

7.2 オブジェクト指向プログラミングの基礎知識 (P.206～P.211)

- 1.クラスを使う場合と使わない場合の比較
- 2.オブジェクト指向プログラミング関連の用語

7.3 クラスの定義 (P.212～P.220)

- 1.オブジェクトの作成とinitializeメソッド
- 2.インスタンスメソッドの定義
- 3.インスタンス変数とアクセサメソッド
- 4.クラスメソッドの定義
- 5.定数

7.2 オブジェクト指向プログラミングの基礎知識（P.206～P.211）

-1.クラスを使う場合と使わない場合の比較

クラスを使わないユーザープログラム

```
userssss = []
userssss << {first_name:'Alice',last_name: 'Ruby',age:20}
userssss << {first_name:'Bob',last_name: 'Python',age:55}

p userssss
出力結果
# [{:first_name=>"Alice", :last_name=>"Ruby", :age=>20} ,
  {:first_name=>"Bob", :last_name=>"Python", :age=>30}]

userssss.each do |user|
  puts " 氏名:#{full_name(user)},年齢：#{user[:age]}"
  ##puts " 氏名:#{user[:first_name]}#{user[:last_name]},年齢：#{user[:age]}" ##
end

def full_name(user)
  "#{user[:first_name]}#{user[:last_name]}"
end
```

①ハッシュキーをタイプミスしてもエラーが出ない(nil扱い)

```
puts " 氏名:#{userssss[0][:middle_name]}"
出力結果
# 氏名： (空白)
```

userssss	first_name	last_name	age	*	*	*
[0]	Alice	Ruby	20			
[1]	Bob	Python	55			
[*]	error	error	error	error	error	error

こういうテーブルを直接見に行ってやり取りしている印象を受ける

②ハッシュは新しいキーを追加できる

```
userssss[0][:country] = 'japan'
```

③内容を変更できる

```
userssss[0][:first_name] = 'Carol'
```

userssss	first_name	last_name	age	country	*	*
[0]	Carol	Ruby	20	japan		
[1]	Bob	Python	55			
[*]	error	error	error	error	error	error

7.2 オブジェクト指向プログラミングの基礎知識（P.206～P.211）

-1.クラスを使う場合と使わない場合の比較

クラスを使用したユーザープログラム

```
class User
  attr_reader :first_name, :last_name, :age ー~~~~ー~~~~→
  def initialize(first_name, last_name , age)
    @first_name = first_name
    @last_name = last_name
    @age = age
  end
end

userssss = []
userssss << User.new('Alice','Ruby',20)
userssss << User.new('Bob','Python',55)

Userクラスからインスタンス化されたオブジェクトをローカル変数userssssに代入してる。

p userssss
[<User:0x00007f9f69159fb8 @first_name="Alice", @last_name="Ruby", @age=20>,
 <User:0x00007f9f69159f40 @first_name="Bob", @last_name="Python", @age=55>]

userssss.each do |user|
  puts " 氏名:#{full_name(user)},年齢：#{user.age}"
  ## puts " 氏名:#{user.first_name}#{user.last_name},年齢：#{user.age}" ##
end

def full_name(user)
  "#{user.first_name}#{user.last_name}"
end
```

【読み込み専用
メソッド化】
def first_name
 @first_name
end
def last_name
 @last_name
end
def age
 @age
end

①クラスを導入するとデータ参照にメソッドが必要
= タイプエラーするとNoMethodが出る

```
userssss[0].first_name => Alice
userssss[0].middle_nam => NoMethodError
```

②新しくデータを追加したり内容変更を防止することもできる

```
class User
  attr_reader :first_name, :last_name, :age
end
```

このメソッドで読み取り専用になってる
(このアクセサメソッドは後述)

それぞれのメリット/デメリットは自分で考えてください。
わたしはまだイメージが湧かないです。

7.2 オブジェクト指向プログラミングの基礎知識 (P.206～P.211)

-2.オブジェクト指向プログラミング関連の用語

```
class User
  attr_reader :first_name,:age

  def initialize(first_name, last_name , age)
    @first_name = first_name
    @last_name = last_name
    @age = age
  end
end
```

```
userssss = []
userssss << User.new('Alice','Ruby',20)
userssss << User.new('Bob','Python',55)
```

Userクラスからインスタンス化されたオブジェクトをローカル変数userssssに代入してる。

```
p userssss
[#<User:0x00007f9f69159fb8 @first_name="Alice", @last_name="Ruby", @age=20>,
 #<User:0x00007f9f69159f40 @first_name="Bob", @last_name="Python", @age=55>]
```

```
userssss.each do |user|
  puts " 氏名:#{full_name(user)},年齢:#{user.age}"
  ## puts " 氏名:#{user.first_name}#{user.last_name},年齢:#{user.age}" ##
end
```

```
def full_name(user)
  "#{user.first_name}#{user.last_name}"
end
```

取得できるようにするメソッド

→Userクラスの属性(first_name,age)

クラス

Userオブジェクト×2
(Userインスタンス×2)

Userクラスの状態

(first_nameにAliceはってんよ～)

レシーバー メソッド

- ・クラス
- ・オブジェクト
- ・インスタンス
- ・レシーバー
- ・メソッド
- ・メッセージ →テキスト読んで
- ・状態 (ステート)
- ・属性 (アトリビュート、プロパティ)

7.3 クラスの定義 (P.212~P.220)

-1.オブジェクトの作成とinitializeメソッド

クラスがインスタンス化された際に実行されるメソッド

例 1

```
class User
  def initialize
    puts "Userクラスを使ってオブジェクトを作成しました"
  end
end
```

@user = User.new この時点でInitialize発火

出力結果

Userクラスを使ってオブジェクトを作成しました。

例 2 - 引数ありバージョン

```
class User
  def initialize(first_name, last_name , age)
    puts "こんにちは,#{first_name}"
  end
end
```

userssss = []

userssss << User.new('Bob','Python',55) この時点でInitialize発火

出力結果

こんにちは、 Bob

-2.インスタンスメソッドの定義

クラスの内部でメソッド定義するとインスタンスメソッドになる

```
class User
  def こんばんわ
    puts "ぼんそわーる"
  end
end
```

@user = User.new

@user.こんばんわ

出力結果

ぼんそわーる

“こんばんわ”という
インスタンスメソッド作成

7.3 クラスの定義（P.212～P.220）

-3.インスタンス変数とアクセサメソッド

インスタンス変数とは

同じオブジェクトの内部で共有される変数

変数名は@から始める。変数の中身がなくてもnilで返してくる（エラーが出ない）

イン変 例1

```
class User
  def initialize(hello)
    @hello = hello
  end

  def こんばんわ
    puts @hello
  end
end

user111 = User.new('ぼんそわーる')
user111 = User.new('ぐーてんあーべんと')
user111.こんばんわ → (puts @hello)
user111.こんばんわ → (puts @hello)
```

出力結果

```
ぐーてんあーべんと
ぐーてんあーべんと
```

当たり前だけど同じ出力

イン変 例2

```
class User
  def initialize(hello)
    @hello = hello
  end

  def こんばんわ
    puts @hello
  end
end

user111 = User.new('ぼんそわーる')
user222 = User.new('ぐーてんあーべんと')
user111.こんばんわ → (puts @hello)
user222.こんばんわ → (puts @hello)
```

出力結果

```
ぼんそわーる
ぐーてんあーべんと
```

同じクラスからuser111,user222を作ってるけど
オブジェクトは別だから出力結果も別

ローカル変数とは

メソッドやブロック内部(P.125)でのみ有効な変数
変数名は小文字or_(アンスコ)から始める。変数の中身がないとエラーが出る。

ロー変 例1

```
_おひるごはん = "たこやき"

puts _おひるごはん
```

出力結果

```
たこやき
```

ロー変 例2

```
puts _おひるごはん

_おひるごはん = "たこやき"
```

出力結果

```
`<main>': undefined local variable or method
```

7.3 クラスの定義 (P.212~P.220)

-3.インスタンス変数とアクセサメソッド

クラスの外部からインスタンス変数を参照するには下記メソッドを追加してください
イン変の値を読み書きするメソッドをアクセサメソッドという

読み込みするメソッド

attr_reader

```
class User
  attr_reader :first_name
  def initialize(first_name)
    @first_name = first_name
  end
end

user = User.new('Alice')
[#<User:0x00007f9f69159fb8 @first_name="Alice">

puts user.first_name メソッドで読み出しできる
出力結果
Alice
```

この一文で以下メソッドが定義

```
def first_name
  @first_name
end
```

読み書きするメソッド

attr_accessor

```
class User
  attr_accessor :first_name
  def initialize(first_name)
    @first_name = first_name
  end
end

user = User.new('Alice')
[#<User:0x00007f9f69159fb8 @first_name="Alice">

user.first_name = "Bob" いつもの感じで変更可能

puts user.first_name メソッドで読み出しできる
出力結果
Bob
```

この一文で以下メソッドが定義

```
def first_name
  @first_name
end

def first_name=(value)
  @first_name = value
end
```

書き込みするメソッド

attr_writer

```
class User
  attr_writer :first_name
  def initialize(first_name)
    @first_name = first_name
  end
end

user = User.new('Alice')
[#<User:0x00007f9f69159fb8 @first_name="Alice">

user.first_name = "Bob" いつもの感じで変更可能

puts user.first_name 読み込み許可はしていない
出力結果
undefined method (そんなメソッドない)

p user
#<User:0x00007f9f69159fb8 @first_name="Bob">
"user"の中身を見るとちゃんと"Bob"化してる
```

この一文で以下メソッドが定義

```
def first_name=(value)
  @first_name = value
end
```

この三つのメソッドの中で
何が行われているのか知りたい人はP.215~216をお読みください

7.3 クラスの定義（P.212～P.220）

-4.クラスメソッドの定義

メソッド名に self.○○○をつけるとクラスメソッドが作成される

```
class User
  def self.ううう
    puts "わたしはクラスメソッド"
  end

  def たんす
    puts "わたしはインスタンスメソッド"
  end
end
```

“ううう”という
クラスメソッド作成

“たんす”という
インスタンスメソッド作成

User.ううう Userクラスに“ううう”メソッド

@user = User.new
@user.たんす @userインスタンスに“たんす”メソッド

出力結果
わたしはクラスメソッド
わたしはインスタンスメソッド

@user.methods User.methods

```
たんす
instance_variable_defined?
remove_instance_variable
instance_of?
kind_of?
is_a?
tap
instance_variable_get
instance_variable_set
instance_variables
singleton_method
method
public_send
define_singleton_method
public_method
extend
to_enum
enum_for
<=>
===
==
|-
eq?
respond_to?
freeze
inspect
object_id
send
to_s
display
nil?
hash
class
singleton_class
clone
dup
itself
yield_self
then
taint
tainted?
untaint
untrust
untrusted?
trust
frozen?
methods
singleton_methods
protected_methods
private_methods
public_methods
equal?
!
```

```
ううう
allocate
superclass
new
<=>
<=
>=
==
===
included_modules
include?
```

```
name
ancestors
attr
attr_reader
attr_writer
attr_accessor
instance_methods
public_instance_methods
protected_instance_methods
private_instance_methods
constants
const_get
const_set
const_defined?
class_variables
remove_class_variable
class_variable_get
class_variable_set
class_variable_defined?
freeze
inspect
private_constant
public_constant
const_missing
deprecate_constant
include
singleton_class?
prepend
module_exec
module_eval
class_eval
remove_method
<
>
undef_method
class_exec
method_defined?
alias_method
to_s
private_class_method
public_method_defined?
private_method_defined?
protected_method_defined?
public_class_method
public_instance_method
define_method
autoload
autoload?
instance_method
instance_variable_defined?
remove_instance_variable
instance_of?
kind_of?
is_a?
tap
instance_variable_get
instance_variable_set
instance_variables
```

-5.定数

定数は大文字で始めると定義できます
説明はありません

（Rubyの定数は変更できる）

DEF = 100
puts DEF

DEF = DEF * 5
puts DEF

出力結果
rb4 warning: already initialized constant DEF
(警告：定数DEFはすでに初期化されています)
rb1 warning: previous definition of DEF was here
(警告：DEFの以前の定義はここにありました)
100
500

詳細はP.254 - 定数についてもっと詳しく

使い所のイメージつきませんでした。。。
テキストによるとインスタンスに含まれるデータは使わないメソッドを定義したい場合は
クラスメソッドを定義したほうが使い勝手がよくなるらしい