Introduzione al linguaggio Ruby

- È un linguaggio di scripting, di solito considerato distinto da un linguaggio di programmazione
- il computer legge le istruzioni e le esegue immediatamente, invece di leggerle tutte e poi eseguirle
- le istruzioni cosi' sono meno ottimizzate, ma lo sviluppo è più rapido
- al contrario del C++, i tipi delle variabili non sono predeterminati
- i tipi sono determinati al momento dell'esecuzione
- anche qui esistono gli array, ma l'allocazione e deallocazione della memoria avviene automaticamente
- anche qui esiste la programmazione a oggetti con le classi, e in più ci sono anche i moduli
- esiste l'ereditarietà, ma solo singola
- uno dei punti forti sono i cicli
- un'altro, il modo di specificare i range

Tipizzazione dinamica

- Voglio che x contenga un intero
- non la dichiaro da nessuna parte, ma la utilizzo, ad esempio, scrivendo $x = 1$
- Ruby capisce che x è intero.
- se scrivevo $x = 3.14$ sarebbe stata una variabile reale
- scrivendo $x =$ "La mia stringa preferita", x sarebbe considerata una stringa, e così via
- Assegnare ad x una stringa, dopo avergli assegnato un intero, non è un errore

Range

- Una variabile compresa tra 3 e 7 (inclusi) è nel range

```
3..7 (punto punto)
```

- se il 7 è escluso

```
3...7 (punto punto punto)
```

- Ad esempio, un programma per calcolare il fattoriale di n si può scrivere come

```
i = 1
for k in 1..n
   i = i*k
end
```

Funzioni

- Si definiscono ponendo le istruzioni tra def ed end;
- ogni istruzione termina con una nuova linea oppure, se più istruzioni stanno sulla stessa linea, con un punto e virgola;
- i parametri della funzione possono stare tra parentesi oppure no, secondo i gusti del programmatore;
- si può restituire anche più di un valore;
- esempio: funzione che calcola $\sin(x)$ e $\cos(x)$

```
def sincos x
   return Math.sin(x), Math.cos(x)
end
a, b = sincos 1.57
```

Istruzioni condizionali

- assomigliano abbastanza a quelle del C++
- anche queste sono terminate da *end*

```
if x > 0
   ...fa qualcosa...
elsif x < -1  # non "elseif"
   ...altro...

else
   ...altro ancora...
end
```

```
case k
when 1..10
  ....
# non e' necessario "break"
when 11...100, 144
   ....
else
   ....
end
```

- esiste anche *unless*, che è il contrario di *if*

Cicli

- si possono fare cicli for, while e until

```
for j in 1..100
  x += 1
end
```

```
while x <= 100
   x = x+1
end
```

```
until x> 100
   x = x+1
end
```

- si possono anche fare cicli infiniti con loop do ...end;

- esistono le istruzioni break e next, analoghe a break e continue del C++, e l'istruzione redo che fa rifare l'ultima iterazione senza aumentare l'indice (come farebbe continue);

- non si può scrivere i++, ma solo i=i+1;

- le variabili definite all'interno del ciclo saranno visibili anche quando il ciclo è terminato (al contrario del C++).

Array

- Partono dall'indice zero e si dichiarano in modo simile al C++

```
arr = Array.new oppure arr = []
```

- lo stesso array può contenere dati di vari tipi: posso scrivere

```
arr[0] = 14
arr[1] = "E ora?"
arr[5] = 3.14
```

- il tipo di contenuto può essere modificato durante l'esecuzione.

Classi

- Possono essere definite tra le istruzioni class e end;
- hanno istanze (oggetti) e metodi come nel C++;
- esistono variabili locali, variabili associate a un'istanza, variabili associate ad una classe. Esempi sono:

```
# Questo e' un commento
class Miotest          # nome sempre in maiuscolo
   # due @@ indicano variabili di classe
   @@variabile_di_classe
   def initialize (i)
      @i = 1           # un @ indica variabile d'istanza
      for j in 1..i    # ciclo
         @i *= j       # @i non e' i
      end              # fine ciclo
   end                 # fine funzione
end                    # fine classe
```

- il nome della classe va sempre maiuscolo: questa è anche la convenzione per le costanti;
- il costruttore si chiama sempre *initialize*;
- i metodi di una classe sono *private* se visibili solo dall'oggetto su cui sono chiamati, *protected* se visibili da tutti gli oggetti della classe e *public* se sono visibili da tutti;
- non esistono funzioni *friend*

Suggerimenti vari

- Le istruzioni condizionali if e unless possono essere postfisse

```
x = x+1 if x > 0
x = 1./y unless y==0
```

- un range può essere trasformato in array e usato come contatore

```
arr = (1..10).to_a        # converte il range in array
arr.each {|i| print i}    # stampa gli interi
```

- questo mostra anche l'uso di un blocco con le istruzioni racchiuse tra parentesi graffe
- esistono funzioni per operare su stringhe

```
s = "strINGA"
s.capitalize            # "Stringa"
s.upcase                # "STRINGA"
s.downcase              # "stringa"
s.capitalize.reverse    # "agnirtS"
s                       # "Stringa"
s.capitalize!.reverse!  # "agnirtS"
s                       # "agnirtS"
```

Operazioni sugli array

- arr.reverse (o reverse!, se voglio il metodo distruttivo)
- arr.map {|i| 3*i+1} (anche arr.map!) fa una stessa operazione su tutti gli elementi
- arr.select{|i| i%7==0} seleziona gli elementi che soddisfano una condizione
- arr.reject{|i| i%7==0} seleziona gli elementi che NON soddisfano una condizione
- a.inject{|somma, i| somma+i} oppure a.inject(1){|prod, i| prod*i} danno rispettivamente la somma ed il prodotto degli elementi di un array
- a.sort ordina l'array

Utility

- È possibile provare i programmi usando una shell interattiva: per farlo lanciare dal terminale il comando irb;

- ci si può documentare su di una funzione, ancora da terminale, con il comando ri nome_funzione;

- il comando rdoc programma.rb, lanciato nella cartella in cui si trova il programma, crea una documentazione in elegante formato html in una sottocartella di nome doc.