1) (defun cic (elem lista pos) (cond ((equal elem (nth (1- pos) lista)) T) (T NIL))

2) Nota: A partir de este programa entendí que identar era esencial para no tener problemas con los paréntesis

\*\*\*\*\*\*\*\*\*Version1 - Al principio tenía la idea de que esto era lo que se solicitaba\*\*\*\*\*\*\*\*\*\*

(defun inicioen (elem lista)

(if (member elem lista)

(let ((resultado (list elem)) (aux lista) )

(do

((n 0 (1+ n ))) ((= (position elem lista) n) (append (reverse resultado) (rest aux)))

(setq resultado (cons (first aux) resultado))

(setq aux (rest aux))

)

)

"El elemento no existe en la lista proporcionada"

)

)

\*\*\*\*\*\*\*\*\*\*Version2 – Esta versión hace lo que realmente se solicitaba\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun inicioen (elem lista)

(if (member elem lista)

(let ((resultado ()) (aux lista) )

(do

((n 0 (1+ n ))) ((= (position elem lista) n) (append aux (reverse resultado)))

(setq resultado (cons (first aux) resultado))

(setq aux (rest aux))

)

)

"El elemento no existe en la lista proporcionada"

)

)

3) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun terminaen (elem lista)

(if (member elem lista)

(let ((resultado ()) (aux (reverse lista)) )

(do

((n 0 (1+ n ))) ((= (position elem (reverse lista)) n) (reverse (append aux (reverse resultado))))

(setq resultado (cons (first aux) resultado))

(setq aux (rest aux))

)

)

"El elemento no existe en la lista proporcionada"

)

)

4)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun primerimpar (lista)

(let ((resultado ()) (aux lista) )

(do

((n 0 (1+ n ))) ((<= (length lista) n) resultado)

(if (oddp (first aux))

(progn (setq resultado (append (list (first aux)) (list n))) (setq n (1+ (length lista))))

(setq aux (rest aux))

)

)

)

)

5)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun lol (lista)

(let ((resultado ()) (aux (reverse lista)) (veces 1) )

(do

((n 0 (1+ n ))) ((< (length lista) n) resultado)

(progn

(if (and (not (equal resultado ())) (equal (first resultado) (first aux)))

(setq veces (1+ veces))

)

(if (and (equal resultado ()) (realp (first aux)) (>= (first aux) 0))

(progn

(setq resultado (list (first aux)))

(print (first aux))

)

)

(if (= n (1- (length lista )))

(progn (setq resultado (append resultado (list veces))) (print veces) )

)

(setq aux (rest aux))

)

)

)

)

6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Opcion1---- No se especifico si al ser una lista propia debía contar el nil al final de la lista, la opción 1 lo cuenta

(defun conteo (lista)

(let ((aux lista) (nentero 0) (nlis 0))

(do

((n 0 (1+ n ))) ((< (length lista) n) (append (list nentero) (list nlis)))

(progn

(if (integerp (first aux))

(setq nentero (1+ nentero))

)

(if (listp (first aux))

(setq nlis (1+ nlis))

)

(setq aux (rest aux))

)

)

)

)

Opcion2---- No cuenta el nil al final de la lista

(defun conteo (lista)

(let ((aux lista) (nentero 0) (nlis 0))

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) (append (list nentero) (list nlis)))

(progn

(if (integerp (first aux))

(setq nentero (1+ nentero))

)

(if (listp (first aux))

(setq nlis (1+ nlis))

)

(setq aux (rest aux))

)

)

)

)

7\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Nota: No halle forma iterativa de hacerlo

(defun aplana (lista)

(let ((aux lista) (resultado ()))

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(setq resultado (append resultado (aplana (first aux))))

(setq resultado (append resultado (list (first aux))))

)

(setq aux (rest aux))

)

)

)

)

8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun diagonal (lista)

(let ((aux lista) (resultado ()) )

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(setq resultado (append resultado (list (nth n (first aux)))))

(setq aux (rest aux))

)

)

)

)

9\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun verif (lista)

(let ((aux lista) (resultado ()) )

(do

((n 0 (1+ n ))) ((< (length lista) n) resultado)

(progn

(cond

((and (listp (first aux)) (> (length (first aux)) 0)) (setq resultado (append resultado (list 'L))))

((and (atom (first aux)) (not (equal (first aux) ()))) (setq resultado (append resultado (list 'A))))

(T (setq resultado (append resultado (list 'N))))

)

(setq aux (rest aux))

)

)

)

)

10\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun s-num (lista)

(let ((aux lista) (resultado 0) )

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(if (numberp (first aux))

(setq resultado (+ resultado (first aux)))

)

(setq aux (rest aux))

)

)

)

)

11\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun fvocales (lista)

(let ((aux (reverse lista)) (resultado ()))

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(if (characterp (first aux))

(if (or (char-equal (first aux) #\a) (char-equal (first aux) #\e) (char-equal (first aux) #\i) (char-equal (first aux) #\o) (char-equal (first aux) #\u))

()

(push (first aux) resultado)

)

(if (and (listp (first aux)) (> (length (first aux)) 0))

(push (fvocales (first aux)) resultado)

(push (first aux) resultado)

)

)

(setq aux (rest aux))

)

)

)

)

12\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun fmult (lista num)

(let ((aux lista) (resultado ()) )

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(cond

((/= (mod (first aux) num) 0) (setq resultado (append resultado (list (first aux)))))

)

(setq aux (rest aux))

)

)

)

)

13\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun celdas (lista)

(let ((aux lista) (resultado 0))

(do

((n 0 (1+ n ))) ((< (1- (length lista)) n) resultado)

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(setq resultado (+ resultado (celdas (first aux))))

(setq resultado (1+ resultado))

)

(setq aux (rest aux))

)

)

)

)

14\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun matrix (lista1 lista2)

(let ((aux1 lista1) (aux2 lista2) (resultado ()))

(if (= (length lista1) (length lista2))

(do ((n 0 (1+ n ))) ((< (1- (length lista1)) n) resultado)

(if (= (length (nth n lista1)) (length (nth n lista2)))

(do ((n 0 (1+ n ))) ((< (1- (length (first aux1))) n) resultado)

(setq resultado (append resultado (list (mapcar #'\* (first aux1) (first aux2)))))

(setq aux1 (rest aux1))

(setq aux2 (rest aux2))

)

(progn

(setq resultado ())

(setq n (1+ (length (nth n lista2))))

)

)

)

)

)

)