Programmazione funzionale

PROGRAMMA = FUNZIONE

La programmazione funzionale o programmazione orientata alle espressioni tratta espressioni, le funzioni vengono usate come oggetti, ovvero vengono trattate in un programma in modo strettamente analogo alle variabili.

Un programma puramente funzionale viene strutturato definendo un'espressione che racchiude l'obiettivo del programma.

Ogni termine dell'espressione è a sua volta un'affermazione relativa ad una caratteristica del problema (magari incapsulata dentro un'altra espressione) e la valutazione di tutte queste espressioni alla fine porta al risultato.
Il flusso di esecuzione del programma assume la forma di una serie di valutazioni di funzioni matematiche.

L'esempio più vecchio di linguaggio funzionale è il Lisp(List Processor) spesso usato nei progetti di intelligenza artificiale, anche se né il Lisp originale né i Lisp moderni, come il Common Lisp e le sue varianti (Logo, Scheme, Dylan e Clojure) sono puramente funzionali.

Altri linguaggi funzionali sono l'Haskell e i linguaggi della famiglia ML(ML e Ocaml, F# sviluppato da Microsoft all'interno del framework .NET).

Altri linguaggi, come per esempio Ruby, Python, Perl e TCL, possono essere usati in stile funzionale.

Il Lisp è stato ideato nel 1958 da John McCarthy come linguaggio formale, per studiare le equazioni di ricorsione in un modello computazionale.
Il Lisp si presenta con un Prompt di inserimento comandi esattamente come farebbe una shell a comandi.

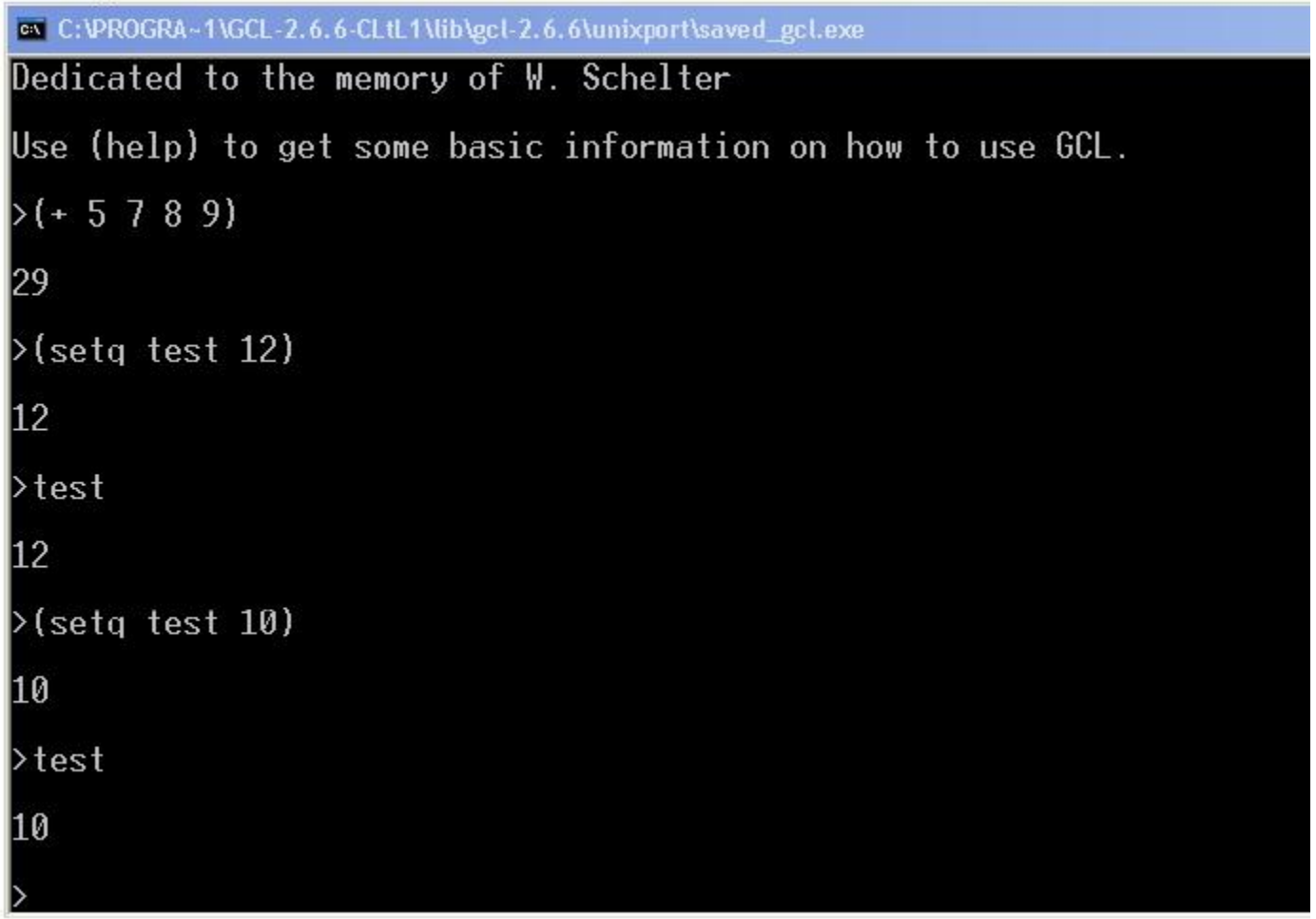
Inserendo una lista composta da elementi separati da spazi tra parentesi ( ) otteniamo l'esecuzione, il processore Lisp valuta il comando inserito, sommando tutti termini che compongono la lista.

Nel Lisp gli oggetti usati sono le S-expression (symbolic expressions) la cui sintassi è:

(comando liste liste liste ...)

Il Lisp non tipizza il dato delle variabili.
Il codice di un programma può essere scritto sottoforma di S-expression, utilizzando la notazione prefissa

Il Lisp valuta il contenuto ed il tipo delle variabili solo quando una funzione esegue una operazione con il contenuto della S-expression.



Variabili e Comandi basilari in Lisp

- Ogni variabile deve possedere un nome (sequenza alfanumerica di caratteri, non solo numeri) che la distingue dalla altre, in modo da poterla utilizzare semplicemente indicandone l'identificativo (nome).
- I dati immagazzinabili in una variabile sono :

  numeri reali, numeri interi, stringhe di testo, liste di valori (cioè variabili contenenti più dati contemporaneamente)

  - Assegnamento: (setq x 5) questa istruzione si limita ad inserire nella variabile di nome x il numero intero 5
  - Scrivere messaggi per l'utente: (print “ciao”)
  - Caricare file: (load "c:/test.lsp")

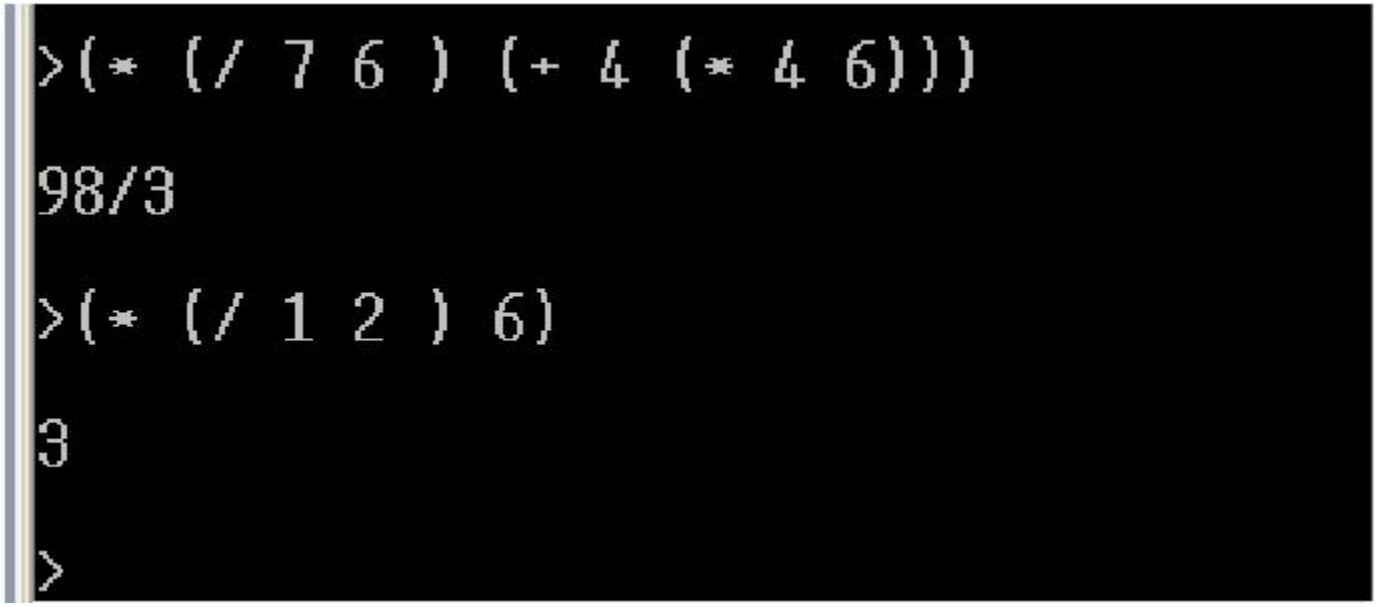
Le funzioni matematiche in Lisp

Lista di alcune funzioni disponibili per l'elaborazione numerica :
+ ,- ,* ,/ , 1+, 1-, abs, gcd, min, max, zerop, atan, log,
cos, sin, exp, sqrt …

Es: (+ 10 7.5) esegue l'addizione tra i numero 10 e 7.5

La sintassi utilizzata: ([funzione] [valore1] [valore2]) è applicabile solamente alla quattro operazioni ”+”, ”-”, “*”, ”/”.

Il Lisp utilizza per gli operatori aritmetici e binari la notazione infissa "polacca di Cambrige".



Espressioni di confronto

•”=” valuta su due o più valori, passati come argomenti, sono uguali, in questo caso ritorna T altrimenti nil.

•”/=” valuta su due o più valori sono diversi, in questo caso ritorna T altrimenti nil.

•”<” e ”>” valutano se un valore è minore o maggiore dell'elemento alla sua destra, in questo caso viene restituito T, altrimenti nil.

•AND OR NOT NULL usuali espressioni booleane
(in particolare “NULL” ritorna T se il suo argomento è nil (falso) e nil se il suo argomento è T (vero).)

Istruzioni di controllo

Istruzione condizionale IF

(if [condizione]
(progn [codice eseguito se condizione = vero])
(progn [codice eseguito se condizione = falso]))

Il costrutto PROGN permette l'esecuzione di più linee di codice.
Se non si utilizza PROGN viene eseguita una sola riga di codice per ognuno dei due stati.

I Cicli

(loop **for** I **from** N1 **to** N2 **do** ...)
(loop **until** Condition do ...)
(loop **while** Condition do ...)

RETURN rompe il loop a qualsiasi punto ritornando un valore come risultato del loop.
FINALLY specifica cosa fare una volta il loop è terminato, di solito è usato per Specificare un valore di ritorno.

Funzioni personalizzate

Una funzione in Lisp così come in tutti gli altri linguaggi di programmazione non è altro che una parte di codice richiamabile attraverso l'utilizzo di un identificatore.

Sintassi:
(defun *NomeFunzione* (*argomenti*) *Corpo*)

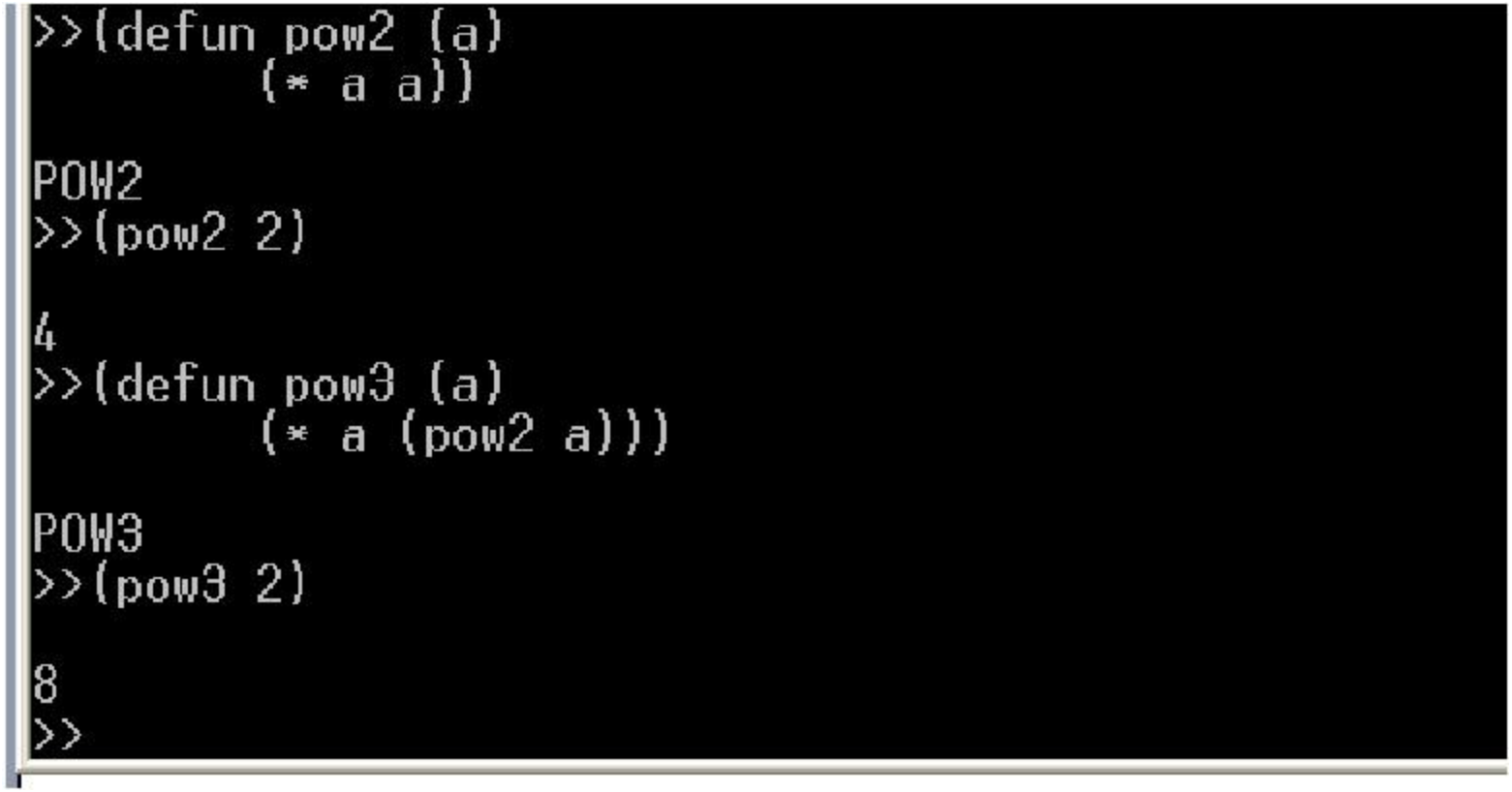
Il valore che la funzione ritorna è il valore dell'ultima istruzione nel *Corpo*

Es1: (defun stampamessaggio() (print "ciao"))
Es2: (defun somma(x) (+ x x))

La funzione somma per x uguale a 3 si richiama ad esempio così: (somma 3)

Esercizio1: scrivere una funzione che calcola la potenza quadrata di un numero
Esercizio2: scrivere una funzione che calcola la potenza cubica di un numero

La funzione per definire una funzione è defun, ad esempio definiamo la funzione pow2 per calcolare il quadrato , e la funzione pow3 per calcolare il cubo utilizzando la funzione pow2.



Per valutare la correttezza di una funzione ricorsiva è possibile mettere sotto traccia una funzione attraverso la funzione trace.

Ad esempio se *funric* è una funzione ricorsiva allora *(trace funric)* farà vedere tutte le chiamate ricorsive.

Esercizi:

- Scrivere funzione per la somma dei primi n numeri
- Scrivere funzione fattoriale ricorsivamente

Soluzione degli esercizi:

- Funzione per la somma dei primi n numeri

```
(defun somma(x)
(setq s 0)
(loop for i from 1 to x do
(setq s (+ s i))
finally (return s))
```

- Funzione fattoriale (ricorsiva)

```
(defun fattr(n)
(if (zerop n)
1
(* n (fattr (1- n)))))
```