## MACRO

Le funzioni lisp prima di eseguire il body valutano sempre i parametri
Le macro usano gli argomenti per produrre delle forme intermedie, quindi valutano la forma intermedia prodotta

Calcolare una funzione solo se un parametro è sotto una soglia

```
(defun valuta-se (val soglia forma)
  (if (< val soglia)
    forma))
```

Non va perchè forma viene sempre valutata

```
? (valuta-se 10 1 (princ "siamo sotto la soglia"))
siamo sotto la soglia
NIL
```

## Esempio

Si potrebbe risolvere il problema così, intendendo che la forma si passa sempre quotata

```
(defun valuta-se1 (val soglia forma)
  (if (< val soglia)
    (eval forma)))

? (valuta-se1 10 1 '(princ "siamo sotto la soglia"))
NIL
? (valuta-se1 10 11 '(princ "siamo sotto la soglia"))
siamo sotto la soglia
"siamo sotto la soglia"
```

Esiste però un modo migliore di risolvere questo tipo di problemi

## MACRO

```
(defmacro valuta-se2 (val soglia forma)
  (list 'if
        (list '< val soglia)
        forma))
```

`val`, `soglia` e `forma` non vengono valutati nell'espansione della macro

```
(defmacro valuta-se3 (val soglia forma)
  `(if (< ,val ,soglia) ,forma))
```

Il backquote ` impedisce la valutazione ma se c'è una espressione preceduta da una virgola questa viene valutata

```
(defmacro my-pop (lst)
  `(prog1 (car ,lst) (setq ,lst (cdr ,lst))))
```

Se avessimo usato una `defun` non avrebbe funzionato

## Strutture

```
(DEFSTRUCT {name |
            (name{(option-keyword {argument}*)}*)}*
   {slot-name | {(slot-name [default-init])}}*)
```

Definisce `name` come un nuovo tipo di dato e ritorna `name`

Definisce un nuovo predicato `[name]-P`, che ritorna T sugli oggetti di questo tipo

Definisce una funzione `make-name` che crea istanze di questo tipo di dato con gli slot inizializzati al valore `default-init`

Definisce una funzione `copy-name` che prende una istanza di questa struttura e ne ritorna una copia

Per ogni slot costruisce una funzione accessore `[name]-[slot-name]`.
I valori di slot sono modificabili tramite questa funzione e `SETF`

## Esempio

```
? (defstruct compact
    compositore
    opera
    esecutore
    (data-incisione 1995))
COMPACT
? (setf cd1 (make-compact :compositore "L.V. Beethoven"
                          :opera "Concerto per violino"
                          :esecutore "I. Oistrakh"
                          :data-incisione "31/3/69"))
#S(COMPACT :COMPOSITORE "L.V. Beethoven" :OPERA "Concerto
per violino" :ESECUTORE "I. Oistrakh" :DATA-INCISIONE
"31/3/69")
? (compact-esecutore cd1)
"I. Oistrakh"
? (setf (compact-esecutore cd1) "I. Perlman")
"I. Perlman"
?  (compact-esecutore cd1)
"I. Perlman"
```

## Esempio

```
? (compact-p cd1)
T
? (setf cd2 (copy-compact cd1))
#S(COMPACT :COMPOSITORE "L.V. Beethoven" :OPERA "Concerto
per violino" :ESECUTORE "I. Perlman" :DATA-INCISIONE
"31/3/69")
? (setf (compact-data-incisione cd2) "4/5/77")
"4/5/77"
? cd1
#S(COMPACT :COMPOSITORE "L.V. Beethoven" :OPERA "Concerto
per violino" :ESECUTORE "I. Perlman" :DATA-INCISIONE
"31/3/69")
? cd2
#S(COMPACT :COMPOSITORE "L.V. Beethoven" :OPERA "Concerto
per violino" :ESECUTORE "I. Perlman" :DATA-INCISIONE
"4/5/77")
```

## Esempio

```
? (defstruct (compact-plus (:include compact (data-incisione
nil))) (etichetta "EMI"))
COMPACT-PLUS
? (setf cd3 (make-compact-plus :compositore "F. Chopin"
:esecutore "A. Benedetti Michelangeli" :opera "Mazurche"))
#S(COMPACT-PLUS :COMPOSITORE "F. Chopin" :OPERA "Mazurche"
:ESECUTORE "A. Benedetti Michelangeli" :DATA-INCISIONE NIL
:ETICHETTA "EMI")
? (compact-p cd3)
T
?(describe cd3)
Type: COMPACT-PLUS
Class: #<STRUCTURE-CLASS COMPACT-PLUS>
COMPOSITORE: "F. Chopin"
OPERA: "Mazurche"
ESECUTORE: "A. Benedetti Michelangeli"
DATA-INCISIONE: NIL
ETICHETTA: "EMI"
```