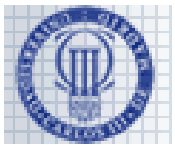
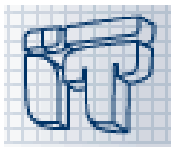


Inteligencia en Redes de Comunicaciones

Prolog



El objetivo es poner en práctica los conocimientos de Prolog estudiados en teoría. Vamos a utilizar SWI-Prolog:

http://www.swi-prolog.org/

La versión para Linux está instalada en

/var/home/asig/ircit/pl-5.6.23

y para ejecutar un programa sólo tienes que hacer:

/var/home/asig/ircit/pl-5.6.23/bin/pl –s <programa.pl>

También puedes descargar e instalar la versión para Windows, si tienes suficientes permisos de instalación.

Primero, a tu ritmo, estudia y prueba en el ordenador los siguientes programas Prolog, los que más te gusten y hasta donde llegues, no te preocupes:

1. **MONTY PYTHON AND THE HOLY GRAIL (1975)**
2. **CANCIÓN ESTÚPIDA**
3. **DIAGNÓSTICO DE ENFERMEDADES**
4. **PLANIFICADOR DE GASTOS**
5. **ÁRBOL GENEALÓGICO**
6. **RED SEMÁNTICA**
7. **HORÓSCOPO**
8. **OPERADOR DE CORTE**
9. **MANEJO DE LISTAS**
10. **ANALIZADOR MORFOLÓGICO**
11. **JUEGO DE LÓGICA**

Una vez que te hayas familiarizado un poco con el lenguaje, aborda el diseño e implementación de un sistema experto en Prolog, ya sea un sistema de diagnóstico mediante reglas (ejercicio 3) o bien que sea capaz de responder a preguntas basado internamente en una red semántica (ejercicio 6). Hazlo en el dominio que quieras. Como único requisito, el sistema debe disponer de reglas para al menos tres niveles de inferencia.

Calificación de la práctica (a sumar directamente a la nota final):

- Construir un sistema experto básico: +0.1 puntos
- Sistema avanzado: +0.1 puntos

1. Monty Python and the Holy Grail (1975)

A witch is a female who burns. Witches burn - because they're made of wood. Wood floats. What else floats on water? A duck; if something has the same weight as a duck it must float. A duck and scales are fetched. The girl and the duck balance perfectly. "It's a fair cop."

```
witch(X)  :- burns(X),female(X).
burns(X)  :- wooden(X).
```

```
wooden(X) :- floats(X).
floats(X) :- sameweight(duck, X).

female(girl).
sameweight(duck,girl).

? witch(girl).
```

2. Canción estúpida

```
cancionestupida(0):-nl,write('Gomo ya no gueda shevvezza, -hic- be boy a doddmig...').

cancionestupida(N):-N>1,nl,write(N),write(' botellas de cerveza en el suelo'),nl,
                    write(N),write(' botellas de cerveza'),nl,
                    write('Cojo una y me la bebo'),nl,
                    A is N-1, cancionestupida(A).

cancionestupida(N):-N=1,nl,write(N),write(' bodellia de shegvezza en el zsduelo'),nl,
                    write(N),write(' bodella de segbezha'),nl,
                    write('La gojo y be la bhebo'),nl,
                    A is N-1, cancionestupida(A).
```

3. Diagnóstico de enfermedades

```
enfermo_de(manuel,gripe).
tiene_sintoma(alicia,cansancio).
sintoma_de(fiebre,gripe).
sintoma_de(tos,gripe).
sintoma_de(cansancio,anemia).
elimina(vitaminas,cansancio).
elimina(aspirinas,fiebre).
elimina(jarabe,tos).
recetar_a(X,Y):-enfermo_de(Y,A),alivia(X,A).
alivia(X,Y):-elimina(X,A),sintoma_de(A,Y).

enfermo_de(X,Y):-tiene_sintoma(X,Z),sintoma_de(Z,Y).
```

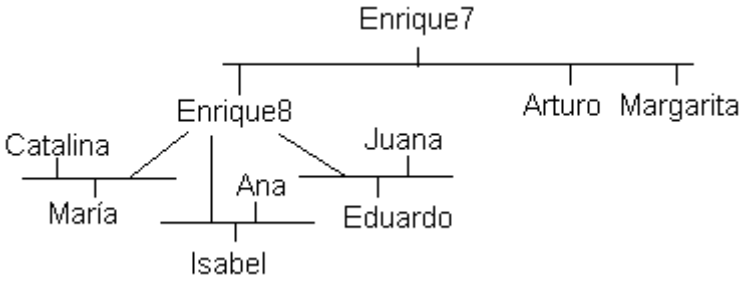
4. Planificador de gastos

```
transporte(roma,20000).
transporte(londres,25000).
transporte(tunez,15000).

alojamiento(hotel,roma,25000).
alojamiento(hotel,londres,15000).
alojamiento(hotel,tunez,10000).
alojamiento(hostal,roma,15000).
alojamiento(hostal,londres,10000).
alojamiento(hostal,tunez,8000).
alojamiento(camping,roma,10000).
alojamiento(camping,londres,5000).
alojamiento(camping,tunez,5000).

viaje(W,X,Y,Z):-transporte(W,A),alojamiento(Y,W,C),B is C*X,Z is A+B.
```

5. Árbol genealógico



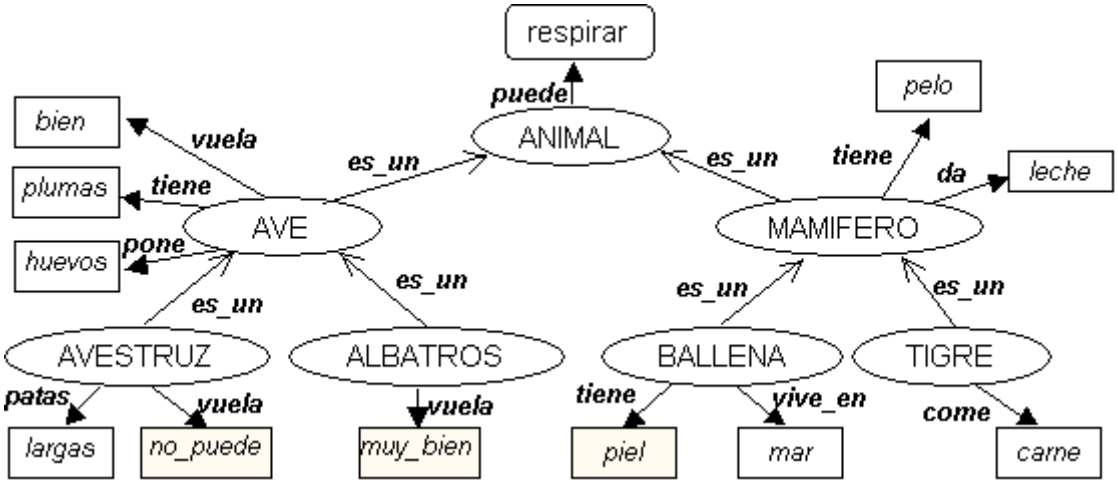
Definir en Prolog los predicados que definen por extensión todas las relaciones familiares directas, **padre(Padre, Hijo)** y **madre(Madre, Hijo)** del árbol genealógico de la familia Tudor.

Definir la relación **progenitor**, utilizando las relaciones de padre y madre. Definir recursivamente la relación **antepasado**. Probar definiciones alternativas de esta relación cambiando el orden de los predicados. Comprobar cómo afecta al comportamiento del programa el orden usado en las distintas definiciones de antepasado.

Definir nuevas relaciones (como **hermano**, **hermana**, **abuelo**, **abuela**) añadiendo los predicados (por ejemplo **mujer**, **hombre**) y reglas necesarios.

6. Red semántica

Prolog es un lenguaje muy adaptado para el desarrollo de aplicaciones en Inteligencia Artificial, en las que un problema básico es representar el conocimiento de un dominio concreto de forma que pueda ser interpretado correctamente en el ordenador. Uno de los métodos de representación, basado en modelos de psicología cognitiva, son las redes semánticas. Las redes semánticas son grafos orientados que proporcionan una representación declarativa de objetos, propiedades y relaciones. Los nodos se utilizan para representar objetos o propiedades. Los arcos representan relaciones entre nodos del tipo **es_un**, **es_parte_de**, etc. El mecanismo de inferencia básico en las redes semánticas es la herencia de propiedades. La figura representa esquemáticamente un ejemplo de red semántica:



Representar en Prolog la red semántica de la figura. Se sugiere emplear un predicado binario por cada relación. Para las propiedades se sugiere que se emplee el predicado **atributo(Objeto, Atributo, Valor)**. Por ejemplo, **atributo(albatros, vuela, muy_bien)**.

Incluir las reglas necesarias para que todo objeto herede los atributos y las propiedades de todas las clases a las que pertenece.

En ocasiones algunos de los atributos heredados por un objeto deben ser sustituidos por otros particulares. Así es posible tratar las excepciones, por ejemplo, las aves que no vuelan (como el avestruz). Representar adecuadamente las excepciones que se señalan en el ejemplo. Se sugiere emplear el predicado **particular(Objeto, Atributo, Valor)**, con el que se especifica que ese atributo es particular de ese objeto y sustituye al heredado.

Emplear Prolog para hacer consultas sobre la información almacenada en la red semántica. Por ejemplo, ¿tiene pelo el avestruz?, ¿quiénes pueden volar?, etc.

7. Horóscopo

```
/* horoscopo(Signo,DiaIni,MesIni,DiaFin,MesFin)
    <- son del signo Signo los nacidos entre el DiaIni/MesIni y el DiaFin/MesFin */
horoscopo(aries,21,3,21,4).
horoscopo(tauro,21,4,21,5).
horoscopo(geminis,21,5,21,6).
horoscopo(cancer,21,6,21,7).
horoscopo(leo,21,7,21,8).
horoscopo(virgo,21,8,21,9).
horoscopo(libra,21,9,21,10).
horoscopo(escorpio,21,10,21,11).
horoscopo(sagitario,21,11,21,12).
horoscopo(capricornio,21,12,21,1).
horoscopo(acuario,21,1,21,2).
horoscopo(piscis,21,2,21,3).

/* signo(Dia,Mes,Signo) <- los nacidos el Dia/Mes pertenecen al signo Signo */
signo(Dia,Mes,Signo) :- horoscopo(Signo,D1,M1,D2,M2),
                        ( (Mes=M1,Dia>=D1) ; (Mes=M2,Dia=<D2) ).

?- signo(8,5,tauro).
```

```
?- signo(7,8,Signo).
?- signo(7,X,Signo).
?- signo(X,7,Signo).
```

Realiza las modificaciones pertinentes a este programa para que el día esté comprendido dentro del rango permitido para cada mes. Por ejemplo no podemos poner el 30 de febrero. No es necesario que compruebes si el año es bisiesto. No compliques la regla "signo" con condiciones complejas, sino que añade los hechos que consideres oportunos ("el mes de marzo tiene 31 días").

SOLUCIÓN:

```
/* mes(Mes,Dias,Nombre) <- el mes Nombre, de número Mes tiene Dias */
mes(1,31,enero).
mes(2,28,febrero).
mes(3,31,marzo).
mes(4,30,abril).
mes(5,31,mayo).
mes(6,30,junio).
mes(7,31,julio).
mes(8,31,agosto).
mes(9,30,septiembre).
mes(10,31,octubre).
mes(11,30,noviembre).
mes(12,31,diciembre).

signo(Dia,Mes,Signo) :- horoscopo(Signo,D1,M1,D2,M2),
                        ( (Mes=M1,Dia>=D1,mes(M1,D,_),Dia=<D) ;
                          (Mes=M2,Dia=<D2, Dia>0) ).
```

8. Operador de corte

```
/* sumatorio(Num,Sum) <- Sum es el sumatorio desde 1 hasta Num */
sumatorio(1,1) :- !.
sumatorio(N,S) :- N1 is N-1,
                  sumatorio(N1,S1),
                  S is N+S1.

/* natural(Num) <- Num es un número perteneciente a los Naturales */
natural(0).
natural(X) :- natural(Y),
              X is Y+1.

/* diventera(Dividendo,Divisor,Cociente) <- Cociente es el resultado de la división */
diventera(A,B,C) :- natural(C),
                    Y1 is C*B,
                    Y2 is (C+1)*B,
                    Y1=<A, Y2>A, !.
```

9. Manejo de listas

```
/* miembro(Elem,Lista) <- el término Elem pertenece a la lista Lista */
miembro(X,[X|_]).
miembro(X,[_|Y]) :- miembro(X,Y).

/* nel(Lista,N) <- el número de elementos de la lista Lista es N */
nel([],0).
nel([_|Y],N) :- nel(Y,M),
                N is M+1.

/* es_lista(Lista) <- Lista es una lista */
es_lista([]).
es_lista([_|_]).

/* concatena(L1,L2,L3) <- concatenación de las listas L1 y L2 dando lugar a L3 */
concatena([],L,L).
concatena([X|L1],L2,[X|L3]) :- concatena(L1,L2,L3).
```

```
/* ultimo(Elem,Lista) <- Elem es el último elemento de la lista Lista */
ultimo(X,[X]).
ultimo(X,[_|Y]) :- ultimo(X,Y).

/* inversa(Lista,Inver) <- Inver es la inversa de la lista Lista */
inversa([],[]).
inversa([X|Y],L) :- inversa(Y,Z),
                    concatena(Z,[X],L).

/* borrar(Elem,L1,L2) <- se borra el elemento Elem de la lista L1 obteniendose L2 */
borrar(X,[X|Y],Y).
borrar(X,[Z|L],[Z|M]) :- borrar(X,L,M).

/* subconjunto(L1,L2) <- la lista L1 es un subconjunto de la lista L2 */
subconjunto([X|Y],Z) :- miembro(X,Z),
                        subconjunto(Y,Z).
subconjunto([],_).

/* insertar(Elem,L1,L2) <- se inserta el elemento Elem en la lista L1 obteniendose L2 */
insertar(E,L,[E|L]).
insertar(E,[X|Y],[X|Z]) :- insertar(E,Y,Z).

/* permutacion(L1,L2) <- la lista L2 es una permutación de la lista L1 */
permutacion([],[]).
permutacion([X|Y],Z) :- permutacion(Y,L),
                        insertar(X,L,Z).

/* sust(E1,E2,L1,L2) <- L2 es L1 sustituyendo las ocurrencias del elemento E1 por E2 */
sust(_,_,[],[]).
sust(E1,E2,[E1|L1],[E2|L2]) :- !, sust(E1,E2,L1,L2).
sust(E1,E2,[Y|L1],[Y|L2]) :- sust(E1,E2,L1,L2).

/* union(L1,L2,L3) <- L3 es la lista-conjunto unión de L1 y L2 */
union([],L,L).
union([X|L1],L2,L3) :- miembro(X,L2), !,
                       union(L1,L2,L3).
union([X|L1],L2,[X|L3]) :- union(L1,L2,L3).

?- miembro(d,[a,b,c,d,e]).
?- miembro(d,[a,b,c,[d,e]]).
?- miembro(d,[a,b,c]).
?- miembro(E,[a,b,[c,d]]).
?- nel([a,b,[c,d],e],N).
?- es_lista([a,b,[c,d],e]).
?- concatena([a,b,c],[d,e],L).
?- concatena([a,b,c],L,[a,b,c,d,e]).
?- concatena(L1,L2,[a,b]).
```

1.- Escribe, basándote en el procedimiento "miembro" visto anteriormente, un nuevo procedimiento "miembro" con 3 parámetros. El nuevo argumento haría referencia a la posición ocupada por el elemento Elem en la lista Lista.
2.- Escribe, basándote en el procedimiento "borrar" visto anteriormente, un nuevo procedimiento "borrarN" que borre el elemento que ocupa la posición N en la lista Lista1 obteniéndose Lista2.

SOLUCIÓN:

```
/* miembro(Elem,Lista,Pos) <- el término Elem pertenece a la lista Lista y
                              ocupa la posición Pos */
miembro(X,[X|_],1).
miembro(X,[_|Y],N) :- miembro(X,Y,N1),
                      N is N1+1.

/* borrarN(Pos,L1,L2) <- se borra el elemento Elem que ocupa la posición
                         Pos de la lista L1 obteniendose la lista L2 */
borrarN(1,[_|Y],Y).
borrarN(N,[Z|L],[Z|M]) :- N1 is N-1,
                          borrarN(N1,L,M).
```

10. Analizador morfológico

El analizador morfológico proporciona el análisis morfológico completo de una palabra dada, es decir, su categoría morfológica (nombre, pronombre, adjetivo, determinante, verbo, adverbio, preposición, conjunción, interjección…) con sus rasgos correspondientes (si es nombre: masculino/femenino y singular/plural, si es verbo: persona, número, tiempo y modo…).

Las palabras en español se comportan de acuerdo a modelos de derivación y conjugación, entre los que están, por ejemplo, los verbos regulares en las 3 conjugaciones (ar, er, ir).

En Prolog es fácil realizar analizadores morfológicos, definiendo modelos que proporcionan los lingüistas. Por ejemplo:

```
% modelo “perro”
analisis(perro,nms,o).              -- nombre masculino singular
analisis(perro,nmp,os).             -- nombre masculino plural
analisis(perro,nfs,a).              -- nombre femenino singular
analisis(perro,nfp,as).             -- nombre femenino plural
%
modelo(perr,perro).
modelo(niñ,perro).

% modelo “señor”
analisis(señor,nms,’’).             -- nombre masculino singular
analisis(señor,nmp,es).             -- nombre masculino plural
analisis(señor,nfs,a).
analisis(señor,nfp,as).
%
modelo(señor,señor).

% modelo “balón”
analisis(balón,nms,’’).             -- nombre masculino singular
analisis(balón,nmp,es).             -- nombre masculino plural
%
modelo(balón,señor).
modelo(balcón,señor).


% modelo “cantar”
analisis(cantar,presenteindicativo1s,o).    -- 1ª persona singular presente indicativo
analisis(cantar,presenteindicativo2s,as).   -- 1ª persona singular presente indicativo
analisis(cantar,presenteindicativo3s,a).    -- 1ª persona singular presente indicativo
…
analisis(cantar,futurosimple1s,aré).        -- 1ª persona singular futuro simple indic.
…
modelo(cant,cantar).
modelo(am,cantar).
modelo(gust,cantar).
modelo(estudi,cantar).


% modelo “temer”
analisis(temer,presenteindicativo1s,o).     -- 1ª persona singular presente indicativo
analisis(temer,presenteindicativo2s,es).    -- 1ª persona singular presente indicativo
analisis(temer,presenteindicativo3s,e).     -- 1ª persona singular presente indicativo
…
modelo(tem,temer).
modelo(beb,temer).
modelo(com,temer).
```

Con ayuda de listas, es muy sencillo analizar una palabra: el análisis morfológico lo proporciona la terminación (llamada desinencia), que, concatenada a la raíz (llamada lexema), forma la palabra completa. Escribe la regla de concatenación que da el análisis morfológico: analisismorfologico(palabra, analisis).

Las excepciones se tratan directamente:

```
analisismorfologico(soy,presenteindicativo1s).
analisismorfologico(fui,presenteindicativo1s).
```

11. Juego de lógica

Vamos a implementar en Prolog juegos de lógica que se tienen que resolver por deducción a partir de un conjunto de pistas. El objetivo es correlacionar una serie de propiedades que cumplen distintos elementos de nuestro Dominio (Universo del Discurso). La restricción a la que está sujeto este juego es que dos elementos distintos de un mismo Universo no pueden tener la misma característica.

Nuestro acertijo: *"Un alumno de Informática, debido al nerviosismo del primer día de clase, ha anotado el nombre de sus profesores (María, Jesús y Faraón), las asignaturas que se imparten (Lógica, Programación y Matemáticas) y el día de la semana de las distintas clases (lunes, miércoles y jueves), pero sólo recuerda que:*
- La clase de Programación, impartida por María, es posterior a la de Lógica
- A Faraón no le gusta trabajar los lunes, día en el que no se imparte Lógica
¿Serías capaz de ayudarle a relacionar cada profesor con su asignatura, así como el día de la semana que se imparte?
(Sabemos que cada profesor imparte una única asignatura y que las clases se dan en días diferentes)"

El siguiente programa sirve como programa de control.

```
/*----             JUEGO DE LÓGICA                 ----*/

mensaje :- nl,write('Ejemplo "Juego Lógico" cargado.'),nl,
           write('Se lanza con ?- iniciar.'),
           nl,nl.

/*----             PROGRAMA PRINCIPAL                 ----*/

/* iniciar <- llamada inicial del programa */
iniciar :- write('Base de Conocimientos: '),
           read(BC),
           consult(BC),!,
           nl,write('Base de Conocimientos '),write(BC),
           write(' consultada'),nl,nl,
           numeroPropiedades(N),
           objetosUniverso(M),
           iniciar(N,M).
iniciar :- nl,write('ERROR: Base de Conocimientos no encontrada'),nl.
iniciar(2,M) :- !,ini(M,[],[]).
```

```
iniciar(3,M) :- !,ini(M,[],[],[]).
iniciar(4,M) :- !,ini(M,[],[],[],[]).
iniciar(5,M) :- !,ini(M,[],[],[],[],[]).
iniciar(N,_) :- nl,write('ERROR: Número de Propiedades incorrecto = '),
                write(N),nl.

/* ini(Sol1,Sol2,...) <- Sol1 es una lista con los objetos del dominio 1,
            Sol2 la lista con los objetos del dominio 2, ...
            con las soluciones respectivamente relacionadas. */

/* Correlacionar 2 propiedades */
ini(M,L1,L2) :- nel(L1,M),escribir(L1,L2),nl,pausa,fail.
ini(M,L1,L2) :- r1(Obj1,Obj2),
                nopertenece(Obj1,L1),
                nopertenece(Obj2,L2),
                ini(M,[Obj1|L1],[Obj2|L2]).

/* Correlacionar 3 propiedades */
ini(M,L1,L2,L3) :- nel(L1,M),escribir(L1,L2,L3),nl,pausa,fail.
ini(M,L1,L2,L3) :- r1(Obj1,Obj2),
                   nopertenece(Obj1,L1),
                   nopertenece(Obj2,L2),
                   r2(Obj1,Obj3),
                   nopertenece(Obj3,L3),
                   r3(Obj2,Obj3),
                   ini(M,[Obj1|L1],[Obj2|L2],[Obj3|L3]).

/* Correlacionar 4 propiedades */
ini(M,L1,L2,L3,L4) :- nel(L1,M),escribir(L1,L2,L3,L4),nl,pausa,fail.
ini(M,L1,L2,L3,L4) :- r1(Obj1,Obj2),
                      nopertenece(Obj1,L1),
                      nopertenece(Obj2,L2),
                      r2(Obj1,Obj3),
                      nopertenece(Obj3,L3),
                      r3(Obj1,Obj4),
                      nopertenece(Obj4,L4),
                      r4(Obj2,Obj3),
                      r5(Obj2,Obj4),
                      r6(Obj3,Obj4),
                      ini(M,[Obj1|L1],[Obj2|L2],[Obj3|L3],[Obj4|L4]).

/* Correlacionar 5 propiedades */
ini(M,L1,L2,L3,L4,L5) :- nel(L1,M),escribir(L1,L2,L3,L4,L5),nl,pausa,fail.
ini(M,L1,L2,L3,L4,L5) :- r1(Obj1,Obj2),
                         nopertenece(Obj1,L1),
                         nopertenece(Obj2,L2),
                         r2(Obj1,Obj3),
                         nopertenece(Obj3,L3),
                         r3(Obj1,Obj4),
                         nopertenece(Obj4,L4),
                         r4(Obj1,Obj5),
                         nopertenece(Obj5,L5),
                         r5(Obj2,Obj3),
                         r6(Obj2,Obj4),
                         r7(Obj2,Obj5),
                         r8(Obj3,Obj4),
                         r9(Obj3,Obj5),
                         r10(Obj4,Obj5),
                         ini(M,[Obj1|L1],[Obj2|L2],[Obj3|L3],[Obj4|L4],[Obj5|L5]).

/*----          RUTINAS GENERALES                    ----*/

/* escribir(Lista1,Lista2,...) <- escribe las soluciones correlacionadas
              de las listas: Lista1, Lista2 ...  */
escribir([],[]).
escribir([Obj1|Resto1],[Obj2|Resto2]) :-
                write(Obj1), write(' - '),write(Obj2),nl,
                escribir(Resto1,Resto2).

escribir([],[],[]).
escribir([Obj1|Resto1],[Obj2|Resto2],[Obj3|Resto3]) :-
                write(Obj1), write(' - '),write(Obj2),
                write(' - '), write(Obj3),nl,
                escribir(Resto1,Resto2,Resto3).
```

```
escribir([],[],[],[]).
escribir([Obj1|Resto1],[Obj2|Resto2],[Obj3|Resto3],[Obj4|Resto4]) :-
                write(Obj1), write(' - '),write(Obj2),
                write(' - '), write(Obj3),write(' - '),write(Obj4),nl,
                escribir(Resto1,Resto2,Resto3,Resto4).

escribir([],[],[],[],[]).
escribir([Obj1|Resto1],[Obj2|Resto2],[Obj3|Resto3],[Obj4|Resto4],[Obj5|Resto5]) :-
                write(Obj1), write(' - '),write(Obj2),write(' - '),
                write(Obj3),write(' - '),write(Obj4),write(' - '),
                write(Obj5),nl,
                escribir(Resto1,Resto2,Resto3,Resto4,Resto5).


/* pausa <- detiene la ejecución del programa hasta que se pulse una tecla */
pausa :- write('Pulsa <return> para buscar otra solucion'),
         skip(10),nl.

/*----          RUTINAS DE MANEJO DE LISTAS           ----*/

/* nopertenece(Elem,Lista)) <- el elemento Elem no pertenece a la lista Lista */
nopertenece(_,[]).
nopertenece(E,[X|L]) :- E\=X,
                        nopertenece(E,L).

/* nel(Lista,N) <- el número de elementos de la lista Lista es N */
nel([],0).
nel([_|L],N) :- nel(L,M),
                N is M+1.

:- mensaje.
```

Escribe la base de conocimientos adecuada al acertijo anterior.

SOLUCIÓN:

```
/*----           BASE DE CONOCIMIENTOS                ----*/

numeroPropiedades(3).
objetosUniverso(3).

/*-                     PROPIEDADES                       -*/

/* prof(Profesor) <- Profesor es el nombre de un profesor */
prof(maria).
prof(jesus).
prof(faraon).

/* asig(Asignatura) <- Asignatura es el nombre de una asignatura */
asig(logica).
asig(programacion).
asig(matematicas).

/* dia(Dia) <- Dia es un día de la semana que hay alguna clase */
dia(lunes).
dia(miercoles).
dia(jueves).


/*-                     RELACIONES                        -*/

/* r1(Profesor,Asignatura) <- Profesor imparte la Asignatura */
r1(maria,programacion).
r1(Profesor,Asignatura) :- prof(Profesor), Profesor\=maria,
                           asig(Asignatura).

/* r2(Profesor,Dia) <- Profesor imparte sus clases el Dia de la semana */
r2(faraon,Dia) :- dia(Dia), Dia\=lunes.
r2(Profesor,Dia) :- prof(Profesor), Profesor\=faraon,
                    dia(Dia).
```

```
/* r3(Asignatura,Dia) <- Asignatura se imparte el Dia de la semana */
r3(logica,Dia) :- dia(Dia), Dia\=lunes, Dia\=jueves.
r3(programacion,Dia) :- dia(Dia), Dia\=lunes.
r3(Asignatura,Dia) :- asig(Asignatura), Asignatura\=logica,
                      Asignatura\=programacion, dia(Dia).
```

[Varias fuentes, entre ellas: Departamento de Tecnología Informática y Computación (Universidad de Alicante) y Departamento de Ciencias de la Computación e I.A. (Universidad de Sevilla)]