VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ARTIFICIAL INTELLIGENCE

Submitted by

IMRAN WADRALI (1BM21CS077)

in partial fulfillment 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
Oct 2023-Feb 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

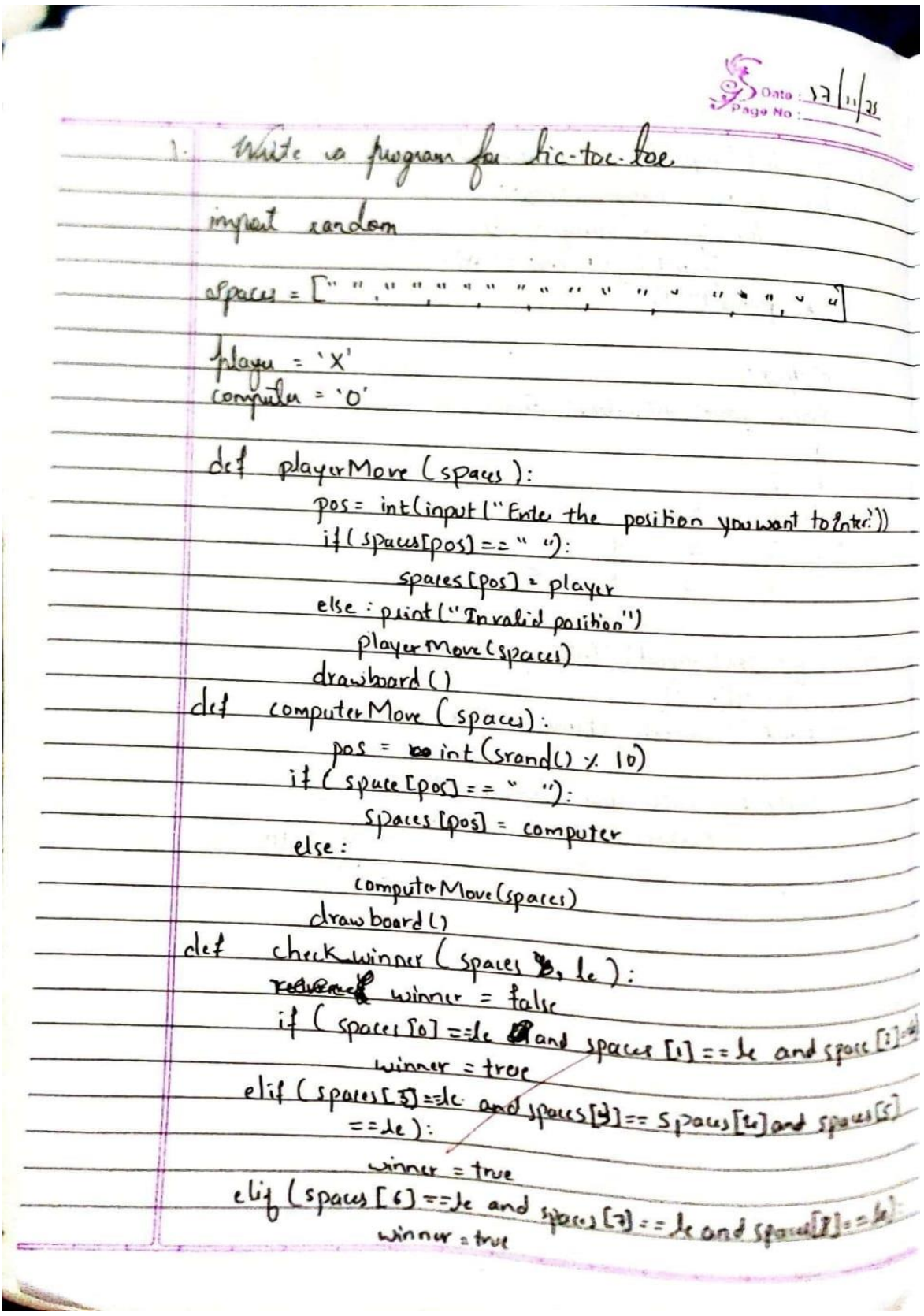
This is to certify that the Lab work entitled **"ARTIFICIAL INTELLIGENCE"** carried out by **IMRAN WADRALI (1BM21CS077)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of Artificial Intelligence Lab - **(22CS5PCAIN)**work prescribed for the said degree.

Swathi Sridharan
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table of Contents

1.Implement Tic –Tac –Toe Game.



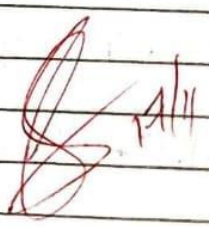
1. Write a program for tic-tac-toe

```
import random

spaces = [" ", " ", " ", " ", " ", " ", " ", " ", " ", " "]

player = 'X'
computer = 'O'

def playerMove (spaces):
    pos = int(input("Enter the position you want to enter:"))
    if (spaces[pos] == " "):
        spaces [pos] = player
    else : print ("Invalid position")
        playerMove (spaces)
    drawboard ()
def computerMove (spaces):
    pos = int (srand() % 10)
    if ( space [pos] == " "):
        spaces [pos] = computer
    else:
        computerMove (spaces)
    drawboard ()
def check_winner ( spaces, le ):
    winner = false
    if ( spaces [0] == le and spaces [1] == le and spaces [2] ==
        winner = true
    elif ( spaces [3] == le and spaces [4] == spaces [5] and spaces [5]
        == le ):
        winner = true
    elif (spaces [6] == le and spaces [7] == le and spaces [8] == le):
        winner = true
```

main()

Algorithm:

Step 1: make a spaces of 9 elements.
Step 2: Take input player move
Step 3: Check winner (if win stop the game)
Step 4: Take computer's random input
Step 5: Check winner. (if won stop the game)



```
elif(spaces[0] == le and spaces[3] == le and spaces[6] == le):
        winner = true
elif(spaces[1] == le and spaces[4] == le and spaces[7] == le):
        winner = true
elif(spaces[2] == le and spaces[5] == le and spaces[8] == le):
        winner = true
elif(spaces[0] == le and spaces[4] == le and spaces[8] == le):
        winner = true
elif(spaces[2] == le and spaces[4] == le and spaces[6] == le):
        winner = true
return winner

def drawboard():
    print(spaces[0] + "|" + spaces[1] + "|" + spaces[2])
    print("-----")
    print(spaces[3] + "|" + spaces[4] + "|" + spaces[5])
    print("-----")
    print(spaces[6] + "|" + spaces[7] + "|" + spaces[8])

def main():
    running = true
    drawboard()
    while(running):
        playermove(spaces)
        if(checkwinner(spaces, player)):
            running = false
            break()
        computerMove(spaces)
        if(checkwinner(spaces, computer)):
            running = false
            break()
```

2 .Solve 8 puzzle problems.

Date 24/11/23
Page No

Lab-3 : 8- puzzle

1) Define the state representation: Represent the initial & final states of the puzzle.
like
initial = [1, 2, 3, 4, 5, 6, 7, 0, 8]
Goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]

2) Initialize the Queue: Begin by making a Queue and store the states. Add the initial state of the puzzle to the queue.

3) Make rules for Moving tiles: The empty tile can move only in 4 directions.
up, down, left, right.
It should take one step at a time
Divide rules like

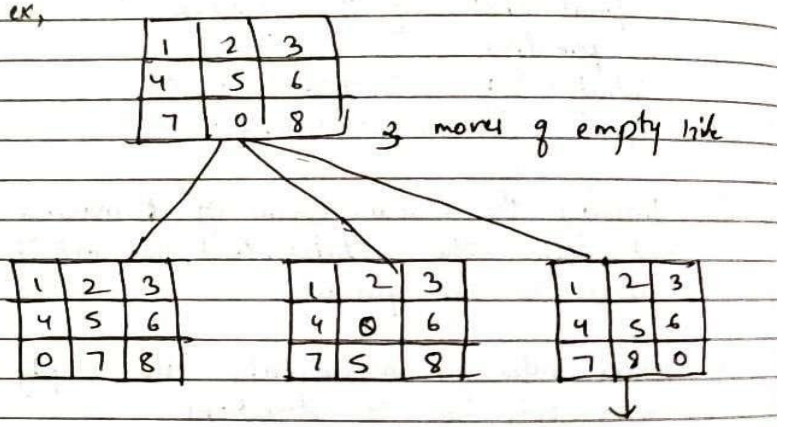
2	3	2
3	4	3
2	3	2

→ Like the corner elements can move only in 2 directions
→ The centre element can move in all 4 directions
→ The corner middle elements can move in 3 directions

4) Next, every time load the current state of board and create the possible moves until current state matches goal state and make a path for the same. Also ensure the states are not already visited to avoid cycles.

5] If BFS reach a solution give the path of solution else print no solution.

For ex,



Program:

```
from collections import deque

def bfs_8puzzle(initial_state, goal_state):
    queue = deque([[initial_state, []]])
    visited = set()
    while queue:
        current_state, path = queue.popleft()
    if (current_state == goal_state): return path
```

Output.

Solution:

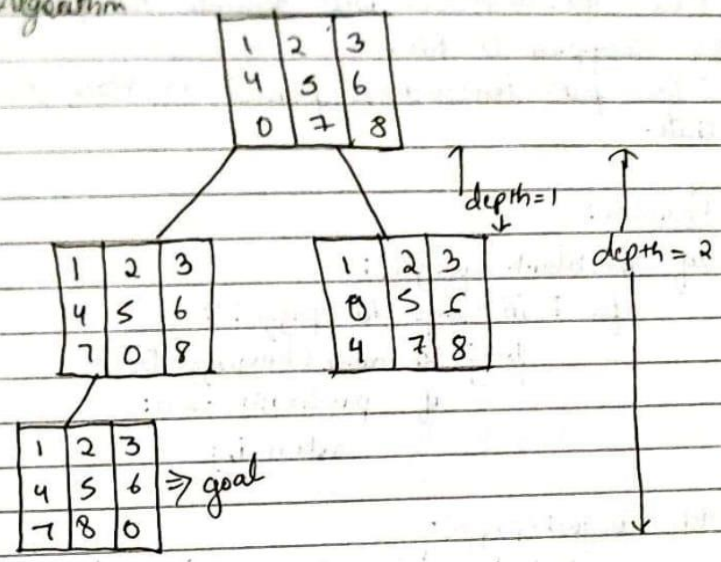
[(1, 2, 3, 4, 0, 6, 7, 5, 8)
(1, 2, 3, 4, 5, 6, 7, 0, 8)
(1, 2, 3, 4, 5, 6, 7, 8, 0)]

3. Implement Iterative deepening search algorithm.

8/12/23

8-puzzle iterative deepening search algorithm

Algorithm



Algorithm:

1] Initialize the initial state = [] and goal state for the 8puzzle

goal_state = [1,2,3,4,5,6,7,8,0] #0 is empty tile

2] Set the depth = 1 and expand the initial state. The depth_limited_search(depth) is performed

```
if node.state = goal
    return node
else
    for neighbour in getneighnode(state)
        child = Puzzlenode(neighbour, node)
        result = depth_limited_search(depth-1)
    if result == True:
        return result;
```

3) After one iteration when depth = 1 increment the depth by 1 and perform limited search again.

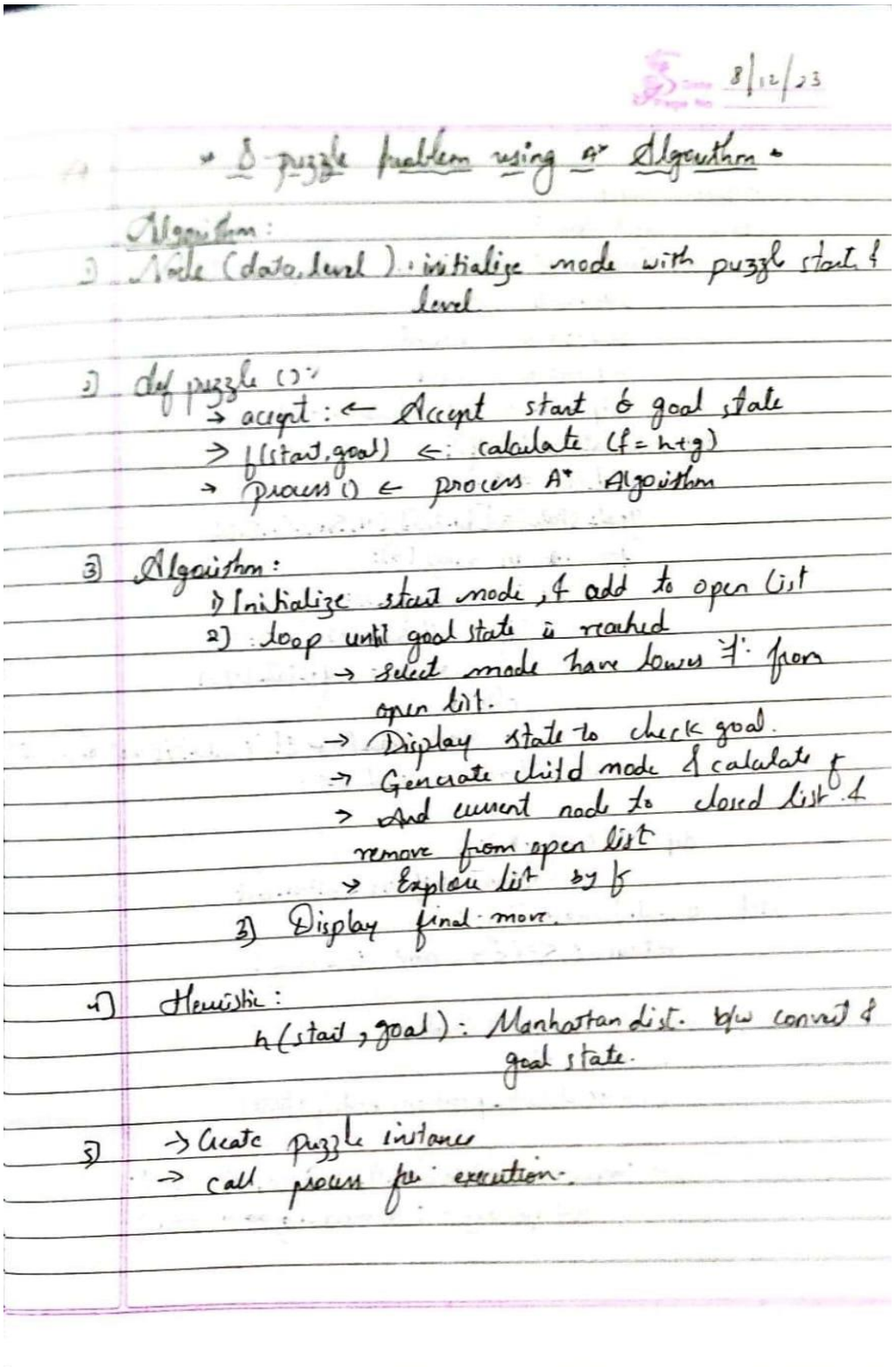
4) Here get_neighbour will generate the possible moves by swapping '0' tile.

5) The path traversed is printed to reach the goal state.

Program:

```
def find_blank (puzzle):
    for i in range (len (puzzle)):
        for j in range (len(puzzle[i])):
            if puzzle[i][j] == 0:
                return i, j

def is_goal (puzzle):
    goal_state = [[1,2,3],[4,5,6],[7,8,0]]
    return puzzle == goal_state

def move_tile (puzzle, direction):
    blank_row, blank_col = find_blank (puzzle)
    new_puzzle = [row[:] for row in puzzle]

    if direction == 'up' and blank_row > 0:
        new_puzzle [blank_row][blank_col], new_puzzle
        [blank_row-1][blank_col] = new_puzzle[row...
            [blank_col], new_puzzle [blank_row...
    elif direction == 'down' and blank_row < 2:
        new_puzzle [blank_row][blank_col], new_puzzle
        [blank_row-1][blank_col], new_puzzle
```

4. Implement A* search algorithm.



8/12/23

* 8 puzzle problem using A* Algorithm *

Algorithm:

1) Node (data, level): initialize node with puzzle state & level

2) def puzzle ():
   → accept: ← Accept start & goal state
   → f(start, goal) ← calculate (f = h+g)
   → process () ← process A* Algorithm

3) Algorithm:
   1) Initialize start node, & add to open list
   2) loop until goal state is reached
      → Select node have lower f: from open list.
      → Display state to check goal.
      → Generate child node & calculate f
      → And current node to closed list & remove from open list
      → Explore list by f
   3) Display final move.

4) Heuristic:
   h(start, goal): Manhattan dist. b/w current & goal state.

5) → Create puzzle instance
   → call process for execution.

5. Implement vaccum cleaner agent.

Date: 22/12/23
Page No:

* Vacuum Cleaner Problem *

Algorithm:

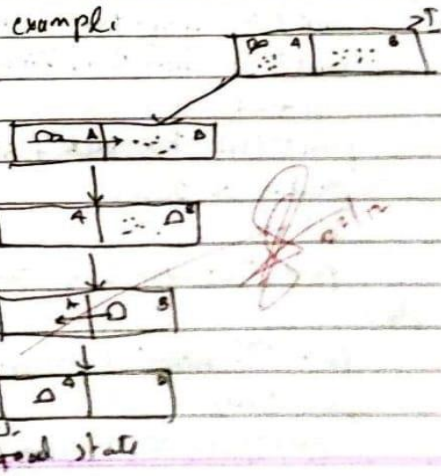
1] Initalize the starting and goal state, the goal state ~~begin~~ being to have both rooms ~~clean~~ clean.

2] If status = Dirty then clean
else if location = A & status = clean ~~return~~ return right
else if location = B & status = clean return left
else exit.

3] Based on above rules, take user input on cleanliness status and show the actions of vacuum cleaner.

4] Once both rooms are clean ~~exit~~ the vacuum cleaner has accomplished the task and reached goal state and then exits.

For example:

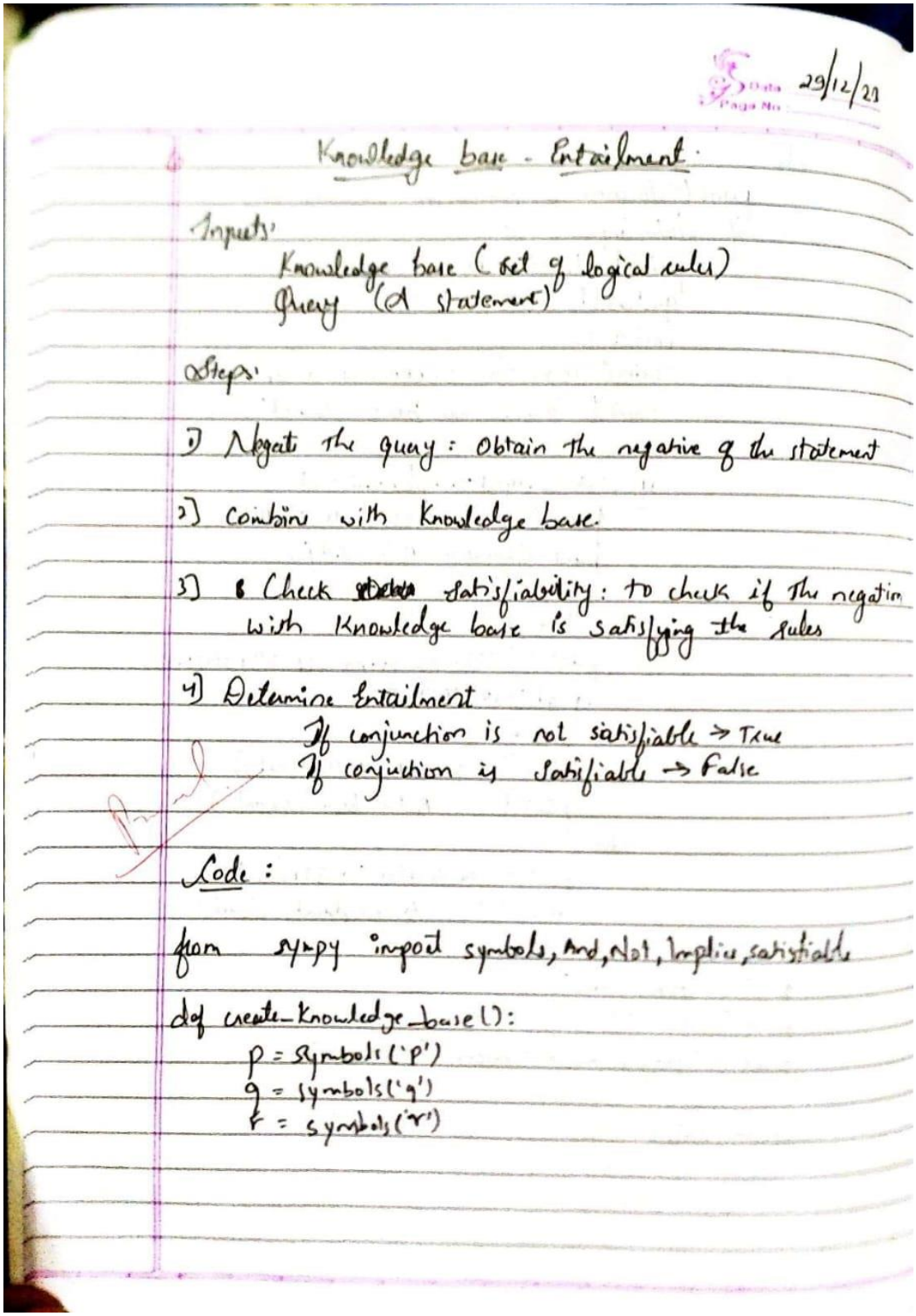


Code:

```
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 the status of the other room"))
    print("Initial Location 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("Loc A has been cleared")

            if status_input_complement == 1:
                print("Moving right to location B")
                print("Location B is Dirty")
                cost += 1
                print("Cost for suck" + str(cost))
                goal_state['B'] = 0
                cost += 1
                print("Location B is cleaned")
            else:
                print("No action" + str(cost))
                print(cost)
                print("Loc B is already clean")
```

```
else:
    print("Vaccum is placed in location B")
    if status_input == 1
        print("Location B is Dirty.")
        good_state['B'] = 0
        cost += 1
        print("COST for CLEANING" + str(cost))
        print("loc B has been cleaned")

        if status_input_complement = = 1
            print("Moving LEFT to locA")
            print("Location A is dirty")

            cost += 1
            print("COST of moving LEFT" + str(cost))
            good_state['A'] = 0
            cost += 1
            print("Cost of suck" + str(cost))
            print("Loc A has been cleaned")
        else:
            print("No action" + str(cost))
            print("Loc A is already clean")

print("GOAL STATE")
print(good_state)
print("performance measurement" + str(cost))

vaccum world()
```

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



Date 29/12/21
Page No.

Knowledge base - Entailment

Inputs:
Knowledge base (Set of logical rules)
Query (A statement)

Steps:

1] Negate the query: Obtain the negative of the statement

2] Combine with Knowledge base.

3] Check satisfiability: to check if the negation with Knowledge base is satisfying the rules

4] Determine Entailment
If conjunction is not satisfiable → True
If conjuction is satisfiable → False

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, query):
    entailment = Satisfiable(And(Knowledge_base, Not(query)))

    return not entailment

if __name__ == "__main__":
    kb = create_knowledgebase()

    query = symbols('not p')

    result = query_entails(kb, query)

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

Output

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

Query : (not, p)

Query entails knowledge base : False

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

Knowledge Base. Propositional Logic

```
7. Code:
def negate_literal(literal)
    if literal[0] == '~':
        return literal[1:]
    else:
        return '~' + literal

def resolve(C1, C2):
    resolved_clause = set(C1) | set(C2)
    for literal in C1:
        if negate_literal(literal) in C2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal(literal))
    return tuple(resolved_clause)

def resolution(knowledge_base):
    while True:
        new_clauses = set()
        for i, C1 in enumerate(knowledge_base)
            for j, C2 in enumerate(knowledge_base)
                if i != j:
                    newclause = resolve(C1, C2)
                if len(new_clause) > 0 & new_clause
                        not in knowledge_base:
                    new_clause.add(new_clause)

        if not new_clause
            break.
```

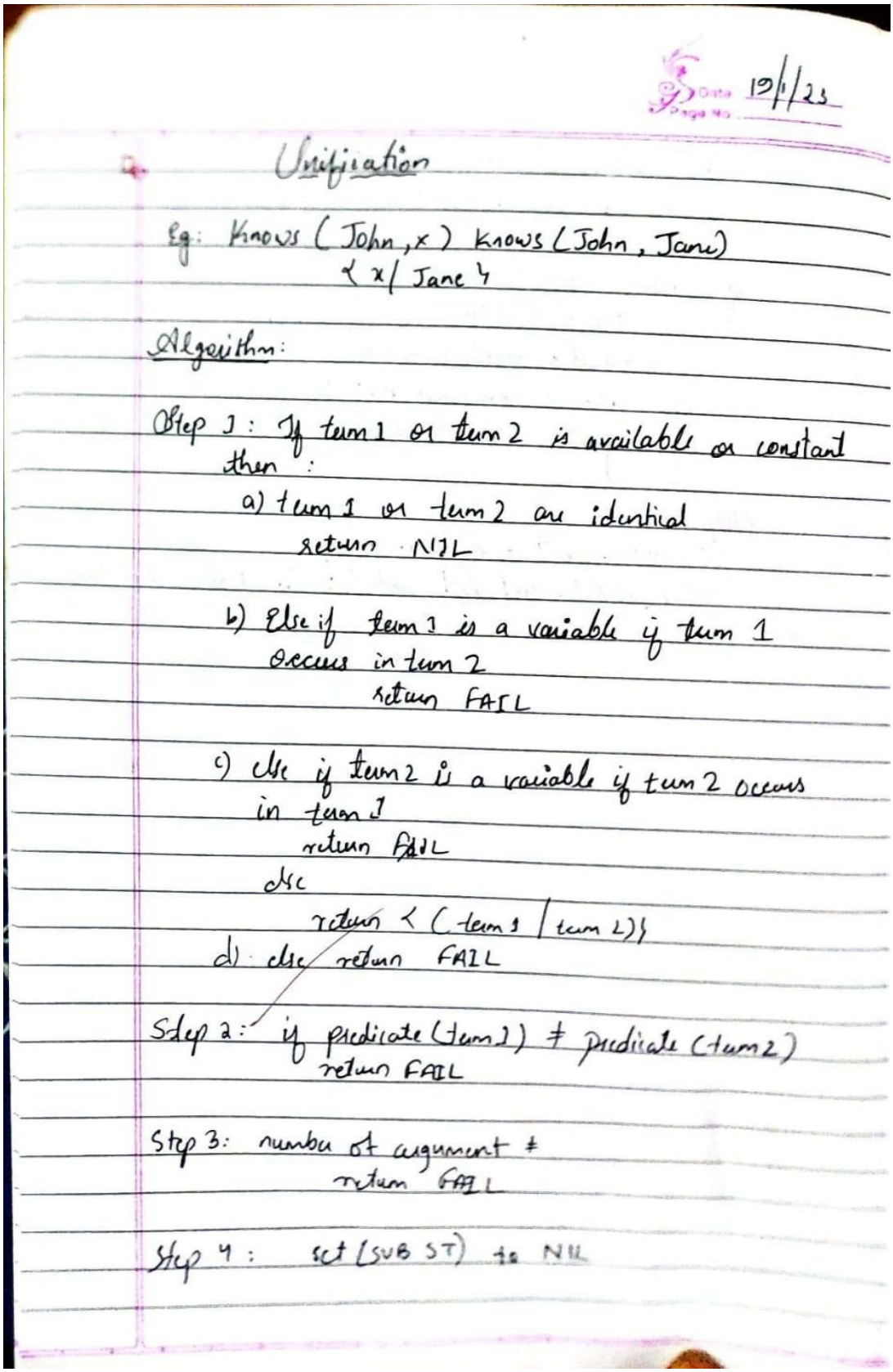
```
        Knowledge_base != new_clause
        Return Knowledge_base

if __name__ == '__main__':
        Kb = [('p', b), ('~p', 'r'), ('~q', '~r')]
        result = resolution(Kb)
        print('original Kb', Kb)
        print('Resolved Kb', result)
```

Output:

Enter Statement = negation

The Statement not entitled in Knowledge base

8. Implement unification in first order logic :



Unification

Eg: Knows (John, x) Knows (John, Jane)
{x/ Jane}

Algorithm:

Step 1: If term 1 or term 2 is 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 in term 2
return FAIL

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

Step 2: if predicate (term 1) ≠ predicate (term 2)
return FAIL

Step 3: number of argument ≠
return FAIL

Step 4: set (SUB ST) to NIL

Step 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
S = FAIL
return FAIL.

b) if S ≠ NIL
a) Apply S to the remainder of both L_1 & L_2
b) SUBST. APPEND (S, SUBST)

step 6: Return SUBST

Code:

```
import re

def getAttributes(expression):
    expression = expression.split("(")[1:]
    expression = "(".join(expression)
    expression = expression[:-1]
    expression = re.split("(?<!\(),(?!\))", expression)
    return expression

def getInitialPredicate(expression):
    return expression.split("(")[0]

def isVariable(char):
    return char.islower() and len(char)==1
```

```
def apply(exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes(exp, old, new)
    return exp

def checkOccurs(var, exp):
    if exp.find(var) == -1
        return false
    return true

def getFirstPart(expression):
    attributes = getAttributes(expression)
    return Attributes[0]

def unify(exp1, exp2):
    if exp1 == exp2
        return []

    if isConstant(exp1):
        return [(exp1, exp2)]

    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return false
        else:
            return [(exp2, exp1)]

    if initialSubstitution != []:
        tail1 = apply(tail1, initial)
        tail2 = apply(tail1, initial)
```

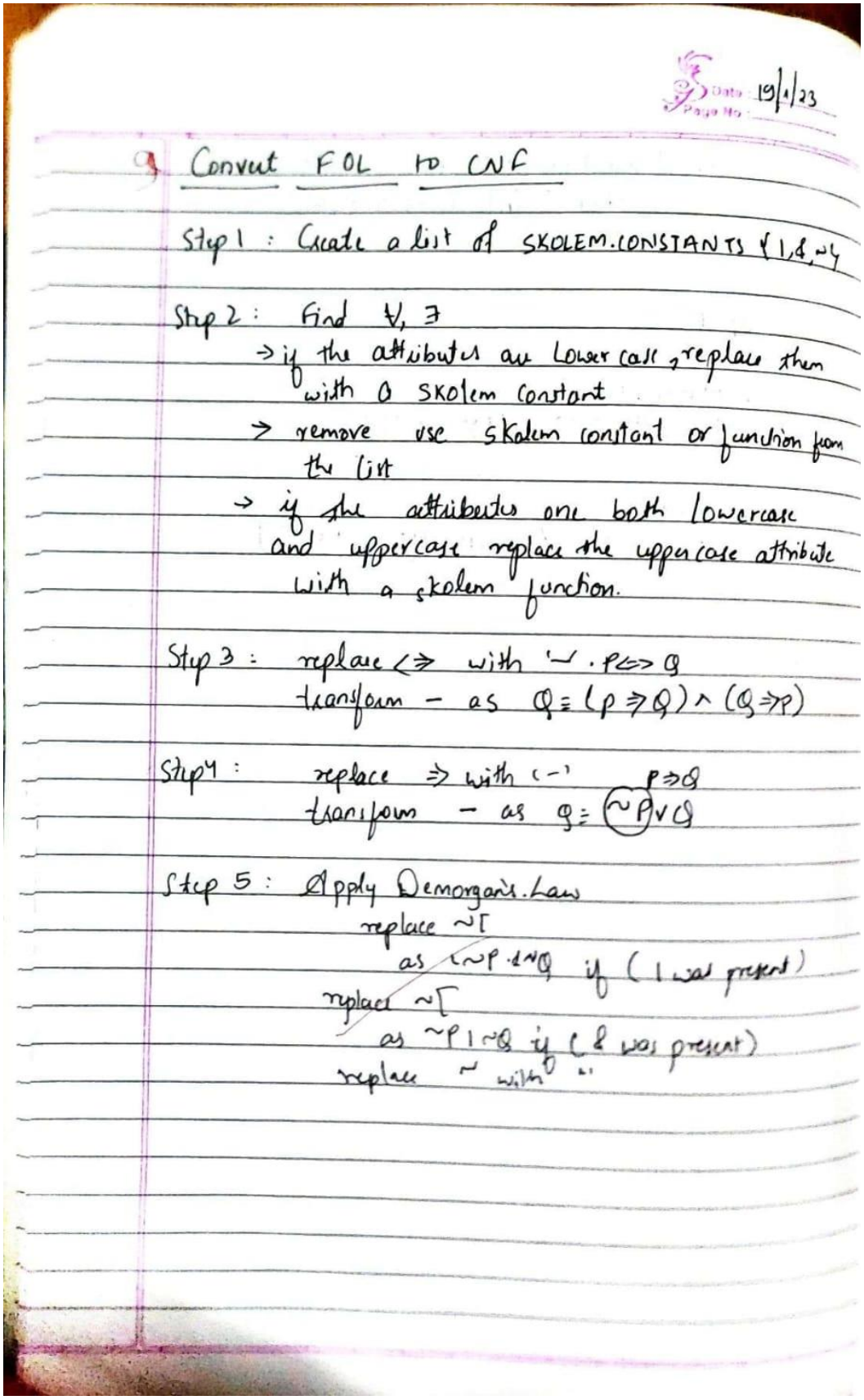
initial Substitution extend (renaming sub)
return initialSubstitution.

Output:

Substitutions:
[('A','Y'), ('Y','x')]

Substitutions:
[('A','Y'), ('Mother(Y)','x')]

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



Date 19/1/23
Page No.

Convert FOL to CNF

Step 1 : Create a list of SKOLEM.CONSTANTS {1,2,...}

Step 2 : Find ∀, ∃
→ if the attributes are Lower case, replace them with a skolem constant
→ remove use skolem constant or function from the list
→ if the attributes are both lowercase and uppercase replace the uppercase attribute with a skolem function.

Step 3 : replace <⇒ with '~' . P<=>Q
transform - as Q≡(P⇒Q)∧(Q⇒P)

Step 4 : replace ⇒ with (~) P⇒Q
transform - as Q≡(~P∨Q)

Step 5 : Apply Demorgan's Law
replace ~[
as (~P.~Q if (was present)
replace ~[
as ~P|~Q if (was present)
replace ~ with '.'

Code:

```
def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findAll(expr, string)
    return [m for m in str(matches) if m.isalpha()]

def getpredicate(string):
    expr = '[a-zA-Z]+\([A-Za-z,]+\)'
    return re.findall(expr, string)

def deMorgan(sentence):
    string = ''.join(list(sentence).copy())
    string = string.replace('~~', '')
    flag = '~' in string

    for predicate in getPredicate(string):
        string = string.replace(predicate, '~'+predicate)

    S = list(string)

    for i,c in enumerate(string):
        if c == '|':
            S[i] = '&'
        elif c == '&':
            S[i] = '|'

    return ''.join(S)[1:-1] if flag else '~('+''.join(S)+')'
```

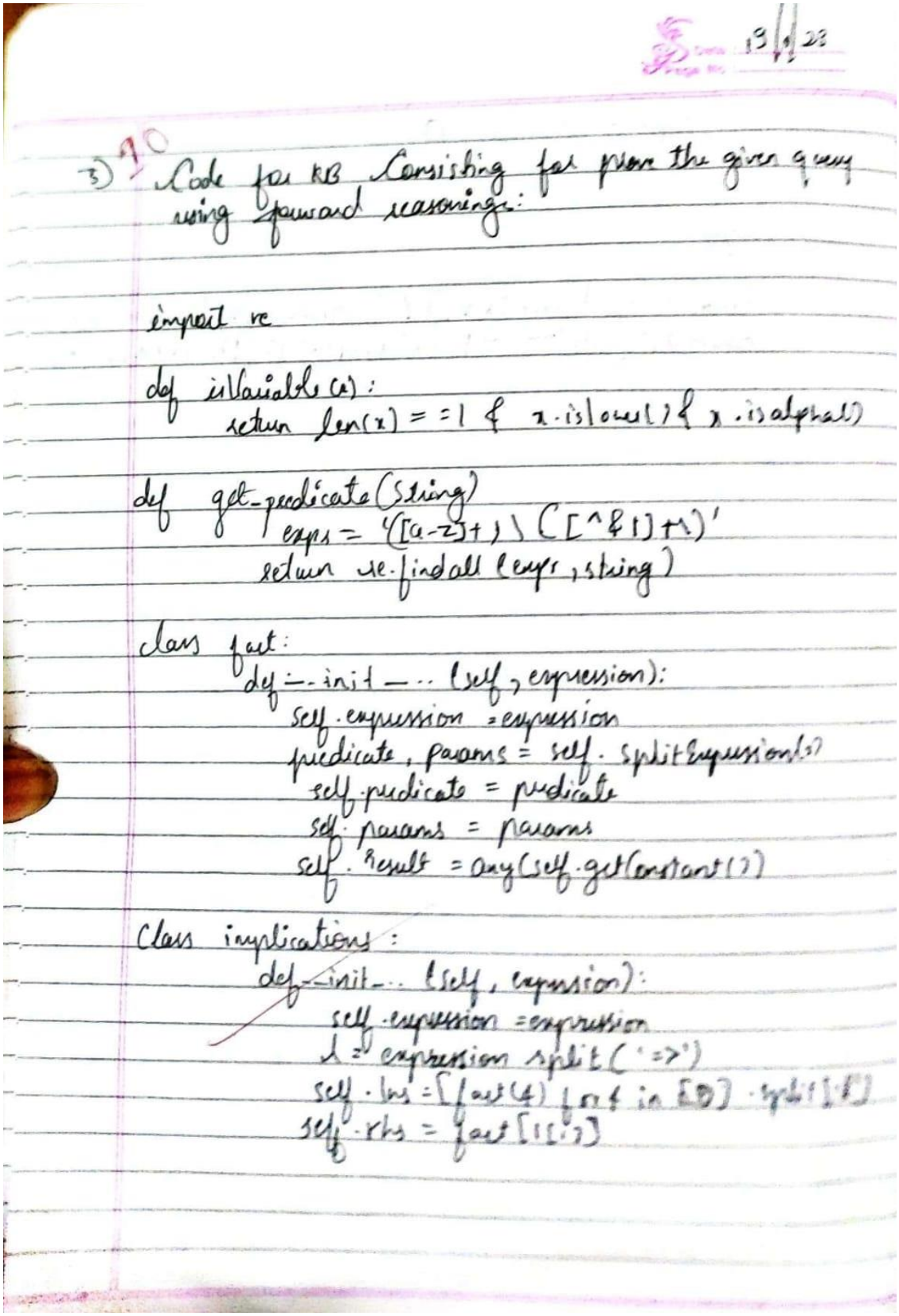
```
def skolemization(sentence):
    statement = ''.join(list(sentence).copy())
    matches = re.findall('[A-Z]', statement)

    for match in matches[::-1]:
        statement = statement.replace(match, '')

    for s in statements:
        statement = statement.replace(s, s[1:-1])


import re

def fol_to_cnf(fol):

    statement = fol.replace("<=>", "-")

    while '-' in statement:
        i = statement.index('-')
        statement = new_statement

    statement = statement.replace('=>', '-')

    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[') + '[' in statement

    while '~A' in statement:
        i = statement.index('~A')
        statement = list(statement)
        statement[i:] = statement[i+2] + "A"
```

expr = (∀ = [A | ∃])

Output:

[¬animals(g) | love(x,y)] ∉ [¬love[x,y] | animals(y)]
[animals[F(z)]] — ¬loves(x, G(x)), [loves(A(x), x]

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



3) Code for KB consisting for prove the given query using forward reasoning:

```
import re

def isVariable(x):
    return len(x) == 1 & x.islower() & x.isalpha()

def getpredicate(string)
    expr = '([a-z]+)\([^&|]+\)'
    return re.findall(expr, string)

class fact:
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression()
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstant())

Class implications:
    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, fact):
    constants = {}
    new_lhs = []

    for key in constants:
        if constants[key]:
            attributes = attributes.pop(key)

    return fact(args) if len(new_line) & all
    ([t.getResult() for t in new])
    else none.

class KB:

    def __init__(self):
        self.fact = set()
        self.fact = set()

    def tell(self, e):
        if '=>' in e:
            self.implication.add(implications[e])
        else:
            self.facts.add(facts(e))

        for i in self.implications:
            res = i.evaluate(self.facts)
        if res:
            self.facts.add(rs)
```

```
def display(self):
    print("All facts:")
    for i, f in enumerate(set([f.expression
                for f in self.facts])):
        print(f'{i+1}: {f}')

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)')
```

Output:

All facts:

Query: evil(x)

1. evil(John)