

プログラミング言語

第3回 4月28日

担当: 篠沢 佳久
櫻井 彰人

平成20年度: 春学期

1

本日の内容

- ▲ interactive Ruby の使い方(復習)
- ▲ 変数
- ▲ 式の記述方法
- ▲ 第一回レポート課題

2

interactive Ruby

第二回の資料より

3

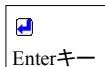
まずは: irb

- ▲ プログラムは、多数の行で構成されている。
- ▲ しかし、中には、実行したいことが、一行で書けてしまうこともある
- ▲ Ruby には、この「一行プログラム」の実行ができるツールが提供されている。
 - 実は、複数行に渡っても実行できる、優れたもの
- ▲ それが、irb (interactive Ruby)

4

irb の起動①

- ▲ コマンドプロンプトで
irb 
と入力

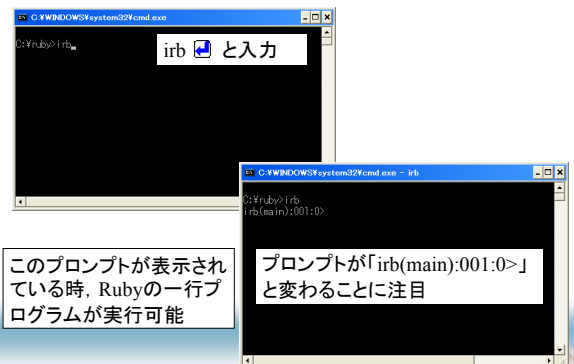


Enterキー

- ▲ あとは interactive (会話的に)
 - 「コマンド」の入力
 - 実行結果の出力が無限に行われます
- ▲ 終了したいときには
exit 
と入力

5

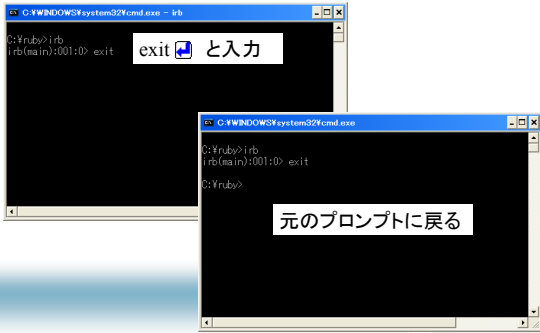
irb の起動②



このプロンプトが表示されている時, Rubyの一行プログラムが実行可能

プロンプトが「irb(main):001:0>」と変わることに注目

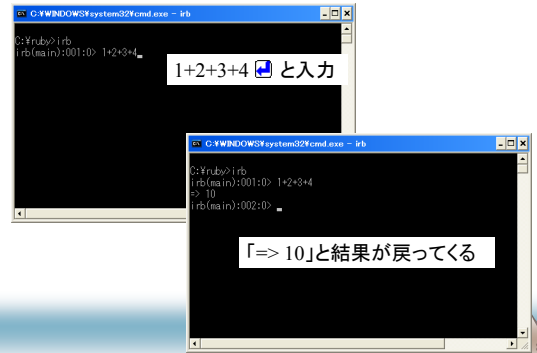
irbの終了



```
C:\WINDOWS\system32\cmd.exe - irb
C:\Ruby> irb
irb(main):001:0> exit
exit と入力

C:\WINDOWS\system32\cmd.exe - irb
C:\Ruby> irb
irb(main):001:0> exit
irb(main):002:0>
元のプロンプトに戻る
```

irb での入力方法



```
C:\WINDOWS\system32\cmd.exe - irb
C:\Ruby> irb
irb(main):001:0> 1+2+3+4
1+2+3+4 と入力

C:\WINDOWS\system32\cmd.exe - irb
C:\Ruby> irb
irb(main):001:0> 1+2+3+4
=> 10
irb(main):002:0>
「=> 10」と結果が戻ってくる
```

変数

変数と代入式

9

変数とは

- ▲ 今日覚える重要なことに「変数」があります。
- ▲ 変数は、中学から代数で慣れ親しんだ変数とそっくりな概念です。そっくりですが、随分違いもあります。よく注意してください。
- ▲ 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(識別子)をつけます。
- ▲ Rubyの変数の型は記憶しているデータの型で決まります(重要!)
 - Rubyの変数は、単に、場所の名前と思えばよい

10

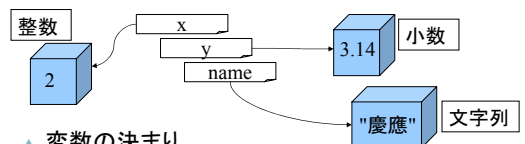
変数①

- ▲ $x = 2$
 - 変数 x は整数型で値は2
- ▲ $y = 3.14$
 - 変数 y は小数型で値は3.14
- ▲ $name = \text{"慶應"}$
 - 変数 $name$ は文字列型で値は"慶應"

11

変数②

- ▲ コンピュータにデータを記憶させる機能のこと
 - 変数: 名前を作って、データが入った箱を区別するイメージ



- ▲ 変数の決まり
 - 変数に「名前」(識別子)をつける
 - 変数に値を代入する、とは、割当てる or 割付けること

12

変数が使えるためには

▲ 変数を使うための準備

- 変数に「名前」(識別子)をつける
 - 名前がないと、何もできない。
 - 名前は使えばよい
 - 「出生届け」はいらない
- 変数を使えるようにする
 - 値を代入すればよい

13

識別子

▲ 変数につける名前のこと

- 通常、英字・数字・アンダースコアを用いる
例: num
- 数字で始めることはできない
例: 1num
- 大文字と小文字は区別される
例: num と Num は区別される
- Rubyで使う予約語(keyword)は使えない
例: ~~class~~ ~~return~~ ~~then~~ ~~else~~

14

予約語の一覧

```
BEGIN class ensure nil self when END
def false not super while alias defined?
for or then yield and do if redo true
begin else in rescue undef break elsif
module retry unless case end next
return until __FILE__ __LINE__
```

15

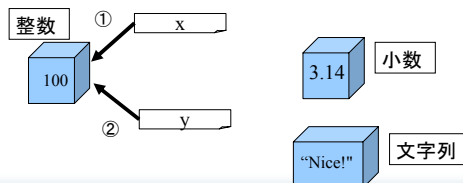
変数への値の代入

- ▲ 「識別子 = 値」で変数に値を代入
- ▲ 数学のイコールとは意味が違うことに注意

```
irb(main):005:0> x = 100
=> 100
irb(main):005:0> y = 100
=> 100
irb(main):006:0> x = "Nice !"
=> "Nice !"
irb(main):007:0> x = 3.14
=> 3.14
irb(main):008:0>
```

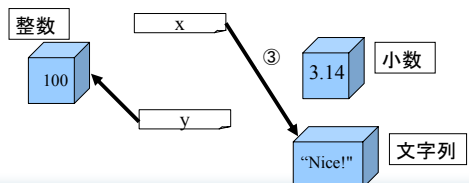
16

```
irb(main):005:0> x = 100 ①
=> 100
irb(main):005:0> y = 100 ②
=> 100
irb(main):006:0> x = "Nice !"
=> "Nice !"
irb(main):007:0> x = 3.14
=> 3.14
irb(main):008:0>
```



17

```
irb(main):005:0> x = 100
=> 100
irb(main):005:0> y = 100
=> 100
irb(main):006:0> x = "Nice !" ③
=> "Nice !"
irb(main):007:0> x = 3.14
=> 3.14
irb(main):008:0>
```

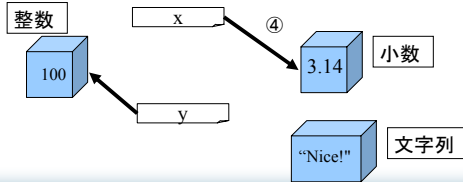


18

```

irb(main):005:0> x = 100
=> 100
irb(main):005:0> y = 100
=> 100
irb(main):006:0> x = "Nice !"
=> "Nice !"
irb(main):007:0> x = 3.14 ④
=> 3.14
irb(main):008:0>

```



19

識別子の注意点

```

irb(main):035:0> else = 5
SyntaxError: compile error
(irb):35: syntax error, unexpected kELSE
else = 5
^

```

elseは予約
語なので利
用できない

```

from (irb):35
from :0
irb(main):036:0> Num = 3
=> 3
irb(main):037:0> num = 13
=> 13
irb(main):038:0> num * Num
=> 39

```

大文字, 小
文字は区別

20

変数の表示

- ▲ 変数に入っている値を表示させたい場合
- ▲ print(変数名) or print 変数名
- ▲ puts(変数名) or puts 変数名

```

irb(main):001:0> x=3
=> 3
irb(main):002:0> print(x)
3=> nil
irb(main):003:0> puts(x)
3
=> nil
irb(main):004:0>

```

xに3を代入

xの中身を表示

21

変数の例①

```

irb(main):056:0> x=25
=> 25
irb(main):057:0> print( "xの値は ",x," です" )
xの値は 25 です=> nil
irb(main):058:0> x=30
=> 30
irb(main):059:0> print( "xの値は ",x," に変わりました" )
xの値は 30 に変わりました=> nil

```

変数の中身は上書きされる

```

print( "コメント" )
print( "コメント", 変数名 )
print( "コメント1", 変数名 , "コメント2" )

```

22

変数の例②

```

irb(main):056:0> x=25
=> 25
irb(main):057:0> print( "xの値は ",x," です" )
xの値は 25 です=> nil

```

一行で書く場合

```

irb(main):060:0> x=20; print( "xの値は ",x," です" )
xの値は 20 です=> nil

```

二つの式を一行で書きたい場合は,「;」(セミ
コロン)で区切る

23

変数の例③

```

irb(main):058:0> x=30
=> 30
irb(main):059:0> print( "xの値は ",x," に変わりました" )
xの値は 30 に変わりました=> nil

```

一行で書く場合

```

irb(main):061:0> x=30; print( "xの値は ",x," に変わりました" )
xの値は 30 に変わりました=> nil

```

24

変数の例④



```
irb(main):056:0> x=25
=> 25
irb(main):057:0> print( "xの値は ",x," です" )
xの値は 25 です=> nil
irb(main):058:0> x=30
=> 30
irb(main):059:0> print( "xの値は ",x," に変わりました" )
xの値は 30 に変わりました=> nil
```



一行で書く場合

```
irb(main):062:0> x=20; print( "xの値は ",x," です" );
x=30; print( "xの値は ",x," に変わりました" )
xの値は 20 ですxの値は 30 に変わりました=> nil
```

25

変数の実例①

▲ 変数はあると便利

```
# sample22.rb
puts "僕の名前は、寿限無寿限無五劫の擦り切れ海砂利水魚の水行末雲来末風来末食う寝る処に住む処やぶら小路の藪柑子パイポパイポパイポのシューリンガンシューリンガンのグーリンダイグーリンダイのポンポコピーのポンポコナーの長久命の長助だ〜い"
puts "え、君の名前は、寿限無寿限無五劫の擦り切れ海砂利水魚の水行末雲来末風来末食う寝る処に住む処やぶら小路の藪柑子パイポパイポパイポのシューリンガンシューリンガンのグーリンダイグーリンダイのポンポコピーのポンポコナーの長久命の長助かい？ 長いなあ"
```



変数 name に値を代入しておく

```
# sample23.rb
name= "寿限無寿限無五劫の擦り切れ海砂利水魚の水行末雲来末風来末食う寝る処に住む処やぶら小路の藪柑子パイポパイポパイポのシューリンガンシューリンガンのグーリンダイグーリンダイのポンポコピーのポンポコナーの長久命の長助"
puts "僕の名前は、" + name + "だ〜い"
puts "え、君の名前は、" + name + "かい？ 長いなあ"
```

文字列を足し算した後、出力

変数の実例②

▲ 変数はあると便利

```
# sample22.rb
puts "僕の名前は、パブロ・ディエゴ・ホセ・フランシスコ・ド・ポール・ジャン・ネポムチェーノ・クリスパン・クリスピアノ・ド・ラ・ンチシュ・トリニダット・ルイス・イ・ピカソだ〜い"
puts "え、君の名前は、パブロ・ディエゴ・ホセ・フランシスコ・ド・ポール・ジャン・ネポムチェーノ・クリスパン・クリスピアノ・ド・ラ・ンチシュ・トリニダット・ルイス・イ・ピカソかい？ 長いなあ"
```



変数 name に値を代入しておく

```
# sample23.rb
name= "パブロ・ディエゴ・ホセ・フランシスコ・ド・ポール・ジャン・ネポムチェーノ・クリスパン・クリスピアノ・ド・ラ・ンチシュ・トリニダット・ルイス・イ・ピカソ"
puts "僕の名前は、" + name + "だ〜い"
puts "え、君の名前は、" + name + "かい？ 長いなあ"
```

27

変数の実例③

▲ 変数はあると便利

```
# sample22.rb
puts "My name is Pablo Diego José Santiago Francisco de Paula Juan Nepomuceno Crispín Crispiniano de los Remedios Cipriano de la Santísima Trinidad Ruiz Blasco y Picasso"
puts "What? Is your name Pablo Diego José Santiago Francisco de Paula Juan Nepomuceno Crispín Crispiniano de los Remedios Cipriano de la Santísima Trinidad Ruiz Blasco y Picasso ? It's too long."
```



変数 name に値を代入しておく

```
# sample23.rb
name= "Pablo Diego José Santiago Francisco de Paula Juan Nepomuceno Crispín Crispiniano de los Remedios Cipriano de la Santísima Trinidad Ruiz Blasco y Picasso"
puts "My name is " + name + "."
puts "What? Is your name " + name + "? It's too long."
```

28

いろいろなデータを一つの変数に



この行は、前の行からの続きであることを表している。
Ruby は行が終わるのを待っている

このセミicolonがあると、Ruby は、まだ行が続くと考え

```
irb(main):063:0> x=20;
irb(main):064:0> print( "x is now ",x );
irb(main):065:0> x=3.14;
irb(main):066:0> print( " but x is now ",x );
irb(main):067:0> x="Hi!";
irb(main):068:0> print( " but now x is ",x );
x is now 20 but x is now 3.14 but now x is Hi! => nil
```

ここにセミicolonはないし、前に書いてある式も一ままとまりになっているので、Ruby は、これで終わり！と考える

29

再び: データ型

▲ コンピュータ内部ではどう表現しているのだろうか？どう演算しているのだろうか？

- 詳しくは後の講義の楽しみ。

▲ 整数型の場合

- 整数表現。2進数。8/16/32/(64)ビットを一まとめにして記憶している。

30

注：有限性

- ▲ 我々は有限なものしか認識できない(かどうかは、哲学に任せよう)。
- ▲ コンピュータで取り扱えるものは、有限のみ。
- ▲ 循環小数はどう表現しよう？
 - ・ 分数で表せば有限なんだけどね。それは別の話。
- ▲ 無理数はどう表現しよう？
 - 有限桁数の近似表現で我慢しよう
- ▲ 非常に大きな整数はどう表現しよう
 - できるだけ表現しよう(Rubyの場合)

31

整数型の可能性①

- ▲ Rubyの整数型は、非常に大きな数を表現できます。
- ▲ まあ、皆さんの常識からすれば、これは当たり前なのですが、コンピュータ言語の世界では、常識ではありません。
- ▲ まずは、試してみましょう

```

irb(main):018:0> 2 ** 1000
=>10715086071862673209944625504906000181056140481170553360
74437503883703510511249361224931983788156958851275946729
175531468251871458569231404359847574569857480393456777
4824230985421074625062371418779541821530464749835819412
6739876755916543646077062914571196477686542167660429831
652624386837205968069376

```

2 ** 10000 も試してみましょう

32

整数型の可能性②

階乗を求めるプログラム

これは、後ほど

```
irb(main):079:> def factorial(n) (1..n).inject{|a,b| a*b} end  
=> nil  
irb(main):080:> factorial(10)  
=> 3628800  
irb(main):081:> factorial(100)  
=>  
933262154439441526816992388562667004907159682643816214685929638  
105177599993229915608941463976156518286253697208272327582511852  
1991686400000000000000000000000000
```

33

進んで考えたい人のために

階乗を求めるプログラム

```
irb(main):002:0> def factorial(n) if n<=1 then 1 else n*factorial(n-1) end end
=> nil
irb(main):003:0> factorial(10)
=> 3628800
```

```
irb(main):004:0> def factorial(n) r=1; (2..n).each{ |i| r*=i }; r end
=> nil
irb(main):005:0> factorial(10)
=> 3628800
```

```
irb(main):007:0> def factorial(n) r=1; for i in 2..n; r*=i end; r end
=> nil
irb(main):008:0> factorial(10)
=> 3628800
```

34

暇な人向けの問題

階乗を求めるプログラム

```
irb(main):002:0> def fct(n,f=1) if n<=1 then f else fct(n-1,n*f) end end
=> nil
irb(main):003:0> fct(10)
=> 3628800
```

```
irb(main):002:0> def factorial(n) (1..n).inject{|x,y| x*y} end
=> nil
irb(main):003:0> factorial(10)
=> 3628800
```

```
irb(main):002:0> def factorial(n) eval( [*(1..n)].join("*") ) end
=> nil
irb(main):003:0> factorial(10)
=> 3628800
```

34

浮動小数点数型

- ▲ 浮動小数点数 floating point number
- ▲ コンピュータのなかで、整数以外の有限桁の数を表現する工夫です。
物理や化学で絶対値の大きな数や小さな数を表す工夫と同じです。

- 6.0221367×10^{23} って何ですか？

▲ **整数** × 2 **整数** で表現します。

仮数部

指数部

- ▲ 仮数部は有限桁です。最近は、53ビットが多い。
- ▲ 指数部も有限桁です。

浮動小数点型 Ruby の場合

- ▲ 10進数で約15桁の精度があります。
(C や Java の double と同じです)

これは、後ほど。
小数点以下50桁印字する

```
irb(main):009:0> printf("%.50f\n", 1.0/3)
0.3333333333333333100000000000000000000000000000000000000
=> nil
```

37

浮動小数点数の指数の制限

- ▲ 指数だって、有限の長さしかない

```
irb(main):012:0> 10.0**308
=> 1.0e+308
irb(main):013:0> 10.0**309
=> Infinity
```

1.0e+308
1.0 × 10³⁰⁸

```
irb(main):014:0> 10.0**(-323)
=> 9.88131291682493e-324
irb(main):015:0> 10.0**(-324)
=> 0.0
```

e-308
10⁻³⁰⁸

38

変数の型変換①

- ▲ 変数においても型変換が可能

x=3	整数型	x=3.1415	小数型
x.to_f	小数へ変換	x.to_i	整数へ変換
x.to_s	文字列へ変換	x.to_s	文字列へ変換

39

```
irb(main):019:0> x=3
=> 3
irb(main):020:0> x.to_f
=> 3.0
irb(main):021:0> x.to_s
=> "3"
irb(main):022:0> x
```

型変換しても、変数の中身を変えているわけではない

```
irb(main):033:0> x=3.141
=> 3.141
irb(main):034:0> x.to_i
=> 3
irb(main):035:0> x.to_s
=> "3.141"
irb(main):036:0> x
=> 3.141
```

40

変数の型変換②

- ▲ 文字列型の変数においても型変換が可能

x="3.14"	文字列型	x="abcd"	文字列型
x.to_f	小数へ変換	x.to_f	小数へ変換
x.to_i	整数へ変換	x.to_i	整数へ変換

どうなるか試してみてください

41

式の記述方法

式と値

数式, 文字列式, 論理式, 条件式

42

式

▲ 式は**演算子**(オペレータ。演算内容を表すもの)と**オペランド**(演算の対象)の組み合わせ

▲ 例: $3+4$

- 「3」と「4」がオペランド、「+」が演算子

43

式と値

▲ 今まで曖昧にしてきたが、重要なこと

▲ 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

▲ 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

44

式の値

値

```
irb(main):001:0> 2+3 式
=> 5
irb(main):002:0> x=3
=> 3
irb(main):003:0> x=x*7
=> 21
irb(main):004:0> x==21
=> true
irb(main):005:0> y = 2; y = 5
=> 5
irb(main):006:0>
```

45

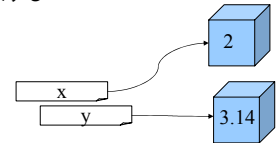
他の変数の値を代入

▲ 変数に他の変数の値を代入する

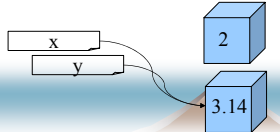
- 等式ではない!

▲ イメージでは、

```
irb(main):016:0> x=2
=> 2
irb(main):017:0> y=3.14
=> 3.14
```



```
irb(main):018:0> x=y
=> 3.14
```

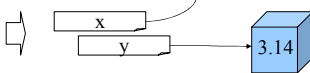


式の値を代入

▲ 「式」の結果の値を作る。そして、代入

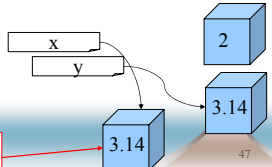
▲ イメージでは、

```
irb(main):016:0> x=2
=> 2
irb(main):017:0> y=3.14
=> 3.14
```



```
irb(main):018:0> x=y+0
=> 3.14
```

y+0 を計算した結果得られた
新たな値に対応して作った数値



47

複雑な式

▲ 四則演算と剰余演算と括弧とを自由に(勿論、正しく)組み合わせたものは正しい式になる

- 但し、型の混合があると、ちょっと、面倒

▲ そのほか、

- 三角関数 Math::PI, Math.sin, Math.cos, Math.tan
- 対数・指数関数 Math::E, Math.log, Math.exp

```
irb(main):033:0> x=Math.sin(Math::PI*0.1) +
Math.exp(Math.log(Math::PI+0.1))
=> 3.55060964796474
irb(main):034:0> x
=> 3.55060964796474
```

48

練習問題①

▲ 次の計算を行ないなさい

- ① $x = \pi / 4$
 $y = \cos(x)^2 + \sin(x)^2$
- ② $x = \pi / 2$
 $z = (1 - \cos(x)) / 2$
- ③ $x = 2$
 $w = \log_e e^x$
- ④ $x = \pi / 4$
 $x = 2^{e^{\sin(x)}}$
- ⑤ $a = x + y + z$
- ⑥ $b = x * y * z$

xに代入している
ことに注意

49

式の種類

- ▲ これまでのように、数値の計算を行わせる式を数式と呼ぶことにします。
- ▲ 文字列の計算を行わせる式を文字列式と呼ぶことにします。
- ▲ 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
- ▲ また、if 論理式 then 数式 else 数式 end とすると数式になります。

```
irb(main):003:0> x=-10
=> -10
irb(main):004:0> if x<0 then -x else x end
=> 10
```

数式

50

論理式

比較演算子

- ▲ $3 == 3$
- 3と3は等しい
- ▲ $3 > 2$ and $4 > 3$
- $3 > 2$ かつ $4 > 3$
- ▲ $"abc" == "abc"$ or $"xyz" == "XYZ"$
- $"abc" == "abc"$ または $"xyz" == "XYZ"$

論理演算子

51

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

52

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	$\text{not } 3 == 2$	true
and	かつ	$2 == 2 \ \text{and} \ 4 > 2$	true
or	または	$2 == 3 \ \text{or} \ 4 > 2$	true

53

論理式の例

```
irb(main):001:0> 3 > 2
=> true
irb(main):002:0> 3 != 3
=> false
irb(main):003:0> 3 > 2 and 3 == 3
=> true
irb(main):004:0> 3 > 2 and 3 != 2
=> true
irb(main):005:0> 3 > 2 and 3 != 3
=> false
irb(main):006:0> 3 > 2 or 2 != 2
=> true
irb(main):007:0> 3 > 2 and 3 == 2
=> false
irb(main):008:0> 3 > 2 or 3 == 2
=> true
```

演算式が正しい場合
→「true」を返す

演算式が正しい場合
→「false」を返す

54

練習問題②

- ▲ 次の論理式の結果はどうなるでしょうか。理由も合わせて考えなさい

- ① $3 + 2 > 5$
- ② $3 + 2 > 4 - 2$
- ③ $3 + 2 > 4 - 2$ and $3 + 2 > 5$
- ④ $3 + 2 > 4 - 2$ or $3 + 2 > 5$
- ⑤ $5 > 2$ and $1 > 2$ or $3 > 2$
- ⑥ $5 > 2$ and $1 > 2$ and $3 > 2$

55

複雑な論理式

- ▲ 比較演算子を用いて、論理式を作り、さらに、論理演算子と組み合わせて、複雑な論理式を書くことができる
- ▲ 括弧を用いて優先順位も決められる

```
irb(main):066:0> Math::PI > 3.14 and
irb(main):067:0* ( Math::E > 2.7 or 3 != 2 or
irb(main):068:1* Math.sin(3.4) > 0.0
irb(main):069:1> )
=> true
```

ここにセミコロンはない。しかし、前に書いてある式が一まとまりになっていない！ Ruby は、この式は終わっていない！と考える

56

if then else end①

- ▲ 「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

57

if then else end②

x = -10

if x < 0 then

-x

else

x

end

x<0 ならば -x

そうでなければ x

xは-10なのでこちらの式が実行される

58

xが3ならば4、違っていれば6

その結果をyに代入

```
irb(main):070:0> x=3
=> 3
irb(main):071:0> y = if x==3 then 4 else 6 end
=> 4
irb(main):072:0> x=4
=> 4
irb(main):073:0> y = if x==3 then 4 else 6 end
=> 6
```



下記と同じです

```
irb(main):007:0> x = 3
=> 3
irb(main):008:0> if x == 3 then y = 4 else y = 6 end
=> 4
irb(main):009:0> x = 4
=> 4
irb(main):010:0> if x == 3 then y = 4 else y = 6 end
=> 6
```

59

もっともっと複雑な式

```
irb(main):081:0> y = if Math.sin(Math::PI/4) >= 0.1 and
irb(main):082:1* ( if Math::E == 2.7 then 2 else 1 end ) > 1.5
irb(main):083:1> then if Math.exp(0.5) >= 0.3 then 0.3 else 0.5 end
irb(main):084:1> else Math.log(23)
irb(main):085:1> end
=> 3.13549421592915
```

Math.sin(Math::PI/4) >= 0.1

and

(if Math::E == 2.7 then 2 else 1 end) > 1.5

二つの式が成立していた場合

成立していない場合

if Math.exp(0.5) >= 0.3 then 0.3 else 0.5

Math.log(23)

60

```

irb(main):081:0> y = if Math.sin(Math::PI/4) >= 0.1 and
irb(main):082:1* ( if Math::E == 2.7 then 2 else 1 end ) > 1.5
irb(main):083:1> then if Math.exp(0.5) >= 0.3 then 0.3 else 0.5 end
irb(main):084:1> else Math.log(23)
irb(main):085:1> end
=> 3.13549421592915

```



```

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

```

実際にプログラムを書く
場合には、まとまっている
構造ごとに段落をそろ
えて書くと分かりやすい

61

練習③

- ①整数型の変数 x の値が偶数の場合は、 x を10倍し、奇数の場合は100倍する式を書きなさい
- ②文字列型の変数 x の長さが5以上の場合は、変数 x を2回繰り返し、5未満の場合は10回繰り返す式を書きなさい
文字列型の変数の長さ $x.length$
文字列型の変数を N 回繰り返し $x*N$

62

練習③

- ③整数型の変数 x , y において、 x と y の大小を判定する式を書きなさい
- ④整数型の変数 x , y において x が y より大きい場合、 x と y の値を入れ換える式を書きなさい

63

Rubyプログラム

- ▲ Ruby のプログラムは、「式」を並べたもの
 - 先回の print 何とかとか puts 何とかとはそれらしくはありません。しかし、今回、irb で入力した各行は「式」らしいですね。
- ▲ 「式」一個でもプログラムです。
 - print 何とかとか puts 何とかも「式」です。
- ▲ 次回のお楽しみ

64

練習問題

65

練習問題

練習①～④を行ないなさい

- ④ 次の式を計算しなさい(レポート課題2と関連)

$x=7; x += x$

$x=7; x += x * x$

$x=7; x += x * x * x$

$x=7; x += x * x * x / x$

$x=7; x += x * x * x / x \% 4$

66

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

67

代入演算子

- ▲ 実は、代入記号(「=」など)も演算子なのです。そして、種類がたくさんあります

演算子	用途	例	演算結果
=	代入	x = 4.3	x ← 4.3
+=	加算後代入	x +=3.1	x ← x + 3.1
-=	減算後代入	x -=3	x ← x - 3
*=	乗算後代入	x *=3	x ← x * 3
/=	除算後代入	x /=3	x ← x / 3
%=	剰余の代入	x %=3	x ← x % 3
**=	幂の代入	x **=3	x ← x ** 3

68

第一回レポート課題

69

問題(1)

- ① 三角関数・指数関数・対数関数・四則演算を組み合わせた複雑な式(要素数で20個以上)を作成し、結果を求めよう。その結果をExcel(あるいは電卓)で計算した結果と比較しよう。
- ② if then else end も含めて複雑な式(要素数20個以上)を作って計算し、式が正しいかどうかを確認しよう。

70

問題(2)

- ▲ どうして、

```
irb(main):006:0> x=7
=> 7
irb(main):007:0> x += x *= x **= x /= x%4
=> 350
```

となるか理由を推定せよ

- もっと短い式、例えば、x += x *= 3 などから初めて、調べてみるとよい(練習④)
- ▲ 自分でも、代入演算子を用いて複雑な式(要素数10個以上)を作って、計算してみよ。

71

第1回レポート提出①

- ▲ 「レポート提出システム」で提出
 - レポート(ワープロ)、プログラムファイルおよび実行結果を電子的に提出してください。
 - レポート本文
 - 先頭の一行は、「プログラミング言語第1回レポート」とし、以下に学籍番号・氏名を記してください。
 - それ以下に、課題内容、各プログラムの説明とプログラムをおよび実行結果、そしてプログラムを書く上で工夫した点、悩んだ点(考察)を書いてください。
 - 最後に、感想を書いてください。

72

第1回レポート提出②

▲ 課題: 問題(1)(2)

▲ 締め切り

- 5/12(月) 10時半(講義開始前まで)

73

レポートの書き方①

A4

プログラミング言語 第一回レポート
学籍番号 氏名

1. 課題内容

何を目的としたプログラムか
入力に対してどのような出力をするプログラムなのか

2. プログラムの説明

使用した変数の説明
問題を解く上でどのようにプログラミングしたのか
プログラムの各行の説明
(ただしプログラムが長い場合、一行ごとに説明する必要はない、まとまった部分ごとで説明すればよい)

74

レポートの書き方②

3. プログラムと実行結果

```
irb(main):081:0> y = if Math.sin(Math::PI/4)
>= 0.1 and
irb(main):082:1* ( if Math::E == 2.7 then 2
else 1 end ) > 1.5
irb(main):083:1> then if Math.exp(0.5) >= 0.3
then 0.3 else 0.5 end
irb(main):084:1> else Math.log(23)
irb(main):085:1> end
=> 3.13549421592915
```

今回の課題ではirbでの実行画面でよい

75

レポートの書き方③

4. 考察

目的とした出力が適切に得られているか
プログラムの作成上において工夫した点
プログラムの作成上において悩んだ点(分からなかった点)
自分がこの課題を作成する上で苦労した点、
他の人とは違うという点を主張して下さい

5. 感想

意見, 要望など

76

レポートを書く上での注意

- ▲ プログラム, レポートのコピーは厳禁
- ▲ 相談するのはかまわない
- ▲ 完成しなかった場合, どこまでできたのか, うまく動かなかったこと, 悩んだことを考察に記述して下さい

77