

Final 15/02/2025

## Parte A

Se tiene el siguiente código en Prolog para un sistema de compras de supermercados

marca(cindor, laSerenisima). marca(latuna, nereida). marca(serenito, laSerenisima). cliente(Cliente) :- compro(Cliente, _)	compro(martina, latuna). compro(martina, cindor). compro(aye, cindor). compro(aye, serenito)
---	---

1. Se desea conocer qué clientes son obsesivos, lo que ocurre cuando **solo compran productos de una marca**. En el ejemplo, solo aye es obsesiva, porque compra sólo cosas de laSerenisima, mientras que martina compra de laSerenisima y nereida.

Se tienen las siguientes soluciones con problemas. **Para cada caso** justificar conceptualmente por qué tiene problemas:

- obsesivo(Cliente) :- cliente(Cliente),  
forall( compro(Cliente, Producto), marca(Producto, Marca) ).
- obsesivo(Cliente) :- marca( \_, Marca),  
forall( compro(Cliente, Producto), marca(Producto, Marca) ).
- obsesivo(Cliente) :- marca( Producto , \_),  
forall( compro(Cliente, Producto), marca(Producto, Marca) ).

2. Codificar una solución superadora, correcta conceptualmente, pero que en lugar de usar forall/2 use not/1.

## Parte B

Dado el siguiente código Haskell para analizar la información de los productos de un supermercado:

data Producto = Producto { nombre :: String, marca :: String }	deMarca unaMarca productos = filter (== unaMarca) . marca productos todosSeLlaman unNombre = all ((== unNombre) . nombre) primerosQueCumplen criterio = map nombre . take 3 . filter criterio
---	---

- Una de las 3 funciones dadas no compila por un error de tipos. Identificar cuál, explicar qué problemas tiene y proponer una alternativa que sí type.
- Indicar el tipo de las otras dos funciones dadas y mostrar un ejemplo de invocación para cada una.
- Dada una lista infinita de productos, y asumiendo que la función con errores de tipos fue corregida, explicar para cada una de las funciones dadas si existe algún escenario en el cual pueda terminar de evaluarse y si existe alguno en el cual no pueda hacerlo.

## Parte C

Queremos programar un calendario en donde se pueda averiguar si una fecha dada está libre. En el calendario ya hay eventos agendados que pueden ser, por ahora, eventos de día completo o recordatorios. La fecha está libre cuando ninguno de los eventos de día completo ocupa esa fecha, sabiendo que con los recordatorios no hay conflicto.

Se tiene la siguiente solución en Wollok.

```
class Calendario {
    var eventos
    method estaLibre(fecha) =
        eventos.all({evento =>
            evento.esRecordatorio() or (not evento.esRecordatorio() and evento.fecha() != fecha)
        })
}
class EventoDiaCompleto {
    var property fecha
    method esRecordatorio(){ return false }
}

class Recordatorio {
    method esRecordatorio(){ return true }
}
```

1. Dada la solución anterior, responder verdadero o falso y justificar en todos los casos.
  - a. Es necesario agregar una superclase Evento para que los distintos tipos de eventos sean polimórficos.
  - b. Es posible agregar un nuevo tipo de evento: los de varios días (en ellos se detalla una lista con todos los días que ocupa), sin cambiar el código de la clase calendario.
2. Proponer una nueva solución (código y diagrama) que resuelva los problemas existentes en el código dado (sin introducir nuevos problemas), e incorpore los eventos de varios días.

## Paradigmas de Programación

### Examen Final

01/03/2025

#### Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

**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.



## Parte A

Un clan samurai está reclutando guerreros. Hay distintos criterios según los maestros del clan.

- Hay un maestro para quien un guerrero es candidato si es habilidoso con al menos 7 de habilidad
- Hay otro maestro para quien un guerrero es candidato si es honorable, es decir, que no tiene deshonra entre sus características

Se tiene el siguiente código:

```
data Guerrero = UnGuerrero {  
    nombre :: String,  
    habilidad :: Habilidad,  
    caracteristicas :: [String] }  
  
habilidoso :: Habilidad -> Guerrero -> Guerrero  
habilidoso minimo guerrero = habilidad guerrero >= minimo  
  
honorable :: Guerrero -> Bool  
honorable guerrero = not elem "deshonra" (caracteristicas guerrero )
```

Y se cuenta con la siguiente definición, según la variante de lenguaje que se use

```
type Habilidad = Int      -- Haskell normal  
type Habilidad = Number   -- Haskell con pdepreludat
```

- Arreglar los errores que encuentre en el código anterior. (Errores que hacen que el código no funcione, hay al menos uno)
- Completar la codificación de una función que obtenga los nombres de los candidatos a reclutar, dado un maestro y una lista de guerreros.

```
type Maestro =  
reclutas :: Maestro -> [Guerrero] -> [String]  
reclutas maestro guerreros =  
guerreros
```

- Mostrar dos ejemplos de consulta de la función anterior, uno para cada maestro (asumir que ya hay una lista de guerreros de ejemplo)
- Reescribir de una manera alternativa la función honorable utilizando adecuadamente la función . (composición). (y arreglando los errores si los hubiera detectado en el ítem 1)

## Parte B

Un grupo de músicos quiere formar una banda, pero deben asegurarse de que haya al menos un integrante que toque cada instrumento necesario para la banda. Han decidido modelar esto bajo el paradigma lógico para tener mejor visibilidad de las combinaciones posibles y elegir más fácilmente. Se tiene esta base de conocimientos a modo de ejemplo:

```
musico(luis, guitarra).
musico(luis, bajo).
musico(ana, voz). // la voz se considera un instrumento
musico(ana, teclado).
musico(juan, bateria).
musico(maria, voz).
musico(maria, guitarra).
instrumentosRequeridosParaBandaDe(rock, [guitarra, bateria, bajo, voz]).
instrumentosRequeridosParaBandaDe(jazz, [teclado, saxo, contrabajo, bateria]).
```

El predicado `bandaPosible/2`, pretender verificar que dado un conjunto de músicos y un género musical, se puede conformar con una banda de ese género, ya que los músicos cubren todos los instrumentos requeridos. Se considera que un músico puede tocar más de un instrumento, por lo que la banda podría tener menos integrantes que instrumentos requeridos.

```
bandaPosible(Musicos, Genero) :-
    instrumentosRequeridosParaBandaDe(Genero, InstrumentosRequeridos),
    findall(Instrumento,
        instrumentoDeAlgunMusico(Musicos, Instrumento),
        InstrumentosDisponibles),
    cubreTodos(InstrumentosRequeridos, InstrumentosDisponibles).

instrumentoDeAlgunMusico(Musicos, Instrumento) :-
    member(Musico, Musicos), musico(Musico, Instrumento).

cubreTodos(InstrumentosRequeridos, InstrumentosDisponibles) :-
    forall(member(Instrumento, InstrumentosRequeridos),
        member(Instrumento, InstrumentosDisponibles)).
```

Esta solución resuelve el problema planteado correctamente pero alguien propone una nueva solución.

```
bandaPosible(G, B) :-
    instrumentosRequeridosParaBandaDe(B, I),
    forall(instrumentoDeAlgunMusico(G, II), member(II, I)).
```

Aun sumando el predicado auxiliar que reutiliza, evidentemente la nueva solución es mas breve que la anterior...

1. Es correcta?
2. Es más expresiva?
3. Es más declarativa?
4. Mostrar ejemplos de consulta y respuesta que muestren qué tan inversibles son ambas soluciones.

## Parte C

Estamos modelando parte del sistema de una **ferretería**, donde se quiere conocer el precio de los productos adquiridos en una compra. Sabemos que el precio de cada producto se calcula como el precio base del mismo, al cual se le hace un recargo de un porcentaje según el país de origen. Por ejemplo, para Uruguay actualmente sería del 10%, mientras que para Brasil sería del 7%.

- El precio base de una herramienta es 10 veces su peso en gramos.
- Para los **materiales de construcción**, el precio base se indica de manera explícita para cada uno.
- Al momento, no hay productos que no sean herramientas o materiales de construcción.

```
class Compra {
    const productos = []

    method totalAPagar() =
}

class Producto {
    var origen

    method precio() =
}
```

```
class Herramienta inherits Producto {
    method precioBase() = peso * 10
}

class MaterialDeConstruccion inherits Producto {
    var property precioBase
}
```

1. Completar la solución, modelando los orígenes e implementando el método **totalAPagar()**, **precio()** y los demás métodos y definiciones que sean necesarios para poder calcular el total a pagar de una compra de productos de la ferretería.
2. Indicar si es V o F y justificar brevemente
  - Hay polimorfismo en **precio()**, a pesar que haya una sola implementación de dicho método que es la misma para todos los productos.
  - Si se agregaran nuevos cosas que se puedan comprar en la ferretería para las cuales las clases ya definidas no fueran adecuadas, la clase de la que se instanciarían estos nuevos objetos debe también heredar de **Producto**.
3. Se agregan algunas herramientas que son de uso profesional, en las que su precio base es también su peso por 10, pero para el precio final, luego de aplicarle el recargo del país de origen, se le suman siempre \$10.000 de la licencia profesional. Modificar la solución para contemplar este nuevo requerimiento.

Paradigmas de Programación

Examen Final

22/02/2025

Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

**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.



Parte A

Se desea conocer cuáles son los jugadores buenos de un equipo. Puede haber distintos jugadores, y para saber cuándo un jugador es bueno depende de su posición. Se sabe que ser un buen mediocampista es difícil, porque todo buen mediocampista debe antes ser buen defensor. El siguiente código resuelve el problema y funciona correctamente:

```
class Equipo {  
    var jugadores  
    method jugadoresBuenos(){  
        var buenos = []  
        jugadores.forEach({ jug =>  
            if (jug.esArquero() && jug.atajaPenales()) { buenos.add(jug) }  
            if (jug.esDefensor() && jug.marcaMucho()) { buenos.add(jug) }  
            if (jug.esMediocampista() && jug.marcaMucho() && jug.pasaBien()) { buenos.add(jug) }  
        })  
        return buenos  
    }  
}  
class Jugador {  
    var porcentajeBuenosPases  
    var penalesAtajados  
    var cantMarcas  
    var posicion  
    method esArquero() = posicion == "Arquero"  
    method esDefensor() = posicion == "Defensor"  
    method esMediocampista() = posicion == "Mediocampista"  
    method atajaPenales() = penalesAtajados > 2  
    method marcaMucho() = cantMarcas > 2  
    method pasaBien() = porcentajeBuenosPases > 0.75  
}
```

1. Responde Verdadero o Falso justificando en todos los casos.
  - a. El método `jugadoresBuenos()` no produce efecto en el estado interno del `Equipo` luego de ejecutarse.
  - b. La solución tiene problemas de declaratividad.
  - c. Para agregar un nuevo tipo de jugador (el delantero, por ejemplo) sólo debo modificar la clase `Jugador`.
  - d. La responsabilidad de saber si un jugador es bueno está mal asignada.
2. Codificar una solución superadora, sin repetición de lógica, en el lenguaje Wollok que resuelva los problemas mencionados anteriormente (y otros que existan) y además **siga permitiendo que un jugador cambie de posición** en cualquier momento y **agregando al delantero**, que es bueno cuando hizo más de 10 goles.

## Parte B

Se tiene el siguiente código Prolog para un sistema de compras de supermercado.

ingrediente(panqueque, huevo). ingrediente(panqueque, harina). ingrediente(flan, huevo). ingrediente(flan, leche). ingrediente(churrasco, carne).	come(jony, panqueque). come(jony, flan). come(moro, tortilla). come(moro, churrasco). come(ivo, flan).
---	--

1. Se desea conocer qué personas son quisquilloosas, lo que ocurre cuando sólo comen comidas con un mismo ingrediente. En el ejemplo, jony es quisquilloso porque sólo come cosas con huevo, ivo también es quisquilloso porque sólo come cosas con leche, pero moro no es quisquilloso.

Se tienen las siguientes soluciones con problemas. Para cada caso justificar conceptualmente por qué tiene problemas:

- a. quisquillosa(Persona) :- come(Persona, \_),  
forall( come(Persona, Comida), ingrediente(Comida, Ingrediente) ).
- b. quisquillosa(Persona) :- ingrediente(\_, Ingrediente),  
forall( come(Persona, Comida), ingrediente(Comida, Ingrediente) ).
- c. quisquillosa(Persona) :- come(\_, Comida),  
forall( come(Persona, Comida), ingrediente(Comida, Ingrediente) ).

2. Codificar una solución superadora, correcta conceptualmente, pero que en lugar de usar forall/2 use not/1.

## Parte C

Dadas las siguientes definiciones

habilidades :: Persona -> [Habilidad]	sirve :: Problema -> Habilidad -> Bool
---------------------------------------	--

1. Usando las funciones habilidades y sirve, definir una función podrianAyudar que dado un problema y una lista de personas permita saber qué personas tienen alguna habilidad que sirva para el problema recibido.
2. Indicar qué concepto/s de los siguientes se está/n utilizando en la solución anterior y para qué:
  - a. Orden Superior
  - b. Composición
  - c. Aplicación Parcial
  - d. Pattern matching
3. Si habilidades devolviera una lista infinita de habilidades para una persona, ¿cómo se comportaría podrianAyudar si dicha persona estuviera en la lista recibida?

Paradigmas de Programación

Examen Final

21/12/2024

Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

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.



## Parte A

Se desea modelar los tipos de cuentas que pueden contratar los usuarios de una plataforma de streaming. Existen tres tipos de cuentas:

- Cuenta Básica: Puede reproducir hasta un límite de horas de contenido al mes, luego no se permite.
- Cuenta Premium: No tiene límite de reproducción. Además, puede descargar contenido con un límite de GB.
- Cuenta Familiar: Es como una cuenta Premium pero permite múltiples perfiles de usuarios, cada uno con límites distintos.

Además, se desea:

- Obtener un string que muestre el tipo de cuenta, el nombre del usuario principal, y la información específica de cada tipo de cuenta: horas restantes (en una cuenta básica), espacio disponible (en cuentas premium o familiar), y perfiles activos (en cuentas familiares).
- Todas las cuentas deben poder cambiar de tipo en cualquier momento, conservando sus datos relevantes.

Contamos con la siguiente solución:

```
class Cuenta {
    var property tipo
    var property usuarioPrincipal
    var property horasConsumidas
    var property limiteHoras
    var property espacioConsumido
    var property limiteEspacio
    var property perfiles

    method estado() {
        if (tipo == "Básica") {
            return "Tipo: " + tipo + ", Usuario: " +
                usuarioPrincipal + ", Horas restantes: " +
                (limiteHoras - horasConsumidas)
        } else if (tipo == "Premium") {
            return "Tipo: " + tipo + ", Usuario: " +
                usuarioPrincipal + ", Espacio disponible: " +
                (limiteEspacio - espacioConsumido) + "GB"
        } else {
            return "Tipo: " + tipo + ", Usuario: " +
                usuarioPrincipal + ", Perfiles: " +
                perfiles.size()
        }
    }
}

// (sigue de la class Cuenta)
method reproducirContenido(horas) {
    if (tipo == "Básica") {
        if (horasConsumidas + horas > limiteHoras) {
            return "No podés reproducir más contenido este mes."
        }
        horasConsumidas += horas
    } else {
        return "Contenido reproducido."
    }
}

method descargarContenido(gb) {
    if (tipo == "Premium" || tipo == "Familiar") {
        if (espacioConsumido + gb > limiteEspacio) {
            return "No hay suficiente espacio."
        }
        espacioConsumido += gb
    } else {
        return "Tu plan no permite descargas."
    }
}
```

1. Responder Verdadero o Falso y justificar:

- a. Sin modificar los métodos existentes, puede agregarse fácilmente un nuevo tipo de cuenta, por ejemplo, una cuenta "Pro" que permite descargas ilimitadas pero no reproducción sin conexión.

- b. Si se agrega un método transferirContenido(cuenta1, cuenta2), que permita transferir contenido descargado entre cuentas premium o familiares, pero no entre cuentas básicas:

```
method transferirContenido(cuenta1, cuenta2) {
    cuenta1.descargarContenido(-5) // Libera 5GB en cuenta1
    cuenta2.descargarContenido(5) // Consumo 5GB en cuenta2
}
```

Si una de las cuentas no tiene suficiente espacio disponible, no se afecta el estado de la otra.

2. Proponer una solución mejorada: Diseñar una solución que respete las buenas prácticas de orientación a objetos, evite repeticiones de lógica y facilite la extensión a nuevos tipos de cuentas sin modificar código existente.

## Paradigmas de Programación

## Examen Final

21/12/2024

### Parte B

Queremos estudiar el comportamiento de la gente al elegir alimentos. Contamos con:

<pre>data Alimento = Comida (     nombre :: [Char],     calorias :: Float,     nutrientes :: [String] ) ana = [esBajoEnCalorías, tieneProteinas]</pre>	<pre>esBajoEnCalorías :: Alimento -&gt; Bool esBajoEnCalorías comida = calorias comida &lt; 100  tieneProteinas :: Alimento -&gt; Bool tieneProteinas = elem "proteinas" . nutrientes</pre>
--	---

1. Definir el tipo de ana.
  2. Queremos determinar si un alimento tiene un nutriente en particular, no necesariamente proteínas:
    - a. Definir la función tieneNutriente y explicitar su tipo.
    - b. Volver a definir ana usando tieneNutriente en vez de tieneProteinas, indicando qué concepto del paradigma funcional se aprovecha en esta solución que no se usaba en la definición anterior de ana.
  3. Se cuenta con una función que permite establecer, dada una lista de alimentos, cuáles elegiría una persona según sus requisitos (por ejemplo, ana), asumiendo que elige un alimento que cumpla al menos 1.
- ```
alimentosElegidos :: [Alimento -> Bool] -> [Alimento] -> [Alimento]
alimentosElegidos requisitos alimentos = filter (f requisitos) alimentos
f :: [Alimento -> Bool] -> Alimento -> Bool
f requisitos alimento = any (\requisito -> requisito alimento) requisitos
```

Responder verdadero o falso, justificar y corregir la implementación en caso de ser necesario:

- a. La solución funciona correctamente.
- b. La solución es poco declarativa.
- c. La solución es poco expresiva.
- d. Asumiendo que funciona como se explica, si se la invoca con una lista de alimentos infinita, no se podrá obtener ninguna respuesta independientemente de cuáles sean los requisitos de la persona.

### Parte C

Dada la siguiente base de conocimiento:

```
%esAutorDe(Autor, Obra).
esAutorDe(tolkien, elSeniorDeLosAnillos).
esAutorDe(tolkien, elHobbit).
esAutorDe(rowling, harryPotterYLaCamaraSecreta).
esAutorDe(rowling, unaVacanteImprevista).
esAutorDe(miguelCervantes, donQuijote).
esAutorDe(borges, elAleph).
```

```
%genero(Obra, Genero).
genero(elSeniorDeLosAnillos, fantasia).
genero(elHobbit, fantasia).
genero(harryPotterYLaCamaraSecreta, fantasia).
genero(unaVacanteImprevista, novela).
genero(donQuijote, novela).
genero(elAleph, cuento).
```

Queremos identificar si un autor se dedica exclusivamente a un único género literario. Por ejemplo, tolkien escribe únicamente fantasía, mientras que rowling no, ya que ha escrito tanto fantasía como novelas. Una consultora propuso la siguiente solución:

```
autorEspecifico(Autor) :- 
    esAutorDe(Autor, _),
    forall(genero(Obra, _), esAutorDe(Autor, Obra)).
```

1. Explicar por qué esta solución no cumple con el problema planteado. Usar lenguaje coloquial para describir qué hace realmente la solución propuesta.
2. Proponer una alternativa que funcione correctamente para resolver consultas como:
  - o autorEspecifico(tolkien)
  - o autorEspecifico(Quien) (y que sea inversible)
3. Indicar en qué partes de su solución aparecen los conceptos de:
  - o Orden Superior
  - o Inversibilidad

Final 14/12/2024

## Parte A

Se necesita implementar un sistema para que los atletas se inscriban a las competencias de un torneo deportivo. Sabemos que un atleta puede inscribirse en una competencia cuando:

1. Ha completado la inscripción general al torneo.
2. No ha participado todavía en esa competencia.
3. Ha participado en todas las competencias requeridas como condición previa para esa competencia, a menos que la inscripción general al torneo se haya realizado hace menos de un año calendario, en cuyo caso las competencias previas no son requeridas.

Se propuso la siguiente solución:

```
puedeInscribirse(Atleta, Competencia, Fecha):-  
    inscripcionGeneral(Atleta, FechaInscripcion),  
    not(participo(Atleta, Competencia, _)),  
    añosCalendarioTranscurridos(FechaInscripcion, Fecha, 0).  
  
puedeInscribirse(Atleta, Competencia, _):-  
    inscripcionGeneral(Atleta, _),  
    not(participo(Atleta, Competencia, _)),  
    competenciaRequerida(CompetenciaPrevia, Competencia),  
    participo(Atleta, CompetenciaPrevia, _).
```

Se sabe que existen los siguientes predicados:

- `inscripcionGeneral/2`: Relaciona un atleta y la fecha en la que completó su inscripción general al torneo. Es inversible.
- `participo/3`: Relaciona un atleta, una competencia, y la fecha en la que participó. Es inversible.
- `competenciaRequerida/2`: Relaciona dos competencias tales que la primera es condición previa para la segunda. Es inversible.
- `añosCalendarioTranscurridos/3`: Relaciona dos fechas cualesquiera con los años calendario que transcurrieron entre ambas. Solo es inversible para su tercer parámetro.

1. ¿La solución propuesta cumple con la lógica pedida? Justifique y plantee algún ejemplo que sirva para fundamentar esa respuesta.
2. Analice la inversibilidad de `puedeInscribirse/3`. En caso de que no sea inversible para uno o más parámetros, explique qué sería necesario modificar o agregar para que lo sea.
3. Realice cualquier corrección que considere necesaria sobre `puedeInscribirse/3`, considerando las respuestas anteriores y eliminando cualquier repetición de lógica existente.

## Parte B

Se desea modelar en el paradigma funcional un sistema de admisión para competencias deportivas. Existen atletas que desean participar en una competencia que tiene una serie de requisitos, de manera que no se permite la inscripción de aquellos que incumplan alguno de ellos. Puede haber diversos requisitos, como por ejemplo que solo se inscriban atletas de una cierta nacionalidad, que no se permita la inscripción de quienes tengan menos de cierta edad, o que no puedan participar atletas con ciertos tipos de equipamiento no permitido. Puede haber competencias con más requisitos que otras, sin requisitos, o con requisitos similares, como restringir la participación por nacionalidad, pero con diferentes criterios específicos. Por lo tanto, de los atletas se debe conocer la edad, la nacionalidad, y el equipamiento que llevan consigo.

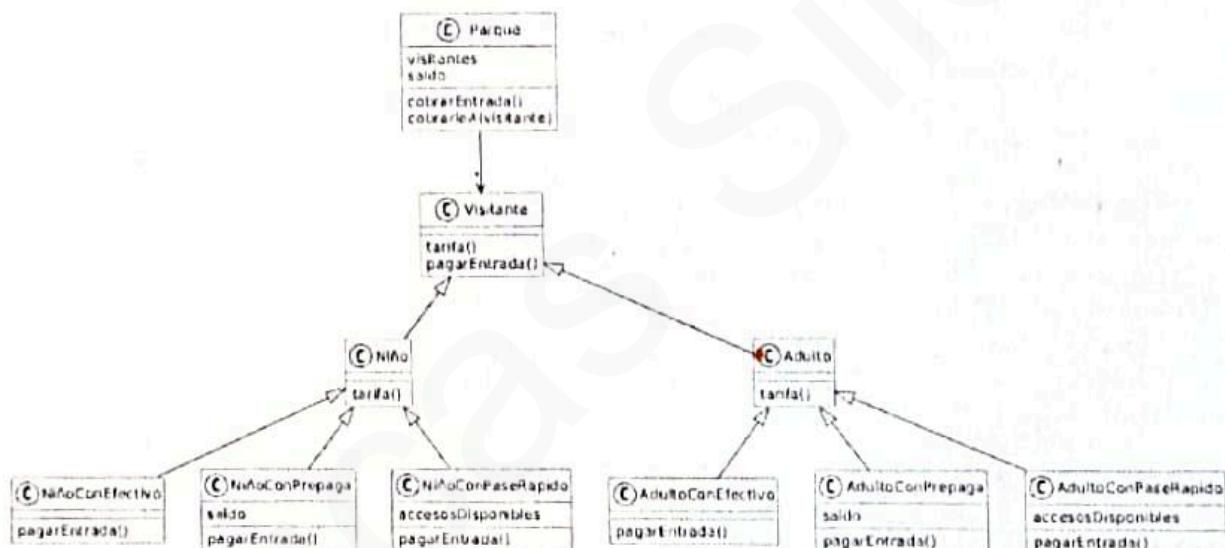
Possible solution: [lucassoliz/practicas-haskell-prolog-wollok](https://github.com/lucassoliz/practicas-haskell-prolog-wollok): Ejercicios y prácticas de Haskell para la facultad

1. Definir tipos de datos y funciones (explicitando el tipo de todas ellas) para cubrir las necesidades explicadas.
2. Mostrar como se representa una competencia de ejemplo que tenga tres requisitos como los mencionados anteriormente
3. Desarrollar la función `controlDeInscripcion`, que permita saber qué atletas de una lista de espera cumplen los requisitos para participar en la competencia.
4. Indicar dónde y para que se utilizaron los siguientes conceptos. **Composición, Aplicación parcial, Orden.**

## Parte C

Se sabe que en un parque de diversiones se cobra entrada a los visitantes. Los adultos y los niños tienen diferentes formas de calcular la tarifa (los adultos pagan una tarifa única de \$100, mientras que los niños pagan \$50), y además hay visitantes que pagan con "Pase Rápido", otros con efectivo y otros con tarjeta prepaga. Al efectuarse el cobro suceden dos cosas: el saldo del parque aumenta, y cada visitante efectúa el pago. Los que pagan con tarjeta prepaga disminuyen el saldo de su tarjeta en un 10% adicional como gasto de servicio (el 10% no va para el parque), y los que pagan con Pase Rápido disminuyen en 1 la cantidad de accesos disponibles. No necesitamos registrar nada adicional para los que pagan en efectivo.

Se tiene el siguiente diagrama de la solución propuesta:



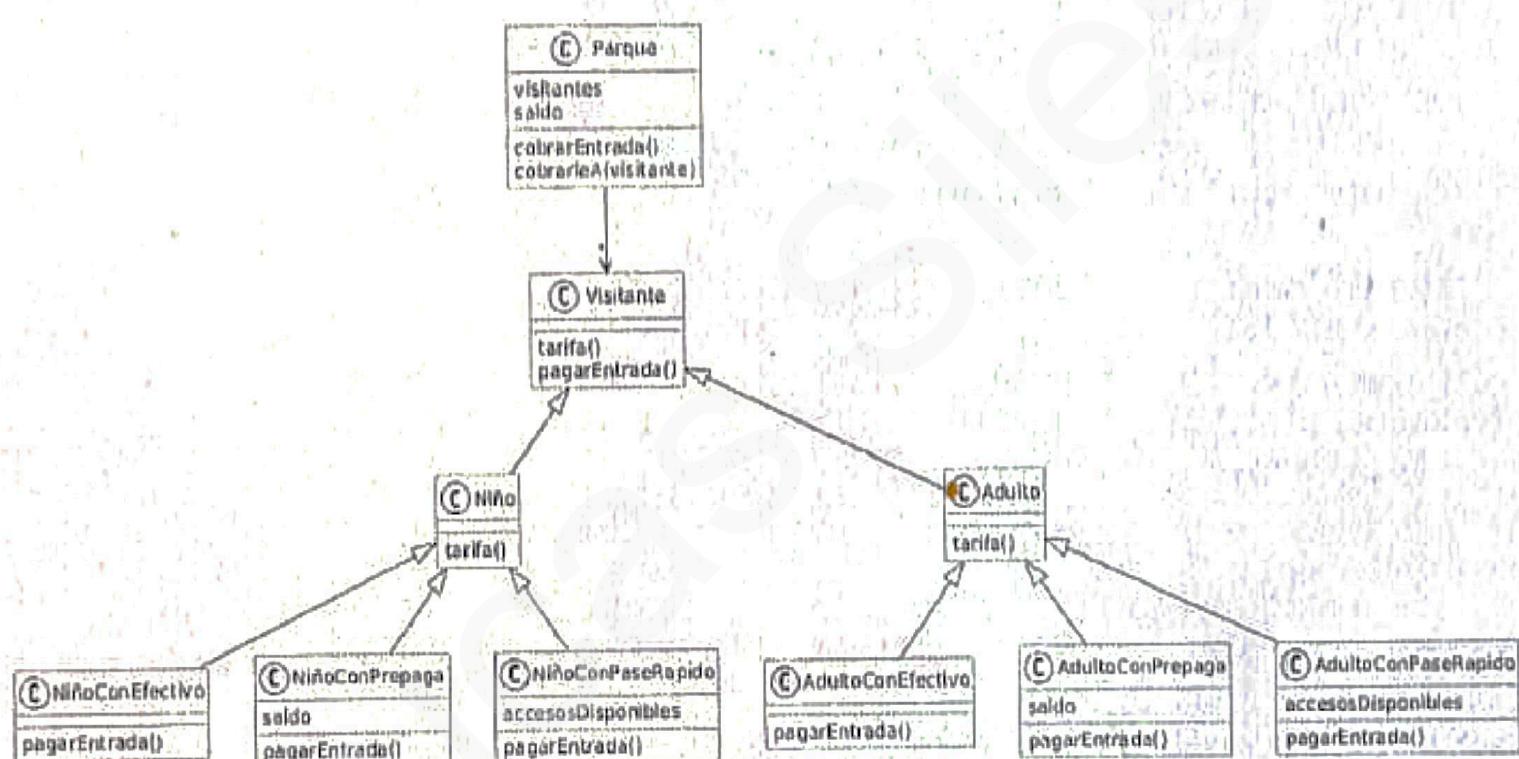
```
class Parque {
    const property visitantes = []
    var property saldo = 0
    method cobrarEntrada() {
        visitantes.forEach { v => self.cobrarleA(v) }
    }
    method cobrarleA(visitante) {
        saldo = saldo + visitante.tarifa()
        visitante.pagarEntrada()
    }
}
```

1. Indicar verdadero o falso y justificar en cada caso:

- Para que la solución propuesta funcione, es necesario que exista la clase Visitante, ya que de lo contrario no podrían incluirse los distintos visitantes en la lista del parque.
- Hay buen uso de polimorfismo en la solución.
- Por cómo se planteó el modelo, se va a repetir lógica en las implementaciones de pagarEntrada().
- Si un adulto que paga con tarjeta prepaga decide pasarse al sistema de Pase Rápido, podrá hacerlo sin problema con este modelo.
- Al cobrarle a un visitante que paga con Pase Rápido y este se queda sin accesos disponibles, se lanza un error, se frena la ejecución y el estado del sistema queda coherente.

2. Desarrollar una nueva solución que resuelva los problemas identificados. Debe incluir el código y un diagrama de clases actualizado.

3. ¿Qué haría falta hacer en cada solución para poder contemplar otros tipos de visitantes, como personas mayores o adolescentes?



## Paradigmas de Programación

## Examen Final

07/12/2024

### Condiciones de aprobación

|                                            |                                                                                                                                                                                                                                                           |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Para aprobar es necesario simultáneamente: | <p><b>En todas tus respuestas sé puntual</b>, no pierdas el foco de lo que se pregunta. Respuestas en exceso generales son tan malas como respuestas incompletas.</p>  |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Se cuenta con información de videojuegos: **título**, **desarrolladora**, un **conjunto de géneros**, y el **año de lanzamiento**. También hay **jugadores** que conforman un comité de evaluación. Cada jugador evalúa videojuegos según su preferencia. Algunos de los jugadores son los siguientes:

- **Juan**. Prefiere los videojuegos desarrollados por Nintendo o que pertenezcan al género de plataformas.
- **Maria**. Prefiere cualquier videojuego lanzado después del año 2015.
- **Pedro**. Prefiere los videojuegos desarrollados por Rockstar Games o que pertenezcan al género de mundo abierto.

Podría haber más jugadores similares a Juan y Pedro, pero fanáticos de otras desarrolladoras o géneros.

El requerimiento principal es obtener el conjunto de videojuegos que sean evaluados positivamente por **todos los jugadores** de un comité.

## Funcional

Se cuenta con las siguientes definiciones:

```
data Videojuego = UnVideojuego {  
    titulo :: String,  
    desarrolladora :: String,  
    generos :: [String],  
    lanzamiento :: Int  
}
```

```
esDesarrolladoPor dev = (==dev).desarrolladora
```

```
perteneceAGenero genero videojuego = elem genero (generos videojuego)
```

```
esReciente videojuego = lanzamiento videojuego > 2015
```

### Tareas:

1.
  - Modelar a los jugadores de ejemplo (Juan, María y Pedro).
  - Hacer la función **preferidosDelComite** que resuelva el requerimiento principal, recibiendo un conjunto de jugadores y varios videojuegos, utilizando una única función auxiliar como máximo. Usar aplicación parcial y expresiones lambda de forma conveniente, indicando por qué se elige una sobre otra en cada caso.
2. Justificar la utilidad del concepto de **orden superior** en la solución planteada (no valen respuestas genéricas, no se pide la definición).
3. Definir los tipos de datos de todas las funciones, incluyendo **esDesarrolladoPor**, **perteneceAGenero** y **esReciente**.

Possible solución: [lucassoliz/practicas-haskell-prolog-wollok](https://github.com/lucassoliz/practicas-haskell-prolog-wollok): Ejercicios y prácticas de Haskell para la facultad

## Objetos

Se cuenta con la clase `Videojuego`, que ya tiene definidos los atributos necesarios y todos los **accessors** que hagan falta (es decir, se pueden usar sin definir). Esta clase puede completarse en los puntos posteriores si se considera necesario.

Tareas:

1.
  - a. Representar a los jugadores y sus formas de evaluar.
  - b. Implementar el método `preferidosDelComite`, que resuelve el requerimiento principal de determinar qué videojuegos cumplen con las preferencias de todos los jugadores del comité.
  - c. ¿Qué objetos se ven involucrados en la resolución del problema (receptor y parámetros de `preferidosDelComite` según corresponda)? Justificar la decisión sobre quién es el receptor del mensaje.
  - d. Si quisieramos tener un jugador flexible, que no tiene una sino **múltiples formas** de evaluar un videojuego y aceptarlo con cualquiera de ellas. ¿Sería posible incorporarlo sin cambiar lo anteriormente desarrollado? Justificar.
2. Se quiere poder representar un comité **exigente**, que además de preferir los videojuegos que cumplen con las preferencias de todos los jugadores que lo componen, los videojuegos preferidos deben pertenecer a al menos **dos géneros destacados**. Realizar los cambios necesarios para resolver este problema.
3. Indicar qué conceptos del paradigma<sup>1</sup> se aplicaron para la solución de los puntos anteriores y dónde se los ve/cómo se los aplica.

## Lógico

La información de los videojuegos se modela mediante un **hecho** por cada videojuego, con la siguiente forma:

```
% videojuego(titulo, desarrolladora, [generos], lanzamiento).
```

Tareas:

1. Definir a los jugadores e implementar **polimórficamente** sus formas de evaluar.
2. Implementar un predicado `preferidoDelComite/1` que, al consultarlo adecuadamente, permita resolver el requerimiento principal. Asumir que el comité está formado por todos los jugadores existentes y se busca analizar a todos los videojuegos, y que se quiere saber solo el título de los videojuegos preferidos.
3. Realizar un predicado `ovejaNegra/1`, que permita saber si un videojuego no es preferido por **ningún jugador**.
4. Para los puntos 2 y 3, mostrar ejemplos de consulta y analizar su **inversibilidad**.

<sup>1</sup> Por favor, no incluir cosas triviales como por ejemplo "objeto", "mensaje" o "referencia". Esos están siempre en cualquier solución. Se pregunta por aquellos para los que se destaca su uso en la solución planteada.

Final 18/10/2024

## Paradigmas de Programación

## Examen Final

18/10/2024

### Condiciones de aprobación

- Para aprobar es necesario simultáneamente:
- completar el 60% del examen, y
  - obtener al menos la mitad de los puntos en cada paradigma.

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.



## Parte A

Queremos estudiar el comportamiento de compra para un vendedor mayorista de productos generales. Contamos con:

```
data Producto = Prod {  
    nombre :: [Char],  
    precio :: Float,  
    precioCantidad :: Float  
}
```

```
queEmpieceConA = (== 'a') . head . nombre  
barato = (< 50) . precio  
  
juan = [ queEmpieceConA , barato ]
```

- a. Modificar la función queEmpieceConA para que acepte un carácter como parámetro, en lugar de que la *a* sea fija.  
b. Indicar qué cambiaría en la representación de juan, y qué concepto está relacionado.  
c. Incluir los tipos de juan y todas las funciones.
2. Codificar una función que permita establecer, dada una lista de productos, qué productos elegiría una persona, asumiendo que elige un producto que cumpla todos los requisitos.

## Parte B

Se tiene el siguiente código Prolog para un sistema de compras de supermercado, con latas y lácteos:

```
precio(lata(atun, 100, nereida), 70).  
precio(lata(atun, 200, nereida), 120).  
precio(lacteo(laSerenisima, leche), 15).  
precio(lacteo(sancor, leche), 22).  
cliente(Cliente):-compro(Cliente, _).
```

```
compro(martina, lata(atun, 100, nereida)).  
compro(martina, lacteo(sancor, leche)).  
compro(aye, lacteo(sancor, leche)).
```

1. Queremos relacionar un cliente y una marca, se cumple si el cliente compró un producto de esa marca. Codificar el predicado que llamaremos comproMarca/2.
2. Se desea saber qué marcas son populares, lo cual se cumple si todos los clientes compraron algún producto de esa marca. Tenemos estas dos soluciones:
  - marcaPopu1(Marca):-forall(cliente(Cliente), comproMarca(Cliente, Marca)).
  - marcaPopu2(Marca):-cliente(Cliente), forall(cliente(Cliente), comproMarca(Cliente, Marca)).Ambas tienen problemas, explique por qué justificando su respuesta para cada una.

## Parte C

Queremos programar un calendario en donde se pueda averiguar si una fecha dada está libre. En el calendario ya hay eventos agendados que pueden ser, por ahora, eventos de día completo o recordatorios. La fecha está libre cuando ninguno de los eventos de día completo ocupa esa fecha, sabiendo que con los recordatorios no hay conflicto. Se tiene la siguiente solución en Wollok.

```
class Calendario {  
    var eventos  
    method estaLibre(fecha) =  
        eventos.all({evento =>  
            evento.esRecordatorio() or (not evento.esRecordatorio() and evento.fecha() != fecha)  
        })  
    }  
class EventoDiaCompleto {  
    var property fecha  
    method esRecordatorio(){ return false }  
}  
  
class Recordatorio {  
    method esRecordatorio(){ return true }  
}
```

1. Dada la solución anterior, responder verdadero o falso y justificar en todos los casos.
  - a. Es necesario agregar la superclase Evento para que los tipos de evento sean polimórficos.
  - b. Es posible agregar un nuevo tipo de evento: los de varios días (en ellos se detalla una lista con todos los días que ocupa) sin cambiar el código de la clase calendario.
2. Proponer una nueva solución (código y diagrama) que resuelva los problemas existentes en el código dado (sin introducir nuevos problemas).

Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

**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.



## Parte A

Queremos estudiar el comportamiento de la gente que se va de vacaciones. Contamos con:

|                                                                                                           |                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>data Destino = Lugar {     nombre :: [Char],     precio :: Float,     atractivos :: [String] }</pre> | <pre>tieneBoliches destino = ((elem 'boliches') . atractivos) destino barato destino = precio destino &lt; 50 alguien = [tieneBoliches, barato]</pre> |
|-----------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|

1. Necesitamos determinar si en un destino hay algún atractivo en particular, no necesariamente boliches:
  - a. Definir la función **tieneAtractivo** y explicitar su tipo.
  - b. Volver a definir **alguien** usando **tieneAtractivo** en vez de **tieneBoliches**, indicando qué concepto del paradigma se aprovecha en esta solución que no se usaba en la definición anterior de **alguien**.
2. Se cuenta con una función que permite establecer, dada una lista de destinos turísticos, cuáles elegiría una persona según sus requisitos (por ejemplo **alguien**), asumiendo que elige un destino que cumpla al menos 1.
 

```
destinosElegidos requisitos destinos = filter (f requisitos) destinos
f requisitos destino = any destino requisitos
```

Responder verdadero o falso, justificar y corregir la implementación en caso de ser posible:

- a. La solución funciona correctamente.
- b. La solución es poco declarativa.
- c. La solución es poco expresiva.
- d. Asumiendo que funciona como se explica, si se la invoca con una lista de destinos infinita no se podrá obtener ninguna respuesta independientemente de cuáles sean los requisitos de la persona.

## Parte B

Dada la siguiente base de conocimiento:

|                                                                                                                                                                  |                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>!genero(Canción, Género). genero(el38, rock). genero(sisters, reggae). genero(muchoPorHacer, rock). genero(tusOjos, reggae). genero(bastara, reggae).</pre> | <pre>!toca(Tema, Banda). toca(el38, divididos). toca(sisters, divididos). toca(muchoPorHacer, riff). toca(tusOjos, losCafres). toca(bastara, losCafres).</pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|

Y el siguiente predicado, que debería ser cierto cuando una banda únicamente toca rock:

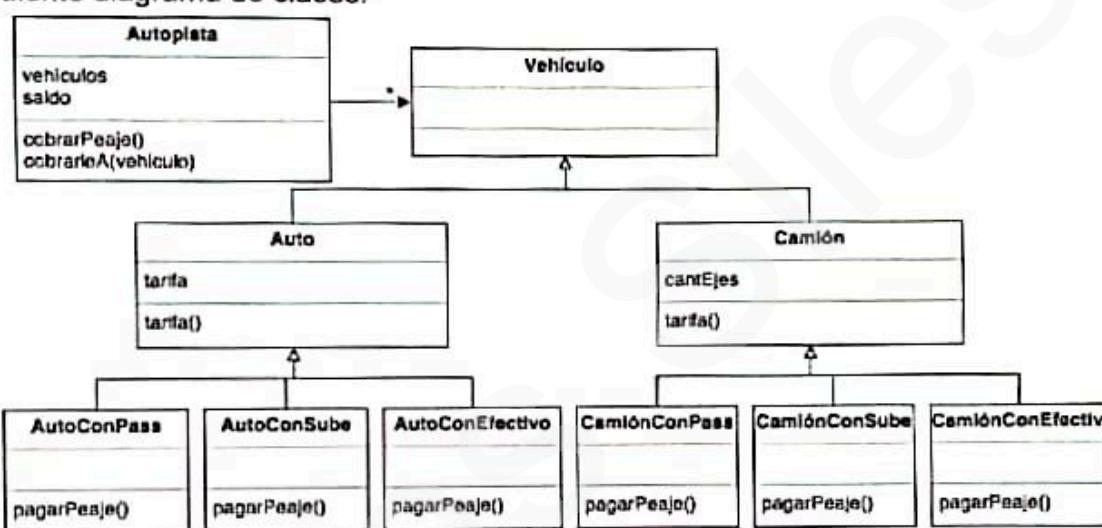
```
rockera(Banda) :-
    findall(Tema, (toca(Tema, Banda), genero(Tema, Genero), Genero\=rock), Temas),
    length(Temas, 0).
```

1. Decir el resultado de las siguientes consultas:
  - a. rockera(riff).
  - b. rockera(divididos).
  - c. rockera(42).
  - d. rockera(Banda).
2. ¿Qué consulta/s no funciona/n como debería/n? Justificar conceptualmente.
3. Escribir una versión más declarativa del predicado **rockera**, solucionando además sus problemas de funcionamiento.

### Parte C

Se sabe que en una autopista se cobra peaje a los vehículos que la usan. Los autos y camiones tienen diferentes formas de calcular la tarifa (en el auto es un valor único y en los camiones según la cantidad de ejes), y además hay vehículos que pagan con telepeaje "Pass", otros con efectivo y otros con la tarjeta Sube. Al efectuarse el cobro suceden dos cosas: el saldo de la empresa que mantiene la autopista aumenta, y además cada vehículo efectúa el pago. Los que pagan con efectivo disminuyen su dinero, los que pagan con sube disminuyen el saldo de la Sube en un 10% más por gasto de servicio (el 10% no va para la autopista), y los que pagan por telepeaje disminuyen en 1 la cantidad de peajes disponibles. La tarifa es \$10 para el auto y \$20 por cada eje en el caso del camión.

Se tiene el siguiente diagrama de clases:



Y suponiendo que el sistema hace lo que debe (funciona) y además esto es parte del código en la clase Autopista:

|                                                                                     |                                                                                                          |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <pre>method cobrarPeaje(){     vehiculos.forEach{ v =&gt; self.cobrarleA(v) }</pre> | <pre>method cobrarleA(vehiculo){     saldo = saldo + vehiculo.tarifa()     vehiculo.pagarPeaje() }</pre> |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|

- Indicar verdadero o falso y justificar en cada caso con prosa y/o código.
  - Para la solución propuesta, es necesario que exista la clase **Vehículo**.
  - Hay forma de implementar el resto del código respetando ese diagrama de clases y sin repetir lógica.
  - Si un auto que paga con Sube decide pasarse al sistema Pass de telepeaje no puede hacerlo con este diseño.
  - Si al realizar el cobro a un vehículo el método **pagarPeaje** tirase error, el sistema quedaría inconsistente.
- Implementar una nueva solución (con código y diagrama) que solucione los problemas encontrados, incluyendo también el código de los vehículos.

### Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen,
- obtener al menos la mitad de los puntos en cada paradigma.

**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.**



## Parte A

Se cuenta con la siguiente información:

```
-- Para cada medicamento:  
amoxicilina = cura "infección"  
bicarbonato = cura "picazón"  
ibuprofeno = cura "dolor" . cura "hinchazón"  
todosLosMedicamentos = [amoxicilina, bicarbonato, ibuprofeno]  
  
cura sintoma = filter (/= sintoma)
```

```
-- Para cada enfermedad / conjunto de síntomas:  
malMovimiento = ["dolor"]  
varicela = repeat "picazón" ^
```

```
-- Asumimos que existe al menos un medicamento capaz de curar cada enfermedad:  
mejorMedicamentoPara sintomas = find (curaTodosLos sintomas) todosLosMedicamentos ^  
curaTodosLos sintomas medicamento = medicamento sintomas == []
```

Se pide:

- Definir el tipo de un medicamento genérico, y de curaTodosLos.
- ¿Cuáles son los dos conceptos funcionales más importantes aplicados en mejorMedicamentoPara y de qué sirvieron?
- ¿Qué responde esta consulta? Justificar conceptualmente. En caso de errores o comportamientos inesperados, indicar cuáles son y dónde ocurren.  
> mejorMedicamentoPara varicela

Si se reemplaza todosLosMedicamentos con lo de abajo, ¿qué respondería la consulta anterior? Justificar conceptualmente.

```
sugestion sintomas = []  
todosLosMedicamentos = [sugestion, amoxicilina, bicarbonato, ibuprofeno]
```

## Parte B

```
% estaEn(Peli,Personaje)  
estaEn(buscandoNemo,nemo).  
estaEn(buscandoNemo,dory).  
estaEn(quienEnganio,rogerRabbit).  
estaEn(quienEnganio,doom).  
estaEn(quienEnganio,eddie).  
estaEn(naufrago,chuck).  
% esAnimado(Personaje)  
esAnimado(nemo).  
esAnimado(dory).  
esAnimado(rogerRabbit).  
esAnimado(doom). %Ups, spoiler
```

```
aptaParaAmargos(Peli):-  
    findall(Pers,(estaEn(Peli,Pers),esAnimado(Pers)),Ps),  
    length(Ps, C), C = 0.  
  
seDivierte(Personaje):-  
    findall(Animado,  
        (estaEn(Peli1,Personaje),  
         estaEn(Peli2,Animado),  
         Peli1 = Peli2,  
         esAnimado(Animado)),  
        Compas),  
    length(Compas, C), C >= 1.
```

- Analizando los predicados aptaParaAmargos y seDivierte

<sup>1</sup>repeat x = x : repeat x

<sup>2</sup>find condicion = head . filter condicion

Paradigmas de Programación

Examen Final

27/07/2024

- seDivierte*
- a. Explicar: ¿Cuándo una peli es apta para amargos? ¿Y cuándo un personaje es buen mime? (No cómo lo hacen, sino qué hacen).
  - b. Dar dos ejemplos de consulta individual para `aptaParaAmargos` y dos para `seDivierte`, una cuya respuesta sea verdadero y una cuya respuesta sea falso, para cada uno. Justifique por cada uno de los casos.
  - c. ¿Qué responde en cada caso la consulta existencial? Justificar conceptualmente.
2. Ambos predicados tienen problemas de declaratividad y de mal uso de pattern matching. Explicar dónde y por qué en cada caso.
  3. Realizar una nueva solución arreglando los problemas mencionados en los puntos anteriores.

### Parte C

Se desea modelar las líneas telefónicas que pueden contratar los usuarios de una empresa para poder tener internet en sus dispositivos. En una línea prepaga, uno puede cargar saldo, que se va agotando con cada kb de internet usado (hay un precio por kb). En una línea con factura, uno va acumulando kb de internet usados, y al final de mes se cobra lo que se debe pagar. En una línea control, que es como la línea con factura más restrictiva, hay un límite de kb y no permite usar internet si se llega a ese límite. Además, se desea poder conocer un string que muestre el estado de la línea, que incluirá en todos los casos el nombre del plan, el número de teléfono, y además si es línea prepaga el saldo, y si es línea factura ó control los kb usados.

**Importante:** una línea debe poder cambiar de plan en cualquier momento.

Se tiene la siguiente solución inicial<sup>3</sup>:

```
class Linea {
    var property plan
    var property numero
    var property saldo
    var property precioKb
    var property kbUsados
    var property limiteKb
    method estado(){
        if (plan == "prepago"){
            return "Tu plan es: " + plan + "tu numero es: " + numero + "tu saldo es: " + saldo
        } else {
            return "Tu plan es: " + plan + "tu numero es: " + numero + "usaste kbs: " + kbUsados
        }
    }
    method usarInternet(kbs){
        if (plan == "prepago"){
            if (saldo < kbs * precioKb){
                return "No alcanza el saldo"
            }
            saldo -= kbs * precioKb
        } else if (plan == "factura") {
            kbUsados += kbs
        } else {
            if(limiteKb < kbUsados + kbs){
                return "Se ha superado el límite"
            }
            kbUsados += kbs
        }
    }
}
```

1. Responder verdadero o falso y justificar en todos los casos:

- a. Sin modificar los métodos existentes puede agregarse fácilmente un nuevo tipo de plan (por ejemplo, un plan "Ilimitado" que siempre permita usar internet), poniendo un string diferente en la variable `plan`.
- b. Luego de agregar un método `llamarPorWhatsapp`, implementado de esta forma:

```
method llamarPorWhatsapp(linea1,linea2){
    linea1.usarInternet(250)
    linea2.usarInternet(250) }
```

Si una de las líneas no tiene saldo o si superó el límite de kb (y, por lo tanto, la llamada no puede ocurrir), no se le gasta indebidamente internet a la otra.

2. Proponer una solución superadora para la solución inicial sin repetición de lógica, manteniendo las funcionalidades mencionadas, y respetando los conceptos y buenas prácticas vistos en la materia.

<sup>3</sup> Nota: Suponer que es posible concatenar un string con un número, usando el método suma (+), por lo que no es un problema cuando se usa el + en el método `estado`.

Final 24/02/2024

## Paradigmas de Programación

## Examen Final

24/02/2024

### Condiciones de aprobación

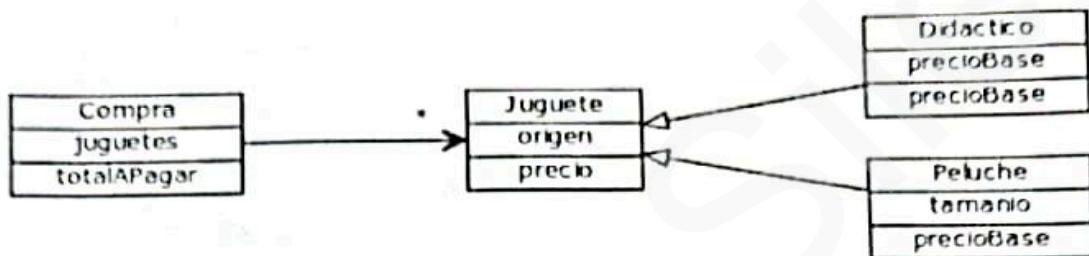
- Para aprobar es necesario simultáneamente
- completar el 60% del examen, y
  - obtener al menos la mitad de los puntos en cada paradigma.

**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



## Parte A

Estamos modelando parte del sistema de una juguetería donde se quiere conocer el precio de los juguetes incluidos en una compra. Sabemos que el precio de cada juguete se calcula como el precio base del mismo, al cual se le hace un recargo de un porcentaje según el país de origen. El precio base de un peluche es 10 veces su tamaño, mientras que para los didácticos se indica para cada uno. Al momento, no hay juguetes que no sean didácticos o peluches. Este es el diseño para esa parte del sistema.



Para conocer el total a pagar, la compra simplemente realiza una sumatoria con el resultado de mandarle el mensaje precio a cada juguete.

Aparece un nuevo requerimiento que indica que el precio de los juguetes existentes se ve afectado por factores adicionales: los que son de primera infancia salen un 10% más caros mientras que los que son para chicos salen 30% más si el mismo tiene publicidades en la tele. No hay juguetes para otras edades.

1. Resolver este nuevo requerimiento incluyendo diagrama de clases y código.
2. En base a la nueva solución responder:
  - a. ¿Cómo se hace para instanciar un juego didáctico de primera infancia?
  - b. ¿Qué impacto tendría sobre tu solución si se quisieran agregar juegos de mesa, que calculan su precio base distinto a los didácticos y los peluches?
  - c. ¿Qué pasa si queremos contemplar juegos de adultos además de los de primera infancia o para chicos?
  - d. ¿Es necesario hacer cambios sobre la lógica inicial de totalAPagar para que el nuevo requerimiento pueda ser soportado? Justificar conceptualmente.

## Parte B

Dada la siguiente expresión:

```
filter ((nombre==).ciudad) destinos
```

1. ¿Cuántas son las funciones involucradas en esa expresión? Decir cuáles son justificando conceptualmente.
2. A partir de la siguiente definición:  

$$f \text{ nombre destinos} = \text{head } (\text{filter } ((\text{nombre}==).\text{ciudad}) \text{ destinos})$$
 Definir el tipo más genérico de ciudad y el tipo más genérico de f.
3. ¿Puedo invocar a f con una lista infinita? ¿Qué casos podrían suceder? Justificar conceptualmente.

## Parte C

Tenemos la siguiente base de conocimientos en Prolog, y nos interesa saber cuáles son los productos que más conviene comprar y en qué local. Para los lavarropas sabemos la marca y el precio, para los televisores además se indica el tamaño en pulgadas. Un precio es más conveniente que otro si es más barato en el caso de los lavarropas, mientras que para los televisores lo es si la relación precio/tamaño es menor.

Siempre la comparación se hace para productos de la misma marca, y el local en el que más conviene comprar será el que tenga el precio más conveniente de todos los locales que vendan el producto comparado. Se necesita poder realizar una única consulta que permita saber qué electrodoméstico conviene comprar en qué local.

```
vende(garparino, lavarropas, ge, 9500).  
vende(rolo, lavarropas, ge, 9000).  
vende(frivola, lavarropas, ge, 12000).  
vende(garparino, tv, lg, 25, 8000).  
vende(rolo, tv, lg, 35, 9000).  
vende(rolo, tv, samsung, 35, 10000).
```

```
masConvenienteEn(lavarropas, Marca, Local):-  
    vende(Local, lavarropas, Marca, PrecioMenor),  
    vende(_, lavarropas, Marca, OtroPrecio),  
    PrecioMenor < OtroPrecio.  
masConvenienteEn(tv, Marca, Pulgadas, Local):-  
    vende(Local, tv, Marca, Pulgadas, PrecioMenor),  
    vende(_, tv, Marca, OtrasPulgadas, OtroPrecio),  
    PrecioMenor / Pulgadas < OtroPrecio / OtrasPulgadas.
```

1. Responder V/F a las siguientes afirmaciones y justificar conceptualmente:
  - a. No hay problemas de inversibilidad en ningún predicado.
  - b. Se está trabajando polimórficamente con los distintos tipos de productos.
  - c. La lógica general para determinar si es más conveniente comprar un producto en un local no es correcta.
2. Realizar los cambios que consideres necesarios para resolver los problemas detectados.

Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

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.

## Parte A

Dadas las siguientes definiciones

habilidades :: Persona -> [Habilidad]

sirve :: Problema -> Habilidad -> Bool

1. Usando las funciones `habilidades` y `sirve`, definir una función que dado un problema y una lista de personas permita saber qué personas tienen alguna habilidad que sirva para el problema recibido.

podrianAyudar problema personas = ...

2. Indicar qué concepto/s de los siguientes se está/n utilizando en la solución anterior y para qué:

- Orden Superior
- Composición
- Aplicación Parcial
- Pattern matching

3. Si una persona tuviera una lista infinita de habilidades, ¿cómo se comportaría `podrianAyudar` si dicha persona estuviera en la lista recibida?

## Parte B

Asumiendo que en la base de conocimiento hay numerosos hechos de los siguientes predicados:

`tiene(Persona, Cosa)`.

`vale(Cosa, Valor)`.

1. Completar lo necesario para que `todoLoQueTieneEsMasValioso` permita saber si todo lo que tiene la primera persona es más valioso que lo que tiene la segunda.

`todoLoQueTieneEsMasValioso( Personal, Persona2 ) :-`

`forall( (.....), ValorCosaValiosa > OtroValor ).`

2. ¿Es posible usar `todoLoQueTieneEsMasValioso` para consultar si nadie cumple con tener todas cosas más valiosas que una persona indicada?

- a. Si es posible, explicar qué es lo que permite dicho uso.

En caso de no ser posible con la solución del punto 1 tal y como está, explicar por qué, y hacer los cambios necesarios de modo que sí lo permita.

- b. Mostrar qué consulta debería realizarse para saber si nadie tiene todas cosas más valiosas que pedro.

## Paradigmas de Programación

## Examen Final

17/02/2024

### Parte C

En nuestro sistema, para que una película rompa estereotipos, en principio su protagonista no debe ser varón. Además se debe cumplir: en el caso de las películas de aventura, que ese protagonista no sea rescatado; en el caso de las películas de terror, que todos los personajes sean rescatados; y en el caso de las de comedia, que sólo tengan un personaje. Se sabe que los personajes animados no tienen género definido y nunca son rescatados, y los personajes actuados pueden tener distintos géneros y ser o no rescatados en la película. Se tiene la siguiente solución:

```
class Pelicula {
    var personajes = []
    method protagonista() = personajes.first()
}

class PeliAventura inherits Pelicula {
    method rompeEstereotipos(){
        return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
            and self.protagonista().esRescatado().negate()
    }
}

class PeliTerror inherits Pelicula {
    method rompeEstereotipos(){
        return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
            and personajes.all({p => p.esRescatado()})
    }
}

class PeliComedia inherits Pelicula {
    method rompeEstereotipos(){
        return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
            and personajes.size() == 1
    }
}

class PersonajeAnimado {
    method sosActuado() = false
    method esRescatado() = false
}

class PersonajeActuado {
    var genero
    var esRescatado
    method sosActuado() = true
    method genero() = genero
    method esRescatado() = esRescatado
}
```

1. Responder verdadero o falso y **justificar conceptualmente** en todos los casos:
  - a. Escribiendo un método `elProtagonistaNoEsVaron` en la superclase y usándolo en las subclases se elimina toda repetición de lógica.
  - b. Se está rompiendo el encapsulamiento de los personajes.
  - c. Como no hay ifs, se está haciendo un buen uso del polimorfismo.
2. Codificar una nueva solución arreglando lo que haga falta en base a las respuestas anteriores. Incluir un diagrama estático de la nueva solución.

## Parte A

Dadas las siguientes definiciones

tomar::: Bebida -> Persona -> Persona  
Devuelve a la persona luego de haberse tomado la bebida

mezclar::: Bebida -> Bebida -> Bebida  
Obtiene la nueva bebida resultante de mezclar las dos bebidas dadas

1. Completar la definición de una función que permita obtener cómo queda un conjunto de personas luego de que cada una se tome un "cóctel" producto de mezclar varias bebidas.

seTomanUnCoctelConEstasBebidas bebidas personas =

map ( .....tomar.....bebidas.....bold1.....bebida.....mezclar..... ) personas

Indicar qué concepto/s se están utilizando en la solución anterior:

- Orden Superior |  Listas infinitas |  Aplicación Parcial |  Efecto

## Parte B

Asumiendo que en la base de conocimiento hay numerosos hechos del siguiente predicado:  
tiene(Persona, Cosa, Cantidad).

por ejemplo:

tiene(tito, arboles, 5).  
tiene(tito, problemas, 2).

predicadoMisterioso( ....Persona.....Cosa.....CantidadM..... ):-  
tiene( ....Persona.....Cosa.....CantidadM..... ),  
forall( tiene( ....Persona.....Cosa.....Cantidad..... ), CantidadM >= Cantidad ).

1. Completar los ... de la solución propuesta del predicadoMisterioso,
2. hacer las consultas adecuadas
3. y, para ganar expresividad, ponerle mejor/es nombre/s a predicadoMisterioso,  
para:

- a. permitir encontrar cuál es la cosa que tito tiene en mayor cantidad;
- b. permitir averiguar la cantidad máxima que cualquier persona tenga de una cosa dada:

16/12/2023

## Paradigmas de Programación

### Examen Final

#### Parte C

Se cuenta con la siguiente definición:

```
class Radio {
    var property recursos
    var property estilo
    method muchosAnunciantes() = // Correctamente implementado
    method esMovilizante () = // Correctamente implementado
    method mejorar() { // Correctamente implementado }
    method cambioDeEpoca() {
        if (estilo == "comunitaria" && self.esMovilizante() )
            recursos = 0
        if (estilo == "anarcoCapitalista")
            self.mejorar()
        if (estilo == "comercial" && self.muchosAnunciantes() && recursos > 100
        )
            estilo = "anarcoCapitalista"
    }
}
```

Se asume que el código faltante (indicado como "// correctamente implementado") está correctamente implementado por lo que la solución pasa todos los tests que pudiera haber.

Si un profesional de sistemas quiere refactorizar la solución, definiendo para ello tres objetos nuevos y delegando en ellos parte de las responsabilidades del sistema.

1. Completar la definición de estos objetos y reescribir el código del método `cambioDeEpoca()`

```
class Radio {
    // Tiene los mismos métodos y propiedades que arriba.
    method cambioDeEpoca() {
        estilo = self.cambioDeEpoca(3self)
    }
}

object comunitaria {
}

object anarcoCapitalista {
}

object comercial {
}
```

2. Indicar qué concepto/s se utilizaron en el cambio de solución:

Polimorfismo |  Herencia |  Redefinición |  Clase abstracta

## Paradigmas de Programación

## Examen Final

27/09/2023

### Condiciones de aprobación

- Para aprobar es necesario simultáneamente:
- completar el 60% del examen, y
  - obtener al menos la mitad de los puntos en cada paradigma.

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.



### PARTE A - Lógico

Dada la siguiente base de conocimiento que modela las visitas de gente a atracciones de un parque de diversiones, y si disfrutaron de las mismas:

```
subioA(prudencio, montaniaRusa(hulk)).  
subioA(prudencio, autosChocadores).  
subioA(hermenegildo, montaniaRusa(rockit)).  
subioA(hermenegildo, simulador(minions)).  
subioA(hermenegildo, autosChocadores).  
subioA(rigoberta, simulador(ikran)).  
subioA(brunilda, simulador(simpsons)).  
  
disfruto(prudencio, montaniaRusa(hulk)).  
disfruto(prudencio, autosChocadores).  
disfruto(hermenegildo, simulador(minions)).  
disfruto(rigoberta, simulador(ikran)).  
foo(Persona) :- forall(subioA(Persona, Atraccion), disfruto(Persona, Atraccion)).
```

Se pide:

- ¿Cuál es el objetivo del predicado `foo/1`? Elegir un nuevo nombre que sea más expresivo y explicar para qué personas debería ser cierto dada la base de conocimientos actual.
- Analizar la inversibilidad del predicado `foo/1` y, en caso de que no sea inversible, modificar todo lo que sea necesario para que lo sea.
- Decimos que una persona es valiente cuando subió a una montaña rusa, sin importar si fue la de hulk o la de rockit. ¿Cómo se implementaría esa definición?

### PARTE B - Funcional

Dado el siguiente fragmento de código en Haskell:

```
data Cafe =  
    Cafe { tipo :: String, tamano :: String, calorias :: Double, precio :: Double } deriving(Show)  
  
type Agregado = Cafe -> Cafe  
  
aumentarPrecio monto cafe = cafe{ precio = precio cafe + monto }  
modificarCalorias modificacion cafe = cafe{ calorias = (modificacion . calorias) cafe }  
  
almendras :: Agregado  
almendras = ... implementar ...  
especialDulceDeLeche :: Agregado  
especialDulceDeLeche = ... implementar ...
```

Se pide:

- Indicar los tipos de las funciones `aumentarPrecio` y `modificarCalorías`.
- Implementar los agregados de `almendras` y `especialDulceDeLeche` usando composición y aplicación parcial de funciones de forma tal que:

## Paradigmas de Programación

## Examen Final

27/09/2023

- a. Las almendras aumenten en 80 las calorías del café y su precio sea 150 más caro.  
b. El especial dulce de leche triplique las calorías del café y su precio sea 350 más caro.
3. Dada una lista infinita de agregados:
- ¿Podríamos conocer el estado final de un café luego de realizar todos los agregados?
  - ¿Sería posible conocer cómo quedaría el café luego de realizar cada agregado individualmente de las otras si sólo nos interesan los primeros resultados?
- Justificar conceptualmente cada respuesta.

## PARTE C - Objetos

Se cuenta con información de los restaurantes: se conocen los platos de su menú, y para cada uno de los platos se sabe el origen, chef que lo cocina, la cantidad de calorías y si contiene gluten. También hay críticos de restaurantes que conforman un jurado. Cada crítico evalúa los restaurantes teniendo en cuenta su preferencia. Algunos de los críticos son los siguientes:

- Anacleto. Prefiere restaurantes con algo de comida de origen chino.
- Eustaquia. Prefiere los restaurantes con variedad, entendiendo esta como más de 50 platos en su menú.
- Margarito. Prefiere los restaurantes "light", que son aquellos que tienen sólo platos de menos de 300 calorías.

Se cuenta con la clase Restaurante, que ya tiene definidos los atributos necesarios para representar la información indicada. También hay una clase Chef con su nombre y si ganó algún premio. Estas clases pueden extenderse en los puntos posteriores, si se considera necesario. Asumir que hay métodos de acceso para todos los atributos que pueden usarse en la solución.

1. Necesitamos implementar un método `elegidosDelJurado` que resuelve el requerimiento principal de determinar qué restaurantes de un festival cumplen con las preferencias de todos los críticos del jurado.
  - Representar a los críticos con sus preferencias de modo que puedan formar parte del jurado.
  - Implementar `elegidosDelJurado` justificando la decisión sobre quién es el receptor del mensaje y qué parámetros recibe.
2. Necesitamos agregar a Rosenda, una crítica que prefiere los restaurantes con variedad y también los que tienen comida peruana. ¿Sería posible incorporar a Rosenda sin cambiar lo anteriormente desarrollado? Justificar y en caso de que la solución no lo permita, explicar el motivo.
3. Se quiere poder representar un jurado exigente, que además de preferir los restaurantes que cumplen con las preferencias de todos los críticos que lo componen, los restaurantes elegidos deben tener algún chef que haya sido premiado.
  - Realizar los cambios necesarios para resolver este problema.
  - Explicar qué conceptos del paradigma se aplicaron para resolverlo y qué beneficios aportan.

## Paradigmas de Programación

## Examen Final

17/12/2022

### Condiciones de aprobación

Para aprobar es necesario simultáneamente:

- completar el 60% del examen, y
- obtener al menos la mitad de los puntos en cada paradigma.

**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.**



## Parte A

Queremos estudiar el comportamiento de compra para un vendedor mayorista de productos generales. Contamos con:

|                                                                             |                                                                                                                                   |
|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <pre>data Producto = Prod {     nombre :: String,     precio :: Int }</pre> | <pre>queEmpieceConA = (== 'a') . head . nombre barato = (&lt; 50) . precio restriccionesDeCompra = [queEmpieceConA, barato]</pre> |
|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|

- Definir los tipos de la función `restriccionesDeCompra`.
- Modificar la función `queEmpieceConA` para que verifique que el nombre del producto comience con una letra que se pase como parámetro. Indicar qué cambiaría en la representación de `restriccionesDeCompra` y qué concepto está relacionado.
- Codificar una función que permita establecer, dada una lista de productos y una lista de restricciones, cuáles productos verifican todas las restricciones. ¿Qué conceptos se destacan en tu solución y en dónde?

## Parte B

Tenemos un predicado `toma/2` que relaciona a una persona con aquella bebida que le gusta tomar. La bebida puede ser cerveza, que tiene una variedad, un amargor y un porcentaje de alcohol (0 si es cerveza sin alcohol), o vino, que tiene un tipo y una cantidad de años de añejamiento. El vino siempre tiene alcohol. Y también existen gaseosas varias.

|                                                                                                                                                                      |                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>toma(juan, coca). toma(juan, vino(malbec, 3)). toma(daiana, cerveza(golden, 18, 0)). toma(gisela, cerveza(ipa, 52, 7)). toma(edu, cerveza(stout, 28, 6)).</pre> | <pre>tieneProblemas(Persona):-      findall(C, (toma(Persona, cerveza(C,_,A)), A&gt;0), Cs),     findall(V, toma(Persona, vino(V,_)), Vs),     findall(T, toma(Persona, T), Ts),     length(Cs, CCs),     length(Vs, CVs),     length(Ts, CTs),     CTs is CCs + CVs.</pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- Justificar V o F
  - No se repite lógica, dado que la estructura de las bebidas alcohólicas son distintas.
  - La solución planteada para `tieneProblemas/1` es declarativa.
  - La solución planteada podría mejorarse con polimorfismo.
- Explique y justifique cuál es el significado de lo que se estaría consultando con el siguiente código:  
?- `tieneProblemas(P).`
- Implemente una solución superadora de `tieneProblemas/1`.

## Parte C

Queremos hacer un programa que nos diga si una persona puede jugar a un determinado juego. Se sabe que para jugar a la pelota la persona tiene que tener más de un año de edad, para los juegos de mesa tiene que estar en el rango de edad que indica el juego y para los videojuegos tiene que tener una edad superior a la mínima determinada para el género del mismo y menor a 80. Además, los videojuegos sólo se pueden jugar si se tiene una consola que lo soporte, con lo cual sólo podrá jugarlo si tiene alguna de esas consolas.

Se tiene el siguiente código Wollok para resolver lo pedido:

Paradigmas de Programación

Examen Final

17/12/2022

```

class Persona {
    var property edad
    var property consolas = []
    method puedeJugar(unJuego) {
        if(unJuego.tipo() = "Juego de mesa")
            return edad.between(unJuego.edadMinima(), unJuego.edadMaxima())
        if(unJuego.tipo() = "Videojuego") {
            if(edad.between(unJuego.genero().edadMinima(), 80)) {
                consolas.forEach{
                    consola => unJuego.consolasPosibles().forEach{
                        otraConsola => if(consola == otraConsola) return true
                    }
                }
                return false
            } else return false
        }
        return edad > 1 // si llega hasta acá, es porque es una pelota
    }
}

```

|                                                                                               |                                                                                                |                                                                       |
|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| class Juego {<br>var property tipo<br>var property edadMinima<br>var property edadMaxima<br>} | class Videojuego inherits Juego {<br>var property consolasPosibles<br>var property genero<br>} | class genero {<br>var property titulo<br>var property edadMinima<br>} |
|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|

1. Critique la solución presentada en términos de **declaratividad, polimorfismo, delegación, encapsulamiento y herencia** marcando los diferentes lugares en los cuales encuentra problemas.
2. Codifique una nueva solución teniendo en cuenta los problemas detectados y fundamente los cambios realizados conceptualmente para cada cambio (no se considerarán respuestas genéricas).

Final 07/09/2022

## Parte A

Queremos hacer un pequeño programa para modelar superhéroes y villanos de la empresa "Maravilla". Nos piden implementar los ataques de los personajes a otros personajes teniendo en cuenta que los personajes pueden ser héroes o villanos, y debe ser posible que un héroe se convierta en villano y viceversa. Un héroe sólo puede atacar a sus enemigos, mientras que un villano puede atacar a cualquier personaje. Luego de dañar al personaje atacado, el otro personaje lo considerará su enemigo.

Un héroe ataca con tanto poder como la suma de los poderes de sus habilidades multiplicada por la cantidad de aliados del héroe. Un villano en cambio usa su arma para atacar, la cual produce un determinado daño que se calcula como su daño base dividido por su área de efecto.

Tenemos el siguiente planteo para la solución:

```
class Personaje {  
    const enemigos = #{}  
    method recibirDaño(cantidad) {  
        /* Asumir que funciona */  
    }  
    method agregarEnemigo(personaje) {  
        enemigos.add(personaje)  
    }  
}  
  
class Villano inherits Personaje {  
    var property arma  
    method atacar(personaje) {  
        personaje.recibirDaño(self.dañoDeArma())  
        personaje.agregarEnemigo(self)  
    }  
    method dañoDeArma() = arma.dañoBase() /  
        arma.areaDeEfecto()  
}  
  
class Arma {  
    var property dañoBase  
    var property areaDeEfecto  
}
```

```
class Heroe inherits Personaje {  
    const habilidades = #{}  
    const aliados = #{}  
  
    method atacar(personaje) {  
        if (not enemigos.contains(personaje)) {  
            return "El personaje atacado  
no es un enemigo"  
        }  
        personaje.recibirDaño(self.poder())  
        personaje.agregarEnemigo(self)  
    }  
  
    method poder() = aliados.size() *  
        habilidades.sum({h => h.poder()})  
}  
  
class Habilidad {  
    var property poder  
}
```

### 1. Responder Verdadero o Falso y justificar:

- El ataque de un héroe no maneja correctamente el caso de atacar a un personaje que no es enemigo.
  - Hay lógica común entre villanos y héroes que podría generalizarse.
  - No hay problemas de delegación en la solución.
  - Los héroes son polimórficos con las habilidades, ya que ambos entienden el mensaje `poder()`.
  - Para que un héroe se convierta en villano alcanza con instanciar `Villano`, inicializándolo con el arma que corresponda y los enemigos que tenía la instancia original de la clase `Heroe`.
2. Proponer una solución alternativa realizando las mejoras que consideres apropiadas en base a las observaciones del punto anterior.

## Parte B

Tenemos un predicado `toma/2` que relaciona a una persona con aquella bebida que le gusta tomar. La bebida puede ser cerveza, que tiene una variedad, un amargor y un porcentaje de alcohol (0 si es cerveza sin alcohol), o vino, que tiene un tipo y una cantidad de años de añejamiento. El vino siempre tiene alcohol. Y también existen gaseosas varías.

```
toma(juan, coca).
toma(juan, vino(malbec, 3)).
toma(daiana, cerveza(golden, 18, 0)).
toma(gisela, cerveza(ipa, 52, 7)).
toma(gisela, vino(malbec, 3)).
toma(edu, cerveza(stout, 28, 6)).
```

```
tieneProblemas(Persona):-  
    findall(C, (toma(Persona, cerveza(C,_,A)), A>0), Cs),  
    findall(V, toma(Persona, vino(V,_)), Vs),  
    findall(T, toma(Persona, T), Ts),  
    length(Cs, CCs),  
    length(Vs, CVs),  
    length(Ts, CTs),  
    CTs is CCs + CVs.
```

1. Explicar cuál es el objetivo del predicado `tieneProblemas/1` (no explicar la forma en que lo resuelve, sólo para qué sirve) y cuál será la respuesta a la consulta:  
`?- tieneProblemas(juan).`
2. Responder Verdadero o Falso y justificar:
  - a. El predicado `tieneProblemas/1` no es inversible.
  - b. La solución planteada para `tieneProblemas/1` es declarativa.
  - c. La solución planteada podría mejorarse usando polimorfismo.
3. Implementar una solución de `tieneProblemas/1` con las mejoras que consideres apropiadas.

## Parte C

Se cuenta con la siguiente información:

-- Para cada medicamento:

`amoxicilina = cura "infección"`

`bicarbonato = cura "picazón"`

`sugestion _ = []`

`cura sintoma = filter (/= sintoma)`

-- Para cada enfermedad / conjunto de síntomas:

`malMovimiento = ["dolor agudo", "hinchazón"]`

`varicela = repeat "picazón" ^ 1`

`mejorMedicamentoPara sintomas = head . filter (idealPara sintomas)`

`idealPara sintomas medicamento = medicamento sintomas == []`

Se pide:

1. Definir el tipo Medicamento en base al modelo dado, y explicitar el tipo de la función `mejorMedicamentoPara`.
2. Explicar qué beneficio aporta el uso de orden superior en la definición de `mejorMedicamentoPara`.
3. Definir y explicitar el tipo del `ibuprofeno`, para que pueda usarse como medicamento, que cure tanto el "dolor agudo" como la "hinchazón" si es de más de 500 miligramos, y de lo contrario cure el "dolor moderado". Armar una lista de medicamentos que incluya amoxicilina e ibuprofeno de 400 miligramos. En caso de que se esté aprovechando algún concepto importante para llevarlo a cabo, mencionarlo.
4. ¿Qué sucederá al evaluar las siguientes consultas? Justificar conceptualmente. En caso de errores o comportamientos inesperados, indicar cuáles son y dónde ocurren.
  - a. `mejorMedicamentoPara malMovimiento (repeat bicarbonato)`
  - b. `mejorMedicamentoPara varicela [sugestion, bicarbonato, amoxicilina]`

Final 22/02/2020

## Parte A

Una empresa inmobiliaria tiene la siguiente base de conocimiento para sus avisos. Aparentemente aseguran que existe cierta relación entre las ubicaciones altas, el precio y el barrio.

Esto quiere decir que a partir de cierto precio en determinado barrio, todos los anuncios les pasa que:

- Todas las casas tienen más de 3 pisos
- Todos los PH tienen más de 2 pisos
- Todos los dptos están ubicados por encima del 5to piso

Para esto se tiene la siguiente solución en prolog

```
% precio(casa(ambientes, pisos), barrio, precio).  
% precio(dpto(ambientes, piso, totalUnidades), barrio, precio).  
% precio(ph(ambientes, pisos, totalUnidades), barrio, precio).  
  
precio(casa(4, 3), Flores, 1400000).  
precio(dpto(4, 6, 28), Palermo, 1800000).  
precio(ph(4, 2, 4), Flores, 1200000).
```

```
todosAltosConPrecioEn(Precio, Barrio):-  
    precio(casa(_, Pisos), Barrio, Precio1),  
    Precio1 >= Precio, Pisos > 3,  
    precio(dpto(_, Piso, _), Barrio, Precio2),  
    Precio2 >= Precio, Piso > 5,  
    precio(ph(_, Pisosp, _), Barrio, Precio3),  
    Precio3 >= Precio, Pisosp > 2,
```

Se pide:

1. Responder verdadero o falso y justificar en todos los casos:
  - a. La solución propuesta para todosAltosConPrecioEn/2 es correcta.
  - b. El uso de polimorfismo para verificar la altura es correcto.
  - c. Como se plantea la solución el predicado es totalmente inversible.
2. Armar una solución superior que mejore los problemas detectados en el punto anterior.

## Parte B

Tenemos estas declaraciones:

edad:: Persona -> Float (no importa cómo se define)

```
f1 _ [] = False  
f1 numero (x:xs) = 2 * x < numero || f1 numero xs  
  
f2 e lista = length (filter (\x -> edad x < (e)) lista) >= 1  
  
f3 tope palabras = any ((< tope).length) palabras
```

Las funciones dadas andan, pero queremos analizarlas más en profundidad:

1. Explicar qué es lo que hace cada una y proponer mejoras en términos de expresividad.
  2. Comparar las funciones f1 y f2, indicando qué conceptos pueden verse en una y no en la otra. En caso de que una sea mejor a la otra en algún sentido, explicar cuál y por qué motivo.
  3. Comparar las funciones f2 y f3, indicando qué conceptos se aprovechan en una y no en la otra. Explicar para qué se están usando.
  4. Indicar para cada una si podría terminar de evaluarse ante una lista infinita como parámetro, justificando el motivo tanto si podrían terminar como si no.
  5. Si bien el objetivo de las funciones f1, f2 y f3 es distinto y se resuelven de formas diferentes, hay una idea general que se repite en las 3.
- Extraer toda la lógica común en una función nueva y volver a definir f1, f2 y f3 usándola adecuadamente.

### Parte C

Necesitamos modelar un sistema de venta de helado para una heladería. Por ahora tienen dos productos: kg de helado y cucuruchos, estos últimos pueden venir con un recargo por cobertura de chocolate. Podemos suponer que los cucuruchos valen \$50 más un recargo según la cobertura de cada uno, y el kg de helado \$200. Se desea realizar una venta acumulando lo facturado y disminuyendo el stock de helado de acuerdo a lo vendido. El cucuricho lleva 200 g de helado.

```
class Heladeria {  
    var facturado  
    var stock  
  
    method vender(prod){  
        if(prod.tipo() == 1){  
            facturado += 200  
            stock = (stock-1).max(0)  
        }  
        if (prod.tipo() == 2){  
            facturado += 50 + prod.recargoCobertura()  
            stock = (stock-0.2).max(0)  
        }  
    }  
}
```

```
class KgDeHelado{  
    method tipo(){ return 1 }  
}  
  
class Cucuricho {  
    var property recargoCobertura  
    method tipo(){ return 2 }  
}
```

1. Responder verdadero o falso y justificar en todos los casos:
  - a. El sistema tiene un problema: cuando se intenta hacer una venta sin stock suficiente, se factura igual. Debería no hacer nada si no hay stock.
  - b. Hay un buen uso de polimorfismo, ya que si quiero agregar un nuevo producto no debo modificar ningún producto existente.
  - c. Las responsabilidades están mal distribuidas.
  - d. Se rompe el encapsulamiento del cucuricho.
2. De acuerdo a lo justificado en el punto anterior, escriba una nueva solución incluyendo diagrama de clases, de manera que se corrijan los problemas mencionados.
3. Al requerimiento inicial se suma un pedido: queremos que se cobre 20% más barato el helado si el mismo es de gustos tradicionales (lo cual se indicará en cada caso). Ese descuento no se aplicaría sobre el recargo de cobertura que tengan los cucuruchos.  
Aregar la lógica pedida a la nueva solución de modo que no haya repetición de lógica.

Possible solución: [lucassoliz/practicas-haskell-prolog-wollok: Ejercicios y prácticas de Haskell para la facultad](https://lucassoliz/practicas-haskell-prolog-wollok)

## Parte A

La fábrica de muebles Armando requiere de su manejo de stock de los productos que realiza en sus distintos locales. Manejan los siguientes productos:

- Sillones: que tiene un tipo (común, cama, reclinable) y cantidad de módulos
- Mesas: forma (rectangular, cuadrada, circular) y material (madera, vidrio)
- Sillas: material (metal, madera)

Se tiene la siguiente base de conocimiento:

|                                                                                                                                                      |                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| %stock(sucursal, producto, cantidad)<br>stock(boedo, sillón(comun, 3), 4).<br>stock(boedo, silla(madera), 12).<br>stock(flores, sillón(cama, 2), 1). | stock(flores, silla(metal), 4).<br>stock(belgrano, sillón(reclinable, 2), 3).<br>stock(belgrano, silla(madera), 8). |
|------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|

Realizar la **codificación** y las **justificaciones** para cada punto:

1. Sabiendo que tenemos los siguientes clientes:

- Mati, que busca una mesa circular de vidrio y 4 sillas de metal.
- Leo, que busca un sillón cama de 2 módulos y otro reclinable de 1.

Agregar la información a la base de conocimientos, sabiendo que se debe poder responder la consulta “¿Qué busca Leo?” (por ejemplo). ¿Hace falta usar listas para representar la información? Si es posible, hacerlo sin usar listas y explicar los conceptos que lo permiten, y en caso contrario hacerlo con listas y explicar por qué son necesarias.

2. Saber si una sucursal trabaja un determinado material. Trabaja el mismo si alguno de sus artículos son de ese material, y se sabe que todos los sillones que trabajan son de madera. ¿Qué concepto resalta en la resolución de este punto y dónde puede verse?
3. Saber si hay una sucursal ideal para un cliente, del cual se conoce su nombre y la información que se agregó en los puntos anteriores. Una sucursal es ideal si tiene en stock todo lo que el cliente busca. ¿Qué concepto aparece que no estaba siendo usado antes?

## Parte B

Se tiene la siguiente función:

```
funcion x y lista = (filter (> x) . map (\ f -> f y)) lista
```

1. Explicar lo que hace y proponer mejoras en términos de expresividad.
2. ¿Sería posible evaluar la función con una lista infinita de modo que dicha evaluación termine? Justificar conceptualmente y plantear un ejemplo para fundamentar la respuesta.
3. Indicar cuáles de las siguientes expresiones son válidas. Para las que son válidas indicar además el tipo de retorno y para las que no son válidas justificar el motivo.
  - a. funcion 3 "hola" []
  - b. funcion 3 7
  - c. funcion "chau" "hola" [length]

## Parte C

Un taller de confección de ropa de moda nos pide un sistema para calcular el tiempo de confección de sus pedidos. Existen faldas, blusas y shorts, y las prendas se pueden hacer utilizando distintas telas: modal, lycra o denim. Las faldas tienen un tiempo de fabricación de 120 minutos, las blusas 200 minutos y 5' adicionales por cada botón, y para los shorts el tiempo depende de cada uno. Ahora bien, realizar prendas en lycra aumenta el tiempo 20% porque el corte y la costura de telas elásticas es más difícil, y como no hay máquinas para coser denim, se terceriza, por lo que el tiempo con denim aumenta 25 minutos por prenda. Con modal no aumenta el tiempo, es el de cada prenda.

Se tiene el siguiente código:

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> class Pedido {     var faldas     var blusas     var shorts     method tiempoDeConfeccion(){         var tiempoTotal = 0         faldas.forEach({falda =&gt;             tiempoTotal += falda.tiempo()})         blusas.forEach({blusa =&gt;             tiempoTotal += blusa.tiempo()})         shorts.forEach({short =&gt;             tiempoTotal += short.tiempo()})         return tiempoTotal     } }  class FaldaModal {     method tiempo(){         return 120     } }  class BlusaModal {     var cantBotones     method tiempo(){         return 200 + cantBotones * 5     } }  class ShortModal {     var tiempoNecesario     method tiempo(){         return tiempo     } } </pre> | <pre> class FaldaLycra {     method tiempo(){         return 120 * 1.2     } }  class BlusaLycra {     var cantBotones     method tiempo(){         return (200 + cantBotones * 5) * 1.2     } }  class ShortLycra {     var tiempoNecesario     method tiempo(){         return tiempoNecesario * 1.2     } }  class FaldaDenim {     method tiempo(){         return 120 + 25     } }  class BlusaDenim {     var cantBotones     method tiempo(){         return 200 + cantBotones * 5 + 25     } }  class ShortDenim {     var tiempoNecesario     method tiempo(){         return tiempoNecesario + 25     } } </pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

1. Responder V o F y justificar conceptualmente en todos los casos.
  - a. No está bien aprovechado el polimorfismo entre las prendas.
  - b. Entre las faldas no hay repetición de lógica.
  - c. Usando herencia puedo resolver toda repetición de lógica de esta solución.
  - d. La solución es poco declarativa.
2. A partir de los problemas descubiertos en el punto anterior, formular una solución superadora que los resuelva, así como también cualquier otro problema detectado.  
Se pide **código y diagrama estático**.