Tarea Independiente 30/10/2025

David Núñez Franco

November 3, 2025

Inventario de Conceptos Claves

- Patrones comunes de recursión en Prolog
- Patrones de Listas
- Patrones de Árboles
- Evaluadores

Ejercicio 1

Escriba un quicksort en Prolog

Solución

```
% quicksort divides the list by choosing (arbitrary) the first element (pivot) and using this element to
% split the list into Left and Right. Left has all the elements smaller than the pivot. Right has all the elements
% larger than the pivot. [Left, pivot, Right].

quicksort([], []).
quicksort([X|Xs], Ys) :-
partition(Xs, X, Left, Right),
quicksort(Left, Ls),
quicksort(Right, Rs),
append(Ls, [X|Rs], Ys).

partition([], _, [], []).
partition([X|Xs], Y, [X|Ls], Rs) :-
X =< Y,
partition(Xs, Y, Ls, Rs).
```

```
        partition([X|Xs], Y, Ls, [X|Rs]) :-
        X > Y,
        partition(Xs, Y, Ls, Rs).

        append([], Ys, Ys).
        append([X|Xs], Ys, [X|Zs]) :-
        append(Xs, Ys, Zs).

        % quicksort(Xs, Ys) sorts list Xs into ascending order list Ys (or Ys is an ordered permutation of Xs).
        % Ys is a sorted [X|Xs] where Left and Right is a result of partitioning Xs by X, Ls and Rs are the sorted
        % Left and Right recursively, and Ys is the result of appending [X|Rs] to Ls.
        % partitioning[X|Xs] with Y gives list Ls (left) and Rs (right), if X is less than or equal Y and partitioning
        % Xs with Y gives Ls and Rs.
        % Base case is the empty list.

        test :-
        quicksort([3,1,4,1,5,9,2,6], Sorted),
        writeln(Sorted).

        :- initialization(test).
```

```
        ?- [quicksort].
        [1,1,2,3,4,5,6,9]
        true.
```

Listing 1: output

Ejercicio 3

Añada exponenciación (** en Prolog)

Solución

```
        % eval(Expr, Context, Result) : Result is the value of evaluating expression Expr

        context_find(C, X, V) :-
        member([X, V], C).

        is_binary(E, Oper, L, R) :-
        E =.. [Oper, L, R],
```

```
        member(Oper, [+, -, *, /, **]).

        % base case
        eval(N, _, N) :-
        number(N). % check if N is a number
        eval(X, C, V) :-
        atom(X),
        context_find(C, X, V).
        eval(E, C, Result) :-
        is_binary(E, Oper, L, R), % check if E is a binary
           expression
        eval(L, C, RL),
        eval(R, C, RR),
        ER =.. [Oper, RL, RR],
        Result is ER.
        eval(- E, C, R) :-
        eval(E, C, VE),
        R is - VE. % change the sign of VE

        :-
        E = x + 10 + -y,
        C = [[x, 20], [y, 30]], % Memory/Context/Environment
        eval(E, C, R),
        format('>>> ~w --eval(~w)--> ~w', [E, C, R]),
        E2 = x ** 2 + y,
        C2 = [[x, 3], [y, 4]],
        eval(E2, C2, R2),
        format("E2=~w C=~w => ~w~n", [E2, C2, R2]).
```

```
        ?- [eval].
        >>> x+10+ -y --eval([[x,20],[y,30]])--> 0E2=x**2+y C=[[x
           ,3],[y,4]] => 13
        true.
```

Listing 2: output