INDEX

Name Darshan. V. Deshbhandari Std _____ Sec _____

Roll No. _____ Subject _____ School/College _____

School/College Tel. No. _____ Parents Tel. No. _____

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
01		Basic programs		
02		8-puzzle using BFS.		
03		8-puzzle using IDDFS.		
04		8-puzzle using A* Alog.		
05		Vacuum Cleaner.		
06		KB entailment.		
07		KB Resolution		
08		Unification using FOL		
09		Convert FOL into CNF		
10		Forward Reasoning.		

3/2/24

3/11/23.

Lab - 1

DATE: __/__/____
PAGE: _____

* Built in Datatypes :-

Text type : str.
Numeric Types : int, float, complex
Sequence Types : list, tuple, range
Mapping Types : dict.
Set Types : set, frozenset
Boolean Type : bool
Binary Types : bytes, bytearray, memoryview
None Type : NoneType

Functions :

```
def function ():
    print ("abcd")

function ()
```

Arguments :

```
def function (fname) :
    print ( fname + "abcd")

function ("Email")
function ("Tobias")
function ("Linus")
```

* Arbitrary Arguments *args

```
def function (*sports)
    print ("Fav sports is" + sports[1])

function ("cricket", "volleyball", "abcd")
```

DATE: / /
PAGE:

Key-word Arguments :

```
def function ( child 3, child 2, child 1)
    print ( " The youngest child is " + child 2)

function (child 1 = "Naveen", child 2 = "Vishwa"
          child 3 = "venbi")
```

Pass Statement :

```
def functions ( ) :
    pass                    // Empty purpose.
```

Lambda :

A lambda function can take any no.of arguments, but can only have one expression.

Syntax :

lambda arguments : expressions

Ex :

Add 10 to arguments a, and return the result :

```
x = lambda a : a+10
print (x(5)).
```

Lab -2

DATE: 10, 11, 23.
PAGE:

1) Using if-else statement :

```
age = int (input (" Enter your age : "))
if age < 0 or age > 150 :
    print (" Enter a valid age ! ")
else :
    if age < 13 :
        print (" You are a child ! ")
    elif age < 20 :
        print (" Your are a teenager ! ")
    elif age < 35 :
        print (" You are an adult ! ")
    elif age < 60 :
        print (" You are a middle aged person
    else :
        print (" You are a senior citizen ! ")
```

Output :

Enter your age : 30.
You are an adult.

DATE: __/__/__
PAGE:

2) Find power of n.

```
x, n = input("Enter the base and power:").split()
x = int(x)
n = int(n)
print(x, "to the power of", n, "is", pow(x,n))
```

Output: 2 3

3 to the power of 2 is 8.

DATE: __/__/__
PAGE:

3) Reversing

```
num = int(input("Enter any integer:"))
sum = 0
while num>0:
    sum = 10*sum + num%10
    num //= 10
print("The number in reverse order is:",sum
```

Output: 2521

The number in reverse order is = 1252.

8

4) Pattern (1, 22, 333 - - -)

```
n = int(input("Enter length of pattern :"))
for i in range(n):
    for j in range(i+1):
        print(i+1, end = "")
print(end = "\n")
```

Output : 3

1, 22, 333.

Lab-3.

① Implement Tic-Tac-Toe Game

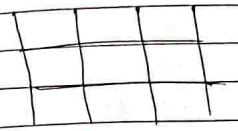
```
board = [' '
 for x in
 range(10)]

def insertLetter(letter, pos):
    board[pos] = letter.

def spaceIsFree(pos):
    return board[pos] == ' '

def printBoard(board):
    print('  |  |  ')
    print(' ' + board[1] + '|' + board[2] + '|'
          + board[3])
    print('  |  |  ')
    print('-----')
    print('  |  |  ')
    print(' ' + board[4] + '|' + board[5] + '|'
          + board[6])
    print('  |  |  ')
    print('-----')
    print(' ' + board[7] + '|' + board[8] +
          '|' + board[9])
    print('  |  |  ')

def isWinner(bo, le):
    return (bo[7] == le and bo[8] == le and
    bo[9] == le) or (bo[4] == le and bo[
    == le and bo[6] == le) or (
    bo[1] == le and bo[2] == le and bo[3]
    le) or (bo[1] == le and bo[4] == le and
    bo[7] == le) or (
```

```
    bo[2] == le and bo[5] == le and bo[8]
    == le or (
    bo[3] == le and bo[6] == le and bo[9] == le
    or ( bo[1] == le and bo[5] == le and
    bo[9] == le ) or bo[3] == le and bo[5] ==
    le and bo[7] == le)

def playerMode():
    run = True
    while run:
        move = input('Please select a position to
        place an \'X\' (1-9): ')
        try:
        move = int(move)
        if move > 0 and move < 10:
            if spaceIsFree(move):
                run = False
                insertLetter('X', move)
            else:
                print('Please type number within the
                    range!')
        except:
            print('Please 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[:]
            boardCopy[i] = let
            if isWinner(boardCopy, let):
                move = i
                return move

    cornersOpen = []
    for i in possibleMoves:
        boardCopy = board[:]
        boardCopy[i] = let

    if len(cornerOpen) > 0:
        move = selectRandom(cornersOpen)
        return move.

    if 5 in possibleMoves:
        move = 5:
        return move

    edgesOpen = []
    for i in possible Moves:
        if i in [2, 4, 6, 8]:
            edgesOpen.append[i]

    if len(edgesOpen) > 0:
        move = selectRandom(edgesOpen)
    return move.

def selectRandom(li):
    import random
    ln = len(i)
    r = random.randrange(0, ln)
    return li[r]

def isBoardFull(board):
    if board.count(' ') > 1:
        return False
```

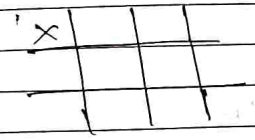
```
    else:
        return True.

def main():
    print ('Welcome to Tic Tac Toe!')
    print (board)

while not (isBoardFull(board)):
    if not (isWinner(board, 'O')):
        playerMove()
        printboard(board)
    else:
        print ('sorry, O's won this time!')
        break:

if isBoardFull(board):
    print ('Tie Game!')

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

Output:

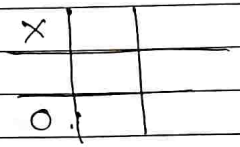
Doyou want to play again? (Y/N) Y



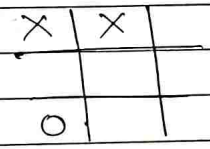
please select a position to place an 'X' (1-9): 1



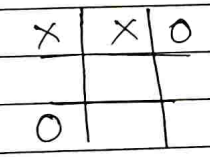
Computer placed an 'O' in position 7:



please Select a position to place an 'X' (1-9): 2



Computer placed an 'O' in position 3:



please select a position to place on 'X' (1-9):

X	X	O
X	●	
O		

Computer placed on 'O' in position 5:

X	X	O
X	O	
O		

Sorry, O's won this time

Lab-1

24/11/23

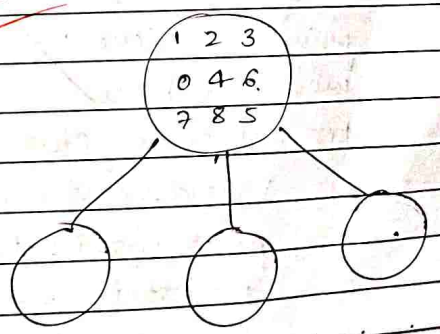
8- Puzzle Game using DFS. Algorithm:

problem: Given a 3×3 board with 8 tiles, where every tile has a number from 1 to 8 and one empty space called as 0. The objective is to slide the tiles adjacent to empty space to match the final configuration.

Algorithm:

1. Start
2. Start from given configuration by generating all child nodes of it
3. Now select the child node by using least cost function where least cost = no. of moves made + number of tiles in non fin configuration
4. Now keep on repeating step 2 and 3 until a final state is reached where we cannot generate any child further
5. Now check if this configuration matches the required one.
6. Return the answer.
7. Stop.

checked



DATE: / /
PAGE:

Program:

```
def print_grid (src):
    State = src.copy()
    State [state.index(-1)] = '_'
    print(f"""
    {State[0]} {State[1]} {State[2]}
    {State[3]} {State[4]} {State[5]}
    {State[6]} {State[7]} {State[8]}
    """)

def h(state, target):

    dist = 0;
    for i in state:
    d1, d2 = state.index(i), target.index(i)
    x1, y1 = d1 % 3, d1 // 3
    x2, y2 = d2 % 3, d2 // 3
    dist += abs(x1 - x2) + abs(y1 - y2)
    return dist

def astar(src, target):
    states = [src]
    q=0
    visited_states = set()

    while len(states):
        print(f"level: {q}")
        moves = []
        for state in states:
            visited_states.add(tuple(state))
            print_grid(state)
            if state == target:
                print("Success")
                return
```

DATE: / /
PAGE:

```
        moves += [move for move in possible_moves(state,
                  visited_states) if move not in moves]

        costs = [q + h(move, target) for move in moves]
        states = [moves[i] for i in range(len(moves))
                  if costs[i] == min(costs)]
        q += 1
    print("No Sucess")

def possible_moves(state, visited_states):
    b = state.index(-1)
    d = []
    if q > b - 3 >= 0:
        d += 'u'
    if q > b + 3 >= 0:
        d += 'd'
    if b not in [2,5,8]:
        d += 'r'
    if b not in [0,3,6]:
        d += 'l'

    pos_moves = []
    for move in d:
        pos_moves.append(gen(state, move, b))
    return [move for move in pos_moves if
            tuple(move) not in visited_states]

def gen(state, direction, b):
    temp = state.copy()
    if direction == 'u':
        temp[b-3], temp[b] = temp[b], temp[b
    if direction == 'd':
        temp[b+3], temp[b] = temp[b], temp[b+
```

```
if direction == 1:
    temp[b-1], temp[b] = temp[b], temp[b-1]
return temp

src = [1, 2, 5, 3, 4, -1, 6, 7, 8]
target = [-1, 1, 2, 3, 4, 5, 6, 7, 8]

astar(src, target)
```

Output :-

Level : 0

1 2 5
3 4 -
6 7 8.

Level 1 :

1 2 -
3 4 5
6 7 8

Level : 2

1 - 2
3 4 5
6 7 8.

Level : 3

- 1 2
3 4 5
6 7 8.

success.

Lab - 6

8- Puzzle using A* Algorithm :-

```
def print_grid(src):
    state = src.copy()
    state[state.index(-1)] = '_'
    print(
        f"""
    {state[0]} {state[1]} {state[2]}
    {state[3]} {state[4]} {state[5]}
    {state[6]} {state[7]} {state[8]}
    """)

def h(state, target):
    dist = 0
    for i in state:
        d1, d2 = state.index(i), target.index(i)
        x1, y1 = d1 % 3, d1 // 3
        x2, y2 = d2 % 3, d2 // 3
        dist += abs(x1 - x2) + abs(y1 - y2)
    return dist

def astar(src, target):
    states = [src]
    g = 0
    visited_states = set()
    while len(states):
        print(f"Level : {g}")
        moves = []
        for state in states:
            visited_states.add(tuple(state))
            print_grid(state)

            if state == target:
                print("success")
```

```
moves += [move for move in possible_moves(
        stack, visited_states) if move not in
        moves]
costs = [g + man(move, target) for move in moves]
states = [moves[i]
        for i in range(len(moves)) if costs[i] ==
        min(costs)]

g += 1
print("No success")

def (possible_moves (state, visited_states):
    b = state.index(-1);
    d = []
    if g > b - 3 >= 0;
        d += 'u'
    if g > b + 3 >= 0;
        d += 'd'
    if b not in [2, 5, 8]:
        d += 'r'
    if b not in [0, 3, 6]:
        d += 'l'
    pos_moves = []
    for move in d:
        pos_moves.append (gen (state, move, b))
return [move for move in pos_move if tuple
(move) not in visited_states].

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

src = [1, 2, 5, 3, 4, -1, 6, 7, 8]
target = [-1, 1, 2, 3, 4, 5, 6, 7, 8]

astar(src, target)
```

Outputs-

level : 0.

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

level 1:

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

level 2:

```
1 - 2
3 4 5
6 7 8.
```

level 3:

```
- 1 2
3 4 5
6 7 8.
```

Success.

DATE: / /
PAGE:

Algorithm:

1) Create the initial state & goal state for the problem. In A*, the heuristics is considered, lower heuristic node is considered in each step.
   fvalue = hvalue + path cost

2) Initially expand the node, find the location of empty & generate the node
   Calculate heuristic function value
   f(x) = h(x) + g(x)

3) Maintain 2 list namely 'open' & 'close'
   The nodes (states) generated are stored in open list, sort using f(x) values.
   The explored nodes are stored in close list & removed from open.

4) The goal is reached when h(x) = 0, implies that all tiks are in correct position.

21/12/23

DATE: / /
PAGE:

Vaccum cleaner Agent Simulation Program

Algorithm:

1) Define Goal state and cost.
   Initialize the goal state representing the cleanliness status of rooms A and B.

2) Take User input for Vaccum's location and Room status.

3) Execute Actions Based on location and status

4) Perform Actions Based on status of other Room.

5) Display the final Results.
   Display the final goal state indicating the cleanliness status of room A and B.

Code:-

```
def vaccum_world():
    goal_state = {'A':'0', 'B':'0'}
    cost = 0

    location_input = input("Enter Location of vaccum")
    status_input = input("Enter status of " + location_input)
    status_input_complement = input("Enter status of other room")
    print("Initial Location Condition" + str(goal_state
```

DATE: __/__/____
PAGE: _____

```
if location_input == 'A':
    print("Vaccum is placed in Location A")
    if status_input == '1':
        print("Location A is Dirty:")
        goal_state['A'] = '0'
        cost += 1
        print("cost for Cleaning A" + str(cost))
        print("Location A has been cleaned.")
        if status_input_complement == '1':
            print("Location B is Dirty.")
            print("Moving right to the Location B.")
            cost += 1
            print("cost for moving RIGHT" + str(cost))
            goal_state['B'] = '0'
            cost += 1
            # print("cost for moving RIGHT" + str(cost))
            print("Cost for Suck + str(cost))
            print("Location B has been cleaned")
    else:
        print("No action" + str(cost))
        print("Location B is already clean")

if status_input_complement == '1':
    print("Location B is Dirty")
    print("Location B is Dirty")
    print("Moving right to the location B")
    cost += 1
    print("cost for moving Right" + str(cost))
    goal_state['B'] = '0'
    cost += 1
    print("cost for suck" + str(cost))
    print("Location B hass been cleaned")
```

DATE: __/__/____
PAGE: _____

```
    print("Location B is already clean")

if status_input == '0':
    print("Location A is already clean")
    if status_input_complement == '1':
        print("Location B is Dirty")
        print("Moving Right to the location B")
        cos += 1
        print("cost for moving right" + str(cost))
        goal_state['B'] = '0'
        cos += 1
        print("cost for suck" + str(cost))
        print("Location B has been cleaned")
    else:
        print("No action" + str(cost))
        print(cost)
        print("Location B is already clean")

else:
    print("Vaccum is placed in location B")

    if status_input == '1':
        print("Location B is dirty")
        goal_state['B'] = '0'
        cos += 1
        print("cost for cleaning" + str(cost))
        print("Location B has been cleaned")

        if status_input_complement == '1':
            print("Location A is Dirty")
            print("Moving left to the location A")
            cos += 1
            print("cost for moving left" + str(cost))
```

```
goal_state['A'] = '0'
cost += 1
print("Cost for suck " + str(cost))
print("Location A has been cleaned")
else:
print("No Action " + str(cost))
print("Location A is already clean")
print("Goal State")
print(goal_state)
print("Performance Measurement " + str(cost))
vaccum_world()
```

O/p:-

```
Enter location of vaccum B
Enter status of B 1
Enter status of other room 1
Initial location condition {A : 1, B : 0}
vaccum is placed in location B
B is dirty.
cost for cleaning 1
B has been cleaned.
Location A is dirty.
moving left to the location A
Cost for moving left 2
Cost for Suck 3
Location A has been cleaned
Goal State:
{'A':'0', 'B':'0'}
Performance measurement : 3
```

Knowledge based entailment :-

```
variable = {"sunny":0, "cloudy":1, "raining":2}
priority = {'(':3, "V":1, "^":2}

def eval(i, val1, val2):
    if i == "^":
        return val2 and val1
    return val2 or val1

def isOperand(c):
    return c.isalpha() and c != "V"

def isLeftparanthesis(c):
    return c == "("

def isRightparanthesis(c):
    return c == ")"

def isEmpty(stack):
    return len(stack) == 0

def peek(stack):
    return stack[-1]

def hasLesserEqualPriority(c1, c2):
    try:
        return priority[c1] <= priority[c2]
    except KeyError:
        return False
```

```
def toPostfix(infix):
    stack = []
    postfix = ""
    for c in infix:
        if isOperand(c):
            postfix += c
        else:
            if isLeftparanthesis(c):
                stack.append(c)
            else if isRightparanthesis(c):
                operator = stack.pop()
                while not isLeftparanthesis(operator):
                    postfix += operator
                    operator = stack.pop()
            else:
                while (not isEmpty(stack)) and
                hasLessorEqualPriority(c, peek(stack)):
                    postfix += stack.pop()
                stack.append(c)
    while not isEmpty(stack):
        postfix += stack.pop()
    return postfix

def evaluatePostfix(exp, comb):
    stack = []
    for i in exp:
        if isOperand(i):
            stack.append(comb[variable.index(i)])
        elif i == "~":
            val1 = stack.pop()
            stack.append(not val1)
        else:
            val1 = stack.pop()
            val2 = stack.pop()
            stack.append(_eval(i, val2, val1))
    return stack.pop()

def checkEntailment():
    kb = input("Enter the knowledge base:")
    query = input("Enter the query:")

    combinations = [
        [T, T, T],
        [T, T, F],
        [T, F, T],
        [T, F, F],
        [F, T, T],
        [F, T, F],
        [F, F, T],
        [F, F, F],
    ]

    postfix_kb = toPostfix(kb)
    postfix_q = toPostfix(query)
    for combination in combinations:
        eval_kb = evaluatePostfix(postfix_kb, combination)
        print(combination, "kb=", eval_kb, "; q="eval
        if eval_kb == True:
            if eval_q == False:
                print("Doesn't entail!!")
                return False
    print("Entails")

if name_ == "main":
    checkEntailment()
```

DATE: __/__/____
PAGE: _____

Output :- Enter KB : (cloudy ∨ Rainy) ∧ (Sunny ∨ cloudy)
Enter query : cloudy ∨ Rainy ∨ Sunny.

Cloudy	Rainy	Sunny	KB	Query
F	F	F	F	F
F	F	T	F	T
F	T	F	T	T
F	T	T	F	T
T	F	F	T	T
T	F	T	T	T
T	T	F	T	T
T	T	T	T	T

Entails.

DATE: __/__/____
PAGE: _____

Knowledge Based Resolution

```
def disjunctify(clauses):
    disjuncts = []
    for clause in clauses:
        disjuncts.append(tuple(clause.split('v')))
    return disjuncts

def getResolvant(ci, cj, di, dj):
    resolvant = list(ci) + list(cj)
    resolvant.remove(di)
    resolvant.remove(dj)
    return tuple(resolvant)

def resolve(ci, cj):
    for di in ci:
        for dj in cj:
            if dj == '~' + dj or dj == '~' + di:
                return getResolvant(ci, cj, di, dj)

def checkResolution(clause, query):
    clauses += [query if query.startswith('~')
                else '~' + query]
    proposition = '^'.join[['('+clause+')'
                  clause in clauses]]
    print('Trying to prove {proposition} by
                  contradiction...')
    clauses = disjunctify(clauses)
    resolved = False
    new = set()

while not resolved:
    n = len(clauses)
```

```
for (ci, cj) in pairs:
    resolvant = resolve(ci, cj)
    if not resolvant:
        resolved = True
        break
    new = new.union(set(resolvant))
if new.issubset(set(clauses)):
    break
for clause in new:
    if clause not in clauses:
        clauses.append(clause)

if resolved:
    print('Knowledge Base entails the query
        proval by resolution')
```

O/p:-

Enter the clauses seperated by a space (AVB)^(CVD
V j) ^ (EVG)

Enter the query: A V B V C

Trying to prove ((A) ^ (V) ^ (B^(C)^(D)^(J)^(E)
(V)^(G))^(A V B V C) by contradiction

Knowledge Base entails the query proved by
resolution

DATE: / /
PAGE:

Implementation of unification in FO Logic.

DATE: / /
PAGE:

19/1/24

Unification Algorithm:

Step 1: If x_1 or x_2 is a variable or constant, then:

a) Else If x_1 is a identical variable, then return NIL.

b) Else if x_1 is a variable,
   a. then if x_1 occurs in x_2, then return FAILURE
   b. Else return {(x_2/x_1)}.

c) Else if x_2 is a variable,
   a. If x_2 occurs in x_1, then return FAILURE,
   b. Else return {(x_1/x_2)}.

d) Else return FAILURE.

Step 2: If the initial Predicate symbol in x_1 and x_2 are not same, then return FAILURE.

Step 3: If x_1 and x_2 have a different number of arguments, then return FAILURE.

Step 4: Set substitution set (SUBST) to NIL.

Step 5: For i=1 to the number of elements in x_1.

a) Call unify function with the ith element of x_1 and ith element of x_2, and put the result into S.

b) If S = failure then returns failure

c) If S ≠ NIL then do,
   a. Apply S to the remainder of both L1 and L2.
   b. SUBST = APPEND (S, SUBST).

Step 6: Return SUBST.

```
import re

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

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

def isConstant(char):
    return char.isupper() and len(char)==1

def replaceAttributes(expr, old, new):
    attr = getAttributes(expr)
    for index, val in enumerate(attr):
        if val == old:
            attr[index] = new
    predicate = getInitialPredicate(expr)
    return predicate + "(" + ",".join(attr)+")"

def getFirstPart(expr):
    attr = getAttributes(expr)
    return attr[0]

def unify(exp1, exp2):
    if exp1 == exp2:
        return []
    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False
    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False
        else:
            return [(exp2, exp1)]
    if isVariable(exp2):
        if checkOccurs(exp2, exp1):
            return False
        else:
            return [(exp1, exp2)]
    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Predicates do not match. Cannot be unified")
        return False
    attributeCount1 = len(getAttributes(exp1))
    attributeCount2 = len(getAttributes(exp2))
    if attributeCount1 != attributeCount2:
        return False
    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSub = unify(head1, head2)
    if not initialSub:
        return False
```

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

remainingSub = unify (tail1, tail2)
if not remainingsub:
    return False

initialSub.extend (remainingsub)
return initialSub

exp1 = input ("Exp 1:");
exp2 = input ("Exp 2")
subs = unify (exp1, exp2)
print ("substitution:")
print (subs)
```

Output:

```
Exp 1: knows (x, john)
Exp 2: knows (smith, y)

Substitution:
[('smith', 'x'), ('john', 'y')]
```

Forward Reasoning

```
import re

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

def getAttributes (string):
    expr = '\([^)]+\)'
    matches = re.findall (expr, string)
    return matches

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

Fact:
Class Fact:
    def __init (self, expression):
        self.expression = expression
        predicate, params = self.splitExpression
                            (expression)
        self.result = any (self.getConstants())

    def getresult (self):
        return self.result

    def getConstants (self):
        return [None if isVariable (c) else c for c
                in self.params]
```

```
def substitute(self, constants):
    c = constants.copy()
    f = f"{self.predicate}({','.join([constants.pop(0) if isVariable(p) else p for p in self.params])})"
    return Fact(f)

class KB:

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

```
def main():
    kb = KB()
    n = int(input("Enter no. of statements in KB:"))

for i in range(n):
    kb.tell(input())

print("Enter query")
query = input()
kb.query(query)

kb.display()

main()
```

Output

Enter FOLs

Exp 1: know(x, john)

Exp 2: know(smith, y)

Substitution:

First Order Logic to CNF:

```
import re

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


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


def DeMorgan(Sentence):
    string = ''.join(list(Sentence).copy())
    string = string.replace('~~', '')
    flag = '[' in string
    string = string.replace('~[', '[')
    string = string.strip(']')
    for predicate in getPredicates(string):
        string = string.replace(predicate, f'~
                                {predicate}')
    s = list(string)
    for i, c in enumerate(string):

        if c == ']':
            s[i] = '['
        elif c == '[':
            s[i] = ']'
    string = ''.join(s)

    string = string.replace('~~', '')
```

```
def fol_to_cnf(fol):
    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] + '=>'
        + statement[i+1:] + ']&[' + state
```

Output:

Enter FOL statement : ∀x(P(x) → Q(x) → happy(x))

FOL to CNF : P(x) → Q(x) → Happy(x)