P2 4A => Having 2 numbers reprezented as lists, add them. [1, 2, 3, 4] + [2, 4, 6] = [1, 4, 8, 0]

Mathematical model **reverseList(l1..ln)** =

{reverseList(l2..ln) U l1

```
% append(L - list, E - elemnt

% reverseList(L - list, C - colector, R - reversed list)
% flow model (i, o), (i, i)
reverseList([], C, C).
reverseList([H|T], C, R):-
    NewC = [H|C],
    reverseList(T, NewC, R).

reverseListCaller(L, R):-
    reverseList(L, [], R).
```

```
Singleton variables: [H]
```

```
?- reverseListCaller([1, 2, 3, 4], R).
```

Mathematical model **adder(l1..ln, k1..km, Carry)** =

{(l1 + Carry) / 10 U adder(l2..ln, [], (l1 + Carry) mod 10), if m = 0

{(k1 + Carry) / 10 U adder([], k2..kn, (k1 + Carry) mod 10), if n = 0

{(l1 + k1 + Carry) / 10 U adder(l2..ln, k2..km, (l1 + k1 + Carry) mod 10), otherwise

```
% adder(L1 - list 1, L2 - list 2, C - carry, R - resulted list
% flow model (i, i, i, o) (i, i, i, i)
adder([], [], 0, []).
adder([], [], C, [C]).
adder([H|T], [], C, R):-
    NewC is (H + C) // 10,
    %write('NewC = '), write(NewC), nl,
```

```
    adder(T, [], NewC, NewR),
    NewH is (H + C) mod 10,
    R = [NewH|NewR].
adder([], [H|T], C, R):-
    NewC is (H + C) // 10,
    %write('NewC = '), write(NewC), nl,
    adder([], T, NewC, NewR),
    NewH is (H + C) mod 10,
    R = [NewH|NewR].
adder([H1|T1], [H2|T2], C, R):-
    NewC is (H1 + H2 + C) // 10,
    %write('NewC = '), write(NewC), nl,
    adder(T1, T2, NewC, NewR),
    NewH is (H1 + H2 + C) mod 10,
    R = [NewH|NewR].

adderCaller(L1, L2, R):-
    reverseListCaller(L1, L1R),
    reverseListCaller(L2, L2R),
    adder(L1R, L2R, 0, NewR),
```

```
?- adderCaller([1, 2, 3, 4], [1, 2, 3], R).
```

```
?- N is 5 // 2.
```

2B => Define a predicate to produce a list of pairs (atom n) from an initial list of atoms. In this initial list atom has n occurrences. - 27 min

Mathematical model **count_occurrences(l1..ln, E)** =

> {1 + count_occurrences(l2..ln, E), if l1 = E

> {count_occurrences(l2..ln, E), otherwise

```
% countE(L - list, E - element, C - contor)
% flow model (i, i, o, o) (i, i, i, i)
count_occurrences([], _, 0).
```

```
count_occurrences([H|T], E, C):-
    H =:= E,
    count_occurrences(T, E, NewC),
    C is NewC + 1, !.
count_occurrences([_|T], E, C):-
    count_occurrences(T, E, C).
```

```
?- countE([1, 2, 1, 3, 4, 1, 1, 1, 2], 1, C).
```

Mathematical model **removeE(l1..ln, E)** =

> {l1 U removeE(l2..ln, E), if l1 =/= E

> {removeE(l2..ln, E), otherwise

```
% removeE(L - list, E - element, R - resulted list)
% flow model
removeE([], _, []).
removeE([E|T], E, R):- % if the head and the element are equal
    removeE(T, E, R).
removeE([H|T], E, R):-
    removeE(T, E, NewR),
    R = [H|NewR].
```

```
?- removeE([1, 2, 3, 1, 2, 3, 1], 1, R).
```

Empty markdown cell. Double click to edit

```
% frequency(L - list, R - resulted list)
% flow model (i, o), (i, i)
frequency([], []).
frequency([H|T], R):-
    count_occurrences([H|T], H, C),
    removeE([H|T], H, NewList),
    frequency(NewList, NewR),
    Pair = [H,C],
    R = [Pair|NewR].
```

```
?- frequency([1, 2, 3, 1, 2, 4, 5, 1], R).
```

P2 7A=> Display the indexes of the occurences of the maximum element - 30 min

Mathematical model **findMax(l1..ln, M)** =

{M = l1 & findMax(l2..ln, M), if l1 > M

{findMax(l2..ln, M), otherwise

```
% findMax(L - list, M - temporary Maximum)
% flow model (i, o), (i, i)
findMax([], M, M). % if the list is empty simply
findMax([H|T], M, R):-
    H > M,
    NewM is H,
    %write(NewM), nl,
    findMax(T, NewM, R), !.
findMax([_|T], M, R):-
    findMax(T, M, R).
findMaxCaller([H|T], R):-
    findMax(T, H, R).
```

```
?- findMaxCaller([1, 2, 3, 20, 4, 5, 10], M).
```

Mathematical model **indexesOfElem(l1..ln, I, E)** =

{I U indexesOfElem(l2..ln, I + 1, E), if l1 = E

{indexesOfElem(l2..ln, I + 1, E), otherwise

```
% indexesOfElem(L - list, I - index, E - element, R - list of indexes)
% flow model (i, i, i, o)
indexesOfElem([], _, _, []).
indexesOfElem([E|T], I, E, R):- % the head is equal to the element
    NewI is I + 1,
```

```
    indexesOfElem(T, NewI, E, NewR),
    R = [I|NewR], !.
indexesOfElem([_|T], I, E, R):- % the head is not equal to the element
    NewI is I + 1,
    indexesOfElem(T, NewI, E, R).
indexesOfElemCaller(L, E, R):-
    indexesOfElem(L, 1, E, R).
```

```
?- indexesOfElemCaller([1, 2, 3, 1, 2, 1, 1], 1, R).
```

Mathematical model **indexMaxElem(l1..ln)** = indexesOfElem(l1..ln, findMaxCaller(l1..ln))

```
% indexMaxElem(L - list, R - resulted list)
% flow model (i, o), (i, i)
indexMaxElem(L, R):-
    findMaxCaller(L, Max),
    indexesOfElemCaller(L, Max, R).
```

```
?- indexMaxElem([1, 2, 5, 3, 5, 5], R).
```

P2 5A => Substitute all the occurences of an element in the first list with all the elements in the other list L1 = [1, 2, 1], nr = 1, L2 = [8, 9] => R = [8, 9, 2, 8, 9] - 30 min

Mathematical Model **appendList(l1..ln, k1..km)** =

> { k1..km, if n = 0

> { l1 U appendList(l2..ln, k1..km), otherwise

```
% appendList(L1 - first list, L2 - second list, R - resulted list)
% flow model (i, i, o), (i, i, i)
appendList([], L, L).
appendList([H|T], L, R):-
    appendList(T, L, NewR),
```

```
    R = [H|NewR].
```

≡ ?- appendList([1, 2, 3], [4, 5, 6], **R**).

Create a Program | Query | Markdown | HTML **cell here**

Mathematical Model **subEList(l1..ln, E, k1..km)** =

{ k1..km U subEList(l2..ln, E, k1..km), if l1 = E

{ l1 U subEList(l2..ln, E, k1..km), otherwise

```
% substitutes every element E with the second list
% subEList(L1 - first list, E - element, L2 - second list, R - resulted list/ cc
% flow model (i, i, i, o), (i, i, i, i)
subEList([], _, _, R, R).
subEList([Elem|T], Elem, L2, R, FR):- % when the head is equal with Element
    appendList(R, L2, NewR),
    % write(NewR), nl,
    subEList(T, Elem, L2, NewR, FR), !.
subEList([H|T], Elem, L2, R, FR):-
    appendList(R, [H], NewR),
    subEList(T, Elem, L2, NewR, FR), !.

subEListCaller(L1, E, L2, R):-
    subEList(L1, E, L2, [], R).
```

≡ ?- subEListCaller([1, 2, 1], 1, [8, 9], **R**).

---
15A => Transform a list in a set keeping the first appearance of the elements [1, 2, 3, 2, 1] -> [1, 2, 3]

Empty markdown cell. Double click to edit

```
% toSet(L - list, R - resulted list
% flow model
toSet([], []).
toSet([H|T], R):-
    removeE(T, H, NewL),
    toSet(NewL, NewR),
    R = [H|NewR], !.
```

≡ ?-

```
toSet([1, 2, 3, 2, 1], R).
```

R = [1, 2, 3]

13A => Transform a list in a set keeping the last appearance of the elements [1, 2, 3, 2, 1] -> [3, 2, 1]

```
% toSetLast(L - list, R - resulted list)
% flow model (i, o), (i, i)
toSetLast([], []).
toSetLast([H|T], R):-
    count_occurrences(T, H, C),
    C =:= 0, % if there are not any duplicates of this number
    toSetLast(T, NewR),
    R = [H|NewR], !.
toSetLast([_|T], R):-
    toSetLast(T, R).
```

```
?- toSetLast([1, 2, 3, 2, 1], R).
```

R = [3, 2, 1]

3A => Define a predicate to remove from a list all repetitive elements.

```
% removeDuplicates(L - list, R - resulted list)
removeDuplicates([], []).
removeDuplicates([H|T], R):-
    count_occurrences(T, H, C),
    C > 0, % if there are any duplicates
    removeE([H|T], H, R1),
    removeDuplicates(R1, R), !.
removeDuplicates([H|T], R):-
    removeDuplicates(T, NewR),
    R = [H|NewR], !.
```

```
?-
```

```
removeDuplicates([1,2,1,4,1,3,4], R).
```

R = [2, 3]