

# Ruby 講義

## 第9回 クラス、インスタンス

五十嵐邦明

twitter : igaiga555

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



2013.6.6 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

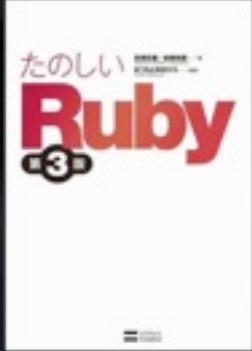
<https://github.com/hmsk/>

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

# 先週の復習

★TODO

# ここから今週の内容



教科書

p.120~152



目次

class

クラスとインスタンス

ローカル変数と  
インスタンス変数

attr\_accessor

initializeメソッド

クラスメソッドと  
インスタンスメソッド

# class

クラス：オブジェクトの種類を表すもの。型。種族。

[1,2,3]

["a","b"]

[]

"abc"

"ちはやふる"

""

**Array**オブジェクト  
= **Array**クラスのオブジェクト

**String**オブジェクト  
= **String**クラスのオブジェクト

"abc"や"ちはやふる"は別オブジェクトですが、同じ**String**クラスに属してるので、同じメソッド群が用意されています。

「○○クラスのオブジェクト」を

「○○クラスのインスタンス」と言うこともあります。インスタンスって？

# クラスとインスタンス

Array

String

[1,2,3]

"ちはやふる"

クラス  
オブジェクトの種類を表すもの。  
型。 種族。 設計図。

インスタンス  
クラスから作られたもの。

# クラスとインスタンス



<http://www.flickr.com/photos/tonomura/2391806827/>



[http://www.flickr.com/photos/\\_yoggy0/4406076358/](http://www.flickr.com/photos/_yoggy0/4406076358/)

クラス  
設計図

インスタンス  
classから作られたもの

# インスタンスをつくる newメソッド

**new** : クラスからインスタンスオブジェクトを作るメソッド  
(=たい焼きの型を使って、たい焼きを焼く)

**Array.new #=> []**

**String.new #=> ""**

今まで書いていたように、インスタンスオブジェクトを直接作ることも可能です。

[1,2,3]

"ちはやふる"

では、クラスを実際に  
作ってみましょう。

# 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

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

卵

1個弱

パン粉

100gくらい

### ■ ソース

ケチャップ

大さじ2

マヨネーズ

100ccくらい

Hashの例で出てきたレシピを  
今回はクラスを使って表現します。

# classのつくりかた

class(たい焼きの型)を自分で作る場合の書き方です。

```
class クラス名  
  クラスの定義  
end
```

クラス名は必ず大文字から始める。  
例：Array, String, Recipe, Book

例：

```
class Recipe  
  #定義を書く  
end
```

# つかったclassを newしてみる

自作のclass(たい焼きの型)からnewしてインスタンス  
(たい焼き)をつくるてみます。

```
class Recipe  
end
```

```
recipe = Recipe.new
```

小文字の **recipe** は変数で、インスタンスを代入しています。  
大文字始まりの **Recipe** はクラス名です。

# classにメソッドをつくる

Recipeクラスはまだ何も仕事ができないので、  
メソッドを実装して仕事ができるように育てます。

最初に、タイトルを取得できるようにtitleメソッドを作ります。

```
class Recipe
  def title
    "cheese cake"
  end
end
```

```
recipe = Recipe.new
p recipe.title #=> "cheese cake"
```

メソッドが返す値は、最後の文の実行結果です。

タイトルを返せるようになりました。

しかし決まったタイトルしか返せないので、次はタイトルを設定できるようにしましょう。

# Recipeクラスにタイトルを設定できるようにしてみる

次は、タイトルを設定、取得できるようにします。

※注：このコードには誤りがあります。

```
class Recipe
  def title=(t)
    recipe_title = t
  end
  def title
    recipe_title
  end
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
```

6:in `title': undefined local variable or method  
`recipe\_title' for #<Recipe:0x10796d3d8>  
(NameError)

実行結果

**title=** メソッドを実装してタイトルを設定できるようにしてみました。これで **recipe\_title**変数へ代入することができます。

しかし、**title**メソッドで **recipe\_title**変数の中身を返すようにしましたが、**recipe\_title**変数がないエラーがでました。なんで？？

# エラーの理由は 変数のスコープ（有効範囲）

変数には有効範囲があります。

```
class Recipe
  def title=(t)
    recipe_title = t # ※1
  end
  def title
    recipe_title
  end
end
```



ここでは上記の  
recipe\_title変数  
にアクセスできない

変数には有効範囲、生存  
期間があります。メソッ  
ド内で作成した変数は、  
そのメソッドの中だけが  
有効範囲です。

このような変数を  
ローカル変数と呼びます。

※1 の行の変数 **recipe\_title** は、そのメソッ  
ドの中だけがスコープ（有効範囲）です。

では、スコープの広い  
変数を作るにはどうす  
ればいいでしょうか？

# インスタンス変数

インスタンスオブジェクトが生存している間ずっと使える変数が  
インスタンス変数です。変数名の頭に @ をつけます。

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title #=> "cheese cake"
```

@recipe\_title  
のスコープ

ここでも@recipe\_title変数  
にアクセスできる

# インスタンス変数の性質 1

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end

recipe1 = Recipe.new
recipe1.title = "cheese cake"
recipe2 = Recipe.new
recipe2.title = "macaroon"

p recipe1.title #=> "cheese cake"
p recipe2.title #=> "macaroon"
```

インスタンスオブジェクト（たい焼き）ごとに  
インスタンス変数を  
別々に持っています。

# インスタンス変数の性質 2

※このコードには誤りがあります

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.recipe_title ✗
```

```
undefined method `recipe_title' for #<Recipe:
0x108650280 @recipe_title="cheese cake">
(NoMethodError)
```

実行結果

インスタンス変数は  
オブジェクトの外か  
ら直接アクセスでき  
ません。

アクセスする時は、  
そのオブジェクトの  
メソッドを通じてア  
クセスします。

# こんなコードになりました

整理すると、こんなコードになりました。

**Recipe**クラスはレシピのタイトルを格納したり取り出せたりします。

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title #=> "cheese cake"
```

これでも機能としては足りる  
のですが、コードを少し読み  
やすく改良します。

# コードをちょっと良くします！

@recipe\_titleという変数の名前を変えましょう。

レシピクラスの中なのでレシピなのは当たり前。

ただの @title という名前に変更します。

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```

```
class Recipe
  def title=(t)
    @title = t
  end
  def title
    @title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```



同じ動作

# コードをちょっと良くします2

インスタンス変数を同名のメソッドで読み書きするコードはよく使うので、便利な書き方 `attr_accessor` が用意されています。

```
class Recipe
  def title=(t)
    @title = t
  end
  def title
    @title
  end
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```

書き方：  
`attr_accessor` インスタンス変数名のシンボル

```
class Recipe
  attr_accessor :title
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```

同じ動作

とても短くなりました。

# レシピクラスができました

ほかの要素 :author, :description, :ingredients も追加します。

```
# -*- coding: utf-8 -*-
class Recipe
  attr_accessor :title, :author, :description, :ingredients
end

recipe = Recipe.new
recipe.title = "cheese cake"
recipe.author = "igarashi"
recipe.description = "やばい"
recipe.ingredients = "（略）"

p recipe.title #=> "cheese cake"
p recipe.author #=> "igarashi"
p recipe.description #=> "やばい"
p recipe.ingredients #=> "（略）"
```

title や author などを格納したり取り出したりできます。

# 演習問題

```
class Book  
...  
end
```

**Book クラスを作ります。(以下、全て同じコードへ追記して構いません)**

- a1. メソッド **category** を実装して、"novel"という文字列を返してください。
- a2. **Book** クラスのインスタンスを作ってください。さきほど作った **category** メソッドを呼び、結果を画面に表示してください。
- a3. **Book** クラスのインスタンス変数 **@title** に "Momo" をセットできるようにしてください。**(attr\_accessorを使わずに、title=メソッドを使って実装してください。)**
- a4. 同じく **@author** に "Michael Ende" をセットできるようにしてください。**(attr\_accessorを使ってください)**
- a5. a3, a4で作ったインスタンス変数の **@title** と **@author** を表示してください。

次のページに  
解答があります

# 演習問題解答

a1. メソッド `category` を実装して、"novel"という文字列を返してください。

a2. `Book` クラスのインスタンスを作ってください。さきほど作った `category` メソッドを呼び、結果を画面に表示してください。

```
class Book
  def category
    "novel"
  end
end
```

```
book = Book.new
p book.category #=> "novel"
```

# 演習問題解答

- a3. Book クラスのインスタンス変数 @title に "Momo" をセットできるようにしてください。  
(attr\_accessorを使わずに、title=メソッドを使って実装してください。)
- a4. 同じく @author に "Michael Ende" をセットできるようにしてください。(attr\_accessorを使ってください)
- a5. a3, a4で作ったインスタンス変数の @title と @author を表示してください。

```
class Book
  def title
    @title
  end
  def title=(t)
    @title = t
  end
  attr_accessor :author
end

book = Book.new
book.title = "Momo"
book.author = "Michael Ende"
p book.title #=> "Momo"
p book.author #=> "Michael Ende"
```

# 命名規則

クラス名は大文字始まり、変数名は小文字のみ

クラス名の例：**Array, String, Recipe, Book**

2単語以上を組み合わせるときは、単語境界文字を大文字にします

例：**RecipeSite**

ちなみに、このタイプの規則を**Camel Case**といいます。（らくだ）

変数名の例：**array, string, recipe, book**

2単語以上を組み合わせるときは、単語を **\_** でつなぎます

例：**recipe\_site**

ちなみに、このタイプの規則を**Snake Case**といいます。（へび）

# initialize メソッド

**initialize** という特別な名前のメソッドを作ると、  
インスタンスを作る際(**new**メソッドが呼ばれた際)に  
自動で実行するメソッドを作ることができます。

```
class Recipe
  attr_accessor :title, :author
  def initialize
    puts "initialize!!"
  end
end

recipe = Recipe.new #=> "initialize!!"
```

# initialize メソッド

newメソッドに引数を渡すと、initializeメソッドの引数として受け取ることができます。

```
class Recipe
  attr_accessor :title, :author
  def initialize(title, author)
    @title = title
    @author = author
  end
end
```

```
recipe = Recipe.new("cheese cake","igarashi")
p recipe.title #=> "cheese cake"
p recipe.author #=> "igarashi"
```

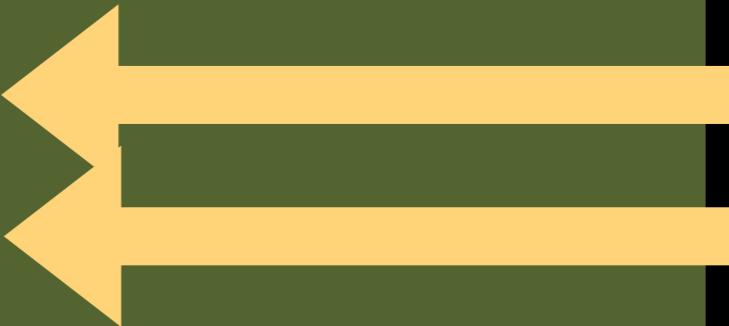
# インスタンスメソッド

ここまで見てきたような、クラスの中で普通に定義したメソッドをインスタンスメソッドといいます。インスタンスに対して呼ぶことができるメソッドです。

```
class Recipe
  attr_accessor :title, :author

  def title_and_author
    @title + " - " + @author
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
recipe.author = "igarashi"
p recipe.title_and_author #=> "cheese cake - igarashi"
```



`attr_accessor` で作られるメソッドもインスタンスメソッドです。

`title_and_author` は  
インスタンスメソッド

インスタンスメソッドは  
インスタンス(たい焼き)に対して呼ぶことができます。

# クラスメソッド

もう1種類、クラスメソッドというのも存在します。クラスメソッドはクラス（たい焼きの型）に対して呼び出します。  
self.メソッド名で定義します。

```
class クラス名  
  def self.メソッド名  
  end  
end
```

```
class Recipe  
  attr_accessor :title, :author  
  def self.published_in  
    "COOKPAD"  
  end  
end  
  
p Recipe.published_in #=> "COOKPAD"
```

クラスに対して呼ぶ。  
newしないで呼ぶ。

# インスタンスマソッドと クラスメソッド

インスタンスマソッドはインスタンス（たい焼き）に対して呼び、  
クラスメソッドはクラス（たい焼きの型）に対して呼びます。

```
class Recipe
  attr_accessor :title, :author
  def self.published_in
    "COOKPAD"
  end
end
```

```
p Recipe.published_in #=> "COOKPAD"
recipe = Recipe.new
recipe.title = "cheese cake"
```

```
recipe = Recipe.new
p recipe.published_in #=> "COOKPAD" X
Recipe.title = "cheese cake" X
```

クラスメソッドは  
クラスに対して呼ぶ。

インスタンスマソッドは  
インスタンスに対して呼ぶ。

逆は呼べない

# インスタンス変数にアクセスできる のはインスタンスマソッドだけ

インスタンス変数にアクセスできるのはインスタンスマソッドだけです。  
クラスメソッドの中ではインスタンス変数にアクセスできません。

```
class Recipe
  attr_accessor :title, :author

  def title_and_author
    @title + " - " + @author
  end

  def self.published_in
    @title + " - " + @author X
  end
end
```

インスタンスマソッドなので  
インスタンス変数にアクセス可

クラスメソッドなので  
インスタンス変数にアクセス不可

たい焼きの型に、「あんこ多い？」って聞い  
ても答えられないのと同じです。たぶん。  
(あんこはインスタンス変数なので)

# 記法

今までこっそり (?) 使ってましたが、インスタンスマソッドとクラスマソッドはこのような記法で表すこともあります。

クラス名#インスタンスマソッド名

クラス名.クラスマソッド名

```
class Recipe
  def self.published_in
    "COOKPAD"
  end
  def title
    @title
  end
end
```

**Recipe#title**  
**Recipe.published\_in**

# 演習問題

※ 以下は1つのコードで書いてもOKです。

Book クラスを作って、以下を実装してください。

**b1.** `info`という名前のクラスメソッドを作り、"This is Book class" という文字列を返してください。

**b2.** `initialize` メソッドを作ってください。1つの引数 `title` を受け取り、インスタンス変数 `@title` へ代入してください。

**b3.** b2 で作った `@title` 変数を返すメソッドを作ってください。

次のページに  
解答があります

# 演習問題解答

**Book** クラスを作って、以下を実装してください。

**b1.** `info`という名前のクラスメソッドを作り、"Book class" という文字列を返してください。

**b2.** `initialize` メソッドを作ってください。1つの引数 `title` を受け取り、インスタンス変数 `@title` へ代入してください。

**b3.** b2 で作った `@title` を返すメソッドを作ってください。

```
class Book
attr_accessor :title
def self.info
  "Book class"
end
def initialize(title)
  @title = title
end
end
```

```
p Book.info #=> "Book class"
book = Book.new("Momo")
p book.title #=> "Momo"
```

**b3.** は `attr_accessor` を使うと1行で書けます。  
または、以下のように書いてもOKです。

```
def title
  @title
end
```

まとめ

# クラスとインスタンス

Array

String

[1,2,3]

"ちはやふる"

クラス  
オブジェクトの種類を表すもの。  
型。 種族。 設計図。

インスタンス  
クラスから作られたもの。

# クラスとインスタンス



<http://www.flickr.com/photos/tonomura/2391806827/>



[http://www.flickr.com/photos/\\_yoggy0/4406076358/](http://www.flickr.com/photos/_yoggy0/4406076358/)

クラス  
設計図

インスタンス  
classから作られたもの

# インスタンスをつくる newメソッド

**new**: クラスからインスタンスオブジェクトを作るメソッド

**Array.new #=> []**

**String.new #=> ""**

今まで書いていたように、インスタンスオブジェクトを直接作ることも可能です。

[1,2,3]

"ちはやふる"

# classのつくりかた

class(たい焼きの型)を自分で作る場合の書き方です。

```
class クラス名  
  クラスの定義  
end
```

クラス名は必ず大文字から始める。  
例：Array, String, Recipe, Book

例：

```
class Recipe  
  #定義を書く  
end
```

# つかったclassを newしてみる

自作のclass(たい焼きの型)からnewしてインスタンス  
(たい焼き)をつくるてみます。

```
class Recipe  
end
```

```
recipe = Recipe.new
```

小文字のrecipe は変数で、 インスタンスを代入しています。  
大文字始まりのRecipeはクラス名です。

# エラーの理由は 変数のスコープ（有効範囲）

変数には有効範囲があります。

```
class Recipe
  def title=(t)
    recipe_title = t # ※1
  end
  def title
    recipe_title
  end
end
```



ここでは上記の  
recipe\_title変数  
にアクセスできない

変数には有効範囲、生存  
期間があります。メソッ  
ド内で作成した変数は、  
そのメソッドの中だけが  
有効範囲です。

このような変数を  
ローカル変数と呼びます。

※1 の行の変数 **recipe\_title** は、そのメソッ  
ドの中だけがスコープ（有効範囲）です。

では、スコープの広い  
変数を作るにはどうす  
ればいいでしょうか？

# インスタンス変数

インスタンスオブジェクトが生存している間ずっと使える変数が  
インスタンス変数です。変数名の頭に @ をつけます。

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title #=> "cheese cake"
```

@recipe\_title  
のスコープ

ここでも@recipe\_title変数  
にアクセスできる

# インスタンス変数の性質 1

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end

recipe1 = Recipe.new
recipe1.title = "cheese cake"
recipe2 = Recipe.new
recipe2.title = "macaroon"

p recipe1.title #=> "cheese cake"
p recipe2.title #=> "macaroon"
```

インスタンスオブジェクト（たい焼き）ごとに  
インスタンス変数を  
別々に持っています。

# インスタンス変数の性質 2

※このコードには誤りがあります

```
class Recipe
  def title=(t)
    @recipe_title = t
  end
  def title
    @recipe_title
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.recipe_title ✗
```

undefined method `recipe\_title' for #<Recipe:  
0x108650280 @recipe\_title="cheese cake">  
(NoMethodError)

実行結果

インスタンス変数は  
オブジェクトの外か  
ら直接アクセスでき  
ません。

アクセスする時は、  
そのオブジェクトの  
メソッドを通じてア  
クセスします。

# attr\_accessor

インスタンス変数を同名のメソッドで読み書きするコードはよく使うので、便利な書き方 attr\_accessor が用意されています。

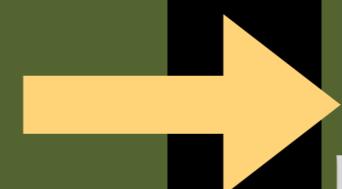
```
class Recipe
  def title=(t)
    @title = t
  end
  def title
    @title
  end
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```

書き方：  
attr\_accessor インスタンス変数名のシンボル

```
class Recipe
  attr_accessor :title
end

recipe = Recipe.new
recipe.title = "cheese cake"
p recipe.title
#=> "cheese cake"
```



同じ動作

とても短くなりました。

# 命名規則

クラス名は大文字始まり、変数名は小文字のみ

クラス名の例：**Array, String, Recipe, Book**

2単語以上を組み合わせるときは、単語境界文字を大文字にします

例：**RecipeSite**

ちなみに、このタイプの規則を**Camel Case**といいます。（らくだ）

変数名の例：**array, string, recipe, book**

2単語以上を組み合わせるときは、単語を **\_** でつなぎます

例：**recipe\_site**

ちなみに、このタイプの規則を**Snake Case**といいます。（へび）

# initialize メソッド

**initialize** という特別な名前のメソッドを作ると、  
インスタンスを作る際(**new**メソッドが呼ばれた際)に  
自動で実行するメソッドを作ることができます。

```
class Recipe
  attr_accessor :title, :author
  def initialize
    puts "initialize!!"
  end
end

recipe = Recipe.new #=> "initialize!!"
```

# initialize メソッド

newメソッドに引数を渡すと、initializeメソッドの引数として受け取ることができます。

```
class Recipe
  attr_accessor :title, :author
  def initialize(title, author)
    @title = title
    @author = author
  end
end
```

```
recipe = Recipe.new("cheese cake","igarashi")
p recipe.title #=> "cheese cake"
p recipe.author #=> "igarashi"
```

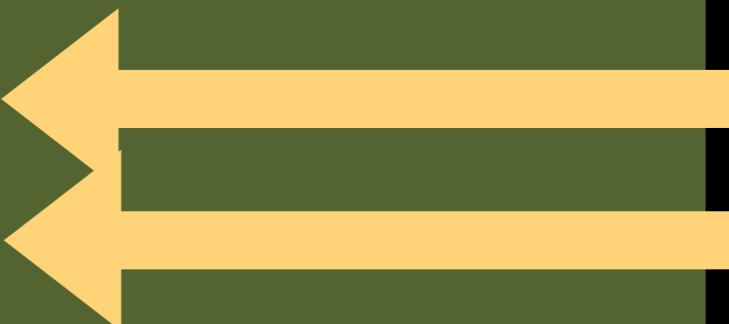
# インスタンスメソッド

ここまで見てきたような、クラスの中で普通に定義したメソッドをインスタンスメソッドといいます。インスタンスに対して呼ぶことができるメソッドです。

```
class Recipe
  attr_accessor :title, :author

  def title_and_author
    @title + " - " + @author
  end
end
```

```
recipe = Recipe.new
recipe.title = "cheese cake"
recipe.author = "igarashi"
p recipe.title_and_author #=> "cheese cake - igarashi"
```



`attr_accessor` で作られるメソッドもインスタンスメソッドです。

`title_and_author` は  
インスタンスメソッド

インスタンスメソッドは  
インスタンス(たい焼き)に対して呼ぶことができます。

# クラスメソッド

もう1種類、クラスメソッドというのも存在します。クラスメソッドはクラス（たい焼きの型）に対して呼び出します。  
self.メソッド名で定義します。

```
class クラス名
  def self.メソッド名
  end
end
```

```
class Recipe
  attr_accessor :title, :author
  def self.published_in
    "COOKPAD"
  end
end

p Recipe.published_in #=> "COOKPAD"
```

クラスに対して呼ぶ。  
newしないで呼ぶ。

# インスタンスマソッドと クラスメソッド

インスタンスマソッドはインスタンス（たい焼き）に対して呼び、  
クラスメソッドはクラス（たい焼きの型）に対して呼びます。

```
class Recipe
  attr_accessor :title, :author
  def self.published_in
    "COOKPAD"
  end
end
```

```
p Recipe.published_in #=> "COOKPAD"
recipe = Recipe.new
recipe.title = "cheese cake"
```

```
recipe = Recipe.new
p recipe.published_in #=> "COOKPAD" X
Recipe.title = "cheese cake" X
```

クラスメソッドは  
クラスに対して呼ぶ。

インスタンスマソッドは  
インスタンスに対して呼ぶ。

逆は呼べない

# インスタンス変数にアクセスできる のはインスタンスマソッドだけ

インスタンス変数にアクセスできるのはインスタンスマソッドだけです。  
クラスメソッドの中ではインスタンスマソッドにアクセスできません。

```
class Recipe
  attr_accessor :title, :author

  def title_and_author
    @title + " - " + @author
  end

  def self.published_in
    @title + " - " + @author X
  end
end
```

インスタンスマソッドなので  
インスタンス変数にアクセス可

クラスメソッドなので  
インスタンス変数にアクセス不可

たい焼きの型に、「あんこ多い？」って聞い  
ても答えられないのと同じです。たぶん。  
(あんこはインスタンス変数なので)



# 參考資料

# 書式

Rubyコード

`puts "abc"`

実行結果

"abc"

実行結果

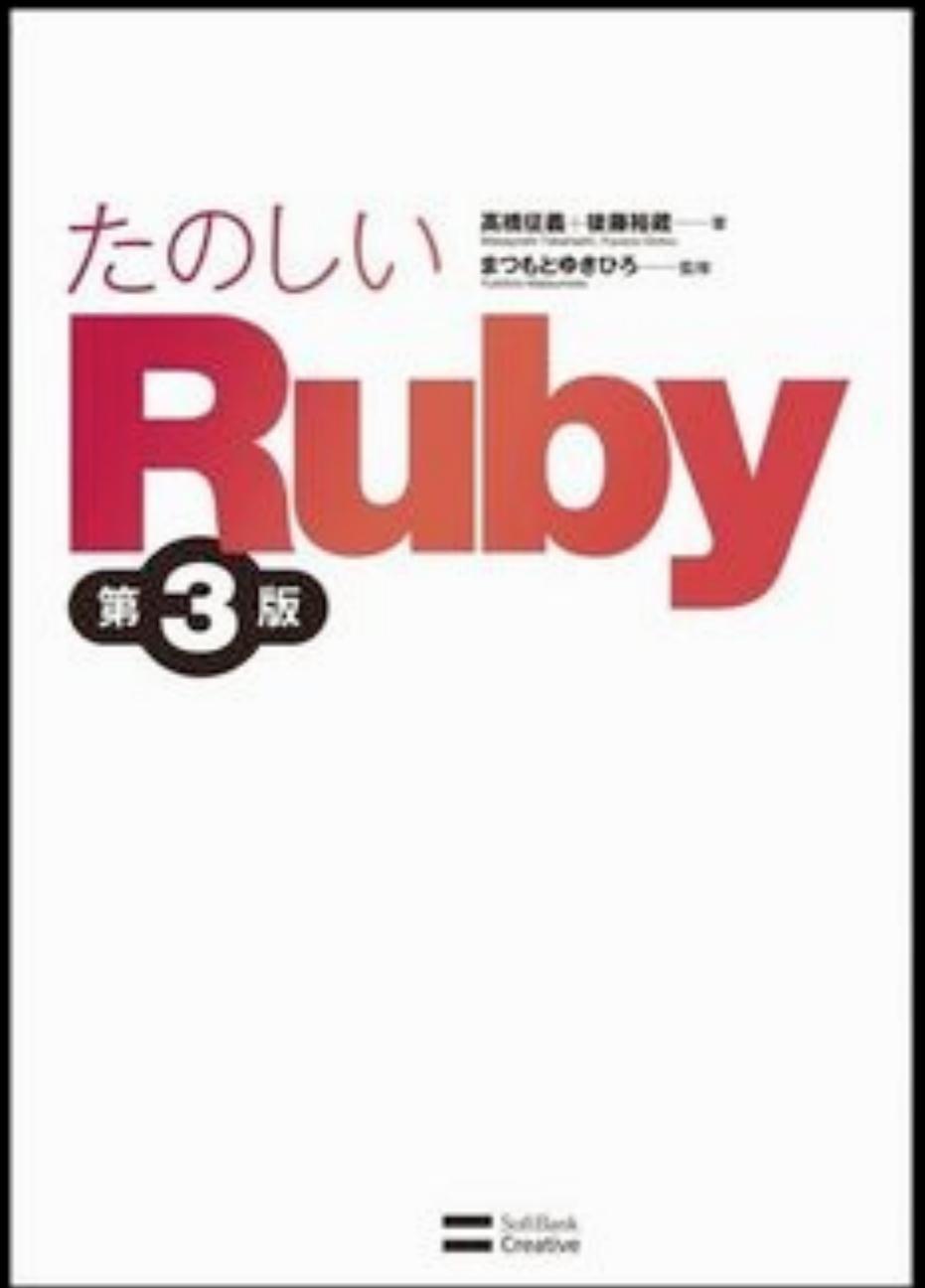
文中では `#=>` で書きます

`p 1+2 #=> 3`

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