DATE: __/__/____
PAGE: _____

Lab-1

1. Using if else ladder create python code related to age.
2. Print the integer values in reverse order
3. Printing Patterns
   i/p = 4 = o/p a) 1, 22, 333, 4444
   b) 1, 12, 123, 1234,
4. Power of x (x^n)
5. Switch case using python.

DATE: __/__/__
PAGE:

1)

```
age = int (input ("Enter your age"))
if (age < 1" or age > 150):
    Print ("enter Valid age")

elif (age <= 14):
    Print ("u are a child")
elif (age <= 20):
    Print ("u are an adult")
elif (age <= 30):
    Print ("u are a teenager")
elif (age <= 60) :
    Print ("u are middle aged")
else :
    Print ("u are a aged person")
```

O/P: Enter your age 50
u are a middle aged

2)

```
num = int ( input ("enter integer no:")).
reversed_num = 0

While num != 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
Print ("Reversed No:" + str (reversed_num))
```

O/P:

Enter Integer no: 1 2 5 8 0 7
7 0 8 5 2 1

DATE: __/__/____
PAGE: _____

3)

a)

```
n = int(input("enter length of Pattern"))
for i in range(n):
    for j in range(i+1):
        print(i+1, end=' ')
    print(end = "\t")
```

O/P: 1, 2 2, 3 3 3, 4 4 4 4

b)

```
n = int(input("Enter length of Pattern"))
print("The Pattern is:", end = " ")
for i in range(n):
    for j in range(i+1):
        print(j+1, end = ' ')
    print(end = "\t")
```

Output:

5

1, 1 2, 1 2 3, 1 2 3 4, 1 2 3 4 5

DATE: __/__/__
PAGE:

4) 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:

Enter the base and power :

base, Power = 2, 3

o/p = 8

base, Power = 3, 4

o/p = 81

10/11/23

DATE: __/__/____
PAGE: _____

(8) switch-case

```
lang = input ("Enter the language u want")

match lang:
    case "Javascript"
        Print ("You can become web developer")
    case "Python"
        Print ("You can become Data Scientist")
    case "PHP"
        Print ("You can become backender")
    case "Java"
        Print ("You can become mobile app developer")
    case _:
        Print ("Lang doesn't matter Problem solving matters")
```

O/P

Enter the language u want
— PHP
and You can become Data Scientist

Lab - 2

Program:

DATE: / /
PAGE:

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

def playMove():
    run = True
```

```
while runn:
    move = input ("Please select a position (1-9): ")
    try:
        move = int (move)
        if move > 0 and move < 10:
            if spaceIsFree (move):
                runn false
                insertLetter ('X', move)
            else:
                Print ('Sorry, this space is occupied')
        else:
            Print ('Please type a num within the range')
    except:
        print ('Please type a num')

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

cornerOpen = []
    for i in possibleMoves:
        if i in [1, 3, 7, 9]:
            cornersOpen.append (i)
```

DATE: ___/___/___
PAGE: _____

```
edges_open:[]
    for i in possible_moves:
        if i in [2,4,6,8]:
            edges_open.append(i)

def selectRandom(li):
    import random
    ln = len(li)
    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 the tic tac toe!')
    printBoard(board)

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

if not(isWinner(board,'X')):
    move = compMove()
    if move == 0:
        print('Tie game!')
```

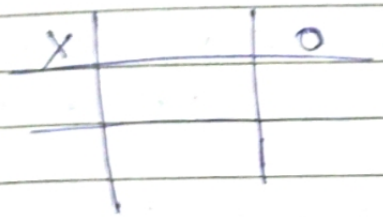
```
else:
    insertLetter('O', move)
    print('Computer placed an "O" is position',
          move, ':')
    printBoard(board)

else:
    print('X"J won this time! Good Job!')
    break

if isBoardFull(board):
    print('The game')

while True:
    answer = input('Do you want to play again?
                   (Y/N)')
if answer.lower() == 'y' or answer.lower() ==
                   'yes':

board = [' ' for x in range(10)]
print('-----------')
main()
else:
    break
```

DATE: ___/___/___
PAGE: _____

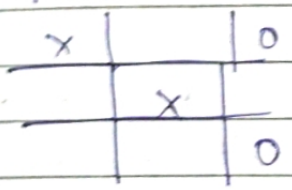
Output:

Do you want to play again ? (Y/N)
YES
Welcome to Tic Tac Toe !

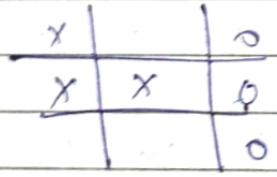
Please select a position to
Place an 'X' (1-9) : 1



Please select Position to Place an 'X' (1-9) : 5



Please select Position to place on 'X' (1-9) = 4



Please select Position to place on 'X' (1-9) = 7
Sorry O's won this time !!

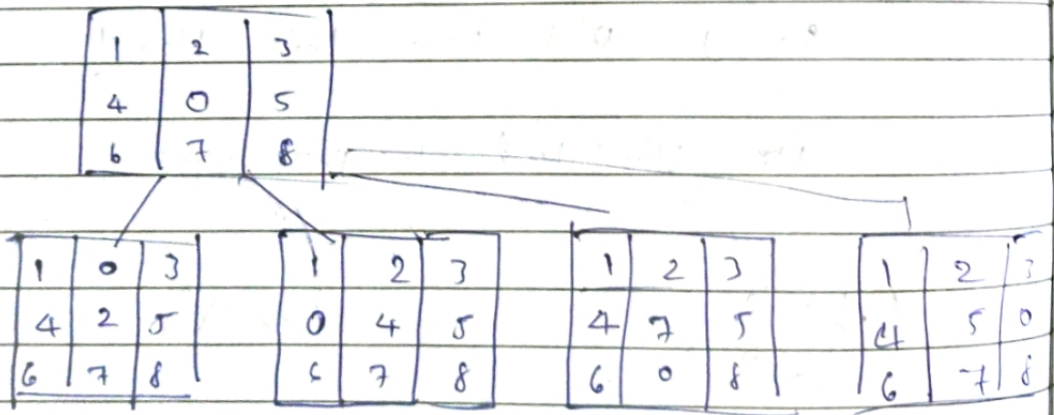
Do you want to play again ?

LAB: 3

DATE / /
PAGE

Algorithm:- 8-Puzzle game using BFS

* Initialize the puzzle.
* Create a puzzle class. Define the board configuration and the goal state.
* Shuffle the numbers to generate a random initial state.
* Print the current board.
* Implement a method to print the current state of the board in a 3 x 3 grid format.
* Move tiles.
* Implement a method to move tiles based on user i/p (up, down, left, right).
* Check for the validity of move to avoid moving tiles beyond the edges of the board. Check if it is solved.
* Game loop: Create a game loop that continues until the puzzle is solved. Move accordingly the tiles. If solved exit from loop.
* Instantiate the puzzle class. Use the game loop to interact with the user, allowing them to make moves until puzzle is solved.
* The implementation uses a simple console based interface and takes i/p from the user to specify the direction of the move.
* The game loop continues until the user solves the puzzle.



DATE / /
PAGE

8-Puzzle Game using BFS Algorithm :-

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

Algorithm

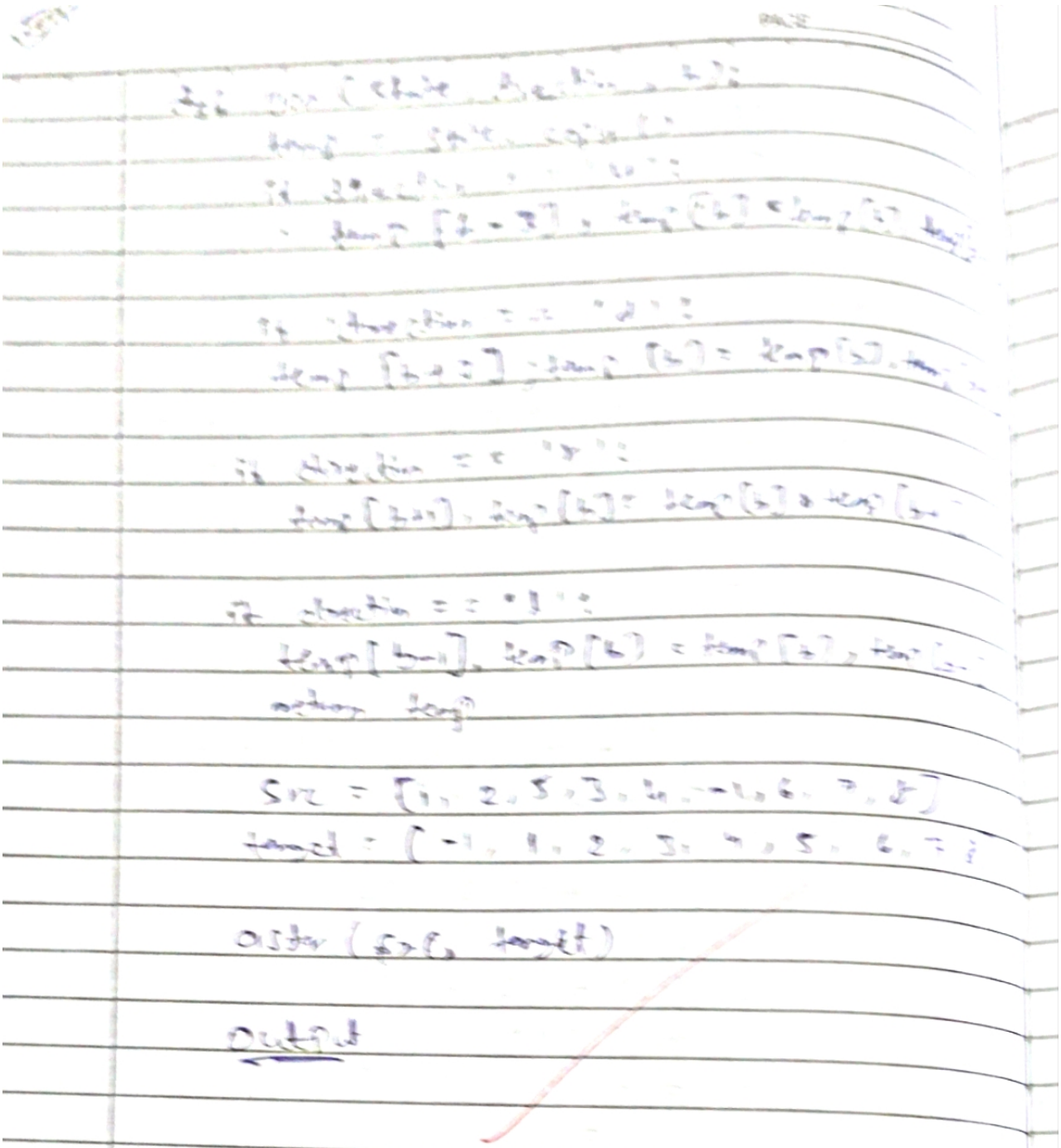
1) Start
2) Start from given configuration by generating all child nodes of it
3) Now select the child node by using least cost function where least cost = no. of moves made + number of tiles in non final configuration
4) Now keep on repeating step 2 & 3 until a final state is reached where we cannot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

Program:

Code:-

```
def Print_Grid (src):
    state = src.copy()
    state[state.index(-1)] = '_'
    print("""
    {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
        x2, y2 = d2 % 3, d2
        dist += abs(x1 - x2) + abs(y1 - y2)
    return dist

def astr(src, target):
    state = [src]
    g = 0
    visited_states = set()

    while len(states):
        print("Level : {g}")

        moves = []
        for state in states:
            visited_states.add(tuple(state))
            print_Grid(state)
```

```
if state == target:
    print("Success")
    return

moves = [move for move in possible_moves(state,
                visited_states) if move not in moves]
costs = [g + h(move, target) for move in moves]
states = [move[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 0 <= b-3 >= 0:
        d += 'u'
    if 0 <= 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]
```



Output

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-4

DATE: / /
PAGE

8-Puzzle using iterative deepening search (IDDS):

Algorithm:

1) Initialize the initial state = [ ] and goal state for the 8 Puzzle
Goal state = [1, 2, 3, 4, 5, 6, 7, 8, 0] # 0 is blank

2) Set the depth = 1 and expand the initial state the depth limit search is performed.
if state = goal.
return node
else
for neighbour in get neighbours (state):
child = PuzzleState (neighbor node)
result = dls (depth=1)
if result != True:
return result

3) After one iteration where depth = 1 increment the depth by 1 & perform 'dls'.

4) these the neighbours will generate the possible moves by swapping the '0' tile.

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

Code:-

```
def dfs( src, target, limit, visited_states ):
    if src == target:
        return True
    if limit <= 0:
        return False
    visited_states.append(src)
    moves = possible_moves(src, visited_states)
    for move in moves:
        if dfs(move, target, limit-1, visited_states):
            return True
    return False

def possible_moves(state, visited_states):
    b = state.index(-1)
    d = []
    if b not in [0, 1, 2]:
        d += 'u'
    if b not in [6, 7, 8]:
        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 move not in
                                    visited_states]
def gen(state, move, blank):
    temp = state.copy()
    if move == 'u':
        temp[blank-3], temp[blank] = temp[blank], temp
    if move == 'd':
        temp[blank+3], temp[blank] = temp[blank], temp[blank+3]
```

```
if move == 'r':
    temp[blank+1], temp[blank] = temp[blank], temp[blank+1]
if move == 'l':
    temp[blank-1], temp[blank] = temp[blank], temp[blank-1]
    return temp
def iddfs(src, target, depth):
    for i in range(depth):
        visited_states = []
        if dfs(src, target, i+1, visited_states):
            print(True)
            return
    print(False)

def get_user_input():
    print("enter the initial state")
    puzzle = []
    for i in range(3):
        row = []
        for j in range(3):
            val = int(input("enter val for position ({i},{j}): "))
            row.append(val)
        puzzle.append(row)
    return puzzle

initial_puzzle = get_user_input()
solution = iterative_deepening_search(initial_puzzle)
if solution is not None:
    print("Solution found:")
    for row in solution:
        print(row)
```

DATE
PAGE

X Output:-

Enter value for P.S (1,1) = 1
(1,2) = 2
(1,3) = 3
(2,1) = 4
(2,2) = 5
(2,3) = 6
(3,1) = 7
(3,2) = 8
(3,3) = 0

Matrix is:

1	2	3
4	5	6
7	8	9

diagonal

1	2	3
4	5	6
7	8	0

Solution found

5/12/23

DATE: / /
PAGE

* 8 - puzzle using A* Algorithm

Algorithm:

1) Create the initial state and Goal state for the problem. In A* the heuristic is combination of lower heuristic node is considered in each step. f value = h value + path cost

2) Initially expand the node find the location of empty tile and generate the nodes. Calculate the heuristic function value.

$f(x) = h(x) + g(x)$

$h(x)$ → misplaced tiles

$g(x)$ → depth from starting node (path cost)

3) Maintain two list namely open and close the explored list.
The nodes (states) generated are stored in the open list ; sort using the $f(x)$ values.
The explored nodes are state in close list and removed from open.

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

9.1

1	2	3
4	5	6
7	8	0

h. = (heuristic : manhattan distance)

1	2	3
0	4	6
7	5	8

h: 4

1	2	3
4	0	6
7	5	8

h: 3

0	2	3
1	4	6
7	5	8

h: 5

1	2	3
7	4	6
0	5	8

h: 4

Code:

```
def Print_grid (src):
    state = src.copy():
    state = [state.index(0)] = ' '
    Print (
        """
        {state[0]} {state[1]} {state[2]}
        {state[3]} {state[4]} {state[5]}
        {state[6]} {state[7]}
        ...
        """
    )
def h(state, target):
    dis = 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
```

DATE: / /
PAGE:

```
def solve(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")
                return
            moves += [move for move in possible_moves(
                state, visited_states) if move not in moves]
        costs = [g + h(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 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
```

DATE / /
PAGE

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

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

askr (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

8/12/23

Lab-5

DATE / /
PAGE

Problem: Vacuum Cleaner Problem:

Algorithm:

→ * Define Goal state and cost
* Initialize the goal state according to the cleanliness status of rooms A & B
* Set the cost counter to track the agent actions.

→ * Take user I/P for vacuum's location and room status

→ * Prompt the user to input the location of the vacuum and the status (clean/dirty) of that room location.
* I/P the complement status (clean/dirty) of the other room.

→ Execute Actions based on location and status
* Check the location of the vacuum and the status of that room
* clean the room if it's dirty and update the goal state.
* increment the cost based on the performed

→ Perform Actions based on status of other room
* check the status of the other room based on user I/P.
* clean the other room if it's dirty and set the goal state.

→ Display final results
* Display the final goal state indicating the cleanliness status of room A & B
* Show the performance measurement which is the accumulated cost of actions taken by the vacuum agent.

Code:

```
def vacuum_world():

goal_state = {'A': '0', 'B': '0'}
cost = 0

location_input = input("Enter location of vacuum:")
status_input = input("Enter status of" + location_input)
status_input_complement = input("Enter status of other room:")

if location_input == 'A':
    goal_state['A'] = status_input
    goal_state['B'] = status_input_complement
else:
    goal_state['A'] = status_input_complement
    goal_state['B'] = status_input

print("Initial Location condition:" + str(goal_state))
print("\n")

if location_input == 'A':
    print("Vacuum is placed in location A")
    if status_input == '1':
        print("Location A is dirty")
        if status_input == '1':
            goal_state['A'] = '0'
            cost += 1
            print("Cost after cleaning A:" + str(cost))
            print("Location A has been cleaned \n")

    if status_input_complement == '1':
        print("Location B is Dirty")
        print("Moving right to the loc B")
        cost += 1
```

if input == '1':
print ("Location B is dirty")

goal state ['B'] = '0'
cost += 1
Print ("cost after cleaning B" + str(cost))
print ("Location B has been cleaned.")

if status_complement == '1':
Print ("Location A is dirty")
Print ("Moving right to loc A")
cost += 1
Print ("Cost after moving right:" + str(cost))

goal state ['A'] = '0'
Cost += 1
Print ("cost for cleaning A" + str(cost))
Print ("Location A has been cleaned")

else:
Print ("no action" + str(cost))
Print ("A has already clean")

Print ("\n GOAL STATE")
print (goal_state)
print ("performance cost" + str(cost))

vacuum_cleaner()

DATE: / /
PAGE:

Output:

Enter the location of vacuum : A
Enter the status of A : 1
Enter status of other room : 1
Initial Condition : {'A': '1', 'B': '1'}

Vacuum is placed in Loc A
Location A is dirty
Cost after cleaning A : 1
Location A has been cleaned.

No Action !
Location B is already clean.

Goal state : {'A': '0', 'B': '0'}
Performance measurement : 1

22/12/23

Lab - 6

DATE / /
PAGE

* Knowledge Base (Forward chaining)

```
Variable = {"Rody": 0, "Rainy": 1, "sunny": 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()

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

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

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

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

def hasLessOrEqualPriority(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 isLeftParenthesis(c):
            stack.append(c)
        elif isRightParenthesis(c):
            operator = stack.pop()
            while not isLeftParenthesis(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 checkEntailment():
    kb = input("Enter the knowledge base: ")
    query = input("Enter the query")
    combinations = [
        [True, True, True],
        [True, True, False],
        [True, False, True],
        [True, False, False],
        [False, True, True],
        [False, True, False],
        [False, False, True],
        [False, False, False],
    ]
```

```
postfix_kb = toPostfix(kb)
postfix_q = toPostfix(query)
for combination in combinations:
    eval_kb = evaluatePostfix(postfix, combination)
    eval_q = evaluatePostfix(postfix, combination)
print(combination, "kb:", eval_kb, "q:", eval_q)

    if eval_kb == True:
        if eval_q == False:
            print("Doesn't entail!!")
            return False
print("Entails")

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

* OUTPUT:

Cloudy ∨ Rainy ∧ Sunny ∨ Cloudy

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	F	T
T	F	T	T	T
T	T	F	T	T
T	T	T	T	T

Entails

DATE / /
PAGE

* Knowledge Base Resolution:

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

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

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

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

    clauses = disjunctify (clauses)
    resolved = False
    new = set ()

    while not resolved:
        n = len (clauses)
pairs = [(clauses [i], clauses [j]) for i in range (n) for j in range(
                                                                      i+1, n)
```

```
for (ci, cj) in pairs:
    resolvent = resolve(ci, cj)
        if not resolvent:
            resolvent = True
            break
    new = new.union(set(resolvent))
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, proved by res
else:
    Print("knowledge base doesn't entail the query")


clauses = input("Enter the clauses seperated by
query = input("Enter a space:").split()

checkResolution(clauses, query)
```

DATE / /
PAGE

output:-

Enter the clauses seperated by a space:

(A ∨ B) ∧ (C ∨ D ∼ J) ∧ (E ∨ G)

Enter the query: A ∨ B ∨ C

Trying to Prove ((A) ∧ (∨) ∧ (D) ∧ (() ∧ (∨) ∧ (D) ∧ (∨) ∧ (J) ∧ (E) ∧ (∨) ∧ (G)) ∧ (A ∨ B ∨ C)
by Contradiction ...

Knowledge Base entails the query, Proved by resolution.

Lab - 7

DATE: __/__/__
PAGE: _____

1. Unification using FOL

Important Algorithm

Step 1: If Ψ_1 or Ψ_2 is a variable or constant then
a) If Ψ_1 or Ψ_2 are identical, then return NIL
b) Else if Ψ_1 is a variable,
a) then if Ψ_1 occurs in Ψ_2, then return FAILURE
b. Else return $\{(\Psi_2 / \Psi_1)\}$.
c. Else if Ψ_2 is a variable.
a. If Ψ_2 occurs in Ψ_1 then return FAILURE.
b. Else return $\{(\Psi_1 / \Psi_2)\}$.
d. Else return FAILURE.

Step 2:
If the initial predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step 3: If Ψ_1 and Ψ_2 have a different no of arguments then return FAILURE.

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

Step 5 For i=1 to no of elements in Ψ_1
a) Call unify function with ith element of Ψ_1 and ith element of Ψ_2 result into S.
b) If S = failure then return Failure.
c = If NIL then do,
Apply S to the remainder of both L_1 & L_2
SUBST = APPEND (S, SUBST).

Step 6: return SUBST

DATE / /
PAGE

Code:

import re

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

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

def replaceAttribute(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 apply(expr, subs):
    for sub in subs:
        new, old = sub
    expr = replaceAttributes(expr, old, new)
    return expr

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

DATE: ___/___/___
PAGE: ___

```
if initial_sub != []:

    tail1 = apply(tail1, initial_sub)
    tail2 = apply(tail2, initial_sub)

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

initial_sub.extend(remaining_sub)

return False

exp1 = input('Expression 1')
exp2 = input('Expression 2:')

subs = unify(exp1, exp2)
print('Substitutions:')
print(subs)
```

OUTPUT:

```
Expression 1: knows(x)
Expression 2: knows(Richard)

Substitutions:

[('x', 'Richard')]
```

DATE: __/__/____
PAGE: _____

(2) Convert FOL into Conjunctive Normal Form (CNF)

Code

```
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)
```

DATE:
PAGE:

```
def Skolemization (Sentence):
    Skolem_Constants = [f'{chr(c)}' for c in ...]

    Statement = ''.join (list (Sentence/.copy(),)

    for match in matches[::-1]:
    Statement = re.findall ('\[\[\[^1]]]+\]', statement)

    for s in Statements:
        Statements = Statements.replace (s, s[1:-1])
    for Predicate in getPredicate (Statement)
    attributes = set Attributes (Predicate)
    return Statement

def fol-CNF (fol):
    Statement = fol.replace ("<=>", "_")

    While '_' in Statement:
    i = Statement.index ('_')
new_Statement = '(' + Statement [::] + '=>' + Statement

    Statement [::] + ']'
    Statement = new_Statement

for i enumerate (statements)

    if '[' in s and ']' not in s:
        Statements[:] + = ']'

for s in Statements:
    Statement = Statement.replace (s, fol-to-cnf)
```

```
for s in statements:
    statement = statement.replace(s, fol_to_cnf(s))
expr = '\s\([^]]+\)'

statement = re.findall(expr, statement)
for s in statements:
    statement = statement.replace(s, Skolemize(s))
return statement

def main():
    statement = input("Enter FOL statement: ")
print("FOL converted to CNF:")
    Skolemization(fol_to_cnf(statement))
main()
```

OUTPUT:

```
Enter FOL statements: ∀x[P(x) -> (Q(x) -> Happy(A))]
FOL to CNF: (P(x) -> (Q(x) -> Happy(A)))

Enter FOL statement: ∀x (P(x) => Q(x) & ∃y R(y))
FOL to CNF: ∀x(P(x) | Q(x) & R(y))
```

DATE: / /
PAGE:

(3) Forward Reasoning

Algorithm:

Step 1: Let us your initial knowledge with relevant information.

Step 2: choose a rule or fact to start reasoning process

Step 3: Use the selected rule or fact to determine information through logical reasoning.

Step 4: Incorporate the newly derived information into the knowledge base

Step 5: Repeat step 2 to 4 until goal is reached or until to further reference can to made

Output:

Enter the no of statements in KB : 4
Animal (dog)
Animal (cat)
Loves (John, cat)
Friend (John)

Enter query : Friend (x)
Querying Friend (x):
1. Friend (John)

All Facts
1. Loves (John, Cat)
2. Animal (Cat)
3. Friend (John)
4. Animal (dog)

19.01.24