

# Ruby 講義

# 第10回 Ruby入門

Kuniaki IGARASHI/igaiga

2012.6.14 at 一橋大学

社会科学における情報技術とコンテンツ作成III  
(ニフティ株式会社寄附講義)

○ 剰余金の配当に関するお知らせ

○ ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始

○ 「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開

○ 「スマブレ！」のサービス停止について

○ ニフティとサンリオウェーブ、iOS向けアプリ「Hello Kitty Worl...」

○ 平成24年3月期 決算短信

○ 特別損失の計上に関するお知らせ

○ 「シユフモ」登録会員数150万人を突破、「2012年主婦の全国節電調査（冬季...」

ニフティとなら、きっとかなう。  
With Us, You Can.

# ニフティ株式会社

アット・ニフティ  
楽しいサービスがいっぱい

アクセスマップ  
大森から西新宿へ移転いたしました

@nifty Web募金  
東日本大震災復興支援  
募金受付中

- 2012年4月25日 IR [特別損失の計上に関するお知らせ](#)
- 2012年4月25日 IR [剰余金の配当に関するお知らせ](#)
- 2012年4月19日 IR [「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始](#)
- 2012年4月19日 IR [ニフティとサンリオウェーブ、iOS向けアプリ『Hello Kitty World』を台湾で提供開始](#)
- 2012年4月10日 お知らせ [「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開](#)

# 提供

講師

# 五十嵐邦明

株式会社万葉

エンジニア



GARASHI

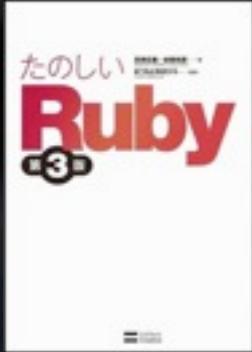
.9.25 at 高専カンファ

いがいが  
⑤

Teaching Assistant 演習 健二  
クックパッド株式会社 エンジニア



先週の  
おさらい



# 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

# 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)

**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

レシピID:1188379



## description ↑

揚げ物楽しい、豚肉安い、ソースも簡単

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

マヨネーズ

100ccくらい

## ingredients →

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。

# 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)
```

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

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  
newメソッドに引数を渡すと、 initializeメソッドで受け取ること  
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"
```

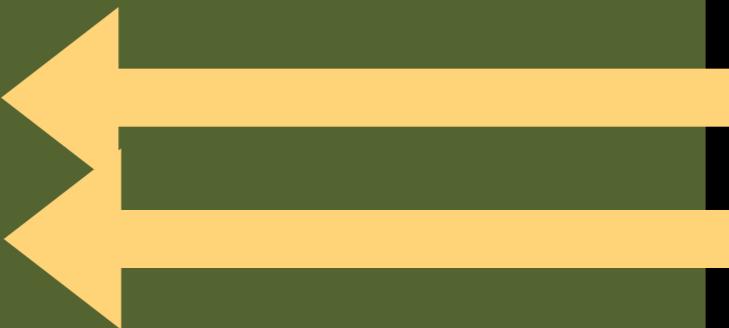
# インスタンスメソッド

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" ✗
Recipe.title = "cheese cake" ✗
```

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

※ 以下を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。

1. `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。
2. `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。
3. 2.で作った `@title` 変数を返すメソッドを作ってください。

# 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。

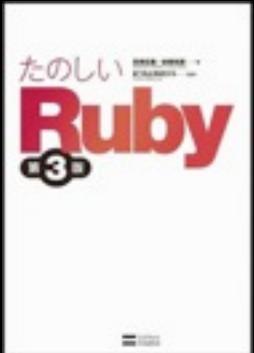
1. **info**という名前のクラスメソッドを作り、"Book class" という文字列を返してください。
2. **initialize** メソッドを作ってください。1つの引数 **title** を受け取り、インスタンス変数 **@title** へ代入してください。
3. 2.で作った **@title** を返すメソッドを作ってください。

```
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"
```

3. 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
```



教科書

p.120~152

# 目次

## クラスの設計方針

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**

継承

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  
ユードだも口

いやダメです



もしもコードの世界が  
village vanguard  
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！！



あー、バグがあった ><  
コードの問題箇所を探すぜー！  
って、みつかるかー！！



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

# 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

味噌

大さじ1

みりん

大さじ1

先週の例でてきたRecipeクラス  
のオブジェクト




検索

人気検索

[塩麹](#) [蒲焼き](#) [親子丼](#) [梅酒](#) [リメイク](#) [もっと見る...](#)

[MYニュース](#)
[MYフォルダ](#)
[MYキッチン](#)
[レシピをさがす](#)
[人気順でさがす](#)
[レシピをのせる](#)
[クックパッドID \(無料\) を登録する](#) | [ログイン](#) | [ヘルプ](#)
[«hmskpad のレシピ \(13品\)](#)

## オーソドックスなとんかつ



レシピID: 1188379

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
卵	1個弱
パン粉	100gくらい

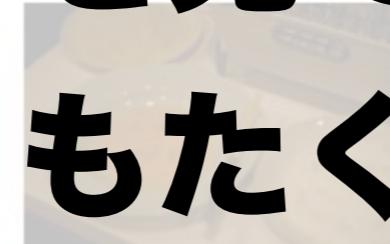
### ■ ソース

ケチャップ	大さじ2
ウスターソース	100ccくらい

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。



包丁の峰で全体をよく叩きながらこしょく全体にかけます。両面あります。



衣を付ける準備をし



小麦粉は全体を押さ

P.S.  
FA

Perfect Suit FAcotry

ネットショップ限定  
**アウトレット** × **スーツ**  
 Special Price **¥10,500 (税込) ~**  
 上質でスタイリッシュなのに低価格。  
+1 オススメするには十分な理由がある!

 毎週更新！おすすめレシピ特集 PR [一覧はこちら](#)
[太りたくない人の夕食レシピ](#) ▶
 
[塩こうじレシピ大集合！](#) ▶
 
[今日のメインに！お肉レシピ](#) ▶
 
[人気ユーザーいちおしレシピ](#) ▶
 
[お酒に合う♪パーティ料理](#) ▶
 
[森三中の♪簡単ヘルシー料理](#) ▶
 
[食アート♪ジ麺レシピ](#) ▶
 
[もっと見る](#)

# メソニュークラスオブジェクト

[クックパッドID \(無料\) を登録する](#) | [ログイン](#) | [ヘルプ](#)

# レシピクラスオブジェクト

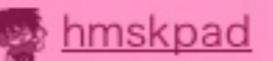
オーソドックスなとんかつ



揚げ物楽しい、豚肉安い、ソースも簡単



レシピID: 1188379



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
ウスターソース	100ccくらい

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。



包丁の峰で全体をよく叩いてから、こまごまと細かく。それを全体にかけます。両面やります。

豚をまな板に出し、



衣を付ける準備をし



小麦粉は全体を押さ

# 広告クラス オブジェクト

Perfect Suit FAcotry

アウトレット × スーツ

Special Price ¥10,500 (税込) ~

上質でスタイリッシュなのに低価格。  
オススメするには十分な理由がある!

+1

おすすめレシピ特集 PR 一覧はこちら

# 特集クラス オブジェクト

今日のメインに！お肉レシピ

人気ユーザーいちおしレシピ

お酒に合う♪パーティ料理

森三中の♪簡単ヘルシー料理

簡単♪アレンジ麺レシピ

▶ [もっと見る](#)

## メソニュークラスオブジェクト

[クックパッドID \(無料\) を登録する](#) | [ログイン](#) | [ヘルプ](#)

# レシピクラスオブジェクト

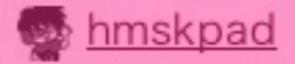
オーソドックスなとんかつ



揚げ物楽しい、豚肉安い、ソースも簡単



レシピID:1188379



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
ウスターソース	100ccくらい

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。

# 広告クラスオブジェクト



Special Price ¥10,500 (税込) ~

上質でスタイリッシュなのに低価格。  
オススメするには十分な理由がある!

# 特集クラスオブジェクト

今日のメインに！お肉レシピ

人気ユーザーいちおしレシピ

# 役割 と 責任



# 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
```

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

レシピに関するデータと手続きを持ちます。

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。

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

もし、レシピオブジェクト  
が全公開だとすると、イン  
スタンス変数を他の人に操  
作されて・・・

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
```

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。

といった設計指針は  
オブジェクト指向  
と呼ばれています

# オブジェクト指向

プログラムを書くときの基本となる考え方

プログラムの処理の対象をオブジェクトとして考える  
オブジェクトは、データ(変数)と手続き(メソッド)をまとめたもの

言葉で言うと難しいですが、実は既にみなさんが書いている  
コードもオブジェクト指向の思想に則っています。

Rubyはオブジェクト指向を考慮して設計されているので、自然とオブジェクト指向の考え方へ沿ったコードを書けます。

```
f = 3.14
```

```
f.round #=> 3 (四捨五入)
```

```
f.ceil #=> 4 (切り上げ)
```

Floatオブジェクト3.14は、  
データ(3.14)と、  
手続き(round, ceilメソッドなど)を持つ

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
    def pub
      puts "public method"
    end
    #ここに書いたメソッドはpublic

  private
  def priv
    puts "private method"
  end
  # ここに書いたメソッドはprivate
end

obj = AccessTest.new
obj.pub #=> "public method"
obj.priv ✗
```

### 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 pub
```

```
    priv # ここは内部
```

```
  end
```

```
private
```

```
def priv
```

```
  puts "private method"
```

```
end
```

```
end
```

```
obj = AccessTest.new
```

```
obj.priv # ここは外部 X
```



## 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、

priv のようにメソッド名だけで呼べると  
ころが内部、

obj.priv 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。

# 演習問題

**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。

**Recipe**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。

Recipe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
  def priv
    puts "priv!!"
  end
end
```

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

**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

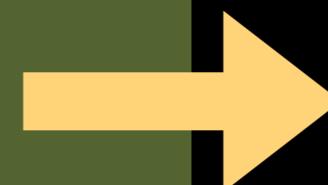
`attr_reader`、書き込みだけを許可する

`attr_writer` も用意されています。

# attr\_accessor

```
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
```

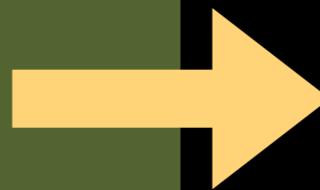
```
recipe = Recipe.new  
recipe.title = "cheese cake"  
p recipe.title  
#=> "cheese cake"
```

# 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
```

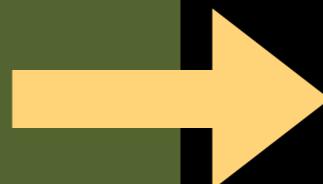
```
recipe = Recipe.new  
p recipe.title
```

# 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"
```

# 継承

既に定義されているクラスを拡張して新しいクラスを作ることを継承といいます。

(既にあるたい焼きの型を利用して、少し違う新しいたい焼きの型をつくるようなものです。)

# 継承

たとえばBookクラスとMagazineクラス  
(雑誌)を作るとします。

```
class Book
  attr_accessor :title, :price
end
```

```
class Magazie
  attr_accessor :title, :price, :number
end
```

別々に定義を書いてもいいのですが、共通項  
もたくさんあります。

# 継承

そんなときは、継承を使うと便利です。

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Magazine < Book  
  attr_accessor :number  
end
```

```
class Magazine  
  attr_accessor :title, :price, :number  
end
```



`class Magazine < Book` と書くことで、Bookクラスを継承したMagazineクラスを定義できます。MagazineクラスはBookクラスの性質を受け継ぎます。何を受け継ぐかを次のページで解説します。

# 継承

## 継承する場合の書式

```
class クラス名 < スーパークラス名  
  クラスの定義  
end
```

スーパークラスとは、継承元の  
クラス(親クラス)です。

継承したクラスは、親クラスの全てのインスタンス変数、メソッドなどを受け継ぎます。

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Magazine < Book  
  attr_accessor :number  
end
```

例えばBookクラスを継承した  
Magazineクラスは、titleとpriceを  
受け継いでいます。例えば、↓のような  
コードを書くことができます。

```
magazine = Magazine.new  
magazine.title = "CanCam"  
p magazine.title #=> "CanCam"
```

# 演習問題1

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに以下のようなfilenameメソッドを実装してください。DigitalBookクラスのインスタンスオブジェクトを作成して、@titleに"Momo"をセットし、filenameメソッドを呼び出してください。

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

```
def filename
  @title + ".pdf"
end
```

# 演習問題2

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに親クラスにあるメソッドと同名のtitleメソッドを実装するとどうなるでしょうか。DigitalBookクラスのインスタンスオブジェクトを作成して、@titleに"Momo"をセットし、titleメソッドを呼び出してください。

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

```
def title
  "[ebook]" + @title
end
```

# 演習問題1 解答

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに以下のようなfilenameメソッドを実装してください。 DigitalBookクラスのインスタンスオブジェクトを作成して、 @title に"Momo"をセットし、 filenameメソッドを呼び出してください。

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

```
class DigitalBook < Book
  def filename
    @title + ".pdf"
  end
end
```

```
ebook = DigitalBook.new
ebook.title = "Momo"
p ebook.filename #=> "Momo.pdf"
p ebook.title #=> "Momo"
```

# 演習問題2解答

以下のBook クラスを継承した DigitalBook(電子書籍) クラスを作ってください。

DigitalBookクラスに親クラスにあるメソッドと同名のtitleメソッドを実装するとどうなるでしょうか。DigitalBookクラスのインスタンスオブジェクトを作成して、@titleに"Momo"をセットし、titleメソッドを呼び出してください。

継承したクラスで親クラスと同名のメソッドがある場合は、メソッド定義が上書きされます。以下のように、Bookクラスには影響は出ません。

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

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

```
class DigitalBook < Book  
  def title  
    "[ebook]" + @title  
  end  
end
```

```
ebook = DigitalBook.new  
ebook.title = "Momo"  
p ebook.title #=> "[ebook]Momo"
```

まとめ

# 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
```

レシピに関するデータと手続きを持ちます。

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。

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  
= Rubyのclassは最初からそうなるように設計されています

# public, private

## メソッドの公開・非公開を制御できます。

```
class AccessTest
  public
    def pub
      puts "public method"
    end
    #ここに書いたメソッドはpublic

  private
  def priv
    puts "private method"
  end
  # ここに書いたメソッドはprivate
end

obj = AccessTest.new
obj.pub #=> "public method"
obj.priv ✗
```

### 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

```
class AccessTest
```

```
  def pub
```

```
    priv # ここは内部
```

```
  end
```

```
private
```

```
def priv
```

```
  puts "private method"
```

```
end
```

```
end
```

```
obj = AccessTest.new
```

```
obj.priv # ここは外部 X
```



## 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、

priv のようにメソッド名だけで呼べると  
ころが内部、

obj.priv 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。

# 継承

## 継承する場合の書式

```
class クラス名 < スーパークラス名  
  クラスの定義  
end
```

スーパークラスとは、継承元の  
クラス(親クラス)です。

継承したクラスは、親クラスの全てのインスタンス変数、メソッドなどを受け継ぎます。

```
class Book  
  attr_accessor :title, :price  
end
```

```
class Magazine < Book  
  attr_accessor :number  
end
```

例えばBookクラスを継承した  
Magazineクラスは、titleとpriceを  
受け継いでいます。例えば、↓のような  
コードを書くことができます。

```
magazine = Magazine.new  
magazine.title = "CanCam"  
p magazine.title #=> "CanCam"
```



# private メソッドの使い方の例

```
# -*- coding: utf-8 -*-
class Omikuji
  def get
    result = rand 3
    if result < 1
      sho_kichi
    elsif result < 2
      chu_kichi
    else
      dai_kichi
    end
  end

  def dai_kichi
    puts "大吉"
  end
  def chu_kichi
    puts "中吉"
  end
  def sho_kichi
    puts "小吉"
  end
end
```

Omikuji オブジェクトがあります。  
getメソッドで呼び出してくれることを期  
待しています。

```
omikuji = Omikuji.new
omikuji.get #=> "小吉"
```

しかし、dai\_kichiメソッドもpublicな  
ので、大吉を出したい人は以下のように書  
けば、ずるができるでしょう。

```
omikuji = Omikuji.new
omikuji.dai_kichi #=> "大吉"
```

# private メソッドの使い方の例

privateを使えば外部からのアクセスを制限できます。

```
# -*- coding: utf-8 -*-
class Omikuji
  def get
    result = rand 3
    if result < 1
      sho_kichi
    elsif result < 2
      chu_kichi
    else
      dai_kichi
    end
  end
```

## private

```
def dai_kichi
  puts "大吉"
end
def chu_kichi
  puts "中吉"
end
def sho_kichi
  puts "小吉"
end
end
```

オブジェクトの内部からはprivateなメソッドも呼び出せる

オブジェクト外部からのprivateメソッド呼び出しはエラーになる

```
omikuji = Omikuji.new
omikuji.dai_kichi ✗
```

# 講義資料置き場

講義資料置き場をつくりました。  
過去の資料がDLできます。

<https://github.com/hitotsubashi-ruby/lecture2012>

or

<http://bit.ly/ruby-lecture>

# 雑談・質問用facebookグループ

facebookグループを作りました

<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
- ・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