**STEP1 : Học Ruby**

**Ngày 17/8/2015**

Từ hôm nay thì mình sẽ thực hiện việc học Ruby theo giáo trình Video của Chicken Rain. Mình sẽ ghi chú lại những gì mình học và hiểu được, đồng thời sẽ đưa ra comment

**Video1.Giới thiệu môi trường IRB**

Đây là môi trường có tác dụng giúp ta có thể thực thi trực tiếp các dòng lệnh tức thời mà ko cần phải tạo file và chạy file đó. Để vào môi trường irb này thì trong commandline ta chỉ cần gõ lệnh

>irb

Và nhấn Enter, như vậy là mình đã tạo ra môi trường irb

Lệnh puts được sử dụng để in ra giá trị

>puts “Xin chao cac ban” (ấn Enter)

“Xin chao cac ban”

>nil (nil nghĩa là khi ta in ra 1 biến thì giá trị đã được hiển thị ở phía trên và ko có giá trị nào trả về - trả về giá trị rỗng)

Nếu muốn thoát khỏi môi trường irb thì chỉ cần gõ

>exit

Hoặc là tắt màn hình commandline đi hoặc tổ hợp phím ctr + D

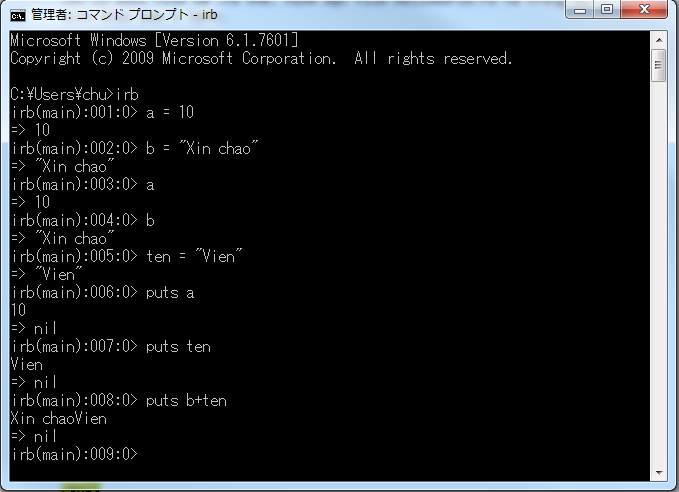
Video2.Biến trong Ruby

Tham khảo: <http://alo30s.com/ruby-bai-4-cach-su-dung-bien-va-tim-hieu-mot-so-quy-dinh-trong-ruby/>

Sử dụng môi trường irb để thực hành phần này thì trong Ruby và kết quả như sau

Biến trong Ruby không cần khai báo kiểu mà chỉ cần:

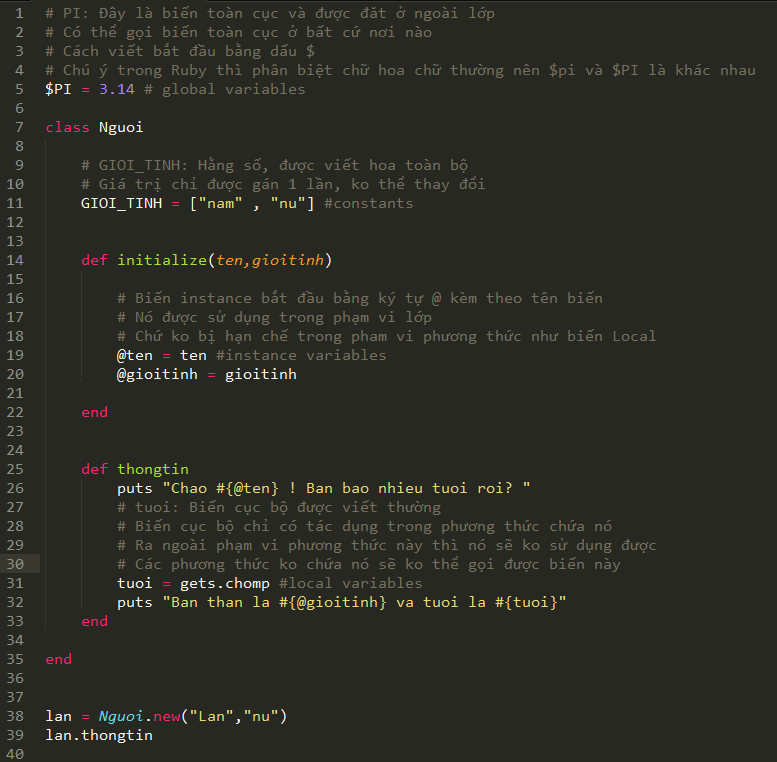
Ten\_bien = gia\_tri\_khoi\_tao



Chú ý khi code Ruby trong file thì cần khai báo biến và lệnh rõ ràng chứ ko thể viết trực tiếp như trong irb

|  |
| --- |
| a = 10  a (X) // Sẽ chạy được trong irb nhưng viết trong file thì không có ý nghĩa và khi run nó sẽ ko hiển thị kết quả  puts a (O) // Sử dụng cả trong irb và viết file trực tiếp với yêu cầu in ra giá trị của biến khai báo sau nó. Muốn in ra chuỗi thì ta sử dụng toán tử “+” để thực hiện ghép chuỗi như trong ví dụ |

Các loại biến trong Ruby



Không cần quan tâm nội dung ví dụ mà chỉ cần quan tâm các biến được khai báo bên trong ví dụ và thuyết minh về cách khai báo cũng như phạm vi sử dụng của chúng.

**Biến toàn cục (Global Variables)** được đặt ở ngoài class và có thể được gọi bất cứ nơi nào, và cách viết được bắt đầu bằng dấu “$” + tên biến và tên biến thì phân biệt hoa thường.

**Hằng số (Constrants)** được viết in toàn bộ và giá trị chỉ được gán một lần và không thể thay đổi.

**Biến cục bộ (Local Variables)** được viết thường và biến này “chỉ có” tác dụng trong phương thức chứa nó. Nếu những phương thức không chứa nó sẽ không gọi được biến này.

**Instance Variables( Biến dùng trong Class)** được viết bắt đầu bằng ký tự “@”, biến có tác dụng trong toàn lớp chưa nó.Như vậy thì biến instance sẽ có phạm vi sử dụng rộng hơn so với biến Local Variables.

Để có thể hiển thị các loại ngôn ngữ ko thông dụng như tiếng Việt, tiếng nhật,.. thì việc hiển thị trên command prompt sẽ phát sinh lỗi ko đọc được. Vậy để khắc phục việc này thì ta phải sử dụng KCODE của Ruby. Ta sẽ phải chạy file ở dạng utf8

>ruby –Ku tên\_file.rb

Tham khảo: <http://www.rubylife.jp/ini/japan/index3.html>

# Thiết lập KCODE để cmd có thể định đạng và đọc được những ký tự ko phải tiếng anh

# Sử dụng nếu trong code có tiếng Nhật, tiếng Việt

$KCODE = "utf-8" # $KCODE = "u" cũng có tác dụng tương tự

vien = "Viên"

puts vien

Nếu khai báo như trên thì ko cần lệnh –Ku trong cmd

**Video3.Các câu lệnh đơn giản và thường gặp trong Ruby**

Ở bài trước mình đã giới thiệu cho các bạn cách gán biến trong Ruby, ở bài này mình sẽ giới thiệu cho các bạn các câu lệnh đơn giản và thường gặp trong Ruby.

**Gán biến hoặc là gán giá trị**

Ví dụ ta có biến a và ta muốn gán giá trị là 10 thì ta dùng lệnh

a = 10

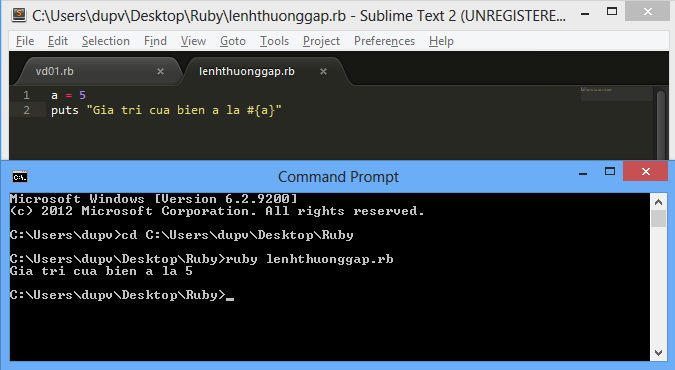
**Xuất chuỗi hoặc giá trị ra màn hình**

Để xuất chuỗi hoặc giá trị ra màn hình các bạn dữ dụng câu lệnh **puts**

|  |
| --- |
| a = 5  # Xuất ra giá trị của 1 biến  Puts a  #Xuất ra chuỗi kèm theo giá trị của 1 biến  puts "Gia tri cua bien a la #{a}" |

Để in ra giá trị của biến trong chuỗi thì ta sử dụng dấu #{tên\_biến}

Sau khi chạy file **Ruby** các bạn sẽ có kết quả như hình bên dưới.

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-01.png)

##################################################################

## Phân biệt puts và print trong Ruby

##################################################################

# puts là lệnh in ra màn hình, nó có thể in tất cả những gì được khai báo ngay sau nó

# puts sẽ tự động xuống dòng cứ mỗi khi kết thúc một chuỗi String

puts "Say","Hello","to","the","World"

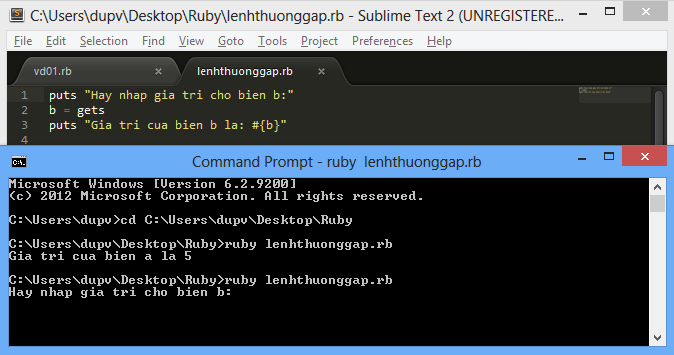
# print cũng có chức năng in ra màn hình, nhưng khác với puts là nó sẽ cho in liền các đoạn string được khai báo

# Và chỉ xuống dòng khi có ký tự xuống dòng

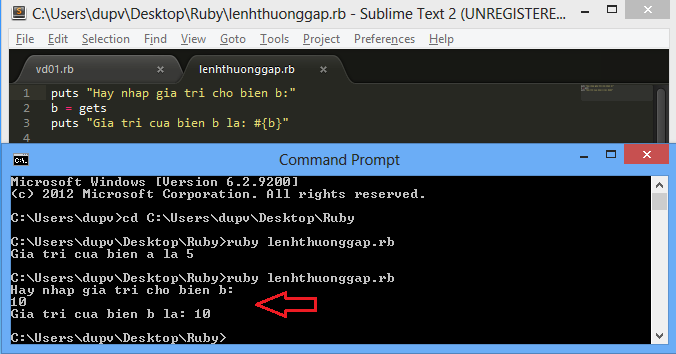
print "Say","Hello","to","the","World \n"

**Lấy giá trị nhập từ bàn phím**

Để lấy giá trị nhập từ bàn phím ta dùng lệnh **gets** ở ví dụ dưới đây mình sẽ cho hiển thị một chuỗi và bắt người dùng nhập giá trị vào và xuất ra chuỗi mới.

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-02.png)

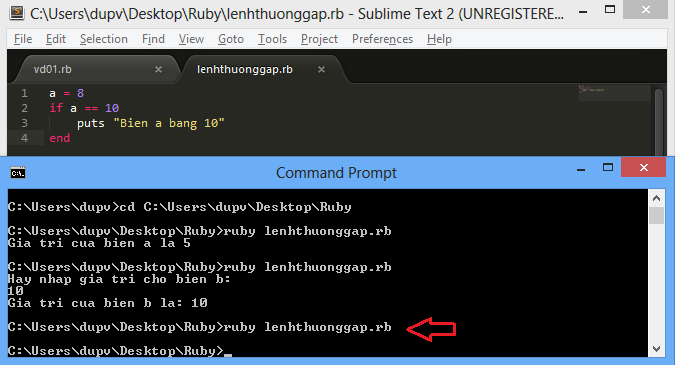
Khi thực thi chương trình sẽ yêu cầu người dùng nhập giá trị cho biến, sau khi nhập giá trị cho biến chương trình sẽ trả về kết quả như hình bên dưới.

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-03.png)

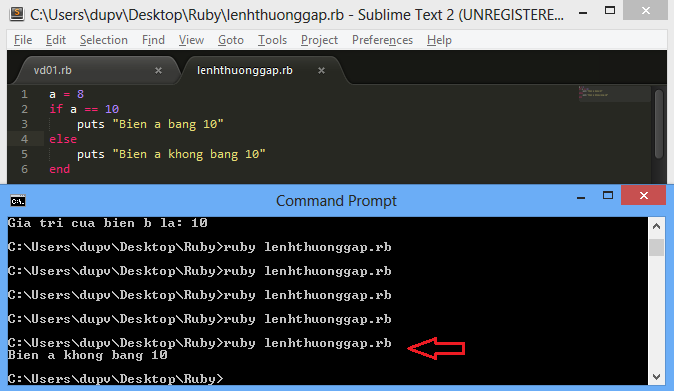
**Câu điều kiện so sánh if else**

Để sử dụng câu điều khiện so sánh, ta sử dụng cú pháp **if**kèmsau đó là điều kiện, và để kết thúc câu điều kiện ta dùng **end.**

**VD1:** Mình sẽ khai báo biến a = 8 và kiểm tra xem biến a có = 10 hay không, nếu bằng 10 thì sẽ in ra kết quả **“Biến a bằng 10″**

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-04.png)

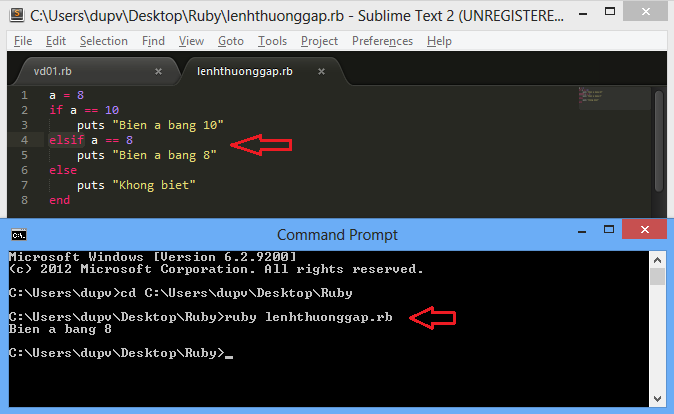
**VD2:** Mình sẽ khai báo biến a = 10 và kiểm tra nếu biến a = 10 thì sẽ xuất ra kết quả **“Biến a bằng 10″** ngược lại nếu biến khác 10 thì sẽ xuất ra kết quả **“Biến a không bằng 10″**

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-05.png)

Ở VD1 vì khi thực thi, câu lệnh điều kiện không trả về kết quả, nên command line không có kết quả trả về, ở VD2 khi thực thi thì có kết quả nên command line có kết quả trả về.

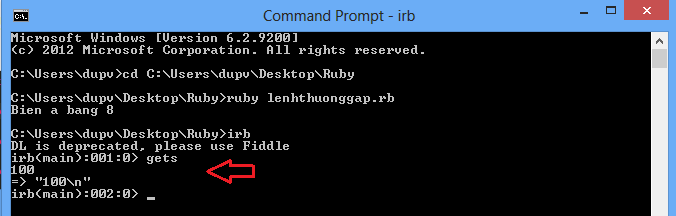
**VD3:** Ở ví dụ này mình sẽ sử dụng câu lệnh **elsif**để kiểm tra giá trị.

Đầu tiên chương trình sẽ khai báo biến a có giá trị là 8, sau đó kiểm tra biến a có bằng 10 hay không và phát hiện biến a không bằng 10. Sau đó chương trình thực thi dòng tiếp theo và kiểm tra biến a có bằng 8 hay không, và phát hiện biến a bằng 8 và trả về kết quả ở command line như hình bên dưới.

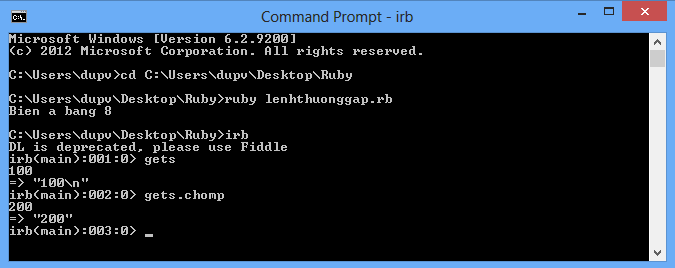
[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-06.png)

**Lưu ý khi sử dụng hàm gets**

Bây giờ mình sẽ vào **môi trường IRB** để ví dụ cho các bạn thấy, ở đây mình sẽ dùng hàm **gets** và nhập giá trị 100 vào. Nhưng khi nhập giá trị 100 thì kết quả lại xuất ra **“100\n”**vì lệnh **gets** đã vô tình lấy giá trị nhập tính luôn phím **Enter**.

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-07.png)

Nên để khắc phục trường hợp này ta dùng lệnh gets.chomp thì sẽ không còn thêm “\n”. Và mình sẽ minh họa cho các bạn thêm một ví dụ về việc sử dụng hàm gets.chomp, khi nhập vào giá trị 200 sẽ không có thêm “\n”

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-08.png)

Từ giờ chú ý là khi sử dụng hàm gets thì chuỗi nhập vào sẽ có thêm “\n” ở đuôi, còn gets.chomp thì sẽ không có.

Như vậy qua bài hướng dẫn này mình đã giới thiệu cho các bạn về **các câu lệnh đơn giản và thường gặp trong Ruby**, ở phần này mình sẽ cho các bạn một bài tập nho nhỏ để giúp các bạn làm quen với các câu lệnh thường gặp trong Ruby.

**Bài tập**

1. Tạo file Ruby nhapten.rb

2. Khi chạy file đó thì kết quả sẽ như thế này:

In ra dòng chữ: “**Ten cua ban la gi**“

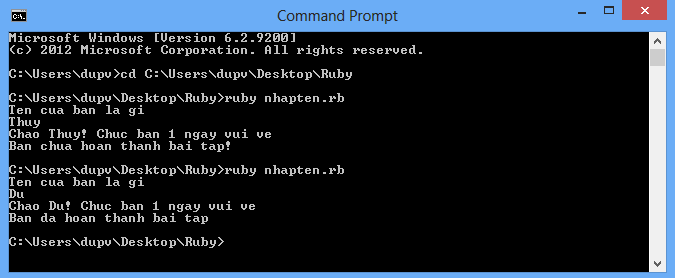
Sau đó đợi dữ liệu nhập từ bàn phím.

Khi nhập xong thì sẽ hiện ra dòng chữ: “Chao **giá-trị-nhập-từ-bàn-phím**! Chuc ban 1 ngay vui ve”

3. Các trường hợp:

Nếu tên nhập vào là tên của bạn thì sẽ hiện thông báo: **Ban da hoan thanh bai tap**

Còn bất kì tên, giá trị khác thì sẽ hiện thông báo: **Ban chua hoan thanh bai tap!**

[](http://idev.vn/wp-content/uploads/2014/06/lenh-thuong-gap-trong-ruby-091.png)

Đáp án

|  |
| --- |
| puts "Ten ban la gi?"  ten = gets.chomp  puts "Xin chao #{ten}! Chuc ban 1 ngay vui ve !"  if ten == "vien"  puts "Ban da hoan thanh bai tap"  else  puts "Ban chua hoan thanh bai tap"  end |

Video4.Số và các phép toán

Trong bài học này thì sẽ học về số, các phép toán, các phép so sánh , cũng như các thành phần khác liên quan đến số trong Ruby.

Trong Ruby thì có rất nhiều kiểu dữ liệu, tuy nhiên khi khai báo biến trong Ruby thì ko cần khai báo kiểu dữ liệu nên chính giá trị mà biến được gán sẽ định nghĩa kiểu dữ liệu cho biến.

Ví dụ

>a = 10

>b = 20.0

Khi muốn kiểm tra a thuộc kiểu dữ liệu nào thì ta có lệnh

>a.class

Fixnum (Ruby gom chung kiểu Integer và Bignum vào chung thành kiểu Fixnum, để trong trường hợp khi biến nhận giá trị lớn hơn kiểu integer , thì nó sẽ tự động chuyển kiểu thành bignum để lưu trữ)

>b.clas

Float

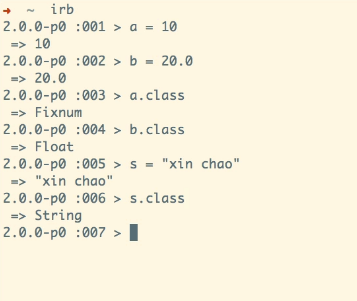
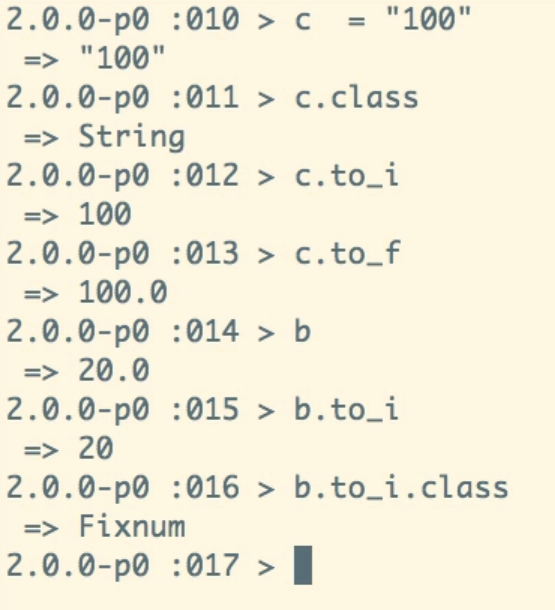
Cú pháp class này có thể tác động nên nhiều kiểu dữ liệu khác nhau

>c = “vien”

“vien”

>c.class

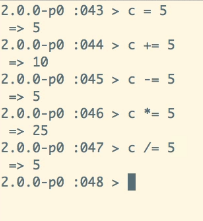
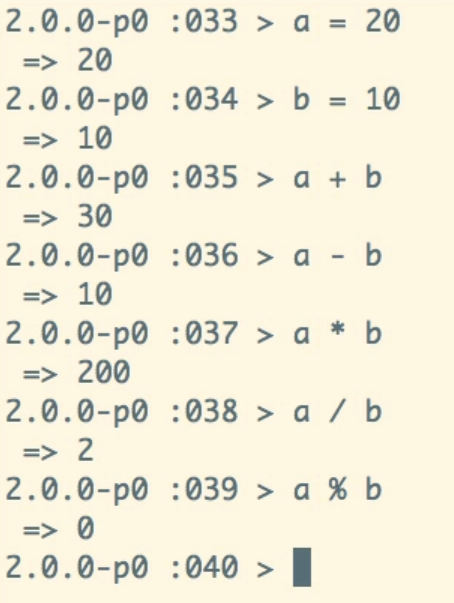
String

Muốn chuyển 1 kiểu dữ liêu sang kiểu số nguyên ta có phương thức tên\_biến.to\_i , nếu muốn chuyển qua kiểu số thực thì ta có phương thức tên\_biến.to\_f

**Toán tử**

Trong Ruby thì có các sử dụng toán tử: + , = , \* , %(lấy phần dư) , /(lấy phần nguyên).Ngoài ra thì Ruby cũng hỗ trợ các toán tử gán như là += , -= , /= , \*=



Trong Ruby thì cũng như các ngôn ngữ lập trình khác thì thứ tự ưu tiên thực hiện sẽ là nhân chia trước, công trừ sau. Nếu thực hiện đồng thời nhân chia thì sẽ thực hiện từ trái qua phải. Phép toán mũ trong Ruby được thực hiện bằng toán tử \*\*

Ví dụ:

> 5\*\*2 (5^2)

25

Ngoài ra còn có cách viết tắt như là

>1e4 (10^4)

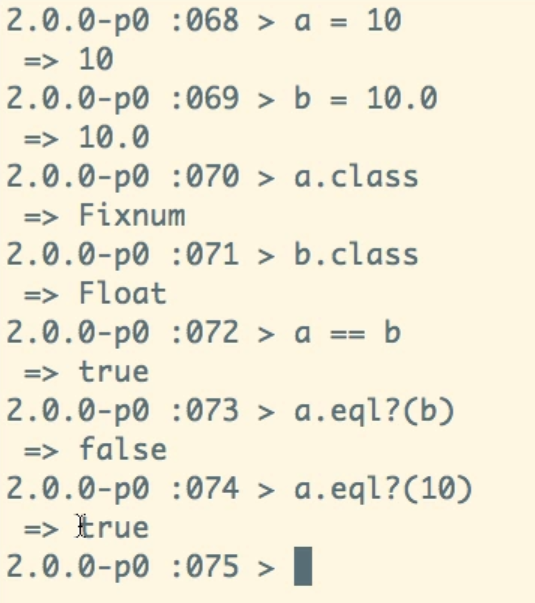
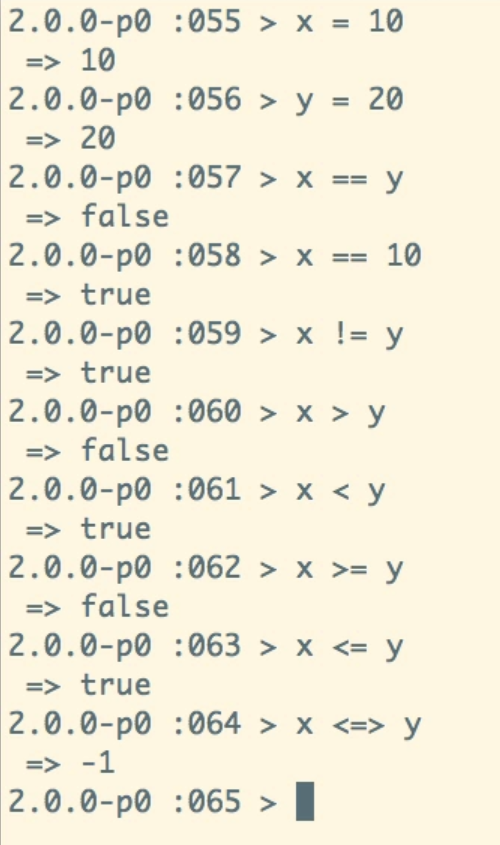
10000.0

>1e6 (10^6)

10000000.0

Các phép so sánh trong Ruby

So sánh bằng: == , so sánh lớn hơn > , so sánh kém < , so sánh khác != , so sánh lớn hơn bằng >= , so sánh nhỏ hơn bang <=



Trong Ruby có 1 phép so sánh đặc biệt với cú pháp a <=> b.Xét 2 giá trị bên trái và phải và trả về 3 trương hợp

Nếu bên trái lớn hơn phải thì trả về 1

Nếu bên trái nhỏ hơn phải thì trả về -1

Nếu bằng nhau thì trả về 0

Trong Ruby ngoài việc so sánh về giá trị thì chúng ta có thể so sánh cả kiểu dữ liệu theo cú pháp

tên\_biên\_1.eql?(tên\_biến\_2)

Ngoài ra có 1 điều thú vị nữa là nếu thực thi lệnh

>0.9 – 0.8

Thông thường thì ta sẽ nghĩ rằng kết quả là 0.1 , nhưng mà kết quả nhận đc lại là: 0.0999999999998

Nguyên nhân là bởi vì trong máy tính khi thực hiện tính toán thì giá trị thực hiện đã được chuyển qua số nhị phân nên kết quả nhận được mới như vây. Để khắc phục tình huống này thì ta sẽ có 2 cách: +Làm tròn kết quả thu được bằng lệnh: >kết\_quả.round(số\_chữ\_số\_thập\_phân\_muốn\_làm\_tròn)

|  |
| --- |
| >a = 0.9 – 0.8  0.09999999998  >a.round(1)  0.1 |

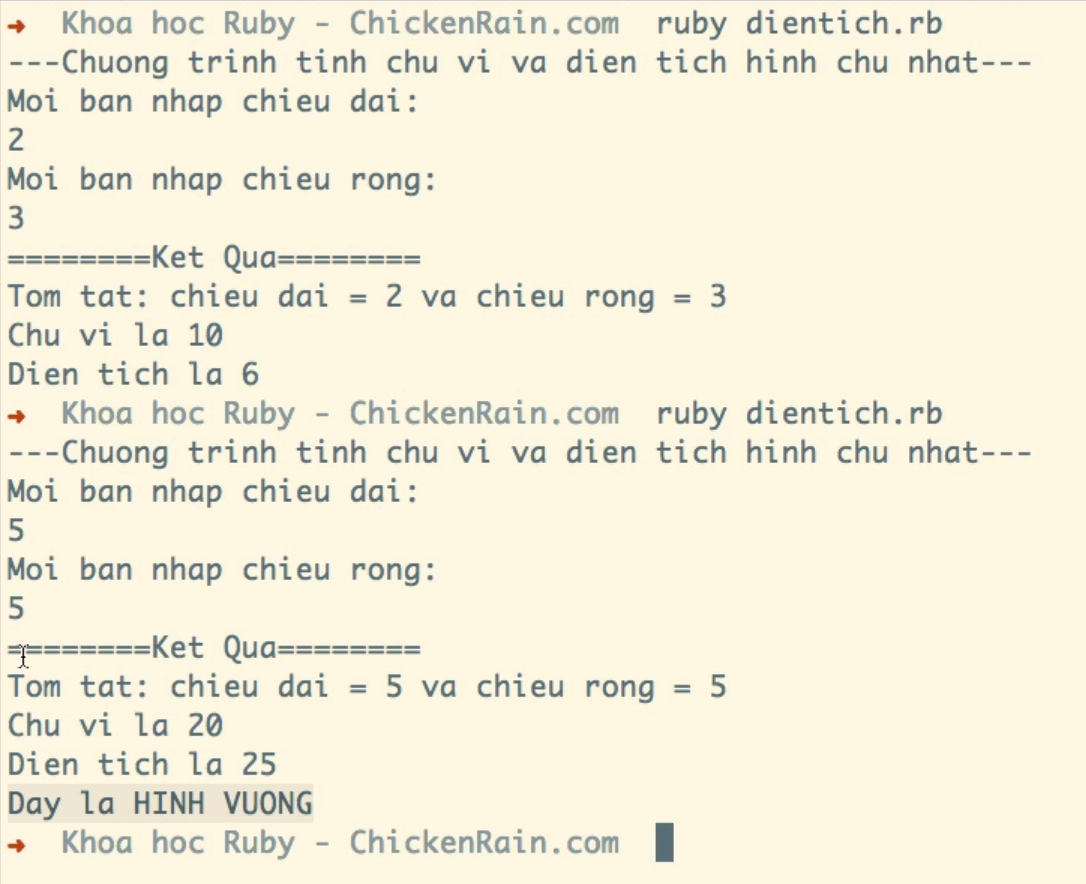
+Sử dụng thư viện trong Ruby

Để gọi thư viện thì ta sử dụng từ khóa require “tên\_thư\_viện”.

|  |
| --- |
| # Khai báo thư viện sử dụng bằng từ khóa require  require "bigdecimal"  x = BigDecimal.new("0.9")  y = BigDecimal.new("0.8")  puts x - y |

Để hiểu rõ hơn về 1 số thư viện trong Ruby thì mình có thể tham khảo tại: <http://ruby-doc.org/core-2.2.2/> (phiên bản ruby mình đang sử dụng là 2.2.2)

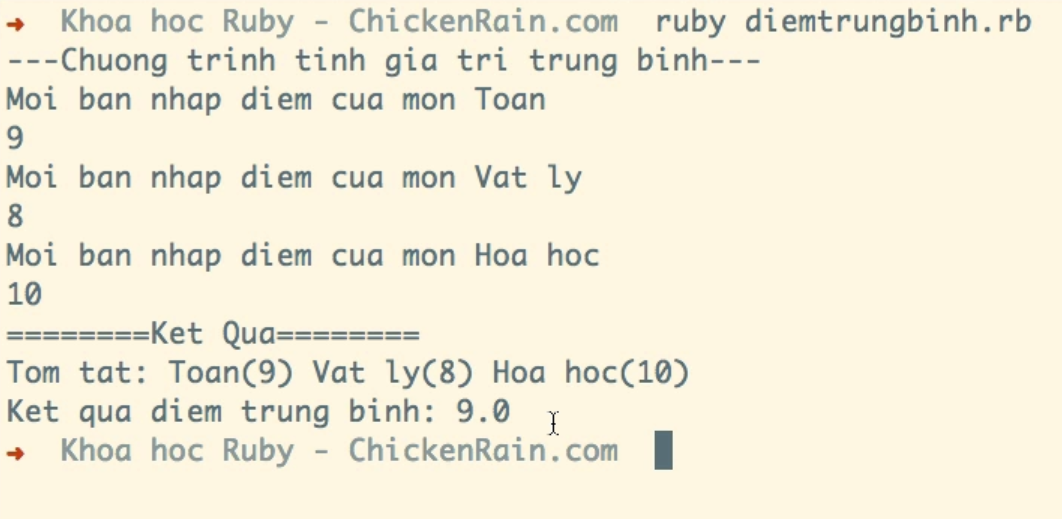
Bài tập 1: Tính diện tích



Đáp án

|  |
| --- |
| #Bài 1: Tính diện tích  puts "========Chuong trinh tinh chu vi va dien tich hinh chu nhat========="  puts "Moi ban nhap vao chieu dai:"  chieudai = gets.chomp # Nhập chiều dài từ bàn phím  puts "Moi ban nhap vao chieu rong:"  chieurong = gets.chomp # Nhập chiều rộng từ bàn phím  puts "========KET QUA========"  puts "Tom tat: chieu dai = #{chieudai} va chieu rong = #{chieurong}"  chieudai = chieudai.to\_i # convert từ string về fixnum  chieurong = chieurong.to\_i #convert từ string về fixnum  chuvi = (chieudai + chieurong) \* 2  puts "Chu vi la #{chuvi}"  dientich = chieudai \* chieurong  puts "Dien tich la #{dientich}"  if chieudai == chieurong  puts "DAY LA HINH VUONG"  end |

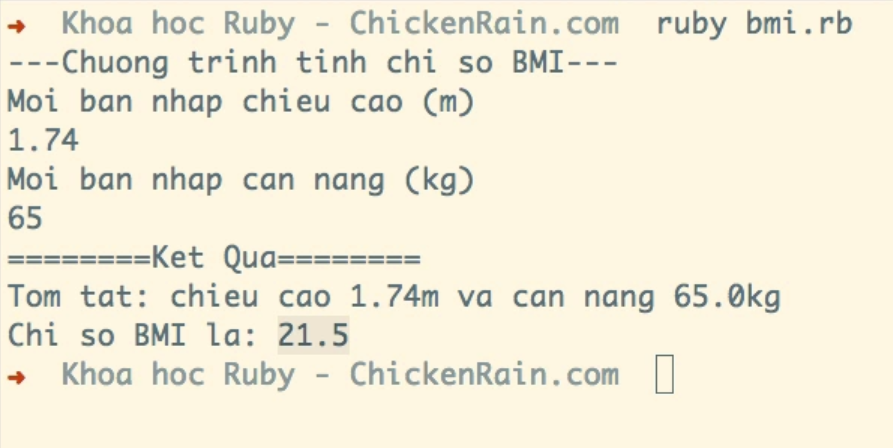
Bài 2: Tính điểm trung bình



Đáp án:

|  |
| --- |
| # Bài 2: Bài tập tính điểm trung bình  puts "===== Chuong trinh tinh diem trung binh ====="  puts "Moi ban nhap vao diem mon Toan"  toan = gets.chomp  puts "Moi ban nhap vao diem mon Vatly"  vatly = gets.chomp  puts "Moi ban nhap vao diem mon Hoa"  hoahoc = gets.chomp  puts "========== KET QUA ==========="  toan = toan.to\_i  vatly = vatly.to\_i  hoahoc = hoahoc.to\_i  puts "Tom tat : Toan (#{toan}) Vat ly(#{vatly}) Hoa hoc(#{hoahoc})"  trungbinh = ((toan.to\_f + vatly.to\_f + hoahoc.to\_f)/3).round(1)  puts "Ket qua diem trung binh: #{trungbinh}" |

Bài 3: Tính chỉ số BMI



Đáp án

|  |
| --- |
| # Bài 3: Tính chỉ số BMI  puts "===== Chuong trinh tinh chi so BMI ===="  puts "Moi ban nhap vao chieu cao(m)"  chieucao = gets.chomp  puts "Moi ban nhap vao can nang(kg)"  cannang = gets.chomp  puts "========== KET QUA ========="  chieucao = chieucao.to\_f  cannang = cannang.to\_f  puts "Tom tat: chieu cao: #{chieucao}m va can nang #{cannang}kg "  bmi = cannang / (chieucao\*\*2)  bmi = bmi.round(1)  puts "Chi so BMI la : #{bmi}"  # Phần này tự bổ sung  if 0 < bmi && bmi < 18.5  puts "Ban dang o duoi chuan"  elsif 18.5 <= bmi && bmi < 25  puts "BMI cua ban la chuan"  elsif 25 <= bmi && bmi< 30  puts "Ban bi thua can roi"  elsif 30 <= bmi && bmi < 40  puts "Ban bi beo - nen giam can"  else  puts "Ban bi beo phi - giam can ngay"  end |

Video5. Chuỗi trong Ruby và các thành phần liên quan

Cũng như các thành phần khác trong Ruby thì chuỗi cũng là 1 đối tượng.Có 2 cách để khai báo chuỗi trong Ruby đó là

>s = String.new(“giá\_trị\_của\_chuỗi”)

Ví dụ: >s = String.new(“Xin chao”)

# Với cách khai báo này là ta đã tạo ra 1 phần tử mới trong lớp String được định nghĩa sẵn của Ruby

Ngoài ra ta có thể khai báo nó theo cách thông thường

>c = “giá\_trị\_của\_chuỗi”

Ví dụ: >c = “Xin chao”

In chuỗi ra màn hình

>puts “chuỗi\_cần\_in” #puts “Xin chao”

Hoặc

>puts biến\_chứa\_chuỗi

Ví dụ:

>b = “Xin chao”

>puts b

Có 1 chút ý với việc sử dụng dấu ngoặc kép và ngoặc đơn trong khi muốn in chuỗi ra màn hình đó là nếu sử dụng dấu ngoặc kép thì #{tên\_biến} sử dụng để in giá trị của biến sẽ có tác dụng, nhưng nếu đặt nó trong dấu ngoặc đơn thì bên trong ngoặc đơn có gì thì nó sẽ in ra như thế.



In ra những ký tự đặc biệt:

Để in ra dấu ngoặc kép bên trong dấu ngoặc kép thì ta sử dụng

\” để in ra dấu ngoặc kép.

\n dùng để xuống dòng.

\t là dùng 1 khoảng tab làm khoảng trống.

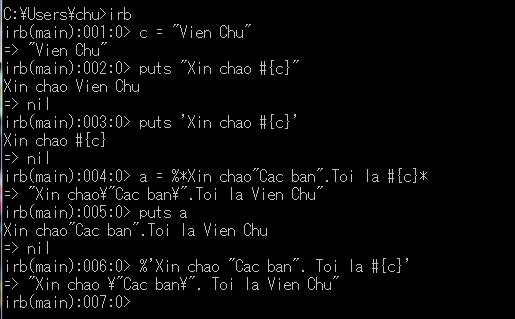
\v là vừa xuống dòng và vừa thụt vào đầu dòng 1 tab

Ngoài ra Ruby hỗ trợ kiểu tự định nghĩa theo cách sau:

%Ký\_tự\_tự\_định\_nghĩa Chuỗi Ký\_tự\_tự\_định\_nghĩa

Ví dụ : %\*Chuỗi\_cần\_in\* , %-Chuỗi\_cần\_in-

Sau dấu % là ký tự nào thì kết thúc bằng ký tự đó(do lập trình viên tự định nghĩa, có thể là \* , - , ‘ ,… đều được). Với kiểu tự định nghĩa này thì nó có thể in được giá trị của biến. Với cách in chuỗi tự định nghĩa này thì ta có thể in ra giá trị của biến trong dấu ngoặc đơn



So sánh chuỗi

Tương tự như so sánh số trong Ruby, ta cũng sử dụng các toán tử so sánh như == , != để so sánh. Ngoài ra thì toán sử <=> sẽ so sánh số lượng ký tự trong 2 chuỗi và đưa ra 3 kết quả.

Nếu về trái nhiều hơn vế phải sẽ trả về 1, ít hơn vế phải trả về -1 và bằng nhau sẽ trả về 0

Kiểm tra chuỗi rỗng hay không

>tên\_biến.empty?

true(nếu rỗng)/false(nếu không rỗng)

Nối chuỗi

>a = “Chao ban”

>b = “Vien Chu”

>a + b

# a << b (Cách viết này tương tự với a = a + b)

“Chao banVien Chu”

>a+ “ ” +b

# a = “Chao ban”

# a << “ ” << b

“Chao ban Vien Chu”

Phép toán nhân chuỗi

>a = “Ruby”

>a\*3

“RubyRubyRuby”

Nhập dữ liệu từ bàn phím.

Theo các bài học trước thì ta đã biết sử dụng hàm gets để nhập dữ liệu từ bàn phím. Tuy nhiên nếu sử dụng hàm gets để nhập thì kết quả trả về sẽ có dạng “dữ\_liệu\_nhập\n”. Vì vậy để không bị tình trạng này ta có thể nhập bằng hàm gets.chomp.

Tuy nhiên nếu chuỗi có sẵn ký tự đặc biệt \n thì làm sao để loại bỏ nó. Ruby cũng cung cấp cho ta phương pháp đó là sử dụng lệnh

>tên\_biến.chomp

Ví dụ:

>a = “xin chao\n”

>a.chomp

“xin chao”

In hoa,in thường chuỗi,định dạng kiểu chữ

Để định dạng để in ra cho đẹp hơn, hoặc định dạng chuỗi từ bên ngoài để thao tác dễ dàng hơn

In hoa: >tên\_biến.upcase

In thường: >tên\_biến.downcase

Chữ đầu tiên in hoa: >tên\_biến.capitalize

Đổi chữ thường thành hoa,hoa thành thường trong 1 chuỗi >chuỗi.swapcase

Hàm đảo ký tự của chuỗi >chuỗi.reverse (Giá trị vẫn ko thay đổi mà chỉ thay đổi dạng hiển thị khi in ra)

Các hàm trên chỉ làm thay đổi định dạng hiển thị chứ ko làm thay đổi giá trị của biến. Vậy để thay đổi giá trị của biến thì mình cần làm thế nào?

>c = c.downcase # Ở đây ta gán cho biến c giá trị mới

>c.downcase! # Dấu chấm than có tác dụng gán luôn giá trị mới vào cho biến c. Vì vậy khi dùng dấu ! thì nên cẩn thận vì giá trị cũ sẽ hoàn toàn bị thay thế và mất đi.

Đếm chiều dài chuỗi

>chuỗi.length

>chuỗi.size

Tìm kiếm chuỗi trong chuỗi

>chuỗi\_cha.include?(“chuỗi\_con”)

true(có tồn tại)/false(không tồn tại). Do Ruby có phân biệt chũ hoa, chữ thường nên chỉ cần 1 ký tự khác thì nó cũng ko tìm thấy

Tìm và thay thế bằng chuỗi khác

>c = “Xin chao cac ban”

>c.sub(“gia\_tri\_can\_tim” , “gia\_tri\_thay\_the”)

Giá trị cần tìm cũng phải giữ đúng các ký tự thì mới có thể tìm thấy và thay thế

Tách chuỗi thành các chuỗi con theo ký tự đặc biệt(“ ” , | , …)

>c = “Xin chao cac ban”

>c.split(“ ”) # Tách chuỗi theo khoảng trắng

[“Xin”,”chao”,”cac”,”ban”]

Để nhập, in ra 1 chuỗi dài thì Ruby có hỗ trợ bạn bằng cách sử dụng ký tự tự định nghĩa

>chuoidai = << AB #AB ở đây là ký tự tự định nghĩa

>”mình sẽ nhập chuỗi tại đây

>”dài bao nhiêu cũng được

>AB # Bắt đầu bằng gì thì kết thúc bằng cái đó

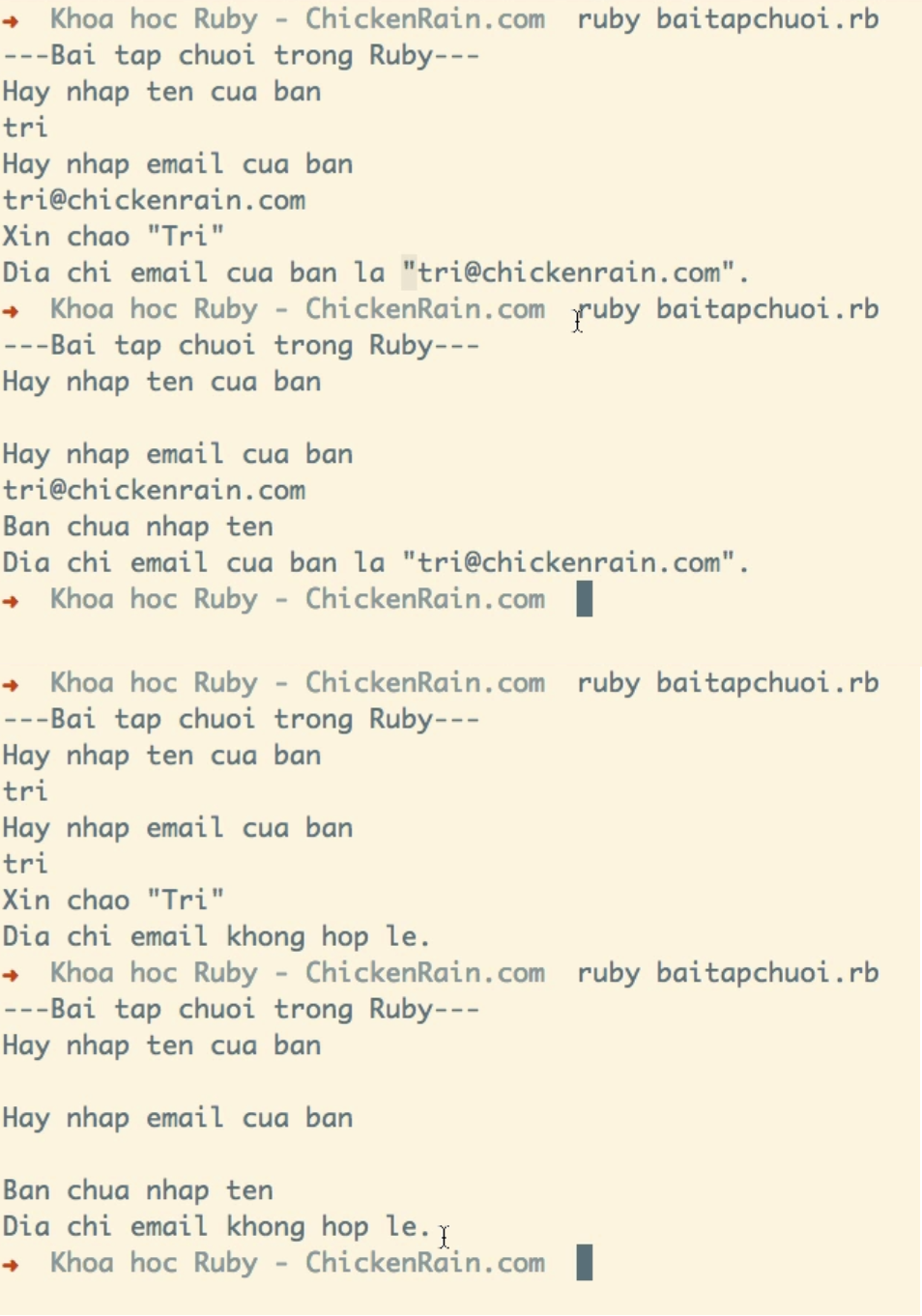
Như vậy ta sẽ kết thúc bài học về chuỗi tại đây.

**Ngày 18/08/2015**

Bài 1:

Nhập tên và địa chỉ email từ bàn phím. Xuất ra tên và địa chỉ email. Nhập vào thế nào cũng được nhưng nếu xuất ra thì phải xuất chữ đầu tiên trong tên là chữ Hoa và tên,email được đặt trong “”.

Trong trường hợp không nhập tên thì sẽ in ra “Ban chưa nhập tên”. Trong trường hợp email ko có ký tự @ hoặc rỗng thì hiển thị “Email chưa hợp lệ”.



Đáp án

|  |
| --- |
| #Bài 1: Nhập và check tên, địa chỉ email  puts "Hay nhap ten cua ban :"  ten = gets.chomp  puts "Hay nhap email cua ban: "  email = gets.chomp  ten = ten.downcase  mangten = ten.split("")  ten = ten.sub(mangten[0] , mangten[0].upcase)  if (ten.empty? == true)  puts "\nBan chua nhap ten"  else  puts "\nXin chao \"#{ten}\" "  end  if (email.empty? == true || email.include?("@") == false)  puts "Dia chi Email chua hop le"  else  puts "Dia chi Email cua ban la \"#{email}\"."  end |

Video6: Mảng trong Ruby

Mảng sẽ giúp ta lưu trữ dữ liệu dễ dàng hơn.

Một số cú pháp khai báo mảng trong Ruby

C1:

>mang = Array.new

[] # Đây là 1 mảng

>mang = Array.new(7) # Khai báo mảng có 7 phần tử

[nil, nil, nil, nil, nil, nil, nil]

C2:

>a = []

[]

>b = [“thu hai” , “thu ba”] # Khai báo trực tiếp giá trị của mảng

[“thu hai” , “thu ba”]

Thêm phần tử vào mảng

>ar = []

>ar.push(5) #Thêm giá trị 5

>ar.push(10)

>ar.push(“ruby”)

[5, 10, “ruby”]

Ngoài rat hay vì dùng lệnh push thì ta có thể dùng toán tử << để thêm giá trị vào cuối mảng

>a << “hello”

[5, 10, “ruby”, “hello”]

Hay ta có thể thêm mảng vào mảng

>a << [“mot”, “hai”]

[5, 10, “ruby”, “hello”, [“mot”, “hai”]]

Có thể sử dụng phép toán gán để đưa giá trị vào mảng

>a += [100] # Tuy nhiên ta phải biết giá trị của phần tử sẽ thêm vào mảng

[5, 10, “ruby”, “hello”, [“mot”, “hai”], 100]

Xoá phần tử trong mảng

>ma = [5,10,20]

>ma.pop

20 # Đây là giá trị bị xóa ra khỏi mảng

Pop sẽ xóa đi phần tử cuối cùng trong mảng

Xác định số lượng phần tử trong mảng

>a = [5,10,20, “ruby”]

>a.size hoặc >a.length

Lấy vị trí của phần tử trong mảng. Mảng trong ruby được đánh số index từ 0 và tính từ trái qua phải

>a.index(10)

1 # Đây là vị trí của phần tử 10 trong mảng

>a.index(“ruby”)

3 # Vị trí của phần tử “ruby” trong mảng

Gọi phần tử trong mảng thì ta sẽ gọi bằng chỉ số của phần tử đó

>a[1]

10

Thay vì 0 là vị trí đầu tiên thì -1 sẽ là vị trí cuối cùng trong mảng và -2,-2 là các phần tử liền ngay trước nó

Để xác đinh nhanh phần tử đầu tiên và phần tử cuối cùng của mảng trong Ruby thì Ruby có hỗ trợ 2 hàm là

>a.first #Lấy giá trị đầu tiên

>a.last #Lấy giá trị phần tử cuối cùng

Nếu muốn thêm phần tử vào đầu mảng thì ta dùng lệnh

>a.unshift(22) # Thêm phần tử 22 vào đầu mảng

[22,5,10,20, “ruby”]

Thay đổi giá trị của 1 phần tử trong mảng

>a[1] = 10 # Thay giá trị phần tử thứ 1 trong mảng thành 10

[22, 10, 10, 20, “ruby”]

Nếu muốn xóa phần tử trong mảng thì ta gán phần tử đó bằng nil

>a[1] = nil

[22, nil ,10, 20 , “ruby”] # Phần tử thứ 1 đã bị xóa

Chúng ta có thể gán 1 mảng vào trong 1 mảng khác

>a = [1,2,3,[5,10],4]

Ghép mảng con vào trong mảng cha

>a.flatten

[1,2,3,5,10,4]

Xóa mảng

>a.clear

[] # Mảng a giờ sẽ là mảng rỗng, ko có phần tử nào cả

Xác định phần tử giao nhau của 2 mảng

>a = [1,2,3,4]

>b = [4,5,6,7,8]

>a&b

[4] # 4 là phần tử chung của 2 mảng

>c = [20]

>a&c

[] # Không giao nhau

Sắp xếp mảng

>mang = [2,50,40,1,23]

>mang.sort # Sắp xếp từ nhỏ đến lớn

[1,2,23,40,50]

>mang.sort.reverse

[50,40,23,2,1] # Sắp xếp từ nhỏ đến lớn rồi đảo ngược mảng lại

Tuy nhiên thì thứ tự các phần tử trong mảng vẫn giữ nguyên ko thay đổi, nên nếu ta muốn sắp xếp và giữ nguyên thứ tự sắp xếp đó thì ta có thể sử dụng dấu !

>mang.sort!

[1,2,23,40,50]

>mang

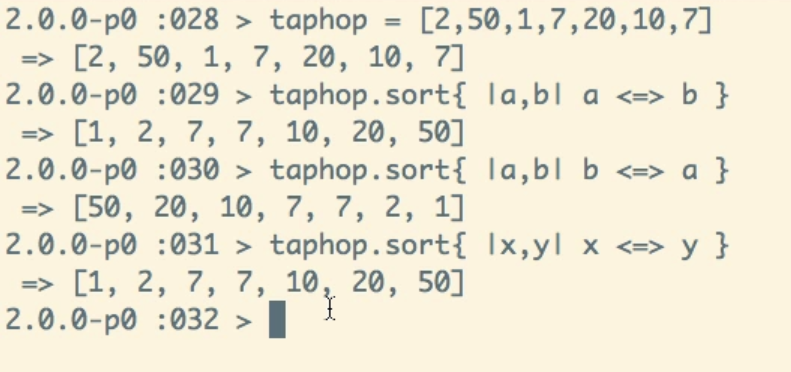
[1,2,23,40,50]

Ngoài ta thì Ruby có hỗ trợ thêm các cách khác để có thể sắp xếp 1 mảng theo ý của mình (cú pháp hơi khó nhớ hơn so với việc sử dụng hàm sort và reverse)

>mang.sort{ |a,b| a <=> b }

# Giải thích ý nghĩa của lệnh trên: Trong lệnh trên mình thực hiện sort với điều kiện sort được khai báo bên trong {}. |a,b| ở đây mình dùng a,b còn có thể x,y,z,m,n gì cũng được, tùy theo sở thích. Và thứ tự là a- nhỏ, b- lớn. Tiếp theo a <=> b nghĩa là sắp xếp từ a đến b tức là từ nhỏ, đến lớn. Còn nếu mình viết b<=>a thì có nghĩa là mình sắp xếp từ lớn đến nhỏ

Ví dụ:



Nếu trong mảng có những phần tử giống nhau thì mình có thể gộp những phần tử ấy vào làm 1 rồi sắp xếp

>mang = [11,21,3,7,7,7,9,12]

>mang.uniq

[11,21,3,7,9,12]

>mang.uniq.sort

[3,7,8,11,12,21]

Nhưng vẫn luôn phải nhớ là nó sẽ chẳng ảnh hưởng gì đến mảng thật sự, nó chỉ ảnh hưởng khi ta thêm ! vào sau lệnh

>mang.uniq! #Loại bỏ phần tử giống và lưu lại vào mảng

[11,21,3,7,9,12]

>mang

[11,21,3,7,9,12]

Duyệt mảng

Trong các trường hợp số lượng phần tử trong mảng quá nhiều thì việc sử dụng index để truy vấn tới phần tử trong mảng là không tốt.Vậy ta phải làm thế nào? Ta dùng vòng lặp để duyệt mảng

>danhsach = [5,10,200,20 ,"ruby",21]

# Duyệt mảng

>danhsach.each do

|ds|

# Tương ứng với mỗi giá trị ds mình sẽ thực hiện 1 hành đông

# Ở đây ta sẽ in ra

puts "Gia tri tai phan tu nay la #{ds}"

end

#Thay do end thành dấu ngoặc {}

>danhsach.each { # Dùng { thay cho do

|ds|

puts "Gia tri tai phan tu la #{ds}"

} #Dùng } thay cho end

Ngoài ra thì ta có thể thực hiện so sánh phần tử trong mảng với 1 giá trị có sẵn thì ta dùng lệnh

>danhsach.all? {|ds| ds>4}

false

# Kiểm tra tất cả các phần tử trong mảng, điều kiện so sánh sẽ nằm trong ngoặc nhọn. Nếu tất cả các phần tử trong mảng thỏa mãn đk mà ta đưa ra thì sẽ trả về true, nếu ko sẽ trả về false

# Vì sao trả về flase ? Vì ruby là chuỗi nên ko so sánh được

Nếu ko kiểm tra tất cả mà chỉ muốn xác định có phần tử nào lớn hay nhỏ hơn giá trị cho sẵn thì ta dùng lệnh

>danhsach = [200,20,21,5,12]

>danhsach.any? {|ds| ds>20}

True

Với cú pháp any thì chỉ cần 1 phần tử thỏa mãn đk đưa ra là trả về kết quả true, ko thì trả về false.

Lấy ra những phần tử trong mảng thỏa mãn điều kiện(mảng cùng kiểu, nếu mảng vừa chứa chuỗi, vừa chứa số thì lệnh này sẽ lỗi)

>danhsach.select {|ds| ds > 15}

[200,20,21] # Trả về mảng các giá trị thỏa mãn đk

Ngược lại với select thì lênh reject sẽ loại ra những giá trị thỏa mãn đk mà lấy những giá thị ngược lại

>danhsach.reject {|item| item>20}

[5,12] # Trả về mảng các giá trị nhỏ hơn 20

Map sẽ tác động đến phần tử trong mảng khi thực thi

>danhsach.map {|ds| ds+10}

# Khi đi qua các phần tử trong mảng thì sẽ cộng vào từng phần tử 10 đơn vị

[210,30,31,15,22]

Và vẫn như cũ,muốn lưu ta phải dùng !

>danhsach.map! {|ds| ds-1}

[199,19,20,4,11]

>danhsach

[199,19,20,4,11]

Như vậy là ta đã kết thúc bài học về mảng tại đây

Tham khảo: <http://www.rubylife.jp/ini/array/>

Trong bài viết về mảng trong Ruby này, tác giả chia ra làm

Video7:Hash trong Ruby (Phần 1)

Hash cũng gần như là mảng, tuy nhiên thay vì sử dụng các index như trong mảng thì nó sử dụng key

Có nhiiều cách khai báo hash trong Ruby

>h = Hash.new # Khai báo 1 đối tương hash chưa có dữ liệu

#=>{}

>h = {} # Tương tự với khai báo trên nhưng ko gọi class Hash mà khởi tạo trực tiếp bằng {} , khác với mảng là dữ liệu nằm trong [] thì với hash, dữ liệu nằm trong {}

#=>{}

Đây là 2 cách để khai báo hash trong Ruby

Tuy nhiên ta thường phải khởi tạo thêm giá trị cho hash như sau

>h = {“name” => “Vien Chu”}

{“name” => “Vien Chu”}

>h[“email”] = “vienbk91@boi.jp”

vienbk91@boi.jp

>h

{“name” => “Vien Chu” , “email” => “vienbk91@boi.jp”}

Như vậy ta đã hình dung ra được phần tử trong hash được khai báo với cặp “key” => “value”

Các thao tác liên quan đến key và value trong hash

+Để liệt kê các khóa trong hash

>hash.keys

[] # Trả về 1 mảng chứa tên các khóa trong hash

>hash.values

[] #Trả về 1 mảng chứa các giá trị trong hash

+Nếu muốn lấy giá trị khi đã biết key thì ta truy cập như trong mảng

>h[“name”]

“Vien Chu”

>h[“location”]

nil # Do khóa này không tồn tại trong hash nên kết quả là nil

+Lấy key khi đã biết value

>h.key(“Vien Chu”)

“name” # Trả về khóa đã lưu cái giá trị này

Sử dụng thư viện để hiển thị hash hiển thị đẹp hơn. Tuy nhiên thư viện này ko phải thư việc mặc định có sẵn của Ruby nên ta phải cài đặt và sau đó là cài đặt nó vào trong môi trường irb.

Cách cài đặt như sau: Vào command prompt và gõ lệnh

>gem install awesome\_print

Đợi nó cài đặt, và bây giờ ta sẽ vào môi trường irb

>require “rubygems” #Tải thư viện rubygems vào

>gem “awesome\_print”

>require “awesome\_print” # Đã cài đặt thư viện awesome\_print vào

Để hiển thị hash thì ta dùng lệnh

>ap tên\_hash



Bài tập: Viết chương trình hiển thị thông tin cá nhân như trong ảnh



Đáp án

|  |
| --- |
| h = {"name" => "Vien Chu" , "email" => "vienbk91@boi.jp" , "wwebsite" => "ChickenRain.com"}  require "awesome\_print"  ap h |



Video8: Hash trong Ruby (Phần 2)

+ Kiểm tra 1 key có tồn tại trong hash

>tên\_biến\_hash.has\_key?(“tên\_key”)

true(nếu tồn tại) / false(nếu không tồn tại)

+ Kiểm tra 1 value có tồn tại trong hash

>tên\_biến\_hash.has\_value?(“giá\_trị\_value”)

true (nếu tồn tại) / false (nếu không tồn tại)

+Vòng lặp với hash

Nhắc lại vòng lặp với mảng

rên\_mảng.each do

|item| #Tên do mình tự đặt, mỗi item đại diện cho 1 ptu trong mảng

/\*Viết mã xử lý item\*/

end

Ngoài ra ta có thể thay thế do..end bằng {}

tên\_mảng.each {

|item|

/\*Viết mã xử lý item\*/

}

Tương tự như thế, hash cũng sử dụng vòng lặp như vậy

tên\_hash.each do

|key.value| # Tên do mình tự đặt, key sẽ đại diện cho từ khóa, value đại diện cho giá trị của từng phần tử trong hash

/\*Thực hiện mã xử lý với key và value\*/

end

Và ta cũng có thể dùng {} để thay thế cho do…end

Ở trên là ta thao tác với cả khóa và giá trị của khóa, nhưng nếu ta chỉ muốn thao tác vơi khóa hoặc giá trị thì ta có thể dùng như sau

tên\_hash.each\_key do

|key| # Chỉ là key thôi

/\*Todo\*/

end

tên\_hash.each\_value do

|value| # Chỉ là value thôi

/\*Todo\*/

end

+Truy vấn phần tử trong hash có thỏa mãn đk cho trước

tên\_hash.select {

|key,value|

/\*Điều kiện\*/

# key.size > 5 , value.length > 10,…

# key == “name”…

}

{“key” => “value”,…} # Kết quả trả về là hash chứa key và value nếu đk là đúng

+Nếu không dùng hàm select thì ta có thể sử dụng hàm

>find{ |key,value| #Điều kiện tìm kiếm}

tuy nhiên nó chỉ trả về 1 kết qua đầu tiên nó tìm thấy và sẽ trả về ở dạng mảng .Nếu muốn nó trả về tất cả các phần tử thỏa mãn đk thì dùng

>find\_all{ |key,value| #Điều kiện tìm kiếm}

+ Sử dụng

>tên\_hash.all?(|key,value| #Tiếp theo là đk)

Thì nó sẽ trả về true nếu các phần tử trong hash thỏa mãn đk

Trả về false nếu 1 phần tử bất kỳ ko thỏa mãn đk

+ Sử dụng

>tên\_hash.any?(|key,value| #Tiếp theo là đk)

Thì nó sẽ trả về true nếu một phần tử bất kỳ trong hash thỏa mãn đk

Trả về false nếu 0 phần tử nào thỏa mãn đk

+Để cập nhật thay đổi chung phần tử trong hash thì cũng giống như array ta sử dụng map

>tên\_hash.map {|key,value| #Thao tác thay đổi phần tử của hash}



Bổ sung về Hash:

Tham khảo: <http://ruby-doc.org/core-1.9.3/Hash.html>

Đây là 1 số ví dụ về hash {key => value}

Hash["a", 100, "b", 200] #=> {"a"=>100, "b"=>200}

Hash[ [ ["a", 100], ["b", 200] ] ] #=> {"a"=>100, "b"=>200}

Hash["a" => 100, "b" => 200] #=> {"a"=>100, "b"=>200}

Có 3 cách để khai báo 1 hash.

h = Hash.new

h = Hash.new(obj) # Truyền vào 1 đối tương

h = Hash.new(|key,value| block)

Ví dụ

h = Hash.new("Go Fish")

h["a"] = 100

h["b"] = 200

h["a"] #=> 100

h["c"] #=> "Go Fish"

Có lẽ nhiều người thắc mắc tại sao lại vậy đúng ko ? Mình có đọc trên mạng và được giải thích như thế này. Vì khi mình khai báo

h = Hash.new(“Go Finish”) thì “Go Finish” trở thành giá trị mặc định gán cho tất cả các phần tử thuộc h. Nên nếu ko được quy định thì nó sẽ tự động nhận default values làm giá trị.

h = { "a" => 100, "b" => 200 }

h["a"] #=> 100

h["c"] #=> nil

Tuy nhiên với cách khai báo như trên thì khóa “c” chưa được khởi tạo giá trị nên nó sẽ là nil(giá trị rỗng)

Ta có 1 cách sử dụng hash rất hay(vì ta có thể dùng hash như 1 hàm)

Tham khảo ví dụ:

fibonacci = Hash.new{ |h,k| h[k] = k < 2 ? k : h[k-1] + h[k-2] }

( 1..6 ).each{ |i| puts fibonacci[i] }

#=> 1 1 2 3 5 8

**Ngày 19/08/2015**

Video9: Câu điều kiện

Có 2 điều kiện đó là

AND(&&): Đúng khi tất cả cùng đúng, sai nếu 1 trong các đk là sai

OR(||): Đúng khi 1 trong các đk là đúng, sai khi tất cả cùng sai

+ Điều kiện if else

if condisiton

/\*Code\*/

elsif

/\*Code\*/

else

/\*Code\*/

end

+ Điều kiện case when

Trong Ruby có hỗ trợ thêm 1 loại câu điều kiện đó là case…when

Nếu ai học C,Java thì có thể hiểu nó chính là switch…case, tuy nhiên có 1 chút thay đổi

+Điều kiện unless

Đây là phủ định của điều kiện if. Cú pháp tương tự if (thay if thành unless), tuy nhiên kết quả nó sẽ trả về ngược với if.

+Điều kiện phủ định sử dụng dấu !(giống C,Java)

>if !(x ==100)

puts “Dung”

end

Có nghĩa là x != 100 thì xuất ra màn hình “Dung”

+ Trong Ruby thì việc đặt câu lệnh if/unless được tùy biến để có thể đặt mọi vị trí

Ví dụ:

>x = 100

>puts “x = 100” if x== 100

x=100

Điều kiện có thể đặt ở sau mà vẫn thực thi :D Cảm giác rất giống ngôn ngữ tự nhiên đúng không

Video10: Vòng lặp trong Ruby

#Vòng lặp while

# Check điều kiện rồi mới thực thi, nếu đk sai thì thoát ra khỏi vòng lặp

puts "======= TEST WHILE======="

i=0

while i <= 10 # Nếu i <=1o thì thực hiện

puts "Gia tri cua i luc nay la #{i}"

i += 1

end

puts "Gia tri cuoi cung cua i la: #{i}"

#Vòng lặp until

# Vòng lặp phủ định với while, nếu đk sau nó sai thì nó mới thực hiện hành động

puts "======= TEST UNTIL======="

k = 0

until k > 10 # Nếu k<=10 thì mới thưc hiện

puts "Gia tri cua k luc nay la #{k}"

k += 1

end

puts "Gia tri cuoi cung cua k la #{k}"

#Vòng lặp for

#Nếu dùng for thì ta có thể xác định được số phần tử cần duyệt trong vòng lặp

puts "======= TEST FOR======="

array = [0,100,2,3,4,5,6,7,8]

for i in array #array là tên mảng vừa khai báo

puts "Gia tri cua i la #{i}"

end

# Sử dụng vong lặp for với số lượng phần tử được xác định

# in 0..5 : Duyệt từ phần tử 0 tới phần tử thứ 5

# do :Khai báo hành động thực thi khi lặp

for i in 0..5 do

puts "Gia tri cua i la #{i}"

end

puts "======= TEST Vong lap long nhau ======="

#Sử dụng vòng lặp bên trong vòng lặp

for i in 0..5 do

for j in 5..10 do

puts "Gia tri cua (i , j) la: (#{i} , #{j} )"

end

end

# Cú pháp điều khiển next

# Khi next được gọi thì nó sẽ bỏ qua tất cả các cú pháp ở bên dưới nó và thực hiện tiếp vòng lặp

puts "======= TEST NEXT ======="

for i in 1..10 do

next if i == 2 # Bỏ qua i = 2 và thực hiện tiếp i=3

puts "Gia tri cua i la #{i}"

end

# Sử dụng next trong While sẽ phát sinh lỗi

# Vì sao lại phát sinh lỗi? Vì lệnh next sẽ bỏ qua tất cả các câu lệnh ở dưới nó

# Vì thế m tăng nhưng ko được thực hiện nhưng m luôn nhỏ hơn 10 nên vòng lặp sẽ ko dừng

puts "======= TEST NEXT IN WHILE ======="

m = 0

while m < 5

#next if m == 2

puts "Gia tri cua m la #{m}"

m += 1

end

# Sử dụng break

# Khi nó được chạy thì nó sẽ dừng vòng lặp lại và thoát khỏi vòng lặp

puts "======= TEST BREAK ======="

for i in 1..6 do

break if i == 4

puts "Gia tri cua i la #{i}"

end

# Một số lệnh lặp được Ruby hỗ trợ

# Lặp khi biết số lần lặp cố định(test thử trong môi trường IRB)

puts "======= TEST TIMES ======="

5.times { # 5 là số lần muốn lắp

puts "Ruby"

}

#Lặp từ a đến b với a,b xác định và a < b

puts "======= TEST UPTO ======="

1.upto(4) {

puts "Ruby"

}

#Lặp từ a đến b với a,b xác định và a > b

puts "======= TEST DOWNTO ======="

4.downto(2){

puts "Ruby"

}

#Lặp từ a đến b

puts "======= TEST EACH ======="

(1..5).each{

puts "Ruby"

}

# Tiếp theo đó là ta có thể sử dụng giá trị của phần tử trong vòng lặp bằng cách truyền vào 1 biến đặt trong || , thì nó sẽ đại diện cho các phần tử trong vòng lặp. Cách này có thể sử dụng cho cả times, upto,downto,each

(2..8).each{

|i|

puts "Ruby #{i}"

}

# Vòng lặp vô tận loop , ta chỉ có thể dừng nó lại nếu có điều kiện ngừng vòng lặp và break nó

puts "======= TEST LOOP ======="

loop do

puts "Nhap vao mat khau :"

matkhau = gets.chomp

if matkhau == "vienbk91"

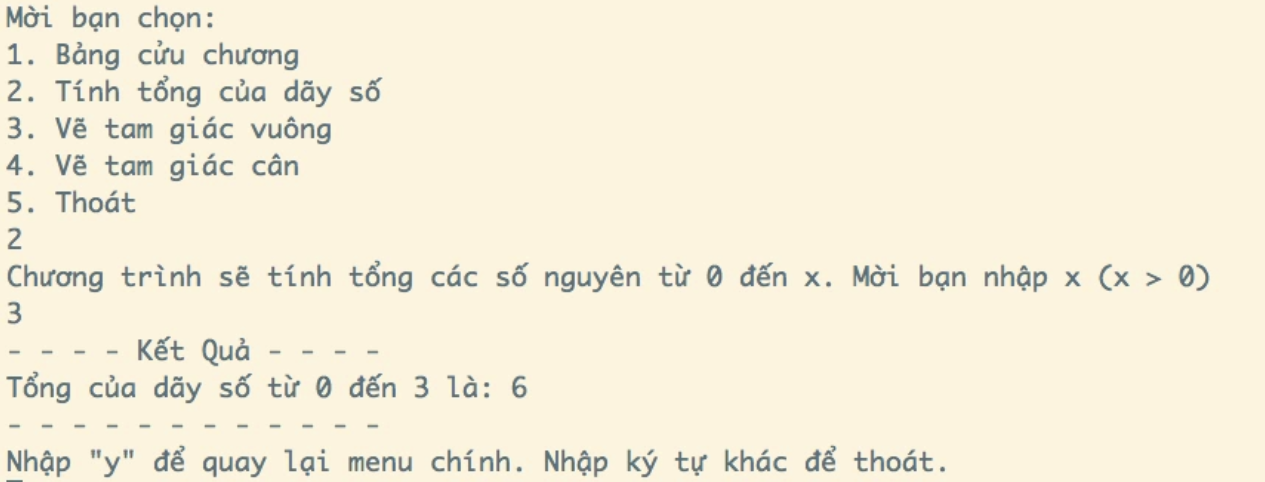
break

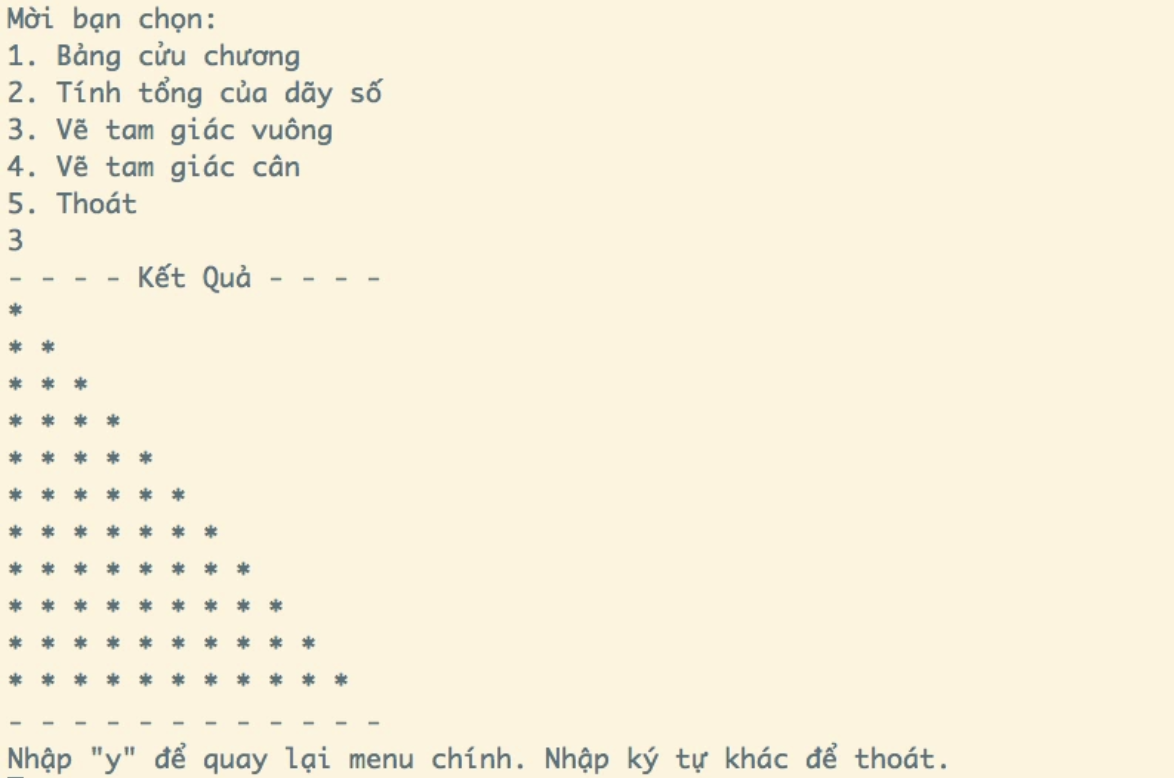
end

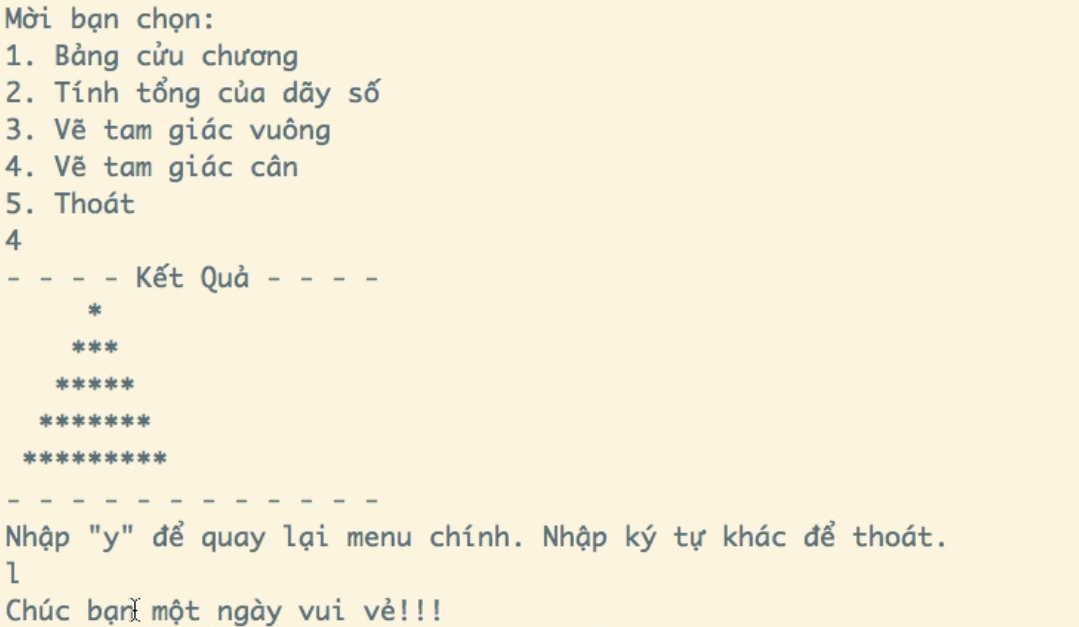
end

Video11: Baì tập tổng hợp











Đáp án:

|  |
| --- |
| # Bài 1: Nhập số và in ra bảng cứu chương  # Bài 2: Tính tổng của dãy số  # Bài 3: Vẽ tam giác vuông  # Bài 4: Vẽ tam giác cân  # Bài 5: Thoát và xuất lời chào  loop do  puts "Moi ban chon:"  puts "1. Bang cuu chuong"  puts "2. Tinh tong cua day so"  puts "3. Ve tam giac vuong"  puts "4. Ve tam giac can"  puts "5. Thoat"  # Nhập vào lựa chọn tương ứng menu  index = gets.chomp  if index.to\_i == 1  # Tạo bảng cửu chương của x với x nhập từ bàn phím  puts "Chuong trinh in ra bang cuu chuong cua x. Moi ban nhap x (1 < x < 9)"  x = gets.chomp  puts "- - - - Ket qua - - - -"  (1..9).each{  |i|  tich = x.to\_i \* i  puts "#{x} x #{i} = #{tich}"  }  puts "- - - - - - - - - - - -"  puts "Nhap \"y\" de quay lai menu chinh . Nhap ky tu khac de thoat."  thoat = gets.chomp  if thoat == "y"  next  else  puts "Chuc ban 1 ngay vui ve !!!"  break  end  elsif index.to\_i == 2  # Tính tổng của các số nguyên từ 0 đến x với x nhập từ bàn phím  puts "Chuong trinh se tinh tong cac so nguyen tu 0 den x. Moi ban nhap x (x > 0)"  a = gets.chomp  puts "- - - - Ket qua - - - -"  a = a.to\_i  tong = 0  (0..a).each{  |a|  tong += a  }  puts "Tong cua day so tu 0 den #{a} la : #{tong}"  puts "- - - - - - - - - - - -"  puts "Nhap \"y\" de quay lai menu chinh . Nhap ky tu khac de thoat."  thoat = gets.chomp  if thoat == "y"  next  else  puts "Chuc ban 1 ngay vui ve !!!"  break  end  elsif index.to\_i == 3  # In ra tam giác vuông  puts "- - - - Ket qua - - - -"  for i in 1..11 do  puts "\* " \* i  end  puts "- - - - - - - - - - - -"  puts "Nhap \"y\" de quay lai menu chinh . Nhap ky tu khac de thoat."  thoat = gets.chomp  if thoat == "y"  next  else  puts "Chuc ban 1 ngay vui ve !!!"  break  end  elsif index.to\_i == 4  puts "- - - - Ket qua - - - -"  # Phần code xử lý cho bài 4 chưa được hoàn thành  # Start code  #End code  puts "- - - - - - - - - - - -"  puts "Nhap \"y\" de quay lai menu chinh . Nhap ky tu khac de thoat."  thoat = gets.chomp  if thoat == "y"  next  else  puts "Chuc ban 1 ngay vui ve !!!"  break  end  elsif index.to\_i == 5  puts "Chuc ban 1 ngay vui ve !!!"  break  end  end |

Qua bài tập tổng hợp ta có thể rút ra 1 số chú ý khi làm bài sau:

* Dữ liệu nhập từ bàn phím luôn là String nên khi thao tác ta phải chuyển đổi về Fixnum hay Float tùy theo kiểu dữ liệu mà ta muốn thao tác
* Cần luyện tập nhiều hơn về phần vòng lặp vì vòng lặp trong Ruby rất đa dạng và nhiều biến hóa, cảm thấy chưa quen trong thời gian ngắn.

Video11: Hàm trong Ruby

# Bài tập giải phương trình a+ x = b. Tìm x

#Cú pháp viết hàm là def tên\_ham /\*Thân hàm\*/ end

#Truyền tham số bằng cách đặt tham số trong () bên cạnh tên hàm

def phuongtrinh(a,b)

#puts "Giai phuong trinh #{a} + x = #{b}: Ta duoc ket qua x = #{b-a}"

return "Giai phuong trinh #{a} + x = #{b}: Ta duoc ket qua x = #{b-a}"

end

#Gọi hàm

phuongtrinh(10,20)

# Nếu muốn có kết quả trả về thì sử dụng return

ketqua = phuongtrinh(3,10)

puts ketqua

# Nếu khi gọi làm mà ta ko truyền vào tham số thì sẽ báo lỗi

# ketqua = phuongtrinh 🡨 Không có giá trị tham số truyền vào

# Vì tham số trong phương trình chưa có giá trị mặc định. Khi ko có tham số được truyền vào thì khi gọi hàm, giá trị của tham số sẽ được lấy theo giá trị mặc định

Ví dụ:

def phuongtrinhbacnhat(a = 0 , b = 0)

puts "Nhap gia tri cua a:"

a = gets.chomp

a = a.to\_i # Convert về dữ liệu integer

puts "Nhap gia tri cua b:"

b = gets.chomp

b = b.to\_i # Convert về dữ liệu integer

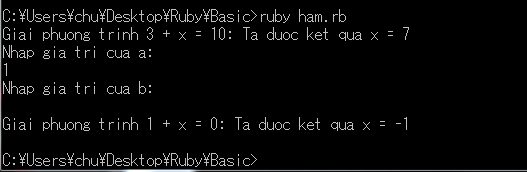
return "Giai phuong trinh #{a} + x = #{b}: Ta duoc ket qua x = #{b-a}"

end

# Gọi hàm mà ko truyền tham số thì giá trị của tham số sẽ nhận giá trị mặc định

puts phuongtrinhbacnhat

Kết quả thu được



Bài tập:

+ Để xác định 1 số có phải chẵn,lẻ hay không:

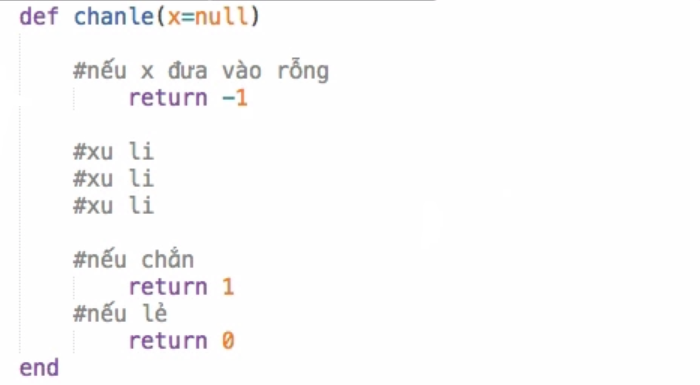
>x.odd?

true(nếu là số lẻ)/false(nếu là số chẵn)

>x.even?

true(nếu là số chẵn)/false(nếu là số lẻ)

Tuy nhiên bây giờ hãy xây dựng hàm xác định chẵn lẻ bằng Ruby



Đáp án:

|  |
| --- |
| # Hàm xác định 1 số là chăn hay lẻ  puts "================ Xac dinh chan le =================="  def chanle(x =null)  if x.empty? == true # Xác định chuỗi nhập vào là rỗng hay ko  return -1  # x nhập từ bàn phím nên x là String, phải chuyển về integer  elsif (x.to\_i % 2) == 0  return 1 #x là số chẵn  else  return 0 #x là số lẻ  end  end  # Main  puts "Nhap tu ban phim:"  a = gets.chomp  b = chanle(a)  puts "Ket qua : "  if b == -1  puts "Ban chua nhap gia tri"  elsif b == 0  puts "#{a} la so le"  else  puts "#{a} la so chan"  end |

Như vậy là mình đã kết thúc phần Ruby Basic tại đây, tiếp theo mình sẽ bắt đầu với phần Ruby Advance. Trong phần này thì mình sẽ học chủ yếu về hướng đối tượng của Ruby như là lớp, phương thức, thuộc tính, sử dụng các thư viện có sẵn, kế thừa,…

Video12: Lớp trong Ruby

1. Thuộc tính và hàm khởi tạo

Để khái báo lớp thì ta có thể khai báo

# Lớp trong Ruby

################################################

## Lớp cơ bản

################################################

# Xây dựng bằng từ khóa class và kết thúc bằng end

class test

# Thân class

end

# Tạo ra 1 đối tượng vienchu thuộc lớp người

vienchu = test.new

# Khởi tạo giá trị cho các thuộc tính của đối tượng vienchu

vienchu.ten = "Vien"

vienchu.ho = "Chu"

vienchu.sothich = "Doc sach"

vienchu.website = http://ChickenRain.com

# Gọi đối tượng và lấy giá trị của thuộc tính(sử dụng toán tử “.”)

puts vienchu.ten + vienchu.ho #VienChu

################################################

## Lớp có phương thức khởi tạo

################################################

class Nguoi

# Để lưu các thông tin của lớp người thì ta sẽ thêm thuộc tính cho nó

# Khai báo thuộc tính bằng từ khóa attr\_accessor

attr\_accessor :ten , :ho , :sothich , :website

# Hàm khởi tạo được bắt đầu với từ khóa

# def initialize(các thuộc tính muốn khởi tạo)

def initialize(ten,ho,sothich,website)

# Biến instance bắt đầu bằng ký tự @ đi kèm tên biến

# Sử dụng trong phạm vi lớp

@ten = ten

@ho = ho

@sothich = sothich

@website = website

end

end

# Bên cạnh việc ta đi gán giá trị cho từng thuộc tính của đối tượng

# Thì ta có thể sử dụng hàm khởi tạo để có thể khởi tạo trực tiếp giá trị cho các thuộc tính

#Việc sử dụng hàm khởi tạo contructor sẽ giúp ta rút ngắn được thời gian rất nhiều

# Thay vì phải ngồi khởi tạo giá trị cho các thuộc tính thì ta có thể gán trực tiếp nó vào

kimsan = Nguoi.new("Kim","Jongun","Doc sach", “ChickenRain.com”)

puts kimsan

Mình thắc mắc là liệ trong Ruby mình có thể viết nhiều phương thức khởi tạo như trong Java ? Và mình đã thử viết nhưng khi chương trình run sẽ báo lỗi mình thiếu parameter. Vậy thì phải làm cách nào ??

Và mình đã tìm ra 1 cách “chống chế” tạm thời như sau:

class Nguoi

attr\_accessor :ten , :tuoi , :sothich , :nangluong

def initialize(ten = nil , tuoi = nil , sothich = nil)

@ten = ten unless ten.nil?

@tuoi = tuoi unless tuoi.nil?

@sothich = sothich unless sothich.nil?

@nangluong = 100

end

def lam\_viec

puts "#{@ten} lam viec tu 10h sang den 20h dem! Vat va thiet do"

@nangluong -= 10

end

def an(ban = nil)

puts "#{@ten} di an cung #{ban}" unless ban.nil?

@nangluong += 10

end

end

vien = Nguoi.new("Vien Chu" ,23 , "Doc sach")

puts vien.nangluong #=> 100

vien.an #=> 110

vien.lam\_viec #=> 100

vien.an("Long") #=> 110

puts vien.nangluong #=>110

long = Nguoi.new("Long")

long.lam\_viec

Như vậy thì cả đối tượng vien và long đều được khởi tạo với đầu vào khác nhau mà vẫn không báo lỗi :D

1. Phương thức

#Phương thức trong Ruby

class Nguoi

# Biến instance sẽ có tác dụng trong tất cả các phương thức và toàn bộ lớp

# Khai báo các thuộc tính

attr\_accessor :ho, :ten , :tuoi , :sothich, :website , :nangluong

# Hàm khởi tạo Contructor

def initialize(ho,ten,tuoi,sothich,website)

@ho = ho

@ten = ten

@tuoi = tuoi

@sothich = sothich

@website = website

end

# Các phương thức của class

# Phương thức cũng như hàm. Được bắt đầu bằng từ khóa def và kết thúc bằng end

def thucday

puts "#{@ho + @ten} vua thuc day"

@nangluong = 100

end

def an

puts "#{@ho + @ten} dang an"

@nangluong += 5

end

def lamviec

puts "#{@ho + @ten} dang lam viec"

@nangluong -= 5

end

end

# Khai bao đối tượng vienchu với các thuộc tính

vienchu = Nguoi.new("Chu","Vien",23,"Nau an","http://xvideos.com")

puts vienchu.ten

# Thực thi các phương thức bằng cách sử dụng toán tử "."

puts "Nang luong ban dau : #{vienchu.nangluong}"

vienchu.thucday

puts "Nang luong sau khi thuc day: #{vienchu.nangluong}"

vienchu.an

puts "Nang luong sau khi an : #{vienchu.nangluong}"

vienchu.lamviec

puts "Nang luong sau khi lam viec: #{vienchu.nangluong}"

1. Thừa kế

# Kế thừa trong Ruby

class Nguoi

# Biến instance sẽ có tác dụng trong tất cả các phương thức và toàn bộ lớp

# Khai báo các thuộc tính

attr\_accessor :ho, :ten , :tuoi , :sothich, :website , :nangluong

# Hàm khởi tạo Contructor

def initialize(ho,ten,tuoi,sothich,website)

@ho = ho

@ten = ten

@tuoi = tuoi

@sothich = sothich

@website = website

end

# Các phương thức của class

# Phương thức cũng như hàm. Được bắt đầu bằng từ khóa def và kết thúc bằng end

def thucday

puts "#{@ho + @ten} vua thuc day"

@nangluong = 100

end

def an

puts "#{@ho + @ten} dang an"

@nangluong += 5

end

def lamviec

puts "#{@ho + @ten} dang lam viec"

@nangluong -= 5

end

end

# Kế thừa bằng cách < tên\_lớp\_được\_kế\_thừa

class GiaoVien < Nguoi

# Giáo viên cũng có những thuộc tính và phương thức như lớp Con người

# Nên thay vì phải coppy toàn bộ code của lớp connguoi thì ta sẽ thừa kế nó

#Ngoài những thuộc tính có sẵn thì mình có thể khai báo thêm các thuộc tính cho lớp GiaoVien

attr\_accessor :chuyennganh

# Ta có thể định nghĩa lại phương thức khởi tạo

def initialize(ho,ten,tuoi,sothich,chuyennganh)

@ho = ho

@ten = ten

@tuoi = tuoi

@sothich = sothich

@chuyennganh = chuyennganh

end

# Định nghĩa 1 phương thức mới

def code

@nangluong -= 5

end

#Ghi đè 1 phương thức đã tồn tại của lớp cha

def an

@nangluong += 10

end

end

kokubosan = GiaoVien.new("Hajime,"Kokubo",23,"Xem phim","Coder”, Engineer")

puts kokubosan.ten

kokubosan.thucday

kokubosan.code

puts "Nang luong sau khi code : #{kokubosan.nangluong}"

kokubosan.an

puts "Nang luong sau khi an : #{kokubosan.nangluong}"

Thừa kế sẽ giúp lập trình trở nên nhanh và dễ dàng hơn bằng cách sử dụng thuộc tính và phương thức từ những lớp thừa kế. Ngoài ra mình còn có thể tạo ra các phương thức mới.

Video13: Blocks trong Ruby

# Block chính là những thành phần câu lệnh nằm trong dấu ngoặc nhọn {} nếu nó nằm trên 1 dòng

# Hoặc nằm trong do...end nếu là nhiều dòng

#Ví du:

# Một dòng

# { puts "Day la Ruby Advance"}

# Nhiều dòng

# do

# puts “Ruby1”

# puts “Ruby2”

# end

# Sử dụng block trong hàm

def chickenrain(&block)

if block\_given? # Kiểm tra block có tồn tại hay ko

block.call #block ở đây chính là tên biến truyền vào

else

puts "Khong co block"

end

end

#Gọi method chickenrain và trong đó ta sẽ khai báo nội dung block

chickenrain{puts "Day la bai hoc ve Block"}

#Khai báo không có block

chickenrain # Kết quả: Không có block

# Không có block nên chương trình đã báo lỗi

#Vì vậy ta sử dụng 1 hàm check block có tôn tại hay chưa đó là block\_given?

#Nếu kết quả trả về tru nghĩa là có block , flase nghĩa là ko có block

# Ngoài ra thì Ruby có hỗ trợ thêm 1 từ khóa yield để lập trình nhanh hơn

# Nó giúp ta có thể ko cần nhớ tên block và cũng ko cần khai báo block

# yield sẽ đại diện cho nội dung bên trong block

# Nếu muốn truyền tham số vào trong yield thì ta viết tham số bên cạnh yield

def hocruby(name)

if block\_given?

yield name

else

puts "Khong co block"

end

end

# Tham số sẽ được gọi như sau

# "Vien Chu" - giá trị của tham số truyền vào

# |name| - biến đại diện cho tham số truyền vào

hocruby("Vien Chu"){ |name| puts "Xin chao #{name}"}

**Ngày 20/08/2015**

Video14: Procs và Lambda

# Proc và Lambda trong Ruby

#######################################################

## Block

#######################################################

m = [1,2,3,4,5]

n = [10,20,30,40,50]

# Xây dựng 1 hàm có sử dụng block

def double\_block(x)

if block\_given?

yield x\*2

else

puts "Khong co block"

end

end

puts "Mang m\_block"

puts m.map {

|arr| # Đưa từng phần tử của mảng m vào arr

#Gọi tới hàm double\_block

double\_block(arr){ # Cho arr trở thành tham số của block

|x| # Gán giấ trị tham số truyền vào trong yield vào biến x, ở đây x = arr\*2

"#{x}" # Thực thi in ra x

}

}

puts "Mang n\_block"

puts n.map{ |arr| double\_block(arr){|x| "#{x}"}}

#######################################################

## Proc

#######################################################

# Ta nhận thấy 2 đoạn code có nội dung hoàn toàn giống nhau

# Vì vậy ta sử dụng proc để ghép nó lại

double\_proc = Proc.new do

|t|

t\*2

end

#Bây giờ thay vì viết đoạn code dài như trên ta chỉ cần gọi proc đã tạo

# Chú ý proc được đặt trong () thay vì {} như trong block

puts "Mang m\_proc"

puts m.map(&double\_proc)

puts "Mang n\_proc"

puts n.map(&double\_proc)

#######################################################

## Lambda

#######################################################

double\_lambda = lambda do

|k|

k\*2

end

puts "Mang m\_lambda"

puts m.map(&double\_lambda)

puts "Mang n\_lambda"

puts n.map(&double\_lambda)

#######################################################

## Phân biệt Proc và Lambda

#######################################################

=begin

Ta có thể truyền nhiều tham số vào trong proc mà ko sử dụng thì khi run chương trình

cũng sẽ không có thông báo lỗi

double\_proc = Proc.new do

|t,k,x,y| # 4 tham số truyền vào

t\*2

end

puts m.map(&double\_proc) # Không phát sinh lỗi

Tuy nhiên Lambda thì có bao nhiêu tham số được khai báo thì cần sử dụng toàn bộ, nếu

không sẽ có lỗi

double\_lambda = lambda do

|k,m| # Truyền vào lambda 2 tham số

k\*2

end

puts m.map(&double\_lambda) # Thông báo lỗi do thiếu argument

=end

Ngoài ra mình có tham khảo thêm 1 số bài viết sau đây:

<http://rubysg.com/2015/04/17/su-dung-proc-va-lamda-cac-diem-khac-nhau/>

<https://techlistfortoday.wordpress.com/2015/07/04/phan-biet-block-proc-va-lambda-trong-ruby/>

Video15: Thao tác với File trong Ruby

# Trong phần này sẽ luyện tập về cách Ruby làm việc với các tệp tin(file)

# Đọc nội dung file

# Xóa file

# Thay đổi nội dung file

#########################################################

## Đọc file

#########################################################

# Cú pháp đọc file: File.read "tên\_file.txt"

puts File.read "test.txt"

# Kết quả thu được

#=> Xin chao, minh la Vien ,minh dang hoc Ruby de tro thanh Server Engineer

#########################################################

## Ghi file

#########################################################

# Ghi file bằng cú pháp

# File.write "new\_file.txt" , "nội\_dung\_file"

File.write "vidu.txt" , "Day la file moi"

File.write "vidu1.txt" , "Day la file moi tao ra"

#########################################################

## Đổi tên file đã có

#########################################################

# Mình sẽ đổi tên vidu.txt thành vidu1.txt

# File.rename "tên\_cũ.txt","tên\_mới.txt"

File.rename "vidu1.txt" , "vidu2.txt"

#########################################################

## Xóa file đã có

#########################################################

#Cú pháp: File.delete "tên\_file\_muốn\_xóa.txt"

File.delete "vidu2.txt"

#########################################################

## Gọi và in ra tên các phương thức có trong class File

#########################################################

puts File.methods

**Ngày 21/08/2015**

Ruby Regular Expertion

Link : <http://laptrinh.vn/d/5105-tim-hieu-ruby-regular-expression>

Symbol trong Ruby

Lúc nãy mình có làm 1 ví dụ với hash và thấy có cách khai báo như sau:

# Chú ý cách viết và khai báo hash này

hash = { Ruby: 1 , Python: 2 , Java: 3 }

ap hash

#######################

## each\_key , each\_value : Lặp với key , value

#######################

hash.each\_key do

|key|

puts "Key : #{key}"

end

hash.each\_value do

|value|

puts "Value : #{value}"

end

Trên đây mình tạm cung cấp 1 ví dụ, nhưng mình thấy lạ ở chỗ tại sao hash lại có thể khai báo như thế kia nhỉ?

Không phải cú pháp chính thống của Ruby khi khai báo hash là {key => value} hay sao ? Tại sao khi báo hash ={Ruby: 1} lại vẫn được chấp nhận nhỉ ? (Chú ý là dấu : phải đặt gần key). Thì hóa ra đây chính là symbol.Symbol là gì ? Symbol có thể được mô tả như là 1 định danh.Symbol sẽ diễn tả thông tin về “ai đó” , chứ ko phải về “cái gì đó”.

Để kiểm tra điều mình nói ở trên ta vào irb và test:

>:ruby.object\_id == :ruby.object\_id

#=> true

> “ruby”.object\_id == “ruby”.object\_id

#=> false

object\_id là phương thức trả về định danh của đối tượng. Nếu 2 đối tượng có chung 1 object\_id, thì 2 đối tượng là như nhau(cùng trỏ đến 1 đối tượng trong vùng nhớ). Như bạn có thể thấy, một khi đã sử dụng Symbol một lần, thì những Symbol có cùng ký tự đều tham chiếu đến cùng một đối tượng trong bộ nhớ. Với bất kỳ 2 Symbol nào có ký tự giống nhau, thì object\_id cũng giống nhau.

Bây giờ hãy xem chuỗi String (“ruby”). Hai object\_id không giống nhau. Điều đó nghĩa là nó được tham chiếu tới 2 đối tượng khác nhau trong vùng nhớ. Bất cứ khi nào bạn sử dụng new String, Ruby sẽ cấp phát vùng nhớ mới cho nó.

Nếu bạn phân vân trong việc sử dụng Symbol hay String, hãy xem xét xem cái gì quan trọng hơn: định danh của một đối tượng (i.e một Hash key), hay là nội dung của nó (như ví dụ bên trên là “ruby”).

Dấu : được dùng để định nghĩa symbol

Như vậy là mình đã hoàn thành việc học ngôn ngữ Ruby trong 6 ngày. Mặc dù chưa thực sự nắm vững ngôn ngữ này nhưng mình có thể dùng Ruby để viết 1 số chương trình đơn giản.

Ngày hôm nay mình chỉ toàn ngồi không, mà càng làm mình càng thấy kiến thức Ruby của mình chưa được chắc, còn rất nhiều chỗ bị hổng. Nên có lẽ mình nên giành thời gian để viết lại các kiến thức này. Mình sẽ học từ chính trang của tác giả viết:

<https://en.wikibooks.org/wiki/Ruby_Programming>

**Bài 1: Cú pháp trong Ruby**

1. Định danh trong Ruby

Cách thức đặt tên biến: Như hầu hết các ngôn ngữ lập trình khác thì định danh hợp lệ bao gồm các ký tự và các số (A-Za-z0-9) và dấu gạch dưới. Tuy nhiên trong Ruby, tên định danh không được bắt đầu bằng 1 chữ số. Độ dài định danh là ko giới hạn, tùy theo sở thích của lập trình viên mà có cách thức đặt tên khác nhau. Đô dài tên chỉ bị giới hạn bởi bộ nhớ của máy tính.

VD: foobar , ruby\_is\_simple , …

Có 1 chú ý trong việc đặt tên phương thức đó là tên phương thức có thể kết thúc bởi các ký tự đặc biệt như ? , ! và =

Sau này khi bạn đã nắm vững Ruby thì bàn sẽ hiểu mục đích sử dụng của những ký tự kết thúc đặc biệt này.

Có 1 vấn đề nữa trong việc khai báo tên biến đó là từ các phiển bản 2.x trở đi thì các tên định danh bắt đầu bằng chữ hoa được xem là Hằng, vì vậy mà các biến cục bộ ta nên bắt đầu bằng ký tự thường.

1. Comment trong Ruby

Trong Ruby thì comment được đắt đầu bằng # đối với những comment 1 dòng.

VD: # Đây là comment trong Ruby

1. Nhúng đoạn văn bản trong Ruby(comment dài)

Tuy nhiên để hỗ trợ những comment dài thì Ruby sử dụng 1 cặp ký tự

=begin # Định nghĩa cho việc bắt đầu đoạn comment

=end # Định nghĩa cho việc kết thúc comment

1. Extension (Biểu thức)

Tất cả các biến, các chữ, các cấu trúc điều khiển,.. đều là những biểu thứ. Chúng được sử dụng kết hợp với nhau để tạo thành chương trình. Bạn có thể chia các biểu thức ra thành các dòng mới, hoặc sử dụng dấu ; để ngăn cách chúng. Tuy nhiên nếu biểu thức vẫn thuộc 1 khối mà bạn muốn xuống dòng thì bạn cần đặt dấu \ ở phía trước phần mã thuộc các dòng tiếp theo.

VD:

puts “Toi” ; puts “la” # Các cấu trúc điều khiển nằm trong cùng 1 dòng

# Minh họa việc xuống dòng của biểu thức (dạng cấu trúc điều khiển)

puts “Toi la Chu Quang Vien , Hien tai

\ toi dang lam Game Engineer tai BOI”

**Bài 2: Biến và hằng**

1. Biến cục bộ (Local variables)

Biến cục bộ được bắt đầu bằng 1 chữ cái thường hoặc một dấu gạch dưới \_

VD: vien, \_vien

Phạm vi của 1 biến cục bộ đó là nó chỉ có thể được truy cập và sử dụng trong phạm vi khối khởi tạo nó. Nếu nó được khởi tạo trong phương thức thì chỉ có thể sử dụng trong phương thức , ra ngoài thì nó sẽ ko có giá trị. Nếu nó được khởi tạo trong lớp thì sẽ có giá trị trong lớp.

1. Biến toàn cục (global variables)

Biến toàn cục được bắt đầu bằng ký tự $

VD: $foobar , $Foobar

Biến toàn cục có thể được truy cập từ bất cứ nơi nào trong chương trình (thường được khai báo ở ngoài lớp).

VD:

$localvar = "hello" # khai báo biến localvar và gán giá trị

def var\_test{

$localvar = 10 # khai báo biến localvar và gán giá trị

puts $localvar # in giá trị của localvar ra màn hình

}

var\_test # gọi phương thức var\_test

puts $localvar #in giá trị của localvar ra màn hình

------

Kết quả :

#==> 10

#==> 10

Ta thấy, khi giá trị của biến toàn cục thay đổi ở trong phương thức, thì sự thay đổi này ảnh hưởng ra tới bên ngoài phương thức.

Như vậy, phạm vi của biến toàn cục là toàn bộ chương trình.

1. Biến đối tượng (Instance Variables)

Biến cá thể hay còn gọi là biến đối tượng, được bắt đầu bằng ký tự @

VD: @foobar

Biến đối tượng có nghĩa là nó chỉ thuộc về 1 đối tượng riêng lẻ hoặc một đối tượng của 1 lớp. Nó có thể được sử dụng giữa các method trong cùng 1 class(nhưng chỉ trong phạm vi class)

VD:

class Dog

def set\_name( aName )

@myname = aName

end

def get\_name

return @myname

end

def talk

return 'woof!'

end

end

Ta thấy, biến được bắt đầu bằng ký tự @ chỉ sử dụng trong các phương thức của lớp, và nó là dành riêng cho lớp. Ngoài ra thì có 1 chú ý đó là không thể truy cập trực tiếp 1 biến thể hiện nếu ở bên ngoài lớp. Bắt buộc phải viết các Setter và Getter cho 1 biến nếu muốn truy cập đến nó. Tuy nhiên Ruby đã hỗ trợ điều đó bằng cách sử dụng attr\_accessor giúp ta có thể truy cập đến 1 biến đối tượng ở ngoài lớp. Khi sử dụng method attr\_accessor thì Ruby đã ngầm định tạo ra 2 method là getter và setter.

1. Biến lớp (Class Variables)

Biến của lớp được bắt đầu bằng kí tự @@ được chia sẻ bởi tất cả các đối tượng trong lớp. Nó được tạo ra tại bất kì chỗ nào trong class.

VD : @@foobar

So sánh Instance Variable và Class Variable

Sự khác nhau giữa biến đối tượng (Instance Variables) và biến lớp (Class Variables) đó là biến class thì được chia sẻ cho cả các kế thừa của lớp đó, còn biến đối tượng thì ko, nó chỉ được sử dụng trong phạm vi của lớp đó mà thôi.

Tham khảo: <http://stackoverflow.com/questions/3802540/difference-between-class-variables-and-class-instance-variables>

Ví dụ

|  |  |
| --- | --- |
| Class Variables | Instance Variables |
| class Foo  @@i = 1  def self.i  @@i  end  def self.i=(value)  @@i = value  end  end  class Bar < Foo  end | class Foo  @i = 1  def self.i  @i  end  def self.i=(value)  @i = value  end  end  class Bar < Foo  end |
| p Foo.i # => 1  p Bar.i # => 1  Bar.i = 2  p Foo.i # => 2  p Bar.i # => 2 | p Foo.i # => 1  p Bar.i # => nil  Bar.i = 2  p Foo.i # => 1  p Bar.i # => 2 |

1. Hằng số

Một biến có định danh bắt đầu bằng 1 chữ cái viết hoa(A-Z) thì gọi là 1 hằng số. Tuy nhiên thông thường để dễ phân biệt thì ta viết hoa toàn bộ các chữ cái trong tên biến. Một hằng số có thể được gán giá trị sau khi khởi tạo, nhưng như vậy thì chương trình sẽ đưa ra cảnh báo, nên ta thường gán giá trị cho hằng ngay khi khởi tạo. Tất cả các class đều là 1 hằng (Vì tên class được bắt đầu bằng chữ in hoa). Việc cố gắng truy cập một hằng số chưa được khởi tạo thì sẽ đưa ra thông báo lỗi “NameError”.

Làm thế nào để sử dụng hằng số

Hằng số được truy cập trực tiếp trong phạm vi mà nó làm thành phần, thông qua toán tử “::”

VD:

class Foo

BAR = 123

end

puts Foo::BAR

#=> 123

Còn tiếp…