Ruby Style

Breno Martinusso
martinusso.com

WHY?

Programas são muito mais fáceis de manter quando todos os seus componentes têm um estilo consistente.

File organization

- file header
- require
- include
- definição de classes e modules
- main
- testing code
- file footer

```
# testing code (?)
# using FILE idiom
if FILE == $0

end
```

comments

- Comentários begin-end devem ser usados para comentários de documentação
- Comentários # podem ser usado para comentários de documentação e implementação

File Header or Footer

```
=begin
  * Name: Facture-e
  * Description: Aplicativo de NF-e e CT-e
  * Author: Tecsystem Tecnologia em
Software
  * Date: march 14, 2013
  * License: Proprietary
=end
```

Class/Module and Method Comments

```
# class Jedi
#
class Jedi
  # constructor of method
  #
  def initialize()
```

Indentation

Comprimento da linha (colunas)

- Máximo de 80 caracteres

Quebra de linha

- Após vírgula
- Após operador

variables

- Variáveis de Ruby não têm "definições". Então, você deve inicializar variáveis.
- Cada linha deve conter no máximo uma instrução

```ruby
# Certo
nome  = 'Breno'
idade = 18

# Errado
nome = 'Breno'; idade = 18
```

BLOCKS

```
# {} style
bar.foo(var){|p|
   ...
}
# do-end style
bar.foo(var) do |p|
   ...
end
# one-line block
bar.foo(var){|p|   }
```

case-when

case-when em Ruby não precisa do comando break!

```
case foo
when condition1
  ...
when condition2
  ...
else
  ...
end
```

Begin-rescue-ensure

try...except...finally

```
begin
  # instruções
rescue FooError => e
  # except
rescue BazError => e2
  # except
ensure
  # finally
end
```

Blank Lines

- O número de espaços deve ser equilibrada.
- Entre as seções de um arquivo de origem
- Entre classe e definições de módulo
- Entre os métodos
- Antes de blocos ou comentários de uma linha
- Entre seções lógicas dentro de um método para melhorar a legibilidade
-

Blank spaces

- Uma palavra-chave seguida de um parênteses devem ser separados por um espaço.
- O número de espaços deve ser equilibrada. (?)

```
# Certo
a+b
a + b

# Errado
a+ b
# interpreted as a(+b))
a +b
```

Naming conventions

classes/modules

- Nomes de classes e módulos devem ser substantivos
- UpperCamelCase

```
class Pessoa

class Carro::CarroPopular
```

Naming conventions

Methods

- Nomes de metódos devem ser verbos
- Todas as letras lower case
- Palavras separadas por underscores _

```
def fala

def fazer_algo

def fazer_algo_importante
```

Naming conventions

variables

- Todas as letras lower case
- Palavras separadas por underscores _

```ruby
nome = 'Breno'

idade = 1

sobre_nome = 'Martinusso'
```

Naming conventions

constants

- Todas as letras upper case
- Palavras separadas por underscores _

```
SISTEMA = 'Facture-e'
```

Pragmatic Programming Practices

Using attr_* to access

- Use attr_* como propriedade

```
def foo()
  @foo
end

attr_reader :foo
```

Pragmatic Programming Practices

without Parenthesis

- Alguns métodos são usados sem parenteses

```
# require
require 'foo/bar'

# include
include FooModule
```

Pragmatic Programming Practices

Reduce Repetition

```
# Errado
x = ModuleA::ClassB::method_c(a)
y = ModuleA::ClassB::method_d(b)

# Certo
cb = ModuleA::ClassB
x = cb::method_c(a)
y = cb::method_d(b)
```

DON'T PANIC