Ruby

commenti

Commento su una riga

#commenti

Commento su più righe

=begin

…

=end

OSS il return nelle funzioni è implicito, ovvero ritornano il valore dell'ultima espressione

NULL => NIL

Assegnare valore a variabile solo se diversa da nil

variabile **||=** *valore*

Input / output da terminale

Input da terminale

variable **= gets.chomp**

; si usa solo per scrivere più comandi sulla stessa linea

Output da terminale

print *variable / "stringa"*

p *variable / "stringa"*

puts *variable / "stringa"* (aggiunge new line)

Interpolazione stringa

Per inserire variabili in stringhe

" #{ *variable* **} "** (funziona solo con " non ')

Metodi

Per invocare un metodo si usa

*variable***.***metodo*

Inoltre esistono

*variable***.***metodo***?** => ritorna condizione true / false

*variable***.***metodo***!** => modifica direttamente oggetto invece di creare copia

Conversione classi

Conversione a intero / float

*variable***.to_i** / *variabile***.to_f**

Conversione a stringa

*variable***.to_s**

Conversione a simbolo

variable.**to_sym**

Conversione a array

(*variable1***..***variable2***).to_a**

Simboli

Classe speciale di Ruby simile a stringa

:*simbolo*

Comparazione tipo variabili

*variabile***.eql?** *Variabile*

Controllo classe

*variabile***.is_a?(***Classe***)**

Control flow

If *condizione* … **elsif** *condizione* … … **else** … **end**	**case** *variabile_condizioni* **when** *condizione* … **when** *condizione* … … **end**

linea_codice **if** *condizione*
linea_codice **unless** *condizione*

Equivalenti	
{ … }	do … end

Cicli

loop do … **end**	Equivale a while(true)
while *condizione* **do** … **end**	
until *condizione* **do** … **end**	Come while ma itera finchè la condizione è false, termina quando diventa true
for *var* **in** *n1…n2* **do** **…** **end**	- var rappresenta variabile corrente - Usando **..** anche l'ultimo valore è compreso nel loop - È possibile usare un array o hash al posto si n1…n2 per iterare sui sui elementi
n**.times do** **…** **end**	Itera n volte - si può mettere un oggetto al posto di n in modo che times iteri un numero pari agli elementi contenuti nell'oggetto
*object***.each do** |*var*| … **end**	Itera su ogni variabile del array o hash, la variabile in |var| rappresenta la variabile corrente. Nel caso di hash si usa |key, value|
*object***.collect do** |*var*| … **end**	come each ma permette di creare una copia dell'array o di modificarlo nel caso di collect!

break if condizione => esce dal ciclo quando si verifica condizione
next if condizione => salta iterazione se si verifica condizione (equivalente continue)

Funzioni (metodi)

def *nome_metodo* **(** *param1, param2 = valore, *param3 …* **)**
….
end

param **=** *valore* => definisce valore di default
param => si possono assegnare più valori al parametro

Hash

Creazione nuovo hash

variable_hash **= Hash.new(** *valore_default* **)**

OSS è possibile inserire un valore di default da assegnare ad ogni nuovo elemento

variable_hash **= {** *key1* **=>** *val1*, *key2* **=>** *val2*, … **}**
variable_hash **= {** *key1***:** *val1*, *key2***:** *val2*, … **}**

OSS usando i simboli è possibile usare anche la seconda notazione

Accede ad un elemento

*variable_hash***[***key_elemento* **]**

OSS simili per array

Aggiungere nuovo elemento

*variable_hash***[:***key_elemento* **] =** *valore_elemento*

Rimozione di un elemento

*variable_hash***.delete(** *elemento* **)**

Controllare presenza elemento

*variable_hash***.include?(** *elemento* **)**

Selezionare elementi con determinate proprietà in hash

*variable_hash***.select { |**key**,** *value***|** *condizione* **}**

Cicli speciali

*variable_hash***.each_key** => itera solo su chiavi
*variable_hash***.each_value** => itera solo su valori

Yield

Permette di collegare blocchi di codice alle funzioni del tipo

metodo **{ |***var1***, … |** *codice*… **}**

Quindi il codice della funzione conterrà nella sua definizione

yield(*param1***, …)** => param corrisponde a |var|

Ogni volta che la funzione incontra yield esegue il codice contenuto nel blocco

Procs

Permettono di salvare blocchi di codice da passare ai metodi

variable_proc **= Proc.new { |***var***|** *codice*… **}**

Una volta creati vengono passati ai metodi tramite

*metodo***($***variabile_proc***)** => & può essere usato anche per convertire simboli (di metodi) in procedure

Oppure si può chiamare direttamente la procedura con

*variable_proc***.call**

Lambda

Simile a procs nel funzionamento ed uguale nei metodi di chiamata

variabile_lambda **= lambda { |***var***|** *codice…* **}**

Differenze tra procs e lambda:

- lambda ritorna un errore se il numero di argomenti passati è diverso da quelli dichiarati, i procs ignorano l'errore
- Quando lambda ritorna ritorna il controllo alla funzione chiamante facendole eseguire le altre istruzioni, proc esce definitivamente ritornando il suo valore di ritorno

Creazione classi

```
Class NomeClasse
    include NomeModulo, …                          => per usare moduli
    extends NomeModulo, …

    $global_variable
    …
    @@class_variable1
    @istance_variable1
    …

    attr_reader :param1, :param2, …                => getters
    attr_writer :param1, :param2, …                => setters
    attr_accessor :param1, :param2, …              => getters e setters insieme

    def initialize ( param1, param2, … )
        @istance_variable1 = paramk
        …
        codice …
    end

    def metodo1 ( param1, param2, … )              => definisce metodo di
        codice …                                      istanza
    end

    def self.metodo1 ( param1, param2, … )         => definisce metodo di
        codice …                                      classe
    end

    private                                        => iniziano metodi privati
    def metodo1 ( param1, param2, … )
        codice …
    end

end
```

Visibilità variabili

$	=> globale (visibile anche all'esterno della classe)
@@	=> di classe (stessa per tutte le istanze)
@	=> d'istanza (unica per l'istanza)

ATTENZIONE
Il simbolo di visibilità diventa parte integrante della variabile

Per creare un istanza della classe

variable **=** *NomeClasse***.new(** *param1, param2, …* **)**

Ereditarietà

Class *NomeClasse* **<** *NomeClasseGenitore*
    codice…

    def initialize (*param1, param2, …* **)**
        codice …
        super => i parametri vengono assegnati come nella classe Genitore
    end

    def *metodo1* **(** *param1, param2, …* **)**
        codice …
        super => utilizza codice metodo nella classe genitore
    end
    …

end

Moduli

Simili alle classi ma non possono essere istanziati o avere sottoclassi, servono solo come contenitori di metodi e costanti

Module *NomeModulo*
    COSTANT **=** *val* => le costanti tutte maiuscole senza $
    …

    def self.*metodo1* **(** *param1, param2, …* **)** => definisce metodo di classe
        codice …
    end
    …

end

Per prelevare una costante da un modulo specifico si usa lo **scope resolution operator**
    NomeModulo **::** *Costante*

Nel caso in cui si volesse importare tutto il modulo (o gemma) in un file, si usa
    require '*NomeModulo***'**

Oppure lo si include nella classe con
- **Include** => per usare metodi d'istanza
- **Extend** => per usare metodi di classe

Installazione gemma

Su terminale
    gem install *nome_gemma*

Extra

“*string***” * *n*** => la stringa verrà riscritta *n* volte

x **=** *n1***..***n2* => tipo range convertibile in array tramite *var*.to_a

Time.now => permette di avere data e orario corrente

Gemme utili

bcrypt => per criptaggio password