Scheme: liste e vettori

Liste e coppie 2451
Dichiarazione di una lista 2454
Caratteristiche esteriori di una lista 2454
Operazioni fondamentali con le liste 2455
Funzioni tipiche sulle liste 2458
Vettori 2462
Strutture di controllo applicate alle liste 2463
Funzione «apply» 2463
Funzione «map» 2464
Funzione «for-each» 2464
Riferimenti 2465

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

Scheme dispone di due strutture di dati particolari: liste e vettori. Le liste sono una sequenza di elementi a cui si accede con una certa difficoltà, senza la possibilità di utilizzare un indice, mentre i vettori sono l'equivalente degli array degli altri linguaggi.

Liste e coppie

La lista è la struttura di dati fondamentale di Scheme. In questo linguaggio, le stesse istruzioni (le chiamate delle funzioni) sono espresse in forma di lista:

```
(elemento...)
```

La lista è un elenco di elementi ordinati. Gli elementi di una lista possono essere oggetti di qualunque tipo, comprese altre liste. Ci sono molte situazioni in cui i parametri di una funzione di Scheme sono delle liste; per esempio la dichiarazione di una funzione, attraverso '**`define`**':

```
(define (nome_funzione elenco_parametri_formali)
    corpo
)
```

Come si vede, il primo parametro della funzione '**`define`**' è una lista, in cui il primo elemento è il nome della funzione che si crea, mentre gli elementi successivi sono la descrizione dei parametri formali.

Le liste vuote, sono rappresentate da una coppia di parentesi aperta e chiusa, '**`()`**', rappresentando degli oggetti speciali nella filosofia di Scheme.

Tabella u129.1. Elenco di alcune funzioni specifiche per la gestione delle stringhe.

Funzione	Descrizione
`(list?` ***oggetto***`)`	*Vero* se l'oggetto è una lista.
`(pair?` ***oggetto***`)`	*Vero* se l'oggetto è una coppia (una lista non vuota).
`(null?` ***lista***`)`	*Vero* se la lista è vuota.
`(length` ***lista***`)`	Restituisce il numero di elementi della lista.
`(car` ***lista***`)`	Restituisce il primo elemento di una lista.

Funzione	Descrizione
`(cdr` ***lista***`)`	Restituisce una lista da cui è stato tolto il primo elemento.
`(cadr` ***lista***`)`	`(car (cdr` ***lista***`))`
`(cddr` ***lista***`)`	`(cdr (cdr` ***lista***`))`
`(caadr` ***lista***`)`	`(car (car (cdr` ***lista***`)))`
`(caddr` ***lista***`)`	`(car (cdr (cdr` ***lista***`)))`
`(cons` ***elemento lista***`)`	Restituisce una lista in cui inserisce al primo posto l'elemento indicato.
`(list` ***elemento***...`)`	Restituisce una lista composta dagli elementi indicati.
`(append` ***lista lista***`)`	Restituisce una lista composta dagli elementi delle due liste indicate.
`(reverse` ***lista***`)`	Restituisce una lista con gli elementi in ordine inverso.
`(set-car!` ***lista oggetto***`)`	Memorizza nella prima posizione della lista l'oggetto indicato.
`(set-cdr!` ***lista oggetto***`)`	Memorizza nella parte successiva al primo elemento l'oggetto indicato.
`(list-tail` ***lista k***`)`	Restituisce una lista in cui mancano i primi ***k*** elementi.
`(list-ref` ***lista k***`)`	Restituisce l'elemento (***k*** + 1)-esimo della lista.
`(vector->list` ***vettore***`)`	Converte il vettore in lista.
`(list->vector` ***list***`)`	Converte la lista in vettore.

Dichiarazione di una lista

«

La dichiarazione di una lista avviene nello stesso modo in cui si dichiara una variabile normale:

```
(define variabile lista_costante)
```

Tuttavia, occorre tenere presente che una lista può essere interpretata come la chiamata di una funzione e come tale verrebbe intesa in questa situazione. Per evitare che ciò avvenga, la si indica attraverso un'espressione costante, cioè la si fa precedere da un apostrofo, o la si inserisce in una funzione '**`quote`**'. L'esempio seguente dichiara la lista '**`lis`**' composta dall'elenco dei numeri interi da uno a sei:

```
(define lis '(1 2 3 4 5 6))
```

In questo caso, se la lista non venisse indicata con l'apostrofo, si otterrebbe la valutazione della lista stessa, prima dell'inizializzazione della variabile '**`lis`**', provocando un errore, dal momento che l'oggetto '**`1`**' (uno) non esiste.

Caratteristiche esteriori di una lista

«

Le caratteristiche esteriori di una lista sono semplicemente la lunghezza, espressa in numero di elementi, e il fatto che contengano o meno qualcosa. Per verificare queste caratteristiche sono disponibili due funzioni, '**`null?`**' e '**`length`**', che richiedono come argomento una lista. Si osservino gli esempi seguenti.

```
; dichiara la lista «lis»
(define lis (1 2 3 4 5 6)

; verifica se la lista «lis» è vuota
(null? lis)                                   ===> #f

; calcola la lunghezza della lista
(length lis)                                  ===> 6
```

Se fosse stata fornita la lista in modo letterale, senza la variabile '**lis**', la stessa cosa avrebbe dovuto essere scritta nel modo seguente:

```
; verifica se la lista è vuota
(null? '(1 2 3 4 5 6))                        ===> #f

; calcola la lunghezza della lista
(length '(1 2 3 4 5 6))                       ===> 6
```

Operazioni fondamentali con le liste

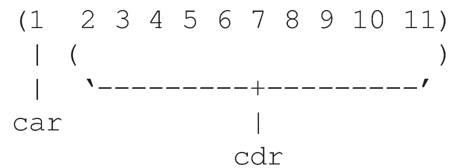
«

L'accesso agli elementi singoli di una lista è un'impresa piuttosto complessa che si attua fondamentalmente con le funzioni '**car**' e '**cdr**'. A queste due si affianca anche '**cons**', il cui scopo è quello di «costruire» una lista.

Per comprendere il senso di queste funzioni, occorre tenere presente che per Scheme una lista è una ***coppia*** composta dal primo elemento, ovvero l'elemento '**car**', e dalla parte restante, ovvero la parte '**cdr**'.

Per la precisione, una coppia è una lista, mentre la lista vuota non è una coppia. La lista contenente un solo elemento, è la composizione dell'unico elemento a disposizione e della lista vuota.

Figura u129.5. La parte «car» e la parte «cdr» che compongono le liste di Scheme.




```
(car lista)
```

```
(cdr lista)
```

Le due funzioni '**car**' e '**cdr**' hanno come argomento una lista, della quale restituiscono, rispettivamente, il primo elemento e la lista restante quando si elimina il primo elemento. Si osservino gli esempi seguenti.[1]

```
(car '(1 2 3 4 5 6))           ===> 1
(cdr '(1 2 3 4 5 6))           ===> (2 3 4 5 6)
```

Data l'idea che ha Scheme sulle liste, la funzione '**cons**' crea una lista a partire dalle sue parti '**car**' e '**cdr**':

```
(cons elemento_car lista_cdr)
```

In altri termini, '**cons**' aggiunge un elemento all'inizio della lista indicata come secondo argomento. Si osservi l'esempio.

```
(cons 0 '(1 2 3 4 5 6))        ===> (0 1 2 3 4 5 6)
```

Le tre funzioni '**car**', '**cdr**' e '**cons**' si completano a vicenda, in base alla relazione schematizzata dalla figura u129.9.

Se viene fornita una lista come primo argomento della funzione '**car**', questa viene inserita come primo elemento della lista risultante.

```
(cons '(0 1 2) '(1 2 3 4 5 6))          ===> ((0 1 2) 1 2 3 4 5 6)
```

Figura u129.9. Relazione che lega le funzioni '**car**', '**cdr**' e '**cons**'. In particolare, «x» e «y» sono liste non vuote; «a» è un elemento ipotetico di una lista.

```
(cons (car x) (cdr x)) = x
(car (cons a y)) = a
(cdr (cons a y)) = y
```

Altri modi per creare una lista sono dati dalle funzioni '**list**' e '**append**'.

(list ***elemento***...)

(append ***lista*** ***lista***)

La funzione '**list**' restituisce una lista composta dai suoi argomenti (se non si vuole che questi siano valutati prima, occorre ricordare di usare l'apostrofo); la funzione '**append**' restituisce una lista composta dagli elementi delle due liste indicate come argomento (se le liste vengono fornite in modo letterale, occorre ricordare di usare l'apostrofo, per evitare che vengano valutate come funzioni).

```
(list 1 2 3 4 5 6)                      ===> (1 2 3 4 5 6)
(append '(1 2 3 4 5 6) '(7 8 9))        ===> (1 2 3 4 5 6 7 8 9)
```

Per verificare che un oggetto sia una lista, è disponibile il predicato '**list?**'. Si osservi l'esempio seguente, con il quale si inten-

de ribadire il significato dell'apostrofo per evitare che una lista sia interpretata come funzione:

```
(define a (+ 1 2))
a                                ===> 3

(define b '(+ 1 2))
b                                ===> (+ 1 2)

(list? a)                        ===> #f
(list? b)                        ===> #t
```

Funzioni tipiche sulle liste

«

Dal momento che con le liste di Scheme non è disponibile un accesso diretto all'elemento ***n***-esimo, se non attraverso la funzione di libreria '**`list-ref`**', è importante imparare a gestire le funzioni elementari già mostrate nella sezione precedente.

- Calcolo della lunghezza di una lista:

```
(define (lunghezza x)
    (if (null? x)
        ; se la lista è vuota, restituisce zero
        0
        ; altrimenti esegue una chiamata ricorsiva
        (+ 1 (lunghezza (cdr x)))
    )
)
```

- Ricerca dell'elemento ***i***-esimo, dove il primo è il numero uno (si veda anche la funzione di libreria '**`list-ref`**', descritta più avanti in questa serie di esempi):

```
(define (i-esimo-elemento i x)
    ; «i» è l'indice, «x» è la lista
    (if (null? x)
        ; la lista è più corta di «i» elementi
        "errore: la lista è troppo corta"
        ; altrimenti procede
        (if (= i 1)
            ; se si tratta del primo elemento, basta la funzione
            ; car per prelevarlo
            (car x)
            ; altrimenti, si utilizza una chiamata ricorsiva
            (i-esimo-elemento (- i 1) (cdr x))
        )
    )
)
```

- Estrae l'ultimo elemento:

```
(define (ultimo x)
    (if (null? x)
        ; la lista è vuota e questo è un errore
        "errore: la lista è vuota"
        ; altrimenti si occupa di estrarre l'ultimo elemento
        (if (null? (cdr x))
            ; se si tratta di una lista contenente un solo elemento,
            ; restituisce il primo e unico di questa
            (car x)
            ; altrimenti utilizza una chiamata ricorsiva
            (ultimo (cdr x))
        )
    )
)
```

- Elimina l'ultimo elemento:

```
(define (elimina-ultimo x)
    (if (null? x)
        ; la lista è vuota e questo è un errore
        "errore: la lista è vuota"
        ; altrimenti si occupa di eliminare l'ultimo elemento
        (if (null? (cdr x))
            ; se si tratta di una lista contenente un solo elemento,
            ; restituisce la lista vuota
            '()
            ; altrimenti utilizza una chiamata ricorsiva per comporre
            ; una lista senza l'ultimo elemento
            (cons (car x) (elimina-ultimo (cdr x)))
        )
    )
)
```

- Restituisce la parte finale della lista, escludendo alcuni elementi iniziali. Si tratta precisamente di una funzione di libreria di Scheme, denominata '**`list-tail`**':

```
(define (list-tail x k)
    (if (zero? k)
        ; se «k» è pari a zero, viene restituita tutta la lista
        x
        ; altrimenti occorre eliminare k-1 elementi iniziali
        ; da (cdr x)
        (list-tail (cdr x) (- k 1))
    )
)
```

- Ricerca del (***k***+1)-esimo elemento di una lista. Si tratta di una funzione di libreria di Scheme, denominata '**`list-ref`**' (in pratica, l'indice ***k*** viene usato in modo da indicare il primo elemento con il numero zero):

```
(define (list-ref x k)
    ; si limita a restituire il primo elemento ottenuto
    ; dalla funzione list-tail
    (car (list-tail x k))
)
```

- Scansione di una lista in modo da restituire un'altra lista, contenente i valori restituiti dalla chiamata di una funzione data per

ogni elemento della lista. Si tratta di una semplificazione della funzione di libreria '**`map`**', in questo caso con la possibilità di indicare una sola lista di valori di partenza:

```
(define (map1 f x)
    ; «f» è la funzione da applicare agli elementi della lista «x»
    (if (null? x)
        ; la lista è vuota e restituisce un'altra lista vuota
        '()
        ; altrimenti compone la lista da restituire
        (cons (f (car x)) (map1 f (cdr x)))
    )
)
```

- Descrizione della funzione di libreria '**`append`**':

```
(define (append x y)
    (if (null? x)
        ; se la lista «x» è vuota, restituisce la lista «y»
        y
        ; altrimenti costruisce la lista in modo ricorsivo
        (cons
            (car x)
            (append (cdr x) y)
        )
    )
)
```

- Descrizione della funzione di libreria '**`reverse`**':

```
(define (reverse x)
    (if (null? x)
        ; se la lista «x» è vuota, non c'è nulla da invertire
        '()
        ; altrimenti compone l'inversione con una chiamata ricorsiva
        (append (reverse (cdr x)) (list (car x)))
    )
)
```

Vettori

Scheme gestisce anche i vettori, che sono in pratica gli array dei linguaggi di programmazione normali. Un vettore viene rappresentato in forma costante come una lista preceduta dal simbolo '**#**':

```
#(elemento_1... elemento_n)
```

L'indice dei vettori di Scheme parte da zero. Il funzionamento dei vettori di Scheme non richiede spiegazioni particolari. La tabella u129.21 riassume le funzioni utili con questo tipo di dati.

Tabella u129.21. Elenco di alcune funzioni specifiche per la gestione dei vettori.

Funzione	Descrizione
`(vector? `***oggetto***`)`	*Vero* se l'oggetto è un vettore.
`(make-vector `***k***`)`	Restituisce un vettore di ***k*** elementi indefiniti.
`(make-vector `***k valore***`)`	Restituisce un vettore di ***k*** elementi inizializzati al valore specificato.
`(vector `***elemento***`...)`	Restituisce un vettore degli elementi indicati.
`(vector-length `***vettore***`)`	Restituisce il numero di elementi del vettore.
`(vector-ref `***vettore k***`)`	Restituisce l'elemento nella posizione ***k***, partendo da zero.
`(vector-set! `***vettore k oggetto***`)`	Assegna all'elemento ***k***-esimo l'oggetto indicato.
`(vector->list `***vettore***`)`	Converte il vettore in lista.

Funzione	Descrizione
`(list->vector` ***lista***`)`	Converte la lista in vettore.

Strutture di controllo applicate alle liste

Alcune funzioni tipiche di Scheme servono ad applicare una funzione a un gruppo di valori contenuto in una lista.

Tabella u129.22. Elenco di alcune funzioni specifiche per la scansione degli elementi di una lista, allo scopo di applicarvi una funzione.

Funzione	Descrizione
`(apply` ***funzione lista***`)`	Esegue la funzione utilizzando gli elementi della lista come argomenti.
`(map` ***funzione lista***...`)`	Esegue la funzione iterativamente per gli elementi delle liste.
`(for-each` ***funzione lista***...`)`	Esegue la funzione iterativamente per gli elementi delle liste.

Funzione «apply»

```
(apply funzione lista)
```

La funzione '**`apply`**' esegue una funzione a cui affida gli elementi di una lista come altrettanti argomenti. Si osservi il modello seguente:

```
(apply funzione '(elem_1 elem_2... elem_n))
```

Questo equivale in pratica a:

```
(funzione elem_1 elem_2... elem_n)
```

Per esempio:

```
(apply + '(1 2))                    ===> 3
```

Funzione «map»

```
(map funzione lista...)
```

La funzione '**map**' scandisce una o più liste, tutte con la stessa quantità di elementi, in modo tale che, a ogni ciclo, viene passato alla funzione l'insieme ordinato dell'***i***-esimo elemento di ognuna di queste liste. La funzione restituisce una lista contenente i valori restituiti dalla funzione a ogni ciclo.

Anche se viene rispettato l'ordine delle varie liste, '**dat**' non garantisce che la scansione avvenga dal primo elemento all'ultimo.

L'esempio seguente esegue la somma di una serie di coppie di valori, restituendo la lista dei risultati:

```
(map + '(1 2 3) '(4 5 6))                    ===> (5 7 9)
```

Funzione «for-each»

```
(for-each funzione lista...)
```

La funzione '**`for-each`**' è molto simile a '**`map`**', nel senso che avvia una funzione ripetutamente, quanti sono gli elementi delle liste successive, garantendo di eseguire l'operazione in ordine, secondo la sequenza degli elementi nelle liste. Tuttavia, non restituisce nulla.

Riferimenti

- A. Aaby, *Scheme Tutorial*, 1996
  http://cs.wwc.edu/~cs_dept/KU/PR/Scheme.html
- *R⁵RS -- Revised-5 Report on the Algorithmic Language Scheme*, 1998
  http://www.swiss.ai.mit.edu/~jaffer/r5rs_toc.html
  http://www.swiss.ai.mit.edu/ftpdir/scheme-reports/r5rs.ps.gz

[1] A questo punto si intende ormai chiarito il significato dell'apostrofo posto di fronte a una lista, quando questa non deve essere valutata, prima di essere fornita come argomento di una funzione.