1) Para una aplicación que modela el juego de fútbol Gran DT, se necesita almacenar la información de los jugadores. **Ejemplos**:

- Juan Forlín, defensor de Boca Juniors, vale 1.700.000
- Martín Morel, volante de Tigre, vale 4.700.000.

A un defensor se lo considera caro si vale más de 1.5M, a un volante se lo considera caro si vale más de un monto que depende del equipo, para Boca es 9.2M y para Tigre es 5.1M.

1.a) Implementar lo que haga falta para poder preguntar si Morel es caro o no en los paradigmas lógico y objetos, y escribir la pregunta en ambos casos.
1.b) Lo que recién hiciste son dos pequeños modelos de la misma situación, usando dos paradigmas distintos. Describir cómo se representa
- al jugador Morel
- a la característica de los jugadores de tener un precio que es un número
- a la característica de los jugadores de ser o no caros

en cada paradigma, relacionando con conceptos vistos en la materia, y marcando las diferencias que te parezcan más relevantes entre las dos representaciones.
1.c) Describir, a partir de los conceptos de cada paradigma usado, cómo se hace la pregunta de si Morel es caro o no. A partir del ejemplo, generalizar cómo se hace para preguntar algo en objetos y en lógico.
1.d) Ahora nos dicen que Morel aumentó su cotización a 7M. Describir cómo se refleja el cambio en cada uno de los dos modelos. Hay al menos una diferencia importante, cuál es, con qué concepto visto en la materia está relacionado.
1.e) Se agrega este requerimiento: obtener la cantidad de jugadores caros de un equipo. Resuelva el requerimiento en objetos -usando los mensajes select: y size- y en lógico –usando findall y length-. Qué diferencias y qué similitudes encuentra respecto al concepto de orden superior.

2) Para el requerimiento: “Determinar la cantidad de números pares de una lista” tenemos estas dos soluciones:

Pseudocódigo

```
type LISTA = array[1..longitudMaxima] of
integer;

function cantidadPares(listaNumeros: LISTA):
int
begin
  n ← longitud(listaNumeros);
  count ← 0;
  recorrer i de 1 a n
  begin
    if (listaNumeros[i] es par)
    begin
      count ← count + 1;
    end
  end
  return count;
end
```

Haskell

```
cantidadPares lista =
  (size . filter even)
  lista
```

a) Se pide que analice ambas soluciones a partir del concepto de declaratividad. No vale hablar en abstracto, hay que basarse en las soluciones propuestas.

b) ¿Qué conceptos del paradigma funcional le parece más relevante mencionar en la solución de la función cantidadPares? Indique dónde aparecen en el código.

c) La solución en pseudocódigo, ¿respeta la transparencia referencial? Justifique.

3) Tenemos el código que genera una multa en el sistema de infracciones, teniendo en cuenta que el monto de la multa se calcula:

- $ 50 para los automovilistas
- $ 40 * cantidad de infracciones para los taxistas
- El máximo entre $ 150 y $ 5 por cada infracción para los colectiveros

```
registrarInfraccion (#Conductor)
   | multa cantidadDeInfracciones |
  multa := Multa new.
  multa fecha: Date today.
  self codigo = 1  "Soy Automovilista"
        ifTrue: [ multa monto: 50  ].
  self codigo = 2  "Soy Taxista"
        ifTrue: [  cantidadDeInfracciones := self multas size.
                   multa monto: cantidadDeInfracciones * 40  ].
  self codigo = 3  "Soy Colectivero"
        ifTrue: [  cantidadDeInfracciones := self multas size.
                   multa monto: (150 max: cantidadDeInfracciones * 5) ].
   self multas add: multa.
```

En un workspace se hizo la prueba de generar una infracción, de la siguiente manera:

```
rolandoRivas := Conductor new.
rolandoRivas codigo: 2.                    "Rolando Rivas, taxista"
rolandoRivas multas: OrderedCollection new.  "Inicializo colección de multas"
rolandoRivas registrarInfraccion.          "Semáforo en rojo"
```

(los setters y getters están codificados)

¿Qué cosas mejoraría de la solución (tanto código como Workspace)? Relacione cada mejora con conceptos propios del paradigma de objetos, en especial contestar cada pregunta entre paréntesis:

- Delegación (¿a qué objetos delego qué responsabilidad?)
- Polimorfismo (¿quién se beneficia al aplicar este concepto?)
- Herencia (¿en qué me beneficia y dónde?)

4) Tenemos la siguiente función:

```
funcion1 f g xs = sum (map f (filter g xs))
```

Dada la siguiente información de una empresa de transportes:

```
transportes = [ ("taxi", 200.50), ("colectivo", 563.45), ... ]
```

donde la tupla es: (tipo de transporte, total recaudado)

Queremos resolver:

- La recaudación de la flota de taxis

  funcion1 snd ((== "taxi") . fst) transportes

¿Utilizando qué conceptos evitamos crear funciones auxiliares? Justifique y muestre dónde aparece cada uno de ellos.

Nota: Sólo indicar los conceptos que ayudan a no tener que crear una función auxiliar aparte, además de funcion1 que sí la definimos.