Ejercicio 4.1. Definir la relación `diferencia(C1,C2,C3)` que se verifica si `C3` es la diferencia de los conjuntos `C1` y `C2`. Por ejemplo,

```
?- diferencia([a,b],[b,c],X)
X = [a] ;
No
```

Hacer una versión con negación y otra con corte.

Ejercicio 4.2. Definir la relación `agregar(X,L,L1)` que se verifica si `L1` es la lista obtenida añadiéndole `X` a `L`, si `X` no pertenece a `L` y es `L` en caso contrario. Por ejemplo,

```
?- agregar(a,[b,c],L).
L = [a,b,c]
?- agregar(b,[b,c],L).
L = [b,c]
```

Ejercicio 4.3. Definir el predicado `separa(L1,L2,L3)` que separa la lista de números `L1` en dos listas: `L2` formada por los números positivos y `L3` formada por los números negativos o cero. Por ejemplo,

```
?- separa([2,0,-3,5,0,2],L2,L3).
L2 = [2, 5, 2]
L3 = [0, -3, 0]
Yes
```

Proponer dos soluciones, una sin corte y otra con corte.

Ejercicio 4.4. Usar `append` para responder a la siguiente cuestión: *Encontrar todas las listas* `L` *tal que al concatenar* `L` *con* `[c,d]` *nos devuelva* `[a,b,c,d]`.

1. Si obtener prefijos de esta forma fuera el único uso de `append` en nuestro programa, ¿podríamos modificar su definición para mejorar la eficiencia?

2. ¿Serviría esa modificación para el uso general de `append`?

Ejercicio 4.5. Definir el predicado `suma_pares(L,N)` que se verifica si `N` es la suma de todos los números pares de la lista `L`. Por ejemplo,

```
?- suma_pares([2,3,4],N).
N = 6 ;
No
```

Hacer una versión con negación y otra con corte.

Ejercicio 4.6. Definir el predicado `exponente_de_dos(N,E)` que se verifica si `E` es el exponente de 2 en la descomposición de `N` como producto de factores primos. Por ejemplo,

```
?- exponente_de_dos(40,E).
E=3
?- exponente_de_dos(49,E).
E=0
```

Hacer una versión con negación y otra con corte.