

# Ruby 講義

## 第5回 Array, Hash

五十嵐邦明

twitter : igaiga555

<http://www.facebook.com/kuniaki.igarashi>



2013.5.9 at 一橋大学  
ニフティ株式会社寄附講義  
社会科学における情報技術と  
コンテンツ作成III

# 五十嵐邦明 講師 株式会社万葉



twitter: igaiga555

<https://github.com/igaiga/>

<http://www.facebook.com/kuniaki.igarashi>

# 濱崎 健吾

Teaching Assistant  
fluxflex, inc(米国法人)



twitter: hmsk

<https://github.com/hmsk/>

<http://www.facebook.com/hamachang>

# 先週の復習



# 変数

## オブジェクトへのラベル・荷札

```
name = "igarashi"
```

name という名前の変数に  
"igarashi" オブジェクトを代入しています。

変数は代入されたオブジェクトを書いたときと  
同じように振る舞います。  
以下の2つは同じ結果になります。

```
puts name
```

```
puts "igarashi"
```

igarashi

igarashi

実行結果

実行結果

# 名付け重要

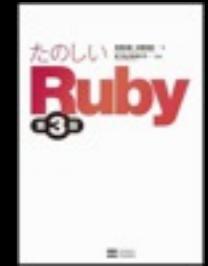
変数名は分かりやすい名前にしよう

良い例

```
width = 20  
height = 3  
area =  
    width * height
```

悪い例

```
a1 = 20  
a2 = 3  
a3 =  
    a1 * a2
```



教科書  
p.23

# コメント文

コードの中にある実行されない文  
コードの説明を書いたりします。

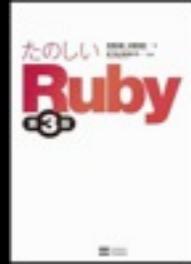
コメント文書式1：# 以降はコメント文

```
# name = "igarashi"  
width = 2 # 文の途中からでもOK
```

↑の場合、以下の色塗りの部分がコメントになります。

```
# name = "igarashi"  
width = 2 # 文の途中からでもOK
```

# 条件判断：if 文



教科書  
p.25-27

**if 条件**

条件が成立したときに実行したい処理

**end**

※教科書には **then** が書いてありますが、省略可能です。

普通は省略します。私は書いたことないです。

条件には値が **true(真)** または **false (偽)** となる式を書くことが一般的

# 条件判断 == 演算子

```
x = 3 - 2
```

```
if x == 1
```

```
  puts "x is 1"
```

```
end
```

x が 1 と同じか判断し  
x が 1 の時に  
puts が実行されます。

== は左辺(x)と右辺(1)が同じかどうか調べて、  
同じならば true、異なる場合は false になります。

== が2個です。= が1つだと代入になってしまってるので注意。  
ちなみに、異なるかを判断する != もあります。  
ほかにも >, >=, <, <= なども使えます。

# インデント(字下げ)

```
if x == 1  
  puts "x is 1"
```

↑  
ind

例えばif文中など、こういう風に先頭にスペースを入れて書くことをインデントするといいます。

プログラムの実行には不要なのですが、

**絶対**に入れてください！

無いと人が読めないので・・・

ちなみにスペースの個数には流派がありますが、2個が主流のようです。

# 条件判断 if - else - end

**if 条件**

条件が成立した時に実行したい処理

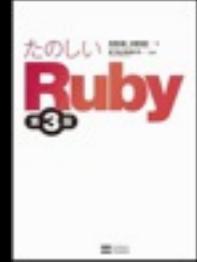
**else**

条件が不成立の時に実行したい処理

**end**

条件が不成立の時に実行したい処理を書く  
こともできます。

# 繰り返し : while文



教科書  
p.27

while 繰り返し続ける条件

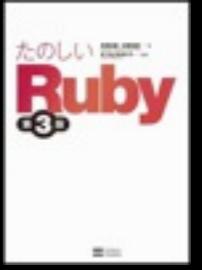
繰り返したい処理

end

1から10までの数を  
順番に表示するコード

```
i = 1
while i <= 10
    puts i
    i = i + 1
end
```

# 繰り返し : times文



教科書  
p.28

繰り返す回数.times do

繰り返したい処理

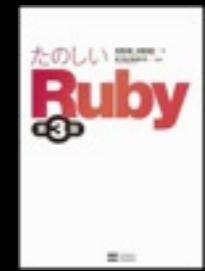
end

※doを  
忘れずに!

Ruby と5回表示する

```
5.times do
  puts "Ruby"
end
```

# メソッドの定義、呼び出し



教科書  
p.28

メソッド定義には  
**def** を使います

```
def メソッド名
  メソッドで実行したい処理
end
```

定義

```
def hello
  puts "Hello"
end
```

呼び出し

```
hello()
```

hello()を呼ぶと、事前に定義していたhelloメソッドが呼ばれ実行されます。 (ここでは puts "Hello")  
hello()の()は省略可能です。 (曖昧にならない限り)

# 引数付きのメソッド

引数の仕組みを使ってメソッドにデータを渡すことができます。

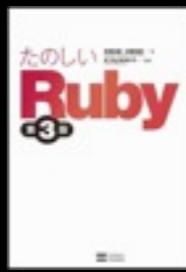
```
def メソッド名(引数)  
    メソッドで実行したい処理  
end
```

定義

```
def hello(word)  
    puts word  
end
```

呼び出し

```
hello("Hi")
```



# require

メソッド定義などを別のファイルに書き、読み込むことができます。

**hello.rb**

```
def hello
  puts "Hello"
end
```

**use\_hello.rb**

```
require "./hello"
hello()
```

↑教科書は `require "hello"` になっていますが、Ruby1.9.2以降だとエラーになるので `"./hello"` としてください。 `./` は(shellで)今いるフォルダの意味です。

**実行**

上記2つのファイルを同じフォルダに置いて以下のコマンド

```
$ ruby use_hello.rb
```

# requireのメリット

メソッドを別のファイルに切り出しておくと、  
複数のプログラムから利用することが可能になります。  
重複して書く必要がなくなったり、  
コードを再利用できたりします。

hello.rb

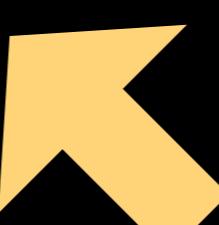
```
def hello  
  puts "Hello"  
end
```

a.rb

```
require "./hello"  
hello()
```

b.rb

```
require "./hello"  
hello()
```



# 課題チェック用の付箋の書き方

上の方に学籍番号と名前を書いてください

学籍番号

名前

※この辺に私と濱崎さん  
がクリアした課題番  
号を書いていきます。

今週

ここから

おしながき

Array

Hash



# 配列(Array)

## ほかのオブジェクトの入れもの

例)

```
names = ["五十嵐", "濱崎"]  
numbers = [1,3,5]
```

[と]で囲い, で区切る。

文字列や数字ほか、どんなオブジェクトも入ります。

空っぽの配列は [] です。

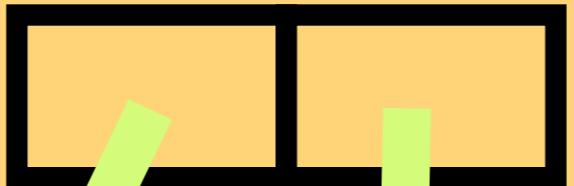
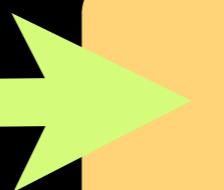
概念図を次のページに示します。

# 配列(Array)概念図

```
names = ["五十嵐", "濱崎"]
```

変数

● names



Array  
オブジェクト

五十嵐

濱崎

String  
オブジェクト

(教科書p.35 図2.2 も参照。)

# 配列から読み込む

番号(index)を指定して読み込み

`names = ["五十嵐", "濱崎"]`

`names[0] → "五十嵐"`

`names[1] → "濱崎"`

配列に[n]と書くと、n番目の要素を返します。

最初の要素は0番です。1始まりではないので注意です。

負数を指定すると末尾から読み込みます。(-1始まり)

`names[-1] → "濱崎"`

`names[-2] → "五十嵐"`

# 配列へ追加する

配列の末尾にオブジェクトを追加するには  
pushメソッドを使います。

```
names = ["五十嵐", "濱崎"]
```

```
names.push("山田")
```

```
p names → ["五十嵐", "濱崎", "山田"]
```

配列に入っている要素数を調べるには sizeメソッド

```
names = ["五十嵐", "濱崎", "山田"]
```

```
p names.size → 3
```

# 配列の繰り返し処理



教科書  
p.38

eachを使うと中身を順番に処理することができます。  
ものすごーーーく大事！！！

配列.each do |変数|

繰り返したい処理

end

# 配列の繰り返し処理



教科書  
p.38

```
names = ["五十嵐", "濱崎", "山田"]  
names.each do |name|  
  puts name  
end
```

"五十嵐"	実行結果
"濱崎"	
"山田"	

nameの両脇  
にある記号 |  
はパイプと読み  
ます。  
キーボードの右  
上の方にある  
(たぶん)。

変数nameの中身が1回目は"五十嵐"、2回目は "濱崎",  
3回目は"山田"となり、繰り返しputs文を実行

# 配列(Array)の演習

a1. 配列 [1,3,5] の全要素を表示するコードを書いてください。

a2. 配列 [1,3,5] の全要素を加えた結果を表示するコードを書いてください。

ヒント: ある変数に数 N を足すのは  $x = x + N$  です。これは  $x += N$  とも書けます。

# 配列(Array)の演習 【上級】

a3. 【上級】 配列 [1,1,2,2,3]を[1,2,3]にするコードを  
1行で書いてください。

ヒント：リファレンスでArrayのメソッドを探してみましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

a4. 【上級】 前述の演習2.をeachを使わずに書いてください

ヒント：injectを使います。使い方はリファレンスから探ししましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

a5. 【上級】 配列の要素をそれぞれ2倍するコードをeachを使わ  
ずに書いてください。例)[1,2,3] → [2,4,6]

ヒント：Arrayはenumerableモジュールのメソッドも使えます。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!class/-enumerable.html>

# ハッシュ(Hash)



教科書  
p.40

これもArrayと同じく、オブジェクトの入れもので使います。  
使い道の例としては、、、



検索

筍 塩麹 ボンゴレ あさり 新玉ねぎ もっと見る...

レシピをさがす

レシピをのせる

クックル

[«hmskpad のレシピ \(13品\)](#)

レシピID: 1188379

## オーソドックスなとんかつ



揚げ物楽しい、豚肉安い、ソースも簡単

 [hmskpad](#)

### 材料 (1人分)

豚ロース	1枚くらい
こしょう	少々
サラダ油	豚ロースが全て浸かるくらい

### ■ 衣

小麦粉 (薄力粉)	100gくらい
パン粉	100gくらい

# こういうページを表現したいときに

# オーソドックスなとんかつ ←title



## ingredients→

揚げ物楽しい、豚肉安い、ソースも簡単

## description ↑

hmskpad

## author ↑

1枚くらい

材料 (1人分)

豚ロース

こしょう

サラダ油

豚ロースが全て浸かるくらい

### ■ 衣

小麦粉 (薄力粉)

100gくらい

卵

1個弱

パン粉

100gくらい

### ■ ソース

ケチャップ

大さじ2

ウスターソース

100ccくらい

データにラベルを付けると扱い易いです

1

2

3

4

包丁の峰で合体向上

オーソドックスなとんかつ

←title

**description**

揚げ物楽しい、豚肉安い、ソースも簡単

**author**

1枚くらい

豚ロース

こしょう

少々

サラダ油

豚ロースが全て浸かるくらい

## ■ 衣

小麦粉（薄力粉）

100gくらい

卵

1個弱

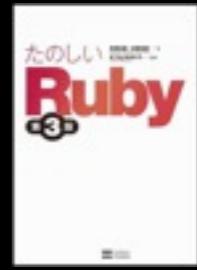
パン粉

100gくらい

**ingredients**→**recipe = {****:title => "オーソドックスなとんかつ",****:author => "hmskpad",****:description => "揚げ物楽しい、豚肉安い、ソースも簡単",****:ingredients => [省略] }**

Hashを使う  
とこんな感じ  
でまとめられ  
ます

# ハッシュ(Hash)



教科書  
p.40

キーと値の組を持てるオブジェクトの入れもの。  
キーをラベル的に使います。なんでも入ります。

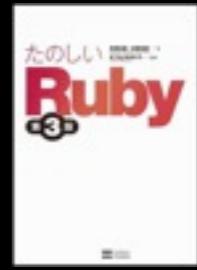
```
recipe = {  
  :title => "♥日向夏のジャム♥",  
  :author => "濱崎" }
```

↑ キー                  ↑ 値

{}で囲う キー => 値 区切りは , 空配列は []

ここでキーに使われている : 始まりのものは何でしょう？

# シンボル



教科書  
p.41

ラベルとして使う文字列的なもの

`:title`

シンボルにするには先頭に`:`を付ける  
ハッシュのキーによく使います

文字列との変換もできます

シンボルへ `"foo".to_sym` → `:foo`

文字列へ `:foo.to_s` → `"foo"`

# ハッシュの別記法(Ruby1.9以降)

```
recipe = {  
  :title => "♥日向夏のジャム♥",  
  :author => "濱崎" }
```

これは、以下のようにも書けます。

```
recipe = {  
  title : "♥日向夏のジャム♥",  
  author : "濱崎" }  
※Ruby1.9以降
```

=> の代わりに : を置き、キーの先頭の : を外します。  
こちらの方が書きやすいので人気です。

# ハッシュから読み込む

```
recipe = { :title => "♥日向夏のジャム♥",
:author => "濱崎" }

p recipe[:title] → "♥日向夏のジャム♥"
p recipe[:author] → "濱崎"
```

ハッシュから値を読むときは、キーを指定します。ハッシュ名[キー]で読み込みます。

# ハッシュへ追加する

ハッシュ名[キー] = 格納したいオブジェクト

同じキーの要素は追加不可(上書き)

ハッシュオブジェクト内でキーは唯一のもの(ユニーク)

```
recipe = { :title => "♥日向夏のジャム♥",
           :author => "濱崎" }
```

```
recipe[:url] = "http://cookpad.com/recipe/xxx"
```

# ↑ ここで追加

```
p recipe → { :title => "♥日向夏のジャム♥",
               :author => "濱崎",
               :url => "http://cookpad.com/recipe/xxx" }
```

# ハッシュの繰り返し処理



教科書  
p.42

Arrayと同じですが、変数を2個となります。

ハッシュ.each do |キーの変数, 値の変数|

繰り返したい処理

end

```
recipe = { :title => "♥日向夏のジャム♥", :author => "濱崎" }
recipe.each do |k, v|
  puts k, " - ", v, "\n"
end
```

title - ♥日向夏のジャム♥  
author - 濱崎

実行結果

# ArrayとHashの使い分け

**Array** : 順番が決まっているいれもの

- ・並び順が重要なものの
- ・データを重複させたい場合にも使える

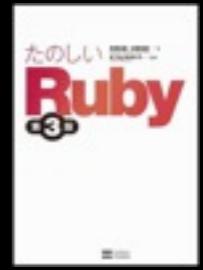
**Hash** : キー(名札)を付けられるいれもの

- ・順番が保持されなくとも困らないもの

※Ruby1.9 からはHashも順番を保持します。

- ・キーが重複しない場合に利用

# nil



教科書  
p.47

「ない」ことを表すオブジェクト  
例えば、ハッシュで存在しないキーを読もう  
とするとこの nil が返ってきます。

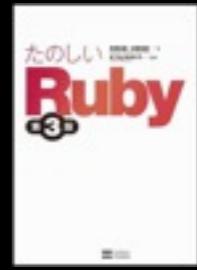
if 文などで条件判断をする場合、

nil は 偽（不成立）になります。

偽（不成立）になるのは false と nil の2つだけです。

それ以外の全ての値は真（成立）になります。

pp



教科書  
p.48

p より見易いデバッグ用メソッド

```
require "pp"
```

```
pp 見たい変数
```

ppには require文 が必要

ハッシュの中身を表示するときに便利です。

# ターミナルの便利機能

ターミナルで使える便利な機能を紹介します。

↑キー：過去に入力したコマンドの履歴を  
遡って実行できます。

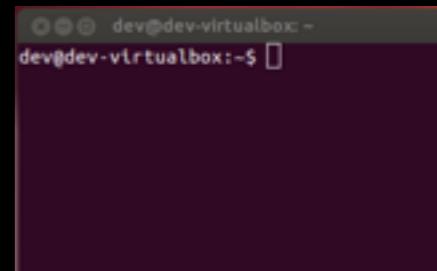
tabキー：途中まで入力した状態で押すと  
残りを補完してくれます。

たとえば、hello.rb というファイルを実行したいときに

\$ ruby he まで打って tab キーを押すと

\$ ruby hello.rb と補完されます。

補完候補が複数ある場合は候補を表示してくれます。



# Hashの演習

b1. ハッシュの全要素を表示する、右のコードを実行してください。

```
# coding: utf-8
recipe = {
:title => "いちごのコンフィ",
:author => "濱崎" }
recipe.each do |k, v|
  print k, " - ", v, "\n"
end
```

b2. b1.のコード中のrecipeの :author キーに対応する値を "五十嵐" とする代入文を書いてください。

# Hashの演習【上級】

b3. 【上級】 右のnumsを値が偶数の要素だけにしてください。

ヒント：リファレンスでHashのメソッドを探してみましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

剰余(割り算した余り)を求める演算子は % です。

```
nums = {  
  :a => 1,  
  :b => 2,  
  :c => 3,  
  :d => 4,  
  :e => 5}
```

b4. 【上級】 右のh1, h2をくっつけて、キーとして :a,:b,:c,:d を持つ Hashを作ってください。

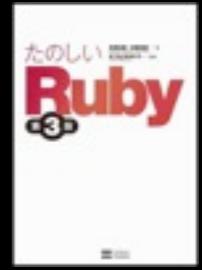
ヒント：リファレンスでHashのメソッドを探してみましょう。

```
h1 = {  
  :a => 1,  
  :b => 2}  
h2 = {  
  :c => 3,  
  :d => 4}
```

結果

```
h = {  
  :a => 1,  
  :b => 2,  
  :c => 3,  
  :d => 4}
```

まとめ



# 配列(Array)

## ほかのオブジェクトの入れもの

例)

```
names = ["五十嵐", "濱崎"]  
numbers = [1,3,5]
```

[と]で囲い, で区切る。

文字列や数字ほか、どんなオブジェクトも入ります。

空っぽの配列は [] です。

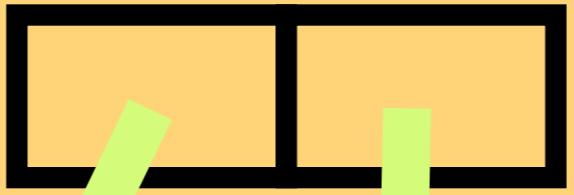
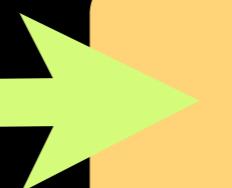
概念図を次のページに示します。(教科書p.35 図2.2 も参照。)

# 配列(Array)概念図

```
names = ["五十嵐", "濱崎"]
```

変数

● names



五十嵐

濱崎

Array  
オブジェクト

String  
オブジェクト

(教科書p.35 図2.2 も参照。)

# 配列から読み込む

番号(index)を指定して読み込み

`names = ["五十嵐", "濱崎"]`

`names[0] → "五十嵐"`

`names[1] → "濱崎"`

配列に[n]と書くと、n番目の要素を返します。

最初の要素は0番です。1始まりではないので注意です。

負数を指定すると末尾から読み込みます。(-1始まり)

`names[-1] → "濱崎"`

`names[-2] → "五十嵐"`

# 配列へ追加する

配列の末尾にオブジェクトを追加するには  
pushメソッドを使います。

```
names = ["五十嵐", "濱崎"]
```

```
names.push("山田")
```

```
p names → ["五十嵐", "濱崎", "山田"]
```

配列に入っている要素数を調べるには sizeメソッド

```
names = ["五十嵐", "濱崎", "山田"]
```

```
p names.size → 3
```

# 配列の繰り返し処理



教科書  
p.38

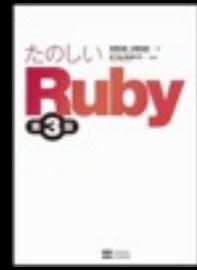
eachを使うと中身を順番に処理することができます。  
ものすごーーーく大事！！！

配列.each do |変数|

繰り返したい処理

end

# ハッシュ(Hash)



教科書  
p.40

キーと値の組を持てるオブジェクトの入れもの。  
キーをラベル的に使います。なんでも入ります。

```
recipe = {  
  :title => "♥日向夏のジャム♥",  
  :author => "濱崎" }
```

↑ キー

↑ 値

{}で囲う キー => 値 区切りは , 空配列は []

ここでキーに使われている : 始まりのものは何でしょう？

オーソドックスなとんかつ

←title

**description**

揚げ物楽しい、豚肉安い、ソースも簡単

**author**

1枚くらい

豚ロース

こしょう

少々

サラダ油

豚ロースが全て浸かるくらい

## ■ 衣

小麦粉（薄力粉）

100gくらい

卵

1個弱

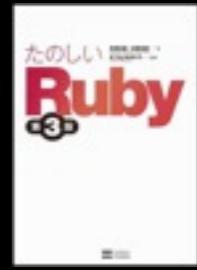
パン粉

100gくらい

**ingredients**→**recipe = {****:title => "オーソドックスなとんかつ",****:author => "hmskpad",****:description => "揚げ物楽しい、豚肉安い、ソースも簡単",****:ingredients => [省略] }**

Hashを使う  
とこんな感じ  
でまとめられ  
ます

# シンボル



教科書  
p.41

ラベルとして使う文字列的なもの

`:title`

シンボルにするには先頭に`:`を付ける  
ハッシュのキーによく使います

文字列との変換もできます

シンボルへ `"foo".to_sym` → `:foo`

文字列へ `:foo.to_s` → `"foo"`

# ハッシュから読み込む

```
recipe = { :title => "♥日向夏のジャム♥",
           :author => "濱崎" }
p recipe[:title] → "♥日向夏のジャム♥"
p recipe[:author] → "濱崎"
```

ハッシュ名[キー]で読み込みます。

# ハッシュへ追加する

ハッシュ名[キー] = 格納したいオブジェクト

同じキーの要素は追加不可(上書き)

ハッシュオブジェクト内でキーは唯一のもの(ユニーク)

```
recipe = { :title => "♥日向夏のジャム♥",
           :author => "濱崎" }
```

```
recipe[:url] = "http://cookpad.com/recipe/xxx"
```

# ↑ ここで追加

```
p recipe → { :title => "♥日向夏のジャム♥",
               :author => "濱崎",
               :url => "http://cookpad.com/recipe/xxx" }
```

# ハッシュの繰り返し処理



教科書  
p.42

Arrayと同じですが、変数を2個となります。

ハッシュ.each do |キーの変数, 値の変数|

繰り返したい処理

end

```
recipe = { :title => "♥日向夏のジャム♥", :author => "濱崎" }
recipe.each do |k, v|
  puts k, " - ", v, "\n"
end
```

title - ♥日向夏のジャム♥  
author - 濱崎

実行結果

# ArrayとHashの使い分け

**Array** : 順番が決まっているいれもの

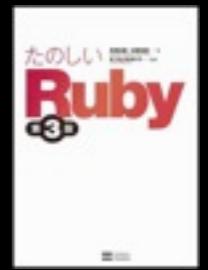
- ・並び順が重要なものの
- ・データを重複させたい場合に利用

**Hash** : キー(名札)を付けられるいれもの

- ・順番が保持されなくとも困らないもの

※Ruby1.9 からはHashも順番を保持します。

- ・キーが重複しない場合に利用



教科書  
p.47

nil

「ない」ことを表すオブジェクト  
例えば、ハッシュで存在しないキーを読もう  
とするとこの nil が返ってきます。

if 文などで条件判断をする場合、

nil は 偽（不成立）になります。

偽（不成立）になるのは false と nil の2つだけです。

それ以外の全ての値は真（成立）になります。

# 演習解答

# 配列(Array)の演習

a1. 配列 [1,3,5] の全要素を表示するコードを書いてください。

a2. 配列 [1,3,5] の全要素を加えた結果を表示するコードを書いてください。

ヒント: ある変数に数 N を足すのは  $x = x + N$  です。これは  $x += N$  とも書けます。

# 配列(Array)の演習 【上級】

a3. 【上級】 配列 [1,1,2,2,3]を[1,2,3]にするコードを  
1行で書いてください。

ヒント：リファレンスでArrayのメソッドを探してみましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

a4. 【上級】 前述の演習2.をeachを使わずに書いてください

ヒント：injectを使います。使い方はリファレンスから探ししましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

a5. 【上級】 配列の要素をそれぞれ2倍するコードをeachを使わ  
ずに書いてください。例)[1,2,3] → [2,4,6]

ヒント：Arrayはenumerableモジュールのメソッドも使えます。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!class/-enumerable.html>

# 配列(Array)の演習解答

a1. 配列 [1,3,5] の全要素を表示するコード  
を書いてください。

```
array = [1,3,5]
array.each do |x|
  puts x
end
```

# 配列(Array)の演習解答

a2. 配列 [1,3,5] の全要素を加えた結果を表示するコードを書いてください。

ヒント: ある変数に数 N を足すのは  $x = x + N$  です。これは  $x += N$  とも書けます。

```
sum = 0
array = [1,3,5]
array.each do |x|
  sum = sum + x
end
puts sum
```

# 配列(Array)の演習解答

a3. 【上級】 配列 [1,1,2,2,3]を[1,2,3]にするコードを1行で書いてください。

ヒント：リファレンスでArrayのメソッドを探してみましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

**p [1,1,2,2,3].uniq!**

配列中の重複を除き、各値1つずつにするには

**Array#uniq!** を使います。

<http://miyamae.github.com/rubydoc-ja/2.0.0/#!method/-array/i/uniq=21.html>

# 配列(Array)の演習解答

a4. 【上級】 上記2.をeachを使わずに書いてください

ヒント : injectを使います。使い方はリファレンスから探しましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

```
p [1,3,5].inject(0){|sum, i| sum + i }
```

inject は便利でかっこいいメソッドです。

<http://miyamae.github.com/rubydoc-ja/2.0.0/#!/method/-enumerable/i/reduce.html>

ちなみに、なぜArrayクラスのメソッドではないinjectを使えるかと  
いうと、ArrayはEnumerableというモジュールのメソッドも使  
えるからです。Arrayだけでなく、eachを持つ全てのクラスは  
Enumerableを使うことができます。

# 配列(Array)の演習解答

a5. 【上級】 配列の要素をそれぞれ2倍するコードをeachを使わずに書いてください。例)[1,2,3] → [2,4,6]

ヒント：Arrayはenumerableモジュールのメソッドも使えます。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!class/-enumerable.html>

```
p [1,2,3].map{|i| i * 2}
```

mapも便利なメソッドです。よく使います。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!method/-enumerable/i/collect.html>

# Hashの演習

b1. ハッシュの全要素を表示する、右のコードを実行してください。

```
# coding: utf-8
recipe = {
:title => "いちごのコンフィ",
:author => "濱崎" }
recipe.each do |k, v|
  print k, " - ", v, "\n"
end
```

b2. b1.のコード中のrecipeの :author キーに対応する値を "五十嵐" とする代入文を書いてください。

# Hashの演習【上級】

b3. 【上級】 右のnumsを値が偶数の要素だけにしてください。

ヒント：リファレンスでHashのメソッドを探してみましょう。

<http://miyamae.github.com/rubydoc-ja/2.0.0/>

剰余(割り算した余り)を求める演算子は % です。

```
nums = {  
  :a => 1,  
  :b => 2,  
  :c => 3,  
  :d => 4,  
  :e => 5}
```

b4. 【上級】 右のh1, h2をくっつけて、キーとして :a,:b,:c,:d を持つ Hashを作ってください。

ヒント：リファレンスでHashのメソッドを探してみましょう。

```
h1 = {  
  :a => 1,  
  :b => 2}  
h2 = {  
  :c => 3,  
  :d => 4}
```

結果

```
h = {  
  :a => 1,  
  :b => 2,  
  :c => 3,  
  :d => 4}
```

# Hashの演習解答

b2. b1.のコード中のrecipeの :author キーに対応する値を "五十嵐" とする代入文を書いてください。

```
recipe = {  
  :title => "いちごのコンフィ",  
  :author => "濱崎" }  
  
recipe[:author] = "五十嵐"
```

この後 p recipe などを書いて確認してみてください。

# Hashの演習解答

b3. 【上級】 右のnumsを値が偶数の要素だけにしてください。

```
nums = { :a => 1, :b => 2,  
         :c => 3, :d => 4, :e => 5}  
nums.select! do |k,v|  
  v % 2 == 0  
end  
p nums
```

`select!` は、あとに続く

`do - end` の結果が`true` の要素だけを残すメソッドです。

偶数かどうかの判定には `v.even?` と書いてもOKです。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!method/-hash/i/select=21.html>

# Hashの演習解答

b4. 【上級】右のh1, h2 をくっつけて、  
キーとして :a,:b,:c,:d を持つHashを作ってください。

```
h1 = { :a => 1, :b => 2}
```

```
h2 = { :c => 3, :d => 4}
```

```
h = h1.merge(h2)
```

```
p h
```

結果

```
{:a => 1, :b => 2, :c => 3, :d => 4}
```

merge は2つのHashをくっつけるメソッドです。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!method/-hash/i/merge.html>



# 參考資料

# 書式

Rubyコード

`puts "abc"`

実行結果

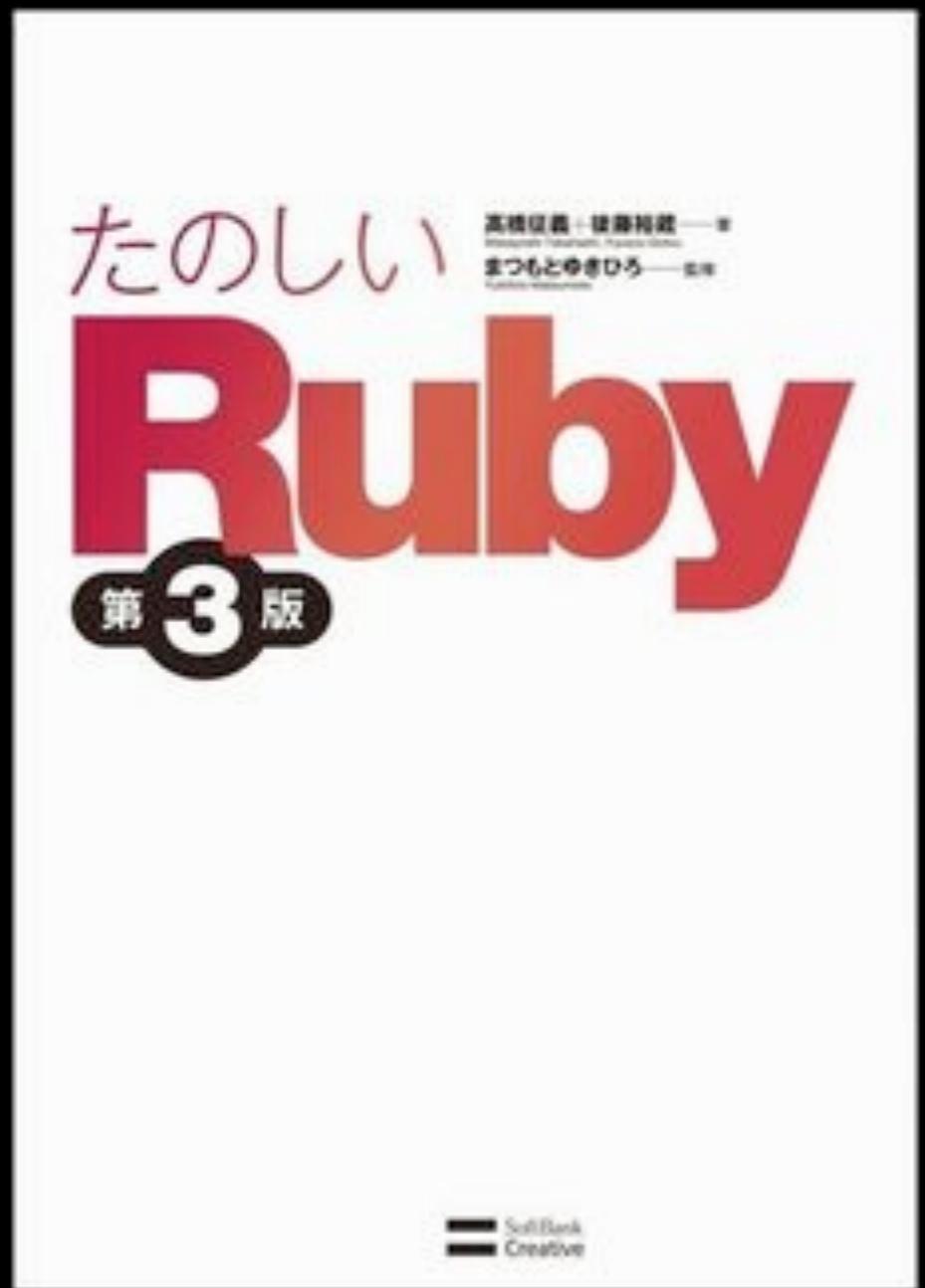
"abc"

実行結果

shellコマンド

`$ ls`

# 教科書：たのしいRuby



<http://www.amazon.co.jp/dp/4797357401/>



お買い求めは  
大学生協または  
ジュンク堂池袋店で

# 講義資料置き場

過去の資料がDLできます。

<https://github.com/igaiga/hitotsubashi-ruby-2013>

# 雑談・質問用facebookグループ

<https://www.facebook.com/groups/hitotsubashi.rb>

- 加入/非加入は自由です
- 加入/非加入は成績に関係しません
- 参加者一覧は公開されます
- 参加者はスタッフ(講師・TA)と昨年、今年の受講者です
- 書き込みは参加者のみ見えます
- 希望者はアクセスして参加申請してください
- 雑談、質問、議論など何でも気にせずどうぞ~
- 質問に答えられる人は答えてあげてください
- 講師陣もお答えします
- 入ったら軽く自己紹介おねがいします