1) Cree un programa en Prolog que permita calcular el factorial de un número con el predicado "factorial/2" en donde el primer argumento unifique con el número al que se le calculará el factorial, y el segundo con el factorial del mismo.

Se debe verificar que el número pasado como primer parámetro sea mayor a cero, en caso contrario, el predicado debe fallar.

Ej.: factorial(5, X). => X = 120.

```
factorial(0, 1).
factorial(X, Y) :- X > 0, X1 is X - 1, factorial(X1, Y1), Y is X * Y1.
```

2) Escribir un programa en Prolog que aplane una lista y la ordene. El predicado ordenar/2 recibe una lista cuyos elementos pueden ser otras listas, los elementos individuales de las listas deben ser números enteros, si se encontrara un elemento que no sea numérico, el predicado debería fallar.

Para ordenar y aplanar la lista se deben construir predicados que lo hagan, no usar predefinidos si los hubiere.

Ejemplos:

```
ordenar([1, 3, 2], L).
L = [1, 2, 3].

ordenar([1, [3, 2]], L).
L = [1, 2, 3].

ordenar([1, [3, a]], L).
fail.

ordenar([1, [3, 2], [1, 5, 4], 3, [5, 1]], L).
L = [1, 1, 1, 2, 3, 3, 4, 5, 5].
```

```
apl([], []).
apl([X|L], L2) :- X = [_|_], !, apl(X, X1), apl(L, L1), append(X1, L1, L2).
apl([X|L], [X|L1])  :- integer(X), apl(L, L1).

menor([X], X) :- !.
menor([X1, X2 | L], X) :- X1 =< X2 -> menor([X1|L], X); menor([X2|L], X).

eliminar(_, [], []) :- !.
eliminar(X, [X|L], L) :- !.
eliminar(X, [Y|L], [Y|L1]) :- eliminar(X, L, L1).

ordenar([], []):-!.
ordenar(L, [X|L1]):- menor(L, X), eliminar(X, L, L2), ordenar(L2, L1).

proc(L, Lf):- apl(L, L1), ordenar(L1, Lf).
```

3) Desarrollar un programa en Prolog que permita calcular las permutaciones de una lista.

Ej.: permutaciones([a, b, c], P).
P=[a,b,c];
P=[a,c,b];
P=[b,a,c];…

```
ins(X, L, [ X | L ]).
ins(X, [ Y | L1 ], [ Y | L2 ]) :- ins(X, L1, L2).

per([],[]).
per([ X | L ], Lp) :- per(L, L1), ins(X, L1, Lp).
```

4) Crear un programa en Prolog que dado un tablero de ajedrez, permita colocar tantas reinas como se quiera en el mismo, para que luego dada una posición nos informe que reinas la acechan.

El programa debe prever que la posición a analizar no debe estar ocupada por una reina, de ser así el programa debe fallar.

Otro punto a tener en cuenta es que las posiciones de la reina y de la posición a verificar deben ser validadas para no permitir que se salgan del tablero (8x8)

Las reinas que participarán deberán incorporarse como una lista, donde a su vez, cada elemento representando una reina, será una lista de dos elementos (X, Y).

La declaración de las reinas se da mediante el predicado “reinas/1”, por ejemplo:

```
reinas([[3, 4], [5, 4], [5, 3]]).
```

La posición a validar con el predicado “validar/3” por ejemplo:

```
validar(6, 7, X).
```

El tercer parámetro deberá unificar con un alista que contendrá las posiciones de las reinas que acechen la casilla en cuestión, en caso de no haber reinas que la acechen se deberá unificar con una lista vacía.

```
reinas([[3, 4], [5, 4], [5, 3]]).

pertenece(X, [X|_]) :- !.
pertenece(X, [_|L]) :- pertenece(X, L).

reina(X, Y):- reinas(Reinas), pertenece([X, Y], Reinas).

afuera(X, Y) :- X > 8; X < 1; Y > 8; Y < 1.


validar(X, Y, _) :- reina(X, Y), !, write('casilla ocupada'), fail.
validar(X, Y, _) :- afuera(X, Y), !, write('celda fuera del rango'), fail.

validar(X, Y, R) :-
                validarDir(X, Y,  0,  1, Rn), append([], Rn, R1),
                validarDir(X, Y,  1,  1, Rne), append(R1, Rne, R2),
                validarDir(X, Y,  1,  0, Re), append(R2, Re, R3),
                validarDir(X, Y,  1, -1, Rse), append(R3, Rse, R4),
                validarDir(X, Y,  0, -1, Rs), append(R4, Rs, R5),
                validarDir(X, Y, -1, -1, Rso), append(R5, Rso, R6),
                validarDir(X, Y, -1,  0, Ro), append(R6, Ro, R7),
                validarDir(X, Y, -1,  1, Rno), append(R7, Rno, R).



validarDir(X, Y, _, _, []) :- afuera(X, Y), !.

validarDir(X, Y, Dx, Dy, [[X, Y]|R1]) :-
                        reina(X, Y), !,
```

```
                Y1 is Y + Dy,
                X1 is X + Dx,
                validarDir(X1, Y1, Dx, Dy, R1).

validarDir(X, Y, Dx, Dy, R1) :-
                Y1 is Y + Dy,
                X1 is X + Dx,
                validarDir(X1, Y1, Dx, Dy, R1).
```