


Condiciones de aprobación:

<p>Para aprobar es necesario simultáneamente:</p> <ul style="list-style-type: none"> • obtener un 60% del puntaje total, y • obtener al menos la mitad de los puntos en cada paradigma. <p>Las preguntas choice o V/F:</p> <ul style="list-style-type: none"> • no serán consideradas si no están justificadas, y • se justifican mediante explicaciones y/o código a criterio del alumno 	<p>En todas tus respuestas sé puntual, no pierdas el foco de lo que se pregunta. Respuestas en exceso generales son tan malas como respuestas incompletas.</p> 
---	--

Parte A

Nos interesa modelar una persona en base a la forma de responder un saludo:

- Algunos saludan siempre con el mismo discurso: “¿Bien o te cuento?”
- Otros responden el mismo saludo que recibieron
- Otros toman la primera palabra y le agregan un: “ es una forma de decir”

La primera solución que se nos ocurrió fue:

```
saludar :: String -> Int -> String
```

```
saludar saludoOriginal 1 = "Bien, ¿o querés que te cuente?"
```

```
saludar saludoOriginal 2 = saludoOriginal
```

```
saludar saludoOriginal 3 = ((flip (++) " es una forma de decir") . head . words1)
                           saludoOriginal
```

- Indique qué conceptos aparecen en la solución propuesta y explicar para qué se utiliza cada uno.
- Comente brevemente puntos positivos y negativos de la solución propuesta, justificando su decisión.
- ¿Sería posible usar saludar con un saludo original infinito en base a las posibles respuestas que soporta el programa? Justificar.

Ni bien lo terminamos de resolver, el usuario nos pide una ligera modificación: “¿Le pueden agregar el nombre de la persona al principio del saludo? Así sé quién es, por ejemplo si le decimos “Hola” a Pepe que responde con el mismo saludo, que responda “Pepe: Hola”, y lo mismo para los otros tipos de saludos”. Por lo tanto:

- Realice los cambios que considere necesarios de manera que no se repita lógica y también sea fácil incorporar nuevos estilos de saludos. Justifique los cambios realizados y dé un ejemplo de uso.

Parte B

Dada la siguiente base de conocimientos:

gusta(charly, milanese). gusta(charly, bondiola). gusta(clari, milanese). gusta(clari, pizza). gusta(erwin, pizza). gusta(franco, sushi).	comida(milanese). comida(higado). comida(bondiola). comida(pizza). comida(brocoli). comida(sushi). comida(bacalao).
--	---

Si tenemos una lista de comidas y queremos relacionar el universo de platos que le gusta a alguno, dos desarrolladores plantearon dos soluciones similares:

Solución A)

```
comidasRicasA(ComidasRicas):-
```

```
    findall(Comida, (comida(Comida), gusta(_, Comida)), ComidasRicas).
```

Solución B)

```
comidasRicasB(ComidasRicas):-
```

```
    findall(Comida, comida(Comida), Comidas), sonRicas(Comidas, ComidasRicas).
```

¹ La función words recibe un string y devuelve una lista de strings equivalente al primero separado por sus espacios en blanco. Ejemplo: words “Hola mundo” --> [“Hola”, “mundo”]

```

sonRicas([], []).
sonRicas([Comida|Comidas], [Comida|ComidasRicas]):-gusta(_, Comida),
    sonRicas(Comidas, ComidasRicas).
sonRicas([_|Comidas], ComidasRicas):-sonRicas(Comidas, ComidasRicas).

```

- 1) ¿Son inversibles ambos predicados principales? Justificar.
- 2) ¿Qué diferencia presentan en las consultas existenciales?
- 3) Relacionar dicha diferencia de funcionalidad con un concepto específico del paradigma que la genera.
- 4) Generar un ejemplo dentro del ejercicio que permita contar el principio de universo cerrado. Sea específico, tiene que usar el código existente (del enunciado base y/o soluciones A y B).

Parte C



Fletrix es una empresa pujante que nos ha encargado relevar su negocio, que consiste en ofrecer por streaming series, películas y documentales.

- Todos los productos tienen un precio mensual que Fletrix debe pagar para ofrecerlo a sus clientes
- También sabemos la fecha de estreno de la serie, película o documental
- Sabemos qué cosas mira o miró un cliente
- Un producto es deficitario si sale más de \$ 100.000 y pasó más de un año de la fecha de estreno
- También sabemos cuándo un producto es interesante
 - En una serie, tiene que tener 4 ó 5 temporadas
 - En una película, porque ganó un Oscar
 - En un documental, si en el título dice la palabra “unofficial”

Queremos saber:

- Si un cliente es un ventajero. Lo es cuando solo mira productos deficitarios
- Y si un cliente mira algo interesante

Para esto se planteó la siguiente solución:

 <pre> class Cliente { method esVentajero() = { productos.all({ producto => (producto.costo() > 100000) && (producto.fechaEstreno().plusYears(1) < new Date() }) } method miraAlgoInteresante() = { productos.any({ producto => producto.esInteresante() }) } class Producto { method esInteresante() { if (tipo == "serie") { return this.cantidadDeTemporadas() == 4 this.cantidadDeTemporadas() == 5 } if (tipo == "pelicula") { return this.ganoOscar() } if (tipo == "documental") { return this.tieneTituloNoOficial() } } } </pre>	 <pre> #Cliente >> esVentajero ^ productos allSatisfy:[:unProducto unProducto costo > 100000 and: [(unProducto fechaEstreno addYears: 1) < Date today]] >> miraAlgoInteresante ^ productos any: [:unProducto unProducto esInteresante] #Producto >> esInteresante tipo = 'serie' ifTrue: [^self cantidadDeTemporadas = 4 or: [self cantidadDeTemporadas = 5]]. tipo = 'pelicula' ifTrue: [^self ganoOscar]. tipo = 'documental' ifTrue: [^self tieneTituloNoOficial] </pre>
--	--

- 1) Responder Verdadero o Falso y justificar conceptualmente en todos los casos:
 - a) En la solución propuesta se aprovecha la delegación de responsabilidades.
 - b) El polimorfismo se utiliza para conocer si un producto cualquiera es o no interesante.
 - c) Si se necesita agregar nuevos productos la solución planteada es fácilmente extensible.
 - d) No hay manera de aprovechar el uso de herencia para este dominio, ya que no hay lógica repetida.

2) Proponer una nueva solución que resuelva todos los problemas que encuentre en la solución original. Incluir codificación y diagrama de clases.