DIGITAL MASS
workshop di cultura digitale *a cura di* ninux.org

Beginning Ruby

Diego Luca Candido

Introduzione

Chi sono

Diego “joxer fracchio” Luca Candido
twitter:@joxer92
email: diego.luca.candido <at> gmail <dot> com
Studente a Roma

Due parole su ninux

La mission di ninux

1.Costruire una rete decentralizzata di proprietà dei partecipanti

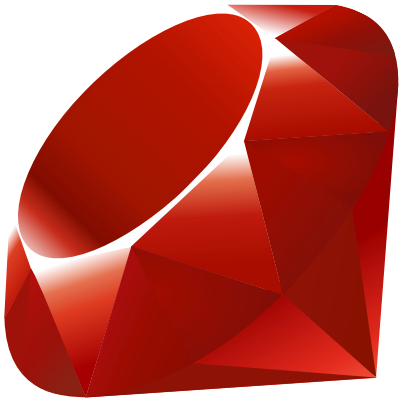
2. Aiutare le realtà colpite da Digital Divide condividendo gratuitamente la connessione ad internet della community

3. Diffondere gli ideali della libertà di comunicazione, la neutralità della rete, la filosofiadel software libero, la collaborazione volontaria per un fine comune e la condivisione dei saper

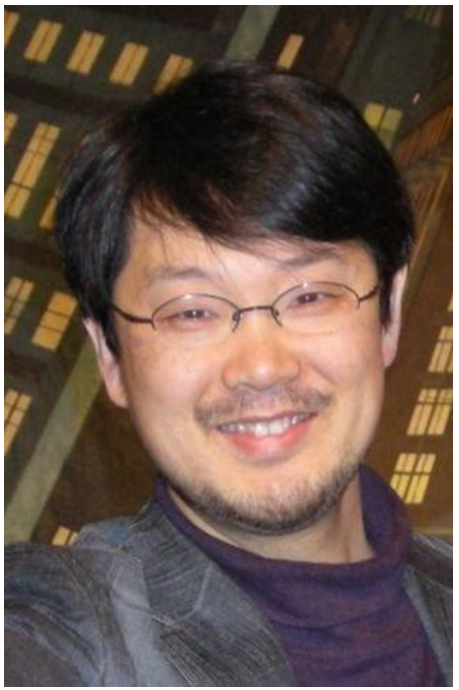
Obiettivi del giorno

- Sapere cosa è ruby
- Sapere cosa è la programmazione ad oggetti
- Capire l'environment di ruby
- Applicare ruby
- Scrivere piccoli script

Cosa è Ruby



Ruby nasce nel 1995 a opera di papà Yukihiro "Matz" Matsumoto



Il simbolo associato a ruby è il rubino

La popolarità di ruby è associata al famoso framework per il web “Rails”

Cosa è Ruby

- È un linguaggio di scripting
- Ha una sintassi semplice e piacevole
- È multipiattaforma
- Multiparadigma: object-oriented, funzionale, riflessiva
- Open source
- Esistono diverse implementazioni della vm

Ruby sì, Ruby No

Perchè sì

- Perche' ha una sintassi _favolosa_
- Perchè ha una grande community dietro
- Perchè esiste sulla maggiorparte delle piattaforme

Perchè no

- È lento
- Il core team di sviluppo è giapponese (non so voi ma io non lo conosco)

Varie implementazioni di Ruby

- Ruby MRI http://ruby-lang.org/
- Jruby http://jruby.org/
- Rubinius http://rubini.us/
- IronRuby http://www.ironruby.net/
- HotRuby e RubyJS http://rubyjs.org
- Rubymotion e MacRuby http://www.rubymotion.com/

Cosa installare sul vostro sistema

Utilizzeremo la vm di Ruby MRI 1.9.3

Linux: apt-get install ruby
yum install ruby
emerge ruby

Windows: installare ruby con RubyInstaller
http://rubyinstaller.org/

Mac OSX: brew install ruby

Se avete Linux o MacOS cercate RVM

Irb: Console interattiva di Ruby

La console interattiva di ruby è chiamata **irb**

Si richiama dal terminale del proprio computer con il comando “irb”, con jruby si chiama “jirb”

Esiste http://tryruby.org che dà una versione di irb simile online

Partiamo con Ruby

Stiamo per entrare nella miniera di Ruby. Pronti?

Hello World!

```
puts “Hello world”

print “Hello world\n”

puts “” + “Hello world”
puts  “”<<”Hello world”

puts String.new("Hello world")
```

In ruby ogni cosa è un oggetto

In ruby ogni cosa è un oggetto, Stringhe, Numeri, vero, falso, il valore nullo

```
"a"irb(main):001:0> "a".class
=> String
irb(main):002:0> Class.class
=> Class
irb(main):003:0> Object.class
=> Class
irb(main):004:0> String.class
=> Class
irb(main):005:0> 1.class
=> Fixnum
irb(main):006:0> (lambda {|x| }).class
=> Proc
```

In generale si può dire che ogni espressione ruby ritorna un oggetto

```
irb(main):007:0> puts
"ciao"
ciao
=> nil
```

Cosa è la programmazione ad oggetti (teoria)

Una classe è una rappresentazione astratta di un concetto

Un concetto nel nostro caso è una persona

ASTRATTA

Cosa è la programmazione ad oggetti (teoria)

Una classe ha dei metodi e degli attributi

Un metodo è una azione che il concetto esegue

Un attributo è una sua caratteristica

Attributi e metodi esistono per tutte le classi ma hanno valori propri per ognuna

Per accedere a metodi e attributi si usa l'operatore “.”

Scriviamo la nostra prima funzione

Le funzioni in ruby sono scritte con:

```
def nome_funzione(argomento1,argomento2)
    # corpo funzione
end

def nome_funzione argomento1,argomento2,argomento3
    # corpo funzione
end

def nome_funzione(argomento1=”abc”, argomento2=1)
    # corpo funzione
end
```

Cosa è la programmazione ad oggetti (teoria)

```
classe persona
  attributi: mionome, miocognome

  metodo inizializza(nome,cognome)
     mionome = nome
     miocognome = cognome
  fine

  metodo mangia()
      scrivi “sto mangiando!”
  fine

  metodo bevi()
    scrivi mionome + “sta bevendo”
  fine
fine_classe
```

Cosa è la programmazione ad oggetti (teoria)

L'istanza è la realizzazione pratica di una classe

mario = Persona.istanzia(nome: “mario”, cognome: “rossi”)

mario.bevi

>> mario sta bevendo

Cosa è la programmazione ad oggetti (pratica)

```
irb(main):001:0> class Persona
irb(main):002:1>  def initialize(nome,cognome)
irb(main):003:2>   @nome,@cognome = nome,co
irb(main):004:2>  end
irb(main):005:1>  def mangia
irb(main):006:2>   puts "sto mangiando"
irb(main):007:2>  end
irb(main):008:1>  def bevi
irb(main):009:2>   puts @nome+" sta bevendo"
irb(main):010:2>  end
irb(main):011:1> end
```

Cosa è la programmazione ad oggetti (pratica)

```
irb(main):012:0> mario =
Persona.new("mario","rossi")
=> #<Persona:0x30ce894 @nome="mario",
@cognome="rossi">
irb(main):013:0> mario.bevi
mario sta bevendo
=> nil
irb(main):014:0> mario.mangia
sto mangiando
=> nil
```

Analizziamo la classe Fixnum

Fixnum: class per gli interi

```
irb(main):001:0> 1 +
1
=> 2
irb(main):002:0>
1.+(1)
=> 2
irb(main):003:0> 1.-
(1)
=> 0
irb(main):004:0> 1 * 1
=> 1
irb(main):005:0>
1.*(1)
=> 1
irb(main):006:0>
2.**(4)
```

Float: class per interi con virgola mobile

```
irb(main):021:0> 1.0
=> 1.0
irb(main):022:0> 1.0.+1
=> 2.0
irb(main):023:0> 1.0 + 2.0
=> 3.0
irb(main):024:0> 0.9 == 1
=> false
irb(main):025:0> 0.9 ==
0.9
=> true
```

Analizziamo la classe String

String: Classe per le stringhe

```
irb(main):007:0> "ciao" + "mondo"
=> "ciaomondo"
irb(main):008:0> "ciao".+ "mondo"
=> "ciaomondo"
irb(main):009:0> "ciao".+ "mondo".+ "secondo"
=> "ciaomondosecondo"
```

In ruby non ho bisogno di mettere le parentesi tonde! (ma in alcuni casi occorrono, altrimenti l'interprete potrebbe fallire)

Scriviamo la nostra prima funzione

```
def nome_funzione(argomento1,argomento2)
    return argomento1
end

def nome_funzione(argomento1,argomento2)
    argomento1
end

def nome_funzione(argomento1,argomento2)
    nil
end
```

Strutture di controllo

Struttura If

```
if 1 > 0
 puts “maggiore”
elsif 1 < 0
 puts “minore”
else
 puts “nessuna condizione
end
```

Costrutto for

```
for x in [1,2,3]
  puts x
end
```

Struttura While

```
while true
 puts “dentro il while”
end
```

Struttura loop

```
loop do
      puts “dentro il loop”
end
```

Operatori di confronto

Operatori di confronto

and oppure && (logico)
or oppure || (logico)
not oppure ! (negazione)
> (maggiore)
< (minore)
>= (maggiore o uguale)
<= (minore o uguale)
== (uguale)
!= (diverso)

Tipi di dato base

Array

E' una lista ordinata di valori che partono dal numero 0

primoarray = [1,2,3]

secondoarray = [“a”,1,-5.0]

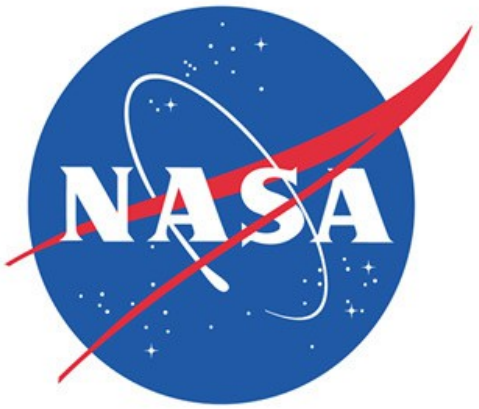
Hash

E' un associazione nome valore

miohash = {1 => “ciao”, “mio” => “mondo”}

Progetto NASA!

Siamo degli ottimi sviluppatori e la NASA ha detto che dovremmo scrivere il nuovo simulatore del progetto spaziale.



Noi siamo bravi e lo scriveremo in ruby.

Costruire una classe

Una classe si costruisce con la keyword "*class*" seguita da il nome della classe con la lettera maiuscola e chiuso il tutto da end

```
class Mondo
end
```

Il primo metodo eseguito dalla classe è "*initialize*"

```
class Mondo
      def initialize
            puts "ciao mondo"
      end
end
```

```
1.9.3p392 :009 > nuovomondo = Mondo.new
ciao
 => #<Mondo:0x000000018c70e8>
```

Metodi della classe

Metodi d'istanza

```
1.9.3p392 :010 > class Mondo
1.9.3p392 :011?>    def initialize
1.9.3p392 :012?>     puts "creo un nuovo mondo"
1.9.3p392 :013?>     end
1.9.3p392 :014?>   def popolo
1.9.3p392 :015?>     puts "siamo 8 miliardi"
1.9.3p392 :016?>     end
1.9.3p392 :017?>   end
 => nil
1.9.3p392 :018 > terra = Mondo.new
creo un nuovo mondo
 => #<Mondo:0x000000017d1198>
1.9.3p392 :019 > terra.popolo
siamo 8 miliardi
```

Metodi e applicazione

```
1.9.3p392 :042 > class Mondo
1.9.3p392 :043?>   def initialize(pianeta,numero)
1.9.3p392 :044?>     @pianeta = pianeta
1.9.3p392 :045?>     @numero = numero
1.9.3p392 :046?>     end
1.9.3p392 :047?>   def popolo
1.9.3p392 :048?>     puts "sul pianeta: "+@pianeta.to_s+"
siamo: "+@numero.to_s
1.9.3p392 :049?>     end
1.9.3p392 :050?>   end
 => nil
```

Metodi e applicazione

```
1.9.3p392 :051 > plutone = Mondo.new
ArgumentError: wrong number of arguments (0 for 2)
    from (irb):43:in `initialize'
    from (irb):51:in `new'
    from (irb):51
    from /home/joxer/.rvm/rubies/ruby-1.9.3-p392/bin/irb:16:in
`<main>'
1.9.3p392 :052 > marte = Mondo.new("Marte",0)
 => #<Mondo:0x000000013e0d68 @pianeta="Marte",
@numero=0>
1.9.3p392 :053 > marte.popolo
Sul pianeta: Marte siamo: 0
```

Metodi di classe

```
1.9.3p392 :054 > class Mondo
1.9.3p392 :055?>   def self.ruota_su_se_stesso(valore)
1.9.3p392 :056?>     # funzione matematica
COMPLICATISSIMA
1.9.3p392 :057 >      valore*2
1.9.3p392 :058?>    end
1.9.3p392 :059?>   end
 => nil
1.9.3p392 :060 > Mondo.ruota_su_se_stesso(42)
 => 84
```

Metodi della classe

Variabili d'istanza

```
1.9.3p392 :042 > class Mondo
1.9.3p392 :043?>   def initialize(pianeta,numero)
1.9.3p392 :044?>     @pianeta = pianeta
1.9.3p392 :045?>     @numero = numero
1.9.3p392 :046?>     end
1.9.3p392 :047?>   def popolo
1.9.3p392 :048?>     puts "sul pianeta: "+@pianeta+" siamo:
"+@numero
1.9.3p392 :049?>     end
1.9.3p392 :050?>   end
 => nil
```

La classe Sistema

```
1.9.3p392 :079 > class Sistema
1.9.3p392 :080?>   def initialize(stella)
1.9.3p392 :081?>     @stella = stella
1.9.3p392 :082?>     @pianeti = []
1.9.3p392 :083?>     end
1.9.3p392 :084?>   def stella #metodo get
1.9.3p392 :085?>     @stella end
1.9.3p392 :086?>   def stella=(nuova_stella) #metodo set
1.9.3p392 :087?>     @stella = nuova_stella
1.9.3p392 :088?>     end
1.9.3p392 :089?>   def aggiungi_pianeta(pianeta)
1.9.3p392 :090?>     @pianeti << pianeta
1.9.3p392 :091?>     end
1.9.3p392 :092?>   def pianeti
1.9.3p392 :093?>     @pianeti
1.9.3p392 :094?>     end
1.9.3p392 :095?>   end
```

La classe Sistema

```
1.9.3p392 :096 > solare = Sistema.new("Sole")
 => #<Sistema:0x000000016304b0 @stella="Sole", @pianeti=[]>
1.9.3p392 :097 > solare.aggiungi_pianeta("Terra")
 => ["Terra"]
1.9.3p392 :098 > solare.aggiungi_pianeta("Mercurio")
 => ["Terra", "Mercurio"]
1.9.3p392 :099 > solare.aggiungi_pianeta("Giove")
 => ["Terra", "Mercurio", "Giove"]
1.9.3p392 :100 > solare.aggiungi_pianeta("Saturno")
 => ["Terra", "Mercurio", "Giove", "Saturno"]
```

Blocchi e iteratori

Vogliamo vedere se su ognuno dei nostri pianeti ci sia la vita, dobbiamo perciò creare una funzione che verifica se esiste la vita

```
1.9.3p392 :116 > for x in solare.pianeti
1.9.3p392 :117?>   puts "pianeta: "+x+" vita:"+vita?(x).to_s
1.9.3p392 :118?>   end
pianeta: Terra vita:true
pianeta: Mercurio vita:false
pianeta: Giove vita:false
pianeta: Saturno vita:false
 => ["Terra", "Mercurio", "Giove", "Saturno"]
```

Questa soluzione è poco rubysta

Blocchi e iteratori

Un vero rubysta userebbe un blocco e un iteratore

Un blocco non è altro che veramente un blocco ^_^ (niente paura)

```
1.9.3p392 :119 > a = [ "a", "b", "c" ]
 => ["a", "b", "c"]
1.9.3p392 :120 > a.each do |x|
1.9.3p392 :121 >       puts x
1.9.3p392 :122?>   end
a
b
c
 => ["a", "b", "c"]
```

Un iteratore è un qualcosa al quale passato il blocco opera un azione su un oggetto

Blocchi e iteratori

```
1.9.3p392 :123 > solare.pianeti.each do |pianeta|
1.9.3p392 :124 >     puts "vita: "+ vita?(pianeta).to_s
1.9.3p392 :125?>   end
vita: true
vita: false
vita: false
vita: false
 => ["Terra", "Mercurio", "Giove", "Saturno"]
```

I blocchi non sono nient'altro che funzioni che esistono solamente nello spazio fra il *do* e l'*end*

Scriviamo il blocco per far partire la missione

La nasa ci ha informato che ci sarà a breve una missione su marte, dobbiamo mostrare a video su quali pianeti ci sarà una missione.

```
1.9.3p392 :138 > class Sistema
 ...
1.9.3p392 :139?>   def missione
1.9.3p392 :140?>     @pianeti.each do |x|
1.9.3p392 :141 >       yield(x)
1.9.3p392 :142?>     end
1.9.3p392 :143?>   end
1.9.3p392 :144?>  end
```

yield esegue il blocco che gli passiamo esternamente

Scriviamo il blocco per far partire la missione

```
1.9.3p392 :157 > solare.missione do |pianeta|
1.9.3p392 :158 >     if pianeta == "Marte"
1.9.3p392 :159?>     puts "c'è una missione su marte"
1.9.3p392 :160?>     else
1.9.3p392 :161 >       puts "non è prevista una missione sul pianeta:"
+pianeta
1.9.3p392 :162?>     end
1.9.3p392 :163?>   end
non è prevista una missione sul pianeta:Terra
non è prevista una missione sul pianeta:Mercurio
non è prevista una missione sul pianeta:Giove
non è prevista una missione sul pianeta:Saturno
c'è una missione su marte
 => ["Terra", "Mercurio", "Giove", "Saturno", "Marte"]
```

Vi sembra troppo complicato? Immaginate di riscrivere sempre lo stesso metodo per ogni nuova missione che viene eseguita.