

## Lezione 3: Introduzione a Ruby

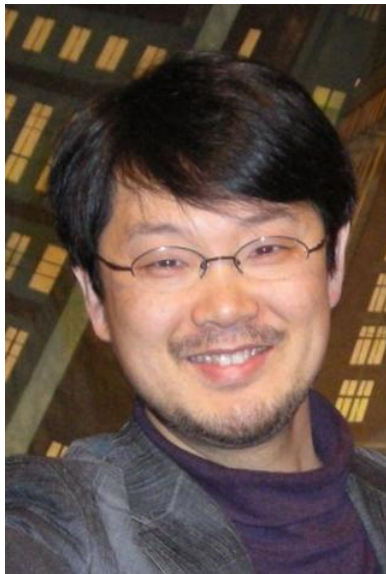
Matteo Vaccari

<http://matteo.vaccari.name/>  
matteo.vaccari@uninsubria.it



(cc) Matteo Vaccari. Published in Italy.  
Attribution – Non commercial – Share alike 2.5

Yukihiro Matsumoto detto “Matz”



# Design Policy of Ruby I

For me, the purpose of life is, at least partly, to have joy. Programmers often feel joy when they can concentrate on the creative side of programming, so Ruby is designed to make programmers happy.

—Matz

# Design Policy of Ruby II

## Principle of Conciseness

I want computers to be my servants, not my masters. Thus, I'd like to give them orders quickly. A good servant should do a lot of work with a short order.

## Principle of Consistency

... a small set of rules covers the whole Ruby language. ... I've tried to follow the principle of "least surprise." Ruby is not too unique, so a programmer with basic knowledge of programming languages can learn it very quickly.

# Design Policy of Ruby III

## Principle of Flexibility

... Ruby consists of an unchangeable small core (that is, syntax) and arbitrary extensible class libraries. Because most things are done in libraries, you can treat user-defined classes and objects just as you treat built-in ones.

Programming is incredibly less stressful in Ruby because of these principles.

<http://www.informit.com/articles/article.aspx?p=18225>

<http://tryruby.hobix.com/>

# Try Ruby!

a hands-on tutorial

```
>> 2 ** 10  
=> 1024  
>> █
```

Numbers & Math

Ruby in 10 minutes

```
STDOUT.puts("Hello, world!");
```

```
STDOUT.puts("Hello, world!")
```

```
STDOUT.puts "Hello, world!"
```

```
puts "Hello, world!"
```

```
puts 'Hello, world!'
```

Ruby è conciso...

```
class Greeting {  
    public static void main(String args[]) {  
        System.out.println("Hello, world!");  
    }  
}
```

```
puts "Hello, world!"
```

## Type checking is dynamic

```
def what?(a, b, c)
  a + b + c
end
```

```
what?(1, 2, 3)      # => 6
```

```
what?("a", "b", "c") # => "abc"
```

```
class Pippo
  puts "Pippo has #{Pippo.new.methods.size} methods"
  def talk
    "Gawrsh!"
  end
  puts "Now it has #{Pippo.new.methods.size} methods"
end

p = Pippo.new
p.talk          # => "Gawrsh!"
```

=====

```
$ ruby pippo.rb
Pippo has 40 methods
$Now it has 41 methods
$
```

# Getters and setters

```
class Point
```

```
  def x
```

```
    @x
```

```
  end
```

```
  def x=(value)
```

```
    @x = value
```

```
  end
```

```
end
```

```
p = Point.new
```

```
p.x          # => nil
```

```
p.x = 3.14
```

```
p.x          # => 3.14
```

```
class Point
```

```
  attr_accessor :x
```

```
end
```

# Getters and setters

```
class Point
```

```
  def x
```

```
    @x
```

```
  end
```

```
  def x=(value)
```

```
    @x = value
```

```
  end
```

```
  def y
```

```
    @y
```

```
  end
```

```
  def y=(value)
```

```
    @y = value
```

```
  end
```

```
end
```

```
class Point
```

```
  attr_accessor :x, :y
```

```
end
```

# Meta programmazione

```
class Pippo
  class_eval "
    def abbaia
      'bau'
    end
  "
end
```

```
Pippo.new.abbaia # => "Bau"
```

# Arrays

```
[1, 2, 3].join(", ") # => "1, 2, 3"
```

```
a = []
```

```
a << "foo"
```

```
a << "bar"
```

```
a
```

```
# => ["foo", "bar"]
```

# Hashes

```
h0 = { 'one' => 1, 'two' => 2, 'three' => 3}  
      # => {"three"=>3, "two"=>2, "one"=>1}
```

```
h0['one']           # => 1
```

```
h1 = Hash.new           # => {}  
h1['gemstone'] = 'ruby' # => "ruby"  
h1['fruit'] = 'banana' # => "banana"  
h1  
      # => {"gemstone"=>"ruby", "fruit"=>"banana"}
```

```
h2 = { :june => 'perl', :july => 'ruby' }  
      # => {:july=>"ruby", :june=>"perl"}  
h2[:july]           # => "ruby"
```

# Blocks

```
3.times { puts "Ho!"}
```

Ho!

Ho!

Ho!

# Blocks

```
sum = 0
for x in [1, 2, 3]
  sum += x
end
sum # => 6
```

```
sum = 0
[1, 2, 3].each {|x| sum += x }
sum # => 6
```

```
[1, 2, 3].map {|x| x * x } # => [1, 4, 9]
```

```
[1, 2, 3].inject(0) {|sum, x| x } # => 6
```

# Blocks

```
[1,2,3,4,5].map { |e| e * e } # => [1, 4, 9, 16, 25]
```

```
[1,2,3,4,5].each do |e|  
  puts e  
end
```

## Vero e falso

Solo **nil** e **false** sono falsi.<sup>1</sup>

```
def is_true(value)
  value ? true : false
end

is_true(false)      # false
is_true(nil)        # false
is_true(true)       # true
is_true(1)          # true
is_true(0)          # true
is_true([0,1,2])    # true
is_true('')         # true
is_true(:a_symbol)  # true
```

---

<sup>1</sup>Some slides from <http://ruby.brian-schroeder.de/course/>

## Documentazione online

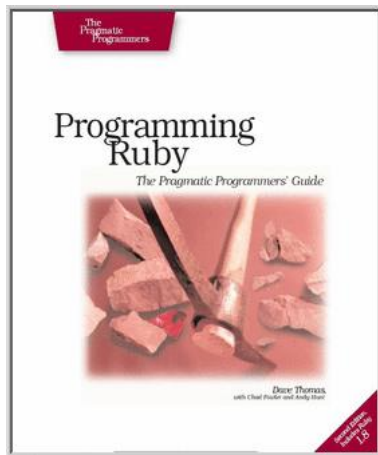
\$ ri String.to\_i

```
----- String#to_i
str.to_i(base=10) => integer
-----
```

Returns the result of interpreting leading characters in `_str_` as an integer base `_base_` (2, 8, 10, or 16). Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of `_str_`, `+0+` is returned. This method never raises an exception.

```
"12345".to_i           #=> 12345
"99 red balloons".to_i #=> 99
"0a".to_i             #=> 0
"0a".to_i(16)         #=> 10
"hello".to_i          #=> 0
"1100101".to_i(2)     #=> 101
"1100101".to_i(8)     #=> 294977
"1100101".to_i(10)    #=> 1100101
"1100101".to_i(16)    #=> 17826049
```

# Documentazione di carta



# Classi

```
class Dog
  def bark
    "bau!"
  end
end
```

```
fido = Dog.new # "new" è un class method
fido.bark      # "bark" è un instance method
```

# Classi ed ereditarietà

```
class Person
  def initialize(name)
    @name = name
  end

  def greet
    "Hello, my name is #{@name}."
  end
end
```

```
matteo = Person.new('Matteo')
puts matteo.greet
# => "Hello, my name is Matteo."
```

```
class Matz < Person
  def initialize
    super('Yukihiro Matsumoto')
  end
end

puts Matz.new.greet
# => "Hello, my name is Yukihiro Matsumoto"
```

# Class and instance methods

```
class Person
  def initialize(name)
    @name = name
  end

  def Person.matteo # class method
    Person.new("Matteo")
  end

  def hello # instance method
    "Hi, I'm #{@name}"
  end
end

m = Person.matteo
m.name      # => "Matteo"
m.hello     # => "Hi I'm Matteo"
```

## Esercizio: String interpolation

Write a file `calculate.rb` containing a function `explain_product(a, b)` that calculates  $a \cdot b$  and returns the string  
"the result of  $\#{a}$  times  $\#{b}$  is  $\#{a \cdot b}$ ".

Create a file `answer` containing the following lines:

```
#!/usr/bin/ruby -w
require 'calculate'
puts explain_product(6, 7)
```

Make the file executable (`chmod +x answer`) and call it (`./answer`).

## Esercizi

ri:

Use `ri` to find out how to make a string all uppercase, and try the function in `irb`.

## Instance variables

```
class Greeter
  def initialize(name)                # constructor
    @name = name
  end

  def say(phrase)
    puts "#{phrase}, #{@name}"
  end
end

g0 = Greeter.new("Fred")
g1 = Greeter.new("Wilma")
g0.say("Ciao")                       # => Ciao, Fred
g1.say("Dammi la clava")             # => Dammi la clava, Wilma
```

Le var di istanza sono **private**

```
class Greeter
  def initialize(name)      # constructor
    @name = name
  end

  def name                  # reader
    @name
  end

  def name=(new_name)     # writer
    @name = new_name
  end
end

g = Greeter.new("Barney")
puts g.name                # => Barney
g.name = "Betty"
puts g.name                # => Betty
```

## Le var di istanza sono davvero private

```
class Greeter
  def initialize(name)
    @name = name
  end

  def uguali?(greeter)
    # ERRORE!
    @name == greeter.@name
  end
end
```

```
class Greeter
  def initialize(name)
    @name = name
  end

  def name
    @name
  end

  def uguali?(greeter)
    @name == greeter.name # OK
  end
end
```

## Private and protected

```
class MyClass
  def m0      # public:    anyone can call
  end
protected
  def m1      # protected: call from instances of this class
  end        #           and subclasses
private
  def m2      # private:   call only from the same instance
  end
end
```

## Altri usi per i blocchi

```
f = File.open('pippo.txt', 'w')  
f.puts "Hello file!"  
f.close
```

```
File.open('pippo.txt', 'w') do |f|  
  f.puts "Hello file!"  
end
```

## Usare i blocchi

```
def take_block(v)
  if block_given?
    yield(v)
  else
    v
  end
end
```

```
take_block("no block")
# => "no block"
```

```
take_block(no block) { |s| s.sub("no ", "") }
# => "block"
```

## Esercizi: blocchi I

### n\_times

Write an iterator function `n_times(n)` that calls the given block `n` times.  
Write an iterator class `Repeat` that is instantiated with a number and has a method `each` that takes a block and calls it as often as declared when creating the object.

### Faculty

Write a one-liner in irb using `Range#inject` to calculate `20!`. Generalize this into a function.

## Esercizi: blocchi II

### Maximum

Write a function to find the longest string in an array of strings.

### `find_it`

Write a function `find_it` that takes an array of strings and a block. The block should take two parameters and return a boolean value.

The function should allow to implement `longest_string`, `shortest_string`, and other functions by changing the block.

## Esercizi: iterazione I

### Fibonacci

Write functions that calculate the fibonacci numbers using different looping constructs

$$fib(i) = \begin{cases} 0 & i = 0 \\ 1 & i = 1 \\ fib(i - 1) + fib(i - 2) & \textit{otherwise} \end{cases}$$

**recursion:** Implement the function using recursion.

**while:** Implement the function using a **while** loop.

**for:** Implement the function using a **for** loop.

**times:** Implement the function using the **times** construct.

**loop:** Implement the function using the **loop** construct.

## Esercizi: iterazione II

### Iterator

Write a fibonacci iterator function.

That is a function that takes a number  $n$  and a block and calls the block with  $fib(0)$ ,  $fib(1)$ ,  $\dots$   $fib(n)$

### Generator

Write a fibonacci generator class.

That is: A class that has a next function which on each call returns the next fibonacci number.