Ruby

Cassiano

13/11/2015

1. Overview

História

- Nasceu em 1993, numa discussão entre Yuhikiro Matsumoto e um colega.
- “Matz” procurava por uma linguagem de script orientada a objetos.
- A primeira versão estável de Ruby (1.2) foi lançada somente em 1998.
- Naquela época a linguagem só era famosa no Japão.
- Em 2005 houve uma explosão de uso. Motivo: Ruby on Rails.
- Naquela época Ruby era usado para a criação de CGI-Scripts, mas Ruby on Rails estava um passo na frente com a prática de MVC’s.

Ruby is simple in appearance, but is very complex inside, just like our human body.

~Matz

Razões para se usar Ruby

- Fácil aprendizagem e adaptação.
- Open Source.
- Contém tratamento de Strings e Expressões Regulares.
- Tudo em Ruby é objeto e possui valor, como deve ser.
- Código elegante, conciso e de alta legibilidade.
- Altamente Portável.

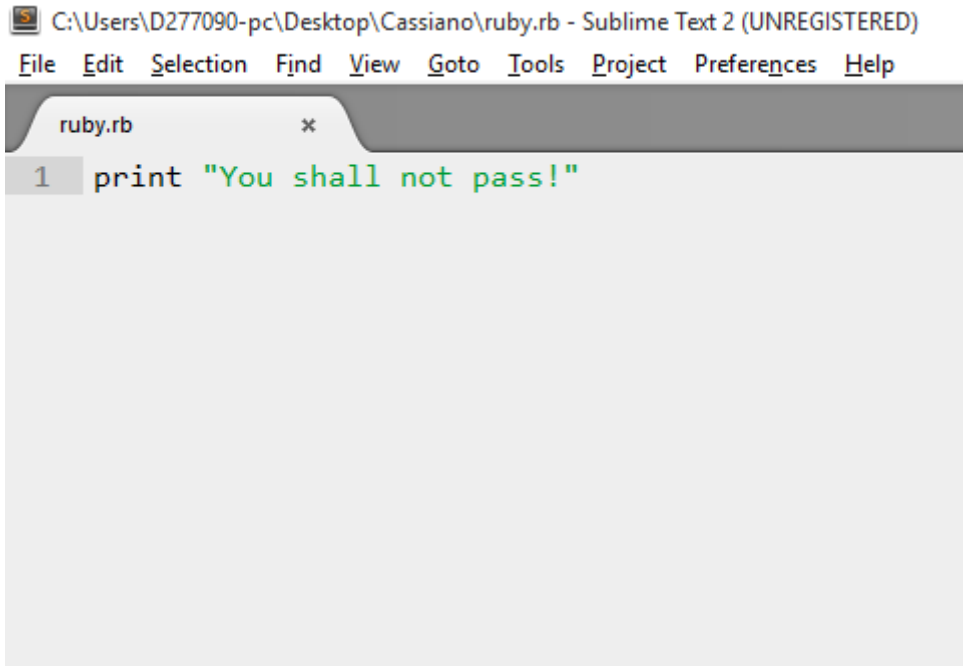
Razões para se usar Ruby

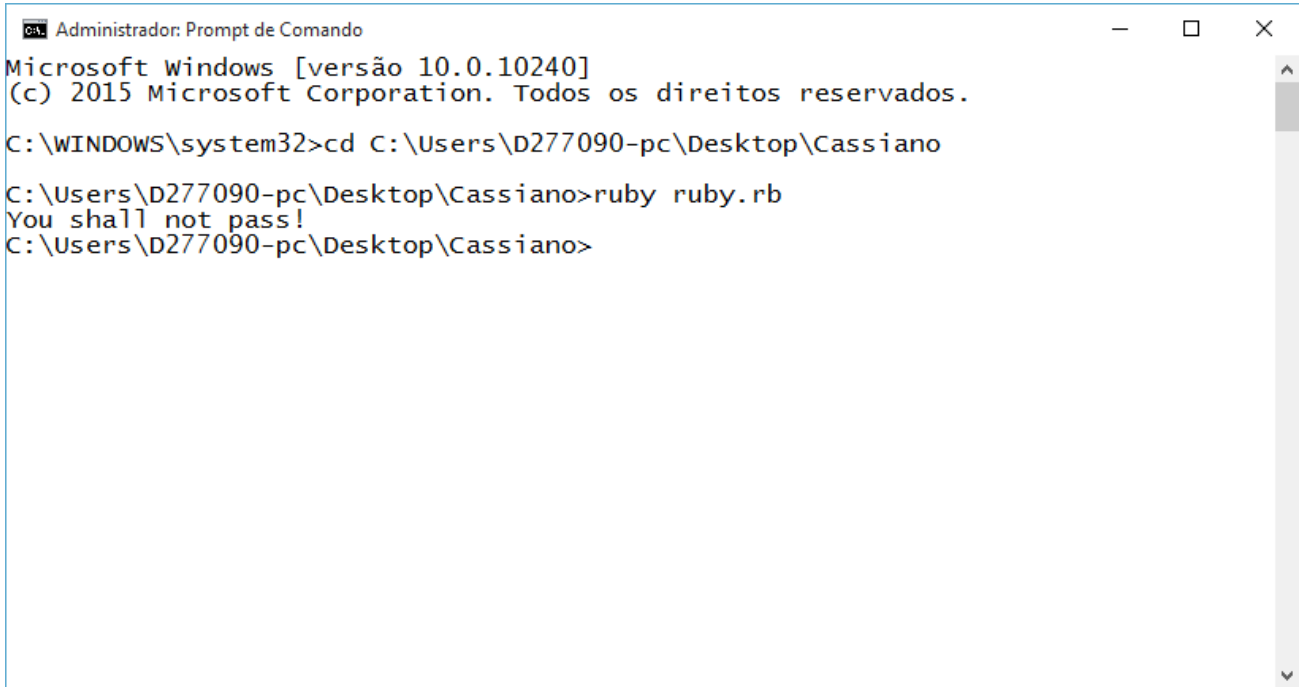
- Robusta aplicação para desenvolvimento Web (Ruby on Rails).
- Linguagem de Script (Tende a ser de fácil manuseio).
- Sistema de Threads independente se o Sistema Operacional permite ou não.
- Comunidade ativa e acolhedora independente da experiência do programador.
- Ecossistema de Plugins (Gems) enorme e livre para download.

Como usar?


Ruby
O MELHOR AMIGO DO PROGRAMADOR
Google™ Pesquisa Personalizada
Procurar
Downloads
Documentação
Módulos
Comunidade
Notícias
Segurança
Sobre Ruby
Baixar o Ruby
Aqui você poderá obter as distribuições mais recentes de Ruby em seus sabores preferidos. A versão estável atual é a 2.2.3. Por favor certifique-se de ter lido a Licença do Ruby.
Formas de instalar o Ruby
Existem diversas ferramentas para instalar o Ruby em cada grande plataforma:
No Linux/UNIX, você pode usar o sistema de gerenciamento de pacotes da sua distribuição ou ferramentas de terceiros (rbenv e RVM).
Em máquinas com OS X, você pode usar ferramentas de terceiros (rbenv e RVM).
Em máquinas com Windows, você pode usar o RubyInstaller ou o pik.
Consulte a página Instalação para mais detalhes sobre como usar sistemas de gerenciamento de pacotes ou ferramentas de terceiros.
É claro, você também pode instalar Ruby a partir do código fonte em todas as principais plataformas.
Primeiros passos, é fácil!
Try Ruby! (in your browser)
Ruby em Vinte Minutos
Ruby a partir de outras linguagens
Explore um novo mundo…
Documentação
Livros
Bibliotecas
Histórias de sucesso
Participe de uma comunidade simpática e em crescimento.
Listas de E-mail: Fale sobre Ruby com programadores de todo o mundo.

Executando um Programa





2. Sintaxe

Separação de Instruções

- A separação de instruções é feita com '\n' ou ';'.
- Não é permitido cadeia de instruções sem separação.
- Espaços em branco e 'tabs' são ignorados, exceto quando aparecem em Strings.
- Operadores no fim da linha extendem a instrução.

ruby.rb

```ruby
if (1 != 1); print "X";
else; print "Y"
end

if(1 == 1) print "X"
else print "Y"
end

if(1 == 1)
    print "X"
else
    print "Y"
end
```

Comentários

- Usa-se # para comentários até o fim da linha.
- Para comentários extendidos ‘=begin’ e ‘=end’.
- Para documentação há um sistema criado pelos usuários chamado RDOC (Ruby Documentation System.

ruby.rb

```
1  # Comentário de linha única
2  # Linguagens de Programação 2015
3
4  =begin
5  Comentário
6  Extendido
7  =end
8
```

3. Operadores

Operadores Aritméticos

- A maioria dos operadores é interpretado como método. E.g. 2 + 3 equivale a 2.+(3).

- + Adição .
- - Subtração.
- * Multiplicação.
- / Divisão.
- % Módulo.
- ** Exponenciação (2^3 == 2**3).

Operadores de Comparação

- == Igual.
- != Diferente.
- > Maior.
- < Menor.
- >= Maior igual.
- <= Menor igual.
- <=> Menor é 1, Igual é 0, Maior é 1.
- === Equivalente.
- eql? Identificador de Hash.
- equal? Identidade.

Operadores Lógicos

- && e and – Operador lógico AND.
- || e or – Operador lógico OR.
- ! e not – Operador lógico NOT.

Operadores de Range

- .. – Cria um Range do início ao fim (inclusive).
  - Exemplo: 1..10 = Range de 1 a 10.

- ... – Cria um Range do início até o penúltimo número.
  - Exemplo: 1..10 = Range de 1 a 9.

Operador Defined

- O operador ‘defined? é um operador especial para determinar de qualquer forma se a expressão passada é definida.

ruby.rb

```ruby
var = "variavel"
if(defined? var) # Verdadeiro
defined? var # local-variable
defined? $_ # global-variable
defined? var2 # nil
defined? var.to_i # method
defined? puts # method

```

Sobrecarga de Operadores

- Ruby permite livremente a sobrecarga de operadores.

- Exemplo de sobrecarga para +

ruby.rb

```ruby
class Tempo
    @segundos

    def initialize(segundos)
        @segundos = segundos
    end

    def +(s)
        @segundos += s
    end
end

a = Tempo.new(2)
a + 3
```

3. Tipos de Dados

Tipagem

Ruby usa tipagem:

- Dinâmica
- Implícita
- Forte

ruby.rb

```ruby
a = 100
b = "Ruby"
(1..5).each{ a = a * 100 ;  puts "#{ a.class} #{a}"  }
a + b #erro
```

```
C:\Users\D277090-pc\Desktop\Cassiano>ruby ruby.rb
Fixnum 10000
Fixnum 1000000
Fixnum 100000000
Bignum 10000000000
Bignum 1000000000000

C:\Users\D277090-pc\Desktop\Cassiano>
```

Fixnum

- Ruby usa a classe Fixnum para representar inteiros,
- Se o número exceder o limite de Fixnum, ele é automaticamente convertido para Bignum
- Contém métodos base como to_s(base=10), to_f, .odd?, zero?
- Não admite a inserção de Singleton Methods.

Fixnum Exemplos

ruby.rb

```ruby
class Fixnum
    def incrementado
        self + 1
    end
end

print 3.incrementado
```

Exemplo saída = 3

ruby.rb

```ruby
def 3.incrementado
        self + 1
    end
end

print 3.incrementado
```

Exemplo saída = unexpected tINTEGER

ruby.rb

```ruby
puts 156.to_s(2) #10011100
puts 156.to_s(8) #234
puts 156.to_s(10) #156
puts 156.to_s(16) #9c
puts 156.to_s(32) #4s
```

Float

- Ruby usa a classe Float para representar números flutuantes.
- São determinados usando arquitetura de dupla-precisão.
- A classe Float possui constantes comuns como EPSILION, INFINITY, MAX, etc.
- Também possui os métodos de conversão to_s, to_i e to_f (self).

String

- A classe String armazena cadeias de caracteres.

- O padrão Encoding de caracteres do Ruby é ASCII. Podendo alterá-lo no início do código escrevendo $KCODE = ‘u’.
  - a – ASCII.
  - e – EUC.
  - n – Nenhum (O mesmo que ASCII).
  - u – UTF8.

- Pode ser facilmente criada das formas a = “STRING” ou a = String.new(“STRING).

String

- A substituição de expressão em uma String é feita da seguinte forma:

ruby.rb

```ruby
var = 100
puts "Eu tenho #{21} anos."      # "Eu tenho 21 anos."
puts "Resultado #{var * 10}."    # "Resultado 1000."
puts "Exponencial #{var ** 2}." # "Exponencial 10000"
```

- A classe String também possui vários métodos comuns. E.g downcase, casecmp, capitalize, capitalize!, empty?, length, etc.

Array

- A classe Array armazena quaisquer objetos por índices inteiros.
- Todas começam em 0, assim como C e Java. Índice negativo significa a última posição relativa, exemplo: -1 = última pos, -2 = penúltima e assim em diante.
- O tamanho de uma Array pode mudar dinamicamente ao adicionar um novo objeto.
- Contém métodos comuns como at, take & drop, length, empty, push & <<, insert, pop, delete, etc.

Array

- As Arrays são criadas das seguinte formas:

ruby.rb

```ruby
nomes = Array.new
nomes = Array.new(20)
nomes = Array.new(4, "LP")
nomes = Array.new(3) { |a| a *= 2 }
nomes = Array.[](1, 2, 3, 4, 5)
nomes = Array[1, 2, 3]
```

Palavras Reservadas

BEGIN	do	next	then
END	else	nil	true
alias	elsif	not	undef
and	end	or	unless
begin	ensure	redo	until
break	false	rescue	when
case	for	retry	while
class	if	return	while
def	in	self	__FILE__
defined?	module	super	__LINE__

Ponteiros

Ruby não usa Aritmética de Ponteiros (A não ser que se use uma extensão para isso).

4. Estruturas de Controle

Condicional – If/else

- Assim como em toda linguagem moderna, Ruby segue o mesmo padrão If then else.

ruby.rb

```ruby
x =2

if x > 2
    puts 'I see dead people'
elsif x < 2
    puts 'Let the pain speak to me'
else
    puts 'On work'
end
```

Condicional - Case

- A condicional Case de Ruby pode ter ou não um método predecessor a ela.
- No lugar de switch() se usa case e no lugar de case se usa when.
- O padrão (default) é, na verdade, um else.
- Não permite múltiplas respostas. (Várias condições podem ser separadas por vírgula num mesmo when).

Condicional - Case

ruby.rb

```ruby
puts case idade = 18
when 0..2
    "Bebê"
when 3..11
    "Criança"
when 12..17
    "Adolescente"
else
    "Adulto"
end
```

Condicional - Unless

- Ruby também tem uma variação de IF chamada Unless, que basicamente faz o contrário do IF.

- Unless não suporta elsif.

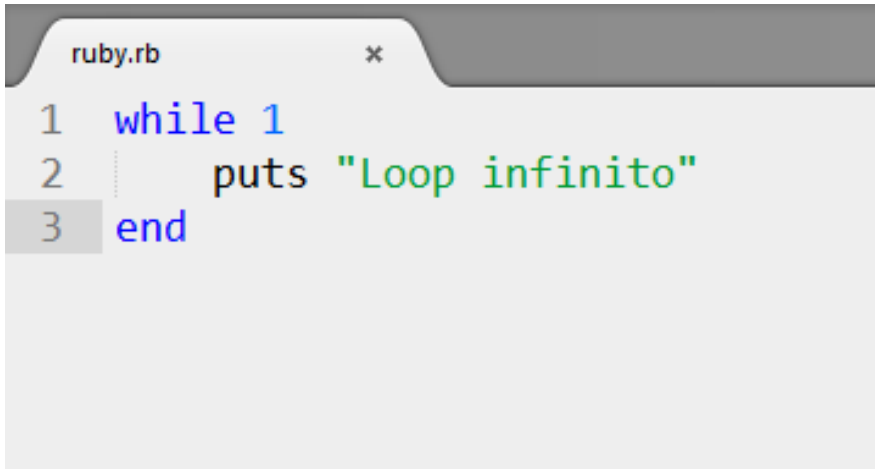
ruby.rb

```ruby
x=2

unless x != 2
    puts 'I see dead people'
else
    puts 'On work'
end
```

Repetição- While

- Executa o bloco de comando enquanto a condição for verdadeira.



Repetição- While

- Ruby também possui duas versões modificadas do tradicional While.

ruby.rb

```ruby
puts "While infinito" while 1

begin
    puts "Esse inacreditável loop infinito não acontecerá"
end while 1
```

Repetição- Until

- A versão contrária de While em Ruby é Until.

ruby.rb

```ruby
until false do
    puts "Sneaky Loop Infinito"
end
```

Repetição- Until

- Until também possui duas versões modificadas.

ruby.rb

```ruby
begin
    puts "Another Infinite Loop"
end until false

puts "Unreachable Loop" until false
```

Repetição- For

- Executa o bloco de código para cada elemento na expressão.
- For de Ruby não possui condicional livre.
- Sempre inicializa a variável com o primeiro inteiro da expressão.
- For assimila-se a .each

Repetição- For

ruby.rb

```ruby
for a in 2..9 do #do opcional
    puts "O valor de a é #{a}"
end
```

Repetição- Each

- Each é um método de Collections em Ruby.

- Segue o mesmo princípio de for e foreach de linguagens mais modernas (iterar).

ruby.rb

```ruby
array = [1,2,3,4,5]
array.each do |i|
   puts i
end
```

4. Variáveis e Constantes

Variáveis Locais

- São iniciadas por letras minúsculas ou _.

- Quando não inicializadas são interpretadas como métodos sem argumentos.

ruby.rb

```ruby
puts "teste"
puts = 2
print puts

Saída: #teste
#2
```

Variáveis Globais

- Variáveis globais se iniciam com prefixo $ e quando não inicializadas recebem ‘nil’.

ruby.rb

```ruby
$var = 10
class Classe1
  def print_global
    puts "#{$var}"
  end
end
class Classe2
  def print_global
    puts "#{$var}"
  end
end

c1 = Classe1.new
c1.print_global # "10"
c2 = Classe2.new
c2.print_global # "10"
```

Variáveis de Instância

- Têm prefixo @ e se não inicializadas recebem nil.

ruby.rb

```ruby
class Teste
    def initialize(nome)
        @nome = nome
    end

    def mostraNome
        puts @nome
    end
end

a = Teste.new("Ruby")
a.mostraNome
```

Variáveis de Classe

- Variáveis de Classe possuem o prefixo @@.

- Precisam ser inicializadas antes de serem usadas.

ruby.rb

```ruby
class QualquerClasse
    @@variavel
    def teste
        print @@variavel
    end
end

a = QualquerClasse.new
a.teste #ERRO
```

Constantes

- Constantes são criadas usando letra maiúscula na primeira letra.

- Quando criadas em Classes ou Módulos só podem ser usadas localmente. Porém quando criadas fora o uso se torna global.

ruby.rb

```ruby
A = 100

class Exemplo
    VAR = 200
    def test
        puts "#{A}"
        puts "#{VAR}"
    end
end

object=Exemplo.new()
object.test
puts VAR #erro
```

5. Funções

Funções

- As funções de Ruby são bem parecidas com as de qualquer linguagem.
- É obrigatório o uso de letra minúscula no começo de uma função.
- Todas funções precisam ser definidos antes de serem chamados.
- Toda função retorna um valor como padrão. Quando não é usado o termo return, a função retorna o último valor usado por ela.

Funções

ruby.rb

```ruby
def metodo (var1 = 1, var2 = 2)
    print var1
    print var2
end

metodo # 12
metodo 25, 20 # 2520
metodo "teste", "oi" # testeoi
```

Passagem de Parâmetros

- Todos os parâmetros são passados como valor, por referência.

- Há uma maneira de passar um número desconhecido de parâmetros para uma função.

ruby.rb

```ruby
def teste (*test)
   puts "A quantidade de parâmetros é #{test.length}"
   for i in 0...test.length
      puts "#{test[i]}"
   end
end
teste "Cassiano", "11", 2.1
teste "Ruindows", "36", "M", 1
teste
```

Garbage Collector

- Ruby possui um módulo para Garbage Collector.
- Pode ser desativado/ativado executando GC.disable / GC.enable
- Possui funções interessantes como stat e count.
- Houve um grande aumento de desempenho na última versão de Ruby (2.2), porém agora o core consome mais memória.

Curto Circuito

- Assim como as outras linguagens, Ruby também possui curto circuito

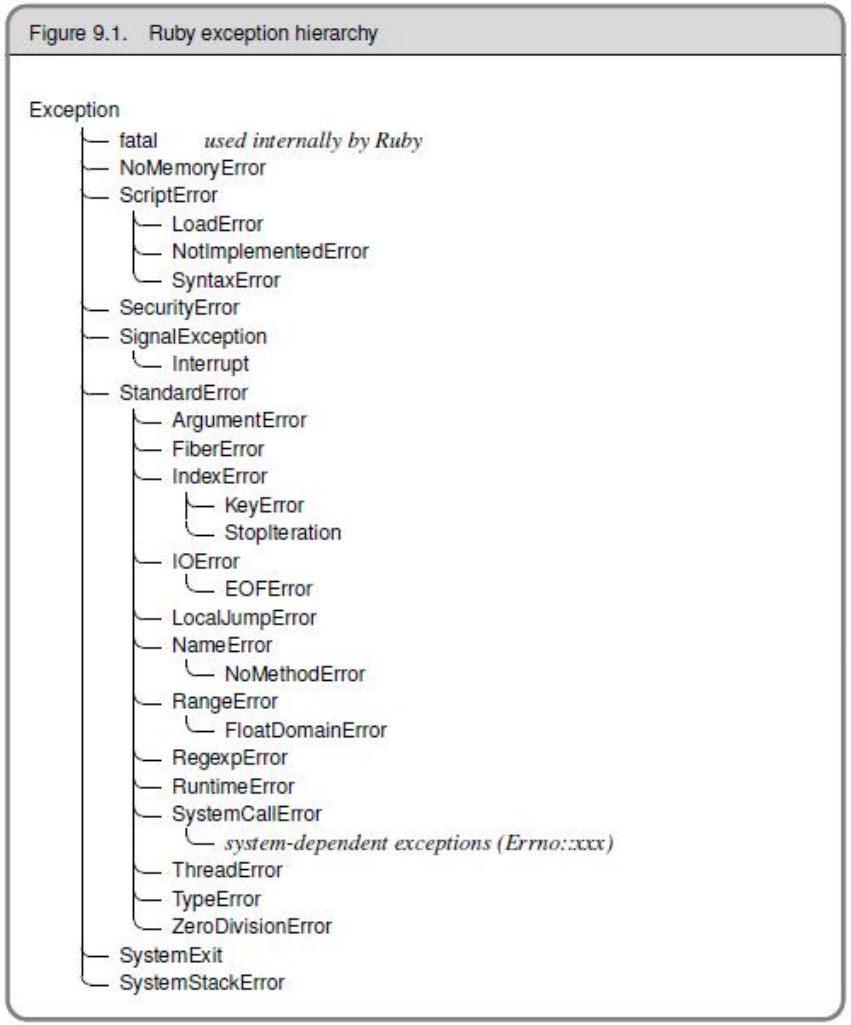
ruby.rb

```ruby
a = 1
b = 1
if a += 1 || b += 1
    puts a # 2
    puts b # 1
end
```

6. Tratamento de Exceções

Tratamento de Exceções

Figure 9.1. Ruby exception hierarchy



Tratamento de Exceções

- Usa-se os métodos raise e rescue para tratamento de exceções.

- Para criar uma classe de Exceção basta herdar de Standard Error.
  - E.g : class ErroCriacao < Standard Error.

Tratamento de Exceções

ruby.rb

```ruby
def levanta_excecao
    puts 'Estou antes do raise.'
    raise 'Um erro ocorreu'
    puts 'Estou depois do raise'
end
levanta_excecao
```

ruby.rb

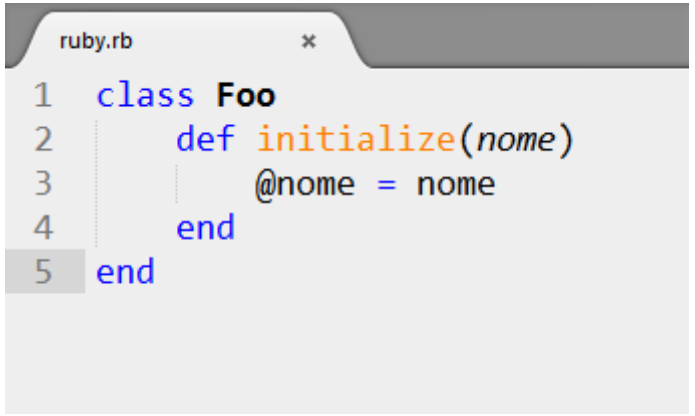
```ruby
def inverte(x)
    raise ArgumentError, 'O argumento não é numérico' unless x.is_a? Numeric
    1.0 / x
end
puts inverte(2)
puts inverte('Não é um número')
```

7. Orientação de Objetos

Classes

- Uma classe é iniciada obrigatoriamente com Letra Maiúscula.

- Da mesma forma que a maioria dos Tipos de Dados já citados, as classes são instanciadas com o método .new().



Herança

- Ruby permite apenas herança Simples.
- Também possui suporte para polimorfismo.

ruby.rb

```ruby
class Foo
    def initialize(nome)
        @nome = nome
    end

    def mostraNome
        puts @nome
    end
end

class Dumb < Foo
    def mostraNome
        puts @nome + " Dokie"
    end
end

a = Foo.new("Cassiano")
b = Dumb.new("Ruby")
a.mostraNome #Cassiano
b.mostraNome #Ruby Dokie
```

Interface

- Não existe suporte para Interfaces em Ruby.

- Mas existe uma Gem chamada Interface, criada por usuários, que pode ser usada.

Avaliação da LP

- Confiabilidade: Parcial.
- Eficiência: Sim.
- Portabilidade: Excelente.
- Reusabilidade: Sim (por ser uma linguagem script e puramente orientada a objetos).
- Concorrência: Sim (com Threads).
- Tipagem: Dinâmica, Implícita e Forte.
- Legidibilidade: Excelente.
- Exceções: Sim.

Referências

- https://www.codecademy.com/pt-BR/learn/ruby
- https://www.ruby-lang.org/pt/
- http://www.tutorialspoint.com/ruby/