17/01/23

Date ___/___/___

Page _____

LAB - 1

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('   |   |')
    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[5] == 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[7] == le)
            or (bo[3] == le and bo[5] == le and bo[7] == le)
```

```
def playersMove():
    run = True
    while run:
        move = int(input("Select a position to place"))
        if move > 0 and move < 10:
            if spaceIsFree(move):
                run = False
                insertLetter('X', move)
            else:
                print("Enter a number within the range")

def compMove():

        else:
            print("Space is already occupied")

def compMove():
    posMoves = [x for x, letter in enumerate(board
                if letter == ' ' and x != 0]
    move = 0
    for let in ['O', 'X']:
        for i in posMoves:
            boardCopy = board[:]
            boardCopy[i] = let
            if isWinner(boardCopy, let):
                return i
    return None

    winning_move = isWinn
    for i in posMoves:
        if i in [1,3,7,9]:
            cornersOpen.append(i)
    return random.random(corners)
```

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

edges = list(set(posMoves) & set([2,4,6,8]))
if edges:
    return random.random(edges)
```

Algorithm:

Player Move:

1. Set run to 'True' and enter the loop.
2. While run is True:
   - Take move input from user
   - If move is in range of 1 to 10, check if the space is free:
     - If the space is free, X is inserted
     - If the space is not free, player is asked to enter another number.
   - The loop continues until run = True.

Computer Move:

1. Create a list of possible moves consisting of empty spaces in the board.
2. For each move of computer:
   - For each possible move, create a board copy and check if it is a winning move.
   - If there is no possible moves:
     - Check if empty corners are there and generate a random number among them
     - Check if '5' position is free. Place 'O' there
     - Check If above two condition dosnt satisfy check for remaining empty positions [2,4,6,8] and generate a random number to place 'O'

```
Harshitha R-1BM21CS075
[1, 2, 3, 4, 5, 6, 7, 8, 9]
+-------------------------------+
|               |               |
|       1       |       2       |       3       |
|               |               |
+-------------------------------+
|               |               |
|       4       |       5       |       6       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       8       |       9       |
|               |               |
+-------------------------------+
computer's turn :
+-------------------------------+
|               |               |
|       X       |       2       |       3       |
|               |               |
+-------------------------------+
|               |               |
|       4       |       5       |       6       |
|               |               |
+-------------------------------+
|               |               |
|       7       |       8       |       9       |
|               |               |
+-------------------------------+
Your turn :
enter a number on the board :5
+-------------------------------+
```

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

```
+---+---+---+
| X | O | 3 |
+---+---+---+
| 4 | O | X |
+---+---+---+
| 7 | 8 | 9 |
+---+---+---+
computer's turn :
+---+---+---+
| X | O | 3 |
+---+---+---+
| 4 | O | X |
+---+---+---+
| 7 | X | 9 |
+---+---+---+
Your turn :
enter a number on the board :7
```

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

```
+---+---+---+
|   |   |   |
| X | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| 4 | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| O | X | O |
|   |   |   |
+---+---+---+
computer's turn :
+---+---+---+
|   |   |   |
| X | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| X | O | X |
|   |   |   |
+---+---+---+
|   |   |   |
| O | X | O |
|   |   |   |
+---+---+---+
```

24/11/23

Date __/__/__
Page _____

LAB - 2

8 Puzzle BFS Algorithm

Initial State:

1	2	3
	4	6
7	5	8

Goal State

1	2	3
4	5	6
7	8	

Rules:

- The empty tile can move in 4 directions (i) up (ii) down (iii) right (iv) left.
- Cannot move diagonally.
- Can take one step at a time.

0	x	0
x	#	x
0	x	0

0 - 2 possible moves
x - 3 possible moves
- 4 possible moves.

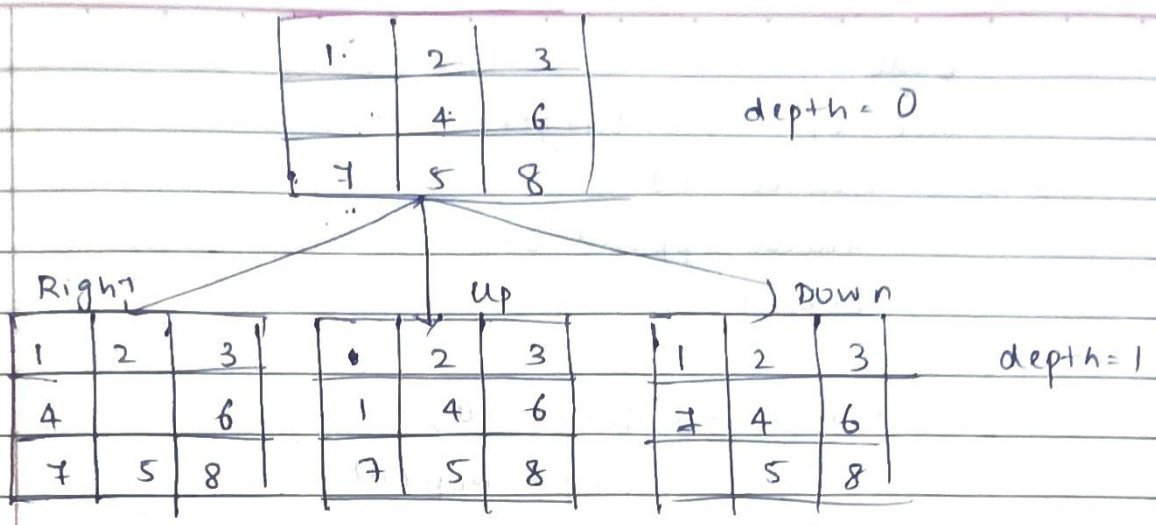
Breadth first Algorithm: (uninformed / Non-heuristic approach).

→ Complexity $O(b^d)$ where
b - branching factor
d - depth factor.

For 8 puzzle problem:

Branching factor b = all possible moves of empty tile at each position / No of tiles

$= \frac{24}{9} \approx 3$



ALGORITHM:

1. Initialize Queue and Explored set:
2. BFS Loop:
   → Dequeue a state from the front of the queue.
   → Add the dequeued state to the visited set.
   → Check if the dequeued state is target. If yes, print "success" and return.
   → Generate possible moves from the current state which is not visited.
   → Enqueue these state to queue.

3. Possible Moves Generation:
   → Find the index of the empty spot.
   → Initialize an empty array to store possible directions.
   → Check for possible moves - up, down, left, right.
   → Generate states based on the possible moves on the current state.

4. Move Generation:
   → Create a copy of the current state.
   → Depending on the specified move, swap the empty spot with the adjacent tile.
   → Return the new state after the move.

Code:

```
def bfs (src, target):
    queue = []
    queue.append(src)
    exp = []

    while len(queue)>0;
        source = queue.pop(0)
        exp.append(source)
        print(source)

        if source == target:
            print("success")
            return

        poss_move_to_do = []
        poss_moves_to_do = possible_moves(source,ex

        for move in poss_moves_to_do:
            if move not in exp and move not in qu
                queue.append(move)

def possible_moves(state, visited_states):
    b = state.index(0)
    d = []
    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_it_can = []
    for i in d:
        pos_moves_it_can.append(gen(state,i,b))
    return [move_it_can for move_it_can in
            pos_moves_it_can if move_it_can
            not in visited_states]

def gen(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[b] = temp[b], temp[b+1]
return temp.

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

24/11

```
Harshitha R-1BM21CS075
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
```

08/12/23

Date __/__/__
Page

LAB - 3

8 Puzzle DFS Algorithm

```
def id_dfs(puzzle, goal, get_moves)
    import itertools

def dfs(route, depth):
    if depth == 0:
        return
    if route[-1] == goal:
        return route
    for moves 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 = []
    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-1], temp[b] = temp[b], temp[b-1]
    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

initial = [1, 2, 3, 0, 4, 6, 9, 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 to find a solution")
```

```
Harshitha R-1BM21CS075
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]]
```

08/12/23

Date __/__/__

Page _____

LAB-4

8 puzzle A* Approach

```
Class Node:
    def (self, state, parent = None, g = 0, h = 0)
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h

    def f(self):
        return self.g + self.h

def astar (start, goal, heuristic):
    openset = [Node (start, None, 0, heuristic (start))]
    visited_set = set()

    while openset:
        current_node = heapq.heappop(openset)

        if current_node.state == goal:
            return reconstruct_path (current_node)

        visitedset.add (current_node.state)
        successors = expand_node (current_node)

        for successor in successors:
            if successor.state in visitedset:
                continue

            if successor.state not in (node.state
                                       for node in open_set)
                heapq.heappush (open_set, successor)
```

```
    else:
        existing_node = next(
            node for node in open_set if
            node.state == successor.
        if successor.g < existing_node.g:
            existing_node.parent = successor.parent
            existing_node.g = successor.g

    return None.

def expand_node(node):
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for move in moves:
        new_state = (node.state[0] + move[0],
                     node.state[1] + move[1])

        if 0 <= new_state[0] < len(grid) and
            0 <= new_state[1] < len(grid)

            if grid[new_state[0]][new_state[1]]
                != "obstacle":
                new_g = node.g + 1

                new_h = heuristic_function(new_state)

                successor = Node(new_state, node,
                                 new_g, new_h)
                successors.append(successor)
    return successor
```

Date __/__/__

Page _____

```
def reconstruct_path(node):
    path = []
    while node:
        path.append(node.state)
        node = node.parent
    return path[::-1]
```

```
Harshitha R-1BM21CS075
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 _
```

22/12/23

Date __/__/__
Page

LAB-5

VACUUM CLEANER

Consider a grid of m x n. This is the configuration of the room. Clean status is given as dirty-1, clean-0.

Algorithm: (Grid)

- Cleaning of dirty cell:
  - → Consider a room.
  - → For each row of the room iterate from left to right if its even row & vice versa.
  - → If floor[i][j] == 1, then the cell is dirty. Make floor[i][j] = 0 to make it clean.

- Check if all the cells of a room are clean.
  - → if all cells of a room are clean return true. So that it can proceed to next room
  - → if any one cell is atleast is dirty return false.

- Main function:
  - → Input the number of rows and columns for each room. Enter clean/dirty status.
  - → Pass room 1 to cleaning().
  - → Once all the cells are clean of both room, check it by passing to check_status() function.
  - → If ~~the~~ all the rows are clean then ~~move~~ end the process else call the clean() function again.

Algorithm: (for single cell room)

1. clean_room (room_name, is_dirty)
   → It will check if the room is dirty: if so it updates the status to 0 (clean=0, dirty=1).
   → If it is already clean, the status is retained

2. Main function
   → Input room status of both room.
   → Pass room1 first followed by room2 to clean.
   → Once both the rooms are clean, call the clean_room function to check if it is dirty again.
   → If it is clean ~~return to room~~ exit the process

Code:

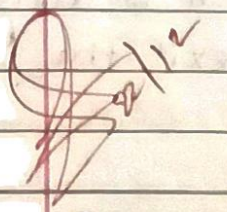
```
def clean_room (room_name, is_dirty):
    if is_dirty:
        print (f"Cleaning {room_name} (Room was dirty)")
        print (f"{room_name} is now clean")
        return 0
    else:
        print (f"{room_name} is already clean")
        return 0

def main():
    rooms = ["Room 1", "Room 2"]
    row_status = []
    for room in rooms:
        status = int (input (f"Enter clean status for {room} (1 for dirty, 0 for clean): "))
        room_status.append ((room, status))
```

```
for i, (room, status) in enumerate(room_statuses):
    room_statuses[i] = (room, clean_room(room, status))

print(f"Returning to {rooms[0]} to check if has become dirty again")
room_statuses[0] = (rooms[0], clean_room(rooms[0], room_statuses[0]))

print(f"{rooms[0]} is {'dirty' if room_statuses[0][1] else 'clean'} after checking")
```



4 rooms:

```
def clean_room(floor, room_row, room_col):
    if(floor[room_row][room_col] == 1):
        print("Room is Dirty")
        print("Cleaning Room")
        floor[room_row][room_col] = 0
    else:
        print("Room is already clean")

def main():
    rows = 2
    cols = 2
    floor = [[0,0],[0,0]]

    for i in range(rows):
        for j in range(cols):
```

Date __/__/__

Page _____

```
        status = int (input (f"Enter clean status
                                for Room"))
        floor[i][j] = status

for i, in range (rows):
    for j in range (cols):
        clean_room (floor, i, j)

# check if the initial room is dirty again
clean_room (floor, 0, 0)
```

```
Harshitha R-1BM21CS075
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.
```

```
Harshitha R-1BM21CS075
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): 1
Enter clean status for Room at (2, 1) (1 for dirty, 0 for clean): 0
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.
Cleaning Room at (1, 2) (Room was dirty)
Room is now clean.
Room at (2, 1) is already 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.
```

29/12/23

Date __/__/__
Page

LAB-6

KNOWLEDGE BASE ENTAILMENT

Entailment refers to the logical relationship between a KB and a query.
If KB entails a statement, it means that whenever the statement in the KB are true, the given query is also true.

$KB \models Q$

Knowledge Base:

- If its raining (P) then the ground is wet.
- If the ground is wet (Q), then the plants will grow (R)
- Its not the case that plants will grow ($\sim R$)

Query: whether it's raining?

Code:

```
from sympy import symbols

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_knowledge_base()
    query = symbols('p')
    result = query_entails(kb, query)

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

Proceed

Explaination: $\models$ = Entails.

→ $\alpha \models \beta$, iff in every model where α is true, β is true

→ $\alpha \models \beta$ if $(\alpha \wedge \sim\beta)$ is unsatisfiable.

Output:

Knowledge base: ~r & (Implies(p,q)) & (Implies(q,r))
Query: p
Query entails knowledge base: False

```
Harshitha R-1BM21CS075
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

29/12/23

Date __/__/__
Page

LAB-7

KNOWLEDGE BASE RESOLUTION.

```
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:
                    new_clause = resolve (c1, c2)
                    if len (new_clause) > 0 and new_clau
                                    not in knowledge_ba
                        new_clauses.add (new_clause)

        if not new_clauses:
            break
        knowledge_base |= new_clauses
    return knowledge_base
```

Date __/__/__

Page _____

```
if __name__ == "__main__":
    kb = {('p','q'),('~p','r'),('~q','~r')}
    result = resolution(kb)
    print("Original KB", kb)
    print("Resolved KB", result)
```

2nd

Harshitha R-1BM21CS075

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.

Harshitha R-1BM21CS075

Step	Clause	Derivation
1.	PvQ	Given.
2.	~PvR	Given.
3.	~QvR	Given.
4.	~R	Negated conclusion.
5.	QvR	Resolved from PvQ and ~PvR.
6.	PvR	Resolved from PvQ and ~QvR.
7.	~P	Resolved from ~PvR and ~R.
8.	~Q	Resolved from ~QvR and ~R.
9.	Q	Resolved from ~R and QvR.
10.	P	Resolved from ~R and PvR.
11.	R	Resolved from QvR and ~Q.
12.		Resolved R and ~R to Rv~R, which is in turn null.

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

Harshitha R-1BM21CS075

Step	Clause	Derivation
1.	PvQ	Given.
2.	PvR	Given.
3.	~PvR	Given.
4.	RvS	Given.
5.	Rv~Q	Given.
6.	~Sv~Q	Given.
7.	~R	Negated conclusion.
8.	QvR	Resolved from PvQ and ~PvR.
9.	Pv~S	Resolved from PvQ and ~Sv~Q.
10.	P	Resolved from PvR and ~R.
11.	~P	Resolved from ~PvR and ~R.
12.	Rv~S	Resolved from ~PvR and Pv~S.
13.	R	Resolved from ~PvR and P.
14.	S	Resolved from RvS and ~R.
15.	~Q	Resolved from Rv~Q and ~R.
16.	Q	Resolved from ~R and QvR.
17.	~S	Resolved from ~R and Rv~S.
18.		Resolved ~R and R to ~RvR, which is in turn null.

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

19/01/24

Date __/__/__
Page

LAB-8

UNIFICATION

Example:
Knows(John, x)
Knows(John, y)

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

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

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

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 getRemainingPart(expression):
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression
    newExpression = predicate + "(" + ",".join
                    (attributes[1:]) + ")"
    return newExpression

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

    if isConstant(exp1) and isConstant(exp2)
        if exp1 != exp2:
            return false
    ← if isConstant(exp1):
            return [(exp1, exp2)]
    ← if isConstant(exp2):
            return [(exp2, exp1)]
    ← 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)
initialSubstitution = unify(head1, head2)
if not initialSubstition:
    return False
if attributeCount1 == 1:
    return initialSubstition

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

if initialSubstition != []:
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution)
remainingSubstition = unify(tail1, tail2)
if not remainingSubstitution:
    return False
initialSubstition.extend(remainingSubstition)
```

```
Harshitha R-1BM21CS075
Substitutions:
[('X', 'Richard')]
```

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```

19/01/24.

Date __/__/__
Page

LAB-9

FOL TO CNF Conversion

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

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

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.strip(']')
    s = list(string)
    for i, c in enumerate (string):
        if c == '|':
            s[i] = '&'
        elif c == '&'
            s[i] = '|'
    string = ''.join(s)
    string = string.replace('~~', '')
    return f'[{string}]' if flag else string
```

```
def Skolemization(sentence):
    SKOLEM_CONSTANTS = [f'{chr(c)}' for c in
                        range(ord('A'), ord('Z')+1)]
    statement = ''.join(list(sentence).copy())
    matches = re.findall('[∀∃].', statement)
    for match in matches[::-1]:
        statement = statement.replace(predicate)
    s = list(string)
    for i, c in enumerate(string):
        statements = re.findall('\([^\(]+\)', statement)
    for s in statements:
        statement = statement.replace(s, s[1:-1])
    for predicate in getPredicates(statement):
        attributes = getAttributes(predicate)
        if ''.join(attributes).islower():
            statement = statement.replace(match[1],
                                          SKOLEM_CONSTANTS.pop())
        else:
            aL = [a for a in attributes if a.islower()]
            aU = [a for a in attributes if not
                  a.islower()][0]
            statement = statement.replace(aU, f'
                {SKOLEM_CONSTANTS.pop(0)}({aL[0] if
                len(aL) else match[1]})')
    return statement
```

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)] | criminal(x)]

Algorithm:

1. Create a List of Skolem_constants
2. Find ∀, ∃
   If the attributes are lowercase, replace them with a skolem constant.
   Removed used skolem constant or function from the list.
   If the attributes are both lowercase and uppercase replace the uppercase attribute with a skolem function.
3. Replace ⟺ with '__'
   transform — as $Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$
4. Replace ⟹ with '_'
5. Apply demorgan's law
   replace ~[
   as ~P & ~Q if (| was present)
   replace ~[
   as ~p | ~Q if (& was present)
   replace ~~ with ' '

Example:

$\forall x$ king(x) ∧ Greedy(x) ⇒ Evil(x)

King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard)

~[King(Richard) ∧ Greedy(Richard)] ∨ Evil(Richard)

~King(Richard) ∨ ~Greedy(Richard) ∨ Evil(Richard)

Harshitha R-1BM21CS075

[~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)

20/1/24

Date __/__/__
Page _____

LAB-10

FORWARD REASONING

```
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()[i]
                    new_lhs.append(fact)
        predicate, attributes = getPredicates(self.rhs.expr)
        for key in constants:
            if constants[key]:
                attributes = attributes.replace(key, constants[key])
        expr = f'{predicate}{attributes}'
        return Fact(expr) if len(new_lhs) and
            all([f.getResult() 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.exp
        for f in self.facts])):
        print(f'\t{i+1}. {f}')
```

Example:

All facts:

1. criminal(West)
2. hostile(Nono)
3. weapon(M1)
4. missile(M1)
5. american(west)
6. sells(West, M1, Nono)
7. owns(Nono, M1)
8. enemy(Nono, America)

Query: criminal(West)

```
Harshitha R-1BM21CS075
Querying criminal(x):
        1. criminal(West)
All facts:
        1. criminal(West)
        2. hostile(Nono)
        3. weapon(M1)
        4. missile(M1)
        5. american(West)
        6. sells(West,M1,Nono)
        7. enemy(Nono,America)
        8. owns(Nono,M1)
```

```
Querying evil(x):
        1. evil(John)
```