

# Ruby 講義

## 第9回 Ruby入門

Kuniaki IGARASHI/igaiga

2012.6.7 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  
@nifty

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

# 破壊的メソッドの説明

## upcase と upcase!

`String#upcase` と `String#upcase!` はどちらも文字列を大文字にするメソッドです。

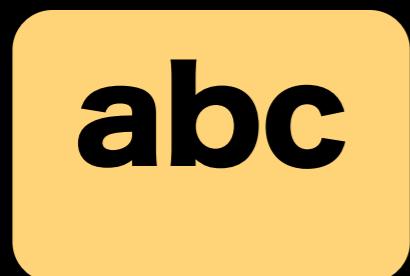
irb で実行するとどちらも同じように見えますが、`!` の有無で何が違うのでしょうか？

```
"abc".upcase #=> "ABC"
```

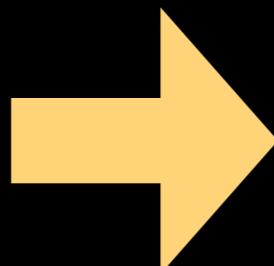
```
"abc".upcase! #=> "ABC"
```

# "abc".upcase!

実行前



upcase!



実行後



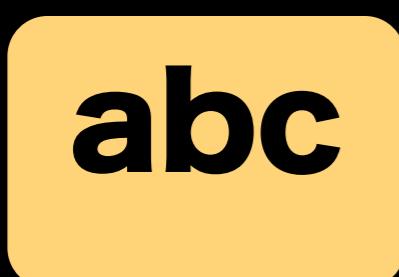
オブジェクト

オブジェクト

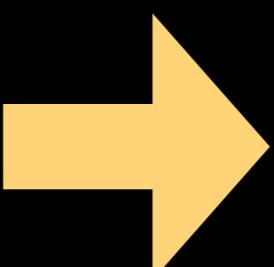
オブジェクト  
が変身する

# "abc".upcase

実行前



upcase



実行後



旧オブジェクト



変換した  
新しい  
オブジェクト  
が複製される

オブジェクト

新オブジェクト

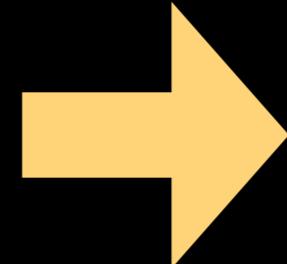
# upcase! を使うコード

```
"abc".upcase!
```

実行前

abc

upcase!



実行後

ABC

オブジェクト  
が変身する

オブジェクト

オブジェクト

```
a = "abc"
```

```
a.upcase!
```

```
puts a
```



# upcase を使うコード

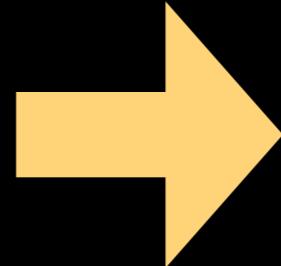
"abc".upcase

実行前

abc

オブジェクト

upcase



実行後

abc

旧オブジェクト

ABC

新オブジェクト

変換した  
新しい  
オブジェクト  
が複製される

a = "abc"

b = a.upcase

puts b

a

abc

a

abc

b

ABC

複製されたオブジェクトを  
別の変数に入れる

# 破壊的メソッド

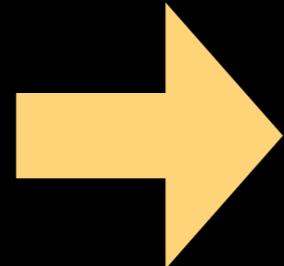
`upcase!` のようにオブジェクトの内容を  
変更するものを「破壊的メソッド」と言います

`"abc".upcase!`

実行前

abc

upcase!



実行後

ABC

オブジェクト  
が変身する

オブジェクト

オブジェクト

メソッド名末尾に `!` がついたら破壊的メソッド  
(ただし、破壊的メソッドでも `!` が付かないものもある)

# 先週の 演習解答

解答は一例です。Rubyのコードはいろいろな書き方が可能です。ここに挙げたコードだけが正解ではないです。  
大切なのは、読み手への心配りと思いやりです。

(いいこと言った)

# 記法の説明

ときどきでてくる

#=>

というマークは実行結果を表します。

p 1+2 #=> 3

p array #=> [1,2,3]

コードではないので打たなくて大丈夫です。もし打ったとしても、#から始まる箇所はコメントとして扱われる  
ので、何も起きません。

# 演習問題 1

Arrayオブジェクトの要素が奇数の項目を全て足すコードを書いてください。  
例えば **array = [2,3,5,7,11]** の場合、 **3+5+7+11 = 26** が表示されれば  
OKです。

ヒント：奇数かどうか調べるのは Fixnum#odd? メソッド

**1.odd? → true, 2.odd? → false**

ちなみに偶数か調べるのは Fixnum#even? メソッドです。

```
array = [2,3,5,7,11]
```

```
sum = 0
```

```
array.each do |i|
```

```
  sum += i if i.odd?
```

```
end
```

```
p sum #=> 26
```

1行でかつこよく書くこともできます。

```
[2,3,5,7,11].select{|i|i.odd?}.inject(:+)
```

# 演習問題 2

```
[{:title => "a", :price => 70},  
 {:title => "b", :price => 200},  
 {:title => "c", :price => 50}]
```

というオブジェクトから、以下のようなオブジェクトを作るコードを書いてください。

```
[{:title=>"a", :price=>70, :special=>"Low price!"},  
 {:title=>"b", :price=>200},  
 {:title=>"c", :price=>50, :special=>"Low price!"}]
```

```
items = [{:title => "a", :price => 70},  
          {:title => "b", :price => 200},  
          {:title => "c", :price => 50}]  
  
items.each do |item|  
  item[:special] = 'Low price!' if item[:price] < 100  
end  
  
p items  
  
#=> [{:title=>"a", :price=>70, :special=>"Low price!"},  
      {:title=>"b", :price=>200},  
      {:title=>"c", :price=>50, :special=>"Low price!"}]
```

# 演習問題 3

あるArrayオブジェクトが与えられたとき、(例えば [6,2,3]) その中で3以下の数字がいくつあるか表示するコードを書いてください。

```
array = [6,2,3]
count = 0
array.each do |i|
  count += 1 if i <= 3
end
p count #=> 2
```

別解 countメソッドにブロックを渡すことでカウントできます。  
ブロックは do end の変わりに {} で書くこともできます。

```
array = [6,2,3]
p array.count { |i| i <= 3 }
```

# 演習問題 4

Hash のバリューの中に！という文字が含まれるときに、そのバリューを大文字に変換したHashを作るコードを書いてください。  
例えば、

```
{:alice=>"yeah", :bob=>"yo!", :linda => "wow!" }
```

↑ というHash を ↓ にできればOKです。

```
{:alice=>"yeah", :bob=>"YO!", :linda => "WOW!" }
```

```
h = {:alice=>"yeah", :bob=>"yo!", :linda => "wow!" }
h.each do |key, value|
  h[key] = value.upcase if value =~ /!/
end
p h
#=> {:alice=>"yeah", :bob=>"YO!", :linda=>"WOW!"}
```

# 演習問題 5

文字列 "write" 中の e を ten に置換し、"written"にするコードを書いてください。  
ヒント：文字列の置換は `String#gsub!`を使います。

```
p "write".gsub!(/e/, "ten") #=> "written"
```

`gsub!`は破壊的メソッドです。対象のオブジェクトを書き換えます。`!のない`  
`gsub` というメソッドもあります。`gsub`は置換した新しい`String`オブジェクトを返します。違いは以下のコードを実行するのが分かり易いです。

```
string = "write"  
string.gsub!(/e/, "ten")  
p string #=> "written"
```

```
string = "write"  
string.gsub(/e/, "ten")  
p string #=> "write"
```

# 演習問題 6

```
text1 = "123"
```

```
text2 = "55"
```

```
text3 = "900"
```

という3つの文字列オブジェクトがあるとき、数値として最も大きいものを表示するコードを書いてください。 (この場合、900 を表示)

考え方の一例：

arrayを作つてこの3つを数値として格納して、maxメソッドを呼ぶと最も大きい数値を返します。

ヒント：文字列オブジェクトを数値オブジェクトにするのはto\_iメソッド

```
"123".to_i #=> 123
```

```
text1 = "123"
```

```
text2 = "55"
```

```
text3 = "900"
```

```
array = []
```

```
array.push text1.to_i
```

```
array.push text2.to_i
```

```
array.push text3.to_i
```

```
p array.max
```

# 演習問題 7

引数にハッシュを受け取るメソッドを書いてください。そのメソッドの中で、引数で受け取ったハッシュにキー :text がなかったとき、"This object does not have :text key." と出力するコードを書いてください。

(例えば、メソッド名を print\_text として、メソッド呼び出し側は以下のようになります。)

```
print_text({:title => "the Ruby book"})
```

```
def print_text(h)
  puts "This object does not have :text key." unless h[:text]
end
print_text({:title => "the Ruby book"})
```

# 演習問題 8

alice,bob,carolの試験の点数が次のようにHashに格納されています。

alice = {**:english => 90, :math => 60, :history => 75**}

bob = {**:english => 50, :math => 90, :history => 80**}

carol = {**:english => 18, :math => 50, :history => 100**}

各科目の平均点を出力してください

alice = {**:english => 90, :math => 60, :history => 75**}

bob = {**:english => 50, :math => 90, :history => 80**}

carol = {**:english => 18, :math => 50, :history => 100**}

**puts "english average", (alice[:english]+bob[:english]+carol[:english])/3.0**

**puts "math average", (alice[:math]+bob[:math]+carol[:math])/3.0**

**puts "history average", (alice[:history]+bob[:history]+carol[:history])/3.0**

かっこいい答え

alice = {**:english => 90, :math => 60, :history => 75**}

bob = {**:english => 50, :math => 90, :history => 80**}

carol = {**:english => 18, :math => 50, :history => 100**}

**students = [alice, bob, carol]**

**[:english, :math, :history].each do |subject|**

**print subject, " average:", students.inject(0){|sum, student| sum + student[subject] } /**

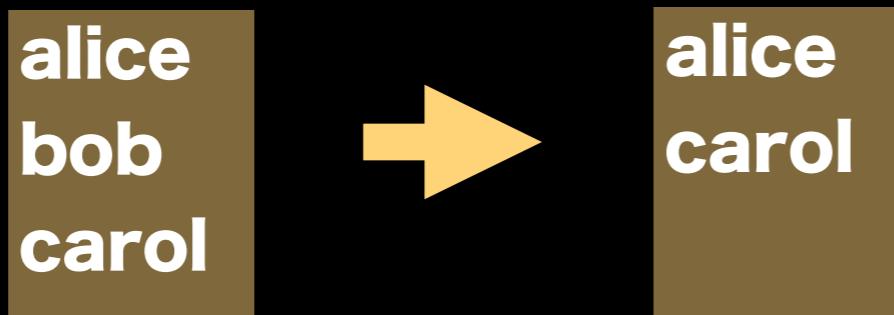
**students.size.to\_f, "\n"**

**end**

# 演習問題【上級】S1 解1

あるテキストファイルから、aという文字列を含む行だけを抽出した別のテキストファイルを作ってください。

例：



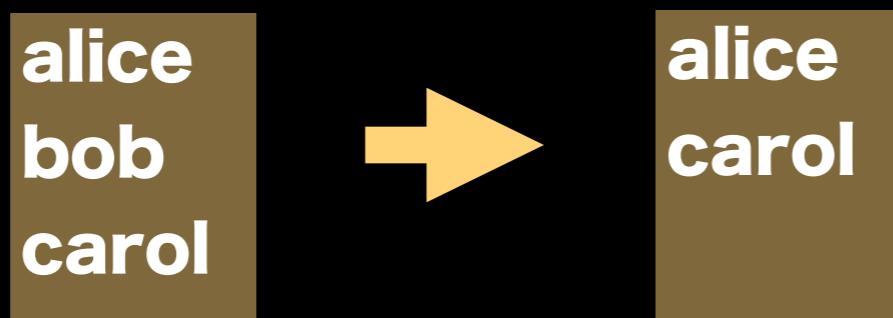
```
# ファイルから読み込み
included = [] # 出力用データを格納するArray
in_filename = 'in.txt'
File.open(in_filename, 'r:UTF-8') do |in_file|
  while text = in_file.gets
    included << text if text =~ /a/
  end
end
# ファイルへ書き込み
out_filename = 'out.txt'
File.open(out_filename, 'w:UTF-8') do |out_file|
  included.each do |i|
    out_file.puts i
  end
end
```

2つのファイルをopenします。  
出力側は"w:UTF-8"(書き込み)  
です。  
また、File.openにブロックを  
渡すとブロック完了時に自動で  
closeします

# 演習問題【上級】S1 解2

あるテキストファイルから、aという文字列を含む行だけを抽出した別のテキストファイルを作ってください。

例：



```
in_filename = 'in.txt'  
out_filename = 'out.txt'  
File.open(in_filename, 'r:UTF-8') do |in_file|  
  File.open(out_filename, 'w:UTF-8') do |out_file|  
    while text = in_file.gets  
      out_file.puts text if text =~ /a/  
    end  
  end  
end
```

in と out を同時に開く書き方です。結果格納用のArrayが不要になるのでスッキリです。

# 演習問題【上級】S2

日本の都道府県で、男性と女性の人口差が最も大きいのはどこか、データから解析してください。

ヒント：データは以下にあります。

<http://www.e-stat.go.jp/SG1/estat>List.do>

男女別人口及び世帯の種類(2区分)別世帯数 の CSV

ヒント：文字例をカンマで区切ってArrayにするには

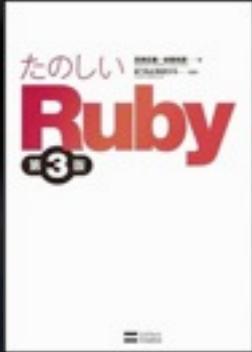
`String#split` を使います。

`"a,b,c".split(",") #=> ["a","b","c"]`

私もまだ書いていないので、誰か書けたらコードを私まで送ってください。 : )

今週

ここから



# 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/>

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。

# Recipeクラスにタイトルを設定できるようにしてみる

次は、タイトルを取得できるようにtitleメソッドを作ります。

※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)
```

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。 recipe\_title 変数へ代入することができました。

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)
```

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)

1. 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。
2. 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。
3. 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メソッドとtitle=メソッドを使って実装してください。)`
4. 同じく `@author` に "Michael Ende" をセットできるようにしてください。
5. 3,4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。

# 演習問題解答

1. 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。
2. `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
puts book.category #=> "novel"
```

# 演習問題解答★TODO

3. Book クラスのインスタンス変数 @title に "Momo" をセットでき  
class Book
4. 同じく @author に "Michael Ende" をセットできるようにして
5. 3,4で作ったクラスのインスタンス変数の @title と @author を表示して  
end

```
book = Book.new
```

```
book.title = "Momo"
```

```
book.author = "Michael Ende"
```

```
puts book.title #=> "Momo"
```

```
puts 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

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!!"
```

次のページでどんな時に使うか見てみましょう。

# initialize 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"
```

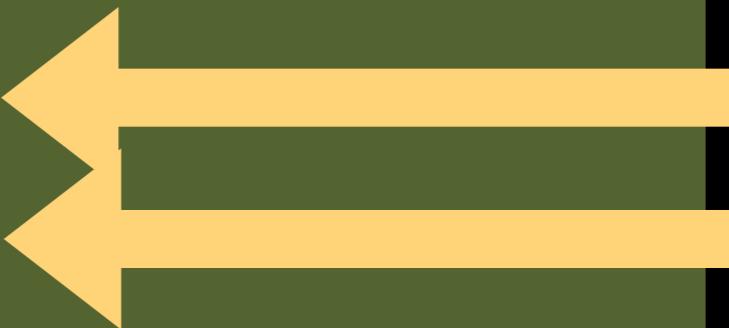
# インスタンスメソッド

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)

# 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
```

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/>

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

**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)
```

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"
```

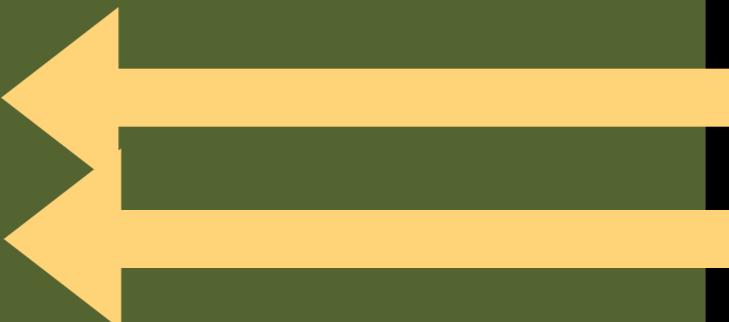
# インスタンスメソッド

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)



# 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