

# Ruby 講義

## 第10回 オブジェクト指向

五十嵐邦明

twitter : igaiga555

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



2013.6.20 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



corporate profile

ニフティとなら、きっとかなう。  
With Us, You Can.

@nifty

# ニフティ 夏の勉強会 秋の発表会

# 夏 の出会い を 秋 に実らせよう

インターナシップ  
参加者募集中

新しく「カタチ」にする  
「チャレンジ」で  
アイデア

### 夏の勉強会

アイデアソン1dayとアプリ企画開発5daysの2コース。世の中を楽しく便利にするアイデアを持ち寄ってカタチにし、Web / アプリ企画・チーム開発の面白さを感じよう！5daysコースで遠方の方には宿泊施設をご用意します。

- 日程：
  - アイデアソン1dayコース  
2013年8月1日(木)、8月2日(金)、8月5日(月)、8月7日(水)
  - アプリ企画開発5daysコース  
2013年8月26日(月)～8月30日(金)
- 場所：
  - ニフティ株式会社 本社(東京都新宿区)  
※5dayコースで遠方の方には宿泊施設をご用意します。
- 内容：
  - サービスの現場社員との交流と、チームでのアプリ企画 / 開発体験
- 定員：
  - アイデアソン1dayコース  
16名 / 回(4回開催)  
※応募多数の場合は日程追加予定。
  - アプリ企画開発5daysコース  
40名
- エントリー締め切り：  
**2013年6月30日(日)**

### 秋の発表会

夏に出会った仲間たちとカタチにしたアイデアの完成度を高めて、秋にお台場のイベントハウスで発表しよう！

- 日程：  
2013年11月9日(土)
- 場所：  
東京カルチャーカルチャー  
(東京お台場)
- 内容：  
アプリ作品発表会
- 応募：  
**2013年9月10日(金)～2013年10月10日(木)**

詳しくはWEBよりご参照下さい。

「ニフティ採用担当Blog」  
<http://niftyjinji.cocolog-nifty.com/blog/2013/06/2013-9ae1.html>

マイナビコード74232 株式公開

## ニフティ(株)

[ネット関連技術](#) [通信事業サービス](#) [広告](#) [情報処理](#)

本社所在地 東京都 資本金 37億4,677万9,000円 売上高 連結：919億5,900万円（2012年3月期）

従業員 単独：637名 連結：731名（2012年9月30日現在）

エントリー

● エントリーボックス

会社概要

インターンシップ

### ○ 夏の勉強会 アイデアソン1 day

開催地域 東京エリア

開催時期 8月1日～8月7日に、1日のプログラムを4回分実施します。

応募締切 6月30日

### ○ 夏の勉強会 アプリ企画開発 5 days

開催地域 東京エリア

開催時期 8月26日～8月30日

応募締切 6月30日



# 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

# 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/>

クラス  
設計図



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

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)



たい焼きの型  
Taiyakiクラス

Taiyaki.new



Taiyaki.new



たい焼きの型(Taiyakiクラス)  
から、たい焼きインスタンスが  
1つできあがります。

newするとその回数だけ  
たい焼きインスタンスが  
できます。

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

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

実際のRubyのコードでは  
newメソッドを使うと、  
そのクラスの空っぽのオブジェクトができる

一方で、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  
  #ここにクラスの内容を書く  
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
```

recipe\_title のスコープ

ここで上記の recipe\_title 变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。

このような变数をローカル変数と呼びます。

※1 の行の变数 **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、スコープの広い变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)
```

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。

実行結果

# 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、单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、单語を **\_** 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。

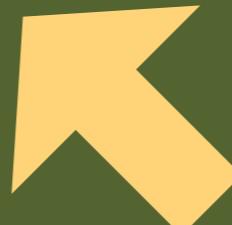
```
class Recipe
  attr_accessor :title, :author
  def initialize
    puts "initialized!"
  end
end

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

# 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
```



newメソッド呼び出し時に渡された引数が initializeメソッドに渡される。

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

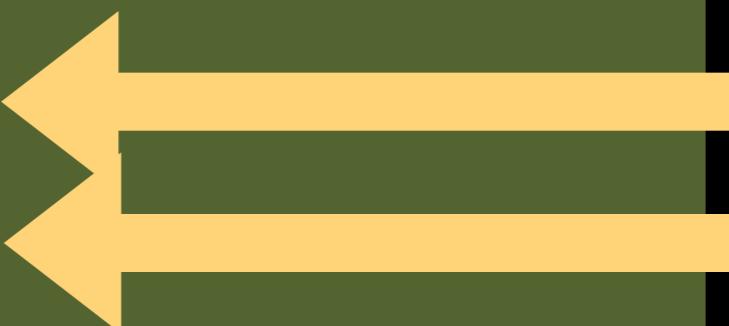
# インスタンスメソッド

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_by  
    "COOKPAD"  
  end  
end
```

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

クラスに対して呼ぶ。  
newしないで呼ぶ。

# インスタンスメソッド、クラスメソッド

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
  def title
    @title
  end
  def title=(t)
    @title = t
  end
  def self.published_by
    "COOKPAD"
  end
end
```

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

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

← インスタンスメソッド

← インスタンスメソッド

self. がついているのでクラスメソッド

クラスメソッド  
(この文ではpublished\_by)  
はクラスに対して呼ぶ。  
インスタンスメソッド  
(この文ではtitle)  
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_by
  @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(たい焼き)のもの)

# ここから今週の内容

# 目次

## オブジェクト指向

なぜクラスを作るのか

オブジェクト指向

メソッドのアクセス制限 : private, public

**attr\_accessor, attr\_writer, attr\_reader**

なんで  
クラス  
つくるの？

コードを  
整理整顿  
したいから

動けばいいじゃないか  
コードだも口

それだと困ることがあります・・・



もしもコードの世界が  
ヴィレッジヴァンガード  
だったら・・・



よーし、新機能つくるぞー！  
新しくコードを加える場所を探すぜー！  
って、みつかるかー！！

※ヴィレッジヴァンガードがお店として悪いってわけ  
じゃないですよ念のため。コードの場合の例えです。





ジャンルごとに分類して  
整頓されていれば  
目的のものを探せます

コードの世界の場合は  
クラスと  
そのオブジェクトを使  
って整理します

# \* 【簡単】爽やかレアチーズケーキ\* ←title

 レシピを保存



先週の例でてきたRecipeクラス  
のオブジェクト

単純作業ばかりの簡単レアチーズケーキ！  
母からのお墨付きをもらいました♪

description ↑  
※オーブン時間に変更しました。  
※ゼラチン多め

 kanan店長

材料	(18cm底の抜ける丸型)	author ↑
小さめビスケット	16枚	
溶かしバター(無塩)	35g	
☆クリームチーズ	200g	
☆水きりヨーグルト	200g	
☆砂糖	90g	
☆レモン汁(ポッカレモン)	大2	
☆牛乳	10g	
☆ゼラチン	200ml	
☆水	16g	
		kanan店長



料理名・食材名



目的・用途

レシピ検索

父の日 夏 夏バテ



レシピをのせる

## \* 【簡単】爽やかレアチーズケーキ\*



レシピを保存



単純作業ばかりの簡単レアチーズケーキ！  
母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め



kanan店長

材料 (18cm底の抜ける丸型)

小さめビスケット	16枚
溶かしバター(無塩)	35g
☆クリームチーズ	200g
☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g
★水	大4

サイト全体を見てみると、ほかのオブジェクトもたくさんあります。



レンジでバターを溶かして、卵の中に入れて、よく混んではじます。

下準備  
卵をよくこねて、卵の中に入れて、よく混んではじます。



る。



クックパッド

食べ過ぎリセット お知らせ

# お知らせオブジェクト

1番人気のレシピが今だけ無料で見放題▼

口グイニ

Q 料理名・食材名

3

目的·用途

レシピ検索

# 父の日 夏 夏バテ 注意

七

## レシピをのせる

# \*【簡単】アートやカラースキーマをオブジェクト化するプロジェクト



単純作業ばかりの簡単レアチーズケーキ！  
母からのお墨付きをもらいました♪

 レシピを保存

※材料、倍量に変更しました。

※ゼラチン多め



kanan店長

### 材料 (18cm底の抜ける丸型)

小さめビスケット	16枚
溶かしバター(無塩)	35g
☆クリームチーズ	200g
☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

このように各クラスのオブジェクトに分割してコードを書くのが定石です。

かして、袋の中に入  
れ揉んでなじませ  
る。



クックパッド

食べ過ぎリセット

お知らせオブジェクト

1番人気のレシピが今だけ無料で見放題▼

ログイン



料理名・食材名



目的・用途

レシピ検索

父の日 夏 夏バテ

注目ワード

レシピをのせる

# レシピクラスオブジェクト

レシピを保存

単純作業ばかりの簡単レアチーズケーキ！

母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め

kanan店長

材料 (18cm底の抜ける丸型)

小さめビスケット	16枚
溶かしバター(無塩)	35g
☆クリームチーズ	200g
☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

水

レモン

袋

ゼラチン

どういうオブジェクトに分割すればいいかは経験によるところも多いのですが、各オブジェクトの役割と責任を明確にするのが1つの方法です。

# 役割 と 責任





料理名・食材名

## **× 目的・用途**

レシピ検索

# 父の日 夏 夏バテ対策

# 注目ワード

## レシピをのせる

# オブジェクト

# レシピクラスオブジェクト



単純作業ばかりの簡単レアチーズケーキ！

母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め

 レシピを保存



 kanaan店長

### 材料 (18cm底の抜ける丸型)

小さめピスケット	16枚
溶かしバター(無塩)	35g
☆クリームチーズ	200g
☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

各オブジェクトに役割と責任を持たせると、  
問題があった場合に怪しそうなコードがある場所を  
推測できます。



クックパッド

食べ過ぎリセット

お知らせオブジェクト

1番人気のレシピが今だけ無料で見放題▼

ログイン



料理名・食材名



目的・用途

レシピ検索

父の日 夏 夏バテ

注目ワード

レシピをのせる

# レシピクラスオブジェクト

レシピを保存

単純作業ばかりの簡単レアチーズケーキ！

母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め

kanan店長

材料 (18cm底の抜ける丸型)

この部分の表示が  
意図通り出ない・・・

☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

こここのデータを管理す  
る役割はRecipeクラス  
が受け持ってるから、そ  
こに何かバグがあるのか  
も？

各オブジェクトに役割と責任を持たせると、  
問題があった場合に怪しそうなコードがある場所を  
推測できます。

かして、袋の中に入  
・ゼラチンをふやか  
る。  
なじませ  
す。



クックパッド

食べ過ぎリセット

## お知らせオブジェクト

1番人気のレシピが今だけ無料で見放題▼

ログイン



料理名・食材名



目的・用途

レシピ検索

父の日 夏 夏バテ

注目ワード



レシピをのせる

## \*【簡単】チーズケーキオブジェクト

レシピを保存

単純作業ばかりの簡単レアチーズケーキ！  
母からのお墨付きをもらいました♪

※材料、倍量に変更しました。

※ゼラチン多め

kanan店長

材料 (18cm底の抜ける丸型)

この部分の表示が  
意図通り出ない・・・

☆水きりヨーグルト	200g
☆砂糖	90g
☆レモン汁(ポッカレモン)	大2
☆蜂蜜	10g
☆牛乳	200ml
★ゼラチン	16g

こここのデータを管理す  
る役割はRecipeクラス  
が受け持ってるから、そ  
こに何かバグがあるのか  
も？

こういった各オブジェクトに役割と責任を持たせる設  
計指針を「オブジェクト指向」といいます。

かして、袋の中に入  
れ揉んでなじませ  
る。

・ゼラチンをふやか  
す。

# レシピオブジェクトの役割と責任

## レシピオブジェクト

```
class Recipe
  def title
    @title
  end
  def title=(t)
    @title = t
    @updated_at = Time.now
  end
  ...
  @title = "cheese cake"
  @author = "igarashi"
  ...
end
```

レシピに関するデータと  
メソッドを持ちます。

レシピに関するデータを管  
理します。

データへのアクセスは所定  
のメソッドを通じてお願  
いします。

# レシピオブジェクトの役割と責任

## レシピオブジェクト

```
class Recipe
```

```
def title      公開ゾーン  
  @title  
end  
  
def title=(t)  
  @title = t  
  @updated_at = Time.now  
end  
  
...  
  
@title = "cheese cake"  
@author = "igarashi"  
  
...  
  
end
```

もし、レシピオブジェクト  
が全公開で、インスタンス  
変数を他の人に操作されて  
しまうと・・・

レシピ『よーし、「チーズケー  
キ」のレシピ表示するぞー、って  
誰だよ「塩麹唐揚げ」に勝手にデ  
ータ変えたのはー！しかも最終更  
新日を変更していないじゃん！』

となり大混乱。

# レシピオブジェクトの役割と責任

## レシピオブジェクト

```
class Recipe
```

```
def title          公開ゾーン  
  @title  
end  
def title=(t)  
  @title = t  
  @updated_at = Time.now  
end
```

```
...               非公開ゾーン
```

```
  @title = "cheese cake"  
  @author = "igarashi"
```

```
...  
end
```

データへのアクセスは所定  
のメソッドを通じてお願  
いします。

→ 公開ゾーンと  
非公開ゾーンを分ける

データを外から勝手に書き換えるとレシピオブジェクトは責任を果たせない  
= データ(変数)は外部からはアクセスできなくすべき(非公開ゾーンに置くべき)  
= Rubyのclassは最初からそうなるように設計されています

# レシピオブジェクトの役割と責任

## レシピオブジェクト

```
class Recipe
```

```
def title          公開ゾーン  
  @title  
end  
  
def title=(t)  
  @title = t  
  @updated_at = Time.now  
end
```

```
...               非公開ゾーン  
  @title = "cheese cake"  
  @author = "igarashi"  
...  
end
```

僕レシピ。

レシピに関する各種業務は  
僕の責務です。

レシピのタイトルを書き換  
えるときは公開している  
title=を使ってください。  
そのときに最終更新日  
(@updated\_at)も更新  
するので。

# レシピオブジェクトの役割と責任

## レシピオブジェクト

```
class Recipe
```

```
def title      公開ゾーン  
  @title  
end  
def title=(t)  
  @title = t  
  @updated_at = Time.now  
end
```

```
...          非公開ゾーン  
  @title = "cheese cake"  
  @author = "igarashi"  
...  
end
```

外部からの操作を公開メソッドからだけに絞ることで

『分かりました、 titleを「塩麹唐揚げ」に変更ですね。データ変えておきます。(・・・タイトルを変えるときは一緒に @updated\_atも変えて・・・、と)』

と責任持って管理することができます。

オブジェクトの各所を外部に公開する・しないを制御するのがオブジェクト指向プログラミングでは大切になります。

そのため、メソッドに1つ1つについても公開・非公開を変更することができます。

# public, private

## メソッドの公開・非公開を制御できます。

```
class AccessTest
```

```
public
```

```
#これ以降に書いたメソッドはpublic
```

```
def show
```

```
  puts "public method"
```

```
end
```

```
#ここに書いたメソッドもpublic
```

```
private
```

```
#これ以降に書いたメソッドはprivate
```

```
def secret
```

```
  puts "private method"
```

```
end
```

```
end
```

```
obj = AccessTest.new
```

```
obj.show #=> "public method"
```

```
obj.secret
```

## public

以降のメソッドを公開メソッドにします。(または、何も書かないと publicになります)

## private

以降のメソッドを非公開メソッドにします。非公開メソッドは、オブジェクト内部からは呼び出せますが、外部からは呼び出せません。

※protectedというのもあるのですが、滅多に使わないので省略

# オブジェクトの内部と外部

# オブジェクトの内部と外部

```
class AccessTest
  def show
    secret #ここは内部
  end

  private
  def secret
    puts "private method"
  end
end
```

```
obj = AccessTest.new
obj.secret #ここは外部 ✗
```

内部

内部

そのオブジェクトクラスのメソッド  
の中。そのオブジェクトクラスの  
class～endの間。

外部

それ以外

または、

secret のようにメソッド名だけで呼べ  
るところが内部、

obj.secret のように  
オブジェクト.メソッド名で  
呼ぶところが外部です。

**attr\_accessor**

**attr\_reader**

**attr\_writer**

# attr一族

先週説明した

`attr_accessor` は

インスタンス変数を読み書きするメソッド  
を提供する文です。

そのほかに、読み込みだけを許可する

`attr_reader`、書き込みだけを許可する

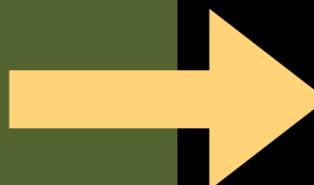
`attr_writer` も用意されています。

# attr\_accessor

読み込みと書き込み用の公開メソッドを用意します。

```
class Recipe  
  attr_accessor :title  
end
```

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



同じ動作

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

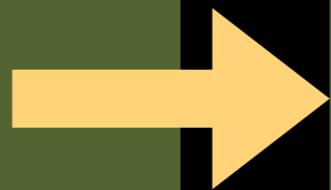
```
recipe = Recipe.new  
recipe.title = "cheese cake"  
p recipe.title  
#=> "cheese cake"
```

# attr\_reader

読み込み用の公開メソッドを用意します。

```
class Recipe  
  attr_reader :title  
end
```

```
recipe = Recipe.new  
p recipe.title
```



同じ動作

```
class Recipe  
  def title  
    @title  
  end  
end
```

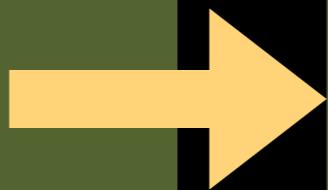
```
recipe = Recipe.new  
p recipe.title
```

# attr\_writer

書き込み用の公開メソッドを用意します。

```
class Recipe  
  attr_writer :title  
end
```

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



同じ動作

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

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

# 演習問題1

**Sample**クラスをつくってください。

**Sample**クラスに**private**なメソッド**priv**を実装してください。  
**priv**メソッドの中で **puts "priv!!"** の1行を実行してください。

**Sample**クラスに**public**なメソッド**pub**を実装してください。  
**pub**メソッドの中で **priv** メソッドを実行してください。

**Sample**クラスのインスタンスオブジェクトを作り、  
**pub**メソッドを呼び出してください。

# 演習問題2

Omikuji クラスがあります。  
getメソッドを呼ぶとランダムで  
大吉と吉と凶をランダムで出します。

2-1: ※の行(omikuji.get)を変更して、  
必ず大吉を出すようにしてください。

2-2: 上記の変更をされてしまうと、  
Omikujiオブジェクトとしては不本意です。  
このようななづるをされないように、  
Omikujiクラスを修正して下さい。

```
class Omikuji
  def get
    case rand(3)
    when 0
      daikichi
    when 1
      kichi
    when 2
      kyou
    end
  end
  def daikichi
    puts "Daikichi!!"
  end
  def kichi
    puts "Kichi"
  end
  def kyou
    puts "Kyou..."
  end
end

omikuji = Omikuji.new
omikuji.get
```

# 演習問題3

以下の実行結果になるように黒塗り部分を実装してください。

```
class Book
```

```
    def decorated_title
```

```
        "***"+@title+"***"
```

```
    end
```

```
end
```

```
book = Book.new
```

```
book.title = "Romeo and Juliet"
```

```
p book.decorated_title #=> "***Romeo and Juliet***"
```

解答

# 演習問題1 解答

Sampleクラスをつくってください。

Sampleクラスにprivateなメソッドprivを実装してください。

privメソッドの中で puts "priv!!" の1行を実行してください。

Sampleクラスにpublicなメソッドpubを実装してください。

pubメソッドの中で priv メソッドを実行してください。

Sampleクラスのインスタンスオブジェクトを作り、 pubメソッドを呼び出してください。

```
class Sample
  def pub
    priv
  end
  private
  def priv
    puts "priv!!"
  end
end
```

```
obj = Sample.new
obj.pub
```

# 演習問題2解答

2-1: ※の行(omikuji.get)を変更して、必ず大吉を出すようにしてください。

omikuji.daikichi

2-2: 上記の変更をされてしまうと、Omikujiオブジェクトとしては不本意です。このようななづるをされないように、Omikujiクラスを修正して下さい。

←部分に **private** を書くことでget以外のメソッドをオブジェクト外から呼び出し不可能にします。

```
class Omikuji
  def get
    case rand(3)
    when 0
      daikichi
    when 1
      kichi
    when 2
      kyou
    end
  end
  def daikichi
    puts "Daikichi!!"
  end
  def kichi
    puts "Kichi"
  end
  def kyou
    puts "Kyou..."
  end
end
```

```
omikuji = Omikuji.new
omikuji.get
```

# 演習問題2補足

ちなみに、getメソッド内で使われている  
**case, when** 文は、  
**case** の後に書かれたコードの結果と  
合致する**when**文を実行します。

**rand(3)**はランダムに 0,1,2を返します。

**rand(3)** が0を返したときは

**daikichi**メソッドを呼び、

1を返したときは**kichi**メソッド、

2を返したときは**kyou**メソッドが

それぞれ呼び出されます。

```
class Omikuji
  def get
    case rand(3)
    when 0
      daikichi
    when 1
      kichi
    when 2
      kyou
    end
  end
  ...
end
```

# 演習問題3解答

以下の実行結果になるように黒塗り部分を実装してください。

```
class Book
```

```
  def decorated_title  
    "***"+@title+"***"  
  end  
end
```

```
book = Book.new  
book.title = "Romeo and Juliet"  
p book.decorated_title  
#=> "***Romeo and Juliet***"
```

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

またはattr\_writerを使うこともできます。

```
attr_writer :title
```



# 參考資料

# 書式

Rubyコード

`puts "abc"`

実行結果

"abc"

実行結果

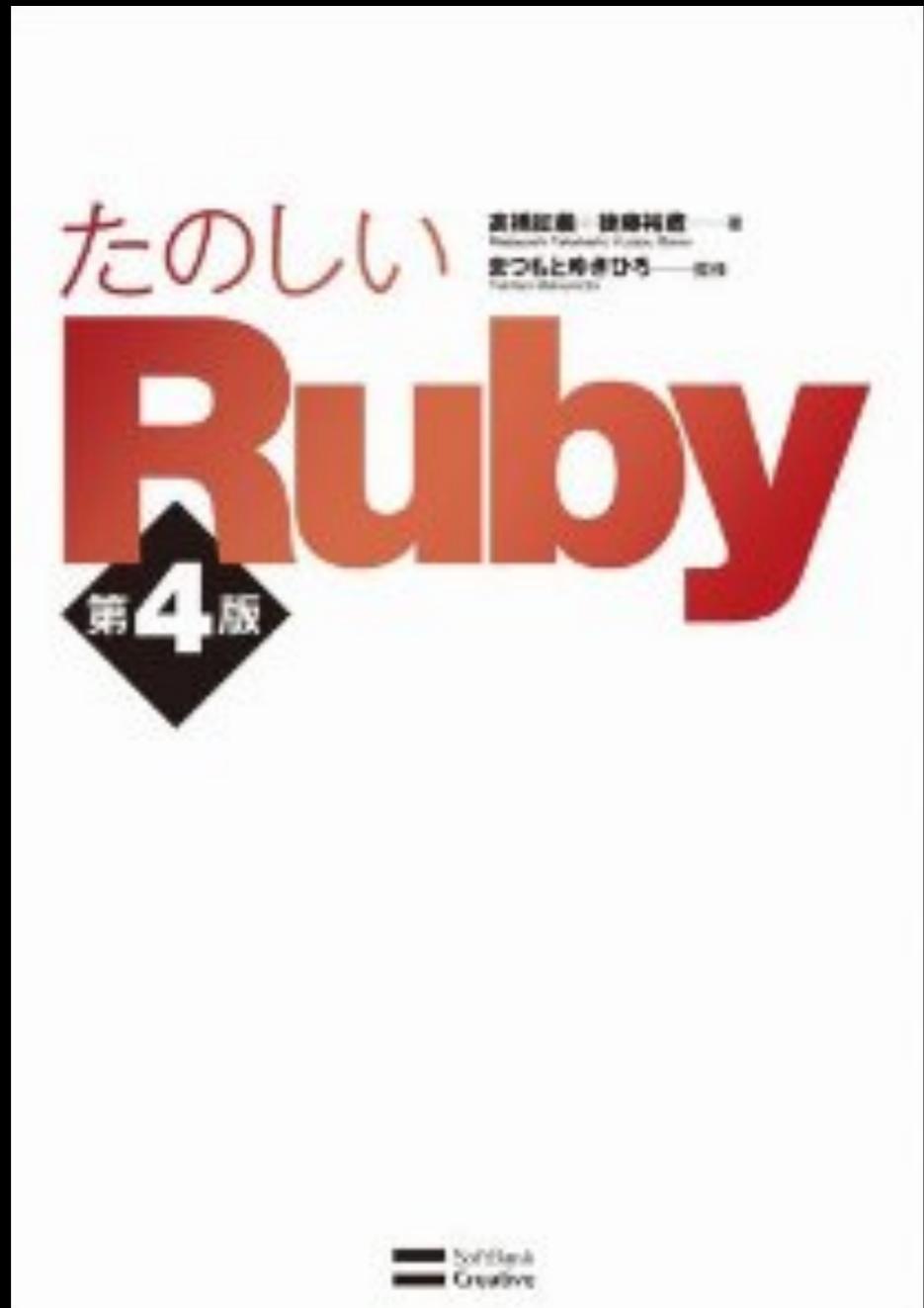
文中では `#=>` で書きます

`p 1+2 #=> 3`

shellコマンド

`$ ls`

# 教科書：たのしいRuby



<http://www.amazon.co.jp/dp/4797372273/>



お買い求めは  
大学生協または  
ジュンク堂池袋店で

# 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

※この辺に講師陣がクリアした課題番号を書いていきます。

# 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