Escriure les funcions en LISP que permetin:

Aplanar una llista (treure tots els parèntesi de la llista):

> (aplanar '(a (b c (d e) f) g))
(a b c d e f g)

```
(defun aplanar (l)
      (cond  ((null l) nil)
             ((listp (car l)) (append (aplanar (car l)) (aplanar
                                      (cdr l))))
             (t (cons (car l) (aplanar (cdr l))))))
```

?-aplanar([a,[b,c,[d,e],f],g], L).

```
aplanar([],[]).
aplanar([X|L1],L2):-is_list(X), aplanar(X,L3),
                     aplanar(L1,L4), append(L3,L4,L2).
aplanar([X|L1],[X|L2]):-aplanar(L1,L2).
```

Agafar l'enèsim element d'una llista

> (agafar-n 3 '(a b c d e f))
c

```
(defun agafar-n (n l)
      (cond  ((null l) nil)
             ((= n 1) (car l))
             (t (agafar-n (- n 1) (cdr l)))))
```

?-agafar(3,[a,b,c,d,e,f],E).
E=c

```
agafar(1,[X|_],X):-!.
agafar(N,[X|L],Y):-N1 is N-1,agafar(N1,L,Y).
```

Rotar els elements d'una llista cap a la dreta

> (rotardreta '(a b c d e f))
(f a b c d e)

```
(defun rotardreta (l)
      (cons (darrer l) (rdc l)))

(defun rdc (l)
      (cond  ((null (cdr l)) nil)
             (t (cons (car l) (rdc (cdr l))))))

(defun darrer (l)
      (cond  ((null (cdr l)) (car l))
             (t (darrer (cdr l)))))
```

?-rotardreta([a,b,c,d,e,f], L).
L=[f,a,b,c,d,e]

```
rotardreta(L1,[Y|L2]):-darrer(L1,Y),rdc(L1,L2).

darrer([X],X).
darrer([X|L],Y):-darrer(L,Y).

rdc([X],[]).
```

```
rdc([X|L1],[X|L2]):-rdc(L1,L2).
```

Rotar els elements d'una llista cap a l'esquerra

> (rotaresquerra ‘(a b c d e f))
(b c d e f a)

```
(defun rotaresquerra (l)
      (snoc (car l) (cdr l)))

(defun snoc (x l)
      (cond  ((null l) (list x))
             (t (cons (car l) (snoc x (cdr l))))))
```

?- rotaresquerra([a,b,c,d,e,f], L).
L=[b,c,d,e,f,a]

```
rotaresquerra([X|L1],L2):-afegir(L1,[X],L2).
```

Sumar tots els elements de les posicions parelles d'una llista

> (sumarparells '(1 2 3 4 5 6 7 8 9))
20

```
(defun sumarparells (l)
      (cond  ((null l) 0)
             ((null (cdr l)) 0)
             (t (+ (cadr l) (sumarparells (cdr (cdr l)))))))
```

?- sumarparells([1,2,3,4,5,6,7,8,9], S).
S=20

```
sumarparells([],0).
sumarparells([X],0).
sumarparells([_,X|L],S):-sumarparells(L,S1),S is S1+X.
```

Sumar tots els elements de les posicions senars d'una llista

> (sumarsenars '(1 2 3 4 5 6 7 8 9))
25

```
(defun sumarsenars (l)
      (cond  ((null l) 0)
             ((null (cdr l)) (car l))
             (t (+ (car l) (sumarsenars (cdr (cdr l)))))))
```

?- sumarsenars ([1,2,3,4,5,6,7,8,9], S).
S=25

```
sumarsenars([],0).
sumarsenars([X],X).
sumarsenars([X,_|L],S):- sumarsenars(L,S1),S is S1+X.
```

Mirar a quina posició d'una llista està un element

>(posicio 'a '(t 2 b c a f g))
5

```
(defun posicio (x l)
      (cond  ((null l) 'no-hi-es)
             ((equal x (car l)) 1)
```

```
                    (t (+ 1 (posicio x (cdr l))))))
```

?-posicio(a, [t,2,b,c,a,f,g],P).
P=5

```
posicio(X,[X|_],1):-!.
posicio(X,[_|L],N):-posicio(X,L,N1), N is N1+1.
```

Donades dues llistes, escriure els elements de la segona indexats per la primera

>(indexa '(1 3 7) '(a b c d e f g h))
(a c g) ; el primer, tercer i setè elements

```
(defun indexa (l1 l2)
      (cond  ((null l1) nil)
             (t (cons (agafar-n (car l1) l2)
                      (indexa (cdr l1) l2)))))
```

?-indexa([1,3,7], [a,b,c,d,e,f,g,h], L).
L=[a,c,g]

```
indexa([],_,[]):-!.
indexa([X|L1],L2,[Y|L3]):-agafar(X,L2,Y),indexa(L1,L2,L3).
```

Eliminar un element d'una llista i de totes les seves subllistes

>(borrarl 'a '(b a c (d a (a)) (f a) g))
(b c (d nil) (f) g)

```
(defun borrarl (x l)
      (cond  ((null l) nil)
             ((listp (car l)) (cons (borrarl x (car l))
                                    (borrarl x (cdr l))))
             ((equal x (car l)) (borrarl x (cdr l)))
             (t (cons (car l) (borrarl x (cdr l))))))
```

?-borrarl(a, [b,a,c,[d,a,[a]],[f,a],g], L).
L=[b,c,[d,[],[f],g]

```
borrarl(_,[],[]).
borrarl(X,[X|L1],L2):-!,borrarl(X,L1,L2).
borrarl(X,[Y|L1],[Z|L2]):-is_list(Y),!,borrarl(X,Y,Z),
                           borrarl(X,L1,L2).
borrarl(X,[Y|L1],[Y|L2]):- borrarl(X,L1,L2).
```

Invertir una llista i totes les subllistes

>(invertirtot '(a (b c) (d e (f g)) h))
(h ((g f) e d) (c b) a)

```
(defun invertirtot (l)
      (cond  ((null l) nil)
             ((listp (car l)) (snoc (invertirtot (car l))
                                    (invertirtot (cdr l))))
                 (t (snoc (car l) (invertirtot (cdr l))))))
```

?-invertirtot([a,[b,c],[d,e,[f,g]],h], L).
L=[h,[[g,f],e,d],[c,b],a]

```
invertirtot([],[]).
invertirtot([Y|L1],L2):-is_list(Y),!,invertirtot(Y,Z),
                        invertirtot(L1,L3),append(L3,[Z],L2.
invertirtot([Y|L1],L2):-invertirtot(L1,L3),append(L3,[Y],L2).
```

Comprimir els elements d'una llista

>(comprimir '(a a a b b b c d d d d))
(3 a 3 b 1 c 4 d)

```
((defun comprimir (l)
   (cond     ((null l) nil)
             (t (cons (vegades (car l) l)
                (cons (car l) (comprimir (botar (car l)
                                         (cdr l))))))))

(defun botar (x l)
      (cond  ((null l) nil)
             ((equal x (car l)) (botar x (cdr l)))
             (t l)))
```

?-comprimir([a,a,a,b,b,b,c,d,d,d,d], L).
L=[3,a,3,b,1,c,4,d]

```
comprimir([],[]).
comprimir([X|L1],[N,X|L2]):-
seguides(X,[X|L1],N,L3),comprimir(L3,L2).

seguides(_,[],0,[]).
seguides(X,[Y|L],0,[Y|L]):-X\=Y.
seguides(X,[X|L1],N,L2):-seguides(X,L1,N1,L2), N is N1+1.
```

Fer una llista amb n vegades un element e

>(replicar 4 'a)
(a a a a)

```
(defun replicar (n x)
      (cond  ((= n 1) (list x))
       (t (cons x (replicar (- n 1) x)))))
```

?-replicar(4, a, L).
L=[a,a,a,a]

```
replicar(1,X,[X]):-!.
replicar(N,X,[X|L]):-N1 is N-1, replicar(N1,X,L).
```

Descomprimir els elements d'una llista

>(descomprimir '(3 a 3 b 1 c 4 d))
(a a a b b b c d d d d))

```
(defun descomprimir (l)
      (cond  ((null l) nil)
             (t (append (replicar (car l) (cadr l))
                        (descomprimir (cdr (cdr l)))))))
```

?- descomprimir([3,a,3,b,1,c,4,d], L).
L=[a,a,a,b,b,b,c,d,d,d,d]

```
descomprimir([],[]).
descomprimir([N,X|L1],L4):-
      replicar(N,X,L2),descomprimir(L1,L3),append(L2,L3,L4).
```

Convertir un número binari a decimal

>(decimal '(1 1 0 1))
13

```
(defun decimal (l)
     (aDecimal (reverse l)))

(defun aDecimal (l)
     (cond   ((null l) 0)
             (t (+ (car l) (* 2 (aDecimal (cdr l)))))))
```

?- decimal([1,1,0,1], N).
N=13

```
decimal(L,N):-reverse(L,L1),aDecimal(L1,N).

aDecimal([],0).
aDecimal([X|L],N):-aDecimal(L,N1),N is N1*2+X.
```

Convertir un número en base 10 a base 2:

>(binari 18)
(1 0 0 1 0)

```
(defun binari (n) (reverse (aBinari n))

(defun aBinari (n)
      (cond ((< n 2) (list n))
            (t (cons (mod n 2) (aBinari (/ n 2))))))
```

?-binari(18, B).
B=[1,0,0,1,0]

```
binari(N,B):-aBinari(N,N1), reverse(N1,B).
binari(N,[N]):-N<2.
binari(N,[B|L]):-B is N % 2, N1 is N/2, binari(L, N1).
```