Lógico 1: descripciones, unificación

Ejercicio 1

Una empresa está buscando candidatos para varios de sus sectores.
Con los requerimientos de cada sector, se arma este programa Prolog:

```
puedeAndar(comercioExterior,P):- habla(ingles,P), habla(frances,P), profesional(P).
puedeAndar(comercioExterior,P):- ambicioso(P).
puedeAndar(contaduria,P):- contador(P), honesto(P).
puedeAndar(ventas,P):- ambicioso(P), conExperiencia(P).
puedeAndar(ventas,lucia).

profesional(P):- contador(P).
profesional(P):- abogado(P).
profesional(P):- ingeniero(P).
ambicioso(P):- contador(P), joven(P).
conExperiencia(P):- trabajoEn(P,_).

contador(roque).
joven(roque).
trabajoEn(roque,acme).
habla(roque,frances).
honesto(roque).
ingeniero(ana).
habla(ana,ingles).
habla(ana,frances).
trabajoEn(ana,omni).

habla(lucia,ingles).
habla(lucia,frances).
trabajoEn(lucia,omni).
abogado(cecilia).
ambicioso(cecilia).
habla(cecilia,frances).
```

A partir de esta base

a. indicar qué predicados aparecen en este programa, y para cada uno si está definido por extensión, por comprensión, o en forma mixta.
b. indicar qué predicados aparecen en la primer cláusula, de ellos cuál se está definiendo en esta cláusula, y qué rol juegan los otros.
c. para cada una de estas consultas, pensar si la respuesta de Prolog va a ser "sí" o "no", justificando si es "no". Verificar después con el SWI Prolog.
    - Roque ¿puede andar para comercio exterior?
    - Ana ¿puede andar para comercio exterior?
    - Lucía ¿puede andar para comercio exterior?
    - Roque ¿puede andar para contaduría?
    - Roque ¿puede andar para ventas?
    - Lucía ¿puede andar para ventas?
d. combinando átomos con incógnitas, mostrar ejemplos de consultas que muestren que el predicado puedeAndar es totalmente inversible.
e. lo mismo con el predicado profesional.
f. agregar la información para las búsquedas de estas secciones:
    - proyectos: pueden andar ingenieros con experiencia y también abogados jóvenes.
    - logística: pueden andar profesionales que o bien sean jóvenes o bien hayan trabajado en Omni.
g. agregar postulantes tales que
    - uno pueda andar para proyectos pero no para logística
    - otro pueda andar para ventas pero no para contaduría.

   en este punto, sólo vale agregar cláusulas a la definición de predicados definidos totalmente por extensión en el programa original.

Ejercicio 2

La siguiente es la nómina de personal de una empresa:

- Departamento de ventas: empleada María, cadetes Juan y Roque
- Departamento de compras: empleada Nora, cadete Pedro
- Departamento de administración: empleados Felipe y Hugo, cadeta Ana.

Escribir un programa Prolog que modele a esta empresa, tal que puedan responderse las siguientes consultas :

- ¿quiénes trabajan en el departamento de compras? ¿y en el de ventas?
- dadas dos personas, ¿trabajan en el mismo departamento?
- dadas dos personas **a** y **b**, ¿puede **a** darle órdenes a **b**?

Decimos que **a** puede darle órdenes a **b** si y sólo si trabajan en el mismo departamento y **a** tiene un cargo superior a **b**. Se considera que “empleado” es un cargo superior a “cadete”.

Ejercicio 3

Escribir un programa Prolog que responda consultas acerca de cuáles son los rivales de una determinada selección en un campeonato mundial.
Una selección tiene como rivales todos los otros equipos de su mismo grupo (¡nunca contra sí misma!).
Incluir en el programa la siguiente información:

- El grupo A está formado por Colombia, Camerún, Jamaica e Italia.
- El grupo B está formado por Argentina, Nigeria, Japón y Escocia.

El programa debe ser capaz de responder, p.ej., a la siguiente consulta:
¿cuáles son los rivales de Argentina?
otorgando como respuestas "Nigeria", "Japón" y "Escocia".

Ejercicio 4

Escribir un programa Prolog que ayude a una agencia matrimonial a armar parejas.
Definimos a una pareja como un par (mujer, varón).
La agencia tiene esta información.

- Las mujeres melancólicas son compatibles con los varones serenos.
- Las mujeres decididas son compatibles con los varones reflexivos.
- Las mujeres soñadoras son compatibles con los varones decididos.
- Juan es sereno y decidido.
- María es melancólica.
- Ursula es decidida.
- Juana es soñadora.
- Pedro es reflexivo.
- José es melancólico.

Se pide:
a. Describir la información que maneja la agencia en Prolog de forma tal que se pueda preguntar qué parejas son compatibles.
P.ej.

- ante la pregunta sobre si la pareja (María, Juan) es compatible debe responder que sí.
- ante la pregunta sobre si la pareja (Ursula, Juan) es compatible debe responder que no.

b. Agregar al programa la siguiente información:

- Cualquier pareja formada por un decidido y un melancólico es compatible.

Según esta nueva información, la pareja (Ursula, José) es compatible, mientras que si nos remitimos al punto a. no lo es.

c. Agregar al programa la posibilidad de responder a consultas sobre si una persona es deseable.
Decimos que una persona (varón o mujer) es deseable si es compatible con, por lo menos, dos personas distintas.
Según la información descripta, Juan es deseable mientras que Juana no lo es.

Ejercicio 5

Dado el siguiente programa

```
1. gustaDe(luis,nora).
2. gustaDe(roque,nora).
3. gustaDe(roque,ana).
4. gustaDe(marcos,zulema).
5. gustaDe(X,zulema):- gustaDe(X,ana).
6. gustaDe(juan,X):- gustaDe(roque,X).
7. gustaDe(X,Y):- gustaDe(X,ana).
8. gustaDe(juan,nuria).
9. compiten(X,Y):- gustaDe(X,Z), gustaDe(Y,Z).
10. debeDinero(juan,marcos).
11. debeDinero(juan,roque).
```

se pide

a. determinar con cuáles de las cláusulas unifica cada una de las consultas de acá abajo. En cada caso que una cláusula unifica con una consulta, indicar qué variables se ligan y con qué átomos.

```
a. ?- gustaDe(juan,A).
b. ?- gustaDe(A,zulema).
c. ?- gustaDe(marcos,ana).
d. ?- gustaDe(juan,zulema).
e. ?- gustaDe(A,B).
```

b. Describir en castellano a quiénes relaciona el predicado compiten, con una descripción por comprensión.

Ejercicio 6

Se está organizando una fiesta para una cátedra. Para saber quién puede ir a la fiesta y quién no, se establece quién es alumno de quién. También se tiene información de otras personas, que no son de la facultad.
La información es esta:

- Luisa y Juan son alumnos de Daniel.
- Ana es alumna de Luisa.
- Diana y Nahuel son alumnos de Nico.
- Tamara es alumna de Nahuel.
- Claudio y José son alumnos de Rubén.
- Alvaro es alumno de José y de Luisa.
- Brad, Leo y Johnny son carilindos.
- Luciano y Lautaro son simpáticos.

A la fiesta pueden ir: Nico, Daniel, y los alumnos de alguien que puede ir.
Como las chicas insistieron, también pueden ir los carilindos.

Armar un programa Prolog al que se le pueda preguntar quiénes pueden ir a la fiesta.

Ejercicio 7

a. Escribir un programa Prolog que responda consultas acerca de qué colores atraen a una determinada persona, de acuerdo con la siguiente información:

- A Mabel y a Ana les atrae el rosa.
- A Mara le atraen el celeste y el lila.
- A Juan le atraen todos los colores pastel.
- A todas las mujeres y a Pablo les atrae el azul.
- A los varones mayores y a Mabel les atrae el rojo.
- A todos los porteños (sin importar el sexo) y a Adrián les atrae el amarillo.
- A Ana y a Juan les atrae el naranja.
- Mabel, Mara y Pablo son porteños.
- Ana y Pablo son mayores.
- El rosa, el celeste y el lila son colores pastel.

El programa debe ser capaz de responder, p.ej., a la siguiente consulta: ¿qué colores atraen a Mabel? obteniendo como respuestas "rosa", "azul", "rojo" y "amarillo".

b. Modificar el programa anterior para que responda consultas acerca de con qué colores puede iluminarse una determinada reunión.
Se dice que una reunión puede iluminarse con un color si entre los asistentes a la reunión hay una pareja (dos personas de distinto sexo) a quienes les atrae el color.
Incluir en el programa la siguiente información:

- Mabel, Ana, Adrián y Pablo asistirán a la reunión del viernes.
- Mara, Mabel, Adrián y Juan asistirán a la reunión del sábado.
- Las personas mayores y Juan asistirán a la reunión del domingo.

El programa debe ser capaz de responder, p.ej., a la siguiente consulta
¿con qué colores puede iluminarse la reunión del sábado?
obteniendo como respuestas "rosa", "celeste", "lila" y "amarillo".

Lógico 2: aritmética, negación

Ejercicio 1

Armar un programa Prolog que le indique a una casa de venta de bebidas a qué precio vender sus productos, a partir del precio de costo de cada uno. Las ventas se realizan a comerciantes y a particulares.
Para los comerciantes, el cálculo del precio es así:

- a las bebidas sin alcohol se les recarga un 25%.
- a los vinos nacionales se les recarga un 30% más un peso (p.ej. de 10 se va a 14).
- a los vinos importados se les recarga el máximo entre un 20% y 3 pesos (p.ej. de 10 se va a 13, de 20 se va a 24).
- a los whiskies se les recarga un 50%

Para los particulares, el cálculo del precio es así:

- a las aguas minerales se les recarga un 30%
- a las gaseosas se les recarga un 40%
- a las bebidas alcohólicas nacionales se les recarga un 60%.
- a las bebidas alcohólicas importadas se les recarga un 70%.

Las bebidas sin alcohol son aguas y gaseosas; las bebidas alcohólicas son vinos y whiskies.

Tener en cuenta estos productos:

- Villavicencio, que es un agua mineral, costo $2.
- Glaciar, que es un agua mineral, costo $3.
- CocaCola, gaseosa, costo $4.
- Goliat, gaseosa, costo $1.
- Bianchi, vino nacional, costo $7.
- Trapiche, vino nacional, costo $12.
- Richelieu, vino importado, costo $13.
- Cucagna, vino importado, costo $18.
- Criadores, whisky nacional, costo $20.
- Grants, whisky importado, costo $30.

y estos clientes

- Luisa y Rubén, que son particulares.
- Zoraida y Ramón, que son comerciantes.

Ejercicio 2

En una competencia de habilidades hay tres pruebas: lanzamiento de precisión, pegarle con el martillo para que suba una pesa, y caminar con una escoba en la nariz sin que se caiga.

A cada competidor se le asigna un puntaje por cada prueba, y finalmente un puntaje total.

El puntaje del lanzamiento de precisión se calcula así:

- entre 9 y 11 metros: 10 puntos.
- entre 7 y 9, o entre 11 y 15: 6 puntos.
- menos de 7 o más de 15: 0 puntos.

El puntaje de la prueba de fuerza (la del martillo), es así:
hasta 5 metros: 3 puntos.
de 5 a 10 metros: 6 puntos.
más de 10 metros: 9 puntos + 1 punto adicional por cada 5 metros extra, p.ej. 14 metros => 9.8 puntos, 19 metros => 10.8 puntos.

El puntaje de la prueba de equilibro (la de la escoba) es 1 punto cada 3 metros recorridos.

El puntaje total se calcula así:

- si ninguno de los tres puntajes llega a 5, entonces es 0
- en caso contrario, es la suma de los 3 puntajes.

Escribir un programa Prolog en el que se represente lo que obtuvo cada competidor en cada prueba, y al que se le pueda preguntar el puntaje total de un competidor.

Ejercicio 3

Armar un programa Prolog que resuelva el siguiente problema lógico:

- Un asesino siempre odia a su víctima y nunca es más rico que ella. El asesino de la tía Agatha, además, vive en la mansión Dreadbury.
- Tía Agatha, el carnicero y Charles son las únicas personas que viven en la mansión Dreadbury.
- Charles odia a todas las personas de la mansión que no son odiadas por la tía Agatha.
- Agatha odia a todos los que viven en la mansión, excepto al carnicero.
- Quien no es odiado por el carnicero y vive en la mansión, es más rico que tía Agatha
- El carnicero odia a las mismas personas que odia tía Agatha.

Al programa le tengo que poder preguntar quién es el asesino de la tía Agatha, y tiene que brindar una sola respuesta.

Lógico 3: listas y functores

Ejercicio 1

En una competencia de saltos, cada competidor puede hacer hasta 5 saltos; a cada salto se le asigna un puntaje de 0 a 10. Se define el predicado puntajes que relaciona cada competidor con los puntajes de sus saltos, p.ej.

```
puntajes(hernan,[3,5,8,6,9]).
puntajes(julio,[9,7,3,9,10,2]).
puntajes(ruben,[3,5,3,8,3]).
puntajes(roque,[7,10,10]).
```

Se pide armar un programa Prolog que a partir de esta información permita consultar:

a. qué puntaje obtuvo un competidor en un salto, p.ej. qué puntaje obtuvo Hernán en el salto 4 (respuesta: 6).

b. si un competidor está descalificado o no. Un competidor está descalificado si hizo más de 5 saltos. En el ejemplo, Julio está descalificado.

c. si un competidor clasifica a la final. Un competidor clasifica a la final si la suma de sus puntajes es mayor o igual a 28, o si tiene dos saltos de 8 o más puntos.

Ayuda: investigar predicado **`nth1/3`** y **`nth0/3`** en el prolog.

Ejercicio 2

Se registra el ingreso de distintas personas en cada mes del año, mediante el predicado ingreso/3, p.ej.

```
ingreso(roque,enero,2000).
ingreso(roque,febrero,3500).
ingreso(roque,marzo,1200).
ingreso(luisa,enero,2500).
ingreso(luisa,febrero,850).
```

y se tiene la relación padre/2 que indica padre(Hijo, Padre), p.ej. para decir que Luisa es la madre de Roque se agrega esta cláusula

```
padre(roque,luisa).
```

(léase "uno de los padres de Roque es Luisa").

Definir estos predicados:

- **buenPasar**/1, que se verifica para una persona si se cumple alguna de estas condiciones:
  - en enero ganó más de 2200,
  - en algún mes ganó más de 3000,
  - ganó más de 2500 en dos meses distintos (esto se puede hacer sin listas, usar el \= para decir que los meses son distintos).
- **mesFilial**/2, que relaciona una persona X con un mes si en ese mes hay algún hijo de X que haya ganado más X.
  P.ej. con la información del ejemplo, la respuesta a la consulta
  ```
  ?- mesFilial(luisa,febrero).
  ```
  debería ser Yes.
  Verificar en el SWI que este predicado es totalmente inversible.
- **ingresoTotal**/2 que relaciona una persona con su ingreso total de una persona en el año; entendiendo que toda la información de la base de conocimiento corresponde al mismo año.
- **ingresoFamiliar**/2, que relaciona una persona con su ingreso familiar en el año; el ingreso familiar de una persona es su ingreso total más la suma de los ingresos totales de sus hijos.

Ejercicio 3 – subtes (orden en las listas)

En este ejercicio viene bien usar el predicado nth1/3, que relaciona un número, una lista, y el elemento de la lista en la posición indicada por el número (empezando a contar de 1).
P.ej. prueben estas consultas

- `?- nth1(X,[a,b,c,d,e],d).`
- `?- nth1(4,[a,b,c,d,e],X).`
- `?- nth1(Orden,[a,b,c,d,e],Elem).`
- `?- nth1(X,[a,b,c,d,e],j).`
- `?- nth1(22,[a,b,c,d,e],X).`

Tenemos un modelo de la red de subtes, por medio de un predicado linea que relaciona el nombre de la linea con la lista de sus estaciones, en orden. P.ej. (reduciendo las lineas)

```
linea(a,[plazaMayo,peru,lima,congreso,miserere,rioJaneiro,primeraJunta,nazca]).
linea(b,[alem,pellegrini,callao,gardel,medrano,malabia,lacroze,losIncas,urquiza]).
linea(c,[retiro,diagNorte,avMayo,independenciaC,plazaC]).
linea(d,[catedral,nueveJulio,medicina,plazaItalia,carranza,congresoTucuman]).
linea(e,[bolivar,independenciaE,pichincha,jujuy,boedo,varela,virreyes]).
linea(h,[once,venezuela,humberto1ro,inclan,caseros]).
combinacion([lima,avMayo]).
combinacion([once,miserere]).
combinacion([pellegrini,diagNorte,nueveJulio]).
combinacion([independenciaC,independenciaE]).
```

No hay dos estaciones con el mismo nombre.

Se pide armar un programa Prolog que a partir de esta información permita consultar:
a. en qué linea está una estación, predicado `estaEn/2`.
b. dadas dos estaciones de la misma línea, cuántas estaciones hay entre ellas, p.ej. entre Perú y Primera Junta hay 5 estaciones. Predicado `distancia/3` (relaciona las dos estaciones y la distancia).
c. dadas dos estaciones de distintas líneas, si están a la misma altura (o sea, las dos terceras, las dos quintas, etc.), p.ej. Independencia C y Jujuy que están las dos cuartas. Predicado `mismaAltura/2`.
d. dadas dos estaciones, si puedo llegar fácil de una a la otra, esto es, si están en la misma línea, o bien puedo llegar con una sola combinación. Predicado `viajeFacil/2`.

Ejercicio 4 – viajes

Una agencia de viajes lleva un registro con todos los vuelos que maneja de la siguiente manera:

```
vuelo(Codigo de vuelo, capacidad en toneladas, [lista de destinos] ).
```

Esta lista de destinos está compuesta de la siguiente manera:

```
escala(ciudad, tiempo de espera)
tramo(tiempo en vuelo)
```

Siempre se comienza de una ciudad (escala) y se termina en otra (no puede terminar en el aire al vuelo), con tiempo de vuelo entre medio de las ciudades. Considerar que los viajes son de ida y de vuelta por la misma ruta.

```
vuelo(ARG845, 30, [escala(rosario,0), tramo(2), escala(buenosAires,0)]).

vuelo(MH101, 95, [escala(kualaLumpur,0), tramo(9), escala(capeTown,2),
tramo(15), escala(buenosAires,0)]).

vuelo(DLH470, 60, [escala(berlin,0), tramo(9), escala(washington,2), tramo(2),
escala(nuevaYork,0)]).

vuelo(AAL1803, 250, [escala(nuevaYork,0), tramo(1), escala(washington,2),
tramo(3), escala(ottawa,3), tramo(15), escala(londres,4), tramo(1),
escala(paris,0)]).

vuelo(BLE849, 175, [escala(paris,0), tramo(2), escala(berlin,1), tramo(3),
escala(kiev,2), tramo(2), escala(moscu,4), tramo(5), escala(seul,2), tramo(3),
escala(tokyo,0)]).

vuelo(NPO556, 150, [escala(kiev,0), tramo(1), escala(moscu,3), tramo(5),
escala(nuevaDelhi,6), tramo(2), escala(hongKong,4), tramo(2),
escala(shanghai,5), tramo(3), escala(tokyo,0)]).

vuelo(DSM3450, 75, [escala(santiagoDeChile,0), tramo(1), escala(buenosAires,2),
tramo(7), escala(washington,4), tramo(15), escala(berlin,3), tramo(15),
escala(tokyo,0)]).
```

Definir los siguientes predicados; en todos vamos a identificar cada vuelo por su código.

- **tiempoTotalVuelo/2** : Relaciona un vuelo con el tiempo que lleva en total, contando las esperas en las escalas (y eventualmente en el origen y/o destino) más el tiempo de vuelo.
- **escalaAburrida/2** : Relaciona un vuelo con cada una de sus escalas aburridas; una escala es aburrida si hay que esperar mas de 3 horas.
- **ciudadesAburridas/2** : Relaciona un vuelo con la lista de ciudades de sus escalas aburridas.
- **vueloLargo/1**: Si un vuelo pasa 10 o más horas en el aire, entonces es un vuelo largo. OJO que dice "en el aire", en este punto no hay que contar las esperas en tierra.
- **conectados/2**: Relaciona 2 vuelos si tienen al menos una ciudad en común.
- **bandaDeTres/3**: Relaciona 3 vuelos si están conectados, el primero con el segundo, y el segundo con el tercero.
- **distanciaEnEscalas/3**: Relaciona dos ciudades que son escalas del mismo vuelo y la cantidad de escalas entre las mismas; si no hay escalas intermedias la distancia es 1. P.ej. París y Berlín están a distancia 1 (por el vuelo BLE849), Berlín y Seúl están a distancia 3 (por el mismo vuelo). No importa de qué vuelo, lo que tiene que pasar es que haya algún vuelo que tenga como escalas a ambas ciudades. Consejo: usar nth1.
- **vueloLento/1**: Un vuelo es lento si no es largo, y además cada escala es aburrida.

Lógico 4: functores, generación, listas con orden

Ejercicio 1

En una empresa multinacional se da servicio de outsorcing a los servidores de varias empresas. De cada servidor se conoce su ubicación en una fila de servidores del Centro de Cómputos, y qué criticidad tiene asociada en el contrato de outsorcing. La criticidad (del 1 al 4) define qué tan rápido se tiene que dar solución a los problemas que se presentan en el server.
En el Centro de Monitoreo de la empresa se reciben las alertas de los problemas que acontence en el Centro de Cómputos. Los eventos que se informa son:

- cortes de luz que afectan toda una fila de servidores
- rebooteos de servers
- "cuelgues" de aplicaciones en un server.

Escribir las cláusulas que hagan falta para definir los predicados requiereAtencionNormal/2 y requiereAtencionInmediata/2, sabiendo que

- ante un corte de luz, todos los servidores de la fila requieren atención inmediata
- ante un rebooteo de server, el server afectado y los otros servidores que sean clientes de ese server requieren atención normal.
- ante un cuelgue de aplicación, el servidor afectado requiere atención inmediata si es crítico (criticidad 1 y 2), o atención normal si el servidor no es crítico (criticidad 3 y 4). Ningún otro servidor requiere atención.

Para los dos predicados, el primer argumento es el servidor del cual se quiere saber si requiere o no atención, y el segundo es un evento; usar functores para distinguir entre distintos eventos.
Lograr para ambos predicados la inversibilidad necesaria para que pueda consultarse qué servidores requieren atención dado un evento.

Los datos de los servidores se pueden modelar así:

```
servidor(PS1, fila1, criticidad1).
servidor(PS2, fila1, criticidad2).
servidor(WAS1, fila2, criticidad1).
servidor(WAS1_2, fila2, criticidad4).
servidor(FS_X48, fila2, criticidad1).
esCliente(PS1,FS_X48).   % acá dice que el servidor PS1 es cliente de FS_X48
esCliente(WAS1,FS_X48).  % acá dice que el servidor WAS1 es cliente de FS_X48
% etc...
```

En este ejemplo:

- ante un corte de luz en la fila1, los servidores PS1 y PS2 requieren atención inmediata.
- ante un rebooteo de FS_X48, tanto ese server como PS1 y WAS1 requieren atención normal.
- ante un cuelgue en PS2, ese server requiere atención inmediata y nadie más.
- ante un cuelgue en WAS1_2, ese server requiere atención normal y nadie más.

Ejercicio 2

La agencia matrimonial del ejercicio 3 de la práctica 1 arma grupos de mujeres y varones en igual cantidad. Para cada persona del grupo se determina, con los métodos científicos que usa la agencia, el orden de preferencia de las personas del otro sexo.

Damos un ejemplo de grupo:

- mujeres: Ana, Nora y Marta
- varones: Luis, Juan y Pedro
- preferencias de Ana : Luis / Juan / Pedro
- preferencias de Nora : Juan / Pedro / Luis
- preferencias de Marta : Pedro / Luis / Juan
- preferencias de Luis : Ana / Nora / Marta
- preferencias de Juan : Marta / Ana / Nora
- preferencias de Pedro : Nora / Marta / Ana

la agencia quiere armar parejas que incluyan a todas las personas de un grupo, y tal que el conjunto de parejas armadas sea estables.

Un conjunto de parejas es estable si todas sus parejas son estables en el contexto del conjunto.
Una pareja es inestable en un contexto si alguno de sus dos integrantes quiere dejar al otro. Una persona A quiere dejar a otra B si existe una tercera C tal que:

- A prefiere a C antes que a B, y
- C prefiere a A antes que a su pareja (la que tiene asignada en el contexto).

P.ej. el conjunto de Ana-Pedro, Nora-Luis, y Marta-Juan es inestable porque Ana quiere dejar a Pedro, dado que Ana prefiere a Luis antes que a Pedro, y Luis prefiere a Ana antes que a Nora (la pareja de Luis en el conjunto).

Se pide:

a. Definir el predicado parejas/3 tal que arma los conjuntos posibles de parejas a partir de un conjunto de mujeres y otro de varones. Hay que definir cómo representar una pareja.

b. Definir el predicado insatisfecho/2, que se verifica para una persona y un conjunto de parejas si la persona quiere dejar a su pareja asignada en el contexto del conjunto.
P.ej. Ana sí está insatisfecha respecto del conjunto de parejas del ejemplo.

c. Definir el predicado estable/1, que indique si un conjunto de parejas es estable o no. Hacerlo de forma tal de garantizar que sea inversible.
Se supone que las mujeres, los varones y los gustos de cada quién están en hechos, que (obviamente) hay que modelar.

Ejercicio 3

Se organiza un juego que consiste en ir buscando distintos objetos por el mundo. Cada participante está en un determinado nivel, cada nivel implica ciertas tareas, cada tarea consiste en buscar un objeto en una ciudad.

Representamos las tareas como functores buscar(Cosa,Ciudad), y definimos el predicado tarea/2 de esta forma:

```
tarea(basico,buscar(libro,jartum)).
tarea(basico,buscar(arbol,patras)).
tarea(basico,buscar(roca,telaviv)).
tarea(intermedio,buscar(arbol,sofia)).
tarea(intermedio,buscar(arbol,bucarest)).
tarea(avanzado,buscar(perro,bari)).
tarea(avanzado,buscar(flor,belgrado)).
```

o sea, si estoy en el nivel básico, mis tareas posibles son buscar un libro en Jartum, un árbol en Patras o una roca en Tel Aviv.

Para definir en qué nivel está cada participante se define el predicado nivelActual/2, de esta forma:

```
nivelActual(pepe,basico).
nivelActual(lucy,intermedio).
nivelActual(juancho,avanzado).
```

También vamos a necesitar saber qué idioma se habla en cada ciudad, qué idiomas habla cada persona, y el capital actual de cada persona. Esto lo representamos con los predicados idioma/2, habla/2 y capital/2:

```
idioma(alejandria,arabe).
idioma(jartum,arabe).
idioma(patras,griego).
idioma(telaviv,hebreo).
idioma(sofia,bulgaro).
idioma(bari,italiano).
idioma(bucarest,rumano).
idioma(belgrado,serbio).

habla(pepe,bulgaro).
habla(pepe,griego).
habla(pepe,italiano).
habla(juancho,arabe).
habla(juancho,griego).
habla(juancho,hebreo).
habla(lucy,griego).

capital(pepe,1200).
capital(lucy,3000).
capital(juancho,500).
```

Definir los siguientes predicados:
a. destinoPosible/2 e **idiomaUtil/2**.
destinoPosible/2 relaciona personas con ciudades; una ciudad es destino posible para un nivel si alguna tarea que tiene que hacer la persona (dado su nivel) se lleva a cabo en la ciudad. P.ej. los destinos posibles para Pepe son: Jartum, Patras y Tel Aviv.
idiomaUtil/2 relaciona niveles con idiomas: un idioma es útil para un nivel si en alguno de los destinos posibles para el nivel se habla el idioma. P.ej. los idiomas útiles para Pepe son: árabe, griego y hebreo.

b. excelenteCompaniero/2, que relaciona dos participantes. P2 es un excelente compañero para P1 si habla los idiomas de todos los destinos posibles del nivel donde está P1.
P.ej. Juancho es un excelente compañero para Pepe, porque habla todos los idiomas de los destinos posibles para el nivel de Pepe.
Asegurar que el predicado sea inversible para los dos parámetros.

c. interesante/1: un nivel es interesante si se cumple alguna de estas condiciones

- todas las cosas posibles para buscar en ese nivel están vivas (las cosas vivas en el ejemplo son: árbol, perro y flor)
- en alguno de los destinos posibles para el nivel se habla italiano.
- la suma del capital de los participantes de ese nivel es mayor a 10000

Asegurar que el predicado sea inversible.

d. complicado/1: un participante está complicado si: no habla ninguno de los idiomas de los destinos posibles para su nivel actual; está en un nivel distinto de básico y su capital es menor a 1500, o está en el nivel básico y su capital es menor a 500.

e. homogeneo/1: un nivel es homogéneo si en todas las opciones la cosa a buscar es la misma. En el ejemplo, el nivel intermedio es homogéneo, porque en las dos opciones el objeto a buscar es un árbol.
Asegurar que el predicado sea inversible.
Ayuda: hay que saber de alguna forma si una lista tiene un único elemento, p.ej. la lista [3,3,3,3] tiene un único elemento (el 3) mientras que la lista [2,1,2,4] no. Tal vez convenga definir un predicado aparte para este problema.

f. poliglota/1: una persona es políglota si habla al menos tres idiomas.

En general: es válido agregar los predicados necesarios para poder garantizar inversibilidad o auxiliares para resolver cada ítem, y usar en un ítem los predicados definidos para resolver ítems anteriores.

Ejercicio 4

Para un proyecto de desarrollo de software se tiene una base de conocimientos con los siguientes hechos:

tarea/3, que relaciona una tarea con su duración y un rol para realizarla.

```
tarea(login, 80, programador).
tarea(cacheDistribuida, 120, arquitecto).
tarea(pruebasPerformance, 100, tester).
tarea(tuning, 30, arquitecto).
```

predece/2, que relaciona dos tareas A y B indicando que la tarea B no puede comenzar hasta que la A no haya terminado.

```
precede(cacheDistribuida, pruebasPerformance).
precede(pruebasPerformance, tuning).
```

realizada/1, que indica las tareas que ya están terminadas.

```
realizada(login).
```

Se pide:

1. Definir el predicado anterior/2 que relaciona dos tareas A y B si estas deben realizarse en un orden específico, ya sea por una dependencia directa o indirecta. Por ejemplo:
?- anterior(T, tuning).
T = pruebasPerformance; (-> dependencia directa, porque pruebasPerformance precede a tuning)
T = cacheDistribuida; (-> dependencia indirecta, a través de pruebasPerformance)

2. Definir los predicados simple/1 y riesgo/1 de forma que:

- Las tareas que realizan los programadores y testers son consideradas simples. También se consideran simples todas las tareas que toman menos de una semana (40 horas).
- Consideramos como riesgos a todas las tareas de más de 40 horas que aún no han sido realizadas.

Por ejemplo:
```
?- simple(T).
T = login; (-> porque la realiza un programador).
T = pruebasPerformance; (-> porque la realiza un tester).
T = tuning; (-> porque tarda menos de cuarenta horas).

?- riesgo(T).
T = cacheDistribuida.
T = pruebasPerformance.
```

3. Definir el predicado meFaltanPara/2 que relaciona a una tarea con la lista de todas las tareas que deben ser terminadas antes de poder comenzar con T y no han sido realizadas.
?- meFaltanPara(tuning, Faltantes).
Faltantes = [pruebasPerformance, cacheDistribuida].

4. Usando el predicado forall definir el predicado puedoHacer/1 que indica si una tarea puede ser realizada. (Una tarea puede ser realizada cuando sus precedentes han sido realizadas). El predicado se debe poder usar de la siguiente forma:

```
?- puedoHacer(T).
T = cacheDistribuida;
```
(es la única tarea que puedo hacer en este momento, ya que login está realizada, para pruebasPerformance me falta cacheDistribuida y para tuning me falta pruebasPerformance).

5. Para poder designar horas de trabajo de varios roles distintos a una misma tarea se realiza la siguiente modificación en el hecho tarea/2:
```
tarea(login, [trabajo (80, programador)]).
tarea(cacheDistribuida, [trabajo(40, arquitecto), trabajo(80, diseniador)]).
tarea(pruebasPerformance, [trabajo(100, tester), trabajo(20, analista)]).
tarea(tuning, [trabajo(40, arquitecto), trabajo(20, tester)]).
```

De esta forma por ejemplo para la tarea cacheDistribuida ahora se necesitan 40 horas de un arquitecto y 80 horas de un diseñador.

Se pide definir el predicado muchoTesting/1 que indique si una tarea tiene más de 40 horas de testing, por ejemplo:

```
?- muchoTesting(pruebasPerformance).
Yes.

?- muchoTesting(tuning).
No. (-> menos de 40 horas de testing).

?- muchoTesting(login).
No. (-> no tiene horas de testing planificadas).
```

Ejercicio 5 - dominó

El juego usual del dominó tiene 28 fichas diferentes. Cada ficha es rectangular y tiene grabado en cada extremo un número de puntos entre 0 y 6. P.ej., estas son algunas fichas:

```
| 0 | 1 |
| 1 | 2 |
| 3 | 6 |
```

todas las fichas son distintas entre sí.

Siguendo las reglas del juego, las fichas se colocan formando una cadena de tal manera que cada par de fichas consecutivas tienen iguales números correspondientes a los dos extremos que se tocan. P.ej. esto podría ser el estado del juego en un momento

```
| 0 | 1 || 1 | 4 || 4 | 2 || 2 | 2 || 2 | 5 |
```

Cada jugador posee un conjunto de piezas, las cuales utiliza por turno para ir agregando a la cadena de piezas del juego; puede agregar su ficha en cualquiera de los dos extremos. P.ej. en el ejemplo anterior las siguientes fichas se pueden agregar

```
| 0 | 4 |  (por la izquierda)
| 0 | 6 |  (por la izquierda)
| 5 | 1 |  (por la derecha)
| 5 | 0 |  (por cualquiera de los dos lados)
```

Cuando un jugador no puede agregar ninguna pieza, este debe ceder su turno.

Lo que sigue es parte de la base de conocimiento

```
tieneFicha(carlos,ficha(0,4)).
tieneFicha(carlos,ficha(0,6)).
tieneFicha(carlos,ficha(5,1)).
tieneFicha(german,ficha(5,0)).
tieneFicha(miguel,ficha(3,2)).
tieneFicha(miguel,ficha(3,3)).
tieneFicha(juan,ficha(1,6)).

estado([ficha(0,1),ficha(1,4),ficha(4,2),ficha(2,2),ficha(2,5)]).
```

(en algún momento les va a convenir agregar algunos hechos más).

Escribir las cláusulas que haga falta para resolver el predicado cedeTurno/1, que se verifica para los jugadores que deben ceder su turno. Lograr que este predicado sea inversible.
En el ejemplo, si consulto

```
?- cedeTurno(X)
```

me debe dar las respuestas

```
X = miguel
X = juan
```

Ayuda: vale definir los predicados auxiliares que hagan falta.

Ejercicio 6 - Liga de Fútbol

Hay que desarrollar un programa en Prolog con el que podamos hacer consultas sobre los partidos desarrollados en una liga de fútbol.

Los hechos están dados por:

%fecha(NroFecha, partido(EquipoLocal,GolesLocal,EquipoVisitante,GolesVisitante)).

Por ejemplo

```
fecha(1,partido(mandiyu,0,losAndes,2)).
fecha(1,partido(yupanqui,2,claypole,1)).
fecha(1,partido(jjurquiza,1,cambaceres,4)).
fecha(2,partido(yupanqui,4,jjurquiza,2)).
fecha(2,partido(losAndes,3,claypole,1)).
fecha(2,partido(cambaceres,2,mandiyu,2)).
fecha(3,partido(jjurquiza,3,losAndes,1)).
fecha(3,partido(mandiyu,2,claypole,2)).
fecha(3,partido(cambaceres,3,yupanqui,2)).
```

Tengan en cuenta que no hay partidos que se hagan ida y vuelta en esta liga, o sea, dados dos equipos hay cargado a lo sumo un partido en el cual se enfrentan.
Suponer que la carga de los hechos es correcta, no hacer validaciones.

Desarrollar los siguientes predicados:

a) rival/3, que nos dice quién fue el rival de un equipo en una fecha. Algunas consultas de ejemplo con sus respuestas

```
?- rival(losAndes,2,claypole).
Yes

?- rival(claypole,2,losAndes).
Yes

?- rival(claypole,2,mandiyu).
No

?- rival(claypole,F,mandiyu).
F = 3;
No
```

b) golesLocalFecha/2, que relaciona un número de fecha con el total de goles que hicieron los equipos locales en sus partidos de dicha fecha.

c) golesFecha/2, que relaciona un número de fecha con el total de goles que se hicieron en los partidos de una fecha.

d) jugaron/2 que relaciona dos equipos y nos responde si jugaron entre sí en alguna fecha de la liga.

e) faltaQueJueguen/2 que relaciona a dos equipos si aún no jugaron entre ellos, p.ej. Mandiyú y J.J.Urquiza, en cualquier orden.

f) gano/2 y **perdio/2** que relaciona dos equipos de acuerdo al resultado del partido jugado entre ambos. Un equipo ganó cuando la cantidad de goles que hizo es mayor que la de su rival.

g) empataron/2 que utilice gano/2 y perdio/2

h) Uno que permita conocer los **puntos** que hizo un equipo (se pueden hacer dos predicados auxiliares **partidosGanados/2 partidosEmpatados/2**).
Cada equipo recibirá 3 puntos por partido ganado, 1 por partido empatado y 0 por partido perdido.

i) equipoImprevisible/1 se dice que un equipo es imprevisible si: le ganó a un equipo A, perdió con otro equipo B, pero B a su vez le ganó a A. Este predicado debe ser inversible, vale agregar hechos.

j) Un predicado **goles/2** que relacione un equipo con la cantidad de goles que convirtió en toda la liga

k) Uno que permita saber si un equipo es **invicto** (si gano todos sus partidos) y otro que permita saber si un equipo es **"lucer"** (si perdió todos sus partidos)

l) masCapoQue/2. Mi equipo es más capo que el tuyo si el mío le ganó al tuyo, y también les ganó a todos a los que le ganó el tuyo.

m) campeon/1. Un equipo es campeón si es el que más puntos tiene y el que convirtió más goles

n) Por último uno con el que podamos conocer la **tabla** de posiciones.
La tabla debe ir de mayor a menor cantidad de puntos relacionando cada equipo con sus puntos asociados.

AYUDA: el predicado **setof/3** tiene una funcionalidad igual que la de **findall/3** con la salvedad que el tercer argumento es una lista sin elementos repetidos

Ejercicio 7 – TEG

En vista de que gran parte de las personas (chicos y grandes) abandonan los juegos clasicos por modernos juegos de PC, la jugueteria de Lanus Quindimil SRL, decide llevar su negocio al terreno digital para poder competir con las nuevas formas de espoarcimiento.
Asi es como se comunican con nosotros solicitandonos que realicemos el desarrollo de ciertas consultas para un tablero de TEG, que se realizan cada cierto tiempo.
Cada vez que se realizan las consultas, vamos a contar con ciertos hechos como los que ejemplificamos a continuación.

```
/* distintos paises */
paisContinente(americaDelSur, argentina).
paisContinente(americaDelSur, bolivia).
paisContinente(americaDelSur, brasil).
paisContinente(americaDelSur, chile).
paisContinente(americaDelSur, ecuador).

paisContinente(europa, alemania).
paisContinente(europa, españa).
paisContinente(europa, francia).
paisContinente(europa, inglaterra).

paisContinente(asia, aral).
paisContinente(asia, china).
paisContinente(asia, gobi).
paisContinente(asia, india).
paisContinente(asia, iran).

/*países importantes*/
paisImportante(argentina).
paisImportante(kamchatka).
paisImportante(alemania).

/*países limítrofes*/
limitrofes([argentina,brasil]).
limitrofes([bolivia,brasil]).
limitrofes([bolivia,argentina]).
limitrofes([argentina,chile]).

limitrofes([espana,francia]).
limitrofes([alemania,francia]).

limitrofes([nepal,india]).
limitrofes([china,india]).
limitrofes([nepal,china]).
limitrofes([afganistan,china]).
limitrofes([iran,afganistan]).

/*distribucion en el tablero */
ocupa(argentina, azul, 4).
ocupa(bolivia, rojo, 1).
ocupa(brasil, verde, 4).
ocupa(chile, negro, 3).
ocupa(ecuador, rojo, 2).

ocupa(alemania, azul, 3).
ocupa(españa, azul, 1).
ocupa(francia, azul, 1).
ocupa(inglaterra, azul, 2).
```

```
ocupa(aral, negro, 2).
ocupa(china, verde, 1).
ocupa(gobi, verde, 2).
ocupa(india, rojo, 3).
ocupa(iran, verde, 1).

/*continentes*/
continente(americaDelSur).
continente(europa).
continente(asia).

/*objetivos*/
objetivo(rojo, ocuparContinente(asia)).
objetivo(azul, ocuparPaises([argentina, bolivia, francia, inglaterra, china])).
objetivo(verde, destruirJugador(rojo)).
objetivo(negro, ocuparContinente(europa)).
```

Se solicita construir un programa que permita la resolución de las siquientes consultas.

Todos los predicados deben ser inversibles, salvo aclaración explícita en contrario.

1) **estaEnContinente/2:** Relaciona un jugador y un continente si el jugador ocupa al menos un país en el continente.
2) **cantidadPaises/2:** Relaciona a un jugador con la cantidad de paises que ocupa.
3) **ocupaContinente/2:** Relaciona un jugador y un continente si el jugador ocupa totalmente al continente.
4) **leFaltaMucho/2:** Relaciona a un jugador y un continente si al jugador le falta ocupar más de 2 países de dicho continente.
5) **sonLimitrofes/2:** Relaciona 2 países si son limítrofes.
6) **esGroso/1:** Un jugador es groso si cumple algunas de estas condiciones:
   - ocupa todos los paises importantes,
   - ocupa mas de 10 paises
   - o tiene mas de 50 ejercitos.
7) **estaEnElHorno/1:** un pais esta en el horno si todos sus paises limitrofes estan ocupados por el mismo jugador que no es el mismo que ocupa ese pais.
8) **esCaotico/1:** un continente es caotico si hay mas de tres jugadores en el.
9) **capoCannoniere/1:** es el jugador que tiene ocupado mas paises.
10) **ganadooor/1:** un jugador es ganador si logro su objetivo

Ejercicio 8 – Subtes

Se parte de una descripción de la red de subtes con esta pinta:

```
linea(a,[plazaMayo,peru,lima,congreso,once,castroBarros,primeraJunta]).
linea(c,[retiro,diagNorte,avMayo,moreno,constitucion]).
linea(d,[catedral,nueveJulio,medicina,pzaItalia,carranza]).
linea(h,[rivadavia,belgrano,sanJuan,parquePatricios]).
combinacion(peru,catedral).
combinacion(lima,avMayo).
combinacion(diagNorte,9julio).
combinacion(once,rivadavia).
```

1. Usando estos hechos, definir los siguientes predicados
   a. `pasaPor(Linea,Estacion)`
   b. `combinan(Linea1,Linea2)`
   c. `cantidadEstacionesEntre(Estacion1,Estacion2,Cant)`
      p.ej. entre Lima y Primera Junta hay 4 estaciones.
      Entre estaciones de distintas líneas no tiene que dar ninguna respuesta.
      Ayuda: predicado nth1/3 que viene con SWI Prolog.

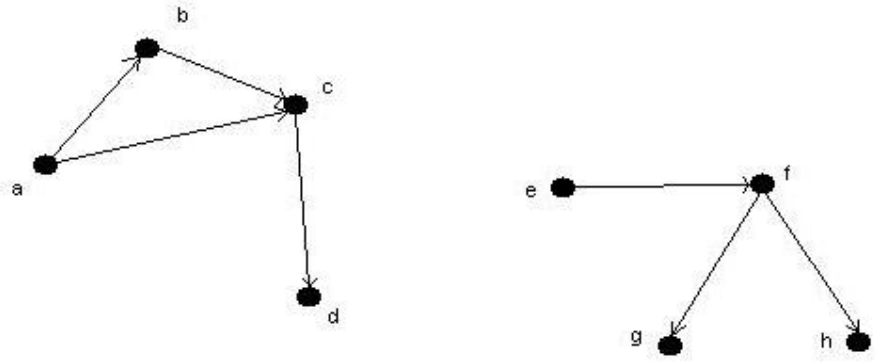
d. `esCabecera(Estacion)`
   P.ej. Retiro y Constitucion son cabeceras, Moreno no.
e. `esLineaUniversal(Linea)`
   Una línea se dice "universal" si combina con todas las otras.

2. Ahora definir estos …
   a. `cabeceraMasCercana(Estacion,Cabecera)` , relaciona cada estación con la cabecera que tiene más cerca (según la diferencia de estaciones).
   b. `hayViajeDirecto(Estacion1,Estacion2)` , si puedo viajar sin necesidad de hacer combinación
   c. `hayViajeConCombinacion(Estacion1,Estacion2)` , si puedo viajar haciendo una combinación
   d. `hayViaje(Estacion1,Estacion2)` , si puedo viajar con hasta una combinación
   e. `longitudViaje(Estacion1,Estacion2,Cant)` , cuántas estaciones, si es combinación sumando los dos tramos

Lógico 5: Recursividad, con y sin listas

Ejercicio 1

Escribir un programa Prolog que describa un grafo dirigido, de forma tal que pueda preguntar si hay camino desde un nodo hasta otro.
P.ej. en este grafo



sí hay camino desde *a* hasta *d*, pero no hay camino ni desde *d* hasta *a* ni desde *a* hasta *f*.
No tener en cuenta ciclos.

Verificar que el predicado definido para la consulta es inversible para todos sus argumentos.

Ejercicio 2

Escribir un programa Prolog al que le pueda pedir el valor de la serie de Fibonacci para cualquier natural.
La serie de Fibonacci se define así:

- fib(1) = 1.
- fib(2) = 1.
- fib(n) = fib(n-1) + fib(n-2), si n > 2.

Ejercicio 3

Armar un programa para una central telefónica que permita realizar derivaciones de llamadas. La central funciona de la siguiente manera:

En la base de conocimientos tenemos registrados a todos los empleados y sus jefes, y el área al que pertencen los jefes. Por ejemplo:

```
empleado(juan).
empleado(marcelo).
empleado(adriana).
empleado(francisco).
empleado(alberto).
empleado(cristian).
empleado(mariana).
jefe(marcelo, juan).
jefe(marcelo, adriana).
jefe(mariana, cristian).
area(marcelo, administracion)
area(mariana, ventas).
```

En la base se registran también los internos de las personas. Por ejemplo:

```
interno(marcelo, 2244).
interno(adriana, 2245).
interno(francisco, 4441).
interno(alberto, 4442).
interno(mariana, 1212).
```

También se registran los gerentes y sus asistentes.

```
gerente(alberto, administracion).
asistente(francisco, alberto).
```

Programar las siguientes reglas:

1- internoDe(Persona, Interno)

Relaciona una persona con su interno. Si la persona no tiene interno, relaciona una persona con el interno de su jefe.

2- quienAtiende(Interno, Persona)

Dice quien atiende un interno. Si llaman al interno de un jefe, hace que atienda cualquiera de sus empleados. Si llaman al interno de un gerente, hace que atienda su asistente. En cualquier otro caso, atiende la persona dueña del interno.

3- dependeDe(Persona, Persona)

Dice si la primer persona depende de la segunda persona. Una persona "A" depende de una persona "B", si "B" es jefe de "A", o si "B" es gerente del área donde trabaja "A".

4- puedeTransferir(interno, interno)

Dos internos se pueden transferir si pertenecen a empleados que dependen de la misma persona. Para resolverla utilizar la regla dependeDe.

5- pertenecen ([internos], [internos]).
"Quita" de una lista de números telefónicos, los que NO pertenecen a internos de la empresa.

6- personas([Personas], [internos])
"Devuelve" de una lista de personas, sus internos. Para resolverla utilizar la regla internoDe.

Ejercicio 4, Recursividad con listas

Definir los siguientes predicados:

a. suma(L,N): relaciona una lista de números L con la suma. La suma de una lista vacía se define como 0.

b. encolar(E,L,LconE): relaciona un elemento E, una lista L, y la lista que resulta de agregar E al final de L.
P.ej.

- ?- encolar([1,3,5],7,[1,3,5,7]). devuelve Yes
- ?- encolar([1,3,5],7,[1,3,7]). devuelve No
- ?- encolar([1,3,5],7,[1,3,5,8]). devuelve No

c. maximo(L,N): relaciona una lista de números L con su elemento más grande. La lista vacía no tiene máximo.

d. elementoEn(L,P,E): relaciona una lista L, un número P y el elemento que está en la posición P de L.
El primer elemento es el 1.
P.ej.

- ?- elementoEn([2,5,8,11,14],3,8). devuelve Yes
- ?- elementoEn([2,5,8,11,14],3,X). devuelve 8
- ?- elementoEn([2,5,8,11,14],7,X). devuelve No (porque la lista tiene menos de 7 elementos)

Lógico 6: Integración

Ejercicio 1

El registro civil cuenta con información sobre nacimientos, casamientos y muertes. La organiza en hechos Prolog de la siguiente manera:

```
nacio(ana,1905).
nacio(mercedes,1906).
nacio(juan,1933).
nacio(pepe,1939).
nacio(angel,1958).
nacio(mario,1960).
nacio(lucia,1908).
nacio(nuria,1941).
nacio(marcela,1931).
seCasaron(ana,lucas,1932).
seCasaron(lucia,pepe,1961).
seCasaron(nuria,pepe,1966).
seCasaron(marcela,pepe,1967).
murio(juan,1993,mendoza).
murio(mercedes,1934,tucuman).
murio(angel,1986,rosario).
anioActual(2006).
```

A partir de esta información, definir los siguientes predicados. En todos los casos, los ejemplos se refieren a los hechos listados arriba.

- **edad**/2.
  Relaciona una persona con su edad. Tener en cuenta que
- para las personas que murieron se considera el año de fallecimiento
- para las que no, se considera el año actual
  P.ej. la edad de Ana es 101, mientras que la de Juan es 60.

- **especial**/1, que indica si una persona es especial o no. Se considera especial a una persona si cumple alguno de los requisitos que siguen:
  - se casó antes de los 18 años o después de los 50.
  - se casó tres veces o más.
  - su edad es mayor a 100.

  P.ej. son especiales Lucía (porque se casó a los 53 años) y Ana (porque su edad es 101).

- **anioConRegistro**/1 y **anioFeliz**/1
  Los años con registro son aquellos en los que se registró algún evento (nacimiento, casamiento o fallecimiento).
  Los años felices son aquellos en los que se registró algún evento, pero no se murió nadie.

- **haySeguidilla**/2
  Una seguidilla es una secuencia de años consecutivos en los cuales pasó algo en todos. P.ej. hay una seguidilla que empieza en 1931 y dura 4 años.
  Los argumentos del predicado haySeguidilla son año inicial y duración. P.ej. si consulto

  ```
  ?- haySeguidilla(1931,4).
  ```

  tiene que responder que sí, mientras que si pregunto

  ```
  ?- haySeguidilla(1931,5).
  ```

  tiene que responder que no.

Ejercicio 2

En un juego de "construya su cañería", hay piezas de distintos tipos: codos, caños y canillas.

- De los codos me interesa el color, p.ej. un codo rojo.
- De los caños me interesan color y longitud, p.ej. un caño rojo de 3 metros.
- De las canillas me interesan: tipo (de la pieza que se gira para abrir/cerrar), color y ancho (de la boca). P.ej. una canilla triangular roja de 4 cm de ancho.

a. Definir un predicado que relacione una cañería con su precio. Una cañería es una lista de piezas.
Los precios son:

- codos: $5.
- caños: $3 el metro.
- canillas: las triangulares $20, del resto $12 hasta 5 cm de ancho, $15 si son de más de 5 cm.

b. Definir el predicado puedoEnchufar/2, tal que

```
puedoEnchufar(P1,P2)
```

se verifique si puedo enchufar P1 a la izquierda de P2.
Puedo enchufar dos piezas si son del mismo color, o si son de colores enchufables.
Las piezas azules pueden enchufarse a la izquierda de las rojas, y las rojas pueden enchufarse a la izquierda de las negras. Las azules no se pueden enchufar a la izquierda de las negras, tiene que haber una roja en el medio.
P.ej.

- sí puedo enchufar (codo rojo, canio negro de 3 m).
- sí puedo enchufar (codo rojo, canio rojo de 3 m) (mismo color).
- no puedo enchufar (canio negro de 3 m, codo rojo) (el rojo tiene que estar a la izquierda del negro).
- no puedo enchufar (codo azul, canio negro de 3 m) (tiene que haber uno rojo en el medio).

c. Modificar el predicado puedoEnchufar/2 de forma tal que pueda preguntar por elementos sueltos o por cañerías ya armadas.
P.ej. una cañería (codo azul, canilla roja) la puedo enchufar a la izquierda de un codo rojo (o negro), y a la derecha de un canio azul.

Ayuda: si tengo una cañería a la izquierda, ¿qué color tengo que mirar? Idem si tengo una cañería a la derecha.

d. Definir un predicado canieriaBienArmada/1, que nos indique si una cañería está bien armada o no.
Una cañería está bien armada si a cada elemento lo puedo enchufar al inmediato siguiente, de acuerdo a lo indicado al definir el predicado puedoEnchufar/2.

e. Modificar el predicado puedoEnchufar/2 para tener en cuenta los extremos, que son piezas que se agregan a las posibilidades.
De los extremos me interesa de qué punta son (izquierdo o derecho), y el color, p.ej. un extremo izquierdo rojo.
Un extremo derecho no puede estar a la izquierda de nada, mientras que un extremo izquierdo no puede estar a la derecha de nada.

Verificar que canieriaBienArmada sigue funcionando.

Ayuda: resuélvanlo primero sin listas, y después agréguenle las listas. Lo de las listas sale en forma análoga a lo que ya hicieron, ¿en qué me tengo que fijar para una lista si la pongo a la izquierda o a la derecha?.

f. (a partir de acá son altamente difíciles!!)
Modificar el predicado canieriaBienArmada/1 para que acepte cañerías formadas por elementos y/u otras cañerías. P.ej. una cañería así: codo azul, [codo rojo, codo negro], codo negro → se considera bien armada.

g. armar las cañerías legales posibles a partir de un conjunto de piezas (si tengo dos veces la misma pieza, la pongo dos veces, p.ej. [codo rojo, codo rojo]).

Lógico: Ejercicios Otros

En estos ejercicios se empiezan a mezclar cosas que vimos en la materia con otras que no vimos, o no pusimos mucho énfasis; están pensados sobre todo para los que quieran seguir aprendiendo sobre programación lógica.

Ejercicio 1 – Mensajes de Texto

Se pide realizar un programa en prolog con la siguiente funcionalidad

a) Se cuenta con la siguiente base de conocimiento

```
tel(1,"!?@").
tel(2,"abc").
tel(3,"def").
tel(4,"ghi").
tel(5,"jkl").
tel(6,"mno").
tel(7,"pqrs").
tel(8,"tuv").
tel(9,"wxyz").
tel(0," ").
```

Implementar el predicado nrosPara/2, según esta especificación y ejemplos de consultas:

% nrosPara(String,ListaDeNumeros)

% String representa una palabra o frase y ListaDeNumeros las teclas asociadas al teléfono para poder representarla

```
?- nrosPara("hola",Numeros).

Numeros = [4, 6, 5, 2] ;
No

?- nrosPara(“hola como estas?”, Numeros).

Numeros = [4, 6, 5, 2, 0, 2, 6, 6, 6|...] [write]

% presionando la tecla W muestra la lista completa

Numeros = [4, 6, 5, 2, 0, 2, 6, 6, 6, 0, 3, 7, 8, 2, 7, 2] ;
No
```

El predicado nrosPara/2 debe ser inversible

```
?- nrosPara(Letras,[4, 6, 5, 2]).
Letras = [103, 109, 106, 97] ;
Letras = [103, 109, 106, 98] ;
Letras = [103, 109, 106, 99] ;
Letras = [103, 109, 107, 97] ;
Letras = [103, 109, 107, 98] ;
Letras = [103, 109, 107, 99] ;
Letras = [103, 109, 108, 97] ;
etc.
```

Tener en cuenta que SWI Prolog considera a un String como la lista de sus números ASCII

asociados.
En las consultas, pueden usar el predicado string_to_list para transformar la lista de ASCII a un string legible

?- nrosPara(Letras,[4, 6, 5, 2]), string_to_list(ResultadoMostrable,,Letras).

Lo malo de string_to_list es que no es inversible =(, por eso no conviene usarlo dentro de nrosPara que sí queremos que sea inversible.

b) Hacer el predicado nroGratis que relaciona una empresa con su número gratuito (el número gratuito tiene el formato 0800)

?- nroGratis(“skip”, Numeros).
Numeros = [0, 8, 0, 0, 7, 5, 4, 7] ;
No

c) Ahora se cambia nuestra base de conocimiento

```
tel(1,'!'). tel(1,'?'). tel(1,'@').
tel(2,'a').tel(2,'b').tel(2,'c').
tel(3,'d').tel(3,'e').tel(3,'f').
tel(4,'g').tel(4,'h').tel(4,'i').
tel(5,'j').tel(5,'k').tel(5,'l').
tel(6,'m').tel(6,'n').tel(6,'o').
tel(7,'p').tel(7,'q').tel(7,'r').tel(7,'s').
tel(8,'t').tel(8,'u').tel(8,'v').
tel(9,'w').tel(9,'x').tel(9,'y').tel(9,'z').
tel(0,' ').
```

Desarrollar nuevamente los predicados nrosPara/2 y nroGratis/2
Ayuda: utilizar los predicados para manejo de átomos atom_chars/2 y atom_concat/3

Ejercicio 2 - Manejo de Matrices en Lógico

Una matriz puede verse como una lista de listas, por ejemplo:

```
[[1,4], [2, 6]]     [[3, 6, 9], [5, 4, 1], [8, 2, 10]]
```

puede verse como las matrices cuadradas:

1	4
2	6

3	6	9
5	4	1
8	2	10

Ahora, veamos un poco más atentamente la segunda matriz. Podemos ver que es una matriz con varias propiedades particulares:

- Propiedad uno: Si intercalamos operaciones de sumas y restas entre los números de sus *filas* vemos que se forman operaciones aritméticas válidas:

3	+	6	=	9
5	-	4	=	1
8	+	2	=	10

- Propiedad dos: Si intercalamos operaciones de sumas y restas entre los números de sus *columnas* vemos que se forman operaciones aritméticas válidas:

3	6	9
+	-	+
5	4	1
=	=	=
8	2	10

Otra forma de expresar esta propiedad es decir que la **transpuesta** de la matriz cumple con la propiedad uno. (Las operaciones aritméticas no tiene por que ser las mismas)

3	+	5	=	8
6	-	4	=	2
9	+	1	=	10

- Propiedad tres: No se repiten los números usados en la matriz

A este tipo de matrices las llamaremos **Cuadrados Aritméticos**.

- Trabajaremos solo con matrices de 3X3, y con números en el rango de 1 a 10 inclusive
- Seria interesante poder hacer un programa en prolog que me permita comprobar/generar cuadrados aritméticos. Esto lo vamos a tratar de hacer a lo largo de la ejecitación.

Planteo General, con enfoque top-down

Pensemos primero en las propiedades que sabemos y como las podemos relacionar en un solo predicado:
Podemos suponer que exite el predicado **cumpleOperaciones(M)** verifica la propiedad uno.
Tambien que tenemos un predicado **transpuesta(M,MT)**, que podemos expresar la propiedad dos.
Y con el predicado **cumpleUnicidad(M)** expresamos la propiedad tres.

Con esto, podemos armar el predicado cuadradoAritmetico:

```
cuadrado_aritmetico(M):-
cumpleOperaciones(M),cumpleUnicidad(M),transpuesta(M,MT),cumpleOperaciones(MT).
```

El predicado cumpleUnicidad(M)

Como podemos determinar que los elementos de una matriz no se repiten?
Tip 1: es fácil determinar si los elementos de una lista no se repiten, si tuvieramos la lista aplanada de la matriz....
Tip 2: dijimos que una matriz es una lista de listas, recordá que en una definición recursiva sobre listas, parto de la definición para la cola, en este caso puedo suponer que sé cuál es el resultado de aplanar la cola...

El predicado transpuesta(M,MT)

Como podemos relacionar una matriz con su transpuesta?
Si pensamos que una matriz es una lista formada por las filas,
suena natural pensar que la trasnpuesta es la *lista fomrmada por las columnas* de la matriz original.
Entonces, tenemos que pensar en una forma de obtener las columnas de una matriz de una forma similar a como prolog nos permite obtener los elementos de una lista. O sea, separo la obtención de la primer columna del resto:

primerColumna([[3, 6, 9], [5, 4, 1], [8, 2, 10]],[3, 5, 8],[[6, 9], [4, 1], [2, 10]]).

primerColumna relaciona una matriz en su primer columna y el resto.

Tip: Parece complicado, pero se aclara si pensamos que la primer columna es la lista de los primeros elementos de cada fila; en particular, el primer elemento de la primer fila, es tambien el primer elemento de la primer columna .
Una ver realizado el predicado primerColumna, es más facil hacer el transpuesta.

Tip 1: La transpuesta de una matriz tendrá
- como primer fila: la primer columna de la matriz.
- como resto de las filas: el resultado de transponer el resto.

Tip 2: Con lo anterior más pruebas en papel debería salir fácil.

El predicado cumpleOperaciones(M)

Este el el gran predicado del problema, pero ya hemos reducido mucho su alcance. Solo debe determinar si:

3	6	9
5	4	1
8	2	10

Cumple con

3	**?**	6	=	9
5	**?**	4	=	1
8	**?**	2	=	10

Donde **?** puede ser en cada caso una suma o una resta.
Como estamos trabajando con matrices de 3X3 podemos asumir que la entrada es una lista de exactamenten 3 elementos.
Cada uno de estos debe de cumplir una operación. Entonces, qué mejor que armar un predicado operacion que relacione tres números si están ligados por alguna operación. Si recuerdo que los números válidos son solo los que estan entre 1 y 10 ambos inclusive, este predicado se puede hacer inversible.

Generación de cuadrados aritméticos

Hacer el predicado cuadrado_aritmetico(X), y verificar que es inversible (o sea que si no ligo el argumento en la consulta, genera los cuadrados).

Cuántos cuadrados aritméticos existen?

Hacer un predicado que permita determinar la cantidad de cuadrados aritméticos posibles a partir de los predicados construidos hasta ahora.