```
Part A
1.
Qs = [A,B,C,D] -> safe_queens([B,C,D],A,1),safe_queens([B,C,D])

safe_queens([B,C,D],A,1):
A != B, abs(A-B) != 1, D1 = 1+1 =2, safe_queens([C,D],A,2).
safe_queens([C,D],A,2):
A != C, abs(A-C) != 2, D1 = 2+1 = 3, safe_queens([D],A,3).
safe_queens([D],A,3):
A != D, abs(A-D) != 3, D1 = 3+1 = 4, safe_queens([],A,4)..return.


Qs = [B,C,D] -> safe_queens([C,D],B,1), safe_queens([C,D])

safe_queens([C,D],B,1):
B != C, abs(B-C) != 1, D1 = 1+1 =2, safe_queens([D],B,2).
safe_queens([D],B,2):
B != D, abs(B-D) != 2, D1 = 2+1 =3, safe_queens([],B,3), return.

safe_queens([D],C,1), safe_queens([]).->return...

safe_queens([D],C,1):
C != D, abs(C-D) != 1, D1 = 1+ 1, safe_queens([],C,2)...return.

constrains:
A != B, abs(A-B) != 1
A != C, abs(A-C) != 2
A != D, abs(A-D) != 3
B != C, abs(B-C) != 1
B != D, abs(B-D) != 2
C != D, abs(C-D) != 1

the role of the third argument is to ensure the distance of those queens in order
to keep those queens in the chessboard.
```

```
2.
sudoku(Rows) :-
        length(Rows, 9), maplist(same_length(Rows), Rows), % 1
        append(Rows, Vs), Vs ins 1..9, % 2
        maplist(all_distinct, Rows), % 3
        transpose(Rows, Columns), % 4
        maplist(all_distinct, Columns), % 5
        Rows = [A,B,C,D,E,F,G,H,I], % 6
        blocks(A, B, C), blocks(D, E, F), blocks(G, H, I). % 7

blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :- % 8
        all_distinct([A,B,C,D,E,F,G,H,I]), % 9
        blocks(Bs1, Bs2, Bs3). % 10

problem(1, [[_,_,_, _,_,_, _,_,_],  % 11
            [_,_,_, _,_,3, _,8,5],
            [_,_,1, _,2,_, _,_,_],

            [_,_,_, 5,_,7, _,_,_],
            [_,_,4, _,_,_, 1,_,_],
            [_,9,_, _,_,_, _,_,_],

            [5,_,_, _,_,_, _,7,3],
            [_,_,2, _,1,_, _,_,_],
            [_,_,_, _,4,_, _,_,9]]).

1. make Rows to be a solution of 9 lists with length of 9.
2. fill Rows with number of 1..9
3. make list in Rows to be distincted to each others.
4. make Columns to be a list of lists that transpose from Rows with same length.
5. make list in Columns to be distincted to each others.
6. represent lists in Row with letters A..I
7. make every number to be different in Rows and Columns.
8. represents first three numbers in arguments with three letters.
9. make the first three numbers of arguments to be distincted.
10. to the same thing on the rest of numbers in arguments.
11. initialize lists
```

```
3.
% Example 6:
% A says: " At least two of us are knights."
% B says: " Exactly one of us is a knight."
% output:
% Ks = [_3536, _3542, _3536],
% sat(1#_3536*_3542)
example_knights(6, Ks) :-
    Ks = [A,B,C],
    sat(A=:=card([2,3],[A,B,C])),
    sat(B=:=card([1],Ks)).

% Example 7:
% A says: "I am a knight, but B isn't"
% C says: " I am the only knight among us."
% output:
% Ks = [0, 0, _3294],
% sat(_3294=:=_3294)
example_knights(7, Ks) :-
    Ks = [A,B,C],
    sat(A=:=(A * ~B)),
    sat(C=:=card([1],Ks)).

% Example 8:
% A says: "none of us is a knave."
% B says: "There is at least one knight among us."
% C says: " I am a knight."
% output:
% Ks = [0, _3872, 0],
% sat(_3872=:=_3872)
example_knights(8, Ks) :-
    Ks = [A,B,C],
    sat(A=:=(A * B * C)),
    sat(B=:=card([2],[~A,~B,~C])),
    sat(C=:= C).
```

```
part 2

:- use_module(library(clpfd)).

crypt1([H1|L1],[H2|L2],[H3|L3],L4) :-
    L4 ins 0..9,% Give constraints on variable values in L4
    H1 #\= 0, H2 #\=0, H3 #\= 0,% Heads cannot have value 0
    all_different(L4), % Variable values are distinct in L4
    reverse([H1|L1],NL1),reverse([H2|L2],NL2),reverse([H3|L3],NL3),% Reverse the
3 input words
    sumL(NL1,NL2,NL3).% Call to helper function that does the sum with reversed
words one iteration at a time

sumL(L1,L2,L3) :- sumh(L1,L2,L3,0,0).
sumh([],[],[],0,_).
sumh([],[],[H3|_],C,Dig) :-
    C #= (10 ^ Dig) * H3.
sumh([],[H2|L2],[H3|L3],C,Dig) :-
    Dig1 #= Dig +1,
    (10 ^ Dig)* H2 + C #= (10 ^ Dig) * H3 + C1, sumh([],L2,L3,C1,Dig1).
sumh([H1|L1],[],[H3|L3],C,Dig) :-
    Dig1 #= Dig +1,
    (10 ^ Dig) * H1 + C #= (10 ^ Dig) * H3 + C1, sumh(L1,[],L3,C1,Dig1).
sumh([H1|L1],[H2|L2],[H3|L3],C,Dig) :-
    Dig1 #= Dig +1,
    (10 ^ Dig) * H1 + (10 ^ Dig)* H2 + C #= (10 ^ Dig) * H3 + C1,
sumh(L1,L2,L3,C1,Dig1).

reverse(L,Res) :- reverseh(L,Res,[]).
reverseh([],L,L).
reverseh([H|T],L,Res) :- reverseh(T,L,[H|Res]).
```

```
% crypt1([S,E,N,D],[M,O,R,E],[M,O,N,E,Y],[D,N,E,S,R,O,M,Y]),
labeling([ff],[D,N,E,S,R,O,M,Y]).
% output:
% D = 7,
% E = 5,
% M = 1,
% N = 6,
% O = 0,
% R = 8,
% S = 9,
% Y = 2

% self-example:
% crypt1([E,D,G,W],[L,P,D,E],[W,G,E,K,L],[E,D,G,W,L,P,K]),
% labeling([ff],[E,D,G,W,L,P,K]).
% output:
% D = 7,
% E = 5,
% G = 2,
% K = 9,
% L = 6,
% P = 8,
% W = 1
```