて理解する

今のところの Ruby プログラムの形

プログラムとして実行した場合

```
H:¥ruby>ruby sample41.rb
x= 0.1, y= 0.3, z= 0.4
```

print文で指定された部分のみ出力される

Ruby プログラムの実行
> ruby プログラム名

今のところの Ruby プログラムの形

irb で一個ずつ実行した場合

```
irb(main):001:0> x = 0.1
=> 0.1
irb(main):002:0> n = 1; y = 0.3
=> 0.3
irb(main):003:0> z = if n==1 then x+y else x*y end
=> 0.4
irb(main):004:0> print( "x= ",x,", y= ",y,", z= ",z,"¥n" )
x= 0.1, y= 0.3, z= 0.4
=> nil
```

式

値

nil
存在しないことを表す特別な値

Ruby プログラムの作成

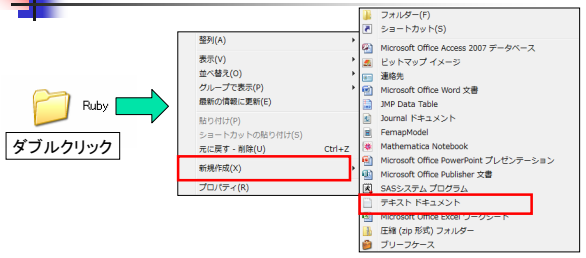
- 練習　sample41.rb を作成し，実行する

H:¥ruby¥sample41.rb

```
x = 0.1
n = 1; y = 0.3
z = if n==1 then x+y else x*y end
print( "x= ",x,", y= ",y,", z= ",z,"¥n" )
```

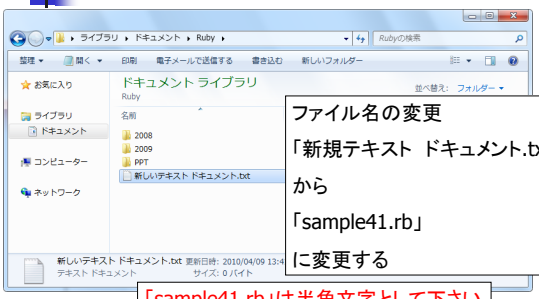
プログラムの書き方その①（復習）

プログラムの記述方法①



「Ruby」のフォルダー内で右クリック→「新規作成」→「テキストドキュメント」

プログラムの記述方法②

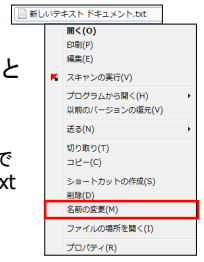


ファイル名の変更
「新規テキスト ドキュメント.txt」
から
「sample41.rb」
に変更する

「sample41.rb」は半角文字として下さい

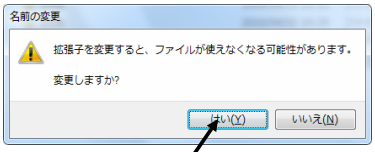
ファイル名の変更方法①

- ファイルを選択→右クリック →「名前の変更(M)」
- ファイルの名前を sample41.rb としてください。
  - 半角文字
  - 今回の講義では、拡張子(この例でいえば(.rb)は .rb でなくても(.txt でも)問題はおこらない(はず)。



ファイル名の変更方法②

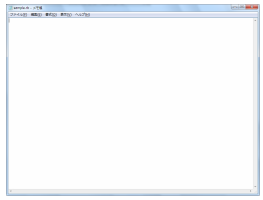
ファイル名を変更すると



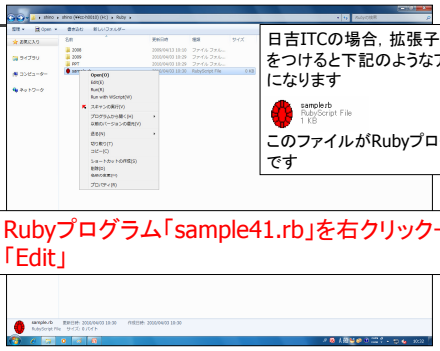
「はい(Y)」をクリック→ファイル名が変更される

エディターの起動①

- sample41.rbが  というアイコンであれば、ダブルクリックしてください。
  - メモ帳が立ち上がるでしょう、きっと。



エディターの起動②
（日吉ITCのPCの場合）

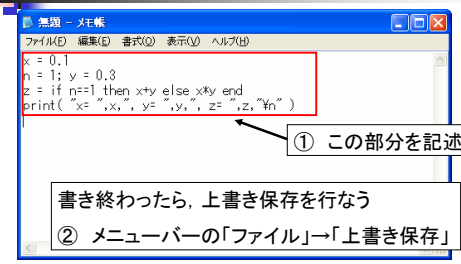


日吉ITCの場合，拡張子に「rb」をつけると下記のようなアイコンになります

このファイルがRubyプログラムです

Rubyプログラム「sample41.rb」を右クリック→「Edit」

プログラムの書き込み



① この部分を記述

書き終わったら，上書き保存を行なう

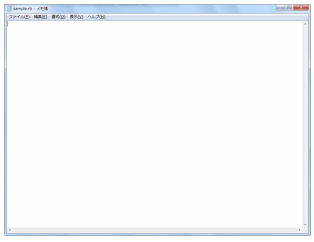
② メニューバーの「ファイル」→「上書き保存」

作成したファイルがRubyプログラム

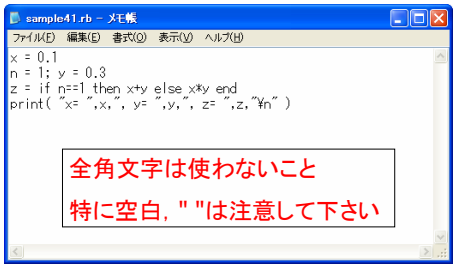
プログラムの書き方その②

エディターの起動

- 「スタートボタン」→「すべてのプログラム」→「アクセサリ」→「メモ帳」



作成したいプログラム



全角文字は使わないこと

特に空白，" "は注意して下さい

プログラムの保存①

- メニューバーの「ファイル」→「名前を付けて保存」

①「保存する場所」:→「コンピュータ」→「Hドライブ」→「Ruby」*

②「ファイルの種類」:すべてのファイル

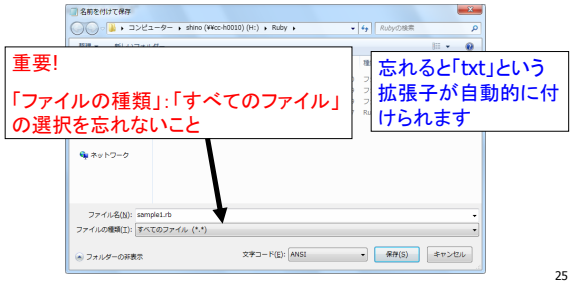
③「ファイル名」: sample41.rb

④「保存」をクリック

*ライブラリ→ドキュメント→Rubyでも可能

プログラムの保存②

- メニューバーの「ファイル」→「名前を付けて保存」



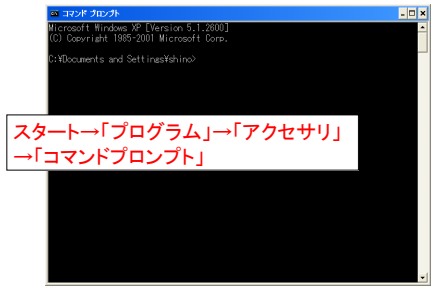
プログラムを保存する上での注意

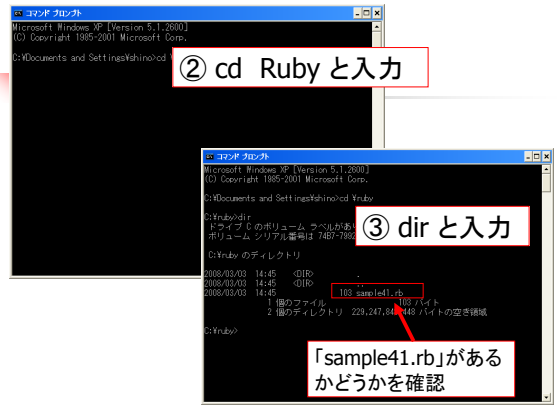
- この講義のプログラムはドキュメント（Hドライブ）の「Ruby」に保存すること
- rubyプログラムの名前の付け方
  - 拡張子に「rb」をつけること
    名前.rb （名前は英数字，自由につけてよい）
- 保存する際にはファイルの種類に「すべてのファイル」を選択すること
  - 選択しない場合，「txt」という拡張子が自動的に付きます

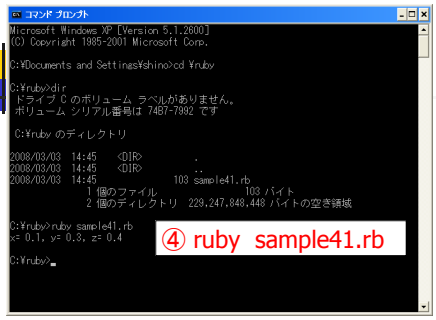
Rubyプログラムの実行

プログラムの実行方法

① コマンドプロンプトの起動





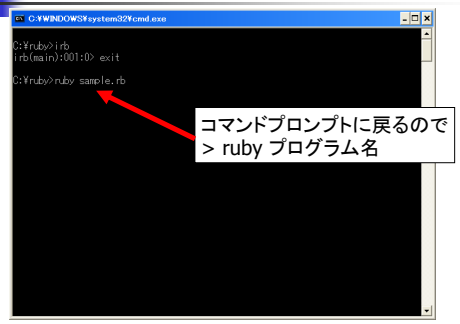


Rubyプログラムの実行方法

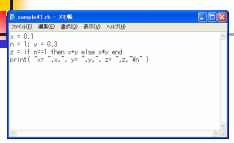
ruby Rubyプログラムのファイル名

irbを終了してrubyコマンドを実行したい場合

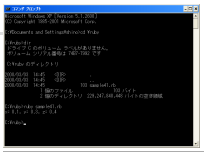
irbを終了してrubyコマンドを実行したい場合



プログラミングと実行



エディター（メモ帳）
プログラムを記述，訂正
保存を忘れずに

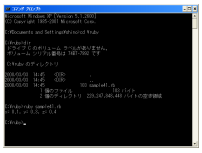


コマンドプロンプト
プログラムを実行（Ruby）

エラーが出た場合

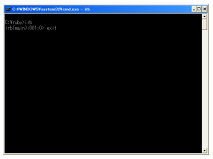
予想通りに動かない場合
追加したい場合

irbとコマンドプロンプト



コマンドプロンプト
エディタで書いたプログラムをrubyコマンドで実行

「irb」を入力
irbの画面へ



irb
rubyの式を一行ごとに実行

「exit」を入力
コマンドプロンプトの画面へ

プログラムが実行できない場合

- エラーメッセージを見て下さい
- 何行目にエラーがあるのか見つけて下さい
- どういう間違えであるのか、ヒントも表示されています

うまく実行できない場合①

- うまく実行結果がでない場合
  - エラーメッセージが表示されるので見ること

```
H:¥>ruby sample41.rb
ruby: No such file or directory -- sample41.rb (LoadError)
```

想定される原因:
ファイル sample41.rb がフォルダ H:¥Ruby にない

さらにその推定原因:
ファイル名または拡張子が違っている
別のフォルダにセーブした
→「dir」でファイルがあるかどうか確認してみる

うまく実行できない場合②

4行目に「invalid char」があるとエラーが表示

```
C:¥ruby>ruby sample41.rb
sample41.rb:4: Invalid char `¥201' in expression
sample41.rb:4: syntax error, unexpected $undefined,
expecting ')'
print(  "x= ",x,", y= ",y,", z= ",z,"¥n" )
      ^
sample41.rb:4: syntax error, unexpected ',', expecting $end
print(  "x= ",x,", y= ",y,", z= ",z,"¥n" )
```

ここにエラーがあると指摘している

→ 半角空白なのに全角空白を使用してしまっていた

うまく実行できない場合③

2行目に「invalid char」があるとエラーが表示

```
H:¥ruby>ruby sample41.rb
sample41.rb:2: Invalid char `¥201' in expression
sample41.rb:2: syntax error, unexpected tCONSTANT, expecting $end
n = 1; y = 0.3
     ^
```

ここにエラーがあると指摘している

→ 半角セミコロンなのに全角セミコロンを使用してしまっていた

うまく実行できない場合④

4行目に「syntax error」があるとエラーが表示

```
H:¥ruby>ruby sample41.rb
sample41.rb:4: syntax error, unexpected $undefined, expecting ')'
print( "x= ",x,", y= ",y,", z= ",z,"¥n" )
                                ^
sample41.rb:4: unterminated string meets end of file
```

ここにエラーがあると指摘している

想定される原因:
実は2番目のダブルクオートが全角になっている。

文字列リテラル（文字列定数）は半角のダブルクオートで始まり、半角のダブルクオートで閉じる必要があります

うまく実行できない場合⑤

3行目に「syntax error」があるとエラーが表示

```
H:¥ruby>ruby sample41.rb
sample41.rb:3: syntax error, unexpected tIDENTIFIER,
expecting kDO or '{' or '('

z = if n==1 then x+y x*y end
                    ^
```

ここにエラーがあると指摘している

想定される原因:
x+y の後に else が抜けている

うまく実行できない場合⑥

4行目に「syntax error」があるとエラーが表示

```
H:¥ruby>ruby sample41.rb
sample41.rb:4: syntax error, unexpected ')', expecting kDO_BLOCK
print( "x= ",x" , y= ",y,", z= ",z,"¥n" )
                                ^
```

ここにエラーがあると指摘している

想定される原因:
x の後にカンマが抜けている

プログラムが実行できない場合

- エラーメッセージが表示されるので注目
  - 何行目にエラーが表示されているのか
  - Invalid char → 不正な文字が含まれていないか
  - Syntax error → 構文的に誤っていないか
- プログラムを修正した後，テキストエディタ上で上書き保存を行ない，再実行する

プログラム構文上の大原則

- 括弧（広い意味での括弧です）は、開いたら、必ず閉じる。
  - Ruby での例外:「#」で始まるコメント（プログラムと関係のない書き込み）は、改行（そして改行のみ）が閉じる記号
- 複数種の括弧が混じるときには、互いに交錯してはならない
  - 例: { ([ ]) }
  - 誤例: {] { ([ )] }
- ダブルクオートも括弧の一種です。文字列を表します。普通の英語でもそうですが、開いたら必ず閉じる必要があります。
  - 例: "this is a book", "これはOK) }{ "
- Ruby では（親切というかおせっかいというか）あるべき括弧が省略できる場所があります

Ruby で括弧が省略できる場所①

- いくつかあるのだが、今回はここ！

```
irb(main):001:0> x=3
=> 3
irb(main):002:0> print( "x=" , x )
x=3=> nil
irb(main):003:0> print "x=" , x
x=3=> nil
```

print(“文字列”) ⇔ print “文字列”

puts(“文字列”) ⇔ puts “文字列”

Ruby で括弧が省略できる場所②

```
irb(main):040:0> print "without () "; print( "or with ()" )
without () or with ()=> nil
irb(main):041:0> puts "without () "; puts( "or with ()" )
without ()
or with ()
=> nil
```

括弧がない場合　括弧がある場合

式の復習

sample41.rb の説明

sample41.rb の説明

```
x = 0.1
n = 1; y = 0.3
z = if n==1 then x+y else x*y end
print( "x= ",x,", y= ",y,", z= ",z,"¥n" )
```

代入式

式は一行に一つだが，「;」でつなげることによって複数個の式を一行に書ける

条件式

出力

変数と型

- 変数:
  - コンピュータにおける変数とは、まず第一に、データを一時的に記憶しておく場所です。
  - そして、場所を区別するために名前をつけます。
  - そして、データには型があります。
- 型:
  - 許される演算によって決まります
  - これまでに出てきたのは、整数、浮動小数点数、文字列です。

データ型の変換

- データの右に、「.」をおき、その右に、「to_i」「to_f」「to_s」を記述した場合、そのデータ(値)を、それぞれ、整数型、浮動小数点数型、文字列への変換を行うことを意味する。
- データの変わりに変数にしても同様

```
irb(main):025:0> print 2
2=> nil
irb(main):026:0> print 2.0
2.0=> nil
irb(main):027:0> print 2.to_s
2=> nil
irb(main):028:0> print 2.to_f
2.0=> nil
irb(main):029:0> print "2".to_f
2.0=> nil
irb(main):030:0> print "2".to_i
2=> nil
```

```
irb(main):031:0> print 2.0.to_i
2=> nil
irb(main):032:0> print "2".to_i.to_f.to_s
2.0=> nil
irb(main):033:0> p "2".to_i.to_f.to_s
"2.0"
=> nil
irb(main):034:0> p "2".to_i.to_f
2.0
=> nil
```

p は、出力するとき、文字列と数値を区別して出力してくれます

自動型変換

- 整数型と浮動小数点数型が混在しているときには、浮動小数点数型に変換される。
  - これは、演算ごとに行われる。演算は左から順番に行われるため、すべての整数型データが浮動小数点型に変換されるわけではない。
  - 浮動小数点数型から整数型への変換は明示的に行わなければならない。

```
irb(main):038:0> 3.0/3 + 2/3
=> 1.0
irb(main):039:0> (3.0+2)/3
=> 1.66666666666667
```

- 文字列との自動変換は行われない

```
irb(main):040:0> "2" + 3
TypeError: can't convert Fixnum into String
        from (irb):40:in `+'
        from (irb):40
        from :0
```


式に関する注意①

- Ruby において「式」の意味する範囲は広い
  - 2 * x +3
  - x = y + x　　#注意　y+x を x に代入するということ
  - print x
  - if x==3 then y else y-1 end
- さらには、複数行にまたがる(またいで書いた方が分かりやすい)場合も、よく、ある。

式に関する注意②

```
if x==3 then y else y-1 end
```

同じ式

```
if x==3 then
    y
else
    y-1
end
```

複数行にまたいで書いた場合

整数型の算術演算子

演算子	用途	例	演算結果
+	加算	3+2	5
-	減算	4-2	2
*	乗算	2*2	4
/	除算	4/2	2
%	剰余	5%2	1
**	冪乗	5**2	25

浮動少数点数型の算術演算子

演算子	用途	例	演算結果
+	加算	3.1+2.2	5.3
-	減算	4.2-2.1	2.1
*	乗算	2.1*2.1	4.41
/	除算	4.2/2.1	2.0
%	剰余	5.0%2.1	0.8
**	冪乗	5.0**2.1	29.3654

代入演算子

- 「a = a 演算子 b」を「**a 演算子= b**」と記述することができる
- これを代入演算子という
  - 注意　=と演算子の間にスペースはおけない

```
a=20; b=10
a=a+b  ⇔ a+=b   # aは30を保持する
a=a-b  ⇔ a-=b   # aは10を保持する
a=a*b  ⇔ a*=b   # aは200を保持する
```


代入演算子の例

代入演算子	用途	例
+=	加算代入	a += b (a=a+b)
-=	減算代入	a -= b (a=a-b)
*=	乗算代入	a *= b (a=a*b)
/=	除算代入	a /= b (a=a/b)
%=	剰余代入	a %= b (a=a%b)
**=	冪乗代入	a **=b (a=a**b)

演算子の優先順位

高 ↑ ↓ 低

( ):括弧，( ): 引数
**:冪乗
+，－:符号（符号同一，符号反転）
*:乗算　/:除算　%:剰余
+:加算　－:減算
=:代入

- 同じレベルの演算子は左から順に計算する。従って
  - 2-4*3 は 2-(4*3)，　3/4*6 は (3/4)*6
  - なお、-a**-b は -(a**(-b))　（**が優先）
    a--+-b は a-(-(+(-b)))　（-と+は同じ）

条件式①

- 「if　論理式 then 式1 else 式2 end」という式がある
- 論理式がtrueならば式1を実行，falseならば式2を実行

```
if a > 0 then
        y = 3
else
        y = -3
end
```

a>0 ならば　y=3

違う場合は　y=-3

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

条件式②

if n==1 then z=x+y else z=x*y end

```
if n==1  then
   z=x+y
else
   z=x*y
end
```

n==1 ならば z=x+y

そうでなければ z=x*y

条件式②'

z = if n==1 then x+y else x*y end

```
z=
   if n==1  then
      x+y
   else
      x*y
   end
```

n==1 ならば x+y

そうでなければ x*y

条件式の値がzに代入

条件式③

nが2で割り切れる、かつnが3で割り切れるかどうか

```
if n % 2 == 0 and n % 3 == 0  then
   n += 2
else
   n *= 2
end
```


条件式④

else がない条件式もあります（詳しくは来週）

```
if n % 2 == 0 then
   print( n , "は2の倍数です¥n" )
end
```

標準出力

print を用いた出力

標準出力（print）

- x=3

① print("x=" , x , "です")

② print("x=" + x.to_s + "です")

③ print("x=は#{x}です")

出力式①

- print(変数)
- print("コメント")
  - コメント(文字列)を表示する場合は" "で囲む
- print("コメント" , 変数)
- print(変数1 , 変数2 , ・・・ , 変数n)
  - 変数，コメントを一行で表示したい場合は，「,」で区切る

出力式②

```
irb(main):042:0> x=3.1415
=> 3.1415
irb(main):043:0> print( "x=" , x )
x=3.1415=> nil
```

文字列　変数

```
irb(main):042:0> x=3.1415
=> 3.1415
irb(main):044:0> y=2
=> 2
irb(main):045:0> print( "x=" , x , " y=" , y )
x=3.1415 y=2=> nil
```

文字列　文字列　変数　変数

変数と変数，文字列は「,」で区切る

出力式③

```
irb(main):002:0> print( "x=" x )
SyntaxError: compile error
(irb):2: syntax error, unexpected tIDENTIFIER, expecting ')'
print( "x=" x )
          ^
        from (irb):2
        from :0
```

「,」がないとエラーが出力

出力式④

```
irb(main):004:0> print( x )
NameError: undefined local variable or method `x' for
main:Object
        from (irb):4
        from :0
```

x の値を設定(宣言)していないため
エラーが生じた

文字列連結演算子による出力

- print("コメント")
- "コメント"は文字列型の値

- 文字列連結演算子
  - 「+」は、文字列を連結する役割がある
  - 「＊」は、文字列を繰り返す役割がある

文字列連結演算子①

- i=3
- print("iは " , i , " です")

- print("iは " + i.to_s + " です")
  - " "で囲まれた文字列または変数の文字列値を連結する
  - 変数を文字列とするには、n.to_s のように変数名の後ろに、「.to_s」をつける
- print("にわ" *4 + "とりがいる")

文字列連結演算子②

```
irb(main):042:0> x=3.1415
=> 3.1415
irb(main):043:0> print( "x=" , x )
x=3.1415=> nil
```

文字列連結演算子を用いた場合

```
irb(main):046:0> x=3.1415
=> 3.1415
irb(main):047:0> print( "x="+x.to_s )
x=3.1415=> nil
```

"x=" + x.to_s

小数 x を文字列に変換し + で連結

文字列連結演算子③

```
irb(main):042:0> x=3.1415
=> 3.1415
irb(main):044:0> y=2
=> 2
irb(main):045:0> print( "x=" , x , " y=" , y )
x=3.1415 y=2=> nil
```

文字列連結演算子を用いた場合

```
irb(main):048:0> x=3.1415
=> 3.1415
irb(main):049:0> y=2
=> 2
irb(main):050:0> print( "x="+x.to_s+" y="+y.to_s )
x=3.1415 y=2=> nil
```


文字列連結演算子④

```
irb(main):001:0> x = "abc"
=> "abc"
irb(main):002:0> y = "xyz"
=> "xyz"
irb(main):003:0> print( " x= " , x , " y= " , y )
 x= abc y= xyz=> nil
```

文字列連結演算子を用いた場合

```
irb(main):001:0> x = "abc"
=> "abc"
irb(main):002:0> y = "xyz"
=> "xyz"
irb(main):003:0> print( " x= " + x + " y= " + y )
 x= abc y= xyz=> nil
```

変数x, yともに文字列型

文字列連結演算子⑤

```
irb(main):001:0> x="3.1415"
=> "3.1415"
irb(main):002:0> print( " x= " , x )
 x= 3.1415=> nil
irb(main):003:0> print( " x= " + x )
 x= 3.1415=> nil
irb(main):004:0> print( " x= " , x.to_f )
 x= 3.1415=> nil
irb(main):005:0> print( " x= " + x.to_f )
TypeError: can't convert Float into String
        from (irb):5:in `+'
        from (irb):5
        from :0
```

+演算子
次の変数は文字列型のみ

Rubyの工夫①

```
irb(main):002:0> n=123; print( "nの値は " , n , " です" )
nの値は 123 です=> nil
```

同じことを文字列で書く場合

```
irb(main):003:0> n=123; print( "n の値は #{n} です" )
n の値は 123 です => nil
```

- Ruby では文字列（これは定数です）中に、変数を書くことができる。
- 上記のように、文字列中に #{ と } で挟んだ式を書けばよい

Rubyの工夫②

- 文字列に、式を書くこともできる。

```
irb(main):018:0* n=123
=> 123
irb(main):019:0> msg="nの2倍は #{n*2} です"; print( msg )
nの2倍は 246 です=> nil
```

msg="nの2倍は #{n*2} です";

```
irb(main):005:0> n=123; print( "nの2倍は #{n*2} です" )
nの2倍は 246 です=> nil
```

print("nの2倍は #{n*2} です")

Rubyの工夫③

```
irb(main):004:0> n=123;
irb(main):005:0* print( "#{n} は #{if n%2==0
irb(main):006:0"  then "even" else "odd" end} です" )
123 は odd です => nil
```

```
"#{n} は #{
        if n%2==0 then
                "even"
        else
                "odd"
        end
} です"
```

条件式も書くことが可能

Rubyの工夫④

前のページと同じ式です

```
irb(main):004:0> n=123
=> 123
irb(main):005:0> msg="#{n} は
 #{if n%2==0 then "even" else "odd" end} です¥n"
=> "123 ¥202¥315 odd ¥202¥305¥202¥267¥n"
irb(main):006:0> print( msg )
123 は odd です
=> nil
```

文字列中に式を記述

改行①

- print("x=" , x)
- print("x=" , x , "¥n")

```
irb(main):006:0> x=2
=> 2
irb(main):007:0> print( "x=" , x )
x=2=> nil
irb(main):008:0> print( "x=" , x , "¥n" )
x=2
=> nil
```

改行されない

改行される

改行②

- 「¥n」
  - 改行文字
  - 一行改行される

```
irb(main):021:0> x=2;y=3
=> 3
irb(main):022:0> print( "x=" , x , " y=" , y )
x=2 y=3=> nil
irb(main):023:0> print( "x=" , x , "¥n y=" , y )
x=2
 y=3=> nil
```

改行される

改行③

```
irb(main):010:0> msg="¥n"
=> "¥n"
irb(main):011:0> print( msg )

=> nil
irb(main):012:0> print( msg*5 )


=> nil
```

1回改行

5回改行

その他の出力①

- puts(変数)
- puts("コメント")
- puts("コメント" , 変数)

- printと何が違うでしょうか？

その他の出力②

- p というのもあります

```
irb(main):031:0> x=Math::PI
=> 3.14159265358979
irb(main):032:0> p x
3.14159265358979
=> nil
irb(main):035:0> x="abcd"
=> "abcd"
irb(main):036:0> p x
"abcd"
=> nil
```

文字列と数値を区別して表示

文字列は""でくくられる

出力フォーマット①

- より凝って出力したい場合には、
  - printf("%+d", 1)

  のように、printf を用います
- " … " の部分がフォーマット(書式)です。
- 詳細は、

  http://www.ruby-lang.org/ja/man/
  ⇒「目次」中の「付録」中の sprintfフォーマット

出力フォーマット②

- %d
- printf("%d" , x)
  - xを10進整数で表示する
- %f
- printf("%f" , x)
  - xを10進浮動小数点数で表示する

出力フォーマット③

- %x
- printf("%x" , x)
  - xを16進整数で表示する
- %s
- printf("%s" , x)
  - xを文字列で表示する

出力フォーマット④

```
irb(main):010:0> x=123
=> 123
irb(main):011:0> printf( "%d" , x )
123=> nil
irb(main):012:0> printf( "%f" , x )
123.000000=> nil
irb(main):013:0> printf( "%x" , x )
7b=> nil
irb(main):014:0> printf( "%s" , x )
123=> nil
```

整数

小数

16進数

文字列

出力フォーマット④'

```
irb(main):005:0> x=123.45
=> 123.45
irb(main):006:0> printf( "%d" , x )
123=> nil
irb(main):007:0> printf( "%f" , x )
123.450000=> nil
irb(main):008:0> printf( "%x" , x )
7b=> nil
irb(main):009:0> printf( "%s" , x )
123.45=> nil
```

整数

小数

16進数

文字列

出力フォーマット⑤

- 桁数の指定
- printf("%5d" , x)
  - xを5桁の整数で表示する
- printf("%5.2f" , x)
  - xを5桁の小数，小数点以下を2桁で表示する

出力フォーマット⑥

- 桁数の指定
- printf("%10d" , x)
  - xを10桁の整数で表示する
  - 指定した桁数に満たない箇所は空白で表示
- printf("%10.5f" , x)
  - xを10桁の小数，小数点以下を5桁で表示する

出力フォーマット⑥

- 桁数の指定
- printf("%-5d" , x)
  - xを左詰で5桁の整数で表示する
- printf("%-5.2f" , x)
  - xを左詰めで5桁の小数，小数点以下を2桁で表示する

出力フォーマット⑦

```
irb(main):019:0> x=123
=> 123
irb(main):020:0> printf( "%5d" , x )
  123=> nil
irb(main):021:0> printf( "%-5d" , x )
123  => nil
```

5桁で表示

左詰め5桁で表示

```
irb(main):022:0> x=3.1415
=> 3.1415
irb(main):023:0> printf( "%5.2f" , x )
 3.14=> nil
irb(main):024:0> printf( "%-5.2f" , x )
3.14 => nil
```

5桁，小数点以下2桁で表示

左詰めで5桁，小数点以下2桁で表示

出力フォーマット⑧

```
irb(main):012:0> x=Math::PI
=> 3.14159265358979
irb(main):013:0> printf( "%5.2f" , x )
 3.14=> nil
irb(main):014:0> printf( "%10.2f" , x )
      3.14=> nil
irb(main):015:0> printf( "%10.5f" , x )
   3.14159=> nil
irb(main):016:0> printf( "%-10.5f" , x )
3.14159    => nil
irb(main):017:0> printf( "%10.8f" , x )
3.14159265=> nil
```

10桁，小数点以下2桁で表示

10桁，小数点以下5桁を左詰で表示

出力フォーマット⑨

```
irb(main):001:0> x=3
=> 3
irb(main):002:0> y=3**2*Math::PI
=> 28.2743338823081
irb(main):03:0> print( "x=" , x , " y= " , y )
x=3 y= 28.2743338823081=> nil
irb(main):04:0> printf( "x=%d y=%8.4f¥n" , x , y )
x=3 y= 28.2743
=> nil
irb(main):05:0> printf( "x=%5.2f y=%10.3f¥n" , x , y )
x= 3.00 y=    28.274
=> nil
irb(main):06:0> printf( "x=%d y=%d" , x , y )
x=3 y=28=> nil
```

xは整数，yは小数

printで出力

xは整数，yは小数で出力

xは小数，yは小数で出力

xは整数，yは整数で出力

標準入力

gets による入力

標準入力①

- キーボードからの入力
- gets

```
irb(main):018:0> gets
34
=> "34¥n"
```

入力

① getsと打つ
② キーボードから入力（最後に改行する）

```
irb(main):019:0> gets
abcd
=> "abcd¥n"
```

入力した値は文字列として処理される

最後に改行「¥n」が入る

標準入力②

入力

```
irb(main):024:0> a=gets
3.1415
=> "3.1415¥n"
irb(main):025:0> p a
"3.1415¥n"
=> nil
```

変数a に入力した値を代入
変数a は文字列型
最後に改行文字が入る

```
irb(main):028:0> x=gets
abcd
=> "abcd¥n"
irb(main):029:0> p x
"abcd¥n"
=> nil
```

変数x に入力した値を代入
変数x は文字列型
最後に改行文字が入る

標準入力③

- getsによる標準入力
- 文字列型で入力される
- 末尾に"¥n"（改行）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④

入力

```
irb(main):041:0> x=gets
3.1415
=> "3.1415¥n"
irb(main):042:0> x.chop
=> "3.1415"
irb(main):043:0> x.chop.to_f
=> 3.1415
irb(main):044:0> 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⑤

入力

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

chop で最後の一文字（改行）を削除

プログラム中で入力する

- 2回読んで、和を求めるプログラムです

G:¥Ruby¥sample0306.rb

```
print( "一番目の数値を入力してください: " )
line = gets.chop  # gets は読み込んだ一行を値とします。
                  # chop は最後の文字(改行文字)を取り除きます。
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 }" )
```

```
G:¥Ruby>ruby sample0306.rb
一番目の数値を入力してください: 123
二番目の数値を入力してください: 456
123.0 + 456.0 = 579.0
G:¥Ruby>
```


キーボードからの入力①

- line = gets.chop
- 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
  - 最後の一文字を削除する
- 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

キーボードからの入力②

```
line = gets.chop
x = line.to_i
print( x , "¥n" )

x = line.to_f
print( x , "¥n" )

x = line.to_s
print( x , "¥n" )
```

整数に変換し，表示

小数に変換し，表示

文字列に変換し，表示

キーボードからの入力④

- 前のページと同じ出力をするプログラムです

```
x = gets.chop.to_i
print( x , "¥n" )

x = gets.chop.to_f
print( x , "¥n" )

x = gets.chop.to_s
print( x , "¥n" )
```

整数に変換し，表示

小数に変換し，表示

文字列に変換し，表示

```
C:¥Ruby>ruby sample.rb
24
24
56
56.0
12
12
```


キーボードからの入力⑤

- chop
  - 最後の一文字を削除する
- chomp
  - 最後の一文字が改行がある場合のみ削除

キーボードからの入力⑤'

```
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

キーボードからの入力⑥

```
a = gets.chomp.to_i
b = gets.chomp.to_i
c = gets.chomp.to_i
average = (a+b+c)/3.0
print( "平均は" , average , "です¥n" )
```

3個の整数を読み込んで平均を出力

```
C:¥Ruby>ruby sample.rb
34
5
56
平均は31.6666666666667です
```


キーボードからの入力⑦

```
a = gets.chomp
b = gets.chomp

if a == b then
        print( a , " と " , b , "は同じです" )
else
        print( a , " と " , b , "は違います" )
end
```

2個の文字列を読み込んで比較

```
C:¥Ruby>ruby sample.rb
abc
xyz
abc と xyzは違います
```


練習問題

練習問題①～③

練習問題①

- 整数x,y を読み込み大小を判定するプログラムを書きなさい

```
C:¥Ruby>ruby sample.rb
31
12   入力
31 は 12 以上です
```

```
C:¥Ruby>ruby sample.rb
12
56   入力
56 は 12 以上です
```

練習問題②

- 円の半径を整数値としてキーボードから読み込み，円周および面積を少数値として求め，表示するプログラムを書きなさい．

```
>ruby en.rb
円の半径は?
10
半径 10 の円周は 62.8318530717959 面積は 314.159265358979 です
```

練習問題③

- 整数Nを入力し，3の倍数，5の倍数かどうかを調べるプログラムを作成しなさい．

```
C:¥Ruby>ruby sample.rb
12
12は3の倍数です

C:¥Ruby>ruby sample.rb
15
15は3の倍数です
15は5の倍数です
```

練習問題④

- 簡単で暇な人はこちらを行ってください。
- 100マス計算ならぬ1マス計算を作ってください
- ランダムな問題を作るには、擬似乱数を使います。
  - ヒント: rand() とすれば 0から1までの一様乱数が得られます。
    - rand(34) とすると 0 から 33 までの整数値がランダムに得られます

```
G:¥Ruby>ruby sample0310.rb
1 + 5 = 6
正解！
7 + 9 = 15
残念
9 + 3 =

G:¥Ruby>
```

「正解！」か「残念」かの印字は下記のようにすればできます。勿論 入力値 と 計算値 のところは、ちゃんと書くのですよ

```
print( if 入力値 == 計算値 then "正解！" else "残念" end )
```

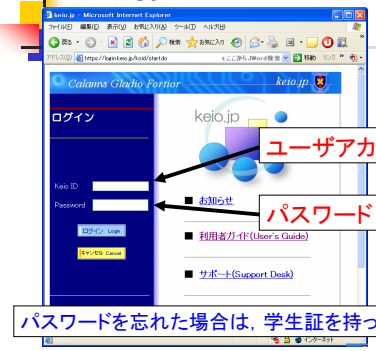

提出方法

- keio.jp 上から提出して下さい
- 練習問題のプログラムを一つのテキストファイル（MS-Wordでもよい）にまとめて提出して下さい
- （時間があったら、プログラムの説明も少し書いて下さい）

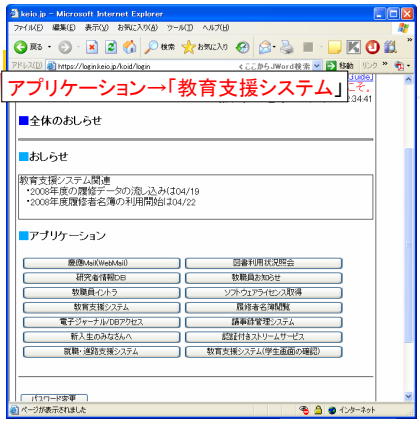
慶應義塾共通認証システム（keio.jp）

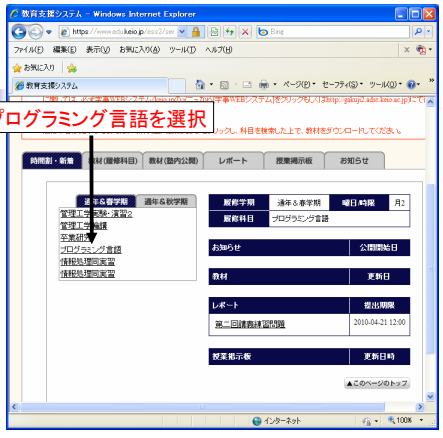
- WEBメール
- 図書館利用状況
- 教育支援システム
- お知らせ
- 各種サービス

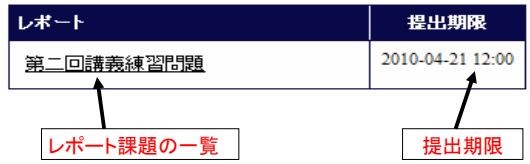
keio.jp へのログオン



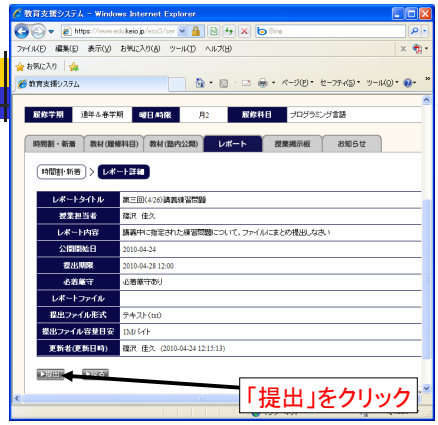
パスワードを忘れた場合は，学生証を持って日吉ITCの受付へ

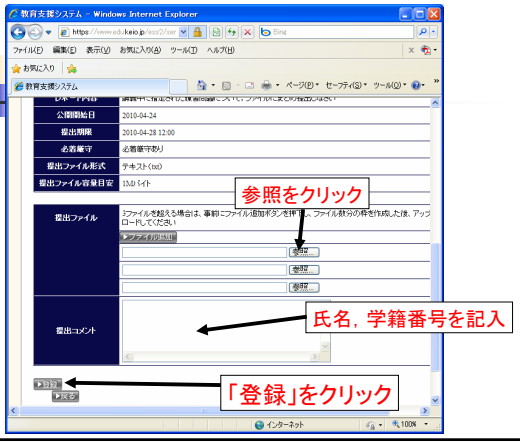


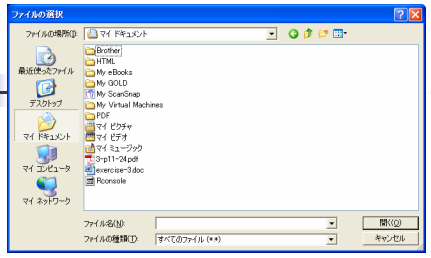




- レポート課題の一覧から，提出する課題のタイトルをクリック
- 提出期限を過ぎると提出は不可能







- 提出するファイルを選択
- 指定されたファイルの形式（MS-Word，テキスト文書など）を確認して提出して下さい