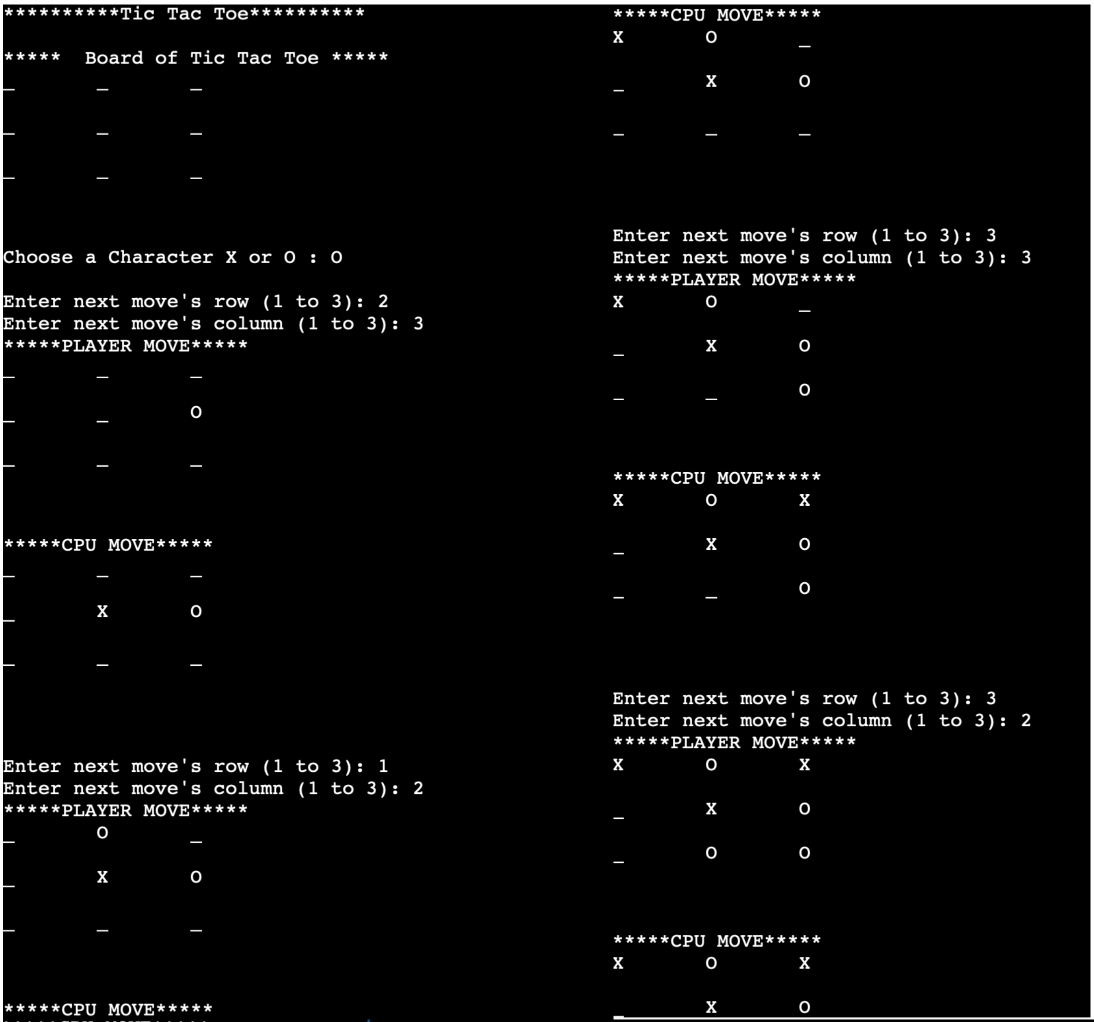
1. Implement Tic -Tac -Toe Game.

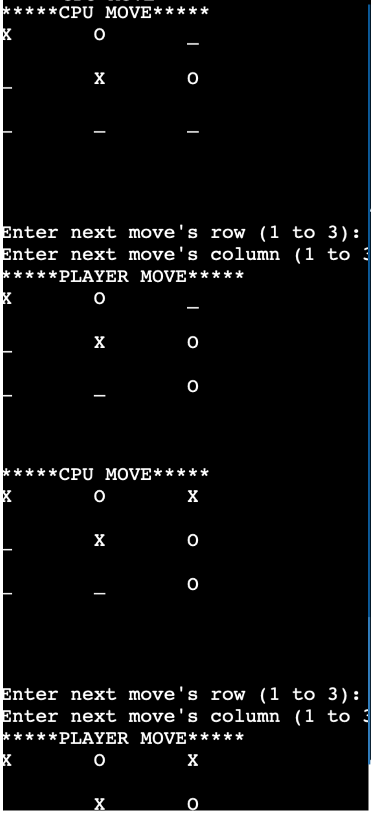
```
import random as r
ai,player='O','X'
board=[['_','_','_'],['_','_','_'],['_','_','_']]
weights=[[3,2,3],[2,4,2],[3,2,3]]
def init():
    global ai,player,board,weights
    ai,player='O','X'
    board=[['_','_','_'],['_','_','_'],['_','_','_']]
    weights=[[3,2,3],[2,4,2],[3,2,3]]
def move(row,col,ch):
    if board[row][col]=='_':
        board[row][col],weights[row][col]=ch,0
        return True
    else : return False

def display(move_type='board'):
    if move_type=='cpu' : print('*'*5+'CPU MOVE'+'*'*5)
    elif move_type=='board': print("*"*5+'  Board of Tic Tac Toe '+'*'*5)
    else :print('*'*5+'PLAYER MOVE'+'*'*5)
    for i in range(3):
        for j in range(3):
             print(board[i][j],end='\t')
        print('\n')
    print('\n')

def compare_line(s1,ch):
    return '_' in s1 and s1.count(ch)==2

def get_position():
    max_value=max([max(x) for x in weights])
    positions=[(i,weights[i].index(max_value)) for i in range(3)  if max_value in weights[i]]
    return positions

def has_tied():
    for row in board:
      if '_' in row: return False
    return True

def attacking_positiion(ch):
        default='_'
        for i in range(3):
            col=[board[0][i],board[1][i],board[2][i]]
            if compare_line(board[i],ch): return (i,board[i].index(default))
            elif compare_line(col,ch): return (col.index(default),i)
        diag1,diag2=[board[0][0],board[1][1],board[2][2]],[board[0][2],board[1][1],board[2][0]]
        if compare_line(diag1,ch):return (diag1.index(default),diag1.index(default))
        elif compare_line(diag2,ch): return (diag2.index(default),2-diag2.index(default))
        return False

def ai_move():
    global ai,player
    pos,f=attacking_positiion(ch=ai),False
    if pos!=False:(row,col),f=pos,True
    else :
        pos=attacking_positiion(ch=player)
        if pos!=False: row,col=pos
        else: row,col=r.choice(get_position())
    move(row,col,ai)
    return f

def run():
    global ai,player
    end,tied,move_type=False,False,None
    print('*'*10+ 'Tic Tac Toe'+'*'*10+'\n')
    display()
    ch=input('Choose a Character X or O : ')
    if ch=='O': ai,player=player,ai
    while(True):
        if tied:
            print('*'*10+'The match is tied'+'*'*10)
            return
        elif end:
            print('*'*10+move_type+' has own '+'*'*10)
            return
        move_type='player'
        r=int(input("\nEnter next move's row (1 to 3): "))
        c=int(input("Enter next move's column (1 to 3): "))
        if not move(r-1,c-1,player):
            print('\nEnter proper positions!!')
        else:
          display(move_type=move_type)
          tied=has_tied()
          if tied: continue
          move_type='cpu'
          end=ai_move()
          display(move_type=move_type)
          tied=has_tied()
def main():
    run()
    f='Y'
    while(f=='Y'or f=='y'):
        f=input('Do you want to play again Y or N: ')
        init()
        if f=='Y' or f=='y':run()
    print('\n\n'+'*'*10+' Thank You '+'*'*10)
main()
```

OUTPUT :

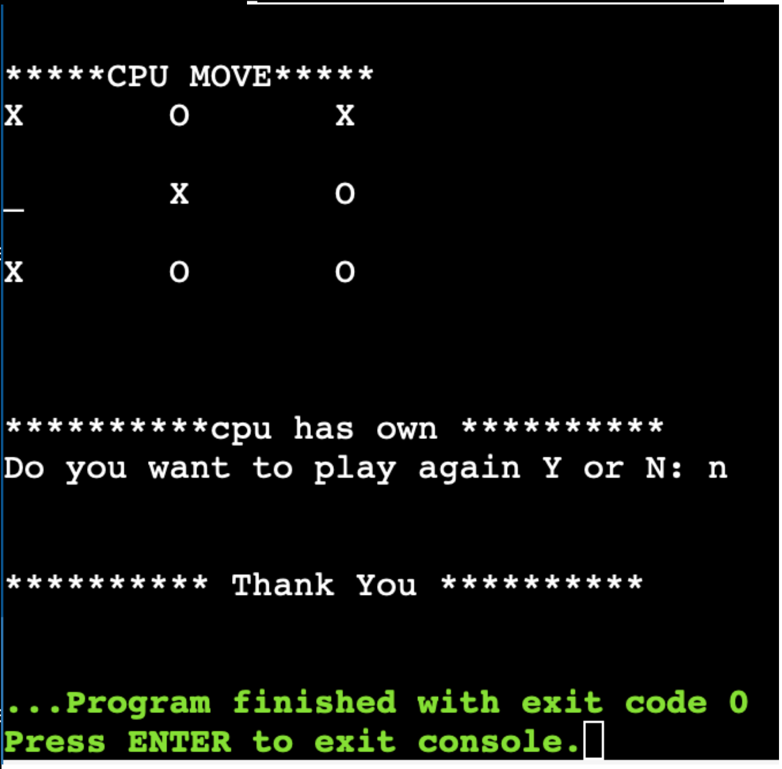


```
**********Tic Tac Toe**********

*****  Board of Tic Tac Toe *****
_       _       _

_       _       _

_       _       _



Choose a Character X or O : O

Enter next move's row (1 to 3): 2
Enter next move's column (1 to 3): 3
*****PLAYER MOVE*****
_       _       _

_       _       O

_       _       _


*****CPU MOVE*****
_       _       _

_       X       O

_       _       _



Enter next move's row (1 to 3): 1
Enter next move's column (1 to 3): 2
*****PLAYER MOVE*****
_       O       _

_       X       O

_       _       _


*****CPU MOVE*****
```

```
*****CPU MOVE*****
X       O       _

_       X       O

_       _       _



Enter next move's row (1 to 3): 3
Enter next move's column (1 to 3): 3
*****PLAYER MOVE*****
X       O       _

_       X       O

_       _       O


*****CPU MOVE*****
X       O       X

_       X       O

_       _       O



Enter next move's row (1 to 3): 3
Enter next move's column (1 to 3): 2
*****PLAYER MOVE*****
X       O       X

_       X       O

_       O       O


*****CPU MOVE*****
X       O       X

_       X       O
```



```
*****CPU MOVE*****
X       O       _

_       X       O

_       _       _



Enter next move's row (1 to 3):
Enter next move's column (1 to
*****PLAYER MOVE*****
X       O       _

_       X       O

_       _       O


*****CPU MOVE*****
X       O       X

_       X       O

_       _       O



Enter next move's row (1 to 3):
Enter next move's column (1 to
*****PLAYER MOVE*****
X       O       X

        X       O
```



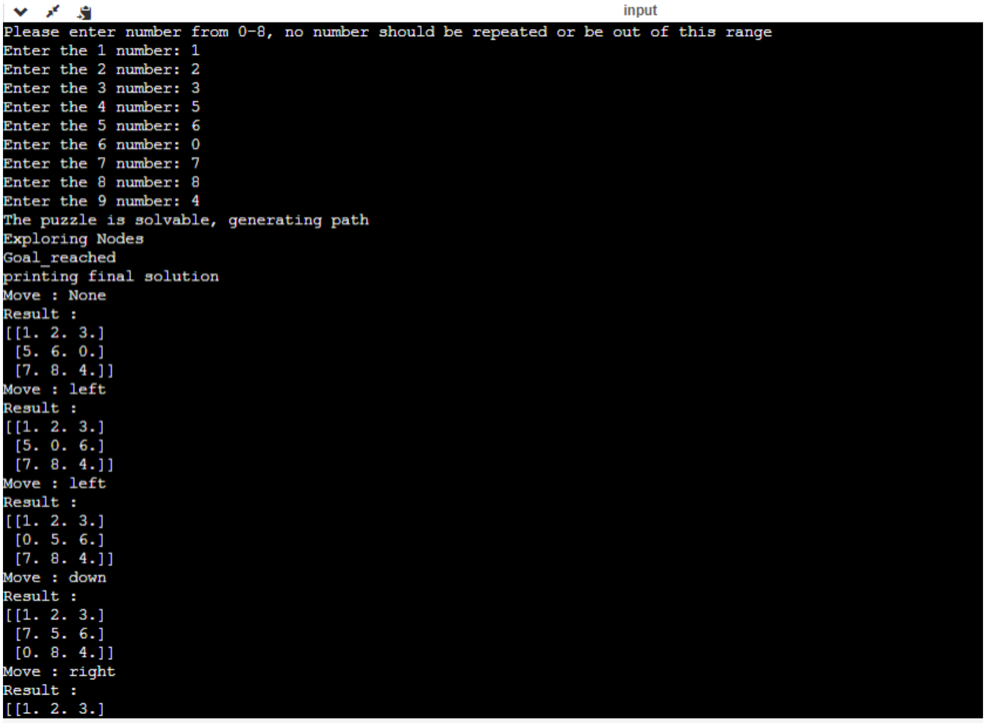
```
*****CPU MOVE*****
X       O       X

_       X       O

X       O       O



**********cpu has own **********
Do you want to play again Y or N: n


********** Thank You **********


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Solve 8 puzzle problem.

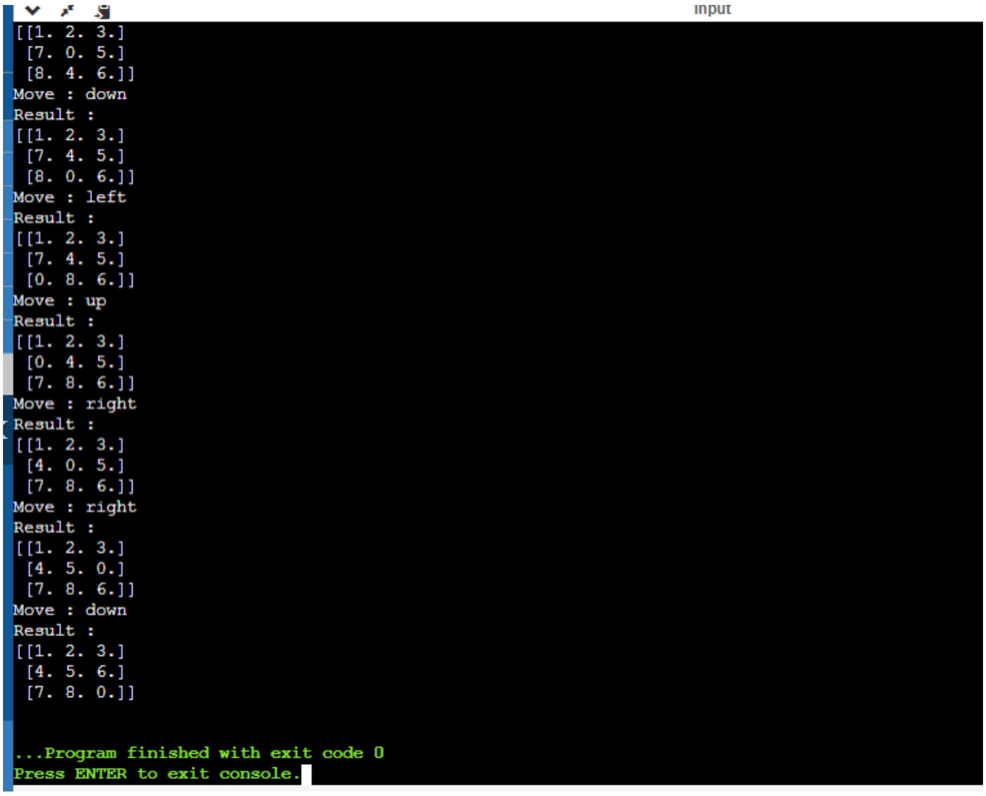
```
import numpy as np  # Used to store the digits in an array
import os  # Used to delete the file created by previous running of the program


class Node:
    def __init__(self, node_no, data, parent, act, cost):
        self.data = data
        self.parent = parent
        self.act = act
        self.cost = cost


def get_initial():
    print("Please enter number from 0-8, no number should be repeated or be out of this range")
    initial_state = np.zeros(9)
    for i in range(9):
        states = int(input("Enter the " + str(i + 1) + " number: "))
        if states < 0 or states > 8:
            print("Please only enter states which are [0-8], run code again")
            exit(0)
        else:
            initial_state[i] = np.array(states)
    return np.reshape(initial_state, (3, 3))


def find_index(puzzle):
    i, j = np.where(puzzle == 0)
    i = int(i)
    j = int(j)
    return i, j


def move_left(data):
    i, j = find_index(data)
    if j == 0:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i, j - 1]
        temp_arr[i, j] = temp
        temp_arr[i, j - 1] = 0
        return temp_arr


def move_right(data):
    i, j = find_index(data)
    if j == 2:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i, j + 1]
        temp_arr[i, j] = temp
        temp_arr[i, j + 1] = 0
        return temp_arr


def move_up(data):
    i, j = find_index(data)
    if i == 0:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i - 1, j]
        temp_arr[i, j] = temp
        temp_arr[i - 1, j] = 0
        return temp_arr


def move_down(data):
    i, j = find_index(data)
    if i == 2:
        return None
    else:
        temp_arr = np.copy(data)
        temp = temp_arr[i + 1, j]
        temp_arr[i, j] = temp
        temp_arr[i + 1, j] = 0
        return temp_arr


def move_tile(action, data):
    if action == 'up':
        return move_up(data)
    if action == 'down':
        return move_down(data)
    if action == 'left':
        return move_left(data)
    if action == 'right':
        return move_right(data)
    else:
        return None


def print_states(list_final):  # To print the final states on the console
    print("printing final solution")
    for l in list_final:
        print("Move : " + str(l.act) + "\n" + "Result : " + "\n" + str(l.data) + "\t")


def path(node):  # To find the path from the goal node to the starting node
    p = []  # Empty list
    p.append(node)
    parent_node = node.parent
    while parent_node is not None:
        p.append(parent_node)
        parent_node = parent_node.parent
    return list(reversed(p))


def exploring_nodes(node):
    print("Exploring Nodes")
    actions = ["down", "up", "left", "right"]
    goal_node = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])
    node_q = [node]
    final_nodes = []
    visited = []
    final_nodes.append(node_q[0].data.tolist())  # Only writing data of nodes in seen
    node_counter = 0  # To define a unique ID to all the nodes formed

    while node_q:
        current_root = node_q.pop(0)  # Pop the element 0 from the list
        if current_root.data.tolist() == goal_node.tolist():
            print("Goal reached")
            return current_root, final_nodes, visited

        for move in actions:
            temp_data = move_tile(move, current_root.data)
            if temp_data is not None:
                node_counter += 1
                 child_node = Node(node_counter, np.array(temp_data), current_root, move, 0)  # Create
a child node

                if child_node.data.tolist() not in final_nodes:  # Add the child node data in final node list
                    node_q.append(child_node)
                    final_nodes.append(child_node.data.tolist())
                    visited.append(child_node)
                    if child_node.data.tolist() == goal_node.tolist():
                        print("Goal_reached")
                        return child_node, final_nodes, visited
    return None, None, None  # return statement if the goal node is not reached


def check_correct_input(l):
    array = np.reshape(l, 9)
    for i in range(9):
        counter_appear = 0
        f = array[i]
        for j in range(9):
            if f == array[j]:
                counter_appear += 1
        if counter_appear >= 2:
            print("invalid input, same number entered 2 times")
            exit(0)


def check_solvable(g):
    arr = np.reshape(g, 9)
    counter_states = 0
    for i in range(9):
        if not arr[i] == 0:
            check_elem = arr[i]
            for x in range(i + 1, 9):
                if check_elem < arr[x] or arr[x] == 0:
                    continue
                else:
                    counter_states += 1
    if counter_states % 2 == 0:
        print("The puzzle is solvable, generating path")
    else:
        print("The puzzle is insolvable, still creating nodes")


# Final Running of the Code
k = get_initial()

check_correct_input(k)
check_solvable(k)
```

```
root = Node(0, k, None, None, 0)

# BFS implementation call
goal, s, v = exploring_nodes(root)

if goal is None and s is None and v is None:
    print("Goal State could not be reached, Sorry")
else:
    # Print and write the final output
    print_states(path(goal))
```

OUTPUT :



```
Please enter number from 0-8, no number should be repeated or be out of this range
Enter the 1 number: 1
Enter the 2 number: 2
Enter the 3 number: 3
Enter the 4 number: 5
Enter the 5 number: 6
Enter the 6 number: 0
Enter the 7 number: 7
Enter the 8 number: 8
Enter the 9 number: 4
The puzzle is solvable, generating path
Exploring Nodes
Goal_reached
printing final solution
Move : None
Result :
[[1. 2. 3.]
 [5. 6. 0.]
 [7. 8. 4.]]
Move : left
Result :
[[1. 2. 3.]
 [5. 0. 6.]
 [7. 8. 4.]]
Move : left
Result :
[[1. 2. 3.]
 [0. 5. 6.]
 [7. 8. 4.]]
Move : down
Result :
[[1. 2. 3.]
 [7. 5. 6.]
 [0. 8. 4.]]
Move : right
Result :
[[1. 2. 3.]
```

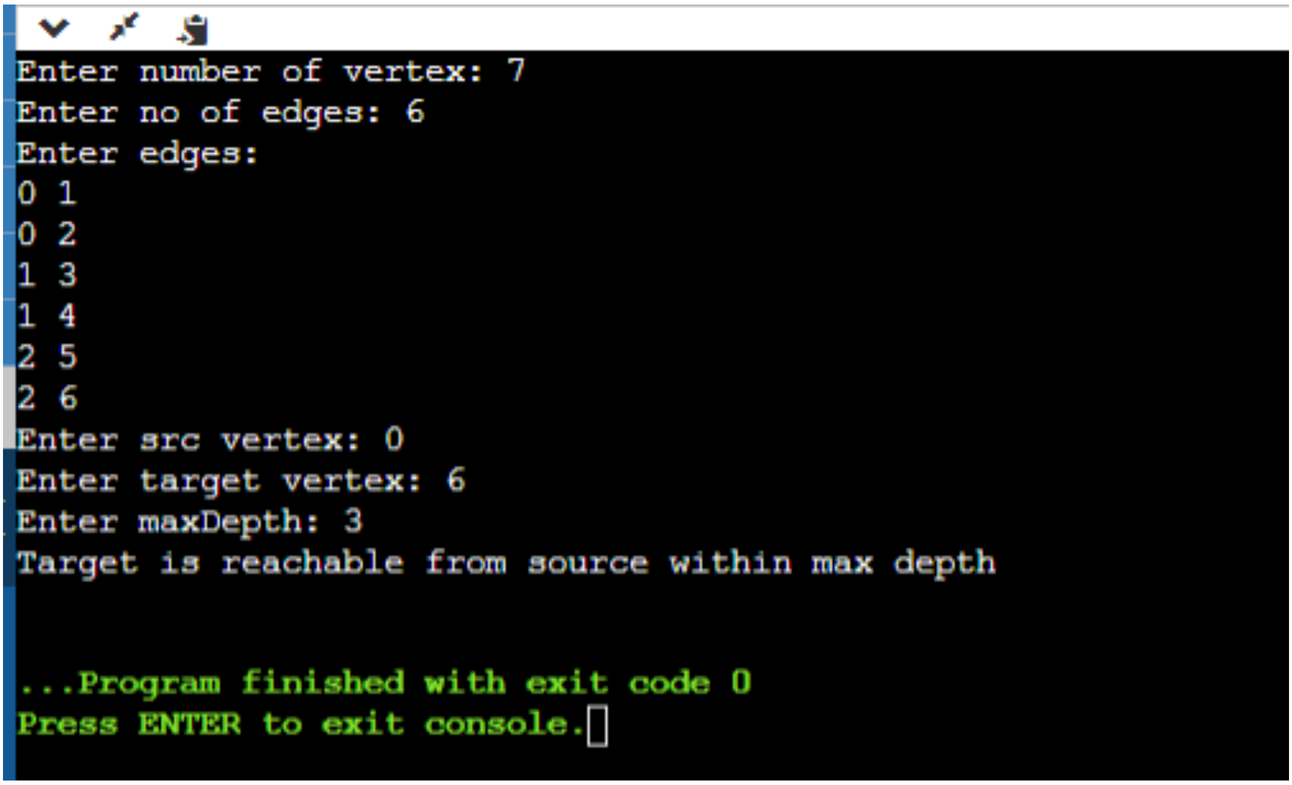


```
[[1. 2. 3.]
 [7. 0. 5.]
 [8. 4. 6.]]
Move : down
Result :
[[1. 2. 3.]
 [7. 4. 5.]
 [8. 0. 6.]]
Move : left
Result :
[[1. 2. 3.]
 [7. 4. 5.]
 [0. 8. 6.]]
Move : up
Result :
[[1. 2. 3.]
 [0. 4. 5.]
 [7. 8. 6.]]
Move : right
Result :
[[1. 2. 3.]
 [4. 0. 5.]
 [7. 8. 6.]]
Move : right
Result :
[[1. 2. 3.]
 [4. 5. 0.]
 [7. 8. 6.]]
Move : down
Result :
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 0.]]


...Program finished with exit code 0
Press ENTER to exit console.
```

3. Implement Iterative deepening search algorithm.

```
from collections import defaultdict

class Graph:
    def __init__(self,vertices):
        self.noOfVertex = vertices
        self.graph = defaultdict(list)

    def addEdge(self,start,end):
        self.graph[start].append(end)

    def search(self,src,target,maxDepth):
        if src == target:
            return True

        if maxDepth<=0:
            return False

        for i in self.graph[src]:
            if self.search(i,target,maxDepth-1):
                return True

        return False

    def iterativeSearch(self,src,target,maxDepth):
        for i in range(maxDepth):
            if self.search(src,target,i):
                return True

        return False

if __name__ == "__main__":
    print("Enter number of vertex: ",end="")
    n = int(input())

    g = Graph(n)

    print("Enter no of edges: ",end="")
    noOfEdge = int(input())

    print("Enter edges: ")

    for i in range(noOfEdge):
        edge = list(map(int, input().split(" ")))
        g.addEdge(edge[0],edge[1])

    print("Enter src vertex: ",end="")
    src = int(input())

    print("Enter target vertex: ",end="")
    target = int(input())

    print("Enter maxDepth: ",end="")
    maxDepth = int(input())

    if g.iterativeSearch(src,target,maxDepth):
        print("Target is reachable from source within max depth")
    else:
        print("Target is NOT reachable from source within max depth")
```

OUTPUT :



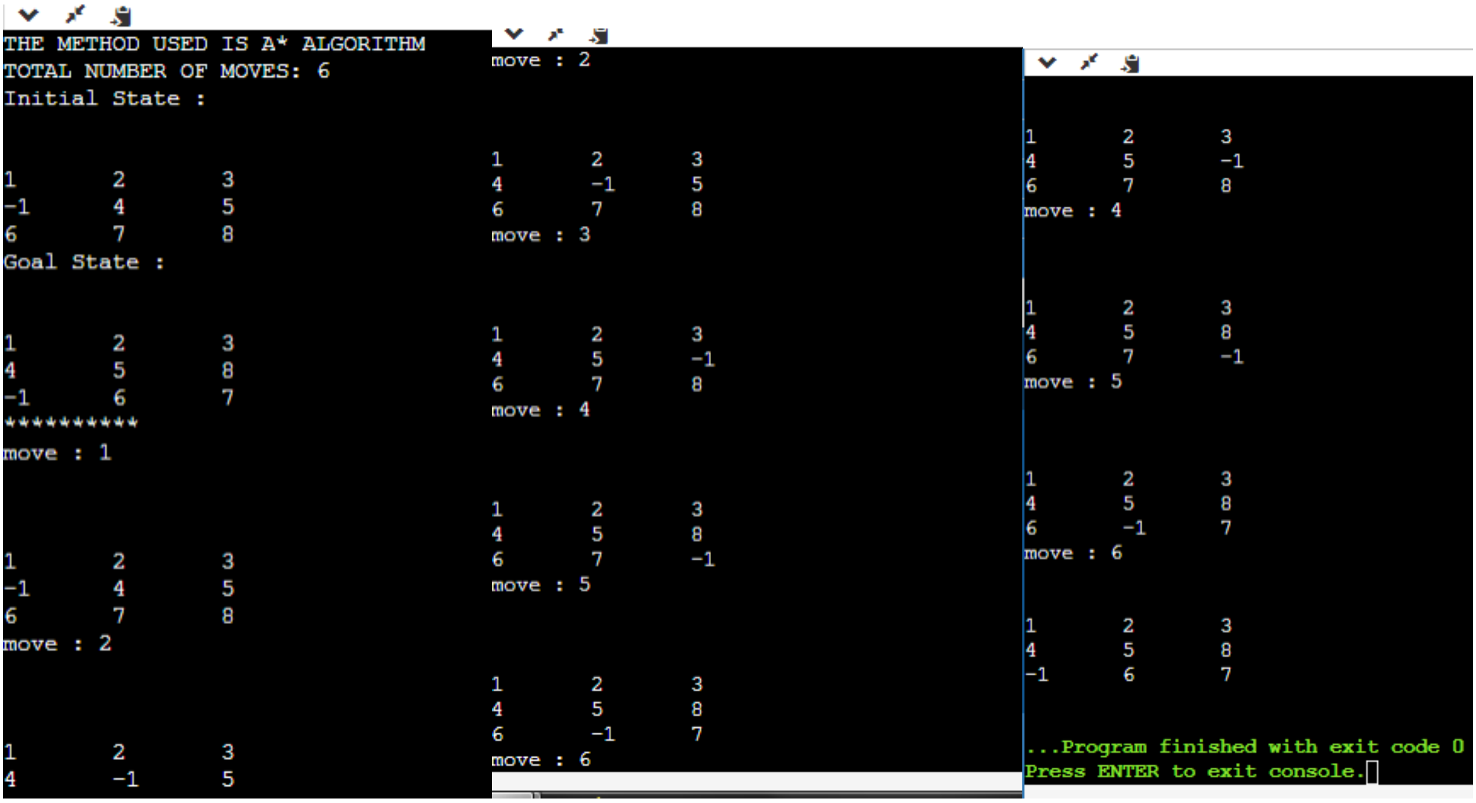
```
Enter number of vertex: 7
Enter no of edges: 6
Enter edges:
0 1
0 2
1 3
1 4
2 5
2 6
Enter src vertex: 0
Enter target vertex: 6
Enter maxDepth: 3
Target is reachable from source within max depth


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Implement A* search algorithm.

```
src=[1,2,3,-1,4,5,6,7,8]
target=[1,2,3,4,5,8,-1,6,7]
print("THE METHOD USED IS A* ALGORITHM")

def h(state):
    res=0
    for i in range(1,9):
        if state.index(i)!=target.index(i): res+=1
    return res

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

def possible_moves(state,visited_states):
    b=state.index(-1)
    d=[]
    pos_moves=[]
    if b<=5: d.append('d')
    if b>=3 : d.append('u')
    if b%3 > 0: d.append('l')
    if b%3 < 2: d.append('r')
    for i in d:
        temp=gen(state,i,b)
        if not temp in visited_states: pos_moves.append(temp)
    return pos_moves


def search(src,target,visited_states,g):
    if src==target: return visited_states
    visited_states.append(src),
    adj=possible_moves(src,visited_states)
    scores=[]
    selected_moves=[]
    for move in adj: scores.append(h(move)+g)
    min_score=min(scores)
    for i in range(len(adj)):
        if scores[i]==min_score: selected_moves.append(adj[i])
    for move in selected_moves:
        if search(move,target,visited_states,g+1):return visited_states
    return 0

def solve(src,target):
    visited_states=[]
    res=search(src,target,visited_states,0)


    if type(res)!=type(int()):
        i=0
        for state in res:
            print('move :',i+1,end="\n")
            print()
            display(state)
```

```
                i+=1
            print('move :',i+1)
            display(target)
print("TOTAL NUMBER OF MOVES:",6)

def display(state):
    print()
    for i in range(9):
        if i%3==0:print()
        if state[i]==-1: print(state[i],end="\t")
        else: print(state[i],end="\t")
    print(end="\n")

print('Initial State :')
display(src)
print('Goal State :')
display(target)
print('*'*10)
solve(src,target)
```

OUTPUT :



```
THE METHOD USED IS A* ALGORITHM
TOTAL NUMBER OF MOVES: 6
Initial State :


1       2       3
-1      4       5
6       7       8
Goal State :


1       2       3
4       5       8
-1      6       7
**********
move : 1


1       2       3
-1      4       5
6       7       8
move : 2


1       2       3
4       -1      5
```

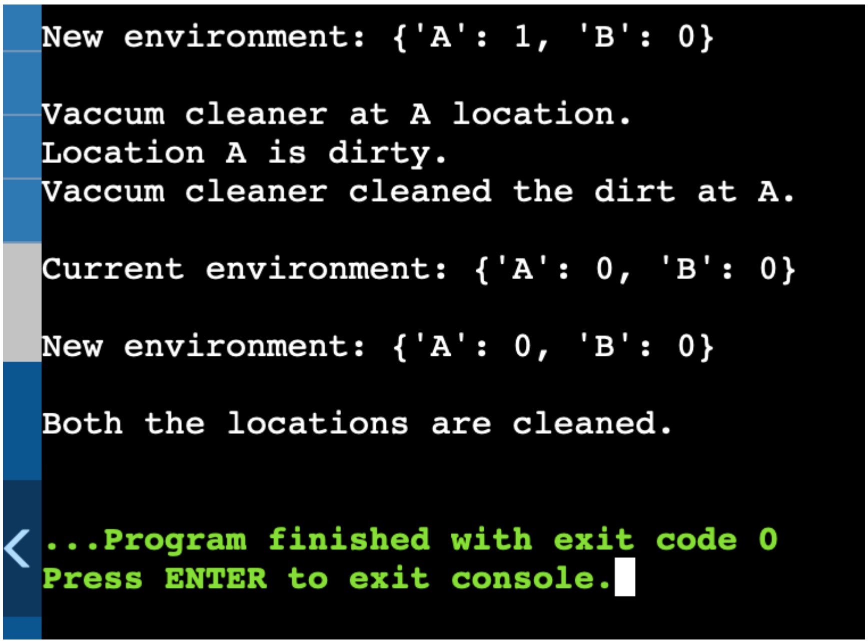
```
move : 2


1       2       3
4       -1      5
6       7       8
move : 3


1       2       3
4       5       -1
6       7       8
move : 4


1       2       3
4       5       8
6       7       -1
move : 5


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

```


1       2       3
4       5       -1
6       7       8
move : 4


1       2       3
4       5       8
6       7       -1
move : 5


1       2       3
4       5       8
6       -1      7
move : 6


1       2       3
4       5       8
-1      6       7


...Program finished with exit code 0
Press ENTER to exit console.
```

5. Implement vacuum cleaner agent.

```
import random

environment = {'A': 0, 'B': 0} # assumed initial state

def checkDirt():
    return random.randint(0,1)

def setEnvironment():
    environment['A'] = checkDirt()
    environment['B'] = checkDirt()

    print("\nNew environment: ",end="")
    print(environment)

def cleaned():
    print("\nBoth the locations are cleaned.")
    exit(0)

def newState():
    setEnvironment()

    if(environment['A'] == 0 and environment['B'] == 0):
        cleaned()
    else:
        if(environment['A']):
            cleanAt(1)
        else:
            cleanAt(0)

def cleanAt(state):
    if state == 1:
        print("\nVaccum cleaner at A location.")
        dirt = environment['A'] # 0-nodirt 1-dirt

        if dirt == 1:
            print("Location A is dirty.")
            print("Vaccum cleaner cleaned the dirt at A.")
            environment['A'] = 0
        else:
            print("Location A is clean.")

        if environment['B']:
            print("Vaccum cleaner moving to B location.")
            cleanAt(0)

    else:
        print("\nVaccum cleaner at B location.")
        dirt = environment['B'] # 0-nodirt 1-dirt

        if dirt == 1:
            print("Location B is dirty.")
            print("Vaccum cleaner cleaned the dirt at B.")
            environment['B'] = 0
        else:
            print("Location B is clean.")

        if environment['A']:
            print("Vaccum cleaner moving to A location.")
            cleanAt(1)

    print("\nCurrent environment: ",end="")
    print(environment)
    newState()

def start():
    newState()

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

OUTPUT :



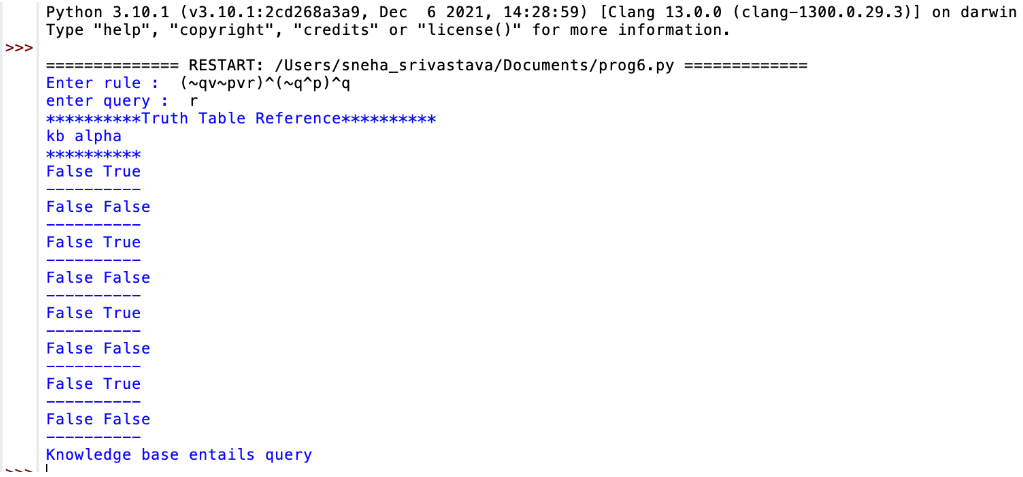
```
New environment: {'A': 1, 'B': 0}

Vaccum cleaner at A location.
Location A is dirty.
Vaccum cleaner cleaned the dirt at A.

Current environment: {'A': 0, 'B': 0}

New environment: {'A': 0, 'B': 0}

Both the locations are cleaned.


...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
combination = [(True,True,True),(True,True,False),(True,False,True),(True,False,False),
(False,True,True),(False,True,False),(False,False,True),(False,False,False)]
variable = {'p':0,'q':1,'r':2}

kb = ''
q = ''

priority = {'~':3,'v':1,'^':2}

def input_rules():
    global kb,q
    kb = (input("Enter rule :  "))
    q = (input("enter query :  "))

def _eval(i,val1,val2):
    if i=='^':
        return val2 and val1
    return val2 or val1


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

    return stack.pop()

def toPostfix(infix):
    stack=[]
    postfix = ''
    for c in infix:
        if isOperand(c):
            postfix += c
        else:
            if isLeftParanthesis(c):
                stack.append(c)
            elif 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 entailment():
    global kb,q
    print('*'*10 + "Truth Table Reference" + '*'*10)
    print('kb','alpha')
    print('*'*10)
    for comb in combination:
        s = evaluatePostfix(toPostfix(kb),comb)
        f = evaluatePostfix(toPostfix(q),comb)
        print(s,f)
        print('-'*10)
        if s and not f:
            return False
    return True

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

input_rules()
ans = entailment()
if ans:
    print("Knowledge base entails query")
else:
    print("Knowledge base does not entail query")
```

OUTPUT :

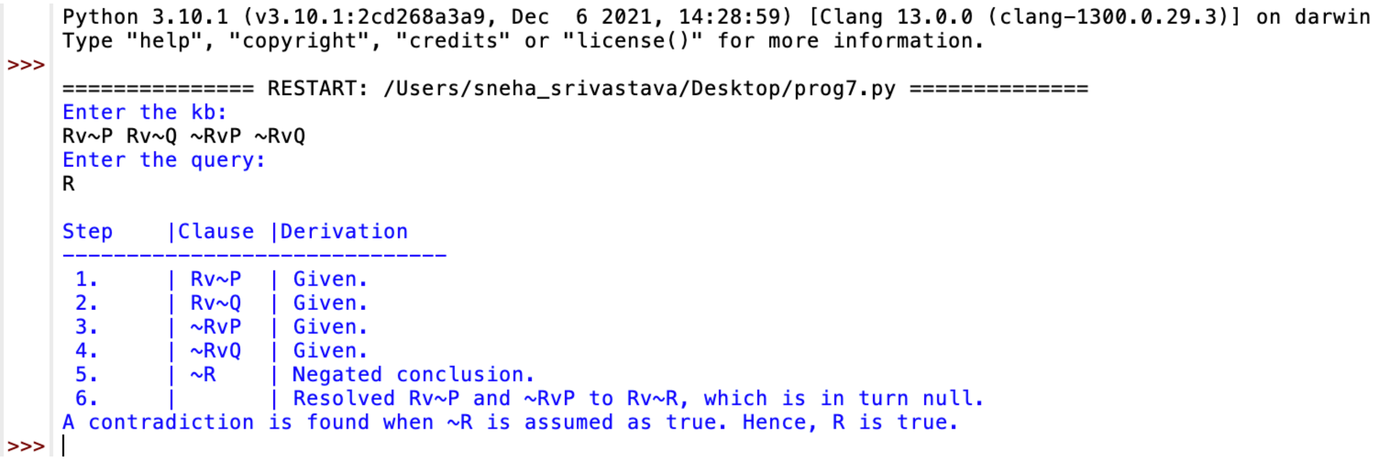


```
    Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: /Users/sneha_srivastava/Documents/prog6.py =============
    Enter rule :  (~qv~pvr)^(~q^p)^q
    enter query :  r
    **********Truth Table Reference**********
    kb alpha
    **********
    False True
    ----------
    False False
    ----------
    False True
    ----------
    False False
    ----------
    False True
    ----------
    False False
    ----------
    False True
    ----------
    False False
    ----------
    Knowledge base entails query
>>>
```

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

```
import re
def negate(term):
    return f'~{term}' if term[0] != '~' else term[1]

def reverse(clause):
    if len(clause) > 2:
        t = split_terms(clause)
        return f'{t[1]}v{t[0]}'
    return ''
def split_terms(rule):
    exp = '(~*[PQRS])'
    terms = re.findall(exp, rule)
    return terms
def contradiction(query, clause):
    contradictions = [ f'{query}v{negate(query)}', f'{negate(query)}v{query}']
    return clause in contradictions or reverse(clause) in contradictions
def resolve(kb, query):
    temp = kb.copy()
    temp += [negate(query)]
    steps = dict()
    for rule in temp:
        steps[rule] = 'Given.'
    steps[negate(query)] = 'Negated conclusion.'
    i = 0
    while i < len(temp):
        n = len(temp)
        j = (i + 1) % n
        clauses = []
        while j != i:
            terms1 = split_terms(temp[i])
            terms2 = split_terms(temp[j])
            for c in terms1:
                if negate(c) in terms2:
                    t1 = [t for t in terms1 if t != c]
                    t2 = [t for t in terms2 if t != negate(c)]
                    gen = t1 + t2
                    if len(gen) == 2:
                        if gen[0] != negate(gen[1]):
                            clauses += [f'{gen[0]}v{gen[1]}']
                        else:
                            if contradiction(query,f'{gen[0]}v{gen[1]}'):
                                temp.append(f'{gen[0]}v{gen[1]}')
                                steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn null.
\
                                  \nA contradiction is found when {negate(query)} is assumed as true. Hence,
{query} is true."
                                return steps
                    elif len(gen) == 1:
                        clauses += [f'{gen[0]}']
                    else:
                        if contradiction(query,f'{terms1[0]}v{terms2[0]}'):
                            temp.append(f'{terms1[0]}v{terms2[0]}')
                            steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn null. \
                                 \nA contradiction is found when {negate(query)} is assumed as true. Hence,
{query} is true."
                            return steps
            for clause in clauses:
                if clause not in temp and clause != reverse(clause) and reverse(clause) not in temp:
                    temp.append(clause)
                    steps[clause] = f'Resolved from {temp[i]} and {temp[j]}.'
            j = (j + 1) % n
        i += 1
    return steps
def resolution(kb, query):
    kb = kb.split(' ')
    steps = resolve(kb, query)
    print('\nStep\t|Clause\t|Derivation\t')
    print('-' * 30)
    i = 1
    for step in steps:
        print(f' {i}.\t| {step}\t| {steps[step]}\t')
        i += 1
def main():
    print("Enter the kb:")
    kb = input()
    print("Enter the query:")
    query = input()
    resolution(kb,query)
main()
```

OUTPUT :



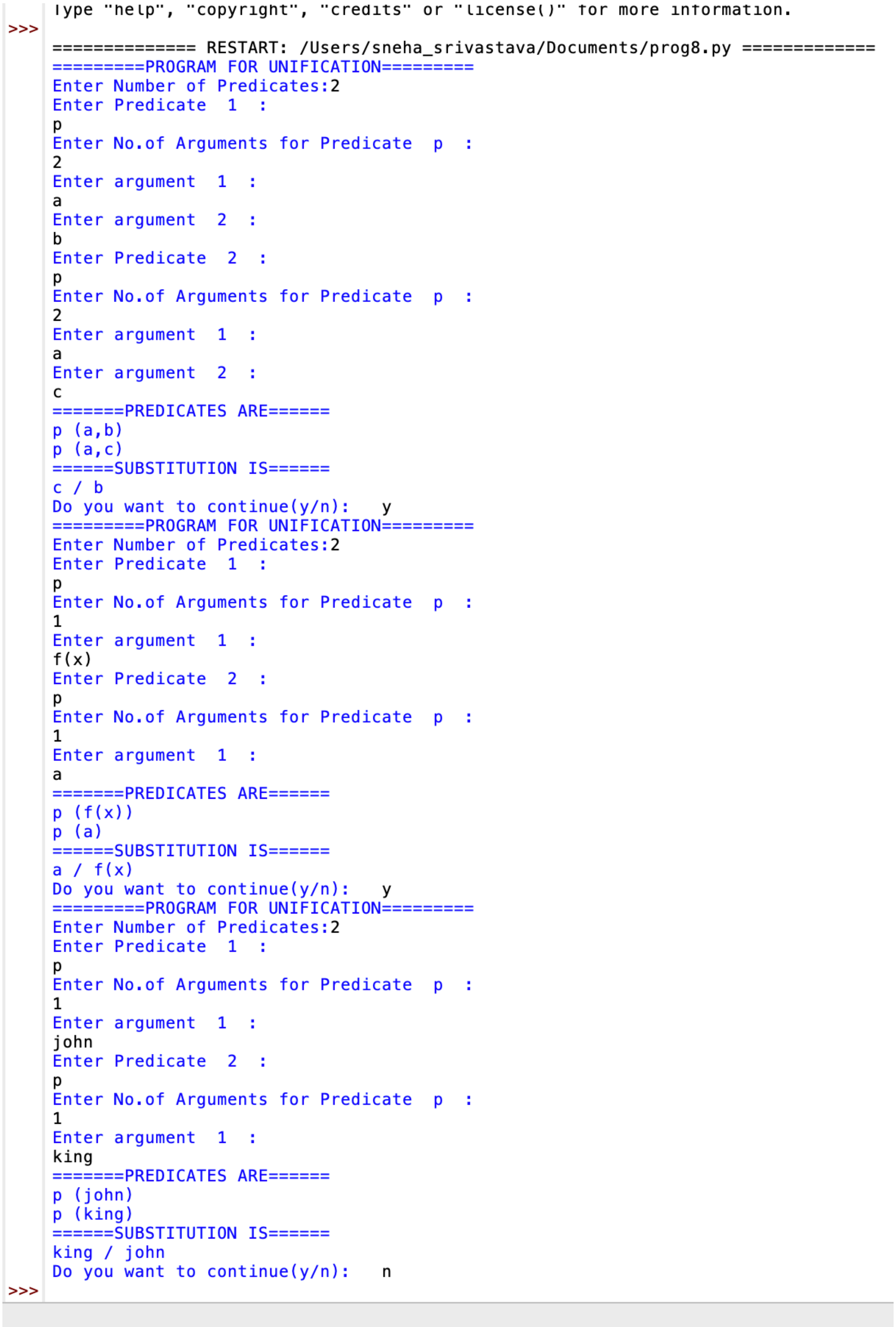
```
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: /Users/sneha_srivastava/Desktop/prog7.py ==============
Enter the kb:
Rv~P Rv~Q ~RvP ~RvQ
Enter the query:
R

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.
>>> |
```

8. Implement unification in first order logic.

```
no_of_pred = 0
no_of_arg = [None for i in range(10)]

predicate = [None for i in range(10)]
argument = [[None for i in range(10)] for i in range(10)]


def main():
    global no_of_pred
    ch = 'y'
    while(ch == 'y'):
        print("=========PROGRAM FOR UNIFICATION=========")
        no_of_pred = int(input("Enter Number of Predicates:"))
        for i in range(no_of_pred):
            # nouse=input() #   //to accept "enter" as a character
            print("Enter Predicate ", (i+1), " :")
            predicate[i] = input()
            print("Enter No.of Arguments for Predicate ", predicate[i], " :")
            no_of_arg[i] = int(input())

            for j in range(no_of_arg[i]):
                print("Enter argument ", j+1, " :")
                argument[i][j] = input()

        display()
        chk_arg_pred()
        ch = input("Do you want to continue(y/n):   ")


def display():

    print("=======PREDICATES ARE======")
    for i in range(no_of_pred):
        print(predicate[i], "(", end="")
        for j in range(no_of_arg[i]):
            print(argument[i][j], end="")
            if(j != no_of_arg[i]-1):
                print(",", end="")
        print(")")


# /*==========UNIFY FUNCTION=========*/

def unify():
    flag = 0
    for i in range(no_of_pred-1):
        for j in range(no_of_arg[i]):
            if(argument[i][j] != argument[i+1][j]):
                if(flag == 0):
                    print("======SUBSTITUTION IS======")
                    print(argument[i+1][j], "/", argument[i][j])
                    flag += 1

        if(flag == 0):
            print("Arguments are Identical...")
            print("No need of Substitution")
```

```
def chk_arg_pred():
    pred_flag = 0
    arg_flag = 0

   # /*======Checking Prediactes========*/
    for i in range(no_of_pred-1):
        if(predicate[i] != predicate[i+1]):
            print("Predicates not same..")
            print("Unification cannot progress!")
            pred_flag = 1
            break

   # /*=====Checking No of Arguments====*/

    if(pred_flag != 1):
        ind = 0
        key = no_of_arg[ind]
        l = len(no_of_arg)
        for i in range(0, key-1):
            if i >= key:
                continue
            if ind != l-1:
                ind += 1
                key = no_of_arg[ind]
            if(no_of_arg[i] != no_of_arg[i+1]):

                print("Arguments Not Same..!")
                arg_flag = 1
                break

        if(arg_flag == 0 and pred_flag != 1):
            unify()


main()
```

OUTPUT :

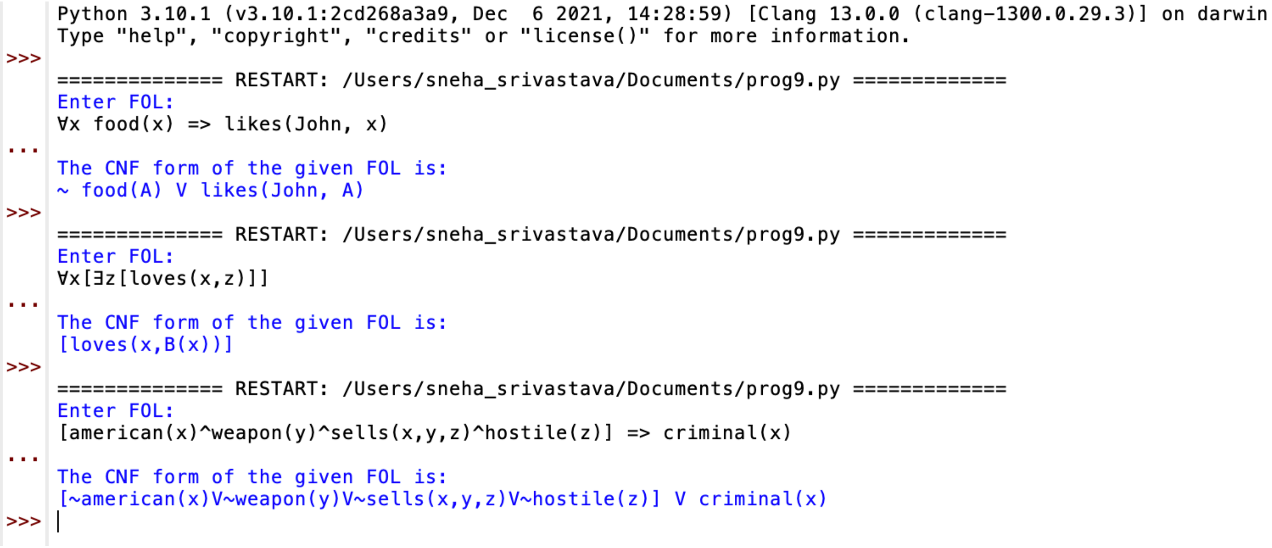


```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/sneha_srivastava/Documents/prog8.py =============
=========PROGRAM FOR UNIFICATION=========
Enter Number of Predicates:2
Enter Predicate  1  :
p
Enter No.of Arguments for Predicate  p  :
2
Enter argument  1  :
a
Enter argument  2  :
b
Enter Predicate  2  :
p
Enter No.of Arguments for Predicate  p  :
2
Enter argument  1  :
a
Enter argument  2  :
c
=======PREDICATES ARE======
p (a,b)
p (a,c)
======SUBSTITUTION IS======
c / b
Do you want to continue(y/n):   y
=========PROGRAM FOR UNIFICATION=========
Enter Number of Predicates:2
Enter Predicate  1  :
p
Enter No.of Arguments for Predicate  p  :
1
Enter argument  1  :
f(x)
Enter Predicate  2  :
p
Enter No.of Arguments for Predicate  p  :
1
Enter argument  1  :
a
=======PREDICATES ARE======
p (f(x))
p (a)
======SUBSTITUTION IS======
a / f(x)
Do you want to continue(y/n):   y
=========PROGRAM FOR UNIFICATION=========
Enter Number of Predicates:2
Enter Predicate  1  :
p
Enter No.of Arguments for Predicate  p  :
1
Enter argument  1  :
john
Enter Predicate  2  :
p
Enter No.of Arguments for Predicate  p  :
1
Enter argument  1  :
king
=======PREDICATES ARE======
p (john)
p (king)
======SUBSTITUTION IS======
king / john
Do you want to continue(y/n):   n
>>>
```

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

```
import re

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~]+\([A-Za-z,]+\)'
    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.replace(predicate, f'~{predicate}')
    s = list(string)
    for i, c in enumerate(string):
        if c == 'V':
            s[i] = '^'
        elif c == '^':
            s[i] = 'V'
    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(match, '')
        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(0))
            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
def fol_to_cnf(fol):

    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
          new_statement = '[' + statement[:i] + '=>' + statement[i+1:] + ']^['+ statement[i+1:] + '=>' +
statement[:i] + ']'
        statement = new_statement
    statement = statement.replace("=>", "-")
    expr = '\[([^]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    while '-' in statement:
        i = statement.index('-')
        br = statement.index('[') if '[' in statement else 0
        new_statement = '~' + statement[br:i] + 'V' + statement[i+1:]
        statement = statement[:br] + new_statement if br > 0 else new_statement
    while '~∀' in statement:
        i = statement.index('~∀')
        statement = list(statement)
        statement[i], statement[i+1], statement[i+2] = '∃', statement[i+2], '~'
        statement = ''.join(statement)
    while '~∃' in statement:
        i = statement.index('~∃')
        s = list(statement)
        s[i], s[i+1], s[i+2] = '∀', s[i+2], '~'
        statement = ''.join(s)
    statement = statement.replace('~[∀','[~∀')
    statement = statement.replace('~[∃','[~∃')
    expr = '(~[∀V∃].)'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
    expr = '~\[[^]]+\]'
    statements = re.findall(expr, statement)
    for s in statements:
        statement = statement.replace(s, DeMorgan(s))
    return statement
def main():
    print("Enter FOL:")
    fol = input()
    print("The CNF form of the given FOL is: ")
    print(Skolemization(fol_to_cnf(fol)))
main()
```

OUTPUT :



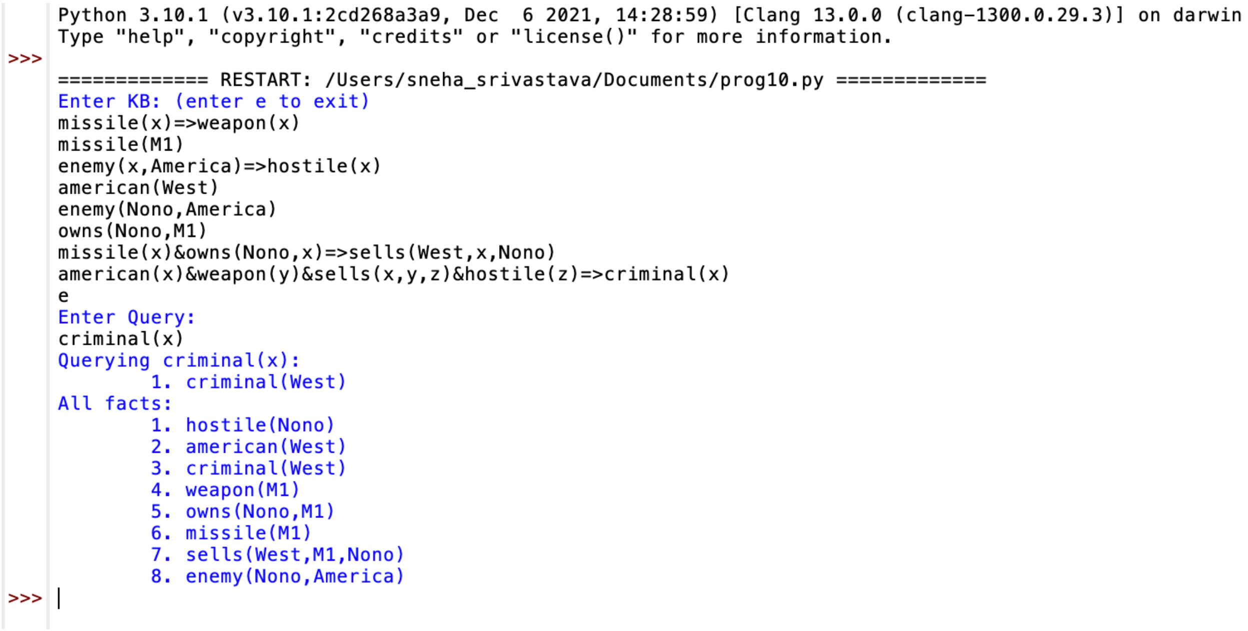
```
    Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: /Users/sneha_srivastava/Documents/prog9.py =============
    Enter FOL:
    ∀x food(x) => likes(John, x)
...
    The CNF form of the given FOL is:
    ~ food(A) V likes(John, A)
>>>
    ============== RESTART: /Users/sneha_srivastava/Documents/prog9.py =============
    Enter FOL:
    ∀x[∃z[loves(x,z)]]
...
    The CNF form of the given FOL is:
    [loves(x,B(x))]
>>>
    ============== RESTART: /Users/sneha_srivastava/Documents/prog9.py =============
    Enter FOL:
    [american(x)^weapon(y)^sells(x,y,z)^hostile(z)] => criminal(x)
...
    The CNF form of the given FOL is:
    [~american(x)V~weapon(y)V~sells(x,y,z)V~hostile(z)] V criminal(x)
>>> |
```

10. Create a knowledge base consisting of first order logic statements and prove the given query using 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-z~]+)\([^&|]+\)'
   return re.findall(expr, string)
class Fact:
   def __init__(self, expression):
      self.expression = expression
      predicate, params = self.splitExpression(expression)
      self.predicate = predicate
      self.params = params
      self.result = any(self.getConstants())

   def splitExpression(self, expression):
      predicate = getPredicates(expression)[0]
      params = getAttributes(expression)[0].strip('()').split(',')
      return [predicate, params]

   def getResult(self):
      return self.result

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

   def getVariables(self):
      return [v if isVariable(v) else None for v 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 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.expression)[0], str(getAttributes(self.rhs.expression)[0])
        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.expression for f in self.facts])):
            print(f'\t{i+1}. {f}')
def main():
    kb = KB()
    print("Enter KB: (enter e to exit)")
    while True:
        t = input()
        if(t == 'e'):
            break
        kb.tell(t)
    print("Enter Query:")
    q = input()
    kb.query(q)
    kb.display()
main()
```

OUTPUT :



```
    Python 3.10.1 (v3.10.1:2cd268a3a9, Dec  6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
    Type "help", "copyright", "credits" or "license()" for more information.
>>> 
    ============= RESTART: /Users/sneha_srivastava/Documents/prog10.py =============
    Enter KB: (enter e to exit)
    missile(x)=>weapon(x)
    missile(M1)
    enemy(x,America)=>hostile(x)
    american(West)
    enemy(Nono,America)
    owns(Nono,M1)
    missile(x)&owns(Nono,x)=>sells(West,x,Nono)
    american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)
    e
    Enter Query:
    criminal(x)
    Querying criminal(x):
            1. criminal(West)
    All facts:
            1. hostile(Nono)
            2. american(West)
            3. criminal(West)
            4. weapon(M1)
            5. owns(Nono,M1)
            6. missile(M1)
            7. sells(West,M1,Nono)
            8. enemy(Nono,America)
>>> |
```