

**UNIVERSIDAD ANDRÉS BELLO**

**FACULTAD DE INGENIERÍA**

**INGENIERÍA CIVIL INFORMÁTICA**

**Trabajo 4 – Prolog/Lisp**

**ANDRÉS EDUARDO VALENZUELA GONZÁLEZ**

**SANTIAGO - CHILE**

**OCTUBRE, 2017**

Contenido

[**1.** **Introducción a Lisp** 3](#_Toc496662006)

[**2.** **Introducción a Prolog** 3](#_Toc496662007)

[**3.** **Desarrollo del problema** 4](#_Toc496662008)

[**4.** **Analisis de Resultados** 4](#_Toc496662009)

[**5.** **Conclusiones** 13](#_Toc496662010)

# **Introducción a Lisp**

En los comienzos de la era computacional resultaba impensable el uso de computadoras para uso doméstico, dado a lo cual no se disponía de una amplia gama de lenguajes de programación. Uno de los primeros (y más conocido hoy en día) lenguajes de alto nivel creado fue *FORTRAN* para propósito general (Desarrollado por *IBM* en el año 1957). Luego, en 1958 fue desarrollado *Lisp* (**LIS**t **P**rocessor) en el Instituto Tecnológico de Massachusetts por el ingeniero y ganador del premio Turing *John McCarthy* con el propósito de representar una notación matemática practica para los programas de computadoras basada en el cálculo *lambda* (de *Alonzo Church*). Mencionado lenguaje se convirtió en el preferido por investigadores de IA (*Inteligencia Artificial)*. También destaca por ser el lenguaje pionero en incluir estructuras de datos tales como los arboles binarios, el almacenamiento automático, tipos de datos dinámicos y compiladores auto contenidos. Pero el elemento fundamental de este lenguaje son las *Listas* (tanto los programas como sus tipos de datos son listas, de ahí su acrónimo List Processor) ya que, para la *IA* es el hecho de que el código y los datos tengan el mismo tratamiento como listas lo que hace especialmente sencillo escribir programas capaces de escribir otros programas según las circunstancias.

# **Introducción a Prolog**

Junto con *LISP*, *Prolog* es un lenguaje utilizado ampliamente en el campo de la inteligencia artificial.  
Creado por Alain Colmerauer a principio de los años 70 en la Universidad de Aix-Marseille I (Marseille, Francia) con el propósito de clasificar lenguajes naturales, cambio su objetivo a ser un lenguaje de programación único por ser declarativo (en donde se declaran/especifican un conjunto de condiciones, proposiciones, afirmaciones, restricciones o transformaciones que describen el problema y detallan su solución) y semi-interpretado, esto significa que también posee un compilador. Todas las sentencias se presentan en forma de reglas y hechos.

La participación de estos lenguajes dentro del campo de la inteligencia artificial es irrefutable, además de haber sido los fundadores de muchas bases y conceptos necesarios para entender el área de *IA*.

Para entender la extensión de este informe, se declara necesario el conocimiento base respecto a estos lenguajes de programación para poder comprender y analizar los siguientes ejemplos.

# **Desarrollo del problema**

Para desarrollar conocimiento y convocar la practica ante ambos lenguajes de programación fueron propuestos 8 ejercicios, los cuales solicitan conocimientos tanto en el uso de estructura de datos como el diseño de algoritmos recursivos.

Para Lisp se solicita la resolución de los siguientes ejercicios:

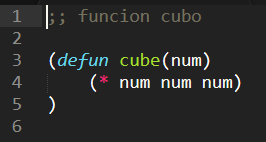
1. Una función a la que se le entregue un numero N y retorne el cubo de ese mismo número.
2. Una función a la que se le entregue un numero N y retorne su valor factorial.
3. Tres funciones que reciban por parámetro una lista y que devuelvan respectivamente el primer, segundo y tercer elemento de mencionada lista.
4. Una función que retorne el promedio cuadrado de dos números.
5. Una función que calcule y resuelva la ecuación cuadrática de la forma .
6. Una función que reciba una lista y la rote un elemento hacia la izquierda.

Se solicita además implementar las funciones en Prolog:

1. Una función que calcule la longitud de una lista.
2. Una función que entregue el número más grande dentro de un árbol.
3. Una función que verifique si un elemento está o no en una lista.

# **Analisis de Resultados**

* 1. Función cubo (LISP):   
     Se presenta el siguiente código:



En la primera línea se definen los comentarios. Siguiendo por la tercera línea se ubica la cabecera de la función, la cual se define con la palabra reservada *defun*, a continuación el nombre de la función y luego los parámetros que esta recibirá.  
El cuerpo de la función (línea 4) se compone de la típica notación polaca de Lisp (o más conocida por notación prefija, en la cual se tipea primero el operador y seguidos los operandos como muestra la imagen). En este ejercicio se vislumbra la multiplicación del número ingresado por parámetro tres veces por sí mismo.

En la siguiente imagen podemos ver su correcto funcionamiento:

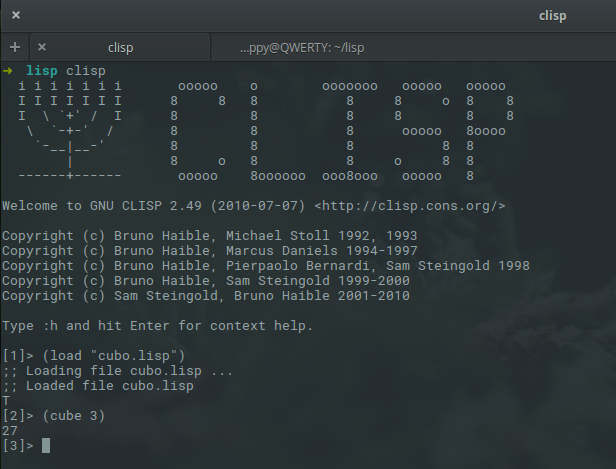
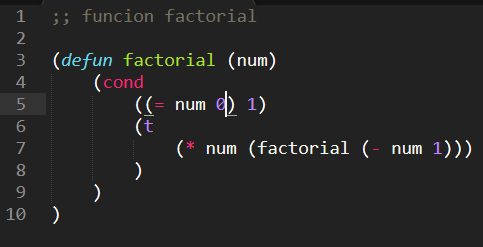


Ilustración .- Función cubo

* 1. Función factorial (LISP):

Se presenta el siguiente código:



En la línea 3 se vuelve a utilizar la palabra reservada *defun* para definir una nueva función, seguida por el nombre de esta y los parámetros a recibir.  
En la línea 4 se puede visualizar el uso de la macro *cond*, la cual permite evaluar la condición de la línea 5. Si esta se cumple, se retorna 1, pero de ser falsa se continua en la línea 6 en donde ***t***indica lo que debe ser retornado por la función, en este caso sería la multiplicación entre el numero recibido por parámetro y la llamada de la función actual reduciendo el número en una unidad. Debido a este decremento, el numero cumplirá luego con la condición de termino *(= num 0)* y luego se multiplicaran los valores retornados.

A continuación se verifica que el código cumple con su propósito:

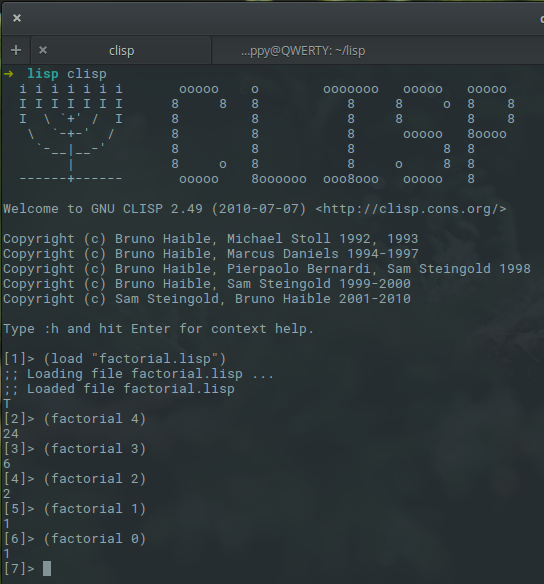
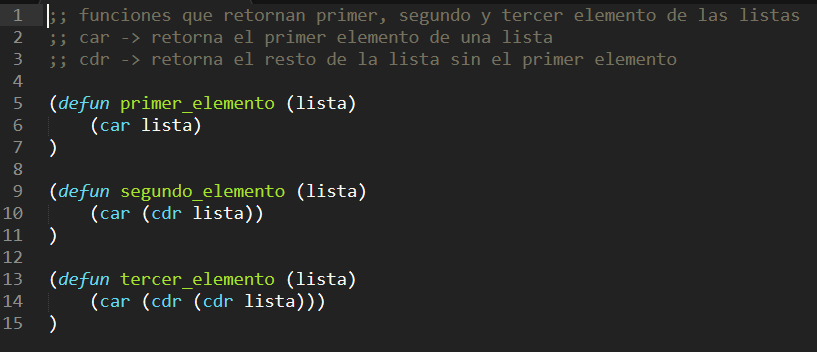


Ilustración .- Función factorial

* 1. Funciones primer, segundo y tercer elemento de una lista (LISP):

Se presenta el siguiente código:



Se parte mencionando que la macros *car* corresponde a una función que retorna el primer elemento de la lista que se le entregue por parámetro. Luego, la macros *cdr* corresponde a una función que retorna el resto de la lista (la misma lista sin el primer elemento), lo cual resulta bastante conveniente en este ejercicio.

En las líneas 5, 9 y 13 se visualiza el uso de la palabra reservada *defun*, el nombre de cada función y por último el parámetro que recibe (nombre a convenir, puede ser el que uno guste. Por convención se escogió “lista”). Consecutivo a la línea 5, se ubica el uso de la macro *car* y luego la lista entregada por parámetro, lo cual retorna el primer elemento de mencionada lista.

En la línea 10 se puede contemplar el uso de las macros *car* y *cdr*, de las cuales se considera su funcionalidad en sentido de derecha a izquierda, es decir, primero la macros *cdr* nos retorna la cola de la lista y luego la macro *car* extrae el primer elemento de esta cola (segundo elemento).

Por último, en la línea 14 se aprecia que se utiliza dos veces la macro *cdr* y finaliza con la llamada a la macro *car*, lo cual se puede interpretar como “retorna la cola de la lista, luego retorna la cola de la lista para finalmente retornar la cabeza de la lista restante (tercer elemento)”.

A continuación se verifica que el código cumple con su propósito:

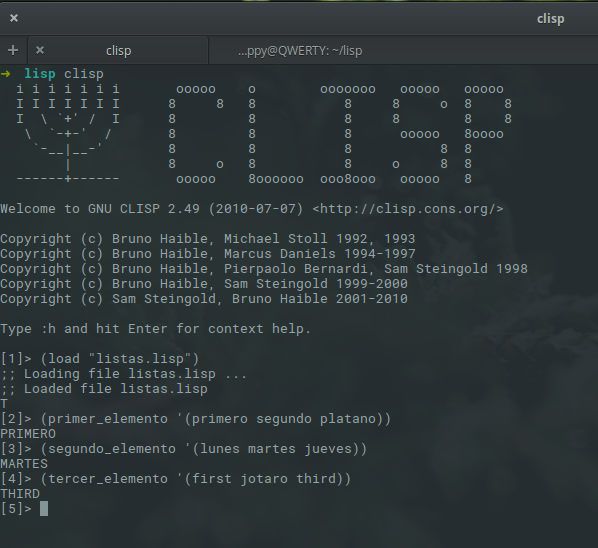
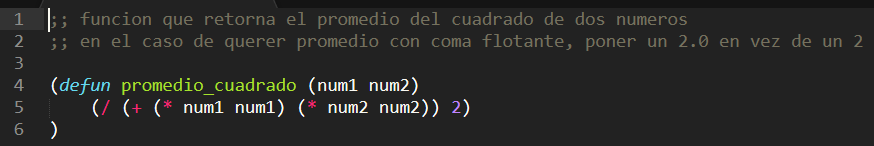


Ilustración .- Función listas

* 1. Función promedio cuadrado de dos números (LISP):

Se presenta el siguiente código:



Esta función cumple con los mismos estándares de los códigos anteriores, ya sean estos la notación polaca, el uso de la palabra reservada *defun* y los paréntesis encapsulando cada función y operaciones, pero en este caso se reciben dos parámetros (num1 y num2) para retornar un solo valor. Nótese que están siendo separados por un espacio y no por una coma (así mismo, la llamada a esta función cumple con la misma estructura). Luego, en la línea 5 podemos ver a la izquierda la multiplicación de la variable num1 por sí misma y a su derecha la multiplicación de la variable num2 por sí misma. Ambas operaciones están englobadas por el operador **+** para ser sumadas, seguido por el operador **/** para ser divididas por el número 2. Se detalla desde ya que si el modulo del resultado por 2 no es cero, el resultado de división corresponderá a una fracción (y se mostrara como fracción), en cambio si el resultado es “cerrado”, será impreso un número entero tal como muestra la siguiente imagen:

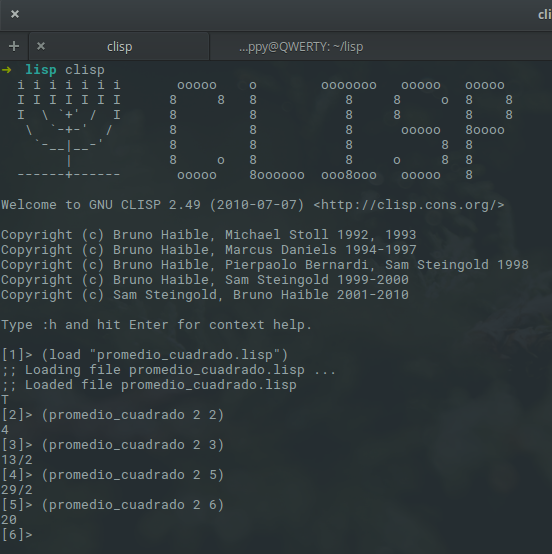
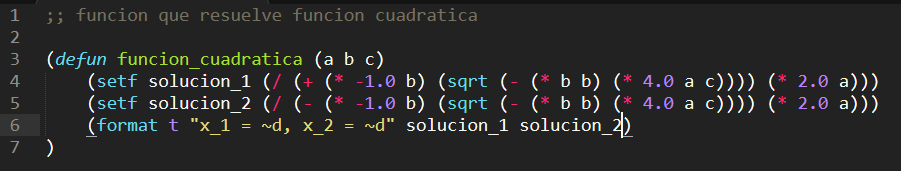


Ilustración .- Función promedio cuadrado

* 1. Función de resolución de función cuadrática (LISP):

Se presenta el siguiente código:



Comenzando por la línea 3, luego de usar la palabra reservada *defun* y dar un nombre representativo a la función, se ubican los parámetros que se recibirán (en este caso serán 3, representados por las letras a, b y c).

Al comienzo de la línea 4 se ubica una nueva macro reservada llamada *setf*, la cual nos permite asignar un valor a una variable de nombre a convenir (en este caso *solución\_1*). Mencionada macro recibe un nombre de variable y el valor de esta variable, el cual será el cálculo de la fórmula de resolución de la función cuadrática.

Se puede contemplar que las líneas 4 y 5 son idénticas **salvo** en el signo + y – que le sigue al signo de división, esto es debido a que la fórmula de la ecuación cuadrática contempla dos resultados (uno positivo y uno negativo).

Siguiendo la misma lógica de las funciones anteriores, se puede llegar a los resultados deseados. Cabe destacar que se utiliza la macro *ser* para extraer el valor de la raíz cuadrada de lo que se le entrega por parámetro.

Luego de hacer los cálculos pertinentes, se imprimen los resultados por consola gracias a la macro *format* tal como muestra la siguiente imagen:

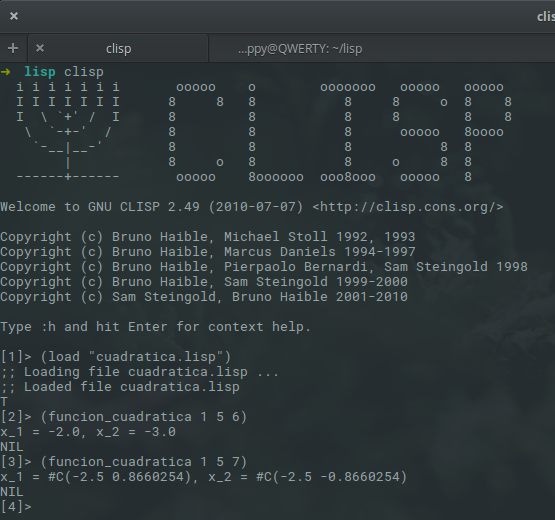
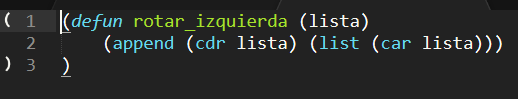


Ilustración .- Función cuadrática

En la última invocación de la función se puede ver como se ingresan valores para una función que no tiene solución, por lo cual *LISP* responde correctamente con una solución con dominio en los números complejos (#C indica que el número pertenece al dominio mencionado).

* 1. Función rotar una lista (LISP):

Se presenta el siguiente código:



Siguiendo la misma estructura que los anteriores códigos, se puede ver que se recibe una lista por parámetro para luego encontrarnos con la lógica en la línea 2.

Recordando que las macros *car* y *cdr* retornan el primer elemento y el resto de la lista respectivamente, se procede a extraer el tope de la lista y anexarlo al final de la misma. Dado a que las funciones retornan una copia del parámetro original, se debe trabajar con lo que retorne cada función (*cdr* y *car*).

Siguiendo con la lógica, se procede a juntar la cola y la cabeza con la macro *append*, la cual recibe dos listas para ser anexadas y, dado a que recibe explícitamente dos listas, se ocupa la macro *list* para transformar el elemento retornado por la macro *car* a una lista.

El resultado de esto se puede apreciar en la siguiente imagen:

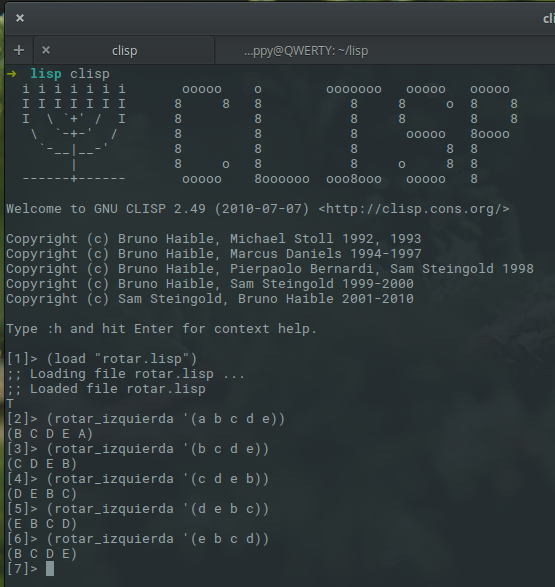
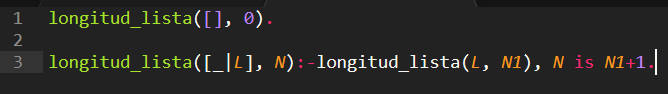


Ilustración .- Función rotar lista

* 1. Función cálculo de longitud de una lista (PROLOG):

Se presenta el siguiente código:



En el cual, en la línea 1 se define nuestro hecho (o más bien conocido por condición de término), el cual se puede traducir a: Si recibe una lista vacía, su longitud corresponde a 0 y en la línea 2 se tiene la estructura de *Regla :- Acción* lo cual se puede traducir a: Se recibe la cabeza de la lista y la cola de la lista, junto con un numero N **tal que** se llame recursivamente a la presente función con L como la cola de la lista y N1 como un numero incrementado en uno **(*N is N1 + 1***).

Cabe destacar que se finaliza cada sentencia con un punto ( **.** )

Se adjunta captura del funcionamiento de este script:

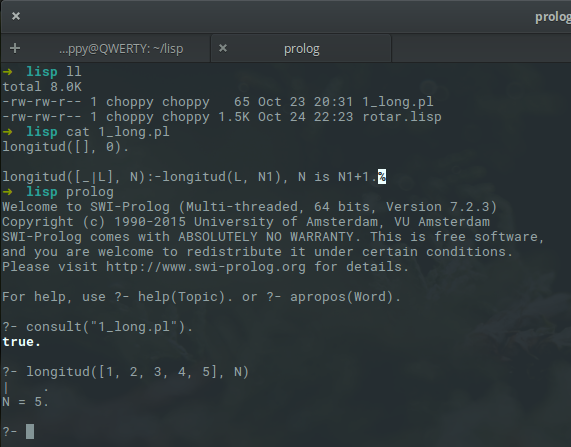
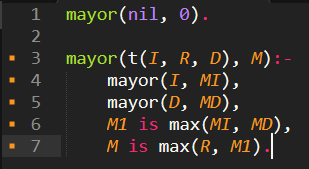


Ilustración .- Función longitud lista

* 1. Función valor máximo de un árbol binario:

Se presenta el siguiente código:



En la primera línea definimos que el máximo de *nil* (comúnmente llamado **NULL**) es 0. Luego en la línea siguiente creamos otra regla en donde le entregamos el árbol como estructura el cual recibe una raíz (R), su nodo izquierdo (I) y su nodo derecho (D). Luego en la línea 4 recursivamente se recorren los nodos izquierdos del árbol y en la línea 5 los derechos. En las líneas 6 y 7 utiliza la función *max* para obtener el valor mayor entre cada hoja (izquierda o derecha) y para calcular el valor final al ver cuál es mayor con la raíz, correspondientemente.

Se detalla funcionalidad en la siguiente captura:

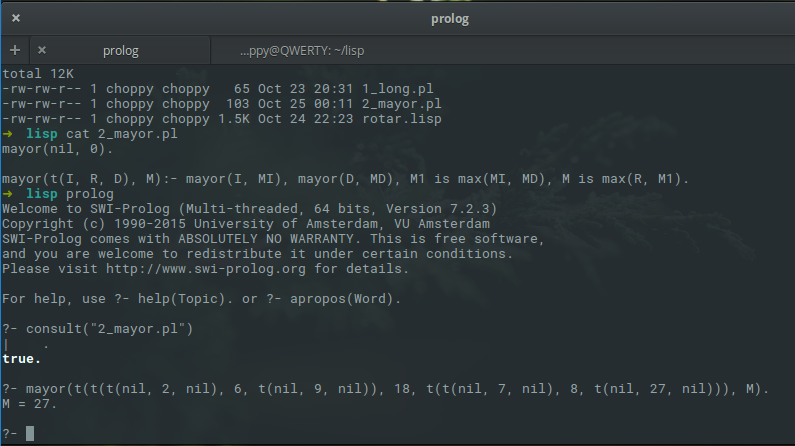
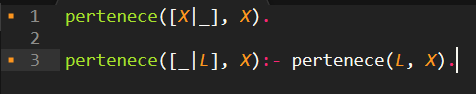


Ilustración .- Función mayor

* 1. Función verificar pertenencia en lista:

Se presenta el siguiente código:



En la línea 1 definimos la primera regla la cual corresponde a que verifique que la cabeza de la lista, es decir, el primer elemento de la lista sea el elemento X que se entrega por parámetro. Luego la segunda regla utiliza la recursión para comprobar, al igual que en el ejercicio 4.7, se le entrega la cola de la lista y se le agrega una condición en la línea 4 que es la misma función pero evaluando solo la cola. Esto hará que revise todas las posiciones hasta el último elemento y, si el último elemento no corresponde al elemento que se busca, devolverá *false*. Del caso contrario, retornara *true* tal como se muestra en la siguiente captura:

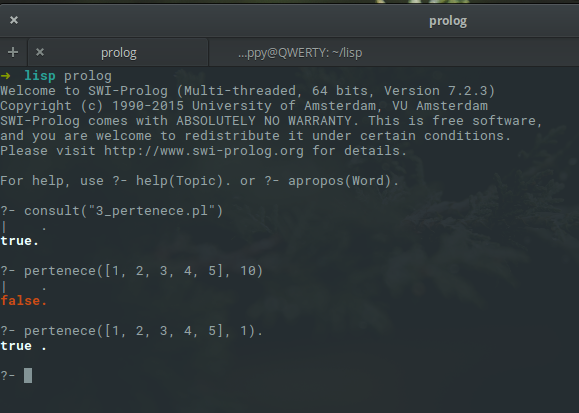


Ilustración .- Función pertenencia lista

# **Conclusiones**

Se mantiene en consideración que ambos lenguajes evaluados en este informe se remontan a finales de los años 50 (LISP) y principios de los años 70 (PROLOG). En el caso de LISP se presenta un lenguaje un poco tosco en sintaxis (por la época tal vez), poderoso en cuanto a estructura de datos (todos los datos son listas y punteros) para la época pero débil en cuanto a funciones propias de los tipos de datos. Se considera útil también la capacidad de poder modularizar el código gracias a la macro *load* y se estima ventajoso que sea un lenguaje interpretado, lo que da para escribir programas que escriban programas. En cuanto a PROLOG se alcanza una sintaxis más cercana a C (y derivados), pero distinta a cualquier otra en cuanto a *hechos* y *reglas*.

Para finalizar, se denota que ambos lenguajes carecen de sentencias iterativas (for, while, do while, foreach, etc), por lo tanto es fundamental el conocimiento de recursividad para extraer lo requerido.