LISP (~)

In lisp esistono atomi e cons-cells. Gli atomi sono tutti i simboli e numeri (ma anche stringhe, caratteri), mentre tutto ciò che non è un atomo, viene chiamato conce-cells (quindi una lista). L'insieme di questi due gruppi forma le symbolic Expression (o Sexp's)

Funzioni principali lisp

- Quote: viene utilizzato per permettere ad una funzione di non essere valutata e viene direttamente ritornata. Viene abbreviato con '
- Atom: ritorna true se l'elemento in questione è un atomo, false altrimenti
- Dotimes: simula il ciclo for, infatti permette di ciclare un numero n di volte

- Cond: viene usato nel caso in cui si hanno numerose condizioni da verificare (praticamente uguale allo swich)
- Setq: permette di assegnare degli oggetti a delle variabili
- let: permette di assegnare dei valori a delle variabili locali
- Defparameter: permette di definire una variabile (inoltre, se lo utilizzo con una lista non piatta, mi permette di assegnarla a una nuova e renderla tale)
- Eval: viene chiamata per far valutare una funzione all'interprete (se anteposta al quote, forza la valutazione di una

funzione non valutata)

- Funcall: esegue una chiamata ad una funzione
- When: è analogo all'if ma, a differenza di questo, inizia direttamente la valutazione
- Format: fornisce un output formattato (~D usato per i decimali, ~S per le stringhe, ~% per andare a capo). La t presente come suo primo argomento indica il luogo in cui andrà stampato l'output (generalmente indica lo standard output, che equivale a System.out di java). Per comodità si passa t
- Coerce: trasforma una varibile in un altra

variabile (ad esempio una lista in un vettore). Esempio: (coerce '(1 2 3) 'vector) -> #(1 2 3). Posso anche trasformare una stringa in una lista

- Stringp: prende come parametro un oggetto e restituisce true se l'oggetto è una stringa, false altrimenti

Funzioni sulle liste

- First/car/1: permette di ricavare il primo elemento della lista
- Rest/cdr: permette di ricavare tutta la lista escluso/tolto il primo elemento
- Last: permette di formare una lista

composta dall'ultimo elemento di una lista

- Append: concatena le due liste passate come parametri
- Butlast: permette di formare una lista contenente tutti gli elementi tranne l'ultimo
- Nthcdr: applica la funziona cdr n volte
- Nth: restituisce l'n-esimo elemento di una lista. Prende come parametri la posizione e la lista da cui si vuole estrarre (parte da zero, non da uno)
- Member: Dato un parametro item, se è contenuto nella lista, restituisce la sottolista che parte da item
- Cons/2: crea una nuova lista tramite puntatori (crea liste non piatte, quindi

liste annidate). Mantiene la struttura in profondità

- Listp: ritorna true se l'argomento è una lista
- Mapcar: utilizza una lambda expression per dare a tutti gli elementi di una lista, la stessa caratteristica
- Length: restituisce la lunghezza della lista
- List: trasforma un oggetto in una lista
- Concatenate: ritorna una sequenza che contiene tutti gli elementi individuali della lista nell'ordine in cui vengono forniti

Esistono inoltre i parametri opzionali che sono indicati con &optional e possono non essere inseriti. A questi parametri possono essere assegnati dei valori di default, che si indicano con : qualcosa (e prendono il nome di keywords)
Esempio :

```
(defun visita-albero (tree &optional (mode :indorder))
```

Viene utilizzato anche nel case, che mi

permette di effettuare una scelta, in questo modo:

```
(defun visita-albero (tree &optional mode)
        (case mode
                (:inoorder (node-inorder tree))
                (:preorder (node-preorder tree))
                (:postorder (node-postorder tree))))
```

Array

Gli array, come in java, possiedono delle funzioni che permettono di semplificare il lavoro.
Alcune di queste funzioni sono:

- Array-dimension/2: restituisce la lunghezza dell'array. Inoltre gli specifico anche la dimensione dell'array (2 se array-bidimensionale, 3 se tridimensionale...).
- La variante Array-dimensions/1 restituisce

tutte le dimensioni, non solo quella specificata

- Aref: sta per array-reference e serve per "incrementare/seguire" i contatori
- Assert: verifica che il risultato del suo argomento sia true. Se così è, avanza l'esecuzione, altrimenti la interrompe
- Make-array: permette di creare una nuova matrice / array, definendo le dimensioni
- Setf: imposta il valore desiderato nella cella in cui mi sposto con aref (in generale inizializza i valori)

Esempio di inizializzazione di una matrice e di un vettore

```
(defparameter matrice-iniziale
      (make-array '(2 2)
                  :initial-contents '((1 0) (0 1))))
(defparameter vettore-iniziale
      (make-array '(5)
                  :initial-contents '((1 2 3 4 5))))
```

Lettura e scrittura su file

Per leggere e scrivere su file in lisp viene usata la macro with-open-file e ha la seguente sintassi:
(with-open-file (<var>
<file> :direction :output) <codice>)
dove <var> rappresenta lo stream aperto su <file> e che può essere usato su <codice>