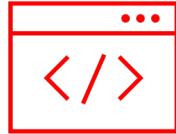




UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN



# Prolog

Prof. Oscar Mauricio Salazar Ospina

**Correo:** omsalazaro@unal.edu.co

3007743 - Programación Lógica y Funcional  
3010426 - Teoría de Lenguajes de Programación

Facultad de Minas  
Departamento de ciencias de la computación y la decisión  
Septiembre 7 de 2022

*Universidad Nacional de Colombia*

---

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN

# Prolog

## Contenido Módulo Prolog



# Prolog

## Contenido



Cláusulas, predicados y términos



Hechos y reglas



Back-chaining



Backtracking



Variables y Unificación



Revisión de elementos del lenguaje PROLOG

# Prolog

## Back-chaining

### Clausulas de Horn (Reglas)

**R1:**  $b \wedge d \wedge e \rightarrow f$

**R2:**  $d \wedge g \rightarrow a$

**R3:**  $c \wedge f \rightarrow a$

**R4:**  $b \rightarrow x$

**R5:**  $d \rightarrow e$

**R6:**  $a \wedge x \rightarrow h$

**R7:**  $c \rightarrow d$

**R8:**  $x \wedge c \rightarrow a$

**R9:**  $x \wedge b \rightarrow d$



### Hechos Iniciales

**b, c**

Ejemplo de hechos

b : "El paciente tiene fiebre"

c : "El paciente tiene cansancio"

...

E : "El paciente tiene tos"

h : "El paciente tiene anemia"

Problema: ¿Se puede verificar **h**?

# Prolog

## Back-chaining

### Clausulas de Horn (Reglas)

**R1:**  $b \wedge d \wedge e \rightarrow f$   
**R2:**  $d \wedge g \rightarrow a$   
**R3:**  $c \wedge f \rightarrow a$   
**R4:**  $b \rightarrow x$   
**R5:**  $d \rightarrow e$   
**R6:**  $a \wedge x \rightarrow h$   
**R7:**  $c \rightarrow d$   
**R8:**  $x \wedge c \rightarrow a$   
**R9:**  $x \wedge b \rightarrow d$

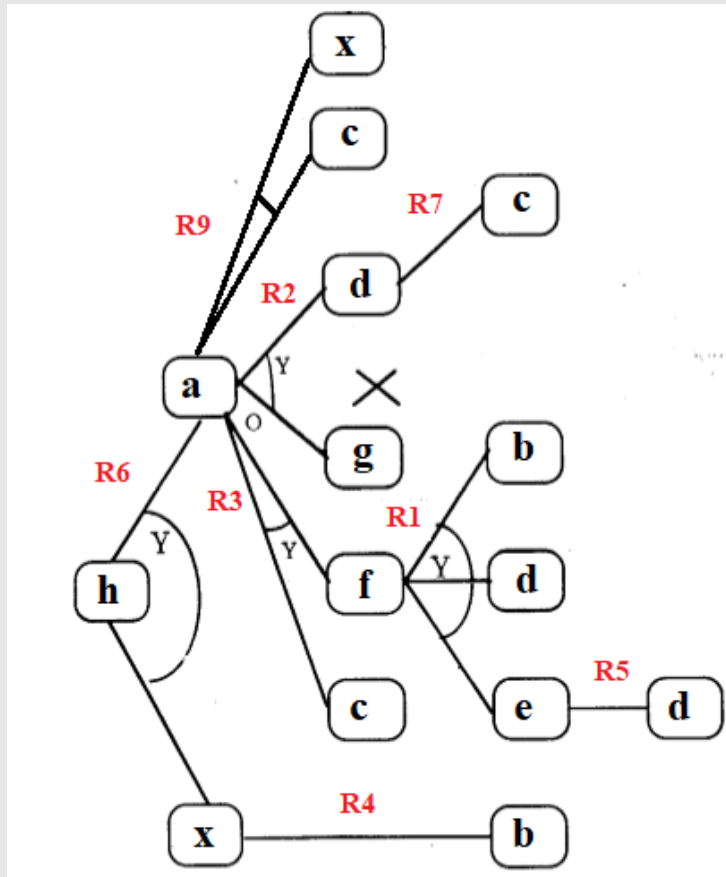


### En PROLOG (Reglas)

**/\* BASE DE HECHOS \*/**  
b. c.

**/\* BASE DE REGLAS \*/**  
f :- b, d, e. **/\* R1 \*/**  
a :- d, g. **/\* R2 \*/**  
a :- c, f. **/\* R3 \*/**  
a :- x, c. **/\* R9 \*/**  
x :- b. **/\* R4 \*/**  
e :- d. **/\* R5 \*/**  
h :- a, x. **/\* R6 \*/**  
d :- c. **/\* R7 \*/**  
d :- x, b. **/\* R8 \*/**  
g :- false. **/\* R10 \*/**

## Back-tracking



## En PROLOG (Reglas)

```
/* BASE DE HECHOS */
b. c.
```

**/\* BASE DE REGLAS \*/**

```
f :- b, d, e. /* R1 */
```

```
a :- d, g. /* R2 */
```

a :- c, f. /\* R3 \*/

a :- x, c. /\* R9 \*/

```
x :- b.    /* R4 */
```

```
e :- d.      /* R5 */
```

```
h :- a, x.    /* R6 */
```

```
d :- c.    /* R7 */
```

```
d :- x,b.    /* R8 */
```

```
g :- false.  /* R10 */
```

# Prolog

## Back-chaining

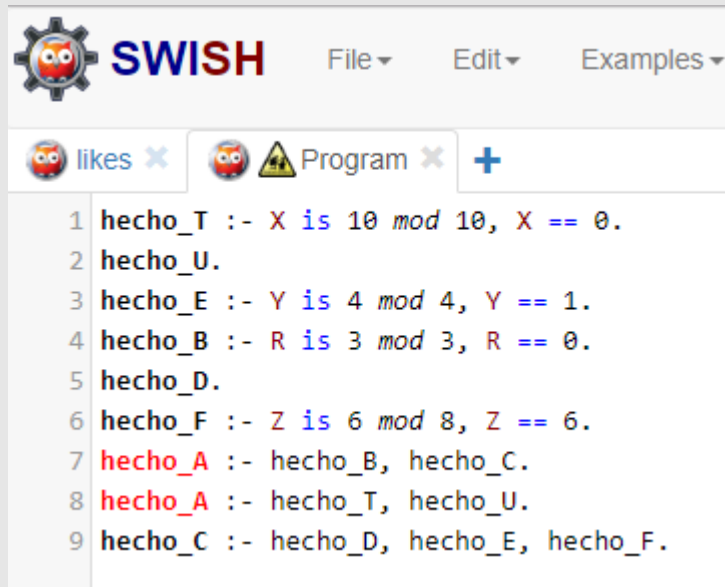
**Backtracking (busca hacia atrás otras soluciones)** aplica **back-chaining** varias veces.





# Prolog

## Back-tracking



```
1 hecho_T :- X is 10 mod 10, X == 0.
2 hecho_U.
3 hecho_E :- Y is 4 mod 4, Y == 1.
4 hecho_B :- R is 3 mod 3, R == 0.
5 hecho_D.
6 hecho_F :- Z is 6 mod 8, Z == 6.
7 hecho_A :- hecho_B, hecho_C.
8 hecho_A :- hecho_T, hecho_U.
9 hecho_C :- hecho_D, hecho_E, hecho_F.
```

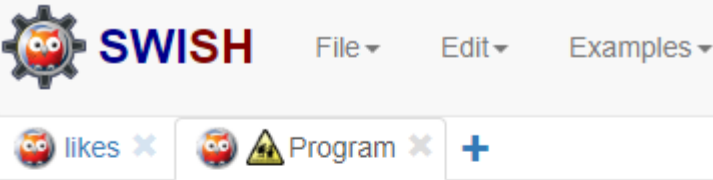


Árbol???

¿Se puede verificar el **hecho\_A** ?

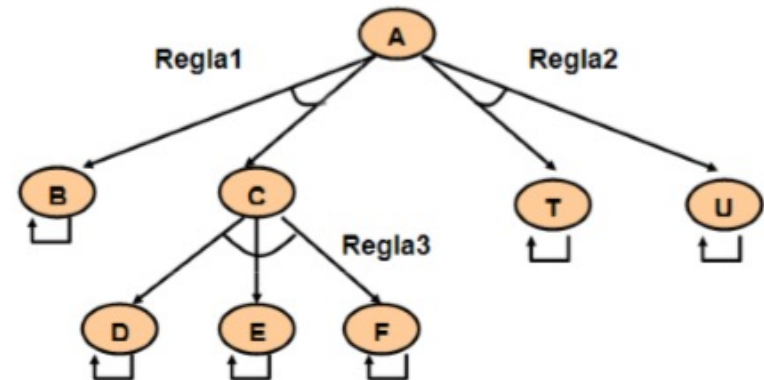
# Prolog

## Back-tracking



```
1 hecho_T :- X is 10 mod 10, X == 0.
2 hecho_U.
3 hecho_E :- Y is 4 mod 4, Y == 1.
4 hecho_B :- R is 3 mod 3, R == 0.
5 hecho_D.
6 hecho_F :- Z is 6 mod 8, Z == 6.
7 hecho_A :- hecho_B, hecho_C.
8 hecho_A :- hecho_T, hecho_U.
9 hecho_C :- hecho_D, hecho_E, hecho_F.
```


```
A :- B , C.           /* Regla 1 */
A :- T , U.           /* Regla 2 */
C :- D , E , F.       /* Regla 3 */
```




¿Se puede verificar el **hecho\_A** ?

# Prolog


## Back-tracking

 `hecho_A.`


`true`

 `hecho_B.`


`true`

 `hecho_D.`


`true`

 `hecho_E.`


`false`

 `hecho_F.`

`true`

 `hecho_T.`

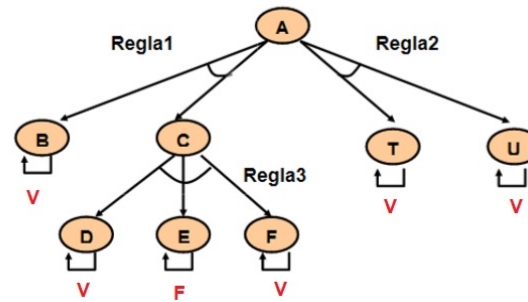
`true`


 `hecho_U.`



`true`

`?- hecho_U.`



```
A :- B, C.           /* Regla 1 */
A :- T, U.           /* Regla 2 */
C :- D, E, F.        /* Regla 3 */
```



 **SWISH** File Edit Examples

 likes  Program +

```
1 hecho_T :- X is 10 mod 10, X == 0.
2 hecho_U.
3 hecho_E :- Y is 4 mod 4, Y == 1.
4 hecho_B :- R is 3 mod 3, R == 0.
5 hecho_D.
6 hecho_F :- Z is 6 mod 8, Z == 6.
7 hecho_A :- hecho_B, hecho_C.
8 hecho_A :- hecho_T, hecho_U.
9 hecho_C :- hecho_D, hecho_E, hecho_F.
```

 `trace, hecho_A.` 

`Call: hecho_A`

`Call: hecho_B`

`Call: _6302 is 3 mod 3`

`Exit: 0 is 3 mod 3`

`Call: 0==0`

`Exit: 0==0`

`Exit: hecho_B`

`Call: hecho_C`

`Call: hecho_D`

`Exit: hecho_D`

`Call: hecho_E`

`Call: _6308 is 4 mod 4`

`Exit: 0 is 4 mod 4`

`Call: 0==1`

`Fail: 0==1`

`Fail: hecho_E`

`Fail: hecho_C`

`Redo: hecho_A`

`Call: hecho_T`

`Call: _6302 is 10 mod 10`

`Exit: 0 is 10 mod 10`

`Call: 0==0`

`Exit: 0==0`

`Exit: hecho_T`

`Call: hecho_U`

`Exit: hecho_U`

`Exit: hecho_A`

`true`

# Prolog

## Hechos con argumentos – predicados poliádicos

### – Hechos:

- `salario(juan, 500).`            `/* Juan gana 500$ */`
- `salario(pepe, 200).`            `/* Pepe gana 200$ */`
- `salario(rosa, 100 * 2).`    `/* Rosa Gana 100$ en 2 horas*/`

### – Realicemos las siguientes consultas:

- `salario(pEPe, 200)`            `/* ¿pEPe gana 200$? */`
- `salario(pepe, 200)`            `/* ¿pepe gana 200$? */`
- `salario(pepe, 50 * 4)`    `/* ¿pepe gana 50$ en 4 horas? */`
- `salario(rosa, 200)`            `/* ¿rosa gana 200$? */`
- `salario(rosa, 100 * 2)`       `/* ¿rosa gana 100$ en 2 horas */`
- `juan(500)`                    `/* ¿Es posible preguntarlo así? */`

# Prolog



## Hechos con argumentos – predicados poliádicos

### – Hechos:

- `salario(juan, 500).`      `/* Juan gana 500$ */`
- `salario(pepe, 200).`      `/* Pepe gana 200$ */`
- `salario(rosa, 100 * 2).`   `/* Rosa Gana 100$ en 2 horas*/`

### – Realicemos las siguientes consultas:

- `salario(pEPe, 200)`      `/* ¿pEPe gana 200$? */`
- `salario(pepe, 200)`      `/* ¿pepe gana 200$? */`
- `salario(pepe, 50 * 4)`   `/* ¿pepe gana 50$ en 4 horas? */`
- `salario(rosa, 200)`      `/* ¿rosa gana 200$? */`
- `salario(rosa, 100 * 2)`   `/* ¿rosa gana 100$ en 2 horas */`
- `juan(500)`              `/* ¿Es posible preguntarlo así? */`

 <code>salario(pEPe, 200).</code>
false
 <code>salario(pepe, 200).</code>
true
 <code>salario(pepe, 50 * 4).</code>
false
 <code>salario(rosa, 200)</code>
false
 <code>salario(rosa, 100 * 2).</code>
true
 <code>juan(500).</code>
procedure `juan(A)' does not exist
?- <code>juan(500).</code>

# Prolog

## Ejercicio



### Ejercicio hechos poliádicos: Libros



**Libros**



**Escritor**



**Editorial**

- ¿Obtener todos los libros?
- ¿Obtener libros con año de publicación 2013?
- ¿Obtener libros con año de publicación mayor al 2000 y número de páginas inferior a 200?
- ¿Obtener libros escritos por Gabriel García y editorial oveja negra?

# Prolog

## Contenido



Operadores **con** y **sin** evaluación



Lógica de predicados vs PROLOG



Variables anónimas ‘\_’



Utilización del cut ‘!’



Recursividad

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	???	X is 10 + 2



# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
==	???	10 + 2 == 5 + 7

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
==	Igualdad	10 + 2 == 5 + 7

# Prolog

Operadores con evaluación  
Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
:=	Igualdad	10 + 2 := 5 + 7
=\=	???	10 + 2 =\= 5 + 8

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
==	Igualdad	10 + 2 == 5 + 7
==\=	Desigualdad	10 + 2 ==\= 5 + 8

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
==	Igualdad	10 + 2 == 5 + 7
!=	Desigualdad	10 + 2 != 5 + 8
>	???	11 * 3 > 3 ^ 2
<	???	2 ** 10 < 5 * 2
>=	???	99.0 >= 0
<=	???	-15 <= 15

# Prolog

## Operadores con evaluación

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
==	Igualdad	10 + 2 == 5 + 7
!=	Desigualdad	10 + 2 != 5 + 8
>	Mayor que	11 * 3 > 3 ^ 2
<	Menor que	2 ** 10 < 5 * 2
>=	Mayor o igual que	99.0 >= 0
<=	Menor o igual que	-15 <= 15

# Prolog

## Operadores sin evaluación

Operador	Significado	Ejemplo
=	Unificación	$X = 10 + 2$
==	Igualdad	$10 + 2 == 10 + 2$
\==	Desigualdad	$10 + 2 \backslash == 5 + 7$
@>	Mayor que	bananon @> bananin
@<	Menor que	parse @< tree
@>=	Mayor o igual que	ser @>= humano
@=<	Menor o igual que	raton @=< teclado

← ASCII

Una **estructura** es **menor** que **otra** si:

- Tiene menor número de argumentos.
- Por los argumentos en orden



# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$  ???

$10 + 4 == 7 + 7$  ???

$10 + 4 =\backslash= 7 + 7$  ???

$10 + 4 \backslash== 7 + 7$  ???

$10 + 4 >= 7 + 7$  ???

### Otros cálculos:

`read(Y), X is Y ** 2`

`read(Y), X is Y ** 2, Z is Y ^ 2, X =:= Z`

`read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y`

\*/

# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$	True
$10 + 4 == 7 + 7$	???
$10 + 4 =\backslash= 7 + 7$	???
$10 + 4 \backslash== 7 + 7$	???
$10 + 4 >= 7 + 7$	???

### Otros cálculos:

read(Y), X is Y \*\* 2  
read(Y), X is Y \*\* 2, Z is Y ^ 2, X =:= Z  
read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y

\*/

# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$

True

$10 + 4 == 7 + 7$

False

$10 + 4 =\backslash= 7 + 7$

???

$10 + 4 \backslash== 7 + 7$

???

$10 + 4 >= 7 + 7$

???

### Otros cálculos:

read(Y), X is Y \*\* 2

read(Y), X is Y \*\* 2, Z is Y ^ 2, X =:= Z

read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y

\*/

# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$

True

$10 + 4 == 7 + 7$

False

$10 + 4 =\backslash= 7 + 7$

False

$10 + 4 \backslash== 7 + 7$

???

$10 + 4 >= 7 + 7$

???

### Otros cálculos:

read(Y), X is Y \*\* 2

read(Y), X is Y \*\* 2, Z is Y ^ 2, X =:= Z

read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y

\*/

# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$

**True**

$10 + 4 == 7 + 7$

**False**

$10 + 4 =\backslash= 7 + 7$

**False**

$10 + 4 \backslash== 7 + 7$

**True**

$10 + 4 >= 7 + 7$

**???**

### Otros cálculos:

read(Y), X is Y \*\* 2

read(Y), X is Y \*\* 2, Z is Y ^ 2, X =:= Z

read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y

\*/

# Prolog

## Operadores

/\*

### Preguntas:

$10 + 4 =:= 7 + 7$

**True**

$10 + 4 == 7 + 7$

**False**

$10 + 4 =\backslash= 7 + 7$

**False**

$10 + 4 \backslash== 7 + 7$

**True**

$10 + 4 >= 7 + 7$

**True**

### Otros cálculos:

read(Y), X is Y \*\* 2

read(Y), X is Y \*\* 2, Z is Y ^ 2, X =:= Z

read(Z), is(X,log(Z)), is(Y,sin(Z)), X < Y

\*/

# Prolog

## Ejercicio # 1

1. Escribir una regla llamada **calcule\_prefijo(X, Y) :- ...**

que **lea** una **expresión matemática X** escrita en **notación polaca** {es decir prefija: **/(+ (5 ,1) ,3)**} por **consola** y devuelva el **resultado** del cálculo en **Result** y la **expresión en notación infija**.

Ejm.

```
/* CONSULTA */  
read(X), calcule_prefijo(X, Result).
```

```
/* lee /(+ (5 ,1) ,3) devuelve Result = 2, X = (5+1)/3
```

Probar con  $X = /(-(200,8),*(3, +(5 ,1)))$  ... Result = ???

# Prolog

## Ejercicio # 1

1. Escribir una regla llamada **calcule\_prefijo(X, Y) :- ...**

que **lea** una **expresión matemática X** escrita en **notación polaca** {es decir prefija: **/(+ (5 ,1) ,3)**} por **consola** y devuelva el **resultado** del cálculo en **Result** y la **expresión en notación infija**.

Ejm.

```
/* CONSULTA */  
read(X), calcular_prefijo(X, Result).
```

```
/* lee /(+ (5 ,1) ,3) devuelve Result = 2, X = (5+1)/3
```

Probar con  $X = /(-(200,8),*(3, +(5 ,1)))$  ... Result = ???

**Solución:** `calcular_prefijo(X, Resultado) :- is(Resultado, X).`



# Prolog

## Ejercicio # 2

2. Escribir una Regla llamada **edad( ... )** de tal forma que se **lea la variable Años** por consola y devuelva:

- Número de Lustros = ???
- Semanas = ???
- Días = ???
- Horas = ???
- Minutos = ???
- Segundos = ???

**vividos por esta persona.**

Probar con Años = 21 (nació el 3 de noviembre de 2000)

**Nota:** Lustros: 5 años; Año: 52 semanas

```
/*CONSULTA */  
read(Años),edad( ....)
```

# Prolog

## Ejercicio # 2

2. Escribir una Regla llamada **edad( ... )** de tal forma que se **lea la variable Años** por consola y devuelva:

- Número de Lustros = ???
- Semanas = ???
- Días = ???
- Horas = ???
- Minutos = ???
- Segundos = ???

**Solución:** edad(Años, Lustros, Semanas, Días, Horas, Minutos, Segundos) :-  
Lustros is Años/5, Semanas is Años\*52,  
Días is Semanas\*7, Horas is Días \* 24,  
Minutos is Horas\*60, Segundos is Minutos\*  
60.

**vividos por esta persona.**

Probar con Años = 21 (nació el 3 de noviembre de 2000)

**Nota:** Lustros: 5 años; Año: 52 semanas

```
/*CONSULTA */  
read(Años),edad( ....)
```

# Prolog

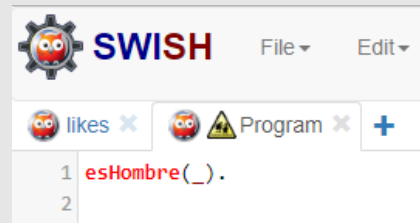
## Variables anónimas (\_)



La **variable anónima** hay que verla como una **variable libre**, sin ninguna cuantificación.

**Se usa** cuando estamos interesados en **saber si existe algún objeto que cumpla un objetivo**, de esta forma **se devuelve la sustitución que primero se encuentre**.

**esHombre(\_).**



?- gusta(X,maria). /\* ¿A quién le gusta María \*/

?- gusta(\_,maria). /\* ¿Hay alguien a quien le gusta María \*/

?- gusta(maria,\_). /\* ¿Hay alguien que le guste a María \*/

# Prolog

## Ejercicio # 3

3. Consideremos las siguientes reglas:

**“todos los dioses son inmortales” y “todos los humanos son mortales”**

En prolog será así:

???

???

La variable X tiene su **contexto o ámbito (*scope*)** en su **correspondiente cláusula o regla**.

# Prolog

## Ejercicio # 3

3. Consideremos las siguientes reglas:

**“todos los dioses son inmortales” y “todos los humanos son mortales”**

En prolog será así:

**inmortal(X):-dios(X).**

**mortal(X):-humano(X).**

La variable X tiene su **contexto o ámbito (*scope*)** en su **correspondiente cláusula o regla.**

# Prolog

## Contexto ó ambito

La variable X tiene su **contexto o ámbito (scope)** en su **correspondiente cláusula o regla**.

# Prolog

## Ejercicio # 3

Program

```
1 /* Reglas */
2 /* Todo dios es inmortal */
3 inmortal(X):-dios(X).
4 /* Todos Los humanos son mortales */
5 mortal(X):-humano(X).
6
7 /* Hechos */
8 dios(loki).
9 dios(thor).
10 humano(pepe).
11 humano(lola).
12 humano(thor).
13
```

mortal(loki)

false

mortal(thor)

true

inmortal(loki)

true

dios(loki)

true

?- dios(loki)

Program

```
1 /* Reglas */
2 /* Todo dios es inmortal */
3 inmortal(X):-dios(X).
4 /* Todos Los humanos son mortales */
5 mortal(X):-humano(X).
6
7 /* Hechos */
8 dios(loki).
9 dios(thor).
10 humano(pepe).
11 humano(lola).
12 humano(thor).
13
```

dios(Dioses)

Dioses = loki

Dioses = thor

inmortal(Y)

Y = loki

Y = thor

mortal(Hs)

Hs = pepe

Hs = lola

Hs = thor

?- mortal(Hs)

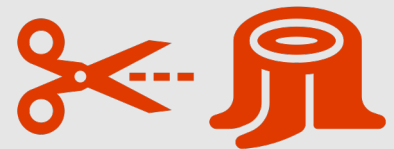
# Prolog

Cut !

El **cut** corta el árbol y garantiza la parada para encontrar una sola solución.

```
esHombre(pedro).  
esHombre(hugo).  
esDios(loki).  
esDios(zeus).
```

```
mortal(X) :- esHombre(X).  
inmortal(X) :- esDios(X),!
```





# Generalidades

## Referencias

- Bratko, Ivan. Prolog Programming for Artificial Intelligence (4th Edition) (International Computer Science Series) Aug 31, 2011
- Programming in Prolog: Using the ISO Standard Oct 4, 2013 by William F. Clocksin and Christopher S. Mellish
- Thinking as Computation (The MIT Press), Jan 6, 2012. by Hector J. Levesque
- Clause and Effect: Prolog Programming for the Working Programmer Apr 29, 2003. by William F. Clocksin
- Programming in Haskell Sep 12, 2016 by Graham Hutton
- Learn You a Haskell for Great Good!: A Beginner's Guide Apr 15, 2011 by Miran Lipovaca
- Learning Haskell Data Analysis, May 28, 2015 by James Church
- Advanced Computer Programming in Python, Advanced Computer Programming in Python Mar 22, 2017, by Karim Pichara and Christian Pieringer
- Functional Python Programming - Create Succinct and Expressive Implementations with Python Jan 31, 2015 by Steven Lott
- Functional Python Programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads, 2nd Edition Apr 13, 2018 by Steven F. Lott
- Building Web Applications with Python and Neo4j, Jul 16, 2015
- Learning Neo4j 3.x - Second Edition: Effective data modeling, performance tuning and data visualization techniques in Neo4j, Oct 20, 2017 by Jerome Baton and Rik Van Bruggen

Gracias 

*Universidad Nacional de Colombia*

---

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN