

Rubyレクチャー

第2回

T313084 水野陽斗

まだTeraPadで消耗してるの？

- ・ 学校のPC含めてまさかまだTeraPadとかJCPad使ってる人はいないよね？
- ・ エディタはVimかEmacsを使いこなせるようになったら多分最強（要努力）
- ・ メモ帳とSublime Textで日本語の文章を書くのはやめた方がいいよ
→ Undo(Ctrl+z)とRedo(Ctrl+y)がゴミ

Ruby

- ・ 日本人のまつもと ゆきひろ（通称Matz）が開発
- ・ 純粋なオブジェクト指向のスクリプト言語
- ・ フレームワークのRuby on Railsで一躍有名に
- ・ 日本 Ruby > Python
海外 Ruby < Python
Perlは...

メソッド呼び出しと引数と返り値

- ・ Rubyのメソッド呼び出しの `()` は省略可能
引数があっても省略可能
- ・ メソッド内の最後に評価された式が暗黙的に返される
- ・ なお、Rubyのインデント幅は基本 2 です

```
def sample(val = "hoge")
  if val == "SUKI" # then 省略可
    "好き"
  elsif val+"kamo" == "Sukikamo" # else if じゃない
    "脈あり"
  elsif val == "hoge"
    p "ほげええええええええええ！！！！"
  else
    "無関心"
  end
end
end
```

```
p sample("s") #sample "s"    sample"s"
p sample "suki".upcase # 大文字にするメソッド
p sample"Suki"
sample; sample(); sample 'hoge'
```

全てがオブジェクト

- ・ 整数も実数も文字列も真偽値も配列もハッシュも
ぜー————んぶオブジェクト
- ・ +とかの演算子はそのクラスのメソッド
- ・ 各クラスには大量のメソッドがあって、
欲しい機能は大体標準で実装されている

```
$ irb
$ 1.class 1.0.class 1.class.superclass
$ 1.+(2) # 1 + 2
$ a = [] a.class
$ "".class true.class nil.class
```

配列 (Arrayクラス)

```
ary = (0..10).to_a  
p ary
```

```
ary[1..4] = 100  
p ary
```

```
ary[1, 0] = [50, -3]  
p ary
```

```
ary[5, 5] = []  
p ary
```

```
p ary.max  
p ary.min
```

```
ary.push(-1000, -11)  
p ary
```

```
ary << 69 << 19  
p ary
```

```
p ary.sort  
p ary.sort.reverse
```

```
ary[12] = "last"  
p ary  
p ary.length
```

Rubyの配列の特徴

- ・ インデックスに負の数が指定可能
- ・ 範囲選択が柔軟
 - [n..m] → nからm
 - [n...m] → nからm未満
 - [n,m] → nからm先まで
- ・ 要素の追加と削除が簡単
 - push, << , pop, shift, delete とか
- ・ 異なるクラス（型）が混在可能
- ・ 便利なメソッドが大量にあるよ
 - 自分で実装せずにリファレンスを見よう

%記法

["a", "b", "c"] みたいな配列は打つのだるいじゃん？
そこで↓

```
s = %w(u n k o)
p s
p "[last] is o" if s[-1] == "o" # 後続if文
```

```
# ()じゃなくても可
# %w{a b c} とか %w!a b c!とか %w$a b c$ とか
```

```
t = %W~#{s[-2]} i na ko~
p t
```

```
# w, W以外も色々あるよ
```

ループ処理

```
a = (1..10).to_a # (1..10)という範囲オブジェクトを配列に変換
sum = 0
for i in (0...a.size) do # forに{}は使えない
  sum += a[i] # iのスコープはブロック外にまであるため注意
end
```

```
j = 0
sum = 0
while j < a.size do
  sum += a[j]
  j += 1
end
```

```
sum = 0
a.size.times { |k| sum += a[k] }
```

```
sum = 0
a.each { |n| sum += n } #配列の合計求めるならa.inject(:+)で一発
```

他にも until, upto, downto, loop などなど
制御にはbreak, next, redo, retry などなどあるよ

演習

- ・ 九九表を表示するプログラムを作成せよ
- ・ 範囲オブジェクトとeachで書いてほしい
- ・ 出力を整形したい場合
 - `print "%3d" % [i*j]`
c言語の `sprintf` と同じ感じ

解答

```
(1..9).each { |i|  
  (1..9).each { |j| print "%3d" % [i*j] }  
  puts  
}
```

```
#1.upto(9) { |n| puts "%3d"*9 % [*n.step(n*9,n)] }  
#kuku = (1..9).map { |a|(1..9).map { |b| a*b} }
```

do endと{ }の違い

- ・ 結合強度が違うため動作が変わることがある
→ 詳しくはググってください
- ・ Rubyは基本的に do end を使う
- ・ 1行の場合は { } 使うことが多い感じ
→ コーディング規約はググってください
- ・ ブロックもメソッドの引数です

破壊的メソッド

```
alphabet = ('a'..'z').to_a  
p alphabet
```

```
random = alphabet[0..-20]  
p random  
random.shuffle # オブジェクトは変わらない  
p random  
random.shuffle! # オブジェクトは変わる  
p random  
#pushなど!が付かなくても変更するメソッドあり
```

演習

- ・ 各要素が英大文字の"J"から"R"の配列 `x` を作成
- ・ 以下のメソッド作成
 - `x` と整数 `n`, `m` の3つを引数に取る
 - `x` の `n`番目から `m-1`番目までの要素を"!"に変更
 - `x`の各要素を逆順で連結した文字列を返す
- ・ ヒント
 - 2回（2重じゃない）ループを使おう
 - ループには`each`メソッドを使うのが楽

解答

```
def connect(a, n=0, m=1)
  (n...m).each { |i| a[i] = "!" }
  # for i in n...m do a[i] = "!"end#←forで書く場合
  # a.fill(n...m) {"!"} #← ループなしでもいける
  p a
  c_string = ""
  a.reverse.each { |char| c_string += char }
  c_string
  # a.reverse.join ↑3行無しでこれだけでいけちゃう
end
```

```
x = ('J'..'R').to_a
p x
p connect(x, 2, 6)
# 結局は下の一行だけでいい
# p ('J'..'R').to_a.fill("!",2...6).reverse.join
```


ハッシュ (連想配列)

- ・ 配列のインデックスが整数じゃないやつ
- ・ キー (key) と値 (value) のセット
- ・ Rubyではキーにシンボルを使うのが一般的
- ・ 大体の言語にあります
JavaとC++にも勿論あるよ

```
member = { kawa: "リア充", :shima => "いずれできるさ",  
           "kuri" => "ライバーかな?" } # 3種類の定義方法
```

```
p member[:kawa] # :から始まるのはシンボル
```

```
p member.size
```

```
p member.values
```

```
p member.keys
```

```
p "sawaはいません" unless member.has_key?(:sawa) #後続unless if !=  
member[:sawa] = "もっと気軽に喋ってくれてええんやで" # 追加
```

```
p "sawaはいます" if member.has_key?(:sawa) #後続if
```

```
p "2年生にコミュ障は多分いません(3年生には...)" unless  
member.has_value?("コミュ障")
```

```
member[:shima] = "公務員" # 代入
```

```
member.each do |key, value| # 2つ指定
```

```
  puts "#{key}は#{value}"
```

```
end
```

```
member = member.to_a # 配列に変換
```

```
p member[0]
```

```
p member = member.to_h # ハッシュに変換
```

```
p member[:kawa]
```

クラス

- ・ クラスは後から拡張可能
組み込みクラスですら拡張可能
→ モンキーパッチというらしい
- ・ 継承元を動的に決めることも可能
- ・ Rubyにオーバーロードはない

組み込みクラスの拡張

```
class Object
  def oppai # 追加
    'おっぱい'
  end
  def class # 上書き
    puts 'うんこぶりぶり'
  end
end
```

コマンドラインで↓

```
$ irb
```

```
$ require "./Object"
```

演習

- ・ C++の時に作ったAnimalクラスのクラス名をCreatureにし、Rubyに書き換えてください
- ・ アクセス修飾子は無視して構いません
- ・ nameだけセッターとゲッターを作成してください（アクセサの方が良い）
- ・ 分からなかったら先週の資料も参考に

解答

```
class Creature
  def initialize(name="不明", age=20, weight=60.0, kind="不明")
    @name = name
    @age = age
    @weight = weight
    @kind = kind
  end

  attr_accessor :name#, :age, :weight, kind
  # def name
  #   @name
  # end
  # def name=(n)
  #   @name = n
  # end

  def show_data
    puts "名前:#{@name}, 年齢:#{@age}, 身長:#{@height}, 体
重:#{@weight}, 種類:#{@kind}"
  end
end
```

演習

- ・ さっきのCreatureクラスを継承してPersonクラスを作成
- ・ インスタンス変数に職業の @job, 身長の @height を追加
- ・ イニシャライザは適宜変更
- ・ show_dataはオーバーライドして職業も表示
- ・ 笑い声を出力するメソッドlaughを作成
- ・ 継承の仕方
class 派生クラス < スーパークラス
違うファイルに書く場合 require './Creature' で読み込み

解答

```
class Person < Creature
  def initialize(name="佐藤", age=20, weight=60.0,
               kind="人間", height=165.0, job="サラリーマン")
#super だけだと引数をそのまま渡す ()付けると引数を渡さない
    super(name,age,weight,kind)
    @height = height
    @job = job
  end

  def show_data
    puts "名前:#{@name}, 年齢:#{@age}, 身長:#{@height}, 体
重:#{@weight}, 種類:#{@kind}, 職業:#{@job}"
  end

  def laugh
    puts "( °∀°)アハハハハハハノヽノヽノヽノヽノヽノヽノヽノヽノ"
  end
end
```


おまけ 先週の演習 俺の解答

```
class Car
  MAX = 5 # 定数

  def initialize
    @count = 0 # 要素数のカウンター
    @car = [] # Array.new()
    # Array.new(5) や Array.new(5){Hash.new()} は show_car でエラーになるためNG
  end

  # 入力された値番号で削除対象を決めるため配列にハッシュを追加
  def add_car(name, price)
    if @count < MAX
      @car[@count] = {name.to_sym => price} # ハッシュ追加
      @count += 1
    else
      # shiftで要素が減るが代入（追加）で要素が増えるため@countの値はそのまま
      @car.shift # 先頭要素削除
      @car[@count-1] = {name.to_sym => price} # 末尾にハッシュ追加 [@MAX-1]とどっちがいいのかな？
    end
  end

  def delete_car(select)
    if 0 < select && select <= @count #入力が0より大きく要素数以下なら
      puts "#{@car.delete_at(select-1)}を削除しました" # delete_at()の戻り値は消した要素
      @count -= 1
    else
      puts "消せません"
    end
  end

  def show_car
    @car.each_with_index do |item, number| #配列に対してのeach_with_index
      item.each do |key,value| #配列の要素 = item = ハッシュに対してのeach
        puts "#{number+1} 車名:#{key} 価格:#{value}"
      end
    end
  end
end
```

おまけ 続き

```
require './Car'

car=Car.new()

loop do # 無限ループ
  print('1:add 2:delete 3:show 4:finish > ' )

  case gets.to_i
  when 1 then
    print 'carname > '
    carname = gets.chomp
    print 'price > '
    price = gets.to_i
    car.add_car(carname, price)

  when 2 then
    car.show_car
    print('deletenumber > ')
    car.delete_car(gets.to_i)

  when 3 then
    car.show_car

  when 4 then
    break

  else
    puts "範囲外の入力です"
  end

  puts
end
```

その他やるうかなと思ったこと

- ・ アクセス修飾子
- ・ nilとfalse以外は "" や 0 も全てtrue
- ・ nil
- ・ 三項演算子（条件演算子）
- ・ **でべき乗
- ・ Stringを*で繰り返し
- ・ 大文字開始で定数
- ・ 1/3rで分数
- ・ 多重代入
- ・ ||, ||=, ==, <=> などの演算子
- ・ スペース、改行に意味がある
- ・ whenの後は複数指定できる
- ・ eachやtimesメソッドの |変数| はなくてもいい

Rubyでためになるサイト

- ・ ドットインストール
言わずと知れた神サイト
dotinstall.com/lessons/basic_ruby_v2
- ・ ミニツク - Rubyのe-ラーニング研修システム
<http://www.minituku.net/>
- ・ Ruby 概論のスライドが素晴らしかったのでまとめてみた(第1部)
- 酒と泪とRubyとRailsと
この人のサイトは色々なページがかなりためになるよ
<http://morizyun.github.io/blog/ruby-generalization-hitotsubashi-univ-1/>