



# **Prácticas Profesionalizantes 3**

# Tema de la clase

Lenguaje de Programación

# RUBY

---

# Historia de Ruby

Yukihiro Matsumoto (MATZ) empezó el desarrollo de Ruby en 1993. En 1995 lanzó la primera versión

Ruby es la fusión de varios lenguajes de programación:

- Smalltalk
- Perl
- LISP

Hasta el 2001, ruby fue conocido sólo en Japón.

El libro Programming Ruby fue el impulsor del lenguaje fuera de Japón.

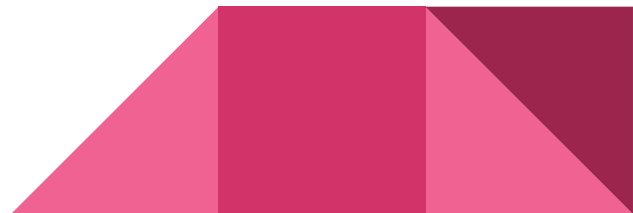
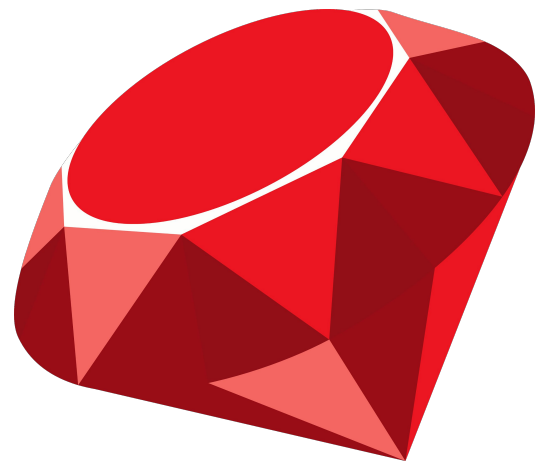
También conocido como [PickAxe](#)



# Ruby is designed to make programmers HAPPY

## Características:

- Dinámico
- Sintaxis concisa y expresiva
- Orientado a objetos
- Capacidades de metaprogramación
- Características funcionales



# Sintaxis y convenciones

`NombreDeClaseOModulo`

`CONSTANTE`

`@nombre_de_atributo`

`@@atributo_de_clase`

`$variable_global`

`nombre_de_metodo`

`metodo_peligroso!`

`metodo_que_pregunta?`

Estas son algunas de las convenciones que encontraremos en el lenguaje.

# Objetos

Todos los valores son objetos

```
"Aprendiendo ruby".length  
=> 16
```

```
1.object_id  
=> 3
```

```
nil.object_id  
=> 4
```

```
([1,2,3] + [4,5,6]).last  
=> 6
```

Arreglos

```
["Mateo", "Lola", "Lihue", "Clio"].sort
```

Números

```
-100.abs
```

Nulos

```
nil.nil?
```

# Números literales

3 # Entero

3.14 # Real

1\_999\_235\_243\_888 == 1999235243888

Podemos usar binario, octal o hexadecimal

0b1000\_1000 # Binario => 132

010 # Octal = 8

0x10 # Hexadecimal => 16

# Cadena de Caracteres (Strings) literales

```
'sin interpolar' => 'sin interpolar'  
"Interpolando: #{'Ja'*3}!" => "Interpolando: JaJaJa!"
```

Podemos usar otras formas

```
%q/Hola/  
%q!Chau!  
%Q{Interpolando: #{3+3}}
```





# Símbolos (Symbols)

Funcionan como variables prefijados con : (dos puntos)

Ejemplos: `:action`, `:line_items`, `:+`

- No es necesario declararlos
- Se garantiza que son únicos
- No es necesario asignarles ningún valor

Veamos un ejemplo:

```
:uno.object_id # siempre devolverá lo mismo
```

```
"uno".object_id # siempre devolverá diferente
```

# Arreglos

```
['Hola', 'Chau']
```

```
%w(Hola Chau #{2+2}) # sin interpolar
```

```
%W(Hola Chau #{2+2}) # interpolando
```

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





# Hashes

Siguen el mismo formato que un objeto JSON

```
{  
  nombre: 'Gonzalo',  
  apellido: 'Goral'  
}
```

Trabajan exactamente igual que los diccionarios de Python

# Expresiones Regulares

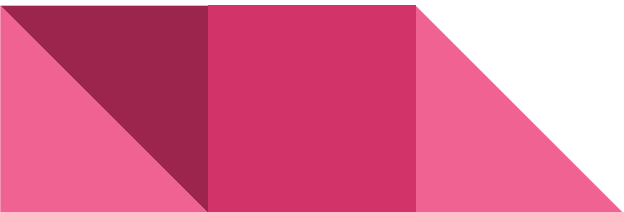
La función principal de una expresión regular es capturar un patrón

Un ejemplo de una expresión regular:

`/^[a-zA-Z]+$` #Probar en rubular que patrón captura está E.R.

Explicación detallada en Pickaxe [enlace\\_a\\_pickaxe](#)

Para probar expresiones regulares puede usar Rubular [enlace\\_a\\_rubular](#)



# Rangos

`0..10`

`"a".."z"`

`"a"..."z"`

Pueden convertirse en arreglos

`("a"..."z").to_a`

Rangos como intervalos

`(1..10) === 5 # => true`

`(1..10) === 15 # => false`

`('a'..'z') === 'z' # => true`

`('a'...'z') === 'z' # => false`

# Funciones Anónimas (Lambdas)

```
uno = lambda { |n| n * 2 }  
dos = ->(n, m){ n * 2 + m }  
tres = ->(n, m=0){ n * 2 + m }
```

# Entonces

```
uno.call 2          # => 4  
dos.call 2,3        # => 7  
tres.call 2         # => 4
```



# Bloques

Rara vez usaremos un for / while

```
3.times do |i|  
  puts i  
end  
# 0  
# 1  
# 2  
# => 3 (retorna el 3 que recibe .times)
```

```
3.times { |x| puts x }
```



# Programación Declarativa

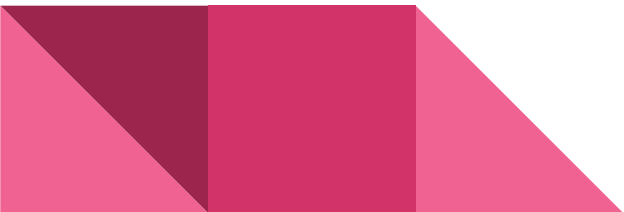
Hay tres métodos que funcionalmente enriquecen mucho al lenguaje, estos los aplicaremos sobre una colección de objetos.

Recordemos que en ruby todo es un objeto.

**Reduce**

**Map**

**Select**





# Programación Declarativa - Select

El método **select** funciona como un filtro, devolviendo una nueva colección con los elementos que fueron seleccionados o cumplen cierta condición.

¿Cómo obtener los números pares de un rango?

```
(1..10).select { |n| n.even? }
```

# o lo que es igual:

```
(1..10).select(& :even?)
```



# Programación Declarativa - Map

El método **map** ó **collect** tiene un funcionamiento simple, aplica una función a todos los elementos de la colección, y devuelve una nueva colección con esa función aplicada.

¿Cómo obtener el cuadrado de los primeros 10 números naturales?

```
(1..10).map { |n| n*2 }
```

# o lo que es igual:

```
(1..10).collect { |n| n*2 }
```



# Programación Declarativa - Reduce

El método **reduce** es un poco más complejo que los dos anteriores. Toma el primer elemento de la colección, aplica la función y guarda el resultado.

El método **inject** funciona igual pero toma un valor inicial.

¿Como sumar los primero 100 números naturales?

```
(1..100).reduce { |sum, n| sum + n }
```

```
(1..100).inject(0) { |sum, n| block }
```

# o lo que es igual:

```
(1..100).reduce(:+)
```

```
(1..100).inject(0, :+)
```



# Instalación

# Historia


Ruby tiene muchas implementaciones

¿Cual conviene usar? ¿Y si necesito o quiero usar más de 1?

## **rvm** - Ruby Version Manager

- Fue la primera herramienta para instalar

## **rbenv** - RuBy ENVironment

- Fue la primera alternativa a **rvm** y rápidamente se hizo muy popular
  - Es más simple que **rvm** y ya no instala rubies ni usa gemsets
  - Se puede agregar funcionalidad con plugins
  - Utiliza shims para cambiar de versión de ruby
- 

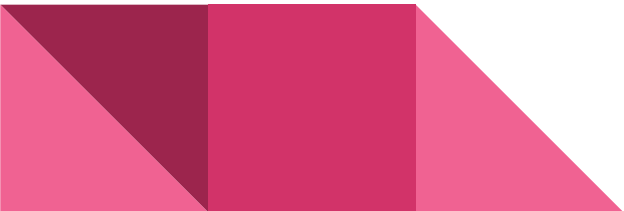
# Instalando rbnbv

Vamos a instalar `rbenv` con `git`

Vamos a instalar los siguientes plugins

`ruby-build`

`rbenv-update`



# Instalando rbenv

1.- Instalamos rbenv en ~/.rbenv

```
$ git clone https://github.com/sstephenson/rbenv.git ~/.rbenv
```

2.- Agregamos ~/.rbenv/bin a \$PATH

```
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
```

En ubuntu, hacer el echo en .bashrc en vez de .bash\_profile

```
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
```

# Instalando rbenv

3.- Agregamos `rbenv init` al shell para habilitar el autocompletado

```
$ echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
```

En ubuntu, hacer el `echo` en `.bashrc` en vez de `.bash_profile`

```
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

4.- Reiniciamos el shell

```
$ exec $SHELL -l
```





# Instalando ruby-build

Descargamos el plugin en el directorio ~/.rbenv/plugins

```
$ git clone https://github.com/sstephenson/ruby-build.git ~/.rbenv/plugins/ruby-build
```



# Instalando rbenv-update

Simplifica la actualización de rbenv y todos sus plugins

Clonamos el plugin en el directorio ~/.rbenv/plugins

```
$ git clone https://github.com/rkh/rbenv-update.git ~/.rbenv/plugins/rbenv-update
```



# Comandos de rbenv

Versiones, para ver las versiones instaladas de ruby (con un \* la versión actual)

```
$ rbenv versions
```

Versión Global, muestra o setea la versión global de ruby

```
$ rbenv global
```

```
$ rbenv global 2.0.0-p247 # setea la versión 2.0.0-p247 como global
```

`rbenv local` idéntico al comando anterior, pero para el directorio actual

Listar todas las versiones disponibles

```
rbenv install -l
```





¿Preguntas?