INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MATO GROSSO
DEPARTAMENTO DA ÁREA DE INFORMÁTICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

MURILLO HENRIQUE SILVA SOARES

INTRODUÇÃO A LINGUAGEM DE PROGRAMAÇÃO RUBY

CUIABÁ
2016

MURILLO HENRIQUE SILVA SOARES

INTRODUÇÃO A LINGUAGEM DE PROGRAMAÇÃO RUBY

Trabalho de linguagem de programação Ruby apresentado para avaliação da Disciplina Linguagem de Programação I do 5º semestre do Curso de Engenharia de Computação sob orientação do Prof. João Paulo.

Área de pesquisa: Linguagem de programação

Cuiabá
2016

“A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original”.

Albert Einstein

CAPÍTULO 1

RESUMO

Este artigo trata de um resumo das características principais da linguagem Ruby, desde a sua arquitetura até a sua sintaxe. Abordando principalmente temas relativos ao contexto de Linguagens de Programação é possível avaliar uma linguagem de forma correta. Este artigo tenta fazer isto de uma forma neutra, avaliando cada ponto da linguagem Ruby.

Keywords: Linguagem Ruby, Ruby, Linguagens de Scripts

Sumário

CAPÍTULO 2

INTRODUÇÃO

Ruby é uma Linguagem de programação interpretada, com tipagem dinâmica e forte, orientada a objetos e, com várias semelhanças com Perl, Python e SmallTalk.

Projetada tanto para a programação em grande escala quanto para codificação rápida, tem um suporte a orientação a objetos simples e prático. A linguagem foi criada pelo japonês Yukihiro Matsumoto, que aproveitou as melhores idéias das outras linguagens da época.

O nome "Ruby", foi decidido durante uma sessão de bate-papo online entre Matsumoto (Matz) e Keiju Ishitsuka em 24 de fevereiro de 1993, antes que qualquer linha de código tivesse sido escrita para a linguagem. Inicialmente foram propostos dois nomes: "Coral"e "Ruby", sendo esse último nome proposto escolhido mais tarde por Matz em um e-mail para Ishitsuka. Matsumoto explicou mais tarde que o motivo de ter escolhido o nome "Ruby"foi porque essa era a pedra zodiacal de um de seus colegas.

2.1 Motivação e objetivos

Pode ser utilizada tanto em scripts mais simples quanto em grandes sistemas. O Ruby é fascinante para os programadores pela sua simplicidade, robustez e pela maneira intuitiva com a qual o programador interage com a linguagem.

Foi criado para desenvolver uma linguagem que enfatiza as necessidades humanas ao invés das necessidades do computador, o que é o motivo de Ruby ser tão fácil de compreender.

2.2 Crescimento

Desde que foi tornado público em 1995, o Ruby arrastou consigo programadores devotos em todo o mundo. Tendo formação de grupos de utilizadores em todas as principais cidades mundiais e as conferências sobre Ruby tiveram lotação esgotada.

O índice TIOBE, que mede o crescimento das linguagens de programação, posiciona o Ruby em 9º entre as linguagens de programação.

O Ruby é também totalmente livre. Não somente livre de custos, mas também livre para utilizar, copiar, modificar e distribuir.

2.3 Aplicabilidade

Algumas aplicações da linguagem Ruby são elaboração de scripts, aplicativos para desktop, aplicativos para web e serviços web, entre outros.

Ruby é uma linguagem interpretada. A sintaxe é enxuta, quase não havendo necessidade de colchetes e outros caracteres. Toda as variáveis são objetos, onde até os "tipos primitivos"(tais como inteiro, real, entre outros) são classes. Estão disponíveis diversos métodos de geração de código em tempo real, como os "attribute acessors". Através do Ruby Gems é possível instalar e atualizar bibliotecas com uma linha de comando, de maneira similiar ao APT do Debian Linux. Code blocks (bloco de código), ajudam o programador a passar um trecho de instruções para um método. A idéia é semelhante aos "callbacks"do Java, mas de uma forma extremamente simples e bem implementada. Mixins, uma forma de emular a herança múltipla, sem cair nos seus problemas. Tipagem dinâmica, mas forte, o que significa que todas as variáveis devem ter um tipo (fazer parte de uma classe), mas a classe pode ser alterada dinâmicamente.

Ruby on Rails é um framework de desenvolvimento web (gratuito de código aberto) otimizado para a produtividade sustentável e a diversão do programador. Ele permite que você escreva código de forma elegante. Favorecendo a convenção ao invés da configuração.

CAPÍTULO 3

CARACTERÍSTICA DA LINGUAGEM

Vamos conhecer agora alguns dos recursos, características, tipos e estruturas básicas da linguagem.

3.1 Tipos de Dados

3.1.1 Números

Ruby divide os números em dois tipos: fixnum (inteiros) e bignum(float). Os fixnum podem ser números na faixa de 2^{-62} até 2^{+62}. Qualquer número fora desta faixa é atribuído à classe bignum. Devido à grande faixa de números atribuídos à classe bignum, sua implementação é feita através de uma seqüência de bits dinâmica, permitindo que cálculos com números extremamente grandes possam ser feitos, como por exemplo fatorial(400) (desde que haja memória suficiente para armazenar o número). Para armazenar um número, são gastos 2 longs + 1 int bytes.

3.1.2 Strings

Strings são seqüências de bytes armazenados dinamicamente sem um limite fixo pré-determinado (somente o limite físico de memória). São gastos 4 long + 1 pointer + tam da string * 1 char para armazenar uma string.

3.1.3 Arrays

Arrays, em Ruby, são conjuntos de objetos (não necessariamente do mesmo tipo) que podem ser acessados através de um índice inteiro, conforme a inserção no array. Devido à heterogeneidade dos arrays, não existem "structs"em Ruby, somente arrays que podem conter elementos de qualquer tipo. São gastos 4 long + 1 pointer + tam do

array * 1 pointer para armazenar um array.

3.1.4 Hashes

Ruby traz uma implementação nativa de tabelas hash, assim como em Perl ou Java. Você pode construir tabelas hash para acelerar o acesso aos seus dados, através da especificação de valores/chaves para os seus dados. Existem algumas restrições que a linguagem faz na construção de tabelas hash. A primeira é que após construída a tabela os valores das chaves não podem ser alterados. Outra restrição é que as chaves sejam todas do mesmo tipo.

3.2 Orientação a Objetos

Características básicas de linguagem orientadas à objetos como o conceito de herança, classes, métodos, polimorfismo e encapsulamento são implementadas em sua totalidade na linguagem Ruby.

Somente herenças simples podem ser feitas na linguagem Ruby (apesar disto, existe um conceito de compartilhamento de métodos entre classes, permitindo desta maneira uma herença múltipla forçada). Utilizando Mixins, por meio de Modulos (Modules), que são classes de classes, Ruby consegue emular a herança multipla.

Os modificadores de acesso é feito através de três construções básicas: private, protected e public. Um método private somente pode ser chamado na sua classe de definição. Um método protected pode ser chamado somente pela sua classe de definição e suas subclasses. Um método public é acessado por qualquer método. Algo que deve ser mencionado aqui é que estas verificações são feitas dinamicamente em tempo de execução. Isto leva programadores menos experientes à run-time errors mais freqüentemente.

3.2.1 Variáveis

Variáveis são somente referências para objetos. Nada mais. Não existem ponteiros. Apesar desta lacuna, aliases de objetos podem ser feitos, como em Java.

As variáveis na linguagem Ruby não possuem tipo, assim como em Smalltalk, BASIC Ou Python. Isto significa que conforme as variáveis são utilizadas no programa

elas vão tendo o seu tipo atribuído. No contexto de Linguagens de Programação isto é chamado de atribuição de tipos dinâmica.

As variáveis não precisam ser declaradas antes de serem utilizadas. Conforme o nome da variável o interpretador sabe qual o escopo da variável utilizada. No contexto de linguagens de programação, estas variáveis são chamadas de variáveis heap-dinâmicas implícitas.

A atribuição de valores à variáveis é feita através de referências à objetos. Isto significa que ao atribuir um valor (lembre-se de que um valor também é um objeto na linguagem Ruby) à um objeto, este valor não precisa ser copiado e instanciado novamente, somente uma referência para o valor que está sendo atribuído é criada.

Uma variável local é declarada normalmente. Uma variável de instância é declarada com um "@"no nome. Uma variável de classe é declarada com "@@", e uma variável global é declarada com cifrão. Variáveis que iniciam com uma letra maiúscula são consideradas constantes.

3.2.2 Métodos

Métodos são os subprogramas das linguagens orientadas à objetos. Seus parâmetros formais são chamados de protocolos, mas a forma de implementação não difere muito entre os dois paradigmas.

Na linguagem Ruby, a procura por um método chamado para ser executado é realizada de forma dinâmica. Isto significa que quando um método não é encontrado em sua classe atual, o próximo lugar a ser procurado será a sua classe pai, e assim sucessivamente, até chegarmos na superclasse.

Os métodos conseguem retornar somente um objeto, não importando que tipo de objeto. Isto significa que um método pode retornar qualquer coisa, devido à capacidade dos arrays serem heterogêneos.

3.2.3 Classe e Objetos

Para criarmos uma classe, usamos a palavra-chave class, seguida pelo nome da classe. Segundo as convenções de Ruby, nos nomes dos métodos deve-se usar letras minúsculas separando as palavras com um sublinhado, porém nos nomes das classes

é utilizado camel case, da mesma maneira que em Java, com maiúsculas separando duas ou mais palavras no nome da classe. Temos então classes com nomes como MinhaClasse, MeuTeste, CarroPersonalizado. As propriedades do nosso objeto são armazenadas no que chamamos variáveis de instância, que são quaisquer variáveis dentro do objeto cujo nome se inicia com @. Se fizermos referência para alguma que ainda não foi criada, ela será. Podemos inicializar várias dessas variáveis dentro do método initialize, que é o construtor do nosso objeto, chamado após o método new, que aloca espaço na memória para o objeto sendo criado. Não temos métodos destrutores em Ruby, mas podemos associar uma Proc para ser chamada em uma instância de objeto cada vez que ela for limpa pelo garbage collector.

3.2.4 Gerenciamento da Memória

O gerenciamento de memória do Ruby é todo feito automaticamente pelo próprio interpretador. Periodicamente, a linguagem Ruby aloca pequenos blocos de memória para que nunca falte memória para um script que esteja rodando, automatizando assim o processo de alocacão de memória. Um garbage collector encarrega-se de todos objetos que não são referenciados à bastante tempo, facilitando assim a vida do programador, que não precisa desalocar blocos de memórias para objetos.

Apesar disto, instruções específicas de alocação/desalocação implementadas nativamente possibilitam ao programador gerenciar casos específicos aonde o gerenciamento automático não se aplica. Uma grande vantagem do garbage collector é que desaparecem os memory leak, acontecem poucos erros ou travamentos. Isto torna a programação mais rápida e menos complicada, porque não é necessário fazer o gerenciamento de memória manual.

3.2.5 Tratamento de Exceções

Exceções nos permitem “cercar” erros que acontecem no nosso programa, em um objeto que depois pode ser analisado e tomadas as devidas providências ao invés de deixar o erro explodir dentro do nosso código levando à resultados indesejados. As palavras-chave para isto são "begin", "rescue"e "ensure". "Begin"inicia um trecho que pode cair em alguma exceção (opcional), "Rescue"determina o comportamento em caso de uma exceção específica ou não e, "Ensure"é o código que será executado independente de ter havido exceção ou não.

CAPÍTULO 4

A SINTAXE DA LINGUAGEM RUBY

4.1 Primeiros Passos

O propósito geral deste artigo é dar uma noção geral sobre a linguagem Ruby, e como não poderia deixar de ser, devemos também abordar como é a sintaxe da linguagem Ruby. Neste mini-tutorial sobre a linguagem já dá pra ter uma noção de como é a sintaxe e o quão fácil é programar em Ruby. Vamos começar com um simples exemplo. Vamos escrever um método que retorna uma string, adicionando à esta string uma outra string passada como parâmetro.

```
def sayHello(name)
result = "Hello, "+ name
return result
end
puts sayGoodnight("World")
```

Antes de tudo, vamos fazer algumas observações. Primeiro, a sintaxe do Ruby é muito simples. Ponto e vírgula como terminadores não são necessários. Comentários são linhas começadas por hashtags. Métodos são definidos pela palavra reservada def, seguido pelo nome do método e seus parâmetros entre parênteses. O método do exemplo acima é simples e é totalmente compreensível para alguém que possui conhecimento em qualquer linguagem de programação estruturada. Vale observar que não é necessário declarar variáveis. Métodos não podem ser declarados fora de classes, como no exemplo acima. Entretanto, por motivos de brevidade, não declarei nenhuma classe. Além disto, você pode chamar o método sayHello de diversas formas, como mostrado abaixo:

```
puts sayHello "World"
puts sayHello("World")
```

```
puts(sayHello "World")
puts(sayHello("World"))
```

Este exemplo também mostra como criar um objeto da classe String. Existem muitas maneiras de criar um objeto String, mas provavelmente a mais simples é utilizar uma palavra entre aspas atribuída ao objeto que desejamos criar. No Ruby, todas aquelas seqüências de caracteres bem conhecidas dos programadores C/C++ para quebra de linhas e tabulação são válidas.

4.2 Arrays e Hashes

Arrays e Hashes no Ruby são coleções indexadas de objetos. Ambos armazenam objetos utilizando uma chave como acesso. Nos arrays, as chaves são números inteiros, como em qualquer outra linguagem. Hashes suportam qualquer objeto como chave. Ambos aumentam de tamanho conforme necessário. Uma observação interessante é que tanto arrays quanto hashes podem guardar qualquer tipo de objeto (inclusive de diferentes tipos).

Você pode criar e inicializar arrays declarando-os literalmente - um conjunto de elementos entre chaves, por exemplo. Com o array criado, você pode acessar seus elementos através de índices, como em qualquer outra linguagem, como no exemplo abaixo:

```
a = [ 1, 'cat', 3.14 ]
a[0] » 1
a[2] = nil
a » [1, "cat", nil]
```

Hashes são muito parecidas com arrays. Uma declaração literal utiliza chaves ao invés de colchetes, conforme ilustrado abaixo:

```
instSection =
'cello' => 'string',
'clarinet' => 'woodwind',
'drum' => 'percussion',
```

```
'oboe' => 'woodwind',
'trumpet' => 'brass',
'violin' => 'string'
```

Para ter acesso aos elementos, utilize a chave declarada:

```
instSection['oboe'] » "woodwind"
instSection['cello'] » "string"
instSection['bassoon'] » nil
```

Como no último exemplo, uma hash retorna nil quando uma chave indexada não é encontrada. Normalmente esta solução não causa muito problema, mas podemos também mudar este valor default. Isto é facilmente feito através do construtor da classe Hash, conforme ilustrado abaixo:

```
histogram = Hash.new(0)
histogram['key1'] » 0
histogram['key1'] = histogram['key1'] + 1
histogram['key1'] » 1
```

4.3 Estruturas de Controle

A linguagem Ruby possui todas as estruturas de controle padrões das linguagens populares hoje em dia, como ifs e whiles. A única diferença é que o Ruby não usa chaves e sim a palavra reservada end para acabar uma estrutura. Por exemplo:

```
if count > 10
puts "Tente novamente"
elsif tries == 3
puts "Você perdeu"
else
puts "Digite um número"
end
```

Da mesma maneira, um while termina com um end, como no exemplo abaixo:

```
while weight < 100 and numPallets <= 30
pallet = nextPallet()
weight += pallet.weight
numPallets += 1
end
```

Para facilitar a legibilidade do código (ou não) existem as construções podem ser feitas de maneiras um pouco diferentes. Veja no exemplo abaixo o exemplo de um if simples:

```
if radiation > 3000
puts "Danger, Will Robinson"
end
puts "Danger, Will Robinson"if radiation > 3000
```

Da mesma forma, um while poderia ser feito da seguinte forma:

```
while square < 1000
square = square*square
end
square = square*square while square < 1000
```

4.4 Algumas Comparações

Para o mesmo problema, Ruby é a linguagem que possui menos linhas de códigos.

Problema: Imprimir somente pares entre 1 e um número qualquer

```c
#include <stdio.h>
void imprime_pares(int limite)
{
    int i;
    for(i = 1; i < = limite; i++)
        if(i%2 == 0)
            printf("%d\n", i);
}
```

Figura 1: Linguagem C

```java
public void imprime_pares(int limite) {
    for(int i = 1; i <= limite; i++)
        if(i%2 == 0)
            System.out.println(i);
}
```

Figura 2: Linguagem Java

```ruby
def imprime_pares(limite)
    1.upto(limite) {|i| puts i if i%2 == 0}
end
```

Figura 3: Linguagem Ruby

CAPÍTULO 5

CONCLUSÃO

Neste trabalho foi trazida a proposta de introdução a linguagem de programação Ruby, descrevendo suas características e o seu suporte a orientação a objetos simples e prático. Foi notado que a linguagem é expressiva, ou seja, diga muito, seja claro e escreva menos código.

Uma pesquisa válida para trabalhos futuros é a implementação da linguagem de programação Ruby para teste de tempo de execução versus tempo de copilação de outras linguagens orientada objetos.

CAPÍTULO 6

REFERÊNCIAS

[1]Matz, falando na lista de e-mails Ruby-Talk, 12 Mai. 2000.

[2]Matz, em An Interview with the Creator of Ruby, 29 Nov. 2001.

[3]Matz, em Blocks and Closures in Ruby, 22 December 2003.

[4] BAGLEI, D., The Great Language Computer Shootout (http://www.bagley.org/~doug/shootc

[5] MATSUMOTO, Y., Programming Ruby - The Pragmatic Programmers Guide Segunda Edicão, Ad
LongMan Inc., 2001

[6] Joshua, D. D., Programming in the Ruby Language, IBM DeveloperWorks (http://www-
106.ibm.com/developerworks/linux/library/l-ruby1.html), 2001

[7] MATSUMOTO, Y., The Ruby FAQ (http://www.ruby-lang.org/en/), 2001