Aprendendo Ruby on Rails

Sejam bem vindos!

Rudiney Altair Franceschi
rudi.atp@gmail.com

Objetivo:

- *Entender como o RoR funciona.*
- *Aprender o máximo possível.*
- *Saber como seguir aprendendo cada dia mais.*
- *Vale a pensa usar profissionalmente?*

O que vamos precisar?

- *Lógica de programação Básica*
- *Banco de Dados / SQL.*
- *WEB / HTTP / Estrutura.*

Vamos precisar pro curso.

Regras.

- *Não fique com dúvidas.*
- *Você faz o curso.*
  - *Quer saber algo que não está neste guia?*

Pergunte.

Início.

inicio/

Ruby / Rails

Ruby *on* *Rails*

Linguagem

Java
PHP
Python
Pearl

Framework

Cake
Grails
Hibernate
JUnit

inicio/

Ruby / Rails

Ruby *on* *Rails*

Yukihiro Matsumoto

1995

Núcleo ruby

http://www.ruby-lang.org/pt/comunidade/nucleo-de-ruby/

David Heinemeier Hansson

2004

Rails Core Team:

http://rubyonrails.org/core

inicio/

A Comunidade.

Site Oficial:

- http://rubyonrails.org/

Documentação Completa:

- http://api.rubyonrails.org/

Guias Oficiais:

- http://guides.rubyonrails.org/

Twitter:

- http://search.twitter.com/search?q=rails

Aprendendo Ruby:

- http://rubylearning.com/

Ruby Inside:

- http://www.rubyinside.com.br/

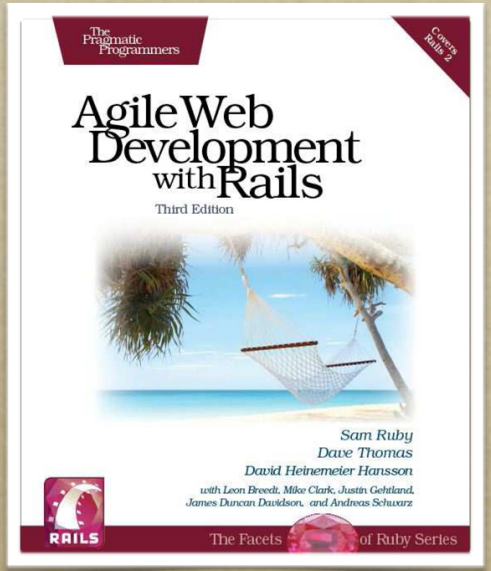
Ruby Flow:

- http://www.rubyflow.com/

Subforúm no GUJ:

- http://www.guj.com.br/forums/show/21.java

Livros.



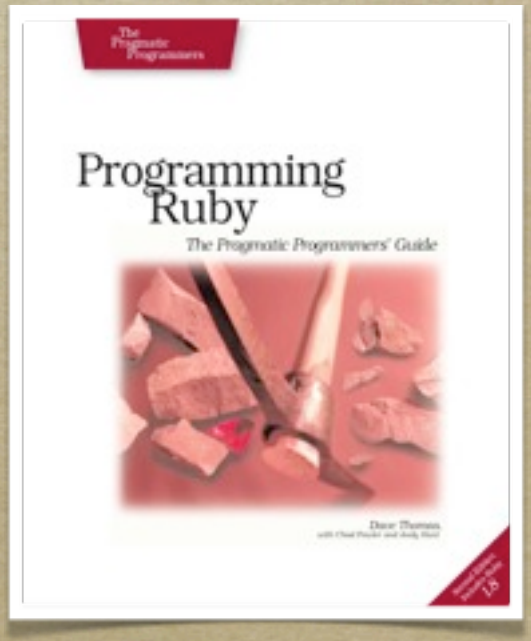
Agile Web Development With Rails.

Sam Ruby
Date Thomas
David Hansson

Programming Ruby.

The Pragmatic Programmer's Guide

Dave Thomas
Chad Fowler
Andy Hunt



Instalação do Ambiente.

ruby/ Instalação do Ruby

No site oficial do Rails tem tudo que precisamos

http://rubyonrails.com.br/

Ali tem a instalação para todos os ambientes...

O Ruby.

ruby/

O Ruby

"É uma linguagem dinâmica, open-source , com foco na ***simplicidade*** *e na* ***produtividade****. Tem uma sintaxe elegante que torna natural de ler e fácil de escrever."*

ruby/

O Ruby

Perl + Smalltalk + Eiffel + Ada + Lisp

ruby/ Interactive Ruby

- *O IRB é um interpretador ao-vivo de comando ruby.*
- *Você pode testar qualquer comando no irb. Basta escrever e apertar [enter]. O resultado é imediato.*
- *É uma ferramenta muuuitooo útil para o programador.*

ruby/ Interactive Ruby

Expressões numéricas tradicionais:

```
>> 3 * (2 + 5) / 8
=> 2
```

Estão disponíveis os operadores:

+ - * / ** %

Soma

Subtração

Multiplicação

Divisão

Potência

Resto da divisão

ruby/ Interactive Ruby

Valores podem ser atribuídos a variáveis:

```
>> resultado = 4 ** 2
```

puts é o método utilizado para imprimir valores.

```
>> puts(resultado)
```

ou

```
>> p resultado
```

String:

Podem ser delimitadas por aspas simples ou duplas:

```
>> texto = “valor inicial”
```

<< append *+ concatenar*

```
>> texto << “ alterado”
```

Interpolação de Strings:

```
>> resultado = 11 * 33
>> texto = “O resultado de 11 * 33 é #{resultado}”
>> puts(texto)
```

Interpolação exige aspas duplas.

É sempre melhor utilizar interpolação, pois o código é mais limpo e mais rápido de ser executado.

Símbolos

Também são texto, como as Strings,
porém devem ser precedidos por :

A diferença é que são imutáveis e compartilhados.

Responda

Por que a comparação entre símbolos é muito mais rápida que entre Strings?

```
>> simbolo1 = :meu_simbolo
>> simbolo2 = :meu_simbolo
>> simbolo1 == simbolo2
```

Boolean

Existem também os tipos booleanos:
true e false

```
>> 23 > 7
=> true
```

Vetores

```
>> puts [1, 2, 3, 4, 5]
1
2
3
4
5
=> nil
```

Hash

```
>> linguagem = {:nome => ‘Ruby’, ‘ano’ => 1995}
>> puts “#{linguagem[:nome]} = #{linguagem[‘ano’]}”
```

Hash pode ser comparado a um objeto JavaScript. É um objeto cuja chave pode ser qualquer outro objeto.

Hash e Vetor são muito utilizados em Ruby on Rails

Estruturas de controle

if

```
>> if (23 > 7)
>>   puts('23 é maior que 7')
>> end
```

Estruturas de controle

case

```
>> idade = 25
>> case idade
>> when 0..17 then
>>   puts ‘Menor’
>> when 18..999 then
>>   puts ‘Maior’
>> end
```

Estruturas de controle

if / elsif

```
>> idade = 25
>> if idade < 18
>>   puts 'Menor'
>> elsif idade == 18
>>   puts 'Tem 18'
>> elsif idade > 18
>>   puts 'Maior'
>> end
```

Ruby / Orientação a Objetos

Orientação a Objetos

Tudo no Ruby é um objeto!

tudo mesmo!

***.class** nos diz qual é a classe de um objeto:*

```
>> “Texto”.class()
=> String
>> 77.class()
=> Fixnum
```

ruby/ Orientação a Objetos

Até mesmo true, false e nil (objeto nulo do Ruby) tem suas próprias classes:

```
>> true.class()
=> TrueClass
>> false.class()
=> FalseClass
>> nil.class()
=> NilClass
```

viu? tudo mesmo!

ruby/ Orientação a Objetos

Para criar um novo objeto, basta invocar o método new

```
>> framework = String.new('Ruby on Rails')
=> "Ruby on Rails"
```

ruby/ Orientação a Objetos

Toda a mágica do OO está no . (ponto)

```
>> objeto.metodo
```

Aqui temos uma cidade

temos o pais do estado da cidade

```
>> cidade.estado.pais.nome
```

finalmente, o nome do pais do estado da cidae

pedimos o estado da cidade

ruby/ Orientação a Objetos

O que acontece quando invocamos um método que não existe?

```
>> linguagem = “Ruby”
>> linguagem.tamanho()
```

ruby/ Orientação a Objetos

Por ser uma linguagem dinâmica, Ruby permite que sejam definidos novos métodos em tempo de execução.

```
>> linguagem = “Ruby”
>> def linguagem.tamanho()
>>   return self.size()
>> end
>> linguagem.tamanho()
```

Orientação a Objetos

Isto é chamado de meta-programação.

Meta-programação é a capacidade de gerar código em tempo de execução.

ruby/ Orientação a Objetos

def é uma palavra chave Ruby para a definição (criação) de métodos.

Métodos podem, é claro, receber parâmetros.

ruby/ Orientação a Objetos

```
>> pessoa = Object.new()
>> def pessoa.vai_para(lugar)
>>   puts “Indo para #{lugar}.”
>> end
>> pessoa.vai_para(‘praia’)
```

Orientação a Objetos

É possível definir valores padrões para os parâmetros dos métodos.

```
>> pessoa = Object.new()
>> def pessoa.vai_para(lugar = ‘praia’)
>>   puts “Indo para #{lugar}.”
>> end
>> pessoa.vai_para()
>> pessoa.vai_para(‘clube’)
```

ruby/ Orientação a Objetos

*Os métodos podem receber n parâmetros dinamicamente. Todos os parâmetros passados para o método são recebidos em um único vetor. Para isso, declara-se o parâmetro com **

```
>> pessoa = Object.new()
>> def pessoa.vai_para(*lugares)
>>   puts “Indo para #{lugares.join(‘ e ’)}.”
>> end
>> pessoa.vai_para()
>> pessoa.vai_para(‘clube’, ‘praia’, ‘casa’)
```

Classes

Para não precisar ficar adicionando métodos em todos os objetos que criamos, utilizamos as classes, que servem como uma fábrica de objetos.
Possibilitam a criação de objetos, incluindo quantos métodos forem necessários.

Orientação a Objetos

Classes

```
>> class Pessoa
>>   def fala
>>     puts 'Ola, eu sou uma pessoa!'
>>   end
>>
>>   def vai_para(lugar = 'praia')
>>     puts "Indo para #{lugar}."
>>   end
>> end
```

Classes

```
>> pessoa = Pessoa.new
>> pessoa.class
>> pessoa.fala
>> pessoa.vai_para('faculdade')
```

ruby/ Orientação a Objetos

O grande diferencial das classes Ruby, é que são classes abertas.

```
>> class Pessoa
>>   def idade
>>     puts “Tenho 27 anos.”
>>   end
>> end
>> pessoa = Pessoa.new
>> pessoa.idade
```

ruby/ Orientação a Objetos

Qualquer classe pode ser aberta e qualquer método pode ser redefinido, incluindo métodos do próprio Ruby.

```
>> class Fixnum
>>   def +(outro_numero)
>>     self - outro_numero
>>   end
>> end
>> 3 + 2
```

ruby/ Orientação a Objetos

Um método pode chamar outro método do próprio objeto. Para isto, basta usar a palavra self, que é uma referência para a própria instância.
Self do Ruby pode ser comparado a this de Java e C#.

Quando não especificamos o objeto em que o método está sendo chamado, o Ruby assume que seja em self.

Atributos e propriedades

Atributos são variáveis da instância de um objeto.
São privados e começam com @.
Não é possível alterar um atributo de fora da classe.

Atributos e propriedades

```
>> class Pessoa
>>   def muda_nome(novo_nome)
>>     @nome = novo_nome
>>   end
>>
>>   def diz_nome
>>     puts “Meu nome é #{@nome}.”
>>   end
>> end
```

ruby/ Orientação a Objetos

Podemos executar um código automaticamente quando uma nova instância da classe é criada. Para isto, utiliza-se um método especial, chamado initialize

```
>> class Pessoa
>>   def initialize
>>     puts “Nasceu uma nova pessoa.”
>>   end
>> end
>> pessoa = Pessoa.new
```

ruby/ Orientação a Objetos

O método initialize é privado, e não pode ser chamado de fora da classe. É possível passar parâmetros:

```
>> class Pessoa
>>   def initialize(nome)
>>     @nome = nome
>>     puts “#{nome} acaba de nascer.”
>>   end
>> end
>> pessoa = Pessoa.new(‘Felipe’)
```

Acessores e Modificadores

Métodos acessores e modificadores são muito comuns e dão a idéia de propriedades do objeto. Existe uma convenção para a definição desses métodos, assim como Java possui sua convenção para definição dos getters e setters.

Acessores e Modificadores

```
>> class Pessoa
>>   def nome
>>     @nome
>>   end
>>
>>   def nome=(novo_nome)
>>     @nome = novo_nome
>>   end
>> end
>> pessoa = Pessoa.new
>> pessoa.nome=('Felipe')
>> pessoa.nome
```

Syntax Sugar

O objetivo de Matz foi fazer com que Ruby fosse uma linguagem extremamente legível.

```
>> pessoa.nome=('Felipe')
>> pessoa.nome = 'Felipe'
>>
>> 10 + 3
>> 10.+(3)
```

```
>> pessoa.[]=('nome','felipe')
>> pessoa.['nome'] = 'felipe'
```

Métodos de Classe

Métodos de classe não necessitam de um objeto instanciado para serem invocados.

```
>> class Pessoa
>>   def self.pessoas_vivas
>>     999
>>   end
>> end
>> Pessoa.pessoas_vivas
```

ruby/ Orientação a Objetos

Levando o dinamismo do Ruby ao extremo, podemos criar métodos que definem métodos em outros objetos.

```
>> class Aluno
>>   # Não sabe nada por enquanto
>> end
>> class Professor
>>   def ensina(aluno)
>>     def aluno.escreve
>>       "Agora eu sei escrever."
>>     end
>>   end
>> end
```

ruby/ Orientação a Objetos

```
>> felipe = Aluno.new
>> felipe.respond_to?(:escreve)
>>
>> professor = Professor.new
>> professor.ensina(felipe)
>>
>> felipe.escreve
```

Herança

Herança é quando uma classe FILHA, herda métodos e atributos públicos da classe PAI.

No Ruby só existe herança simples.

1 filho só pode ter 1 pai.

```
>> class Filho < Pai
..
>> end
```

ruby/ Orientação a Objetos

atributos e métodos publicos?????

uma classe pode ter métodos e atributos públicos e privados

sim... tem outros também, mas ta bom por enquanto.

público:
os filhos herdam
acessível de fora da classe

privado:
os filhos Ñ herdam
Ñ acessível de fora da classe

Ruby / Blocos

E Programação funcional

Blocos

Qualquer método pode receber um bloco de código.

```
>> def meu_metodo
>>   puts “Olá”
>> end
>>
>> meu_metodo { puts “bloco de codigo” }
```

Blocos

É possível passar um bloco de código com várias linhas, utilizando do e end

```
>> def meu_metodo
>>   puts “Olá”
>> end
>>
>> meu_metodo do
>>   puts “Estou aqui dentro do”
>>   puts “bloco de codigo”
>> end
```

Blocos

Você pode decidir se executa ou não o bloco de código, utilizando a palavra reservada yield

```
>> def meu_metodo
>>   puts “Olá”
>>   yield
>> end
>>
>> meu_metodo do
>>   puts “Estou dentro do bloco”
>> end
```

Blocos

Todo vetor possui um método chamado each, que chama um bloco de código associado para cada um dos seus itens:

```
>> vetor = [1, 2, 3, 4, 5]
>> vetor.each do |item|
>>   puts item
>> end
```

Ruby / Exceptions

Exceptions

Tratamento de Exceptions

Uma exception é um tipo especial de objeto que extende ou é instância da classe Exception. Lançar uma exception significa que algo não esperado ou errado aconteceu no fluxo do programa.

Exceptions

Manipulando Exceptions

Crie um arquivo .rb com o conteúdo abaixo:

```
print “Digite um número:”
numero = gets.to_i

begin
  resultado = 100 / numero
rescue
  puts “O número digitado é inválido.”
  exit
end
puts “100/#{numero} é #{resultado}”
```

Exceptions

Manipulando Exceptions

Você pode criar suas próprias Exceptions:

```
>> class MinhaException < Exception
>> end
>>
>> begin
>>   puts “Lançando MinhaException...”
>>   raise MinhaException
>> rescue MinhaException => e
>>   puts “Foi lançada a exception: #{e}”
>> end
```

Exercício:

- *Faça uma classe para cadastramos produtos.*
  - *Cada produto tem um nome, um fabricante e um preço*
  - *Salvo isso em variáveis mesmo...*
- *Temos uma classe de venda que recebe um produto por parâmetro*
  - *depois dizemos a quantidade*
  - *e ela nos díz o total da venda!*

Desafio Ruby

Nosso super sistema será o seguinte:

- *Haverá fábricas de automóveis:*
  - *Cada fábrica possui uma marca;*
  - *A partir da fábrica, posso construir carros e motos;*
  - *A fábrica sabe todos os automóveis que já construiu;*
- *Posso construir Carros e Motos:*
  - *Ambos são automóveis;*
    - *Todo automóvel possui um nome:*
    - *O nome do carro é sempre a marca de sua fábrica + o seu nome;*
    - *Pode ser ligado e desligado;*
    - *Pode andar se estiver ligado;*
    - *O automóvel sabe quem é a sua fábrica;*
  - *Os carros tem 4 rodas*
  - *Os motos tem somente 2 rodas*

Desafio Ruby

Testando:

1) Vou criar um fabrica com um nome:

```
>> gm = Fabrica.new('GM')
```

2) Vou criar um carro dessa fábrica:

```
>> carro = gm.novo_carro('Corsa')
```

3)

```
>> carro.nome
```

deve ser 'GM Corsa'

5) não posso andar sem ligar o carro

```
>> carro.anda
```

6) ligo o carro:

```
>> carro.liga
```

7) agora posso andar:

```
>> carro.anda
```

8) Faço tudo denovo com uma moto.....

9) Mas a moto tem só 2 rodas:

```
>> moto.rodas
```

10) A fábrica sabe o carro e a moto que criamos:

```
>> gm.automoveis_construidos
```

Rails.

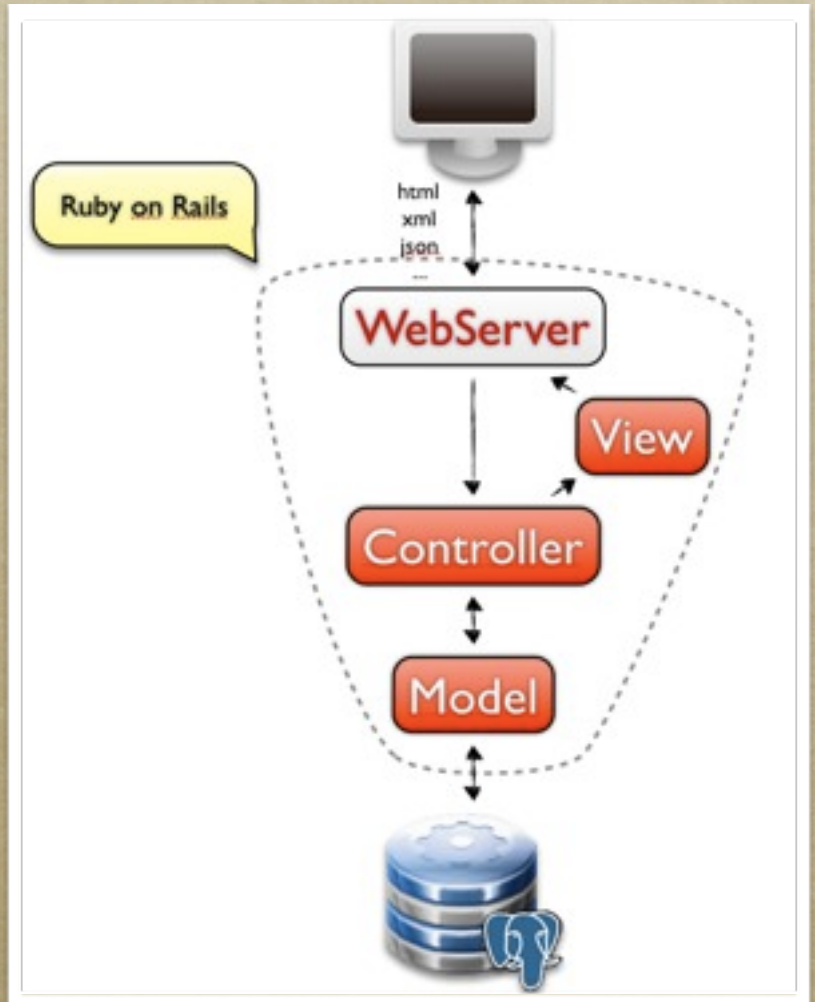
Pilares

- *Don't Repeat Yourself. (DRY)*
  - *Não deve existir CtrlC + CtrlV*
  - *Scopes.*
  - *Filters.*
  - *Callbacks.*
- *Convention Over Configuration.*
  - *O Rails tem seus padrões, se vc deseja algo diferente, só então vc configura.*

MVC

- *Models.*
  - *Mantêm os dados.*
  - *Regras do negócio.*
- *Views.*
  - *Interface.*
- *Controllers.*
  - *Liga tudo.*
  - *Fluxo*

MVC


Ruby on Rails
html
xml
json
...
WebServer
View
Controller
Model

Mão na Massa!

chega de balela.

Mão na Massa!

Para Instalar o Rails:

1 - Instalar o GEMS

site oficial ⟶ *download* ⟶ *RubyGems* ⟶ *Dowload*

Versão 1.6.2!

Link direto:

http://rubyforge.org/frs/download.php/74446/rubygems-1.6.2.zip

descompactar e rodar:

```
ruby setub.rb
```

* *instalação para windows*

Mão na Massa!

Para Instalar o Rails:

2 - Instalar o Rails

```
gem install rails --no-ri --no-rdoc
```

rails/

Mão na Massa!

Crie uma nova aplicação:

```
rails new helloworld
```

Estrutura de diretórios:

```
cd helloworld
```

```
dir
```

Hello World

Em app/controller crie o arquivo:

```
hello_controller.rb
```

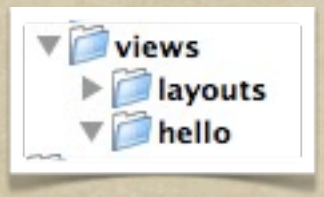
com o seguinte conteudo:

```ruby
class HelloController < ApplicationController
    def world
    end
end
```

rails/

Hello World

Em app/views crie o diretório 'hello'



Em app/views/hello crie o arquivo:

world.html.erb

Com o conteúdo:

```
<div align='center'>
    <p><h1>Hello World!</h1></p>
    <p> Minha primeria página no rails funcionou! </p>
</div>
```

Será que funciona????

rails/

Hello World

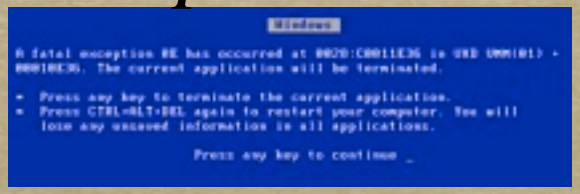
Para testar, precisamos iniciar o servidor.

```
rails server
```

Testando:

http://localhost:3000/hello/world

pãã̃ã!



Rotas

- *As rotas ligam a URL digitada a uma ação e um controlador específicos da nossa aplicação.*
  - *É possível configurar as rotas de diversas maneiras, vamos começar pelo básico:*

no arquivo: `config/routes.rb` *temos a configuração das rotas*

descomente a linha:

```
# match ':controller(/:action(/:id(.:format)))'
```

```
match ':controller(/:action(/:id(.:format)))'
```

rails/

Hello World

Será que funciona????

http://localhost:3000/hello/world

Hello World!

Minha primeria página no rails funcionou!

uhuuul! \o/

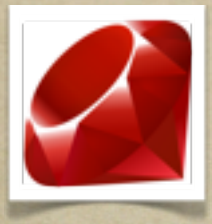
Brincando um pouco

variáveis entre o controller e a view:

```
# em app/controllers/hello_controller.rb
def world
    @variavel_de_instancia = 'Este texto irá aparecer na view.'
    variavel_de_bloco = 'Esta variável vai dar pau!'
end
```

na view (apps/views/hello/world.html.erb)

```
<div align='center'>
    <p><h1>Hello World!</h1></p>
    <p>Variável da instancia: <%= @variavel_de_instancia %></p>
    <p>Variável do bloco: <%= variavel_de_bloco %></p>
</div>
```



rails/ Brincando um pouco

Será que funciona????

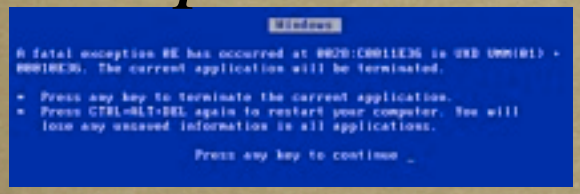
http://localhost:3000/hello/world

caracter inválido!

SyntaxError in HelloController#world

```
/SGI/desenv/helloworld/app/controllers/hello_controller.rb:3: invalid multibyte char (US-ASCII)
/SGI/desenv/helloworld/app/controllers/hello_controller.rb:3: invalid multibyte char (US-ASCII)
/SGI/desenv/helloworld/app/controllers/hello_controller.rb:3: syntax error, unexpected $end, expecting keyword_end
                @variavel_de_instancia = 'Este texto irÃ¡ aparecer na view.'
                                                                          ^
```

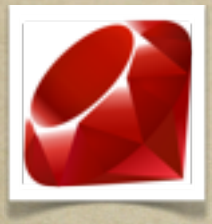
pãã̃ã!



- *O ruby 1.9.2 exige que a codificação dos arquivos fontes seja declarada!*

coloque: `# encoding: UTF-8` *no inicio do controller.*

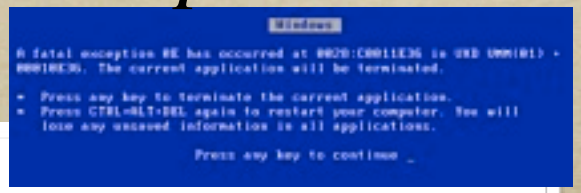
veja a codificaçào que seu editor utiliza



rails/ Brincando um pouco

Será que funciona???? vamo lá denovo.

pãã!



http://localhost:3000/hello/world

NameError in Hello#world

Showing */SGI/desenv/helloworld/app/views/hello/world.html.erb* where line **#5** raised:

```
undefined local variable or method `variavel_de_bloco' for #<#<Class:0x0000010324c030>:0x00000103249538>
```

variavel não existe

na view (apps/views/hello/world.html.erb)

```
<div align='center'>
    <p><h1>Hello World!</h1></p>
    <p>Variável da instancia: <%= @variavel_de_instancia %></p>
</div>
```

rails/ Brincando um pouco

agora vai!

http://localhost:3000/hello/world

Hello World!

Variável da instancia: Este texto irá aparecer na view.

uhuuul! \o/

rails/ Brincando um pouco

mais coisas:

```
# em app/controllers/hello_controller.rb
@alfabeto = ‘a’..’z’
@estrutura = {:chave => 'valor'}
@pessoas = [
    {:nome => 'João', :idade => 15},
    {:nome => 'José', :idade => 20},
    {:nome => 'Maria', :idade => 18}]
```

Brincando um pouco

mais coisas:

na view (apps/views/hello/world.html.erb)

```
<div><p><h1>Hello World!</h1></p>
    <p><b>Eu sei o alfabeto!:</b><br>
        <% @alfabeto.each do |letra| %>
            <%= letra %> ,
        <% end %>
    </p>
    <p><b>Estrutura:</b><br>
        Chave: <i><%= @estrutura[:chave] %>.</i>
    </p>
    <p><b> Lista de Pessoas:</b><br>
        <% @pessoas.each do |pessoa| %>
            <%= pessoa[:nome] %> tem <%= pessoa[:idade] %> anos! <br>
        <% end %>
    </p>
</div>
```

Partials.

Partials

- *Partials são pedaços de telas que podem ser aproveitados em várias views.*
  - *Tem o objetivo de eliminar códigos duplicados nas views.*

vamos criar um novo controller: pessoas_controller.rb

Com duas ações:

```
# encoding: UTF-8
class PessoasController < ApplicationController
    def criancas
    end

    def adultos
    end
end
```

Partials

Crianças:

em apps/controllers/pessoas_controller.rb

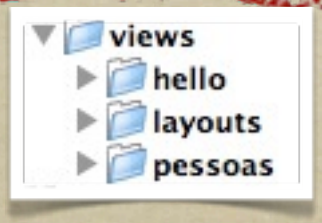
```
@criancas = [
    {:nome => 'Zéquinha', :idade => 7},
    {:nome => 'Pedrinho', :idade => 6}
]
```

Adultos:

```
@adultos = [
    {:nome => 'Marcos', :idade => 30},
    {:nome => 'Pedro', :idade => 35}
]
```

Partials

criar a pasta 'pessoas' em app/views:



em app/views/pessoas/criancas.html.erb:

```
<p><h1>Lista de Crianças </h1></p>
<%= render :partial => 'lista_de_pessoas', :locals =>
{:pessoas => @criancas} %>
```

em app/views/pessoas/adultos.html.erb:

```
<p><h1>Lista de Adultos</h1></p>
<%= render :partial => 'lista_de_pessoas', :locals =>
{:pessoas => @adultos} %>
```

Partials

criar o partial: `_lista_de_pessoas.html.erb` *em app/views/pessoas/*

SEMPRE começa com _

```
<table>
    <tr>
        <th>Nome</th>
        <th>Idade</th>
    </tr>
    <% pessoas.each do |pessoa| %>
        <tr>
            <td><%= pessoa[:nome] %></td>
            <td><%= pessoa[:idade] %> anos</td>
        </tr>
    <% end %>
</table>
```

Espera este parâmetro

Partials

Testando:

http://localhost:3000/pessoas/ciancas

Lista de Crianças

Nome	Idade
Zéquinha	7 anos
Pedrinho	6 anos

http://localhost:3000/pessoas/adultos

Lista de Adultos

Nome	Idade
Marcos	30 anos
Pedro	35 anos

uhuuul! \o/

rails/

Exercício!

1) Crie um Cabeçalho e um Rodapé padrão para as nossas 2 telas anteriores

10 minutos.

Este é o cabeçalho das nossas páginas.
Coloque alguma coisa útil aqui.

Lista de Crianças

Nome	Idade
Zéquinha	7 anos
Pedrinho	6 anos

Desenvolvido por EU, durante o curso de RoR da Unoesc. Todos os direitos reservados.

Este é o cabeçalho das nossas páginas.
sa útil aqui.

Adultos

U, durante o curso de RoR da
reitos reservados.

Dica: Voce pode colocar o partial em outro diretório e no render usar 'diretorio/partial'

Layouts.

Layouts

- *é um HTML padrão para todas as view do nossa aplicação.*
  - *Cada controlador pode ter seu layout.*
  - *Existe o layout padrão*
  - *Todos os layouts ficam no diretório app/views/layouts/*
    - *para layouts específicos, é só criar um arquivo CONTROLADOR.html.erb neste diretóio*

Layouts

Alterando o layout padrão:

em apps/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title>Helloworld</title>
  <%= stylesheet_link_tag :all %>
  <%= javascript_include_tag :defaults %>
  <%= csrf_meta_tag %>
</head>
<body>
<%= yield %>
</body>
</html>
```

Inclusão dos JS e dos CSS necessários

aqui cada view será renderizada

Layouts

Alterando o layout padrão:

deixamos o body assim:

```
<body style='margin:0px; background-color:#600;'  >
<table  width='100%' height='100%' style='border-spacing: 0;'>
    <tr><td> <%= render :partial => '/layouts/cabecalho' %> </td></t
    <tr><td> <%= render :partial => '/layouts/menu' %> </td></tr>
    <tr><td><div style='background-color:white;
                        margin:0px 15px 0px 15px; padding:4px;
                        border-radius: 4px;-moz-border-radius: 5px;'
        <%= yield %>
    </div></td></tr>
    <tr><td><%= render :partial => '/layouts/rodape' %></td></tr>
</table>
</body>
```

Layouts

em app/views/layouts/_cabecalho.html.erb

```
<div style='color:white; font-size:30px; padding: 10px 10px 5px 0px;
        text-shadow: 1px 1px 4px #A1A1A1;' align='right'>
    Minha aplicação em Ruby on Rails.
</div>
```

em app/views/layouts/_rodape.html.erb

```
<div style='color:gray;' align='center'>
    Desenvolvido por Rudiney, todos os direitos reservados.
</div>
```

coloque seu nome

Links

Criando um link:

em app/views/layouts/_menu.html.erb

```
<div style='background-color:white; margin:0px 15px 0px 15px;
        padding:4px; border-radius: 4px;-moz-border-radius: 5px;'>

    <%= link_to 'HelloWorld!', '/hello/world' %> |
    <%= link_to 'Criancas', '/pessoas/criancas' %> |
    <%= link_to 'Adultos', '/pessoas/adultos' %> |
</div>
```

Documentação completa das Views:

api.rubyonrails.org/classes/ActionView/Base.html

Layouts

Ficou mais bonito???

http://localhost:3000/pessoas/ciancas

http://localhost:3000/pessoas/adultos

Todas as páginas devem ter o novo layout!

http://localhost:3000/hello/world

Exercício!

30 minutos.

1) Nova App.

2) Lista de cidades

2.1) Quero digitar cidades/santa_catarina

2.2) Quero digitar cidades/parana

3) Quero uma lista de cidades e seus habitantes

4) Link para alterar entre cidades de SC e PR

Exercício!

5) Quero assim:

tá bom, podem copiar o layout do exemplo anterior....

Banco de dados.

tá começando a fica interessante....

rails/

Banco de Dados

- *Configuração:*

```
config/databases.yml
```

usaremos o sqlite.

  - *Manipulado pelos modelos.*

```
app/models
```

    - *Versionado pelas migrações.*

```
db/migrate
```

Migração

- *Nossa primeira tabela:*

configura o banco de dados

```
config/databases.yml
```

crie o banco

```
rake db:create
```

gerando uma nova migração

```
rails generate migration cria_tabela_cidades
```

Migração

- *Nossa primeira tabela:*

  criando a tabela pela migração

  toda alteração no banco deve ser feita em uma migração!

```
db/migrate/XXXXXXXXcria_tabela_cidades.rb
```

Documentação das Migrações:

http://api.rubyonrails.org/classes/ActiveRecord/Migration.html

Migração

Conteúdo da migração:

isso faz tudo

```
class CriaTabelaCidades < ActiveRecord::Migration
  def up
    create_table :cidades do |t|
      t.string :nome,      :limit => 20, :null => false
      t.string :estado,    :limit => 2,  :null => false
      t.integer :habitantes
      t.timestamps
    end
  end
  def down
    drop_table :cidades
  end
end
```

Migração

Rodando a migração

```
rake db:migrate
```

Pronto! a tabela esta no banco.

qué aposta?

```
sqlite3 db/development.sqlite3
```

comandos no sqlite3...

```
.tables
```

lista as tabelas

```
.mode line
```

sai...

```
.exit
```

Modelo

- *Criando o Modelo:*

```
em app/models/cidade.rb
```

Modelos sempre no singular!

```
class Cidade < ActiveRecord::Base
end
```

Esse é O cara.

é tudo o que precisamos para manipular os dados

Documentação dos Modelos:

http://api.rubyonrails.org/classes/ActiveRecord/Base.html

Modelo

- *Brincando com o modelo*

```
rails console
```

```
c = Cidade.new
```

```
c.save
```

```
Cidade
```

- *Vendo pelo banco*

```
sqlite3 db/development.sqlite3
```

```
select * from cidades;
```

rails/

Cadastro.

- *Criando uma interface para o cadastro*

Controller de cidades

Ação ***index***

```
def index
  @cidades = Cidade.all
end
```

view index..... listar as cidades

tcharam!

Cadastro.

- *Inserindo uma cidade*

Ação ***new***

```
def new
  @produto = Cidade.new
end
```

Tudo sobre formularios.....

http://api.rubyonrails.org/classes/ActionView/Helpers/FormHelper.html

tcharam!

Cadastro.

- *Salvando a cidade*

Ação ***create***

```
def create
  @cidade = Cidade.new
  @cidade.nome = params[:cidade][:nome]
  @cidade.estado = params[:cidade][:estado]
  @cidade.habitantes = params[:cidade][:habitantes]
  @cidade.save
  redirect_to :action => :index
end
```

Cadastro.

- *Deletando a cidade na view* ***index:***

```
<td>
  <%= link_to({:action => :destroy, :id => cidade.id},
  :confirm => "Você tem certeza?") do %>
      <%= image_tag 'close.png' %>
  <% end %>
</td>
```

Ação ***destroy***

```
def destroy
  @cidade = Cidade.find(params[:id])
  @cidade.destroy
  redirect_to :action => :index
end
```

Cadastro.

- *Editando a cidade na view **index:***

```
<td>
  <%= link_to(:action => :edit, :id => cidade.id) do %>
     <%= image_tag 'page_edit.png' %>
  <% end %>
</td>
```

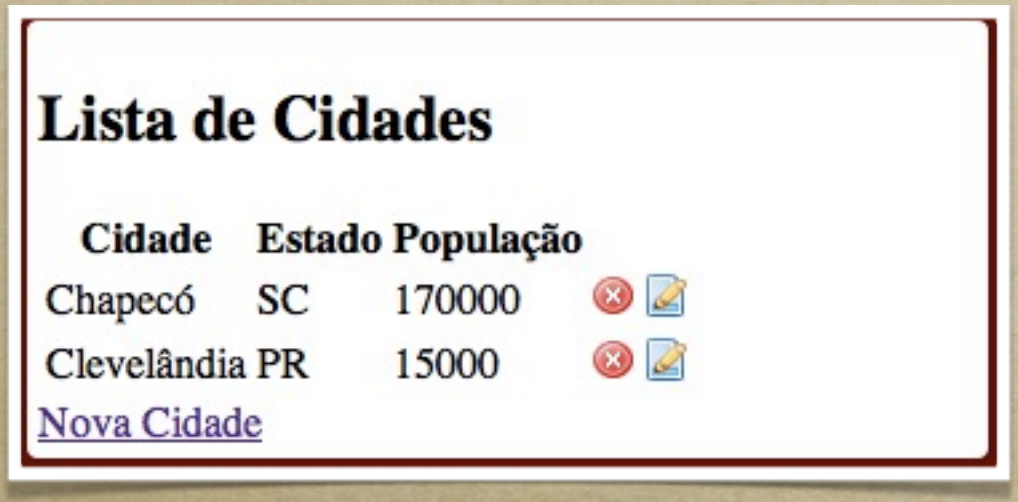
Ações

```
def edit
  @cidade = Cidade.find(params[:id])
end
```

```
def update
  @cidade = Cidade.find(params[:id])
  @cidade.update_attributes(params[:cidade])
  redirect_to :action => :index
end
```

Cadastro.

Temos um Cadastro Completo.



Exercício!

30 minutos.

1) Crie Agora o Cadastro de Estados com 2 informações: Sigla e Nome

2) Faça todas as operações que fizemos nas cidades

rails/

Estragando a diversão

o scaffold e os geradores

rails/

Geradores.

- *O Rails possue script para gerar códigos. Os chamados geradores*

tente executar o seguinte comando:

```
rails generate scaffold pessoa nome:text idade:integer
```

- *Irá gerar tudo o que criamos até agora para um cadastro e um pouco mais....*

Você pode criar seu próprio gerador veja:

http://guides.rubyonrails.org/generators.html

Como estamos aprendendo, não vamos utilizar nenhum gerador!

rails/

Validações

Validações

- *Servem para não deixar existir registros inválidos no banco de dados.*
  - *Seguindo o MVC, as regras de negócio devem ficar nos modelos:*

Validações padrões disponíveis:

http://api.rubyonrails.org/classes/ActiveModel/Validations/HelperMethods.html

Tudo sobre validações:

http://api.rubyonrails.org/classes/ActiveModel/Validations.html

Validações

- *Validando nosso cadastro:*

O nome é obrigatório e deve ter no máximo 20 letras:

```
validates :nome, presence => {:message => "Deve ser Preenchido"},
:length ={:maximum => 20, :message => 'Deve ter no máximo 20 letras'}
```

O estado é obrigatório e deve ter no máximo 2 letras:

```
validates :estado,  :presence => {:message => 'Deve ser Preenchido'},
length => {:maximum => 2, :message => 'Deve ter no máximo 2 letras'}
```

O Número de habitantes deve ser um numero:

```
validates :habitantes, :numericality => {
    :message => 'Deve ser um Número'}
```

Validações

- *Testando...*

```
rails console
```

crie uma cidade....

coloque valores inválidos...

```
cidade.save
=> false
```

O save não salva!

```
cidade.errors
```

Podemos obter todos os erros do registro

os ERRORS tem a seguinte estrutura:

```
:atributo => [
  'Primeiro erro',
  'Segundo Erro'
],
:outro_atributo ...
```

Validações

- *Exibindo os erros no nosso cadastro*

na ação create:

```
if @cidade.save
  redirect_to :action => :index
else
  render :action => 'new'
end
```

no formulário:

```
<% if @cidade.errors.any? %>
<div style='color:red;'>
Não foi possível salvar a cidade por <%= pluralize
(@cidade.errors.size, 'motivo') %>:
  <ul>
    <% @cidade.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
  </ul>
</div>
<% end %>
```

Validações

- *Exibindo os erros no nosso cadastro*

Nova Cidade

Não foi possível salvar a cidade por 2 motivos:

- Nome Deve ser Preenchido
- Estado Deve ser Preenchido

Nome:

Nova Cidade

Não foi possível salvar a cidade por 2 motivos:

- Nome Deve ter no máximo 20 letras
- Estado Deve ter no máximo 2 letras

Nome:

123456789012345678901234

Validações

- *Validações customizadas:*

```
validate :valida_se_estado_existe
private
def valida_se_estado_existe
  estados_existente = %w{ RS SC PR SP RJ MG } #.....
  if !estados_existente.include?(self.estado.upcase)
    self.errors.add(:estado, 'Não Existe')
  end
end
```

Nova Cidade

Não foi possível salvar a cidade por 1 motivo:

- Estado Não Existe

Nome:

Estado não existe

Estado:

DS

Validações

- *Mensagens de sucesso:*

No create e no update:

```
if @cidade.save # ou update_attributes
  flash[:notice] = "Cidade #{@cidade.nome} criada com sucesso!"
```

No Index:

```
<div style='color:green'>
  <%= flash[:notice] %>
</div>
```

Lista de Cidades

Cidade Porto Alegre criada com sucesso!

Cidade	Estado	População
Chapecó	SC	170000
Clevelândia	PR	15000
Porto Alegre	RS	2145345

uhuuul! \o/

Exercício!

1) Validações nos cadastro de estado criado anteriormente.

20 minutos.

A Sigla e o Nome devem ser obrigatórios.

A Sigla deve ter no máximo 2 letras.

DESAFIO: não pode haver 2 estados com a mesma sigla!

dica: antes de fazer uma validação, veja a documentação.

2) Mostre todas as mensagem como fizemos no cadastro de cidade.

rails/

Associações

Associações

- *O Rails abstrai os relacionamentos entre o Modelos*

N - N

1 - 1

1 -N

Tudo sobre associações:

http://api.rubyonrails.org/classes/ActiveRecord/Associations.html

Guia das Associações:

http://guides.rubyonrails.org/association_basics.html

Associações

- *Primeiro vamos renomear a coluna estado da cidade:*

```
rails g migration altera_coluna_estado_na_cidade
```

```ruby
def up
  rename_column :cidades, :estado, :estado_id
end
def down
  rename_column :cidades, :estado_id, :estado
end
```

padrão do rails

```
rake db:migrate
```

Associações

- *Uma cidade pertence a um estado:*

```
class Cidade < ActiveRecord::Base
    belongs_to :estado
```

- *Uma estado tem várias cidades:*

```
class Estado < ActiveRecord::Base
    has_many :cidades
```

testando:

```
rails console
```

```
cidade.estado
```

```
estado.cidades
```

Exercício!

Este é para fazer em casa.

1) Uma nova aplicação.

2) Deve haver um cadastro de restaurantes.

3) Um cadastro de pratos (lasanha, filé, sopa...)

Um restaurante pode ter vários pratos, e um prato pode ser servido em muitos restaurantes!

dica: nunca esqueça da documentação

dica2: nunca esqueça dos guias

faça uma tela onde é exibido o cardápio de cada um dos restaurantes.

tá bom, pode usar o scaffold!