/\*

Homework number 9

Rodrigo Benavente

2021-16-09

\*/

% First exercise, recieves a list and an element in order to see if the element is inside the list

% Base case if list is empty

in\_list([], \_Ele) :-

    false.

% Other cases

in\_list([Head | \_Tail], Ele) :- %Check if the head is the same as the element

    Head == Ele,

    !. %Cut if Head is the same as element

%Else we send the tail as the next list

in\_list([Head | Tail], Ele) :-

    Head \= Ele,

    in\_list(Tail, Ele).

% Second exercise, recieves index and list as arguments \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Base cases

element\_at(\_Index, [], \_R) :- %If list is empty

    fail.

element\_at(Index, [Head | Tail], N) :- %If the index is negative

    Index < 0,

    !,

    Id1 is Index \* -1,

    reverse([Head|Tail], ListR), %Reverse list

    element\_at(Id1, ListR, N, 1). %Start from 1 because we can't start from 0

% Create a counter

element\_at(Index, [Head | Tail], N) :-

    Index >= 0, %Verify the index is positive

    !,

    element\_at(Index, [Head | Tail], N,0). %Start from pos 0

% Work with other cases

element\_at(Index, [\_Head | Tail], N, Count) :-

    Index \= Count, %We haven't reached the index

    !,

    C1 is Count + 1,

    element\_at(Index, Tail, N, C1).

element\_at(Index, [Head | \_Tail], N, Count) :- %We reached the index so N gets assigned the value of Head

    Index = Count,

    !,

    N = Head.

% Third exercise, recieves 3 ints and a list as its arguments \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Base cases

range(Start, End, Inc, []) :- %If the Start is bigger than End and the increment is positive

    Inc > 0,

    Start >= End,

    !.

range(Start, End, Inc, []) :- %If the End is bigger than Start and the increment is positive

    Inc < 0,

    Start =< End,

    !.

range(\_Start, \_End, 0, []) :-  %If the increment is 0 we return an empty list

    !.

% Other cases

range(Start, End, Inc, [Start | Tail]) :- %If the Inc is positive

    Inc > 0,

    Start < End,

    !,

    S1 is Start + Inc,

    range(S1, End, Inc, Tail).

range(Start, End, Inc, [Start | Tail]) :- %If the Inc is negative

    Inc < 0,

    Start >= End,

    !,

    S1 is Start + Inc,

    range(S1, End, Inc, Tail).

% Fourth exercise, recieves a list as its argument \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Base Case when we don't recieve a list

remove\_doubles([], []).

% Case if list ins't empty

remove\_doubles([Head|Tail], R) :- remove\_doubles(Tail, R, Head). %Add 2 temps that will be compared

% Base Cases when we reach last element

remove\_doubles([], [X], X).

% Case where there are no repeated values together

remove\_doubles([Head | Tail], [X | Temp], X) :-

    Y1 = Head,

    X \= Y1,

    !,

    remove\_doubles(Tail, Temp, Y1).

%Case where there are repeated values together

remove\_doubles([Head | Tail], Temp, X) :-

    Y1 = Head,

    X == Y1,

    !,

    remove\_doubles(Tail, Temp, Y1).

% Fifth exercise, recieves an integer as its argument \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%Base Case where we recieve 0 or 1

fibonacci(0, [0]) :-

    !.

fibonacci(Num, R) :-

    Num > 0,

    fibonacci(Num, R, \_X, \_Y).

fibonacci(1, [0, 1], 0, 1).

fibonacci(Num, L, X, Y) :-

    Num > 1,

    Num1 is Num - 1,

    !,

    fibonacci(Num1, List, X1, Y1),

    Sum1 is X1 + Y1,

    X is Y1,

    Y is Sum1,

    append(List, [Sum1], L),

    !.