1. Fer un predicat llista que digui si un element és una llista o no

```
?-lista([a]).
yes
?-llista(3).
no

/* una llista pot ser o una llista buida o una llista que té un cap i una coa
  per tant, tendrem dues regles: */

llista([]).
llista([_|_]).
```

2. Donada una llista, troba el seu n-èssim element

```
?-element(3,[a,e,i,o,u],E).
E= i

/* Cas base de la recursivitat: sabem que el primer element d'una llista que comença
  per X, és X
  cas recursiu: cercan l'element N-1 del resta de la llista. Pensau que no podeu
  modificar
  directament el valor d'una variable, sino que heu d'utilitzar una variable nova */

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

3. Invertir una llista donada en tots els seus nivells

```
?-invertir([a,e,[b,c,d],i,o,u],L).
L=[u,o,i,[d,c,b],e,a]

/* Per invertir una llista ho podeu fer posant el primer element al final del resta
  de la llista invertida o posant el darrer element al principi de la llista menys el darrer
  invertida pero pensau que si l'element és una llista l'heu d'invertir recursivament */

/* Posant el primer element darrera: */
invertir([],[]).
invertir([X|L1],L3):-not(llista(X)),invertir(L1,L2), append(L2,[X],L3).
invertir([X|L1],L3):-llista(X),invertir(X,L4),invertir(L1,L2), append(L2,[L4],L3).

/* Posant el darrer element davant (fa falta definir darrer i rdc): */
invertir1([],[]).
invertir1(L1,[X|L3]):-darrer(L1,X),not(llista(X)),rdc(L1,L2),invertir1(L2,L3).
invertir1(L1,[L4|L3]):-darrer(L1,X),llista(X),invertir1(X,L4),rdc(L1,L2),invertir1(L2,L3).

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

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

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


/* Es pot utilitzar també el cut (!) */
invertir2([],[]).
invertir2([X|L1],L3):-not(llista(X)),!,invertir2(L1,L2), append(L2,[X],L3).
invertir2([X|L1],L3):-invertir2(X,L4),invertir2(L1,L2), append(L2,[L4],L3).

invertir3([],[]).
invertir3(L1,[X|L3]):-darrer(L1,X),not(llista(X)),!,rdc(L1,L2),invertir3(L2,L3).
invertir3(L1,[L4|L3]):-darrer(L1,X),invertir3(X,L4),rdc(L1,L2),invertir3(L2,L3).
```

4. Retorna els n primers elements d'una llista

```
?-retornaNprimers(3,[a,b,c,d,e],L).
L=[a,b,c]

/* Sabem retornar els 0 primers elements de qualsevol llista, és la llista buida.
  Si no, llavors reduïm el número d'elements a retornar i escurçam la llista
  recursivament conservant el primer element en el resultat */

retornaNprimers(0,_,[]).
retornaNprimers(N,[X|L1],[X|L2]):-N1 is N-1, retornaNprimers(N1,L1,L2).
```

5. Retorna tots els elements d'una llista menys els n primers

```
?-extreuNprimers (3,[a,b,c,d,e],L).
L=[d,e]

/* Extreure els 0 primers elements d'una llista és la mateixa llista.
   Si no, llavors reduïm el número d'elements a retornar i escurçam la llista
   recursivament  sense guardar el primer element en el resultat */

extreuNprimers(0,L,L).
extreuNprimers(N,[_|L1],L2):-N1 is N-1, extreuNprimers(N1,L1,L2).
```

6. Retorna els darrers n elements d'una llista

```
?-retornaNdarrers (3,[a,b,c,d,e],L).
L=[c,d,e]

/* Si sabem extreure els N primers, llavors retornar els N darrers és el mateix que
  extreure els (longitud de la lista - N) primers */

retornaNdarrers(0,_,[]).
```

```
retornaNdarrers(N,L,L1):-length(L,A),N1 is A-N, extreuNprimers(N1,L,L1).
```

7. Elimina els elements repetits d'una llista

```
?-eliminarepetits([a,b,a,c,b,d,d,e],L).
L=[a,b,c,d,e]

/* D'una llista buida sabem eliminar els repetits. Si la llista no esta buida llavors
   miram si el primer pertany al resta de la llista. Si hi pertany no el guardam, sino
   el guardam i continuam recursivament.
   Pensau que aquesta solució elimina els repetits per davant:
        eliminarrepetits([a,b,c,a,d], L).   ->  L=[b,c,a,d]
*/

eliminarepetits([],[]).
eliminarepetits([X|L1],L2):-pertany(X,L1),!, eliminarepetits(L1,L2).
eliminarepetits([X|L1],[X|L2]):-eliminarepetits(L1,L2).

/* Si es volen eliminar pel darrera: */

eliminarepetits2([],[]).
eliminarepetits2([X|L1],[X|L3]):-pertany(X,L1),!,borra(X,L1,L2),  eliminarepetits2(L2,L3).
eliminarepetits2([X|L1],[X|L2]):-eliminarepetits2(L1,L2).
```

8. Sumar tots els números d'una llista

```
?-sumar([1,3,5,7], N);
N=16


/* Sumar els números d'una llista buida és 0.
   Si la llista no és buida, llavors sumam el primer a la suma de la resta d'elements */

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

9. Sumar els números de les posicions parelles d'una llista

```
?-sumaparells([1,4,3,2,5,7],N).
N=13

/* La suma dels parells d'una llista buida o d'una llista amb un únic element, és zero
   i sumar els parells d'una llista de més de dos elements és sumar el segon (posició
   parell)  amb la suma dels parells de la resta de la llista : */

sumaparells([],0).
sumaparells([_],0).
sumaparells([_,Y|L],Z):-sumaparells(L,Z1), Z is Z1+Y.
```

10. Sumar els números de les posicions senars d'una llista

```
?-sumasenars([1,4,3,2,5,7],N).
N=9

/* La suma dels senars d'una llista buida és zero.
  La suma dels senars d'una llista d'un únic element és aquest element
  i sumar els senars d'una llista de més de dos elements és sumar el primer (posició
  senar) amb la suma dels senars de la resta de la llista : */

sumasenars([],0).
sumasenars([X],X).
sumasenars([X,_|L],Z):-sumasenars(L,Z1), Z is Z1+X.
```

11. Escriure els predicats unió, intersecció,diferència i diferencia simètrica de conjunts (llistes no ordenades de elements no repetits).

```
?-unio([a,b,c,d,e],[t,u,a,s,d],L).
L=[a,b,c,d,e,t,u,s]
?-interseccio([a,b,c,d,e],[t,u,a,s,d],L).
L=[a,d]
?-diferencia([a,b,c,d,e],[t,u,a,s,d],L).
L=[b,c,e]
?-diferenciasimetrica([a,b,c,d,e],[t,u,a,s,d],L).
L=[b,c,e,t,u,s]

unio([],L,L).
unio([X|L1],L2,L3):-pertany(X,L2),!,unio(L1,L2,L3).
unio([X|L1],L2,[X|L3]):-unio(L1,L2,L3).

interseccio([],_,[]).
interseccio([X|L1],L2,[X|L3]):-pertany(X,L2),!,interseccio(L1,L2,L3).
interseccio([_|L1],L2,L3):-interseccio(L1,L2,L3).

diferencia([],_,[]).
diferencia([X|L1],L2,L3):-pertany(X,L2),!,diferencia(L1,L2,L3).
diferencia([X|L1],L2,[X|L3]):-diferencia(L1,L2,L3).

diferenciasimetrica(A,B,C):-unio(A,B,C1), interseccio(A,B,C2),diferencia(C1,C2,C).
```

12. Escriure un predicat que ens digui totes les llistes de quatre dígits (de 1 a 9) que sumin N.

```
?-sumallista(L,10).
L=[1,1,1,7];
L=[1,1,2,6];
...
```

```
sumallista([A,B,C,D],N):-digit(A), digit(B), digit(C), digit(D), N is A+B+C+D.

digit(X):- pertany(X,[1,2,3,4,5,6,7,8,9]).
```

13. Escriure un predicat vegades que ens diu si un element apareix exactament N vegades dins la llista L.

```
?-vegades(a,[a,b,c,d,a],N).
N=2
?-vegades(X,[a,b,c,d,a],2).
X=a

vegades(_,[],0).
vegades(X,[X|R],N) :- vegades(X,R,M), N is M+1.
vegades(X,[Y|R],N) :-  X\==Y, vegades(X,R,N).
```

14. Escriure un predicat distintaposició que ens digui si dues llistes no tenen cap element en la mateixa posició. Poden ser de distinta longitud.

```
?-distintaposicio([a,b,c],[c,b,a,e,f,g]).
no
?-distintaposicio([a,b,c,d],[d,c,b,a]).
yes


distintaposicio([],_).
distintaposicio(_,[]).
distintaposicio([X|L1],[Y|L2]):-X\=Y, distintaposicio(L1,L2).
```

15. Escriure un programa que calculi el producte cartesià de dos conjunts. Els conjunts estaran representats com a llistes i el producte com una llista de llistes de dos elements. Per exemple:

```
?-productecartesia([a,b,c],[d,e],L).
L = [[a,d], [a,e], [b,d], [b,e], [c,d], [c,e]]

productecartesia([],_,[]).
productecartesia([X|L1],L2,L4):-
combinaparelles(X,L2,Y),productecartesia(L1,L2,L3),afegir(Y,L3,L4).

combinaparelles(_,[],[]).
combinaparelles(X,[Y|L1],[[X,Y]|L2]):- combinaparelles(X,L1,L2).
```