Ruby on Rails (Introducción a Ruby)

Esteban Manchado Velázquez
zoso@foton.es

20 de mayo de 2006

Índice

Índice

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*
- Ruby: Lenguaje interpretado orientado a objetos

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*
- Ruby: Lenguaje interpretado orientado a objetos
- Ruby on Rails está escrito en Ruby, claro

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*
- Ruby: Lenguaje interpretado orientado a objetos
- Ruby on Rails está escrito en Ruby, claro
- Durante los dos primeros días, un poquito de Ruby

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*
- Ruby: Lenguaje interpretado orientado a objetos
- Ruby on Rails está escrito en Ruby, claro
- Durante los dos primeros días, un poquito de Ruby
- Suficiente para que no se pierdan, sólo lo más básico

De qué va todo esto

- Ruby on Rails: plataforma de desarrollo web *muy productiva*
- Ruby: Lenguaje interpretado orientado a objetos
- Ruby on Rails está escrito en Ruby, claro
- Durante los dos primeros días, un poquito de Ruby
- Suficiente para que no se pierdan, sólo lo más básico
- A la larga, recomendable aprovechar la potencia del lenguaje

Índice

Ajax y Javascript integrados en Rails

- Autocompletar:
  http://demo.script.aculo.us/ajax/autocompleter
- Autocompletar personalizado:
  http://demo.script.aculo.us/ajax/autocompleter_customized
- Listas ordenables:
  http://demo.script.aculo.us/ajax/sortable_elements
- Mini tienda:
  http://demo.script.aculo.us/shop

Demo de Backpack

- Backpack: aplicación real escrita en Rails
  pages.mov

Índice

Nacimiento de Ruby

- Diseñado por Yukihiro Matsumoto (`matz`)
- «Ruby» viene de Perl $\longrightarrow$ Pearl
- *I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python*
- *They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves*
- *Don't underestimate the human factor. Even though we are in front of computers, they are media. We are working for human, with human*
- *You want to enjoy life, don't you? If you get your job done quickly and your job is fun, that's good, isn't it? That's the purpose of life, partly. Your life is better*

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura
- Uso de mayúsculas y minúsculas (constantes, variables)

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura
- Uso de mayúsculas y minúsculas (constantes, variables)
- Se usan **mucho** los *bloques* (funciones anónimas)

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura
- Uso de mayúsculas y minúsculas (constantes, variables)
- Se usan **mucho** los *bloques* (funciones anónimas)
- Documentación empotrada

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura
- Uso de mayúsculas y minúsculas (constantes, variables)
- Se usan **mucho** los *bloques* (funciones anónimas)
- Documentación empotrada
- Inmaduro (cambios, pocos módulos *de desarrollo activo*)

Características generales

- «Perl orientado a objetos», «Smalltalk con sintaxis familiar»
- *Completamente* orientado a objetos (¡hasta `nil`!)
- Tipado dinámico (*Duck typing*)
- Sintaxis limpia, modo *poético*, «sufijos» de sentencias
- Sangrado libre, marcas de fin de estructura
- Uso de mayúsculas y minúsculas (constantes, variables)
- Se usan **mucho** los *bloques* (funciones anónimas)
- Documentación empotrada
- Inmaduro (cambios, pocos módulos *de desarrollo activo*)
- Comunidad abierta (muchos «refugiados»)

Para aprender...

- Entre Perl y Python en cuanto a integración

Para aprender...

- Entre Perl y Python en cuanto a integración
- Consola interactiva: `irb`

Para aprender...

- Entre Perl y Python en cuanto a integración
- Consola interactiva: `irb`
- Documentación empotrada de referencia: `rdoc`

Para aprender...

- Entre Perl y Python en cuanto a integración
- Consola interactiva: `irb`
- Documentación empotrada de referencia: `rdoc`
- Consulta de documentación, ayuda interactiva: `ri` e `ihelp`

Para aprender...

- Entre Perl y Python en cuanto a integración
- Consola interactiva: `irb`
- Documentación empotrada de referencia: `rdoc`
- Consulta de documentación, ayuda interactiva: `ri` e `ihelp`
- Consulta de documentación en web: `http://www.ruby-doc.org/find/pickaxe/string`

Para aprender...

- Entre Perl y Python en cuanto a integración
- Consola interactiva: `irb`
- Documentación empotrada de referencia: `rdoc`
- Consulta de documentación, ayuda interactiva: `ri` e `ihelp`
- Consulta de documentación en web: `http://www.ruby-doc.org/find/pickaxe/string`
- En el caso de Rails y otros, de moda los *vídeos*

Índice

Tipado en Ruby

- *Dinámico* pero *fuerte*

Tipado en Ruby

- *Dinámico* pero *fuerte*
- Las variables no tienen tipo

Tipado en Ruby

- *Dinámico* pero *fuerte*
- Las variables no tienen tipo
- Son como referencias/punteros en otros lenguajes

Tipado en Ruby

- *Dinámico* pero *fuerte*
- Las variables no tienen tipo
- Son como referencias/punteros en otros lenguajes
- Los objetos a los que apuntan sí tienen tipo

Tipado en Ruby

- *Dinámico* pero *fuerte*
- Las variables no tienen tipo
- Son como referencias/punteros en otros lenguajes
- Los objetos a los que apuntan sí tienen tipo
- Las variables no se declaran, «aparecen» al asignarles un valor

Tipado en Ruby

- *Dinámico* pero *fuerte*
- Las variables no tienen tipo
- Son como referencias/punteros en otros lenguajes
- Los objetos a los que apuntan sí tienen tipo
- Las variables no se declaran, «aparecen» al asignarles un valor
- No hay conversiones *automáticas* entre distintos tipos

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

Ruby básico (I)

```
irb(main):001:0> lista = [3,2,1]
=> [3, 2, 1]
irb(main):002:0> lista2 = lista
=> [3, 2, 1]
irb(main):003:0> lista.sort!
=> [1, 2, 3]
irb(main):004:0> lista
=> [1, 2, 3]
irb(main):005:0> lista2
=> [1, 2, 3]
```

- ¿Sorpresa?
- Normalmente no es un problema (estilo funcional *POWA*!)

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Ruby básico (II)

```
irb(main):001:0> msj = "Hola, mundo"
=> "Hola, mundo"
irb(main):002:0> msj[3..4]
=> "a,"
irb(main):003:0> msj[3..-1]
=> "a, mundo"
irb(main):004:0> may = msj.upcase
=> "HOLA, MUNDO"
irb(main):005:0> may
=> "HOLA, MUNDO"
irb(main):006:0> msj
=> "Hola, mundo"
```

Índice

Duck Typing

- Expresión acuñada por Dave Thomas

Duck Typing

- Expresión acuñada por Dave Thomas
- «Si camina como un pato, nada como un pato, tiene pico de pato, ...»

Duck Typing

- Expresión acuñada por Dave Thomas
- «Si camina como un pato, nada como un pato, tiene pico de pato, ...»
- Se juzga a los objetos por sus métodos, no por *tipos*

Duck Typing

- Expresión acuñada por Dave Thomas
- «Si camina como un pato, nada como un pato, tiene pico de pato, ...»
- Se juzga a los objetos por sus métodos, no por *tipos*
- Da polimorfismo «gratis» y facilita escribir código genérico

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos
end

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos
end

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos
end

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos
end

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos => 1.next + 2 => 2 + 2
end

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Ruby básico (cuac)

```
# Duck Typing r0x!
def metodo_patoso(uno, dos)
  uno.next + dos  => "algo".next + "otro algo" =>
end                    "algp" + "otro algo"

irb(main):001:0> metodo_patoso(1, 2)
=> 4
irb(main):002:0> metodo_patoso("algo", "otro algo")
=> "algpotro algo"
```

Índice

Micro-intro a la OO

- *Clases* de *objetos*

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*
- Los objetos tienen *atributos*

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*
- Los objetos tienen *atributos*
- Las clases *heredan* de otras

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*
- Los objetos tienen *atributos*
- Las clases *heredan* de otras
- Al menos teóricamente, la OO nos hace natural pensar en términos que *facilitan* reducir el acoplamiento entre conceptos diferentes

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*
- Los objetos tienen *atributos*
- Las clases *heredan* de otras
- Al menos teóricamente, la OO nos hace natural pensar en términos que *facilitan* reducir el acoplamiento entre conceptos diferentes
- **No les culparé si no se lo tragan**

Micro-intro a la OO

- *Clases* de *objetos*
- Los objetos responden a *métodos*
- Los objetos tienen *atributos*
- Las clases *heredan* de otras
- Al menos teóricamente, la OO nos hace natural pensar en términos que *facilitan* reducir el acoplamiento entre conceptos diferentes
- **No les culparé si no se lo tragan (o no lo entienden)**

OO en Ruby

- Simple, cómoda de escribir (más vale)

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple
- *No existen* los atributos (desde fuera)

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple
- *No existen* los atributos (desde fuera)
- Métodos terminados en «!» y «?»

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple
- *No existen* los atributos (desde fuera)
- Métodos terminados en «!» y «?»
- Métodos especiales «=» para caramelos sintácticos

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple
- *No existen* los atributos (desde fuera)
- Métodos terminados en «!» y «?»
- Métodos especiales «=» para caramelos sintácticos
- Se usa «@» y «@@» para los atributos de objeto/clase

OO en Ruby

- Simple, cómoda de escribir (más vale)
- Es la forma natural de resolver los problemas
- Se puede escribir en estilo *no* OO, pero en realidad es OO
- Herencia simple
- *No existen* los atributos (desde fuera)
- Métodos terminados en «!» y «?»
- Métodos especiales «=» para caramelos sintácticos
- Se usa «@» y «@@» para los atributos de objeto/clase
- Se usa «$» para variables globales (vale, no OO)

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

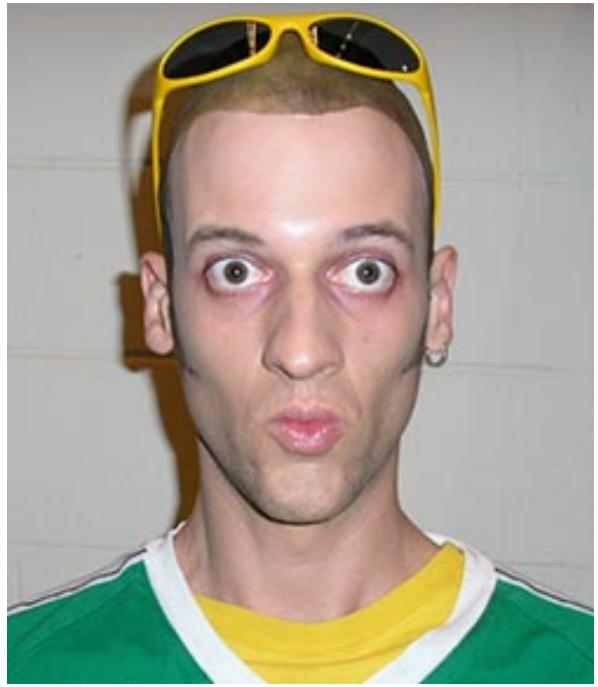
Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (I)

```
class SerVivo
  attr_accessor :nombre
  attr_reader   :edad
  def initialize(nombre, edad)
    @nombre  = nombre
    @edad    = edad
    @persona = false
  end
  def saludar
    puts "¿Qué pasa, #{@nombre}?"
  end
  def persona?; @persona end
  def personizar!;  @persona = true end
  def edad=(nuevaEdad) @edad = nuevaEdad.to_i end
end
```

Ejemplo algo más «real» (II)



Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (II)

```
irb(main):030:0> neng = SerVivo.new('Neng', 25)
=> #<Persona:0xb7a254a4 @persona=false, @edad=25, ...>
irb(main):031:0> neng.saludar
¿Qué pasa, Neng?
=> nil
irb(main):032:0> neng.persona?
=> false
irb(main):033:0> neng.personizar!
=> true
irb(main):034:0> neng.persona?
=> true
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = ’26’
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = ’Pepita’
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

Ejemplo algo más «real» (III)

```
irb(main):035:0> neng.edad = '26'
=> "26"
irb(main):036:0> neng.edad
=> 26
irb(main):037:0> neng.nombre
=> "Neng"
irb(main):038:0> neng.nombre = 'Pepita'
=> "Pepita"
irb(main):039:0> neng.nombre
=> "Pepita"
irb(main):040:0> neng.sexo
=> :femenino
```

(Es broma)

Índice

Estructuras

- Pocas, en parte por los bloques

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`
- `if` tiene `elsif` aparte de `else`, y devuelve un valor

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`
- `if` tiene `elsif` aparte de `else`, y devuelve un valor
- `if` tiene `then` optativo (sintaxis de una línea)

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`
- `if` tiene `elsif` aparte de `else`, y devuelve un valor
- `if` tiene `then` optativo (sintaxis de una línea)
- `case` tiene una forma curiosa de comparar (operador `===` *de lo que va en el* `when`)

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`
- `if` tiene `elsif` aparte de `else`, y devuelve un valor
- `if` tiene `then` optativo (sintaxis de una línea)
- `case` tiene una forma curiosa de comparar (operador `===` *de lo que va en el* `when`)
- `case` devuelve un valor

Estructuras

- Pocas, en parte por los bloques
- `if`, `case`, `while`, `loop`, `for`
- `if` tiene `elsif` aparte de `else`, y devuelve un valor
- `if` tiene `then` optativo (sintaxis de una línea)
- `case` tiene una forma curiosa de comparar (operador `===` *de lo que va en el* `when`)
- `case` devuelve un valor
- Las tres últimas *rara vez se usan*

Ejemplo

```
if 0 then puts "Sorpresa" else puts "0 es falso" end

if 0
   puts "No te enteras..."
else
   puts "Solamente false y nil son falsos"
end
```

Ejemplo

```
if 0 then puts "Sorpresa" else puts "0 es falso" end

if 0
   puts "No te enteras..."
else
   puts "Solamente false y nil son falsos"
end
```

Ejemplo

```
if 0 then puts "Sorpresa" else puts "0 es falso" end

if 0
   puts "No te enteras..."
else
   puts "Solamente false y nil son falsos"
end
```

Más ejemplos

```
case 'pepito'
   when /pep/
      puts "Guardiola"
   when Integer
      puts "Un entero"
   when "a" .. "z"
      puts "Una sola letra"
end
```

Más ejemplos

```
case 'pepito'
   when /pep/
      puts "Guardiola"
   when Integer
      puts "Un entero"
   when "a" .. "z"
      puts "Una sola letra"
end
```

Más ejemplos

```
case 'pepito'
   when /pep/
      puts "Guardiola"
   when Integer
      puts "Un entero"
   when "a" .. "z"
      puts "Una sola letra"
end
```

Más ejemplos

```
case 'pepito'
   when /pep/
      puts "Guardiola"
   when Integer
      puts "Un entero"
   when "a" .. "z"
      puts "Una sola letra"
end
```

Más ejemplos todavía

```
a = 0
while a < 10 do a += 1 end

loop do
   a += 1
   puts "Bucle manual y aburrido"
   break if a > 20
end

for i in 35..40 do puts "Mejor, pero no" end
```

Bucles estilo Ruby

```
0.upto(9) do puts "Mejor" end

loop do
   a += 1
   puts "Bucle manual y aburrido"
   break if a > 20
end

for i in 35..40 do puts "Mejor, pero no" end
```

Bucles estilo Ruby

```
0.upto(9) do puts "Mejor" end
```

```
10.times do puts "Bucle rubyano" end
```

```
for i in 35..40 do puts "Mejor, pero no" end
```

Bucles estilo Ruby

```
0.upto(9) do puts "Mejor" end
```

```
10.times do puts "Bucle rubyano" end
```

```
(35..40).each do |i| puts "Mucho mejor, #{i}" end
```

Índice

Ejercicio 1

- Pedir cinco palabras y meterlas en una lista
- Recorrer la lista, e imprimir las que tienen más de 3 letras
- Métodos útiles: `gets`, `puts`, `Array#push` y `String#size`
- ¿Algún problema sutil?

Índice

Ejercicio 2

- Escribir una clase `Traductor`
- En el constructor, recibirá el idioma al que traducir (`"inglés"`, `"francés"`, `"alemán"`)
- Tendrá un método `traducir`
- Para traducir al inglés, se añade `eishon` al final
- Para traducir al francés, se añade é al final
- Para traducir al alemán, siempre se devuelve `frufrunguen`
- ¿Demasiado fácil? Extra: en inglés y francés, quitar la última letra si es una vocal (`Array#include?`, `String#[]`)

Índice

Ejercicio 3

- Escribir una clase `Dado`
- En el constructor recibirá el número de caras
- Tendrá un método tirar, que devolverá un número aleatorio
- Escribir otra clase `Personaje`
- En el constructor recibirá el nombre
- Al crearse uno, se calcularán las características fuerza, destreza e inteligencia (tirando un dado de 20)
- Las características tendrán que ser visibles desde fuera
- ¿Demasiado fácil? Extra: por cada característica, tirar tres veces el dado y quedarse con la tirada más alta
- Métodos útiles: `rand`, `Array#max`

ツテボミヴス