PROGRAMAÇÃO BÁSICA

GUILHERME ALMEIDA - NACCIB

FÁBRICA DE NOOBS

6 INTRODUÇÃO AO RUBY - PARTE 1

Introdução

Ruby é uma linguagem de programação **dinâmica**, **orientada à objeto** e **intepretada**.
Eu, Guilherme Almeida a.k.a naccib, fiz uma apostila, com apoio do Natanael, com uma introdução sobre essa belíssima linguagem.
Fora Temer.

Obtenção

Windows

Para baixar Ruby no Windows você precisará ir ir nesse site e baixar o *RubyInstaller*, é recomendado que você baixe o *RubyInstaller* 2.3.3 (32-bit), pois muitas bibliotecas são melhores implementadas na versão de 32 bits do Windows.

OSX

No OSX, basta rodar esses comandos para instalar a versão mais nova do Ruby:

```
brew install rbenv ruby-build

# Add rbenv to bash so that it loads every time you open
a terminal
echo 'if which rbenv > /dev/null; then eval "$(rbenv init
```

```
 -)"; fi' >> ~/.bash_profile
source ~/.bash_profile

# Install Ruby
rbenv install 2.3.3
rbenv global 2.3.3
```

Linux

Existe uma forma de instalar Ruby em *qualquer* distro do Linux, essa é rodando os seguintes comandos:

```
cd
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
exec $SHELL

git clone https://github.com/rbenv/ruby-build.git ~/.rben
v/plugins/ruby-build
echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$P
ATH"' >> ~/.bashrc
exec $SHELL

rbenv install 2.3.3
rbenv global 2.3.3
```

Executando um script em Ruby

Para excutar um script em Ruby através da linha de comando nós apenas temos que digitar o seguinte:

```
ruby nome_do_arquivo.rb
```

Onde `nome_do_arquivo` é o nome do arquivo com o código para ser executado.
Ou...
Nós podemos executar Ruby interativamente pelo comando `irb`.

Testando

Agora para testar se o Ruby funciona, basta rodar

```
ruby -v
```

A saída vai ser algo como `ruby 2.3.3p222 (2016-11-21 revision 56859) [x86_64-linux]`.

Agora digite `irb` e vamos começar o tutorial!

Variáveis

Em Ruby, não é necessário declarar variáveis antes de seu uso, nem

definir seu tipo — isso já é feito automaticamente.
As variáveis podem ser de tipos diferentes, os mais básicos são:

- *String* — Todo texto em Ruby é uma String, elas são delimitadas por aspas simples `''` ou duplas `""`
- *Integer* — Todo número inteiro em Ruby é uma Integer, como 5, 9 ou 2017
- *Float* — Todo número decimal em Ruby é um Float, como 5.2, 9.321, 2017.0

O sintaxe de declaração de variável em Ruby é bem simples e segue o seguinte formato: `nome_da_variavel = valor_da_variavel`.
O nome da variável tem algumas restrições, ela não pode começar com um número nem podem conter caractéres especiais, tais como o espaço.
Por exemplo, se nós quisermos declarar uma variável chamda `a` com o valor de `5`, fazemos:

```
a = 5
```

Se nós quisermos declarar uma variável `b` com o resultado de uma operação matemática, digamos que `5 × 9`, fazemos:

```
b = 5 * 9
```

O valor de `b` será 45.

Nós podemos declarar números decimais da mesma forma, lembre-se que em Ruby nós usamos `.` para separar a parte inteira da decimal, não `,` (como se usa aqui no Brasil).

```
c = 5.5
```

É possível também declarar variáveis baseadas em *outras* variáveis, por exemplo:

```
d = c + 1.5
```

O valor de `d` será 7.

> *Ah, `#` define comentários em Ruby, isso quer dizer que tudo que vem depois de um `#` é ignorado.*
> *Isso é útil para comentar nosso código.*

Strings

Agora vamos falar das strings.
Uma `String` é uma variável que guarda um texto como valor, esse texto é delimitado por áspas simples `''` ou aspas duplas `""`, da seguinte maneira:

```
pessoa = 'naccib'
```

Nesse caso, uma variável de nome `pessoa` tem o valor `naccib`, simples e fácil.
Nós podemos juntar strings usando o operador `+` da seguinte forma:

```
frase = 'A vaca atravessou a rua e ' + pessoa + ' viu.'
```

O valor guardado na variável `frase` é `A vaca atravessou a rua e naccib viu.`
Porém, existe uma maneira mais fácil juntar strings em Ruby, se chama interpolação de strings. O nome pode parecer complicado, mas é *muito* simples, veja um exemplo:

```
idade = 21
nome = 'Natanael Antônio-Ollie'
frase = "#{nome} já tem #{idade} anos, eu tô c h o c a d a."
```

A variável `frase` tem o valor de `Natanael Antônio-Ollie já tem 21 anos, eu tô c h o c a d a`.
Isso ocorre porque `#{nome_da_variavel}` dentro de uma string delimitada por aspas duplas (`""`) vai ser substituido pelo **valor** da variável.
Isso facilita *muito* na hora de formatar strings.

> *Mas, eai, como eu faço para escrever o valor das variáveis que eu tenho na tela?*

Bem nós usamos **métodos** (vou falar mais deles depois) para escrever texto na tela.
Uma dessas funções é a `puts`, que é uma abreviação para o nome `put string` (botar string), veja:

```
a = 61
puts a
```

Vai escrever `61` na tela.
Você pode jogar qualquer tipo de variável nessa função, seja ela uma float, integer ou string, vejamos um exemplo mais complexo:

```
criador = 'Natanael'
programador = 'naccib'
frase = "Esse curso foi desenvolvido por:\n#{criador} e #{programador}"

puts frase
```

Esse código escreve o seguinte na tela:

```
Esse curso foi desenvolvido por:
Natanael e naccib
```

Você pode ter estranhado o `\n`, em programação isso significa quebra de linha, ou seja, ele separa as linhas.

Existem *vários* métodos para manipulas strings em Ruby, tais como

- `upcase` — Retorna a `String` em caixa alta
- `downcase` — Retorna a `String` em caixa baixa
- `length` — Retorna o tamanho (em caractéres) da `String`.

```
frase = 'Se o boy sorrir, é mandioca no Edir.'

puts frase.upcase # Escreve a frase para caixa alta
puts frase.downcase # Escreve a frase em caixa baixa
puts frase.length # Escreve o tamanho da frase (em caract
eres)
```

Operações Matemáticas

A linguagem Ruby possui uma poderosa e rápida biblioteca matemática, isso quer dizer que você consegue escrever código rápido e simples para fazer várias operações.

Operadores Básicos

Existem quatro operadores básicos: `+`, `-`, `*` e `/`.
O `+` serve para somar dois números, por exemplo:

```
2 + 9     # Isso é igual a 11
9.2 + 1.0 # Isso é igual a 10.2
```

O `-` serve para diminuir dois números, por exemplo:

```
60 - 20     # Isso é igual a 40
50.5 - 10.5 # Isso é igual a 40 também
```

O `*` serve para multiplicar dois números, por exemplo:

```
70 * 2     # Isso é igual a 140
90.5 * 8.2 # Isso é igual a 742.0999999999999
```

O `/` serve para dividir dois números, por exemplo:

```
-60 / 2   # Isso é igual a -30
140.6 / 2 # Isso é igual a 70.3
```

Para pegar apenas a parte inteira de um número usamos a função `to_i`, da seguinte forma:

```
peso = 90.82
inteiro = peso.to_i
puts inteiro # 90
```

Existem também o operador `**` que é o mesmo que exponenciação:

```
4 ** 2    # Isso é a mesma coisa que 4², que é 16.
36 ** 0.5 # Isso é a mesma coisa que 36 elavado a 1/2, qu
```

```
e é 6.
```

Nós podemos fazer operações mais complexas em Ruby usando parênteses, eles funcionam da mesma forma que na matemática:

```ruby
a = 9
b = 2
c = -4

resultado = (a + b) / c
puts "O resultado é #{resultado}" # O resultado é -3
```

Por fim, existe também os atalhos para operadores.
Digamos que você tenha uma variável `a` de valor `90` e você queira dividir `a` por dois, seria chato ter que digitar `a = a / 2`.
Para resolver esse problema nós usamos os atalhos `+=`, `-=`, `*=` e `/=`, por exemplo:

```ruby
a = 80
a /= 2  # a mesma coisa que a = a / 2, a agora tem valor de 40
a += 9  # a mesma coisa que a = a + 9, a agora tem valor de 49
a -= 19 # a agora tem valor de 30
a *= 2  # a agora tem valor de 80 novamente
```

Comparação

Comparação é uma parte *muito* importante em qualquer linguagem, ela serve para comparar valores e dizer se um é igual, diferente, maior ou menor que outro.

Em Ruby existem os seguintes comparadores:

Operador	Descrição
==	Testa por equalidade, retorna `true` se ambos os operandos forem iguais, `false` do contrário.
!=	Testa por inequalidade, retorna `true` se ambos os operandos forem diferentes, `false` do contrário.
>	Maior, retorna `true` se o primeiro operando for maior que o segundo, `false` do contrário.
<	Menor, retorna `true` se o primeiro operando for menor que o segundo, `false` do contrário.
>=	Maior ou igual, retorna `true` se o primeiro operando for maior ou igual ao segundo, `false` do contrário.
<=	Menor ou igual, retorna `true` se o primeiro operando for menor ou igual ao segundo, `false` do contrário.

Veja alguns desses operadores em ação:

```
irb(main):001:0> 2 == 2
=> true
irb(main):002:0> 9 == 1
=> false
```

```
irb(main):003:0> 9 > 1
=> true
irb(main):004:0> 80 >= 90
=> false
irb(main):005:0> 80 >= 80
=> true

irb(main):006:0> a = 5
=> 5
irb(main):007:0> b = 9
=> 9
irb(main):008:0> a == b
=> false
irb(main):009:0> a + 4 == b
=> true

irb(main):010:0> "naccib" == 'naccib'
=> true
```

Coleções

Uma coleção em Ruby é um conjunto de dados, por exemplo, os números 0, 2, 4 são a coleção dos 3 primeiros números pares.

Existem dois tipos principais de coleções em Ruby: **Array** e **Hash**.

Array

Arrays são objetos que guardam valores *ondernadamente*, isso quer dizer que para cada valor em uma array, existe um número correspondente a ele.
É assim que se declara uma `Array`:

```
minha_array = []
```

Nós podemos adicionar elementos à array da seguinte maneira:

```
minha_array[0] = 'naccib'
minha_array[1] = 'Krossvaca'
minha_array[2] = 'Luis'
```

Ou, de uma forma mais compacta (e preferível)...

```
minha_array = ['naccib', 'Krossvaca', 'Luis']
```

Nas arrays os elementos são acessados por seu índice (index) numérico, exemplo:

```
terceiro = minha_array[2]
puts terceiro # Luis
puts minha_array[0] # naccib
```

Se nós quisermos pegar o tamanho da array (quantos itens ela contém), usamos o método `length`:

```
puts minha_array.length # 3
```

Para inverter a ordem dos itens de uma array, usamos o método `reverse`:

```
nova_array = minha_array.reverse
puts nova_array # Escreve ["Luis", "Krossvaca", "naccib"]
```

É importante saber que:

> *Colchetes (`[]`) denotam o começo e o final de uma Array*
> *Cada lugar da array pode conter qualquer tipo de objeto, você não está misturando objetos, você está* ***fazendo uma coleção*** *deles.*
> *Os itens de uma array são acessados pelo seu índice numérico, lembre que em Ruby nós começamos a contar do 0. Isso quer dizer que o primeiro item está no index de número `0`.*

Existe uma coisa em Ruby chamada `Range`, que é um conjunto de números declarado da seguinte forma:

```
zero_ate_cem = 0..100
```

Isso quer dizer que `zero_ate_cem` é uma array com **todos** os números inteiros de `0` até `100`, nós vamos usar isso nos próximos exemplos.

Iteração por Loop

Iterar algo é percorrer todos ou alguns itens de algo, sendo que algo é uma coleção.
Em uma array nós temos *vários* métodos de iterar sobre os itens, vou mostrar três

For Loop

Um `for` loop é um loop declarado da seguinte forma:

```
for item in colecao do
   # bloco de código
end
```

Exemplo:

```
for numero in 0..10 do
   puts numero
end
```

Escreve o seguinte na tela:

```
0
1
2
3
4
5
6
```

```
7
8
9
10
```

Você pode ler isso como:

> *Para cada item, chamado numero, dentro dos números de 0 até 10, escreve esse numero na tela.*

While Loop

O `while` loop é um loop que não para enquanto a condição dada for mantida. Ele é declarado assim:

```ruby
while condition do
   # bloco de código
end
```

Exemplo:

```ruby
minha_array = [5, 'Kross', 9.52, 'Vaca']

n = 0
while n < minha_array.length do
  puts minha_array[n]
  n += 1
end
```

Esse loop é bem menos eficiente, mas ele tem suas vantagens.
É vantajoso usar um `while` em situações que você não sabe *quantas* vezes você vai iterar em uma situação.
O exemplo acima pode ser lido como:

> *Enquanto `n` for menor que o tamanho de `minha_array` escreva o item de índice `n` da array na tela e adicione um a `n`*

Until Loop

O `until` loop é um loop que para *quando* a condição for verdadeira.
Segue o sintaxe dele:

```
until condicao
   # código
end
```

Método `each`

O método `each` é uma maneira mais compacta de iterar em uma array, pode-se escrever um `each` de duas maneiras:

```
minha_array = 2..6
minha_array.each { |n| puts n ** 2 }
```

Ou...

```ruby
minha_array = 2..6
minha_array.each do |n|
  puts n ** 2
end
```

Os dois produzem o mesmo resultado, que é:

```
4
9
16
25
36
```

Manipulando Arrays

Existe um número de maneiras de manipular arrays: adicionando itens, removendo itens, trocando itens e etc.

Convertendo Arrays para String

Uma delas é usando o método `to_s`, assim:

```ruby
minha_array = [2, 'Kross', 5.2, 'Vaca']
puts minha_array.to_s
```

Escreve `[2, "Kross", 5.2, "Vaca"]` na tela.

Outra maneira é usando o método `join(separator)`, da seguinte maneira:

```
minha_array = [2, 'Kross', 5.2, 'Vaca']
puts minha_array.join '-'
```

Escreve `2-Kross-5.2-Vaca` na tela.
Se você quiser escrever cada elemento da array em uma linha separada, pode usar o código:

```
minha_array = [2, 'Kross', 5.2, 'Vaca']
puts minha_array.join '\n'
```

Isso vai escrever cada elemento em uma linha pois ele juntará cada elemento por `\n`, que é o separador de linhas.

Adicionando elementos a uma Array

Você pode adicionar elementos a uma array por dois métodos, um deles é usando o operador `+`:

```
minha_array = [2, 'Kross', 5, 'Vaca']
nova_array = minha_array + ['naccib', 11]

puts nova_array.join ', '
```

Escreve `2, Kross, 5, Vaca, naccib, 11` na tela.

O outro, que funciona da mesma forma que o operador `+`, é usando o método `push`.
O benefício é que a gente não precisa usar os parênteses (`[]`), ai fica um pouco mais legível:

```
minha_array = [2, 'Kross', 5, 'Vaca']
nova_array = minha_array.push 92, -1, 'Luis'

puts nova_array.join ', '
```

Isso escreve `2, Kross, 5, Vaca, 92, -1, Luis` na tela.

Outros métodos

Existem *vários* outros métodos para usar em Arrays, eis alguns.

```
minha_array = [2, 'Kross', 5, -6, 'Vaca']

minha_array.pop # Remove o ultimo item
puts minha_array.join ', '

minha_array.delete 'Kross' # Remove o elemento com valor
'Kross'
puts minha_array.join ', '

minha_array = minha_array.sort # Organiza os itens por or
dem na linha numérica
puts minha_array.join ', '
```

Apostila 2 — Vem ai

A segunda apostila vai conter o seguinte:

- Métodos
- Classes
- Hierarquia
- O que vier na telha

Fica ligado nos links do **Sobre** ali em baixo.

Sobre

Essa apostila foi criada por naccib e narrada pela bela voz e discurso do Natanael — Fábrica de Noobs.
Revisão por Krossbow, o maior memeguy do Brasil ocidental.

Fora Temer.