

/Tutorial Prolog

Julián Andrés Pereira
Jefferson Daniel Castro
Nicolás Andrés Caicedo
Gabriel Enrique Ramirez



/CONTENIDO

/01 /INTRODUCCIÓN

- > ¿Qué es?, tipo, propósito, aplicaciones, ventajas, desventajas e historia

/03 /INSTALACIÓN

- > Swi-prolog, pasos, version en linea

/02 /CONCEPTOS BÁSICOS

- > Hechos, consultas, reglas, expresiones, predicados, listas, árboles

/04 /EJEMPLOS

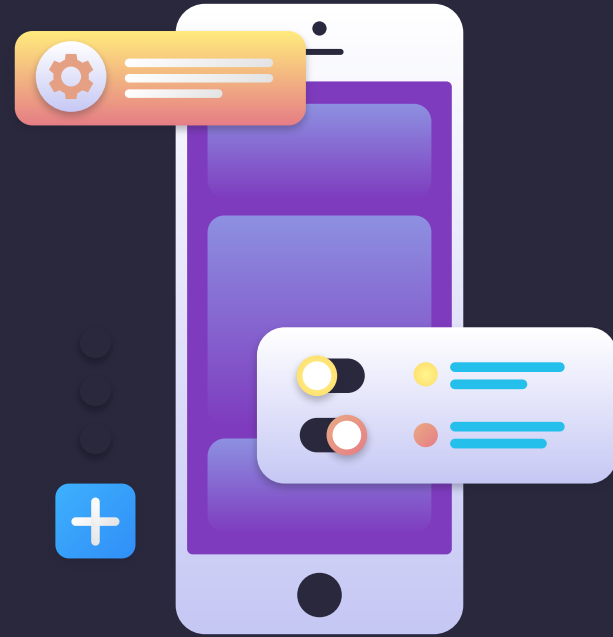
- > Simples, intermedios y uno avanzado con ejemplos en swi-prolog



/01

/INTRODUCCIÓN

¿Qué es?, tipo, propósito,
aplicaciones, ventajas,
desventajas e historia





¿Qué es PROLOG?

- Es un lenguaje de programación lógico.
- Busca expresar de forma lógica problemas complejos donde la programación imperativa ha fracasado

- Existen variaciones de este lenguaje con la capacidad de resolver sistemas de ecuaciones
- Su nombre viene del francés **P**rogrammation en **L**ogique (Programación Lógica).



¿Paradigma de Programación?

- Paradigma de programación lógico (declarativo)

- Fundamentado en la lógica de primer orden o de predicados



¿Paradigma de Programación?

- Se parte de un conjunto de **Hechos** como conocimiento base(predicados → Verdadero)

- Se definen **Reglas** para el cálculo de predicados
- Se realizan **Consultas** como forma de generar más conocimiento.

- **Hechos**

**SWISH**
Program ✕ +

```
1 connected(bond_street,oxford_circus,central).  
2 connected(oxford_circus,tottenham_court_road,central).  
3 connected(bond_street,green_park,jubilee).  
4 connected(green_park,charing_cross,jubilee).  
5 connected(green_park,piccadilly_circus,piccadilly).  
6 connected(piccadilly_circus,leicester_square,piccadilly).  
7 connected(green_park,oxford_circus,victoria).  
8 connected(oxford_circus,piccadilly_circus,bakerloo).  
9 connected(piccadilly_circus,charing_cross,bakerloo).  
10 connected(tottenham_court_road,leicester_square,northern).  
11 connected(leicester_square,charing_cross,northern).  
12
```

- **Reglas**

```
15 nearby(X,Y):-connected(X,Y,_L).  
16 nearby(X,Y):-connected(X,Z,L),connected(Z,Y,L).
```

- **Consultas**

```
?- nearby(tottenham_court_road,leicester_square).
```

[Examples](#)[History](#)[Solutions](#)[Loadable results](#)[Run!](#)



¿Paradigma de Programación?

- Para poder hacer la inferencia lógica, Prolog se basa en tres mecanismos: Unificación, Backtracking y Resolución SLD.
- Unificación: Trata de hacer match entre los funtores o átomos.

- Backtracking: Mecanismo con el que Prolog trata de buscar una solución en el árbol SLD.
- Resolución SLD: Mecanismo con el cual se construyen todas las posibles soluciones con base en los hechos y las reglas.



¿Proposito?

- Busca expresar de forma lógica problemas complejos donde la programación imperativa ha fracasado

- Existen variaciones de este lenguaje con la capacidad de resolver sistemas de ecuaciones



Aplicaciones

- Intelligent Database Retrieval
- Natural Language Understanding
- Specification Language

- Machine Learning
- Robot Planning
- Automation System
- Problem Solving



Ventajas y Desventajas

- Sintaxis fácil de Aprender
- Poco tiempo escribiendo código
- Fácil lectura y modificación

- Curva larga de aprendizaje de lógica matemática
- Una forma diferente de pensar a la usual



Ventajas y Desventajas

- Manejo fácilmente de estructuras de datos complejas
- Modelado de problemas más fácil y rápido.

- Puede ser ineficiente si no se estructura bien el problema



Un poco de Historia...

- Creado por Alain Colmerauer with Philippe Roussel alrededor de 1972.
- Fue diseñado para el procesamiento de lenguaje natural y el razonamiento lógico.

- La primera implementación fue un intérprete hecho en Fortran by Gerard Battani and Henri Meloni.
- David H. D. Warren construyó el primer compilador.

/02

/CONCEPTOS BÁSICOS

Hechos, consultas, reglas,
expresiones, backtracking,
predicados, listas, árboles.





PROLOG

Diseñado para representar y utilizar el conocimiento que se tiene sobre un determinado dominio. Los programas en Prolog **responden preguntas** sobre el tema del cual se tiene conocimiento.

La popularidad del lenguaje se debe a su **capacidad de deducción** y además es un lenguaje fácil de usar por su **semántica y sintaxis**. Solo busca relaciones entre los objetos creados, las variables y las listas, que son su estructura básica.



TÉRMINOS

Único elemento del lenguaje, está compuesto de un functor seguido de cero a **n** argumentos entre paréntesis y separados por comas

Ejemplo: `functor(2)`

- **Constantes**
 - Números
 - Átomos o funtores
- **Variables**
- **Estructuras**

/TÉRMINOS

/CONSTANTES

Números: Representan valores reales y enteros para realizar operaciones aritméticas

Functores: Utilizados para nombrar objetos, propiedades o relaciones. Deben empezar en minúscula

```
8 isNumber(3e-24).  
9 isNumber(2e2).  
10 isNumber(.87767).
```

Syntax error: Operator expected

```
4 isAtom(atOmMm12345).  
5 isAtom(i_am_AToMm_23).
```

/VARIABLES

Se representan mediante cadenas formadas por letras, dígitos y el símbolo de subrayado, también con una letra mayúscula en su inicio.

```
1 isVariable(Var).  
2 isVariable(_var).  
3 isVariable(Va2323sdf).  
4 isVariable(V_a_w_3_).  
5 isVariable(_).  
6
```

/ESTRUCTURAS

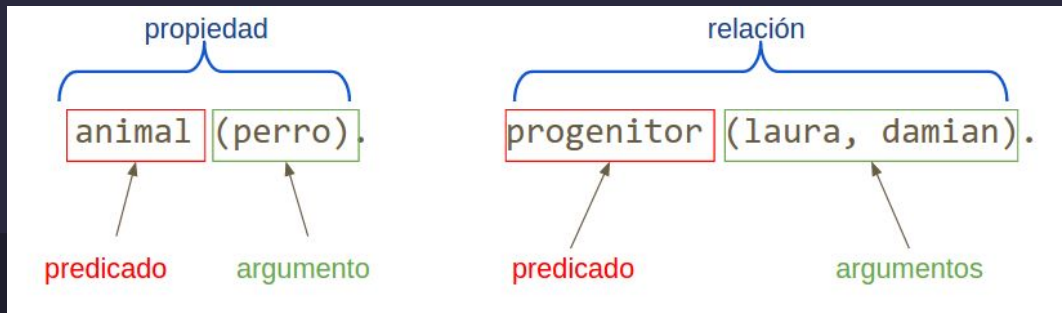
Son términos compuestos por otros términos. Se construyen mediante un símbolo de función, denominado functor, seguido entre paréntesis, por un conjunto de términos separados por comas.

```
1 name(arg1).  
2 Functor Argumentos  
3 name(arg1, arg2, arg3).  
4 Functor Argumentos
```

HECHOS

Cláusula sin cuerpo que muestra una **relación** explícita entre **objetos** y declaran los valores que son verdaderos para un predicado. Su estructura se compone de un **predicado** y un **argumento** u **objeto**.

- El predicado debe iniciar con minúscula.
- Los argumentos se escriben separados por comas, en minúscula y encerrados entre paréntesis.
- Todos los hechos deben terminar en punto.



CONSULTAS

Un `conjunto de hechos` genera una `base de datos` a la cual se puede realizar una serie de preguntas para extraer el `conocimiento generado` por esta. Comienzan con un signo de interrogación seguido de un guión `?-` y terminan en punto. Ante una consulta, Prolog intenta hacer un matching sobre la `base de conocimiento`.

Prolog devuelve la `primera ocurrencia`, se obtienen las demás insertando el token `;`. Las respuestas a una consulta puede ser `true`, `false` o los elementos que pueden tomar una `variable definida`.

CONSULTAS

```
progenitor(clara,jose).      %Hecho 1
progenitor(tomas, jose).    %Hecho 2
progenitor(tomas,isabel).   %Hecho 3
progenitor(jose, ana).      %Hecho 4
progenitor(jose, patricia). %Hecho 5
progenitor(patricia,jaime). %Hecho 6
```

```
?- progenitor(patricia,jaime).
```

Singleton variables: [X]

true

```
?- progenitor(X,jaime).
```

Singleton variables: [X]

X = patricia

```
?- progenitor(tomas,X), progenitor(X,Y), progenitor(Y,Z).
```

Singleton variables: [X]

X = jose,
Y = patricia,
Z = jaime

REGLAS

Cuando la **verdad de un hecho** depende de la verdad de otro hecho o de un grupo de hechos se usa una **regla**. Declaran las condiciones para que un **predicado sea cierto**, con una implicación que pueden relacionar hechos para dar los valores de verdad a un predicado. Una regla está compuesta por una **cabeza y un cuerpo**. El cuerpo puede estar formado por varios hechos y objetivos. Su sintaxis general es:

```
cabeza :- objetivo1, objetivo2, ..., objetivon.
```

/REGLAS

/CONJUNCIONES

- Emplea el operador lógico AND
- Se utiliza la coma (,)

```
tia(X,Y):-hermana(X,Z),padre(Z,Y).
```

Regla

AND

/DISYUNCIONES

- Emplea el operador lógico OR
- Se utiliza el punto y coma (;)

```
hijo(X,Y):-padre(Y,X);madre(Y,X).
```

Regla

OR



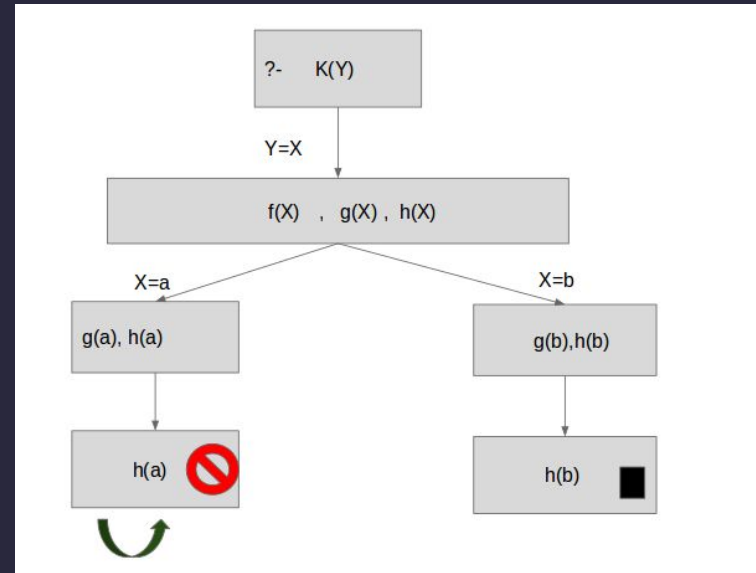
REGLAS RECURSIVAS

```
predecesor(X,Y):-progenitor(X,Y).                %padre
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,Y). %abuelo
predecesor(X,Y):-progenitor(X,Z), progenitor(Z,V), progenitor(V,Y). %bisabuelo
```

```
predecesor(X,Y):-progenitor(X,Y).                %caso base
predecesor(X,Y):-progenitor(X,Z), predecesor(Z,Y). %caso recursivo
```


BACKTRACKING

Prolog siempre está consultando su **base de datos**, para verificar que hechos son verdaderos y que nos permitirá la construcción de las posibles reglas. Para problemas de recursión, prolog se devuelve hasta que encuentra que un hecho base es verdadero y de ahí construye la respuesta.



/EXPRESIONES

| Expresión | Operación |
|---------------------|---|
| $X+Y$ | suma de X e Y |
| $X-Y$ | X menos Y |
| $X*Y$ | producto de X por Y |
| X/Y | cociente real de la división de X por Y |
| $X//Y$ | cociente entero de la división de X por Y |
| X^Y | potencia entera de X a la Y |
| $X^{**}Y$ | potencia real de X a la Y |
| $X \bmod Y$ | resto de la división entera de X por Y |
| $\text{abs}(X)$ | valor absoluto de X |
| $\text{acos}(X)$ | arco coseno de X |
| $\text{asen}(X)$ | arco seno de X |
| $\text{atan}(X)$ | arco tangente de X |
| $\text{cos}(X)$ | coseno de X |
| $\text{exp}(X)$ | exponencial de X; $[e^X]$ |
| $\text{ln}(X)$ | logaritmo neperiano de X |
| $\text{log}(X)$ | logaritmo en base 2 de X |
| $\text{sin}(X)$ | seno de X |
| $\text{sqrt}(X)$ | raíz cuadrada de X |
| $\text{tan}(X)$ | tangente de X |
| $\text{round}(X,N)$ | redondeo del real X con N decimales |

| Expresión | Operación |
|------------|---|
| $X < Y$ | cierto si el valor numérico de X es menor que el de Y |
| $X > Y$ | cierto si el valor numérico de X es mayor que el de Y |
| $X \leq Y$ | cierto si el valor numérico de X es menor o igual que el de Y |
| $X \geq Y$ | cierto si el valor numérico de X es mayor o igual que el de Y |

| Expresión | Operación |
|-------------------|--|
| $X == Y$ | la expresión X es igual que la expresión Y |
| $X != Y$ | la expresión X es distinta que la expresión Y |
| $X @ < Y$ | la expresión X es menor que la expresión Y |
| $X @ > Y$ | la expresión X es mayor que la expresión Y |
| $X @ \leq Y$ | la expresión X es menor o igual que la expresión Y |
| $X @ \geq Y$ | la expresión X es mayor o igual que la expresión Y |
| $X \text{ is } Y$ | Si Y es una expresión aritmética, ésta se evalúa y el resultado se intenta unificar con X. |



/ARITMETICAS

/COMP_TERMINOS

/COMP_EXPRESIONES

PREDICADOS

Conjunto de predicados que permiten determinar el tipo de términos que estamos usando

| Predicado | Función |
|-----------|---|
| var | El objetivo var(X) se cumple si X es una variable no instanciada. |
| nonvar | El objetivo nonvar(X) se cumple si X es una variable instanciada |
| atom | El objetivo atom(X) se cumple si X representa un átomo. |
| integer | El objetivo integer(X) se cumple si X representa un número entero. |
| atomic | El objetivo atomic(X) se cumple si X representa un entero o un átomo. |

Los siguientes son predicados predefinidos que permiten controlar otros predicados.

PREDICADOS

Conjunto de predicados que permiten **controlar** otros predicados

| Predicado | Función |
|-----------|--|
| !(cut) | Fuerza al sistema a mantener ciertas elecciones que ha realizado. |
| true | Este objetivo siempre se cumple. |
| fail | Este objetivo siempre fracasa. |
| not | El objetivo not(X) se cumple si fracasa el intento de satisfacer X. El objetivo not(X) fracasa si el intento de satisfacer X tiene éxito. Es similar a la negación en la lógica de predicados. |
| repeat | forma auxiliar para generar soluciones múltiples mediante el mecanismo de reevaluación. |
| call | Se cumple si tiene éxito el intento de satisfacer X. |
| ; | Especifica una disyunción de objetivos |
| , | Especifica una conjunción de objetivos |

PREDICADOS

Conjunto de predicados de
lectura y escritura

| Predicado | Función |
|-----------|--|
| write | escribe el término X en la consola de salida. |
| nl | genera una nueva línea en la consola de salida. |
| read | lee el siguiente término en la consola de entrada. |
| display | funciona exactamente igual que write, excepto que pasa por alto las declaraciones de operadores. |

LISTAS

Es una secuencia ordenada de elementos que puede ser de cualquier longitud.

En prolog están formadas por cabeza y cola, se representan como una serie de elementos separados por `com` y entre corchetes rectangulares.

| Lista | Cabeza | Cola |
|--------------------------|--------------------|------------------------|
| <code>[a,b,c,d]</code> | <code>a</code> | <code>[b,c,d]</code> |
| <code>[a]</code> | <code>a</code> | <code>[]</code> |
| <code>[]</code> | no tiene | no tiene |
| <code>[[a,b],c]</code> | <code>[a,b]</code> | <code>[c]</code> |
| <code>[a,[b,c]]</code> | <code>a</code> | <code>[[b,c]]</code> |
| <code>[a,b,[c,d]]</code> | <code>a</code> | <code>[b,[c,d]]</code> |

/LISTAS

/unificación

En Prolog se puede unificar una lista con otra.

```
[X,Y,Z]= [a,b,c]
```

```
X = a
```

```
Y = b
```

```
Z = c
```

Una variable no instanciada, se puede unificar con cualquier objeto.

```
X= [a,b,c]
```

Para unificar una variable con una lista pero separando su cabeza y cola se debe hacer de la forma [A | B]

```
[a,b,c] = [Cabeza|Cola]
```

```
Cabeza = a
```

```
Cola = [b,c]
```

/LISTAS

/recursion

Criterios de terminación en Prolog.

Cuando la lista
está vacía.

```
/* Regla de terminación */  
predicado([ ]):- procesar([ ]).  
/* Regla recursiva */  
predicado([Cabeza | Cola]):- procesar(Cabeza), predicado(Cola).
```

Cuando un elemento
es encontrado.

```
/* Regla de terminación */  
predicado(Cabeza, [Cabeza | Cola]):- procesar algo.  
/* Regla recursiva */  
predicado(X, [Cabeza | Cola]):- procesar algo, predicado(X, Cola).
```

Cuando una posición
es encontrada.

```
/* Regla de terminación */  
predicado(1,Cabeza, [Cabeza | Cola]):- procesar algo.  
/* Regla recursiva */  
predicado(P, X, [ | L]):- P1=P-1, predicado(P1,X, L).
```


ÁRBOLES

Un árbol binario es una estructura que contiene un nodo padre y dos hijos.

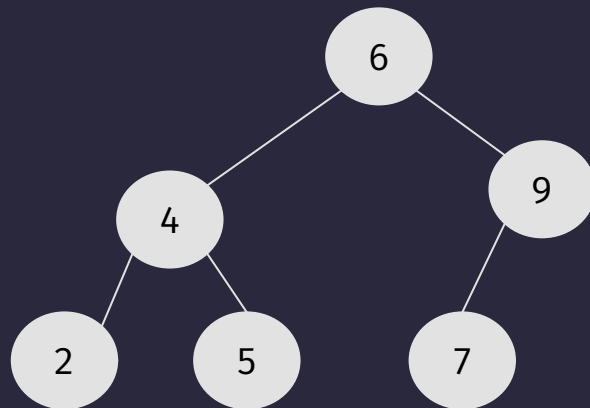
También definido como el elemento raíz cuyos hijos son árboles.

Como un árbol es una estructura recursiva, se necesita de un caso base y un caso recursivo.

```
binary_tree(void).                %caso base
binary_tree(t(K,L,R)) :-          %caso recursivo
    binary_tree(L),
    binary_tree(R).
```

/ÁRBOLES

/representación



```
tree1(t(6, t(4, t(2, nil, nil), t(5, nil, nil)), t(9, t(7, nil, nil), nil))).
```

/ÁRBOLES /reglas

También se pueden definir las reglas de los posibles recorridos de un árbol.

Inorder:

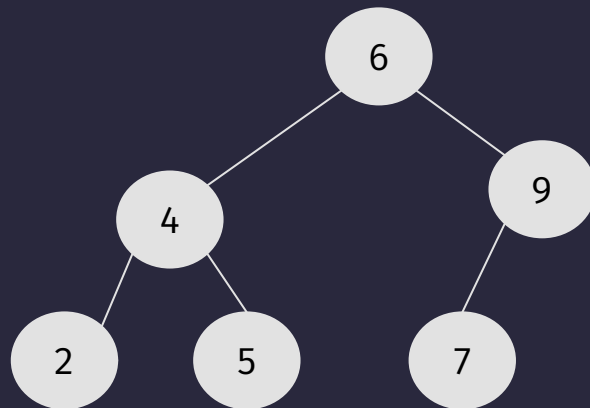
```
inorder(nil, []).  
inorder(t(K,L,R), List):-inorder(L,LL),  
                           inorder(R, LR),  
                           append(LL, [K|LR],List).
```

Preorder:

```
preorder(nil, []).  
preorder(t(K,L,R), List):-preorder(L,LL),  
                           preorder(R, LR),  
                           append([K|LL], LR, List).
```

/ÁRBOLES /otra representación

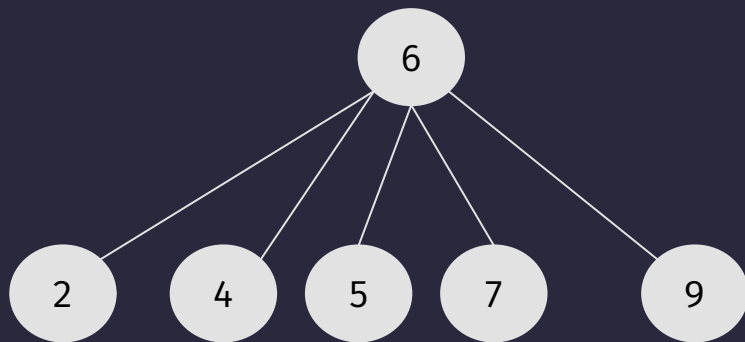
[left, data, right]



[[[],2,[],4,[],5,[]],6,[[],7,[],9,[]]]

/ÁRBOLES /N-arios

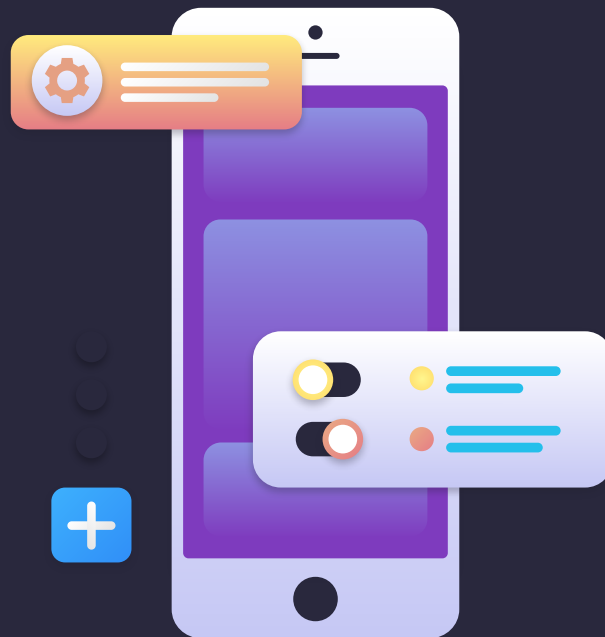
Para un árbol n-ario, se utilizan listas, en las cuales la cabeza será el nodo padre y todos los elementos de la cola serán el conjunto de nodos hijos.



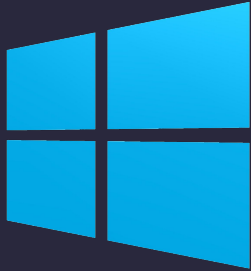
/03

/INSTALACIÓN

SWI-PROLOG, PASOS, VERSION EN
LINEA



SWI-PROLOG EN LOCAL



Windows



MacOS

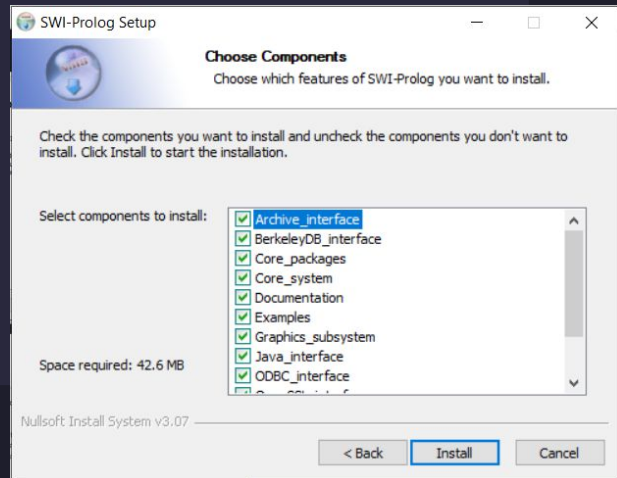
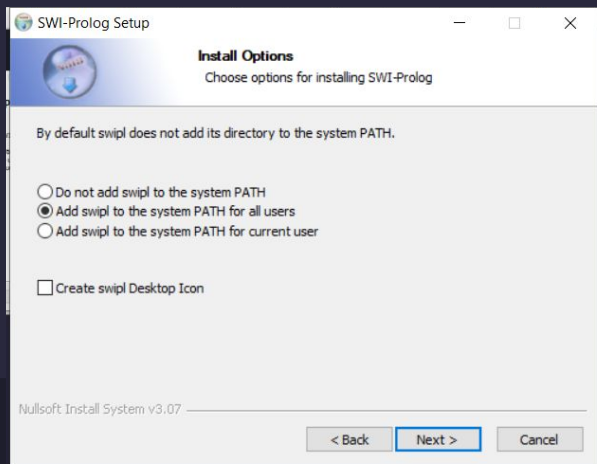
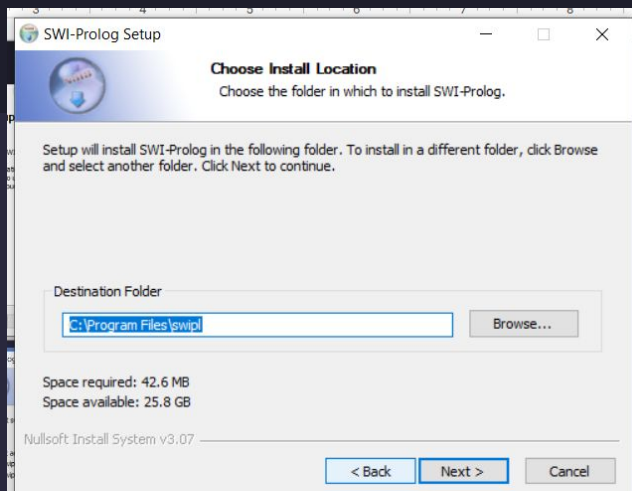
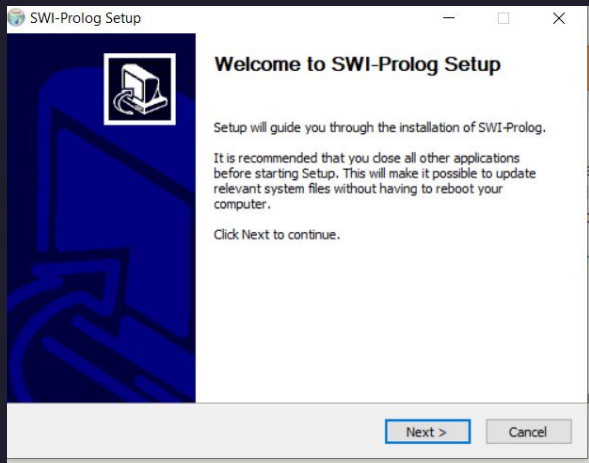


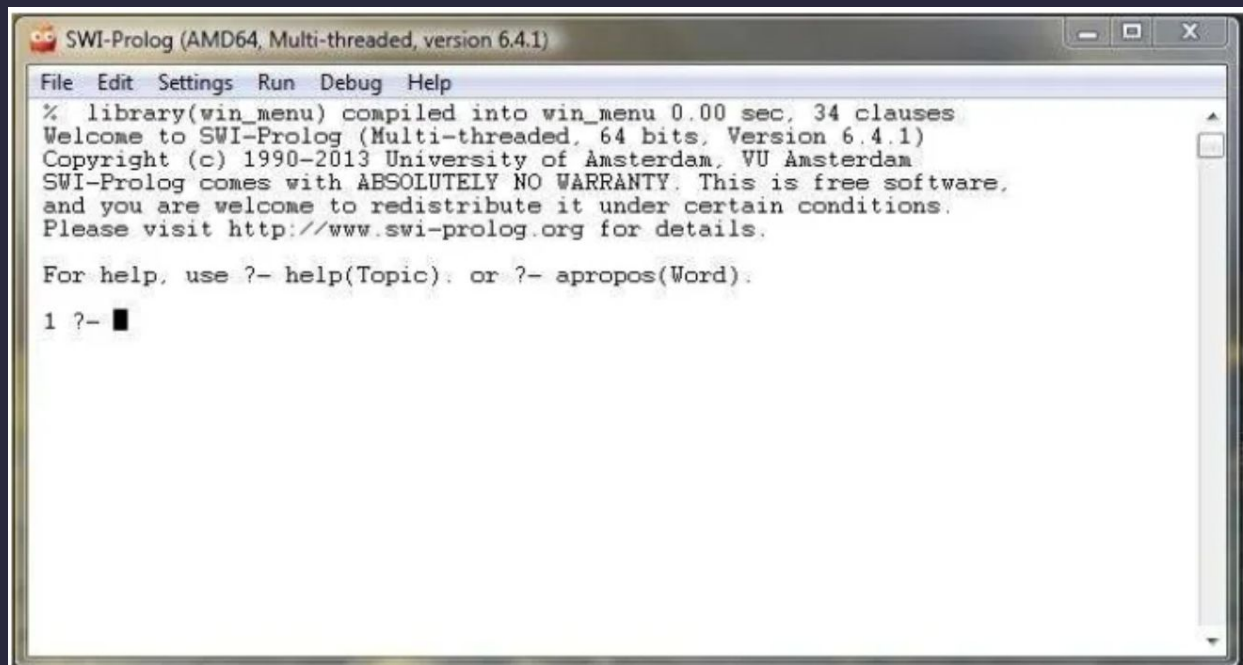
Linux



Termux

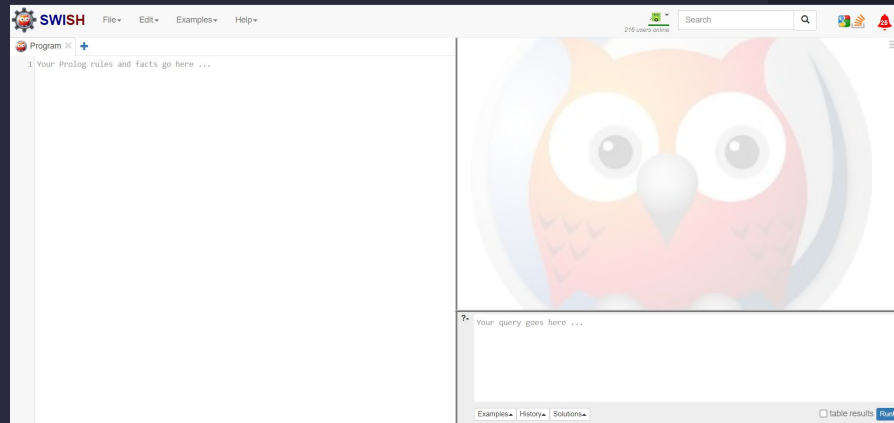
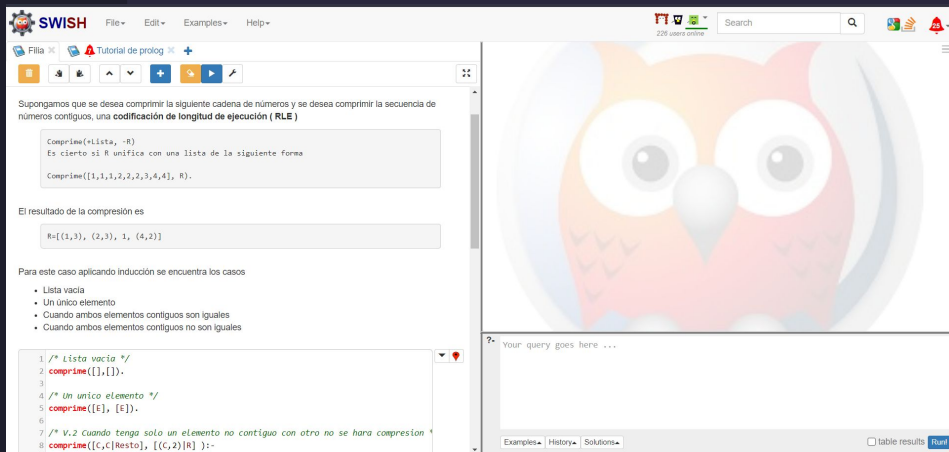
<https://www.swi-prolog.org/download/stable>



A screenshot of a SWI-Prolog window. The title bar reads "SWI-Prolog (AMD64, Multi-threaded, version 6.4.1)". The menu bar contains "File", "Edit", "Settings", "Run", "Debug", and "Help". The main text area displays the following:

```
% library(win_menu) compiled into win_menu 0.00 sec, 34 clauses  
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.4.1)  
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.  
Please visit http://www.swi-prolog.org for details.  
  
For help, use ?- help(Topic): or ?- apropos(Word).  
  
1 ?- █
```

PROLOG EN LINEA

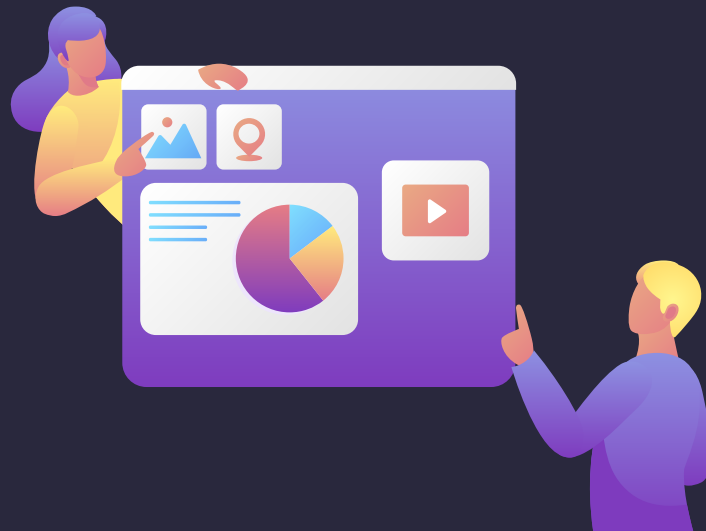


<https://swish.swi-prolog.org>

/04

/EJEMPLOS

`https://swish.swi-prolog.org
/p/PL2022-Practica.pl`



Ejemplo 1:

Cadena alimenticia

```
animal1(leon).  
animal1(coyote).  
animal1(zebra).  
carnivoro(leon).  
carnivoro(coyote).  
masDebil(coyote,leon).  
herbivoro(zebra).  
plantaComestible(hojas).  
  
come(A,B) :-  
    carnivoro(A), animal1(B), masDebil(B, A);  
    herbivoro(A), plantaComestible(B).
```

Ejemplo 2:

A Elena no le gustan las
serpientes

```
animal(snoopy).  
animal(lamia).  
serpiente(lamia).  
gusta1(elena, X):- serpiente(X), !, fail.  
gusta1(elena, X):-animal(X).
```

Ejemplo 3:

Maximo comun divisor

```
mcd(X,X,X).  
mcd(X,Y,Z) :-  
    X < Y,  
    Y1 is Y-X,  
    mcd(X,Y1,Z).  
mcd(X,Y,Z) :-  
    X > Y,  
    mcd(Y,X,Z).
```

Ejemplo 4:

Sucesión de fibonacci

```
fibonacci(0,0).  
fibonacci(1,1).  
fibonacci(N,X) :-  
    N > 1,  
    N1 is N-1,  
    fibonacci(N1,X1),  
    N2 is N-2,  
    fibonacci(N2,X2),  
    X is X1+X2.
```

Ejemplo 5:

Sistema que define qué animal es el que cumple las características dadas por un usuario.

```
empezar:- hypothesis(Animal),  
    write('El animal es: '),  
    write(Animal),  
    nl,  
    undo.
```



```
hypothesis(jaguar):- jaguar, !.  
hypothesis(tigre):- tigre, !.  
hypothesis(jirafa):- jirafa, !.  
hypothesis(zebra):- zebra, !.  
hypothesis(gato):- gato, !.  
hypothesis("Animal desconocido").
```

```
jaguar:-  
    verify("es mamifero"),  
    verify("es carnivoro"),  
    verify("es color ambar"),  
    verify("tiene manchas oscuras").  
  
tigre :-  
    verify("es mamifero"),  
    verify("es carnivoro"),  
    verify("es color ambar"),  
    verify("tiene rayas negras").  
  
jirafa :-  
    verify("es mamifero"),  
    verify("tiene cuello largo"),  
    verify("tiene manchas oscuras").  
  
zebra :-  
    verify("es mamifero"),  
    verify("tiene rayas negras").  
  
gato :-  
    verify("es mamifero").
```

```
ask(Question) :-  
    write('El animal '),  
    write(Question),  
    write('? '),  
    read(Response),  
    nl,  
    ( (Response == yes ; Response == y ; Response == si)  
    ->  
        assert(yes(Question)) ;  
        assert(no(Question)), fail).  
  
:- dynamic yes/1,no/1.
```

```
verify(S) :- (yes(S) -> true ;  
              (no(S) -> fail ;  
               ask(S))).  
  
/* eliminar las clausulas creadas */  
undo :- retract(yes(_)),fail.  
undo :- retract(no(_)),fail.  
undo.
```

/REFERENCIAS

- <https://www.swi-prolog.org/>
- <https://www.dsi.fceia.unr.edu.ar/downloads/IIA/recursos/Tutorial%20de%20%20Prolog.pdf>
- <https://ferestrepoca.github.io/paradigmas-de-programacion/prologica/tutoriales/prolog-gh-pages/index.html#22-programa>
- <http://fcqi.tij.uabc.mx/usuarios/ardiaz/conceptos.html>
- <https://swish.swi-prolog.org/p/PL2019-I.swinb>
- <https://elvex.ugr.es/decsai/intelligent/workbook/ai/PROLOG.pdf>

/¡Gracias!

/¿Preguntas?

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

> Please keep this slide for attribution