Corso di Linguaggi di Programmazione

Lezione 5

Alberto Ceselli
alberto.ceselli@unimi.it

Università degli Studi di Milano

12 Marzo 2013

Le funzioni sono oggetti di prima classe in Scheme

In Scheme le funzioni sono trattate allo stesso modo degli altri oggetti (numeri, liste, stringhe, simboli, ecc.):

- le funzioni possono prendere *funzioni come argomento*
- la forma speciale `lambda` *permette di costruire* funzioni
- le funzioni possono *restituire funzioni*

Funzione come argomento

- Uno dei parametri nella definizione della funzione rappresenta una funzione, e come tale viene utilizzato all'interno del corpo
  - `(define (function-application nome-fun x y) (nome-fun x y) )`
- L'associazione tra il parametro e la funzione avviene in chiamata di funzione
  - `(function-application + 3 4)` associa `+` a `nome-fun`
  - `(function-application list 3 4)` associa `list` a `nome-fun`

Esempio1: realizzare un programma Scheme che riceve una funzione ed una lista; se la lista è vuota restituisce '() altrimenti applica la funzione alla lista.
Esempio2: realizzare un programma Scheme che controlla alcune proprietà di coppie di elementi in una lista (alla lavagna).

Funzione predefinita `map`

- La funzione `map` viene utilizzata per applicare una funzione ad *ogni elemento in una lista*.
- Restituisce una *lista* i cui elementi sono il risultato che si ottiene applicando il suo primo argomento (la funzione) a ciascun elemento del secondo argomento (la lista).
- Esempi:
  - `(map odd?  '(2 3 4 5 6))` restituisce `(#f #t #f #t #f)`
  - `(map + '(1 2 3) '(10 20 30))` restituisce `(11 22 33)`
- Non è specificato in quale ordine vengono considerati gli elementi della lista.

Funzione predefinita `apply`

- La funzione predefinita `apply` restituisce il risultato che si ottiene applicando il suo primo argomento al secondo argomento
- Esempi:
  - `(apply + '(7 5))` $\rightarrow$ `12`
  - `(apply max '(3 7 2 9))` $\rightarrow$ `9`
- Limitazione: non funziona se al posto di una funzione viene passato come argomento una *forma normale* (e.g., `and` o `or`)

Funzioni che restituiscono funzioni - `lambda` (1)

- Dal λ-calcolo, dove una funzione $f(x) = x + 2$ viene espressa come $\lambda x.x + 2$, e $f(3)$ come $(\lambda x.x + 2)3$
- **definizione**: `(lambda (lista-argomenti) corpo)`
  - Es.: `(lambda (x) (+ 2 x))`
- **chiamata**: `((lambda (lista-argomenti) corpo) args-formali)`
  - Es.: `((lambda (x) (+ 2 x)) 3)` restituisce 5
- **definizione funzione con nome**:

```
(define nome-funzione
(lambda (lista-argomenti) corpo))
```

  - Es.: `(define somma2 (lambda (x) (+ 2 x)))`

Funzioni che definiscono funzioni - `lambda` (2)

Utilizzo della `lambda`-expression per definire funzioni (anche senza nome):

- come argomento a funzione
  - `(map (lambda (num) (>= num 10)) '(15 17) )`
- per creare funzioni che sono la combinazione di `and` oppure `or` di altre funzioni
  - `(lambda (element) (and (number? element)(positive? element)) )`

Esempio: (1) realizzare un programma Scheme che, presi due caratteri, restituisce la funzione *sostituzione* del primo carattere con il secondo (2) realizzare un programma Scheme che, presi due caratteri ed una lista, sostituisce ogni occorrenza del primo carattere nella lista con un'occorrenza del secondo (alla lavagna).

Tipi di dato:

- int: 3
- float: 2.5
- str: 'abc', "abc"
- list: [0, 1, 2], [0, 1, 'the']
- tuple: (0, 1, 2), (0, 1, 'the')
- dict: {'a': 1, 'mamma':'anna', 2:'b'}

- Python *usa gli spazi per identificare blocchi di codice*
- si indica l'inizio di un blocco con ':' (vedi codice)
- costrutto *if* simile al C
- costrutto *switch ... case* sostituito con *if ... elif*
- due costrutti di iterazione: *while* e *for*

(codice di esempio)

Alcune operazioni su liste

- *lst.append(x)*: include l'elemento x in coda a lst
- *lst.extend(L)*: include la lista L in coda a lst
- *list.remove(x)*: rimuove la prima occorrenza di x in lst
- *list.count(x)*: restituisce il numero di occorrenze di x in lst

Definizione di funzioni

Definizione di funzioni

- parola chiave *def*
- def <nome funzione>(<argomenti>):
- statement *return* al termine della funzione

(codice di esempio)

Funzioni come oggetti di primo livello

Le funzioni sono *oggetti di primo livello*:

- possono essere *passate come argomenti* ad altre funzioni
- possono essere *costruite* e restituite da altre funzioni

Funzioni che costruiscono funzioni

- parola chiave *lambda*
- f = lambda <argomenti>: <corpo della funzione da costruire>
- a questo punto f contiene una *funzione*

(codice di esempio)

Le funzioni Python possono *restituire un numero arbitrario di valori*

```
def computexy()
...
return x,y

>> a,b = computexy()
```