**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

BÁO CÁO SEMINAR

NGÔN NGỮ LẬP TRÌNH RUBY

**Môn học :** Nguyên lý các ngôn ngữ lập trình

**GVHD :** Phạm Trọng Nghĩa

**NGƯỜI THỰC HIỆN : 1512002 – Lê Dương Tuấn Anh**

**1512683 – Lý Anh Vũ**

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên, ĐHQG-HCM

Tháng 5/2019

**MỤC LỤC**

[1 Tổng quan 4](#_Toc10968695)

[Thông tin cá nhân 4](#_Toc10968696)

[Công cụ hỗ trợ 4](#_Toc10968697)

[Thông tin seminar 4](#_Toc10968698)

[2 Nội dung báo cáo 5](#_Toc10968699)

[2.1 Giới thiệu NNLT Ruby 5](#_Toc10968700)

[RUBY 5](#_Toc10968701)

[Cái tên Ruby 5](#_Toc10968702)

[Ngữ nghĩa 5](#_Toc10968703)

[2.2 Lịch sử phát triển 6](#_Toc10968704)

[2.3 Tính năng và một số vấn đề 7](#_Toc10968705)

[Tính năng 7](#_Toc10968706)

[Một số vấn đề 7](#_Toc10968707)

[2.4 Cách cài đặt setup môi trường 8](#_Toc10968708)

[2.5 Các thành phần của Ruby 9](#_Toc10968709)

[2.6 Hướng phát triển trong tương lai 12](#_Toc10968710)

[Ưu điểm 12](#_Toc10968711)

[Nhược điểm 12](#_Toc10968712)

[Ai sử dụng Ruby? 13](#_Toc10968713)

[Mức độ phổ biến 13](#_Toc10968714)

[2.7 Ruby on Rails 13](#_Toc10968715)

[Ruby on Rails là gì? 13](#_Toc10968716)

[Nguồn gốc của Rails 13](#_Toc10968717)

[Lợi ích khi sử dụng Ruby on Rails 14](#_Toc10968718)

[Nhược điểm của Ruby on Rails 14](#_Toc10968719)

[2.8 Demo 15](#_Toc10968720)

[Link demo 15](#_Toc10968721)

[Thuật toán tìm kiếm tuần tự 15](#_Toc10968722)

[Xây dựng một Class cơ bản 17](#_Toc10968723)

[Tính hướng đối tượng trong Ruby 19](#_Toc10968724)

[Hướng đối tượng nâng cao – Mẫu Singleton 22](#_Toc10968725)

[3 Tự đánh giá 23](#_Toc10968726)

# Tổng quan

### Thông tin cá nhân

|  |  |  |  |
| --- | --- | --- | --- |
| MSSV | Họ tên | Email | SĐT |
| 1512002 | Lê Dương Tuấn Anh | [leduongtuananh97@gmail.com](mailto:leduongtuananh97@gmail.com) | 0336062018 |
| 1512683 | Lý Anh Vũ | lyanhvu1512@gmail.com | 0972662851 |

### Công cụ hỗ trợ

- Bộ thông dịch cho Ruby, phiên bản 2.5.5 (<https://rubyinstaller.org/>).

- Sublime Text (Code Editor).

### Thông tin seminar

Giới thiệu ngôn ngữ lập trình RUBY, cách cài đặt và cách sử dụng. Lịch sử hình thành và khả năng phát triển trong tương lai của ngôn ngữ.

# Nội dung báo cáo

## Giới thiệu NNLT Ruby

### RUBY

**Ruby** là một ngôn ngữ lập trình hướng đối tượng được sử dụng để tạo ra rất nhiều loại ứng dụng web và di động. Theo tác giả, Ruby chịu ảnh hưởng bởi Perl, Smalltalk, Eiffel, Ada và Lisp. Ruby là ngôn ngữ thông dịch và hướng đối tượng.

Ruby cung cấp nhiều mẫu hình lập trình, bao gồm lập trình hàm, hướng đối tượng, mệnh lệnh, phản xạ. Nó sử dụng hệ thống kiểu biến động và tự động quản lý bộ nhớ tự động.

### Cái tên Ruby

Nguồn gốc của cái tên "Ruby" là từ một phiên chat online giữa Matsumoto và Ishitsuka Keiji vào ngày 24 tháng 2 năm 1993, trước khi viết code cho ngôn ngữ này. Ban đầu, "Coral" và "Ruby" là hai cái tên được đề xuất. Matsumoto chọn cái cuối cùng trong email gửi đến Ishitsuka. Sau này Matz cũng bất ngờ khi phát hiện ra Pearl là viên đá quý tượng trưng cho những người sinh tháng 6, còn Ruby thì tượng trưng cho những người sinh tháng 7. Anh cho rằng cái tên Ruby như thế là phù hợp vì Ruby kế thừa và phát triển nhiều đặc tính từ Perl.

### Ngữ nghĩa

Ruby là một ngôn ngữ hướng đối tượng: mỗi bit dữ liệu đều là một đối tượng (object), bao gồm các kiểu dữ liệu mà đối với các ngôn ngữ khác, chúng là kiểu cơ bản (primitive) như integer. Mỗi hàm (function) là một phương thức (method). Tên biến (variables) chính là tham chiếu (references) đến các đối tượng, bản thân nó không phải là đối tượng. Ruby hỗ trợ kế thừa (inheritance) với dynamic dispatch, mixin và singleton method (thuộc về, và để định nghĩa cho, một instance đơn hơn là định nghĩa dành cho lớp). Mặc dù Ruby không hỗ trợ đa kế thừa, các lớp vẫn có thể được đưa vào các module dưới dạng các mixins. Cú pháp dạng thủ tục (procedural syntax) vẫn còn được hỗ trợ, có vẻ như là ngoài tầm vực của mọi đối tượng, nhưng thực sự là thuộc một thể hiện của class Object tên là 'main'. Vì class này là cha của mọi class khác, nó trở trên ẩn đối với mọi lớp và đối tượng.

Ruby được xem là một ngôn ngữ lập trình đa mẫu hình (multi-paradigm programming language): nó cho phép bạn lập trình dạng thủ tục (tạo ra các hàm/biến nằm ngoài phạm vi của các lớp và biến chúng thành một phần của đối tượng gốc, 'self' Object), với khả năng hướng đối tượng (mọi thứ đều là đối tượng) hay hàm (nó có các hàm không có tên (anonymous functions), closures, và continuations; mọi câu lệnh đều có giá trị trả về, và các hàm đều trả về kết quả ước lượng cuối cùng). Nó hỗ trợ mạnh cho tự định kiểu (type introspection), reflection và meta-programming.

## Lịch sử phát triển

Ruby được tạo ra bởi Yukihiro "Matz" Matsumoto với mục đích “Làm cho lập trình viên hạnh phúc” từ 24 tháng 2 năm 1993 và đưa ra phiên bản chính thức 0.95 (Đây là phiên bản công khai đầu tiên của Ruby được công bố tại Nhật Bản) vào ngày 21 tháng 12 năm 1995. Ở phiên bản Ruby 0.95, nó đã có nhiều tính năng quen thuộc trong các phiên bản sau của Ruby, bao gồm thiết kế hướng đối tượng, các lớp với kế thừa, mixin, vòng lặp, xử lý ngoại lệ và thu gom rác.

Sau khi phát hành Ruby 0.95 vào năm 1995, một số phiên bản ổn định của Ruby đã được phát hành trong những năm tiếp theo:

* Ruby 1.0: ngày 25 tháng 12 năm 1996.
* Ruby 1.2: tháng 12 năm 1998.
* Ruby 1.4: tháng 8 năm 1999.
* Ruby 1.6: tháng 9 năm 2000.

Hiện nay, phiên bản ổn định mới nhất là 2.6.3.

## Tính năng và một số vấn đề

### Tính năng

* Hướng đối tượng
* Xử lý ngoại lệ
* Iterator và closures (dựa vào cách truyền một khối lệnh)
* Native, biểu thức chính quy như Perl ở mức độ ngôn ngữ
* Chồng toán tử (operator overloading)
* Thu hồi rác tự động
* Tính khả chuyển cao (portable)
* Có thể chạy đa luồng (multi-threading) trên mọi platforms
* Khả năng nạp động các DLL/thư viện chia sẻ trên hầu hết mọi platforms.
* Tự kiểm tra (introspection), reflection và meta-programming
* Rất nhiều thư viện chuẩn
* Hỗ trợ dependency injection
* Continuation và generator
* Dùng khối lệnh (code block)

Ruby hiện tại vẫn chưa hỗ trợ Unicode, dù chỉ mới hỗ trợ tạm thời cho UTF-8.

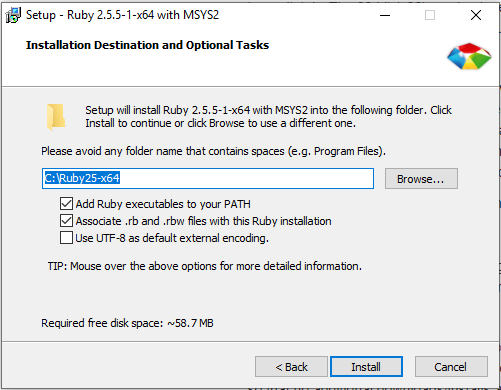
### Một số vấn đề

* Các tên bắt đầu bằng ký tự hoa được xem là hằng, vì thế biến cục bộ nên bắt đầu bằng ký tự thường.
* Việc đánh giá Boolean đối với các dữ liệu không phải bool rất chặt chẽ.
* Để biểu diễn một số thực dấu chấm động, ta phải theo quy tắc dùng ký số zero (99.0) hay chuyển đổi tường minh (99.to\_f). Việc dùng dấu chấm là không đủ (99.) vì các số vẫn có thể nhận cú pháp có phương thức.
* Không có kiểu dữ liệu kí tự (char). Khi duyệt qua chuỗi: "abc"[0] cho ra 97 (một số nguyên, biểu diễn mã ASCII của ký tự đầu tiên trong chuỗi); để lấy được "a" dùng "abc"[0,1] (chuỗi con có chiều dài 1) hay "abc"[0].chr.

## Cách cài đặt setup môi trường

Download trình thông dịch Ruby tại <https://rubyinstaller.org/downloads/>. Tại trang này còn cung cấp cho người dùng Devkit (trên nền tảng MSYS2 Toolchain). Devkit này là bắt buộc nếu người lập trình Ruby muốn sử dụng extensions native C/C++, đặc biệt cần thiết cho framework Ruby on Rail và rất nhiều thư viện khác.

Lưu ý, cần tick vào tùy chọn “Add Ruby executables to your PATH” để có thể dùng Ruby từ Command Line.



Sau khi cài đặt xong, mở Command Line, gõ lệnh **irb -v (Interative Ruby)** để kiểm tra cài đặt thành công như hình hay chưa.

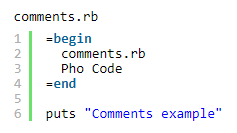


Nếu như hình thì việc cài đặt đã hoàn tất!

## Các thành phần của Ruby

#### Bình luận – Comment

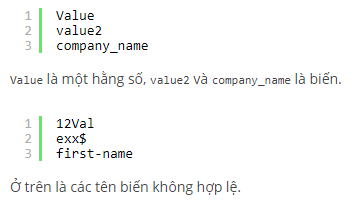
Các đoạn comment được dùng để ghi chú mã nguồn. Cú pháp comment của Ruby có 2 loại là comment cho một dòng và comment cho nhiều dòng. Comment trên một dòng được bắt đầu bởi dấu #, comment trên nhiều dòng được bọc bởi cặp kí hiệu =begin và =end.



#### Biến – Variable

Biến chỉ là một cái tên, đại diện cho một thứ gì đó có công việc là lưu trữ một giá trị nào đó. Trong lập trình thì chúng ta nói là “gán giá trị cho biến”, giá trị ở đây có thể là một đoạn text, một con số hay một đối tượng nào đó.

Tên của các biến được đặt bằng các kí tự trong bảng chữ cái và dấu gạch dưới, nhưng không được bắt đầu bằng kí tự số, cũng không được bắt đầu bằng kí tự viết HOA, nếu dùng chữ HOA thì Ruby sẽ gọi đây là “hằng số”.



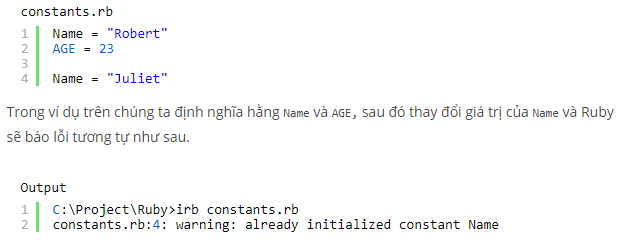
Tên biến có thể được bắt đầu bởi 2 kí tự đặc biệt là @ và $.

Biến trong Ruby là có phân biệt HOA-thường, tức là price và pRice là 2 biến khác nhau.

#### Hằng số - Constant

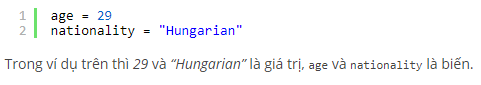
Giá trị của các hằng số trong Ruby là có thể thay đổi được, khi chúng ta thay đổi giá trị hằng thì Ruby không báo lỗi mà chỉ đưa ra mấy dòng cảnh báo.

Tên hằng số được bắt đầu bởi một kí tự viết HOA, thường thì khi đặt tên hằng chúng ta luôn viết hoa toàn bộ các kí tự trong tên.



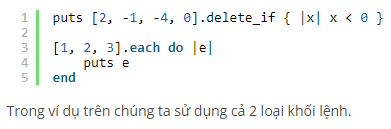
#### Giá trị - Literal

Giá trị là các kí tự mô tả một giá trị của một kiểu dữ liệu nào đó, có thể là một con số, một đoạn text… dùng để gán cho các biến.



#### Khối lệnh - Block

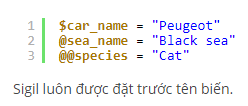
Khối lệnh là cách để chúng ta gộp nhóm các lệnh lại với nhau. Khối lệnh trong Ruby được bắt đầu và kết thúc bởi cặp dấu {} hoặc cặp từ khóa do-end.



Ngoài các câu lệnh tính toán bình thường thì trong lập trình còn có các câu lệnh điều khiển, ví dụ như câu lệnh if, đây là câu lệnh điều kiện, theo sau if là một biểu thức rồi tới một khối lệnh nằm trong cặp từ khóa then-end.

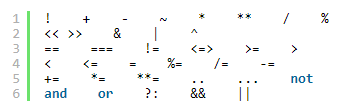
#### Sigil

Sigil là các kí tự $ và @ dùng để khai báo phạm vi hoạt động của biến. Trong đó $ cho biết biến đó là một biến toàn cục, @ cho biết đó là biến instance, @@ là biến class.



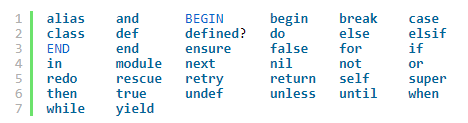
#### Toán tử

Toán tử là các kí tự thực hiện các hành động nào đó trên một giá trị nào đó.



#### Từ khóa

Từ khóa là các từ ưu tiên trong Ruby, thường dùng làm các câu lệnh thực hiện hành động nào đó, chẳng hạn như in giá trị ra màn hình, thực hiện các công việc lặp đi lặp lại hay thực hiện tính toán. Khi đặt tên biến chúng ta không được đặt tên trùng với từ khóa.



## Hướng phát triển trong tương lai

Ruby (ngôn ngữ lập trình) chạy với Ruby on Rails là một mã nguồn mở, full-stack web application framework. Đây là một ngôn ngữ lập trình hướng đối tượng mệnh lệnh năng động. Hệ thống loại năng động và quản lý bộ nhớ tự động.

### Ưu điểm

* Mã nguồn mở
* Hoạt động trên nhiều nền tảng
* Có thể nhúng vào HTML
* Ngôn ngữ cấp cao
* Cung cấp các phương pháp đóng gói dữ liệu trong các đối tượng
* OOP tinh khiết (Lập trình hướng đối tượng)
* Kỹ thuật chuỗi và văn bản thao tác siêu tiên tiến
* Có thể dễ dàng kết nối với DB2, MySQL, Oracle, và Sybase
* Các chương trình lớn và mở rộng cao dễ dàng được bảo trì
* Có một cú pháp sạch sẽ và dễ dàng cho phép các nhà phát triển mới để tìm hiểu Ruby rất nhanh chóng và dễ dàng
* Có khả năng viết các ứng dụng đa luồng với một API đơn giản
* Cung cấp lớp màn tiên tiến
* Có thể viết thư viện bên ngoài Ruby hoặc C
* Tính năng bảo mật tốt hơn
* Có một trình gỡ lỗi
* Cú pháp linh hoạt
* Chuỗi xử lý mạnh mẽ

### Nhược điểm

* Việc học ngôn ngữ Ruby có thể khó khăn
* Thiếu nguồn thông tin, tài liệu
* Thời gian xử lý chậm hơn (thời gian CPU) so với các ngôn ngữ lập trình khác
* Phát triển và cập nhật chậm hơn
* Lập trình nhanh hơn do ít dòng lệnh hơn. Tuy nhiên nó sẽ khó cho việc bảo trì và tốc độ chạy chương trình cũng sẽ chậm hơn

### Ai sử dụng Ruby?

* Google Sketchup
* 37signals
* GitHub
* Shopify
* Indiegogo
* Basecamp

### Mức độ phổ biến

Theo trang web của Ruby , nó được xếp hạng 10 trong hầu hết các ngôn ngữ lập trình phổ biến trên toàn thế giới. Phần lớn sự tăng trưởng là do mức độ phổ biến của phần mềm viết bằng Ruby, đặc biệt là web framework: ***Ruby on Rails***.

## Ruby on Rails

### Ruby on Rails là gì?

Ruby on Rails là một Framework cho phép phát triển ứng dụng Web gồm 2 phần cơ bản:

* Phần ngôn ngữ Ruby: “Ruby là một ngôn lập trình mã nguồn mở, linh hoạt, với một sự nổi bật về sự đơn giản dễ dùng và hữu ích. Nó có cú pháp “tao nhã” và tự nhiên dễ đọc và dễ dàng để viết”.
* Phần Framework Rails bao gồm nhiều thư viện liên kết.

Ruby on Rails được ra đời một cách hoàn hảo dành cho những ai luôn “mải miết đi tìm” câu trả lời: tại sao những lập trình viên luôn tìm mọi cách để phát triển những phần mềm, ứng dụng Web, … để người sử dụng được thuận tiện nhất, nhưng lại không “thuận tiện” cho người lập trình mỗi khi chương trình càng đơn giản thì bên trong nó lại càng phức tạp, khó chỉnh sửa khi cần thiết, làm cho công việc của người lập trình viên “cảm thấy mệt mỏi” hơn. Ruby on Rails đã trả lời được câu hỏi đó.

### Nguồn gốc của Rails

Rails ra mắt công chúng lần đầu tiên vào năm 2004, Rails thoạt đầu được dùng như là nền tảng cho một công cụ quản lý dự án được đặt tên là Basecamp và được tạo ra bởi nhà phát triển web David Heinemeier Hansson, một nhân viên của công ty phát triển web 37signals (Mỹ). Ban đầu họ xây dựng Rails không phải với mục đích là xây dựng ra một framework riêng, chủ tâm ban đầu là dùng nó để xây dựng các ứng dụng khác của 37signals. Sau đó Heinemeier Hansson thấy tiềm năng của nó giúp cho anh ấy làm các công việc dễ dàng hơn bằng cách rút ra các tính năng phổ biến như trừu tượng cơ sở dữ liệu và khuôn mẫu(template) bên trong, và sau đó nó trở thành phiên bản đầu tiên được tung ra của Ruby on Rails.

### Lợi ích khi sử dụng Ruby on Rails

Ruby cung cấp cho chúng ta một sự kết hợp giữa những các công cụ tốt nhất, thư viện code chất lượng và cách tiếp cận tốt tới phần mềm. Bên cạnh đó cộng đồng Ruby cũng cực kỳ lớn.

* **Công cụ:** Rails cung cấp cho ta công cụ tuyệt vời giúp chúng ta triển khai được nhiều tính năng hơn mà lại tốn ít thời gian hơn. Nó cung cấp cho ta một cấu trúc chuẩn cho ứng dụng web.
* **Thư viện:** Rails cung cấp cho ta gem, thứ có thẻ thay thế hầu hết cho mọi thứ. Hơn nữa, tất cả gem đều có thể sử dụng một cách hoàn toàn miền phí và có thể dễ dàng tra cứu tại <https://rubygems.org/>.
* **Chất lượng code:** Chất lượng của các phần mềm hỗ trợ Ruby code là cao hơn so với một số ngôn ngữ khác.
* **Cộng đồng:** Cộng đồng Ruby rất lớn. Điều này giúp cải thiện những sản phẩm của Ruby rất nhiều và đây cũng là một lý do mà thư viện của Ruby lại tuyệt vời như vậy. Ruby cũng là một trong số những ngôn ngữ lập trình phổ biến nhất trên Github.
* **Hiệu suất:** RoR là một ngôn ngữ gọn gàng, khi mà sử dụng kết hợp cùng các thư viện hỗ trợ, RoR cho phép bạn phát triển một cách khá là nhanh chóng.
* **Tương lai:** RoR có thể được xem là ngôn ngữ được lựa chọn trong các trang học lập trình trực tuyến phổ biến như [Makers Academy](http://www.makersacademy.com/), [CodeCademy](https://www.codecademy.com/). Điều này có nghĩa là số lượng lập trình viên tài giỏi tham gia vào cộng đồng Ruby sẽ tăng lên trong các năm tới.

### Nhược điểm của Ruby on Rails

Đương nhiên không có một ngôn ngữ nào là hoàn hảo và Rail cũng không phải là một ngoại lệ. Dưới đây là những nhược điểm đáng lưu ý của Rails:

* **Thời gian chạy:** Phần lớn những cuộc tranh luận về Rail đều có những ý kiến rằng Rails chậm. Điều này hoàn toàn đúng, so với NodeJS hay GoLang, Rails có tốc độ chạy khá chậm. Mặc dù trong thực thế, hiệu suất của ứng dụng Rails rất khó làm chậm lại việc kinh doanh, Vấn đề thường nằm ở việc thiết kế server, thậm chí là cả cơ sở dữ liệu.
* **Tốc độ boot:** Vấn đề chính gây khó chịu khi làm việc với Rails là tốc độ boot của Rails framework. Phụ thuộc vào số lượng gem và file, chúng ta có thể mất một khoảng thời gian kha khá để khởi động, điều này có thể ảnh hưởng trực tiếp tới hiệu suất làm việc. Với việc Spring ra đời, tốc độ boot đã được cải thiện nhưng có vẻ như nó vẫn còn có thể nhanh hơn nữa.

## Demo

### Link demo

<https://youtu.be/G83KttPEVjI>

### Thuật toán tìm kiếm tuần tự

Ta thử xây dựng một đoạn code ngắn với mục đích tìm kiếm xem một giá trị nào đó có nằm trong mảng cho trước hay là không, bằng ngôn ngữ Ruby.

Thuật toán này sẽ quét qua tất cả các phần tử trong mảng, in ra vị trí và giá trị tương ứng tìm thấy.

# Có thể comment trong Ruby bằng dấu #

# Khai báo một mảng Array có phần tử là các loại khác nhau

array **=** ['x', 0, 123, "123", **nil**, **true**]

# Lấy giá trị độ dài mảng thông qua giá trị .length

n **=** array.length

# Key là giá trị phần tử cần tìm

key **=** 123

# Duyệt qua các phần tử từ 0 đến n-1

**for** i **in** 0..n**-**1 **do**

# Kiểm tra điều kiện

**if** array[i] **==** key

# In ra vị trí và giá trị

# Hàm .to\_s là để chuyển đổi integer sang string

puts 'Found at ' **+** i.to\_s **+** ', with value: ' **+** array[i].to\_s **+** '.'

**end**

**end**

Lưu file với tên **search.rb (đuôi cho các chương trình Ruby thường là \*.rb).** Thông dịch với câu lệnh **irb a.rb**. Kết quả:

D:\Code**>**irb search.rb

a.rb(main):001:0**>** # Có thể comment trong Ruby bằng dấu #

**=>** nil

a.rb(main):002:0**>** # Khai báo một mảng Array có phần tử là các loại khác nhau

**=>** nil

a.rb(main):003:0**>** array **=** ['x', 0, 123, "123", nil, **true**]

**=>** ["x", 0, 123, "123", nil, **true**]

a.rb(main):004:0**>** # Lấy giá trị độ dài mảng thông qua giá trị .length

**=>** nil

a.rb(main):005:0**>** n **=** array.length

**=>** 6

a.rb(main):006:0**>** # Key là giá trị phần tử cần tìm

**=>** nil

a.rb(main):007:0**>** key **=** 123

**=>** 123

a.rb(main):008:0**>**

**=>** nil

a.rb(main):009:0**>** # Duyệt qua các phần tử từ 0 đến n**-**1

**=>** nil

a.rb(main):010:0**>** **for** i in 0..n**-**1 **do**

a.rb(main):011:1**\*** # Kiểm tra điều kiện

a.rb(main):012:1**\*** **if** array[i] **==** key

a.rb(main):013:2**>** # In ra vị trí và giá trị

a.rb(main):014:2**>** # Hàm .to\_s là để chuyển đổi integer sang string

a.rb(main):015:2**>** puts 'Found at ' **+** i.to\_s **+** ', with value: ' **+** array[i].to\_s **+** '.'

a.rb(main):016:2**>** end

a.rb(main):017:1**>** end

Found at 2, with value: 123 // Tìm được ở vị trí 2 trong mảng (bắt đầu từ 0)

**=>** 0..5

Phân tích đoạn code trên, ta thấy được:

- Với các dòng để comment, Ruby sẽ bỏ qua, trả về “nil” tương ứng.

a.rb(main):001:0**>** # Có thể comment trong Ruby bằng dấu #

**=>** nil

a.rb(main):002:0**>** # Khai báo một mảng Array có phần tử là các loại khác nhau

**=>** nil

- Các dòng khai báo các biến, Ruby sẽ in ra màn hình giá trị được gán tương ứng:

a.rb(main):003:0**>** array **=** ['x', 0, 123, "123", nil, **true**]

**=>** ["x", 0, 123, "123", nil, **true**]

- Với các khối lệnh như **if, for, while, v.v…** ruby sẽ chạy một lượt cho đến kết thúc (là từ khóa **end**), không in ra từng dòng. Kết quả in nếu có sẽ in ra sau đó:

a.rb(main):010:0**>** **for** i in 0..n**-**1 **do**

a.rb(main):011:1**\*** # Kiểm tra điều kiện

a.rb(main):012:1**\*** **if** array[i] **==** key

a.rb(main):013:2**>** # In ra vị trí và giá trị

a.rb(main):014:2**>** # Hàm .to\_s là để chuyển đổi integer sang string

a.rb(main):015:2**>** puts 'Found at ' **+** i.to\_s **+** ', with value: ' **+** array[i].to\_s **+** '.'

a.rb(main):016:2**>** end

a.rb(main):017:1**>** end

Found at 2, with value: 123 // Đến đây mới in ra nếu cần

**=>** 0..5 // Loop-Range

### Xây dựng một Class cơ bản

Ta sẽ xây dựng một class cơ bản **Song** sẽ chứa các thông tin cơ bản của một bài hát như tên (**name**), tên tác giả (**artist**) và độ dài bài hát (**duration**). Các định nghĩa như sau:

**class** Song

    # initialize là hàm khởi tạo, sẽ được chạy khi gọi new Song

**def** initialize(name, artist, duration)

        # Các biến bắt đầu bằng dấu @ là biến trong class, cho mỗi instance khácn nhau

        @name **=** name

        @artist **=** artist

        @duration **=** duration

**end**

**end**

# Tạo một bài hát mới

quoc\_ca **=** Song.**new**('Quoc Ca', 'Van Cao', '3:45')

# Thử in class Song dưới dạng String

# Vì mọi Class trong Ruby đều kế thừa class object

# Nên mặc định sẽ có hàm to\_s (to\_string)

puts(quoc\_ca)

Kết quả khi chạy chương trình:

a.rb(main):001:0**>** class Song

a.rb(main):002:1**>** # initialize là hàm khởi tạo, sẽ được chạy khi gọi new Song

a.rb(main):003:1**>** def initialize(name, artist, duration)

a.rb(main):004:2**>** # Các biến bắt đầu bằng dấu @ là biến trong class, cho mỗi instance khácn nhau

a.rb(main):005:2**>** @name **=** name

a.rb(main):006:2**>** @artist **=** artist

a.rb(main):007:2**>** @duration **=** duration

a.rb(main):008:2**>** end

a.rb(main):009:1**>** end

**=>** :initialize

a.rb(main):010:0**>**

**=>** nil

a.rb(main):011:0**>** # Tạo một bài hát mới

**=>** nil

a.rb(main):012:0**>** quoc\_ca **=** Song.new('Quoc Ca', 'Van Cao', '3:45')

**=>** #**<**Song:0x000000000313b890 @name**=**"Quoc Ca", @artist**=**"Van Cao", @duration**=**"3:45"**>**

a.rb(main):013:0**>** # Thử in class Song dưới dạng String

**=>** nil

a.rb(main):014:0**>** # Vì mọi Class trong Ruby đều kế thừa class object

**=>** nil

a.rb(main):015:0**>** # Nên mặc định sẽ có hàm to\_s (to\_string)

**=>** nil

a.rb(main):016:0**>** puts(quoc\_ca)a.rb(main):016:0**>**

#**<**Song:0x000000000313b890**>**

**=>** nil

a.rb(main):016:0**>**

Dễ thấy, hàm **to\_s** đã hoạt động không hiệu quả. Ta sẽ override lại phương thức này ngay trong lớp Song:

class Song

    def initialize(name, artist, duration)

        @name = name

        @artist = artist

        @duration = duration

    end

    # Override lại to\_s

    def to\_s

        # Cú pháp #{} cho phép đặt các biến, phép tính, v.v...

        # với cấu trúc Ruby vào bên trong

        # Ở đây ta dùng các biến Instance

        "Song: #{@name} - By #{@artist} (#{@duration})"

    end

end

quoc\_ca = Song.new('Quoc Ca', 'Van Cao', '3:45')

puts(quoc\_ca)

Lúc này, kết quả:

a.rb(main):018:0**>** puts(quoc\_ca)a.rb(main):018:0**>**

Song: Quoc Ca **-** By Van Cao (3:45)

**=>** nil

### Tính hướng đối tượng trong Ruby

Ta sẽ xây dựng một lớp **KaraokeSong** kế thừa trực tiếp lớp **Song** đã dùng ở trên

class Song

    def initialize(name, artist, duration)

        @name = name

        @artist = artist

        @duration = duration

    end

    def to\_s

        "Song: #{@name} - By #{@artist} (#{@duration})"

    end

end

# Kế thừa. Kí tự < chỉ việc kế thừa

class KaraokeSong < Song

    # Các tham số vẫn truyền vào hàm initialize

    def initialize(name, artist, duration, lyrics)

        # Hàm super cho phép truyền các tham số trên

        # Vào hàm khởi tạo của lớp cha

        # đồng thời trả ngược các biến instance vào lớp này

        super(name, artist, duration)

        # Ta có thêm biến lyrics

        @lyrics = lyrics

    end

end

quoc\_ca = KaraokeSong.new('Quoc Ca', 'Van Cao', '3:45', 'Doan quan Viet Nam di...')

puts(quoc\_ca.to\_s)

Kết quả:

a.rb(main):026:0**>** quoc\_ca **=** KaraokeSong.new('Quoc Ca', 'Van Cao', '3:45', 'Doan quan Viet Nam di...')

**=>** #**<**KaraokeSong:0x000000000337b7e0 @name**=**"Quoc Ca", @artist**=**"Van Cao", @duration**=**"3:45", @lyrics**=**"Doan quan Viet Nam di..."**>**

a.rb(main):027:0**>** puts(quoc\_ca.to\_s)a.rb(main):027:0**>**

Song: Quoc Ca **-** By Van Cao (3:45)

**=>** nil

Ta thấy được biến **lyrics** chưa xuất hiện, lí do vì hàm **to\_s** được kế thừa từ lớp **Song** không có thông tin này. Tương tự, ta sẽ khai báo một hàm **to\_s** cho lớp **KaraokeSong**:

def to\_s

        "Song: #{@name} - By #{@artist} (#{@duration}) : #{@lyrics}"

    end

Kết quả đạt đuơc:

a.rb(main):031:0**>** puts(quoc\_ca.to\_s)a.rb(main):031:0**>**

Song: Quoc Ca **-** By Van Cao (3:45) : Doan quan Viet Nam di...

**=>** nil

Tuy nhiên, rõ ràng cách này rất “hard-core”! Bởi ta hoàn toàn có thể kế thừa **1 phần** hàm **to\_s** của lớp cha. Ruby hỗ trợ rất tốt phần này bằng cách khai báo sau:

def to\_s

        # Hàm super gọi chính to\_s của cha

        super + " : #{@lyrics}"

    end

Ta cũng cho kết quả tương tự. Hàm **super** còn cho phép ta thực hiện rất nhiều thao tác liên quan đến việc kế thừa. Trong khuôn khổ báo cáo này, nhóm xin phép được thực hiện mẫu đến đây.

Ngoài ra, Ruby còn cho phép khai báo **biến toàn cục Class,** tức là **biến này xuất hiện trong tất cả các biến có kiểu là Class tương ứng.** Việc khai báo như sau:

class Song

    # Biến toàn cục Class khai báo bắt đầu bằng 2 dấu @@

    @@plays = 0

    def initialize(name, artist, duration)

        @name = name

        @artist = artist

        @duration = duration

        @plays = 0

    end

    def to\_s

        "Song: #{@name} - By #{@artist} (#{@duration})"

    end

    def play

        @plays += 1

        @@plays += 1

        # Biến @ và @@ có khả năng phân biệt là 2 biến khác nhau

        "This song: #{@name} plays #{@plays} times. Total play: #{@@plays}"

    end

end

baihat1 = Song.new('Bai Hat 1', 'Tac Gia 1', '01:23')

baihat2 = Song.new('Bai Hat 2', 'Tac Gia 2', '45:67')

baihat1.play

baihat1.play

baihat2.play

baihat1.play

baihat2.play

Kết quả:

a.rb(main):026:0**>** baihat1.play

**=>** "This song: Bai Hat 1 plays 1 times. Total play: 1"

a.rb(main):027:0**>** baihat1.play

**=>** "This song: Bai Hat 1 plays 2 times. Total play: 2"

a.rb(main):028:0**>** baihat2.play

**=>** "This song: Bai Hat 2 plays 1 times. Total play: 3"

a.rb(main):029:0**>** baihat1.play

**=>** "This song: Bai Hat 1 plays 3 times. Total play: 4"

a.rb(main):030:0**>** baihat2.play

**=>** "This song: Bai Hat 2 plays 2 times. Total play: 5"

a.rb(main):031:0**>**

**=>** nil

### Hướng đối tượng nâng cao – Mẫu Singleton

Như đã giới thiệu, Ruby hỗ trợ rất tốt trong việc lập trình hướng đối tượng. Một trong những mẫu lập trình hướng đối tượng đơn giản nhất có thể được khai báo đó là mẫu **Singleton** (<https://en.wikipedia.org/wiki/Singleton_pattern>). Ví dụ về class **Logger** có thể được khai báo như sau:

* Khai báo từ khoá **new** trở thành 1 private method, vì vậy không thể gọi từ bên ngoài.
* Truy xuất instance của Logger thông qua hàm **create**. Hàm này sẽ **kiểm tra xem Logger.create có bao giờ được gọi hay chưa. Nếu rồi sẽ trả về instance đã tạo. Nếu chưa sẽ tạo ra instance mới**.
* Dùng loại biến **@@** thay vì **@** cho instance tạo ra trong nội tại class**, bởi instance này là chia sẻ bởi các class instance** (*lưu ý, chỉ có thể truy xuất trực tiếp qua class).*

class Logger

    # Không cho phép gọi new từ bên ngoài class

    private\_class\_method :new

    @@logger = nil

    # Chỉ cho phép truy cập qua hàm này

    def Logger.create

        @@logger = new unless @@logger

        @@logger

    end

end

# Lấy id trực tiếp

Logger.create

# Thử gọi lần 2 xem có khác biệt hay không

Logger.create

Kết quả chạy chương trình:

a.rb(main):001:0**>** class Logger

a.rb(main):002:1**>** # Không cho phép gọi new từ bên ngoài class

a.rb(main):003:1**>** private\_class\_method :new

a.rb(main):004:1**>** @@logger **=** nil

a.rb(main):005:1**>**

a.rb(main):006:1**>** # Chỉ cho phép truy cập qua hàm này

a.rb(main):007:1**>** def Logger.create

a.rb(main):008:2**>** @@logger **=** new unless @@logger

a.rb(main):009:2**>** @@logger

a.rb(main):010:2**>** end

a.rb(main):011:1**>** end

**=>** :create

a.rb(main):012:0**>**

**=>** nil

a.rb(main):013:0**>** # Lấy id trực tiếp

**=>** nil

a.rb(main):014:0**>** Logger.create

**=>** #**<**Logger:0x000000000336eea0**>**

a.rb(main):015:0**>** # Thử gọi lần 2 xem có khác biệt hay không

**=>** nil

a.rb(main):016:0**>** Logger.create

**=>** #**<**Logger:0x000000000336eea0**>**

a.rb(main):017:0**>**

Cả 2 đều có cùng địa chỉ trên bộ nhớ. Như vậy mẫu Singleton đã hoạt động đúng.

# Tự đánh giá

|  |  |  |  |
| --- | --- | --- | --- |
| **Yêu cầu** | **Mô tả các kết quả đạt được** | **Thang điểm** | **Điểm tự đánh giá** |
| Báo cáo | Hoàn thành 100% | 10 | 10 |
| Slide | Hoàn thành 100% | 10 | 10 |
| Demo | Hoàn thành 100% | 10 | 10 |
|  | Tổng | 10 | 10 |