



## Ruby Mola (y por qué)

Sergio Gil

Conferencia Rails  
noviembre de 2006

# Ruby

- Simple
- Completo
- Orientado a objetos
- Dinámico



## Ruby y Ruby on Rails

- Rails aprovecha al máximo la potencia y el dinamismo de Ruby
  - DHH opina que no hubiera podido crear Rails con otro lenguaje.
- 
- Y sin embargo, está **infrautilizado**

1 Un lenguaje muy *dulce*

1 Un lenguaje muy *dulce*

2 '*De serie*'

1 Un lenguaje muy *dulce*

2 '*De serie*'

3 Orientación a objetos

1 Un lenguaje muy *dulce*

2 '*De serie*'

3 Orientación a objetos

4 Un lenguaje dinámico

- 1 Un lenguaje muy *dulce*
- 2 '*De serie*'
- 3 Orientación a objetos
- 4 Un lenguaje dinámico

*“Quien no ha probado lo  
amargo, no sabe lo que es  
dulce”*

PROVERBIO ALEMÁN



- Los idiomas influyen (¿determinan?) la manera de pensar de las personas
- Los lenguajes, la de los programadores

## Ruby

- Cercano a nuestra forma de pensar
  - Lo cual da una experiencia mejor
  - ¡Es divertido!
  - Te hace sentir inteligente
- 
- *'Ruby stays out of your way'*

Algunos ejemplos

Las estructuras **devuelven** el valor de **la última expresión evaluada**:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

Las estructuras **devuelven** el valor de la última expresión evaluada:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

Las estructuras **devuelven** el valor de la última expresión evaluada:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

Las estructuras **devuelven** el valor de la última expresión evaluada:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

- 1 Menos código

Las estructuras **devuelven** el valor de la última expresión evaluada:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

- 1 Menos código
- 2 Menos errores

Las estructuras **devuelven** el valor de la última expresión evaluada:

```
@user = if params[:id]
  User.find(params[:id])
else
  User.new
end
```

- 1 Menos código
- 2 Menos errores
- 3 ¡Más bonito!

Las asignaciones también **devuelven** el valor asignado:

```
@post = Post.find(params[:id])  
if @post  
  @post.destroy  
end
```

Las asignaciones también **devuelven** el valor asignado:

```
@post = Post.find(params[:id])  
if @post  
  @post.destroy  
end
```

Mejor:

```
if @post = Post.find(params[:id])  
  @post.destroy  
end
```

Las asignaciones también **devuelven** el valor asignado:

```
@post = Post.find(params[:id])  
if @post  
  @post.destroy  
end
```

Mejor:

```
if @post = Post.find(params[:id])  
  @post.destroy  
end
```

Las asignaciones también **devuelven** el valor asignado:

```
@post = Post.find(params[:id])  
if @post  
  @post.destroy  
end
```

Mejor:

```
if @post = Post.find(params[:id])  
  @post.destroy  
end
```

- Y además nos repetimos menos

## Asignaciones condicionales

## Asignaciones condicionales

```
@title = "Título genérico" unless defined?(@title)
```

## Asignaciones condicionales

```
@title = "Título genérico" unless defined?(@title)
```

```
▶ @title ||= "Título genérico"
```

## Asignaciones condicionales

```
@title = "Título genérico" unless defined?(@title)
```

```
▶ @title ||= "Título genérico"
```

```
@heading = if defined?(@subsection)
```

```
  @subsection.title
```

```
else
```

```
  @section.title
```

```
end
```

## Asignaciones condicionales

```
@title = "Título genérico" unless defined?(@title)
```

```
▶ @title ||= "Título genérico"
```

```
@heading = if defined?(@subsection)
```

```
  @subsection.title
```

```
else
```

```
  @section.title
```

```
end
```

```
▶ @heading = ( @subsection || @section ).title
```

## Asignaciones condicionales

```
@title = "Título genérico" unless defined?(@title)
```

```
▶ @title ||= "Título genérico"
```

```
@heading = if defined?(@subsection)
```

```
  @subsection.title
```

```
else
```

```
  @section.title
```

```
end
```

```
▶ @heading = ( @subsection || @section ).title
```

- 1 Un lenguaje muy *dulce*
- 2 *'De serie'*
- 3 Orientación a objetos
- 4 Un lenguaje dinámico

*“Cualquier tecnología  
distinguible de la magia es  
insuficientemente avanzada”*

CLEÓN I



- La clase `Array`<sup>1</sup> es una de las joyas de las clases incluidas
- Le daremos un repaso para ejemplificar:
  - La potencia de las clases incluidas
  - La potencia y el uso de los bloques
  - El *'estilo Ruby'*

---

<sup>1</sup>Sin olvidar el módulo `Enumerable`

```
@arr = [ 1, 2, 3, 4, 5 ]
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

Que es el for de toda la vida:

```
for item in @arr  
  puts item  
end
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

Que es el for de toda la vida:

```
for item in @arr  
  puts item  
end
```

```
@arr = [ 1, 2, 3, 4, 5 ]
```

Iterando:

```
@arr.each do |item|  
  puts item  
end
```

Que es el for de toda la vida:

```
for item in @arr  
  puts item  
end
```

Lo bueno empieza ahora:

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`

```
[ 1, 1, 2, 3, 3 ].uniq  
>> [ 1, 2, 3 ]
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`

```
[ [ 1, 1 ], 2, [ 3, 3 ] ].flatten  
>> [ 1, 1, 2, 3, 3 ]
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`
- `.compact`

```
[ 1, nil, 2, nil, 3 ].compact  
>> [ 1, 2, 3 ]
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`
- `.compact`
- `.reverse`

```
[ 1, 1, 2, 3, 3 ].reverse  
>> [ 3, 3, 2, 1, 1 ]
```

Lo bueno empieza ahora:

## Utilidades varias

- .uniq
- .flatten
- .compact
- .reverse
- .sort

```
[ 1, 3, 2, 1, 3 ].sort  
>> [ 1, 1, 2, 3, 3 ]
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`
- `.compact`
- `.reverse`
- `.sort`

## Búsqueda

- `.find`

```
[ 1, 2, 3 ].find {|n| n % 3 == 0 }  
>> 3  
[ 1, 2, 3 ].find {|n| n % 5 == 0 }  
>> nil
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`
- `.compact`
- `.reverse`
- `.sort`

## Búsqueda

- `.find`
- `.find_all`

```
[ 1, 3, 5, 6 ].find_all { |n| n % 3 == 0 }  
>> [ 3, 6 ]  
[ 1, 2, 3 ].find_all { |n| n % 5 == 0 }  
>> []
```

Lo bueno empieza ahora:

## Utilidades varias

- `.uniq`
- `.flatten`
- `.compact`
- `.reverse`
- `.sort`

## Búsqueda

- `.find`
- `.find_all`

```
[ 1, 3, 5, 6 ].reject { |n| n % 3 == 0 }  
>> [ 1, 5 ]  
[ 1, 2, 3 ].reject { |n| n % 5 == 0 }  
>> [ 1, 2, 3 ]
```

## Operaciones

- 'Suma': +

```
[ 1, 2 ] + [ 3, 4 ]  
>> [ 1, 2, 3, 4 ]
```

## Operaciones

- *'Suma'*: +
- *'Resta'*: -

```
[ 1, 2, 3, 4 ] - [ 2, 3 ]  
>> [ 1, 4 ]
```

## Operaciones

- *'Suma'*: +
- *'Resta'*: -
- *'Repetición'*: \*

```
[ 1, 2 ] * 3  
>> [ 1, 2, 1, 2, 1, 2 ]
```

## Operaciones

- 'Suma': +
- 'Resta': -
- 'Repetición': \*
- Intersección ( $\cap$ ): &

```
[ 0, 1, 2, 3, 4 ] & [ 0, 3, 6 ]  
>> [ 0, 3 ]
```

## Operaciones

- 'Suma': +
- 'Resta': -
- 'Repetición': \*
- Intersección ( $\cap$ ): &
- Unión ( $\cup$ ): |

```
[ 1, 2, 3 ] | [ 3, 4, 5 ]  
>> [ 1, 2, 3, 4, 5 ]
```

Nunca más esto:

```
@precios = ...  
@ivas = []  
@precios.each do |precio|  
  @ivas << precio * 0.16  
end
```

Nunca más esto:

```
@precios = ...  
@ivas = []  
@precios.each do |precio|  
  @ivas << precio * 0.16  
end
```

¿No es mejor esto?

```
@ivas = @precios.map {|precio| precio * 0.16 }
```

Por si hace falta otro ejemplo:

```
@precios = ...  
@total = 0  
@precios.each do |precio|  
  @total += precio  
end
```

Por si hace falta otro ejemplo:

```
@precios = ...  
@total = 0  
@precios.each do |precio|  
  @total += precio  
end
```

¿No es más bonito así?

```
@total = @precios.inject {|t, precio| t + precio }
```

¿Y en **Rails**? Algunos ejemplos chulos:

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

```
@last_comments = Comment.find(:all, :order => 'date DESC',  
                             :limit => 10)
```

```
@last_commented_posts = @last_comments.map { |c| c.post }.uniq
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

```
@last_comments = Comment.find(:all, :order => 'date DESC',  
                             :limit => 10)
```

```
@last_commented_posts = @last_comments.map { |c| c.post }.uniq
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

```
@last_comments = Comment.find(:all, :order => 'date DESC',  
                             :limit => 10)
```

```
@last_commented_posts = @last_comments.map { |c| c.post }.uniq
```

```
<p>Comentado por: <%= @post.comments.sort[0..4].map { |c|  
    link_to(c.user.name, c.permlink) }.join(', ') %></p>
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort

@last_comments = Comment.find(:all, :order => 'date DESC',
                               :limit => 10)

@last_commented_posts = @last_comments.map { |c| c.post }.uniq

<p>Comentado por: <%= @post.comments.sort[0..4].map { |c|
  link_to(c.user.name, c.permlink) }.join(', ') %></p>
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

```
@last_comments = Comment.find(:all, :order => 'date DESC',  
                             :limit => 10)
```

```
@last_commented_posts = @last_comments.map { |c| c.post }.uniq
```

```
<p>Comentado por: <%= @post.comments.sort[0..4].map { |c|  
    link_to(c.user.name, c.permlink) }.join(', ') %></p>
```

¿Y en **Rails**? Algunos ejemplos chulos:

```
@comments = @posts.map { |p| p.comments }.flatten.sort
```

```
@last_comments = Comment.find(:all, :order => 'date DESC',  
                             :limit => 10)
```

```
@last_commented_posts = @last_comments.map { |c| c.post }.uniq
```

```
<p>Comentado por: <%= @post.comments.sort[0..4].map { |c|  
    link_to(c.user.name, c.permlink) }.join(', ') %></p>
```

**Merece la pena** conocer bien las clases incluidas

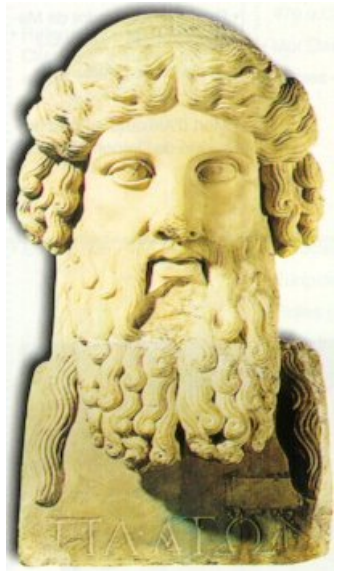
## Algunas clases y módulos que no perder de vista

- Array
- Enumerable
- Hash
- Numeric
- Range
- Regexp
- String
- Time

- 1 Un lenguaje muy *dulce*
- 2 '*De serie*'
- 3 Orientación a objetos**
- 4 Un lenguaje dinámico

*“Los prisioneros de la caverna sólo ven las sombras de los objetos.”*

PLATÓN



En Ruby:

- Todo es un objeto
- Todo
- De verdad, todo

En serio, todo:

- No hay *'tipos primitivos'*
- Los números son objetos `Fixnum` (o `Float`, o...)
- Las cadenas son objetos `String`
- Incluso las clases son objetos `Class`<sup>2</sup>

---

<sup>2</sup>Lo cual permite técnicas avanzadas que no veremos hoy =;-)

De toda la vida:

```
def sanitize_title(title)
  sanitized_title = ...
  return sanitized_title
end
```

...

```
post.title = params[:title]
post.sanitized_title = sanitize_title(post.title)
```

Más bonito y más *DRY*:

```
class Post < ActiveRecord::Base
  def sanitized_title
    ...
  end
end
...
post.title = params[:title]
post.sanitized_title
```

Mejor aún:

```
class String
  def sanitize
    ...
  end
end
...
post.title = params[:title]
post.title.sanitize
```

- 1 Un lenguaje muy *dulce*
- 2 '*De serie*'
- 3 Orientación a objetos
- 4 Un lenguaje dinámico

*“Un hombre no puede bañarse dos veces en el mismo río; ni el río ni el hombre son los mismos, puesto que todo fluye.”*

HERÁCLITO



## En Ruby:

- No hay diferencia entre *'tiempo de compilación'* y *'tiempo de ejecución'*
- Se pueden crear y modificar clases, redefinir o añadir métodos, en cualquier momento
- Hasta los tipos básicos, como `Class` u `Object`

## ¿Cómo creen si no que **Rails**...

- ...añade métodos a clases ActiveRecord tras leer la base de datos?  
(como `name` o incluso `find_by_name()`)
- ...añade unos métodos tras ejecutar otros?  
(`has_many :posts` añade el método `posts`, etc...)
- ...modifica nuestros métodos para que ejecuten los filtros?  
(`before_filter` y `after_filter`)

Nosotros también podemos redefinir

## Nosotros también podemos redefinir

```
class Fixnum
  def +(other)
    self - other
  end
end
```

## Nosotros también podemos redefinir

```
class Fixnum
  def +(other)
    self - other
  end
end
```

```
>> 5 + 4
```

## Nosotros también podemos redefinir

```
class Fixnum
  def +(other)
    self - other
  end
end
```

```
>> 5 + 4
```

```
=> 1
```

Un ejemplo más cabal

## Un ejemplo más cabal

```
class Post < ActiveRecord::Base
```

```
end
```

## Un ejemplo más cabal

```
class Post < ActiveRecord::Base
  def title=(title)
    write_attribute('title', title)

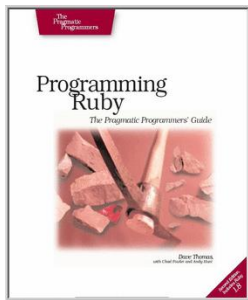
  end
end
```

## Un ejemplo más cabal

```
class Post < ActiveRecord::Base
  def title=(title)
    write_attribute('title', title)
    if self.new?
      write_attribute('sanitized_title', title.sanitize)
    end
  end
end
```

# Preguntas

¿ ... ?



**Moraleja:** Aprendan Ruby

Merece la pena

¡Y gracias por venir!



Salamanca 17

Madrid – 28020  
España

tel. +34 91 567 0605

[www.the-cocktail.com](http://www.the-cocktail.com)

# Referencias

- *"Programming Ruby"*, Dave Thomas
- *"The Pragmatic Programmer"*, Adrew Hunt & Dave Thomas
- <http://www.ruby-lang.org/>
- <http://www.rubyonrails.org/>
- [http://www.cs.byu.edu/colloquia/2006Fall/presentations/Matz\\_slides/](http://www.cs.byu.edu/colloquia/2006Fall/presentations/Matz_slides/)
- <http://es.wikipedia.org/wiki/Plat%C3%B3n>
- <http://es.wikipedia.org/wiki/Her%C3%A1clito>

## Fotografías:

- <http://flickr.com/photos/rtv/290062998/in/pool-sweetcandy/>
- <http://microwave.gotovim.ru/>
- <http://www.xtec.es/lvallmaj/passeig/plato2.htm>
- <http://www.flickr.com/photos/mtnpix/294515665/>