UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

7. Programmazione funzionale in Scala

3 Esercizi risolti

Tutorato di
Programmazione Avanzata

RELATORE
Imberti Federico

SEDE
Dalmine, BG

Key-points della teoria: Programmazione funzionale

Dire alla macchina “**cosa fare**” piuttosto che “come farlo”

- Contare numeri pari in un array: Java Vs Scala

```java
int nEven = 0;
for(int i = 0; i < array.size; i++)
  if(array[i] % 2 == 0)
    nEven++;
```

```scala
array
  .filter(_ % 2 ==0)
  .size;
```

- Stampare il nome di tutti gli studenti che vivono a Dalmine e fanno l’esame di PA: SQL

```sql
SELECT s.name
FROM student as s join exam_ap as e on s.id=e.id
WHERE s.residence=”Dalmine”
```

Key-points della teoria: Programmazione funzionale

- Programmazione funzione si basa sull'applicazione di funzioni pure e sull'immutabilità dei dati

- Molti linguaggi di programmazione moderni sono **multiparadigma**
  - In Java posso ricorrere alla programmazione funzionale (framework "stream")

- Scala viene considerato un linguaggio funzionale "impuro"
  - Integra caratteristiche che non sono parte della programmazione funzionale, come gli **oggetti** e le **variabili**;
  - Possibile poiché è costruito sulla stessa **JVM** su cui si basa Java, infatti i due sono interoperabili (in Scala posso usare librerie di Java e in Java posso ricorrere alle funzioni di Scala);
  - "Scala" viene da "scalability", questo linguaggio permette infatti di lavorare velocemente su grandi moli di dati: i big data.

Key-points della teoria: Scala quick hits

- **Tipizzato staticamente**: type safety rafforzata;

- Possibile sia come **type-inferred** (tipo Python) che tipizzato esplicitamente
  - `Var a:Int  //valore che può essere mutato`
  - `Val b    //valore immutabile type-inferred`

- Non esistono tipi primitivi poiché tutti i tipi sono wrappati attorno a classi Java

- Esiste “**null**” ma viene usato solo per interoperare con Java
  - Un tipo ritornato può tuttavia essere “Optional”: `def foo(): Option[Int] = { … }`

Key-points della teoria: Scala quick hits

- “Tutto deve restituire qualcosa”
  - Al posto di “void” abbiamo “**Unit**”: `def hi():Unit = { println(“hi”) }`

- Possiamo passare parametri:
  - **By-name**: valuto solo i parametri che usiamo, ma tutte le volte che vengono usati

```
def foo(a:=>Int, b:=>Float):Unit = {...}
```

  - **By-value**: valuto sempre tutti i parametri alla chiamata della funzione

```
def foo(a:Int, b:Float):Unit = {...}
```

- Polimorfismo implementato con **duck-typing**
  - *“If it walks like a duck and it quacks like a duck then it is duck”*

Key-points della teoria: Composizione di funzioni

Così come in matematica possiamo comporre due funzioni, cioè applicare il risultato di una nell'altra. Possibile perché le funzioni in Scala sono first-class.

```
def sqr(x: Int) = x * x
def cube(x: Int) = x * x * x

def squareOfTheCube(x: Int) = sqr(cube(x))

//Oppure definendo una funzione che compone funzioni al suo interno
def compose(f: Int=>Int, g:Int=>Int) : Int=>Int = x => {f(g(x))}
```

Key-points della teoria: Currying

Trasformare una funzione che prende *n* parametri in una serie di *n* funzioni che prendono ognuna un solo parametro.

```
def sumOfTwoNums(x:Int, y:Int):Int = {x + y}

def sumOfTwoNumsCurred(x:Int): Int => Int = y => x + y

def sumOfThreeNums(x:Int): Int => (Int => Int) = y => (z => x + y + z)
```

```
def concatenate(l1:List[Int], l2:List[Int]):List[Int] = l1:::l2

def concatenateCurred(l1:List[Int]): List[Int]=>List[Int] = l2 => l1:::l2
```

Key-points della teoria: metodi essenziali

- .map(): applica una funzione a tutti gli elementi di una collezione;
- .filter(): filtra una collezione in base a un predicato;
- .reduce(): riduce una collezione da una serie di elementi a uno solo (e.g. la somma);
- .forAll(): verifica se un predicato è vero per ogni elemento di una collezione;
- .partition(): divide una collezione in base a un predicato;
- .exists(): ritorna true quando esiste almeno un elemento in una collezione che soddisfi un predicato;
- …

Key-points della teoria: further readings

- Exploring Scala (file che dimostrano alcune caratteristiche di Scala)

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Domandine?

Esercizio 1/3

a. Definire in Scala la funzione mult6 che dato un numero intero lo moltiplica per sei, e la funzione mult che dati due numeri interi li moltiplica.
b. Definire la funzione magic che data una funzione *f:Integer -> Integer* e una funzione *g: Integer X Integer -> Integer* restituisce la funzione *g(f(x),f(y))*.
   Dati due numeri interi, utilizzare la funzione magic in combinazione con le funzioni definite in precedenza per moltiplicare i numeri interi per sei e poi moltiplicarli tra di loro.
c. Definire una list di interi contenente gli interi 1,2,3. Moltiplicare ogni elemento della lista per due, utilizzando una funzione anonima
d. Utilizzare map-reduce per sommare la lista dei valori ottenuti al punto (c).

Esercizio 2/3

a. Definire in Scala la funzione f che data una stringa restituisce un'altra stringa contenente i caratteri in posizione 3, 6 e 9 o la stringa vuota se la stringa passata come parametro e' nulla o ha meno di 9 caratteri.
b. Definire la funzione g che data una stringa come parametro restituisce la stringa ottenuta concatenando la stringa a se stessa.
c. Definire la funzione h: String->String che data una funzione l:String -> String e una funzione m: String -> String restituisce la funzione m(l(x)).
d. Definire la funzione "k" utilizzando la funzione "h" passandogli come input le funzioni "f" e "g"definite in precedenza. Utilizzare la funzione "k" sulla stringa "oloboslorat"
e. Definire una lista di stringhe contenente le stringhe "oloboslorat", "olocosiorat", "olomosaor!t". Applicare ad ogni elemento della lista la funzione "k".
f. Utilizzare map-reduce per concatenare la lista delle stringhe ottenute al punto (e).

Esercizio 3/3

a. Definire in Scala la funzione raddoppia: Int=>Int che restituisce il doppio di un valore intero passato come parametro.
b. Definire la funzione somma che data una funzione f: Int=>Int restituisce una funzione che prende due parametri interi (a e b, tali che a<b) e restituisce la somma di f(x) per ogni x compreso tra a e b (a e b inclusi).
c. Definire la funzione raddoppiaESomma che applica la funzione raddoppia e la funzione somma
d. Utilizzare la funzione raddoppiaESomma passando come parametri a=4 e b=6.
e. Definire una lista di interi contenente i valori 1,2,3,4. Applicare la funzione raddoppia per ogni elemento della lista.
f. Utilizzare map-reduce per sommare i valori al punto (e).