Ejercicio 1:

Se tiene la siguiente base de conocimiento que forma parte de un sistema realizado en Prolog:

```
%fabrica(pais, reloj)
fabrica(argentina, reloj(pila, análogico, pulsera)).
fabrica(egipto , reloj(pila, digital, pared)).
fabrica(suiza , reloj(cuerda, pulsera)).
fabrica(brasil , reloj(cuerda, pared)).
fabrica(suiza , reloj(pila, análogico)).
fabrica(argentina , reloj(pila, digital)).
fabrica(egipto , reloj(arena)).
```

Para conocer la cantidad de relojes que funcionan bajo un mecanismo en particular alguien planteo lo siguiente:

```
cantidad(X, C):- findall(X, fabrica(_, reloj(X,_)), L) , length(L, C).
cantidad(X, C):- findall(X, fabrica(_, reloj(X,_, _)), L) , length(L, C).
```

a) ¿Funciona correctamente la siguiente consulta? Justificar.

```
?- cantidad(pila, Cantidad).
```

b) ¿Qué sucede al realizar la consulta? Justificar.

```
?- cantidad(arena, Cantidad).
```

c) Analizar la inversibilidad del predicado cantidad/2 indicando para las variables X y C qué sucede si llegan sin ligar.

d) Plantear una solución aprovechando los conceptos vistos en la materia que resuelva los problemas encontrados. Indicar qué conceptos se usaron en el código propuesto para lograrlo.

Ejercicio 2:

Un sitio de juegos tiene el siguiente código (con getters y setters para las variables de instancia) para una rueda de la fortuna que requiere que la persona pague el costo de jugar y lo premia (o no) en base al casillero en el que caiga la rueda.

```
#Rueda (v.i.: premio, costo)
>> girarPara: unaPersona
      |numero|
      unaPersona dinero: unaPersona dinero - costo.
      numero := (1 to: 6) atRandom[1].
      numero = 1 ifTrue: [ ^ unaPersona dinero: unaPersona dinero + 250 ].
      numero = 2 ifTrue: [ ^ unaPersona dinero: unaPersona dinero + costo ].
      numero = 4 ifTrue: [ ^ unaPersona items add: premio ].
      numero = 5 ifTrue: [ ^ unaPersona dinero: unaPersona dinero +  (costo/2)
].
      ^ unaPersona “no gano nada”

#Persona (v.i.: dinero, items)
```

a) Criticar la solución en términos de delegación y repetición de lógica.

b) Se propone la siguiente solución en Haskell para premiar a la persona luego de girar la rueda:

```
costo (c,_) = c
premio (_,p) = p
premiar 1 _ (dinero, items) = (dinero + 250, items)
premiar 2 rueda (dinero, items) = (dinero + costo rueda, items)
premiar 4 rueda (dinero, items) = (dinero, (premio rueda : items))
premiar 5 rueda (dinero, items) = (dinero + costo rueda / 2, items)
```

1. ¿La solución propuesta funciona correctamente? Justificar.

2. Explicar ventajas y desventajas del uso de pattern matching en la función **premiar**.

3. Comparar las dos implementaciones dadas en términos de efecto colateral y transparencia referencial.

c) El éxito de la rueda de la fortuna llevó al sitio a querer tener otras ruedas del mismo estilo pero con distintos premios. Cada rueda debe poder tener distinta cantidad de casilleros, no necesariamente 6 como la que ya existe.

1. Proponer una alternativa de solución para cada implementación existente que contemple esto, de modo que se puedan tener nuevas ruedas con otros premios sin requerir cambios a la lógica del programa.

2. Definir el tipo de la función **premiar** luego de realizar los cambios del punto anterior.

3. Explicar qué conceptos ayudaron a la solución del nuevo requerimiento en cada paradigma.

[1] Retorna un elemento aleatorio de cualquier tipo de colección, en este caso el intervalo de 1 a 6.