かんたん♪Ruby入門

in 新潟のRubyistイベント

あじぇんだ

前半

RubyでYes!エンジョイプログラミング

後半

RubyでYes!エンジョイWebプログラミング

説明資料環境は Ruby 1.9.3

資料では説明不足な点が多くあるので、疑問に思ったところは尋ねてください

あじぇんだ

- **RubyでYes!エンジョイプログラミング**
  - なぜRuby？
  - 基本的なこと
  - メソッド定義, イテレータ
  - スコープ
  - クラス

なぜRuby?

なぜRuby?

- Ruby講習会にいるからRubyをやります
- Rubyを書くことは楽しい

> “機能や効率性を求めているからといってにRubyでプログラミングをしないでください。使う「べきだ」と考えているからといってRubyでプログラミングをしないでください。好きだったらRubyでプログラミングをして下さい。もし好きでなければ、止めて下さい。”

なぜ、私はRubyでプログラムを書くのか？（そして、なぜ、あなたはRubyを使わない方がいいのか？）より
http://eitoball.blogspot.jp/2009/11/giles-bowkettwhy-i-program-in-ruby-and.html

理屈はよくわからないが、Rubyの風を感じる

なぜRuby?

楽しいよと言われても、具体的なものがないと
取り組む気は起きない

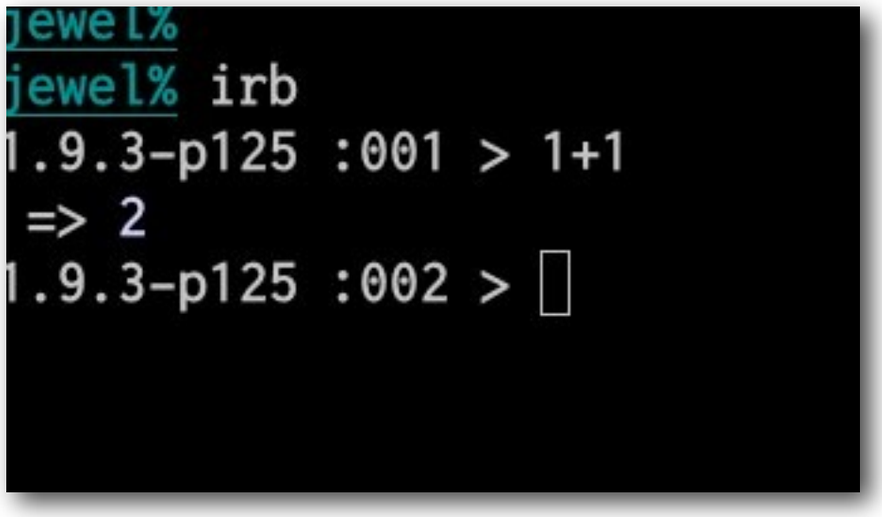
もう少し具体的に

http://www.ruby-lang.org/ja/about/

基本的なこと

irb

- **Interactive Ruby**
  - 1+1 とかを入力すると、すぐに評価されて 2 が返ってくる
  - Rubyの動作を確認するときに便利



Hello World!

- **好きなエディタで「print “Hello World”」と書いて「hello.rb」という名前で保存しよう。**
  - 括弧つけなくていい
  - 余計なセミコロンもいらない
- **端末で「ruby hello.rb」すると「Hello World」と出力される。**
  - puts “Hello World” とすると改行される。

コメント

- # 以降がコメント
- 複数行のコメントは

=begin
coment...
=end

変数

- **型宣言は無い**
- **最初の１文字で変数の種類を区別**
  - kuma => ローカル変数
  - Kuma => 定数
  - $kuma => グローバル変数

```
a = 8
b = 2
a + b #=> 10

a = 8
a = "kuma"
a #=> "kuma"
```

文字列

シングルもしくはダブルクォートで囲むと文字列

```
str1 = "Hello"
str2 = "World"

str3 = str1 + str2 #=> "HelloWorld"

a = 2012
"#{str3} #{a}" #=> "HelloWorld 2012"
```

- 文字列を「+」で連結
- 変数展開は #{}

文字列

```
str = "animation"
str[2..5] #=> "imat"
str.reverse #=> "noitamina"
```

- **[2..5] の部分**
  - 部分文字列を取り出す
- **str.reverse**
  - 文字列をひっくり返す

クイズ: str = “sukiyaki” から
“ikayaki” を作る

文字列

回答

str.reverse[0..2] + str[4..-1] とか

メソッド呼び出し

文字列(String)は

- **reverse** (文字列をひっくり返す)
- **chop** (最後の一文字を除去)

というメソッド(関数)を持つ

こう書くとする↓

String#reverse, String#chop

メソッドは . (ドット)で繋いで呼び出す

“Yamada”.reverse => “adamaY”

メソッド呼び出し

メソッドチェイン

ドットで繋ぐことで、返ってきたオブジェクトのメソッドを呼べる

“Yamada”.reverse.chop

“Yamada”.reverse

↓

“adamaY”.chop

↓

“adama”

配列

普通に生成・参照

```
arr = [1,2,3,4]

arr[2] #=> 3
arr[-1] #=> 4 最後の要素から数える

arr = %w(a b c d) #=> ["a", "b", "c", "d"]

arr[1..2] #=> ["b", "c"] 部分的な配列を生成
```

配列

要素の追加と結合

```
arr = [1,2,3]

arr << 2 #=> [1,2,3,2] 要素の追加

# いろいろな種類のオブジェクトを要素にできる
arr << "kuma" #=> [1,2,3,2,"kuma"]

# 結合
arr + [4,5,6] #=> [1,2,3,2,"kuma",4,5,6]
```

配列

配列の要素を１つずつ取り出す

```
foods = %w(yakiniku sukiyaki sushi)

foods.each do |food|
  puts "#{food} LOVE!!!"
end

#=> yakiniku LOVE!!!
#=> sukiyaki LOVE!!!
#=> sushi LOVE!!!
```

do ~ end は { ~ } とも書ける

ハッシュ

- どんなオブジェクトでもキーにできる
- :文字列 はシンボルと呼ばれ、文字列を直接キーにする代わりに使用される

```
h = {:kuma => 2, :sukiyaki => 12}

h[:kuma] #=> 2

h["usagi"] = "Rabbit"
h[ [1,2] ] = "Array"
#=>
{:kuma => 2, :sukiyaki => 12,
  "usagi"=> "Rabbit",
  [1,2] => "Array"}
```

制御構文

- If

```ruby
if x > 10
  puts "#{x} is bigger than 10"
elsif x == 10
  puts "#{x} is 10"
else
  puts "#{x} is smaller than 10"
end
```

- 後置if

puts “Niigata!” if place == “Niigata”

true / false / nil

- **true / false**

  5.odd? #=> true

  5.even? #=> false

「?」がメソッド名の最後につく場合は、true/false を返すしきたりがある。

- **nil (何者でもないオブジェクト)**

  arr = %w(a b c)

  arr[1] #=> “b”

  arr[3] #=> nil

true / false / nil

nil / false 以外のオブジェクトは全て真

```
arr = %w(a b c)
puts “YES!!” if arr[1] #=> “YES!!”
puts “YES!!” if arr[3] #=> nil
puts “YES!!” if 5.odd? #=> “YES!!”
puts “YES!!” if 5.even? #=> nil

x = 10 #=> 10
x == 10 #=> true
puts “YES!!” if x = 5 #=> “YES!!”
```

制御構文

- **Unless**
  - 評価結果が真でないときに実行

```
unless x >= 10
  puts  “#{x} is smaller than 10”
end

puts  “#{x} is not 10”  unless x == 10
```

- **break**
  - 最も内側のループを抜ける

```
5.times do |i|
  break if i.odd?
  puts i
end
#=> 0 のみ出力
```

制御構文

- **next**
  - 次の繰り返しへ

```
5.times do |i|
  next if i.odd?
  puts i
end
#=> 0, 2, 4 が出力
```

- **for, while もちゃんとあります。**

ファイル入出力

- **テキストファイルを読み込んで内容を出力**

```
puts File.read( “niku.txt” )
```

とか

```
puts open( “niku.txt” ) {|f| f.read}
```

ファイル入出力

- テキストファイルを１行ずつ読み込む

```
open(  “niku.txt”  ) do |file|
  while line = file.gets # １行ずつ読み込む
    puts line
  end
end # ここでファイルが閉じられる
```

file.each {|line| puts line} とも書ける

ファイル入出力

- ファイルに書き込む

```
open(  “niku.txt” ,  “w”  ) do |f|
  f.puts  “yakiniku”
end
```

など

チャレンジ！

標準入力から文字列を読み込む度に「a」が何文字含まれているか出力する。「exit」と入力されたら終了。

Hints:

- gets で入力された文字列を読み込む
- chomp/chomp! で文字列末尾の改行を除去
- String#count, String#scan などを使う

メソッド・クラス等

メソッド定義

```
def add_emoticon( str )
  return str + “( ^ _ ^ )”
end

add_emoticon( “はい” ) #=> “はい ( ^ _ ^ )”
```

関数は最後に評価した結果を返すので、この場合 return はなくても良い。お好みで。

イテレータ

yield でメソッドに渡された処理(do ~ end で囲まれた部分)に引数を渡して実行できる

```
def say_niku_tabe_tai
  yield “肉！”
  yield “食べ！”
  yield “たい！”
end

say_niku_tabe_tai do |str|
  puts str
end
```

スコープクイズ1

スコープ: 変数が参照できる範囲

```
x = 1
if true
  x = 2
  y = 3
end

x #=> ?
y #=> ?
```

スコープクイズ1

- if や unless はスコープを作らない

```
x = 1
if true
  x = 2
  y = 3
end

x #=> 2
y #=> 3
```

スコープクイズ2

```
x = 1


def check_scope
  x += 1
  y = 3
end


x #=> ?
y #=> ?
```

スコープクイズ2

```
x = 1
def check_scope
  x += 1 #=> x が見つからないのでエラー
  y = 3
end
check_scope

x #=> 1
y #=> y が見つからないのでエラー
```

スコープクイズ3

```
x = 1
[1,2,3].each do |i|
  y = 3
  x = i + y
end
```

x #=> ?

y #=> ?

i #=> ?

スコープクイズ3

繰り返し処理の中で使用された変数は外から見えない。繰り返し処理の中から外の変数は見える。

```
x = 1
[1,2,3].each do |i|
  y = 3
  x = i + y
end
x #=> 6
y #=> 見つからない
i #=> 見つからない
```

class

```
class Kuma
  # インスタンスを生成した時に実行される
  def initialize( age )
    @age = age # @をつけるとインスタンス変数
  end

  def introduce_myself # インスタンスメソッド
    puts “#{@age}歳のクマです。”
  end
end

kuma = Kuma.new( 10 )
kuma.introduce_myself
```

classその他

・クラス変数は @@age と @@ を変数名の頭につける

・クラスメソッドは def self.say と self を使って宣言できる

・継承は class Kuma < Mammal と書く

・多重継承(Mix-in)は module を include する形で実現

自画自賛するArray

```
class SugoiArray < Array
  def ziga_zisan
    puts “すごい”
  end
end

sugoi_arr = SugoiArray.new( 3, “kuma” )
#=> [ “kuma” , “kuma” , “kuma” ]

sugoi_arr.join #=> “kumakumakuma”
sugoi_arr.ziga_zisan #=> “すごい”
```

自画自賛するArray

モンキーパッチ(↓は推奨されないやり方)

```
class Array
  def ziga_zisan
    puts “すごい”
  end
end

arr = Array.new( 3, “kuma” )

arr.ziga_zisan #=> “すごい”
[1,2].ziga_zisan #=> “すごい”
```

危険な柔軟さ。柔軟性と危なさのバランスは開発者が判断する。

RubyでYes!エンジョイ Webプログラミング

あじぇんだ

- RubyでYes!ウェブプログラミング
  - Sinatraとは
  - 出席簿を作ろう

やること

簡単なウェブアプリケーションフレームワーク(WAF)を使用して何か作る

WAF: Sinatra
テンプレートエンジン(HTML): Haml

Sinatraのインストール

WEBアプリケーションを手早く作るための
DSL(ドメイン固有言語)

DSLであることを意識する必要はない。Ruby
はRuby

gem install sinatra

RubyGemsはRubyライブラリのパッケージ管
理システム

Sinatraアプリの例

- **TRENDONAU**
  - http://donau.heroku.com/
- **OtaQ**
  - http://otaq.jewelve.com/
- **突然死ぬゲーム**
  - http://totsuzen4.jewelve.com/

個人制作のアプリケーションにはもってこい

ディレクトリ構成

- **SugoiApuri**
  - ワーキングディレクトリ
- **/views**
  - テンプレートのディレクトリ
- **app.rb**
  - アプリ本体

ハローワールド

app.rb

```
# coding: utf-8
require “sinatra”

get “/” do
  “こんにちは！！！！”
end
```

ruby app.rb でサーバが立ち上がる。

http://localhost:4567/ にアクセス！

Haml を使う

- **HTML等を生成するためのマークアップ言語**
  - Haml の他にも Slim 等がある

変換前 (Haml)

```
!!!
%html
 %head
  %title Hello, Haml!
 %body
  #header
   %h1 Hello, Haml!
  #content
   %p
    I use Haml
    %span.version= Haml::VERSION
```

変換後 (Html)

```
<!DOCTYPE html>
<html>
 <head>
  <title>Hello, Haml!</title>
 </head>
 <body>
  <div id='header'>
   <h1>Hello, Haml!</h1>
  </div>
  <div id='content'>
   <p>
    I use Haml
    <span class='version'>3.1.0.alpha.147 (Bleeding Edge)
</span>
   </p>
  </div>
 </body>
</html>
```

日本Haml の会より
(http://haml.ursm.jp/)

Hamlを使う

インデント == タグのネスト
ということを覚えておく

gem install haml

例（app.rb）

```
# coding:utf-8
require 'sinatra'

get “/” do
  @message = ”こんにちは”
  haml :index
end
```

※ get “/hello” do ~~ end と書くと、
http://localhost:4567/hello に
GETでアクセスした時、実行される

例（Haml）

```
!!!
%html
  %head
    %meta{'http-equiv' => 'Content-Type',
          :content => 'text/html', :charset => 'UTF-8'}
    %title SUGOI APURI
  %body
    %h1 すごいウェブアプリケーション Ver. 0.000001
    %p= @message
```

index.haml として views 以下に保存

Rubyコードの埋め込み

- **評価結果がタグの内容になる**

%p= “ただいまの時刻は#{Time.now.to_s}”

- **繰り返し**
  - ハイフンからはじめる
  - インデントがブロックの範囲
  - end は不必要

- @names.each do |name|
  %p= name

出席簿を作ろう

すごいウェブ出席簿

出席!

出席者一覧☆彡

名前	出席時刻
山田	Fri Jun 22 17:56:03 2012
罷	Fri Jun 22 17:56:11 2012
鹿肉	Fri Jun 22 17:56:43 2012

Haml側

さっきの Haml の %p 以降を↓に書き換え

```
%form{:action => "/", :method => "post"}
      %input{:type => "input", :name => "name"}
      %button 出席!
    %p 出席者一覧☆彡
    %table{:border => 1}
      %tr
        %th 名前
        %th 出席時刻
      - @attendances.each do |attendance|
        %tr
          %td= attendance[:name]
          %td= attendance[:time].strftime( "%c" )
```

流れ星とかは要らないです

app.rb側

```ruby
attendances = []

get '/' do
  @attendances = attendances
  haml :index
  # テンプレート内でのローカル変数を明示したい時。
  # @attendances = attendances が要らなくなる。
  # haml :index, :locals => { :attendances => attendances }
end

post '/' do # post のときはこちらが呼ばれる
  attendances << {
    :name => params["name"], # フォームのパラメータは params で取得できる
    :time => Time.now
  }

  @attendances = attendances
  haml :index
end
```

注意

再起動するとデータが消えます！！！！

COOOOOL!!!!

おすすめの本とか

ナウい環境

調べればナウい環境がいっぱい出てきます

使用したことのあるツール等（もはやナウくない）

- **rvm**
  - バージョンを切り替えたり
- **RSpec**
  - テスト
- **Bundler**
  - アプリケーションごとのライブラリの管理

おすすめの本

初めて勉強した本

Rubyプログラミング入門

(手元にないので未チェックですが、内容が古くなっているかも)

Rubyをちゃんと使うようになったら……

メタプログラミング Ruby

と

Ruby ベストプラクティス

もし時間が余ったら

- **Sinatra アプリを改造**
  - DBに保存できるように
  - Web API を組み合わせて
- **個人的なアプリケーションの作成**
  - Twitter TL のフィルタリング
  - あみだくじを作るのが一部で流行ってたらしい