Exercise 2.1:
Consider the following two CSPs
\[ P_1 := \langle x + y \leq z, 4 \leq z < 6; x, y, z \in [2..6] \rangle \]
\[ P_2 := \langle a < z, x + y = a, z \geq 5; a \in [4..6], x, y, z \in [2..6] \rangle \]

a) Fix the order \( X = a, x, y, z \) between variables. Represent each constraint \( C \) of \( P_1 \) and \( P_2 \) as set of projections \( d[Y] \), where \( d \in [4..6] \times [2..6]^3 \) and \( Y \) is the subsequence of \( X \) which exactly contains the variables mentioned in \( C \) (cf. Slides II/3).

b) Give all solutions to \( P_1 \) and \( P_2 \).

c) Are \( P_1 \) and \( P_2 \) equivalent? Are they equivalent with respect to some subsequence of \( X = a, x, y, z \)?

Exercise 2.2:
Consider the following Boolean constraints (see also Slide II/22):
\[ i_1 \land o_2 = y_1 \]
\[ i_2 \land o_1 = y_2 \]
\[ \neg y_1 = o_1 \]
\[ \neg y_2 = o_2 \]

For the above constraints show two successful derivations using the Boolean constraint propagation rules given on Slides 23-24. For each derivation step you should underline the selected constraint and give the used rule. The initial CSPs are:

a) \( \langle i_1 \land o_2 = y_1, i_2 \land o_1 = y_2, \neg y_1 = o_1, \neg y_2 = o_2; i_1 = 0, i_2 = 1 \rangle \)

b) \( \langle i_1 \land o_2 = y_1, i_2 \land o_1 = y_2, \neg y_1 = o_1, \neg y_2 = o_2; o_2 = 1, i_1 = 1 \rangle \)
Exercise 2.3:

a) Implement an Eclipse-Prolog-predicate `permutation(A,B)` that generates a permutation B of a list A. All permutations can be computed by backtracking using ";" when prompted. Use the following queries to check that your program works correctly:

```prolog
:-permutation([1,2,3],B).
:-permutation(A,B).
```

b) Use the constraint solving library `ic` to write a predicate `sorted(L)` for a list L that is using constraints and is true if the elements of the list are sorted in ascending order. Implement a predicate `permsort(L,SL)` that takes a list L and generates a sorted list SL by testing permutations until a sorted permutation is found.

c) Redefine `permsort` such that it first calls `sorted` and then `permutation`. Compare the two ways of implementing `permsort` with respect to run time. Check that `permsort` also works for lists with variables, for example `permsort([1,A,3],SL).`