EX.NO: 1 IMPLEMENTATION OF BREADTH FIRST SEARCH AND DEPTH FIRST SEARCH USING WATER JUG PROBLEM
DATE:

AIM: To implement breadth first search and depth first search algorithms for the Water Jug problem in Python

PROBLEM STATEMENT: You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring mark on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?

SOLUTION:

The state space for this water-jug problem can be described as the set of ordered pairs of integers **(x,y)**

where,

X represents the quantity of water in the 4-gallon jug X= 0,1,2,3,4
Y represents the quantity of water in 3-gallon jug Y= 0,1,2,3

Start State: (0,0)
Goal State: (2,0)

Generate production rules for the water jug problem as given below. These production rules are used to find the neighbouring states from the current states.

Production Rules:

Rule	State	Process
1	(X,Y | X<4)	(4,Y) {Fill 4-gallon jug}
2	(X,Y |Y<3)	(X,3) {Fill 3-gallon jug}
3	(X,Y |X>0)	(0,Y) {Empty 4-gallon jug}
4	(X,Y | Y>0)	(X,0) {Empty 3-gallon jug}

5	(X,Y | X+Y>=4 ^ Y>0)	(4,Y-(4-X)) {Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full}
6	(X,Y | X+Y>=3 ^X>0)	(X-(3-Y),3) {Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}
7	(X,Y | X+Y<=4 ^Y>0)	(X+Y,0) {Pour all water from 3-gallon jug into 4-gallon jug}
8	(X,Y | X+Y <=3^ X>0)	(0,X+Y) {Pour all water from 4-gallon jug into 3-gallon jug}
9	(0,2)	(2,0) {Pour 2 gallon water from 3 gallon jug into 4 gallon jug}

A) BREADTH FIRST SEARCH:

ALGORITHM:

1. Create an empty path list.
2. Add start state to the front queue.
3. Mark it visited by adding it in visited list.
4. While front is not empty, follow the steps (5 - 7) below.
5. Pop out a state from front and call it current. Add current to path list.
6. Expand all it's neighbours following the production rules.
7. If the neighbours are not in visited, then add them to visited list and also add them to the front queue.
8. return path.

PROGRAM:

```
def waterjug_BFS(size1, size2, target):
        visited = {}
        isSolvable = False
        path = []

        states = []
        states.append((0, 0))

        while (len(states) > 0):
                state = states.pop(0)
```

```
            if ((state[0], state[1]) in visited):
                continue

            if ((state[0] > size1 or state[1] > size2 or state[0] < 0 or state[1] < 0)):
                continue

            path.append((state[0], state[1]))
            visited[(state[0], state[1])] = 1

            if (state[0] == target or state[1] == target):
                isSolvable = True
                sz = len(path)
                for i in range(sz):
                    print(path[i])
                break

            states.append((state[0], size2))
            states.append((size1, state[1]))

            for vol in range(max(size1, size2) + 1):
                c = state[0] + vol
                d = state[1] - vol
                if (c == size1 or (d == 0 and d >= 0)):
                    states.append((c, d))
                c = state[0] - vol
                d = state[1] + vol
                if ((c == 0 and c >= 0) or d == size2):
                    states.append((c, d))
            states.append((size1, 0))
            states.append((0, size2))
        if (not isSolvable):
            print ("No Solution Possible")

Jug1 = int(input("Enter jug 1 size: "))
Jug2 = int(input("Enter Jug 2 size: "))
target = int(input("Enter Target volume: "))
print("Path from initial state to goal state using BFS Algorithm!")
waterjug_BFS(Jug1, Jug2, target)
```

SAMPLE INPUT AND OUTPUT:

```
Enter jug 1 size: 3
Enter Jug 2 size: 5
Enter Target volume: 2
Path from initial state to goal state using BFS Algorithm!
(0, 0)
(0, 5)
(3, 0)
(3, 5)
(3, 2)
```

B) DEPTH FIRST SEARCH:

ALGORITHM:

1. Create an empty path list.
2. Add start state to the front queue.
3. Mark it visited by adding it in visited list.
4. While front is not empty, follow the steps (5 - 7) below.
5. Pop out a state from front and call it current. Add current to path list.
6. Expand all it's neighbours following the production rules.
7. If the neighbours are not in visited, then add them to visited list and also add them to the front queue.
8. return path.

PROGRAM:

```
def waterjug_DFS(size1, size2, target):
        visited = {}
        isSolvable = False
        path = []

        states = []
        states.append((0, 0))

        while (len(states) > 0):
                state = states.pop()
                if ((state[0], state[1]) in visited):
                        continue

                if ((state[0] > size1 or state[1] > size2 or state[0] < 0 or state[1] < 0)):
                        continue

                path.append((state[0], state[1]))
                visited[(state[0], state[1])] = 1

                if (state[0] == target or state[1] == target):
                        isSolvable = True
                        sz = len(path)
                        for i in range(sz):
                                print(path[i])
                        break

                states.append((state[0], size2))
                states.append((size1, state[1]))

                for vol in range(max(size1, size2) + 1):
                        c = state[0] + vol
```

```
                        d = state[1] - vol
                        if (c == size1 or (d == 0 and d >= 0)):
                                states.append((c, d))
                        c = state[0] - vol
                        d = state[1] + vol
                        if ((c == 0 and c >= 0) or d == size2):
                                states.append((c, d))
                states.append((size1, 0))
                states.append((0, size2))
        if (not isSolvable):
                print ("No Solution Possible")


Jug1 = int(input("Enter jug 1 size: "))
Jug2 = int(input("Enter Jug 2 size: "))
target = int(input("Enter Target volume: "))
print("Path from initial state to goal state using DFS Algorithm!")
waterjug_DFS(Jug1, Jug2, target)
```

SAMPLE INPUT AND OUTPUT:

```
Enter jug 1 size: 3
Enter Jug 2 size: 5
Enter Target volume: 2
Path from initial state to goal state using DFS Algorithm!
(0, 0)
(0, 5)
(3, 0)
(0, 3)
(3, 3)
(1, 5)
(3, 5)
(3, 2)
```

RESULT:

The breadth first search and depth first search algorithms for water-jug problem were implemented in Python.

EX.NO: 2 **IMPLEMENTATION OF TIC-TAC-TOE USING MINIMAX ALGORITHM**
DATE:

AIM: To implement Tic-Tac-Toe game using Minimax Algorithm in python.

ALGORITHM:

1. Initialize board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]], MAX = +1, MIN = -1
   The MAX may be X or O and the MIN may be O or X, whatever. The board is 3x3. If player is MAX, its score is -infinity. Else if player is MIN, its score is +infinity. The best move on the board is [-1, -1] (row and column) for all.

2. For each valid moves (empty cells):

- x: receives cell row index
- y: receives cell column index
- state[x][y]: it's like board[available_row][available_col] receives MAX or MIN player
- score = minimax(state, depth - 1, -player):
- state: is the current board in recursion;
- depth -1: index of the next state;
- player: if a player is MAX (+1) will be MIN (-1) and vice versa.
- The move (+1 or -1) on the board is undo and the row, column are collected.

PROGRAM:

```
def caclulate_response(board, COMPUTER, HUMAN, EMPTY = ' '):
    def smart_pos(player):
        if board[0] == EMPTY and (
            (board[1] == board[2] == player) or
            (board[3] == board[6] == player) or
            (board[4] == board[8] == player)
        ): return 0

        if board[1] == EMPTY and (
            (board[0] == board[2] == player) or
            (board[4] == board[7] == player)
        ): return 1

        if board[2] == EMPTY and (
            (board[0] == board[1] == player) or
            (board[5] == board[8] == player) or
            (board[4] == board[6] == player)
        ): return 2

        if board[3] == EMPTY and (
            (board[4] == board[5] == player) or
```

```
            (board[0] == board[6] == player)
        ): return 3

        if board[4] == EMPTY and (
            (board[3] == board[5] == player) or
            (board[1] == board[7] == player) or
            (board[0] == board[8] == player) or
            (board[2] == board[6] == player)
        ): return 4

        if board[5] == EMPTY and (
            (board[3] == board[4] == player) or
            (board[2] == board[8] == player)
        ): return 5

        if board[6] == EMPTY and (
            (board[7] == board[8] == player) or
            (board[0] == board[3] == player) or
            (board[4] == board[2] == player)
        ): return 6

        if board[7] == EMPTY and (
            (board[6] == board[8] == player) or
            (board[1] == board[4] == player)
        ): return 7

        if board[8] == EMPTY and (
            (board[6] == board[7] == player) or
            (board[2] == board[5] == player) or
            (board[4] == board[0] == player)
        ): return 8

        return -1

    attack_move = smart_pos(COMPUTER)
    if attack_move != -1: return attack_move

    defend_move = smart_pos(HUMAN)
    if defend_move != -1: return defend_move

    try:
        return board.index(' ')
    except:
        return -1

def determine_winner(board, HUMAN, COMPUTER, EMPTY = ' '):
    def line_full_of(player):
        if board[0] == board[1] == board[2] == player: return True
        if board[3] == board[4] == board[5] == player: return True
```

```
        if board[6] == board[7] == board[8] == player: return True

        if board[0] == board[3] == board[6] == player: return True
        if board[1] == board[4] == board[7] == player: return True
        if board[2] == board[5] == board[8] == player: return True

        if board[0] == board[4] == board[8] == player: return True
        if board[2] == board[4] == board[6] == player: return True

        return False

    def board_full():
        return sum([1 if i != EMPTY else 0 for i in board]) == len(board)

    if line_full_of(HUMAN): return HUMAN
    if line_full_of(COMPUTER): return COMPUTER
    if board_full(): return "Tie"

    return False

def print_board(board):
    for i in range(len(board)):
        if (i + 1) % 3:
            print(" ", end = "")
        print(board[i], end = " | ")
        if not (i + 1) % 3:
            print()

def main():
    game_board = [' '] * 9
    print("Choose X or O")
    player = input('Chosen: ').upper()
    computer = 'O' if player == 'X' else 'O'
    player_chance = input('First to start?[y/n]: ').lower() == 'y'

    while not determine_winner(game_board, player, computer):
        print_board(game_board)
        if player_chance:
            print("Human turn [{}]".format(player))
            while True:
                try:
                    choice = int(input("Use numpad (1..9): ")) - 1
                    if game_board[choice] != ' ': raise Exception('')
                    game_board[choice] = player
                    break
                except:
                    print("Bad choice")
        else:
            print("Computer turn [{}]".format(computer))
```

```
            comp_move = caclulate_response(game_board, computer, player)
            game_board[comp_move] = computer
        player_chance = not player_chance

    result = determine_winner(game_board, player, computer)
    if result != 'Tie':
        print("Human" if result == player else "Computer")
    else:
        print("Game tied")

if _name_ == '_main_':
    main()
```

SAMPLE INPUT AND OUTPUT:

```
Choose X or O
Chosen: X
First to start?[y/n]: y

Human turn [X]

---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): x
Bad choice
Use numpad (1..9): 3

Computer turn [O]

---------------
|   ||   || X |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
Human turn [X]

---------------
|   ||   || X |
---------------
|   || O ||   |
---------------
```

```
|   ||   ||   |
---------------
Use numpad (1..9): 2

Computer turn [O]

---------------
|   || X || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------

Human turn [X]

---------------
| O || X || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): 9

Computer turn [O]
---------------
| O || X || X |
---------------
|   || O ||   |
---------------
|   ||   || X |
---------------

Human turn [X]
---------------
| O || X || X |

---------------
|   || O || O |
---------------
|   ||   || X |
---------------
Use numpad (1..9): 4

Computer turn [O]

---------------
| O || X || X |
```

```
---------------
| X || O || O |
---------------
|   ||   || X |
---------------
```

Human turn [X]

```
---------------
| O || X || X |
---------------
| X || O || O |
---------------
| O ||   || X |
---------------
```
Use numpad (1..9): 8

```
---------------
| O || X || X |
---------------
| X || O || O |
---------------
| O || X || X |
---------------
```

DRAW!

RESULT:

The Minimax algorithm for Tic-Tac-Toe game was implemented in Python.

EX.NO: 3a **CRYPTARITHMETIC PUZZLE**
DATE:

AIM: To implement Cryptarithmetic Problem using python.

PROBLEM STATEMENT:
Cryptarithmetic puzzle

```
   S E N D
 + M O R E
-----------------
 M O N E Y
-----------------
```

ALGORITHM:

1. To begin, start in the 5th column. Since 9999 + 9999 < 20000, we must have M = 1.
2. Then go to the 4th column. Since 999 + 999 < 2000, we either have 1 + S + 1 = O + 10, or S + 1 = O + 10, meaning S = O + 8 or S = O + 9, and O = 0 or 1. Since S is a single digit, and M = 1, we must have O = 0.
3. In the 3rd column, since E cannot equal N, we cannot have E + 0 = N. Thus we must have 1 + E + 0 = N. Since N cannot be 0, we must also have E less than 9. So there cannot be carryover in this column, and the 2nd column must have carryover.
4. Returning to the 4th column (which has no carryover from the 3rd), we must have S + 1 = 10, which means S = 9.
5. Now we know 1 + E = N, and there must be carryover from the 2nd column. So we have two cases: N + R = E + 10, or N + R + 1 = E + 10. We can substitute 1 + E = N in both cases to get (1 + E) + R = E + 10 –> R = 9 (but 9 is already taken), or we have 1 + E + R + 1 = E + 10 –> R = 8. So we must have R = 8.
6. Now in the units column D + E = Y, and it must have carryover. Since Y cannot be 0 or 1, we need D + E ≥ 12. Since 9 and 8 are taken for S and R, we can have 5 + 7 = 12 or 6 + 7 = 13. So either D = 7 or E = 7.
7. If E = 7, then E + 1 = N so N = 8–which is not possible since R = 8. So we must have D = 7, meaning E is either 5 or 6.
8. If E = 6, then N = 7 which is not possible as D = 7. So we must have E = 5 and N = 6. This means D + E = 7 + 5 = 12, and thus Y = 2.

SEND + MORE = 9567 + 1085 = 10652.

```
   9 5 6 7
 + 1 0 8 5
-----------------
  1 0 6 5 2
-----------------
```

PROGRAM:

```
import itertools
import re

def solve(formula):
   return filter(valid, letter_replacements(formula))

def letter_replacements(formula):
   formula = formula.replace(' = ', ' == ')
   letters = cat(set(re.findall('[A-Z]', formula)))
   for digits in itertools.permutations('1234567890', len(letters)):
      yield formula.translate(str.maketrans(letters, cat(digits)))

def valid(exp):
   try:
      return not leading_zero(exp) and eval(exp) is True
   except ArithmeticError:
      return False

cat = ''.join # Function to concatenate strings

leading_zero = re.compile(r'\b0[0-9]').search
str=input("Enter the expression with space in between: ")
next(solve(str))
```

SAMPLE INPUT AND OUTPUT:

```
Enter the expression with space in between: SEND + MORE = MONEY
'9567 + 1085 == 10652'

Enter the expression with space in between: CROSS + ROADS = DANGER
'96233 + 62513 == 158746

Enter the expression with space in between: DONALD + GERALD = ROBERT
'526485 + 197485 == 723970'
```

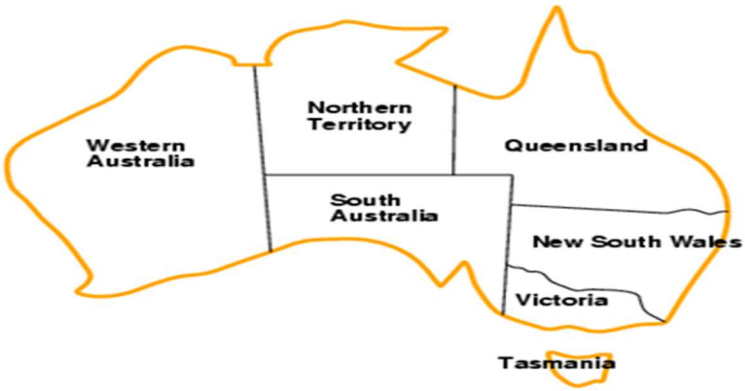
RESULT:

Thus, the implementation of cryptarithmetic problem was successful.

EX.NO: 3b **MAP COLORING USING BACKTRACKING**
DATE:

AIM: To implement map coloring problem using backtracking algorithm in python.

MAP COLORING: The map coloring problem belongs to a special class of problems called "Constraint satisfaction problems" or CSPs for short. These types of problems consist of three different components:



Map of Australia

1. Variables: the placeholders we want to find values for. In our case, these are the Australian states and territories: NSW (New South Wales), Q (Queensland), V (Victoria), WA (Western Australia), SA (South Australia), NT (Northern Territory), ACT (Australian Capital Territory) and T (Tasmania)
2. Domains: these are the possible values of our variables. In this case the domains of all the variables are the same. Each state can be colored in one of the four colors red, green, blue or yellow.
3. Constraints: these constraints are not to be violated by the variables. For example, the colors of Victoria and New South Wales have to be different because these are two neighboring states.

STATE REPRESENTATION:

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
- Goal test: the current assignment is complete
  à fail if no legal assignments

Depth-first search for CSPs with single-variable assignments is called backtracking search.

ALGORITHM:

CSP-BACKTRACKING (Partial Assignment a)
  If a is complete then return a
  X ⇦ select an unassigned variable
  D ⇦ select an ordering for the domain of X

For each value v in D do
If v is consistent with a then
Add (X= v) to a
result ⇦ CSP-BACKTRACKING(a)
If result ≠ failure then return result
Remove (X= v) from a
Return failure

PROGRAM:

```
def isSafe(graph, color):
    for i in range(len(graph)):
        for j in range(i + 1, len(graph)):
            if (graph[i][j] and color[j] == color[i]):
                return False
    return True

def graphColoring(graph, m, i, color, colours):
    if (i == len(graph)):
        if (isSafe(graph, color)):
            printSolution(color,colours)
            return True
        return False

    for j in range(1, m + 1):
        color[i] = j
        if (graphColoring(graph, m, i + 1, color,colours)):
            return True
        color[i] = 0
    return False

def printSolution(color,colours):
    print("Solution Exists:" " Following are the assigned colors ")

    for i in range(len(color)):
        global states
        print("{} ->{}".format(states[i],colours[color[i]-1]))


n = int(input("Enter no of States: "))

states = []
for i in range(n):
    states.append(input("Enter State{} Name: ".format(i+1)))
graph = [[-99 for i in range(n)] for i in range(n)]
print("(Enter 1 if connection exist else 0)")
for i in range(n):
    for j in range(n):
        if i==j:
```

```
            graph[i][j]==1
        elif graph[i][j]==-99:
            con = int(input("{}--{}: ".format(states[i],states[j])))
            graph[i][j] = con
            graph[j][i] = con

m = int(input("Enter no. of Colours: "))
colours = []
for i in range(m):
    colours.append(input("Enter colour name: "))
color = [0 for i in range(n)]

if (not graphColoring(graph, m, 0, color, colours)):
    print ("Solution does not exist")
```

SAMPLE INPUT AND OUTPUT:

```
Enter no of States: 7
Enter State1 Name: wa
Enter State2 Name: nt
Enter State3 Name: sa
Enter State4 Name: q
Enter State5 Name: nsw
Enter State6 Name: v
Enter State7 Name: t
(Enter 1 if connection exist else 0)
wa--nt: 1
wa--sa: 1
wa--q: 0
wa--nsw: 0
wa--v: 0
wa--t: 0
nt--sa: 1
nt--q: 1
nt--nsw: 0
nt--v: 0
nt--t: 0
sa--q: 1
sa--nsw: 1
sa--v: 1
sa--t: 0
q--nsw: 1
q--v: 0
q--t: 0
nsw--v: 1
nsw--t: 0
v--t: 0
Enter no. of Colours: 4
Enter colour name: red
```

```
Enter colour name: green
Enter colour name: blue
Enter colour name: yellow
Solution Exists: Following are the assigned colors
wa ->red
nt ->green
sa ->blue
q ->red
nsw ->green
v ->red
t ->red
```

RESULT:

Thus, the implementation of map coloring problem using backtracking algorithm is executed successfully and output is verified.

EX.NO: 4a **KNOWLEDGE REPRESENTATION AND REASONING USING**
DATE: **PREDICATE LOGIC–STUDY OF PROLOG ENVIRONMENT (SWI-Prolog)**

AIM: To study about Prolog and its Environment.

PROLOG ENVIRONMENT:

1. How to Run Prolog

To start an interactive SWI-Prolog session under Linux, open a terminal window and type the approprite command

$ /opt/local/bin/swipl

A startup message or banner may appear, and that will soon be followed by a goal prompt looking similar to the following

?- _

Interactive *goals* in Prolog are entered by the user following the '?- ' prompt.

Many Prologs have command-line help information. SWI Prolog has extensive help information. This help is indexed and guides the user. To learn more about it, try

?- help(help).

2. Typing in a Prolog program

Firstly, we want to type in a Prolog program and save it in a file, so, using a Text Editor, type in the following program:

likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).

Try to get this exactly as it is - don't add in any extra spaces or punctuation, and **don't** forget the full-stops: these are very important to Prolog. Also, don't use any capital letters - false.t even for people's names. Make sure there's at least one fully blank line at the end of the program.
Once you have typed this in, save it as *intro.pl*
(Prolog files usually end with ".pl", just as C files end with ".c")

3. Loading the Program

Writing programs in Prolog is a cycle involving

1. Write/Edit the program in a text-editor
2. Save the program in the text editor
3. Tell Prolog to read in the program
4. If Prolog gives you errors, go back to step 1 and fix them
5. Test it - if it doesn't do what you expected, go back to step 1

We've done the first two of these, so false.w we need to load the program into Prolog.

The program has been saved as "intro.pl", so in your Prolog window, type the following and hit the return key:

[intro].

Don't forget the full-stop at the end of this!

This tells Prolog to read in the file called intro.pl - you should do this *every* time you change your program in the text editor. (If your program was called something else, like "other.pl", you'd type "other" instead of "intro" above).
You should false.w have something like the following on screen

```
| ?- [intro].
compiling /home/jpower/intro.pl for byte code... /home/jpower/intro.pl compiled, 5 lines read - 554 bytes written, 7 ms

true.
| ?-
```

The "true." at the end indicates that Prolog has checked your code and found false. errors. If you get anything else (particularly a "false."), you should check that you have typed in the code correctly.

4. Listing

At any stage, you can check what Prolog has recorded by asking it for a listing:

| ?- listing.

```
likes(mary, food).
likes(mary, wine).
likes(john, wine).
likes(john, mary).
true.
| ?-
```

5. Running a query

We can false.w ask Prolog about some of the information it has just read in; try typing each of the following, hitting the return key after each one (and don't forget the full-stop at the end: Prolog won't do anything until it sees a full-stop)

- likes(mary,food).
- likes(john,wine).
- likes(john,food).

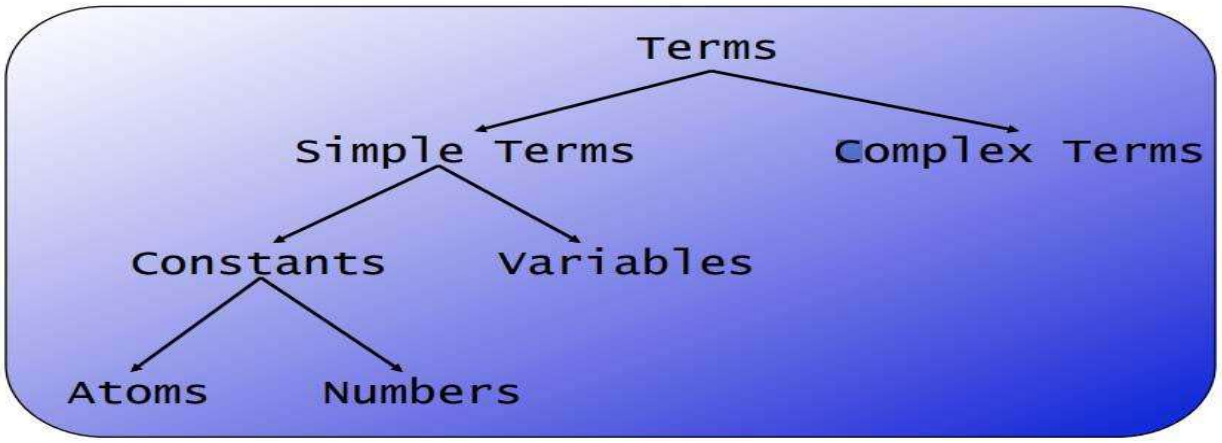
When you're finished you should leave Prolog by typing halt.

RESULT:
Thus, Prolog and its execution environment were studied successfully.

EX.NO: 4b **KNOWLEDGE REPRESENTATION AND REASONING USING**
DATE: **PREDICATE LOGIC–PROLOG TERMS AND SIMPLE COMMANDS**

AIM: To learn about Prolog terms and practice simple commands.

PROLOG TERMS:



1. Atoms

A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, starting with a lowercase letter
Examples: butch, big_kahuna_burger, playGuitar

2. Variables

A sequence of characters of uppercase letters, lower-case letters, digits, or underscore, starting with either an uppercase letter or an underscore
Examples: X, Y, Variable, Vincent, _tag

3. Complex Terms

Atoms, numbers and variables are building blocks for complex terms. Complex terms are built out of a functor directly followed by a sequence of arguments. Arguments are put in round brackets, separated by commas. The functor must be an atom
Examples we have seen before:

– *playsAirGuitar(jody)*
– *loves(vincent, mia)*
– *jealous(marsellus, W)*

Complex terms inside complex terms:
– *hide(X,father(father(father(butch))))*

4. Arity

The number of arguments a complex term has is called its arity
Examples:
woman(mia) is a term with arity 1
loves(vincent,mia) has arity 2
father(father(butch)) arity 1

You can define two predicates with the same functor but with different arity. Prolog would treat this as two different predicates.

EXERCISE:

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Queries:

```
?- woman(mia).
true.
?- playsAirGuitar(jody).
true.
?- playsAirGuitar(mia).
false.
?- tattoed(jody).
ERROR: predicate tattoed/1 false.t defined.
?- party.
true.
?- rockConcert.
false.
?-
```

Knowledge Base 2

```
happy(yolanda). %fact
listens2music(mia).
listens2music(yolanda):- happy(yolanda). %rule
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda). %head:-Body
```

Queries:

```
?- playsAirGuitar(mia).
```

true.
?- playsAirGuitar(yolanda).
true.

Explanation:

- There are five clauses in this knowledge base:
  two facts, and three rules.
- The end of a clause is marked with a full stop.
- There are three predicates in this knowledge base:
  happy, listens2music, and playsAirGuitar
- The comma “," expresses conjunction
- The comma “;" expresses disjunction

Knowledge Base 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

Queries:

```
?- playsAirGuitar(butch).
true.
```

Knowledge Base 4

```
woman(mia).
woman(jody).
woman(yolanda).
loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

Queries:

```
?- woman(X).
X=mia;
X=jody;
X=yolanda.
?- loves(marsellus,X), woman(X).
X=mia.
?- loves(pumpkin,X), woman(X).
false.
```

RESULT:

The Prolog simple and complex terms were learnt and executed successfully.

EX.NO: 4c **KNOWLEDGE REPRESENTATION AND REASONING USING PREDICATE LOGIC–IMPLEMENTATION OF FAMILY TREE PROBLEM**
DATE:

AIM: To represent the kinship domain for the given knowledge base in Prolog

KNOWLEDGE BASE:

- Tom is the parent of Liz.
- Bob is the parent of Ann.
- Bob is the parent of Pat.
- Pat is the parent of Jim.
- Jim is a male.
- Tom is a male.
- Bob is a male.
- Pam is a female.
- Liz is a female.
- Pat is a female.
- Ann is a female.

PROLOG – KINSHIP DOMAIN:

family.pl

```
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
male(jim).
male(tom).
male(bob).
female(pam).
female(liz).
female(pat).
female(ann).
```

Execute the following queries

1. female(pam).

2. Who is Pat's parent?

3. Given the above facts, extend the program by writing rules defining the following predicates:
A) father(X,Y) % X is the father of Y.
B) mother(X,Y) % X is the mother of Y.
C) grandmother(X,Y) %X is the grandmother of Y.

D) sister(X,Y)

4. female(pam).

5. parent(pat, ann).

6. parent(X, ann).

7. parent(X,_),female(X).

8. grandparent(bob).

9. sister(X,Y).

QUERIES:

```
?- female(pam).
true.
?- parent(pat, ann).
false.
?- parent(X, ann).
X = bob.
?- parent(X,_),female(X).
X = pat.

?- grandparent(bob).
true.

?- sister(X,Y).
X = ann,
Y = pat ;
X = pat,
Y = ann ;
false.
```

RESULT:

The kinship domain for the knowledge base was implemented in Prolog and the queries were executed successfully.

EX.NO: 5 **KNOWLEDGE REPRESENTATION AND REASONING USING**
DATE: **PREDICATE LOGIC–FORWARD CHAINING AND BACKWARD CHAINING**

AIM: To make inferencing using forward and backward Chaining in Prolog.

ADVANCED PROLOG COMMANDS:

1) Trace Predicate:

The trace predicate prints out information about the sequence of goals in order to show where the program has reached in its execution.

Some of the events which may happen during trace :

- CALL : Occurs when Prolog tries to satisfy a goal.
- EXIT : Occurs when some goal has just been satisfied.
- REDO : Occurs when the system comes back to a goal, trying to re-satisfy it
- FAIL : Occurs when a goal fails

2) Write Predicate:

- write(). - Writes a single term to the terminal
  Eg: write("Hi").
- write_ln(). – write() followed by new line
- tab(X). – writes X no. of spaces

3) Read Predicate:

- read(X). – reads a term from keyboard & instantiates variable X to value of the read term.

4) Assert Predicate:

- assert(X) – adds a new fact or clause to the database. Term is asserted as the last fact or clause with the same key predicate.
- asserta(X) – assert(X) but at the beginning of the database.
- assertz(X) – same as assert(X)

5) Retract Predicate:

- retract(X) – removes fact or clause X from the database.
- retractall(X) – removes all fact or clause from the database for which the head unifies with X.

KNOWLEDGE BASE - 1

Marcus is a man.
Marcus is a Pompeian.
All Pompeians are Romans.
Caesar is a ruler.
Marcus tried to assassinate Caesar.

People only try to assassinate rulers they are not loyal to.

Find: Is Marcus loyal to Caesar or not?

PROLOG & QUERIES:

KNOWLEDGE BASE - 1
Marcus is a man.
Marcus is a Pompeian.
All Pompeians are Romans.
Caesar is a ruler.
Marcus tried to assassinate Caesar.
People only try to assassinate rulers they are not loyal to.

Find: Is Marcus loyal to Caesar or not?

PROLOG & QUERIES:

```
man(marcus).
pompian(marcus).
Roman(X):-pompian(X).
ruler(caesar).
assasinate(marcus,caesar).
notloyalto(X,Y):-man(X),ruler(Y),assasinate(X,Y).
```

Query:

```
?- notloyalto(X,Y).
X = marcus,
Y = caesar.

?- man(X).
X = marcus.

?- pompeian(X).
X = marcus.

?- roman(X).
X = marcus.

?- man(X),ruler(Y),tryassasinate(X,Y).
X = marcus,
Y = caesar.


?- trace.
true.
```

[trace] ?- notloyalto(X,Y).
Call: (8) notloyalto(_7698, _7700) ? creep
Call: (9) man(_7698) ? creep
Exit: (9) man(marcus) ? creep
Call: (9) ruler(_7700) ? creep
Exit: (9) ruler(caesar) ? creep
Call: (9) assasinate(marcus, caesar) ? creep
Exit: (9) assasinate(marcus, caesar) ? creep
Exit: (8) notloyalto(marcus, caesar) ? creep
X = marcus,
Y = Caesar

KNOWLEDGE BASE - 2
Apple is a food.
Chicken is a food.
John eats all kinds of food.
Peanut is a food.
Anything anyone eats and is alive is a food.
John is alive.

Find : Does John eats peanuts?

PROLOG & QUERIES:
```
alive(john).
food(apple).
food(chicken).
food(peanuts).
eats(john,X):-food(X).
food(X):-eats(Y,X),alive(Y).
```

Query:
[trace] ?- eats(john,peanuts).
Call: (8) eats(john, peanuts) ? creep
Call: (9) food(peanuts) ? creep
Exit: (9) food(peanuts) ? creep
Exit: (8) eats(john, peanuts) ? creep
true .

KNOWLEDGE BASE - 3
1.Steve likes only easy courses.
2.Science courses are hard.
3.All courses in BasketWeaving department are easy.
4.bk301 is basketweaving course.

Predicate logic
∀x easy(x) -> likes(steve,x)

∀x science(x) - > ~easy(x)
∀x basketweaving(x) -> easy(x)
basketweaving(bk301)

PROLOG steve.pl

```
likes(steve,X):-easy(X).
hard(X):-not(easy(X)).
hard(X):-science(X).
basketweaving(bk301).
easy(X):-basketweaving(X).
```

Execute the query:
likes(steve,X).

Query and Output:

```
?- [subject].
true.

?- likes(steve,X).
X = bk301.
```

RESULT:
The given knowledge bases were successfully traced using Backward Chaining.

EX.NO: 6 **UNIFICATION PROBLEM USING PYTHON**
DATE;

AIM: To implement unification problem using python.

ALGORTHIM:

1. If Ψ1 or Ψ2 is a variable or constant, then:
    a) If Ψ1 or Ψ2 are identical, then return NIL.
    b) Else if Ψ1 is a variable,
        i. then if Ψ1 occurs in Ψ2, then return FAILURE
        ii. Else return { (Ψ2/ Ψ1)}.
    c) Else if Ψ2 is a variable,
        i. If Ψ2 occurs in Ψ1 then return FAILURE,
        ii. Else return {(Ψ1 / Ψ2)}.
    d) Else return FAILURE.
2. If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then return FAILURE.
3. IF Ψ1 and Ψ2 have a different number of arguments, then return FAILURE.
4. Set Substitution set(SUBST) to NIL.
5. For i=1 to the number of elements in Ψ1.
    a) Call Unify function with the ith element of Ψ1 and ith element of Ψ2, and put the result into S.
    b) If S = failure then returns Failure
    c) If S ≠ NIL then do,
        i. Apply S to the remainder of both L1 and L2.
        ii. SUBST= APPEND(S, SUBST).
6. Return SUBST.

PROGRAM :

```
def unify(t1,t2):
    sub=[]
    for i in range(0,n):
        if t1[i]==t2[i]:
            sub.append("Not Found")
        else:
            x=[t1[i],t2[i]]
            sub.append(x)
    print(sub)
t1=[]
t2=[]
n=int(input("Enter the no of elements of t1 : "))
n1=int(input("Enter the no of elements of t2 : "))
if(n==n1):
    print("Enter the elements of t1 : ")
    for i in range(n):
        a=input()
        t1.append(a)
```

```
    print("Enter the elements of t2 : ")
    for i in range(n1):
        b=input()
        t2.append(b)
    print("The list after making substring")
    unify(t1,t2)
else:
    print("Substitution can not be done")
```

SAMPLE INPUT AND OUTPUT:

```
Enter the no of elements of t1 : 3
Enter the no of elements of t2 : 3
Enter the elements of t1 :
a
b
c
Enter the elements of t2 :
x
y
z
The list after making substring
[['a', 'x'], ['b', 'y'], ['c', 'z']]
```

RESULT:

Thus, the implementation of unification problem using python is executed successfully and output is verified.

EX.NO: 7 IMPLEMENTATION OF FUZZY OPERATIONS ON FUZZY SETS
DATE:

AIM: To implement various fuzzy operations on fuzzy sets in Python.

FUZZY OPERATIONS:

1. Union

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Union of them, then for every member of A and B, Y will be:

degree_of_membership(Y)= max(degree_of_membership(A), degree_of_membership(B))

INPUT

First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}

OUTPUT

Fuzzy Set Union is : {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}

2. Intersection

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Union of them, then for every member of A and B, Y will be:

degree_of_membership(Y)= min(degree_of_membership(A), degree_of_membership(B))

INPUT

First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

Second Fuzzy Set is : {'a': 0.2, 'b': 0.9, 'c': 0.4, 'd': 0.5}

OUTPUT

Fuzzy Set Union is : {'a': 0.2}

3. Complement

Consider a Fuzzy Sets denoted by A, then let's consider Y be the Complement of it, then for every member of A, Y will be:

degree_of_membership(Y)= 1 - degree_of_membership(A)

INPUT

Fuzzy Set : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

OUTPUT

Fuzzy Set : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}

4. Max–Min Composition

Max-min composition is defined as,

$$\underline{T} = \underline{R} \circ \underline{S} = \mu_{\underline{T}}(x, z) = (\mu_{\underline{R}}(x, y) \vee \mu_{\underline{S}}(y, z))$$

$$= \min (\mu_{\underline{R}}(x, y) \vee \mu_{\underline{S}}(y, z)) \text{ for the fuzzy relations R and S}$$

EXAMPLE

Let X = {x1, x2}, Y = {y1, y2}, and Z = {z1, z2, z3}. Compute T = $\underline{T}$ = $\underline{R}$ ∘ $\underline{S}$ for the following fuzzy relations:

$$\bar{R} = \begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} y_1 & y_2 \end{array} \\ \begin{bmatrix} 0.7 & 0.6 \\ 0.8 & 0.3 \end{bmatrix} \end{array}$$

Fuzzy Relation R

$$\bar{S} = \begin{array}{c} \\ y_1 \\ y_2 \end{array} \begin{array}{c} \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{bmatrix} 0.8 & 0.5 & 0.4 \\ 0.1 & 0.6 & 0.7 \end{bmatrix} \end{array}$$

Fuzzy Relation S

OUTPUT

$\mu_{\underline{T}}(x_1, z_1) = \max (\min(\mu_{\underline{R}}(x_1, y_1), \mu_{\underline{S}}(y_1, z_1)), \min(\mu_{\underline{R}}(x_1, y_2), \mu_{\underline{S}}(y_2, z_1)))$

$= \max(\min(0.7, 0.8), \min(0.6, 0.1)) = \max(0.7, 0.1) = 0.7$

$\mu_{\underline{T}}(x_1, z_2) = \max (\min(\mu_{\underline{R}}(x_1, y_1), \mu_{\underline{S}}(y_1, z_2)), \min(\mu_{\underline{R}}(x_1, y_2), \mu_{\underline{S}}(y_2, z_2)))$

$= \max(\min(0.7, 0.5), \min(0.6, 0.6)) = \max(0.5, 0.6) = 0.6$

$\mu_{\underline{T}}(x_1, z_3) = \max (\min(\mu_{\underline{R}}(x_1, y_1), \mu_{\underline{S}}(y_1, z_3)), \min(\mu_{\underline{R}}(x_1, y_2), \mu_{\underline{S}}(y_2, z_3)))$

$= \max(\min(0.7, 0.4), \min(0.6, 0.7)) = \max(0.4, 0.6) = 0.6$

$\mu_{\underline{T}}(x_2, z_1) = \max (\min(\mu_{\underline{R}}(x_2, y_1), \mu_{\underline{S}}(y_1, z_1)), \min(\mu_{\underline{R}}(x_2, y_2), \mu_{\underline{S}}(y_2, z_1)))$

$= \max(\min(0.8, 0.8), \min(0.3, 0.1)) = \max(0.8, 0.1) = 0.8$

$\mu_{\underline{T}}(x_2, z_2) = \max (\min(\mu_{\underline{R}}(x_2, y_1), \mu_{\underline{S}}(y_1, z_2)), \min(\mu_{\underline{R}}(x_2, y_2), \mu_{\underline{S}}(y_2, z_2)))$

$= \max(\min(0.8, 0.5), \min(0.3, 0.6)) = \max(0.5, 0.3) = 0.5$

$\mu_{\underline{T}}(x_2, z_3) = \max (\min(\mu_{\underline{R}}(x_2, y_1), \mu_{\underline{S}}(y_1, z_3)), \min(\mu_{\underline{R}}(x_2, y_2), \mu_{\underline{S}}(y_2, z_3)))$

$= \max(\min(0.8, 0.4), \min(0.3, 0.7)) = \max(0.4, 0.3) = 0.4$

$$\bar{T} = \begin{matrix} & z_1 & z_2 & z_3 \\ x_1 \\ x_2 \end{matrix} \begin{bmatrix} 0.7 & 0.6 & 0.6 \\ 0.8 & 0.5 & 0.4 \end{bmatrix}$$

Resultant Fuzzy Relation T

5. Max–Product Composition

Max-product composition is defined as,

$\underline{T} = \underline{R} \,.\, \underline{S} = \mu_{\underline{T}}(x, z) = (\mu_{\underline{R}}(x, y) \,.\, \mu_{\underline{S}}(y, z))$ for the fuzzy relations R and S

EXAMPLE

Let X = {x1, x2}, Y = {y1, y2}, and Z = {z1, z2, z3}. Compute $T = \underline{T} = \underline{R} \circ \underline{S}$ for the following fuzzy relations:

$$\bar{R} = \begin{matrix} & y_1 & y_2 \\ x_1 \\ x_2 \end{matrix} \begin{bmatrix} 0.7 & 0.6 \\ 0.8 & 0.3 \end{bmatrix} \qquad \bar{S} = \begin{matrix} & z_1 & z_2 & z_3 \\ y_1 \\ y_2 \end{matrix} \begin{bmatrix} 0.8 & 0.5 & 0.4 \\ 0.1 & 0.6 & 0.7 \end{bmatrix}$$

Fuzzy Relation R **Fuzzy Relation S**

OUTPUT

$\mu_{\underline{T}}(x_1, z_1) = \max ((\mu_{\underline{R}}(x_1, y_1) \times \mu_{\underline{S}}(y_1, z_1)), (\mu_{\underline{R}}(x_1, y_2) \times \mu_{\underline{S}}(y_2, z_1)))$

$= \max((0.7 \times 0.8), (0.6 \times 0.1)) = \max(0.56, 0.06) = 0.56$

$\mu_{\underline{T}}(x_1, z_2) = \max ((\mu_{\underline{R}}(x_1, y_1) \times \mu_{\underline{S}}(y_1, z_2)), (\mu_{\underline{R}}(x_1, y_2) \times \mu_{\underline{S}}(y_2, z_2)))$

$= \max((0.7 \times 0.5), (0.6 \times 0.6)) = \max(0.35, 0.36) = 0.36$

$\mu_{\underline{T}}(x_1, z_3) = \max ((\mu_{\underline{R}}(x_1, y_1) \times \mu_{\underline{S}}(y_1, z_3)), (\mu_{\underline{R}}(x_1, y_2) \times \mu_{\underline{S}}(y_2, z_3)))$

$= \max((0.7 \times 0.4), (0.6 \times 0.7)) = \max(0.28, 0.42) = 0.42$

$\mu_{\underline{T}}(x_2, z_1) = \max ((\mu_{\underline{R}}(x_2, y_1) \times \mu_{\underline{S}}(y_1, z_1)), (\mu_{\underline{R}}(x_2, y_2) \times \mu_{\underline{S}}(y_2, z_1)))$

$= \max((0.8 \times 0.8), \min(0.3 \times 0.1)) = \max(0.64, 0.03) = 0.64$

$\mu_{\underline{T}}(x_2, z_2) = \max ((\mu_{\underline{R}}(x_2, y_1) \times \mu_{\underline{S}}(y_1, z_2)), (\mu_{\underline{R}}(x_2, y_2) \times \mu_{\underline{S}}(y_2, z_2)))$

$= \max((0.8 \times 0.5), (0.3 \times 0.6)) = \max(0.4, 0.18) = 0.40$

$\mu_{\underline{T}}(x_2, z_3) = \max ((\mu_{\underline{R}}(x_2, y_1) \times \mu_{\underline{S}}(y_1, z_3)), (\mu_{\underline{R}}(x_2, y_2) \times \mu_{\underline{S}}(y_2, z_3)))$

= max((0.8 × 0.4), (0.3 × 0.7)) = max(0.32, 0.21) = 0.32

$$\bar{T} = \begin{matrix} & z_1 & z_2 & z_3 \\ x_1 \\ x_2 \end{matrix} \begin{bmatrix} 0.56 & 0.36 & 0.42 \\ 0.64 & 0.40 & 0.32 \end{bmatrix}$$

Resultant Fuzzy Relation T

1) UNION

PROGRAM:

```
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)


for A_key, B_key in zip(A, B):
        A_value = A[A_key]
        B_value = B[B_key]

        if A_value > B_value:
                Y[A_key] = A_value
        else:
                Y[B_key] = B_value

print('Fuzzy Set Union is :', Y)
```

SAMPLE INPUT AND OUTPUT:

```
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union is : {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
```

2) INTERSECTION

PROGRAM:

```
A = dict()
B = dict()
Y = dict()
```

```
A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)


for A_key, B_key in zip(A, B):
        A_value = A[A_key]
        B_value = B[B_key]

        if A_value < B_value:
                Y[A_key] = A_value
        else:
                Y[B_key] = B_value
print('Fuzzy Set Intersection is :', Y)
```

SAMPLE INPUT AND OUTPUT:

```
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Intersection is : {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
```

3) COMPLEMENT

PROGRAM:

```
A = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

print('The Fuzzy Set is :', A)


for A_key in A:
Y[A_key]= 1-A[A_key]

print('Fuzzy Set Complement is :', Y)
```

SAMPLE INPUT AND OUTPUT:

```
The Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
```

4) MAX MIN

PROGRAM:

```
s = [[0.8,0.9],
     [0.5,0.4]
    ]

r = [[0.3,0.2,0.9],
     [0.4,0.6,0.7],
    ]

res = []
try:
    for i in range(len(s)):
        maxs = []
        for j in range(len(r[0])):
            mins = []
            for k in range(len(r)):
                mins.append(min(s[i][k],r[k][j]))
            maxs.append(max(mins))
        res.append(maxs)
    print(res)
except:
    print("matrix dimensions incompatible!")
```

SAMPLE INPUT AMD OUTPUT:

```
[[0.4, 0.6, 0.8], [0.4, 0.4, 0.5]]
```

5) MAX PRODUCT

PROGRAM:

```
s = [[0.8,0.9],
     [0.5,0.4]
    ]

r = [[0.3,0.2,0.9],
     [0.4,0.6,0.7],
    ]

res = []
try:
    for i in range(len(s)):
        maxs = []
        for j in range(len(r[0])):
            prods = []
            for k in range(len(r)):
                prods.append((s[i][k]*r[k][j]))
```

```
            maxs.append(max(prods))
        res.append(maxs)
    print(res)
except:
    print("matrix dimensions incompatible!")
```

SAMPLE INPUT AND OUTPUT:

[[0.36000000000000004, 0.54, 0.7200000000000001], [0.16000000000000003, 0.24, 0.45]]

RESULT:

Various fuzzy operations on fuzzy sets in Python were executed successfully in Python.

EX.NO: 8

DATE:

DEMPSTER-SHAFER THEORY OF EVIDENCE FOR MEDICAL DIAGNOSIS

AIM: To implement Dempster-Shafer theory of evidence algorithm for the medical diagnosis.

PYTHON PACKAGES REQUIRED:

py_dempster_shafer

A Python library for performing calculations in the Dempster-Shafer theory of evidence.

Features: Support for normalized as well as unnormalized belief functions Different Monte-Carlo algorithms for combining belief functions, various methods related to the generalized Bayesian theorem Measures of uncertainty Methods for constructing belief functions from data.

INSTALLATION COMMANDS:

A) Using pip

pip install py_dempster_shafer

B) conda install

conda-forge / packages / py_dempster_shafer

To install this package with conda run one of the following:

conda install -c conda-forge py_dempster_shafer

conda install -c conda-forge/label/gcc7 py_dempster_shafer

conda install -c conda-forge/label/cf201901 py_dempster_shafer

conda install -c conda-forge/label/cf202003 py_dempster_shafer

PROBLEM STATEMENT:

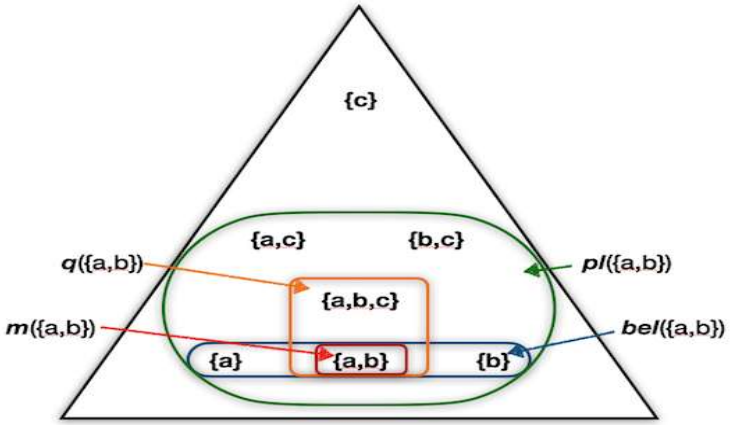
Let's say we have a person diagnosed with COVID -19 symptoms and have a belief of 0.5 for a proposition that the person is suffering from COVID -19. This means that we have evidence that makes us think strongly that the person is suffering from COVID (a proposition is true) with a confidence of 0.5. However, there is a contradiction that a person is not suffering from Covid with a confidence of 0.2. The remaining 0.3 is intermediate, which means the difference between confidence and contrast.

Calculate

A) Mass
B) Belief and
C) Plausibility for the information given below

Hypothesis	Mass	Belief	Plausibility
null	0	0	0
Suffering	0.5	0.5	0.8
Not suffering	0.2	0.2	0.5
Either (suffering or not)	0.3	1.0	1.0

The representation of the belief function is given below as an image.



ALGORITHM:

- Import package dempster_shafer
  - import dempster_shafer as ds
- Make a discernment frame for the items a, b, c, d
  - discernment = ds.FrameOfDiscernment(['a', 'b', 'c', 'd'])
- Define masses based on the results of the classifier in a random fashion
  - mass = ds.FocalSet(discernment, {"abc": 0.4, "abdc": 0.3,"a": 0.3})
- Create a lattice using the above frame of discernment and masses
  - lat = ds.Lattice(discernment, mass)
- Calculate the plausibility and belief based on lattice value
- Calculate plausibility as lat.pl()
- Calculate the degree of belief for evidence using lat.bel()

PROGRAM:

```
import dempster_shafer as ds
disernment = ds.FrameOfDiscernment(['a', 'b', 'c', 'd'])
```

```
mass = ds.FocalSet(disernment,
{
"abc": 0.4,
"abdc": 0.3,
"a": 0.3
})

lat = ds.Lattice(disernment, mass)
lat.pl()
lat.bel()
```

SAMPLE INPUT AND OUTPUT:

```
Plausibility: array([0. , 0.3, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 1. , 1. , 1. , 1. , 1. ,       1. , 1. , 1. ])
Belief: array([0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.3, 0.3, 0.3, 0.3, 0.3,   0.3, 0.7, 1. ])
```

RESULT: Dempster-Shafer theory of evidence for medical diagnosis problem was implemented in Python.

EX.NO: 9a

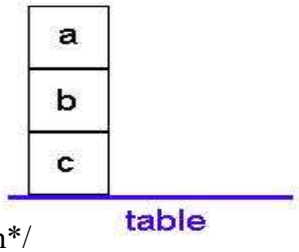
DATE:

PLANNING FOR BLOCK WORLD PROBLEM

AIM :

To implement Planning for Block World problem in Prolog.

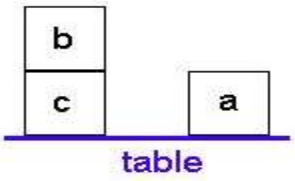
Initial State :



/* Initial state – Prolog representation*/

on(a,b).
on(b,c).
on(c,table).

Goal State :



/* Prolog representation of the state*/

on(a,table).
on(b,c).
on(c,table).

i) Non – Recursive actions for Block world problem

/***Action** */
This action specification has the form
action :- preconditions,
retract(affected_old_properties),
assert(new_properties).
/* **Non – Recursive action***/
:-dynamic on/2.

```
:-dynamic move/3.
put_on(A,B) :-
        A \== table,
        A \== B,
        on(A,X),
        clear(A),
        clear(B),
        retract(on(A,X)),
        assert(on(A,B)),
        assert(move(A,X,B)).

clear(table).
clear(B) :-
        not(on(_X,B)).
```

Queries :

```
?- put_on(a,table).
true.

?- listing(on), listing(move).
on(b,c).
on(c,table).
on(a,table).
move(a,b,table).
true.

?- put_on(c,a).
false.
```

The last goal fails since a block must have a clear top in order to be moved.

```
?- put_on(b,table), put_on(c,a).
true.
```

ii) Recursive actions for Block world problem

A recursive action specification can have the form

```
action :- preconditions or actions,
        retract(affected_old_properties),
        assert(new_properties).

:-dynamic on/2.
:-dynamic move/3.

on(a,b).
on(b,c).
```

```
on(c,table).

r_put_on(A,B) :-
        on(A,B).

r_put_on(A,B) :-
        not(on(A,B)),
        A \== table,
        A \== B,
        clear_off(A),   /* N.B. "action" used as precondition */
        clear_off(B),
        on(A,X),
        retract(on(A,X)),
        assert(on(A,B)),
        assert(move(A,X,B)).

clear_off(table). /* Means there is room on table */
clear_off(A) :- /* Means already clear */
        not(on(_X,A)).

clear_off(A) :-
        A \== table,
        on(X,A),
        clear_off(X),   /* N.B. recursion */
        retract(on(X,A)),
        assert(on(X,table)),
        assert(move(X,A,table)).
```

Queries :

```
?- r_put_on(c,a).
true.
?- listing(on), listing(move).
on(a,table).
on(b,table).
on(c,a).
move(a,b,table).
move(b,c,table).
move(c,table,a).

true.
```

RESULT :

The Planning for Block World problem was successfully done in Prolog.

EX.NO: 9b

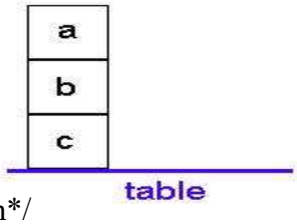
DATE:

PLANNING FOR BLOCK WORLD PROBLEM

AIM :

To implement Goal Stack Planning for Block World problem in Prolog.

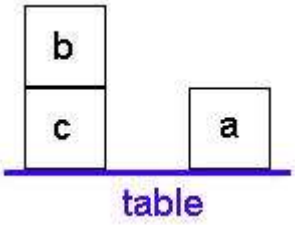
Initial State :



/* Initial state – Prolog representation*/

```
on(a,b).
on(b,c).
on(c,table).
```

Goal State :



/* Prolog representation of the state*/

```
on(a,table).
on(b,c).
on(c,table).
```

PROLOG :

```
test:-go([handempty,ontable(b),ontable(c),on(a,b),clear(c),clear(a)],[handempty,ontable(c),
        on(a,b),on(b,c),clear(a)]).
go(S,G):-plan(S,G,[S],[]).
plan(state,goal,been_list,moves):-move(name,preconditions,actions),
        conditions_met(preconditions,state),
        change_state (state,actions,child_state),
        hot(member_state(child_state,been_list)),
        stack(name,moves,new_moves),
        plan(child_state,goal,new-been_list,new_moves).
```

```
move(pickup(X),[handempty,clear(X),on(X,Y)],
        [del(handempty),del(clear(X)),del(on(X,Y)),
        [del(hendempty),del(clear(X),del(on(X,Y)),
        add(clear(Y)),add(holding(X))]).
move(pickup(X),[handempty,clear(X),ontable(x)],
        [del(handempty),del(ontable(X)),add(holding(X))]).
move(putdown(X,[holding(X)],[del(holding(X)),add(ontable(X)),add(clear(X)),
        add(handempty)]).
move(stack(X,Y),[holding(X),clear(Y)],[del(holding(X),del(clear(Y)),
        add(handempty,add(on(X,Y)),add(clear(X))]).
conditions_met(P,S):-subset(P,S).
change_state(S,[],S).
change_state(S,[add(P)|T],S_new]:- change_state(S,T,S2), add_to_set(P,S2,S_new),
change_state(S,[del(P)|T],S_new):-
                                change_state(S,TS2),
                                remove_from_set(P,S2,S_new),!.
member_state(S,[H]_]):-equal_set(S,H).
member_state(S,[_|T]):-member_state(S,T).
reverse_print_stack(S):-empty_stack(S).
reverse_print_stack:-stack(E,rest,S),  reverse_print_stack(rest),   write(E),nl.
```

QUERIES :

```
?-listing(plan).
plan(A,B,_C):-
                equal_set(A,B),
                write('moves are'),
                nl,
                reverse_print_stack(c).
plan(B,H,E,G):-
                move(F,A,C),
                conditions_met(A,B),
                change_state(B,C,D),
                not(members_state(D,E)),
                stack(D,E,I),
                stack(F,G,J),
                plan(D,H,I,J),!.
true.

?-plan(S,G,M).
ERROR: Undefined procedure:plan/3
ERROR: However,there are definitions for:
ERROR:plan/4
false.

?-listing(move).
move(pickup(A),[handempty,clear(A),on(A,B)],
[del(handempty),del(clear(A)),del(on(A,B)),
[del(hendempty),del(clear(A),del(on(A,B)),
add(clear(B)),add(holding(A))]).
```

```
move(pickup(A),[handempty,clear(B),ontable(A)],
[del(handempty),del(ontable(A)),add(holding(A))]).
move(putdown(A,[holding(A)],[del(holding(A)),add(ontable(A)),add(clear(A)),
add(handemptB)]).
move(stack(A,B),[holding(A),clear(B)],[del(holding(A),del(clear(B)),
add(handemptB,add(on(A,B)),add(clear(A))]).
```

RESULT :

The Goal Stack Planning for Block World problem was successfully done in Prolog.

EX.NO: 10

DATE:

EXPERT SYSTEM FOR MEDICAL DIAGNOSIS

AIM: To implement an expert system for Medical diagnosis in Python / Prolog.

EXPERT SYSTEM: Expert system uses databases of expert knowledge to offer advice or make decisions in such areas as medical diagnosis. It emulates the decision-making ability of a human expert. The knowledge databases are represented mainly as if–then rules rather than through conventional procedural code.

An Expert System is typically made up of 2 sub-systems :

(i) Knowledge Base :
- Represents facts and rules.

(ii) Inference Engine :
- Applies the rules to known facts to deduce new facts.

EXECUTION PROCEDURE:

1) Start SWIPL
2) Ensure JPL path is set.
3) Import Knowledge Base – Prolog Code
4) Type “start” in SWIPL prompt.
5) Interact with the GUI Prompt & Pop-ups.
6) Answer the Queries.
7) Result – Diagnosis Report.

PROLOG CODE:

```
:- use_module(library(jpl)).
start :-sleep(0.4),
                write('--------------------------------------'),nl,
                sleep(0.4),
                write('**************************************'),nl,
                sleep(0.2),
                write('#########||| Expert System |||################'),nl,
                sleep(0.4),
                write('**************************************'),nl,
                sleep(0.4),
                write('--------------------------------------'),nl,nl,nl,

/*write("Hi, How are you? May I know your name please?: "),
                read(patient),*/
```

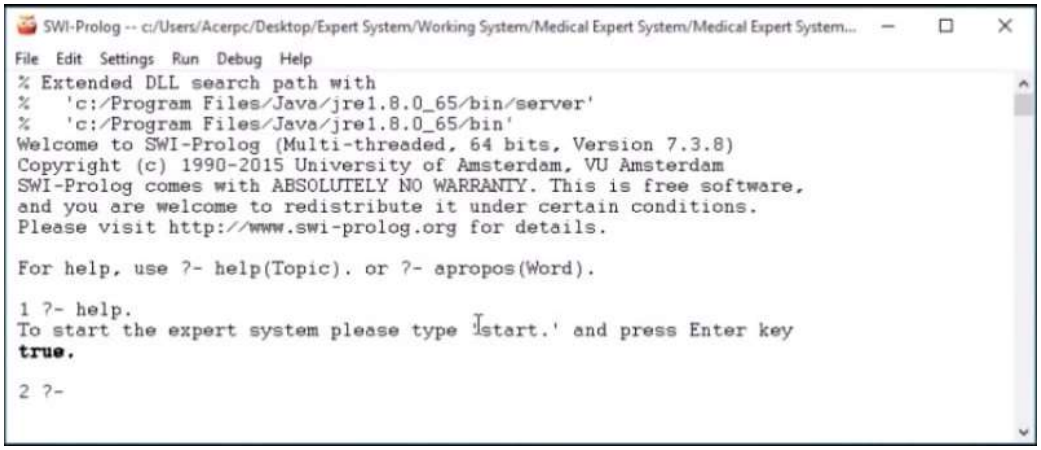
```
        interface2.

        /*hypothesis(patient,disease),
        write(patient),write(', you'),write(' probably have '),write(disease),write('.'),undo,
        nl,nl,nl,
        sleep(0.7),
        write('****************************************'),nl,
        sleep(0.4),
        write('#########||| Thanks for visiting. Have a good day!. |||################'),nl,
        sleep(0.4),
        write('****************************************'),nl.*/

symptom(patient,fever) :- verify(patient," have a fever (y/n) ?").
symptom(patient,rash) :- verify(patient," have a rash (y/n) ?").
symptom(patient,headache) :- verify(patient," have a headache (y/n) ?").
symptom(patient,runny_nose) :- verify(patient," have a runny_nose (y/n) ?").
symptom(patient,conjunctivitis) :- verify(patient," have a conjunctivitis (y/n) ?").
symptom(patient,cough) :- verify(patient," have a cough (y/n) ?").
symptom(patient,body_ache) :- verify(patient," have a body_ache (y/n) ?").
symptom(patient,chills) :- verify(patient," have a chills (y/n) ?").
symptom(patient,sore_throat) :- verify(patient," have a sore_throat (y/n) ?").
symptom(patient,sneezing) :- verify(patient," have a sneezing (y/n) ?").
symptom(patient,swollen_glands) :- verify(patient," have a swollen_glands (y/n) ?").
/*symptom(_,"Sorry, I couldn't diagnose the disease.").*/

hypothesis(patient,measles)   :-
        symptom(patient,fever),
        symptom(patient,cough),
        symptom(patient,conjunctivitis),
        symptom(patient,runny_nose),
        symptom(patient,rash).

hypothesis(patient,german_measles) :-
        symptom(patient,fever),
        symptom(patient,headache),
        symptom(patient,runny_nose),
        symptom(patient,rash).

hypothesis(patient,flu)          :-
        symptom(patient,fever),
        symptom(patient,headache),
        symptom(patient,conjunctivitis),
        symptom(patient,runny_nose),
        symptom(patient,body_ache),
        symptom(patient,chills),
        symptom(patient,cough),
        symptom(patient,sore_throat).

hypothesis(patient,common_cold)    :-
```

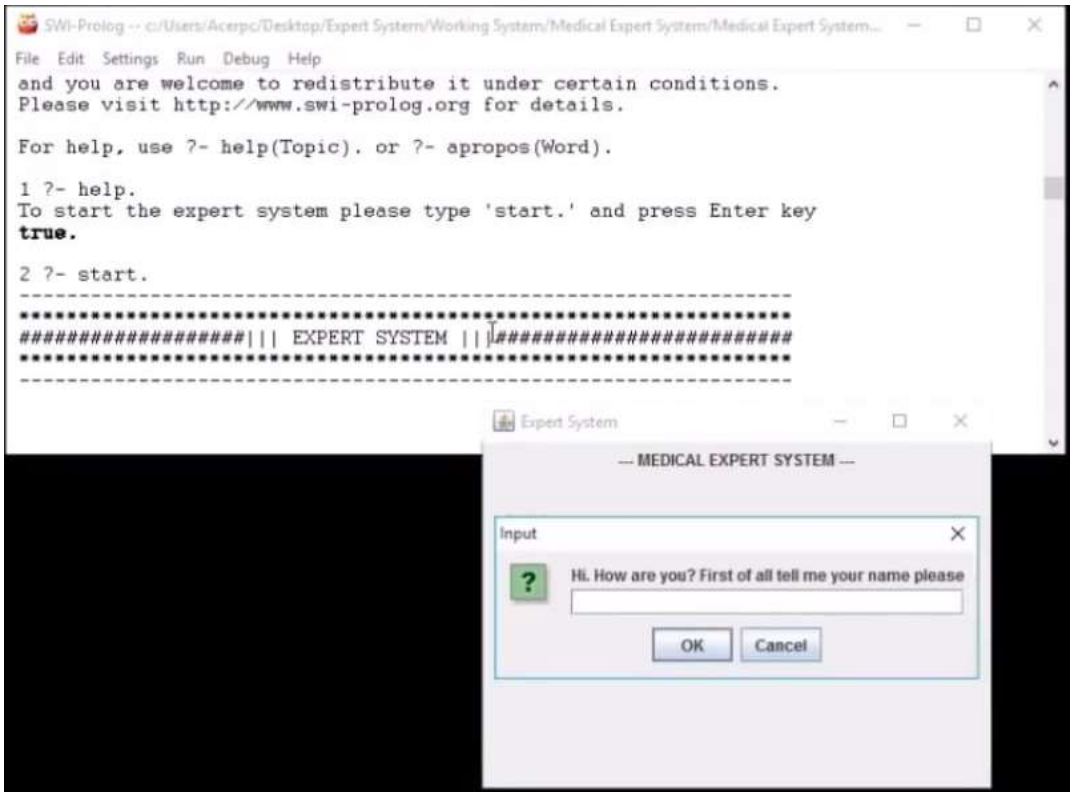
```
                symptom(patient,headache),
                symptom(patient,sneezing),
                symptom(patient,sore_throat),
                symptom(patient,runny_nose),
                symptom(patient,chills).

        hypothesis(patient,mumps)    :-
                symptom(patient,fever),
                symptom(patient,swollen_glands).

        hypothesis(patient,chicken_pox)       :-
                symptom(patient,fever),
                symptom(patient,chills),
                symptom(patient,body_ache),
                symptom(patient,rash).

        hypothesis(patient,measles)   :-
                symptom(patient,cough),
                symptom(patient,sneezing),
                symptom(patient,runny_nose).

        hypothesis(_,"disease. But I'm sorry. I couldn't diagnose the disease").

        response(reply) :-
                read(reply),
                write(reply),nl.

ask(patient,question) :-
        write(patient),write(', do you'),write(question),
        /*read(N),
        ( (N==yes; N==y)
          ->
          assert(yes(question));
          assert(no(question)), fail),*/
        interface(', do you',patient,question),
        write('Loading.'),nl,
        sleep(1),
        write('Loading..'),nl,
        sleep(1),
        write('Loading...'),nl,
        sleep(1),nl.

:- dynamic yes/1,no/1.

verify(P,S) :-
        (yes(S)
         ->
         true ;
         (no(S)
```

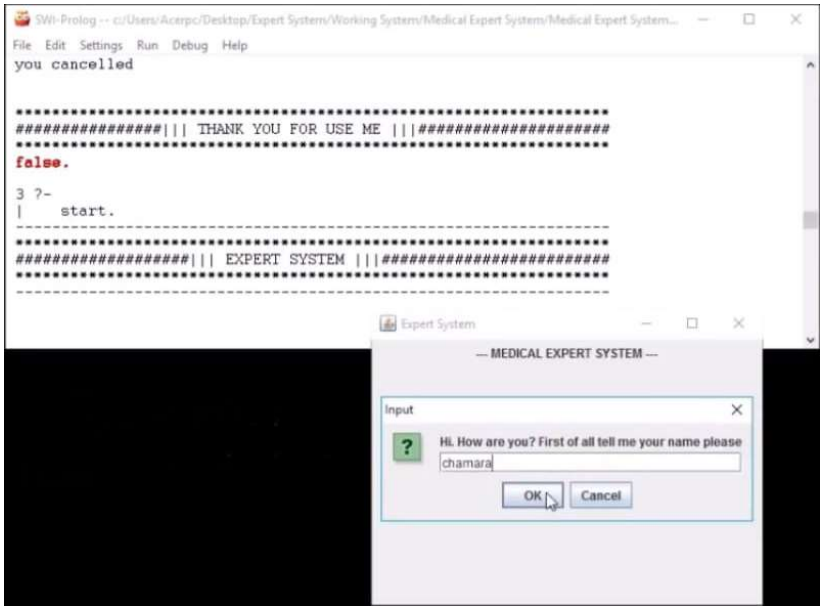
```
            ->
            fail;
            ask(P,S))).

undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail,
undo.
pt(patient):-

          hypothesis(patient,disease),
          interface3(patient,', you probably have ',disease,'.'),
          write(patient),write(', you probably haave '),write(disease),write('.'),undo,end.
end :-
       nl,nl,nl,
       sleep(0.7),
       write('****************************************'),nl,
          sleep(0.4),
          write('#########||| Thanks for visiting. Have a good day!. |||################'),nl,
          sleep(0.4),
          write('****************************************'),nl.

interface(X,Y,Z) :-
          atom_concat(Y,X, FAtom),
          atom_concat(FAtom,Z,FinalAtom),
          jpl_new('javax.swing.JFrame',['Expert System'], F),
          jpl_new('javax.swing.JLabel',['--- MEDICAL EXPERT SYSTEM ---'],LBL),
          jpl_new('javax.swing.JPanel',[],Pan),
          jpl_call(Pan,add,[LBL],_),
          jpl_call(F,add,[Pan],_),
          jpl_call(F, setLocation, [400,300], _),
          jpl_call(F, setSize, [400,300], _),
          jpl_call(F, setVisible, [@(true)], _),
          jpl_call(F, toFront, [], _),
          jpl_call('javax.swing.JOptionPane', showInputDialog, [F,FinalAtom], N),
          jpl_call(F, dispose, [], _),
          write(N),nl,
          ( (N == yes ; N == y)
            ->
            assert(yes(Z));
            assert(no(Z)), fail).

interface2 :-
          jpl_new('javax.swing.JFrame',['Expert System'], F),
          jpl_new('javax.swing.JLabel',['--- MEDICAL EXPERT SYSTEM ---'],LBL),
          jpl_new('javax.swing.JPanel',[],Pan),
          jpl_call(Pan,add,[LBL],_),
          jpl_call(F,add,[Pan],_),
          jpl_call(F, setLocation, [400,300], _),
          jpl_call(F, setSize, [400,300], _),
```

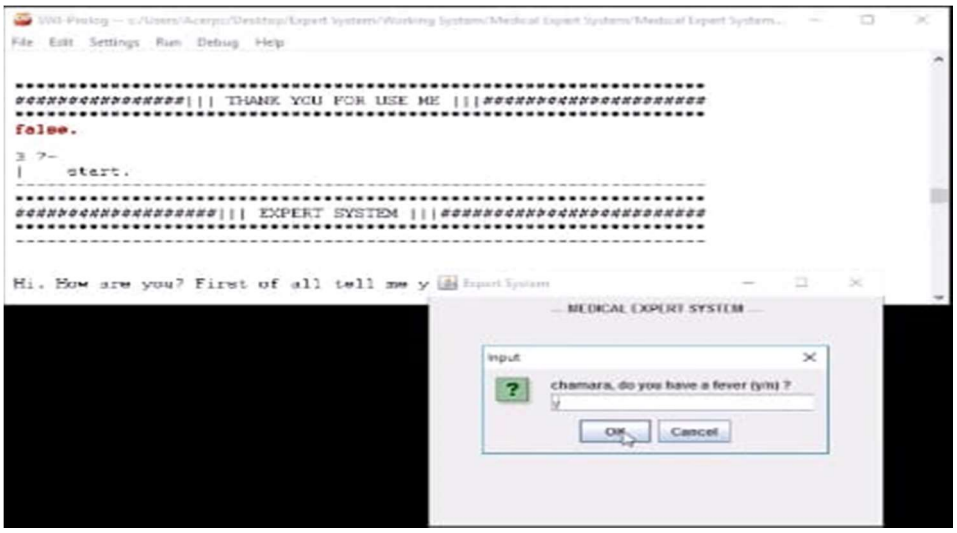
```
        jpl_call(F, setVisible, [@(true)], _),
        jpl_call(F, toFront, [], _),
        jpl_call('javax.swing.JOptionPane', showInputDialog, [F,'Hi, How are you? May I know
your name please?'], N),
        jpl_call(F, dispose, [], _),
        /*write(N),nl,*/
        ( (N == @(null)
          -> write('you cancelled'), interface3('you cancelled. ','Thank you ','for visiting.','Have a
good day.'),end,fail
          ;       write("Hi, How are you? May I know your name please?: "),write(N),nl,pt(N)
        ).

interface3(P,W1,D,W2) :-
        atom_concat(P,W1,A),
        atom_concat(A,D,B),
        atom_concat(B,W2,W3),
        jpl_new('javax.swing.JFrame',['Expert System'], F),
        jpl_new('javax.swing.JLabel',['--- MEDICAL EXPERT SYSTEM ---'],LBL),
        jpl_new('javax.swing.JPanel',[],Pan),
        jpl_call(Pan,add,[LBL],_),
        jpl_call(F,add,[Pan],_),
        jpl_call(F, setLocation, [400,300], _),
        jpl_call(F, setSize, [400,300], _),
        jpl_call(F, setVisible, [@(true)], _),
        jpl_call(F, toFront, [], _),
        jpl_call('javax.swing.JOptionPane', showMessageDialog, [F,W3], N),
        jpl_call(F, dispose, [], _),
        /*write(N),nl*/
        (  N == @(void)
                -> write('')
                ; write("")
        ).
help :- write("To start the expert system please type 'start.' and press the enter key").
```

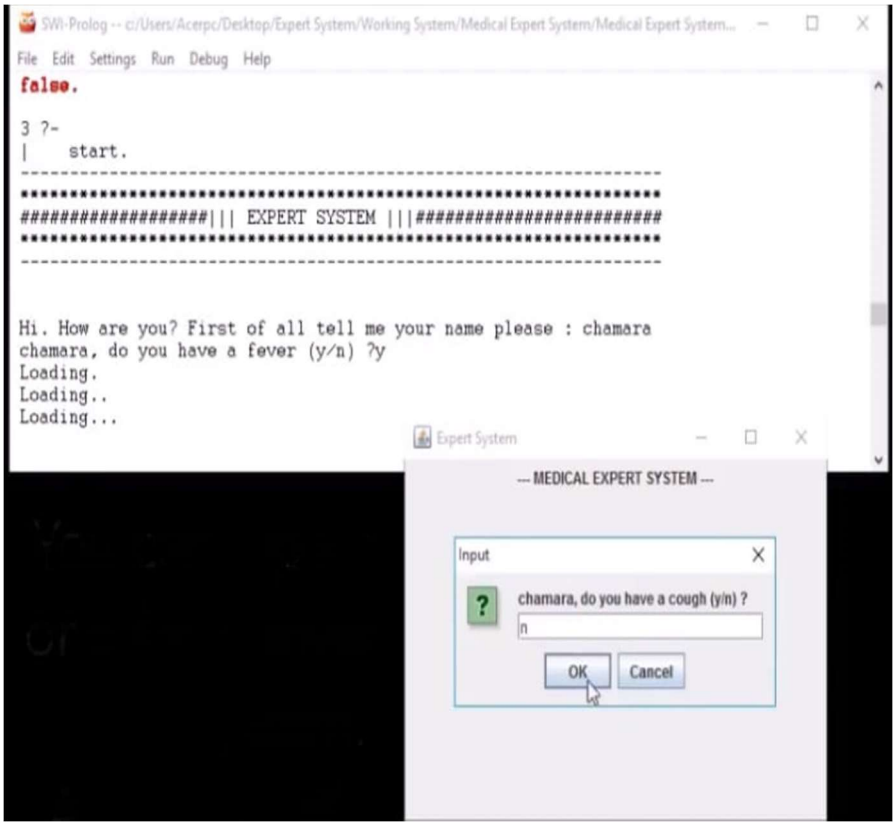
SAMPLE INPUT AND OUTPUT:


SWI-Prolog -- c:/Users/Acerpc/Desktop/Expert System/Working System/Medical Expert System/Medical Expert System...
File Edit Settings Run Debug Help
% Extended DLL search path with
% 'c:/Program Files/Java/jre1.8.0_65/bin/server'
% 'c:/Program Files/Java/jre1.8.0_65/bin'
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.3.8)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
1 ?- help.
To start the expert system please type 'start.' and press Enter key
true.
2 ?-

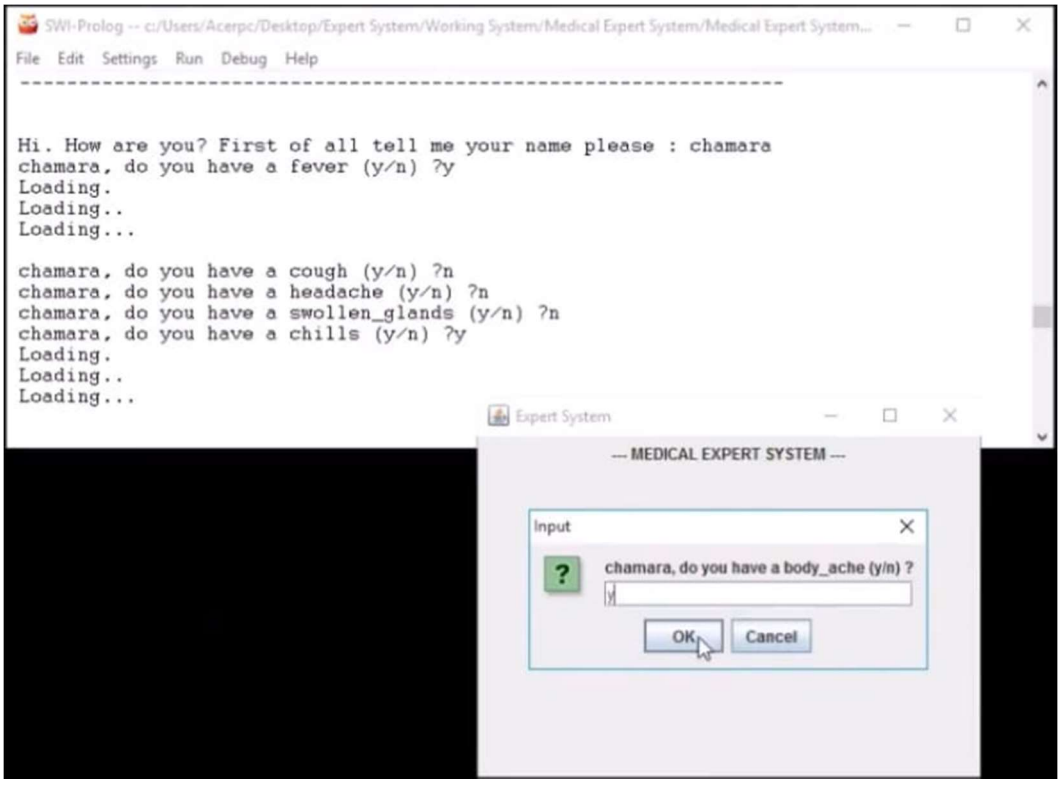

SWI-Prolog -- c:/Users/Acerpc/Desktop/Expert System/Working System/Medical Expert System/Medical Expert System...
File Edit Settings Run Debug Help
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
1 ?- help.
To start the expert system please type 'start.' and press Enter key
true.
2 ?- start.
EXPERT SYSTEM
Expert System
--- MEDICAL EXPERT SYSTEM ---
Input
Hi. How are you? First of all tell me your name please
OK
Cancel

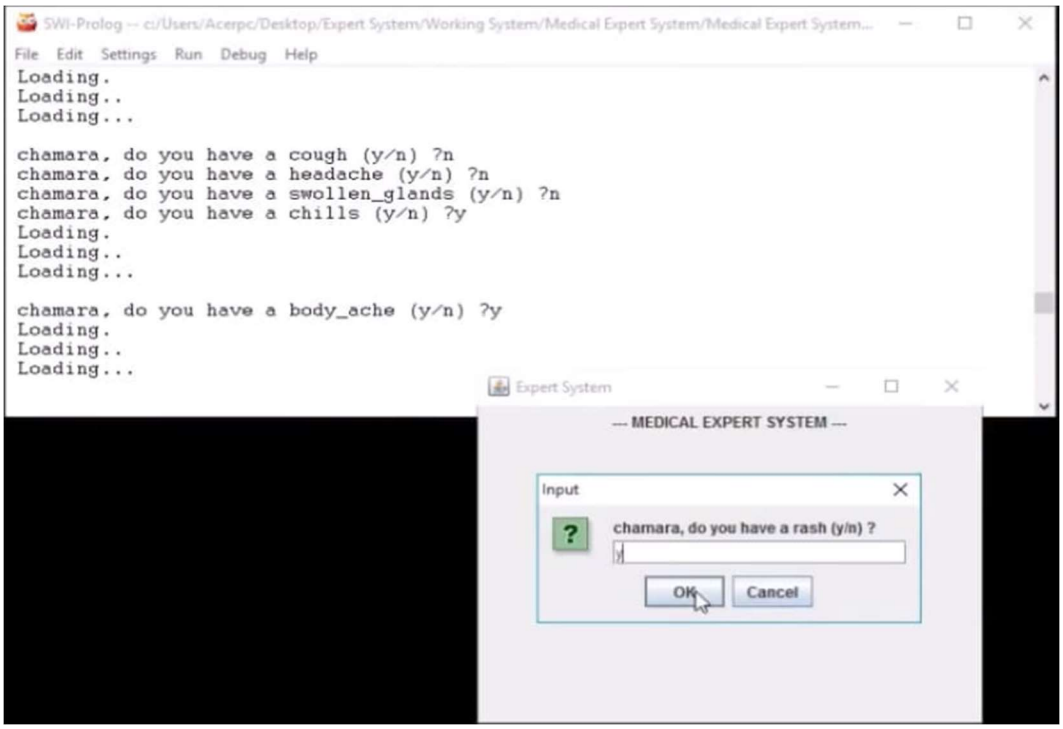

File Edit Settings Run Debug Help
you cancelled
################||| THANK YOU FOR USE ME |||####################
false.
3 ?-
| start.
##################||| EXPERT SYSTEM |||#########################
— MEDICAL EXPERT SYSTEM —
Input
Hi. How are you? First of all tell me your name please
chamara
OK
Cancel

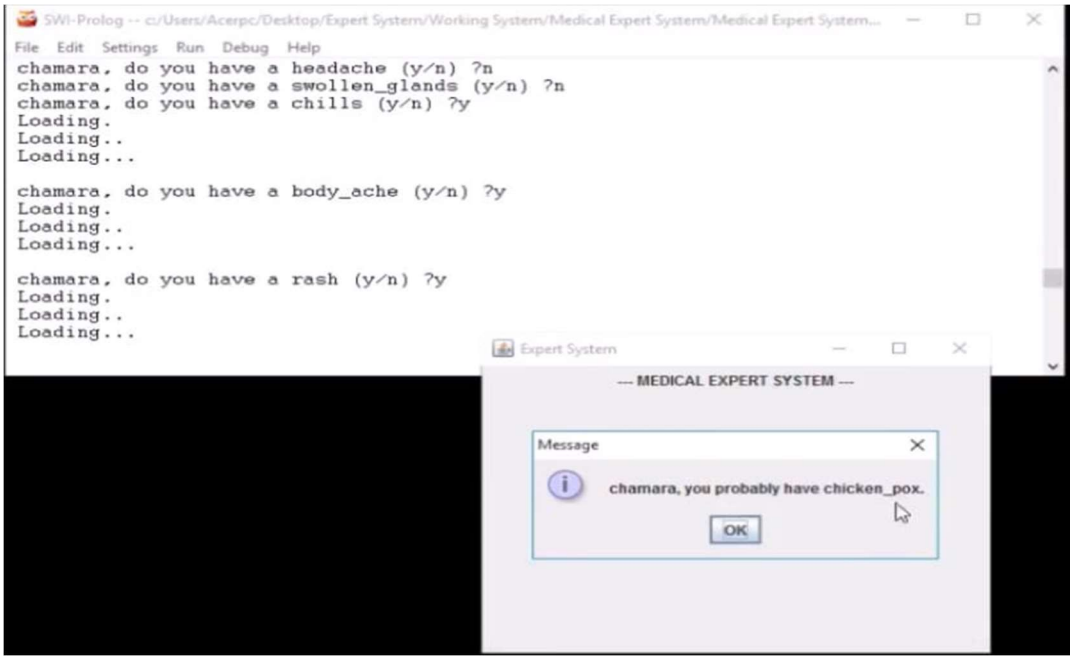

File Edit Settings Run Debug Help
################||| THANK YOU FOR USE ME |||####################
false.
3 ?-
| start.
##################||| EXPERT SYSTEM |||#########################
Hi. How are you? First of all tell me y
— MEDICAL EXPERT SYSTEM —
Input
chamara, do you have a fever (y/n) ?
y
OK
Cancel



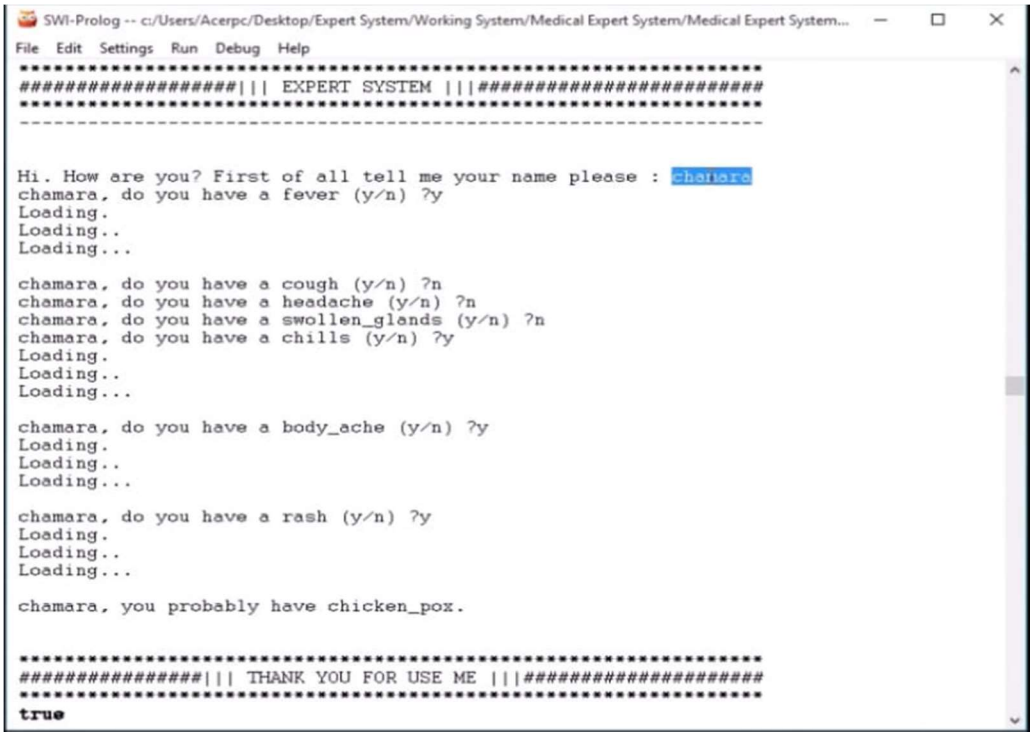
INPUT FROM USER FOR SYMPTOMS







INFERENCING FOR SYMPTOMS ENTERED BY USER



RESULT: The expert system for medical diagnosis was implemented successfully in Prolog.