

Proyecto 1

# Manual de Usuario

---

201612139      Jeralmy Alejandra de León Samayoa

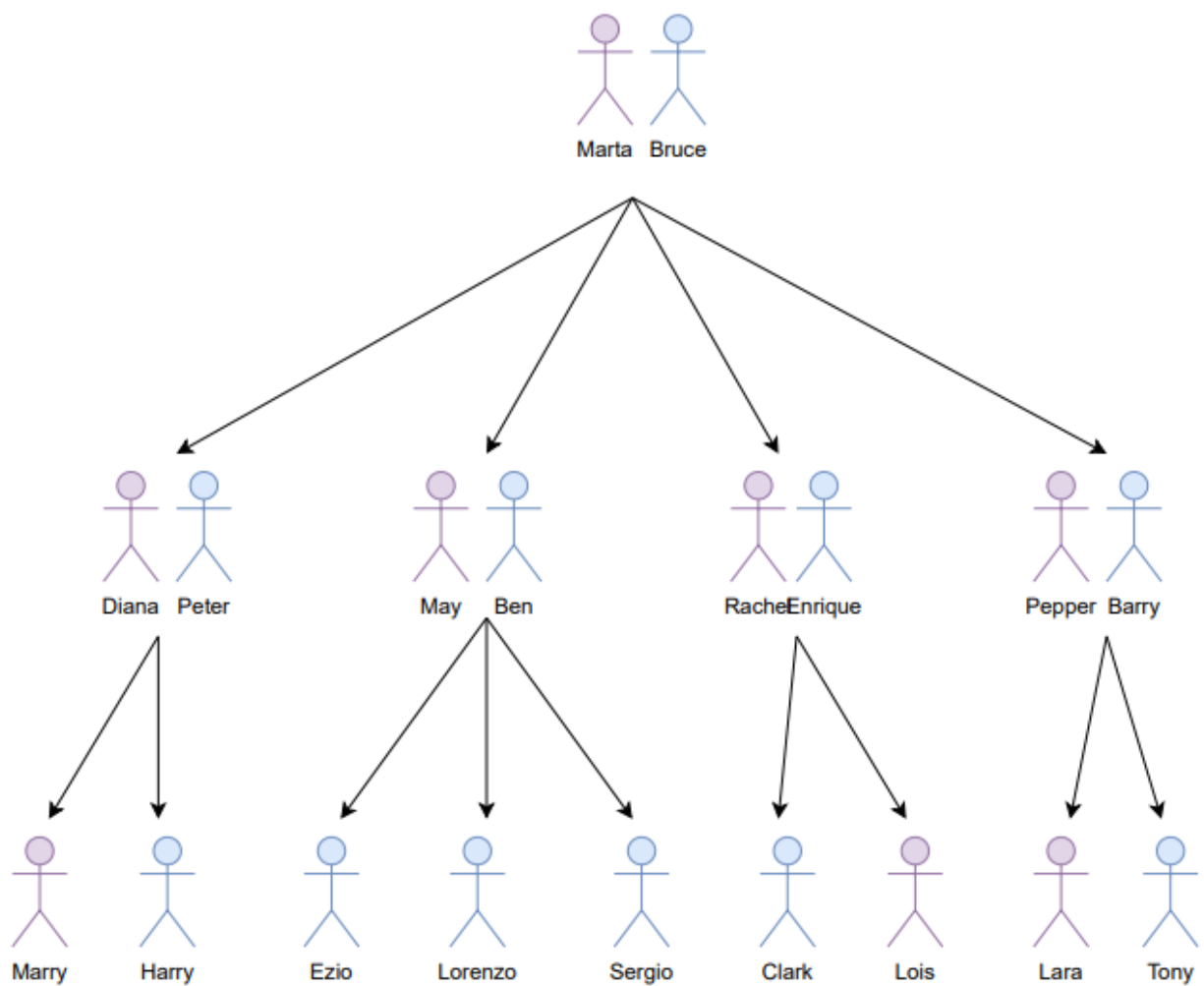
201403532      Ever Eduardo Chicas Prado

07 de septiembre del 2021

## Problema 1

### Solución:

Árbol genealógico:



Con los datos del problema se determinó el árbol anterior, aplicando relaciones de madre, padre, hijo, hermana, hermano y pareja.

## ¿Quién fue la persona culpable del asesinato?

### 1. Integrantes de la familia:

- Marta
- Bruce
- Diana
- Peter
- May
- Ben
- Rachel
- Enrique
- Pepper
- Barry
- Mary
- Harry
- Ezio
- Sergio
- Lorenzo
- Clark
- Lois
- Lara
- Tony

### 2. La hermana de el/la culpable tenía sospechas de dicha persona.

**Solución:** Se descarta a todas las personas que no tienen hermana.

- Diana
- May
- Rachel
- Barry
- Harry
- Clark
- Tony

3. Las sospechas las comento con tres parientes, el primer pariente era el abuelo que se llama Bruce.

**Solución:** Se descartan a todas las personas que no tengan de abuelo a Bruce.

- Harry
- Clark
- Tony

4. Segundo pariente el primo Clark.

**Solución:** Se descartan a todas las personas que no tengan de primo a Clark.

- Harry
- Tony

5. Tercer pariente el tío Barry.

**Solución:** Se descartan a todas las personas que no tengan de tío a Barry.

- Harry

**Resultado:** El culpable del asesinato de la abuela Marta fue el nieto Harry.

## Hechos

- **pareja(X,Y):** La persona X es pareja de la persona Y.

```
pareja(marta,bruce) .  
pareja(bruce,marta) .
```

- **madre(X,Y):** La persona X es madre de la persona Y.

```
madre(marta,diana) .  
madre(marta,may) .
```

- **padre(X,Y):** La persona X es padre de la persona Y.

```
padre(bruce,diana) .  
padre(bruce,may) .
```

- **hijo(X,Y):** La persona X es hijo de la persona Y.

```
hijo(barry,marta) .  
hijo(barry,bruce) .
```

- **hermana(X,Y):** La persona X es hermana de la persona Y.

```
hermana(diana,may) .  
hermana(diana,rachel) .
```

- **hermano(X,Y):** La persona X es hermano de la persona Y.

```
hermano(barry,diana) .  
hermano(barry,may) .
```

## Reglas

6. **Regla para encontrar al culpable:** Verifica que si la persona X es el culpable y cumple con los hechos y reglas. Si es culpable retorna **true**, si no retorna **false**.

```
culpable(X) :- hermana(Y,X) ,  
               abuelo(bruce,Y) ,  
               primo(clark,Y) ,  
               tio(barry,Y) .
```

7. **Regla para determinar al abuelo:** La persona X es abuelo de la persona Y.

```
abuelo(X, Y) :- padre(X, Z) ,  
               (hijo(Y,Z)) .
```

8. **Regla para determinar al primo:** La persona X es primo de la persona Y.

```
primo(X,Y) :- (madre(A,X) , padre(B,X) , pareja(A,B)) ,
```

```
(hijo(X,A);hijo(X,B)),

(madre(C,Y),padre(D,Y),pareja(D,C)),

((hermana(A,C);hermana(A,D));(hermano(B,C))).
```

9. **Regla para determinar al tío:** La persona X es tío de la persona Y.

```
tio(X,Y) :- hermano(X,Z),

           madre(Z,Y).
```

10. **Regla para obtener la lista de hijos de la persona X.**

```
obtener_hijos(X,L) :- (setof(Y,hijo(Y,X),L);

                      L = []).
```

11. **Regla para obtener los parientes a partir de la persona X:**

```
arbol(X) :- (pareja(X,Z) -> graficar_arbol([X], '-> '));

            (hijo(X,Y) -> write('---> '), write(X));

            ((write('No existe el integrante en la familia.'), nl) -> false).
```

Primera opción, si la persona tiene pareja se obtiene la lista de sus parientes a partir de la persona.

Segunda opción, si la persona no tiene pareja solo es hijo se imprime su nombre.

Tercera opción, si la persona no existe en el árbol genealógico ->false.

12. **Regla recursiva para obtener la lista de los parientes a partir de la persona X.**

```
graficar_arbol([], NivelAnterior) :- write('').

graficar_arbol([Cabeza|Cola], NivelAnterior) :-

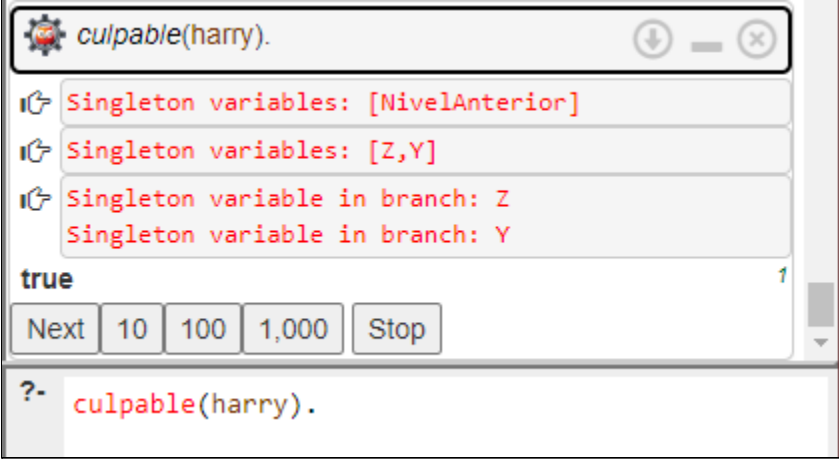
    (pareja(Cabeza,Z) -> atom_concat('-----', NivelAnterior, NivelActual),
    write(NivelActual), write(Cabeza), write(' - '), write(Z), nl;

    atom_concat('-----', NivelAnterior, NivelActual), write(NivelActual),
    write(Cabeza), nl),
```

```
obtener_hijos(Cabeza,Hijos),  
graficar_arbol(Hijos,NivelActual),  
graficar_arbol(Cola, NivelAnterior).
```

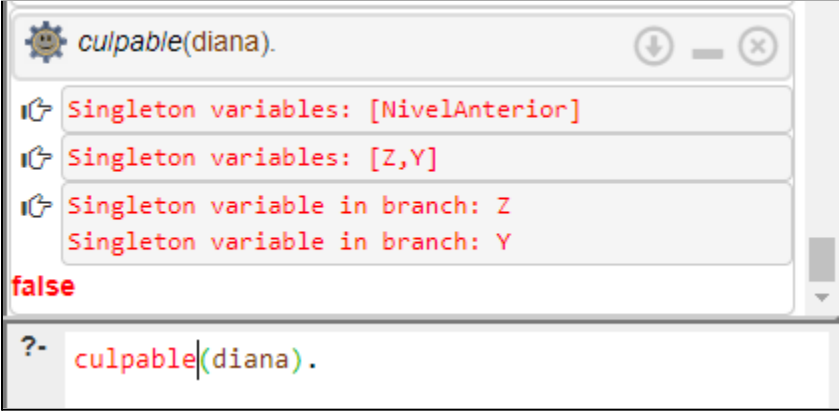
Salida:

Culpable



The screenshot shows a Prolog interpreter window titled `culpable(harry).`. The window displays the following information:

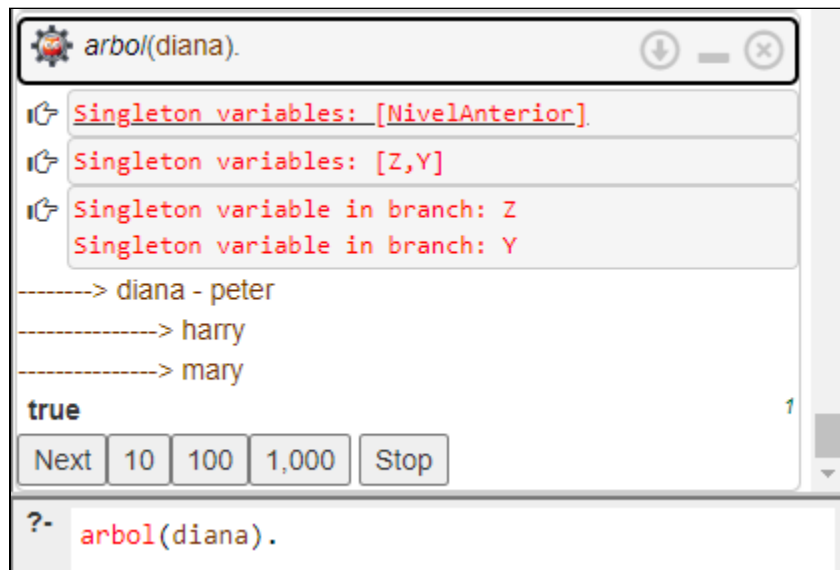
- Singleton variables: `[NivelAnterior]`
- Singleton variables: `[Z,Y]`
- Singleton variable in branch: `Z`
- Singleton variable in branch: `Y`
- The result is `true`.
- Buttons: `Next`, `10`, `100`, `1,000`, and `Stop`.
- The query prompt `?- culpable(harry).` is shown at the bottom.



The screenshot shows a Prolog interpreter window titled `culpable(diana).`. The window displays the following information:

- Singleton variables: `[NivelAnterior]`
- Singleton variables: `[Z,Y]`
- Singleton variable in branch: `Z`
- Singleton variable in branch: `Y`
- The result is `false`.
- The query prompt `?- culpable(diana).` is shown at the bottom.

## Árbol



arbol(diana).

Singleton variables: [NivelAnterior].

Singleton variables: [Z,Y]

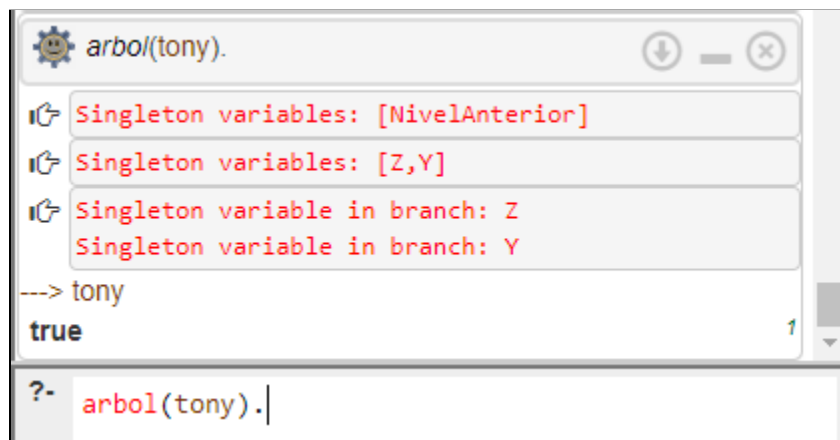
Singleton variable in branch: Z  
Singleton variable in branch: Y

-----> diana - peter  
-----> harry  
-----> mary

true

Next 10 100 1,000 Stop

?- arbol(diana).



arbol(tony).

Singleton variables: [NivelAnterior]

Singleton variables: [Z,Y]

Singleton variable in branch: Z  
Singleton variable in branch: Y

---> tony

true

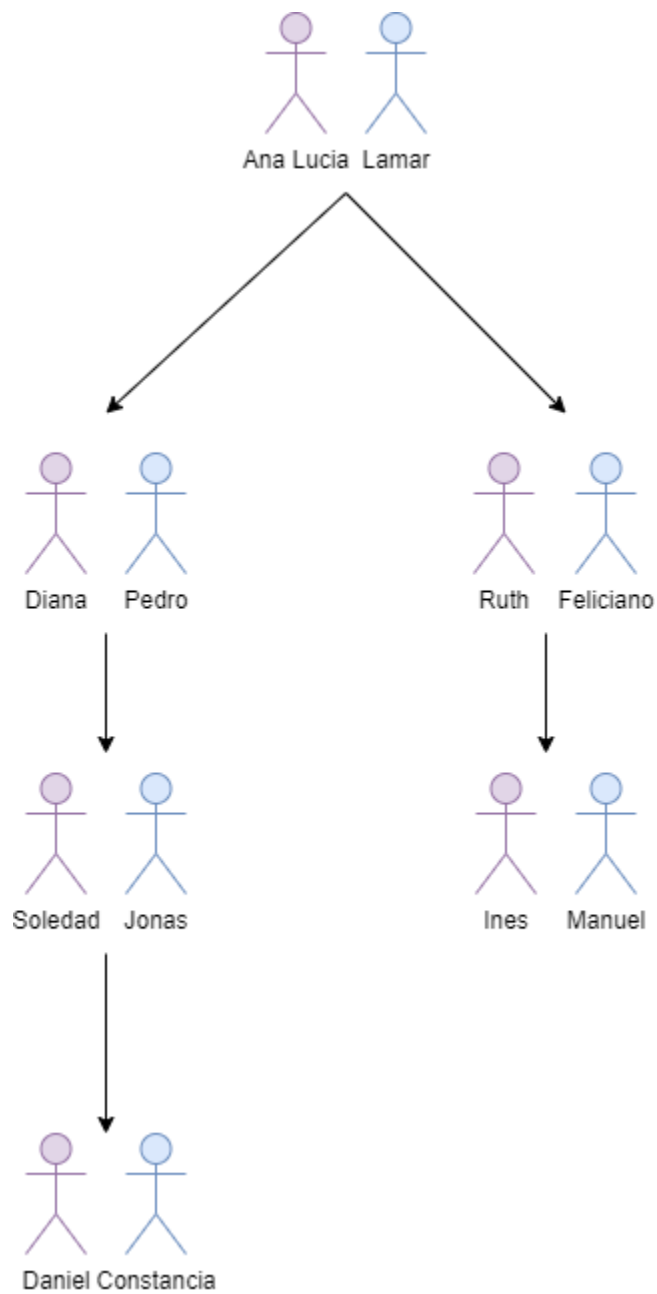
?- arbol(tony).|



## Problema 2

### Solución:

Árbol genealógico:



Con los datos del problema se determinó el árbol anterior, aplicando relaciones de madre, padre, hijo, hermana, hermano, pareja y se buscó el año de cada situación del enunciado para asociar las personas a la línea de tiempo:

Daniel y Constanca nacieron tras establecerse el Euro como moneda en Europa	2002
Manuel es primo hermano de Jonas, el cual nació el año en que Juan Carlos fue proclamado Rey de España	1975
Ana Lucia se casó justo después de que terminó la Guerra Civil en España	1939
Pedro y Feliciano son cuñados y nacieron el mismo año en que acabó la Segunda Guerra Mundial	1945
Soledad nació el mismo año en que se aprobó la última Constitución española.	1978

## Hechos

- **pareja(X,Y):** La persona X es pareja de la persona Y.

```
pareja(analucia,lamar) .  
pareja(lamar,analucia) .
```

- **madre(X,Y):** La persona X es madre de la persona Y.

```
madre(analucia,pedro) .  
madre(analucia,ruth) .
```

- **padre(X,Y):** La persona X es padre de la persona Y.

```
padre(lamar,pedro) .  
padre(lamar,ruth) .
```

- **hijo(X,Y):** La persona X es hijo de la persona Y.

```
hijo(pedro,lamar).
hijo(ruth,lamar).
```

- **hermana(X,Y):** La persona X es hermana de la persona Y.

```
hermana(ruth,pedro).
hermana(ines,manuel).
```

- **hermano(X,Y):** La persona X es hermano de la persona Y.

```
hermano(pedro,ruth).
hermano(manuel,ines).
```

## Reglas

1. **Regla para obtener la lista de hijos de la persona X.**

```
obtener_hijos(X,L):- (setof(Y,hijo(Y,X),L);
                     L = []).
```

2. **Regla para obtener los parientes a partir de la persona X:**

```
arbol(X):- (pareja(X,Z)-> graficar_arbol([X],'-> '));
           (hijo(X,Y)-> write('---> '),write(X));
           ((write('No existe el integrante en la familia.'),nl)-> false).
```

Primera opción, si la persona tiene pareja se obtiene la lista de sus parientes a partir de la persona.

Segunda opción, si la persona no tiene pareja solo es hijo se imprime su nombre.

Tercera opción, si la persona no existe en el árbol genealógico ->false.

3. **Regla recursiva para obtener la lista de los parientes a partir de la persona X.**

```
graficar_arbol([],NivelAnterior):- write('').
```

```

graficar_arbol([Cabeza|Cola],NivelAnterior):-
    (pareja(Cabeza,Z)->atom_concat('-----',NivelAnterior,NivelActual),
    write(NivelActual),write(Cabeza),write(' - '),write(Z),nl;

    atom_concat('-----',NivelAnterior,NivelActual),write(NivelActual),
    write(Cabeza),nl),

    obtener_hijos(Cabeza,Hijos),

    graficar_arbol(Hijos,NivelActual),

    graficar_arbol(Cola, NivelAnterior).

```

Salida:

Árbol

```

?- arbol(analucia).
-----> analucia - lamar
-----> pedro - diana
-----> jonas - soledad
-----> constancia
-----> daniel
-----> ruth - feliciano
-----> ines
-----> manuel
true .

```

## Problema 3

### Reverso de una lista

#### Solución:

```
1 % --- reverso de lista
2 reverso([],[]).
3
4 reverso([CabezaLista|ColaLista],Lista):- reverso(ColaLista,Colainvertida),concatenar(Colainvertida,[CabezaLista],Lista).
5 % Concatenar
6 concatenar([],Lista,Lista).
7 concatenar([CabezaLista|ColaLista],Lista2, [CabezaLista|Lista3]):- concatenar(ColaLista,Lista2,Lista3).
```

#### Hechos

```
2 reverso([],[]).
```

```
6 concatenar([],Lista,Lista).
```

#### Reglas

```
4 reverso([CabezaLista|ColaLista],Lista):- reverso(ColaLista,Colainvertida),concatenar(Colainvertida,[CabezaLista],Lista).
7 concatenar([CabezaLista|ColaLista],Lista2, [CabezaLista|Lista3]):- concatenar(ColaLista,Lista2,Lista3).
```

#### Ejemplo:

```
?- reverso([a,b,c,d,e,f,g],I).
I = [g, f, e, d, c, b, a].
```

## Lista palindroma

Solución:

```
9 % --- palindromo de una lista
10 palindromo([],[]).
11 es_palindromo(Entrada):- reverso(Entrada,Entrada).
```

Hechos

```
10 palindromo([],[]).
```

Reglas

```
11 es_palindromo(Entrada):- reverso(Entrada,Entrada).
```

Ejemplo:

```
?- es_palindromo([a,b,c,d,e,f,g]).
false.
```

Ejemplo 2:

```
?- es_palindromo([a,n,a]).  
[a,n,a]  
true.
```

## Duplicar lista

Solución:

```
13 % ---- duplicar lista  
14 duplicar_elementos([],[]).  
15 duplicar_elementos([CabezaLista|ColaLista2],[CabezaLista,CabezaLista|Lista3]):-duplicar_elementos(ColaLista2,Lista3).
```

Hechos

```
14 duplicar_elementos([],[]).
```

Reglas

```
duplicar_elementos([CabezaLista|ColaLista2],[CabezaLista,CabezaLista|Lista3]):-  
    duplicar_elementos(ColaLista2,Lista3).
```

Ejemplo:

```
?- duplicar_elementos([a,b,c,d,e,f,g],L).  
L = [a, a, b, b, c, c, d, d, e|...].
```

## Dividir lista en 2

Solución:

```

17 % ----- Dividir lista en 2
18 div(Lista, ListaA, ListaB) :-
19     append(ListaA, ListaB, Lista),
20     length(ListaA, LenghtA),
21     length(ListaB, LenghtB),
22     ((LenghtA-1)==LenghtB; (LenghtA+1)==LenghtB; LenghtA==LenghtB), !.
23

```

## Reglas

```

18 div(Lista, ListaA, ListaB) :-
19     append(ListaA, ListaB, Lista),
20     length(ListaA, LenghtA),
21     length(ListaB, LenghtB),
22     ((LenghtA-1)==LenghtB; (LenghtA+1)==LenghtB; LenghtA==LenghtB), !.

```

## Ejemplo:

```

?- div([a,b,c,d,e,f,g],A,B).
A = [a, b, c].
B = [d, e, f, g].

```

## Insertar valor por índice

### Solución:

```

24 % ----- Insertar valor en index
25 insertarElemento(Elem,[],_Pos,[Elem]).
26 insertarElemento(Elem,Lista,1,[Elem|Lista]).
27 insertarElemento(Elem,[CabezaLista|ColaLista],Pos,[CabezaLista|ColaElem]):- Pos1 is Pos-1, insertarElemento(Elem,ColaLista,Pos1,ColaElem).

```

## Hechos

```

25 insertarElemento(Elem,[],_Pos,[Elem]).
26 insertarElemento(Elem,Lista,1,[Elem|Lista]).
27 insertarElemento(Elem,[CabezaLista|ColaLista],Pos,[CabezaLista|ColaElem]):- Pos1 is Pos-1, insertarElemento(Elem,ColaLista,Pos1,ColaElem).

```



## Reglas

```
insertarElemento(Elem,[CabezaLista|ColaLista],Pos,[CabezaLista|ColaElem):-  
    Pos1 is Pos-1, insertarElemento(Elem,ColaLista,Pos1,ColaElem).
```

## Ejemplo:

```
?- insertarElemento(prueba,[1,2,3,4],2,R).  
R = [1, prueba, 2, 3, 4]
```

## Problema 4

### Solución:

```
1  
2 % Inicio de predicado, llama otros predicados  
3 sudoku(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4) :-  
4     resolver(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4),  
5     printsudoku(F1C1, F1C2, F1C3, F1C4),  
6     printsudoku(F2C1, F2C2, F2C3, F2C4),  
7     printsudoku(F3C1, F3C2, F3C3, F3C4),  
8     printsudoku(F4C1, F4C2, F4C3, F4C4).  
9  
10 % Imprimir los valores en orden  
11 printsudoku(A, B, C, D) :- write(' '), write(A), write(' '), write(B), write(' '), write(C), write(' '), write(D), nl.  
12  
13 % Resuelve los valores para cada posicion  
14 resolver(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4) :-  
15     validarNumeros(F1C1, F1C2, F1C3, F1C4), % primera fila  
16     validarNumeros(F2C1, F2C2, F2C3, F2C4), % segunda fila  
17     validarNumeros(F3C1, F3C2, F3C3, F3C4), % tercera fila  
18     validarNumeros(F4C1, F4C2, F4C3, F4C4), % cuarta fila  
19     validarNumeros(F1C1, F2C1, F3C1, F4C1), % primera columna  
20     validarNumeros(F1C2, F2C2, F3C2, F4C2), % segunda columna  
21     validarNumeros(F1C3, F2C3, F3C3, F4C3), % tercera columna  
22     validarNumeros(F1C4, F2C4, F3C4, F4C4), % cuarta columna  
23     validarNumeros(F1C1, F1C2, F2C1, F2C2), % block de arriba para la izquierda  
24     validarNumeros(F1C3, F1C4, F2C3, F2C4), % block de arriba para la derecha  
25     validarNumeros(F3C1, F3C2, F4C1, F4C2), % block de abajo para la izquierda  
26     validarNumeros(F3C3, F3C4, F4C3, F4C4). % block de abajo para la derecha  
27  
28 % Determine si cada cuadrado de una fila, columna o bloque está  
29  
30 % validarNumeros  
31 validarNumeros(A, B, C, D) :- num(A), num(B), num(C), num(D), A\=B, A\=C, A\=D, B\=C, B\=D, C\=D.  
32  
33 % Initialize numbers  
34 num(1). num(2). num(3). num(4).
```

## Reglas

### sudoku:

```
1
2 % Inicio de predicado, llama otros predicados
3 sudoku(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4) :-
4     resolver(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4),
5     printsudoku(F1C1, F1C2, F1C3, F1C4),
6     printsudoku(F2C1, F2C2, F2C3, F2C4),
7     printsudoku(F3C1, F3C2, F3C3, F3C4),
8     printsudoku(F4C1, F4C2, F4C3, F4C4).
```

### printsudoku:

```
9
10 % Imprimir los valores en orden
11 printsudoku(A, B, C, D) :- write(' '), write(A), write(' '), write(B), write(' '), write(C), write(' '), write(D), nl.
```

### resolver:

```
13 % Resuelve los valores para cada posicion
14 resolver(F1C1, F1C2, F1C3, F1C4, F2C1, F2C2, F2C3, F2C4, F3C1, F3C2, F3C3, F3C4, F4C1, F4C2, F4C3, F4C4) :-
15     validarNumeros(F1C1, F1C2, F1C3, F1C4), % primera fila
16     validarNumeros(F2C1, F2C2, F2C3, F2C4), % segunda fila
17     validarNumeros(F3C1, F3C2, F3C3, F3C4), % tercera fila
18     validarNumeros(F4C1, F4C2, F4C3, F4C4), % cuarta fila
19     validarNumeros(F1C1, F2C1, F3C1, F4C1), % primera columna
20     validarNumeros(F1C2, F2C2, F3C2, F4C2), % segunda columna
21     validarNumeros(F1C3, F2C3, F3C3, F4C3), % tercera columna
22     validarNumeros(F1C4, F2C4, F3C4, F4C4), % cuarta columna
23     validarNumeros(F1C1, F1C2, F2C1, F2C2), % block de arriba para la izquierda
24     validarNumeros(F1C3, F1C4, F2C3, F2C4), % block de arriba para la derecha
25     validarNumeros(F3C1, F3C2, F4C1, F4C2), % block de abajo para la izquierda
26     validarNumeros(F3C3, F3C4, F4C3, F4C4). % block de abajo para la derecha
```

### validarNumeros

```
30 % validarNumeros
31 validarNumeros(A, B, C, D) :- num(A), num(B), num(C), num(D), A\=B, A\=C, A\=D, B\=C, B\=D, C\=D.
```

### Ejemplo:

---

```
?- sudoku(_,_ ,2,3,_,_,_,_,_,_,_,_,3,4,_,_) .  
  4   1   2   3  
  2   3   4   1  
  1   2   3   4  
  3   4   1   2  
true ■
```