Ejercicio 1. Dadas las siguientes reglas:

```
humano(turing).
humano(socrates).
griego(socrates).
mortal(X) :- humano(X).
```

Se pide:
a) Definir un objetivo (consulta) que nos permita conocer todos los mortales que son griegos.
b) Especificar todos los pasos que debe realizar el intérprete Prolog para obtener las soluciones del objetivo anterior.

Ejercicio 2. Dada el siguiente programa:

```
progenitor(joaquin, marcos).
progenitor(joaquin, facundo).
progenitor(florencia, matias).
progenitor(martia, andrea).
hermano(X,Y) :- progenitor(Z,X),progenitor(Z,Y).
```

Se pide:
a) Explique porque es posible que una persona sea hermano de si misma.
b) Modifique la definición para que esto no suceda.
c) Dado el Objetivo: hermano(marcos, X)?, Indicar la salida.

Ejercicio 3. Suponga definidas las siguientes claúsulas:

```
padre (X,Y).       X es padre de Y.
madre (X,Y).       X es madre de Y.
hombre (X).        X es hombre.
mujer (X).         X es mujer.
progenitor (X,Y). X es padre o madre de Y.
```

Se pide definir en base a ellas las siguientes relaciones:

```
es_madre (X). X es madre.
es_padre (X). X es padre.
es_hijo (X). X es hijo.
hija (X,Y). X es hija de Y.
tio (X,Y). X es tío de Y.
sobrino (X,Y). X es sobrino de Y.
prima (X,Y). X es prima de Y.
abuelo_o_abuela (X,Y). X es abuelo o abuela de Y
```

Ejercicio 4. Dado el siguiente programa Prolog:

```
hombre(carlos).
hombre(juan).
hombre(pedro).
hermano(carlos,maria).
hermano(carlos,juan).
hermano(carlos,raul).
madre(maria,luis).
madre(maria,hugo).
padre(raul,jose).
tio(pablo,luis).
tio(X,Y) :- hombre(X), hermano(X,Z), progenitor(Z,Y).
progenitor(juan,pedro).
progenitor(X,Y) :- padre(X,Y).
progenitor(X,Y) :- madre(X,Y).
```

Se pide realizar manualmente los pasos realizados al efectuarse las siguientes consultas:

a) tio (carlos,A)?
b) tio (A,luis)?
c) tio (A,B)?

Ejercicio 5. Defina el predicado POTENCIA:

potencia (3,0,X). ---> X=1
potencia (5,2,25). ---> true

Ejercicio 6. Defina el predicado MODULO:

modulo(10,5,R). ---> R=0
modulo(14,3,R). ---> R=2

Ejercicio 7. Dado la siguiente definición del predicado UNION:

```
union([], L2, L2).
union([X|L1], L2, [X|L3]) :- union(L1, L2, L3).
```

Evalúe las siguientes invocaciones:

a) union([1,2], [3,4,5], [1,2,3,4,5]).
b) union (A, [3,4,5], [1,2,3,4,5]).
c) union ([1,2], [3,4,5], A).
d) union (A, B, [1,2,3,4,5]).

Ejercicio 8. Defina el predicado MIEMBRO:

miembro (1, [3,2,1]). ---> true
miembro (X, [3,2,1]). ---> X=3
X=2
X=1

Ejercicio 9. Defina el predicado ADYAC que verifica si dos elementos son adyacentes en una lista.

adyac(3, 2, [3,2,1]). ---> true
adyac(2, 3, [3,2,1]). ---> true
adyac(X, 2, [3,2,1]). ---> X=3
X=1

Ejercicio 10. Dadas las cláusulas **medico** y **paciente** y una relación entre ellas representada por la cláusula atiende, mediante los siguientes hechos:

```
medico(m1,rosales).
medico(m2,manni).
paciente(p1,juan).
paciente(p2,ana).
atiende(m1,p1).
atiende(m1,p2).
atiende(m2,p2).
```

Construya un programa de manera tal que cuando se invoque como **base,** aparezca en pantalla lo siguiente:

1. Dado un médico, listar los pacientes que atiende.
2. Dado un paciente, listar los médicos que lo atienden.
3. Terminar.

Se espera que:

a) Si se responde 1, el programa deberá preguntar por el nombre del médico y mostrar todos los pacientes que atiende, y luego vuelva a mostrar el menu.
b) Si se responde 2, el programa deberá preguntar por el nombre del paciente y mostrar todos los médicos que lo atienden, y luego vuelva al menú.
c) Si se responde 3, el programa deberá finalizar dando mensaje que diga "ADIOS".

Ejercicio 11. Defina el predicado ELIM que permita eliminar un elemento de una lista.

elim(2, [3,2,1], [3,1]). ---> true
elim(2, [2,3,1], X). ---> X=[3,1]

Ejercicio 12. Defina el predicado REVES:

reves([3,2,1], [1,2,3]). ---> true (Hacer seguimiento)
reves([3,2,1], X). ---> X=[1,2,3]

Ejercicio 13. Dado el siguiente programa:

```
encontrar_pareja :-  persona(H, m, List1),
                     persona(M, f, List2),
                     interes_comun(List1, List2,_),
                     write(H),
                     write('esta muy enamorado con '),
                     write(M), nl.
encontrar_pareja :- write("Fin de lista").
interes_comun(L1, L2, X) :- member(X, L1), member(X, L2), !.
persona(carlos, m, [viajar,libros,basquet]).
persona(maria, f, [viajar,libros,natacion]).
persona(juana, f, [estudiar,libros,tenis]).
member(X, [X|_]).
member(X, [_|L]) :- member(X,L).
```

Si se dispara el objetivo encontrar_pareja, se pide:

a) Decir que muestra por pantalla.
b) Hacer el seguimiento.
c) Decir que mostraría en caso de no tener el !.

Ejercicio 14. Dado el siguiente programa:

```
% auto(codigo,nombre,precio)
auto(10,ford,8000).
auto(20,fiat,9000).
….
% socio (numero, nombre, total abonado)
socio(100, juan,8000).
socio(200, luis, 5000).
socio(300, carlos,9000).
socio(400, raul, 4500).
socio(500, mario, 8000).
…..
% grupo (codigo_auto, lista de socios)
grupo(10, [200,500,400]).
grupo(20, [300,400]).
…..
listado(P) :- auto(A,P,Precio), !,grupo(A, Lista),sigue(Lista, Precio).
sigue([ ], _) :- write(' Listado completo para: ').
sigue([X|Y], Precio) :- socio(X,Nom,Pagado), Pagado=Precio, write(Nom),
sigue(Y,Precio),!.
sigue([_|Y],P) :- sigue(Y,P).
```

Se pide:

1. Dado el objetivo listado (X):
   - **a)** Mostrar el valor, o los valores, de la variable X.
   - **b)** Realizar el árbol de seguimiento.
   - **c)** Indicar las salidas en caso que el predicado no tuviera el !(cut) .

2. Confeccionar un predicado que muestre la cantidad de socios por cada marca de auto.

EJERCICIO PROPUESTO

```
animales(mamifero([vaca,mono])).
animales(reptil([serpiente,coco])).
animales(pez([salmon])).
```

1. Realizar el seguimiento para las siguientes consultas:
   - **a)** ?- animales(mamifero(X)).
   - **b)** ?- animales(Z), !.
   - **c)** ?- animales(pez([salmon])).

2. Construya una función que permita mostrar la clase a que corresponda un animal en caso de ser posible, sino mostrar un cartel de "ERROR".

   Ej.: ?- es (serpiente) reptil
   ?- es (mono) mamifero
   ?- es (tigre) ERROR