Note: Same rules apply as on previous assignments.

P1

(a). (6 pts) Give all resolvents ***for each pair*** of clauses below:

1. $(p \vee q \vee \sim r \vee s)$
2. $(\sim q \vee \sim r \vee \sim s)$
3. (q)
4. $(\sim q)$

(b). (1 pt) What is the result of resolving any clause against itself?
(c). (1 pt) Convert to CNF (show your work): $x \Rightarrow (y \Rightarrow z)$
(d). (1 pt) Convert to CNF (show your work): $(x \vee y) \Rightarrow z$

(e). (2 pts) Convert to CNF (show your work): $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$

(f). (2 pts) As a function of n, how many clauses would be in the CNF conversion of:

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

P2 (9 pts) Use Forward Chaining with the Knowledge Base below to determine which variables it entails and which it does not entail. Show your work! Show the final table (fill in YES if the variable is entailed, NO otherwise.

variable	L	F	B	Y	E	Z	W	T	X	N	P
entailed?	YES	YES									

0. $L \wedge F \Rightarrow B$
1. $Y \wedge E \Rightarrow Z$
2. $W \Rightarrow T$
3. $B \wedge X \Rightarrow W$
4. $F \wedge B \Rightarrow X$
5. $N \Rightarrow Z$
6. $Z \wedge Y \Rightarrow E$
7. $L \wedge P \wedge E \Rightarrow B$
8. $Z \wedge T \Rightarrow E$
9. L
10. F

P3 (10 pts) Write the method `get_resolvents(c1, c2)` which takes two clauses and returns a list of all resolvents between the two clauses. See sample output for examples.

P4 (10 pts) Copy your function from Problem 3 into the Problem 4 template file. Write the function `is_satisfiable(clauses, display)` which will call your `get_resolvents` function repeatedly, maintaining a list (called "all") of the clauses and all resolvents.

In each "round", resolve every clause in "all" against every other clause in "all", and keep track of these in a list called "new_resolvents". If any resolvent is the empty clause, then formula is not satisfiable. If at any point the list of new_resolvents would not add any new clauses to "all", then the formula is satisfiable.

This function takes a boolean variable "display". If set to True, the function should show the work being done, as shown in class and in the sample output. This will help you with debugging.

P5 (5 pts) Copy your functions from Problems 3 and 4 into the Problem 5 template file. Write the function `entails(kb, query)`. The knowledge base "kb" consists of a list of clauses. The function returns True if the knowledge base entails (i.e., implies) the query, and False otherwise.

For this problem, you will simply add the negation of the query to a copy of the knowledge base, and check if that is satisfiable. Assuming the knowedge base is satisfiable, if adding the negation of the query causes it to be unsatisfiable, then the knowledge base necessarily entails the query.

P6 (10 pts) Write the function `forward_chaining(kb, query, display = False)`. This function takes a knowledge base "kb" of clauses that are ***definite clauses*** (clauses with exactly one postive literal).

Use the forward chaining algorithm (gone over in class) to determine if "kb" entails the query. Return True if it does, and False otherwise.

If this function is called with two arguments, "kb" and "query", then the parameter "display" is set to False. You may want to use this "display" parameter to show the work being done by the function to help you debug (and understand what's going on). This is not mandatory for this problem.

P7 (+4% points extra credit) Make a copy your program from Problem 4 and chang the `is_satisfiable` function so that it never calls `get_resolvents` twice (or more) on the same pair of clauses in list "all". .