<!--Universidad Nacional-->

Programación lógica {

<Por="
- Michael Daniels Oviedo
- Nestor Steven Piraquive Garzon
- Julian David Rodriguez Fernandez
- Esteban Barrera Sanabria
"/>

}

Contenidos

PARADIGMAS DE PROGRAMACIÓN {

Un paradigma de programación es una forma o estilo de pensar y organizar el código al momento de desarrollar software. Es como un “modelo” o conjunto de reglas y principios que guían cómo escribir programas.

Si se quiere resolver un problema (por ejemplo, hacer un programa). Un paradigma sería la “estrategia” que eliges para hacerlo.

}

PARADIGMAS DE PROGRAMACIÓN {

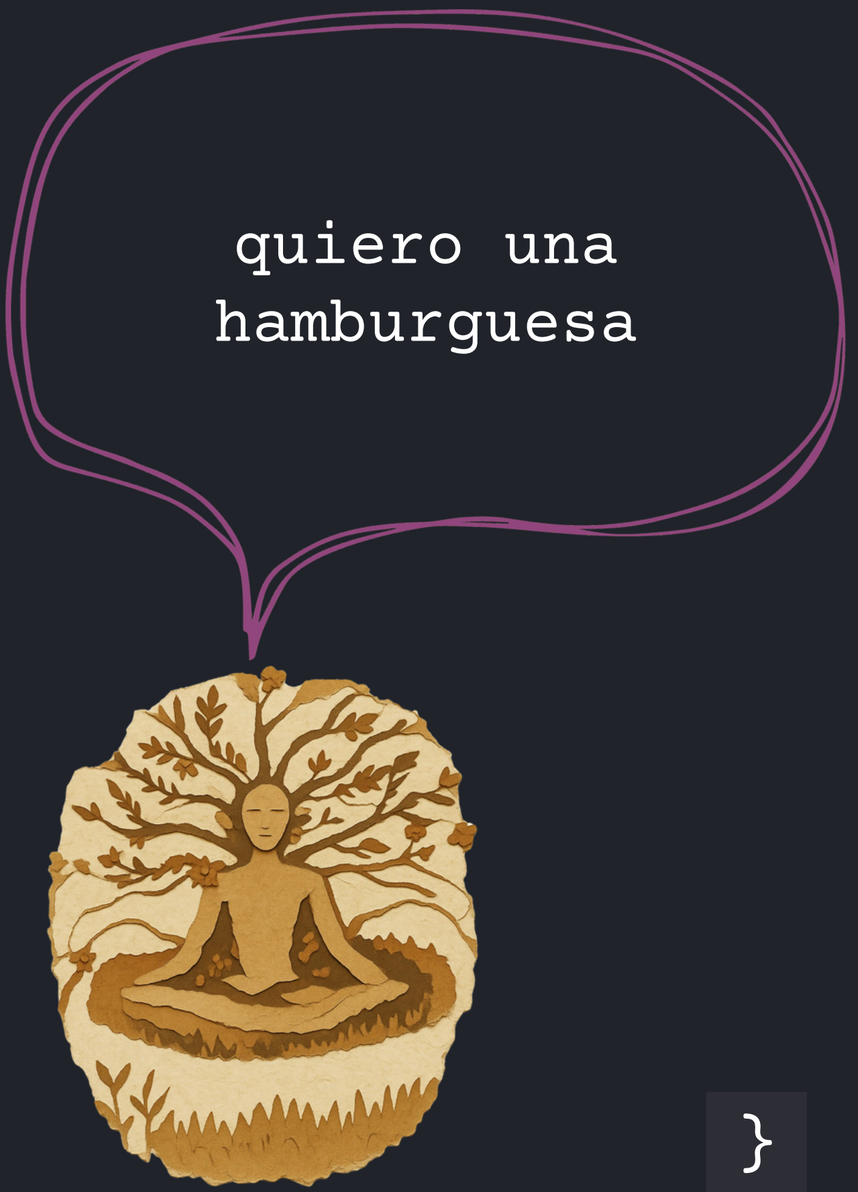
Existen dos grandes grupos:

Programación imperativa

¿Cómo hacer las cosas?

Programación declarativa

¿Qué cosas hacer?



}

PARADIGMAS DE PROGRAMACIÓN {

Existen dos grandes grupos:

Programación imperativa

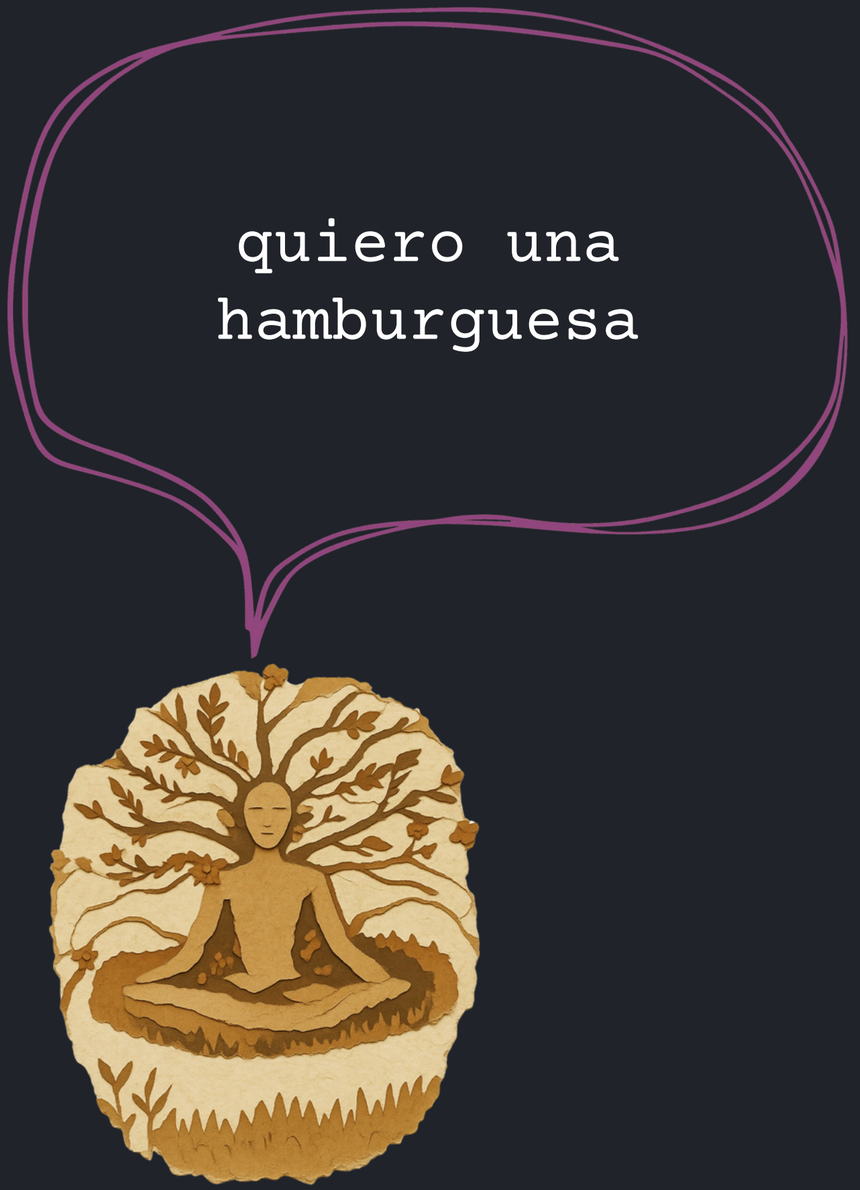
¿Cómo hacer las cosas?

> Sacar dos rodajas de pan
> Sacar carne y queso
> Poner una rodaja de pan
> Poner una rodaja de queso sobre la carne y despues pan ...

Programación declarativa

¿Qué cosas hacer?

> Una hamburguesa tiene un pan, encima va una lechuga, luego una rodaja de tomate, otra de queso, un pedazo de carne y finalmente otra rodaja de pan.
> Haz un hamburguesa.



}

PARADIGMAS DE PROGRAMACIÓN {

Existen dos grandes grupos:

Programación imperativa

> Orientado a Objetos.
> Por procedimientos.
> Programación Estructurada.

Programación declarativa

> Programación Funcional.
> Programación Lógica.
> Programación Reactiva.

Algunos paradigmas

}

FILOSOFÍA PROGRAMACIÓN LÓGICA {

> Paradigma declarativo

> Se definen reglas lógicas para modelar un problema.

> Aplicación de reglas de la lógica para inferir conclusiones o soluciones a partir de datos.

1
Definir reglas Logicas

↓

2
Consultar

↓

3
Soluciones

}

FILOSOFÍA PROGRAMACIÓN LÓGICA {

¿Como se definen reglas lógicas para modelar problema?

Hechos

Reglas

Predicados

}

Conceptos clave {

01

Hechos (Facts)

Los hechos representan información verdadera dentro del sistema.

```
parent_child(charles,william)
```

02

Reglas (Rules)

Las reglas permiten crear nuevas conclusiones a partir de hechos ya existentes.

```
can_drive(Person) :-
 has_license(Person),
 is_adult(Person).
```

03

Consultas o Metas

Las consultas son preguntas hechas al sistema lógico.

```
?- parent_child(X, william).
X = charles
```

04

Predicados

Un predicado representa una relación entre elementos.

```
mother_child(X, Y)
father_child(X, Y)
grandparent_child(X, Y)
```

}

Conceptos clave {

05

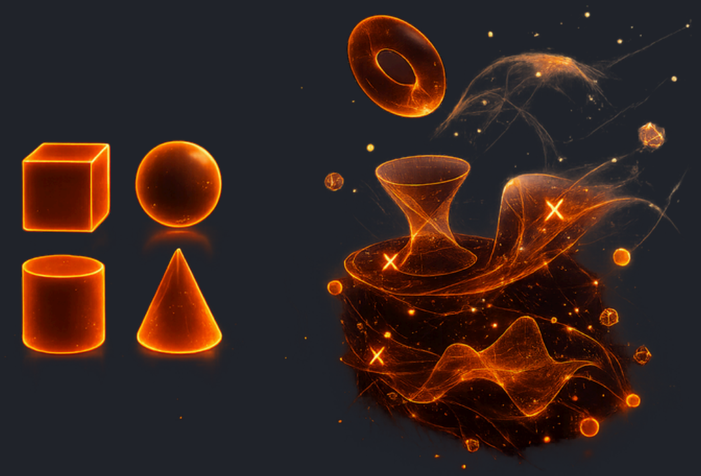
Constantes y variables

Representan valores específicos.

```
elizabeth
charles
william
```

Representan valores desconocidos.

```
X
Y
Z
```



06

Inferencia Lógica

La inferencia es el proceso mediante el cual el sistema deduce nuevas verdades.

```
dog(firulais).
animal(X) :- dog(X).

animal(firulais).
```

}

Conceptos clave {

07

Backtracking

Es el mecanismo mediante el cual el sistema prueba alternativas cuando una solución falla.

```
father_child(charles, william).
father_child(charles, harry).
?- father_child(charles, X).
```

```
X = william;
X = harry.
```

08

Unificación

Es el proceso de hacer coincidir términos lógicos.

```
color(carro, rojo)
?- color(carro, X).
```

```
color(carro, X).
```

```
X = rojo
```

}

Conceptos clave {

09

Declaratividad

La programación lógica es declarativa.

```
FUNCIÓN es_ancestro(objetivo, persona)
    padre = obtener_padre(persona)

    SI no hay padre:
        RETORNAR Falso

    SI padre == objetivo:
        RETORNAR Verdadero
    SINO:
        RETORNAR es_ancestro(objetivo, padre)
FIN FUNCIÓN
```

10

Resolución

Es el método lógico que utiliza Prolog para demostrar si una consulta es verdadera.

```
parent_child(charles, william).
parent_child(elizabeth, charles).


?- grandparent_child(elizabeth, william).


grandparent_child(X,Y) :-
    parent_child(X,Z),
    parent_child(Z,Y).
```

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

}

Ventajas y desventajas

Ventajas

- Código declarativo y fácil de entender
- Excelente para inteligencia artificial
- Inferencia automática
- Backtracking automático
- Muy útil para problemas simbólicos

Desventajas

- Menor rendimiento en algunos problemas
- Difícil de aprender al inicio
- Puede generar recursiones infinitas
- Depuración compleja
- Menor popularidad industrial
- Control limitado de ejecución

}

Lenguajes de programación {

Cada lenguaje representa una evolución del paradigma lógico, adaptándose a necesidades específicas de la industria y la academia.

}

Soufflé

2016 | Datalog | Análisis estático de código

PROLOG

1972 | Predicados | IA clásica / NLP

CURRY

1995 | Lógico-funcional | Programacion hibrida

Prolog

- Creado en 1972 por Alain Colmerauer y Robert Kowalski
- Su nombre viene de PROgramming in LOGic
- Es el lenguaje lógico más usado e influyente de la historia
- Implementación más popular: SWI-Prolog (gratuita y activa hasta hoy)
- Usado en:
  - Procesamiento de lenguaje natural
  - Sistemas expertos médicos
  - Parsing de lenguajes formales
  - Inteligencia artificial clásica

```prolog
% --- HECHOS ---
padre(tom, bob).
padre(tom, liz).
padre(bob, ann).
padre(bob, pat).

% --- REGLAS ---
abuelo(X, Y) :-
    padre(X, Z),
    padre(Z, Y).

hermano(X, Y) :-
    padre(Z, X),
    padre(Z, Y),
    X \= Y.

ancestro(X, Y) :- padre(X, Y).
ancestro(X, Y) :-
    padre(X, Z),
    ancestro(Z, Y).
```

```prolog
?- %Cual es el abuelo de tom ?
abuelo(tom, X).
```

```
X = ann
```

```prolog
?- %Quien es el hermano de ann ?
hermano(ann, X).
```

```
X = pat
```

```prolog
?- %Quienes son los ancestros de ann ?
ancestro(X, ann).
```

```
X = bob
X = tom
```

SOUFFLE

Soufflé

- Creado por Oracle Labs en 2016
- Adoptado por Meta y Amazon para análisis de seguridad
- Es Datalog moderno con ejecución en paralelo
- Procesa millones de hechos en segundos
- Se usa para encontrar vulnerabilidades en código antes de que lleguen a producción

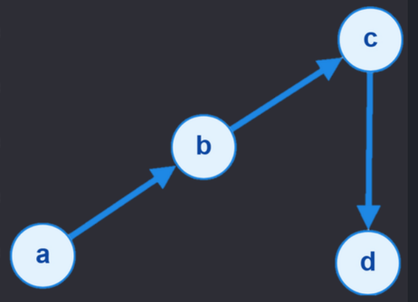
```
// Encontrar todos los nodos
// alcanzables en un grafo
.decl arista(x:symbol, y:symbol)
.decl alcanzable(x:symbol, y:symbol)
.output alcanzable

arista("a", "b").
arista("b", "c").
arista("c", "d").

alcanzable(x, y) :- arista(x, y).
alcanzable(x, z) :- alcanzable(x, y),
                    arista(y, z).
```

```
# RESULTADO

alcanzable("a","b")
alcanzable("a","c")
alcanzable("a","d")
alcanzable("b","c")
alcanzable("b","d")
alcanzable("c","d")
```



CURRRY

- Creado en 1995, nombrado en honor a Haskell Curry
- Combina programación funcional + lógica en un solo lenguaje
- Basado en Haskell pero con capacidades lógicas
- Su superpoder: funciones inversas — le das el resultado y él encuentra la entrada
- Implementaciones: KiCS2, PAKCS

```
1   -- Lado funcional: función normal
2   doble :: Int -> Int
3   doble x = x * 2
4
5   -- Lado lógico: función INVERSA
6   -- ¿Qué x hace que doble x = n?
7   mitad :: Int -> Int
8   mitad n | doble x =:= n = x
9     where x free
10
11  -- Búsqueda no determinista
12  unElemento :: [a] -> a
13  unElemento (x:_)  = x
14  unElemento (_:xs) = unElemento xs
15
16  -- > doble 5            => 10
17  -- > mitad 10           => 5
18  -- > unElemento [1,2,3] => 1, 2 o 3
```

miniKanren

- Creado por William Byrd y Daniel Friedman en 2005
- No es un lenguaje independiente, se embebe dentro de otros lenguajes
- Existe en Python, JavaScript, Scheme, Haskell
- Su superpoder: correr programas al revés
- Si tienes una función que suma, miniKanren puede decirte qué números suman un resultado dado

```python
from kanren import run, eq, var, membero

x = var()
y = var()
q = var()

# Unificación: deduce y comparando
# las listas posición por posición
resultado = run(1, y, eq([1, 2, y], [1, 2, 3]))
print(resultado[0])
# 3

# Encadenar variables:
# x = y, y = 42, entonces x = ?
resultado = run(1, x, eq(x, y), eq(y, 42))
print(resultado[0])
# 42

# Intersección: q debe cumplir
# ambas condiciones al mismo tiempo
pares   = [2, 4, 6, 8, 10, 12]
mayores = [6, 7, 8, 9, 10, 11, 12]
resultado = run(10, q,
    membero(q, pares),
    membero(q, mayores))
print(resultado)
# (6, 8, 10, 12)
```

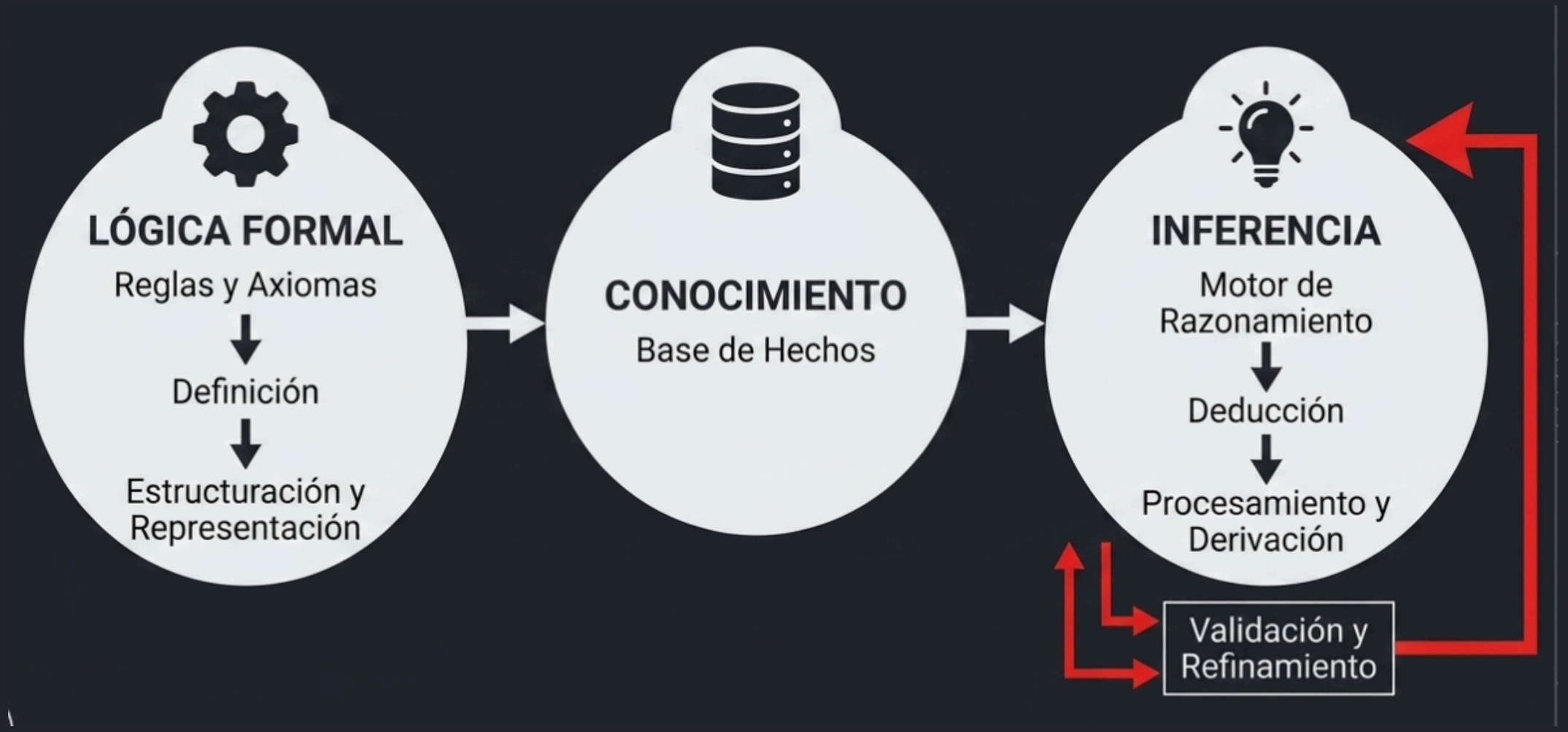
Otros Lenguajes {

LENGUAJE	AÑO	CARACTERISTICA PRINIPAL
Datalog	**1977**	**Subconjunto de Prolog, base de Souffle**
Mercury	1995	Prolog con tipos estrictos, más seguro
Twelf	1999	Verificación formal de pruebas matematicas
ASP/CLINGO	1999	Resolver problemos de optimización complejos
Oz/Mozart	1991	Multiparadigma, lógico + concurrente

}

Aplicaciones de este paradigma {

IA y sistemas expertos



}

Aplicaciones de este paradigma {

Sistemas Expertos

EJEMPLO LÓGICO

```
animal(perro) :-
    tiene_pelaje,
    dice(guau).

animal(gato) :-
    tiene_pelaje,
    dice(miau).

animal(pato) :-
    tiene_plumas,
    dice(cuac).
```

EXECUTION STATUS: READY

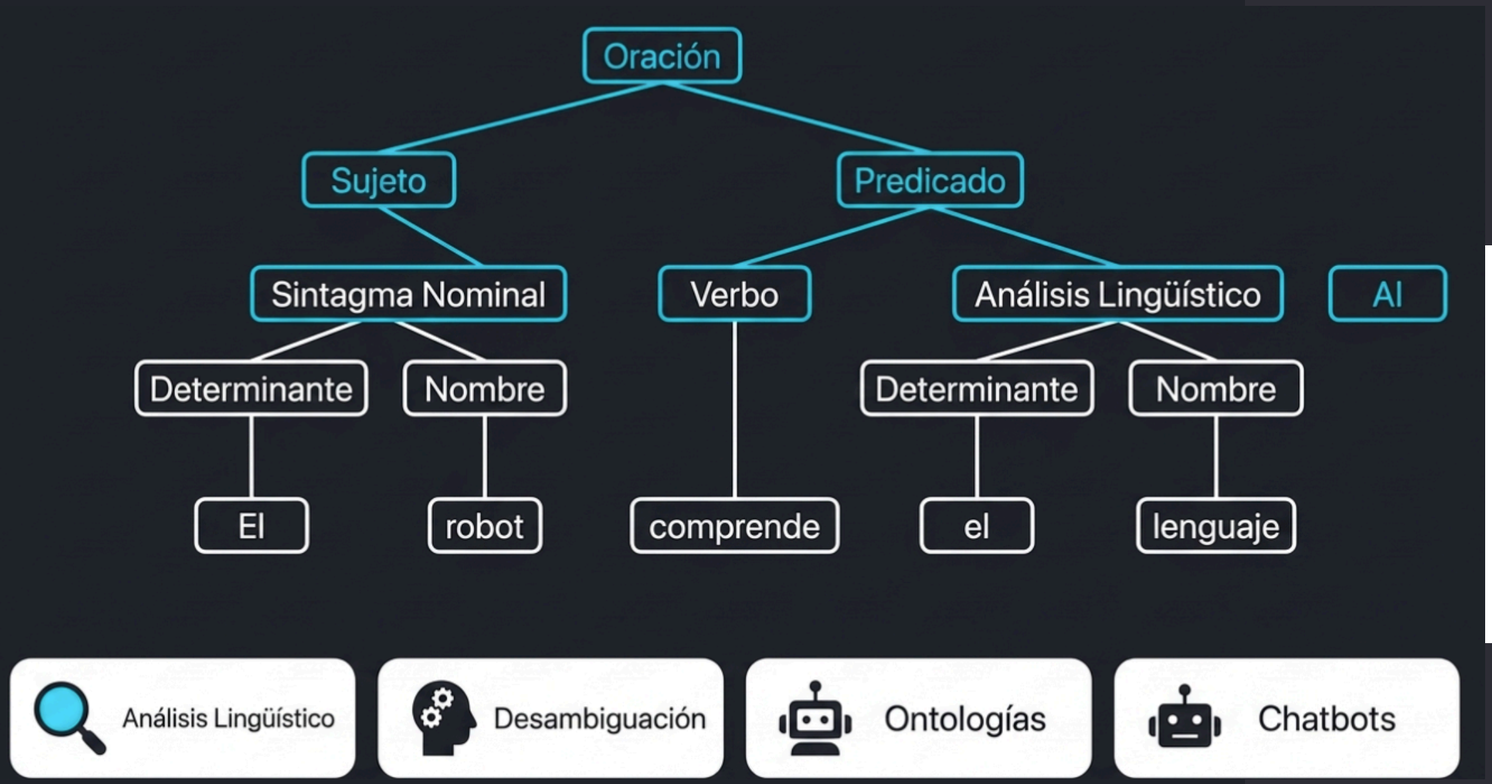
SISTEMAS NOTABLES

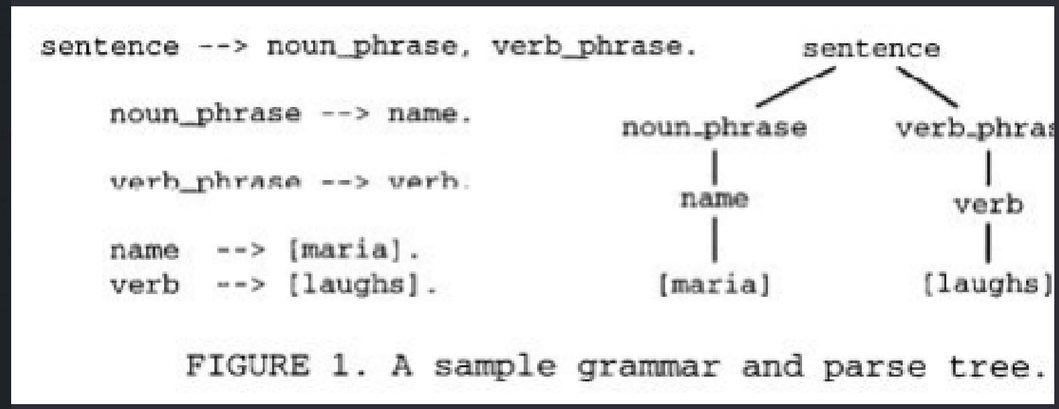
- PXDES
- CaDeT
- Dendral
- DXplain
- MYCIN

}

Aplicaciones de este paradigma {

Generacion y reconocimiento de lenguaje natural





}

Aplicaciones de este paradigma {

Bases de Datos Relacionales

PROLOG

Evaluación:
Fila por fila (tupla a tupla).

Lógica:
Razonamiento y reglas recursivas.

UNIFICACIÓN:
Mecanismo central de emparejamiento.

SQL

Evaluación:
Operaciones sobre conjuntos de datos.

Lógica:
Qué datos obtener, no cómo.

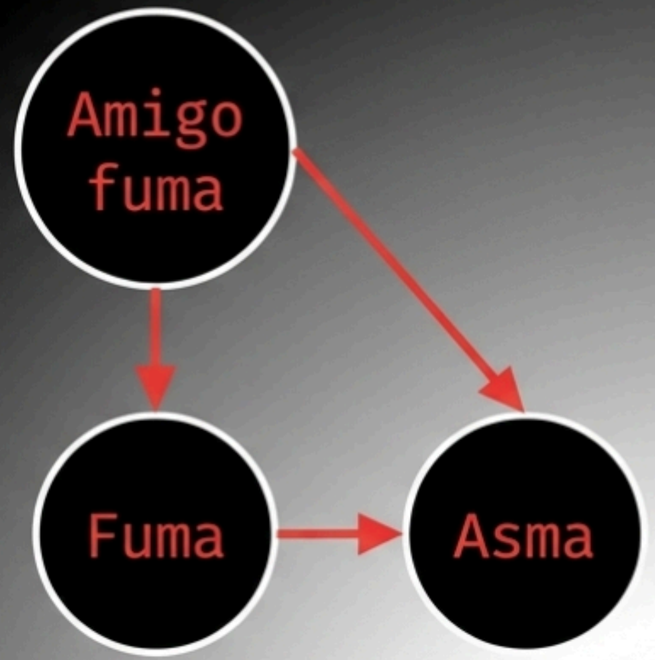
}

Aplicaciones de este paradigma {

Programación Lógica Probabilística

```
0.3::fuma(X).
0.2::fuma(X) :- amigo(X,Y), fuma(Y).
0.4::asma(X) :- fuma(X).

?- asma(2).    ⟶  P = 0.15
*  evidencia   ⟶  P = 0.19
```



- Diagnóstico médico
- Reconocimiento actividades
- Detección fraude

}

Aplicaciones de este paradigma {

¿QUE PASA EN EL MUNDO REAL?

SIEMENS

Automatización.

IBM WATSON

Asistente.

GENEXUS

Low-code.

CYC

Sentido común.

TERMINUSDB

Base versionada.

PROLOG

Núcleo IA.

}

<!--Universidad Nacional-->

Bibliografía

- F. Alonso Amo y M. Villalobos Abarca, “Programación lógica: un enfoque para desarrollar aplicaciones,” Conciencia Tecnológica, no. 14, pp. 3-11, ago. 2000. [En línea]. Disponible en: Redalc

Dahl, Veronica, and Alejandro Javier García. "Programación Lógica." Triangle: llenguatge, literatura, computació 2 (2010): 1-64.

- https://github.com/souffle-lang/souffle
- https://keepcoding.io/blog/que-es-la-programacion-logica/
- https://github.com/ferestrepoca/paradigmas-de-programacion/blob/gh-pages/proglogica/logica_teoria/aplicaciones.html

}

<!--Universidad Nacional-->

Gracias.