

[Wikidot.com](http://Wikidot.com)

.wikidot.com

Share on      

[Edit](#) [History](#) [Tags](#) [Source](#)

[Explore »](#)

# Ruby Tutorial

## ...o como pasar un buen rato programando

- [admin](#)
  - [site manager](#)

[Create account](#) or [Sign in](#)



## Lección 1

- [Introducción](#)
- [Instalación](#)
- [El Primer Programa](#)
- [Números en Ruby](#)
- [Strings y diversión](#)
- [Variables](#)
- [Alcance de las variables](#)

## Lección 2

- [Introduciendo Datos](#)
- [Normas en los nombres](#)

- [Los métodos](#)
- [Los métodos: argumentos](#)
- [Rangos](#)
- [Arrays](#)

## Lección 3

- [Bloques](#)
- [Más malabares con strings](#)
- [Expresiones Regulares](#)
- [Condicionales](#)
- [Bucles](#)
- [Números Aleatorios](#)

## Lección 4

- [Clases y Objetos](#)
- [Accesores](#)
- [Ficheros: lectura/escritura](#)
- [Cargando librerías](#)
- [Herencia de clases](#)
- [Modificando clases](#)
- [Congelando objetos](#)
- [Serializando objetos](#)

## Lección 5

- [Control de acceso](#)
- [Excepciones](#)
- [Módulos](#)
- [Constantes](#)
- [Hashes y Símbolos](#)
- [La clase Time](#)

## Lección 6

- [self](#)
- [Duck Typing](#)
- [Azúcar Sintáctico](#)
- [Test de unidades](#)

## contacto

[e-mail](#)



## Herencia de clases

La **herencia de clases** es una relación entre dos clases. La ventaja de la herencia es que las clases que en una jerarquía están en un nivel inferior, heredan las características de las clases de niveles superiores; y además, pueden añadir sus propias características.

Por ejemplo: todos los gatos son mamíferos. Si todos los mamíferos respiran, la clase gato por "descender" de la clase mamífero hereda esta característica: los gatos respiran.

Esto puede programarse así:

```
class Mamifero
  def respirar
    puts 'inspirar, respirar'
  end
end
```

```
# el símbolo < indica que
# Gato es una subclase de Mamifero

class Gato < Mamifero
  def maullar
    puts 'Miaaaaaaaaaaaaau'
  end
end

cribas = Gato.new
cribas.respirar
cribas.hablar
```

Aunque no se especificó que los gatos puedan respirar, todos los gatos herederán esa característica de la clase Mamifero, ya que el gato es una subclase de los Mamiferos. En el argot, Mamifero es la super-clase o clase padre, y Gato es la subclase, o clase hija. Esto es una ventaja para el programador: los gatos tienen la capacidad de respirar, sin haberlo implementado.

En Ruby, como se mostró en [este esquema](#), la clase **Object** es la madre de todas las clases en Ruby; por lo tanto, sus métodos están disponibles en todos los objetos, excepto a aquellos que se han sobreescrito.

## Sobreescritura de métodos (method overriding)

Habrán situaciones donde las propiedades de una super-clase no deberían ser heredadas por una subclase en particular. Por ejemplo, las aves generalmente saben volar, pero los pingüinos son una subclase de Ave, y no vuelan:

```
class Ave
  def asear
    puts 'Me estoy limpiando mis plumas.'
  end

  def volar
    puts 'Estoy volando.'
  end
end

class Pinguino < Ave
  def volar
    puts 'Lo siento, no soy capaz de volar.'

```

```

    end
end

p = Pinguino.new
p.asear
p.volar

```

Se ha sobrescrito el método volar. La gran ventaja que aporta el uso de la herencia de clases, se llama **programación diferencial**: vamos de lo más general a lo más particular, añadiendo y modificando donde sea necesario.

Los dos ejemplos anteriores son traducciones de la guía online [Ruby User's Guide](http://rubyuser.org/)

## Super

```

class Bicicleta
  attr_reader :marchas, :ruedas, :asientos # se hablará de attr_reader
  def initialize(marchas = 1)
    @ruedas = 2
    @asientos = 1
    @marchas = marchas
  end
end

class Tandem < Bicicleta
  def initialize(marchas)
    super
    @asientos = 2
  end
end

t = Tandem.new(2)
puts t.marchas
puts t.ruedas
puts t.asientos
b = Bicicleta.new
puts b.marchas
puts b.ruedas
puts b.asientos

```

Cuando uno usa **super** dentro de un método, Ruby manda un mensaje a la clase madre del objeto al

que pertenece el método, buscando un método con el mismo nombre. Si:

- se invoca con una lista vacía de argumentos (como este caso), `super` ó `super ( )`, no se pasan argumentos al método de la clase madre.
- se invoca con argumentos, `super(a, b, c)`, se mandand los argumentos a, b, c.

En este caso, se usa `super` en el método `initialize` de `Tandem`, lo que provoca el uso del `initialize` de `Bicicleta` para crear instancias de `Tandem`. La salida es:

```
2
2
2
1
2
1
```

**RAILS:** la herencia de clases es una de las claves en el desarrollo de RAILS

page\_revision: 17, last\_edited: 26 Jul 2009, 17:57 GMT-05 (491 days ago)

[EditTags](#) [History](#) [Files](#) [Print](#) [Site tools+ Options](#)

[Help](#) | [Terms of Service](#) | [Privacy](#) | [Report a bug](#) | [Flag as objectionable](#)

Powered by [Wikidot.com](#)

Unless otherwise stated, the content of this page is licensed under [Creative Commons Attribution-ShareAlike 3.0 License](#)

## Other interesting sites



### [A Barrel Full](#)

A Barrel Full of information for Oil Industry Professionals



### [Wiki for 'My Vineyard'](#)

A Facebook Application



### [Gemexpe](#)



### [Timothy Youth Fellowship](#)

re:Christ