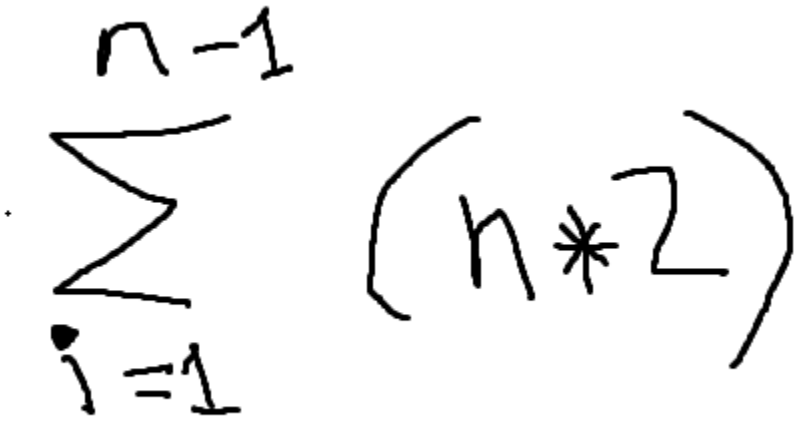
Part 1:

a) The constraints being placed on the variables A,B,C,D are abs() and int() to take the values and instantiate them



The constraints that are generated from value N input are calculated by this formula The third argument is meant to iterate through the values 1-(N-1)

b)

```
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 = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],                %6
        blocks(As, Bs, Cs),                                 %7
        blocks(Ds, Es, Fs),                                 %8
        blocks(Gs, Hs, Is).                                 %9

blocks([], [], []).                                         %10
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :
        all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),         %11
        blocks(Ns1, Ns2, Ns3).                              %12
```

```
problem(1, [[_,_,_, _,_,_, _,_,_],                    %13.
            [_,_,_, _,_,3, _,8,5],
            [_,_,1, _,2,_, _,_,_],
            [_,_,_, 5,_,7, _,_,_],
            [_,_,4, _,_,_, 1,_,_],
            [_,9,_, _,_,_, _,_,_],
            [5,_,_, _,_,_, _,7,3],
            [_,_,2, _,1,_, _,_,_],
            [_,_,_, _,4,_, _,_,9]]).
```

1. Length checks if Rows is of size 9, if it is unbound then it makes it size 9 of unbound variables, maplist then takes those 9 unbound indexes and creates a list for each index of size 9 filled with unbound values so that we essentially get a matrix
2. Append concatenates this new matrix into a new list named vs which is set to a domain of 1-9
3. This line checks if every rows list has only unique values
4. This line will save a copy of the inverting the rows lists and transposed to the columns list
5. This line checks if every list of the newly made columns have unique values within them
6. Creates the list of lists and assigning variable names to the sublists
7. Blocks will allow the 3 x 3 ‘blocks’ to be assigned with unique values
8. Blocks will allow the 3 x 3 ‘blocks’ to be assigned with unique values
9. Blocks will allow the 3 x 3 ‘blocks’ to be assigned with unique values
10. Blocks base case that will end the recursive call
11. Blocks that are imputed are checked to make sure all internal values are unique
12. Recursive call
13. Defines the sudoku output format

c)

```
:- use_module(library(clpb)).
% Example 1: You meet 2 inhabitants, A and B.
%           A says: "B is either a knight or a knave."
%           B says: "I am the opposite of who A says I am."
e1_knights(1, [A,B]) :-
   sat(A=:=card([1],[~B,B])),
   sat(B=:=(~A)).

e2_knights(2, [A,B,C]) :-
   sat(A=:=card([1], [(C * B), (~A * ~B * ~C)])),
   sat(B=:=(A * ~C)).
e3_knights(3, [A,B,C,D,E,F,G,H,I,J,K,L]) :-
   sat(K=:=J),
   sat(J=:=F),
   sat(F=:=E),
   sat(E=:=C),
   sat(C=:=A),
   sat(A=:=(~A * ~B * ~C * ~D * ~E * ~F * ~G * ~H * ~I * ~J * ~K * ~L)),
   sat(L=:=I),
   sat(I=:=H),
   sat(H=:=G),
   sat(G=:=D),
   sat(D=:=B),
   sat(B=:=(~A)).

% ?- e*_knights(Example, Knights).
```

Part 2:

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).
:- use_module(library(apply)).

crypt1([H1|L1],[H2|L2],[H3|L3],L4) :-
   L4 ins 0..9,
   H1#\=0,
   H2#\=0,
   all_distinct(L4),
   reverse([H1|L1],R1),
   reverse([H2|L2],R2),
   reverse([H3|L3],R3),
```

```
        crypt1_(R1, R2, 0, R3).

crypt1_([], [], C, [H3]) :- H3 #= C.
crypt1_([H1|T1], [H2|T2], C2, [H3|T3]) :-
    H3 #= (H1 + H2 + C2) mod 10,
    Carry #= (H1 + H2 + C2) div 10,
    crypt1_(T1, T2, Carry, T3).

% ?- 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]).
```