Padrão de Documentação

- **Nomenclatura de classes:**

```
#mau
class minha_classe
end

#bom
classe MinhaClasse
end

#mau
class classe
end

#bom
class Classe
end
```

- **Nomenclatura de métodos:**

```
#mau
def Metodo
end

#bom
def metodo
end

#mau
def MeuMetodo
end

#bom
def meu_metodo
end
```

- **Nomenclatura de variáveis**

O mais mnemônicas possível.

- **Use UTF-8 como a codificação do arquivo de fonte.**

- **Use dois espaços por nível de recuo (conhecido como soft tabs).**

```
# ruim - quatro espaços
def algum_metodo
    fazer_algo
end

# bom
def algum_metodo
  fazer_algo
end
```

- **Não use ';' para separar as declarações e expressões. Use uma expressão por linha.**

```
# ruim
puts 'foobar'; # ponto e vírgula supérfluo

puts 'foo'; puts 'bar' # duas expressões na mesma linha

# bom
puts 'foobar'

puts 'foo'
puts 'bar'

puts 'foo', 'bar' # isto aplica-se ao puts em particular
```

- **Prefira um formato de linha único para as definições de classe sem corpo.**

```
# ruim
class FooError < StandardError
end

# ok
class FooError < StandardError; end

# bom
FooError = Class.new(StandardError)
```

- **Use espaços em torno de operadores, após vírgulas, dois-pontos e ponto e vírgula, em volta do { e antes do }. Espaço em branco pode ser (na maioria dos casos) irrelevante para o intérprete de Ruby, mas sua**

utilização correcta é a chave para escrever um código facilmente legível.

```
soma = 1 + 2
a, b = 1, 2
1 > 2 ? true : false; puts 'Olá'
[1, 2, 3].each { |e| puts e }
```

- **A única exceção, sobre os operadores, é o operador de expoente:**

```
# ruim
e = M * c * * 2

# bom
e = M * c**2
```

- **Idente when no nível do case.**

```
# ruim
case
  when musica.nome == 'Misty'
    puts 'Não!'
  when musica.duracao > 120
    puts 'Muito tempo'!
  when Time.now.hour > 21
    puts "É tarde demais"
  else
    musica.Play
end

# bom
case
when musica.nome == 'Misty'
  puts 'Não!'
when musica.duracao > 120
  puts 'Muito tempo'!
when Time.now.hour > 21
  puts "É tarde demais"
else
  musica.play
end
```

- **Usar linhas vazias entre as definições de método e também para quebrar um método em parágrafos de lógica internas.**

```
def algum_metodo
```

```
  data = initialize(options)

  data.manipulate!

  data.result
end

def algum_metodo
  result
end
```

- **Evitar a vírgula após o último parâmetro em uma chamada de método, especialmente quando os parâmetros não são em linhas separadas.**

```
# ruim - mais fácil de mover ou adicionar/remover parâmetros, mas ainda não preferível
algum_metodo (
             tamanho,
             contagem,
             cor,
            )

# ruim
algum_metodo (tamanho, contagem, cor, )

# bom
algum_metodo (tamanho, contagem, cor)
```

Use espaços em torno do operador = quando para atribuição de valores padrão para os parâmetros do método:

```
# ruim
def algum_metodo (arg1=:default, arg2=nil, arg3=[])
  # fazer alguma coisa...
end

# bom
def algum_metodo (arg1 =: default, arg2 = nil, arg3 = [])
  # fazer alguma coisa...
end
```

- **Alinhe os elementos do array de literais, abrangendo várias linhas.**

```
# ruim - identação única
menu_item = ['Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam',
  'Feijão', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam']

# bom
```

```
menu_item = [
  'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam',
  'Feijão', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam'
]

# bom
menu_item =
  ['Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam',
   'Feijão', 'Spam', 'Spam', 'Spam', 'Spam', 'Spam']
```

- **Adicione sublinhados para literais numéricos grandes para melhorar a sua legibilidade.**

```
# ruim - quantos 0s existem?
num = 1000000

# bom - muito mais fácil de analisar para o cérebro humano
num = 1_000_000
```

- **Não coloque uma linha vazia entre o bloco de comentários e o def.**

- **Evite o espaço em branco à direita.**

- **Termine cada arquivo com uma nova linha.**

- **Não use comentários em bloco.**

```
# mau
=begin
linha de comentário
outra linha de comentário
=end

# bom
# linha de comentário
# outra linha de comentário
```

- **Aproveitar o fato de que, if e case são expressões que retornam um resultado.**

```
# mau
if condicao
  resultado = x
else
  resultado = y
end
```

```
# bom
resultado =
  if condicao
    x
  else
    y
  end
```

- **Use ! em vez de not.**

```
# mau - chaves são necessárias por causa da precedência de operadores
x = (not algo)

# bom
x = !algo
```

Evitar a utilização de !!.

```
# mau
x = 'teste'
# obscura checagem de nil
if !!x
  # corpo omitido
end

x = false
# dupla negação é inútil em booleanos
!!x # => false

# bom
x = 'teste'
unless x.nil?
  # corpo omitido
end
```

- **Nao usar and e or. Sempre use && e || em vez disso.**

```
# mau
# expressão booleana
if alguma_condicao and alguma_outra_condicao
  faca_alguma_coisa
end

# controle de fluxo
document.saved? or document.save!
```

```
# bom
# expressão booleana
if alguma_condicao && alguma_outra_condicao
  faca_alguma_coisa
end

# controle de fluxo
document.saved? || document.save!
```

- **Nunca use unless com else. Reescreva estas com o caso positivo primeiro.**

```
# mau
unless sucesso?
  puts 'fracasso'
else
  puts 'sucesso'
end

# bom
if sucesso?
  puts 'sucesso'
else
  puts 'fracasso'
end
```

- **Não use parênteses em torno da condição de uma if/unless/while/until.**

```
# mau
if (x > 10)
  # corpo omitido
end

# bom
if x > 10
  # corpo omitido
end
```

- **Omita parênteses para chamadas de método sem argumentos.**

```
# mau
Kernel.exit!()
2.even?()
fork()
'test'.upcase()
```

```
# bom
Kernel.exit!
2.even?
fork
'teste'.upcase
```

- **Evite return, onde não é necessário.**

```
# mau
def algum_metodo(algum_array)
  return algum_array.size
end

# bom
def algum_metodo(algum_array)
  algum_array.size
end
```

- **Use somente simbolos ascii nos comentarios.**