Estudo sobre a Linguagem de Programação Ruby

Giovanna Monteiro de Azevedo e Ibukun Chife Didier Adjitche

[1]Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil

giovanna_monteiro@live.com, adjitchedidier1@gmail.com

Resumo. O presente trabalho trata do aprendizado de uma nova linguagem de programação, Ruby, utilizando-a para implementar um projeto. O objetivo escolhido para o projeto foi realizar o levantamento de dados para obter a quantidade de veículos, preco, distância e tempo de viagem do centro das capitais brasileiras até os seus respectivos aeroportos. Para tanto, necessitou-se efetuar uma conexão com API do Uber a fim de obter os parâmetros de entrada para a manipulação e desenvolvimento do projeto em questão.

1. História do surgimento da linguagem

De uma linguagem basicamente desconhecida em 2004 a uma das linguagens mais utilizadas atualmente, o Ruby percorreu um rápido e espetacular caminho. Criada por Yukihiro "Matz" Matsumoto em 1993, a linguagem cresceu rapidamente no Japão, terra de seu inventor, mas nunca chegou a ter reconhecimento internacional até que um framework novo, chamado Rails, conquistasse o mundo do desenvolvimento Web em meados de 2003.

Chamado simplesmente de Ruby on Rails, esse novo framework fazia um uso inovador do Ruby e de suas características de meta-programação para encaixar as peças do desenvolvimento Web em um todo coerente que poucas vezes surgira no mercado. Desnecessário dizer, o Rails se tornou um sucesso instantâneo trazendo o Ruby para a frente das câmeras e mostrando que existiam outras linguagens, além das tradicionais, que podiam oferecer muito ao programador interessado.

No Japão, o Ruby já era, inclusive, mais popular do que o Python. Sua brilhante aplicação no Rails lhe deu um reconhecimento maior e o tornou a ferramenta de escolha de milhões de programadores, não só para desenvolvimento Web como basicamente para qualquer tarefa que exija uma linguagem dinâmica e produtiva.

O sucesso atual do Ruby se deve, sem dúvida ao Rails. Sem o Rails, é provável que o Ruby tivesse surgido eventualmente no cenário central das linguagens de programação, mas sem todo o hype e visibilidade que o mesmo possui atualmente. O que teria sido uma pena já que o Ruby, como mencionado acima, é uma ferramenta fantástica não só para desenvolvimento Web mas também para a miríade de tarefas que um programador precisa executar do seu dia-a-dia.

Um dos motivos da criação do Ruby, segundo o seu inventor, foi o foco no programador e na necessidade de maior produtividade com menor stress. De acordo com Matz, as linguagens de programação são focadas em extrair o máximo de performance da máquina na qual estão rodando, enquanto o Ruby privilegia o máximo de performance para o programador.

Tudo isso dito, não se pode extrair a conclusão de que o Ruby seja a linguagem perfeita. Como qualquer linguagem, o Ruby tem suas vantagens e desvantagens e deve ser usada apropriadamente.

2. Domínios de aplicação

Aplicações Web:

- BaseCamp: é um software com plataforma Web, que gerencia os projetos que podem ser de diversas áreas. Com o objetivo de diminuir a enorme quantidade de reuniões e otimizar o desenvolvimento dos trabalhos.

Simulação
- NASA Langley Research Center
Negócio:
- Toronto Rehab : universidade
Robótica
- MORPH : grupo de Inteligência em Automação
Redes
- Open Domain Server
Telefônica
- Lucent : empresa ligada ao Alcatel
Administração de Sistemas
- Level 3 Communications: empresa especialisada em Ti

Plataformas:
- Mac OS x
- Linux
- MS-DOS
- BSDs(inclui o freeBSD e o OpenBSD)
- Acorn RISC OS
- Microsoft
- Celulares Symbian Series 60
- Plataformas que contém uma maquina virtual Java (Usando Jruby)

3. Paradigmas suportados pela linguagem

Ruby é uma linguagem Multi-paradigma:

•**Orientada Objetos**

```
class Banco

  def initialize(contas)
    @contas = contas
  end

  def status
    saldo = 0
    for conta in @contas
      saldo += conta
    end
    saldo
```

```
  end
end

banco = Banco.new([200, 300, 400])
banco.status
```

•**Funcional**

```
>> ens.to_s
=> "{ 1, 5, 3, 2 }"
?> ens.map { |x| x * 10 }
=> [10, 50, 30, 20]
>> ens.reject { |x| x.even? }
=> [1, 5, 3]
>> ens.find { |x| x >= 2 }
=> 5
```

•**Programação Imperativa**

```
int soma = 0;
for(int i=0 ; i< array.length ; i++){
      soma += array[i];
}
return soma;
```

4. Variáveis e tipos de dados

Uma variável é um local de armazenamento para um valor. É chamado de variável porque as informações armazenadas nesse local podem ser alteradas quando o programa está sendo executado. Para atribuir um valor a uma variável, use o sinal de igual, também chamado de operador de atribuição. Por exemplo:

```
x = 8
```

Nomes de variáveis podem consistir em caracteres alfanuméricos e o caractere de sublinhado (_), mas não pode começar com uma letra maiúscula. Todavia, também temos as Palavras Reservadas que são aquelas que não podem ser utilizadas como nomes de classes, variáveis ou qualquer outra coisa. As palavras reservadas em Ruby são:

BEGIN	def	for	redo	undef
END	defined	if	rescue	unless
alias	do	in	retry	until
and	else	module	return	when

begin	elsif	next	self	while
break	end	nil	super	yield
case	ensure	not	then	
class	false	or	true	

Variáveis que começam com uma letra maiúscula são chamadas **Constantes**. O valor de uma variável constante não pode ser mudado uma vez que tenha sido atribuído. Por exemplo:

```
MyNum = 42
#Tentar alterar o valor resulta em um aviso

MyNum = 8
#Warning: already initialized constant MyNum
```

Tipos de Dados

Ruby não possui tipos primitivos, mas sim todos tipos são classes, assim como todas variáveis são objetos. Como exemplo, conjunto de caracteres é uma instância da classe String, inteiro é da Fixnum e matriz é Array. Ruby determina automaticamente o tipo de dados pelo valor atribuído à variável. O desenvolvedor pode reatribuir um valor diferente a uma variável a qualquer momento. Para inserir o valor de uma variável em uma sequência de citações duplas (uma string é uma sequência de caracteres, como "olá"), use o símbolo # e os suportes {} com o nome da variável. Por exemplo:

```
age = 22
puts "Ele tem #{33} anos de idade"
```

1. **Integer ou Inteiro**: A classe Integer representa valores numéricos inteiros. Essa é uma classe abstrata, e não é possível instanciar objetos com ela (ausência o método new). Essa classe serve apenas como modelo para as classes que herdam ou estendem dela. As suas subclasses compartilharão os mesmos atributos e comportamentos dela, e são elas que realmente são usadas para instanciar objetos. É dividida em duas subclasses: Fixnum e Bignum, onde a primeira representa números inteiros, é importante ressaltar que possui um limite de valor que ela pode armazenar, mas isso depende da arquitetura do computador. Já a classe Bignum, que também é subclasse de Integer, armazena valores inteiros maiores que Fixnum. O limite dos valores armazenados depende nesse caso da memória disponível. Se os valores designados a uma variável do tipo Fixnum ultrapassarem o limite que o tipo suporta, o Ruby automaticamente converterá a variável para o tipo Bignum.

2. **Float:** A classe Float deriva seu comportamento da classe Numeric e é usada para representar valores de ponto flutuante, ou seja, números com valores fracionados, independentemente do tamanho desse número. Para separar a parte inteira da fracionário usa-se o ponto e não virgula.

3. **String:** As String's armazenam cadeias ou conjuntos de caracteres, e são usadas constantemente para representar textos na aplicação. Nos exemplos apresentados utilizamos objetos String's para retornar uma mensagem para o usuário. Em Ruby, as String's vêm em dois tipos: as delimitadas por aspas simples e as delimitadas por aspas duplas. O conteúdo das que são declaradas entre aspas simples é interpretado literalmente, já as delimitadas por aspas duplas podem conter expressões que inferem para outras coisas.

4. **Arrays:** Um array é uma estrutura de dados para guardar uma coleção de objetos, que podem ser até de tipos diferentes. Cada elemento está associado a um índice que inicia em 0, e pode ser usado para recuperar o elemento. Os elementos do array em Ruby são delimitados por colchetes e separados por (virgula). A sintaxe de declaração de um array segue o seguinte:

5. **Hash:** O hash é uma estrutura semelhante ao array, pois também armazena uma coleção de informações ou elementos. Mas diferentemente do array, os elementos do hash não estão associados a um índice e sim a um objeto qualquer que funciona como uma chave. Os seus elementos são delimitados por "{}" (chaves).

6. **Symbol:** Os objetos desse tipo se assemelham com Strging's, mas não são definidos entre aspas, eles são, na verdade, definidos com um sinal de dois pontos. Dois símbolos idênticos apontam para o mesmo endereço de memória, ou seja, são únicos, justamente por esse motivo os desenvolvedores constantemente os usam como chaves em objetos hash's.

7. **Range:** Um Range (série) representa um intervalo de objetos. É possível instanciar um Range usando os operadores ".." , "..." ou o método new. Os Range's podem ser convertidos para array's com a chamada ao método to_a.

```
Exemplos:
x=42 #integer
y = 1.58 #floating point value
z = "Hello" #string
w = ['elemento1', 'elemento2', 'elemento3'] #array
t = {'objeto_chave1' => 23, 'objeto_chave2' => 30} #hash
puts :simbolo #symbol
a = (1..7).to_a
puts a #[1,2,3,4,5,6,7] #range
```

5. Comandos de controle

As estruturas condicionais são estruturas que mudam pouco entre as linguagens de programação. Para utilizar estruturas de controle em Ruby, precisamos antes conhecer os operadores booleanos, true e false. Os operadores booleanos são: ==, >, <, >= e <=. Expressões booleanas podem ainda ser combinadas com os operadores && (and) e || (or).

1. **If:** O if do ruby aceita qualquer expressão booleana, no entanto, cada objeto em Ruby possui um "valor booleano". Os únicos objetos de valor booleano false são o próprio false e o nil. Portanto, qualquer valor pode ser usado como argumento do if:

```
num = 8
if num == 3
     puts "Número é igual a 3"
elseif num ==10
     puts "Número é igual a 10"
elsif num == 7
     puts "Número é igual a7"
else
     puts "Não encontrado"
end
```

2. **Unless:** a expressão unless é o oposto da expressão if. E executa o código quando a condição é falsa.

```
 unless i >= 10
     puts "menor que 10"
 else
     puts "maior igual que 10"
 end
```

3. **Case:** é a opção mais simplificada e flexível, pq testa o valor nas afirmações de when.

```
case i
when 0..5
     puts " Esta entre 0 e 5"
when 6..10
     puts " Esta entre 6 e 10"
else
```

```
    puts i.to_s
end
```

4. **While**: os loops são usados para executar no mesmo bloco de código um específico número de vezes. O loop While executa um bloco de código quando a condição for verdadeira.

```
a = 0
while a < 10
    puts a
    a+=1
end
```

5. **Until**: O loop until é o oposto de um loop while, ou seja, ele roda quando a condição é falsa.

```
a = 0
until a > 10
    puts "a = #{a}"
    a+=2
end
```

6. **For**: O loop de for consiste em uma variável vazia e um range. Para cada interação do loop, a variável vazia recebe um valor correspondente ao elemento do range.

```
for a in (1..6)
    puts a
end
```

7. **Loop Do**: permite que o código execute até que a condição do break seja atendida.

```
X=0
looop do
    puts x
    x+=1
    break if x>10
end
```

6. Escopo (regras de visibilidade)

Ruby tem uma estrutura de bloco aninhada, ou seja:

— Declarações no nível mais externo têm escopo global (nível 1).

— Declarações dentro de um bloco interno são locais ao bloco; cada bloco está dentro de outro bloco, com um nível a mais.

O escopo (âmbito) de uma variável local em Ruby é um dos seguintes:

- proc{…}
- loop{…}
- def…end
- class…end
- module…end
- O programa inteiro (se não há nenhum dos itens anteriores)

Ruby implementa o escopo **estático** (as variáveis tem seu escopo determinado antes da execução do programa, ou seja, não é determinado durante o tempo de execução, como seria o escopo dinâmico). Mas em Ruby todas as variáveis podem ser local, global – $, atributos de objeto (de instância) – @ (a variável é acessível somente à classe) ou atributos de classe – @@ (a variável é acessível à classe e a todas que herdarem dela). O que pode confudir na hora de determinar qual escopo a linguagem tem. Por exemplo:

Código 1

```
$s = 2;
def scaled(d)
    d*$s;
end
def teste()
   s = 3;
   scaled(3);
end
puts teste();
```

Código 2

```
$s = 2;
def scaled(d)
d*$s;
end
def teste()
$s = 3;
scaled(3);
end
puts teste();
```

No código 1 será impresso 6 e no código 2 será impresso 9. Nesses códigos exemplificam duas diferentes formas de acesso a uma variável, mas o escopo continua

sendo estático. Não está relacionado com o TIPO de escopo, mas com o escopo em si (escopo = faixa de instruções no qual uma variável é visível). Ou seja, está relacionado com a forma de acesso a ambientes locais ou não locais.

Declarações

Uma declaração é uma frase de um programa que ao ser elaborada produz ocorrências e vínculos. Ruby suporta os principais tipos de declarações, com exceção de declarações colaterais.

Definição – é uma declaração cujo único efeito é produzir associações. Como exemplo, tem-se a declaração de constantes. Ex:

```
Constante = 10
```

Declaração de tipos - uma declaração de um novo tipo cria um tipo e produz um vínculo. Como em Ruby TUDO é objeto, pode-se definir um novo tipo através de classes (TAD): Ex:

```
classFoo
  @foo
end
```

Declaração de variáveis – uma declaração de variável nova cria uma variável e produz um vínculo. Como foi dito ateriormente, em Ruby, todas as variáveis podem ser local, global ou atributos de classe. Como Ruby é dinamicamente tipada, a declaração de uma variável é feita quando se associa um valor à mesma:

```
Ex: foo = Foo.new
```

Declaração sequencial – sintaxe semelhante aos comandos sequenciais:

```
Ex: @foo; @foo2 # declaracao da variavel @foo e da variavel @foo2
```

```
obs: o '#' significa início de comentário
```

Declaração recursiva – Uma declaração recursiva usa as próprias declarações que ela produz, pode ser, por exemplo, uma declaração de tipos recursivos ou definições de funções e procedimentos recursivos.

```
Ex:
def fib(n)

return 1 if n <= 1

return fib(n - 1) + fib(n - 2)

end
```

7. Exemplo prático de uso da linguagem de programação

Para este trabalho, foi desenvolvido um programa em Ruby que realiza o quantitativo de veículos do aplicativo Uber no Brasil e também foi executado o levantamento de percursos do centro de todas as capitais do país até os seus respectivos Aeroportos, contendo as variáveis de distância, tempo e valor da corrida.

Para o desenvolvimento do programa foi utilizado o paradigma imperativos, principalmente no que diz respeito da leitura e interpretação dos arquivos JSON fornecidos pelo API do Uber e também pela praticidade da execução dos comandos de controle (if e for).

Primeiramente, foi instalado o Ruby através da linha de comando do ambiente Linux (Ubuntu e Mac), possibilitando a criação vários scripts como extensão .rb e um ambiente (enviroment), este guardando os dados de permissão de acesso da API e os dados a servir de entrada para a execução dos scripts. Com o intuito de facilitar o desenvolvimento do projeto, utilizou-se os editores de texto Geany e SublimeText. Os testes e a execução foram executados na linha de comando através da seguinte entrada:

```
env $(cat .env3 | xargs) ruby nome_do_arquivo.rb;
```

O arquivo em ruby (ubertrackerv4.rb), juntamente com o ambiente utilizado (.env3) estão disponíveis no link do repositório:

https://github.com/IbukunChife/LP_2017_ProjetoFinal_IbkGio.git

8. Conclusões

Diante do que foi apresentado neste projeto, podemos concluir que Ruby é uma linguagem dinâmica, tipada, multiparadigmas e fácil de aprender, pois é bem intuitiva e oferece várias ferramentas com diversas funcionalidades, como o Ruby On Rails. Possui diversas vantagens, em especial a capacidade de migração e de comunicação com outras linguagens como C, HTML, PHP, SQL, CSS, entre outras. Apesar de ser uma linguagem muito parecida com Python, ainda fez-se necessário dispor certo de dedicação rigorosa à literatura para compreender desde os conceitos mais básicos aos mais complexos da linguagem escolhida e da troca de dados com a plataforma API em questão.

No decorrer da construção do projeto, encontraram-se restrições aos acessos do API, que embora o cadastro exigido no site dos desenvolvedores da Uber tenha sido preenchido da forma correta, não foi possível obter acesso ilimitado aos dados dos motoristas, passageiros e das experiências de viagens. Contudo, o projeto foi adaptado a

quantidade de dados oferecidos, resultando basicamente na manipulação dos preços, distâncias e tempo das corridas e das diferentes categorias de viagens.

Ainda mediante às dificuldades, um vez compreendido as limitações do API e das funcionalidades da linguagem Ruby, foi possível concluir o desenvolvimento do programa de forma satisfatória. Podendo este ser incrementado e, futuramente, melhorado quando trabalhado no Framework do Ruby On Rails como um site, um aplicativo mobile ou Web ou até mesmo servir de base para um projeto de pesquisa.

9. Referências

Caelum. Desenvolvimento Ágil para Web com Ruby on Rails 4. Disponível em: <www.caelum.com.br/apostilas/> Acesso em: 15 de julho 2017.

Damasceno, H. **Paradigma Linguagem Ruby**. Curso ciência da computação, Faculdade Anglo Americano. Disponível em: <https://fr.slideshare.net/heverson/ruby-apresentacao/> Acesso em: 17 de julho 2017.

Ligação e Escopo. Disponível em: <https://linguagemruby.wordpress.com/ligacao-e-escopo/> Acesso em 19 de julho 2017.

Menegotto, A. B. e Mierlo F. **A Linguagem Ruby**. UNISINOS - Universidade do Vale do Rio dos Sinos Centro de Ciências Exatas e Técnológicas São Leopoldo - RS – Brasil.

Ruby (linguagem de programação). Disponível em: <https://pt.wikipedia.org/wiki/Ruby_(linguagem_de_programa%C3%A7%C3%A3o) > Acesso em: 17 de julho 2017.

Ruby doc. Disponível em: <http://ruby-doc.org/> Acesso em: 15 de julho 2017.

Site Oficial do Ruby. Disponível em: <https://www.ruby-lang.org/pt/> Acesso em: 15 de julho 2017.

Tipos de dados em Ruby. Disponível em:<http://www.devmedia.com.br/tipos-de-dados-em-ruby/33600>Acesso em: 15 de julho 2017.