VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Artificial Intelligence

Submitted by

Kshitij S(1BM21CS093)

Under the Guidance of

Prof. Asha GR
Assistant Professor, BMSCE

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

November 2023-February 2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum) Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Artificial Intelligence**" carried out by **Kshitij(1BM21CS093)** , who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24.

The Lab report has been approved as it satisfies the academic requirements in respect of **Artificial Intelligence - (22CS5PCAIN)** work prescribed for the said degree.

Prof. Asha GR	Dr. Jyothi Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Kshitij S(1BM21CS093), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Artificial Intelligence** " has been carried out by me under the guidance of Prof. Asha GR, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

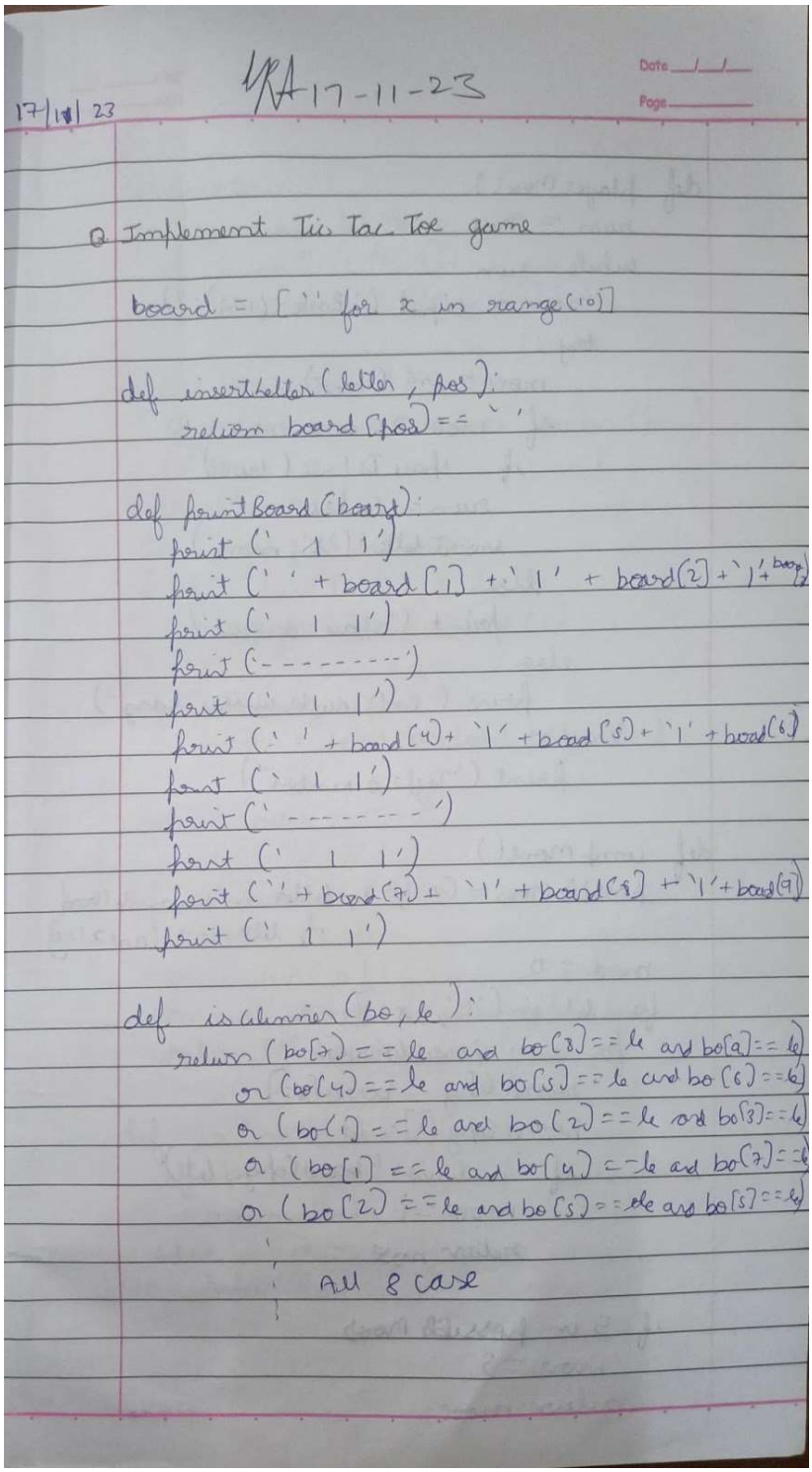
TABLE OF CONTENTS

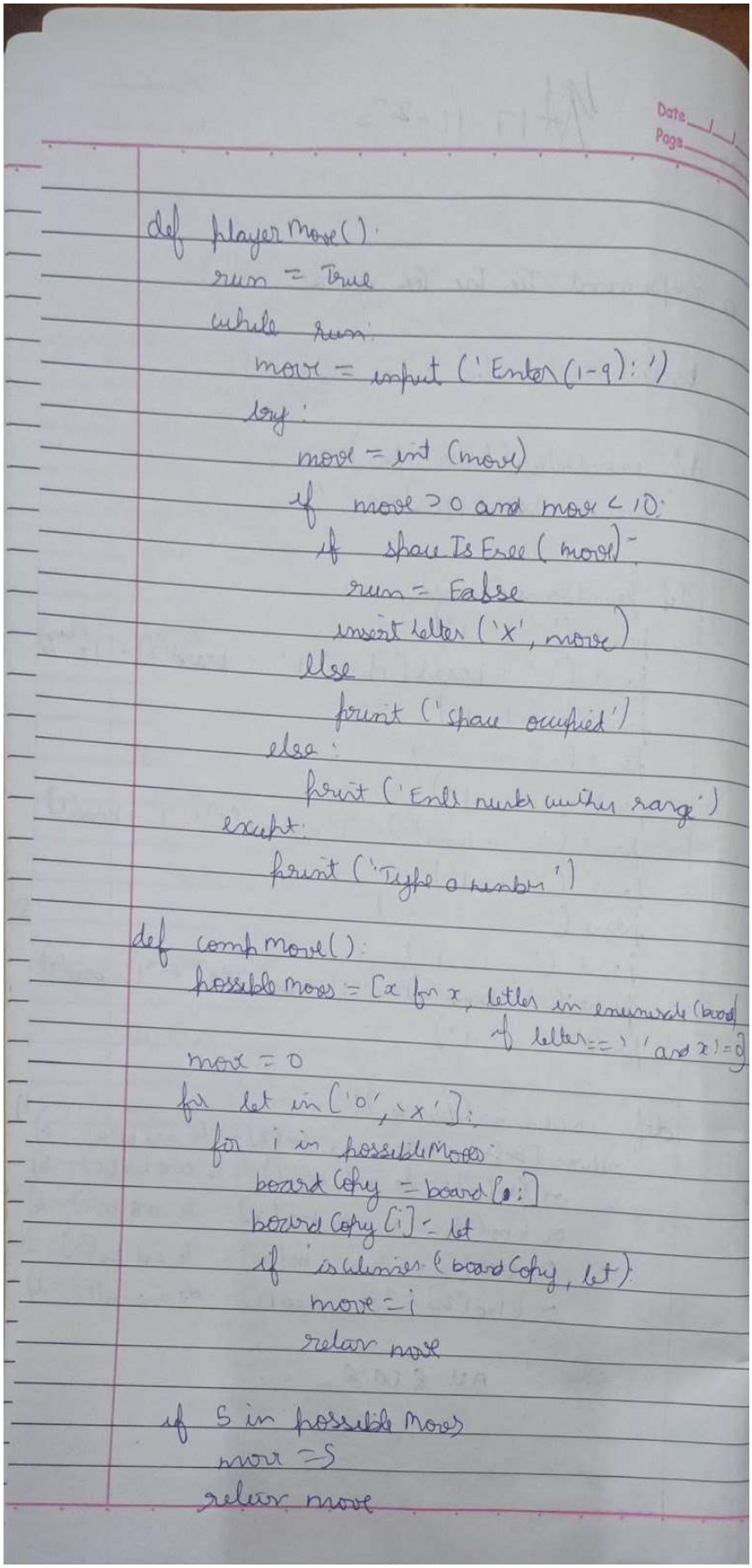
LAB: - 1

Aim

Implement Tic –Tac –Toe Game.

Observation Notebook





```
def playerMove():
    run = True
    while run:
        move = input('Enter(1-9):')
        try:
            move = int(move)
            if move > 0 and move < 10:
                if spaceIsFree(move):
                    run = False
                    insertLetter('X', move)
                else
                    print('space occupied')
            else:
                print('Ents number within rang')
        except:
            print('Type a number')

def compMove():
    possibleMoves = [x for x, letter in enumerate(board)
                     if letter == ' ' and x != 0]
    move = 0
    for let in ['O', 'X']:
        for i in possibleMoves:
            boardCopy = board[0:]
            boardCopy[i] = let
            if isWinner(boardCopy, let):
                move = i
                return move

    if 5 in possibleMoves
        move = 5
    return move
```

```
def main():
    print('Welcome')
    print board(board)
    while not(is Board Full(board)):
        if not(winner(board, 'O')):
            player Move()
            print Board(board)
        else:
            print('O won')
            break

        if not(is winner(board, 'X')):
            move = comp Move()
            if move == 0:
                print('Tie!')
            else:
                insert letter('O', move)
                print('Computer has a O in', move)
                print Board(board)
        else:
            print('X won')
            break
    if is Board Full(board):
        print('Tie Game!')

while True:
    answer = input('Do you want to play (Y/N)')
    if answer.lower() == 'y' or answer.lower() == 'yes':
        board = [' ' for x in range(10)]
        print('-----')
        main()
    else:
        break
```

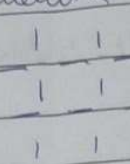
Date __/__/__

Page _____

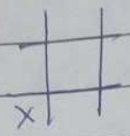
Output

Do you want to play (Y/N) Y

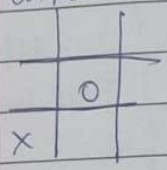
Welcome



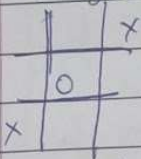
Select Position : 7



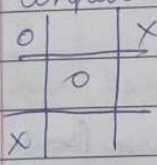
Computer has O in position 5



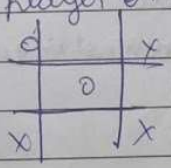
Player Move = 3



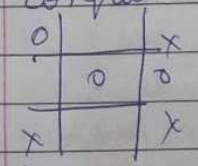
Computer has O in position 1



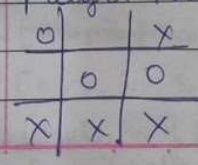
Player Move = 9



Computer move : 6



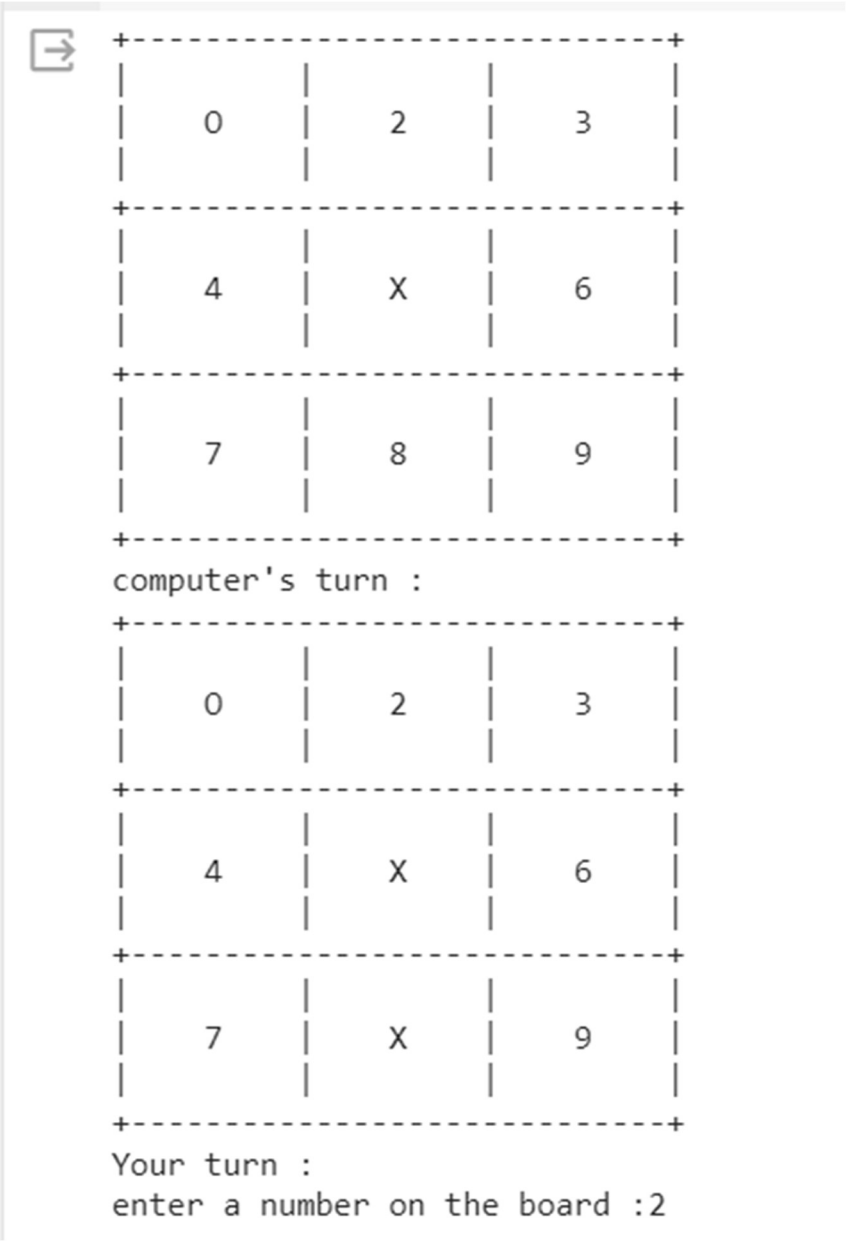
Player move : 8



X wins Do you want to play again (Y/N) N

Output

```
Kshitj S-1BM21CS093
[1, 2, 3, 4, 5, 6, 7, 8, 9]
+-------------------------------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   5   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
computer's turn :
+-------------------------------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------------------------------+
Your turn :
enter a number on the board :1
```


+-----------------------------+
0	2	3
+-----------------------------+
4	X	6
+-----------------------------+
7	8	9
+-----------------------------+
computer's turn :
+-----------------------------+
0	2	3
+-----------------------------+
4	X	6
+-----------------------------+
7	X	9
+-----------------------------+
Your turn :
enter a number on the board :2

```
+-------------------------------+
|               |               |
|       0       |       0       |       3       |
|               |               |
+-------------------------------+
|               |               |
|       4       |       X       |       6       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       X       |       9       |
|               |               |
+-------------------------------+
computer's turn :
+-------------------------------+
|               |               |
|       0       |       0       |       X       |
|               |               |
+-------------------------------+
|               |               |
|       4       |       X       |       6       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       X       |       9       |
|               |               |
+-------------------------------+
Your turn :
enter a number on the board :7
```

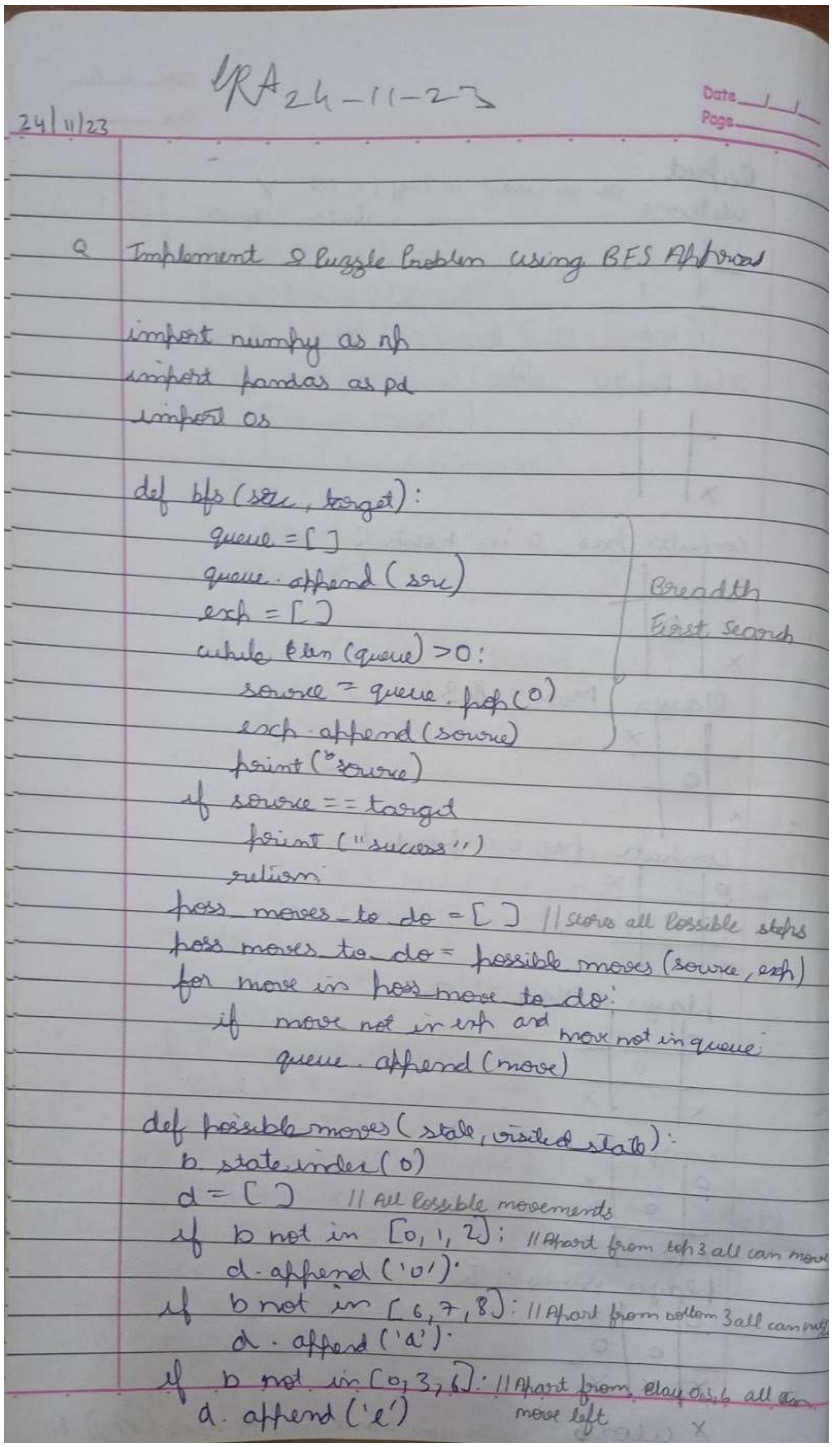
```
+-----------------------------+
|             |             |             |
|      O      |      O      |      X      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      4      |      X      |      6      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      O      |      X      |      9      |
|             |             |             |
+-----------------------------+
computer's turn :
+-----------------------------+
|             |             |             |
|      O      |      O      |      X      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      X      |      X      |      6      |
|             |             |             |
+-----------------------------+
|             |             |             |
|      O      |      X      |      9      |
|             |             |             |
+-----------------------------+
Your turn :
enter a number on the board :6
```

```
+-------------------------------+
|       |       |       |
|   O   |   O   |   X   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   X   |   X   |   O   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   O   |   X   |   9   |
|       |       |       |
+-------------------------------+
computer's turn :
+-------------------------------+
|       |       |       |
|   O   |   O   |   X   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   X   |   X   |   O   |
|       |       |       |
+-------------------------------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------------------------------+
```

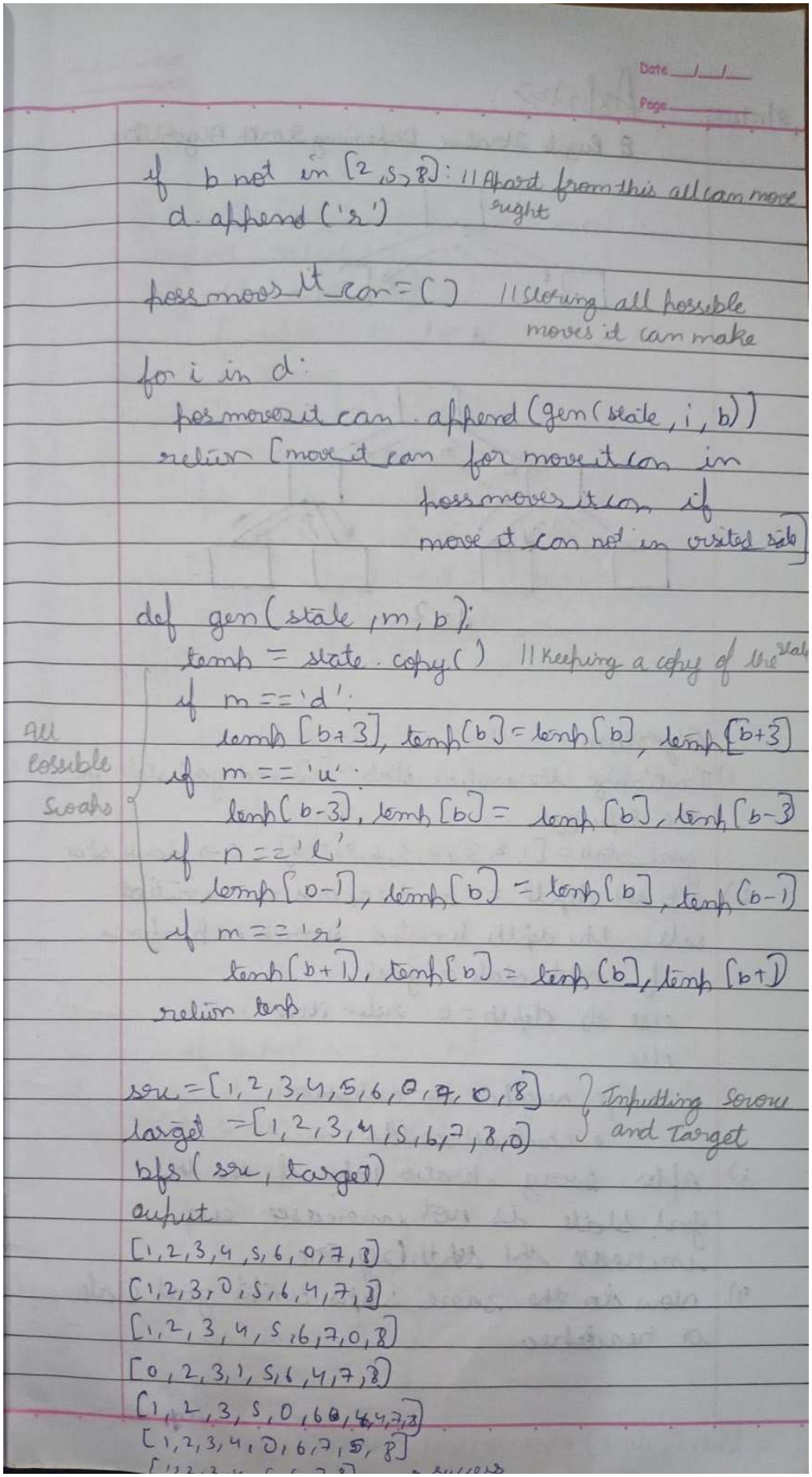
LAB: - 2

Aim
Solve 8 puzzle problems.

Observation Notebook



```
LRA 24-11-23                                   Date __/__/__
24/11/23                                       Page

Q  Implement 8 puzzle problem using BFS Algorithm

import numpy as np
import pandas as pd
import os

def bfs(src, target):
    queue = []
    queue.append(src)                          Breadth
    exp = []                                   First Search
    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)
        print("source")
    if source == target
        print("success")
        return
poss_moves_to_do = []  // stores all possible steps
poss_moves_to_do = possible_moves(source, exp)
for move in poss_move_to_do:
    if move not in exp and move not in queue:
        queue.append(move)

def possible_moves(state, visited_state):
    b.state.index(0)
    d = []  // All possible movements
    if b not in [0, 1, 2]:  // Apart from 1st 3 all can move
        d.append('u').
    if b not in [6, 7, 8]:  // Apart from bottom 3 all can move
        d.append('d').
    if b not in [0, 3, 6]:  // Apart from elay 0, 3, 6 all can
        d.append('l')                          move left
```



```
if b not in [2,5,8]: // Apart from this all can move right
    d.append('r')

poss_moves_it_can = []  // Storing all possible moves it can make
for i in d:
    poss_moves_it_can.append(gen(state, i, b))
return [move_it_can for move_it_can in poss_moves_it_can if move_it_can not in visited_lst]

def gen(state, m, b):
    temp = state.copy()  // Keeping a copy of the state
    if m == 'd':
        temp[b+3], temp[b] = temp[b], temp[b+3]
    if m == 'u':
        temp[b-3], temp[b] = temp[b], temp[b-3]
    if m == 'l':
        temp[b-1], temp[b] = temp[b], temp[b-1]
    if m == 'r':
        temp[b+1], temp[b] = temp[b], temp[b+1]
    return temp
```

All possible swaps

```
src = [1,2,3,4,5,6,0,7,8]
target = [1,2,3,4,5,6,7,8,0]
bfs(src, target)
```

Inputting source and target

output

```
[1,2,3,4,5,6,0,7,8]
[1,2,3,0,5,6,4,7,8]
[1,2,3,4,5,6,7,0,8]
[0,2,3,1,5,6,4,7,8]
[1,2,3,5,0,6,4,7,8]
[1,2,3,4,0,6,7,5,8]
```

Output

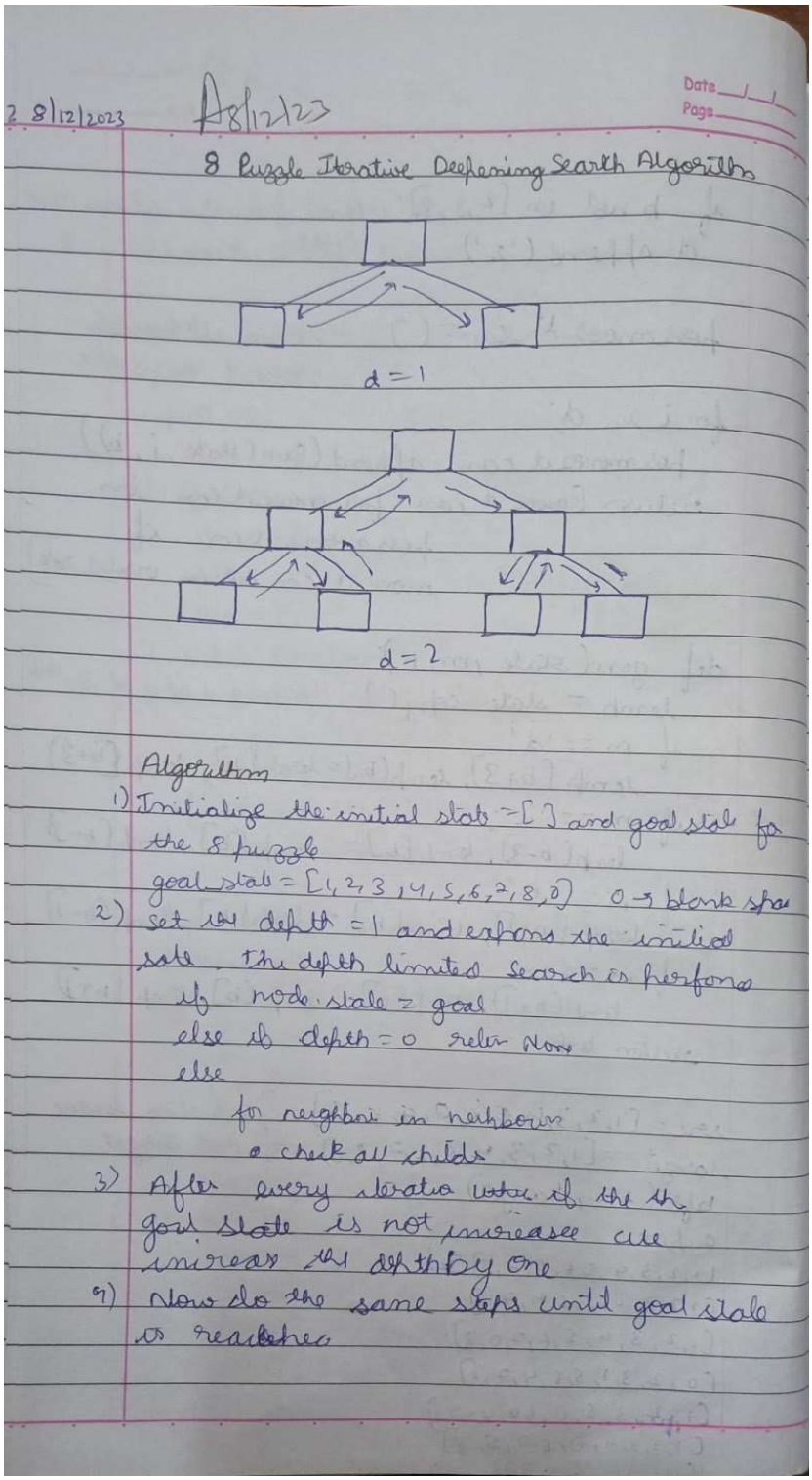
```
Kshitj S-1BM21CS093
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8

1 | 2 | 3
0 | 5 | 6
4 | 7 | 8

1 | 2 | 3
4 | 5 | 6
7 | 0 | 8

0 | 2 | 3
1 | 5 | 6
4 | 7 | 8

1 | 2 | 3
5 | 0 | 6
4 | 7 | 8

1 | 2 | 3
4 | 0 | 6
7 | 5 | 8

1 | 2 | 3
4 | 5 | 6
7 | 8 | 0

success
```

LAB: - 3

Aim

Implement Iterative deepening search algorithm.

Observation Notebook



Date __/__/__

Page _____

Code

```
def id_dfs(puzzle, goal, get_moves)
    import itertools
    def dfs(route, depth):
        if depth == 0:
            return
        if route[-1] == goal:
            return route
        for move in get_moves(route[-1]):
            if move not in route:
                next_route = dfs(route + [move], depth-1)
                if next_route:
                    return next_route

    for depth in itertools.count():
        route = dfs([puzzle], depth)
        if route:
            return route

def possible_moves(state):
    b = state.index(0);
    d = [0]
    if b not in [0,1,2]:
        d.append('u')
    if b not in [6,7,8]:
        d.append('d');
    if b not in [0,3,6]:
        d.append('l')
    if b not in [2,5,8]:
        d.append('r')
    pos_moves = []
    for i in d:
        pos_moves.append(generate(state, i, b))
    return pos_moves
```

```
def generate ( state, m, b ):
    temp = state.copy()
    if m == 'd':
        temp[b+3], temp[b] = temp[b], temp[b+3]
    if m == 'u':
        temp[b-3], temp[b] = temp[b], temp[b-3]
    if m == 'l':
        temp[b-1], temp[b] = temp[b], temp[b-1]
    if m == 'r':
        temp[b+1], temp = temp[b], temp[b+1]
    return temp

initial = [1,2,3,0,4,6,7,5,8]
goal = [1,2,3,4,5,6,7,8,0]

route = id_dfs(initial, goal, possible_moves)
if route:
    print("Success!!")
    print("Path", route)
else:
    print("Failed")
```

Output

Success!!

Path : [[1,2,3,0,4,6,7,5,8], [1,2,3,4,0,6,7,5,8],
[1,2,3,4,5,6,7,0,8], [1,2,3,4,5,6,7,8,0]]

Output

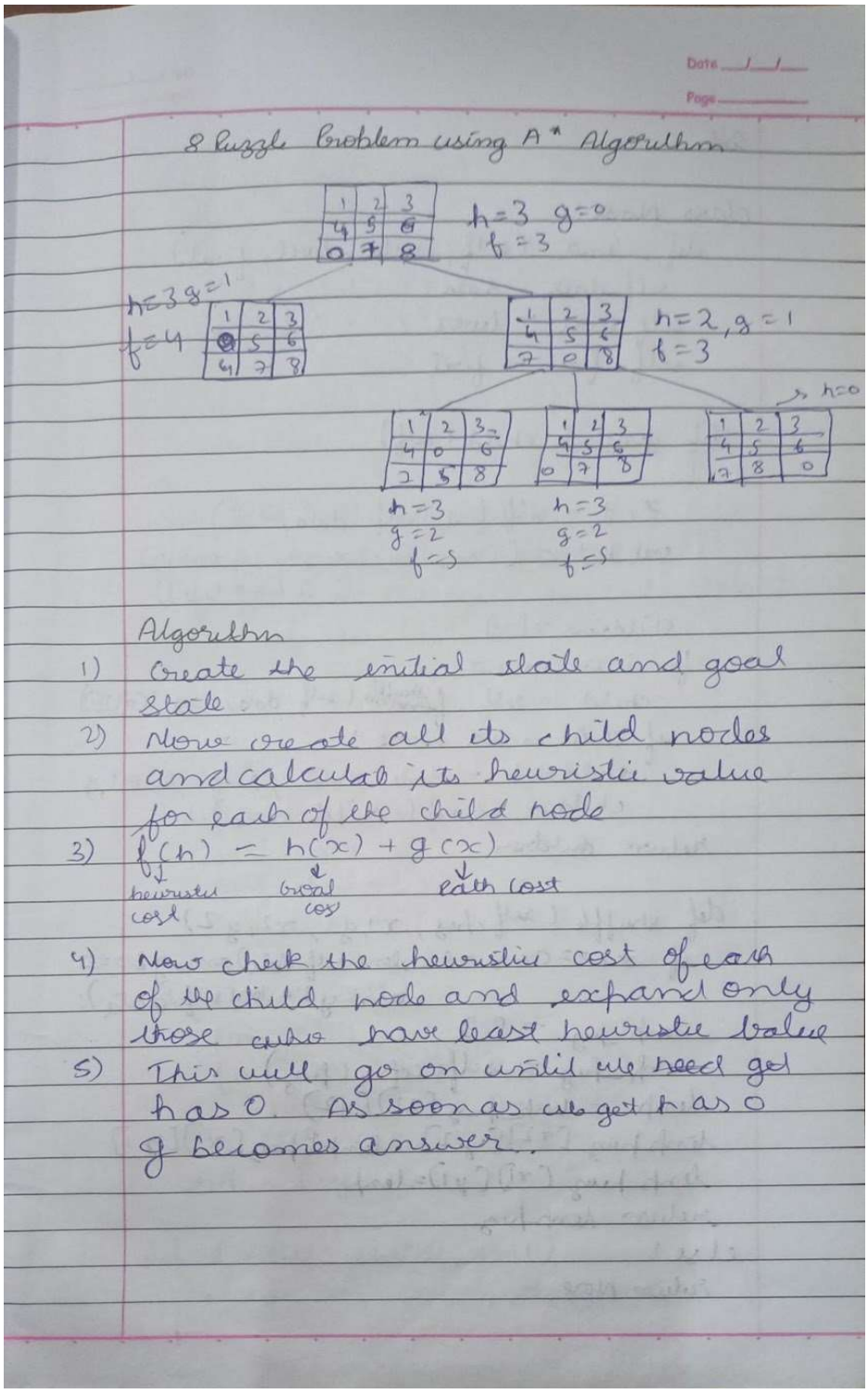
```
Kshitj S-1BM21CS093
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
```

LAB: - 4

Aim

Implement A* search algorithm.

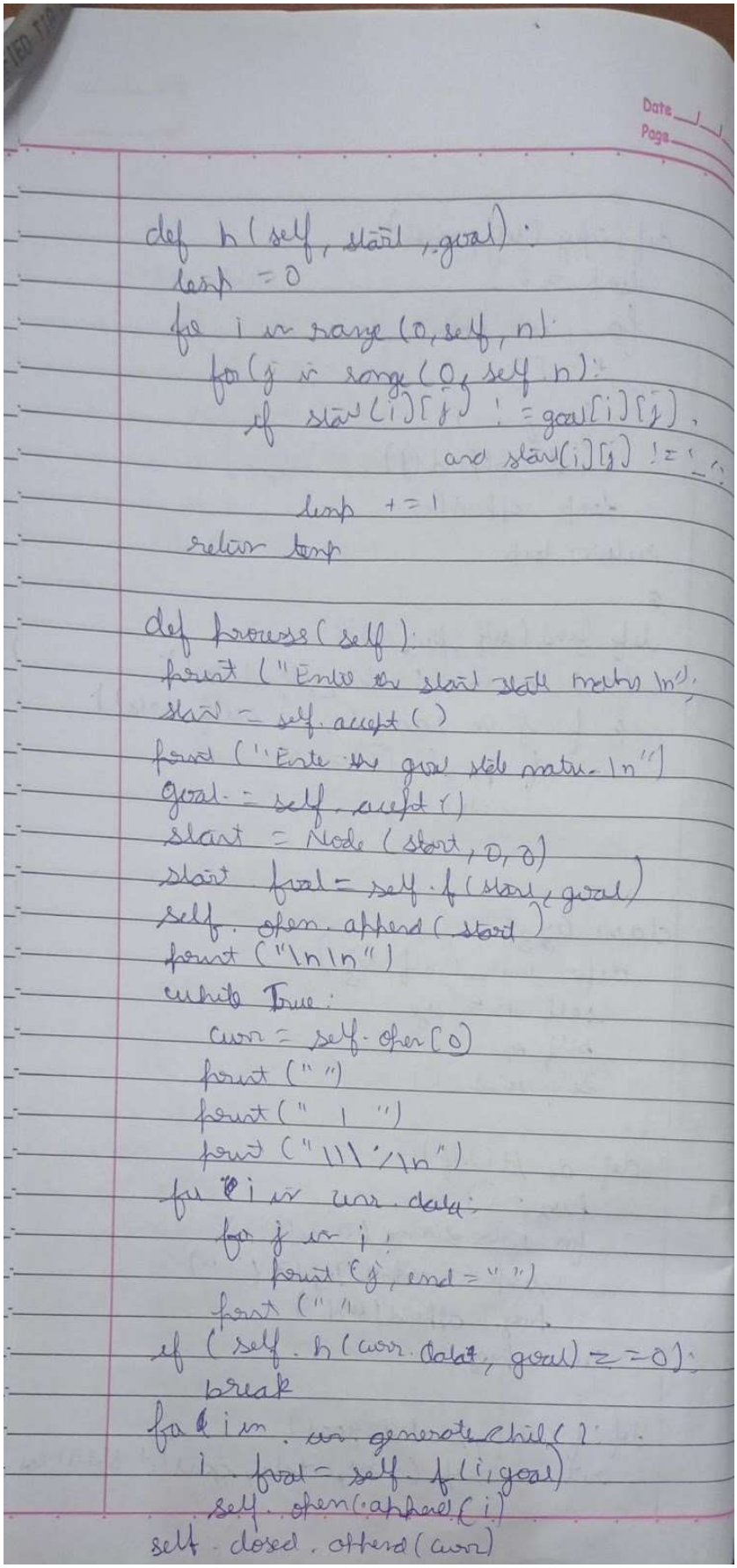
Observation Notebook



Code

```
class Node:
    def __init__(self, data, level, fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):

        x, y = self.find(self.data, '_')
        val_list = [[x, y-1], [x, y+1], [x-1, y],
                    [x+1, y]]

        children = []
        for i in val_list:
            child = self.shuffle(self.data, x, y, i[0], i[1])
            if child is not None:
                child_node = Node(child, self.level+1, 0)
                children.append(child_node)
        return children

    def shuffle(self, puz, x1, y1, x2, y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0
                                    and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None
```

```
def copy(self, root):
    temp = []
    for i in root:
        t = []
        for j in i:
            t.append(j)
        temp.append(t)
    return temp

@
def find(self, puz, x):
    for i in range(0, len(self.data)):
        for j in range(0, len(self.data)):
            if puz[i][j] == x:
                return i,j


class puzzle:
    def __init__(self, size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0, self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self, start, goal):
        return self.h(start.data, goal) + start.lvl
```



```
def h(self, start, goal):
    temp = 0
    for i in range(0, self.n):
        for j in range(0, self.n):
            if start[i][j] != goal[i][j] and start[i][j] != '_':
                temp += 1
    return temp

def process(self):
    print("Enter the start state matrix \n")
    start = self.accept()
    print("Enter the goal state matrix \n")
    goal = self.accept()
    start = Node(start, 0, 0)
    start.fval = self.f(start, goal)
    self.open.append(start)
    print("\n\n")
    while True:
        cur = self.open[0]
        print("")
        print("  |")
        print("  |")
        print(" \\\'/ \n")
        for i in cur.data:
            for j in i:
                print(j, end=" ")
            print("")
        if (self.h(cur.data, goal) == 0):
            break
        for i in cur.generate_child():
            i.fval = self.f(i, goal)
            self.open.append(i)
        self.closed.append(cur)
```

```
del self.open[0]

self.open.sort(key = lambda x: x.fval,
                        reverse = False)


puz = Puzzle(3)
puz.process()
```

Output

Enter the start state matrix

```
1 2 3
4 5 6
_ 7 8
```

Enter the goal state matrix

```
1 2 3
4 5 6
7 8 _
```

↓

```
1 2 3
4 5 6
_ 7 8
```

↓

```
1 2 3
4 5 6
7 _ 8
```

↓

```
1 2 3
4 5 6
7 8 _
```

Output

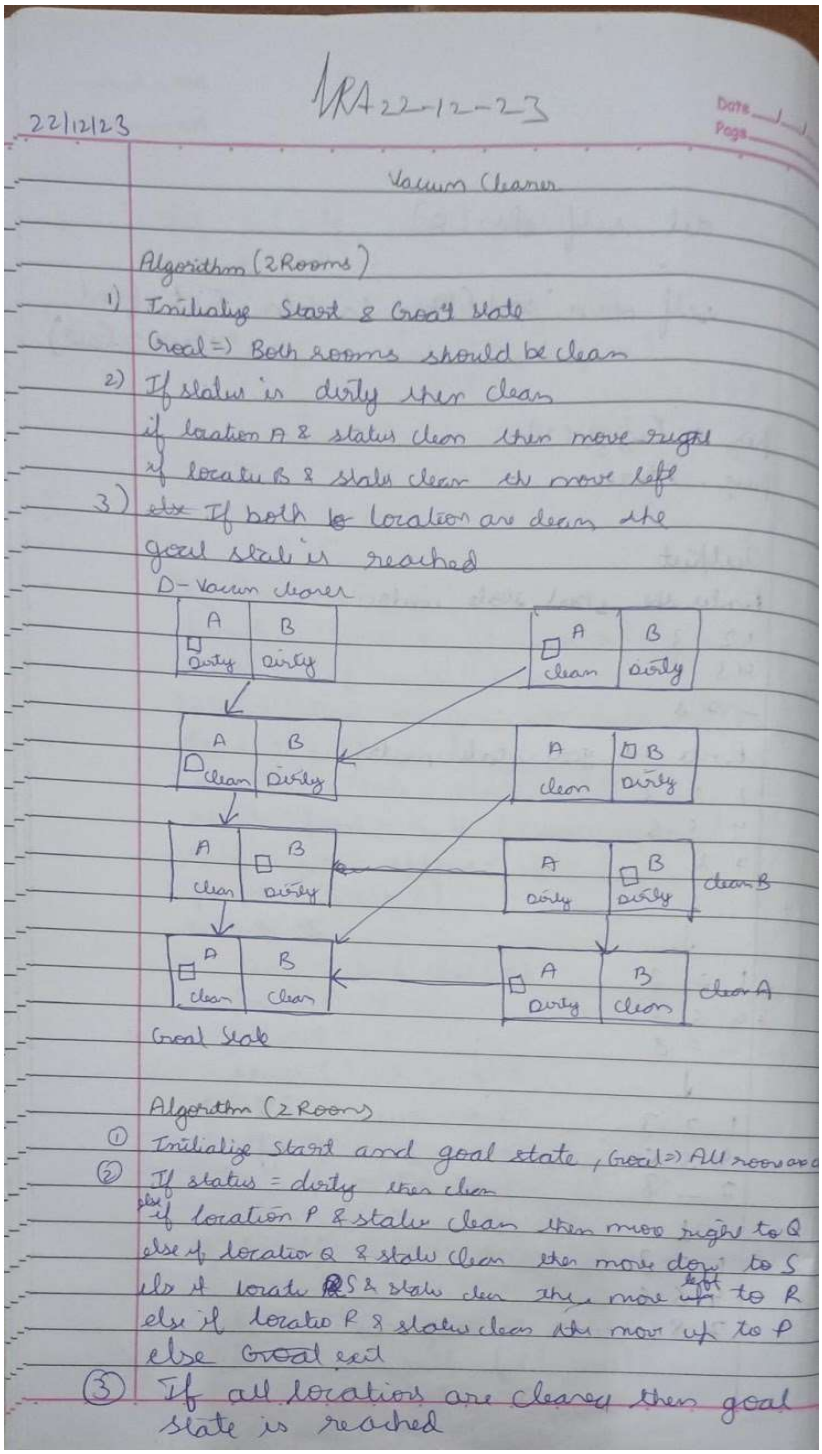
```
Kshitj S-1BM21CS093
Enter the start state matrix

1 2 3
4 5 6
_ 7 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _




  |
  |
 \'/

1 2 3
4 5 6
_ 7 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

1 2 3
4 5 6
7 8 _
```

LAB: - 5

Aim

Implement vacuum cleaner agent.

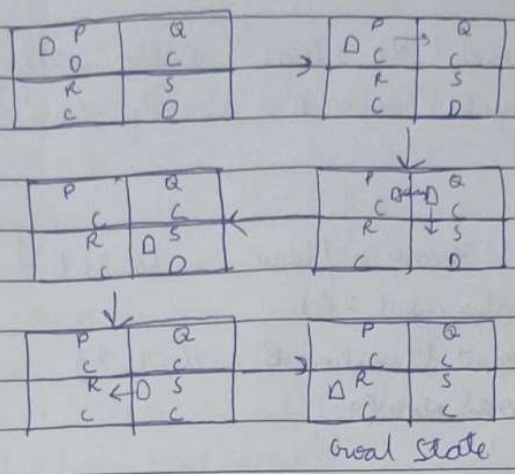
Observation Notebook



Date __/__/__

Page _____

4 Rooms Imput = 1010 (P Q R S) Room No - P Q R S D → Dirty, C → Clean, D → Vacuum



Code for Two Rooms

```
def vacuum_world():
    goal_state = {'A': 0, 'B': 0}
    cost = 0
    location_input = input("Enter location of Vacuum")
    status_input = int(input("Enter status of " + location_input))
    status_input_complement = int(input("Enter status of other room"))
    print("Goal Condition" + str(goal_state))
    if location_input == 'A':
        print("Vacuum is placed in location A")
        if status_input == 1:
            print("Location A is Dirty")
            goal_state['A'] = 0
            cost += 1
            print("cost of cleaning A" + str(cost))
            print("Location A is being cleaned")
    if status_input_complement == 1:
        print("Location B is Dirty")
        print("moving right to Location B")
        cost += 1
        print("cost to Suck" + str(cost))
```

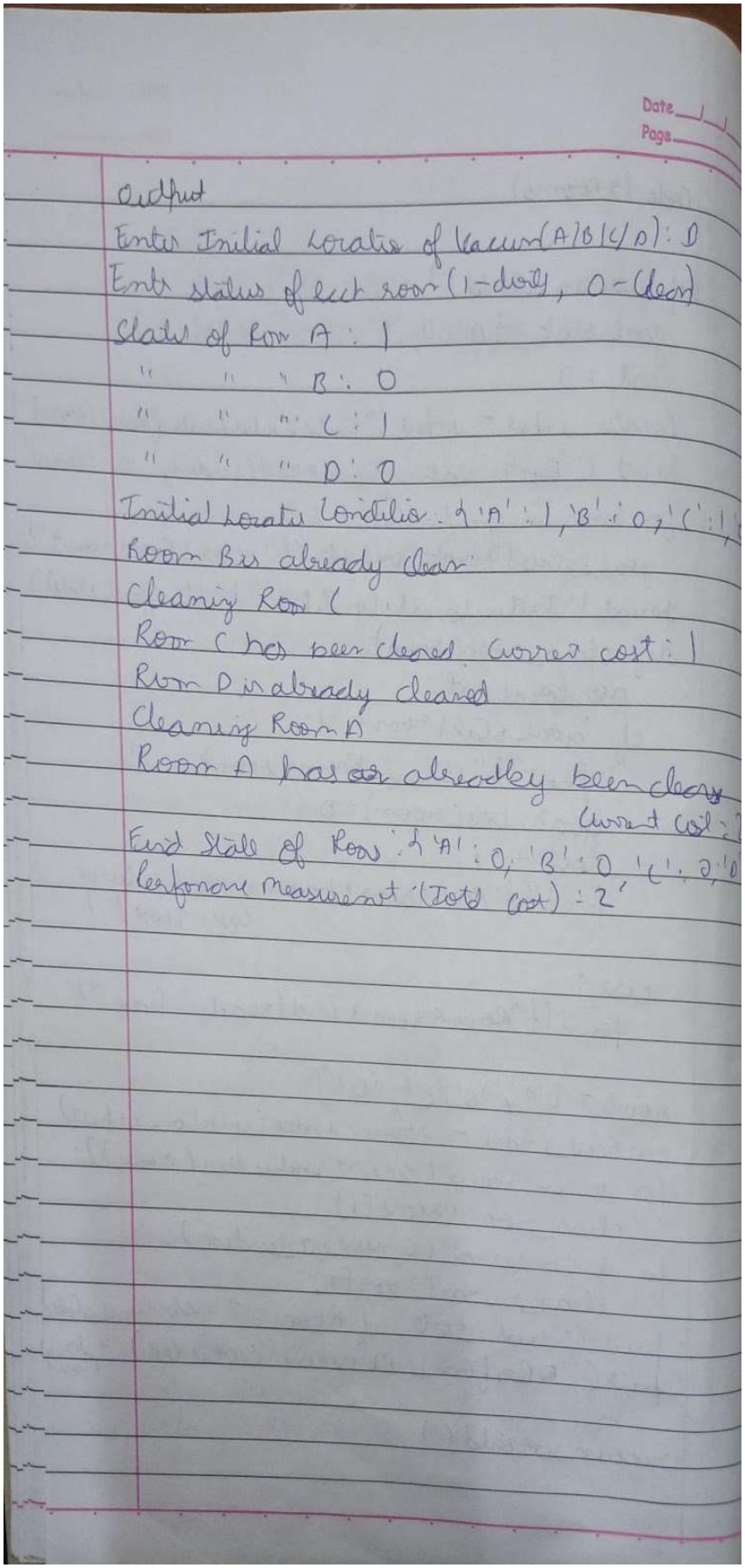
```
else:
    print ("No action " + str(cost))
    print ("location B is already clean")

else:
    print ("Vaccum is placed in locatio B")
    if status_input == 1:
        print ("location B is dirty")
        goal_state[
            :
            :
            \                    Same as A

print ("Goal State:")
print (goal_state)
print ("Performance Measurement:" + str(cost))

vaccum_world()
```

Output

```
Enter location of Vaccum = 2
Enter status of 1 1
Enter status of the other room 0
Goal Condition {'A': 0, 'B': 0}
Vaccum is placed in Locatio B
location B is Dirty
Cost for Cleaning 1
location B has been cleaned
Goal State: {'A': 0, 'B': 0}
Performance Measurement: 1
```

Code (4 Rooms)

```
def vacuum_world():
    goal_state = {'A': 0, 'B': 0, 'C': 0, 'D': 0}
    cost = 0
    location_input = input("Enter Initial Location of Vacuum (A/B/C/D): ")
    print("Enter state of each room (1 - dirty, 0 - clean)")
    for room in goal_state:
        goal_state[room] = int(input(f"State of Room {room}: "))
    print("Initial Location Condition: " + str(goal_state))
    def clean_room(room):
        nonlocal cost
        if goal_state[room] == 1:
            print(f"Cleaning Room {room}...")
            goal_state[room] = 0
            cost += 1
            print(f"Room {room} has been Cleaned. Current cost: {cost}")
        else:
            print(f"Room {room} is already clean")
    rooms = ['A', 'B', 'C', 'D']
    current_index = rooms.index(location_input)
    for i in range(current_index, len(rooms)):
        clean_room(rooms[i])
    for i in range(0, current_index):
        clean_room(rooms[i])
    print("Final State of Rooms: " + str(goal_state))
    print("Performance Measurement (Total Cost): " + str(cost))

vacuum_world()
```



Output

Enter Initial Location of Vacuum (A/B/C/D): D
Enter status of each room (1-dirty, 0-clean)
State of Room A: 1
" " " B: 0
" " " C: 1
" " " D: 0
Initial Location Condition: {'A': 1, 'B': 0, 'C': 1,
Room B is already clean
Cleaning Room C
Room C has been cleaned. Current cost: 1
Room D is already cleaned
Cleaning Room A
Room A has already been clean
Current cost: 2
End State of Room: {'A': 0, 'B': 0, 'C': 0, 'D': 0
Performance Measurement: (Total Cost) = 2

Output

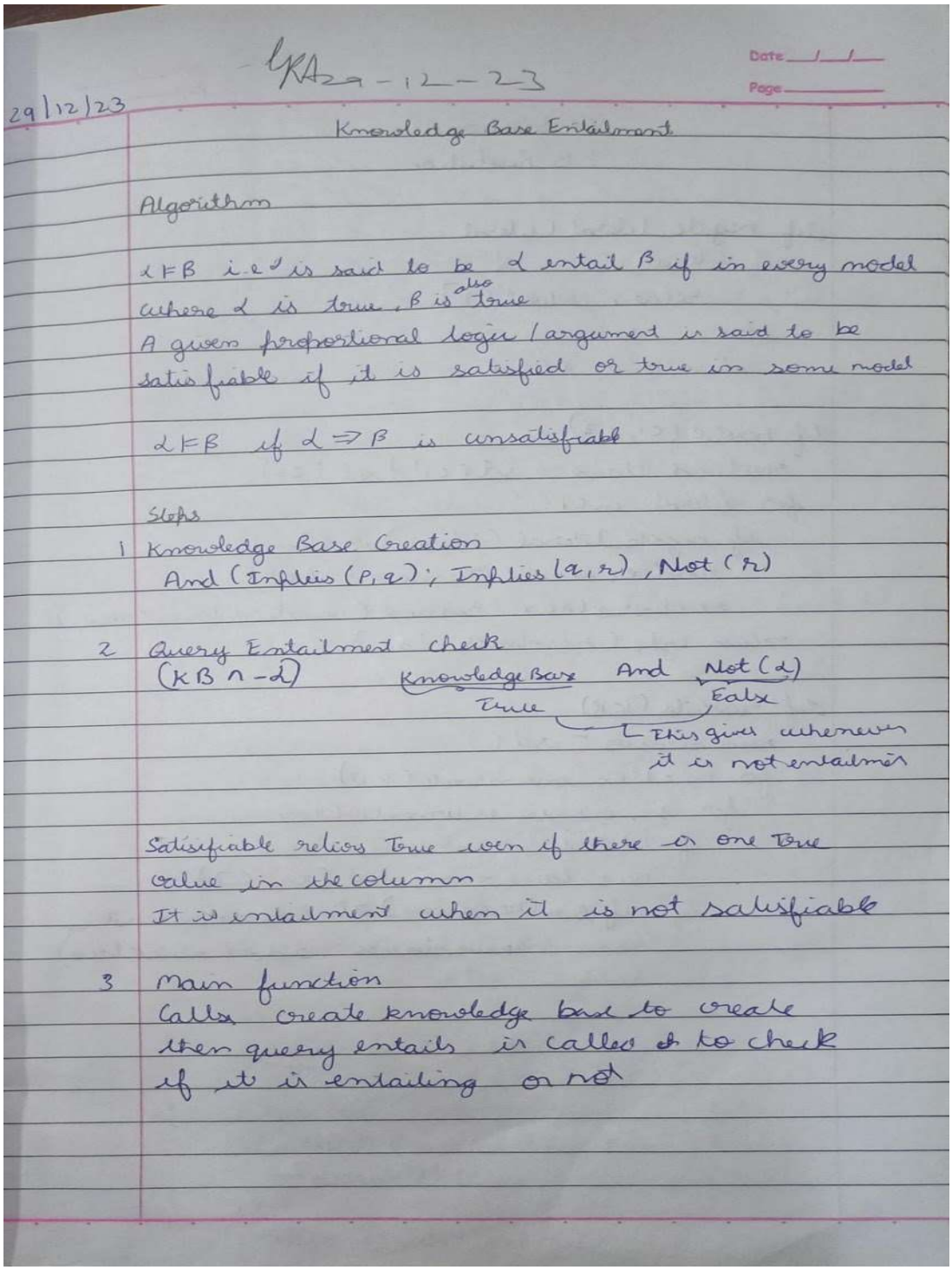
```
Kshitij S-1BM21CS093
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 0
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Room 2 is already clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.


Kshitij S-1BM21CS093
Enter clean status for Room at (1, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (1, 2) (1 for dirty, 0 for clean): 0
Enter clean status for Room at (2, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (2, 2) (1 for dirty, 0 for clean): 1
Cleaning Room at (1, 1) (Room was dirty)
Room is now clean.
Room at (1, 2) is already clean.
Cleaning Room at (2, 1) (Room was dirty)
Room is now clean.
Cleaning Room at (2, 2) (Room was dirty)
Room is now clean.
Returning to Room at (1, 1) to check if it has become dirty again:
Room at (1, 1) is already clean.
```

LAB: - 6

Aim

Create a knowledge base using prepositional logic and show that the given query entails the knowledge base or not.

Observation Notebook



Date __/__/__
Page _____

Knowledge Base Entailment Code

```
from sympy import symbols, And, Not, Implies, satisfiable

def create_knowledge_base():
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    knowledge_base = And(
        Implies(p, q),
        Implies(q, r),
        Not(r)
    )
    return knowledge_base

def query_entails(knowledge_base, Not(query)):
    entailment = satisfiable(And(knowledge_base, Not(query)))
    return not entailment

if __name__ == "__main__":
    kb = create_knowledge_base()
    query = symbols('p')
    result = query_entails(kb, query)

    print("Knowledge Base", kb)
    print("Query:", query)
    print("Query entails Knowledge Base:", result)
```

Output

Knowledge Base: ~r & (Implies(p,q)) & (Implies(q,r))

Query: p

Query entails Knowledge Base: False

Output

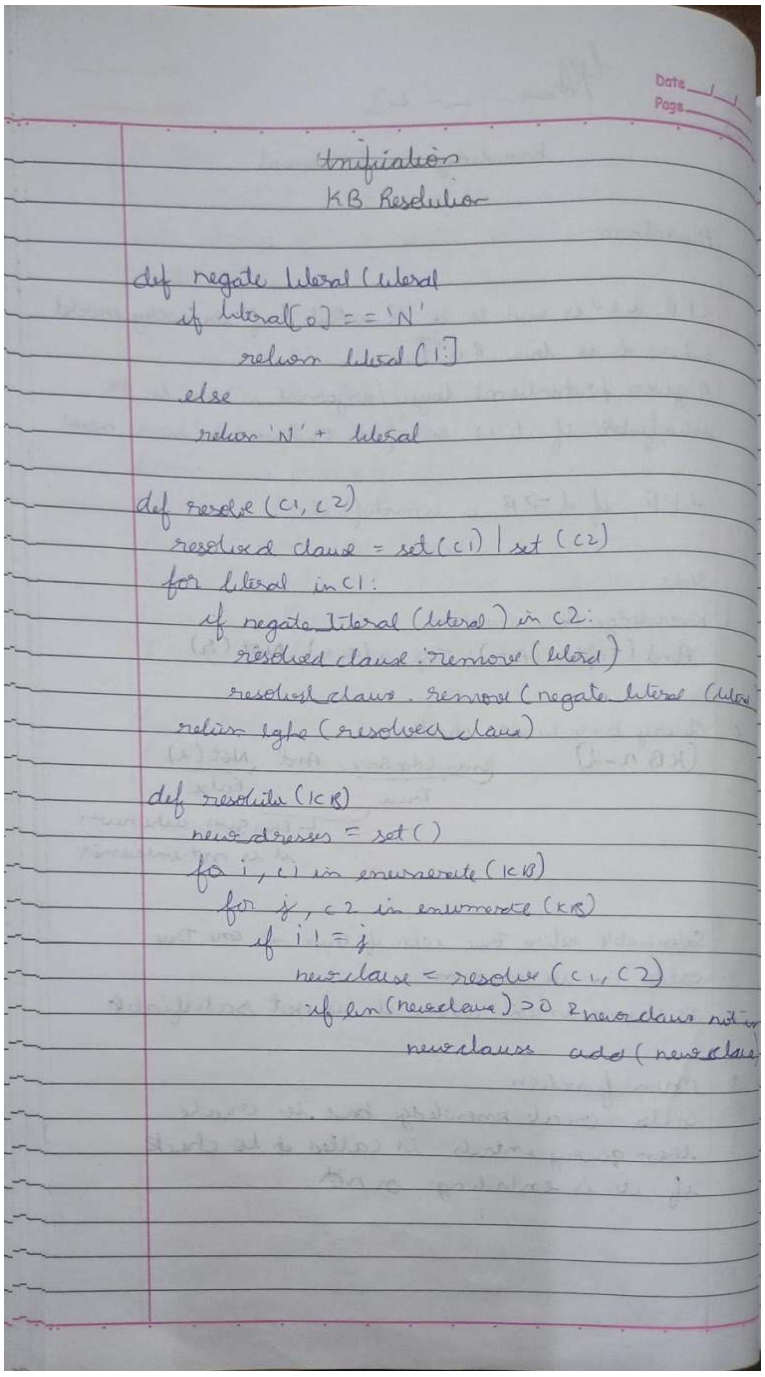
```
Kshitj S-1BM21CS093
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

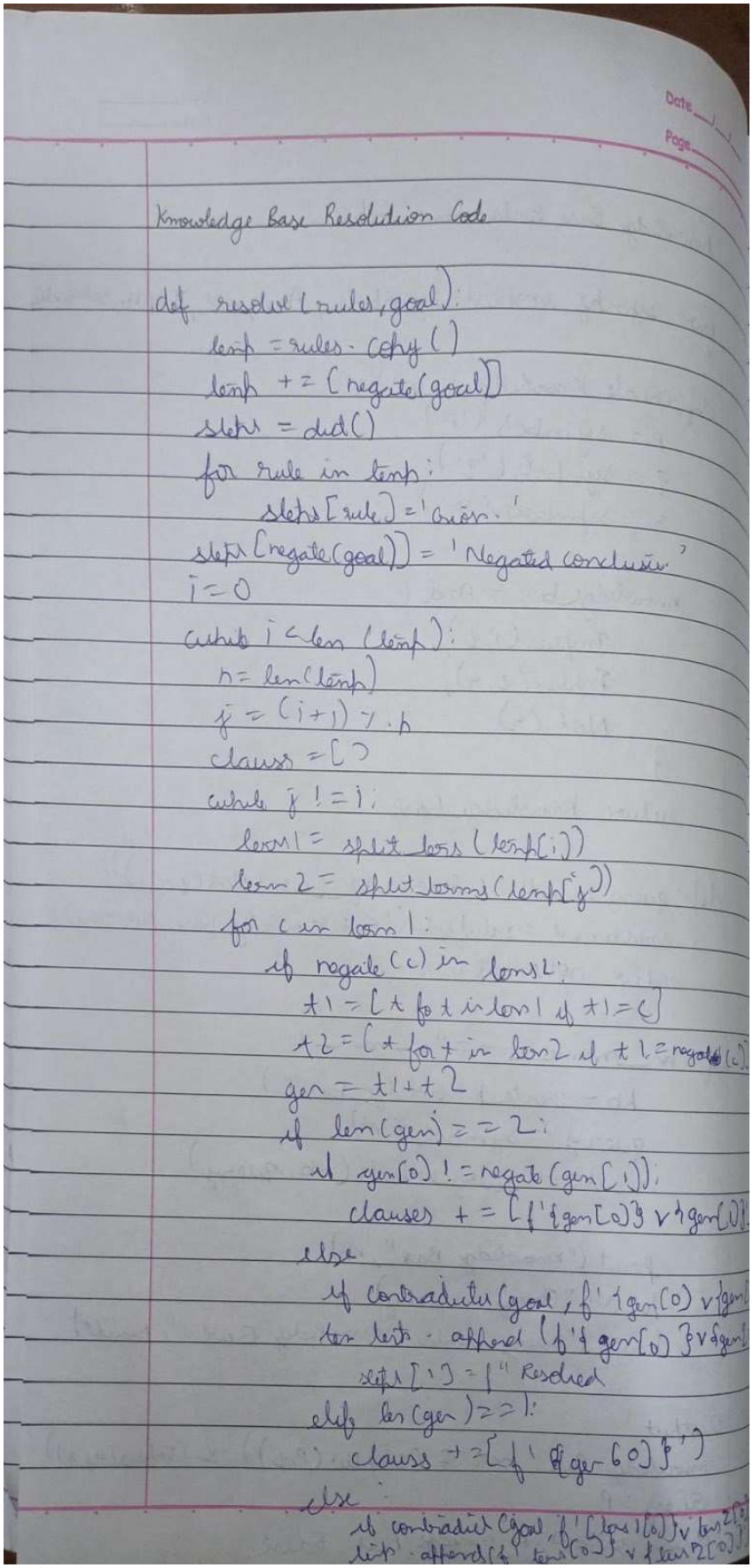
LAB: - 7

Aim

Create a knowledge base using prepositional logic and prove the given query using resolution

Observation Notebook





Knowledge Base Resolution Code

```
def resolve(rules, goal):
    temp = rules.copy()
    temp += [negate(goal)]
    steps = dict()
    for rule in temp:
        steps[rule] = 'Given.'
    steps[negate(goal)] = 'Negated conclusion.'
    i = 0
    while i < len(temp):
        n = len(temp)
        j = (i+1) % n
        clauses = []
        while j != i:
            terms1 = split_terms(temp[i])
            terms2 = split_terms(temp[j])
            for c in terms1:
                if negate(c) in terms2:
                    t1 = [t for t in terms1 if t != c]
                    t2 = [t for t in terms2 if t != negate(c)]
                    gen = t1 + t2
                    if len(gen) == 2:
                        if gen[0] != negate(gen[1]):
                            clauses += [f'{gen[0]} v {gen[1]}']
                        else:
                            if contradict(goal, f'{gen[0]} v {gen[1]}'):
                                temp.append(f'{gen[0]} v {gen[1]}')
                                steps[''] = "Resolved
                    elif len(gen) == 1:
                        clauses += [f'{gen[0]}']
                    else:
                        if contradict(goal, f'{terms1[0]} v {terms2[0]}'):
                            temp.append(f'{terms1[0]} v {terms2[0]}')
                            steps[''] = f"Resolved {terms1[0]} and {terms2[0]}
```

Date __/__/__
Page _____

```
else:
    if contradiction(goal, f'{term1[0]} v {term2[0]}'):
        return steps

for clause in clause:
    if clause not in test and clause != reverse(clause)
            and reverse(clause) not in test:
        test.append(clause)
        steps[clause] = f'Resolve from {list[i]} and {list[j]}'
    j = (j+1) % n
    i += 1
return steps

rules = 'R v ~P   R v ~Q   ~R v P   ~R v Q'
goal = 'R'
main(rules, goal)
```

output

Step	Clause	Derivation
1	R v ~P	Given
2	R v ~Q	Given
3	~R v P	Given
4	~R v Q	Given
5	~R	Negated Conclusion
6		Resolving R v ~P and ~R v P which is in turn null.

A contradiction is found when ~R is assumed as true, Hence, R is true.

Output

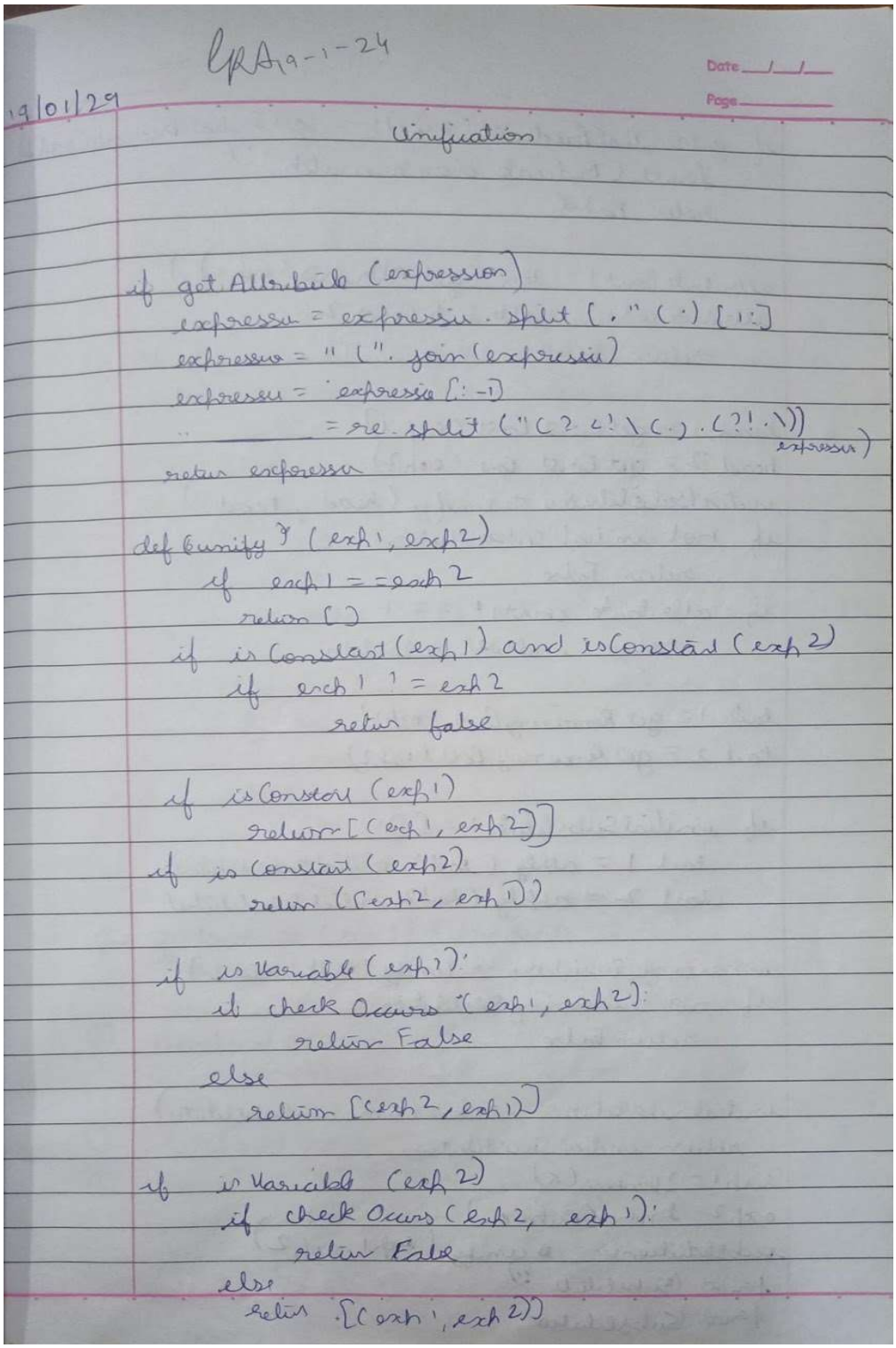
```
Kshitj S-1BM21CS093

Step    |Clause |Derivation
-----------------------------
 1.     | Rv~P  | Given.
 2.     | Rv~Q  | Given.
 3.     | ~RvP  | Given.
 4.     | ~RvQ  | Given.
 5.     | ~R    | Negated conclusion.
 6.     |       | Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

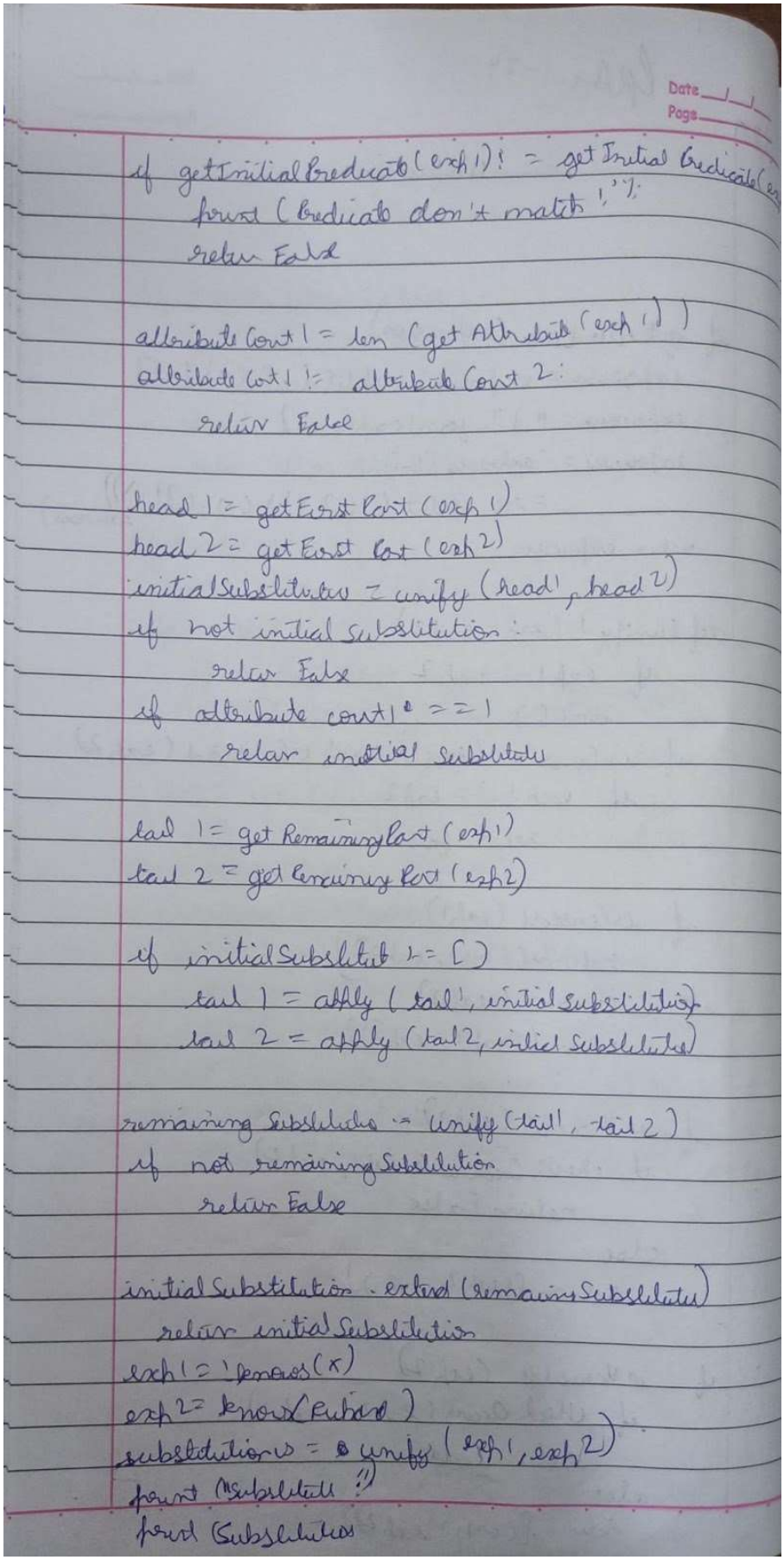
LAB: - 8

Aim

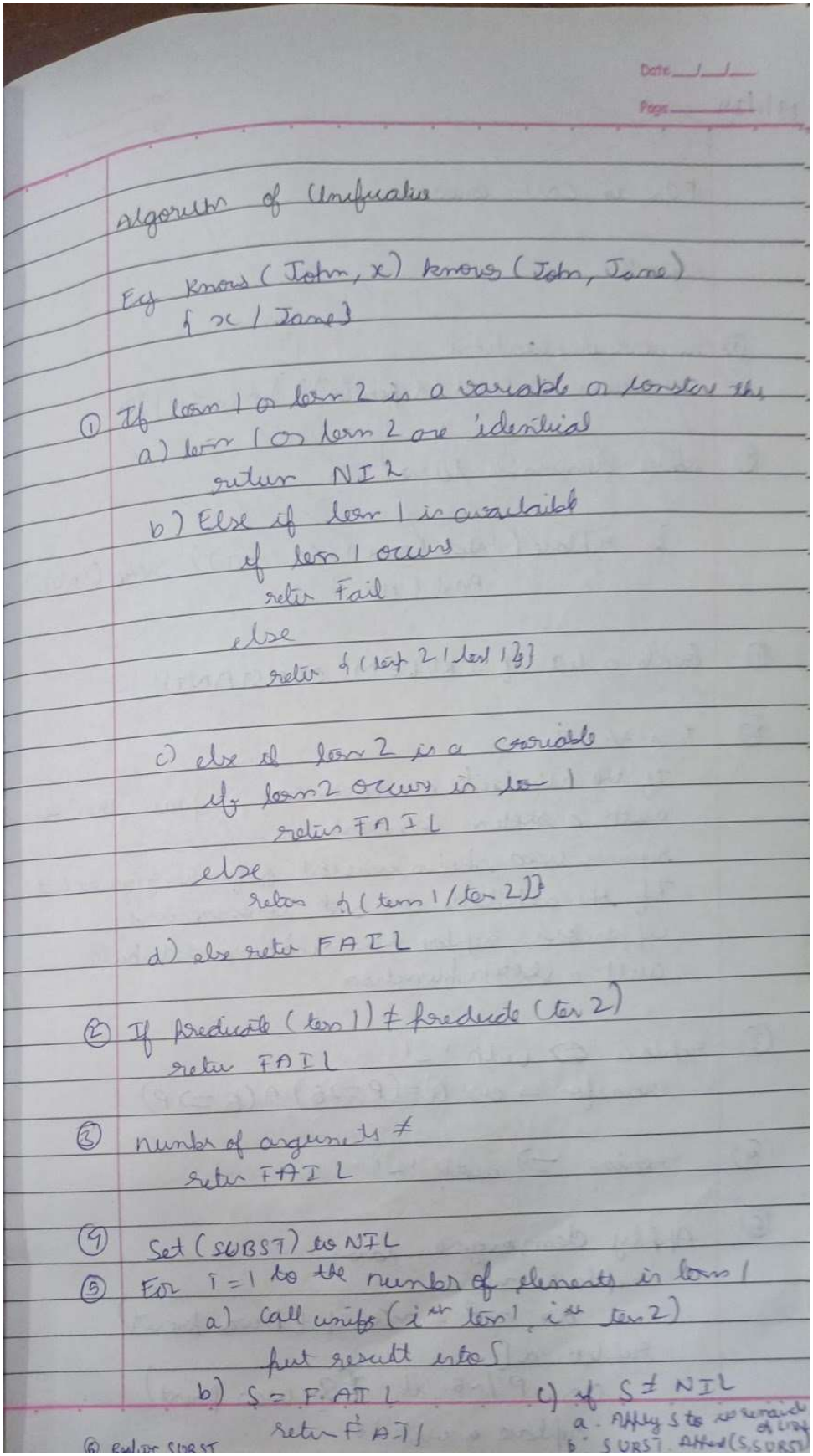
Implement unification in first order logic

Observation Notebook





```
if getInitialPredicate(exp1) != getInitialPredicate(exp
    print ("Predicate don't match");
    return False

attributeCount1 = len(getAttributes(exp1))
attributeCount1 != attributeCount2:
    return False

head1 = getFirstPart(exp1)
head2 = getFirstPart(exp2)
initialSubstitution = unify(head1, head2)
if not initialSubstitution:
    return False
if attributeCount1 == 1
    return initialSubstitution

tail1 = getRemainingPart(exp1)
tail2 = getRemainingPart(exp2)

if initialSubstitution != {}
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution)

remainingSubstitution = unify(tail1, tail2)
if not remainingSubstitution
    return False

initialSubstitution.extend(remainingSubstitution)
return initialSubstitution
exp1 = knows(x)
exp2 = knows(Richard)
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```


Algorithm of Unification

Eg knows (John, x) knows (John, Jane)
{x / Jane}

① If term 1 or term 2 is a variable or constant then
a) term 1 or term 2 are identical
return NIL
b) Else if term 1 is a variable
if term 1 occurs
return Fail
else
return {(term 2 / term 1)}

c) else if term 2 is a variable
if term 2 occurs in term 1
return FAIL
else
return {(term 1 / term 2)}
d) else return FAIL

② If predicate (term 1) ≠ predicate (term 2)
return FAIL

③ number of arguments ≠
return FAIL

④ Set (SUBST) to NIL
⑤ For i = 1 to the number of elements in term 1
a) Call unify (ith term 1, ith term 2)
put result into S
b) S = FAIL
return FAIL
c) if S ≠ NIL
a. Apply S to remainder of list
b. SUBST. Append(S, SUBST)

⑥ Return SUBST

Output

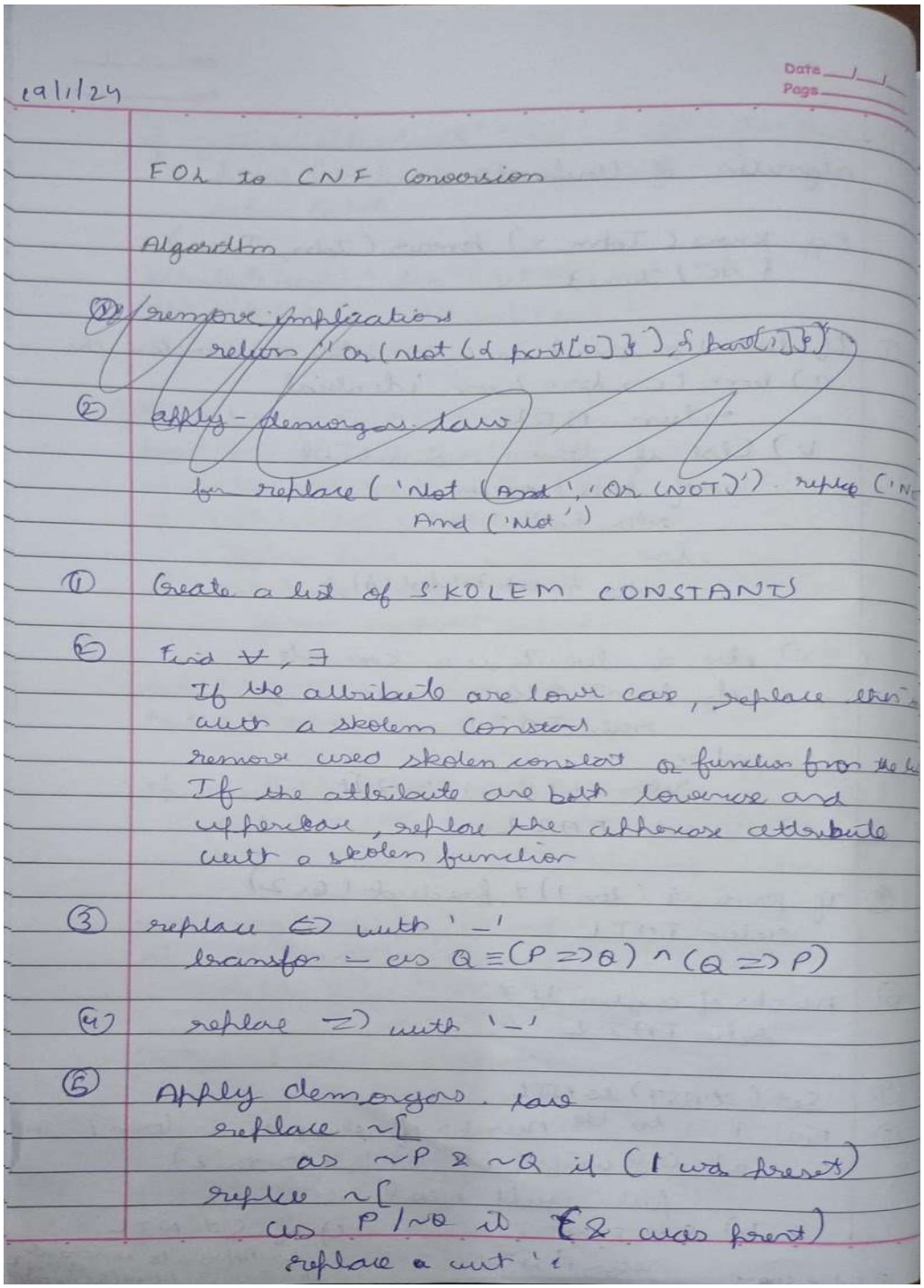
```
Kshitj S-1BM21CS093
Substitutions:
[('X', 'Richard')]
```

LAB: - 9

Aim

Convert a given first order logic statement into Conjunctive Normal Form (CNF).

Observation Notebook



Date __/__/__

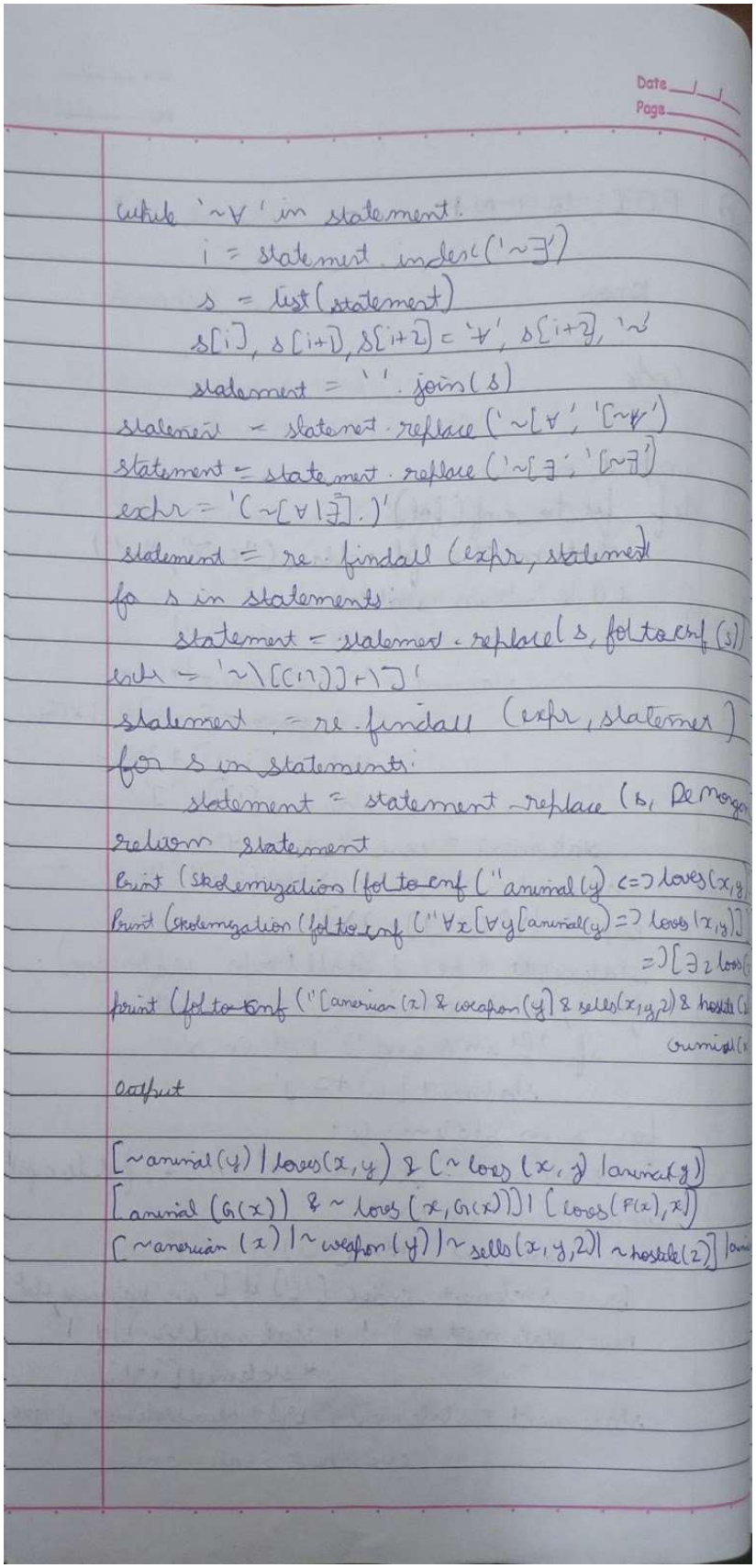
Page _____

④ FOL to CNF

know

Code

```
import re
def fol_to_cnf(fol):
    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = new statement .index('_')
        new_statement = '[' + statement[0:i] +
                        '=>' + statement[i+1:] + ']&[' +
                        statement[i+1:] + '=>' +
                        statement[:i] + ']'
    statement = new_statement
    statement = statement.replace("=>", "_")
    expr = '\[([^\]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '~' in statement:
        i = statement.index('~')
        br = statement.index('[') if '[' in statement else 0
        new_statement = '~' + statement[br:i] + '(' +
                        statement[i+1:] + ')'
        statement = statement[:br] + new_statement if br>0
                    else new_statement
```



while '~∀' in statement:
    i = statement.index('~∃')
    s = list(statement)
    s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
    statement = ''.join(s)
statement = statement.replace('~[∀', '[~∀')
statement = statement.replace('~[∃', '[~∃')
expr = '(~[∀|∃].)'
statement = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, fol_to_cnf(s))
expr = '~\[([^]]+)\]'
statement = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, DeMorgan
return statement
print(skolemization(fol_to_cnf("animal(y) <=> loves(x,y
print(skolemization(fol_to_cnf("∀x[∀y[animal(y) => loves(x,y)]
=>[∃z loves(
print(fol_to_cnf("[american(x) & weapon(y) & sells(x,y,z) & hostile(z
animal(x

output

[~animal(y) | loves(x,y)] & [~loves(x,y) | animal(y)]
[animal(G(x)) & ~loves(x,G(x))] | [loves(F(x),x)]
[~american(x) | ~weapon(y) | ~sells(x,y,z) | ~hostile(z)] | animal

Output

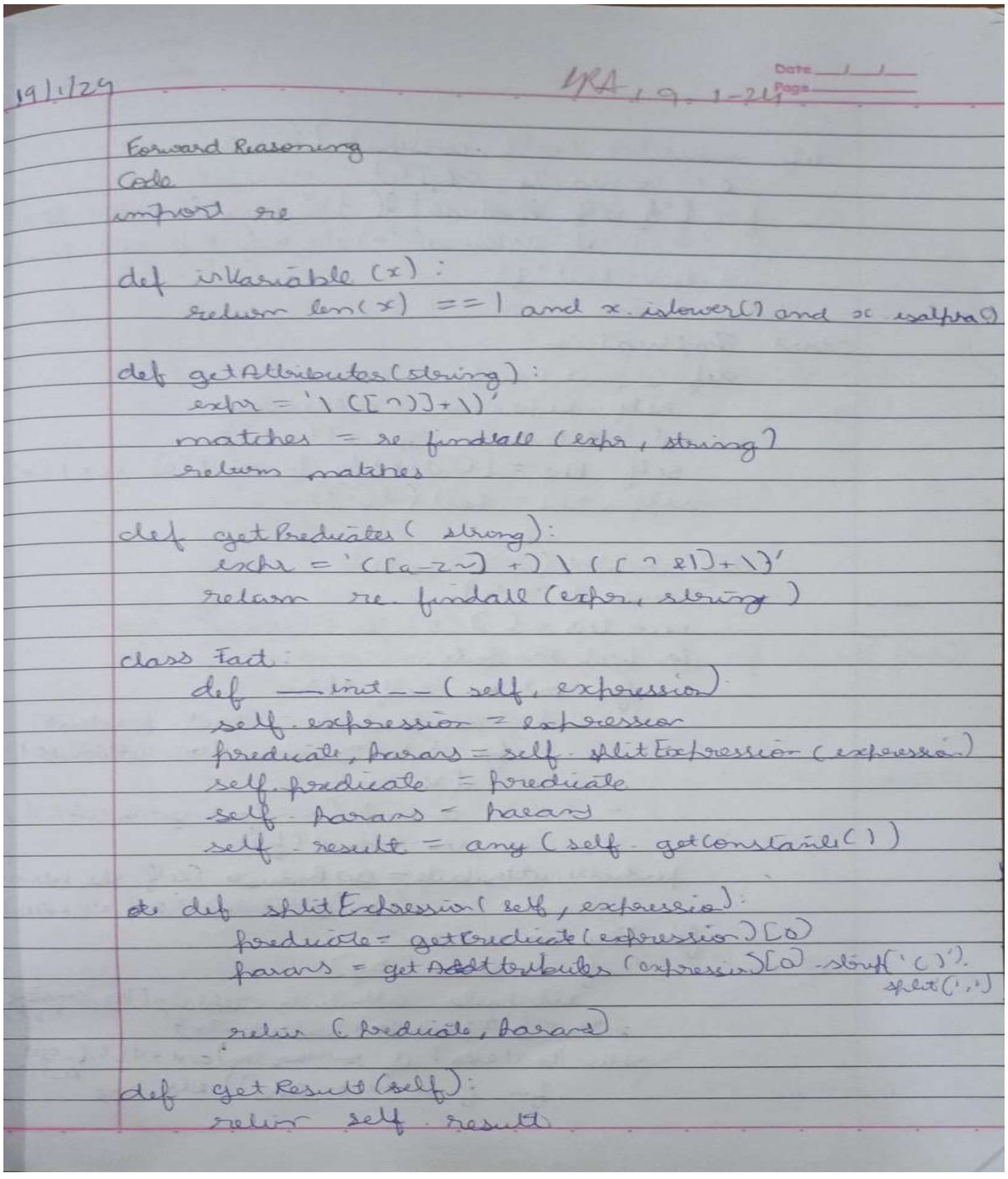
```
Kshitj S-1BM21CS093
[~animal(y)|loves(x,y)]&[~loves(x,y)|animal(y)]
[animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]
[~american(x)|~weapon(y)|~sells(x,y,z)|~hostile(z)]|criminal(x)
```

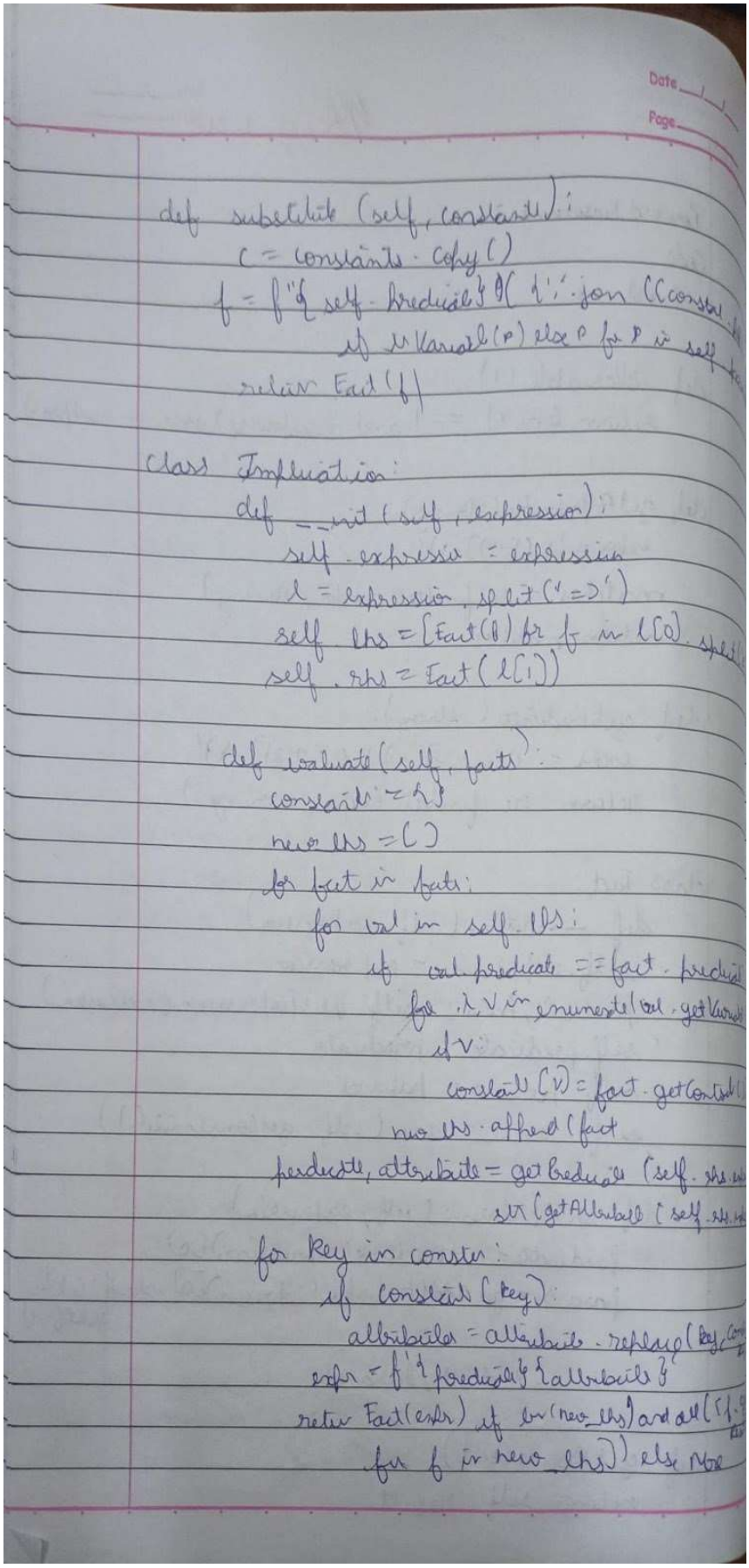
LAB: - 10

Aim

Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Observation Notebook





```
def substitute(self, constants):
    c = constants.copy()
    f = f"{self.predicate}({','.join(c.get(p, p) if is_variable(p) else p for p in self.
    return Fact(f)

class Implication:
    def __init__(self, expression):
        self.expression = expression
        l = expression.split('=>')
        self.lhs = [Fact(f) for f in l[0].split('
        self.rhs = Fact(l[1])

    def evaluate(self, facts):
        constants = {}
        new_lhs = []
        for fact in facts:
            for val in self.lhs:
                if val.predicate == fact.predicate
                    for i, v in enumerate(val.getVariables
                        if v:
                            constants[v] = fact.getConstants
                    new_lhs.append(fact
        predicate, attributes = getPredicate(self.rhs.
                                str(getAttributes(self.rhs.
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key, constants
        expr = f"{predicate}({attributes})"
        return Fact(expr) if len(new_lhs) and all([f.
                for f in new_lhs]) else None
```

```
class KB:
    def init(self):
        self.facts = set()
        self.implications = set()

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
        for i in self.implications:
            res = i.evaluate(self.facts)
            if res:
                self.facts.add(res)

    def query(self, e):
        facts = set([f.expression for f in self.facts])
        i = 1
        print(f'Querying {e}:')
        for f in facts:
            if Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')
                i += 1

    def display(self):
        print("All facts:")
        for i, f in enumerate(set([f.expression for f in self.facts])):
            print(f'\t{i+1}. {f}')
```

Date / /
Page

Output

```
kb_ = KB()
kb_ tell('king(x) & greedy(x) => evil(x)')
kb_ tell('king(John)')
kb_ tell('greedy(John)')
kb_ tell('king(Richard)')
kb_ query('evil(x)')
```

Query: evil(x):
1. evil(John)

Output

```
Kshitj S-1BM21CS093
Querying criminal(x):
        1. criminal(West)
All facts:
        1. criminal(West)
        2. hostile(Nono)
        3. weapon(M1)
        4. missile(M1)
        5. sells(West,M1,Nono)
        6. enemy(Nono,America)
        7. owns(Nono,M1)
        8. american(West)
```