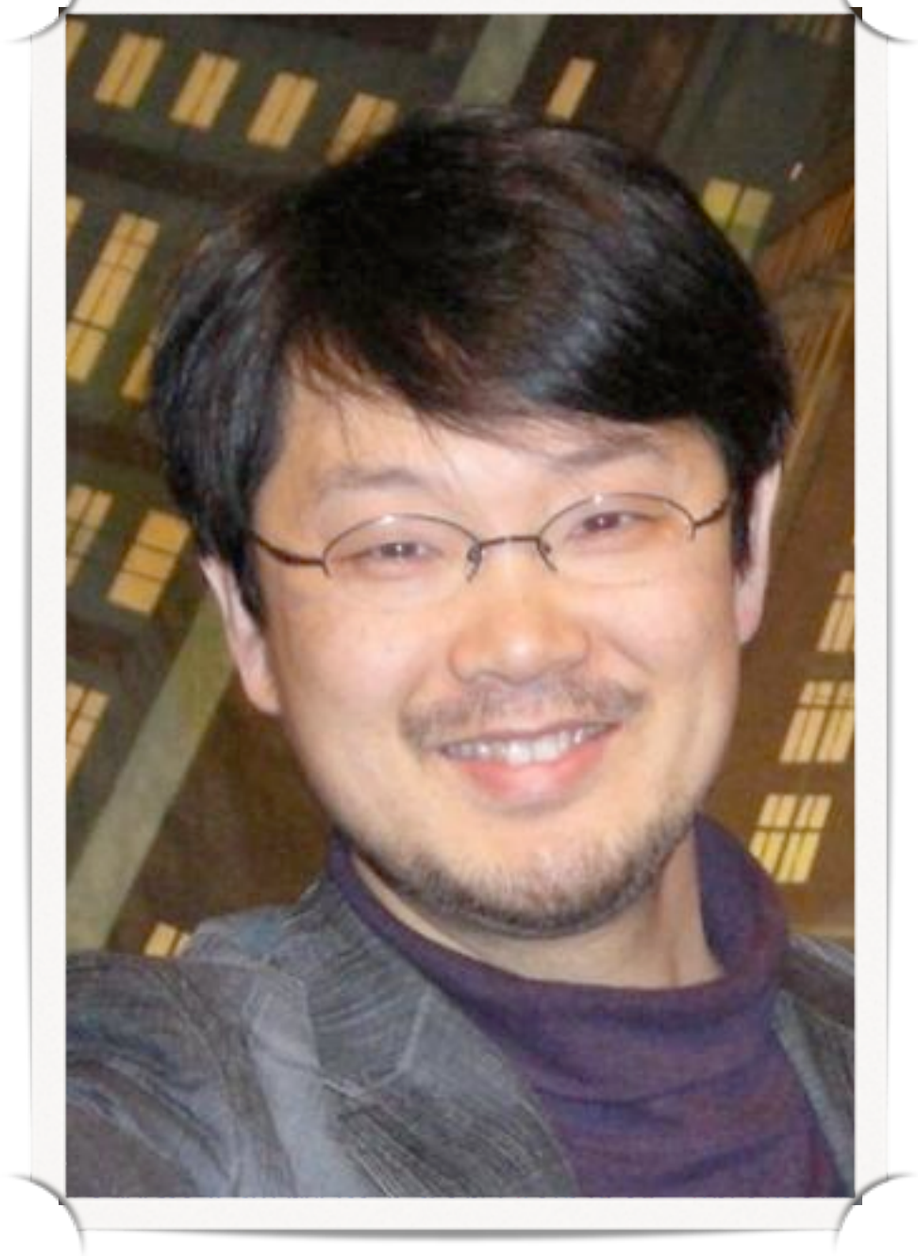
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Raúl Montes T.

Creado en 1995 por



Yukihiro Matsumoto

“Matz”

"... trying to make Ruby natural, not simple"

"Ruby is simple in appearance, but is very complex inside, just like our human body"

TL;DR

Sentencias de asignación de variables
Sentencias de control de flujo: if, loops
Funciones / Métodos
Tipos básicos como numbers, strings, arrays y hashes
Orientado a objetos, con clases

Pero con varias particularidades...

Ruby es flexible

No necesitas terminar las sentencias con “**;**”, pero puedes...

No tienes que encerrar entre **( )** los argumentos, pero puedes...

Puedes redefinir todo el lenguaje si quieres...

Ruby es flexible

Sin embargo...

Decide un estilo de código y procura seguirlo en el proyecto

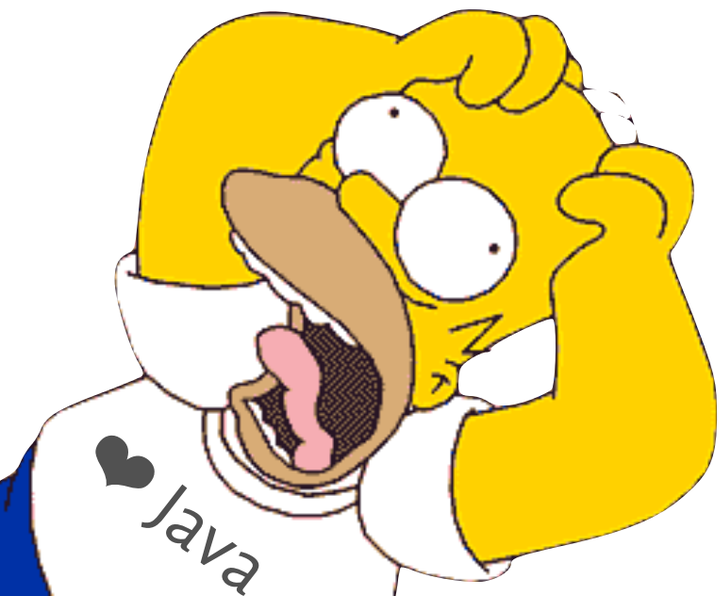
No redefinas / extiendas el core del lenguaje a menos que tengas un muy buen motivo para ello

Ruby is a **dinamycally** typed language

Ruby is a **strongly** typed language

Las variables no se declaran, simplemente se asignan

Pueden ser:

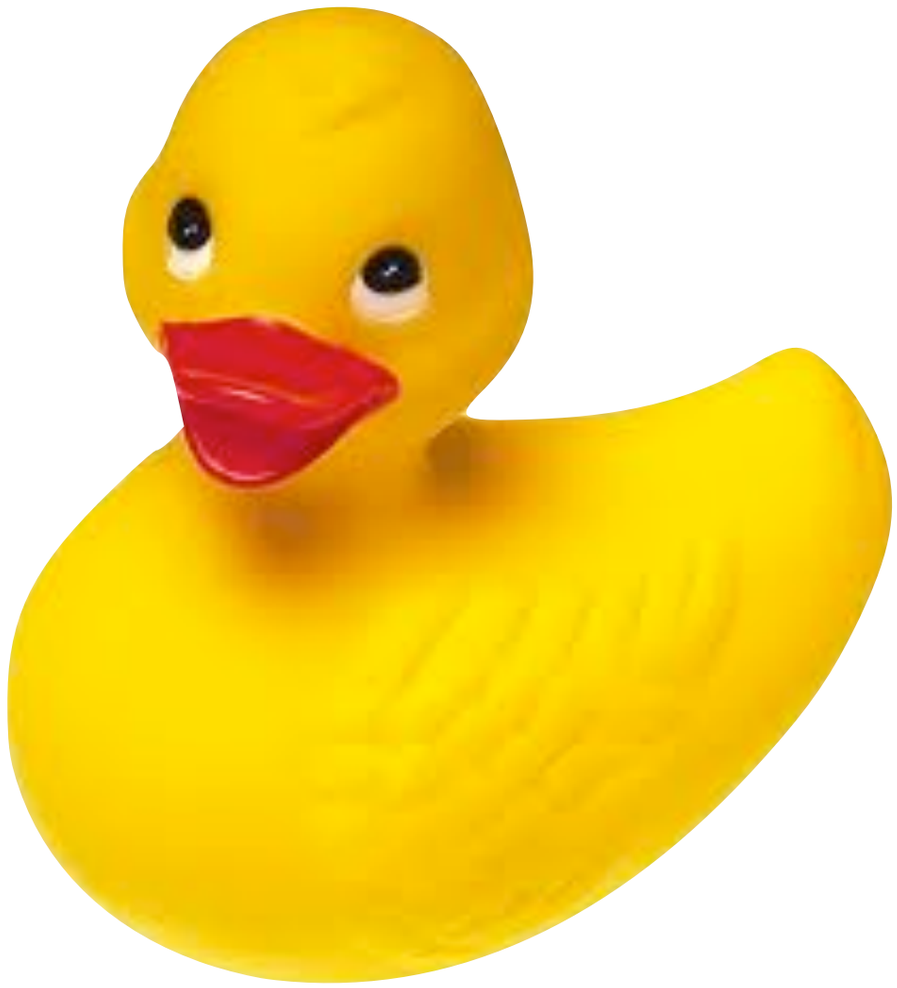



```
locales
CONSTANTES
$GLOBALES
@de_instancia
@@de_clase
```

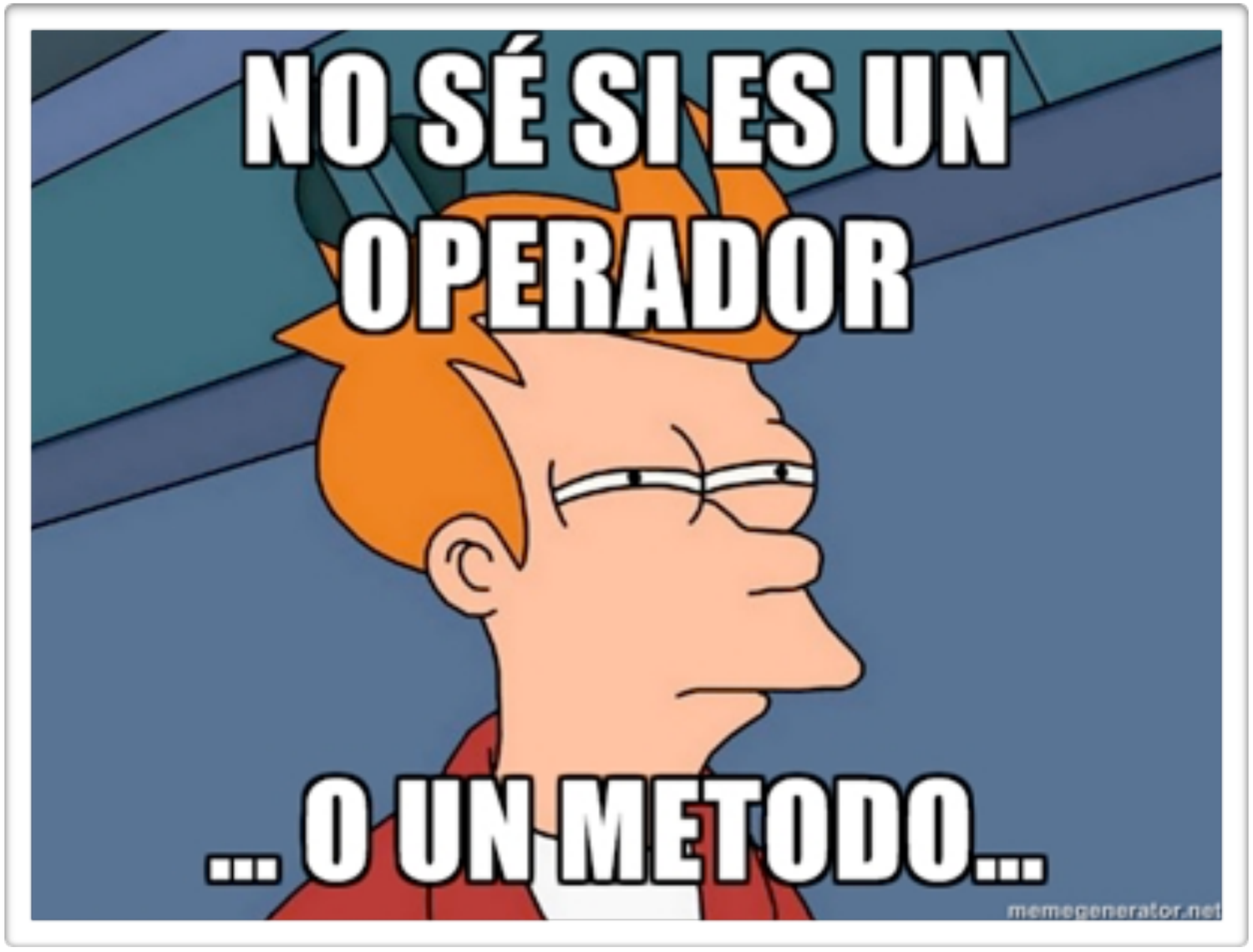
Ruby is **really** Object Oriented

TODO es un objeto

Ruby no tiene interfaces, pero...



Duck Typing



Los operadores son sólo “syntax sugar”... en realidad son métodos...

otros aspectos generales...

- Exception handling
- Garbage collection
- Herencia simple (pero tiene Mixins)
- Bloques, lambdas, procs

Algunos recursos para aprender Ruby

- http://tryruby.org/ (para saborearlo)
- https://www.ruby-lang.org/en/documentation/ (listado de muchos recursos gratuitos)
- Code style guide: https://github.com/bbatsov/ruby-style-guide

VAMOS A LOS DETALLES...

Tipos de Dato

- Text
- Arrays/Hashes
- Symbols
- Objects

Text

- Comillas simples para los literales

```
'This is a simple Ruby string literal'
```

- Comillas dobles permiten interpolación

```
"360 degrees=#{2*Math::PI} radians"
"360 degrees=6.28318530717959 radians"
```

- Delimitadores arbitrarios

```
%q(Don't worry about escaping ' characters!)
%Q|"How are you?", he said|
%q_This string literal contains \_underscores
\__
```

“Operadores” de Strings

- + concatenación
- << append
- [] acceso a caracteres

```ruby
planet = "Earth"
"Hello" + " " + planet  # Produces "Hello Earth"

greeting = "Hello"
greeting << " " << "World"
puts greeting      # Outputs "Hello World"
```

```ruby
s = 'hello'
s[0]                # 'h'
s[s.length-1]       # 'o'
s[-1]               # 'o'
s[-2]               # 'l'
s[-s.length]        # 'h'
s[s.length]         # nil
```

Símbolos

- strings inmutables que representan cosas (eficiente)
- símbolos distintos representan contenidos distintos
  - se usa el prefijo `:`

```
:symbol                       # A Symbol literal
:"symbol"                     # The same literal
:'another long symbol' # useful for symbols with spaces
s = "string"
sym = :"#{s}"                 # The Symbol :string
```

Arrays

- índices de 0 a size -1
- índices negativos cuentan desde el final
- untyped (elementos pueden ser de distinto tipo)
- tamaño dinámico (se ajustan a la necesidad)

```ruby
[1, 2, 3]              # An array that holds three Fixnum objects
[-10...0, 0..10,]      # An array of two ranges; trailing commas
allowed
[[1,2],[3,4],[5]]      # An array of nested arrays
[x+y, x-y, x*y]        # Array elements can be arbitrary expressions
[]                     # The empty array has size 0
words = %w[this is a test]  # Same as: ['this', 'is', 'a', 'test']
open = %w| ( [ { < |        # Same as: ['(', '[', '{', '<']
white = %W(\s \t \r \n)     # Same as: ["\s", "\t", "\r", "\n"]
a = [0, 1, 4, 9, 16]     # Array holds the squares of the indexes
a[0]          # First element is 0
a[-1]         # Last element is 16
a[-2]         # Second to last element is 9
a[a.size-1]  # Another way to query the last element
a[-a.size]   # Another way to query the first element
a[8]          # Querying beyond the end returns nil
a[-8]         # Querying before the start returns nil, too
```

```ruby
a[0] = "zero"      # a is ["zero", 1, 4, 9, 16]
a[-1] = 1..16      # a is ["zero", 1, 4, 9, 1..16]
a[8] = 64          # a is ["zero", 1, 4, 9, 1..16, nil, nil, nil, 64]
a[-9] = 81         # Error: can't assign before the start of an array
a = ('a'..'e').to_a    # Range converted to ['a', 'b', 'c', 'd', 'e']
a[0, 0]                # []: this subarray has zero elements
a[1, 1]                # ['b']: a one-element array
a[-2, 2]               # ['d','e']: the last two elements of the
array
a[0..2]                # ['a', 'b', 'c']: the first three elements
a[-2..-1]              # ['d','e']: the last two elements of the
array
a[0...-1]           # ['a', 'b', 'c', 'd']: all but the last element
```

Operadores de Array

- + concatenación
- – diferencia (como en conjuntos)
- << append al final del array
- * multiplica un elemento
- | union (elimina duplicados)
- & intersección (elimina duplicados)
- métodos: `each`, `sort`, `reverse`, etc.

Ejemplos

```
a = [1, 2, 3] + [4, 5]    # [1, 2, 3, 4, 5]
a = a + [[6, 7, 8]]   # [1, 2, 3, 4, 5, [6, 7, 8]]
a = a + 9                 # Error: righthand side must be an array
a = []                # Start with an empty array
a << 1                # a is [1]
a << 2 << 3           # a is [1, 2, 3]
a << [4,5,6]          # a is [1, 2, 3, [4, 5, 6]]
a = [0] * 8           # [0, 0, 0, 0, 0, 0, 0, 0]
a = [1, 1, 2, 2, 3, 3, 4]
b = [5, 5, 4, 4, 3, 3, 2]
a | b     # [1, 2, 3, 4, 5]: duplicates are removed
b | a     # [5, 4, 3, 2, 1]: same, but order is different
a & b     # [2, 3, 4]
b & a     # [4, 3, 2]
```

Hashes

- también conocidos como maps
- similar a arrays pero mantiene pares key-value (índice es cualquier cosa)

```ruby
numbers = Hash.new      # Create a new, empty, hash object
numbers = {}            # The same
numbers["one"] = 1      # Map the String "one" to the Fixnum 1
numbers["two"] = 2
numbers["three"] = 3
sum = numbers["one"] + numbers["two"]
numbers = { "one" => 1, "two" => 2, "three" => 3 }
numbers = { :one => 1, :two => 2, :three => 3 }  # better
numbers = { one: 1, two: 2, three: 3 }  # the best, new in 1.9
```

Objetos

- todo valor es objeto (no hay tipos primitivos)
- todos los objetos descienden de la clase Object
- siempre se manejan referencias a los objetos (no el objeto mismo)
- si no son literales se crean con new
- garbage collection automático

Ejemplos

```ruby
s = "Ruby"  # Create a String object. Store a reference to it in s.
t = s       # Copy the reference to t. s and t both refer to the same object.
t[-1] = ""  # Modify the object through the reference in t.
print s     # Access the modified object through s. Prints "Rub".
t = "Java"  # t now refers to a different object.
print s, t     # Prints "RubyJava".
a = "Ruby"        # One reference to one String object
b = c = "Ruby"# Two references to another String object
a.equal?(b)# false: a and b are different objects
b.equal?(c)    # true: b and c refer to the same object
a == b         # true: but these two distinct objects have equal values
```