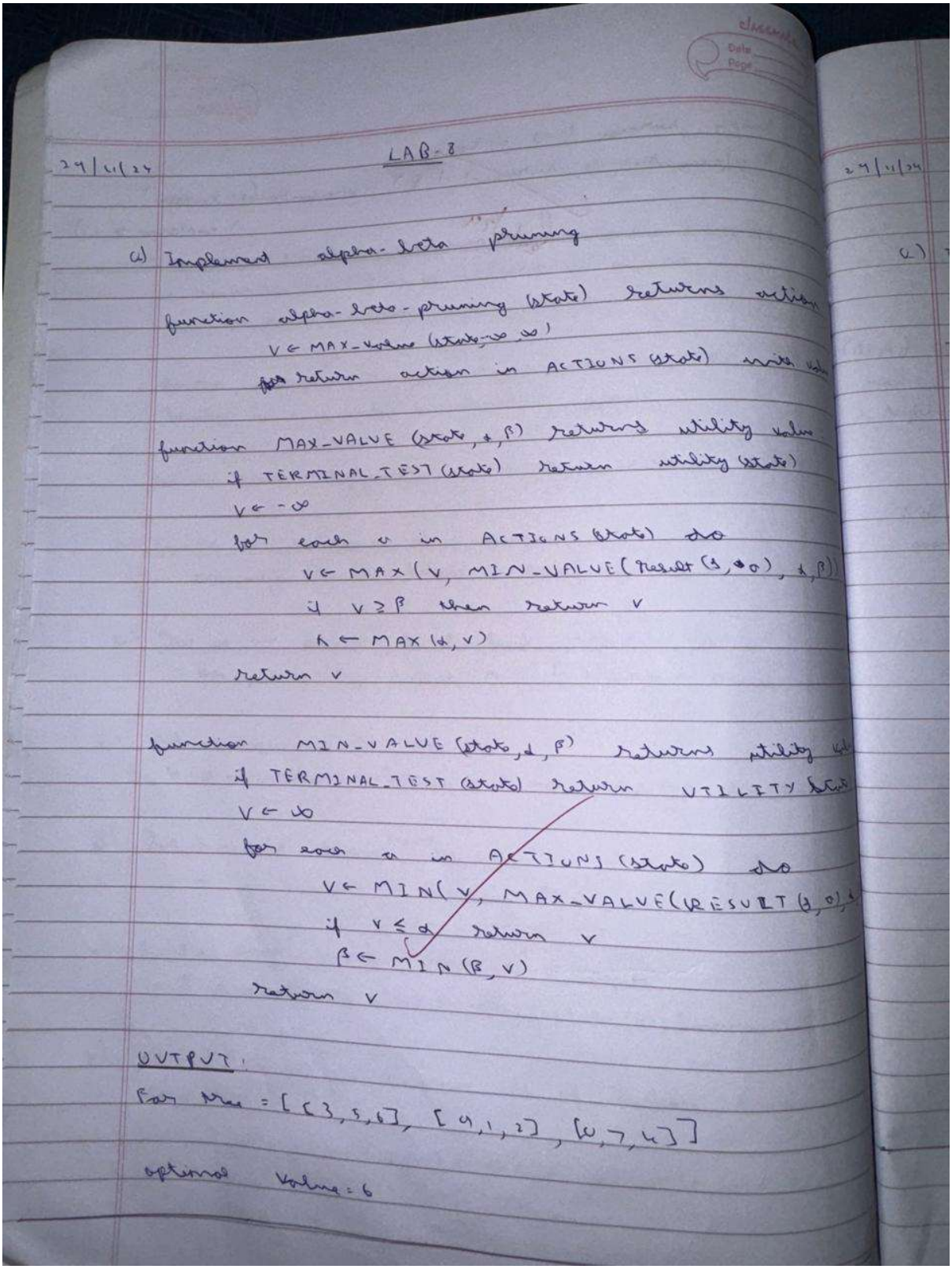
1) Alpha-Beta Pruning



29/11/24

LAB-8

a) Implement alpha-beta pruning

function alpha-beta-pruning (state) returns action
    v ← MAX-value (state, -∞, ∞)
    return action in ACTIONS (state) with value

function MAX-VALUE (state, α, β) returns utility value
    if TERMINAL_TEST (state) return utility (state)
    v ← -∞
    for each a in ACTIONS (state) do
        v ← MAX (v, MIN-VALUE (Result (s, a), α, β))
        if v ≥ β then return v
        α ← MAX (α, v)
    return v

function MIN-VALUE (state, α, β) returns utility value
    if TERMINAL_TEST (state) return UTILITY (state)
    v ← ∞
    for each a in ACTIONS (state) do
        v ← MIN (v, MAX-VALUE (RESULT (s, a), α, β))
        if v ≤ α return v
        β ← MIN (β, v)
    return v

OUTPUT:

For tree = [[3,5,6], [9,1,2], [0,7,4]]

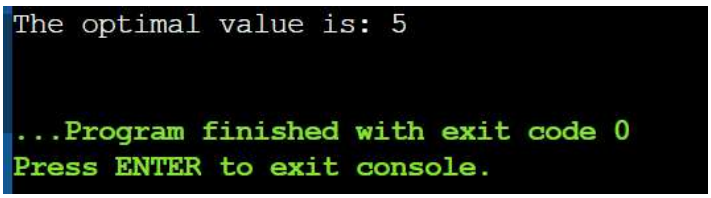
optimal Value = 6

Code

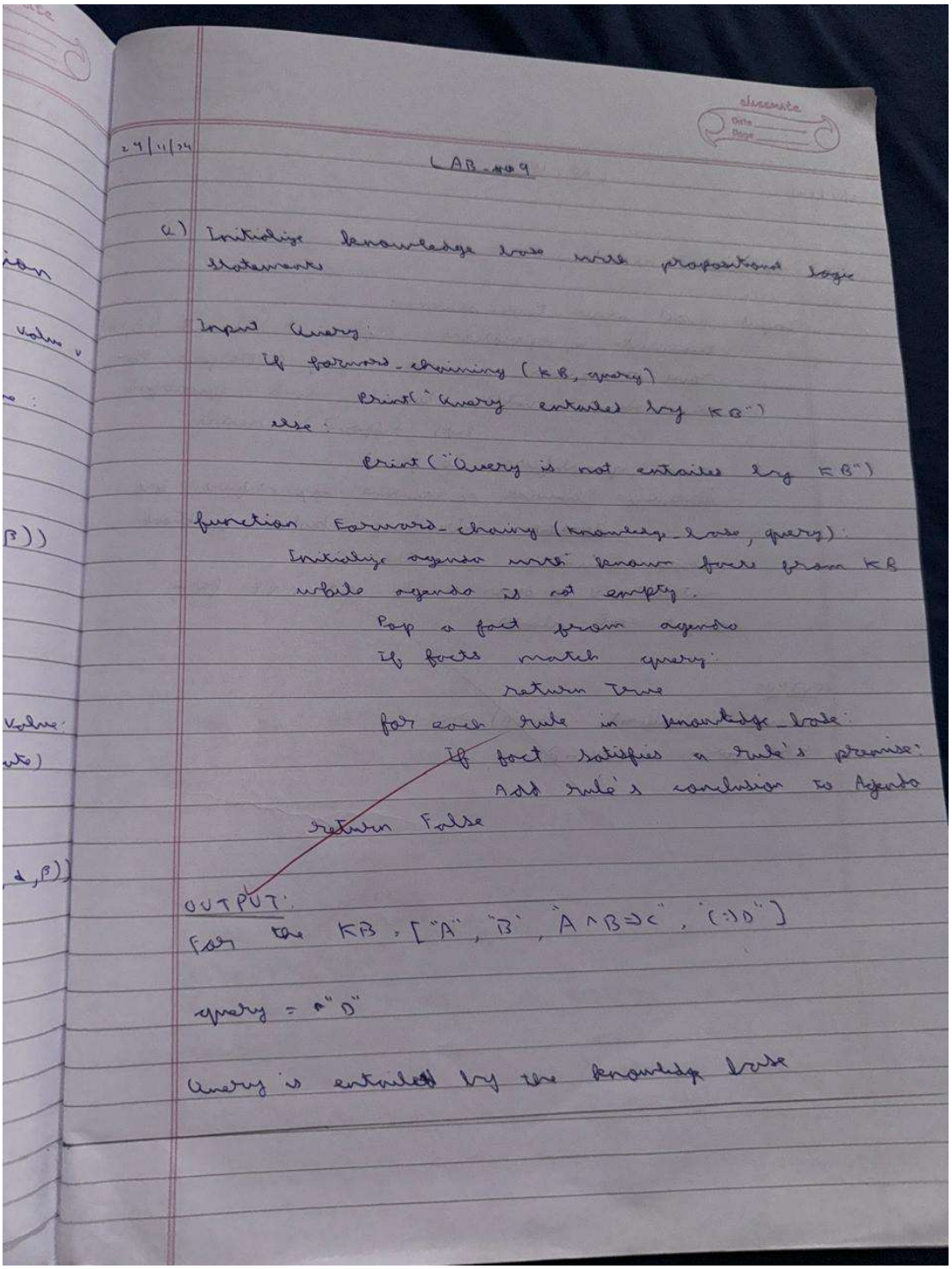
```
import math


def alpha_beta_pruning(depth, node_index, is_maximizing_player, values, alpha, beta, max_depth):
    if depth == max_depth:
        return values[node_index]


    if is_maximizing_player:
        best = -math.inf
        for i in range(2):
            val = alpha_beta_pruning(depth + 1, node_index * 2 + i, False, values, alpha, beta, max_depth)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = math.inf
        for i in range(2):
            val = alpha_beta_pruning(depth + 1, node_index * 2 + i, True, values, alpha, beta, max_depth)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
```

```
            break
        return best

if __name__ == "__main__":
    values = [3, 5, 6, 9, 1, 2, 0, -1]  # Example tree represented as a list of leaf node values
    max_depth = 3  # Height of the tree
    result = alpha_beta_pruning(0, 0, True, values, -math.inf, math.inf, max_depth)
    print("The optimal value is:", result)
```

OUTPUT:



2) Propositional Logic Statement Entailment



27/11/24

LAB-#09

a) Initialize knowledge base with propositional logic statements

Input query:

If forward_chaining (KB, query)

Print("Query entailed by KB")

else:

Print("Query is not entailed by KB")

function Forward_chaining (knowledge_base, query):

Initialize agenda with known facts from KB

while agenda is not empty:

Pop a fact from agenda

If facts match query:

return True

for each rule in knowledge_base:

If fact satisfies a rule's premise:

Add rule's conclusion to Agenda

return False

OUTPUT:

For the KB : ["A", "B", "A ∧ B ⇒ C", "C ⇒ D"]

query = "D"

Query is entailed by the knowledge base

Code:

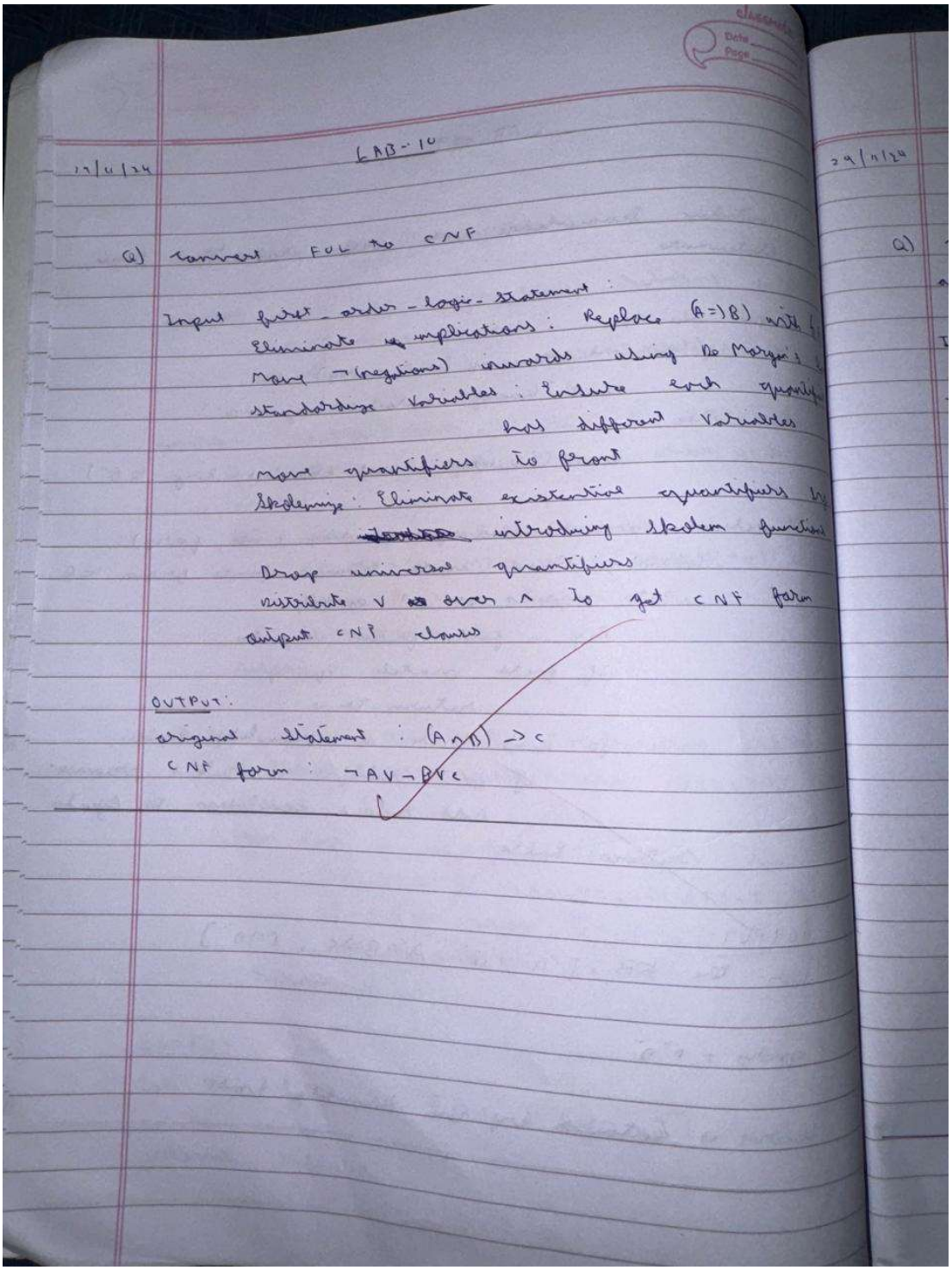
```
from sympy.logic.boolalg import Or, And, Not from sympy.abc import A, B, C, D, E, F

from sympy import simplify_logic



def is_entailment(kb, query): # Negate the query

negated_query = Not(query)



# Combine the knowledge base with the negated query

kb_with_negated_query = And(*kb, negated_query) # Combine all KB clauses and the negated query



# Simplify the combined KB to CNF (Conjunctive Normal Form) simplified_kb = simplify_logic(kb_with_negated_query, form="cnf")



# If the simplified KB evaluates to False, the query is entailed return simplified_kb == False



# Define a larger Knowledge Base (kb) kb = [

Or(A, B),        # A ∨ B Or(Not(A), C),  # ¬A ∨ C

Or(Not(B), D),  # ¬B ∨ D

Or(Not(D), E),  # ¬D ∨ E

Or(Not(E), F),  # ¬E ∨ F

]
```

```
# Query to check (C ∨ F) query = Or(C, F)

# Check entailment

result = is_entailment(kb, query)

# Output the result
print(f"Is the query '{query}' entailed by the knowledge base? {'Yes' if result else 'No'}")
```

OUTPUT:

```
Is the query 'C | F' entailed by the knowledge base? Yes
```

3) FOL to CNF



LAB - 10

17/4/24

Q) Convert FOL to CNF

Input first_order_logic_statement

Eliminate implications: Replace (A=>B) with (

Move ¬(negations) inwards using De Morgan's

standardize variables: Ensure each quantifi

has different variables

move quantifiers to front

Skolemize: Eliminate existential quantifiers by

introducing Skolem functions

Drop universal quantifiers

Distribute ∨ over ∧ to get CNF form

Output CNF clauses

OUTPUT:

original statement : (A∧B) -> C

CNF form : ¬A∨¬B∨C

29/11/24

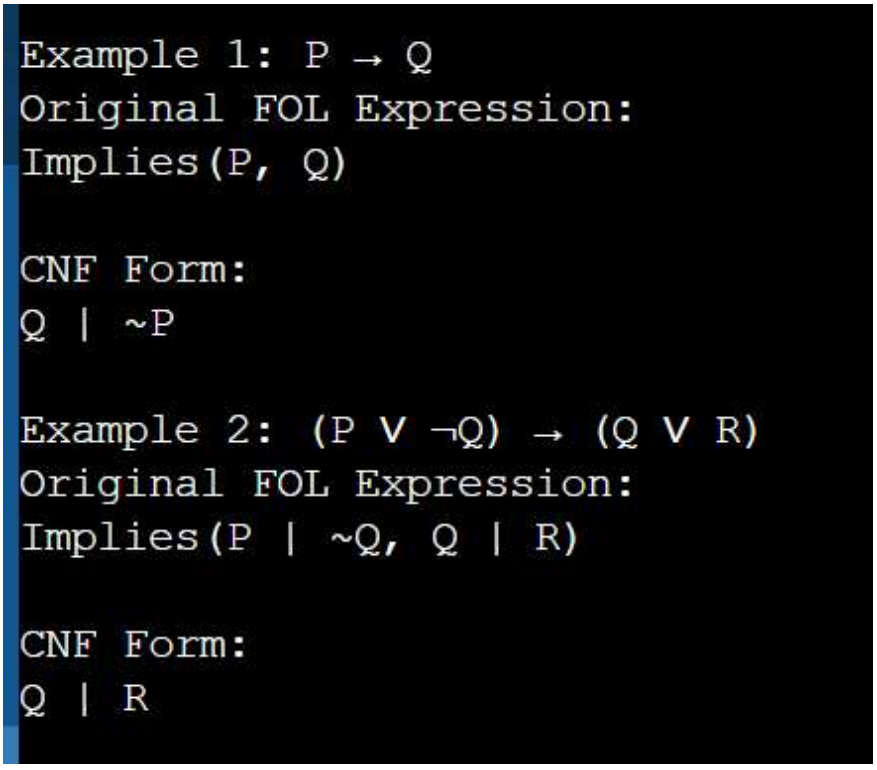
Q)

Code:

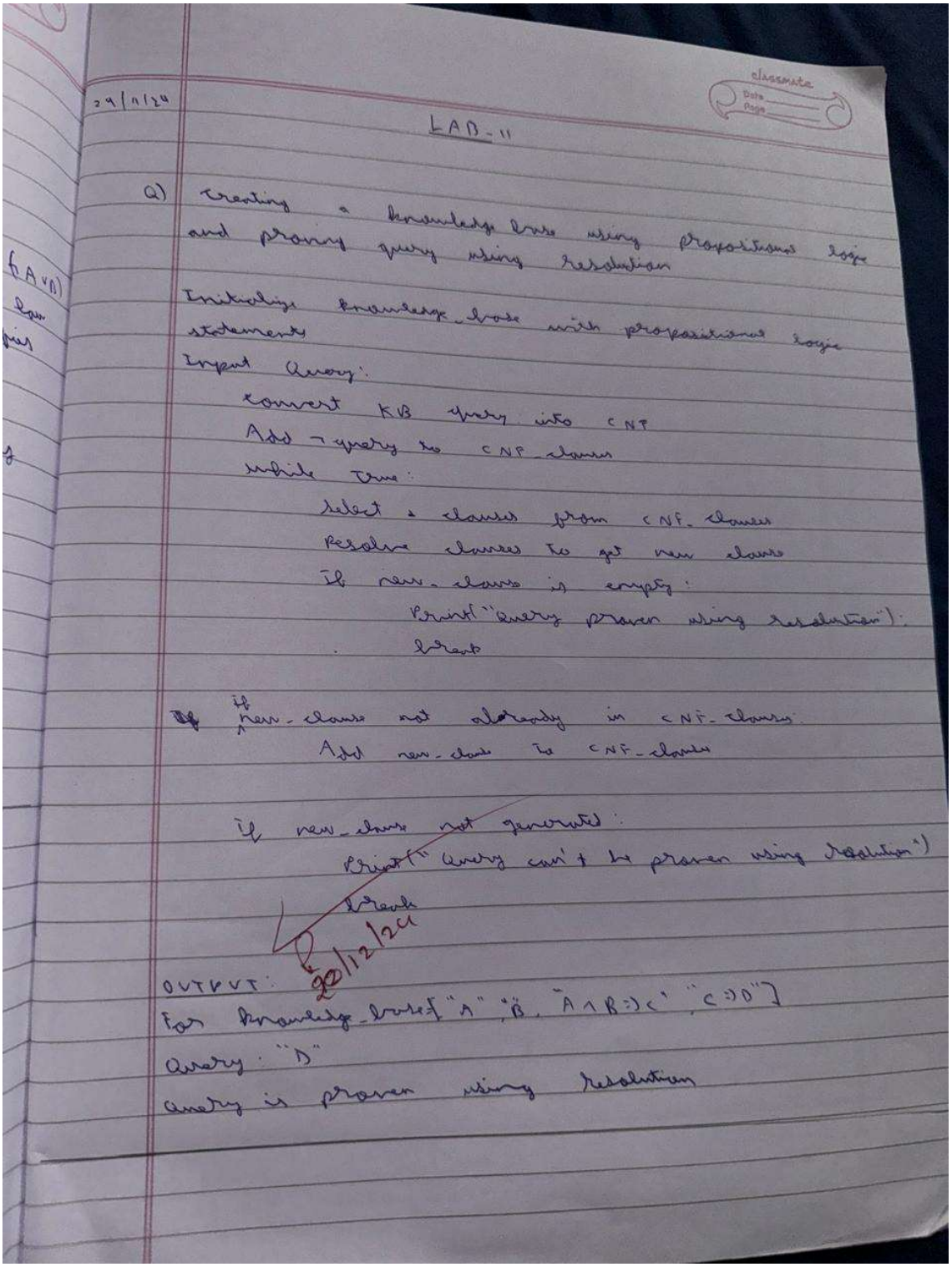
```
from sympy import symbols, Not, Or, And, Implies, Equivalent
from sympy.logic.boolalg import to_cnf

def fol_to_cnf(fol_expr):
    fol_expr = fol_expr.replace(Equivalent, lambda a, b: And(Implies(a, b), Implies(b, a)))
    fol_expr = fol_expr.replace(Implies, lambda a, b: Or(Not(a), b))
    cnf_form = to_cnf(fol_expr, simplify=True)
    return cnf_form

def main():
    P = symbols("P")
    Q = symbols("Q")
    R = symbols("R")

    fol_expr1 = Implies(P, Q)
    print("Example 1: P → Q")
    print("Original FOL Expression:")
    print(fol_expr1)
    cnf1 = fol_to_cnf(fol_expr1)
    print("\nCNF Form:")
    print(cnf1)

    fol_expr2 = Implies(Or(P, Not(Q)), Or(Q, R))
    print("\nExample 2: (P ∨ ¬Q) → (Q ∨ R)")
    print("Original FOL Expression:")
```

```
    print(fol_expr2)
    cnf2 = fol_to_cnf(fol_expr2)
    print("\nCNF Form:")
    print(cnf2)

if __name__ == "__main__":
    main()
```

OUTPUT:



4) Proving by Resolution



29/11/24

LAB - 11

Q) Creating a knowledge base using propositional logic and proving query using resolution

Initialize knowledge_base with propositional logic statements

Input query:

convert KB query into CNF

Add ¬query to CNF_clauses

while True:

select 2 clauses from CNF_clauses

Resolve clauses to get new_clause

If new_clause is empty:

Print("Query proven using resolution")

break

If new_clause not already in CNF_clauses:

Add new_clause to CNF_clauses

If new_clause not generated:

Print("Query can't be proven using resolution")

break

22/12/24

OUTPUT:

for knowledge_base=["A" "B" "A ∧ B => C" "C => D"]

Query: "D"

Query is proven using resolution

Code:

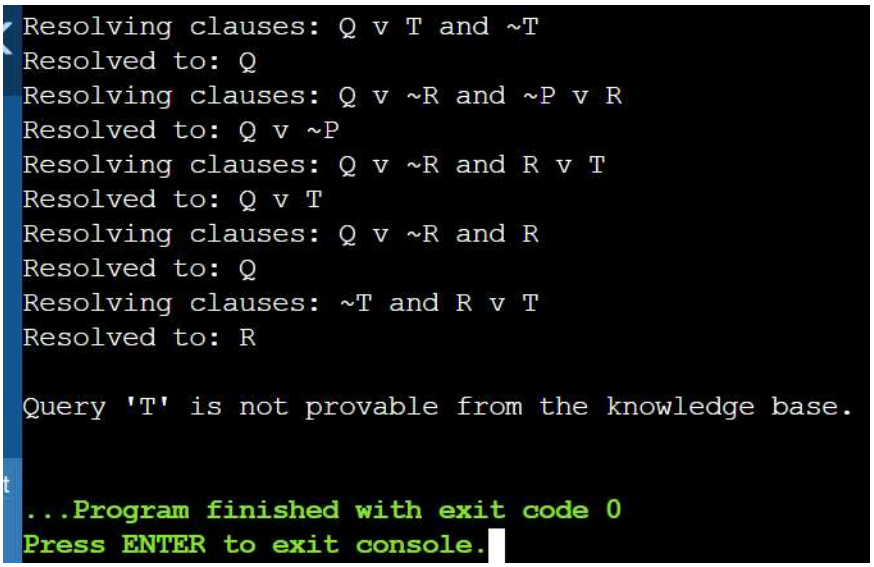
```
def negation(p):
    if p.startswith("~"):
        return p[1:]
    return f"~{p}"

def resolution(kb, query):
    kb.append(negation(query))
    new_clauses = set(kb)
    print(f"Initial Knowledge Base + negation of query: {kb}")

    while True:
        added_new_clause = False
        clauses = list(new_clauses)
        for i in range(len(clauses)):
            for j in range(i + 1, len(clauses)):
                clause1 = clauses[i]
                clause2 = clauses[j]
                resolvent = resolve(clause1, clause2)

                if resolvent is not None:
                    print(f"Resolving clauses: {clause1} and {clause2}")
                    print(f"Resolved to: {resolvent}")

                    if not resolvent:
                        return True

                    if resolvent not in new_clauses:
```

```
                    new_clauses.add(resolvent)
                    added_new_clause = True

        if not added_new_clause:
            break

    return False

def resolve(clause1, clause2):
    literals1 = set(clause1.split(" v "))
    literals2 = set(clause2.split(" v "))

    for literal in literals1:
        neg_literal = negation(literal)
        if neg_literal in literals2:
            new_clause = literals1.union(literals2) - {literal, neg_literal}
            return " v ".join(sorted(new_clause))

    return None

kb = [
    "P v Q",
    "~P v R",
    "Q v ~R",
    "R v T"
]

query = "T"
```

```
result = resolution(kb, query)


if result:
    print(f"\nQuery '{query}' is provable from the knowledge base.")
else:
    print(f"\nQuery '{query}' is not provable from the knowledge base.")
```

OUTPUT:

```
Initial Knowledge Base + negation of query: ['P v Q', '~P v R', 'Q v ~R', 'R v T', '~T']
Resolving clauses: P v Q and ~P v R
Resolved to: Q v R
Resolving clauses: Q v ~R and ~P v R
Resolved to: Q v ~P
Resolving clauses: Q v ~R and R v T
Resolved to: Q v T
Resolving clauses: ~T and R v T
Resolved to: R
Resolving clauses: Q v R and Q v ~R
Resolved to: Q
Resolving clauses: P v Q and Q v ~P
Resolved to: Q
Resolving clauses: P v Q and ~P v R
Resolved to: Q v R
Resolving clauses: Q v T and ~T
Resolved to: Q
Resolving clauses: Q v ~R and ~P v R
Resolved to: Q v ~P
Resolving clauses: Q v ~R and R v T
Resolved to: Q v T
Resolving clauses: Q v ~R and R
Resolved to: Q
Resolving clauses: ~T and R v T
Resolved to: R
Resolving clauses: Q v R and Q v ~R
Resolved to: Q
Resolving clauses: P v Q and Q v ~P
Resolved to: Q
Resolving clauses: P v Q and ~P v R
Resolved to: Q v R
```


Resolving clauses: Q v T and ~T
Resolved to: Q
Resolving clauses: Q v ~R and ~P v R
Resolved to: Q v ~P
Resolving clauses: Q v ~R and R v T
Resolved to: Q v T
Resolving clauses: Q v ~R and R
Resolved to: Q
Resolving clauses: ~T and R v T
Resolved to: R
Query 'T' is not provable from the knowledge base.
...Program finished with exit code 0
Press ENTER to exit console.