Implementation Details:

- The code has been implemented as a python script.
- In the Driver function, we take the path of the cnf file as input. We store the clauses as a list of lists which store the literals present in a clause. We store the assigned value of a variable in an array (an unassigned variable has the value 0). Also, we time our function using Python's time() library. We then call the solve() function. If it returns a None object then the formula is UNSAT else we it returns the variable assignments. Note that a 0 assignment means that either -1 or 1 would make the formula satisfiable so we arbitrarily assign it 1.
- Solve() is a recursive routine which takes a list of remaining clauses to be satisfied and an array to store the variable assignments and returns the variable assignments if a satisfying assignment is possible else None.
- The function solve() is based on backpropagation algorithm which is used as the backbone of DPLL algorithms as well. From the clause list, we pop the clause with fewest number of elements.
- For each of the literal in the popped clause, we assign it the value that will make the popped clause true, then call the function reduce() to remove the occurrences of this literal and its negation from the clause list and then call solve() again recursively on this modified clause list along with the modified variable assignment with the literal being true.
    - The reduce() function takes a list of clauses and a literal as its input and returns a modified list of clauses.
    - If the passed literal is present in any of the clause then that cluse is satisfied and we thus pop it from the clause list. Else, if its negation is present in any of the clauses, we just remove this occurrence from the list.
    - It might so happen that after a reduction step, a clause may become empty and thus unsatisfiable. To optimize our algorithm a bit, we return [[]] the trivial unsatisfiable clause list in that case.
- If we receive a satisfying assignment from solve() we return it and pass it down. Else, we assign this proposition the negation of the literal and move to the next literal in the list.
- If we exhaust the list of literals then the popped clause is not satisfiable and we return None to the calling function.

Assumptions:

- The file containing the cnf formula is not corrupt. We do check if the number of clauses match with the value mentioned against the “p cnf” line and exit if it doesn’t.
- The default recursion depth is sufficient for the entered clause. One may however increase it by uncommenting line 3 and setting it according to their choice, however this may be unsafe.

Limitations:

- Despite being based on backtracking algorithm; it is still extremely slow especially when compared to modern sat solvers which have been optimized heavily. One may solve a formula with say ~40 variables and ~150 clauses of length ~3 satisfactorily with it but not much more than that.

How to run:

- The only prerequisite for running the script is python3, so ascertain that you have it installed.
- Copy the main.py file on your machine to say and then run “python [path]” on your terminal where [path] is the path of your main.py file. Then when prompted, enter the path of the cnf file containing your formula.
- If the formula is satisfiable, a list of variables will output on your terminal. If the variable is positive, then it means it needs to be true while if it is positive then it needs to be false for the satisfying assignment.