Major Elective (RJSPGITE102) 2 Artificial Intelligence 50
Major Elective (RJSPGITE102P) 2 Artificial Intelligence Practical 50

Practical No	Practical	Date
1	Implementation of following uninformed search algorithms: a. Breadth First Search b. Depth First Search c. Bidirectional search.	8/7/24
2.	Implementation of Heuristic(informed) search algorithms: a. Hill Climbing search b. Best first search c. A* Search- d. AO*- https://www.geeksforgeeks.org/ao-algorithm-artificial-intelligence/	16/9/24
3.	Implementation of a. Tic-Tac-Toe game problem b. 8-Puzzle problem c. Water-Jug problem-	22/7/24
4.	Implementation of a. Map/Region Coloring Problem in AI. b. Construct a Maze Problem Solver	29/7/24
5.	Implementation of N-Queens Problem.	16/9/24
6.	Implementation of constraint satisfaction problems using Prolog. SEND+MORE=MONEY	23/9/24
7.	Implementation of logic programming using Prolog.	COMPLETED in multiple sessions

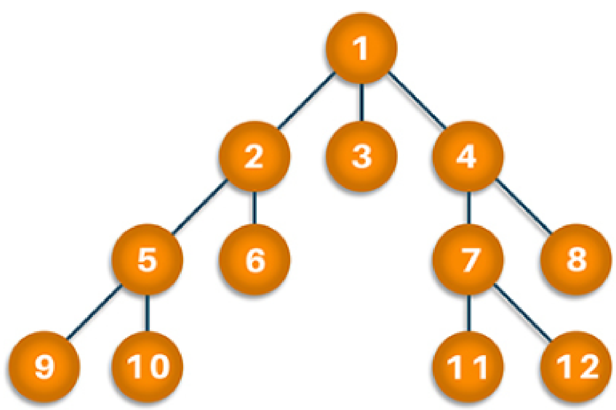
8.	**Implementation of a fuzzy-based application using Python / R.**	23/9/24

THEORY SYLLABUS

Unit	Topics	Lectures
I	**Introduction to Artificial Intelligence:** Introduction to AI and AI problems, The Foundations of AI: Philosophy, Mathematics, Economics, Neuroscience, Psychology, Computer Engineering, AI Applications. **Intelligent Agents:** Agents and Environments, The concept of rationality, The Nature of Environments, The structure of Agents.	
II	**Uninformed Search Strategies:** Breadth-first, Depth-first, Uniform-cost search, Iterative deepening depth-first search, Bidirectional search. **Informed (Heuristic) Search Techniques:** Heuristic Function, Best-first Search, Greedy best-first search, Generate-and-Test, Local Search Algorithm- Hill-climbing search, Simulated annealing, Problem Reduction- And-OR Search, A* search: Minimizing the total estimated solution cost,	
III	**Constraint Satisfaction problem:** Map Coloring, cryptarithmetic problem. **Adversarial Search:** Games, Optimal Decision in Games, Alpha-Beta Pruning Minimax Search Procedure, Adding Alpha-beta Cut-offs, Iterative Deepening. Chapter No-5, 6 from reference book	

IV	**Knowledge Representation, Reasoning, and Planning:** Logic: Propositional Logic, Propositional Theorem Proving, First Order Logic: Predicate Logic, Inference in First-Order Logic, Forward chaining, Backward chaining, Resolution. **Uncertain Knowledge and Probabilistic reasoning:** Quantifying uncertainty: Acting under uncertainty Basic probability notation, Inference using full joint distributions, independence, Bayes' rule and its use, fuzzy logic.	

AI Practical - Index (2023-2024)

Practical No	Details	Date
1.	Implementation of following search algorithms for the given tree:  a. Breadth First Search b. Depth First Search c. Bidirectional search. https://favtutor.com/blogs/breadth-first-search-python https://www.youtube.com/watch?v=U5-bRX2AHNY	15/7/24
2.	Implementation of Heuristic search algorithms: a. Hill Climbing search b. Best first search c. A* Search	

D. AO*

A Star Algorithm-

```
import heapq
import math

def heuristic_distance(point1, point2):
    #using manhaten distance
    #return abs(point1[0] - point2[0]) + abs(point1[1] - point2[1])
    #using eucledian distance
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)

def a_star(grid, start, goal):
    open_list = [(0, start)]
    came_from = {}
    g_score = {node: float('inf') for node in grid}
    g_score[start] = 0

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == goal:
            path = []
            while current in came_from:
                path.insert(0, current)
                current = came_from[current]
            path.insert(0, start)
            return path

        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            neighbor = current[0] + dx, current[1] + dy
            if neighbor in grid:
                tentative_g_score = g_score[current] + 1
                if tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    heapq.heappush(open_list, (g_score[neighbor] + heuristic_distance(neighbor, goal), neighbor))
```

```
    return None  # No path found

# Example grid (dict with (x, y) as keys)
grid = {
    (0, 0), (0, 1), (0, 2),(0,3),
    (1, 0), (1, 1), (1, 2),(1,3),
    (2, 0), (2, 1), (2, 2),(2,3)
}

start = (1, 0)
goal = (0, 3)
path = a_star(grid, start, goal)

if path:
    print("Path found:", path)
else:
    print("No path found.")
```

Output:

```
====================== RESTART: F:/Punam Part1/aSTAR.py =========
Path found: [(1, 0), (1, 1), (1, 2), (0, 2), (0, 3)]
```

AO * Algorithm:

```
import heapq

def ao_star(start, goal, heuristic, neighbors):
    open_list = [(0, start)]
# Priority queue initialized with the start node
    g_costs = {start: 0}
# Dictionary to store the cost of the shortest path to each node
    came_from = {}
 # Dictionary to reconstruct the path

    while open_list:
        _, current = heapq.heappop(open_list)
```

```
        if current == goal:
            path = [current]
            while current in came_from:
                current = came_from[current]
                path.append(current)
            return list(reversed(path))
# Return the path from start to goal

        for neighbor in neighbors(current):
            tentative_g = g_costs[current] + 1
# Assuming uniform cost for each step
            if tentative_g < g_costs.get(neighbor, float('inf')):
                g_costs[neighbor] = tentative_g
                f_cost = tentative_g + heuristic(neighbor, goal)
                came_from[neighbor] = current
                heapq.heappush(open_list, (f_cost, neighbor))

    return None
# Return None if no path is found

# Example usage:

def heuristic(point1, point2):
    # Manhattan distance heuristic for grid-based pathfinding
    return abs(point1[0] - point2[0]) + abs(point1[1] - point2[1])

def neighbors(node):
    # Example neighbor function for a grid (4-connected)
    x, y = node
    return [(x + dx, y + dy) for dx, dy in [(-1, 0), (1, 0), (0, -1), (0,
1)]]

# Example grid and start/goal points
start = (0, 0)
goal = (2, 2)

path = ao_star(start, goal, heuristic, neighbors)
print("Path found:", path)
```

	OUTPUT
	
	`========================= RESTART: F:/Punam Part1/AOStar1.py ================`
	`Path found: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2)]`	

3.	Implementation of a. Tic-Tac-Toe game problem https://drive.google.com/file/d/1vZqfLGEY35ey_qdk6hQPYaBobgwLMLGi/view?usp=sharing b. 8-Puzzle problem c. Water-Jug problem-	22/7/24

```
from collections import defaultdict
jug1=4, jug2=3, aim = 2
visited = defaultdict(lambda: False)
#print(visited )

def waterJugSolver(amt1, amt2):
   if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 ==
0):
      print(amt1, amt2)
      return True

   if visited[(amt1, amt2)] == False:
      print(amt1, amt2)
      visited[(amt1, amt2)] = True
      return (waterJugSolver(0, amt2) or
            waterJugSolver(amt1, 0) or
            waterJugSolver(jug1, amt2) or
            waterJugSolver(amt1, jug2) or
            waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                  amt2 - min(amt2, (jug1-amt1))) or
            waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                  amt2 + min(amt1, (jug2-amt2))))
   else:
      return False

print("Steps: ")
```

	waterJugSolver(0, 0)
	
	`======================= RESTART: D:\Punam AI\WaterJug.py ==`
	`Steps:`
	`0 0`
	`4 0`
	`4 3`
	`0 3`
	`3 0`
	`3 3`
	`4 2`
	`0 2`
	`>`	

	NOTE: if two versions of python are installed on the machine, verify that the package is installed in the required version. 1. Pip install simpleai 2. Check the multiple version of AI- a. C:\Users\Admin\AppData\Local\Programs\Python 3. Check the package is installed at this location- a. C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages 4. Copy and paste in other version also	

```
from simpleai.search import CspProblem, backtrack
def constraint_func(names, values):
    return values[0] != values[1]
if __name__=='__main__':
      names = ('WA', 'NT','SA', 'QL','Vict', 'Tasmania'
)
      colors = dict((name, ['red', 'green', 'blue' ]) for
name in names)
      constraints = [
         (('WA', 'NT'), constraint_func),
         (('WA', 'SA'), constraint_func),
         (('SA', 'QL'), constraint_func),
         (('SA', 'NT'), constraint_func),
         (('SA', 'Vict'), constraint_func),
         (('QL', 'NT'), constraint_func),
         (('QL', 'Vict'), constraint_func),
         (('Tasmania'), constraint_func)
      ]
      problem = CspProblem(names, colors,
```

constraints)

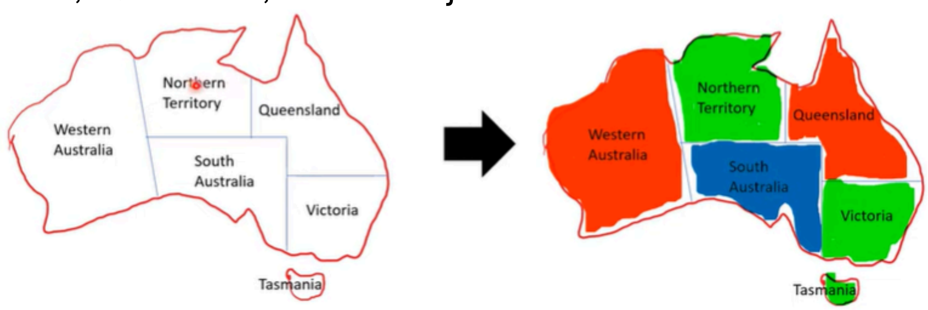
```
output = backtrack(problem)
print('\nColor mapping:\n')
for k, v in output.items():
    print(k, '==>', v)
```

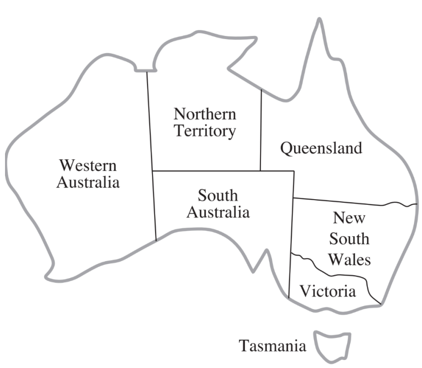
```
========================= RESTART: D:\Punam AI\csp.py ========

Color mapping:

WA ==> red
NT ==> green
SA ==> blue
QL ==> red
Vict ==> green
Tasmania ==> red
```

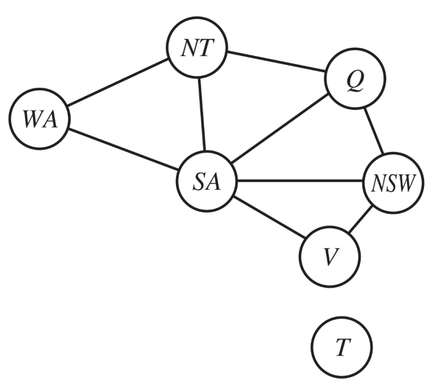
4. Map Coloring Problem Solution.

Region Coloring in AI

C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW , SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW , NSW ≠ V } .





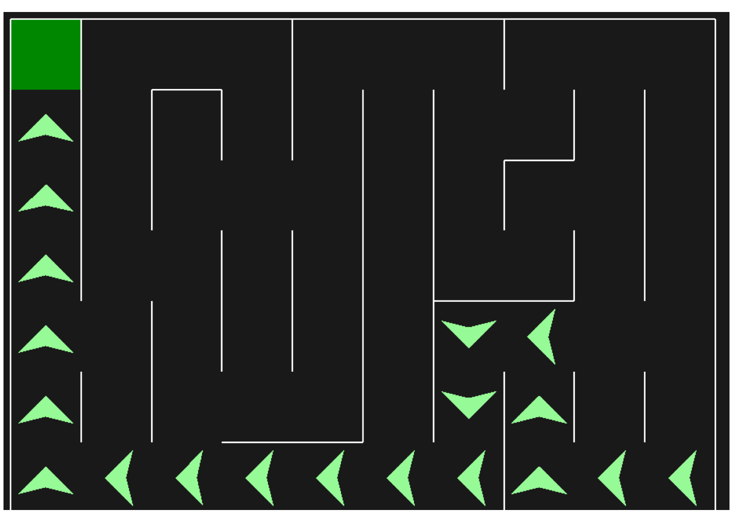


```python
from simpleai.search import CspProblem, backtrack

# Define the constraint function
def different_colors_constraint(names, values):
    return values[0] != values[1]

# Define the problem
def main():
    # Define variables
    variables = ['A', 'B', 'C']

    # Define domains (color choices)
    domains = {
        'A': ['red', 'green', 'blue'],
        'B': ['red', 'green', 'blue'],
        'C': ['red', 'green', 'blue']
    }

    # Define constraints
    constraints = [
        (('A', 'B'), different_colors_constraint),
        (('A', 'C'), different_colors_constraint),
        (('B', 'C'), different_colors_constraint)
    ]

    # Create CSP problem
    problem = CspProblem(variables, domains, constraints)

    # Solve the problem
    solution = backtrack(problem)

    print("Solution:", solution)

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

Map Coloring

```
from simpleai.search import CspProblem, backtrack

# Define the constraint function
def different_colors_constraint(names, values):
    return values[0] != values[1]

# Define the problem
def main():
    # Define variables
    variables = ['SA', 'WA', 'NT', 'Q', 'NSW', 'V' ]

    # Define domains (color choices)
    domains = {
        'SA': ['red', 'green', 'blue'],
        'WA': ['red', 'green', 'blue'],
        'NT': ['red', 'green', 'blue'],
        'Q':['red', 'green', 'blue'],
        'NSW':['red', 'green', 'blue'],
        'V':['red', 'green', 'blue']
    }

    # Define constraints
    constraints = [
        (('SA', 'WA'), different_colors_constraint),
        (('SA', 'NT'), different_colors_constraint),
        (('SA', 'Q'), different_colors_constraint),
        (('SA', 'NSW'), different_colors_constraint),
        (('SA', 'V'), different_colors_constraint),
        (('WA', 'NT'), different_colors_constraint),
        (('NT', 'Q'), different_colors_constraint),
        (('Q', 'NSW'), different_colors_constraint),
        (('NSW', 'V'), different_colors_constraint)

    ]

    # Create CSP problem
    problem = CspProblem(variables, domains, constraints)
```

	# Solve the problem
	solution = backtrack(problem)
	
	print("Solution:", solution)
	
	if __name__ == "__main__":
	main()
	
	3. Implementation of the Traveling Salesman Problem using Python.	

5	Construct a Maze Problem Solver - https://towardsdatascience.com/a-python-module-for-maze-search-algorithms-64e7d1297c96 from pyamaze import maze,agent,COLOR m=maze(7,10) m.CreateMaze(pattern='v', loopPercent=40, theme=COLOR.dark) a=agent(m,filled=True,shape = 'arrow', footprints=True, color='green') m.tracePath({a:m.path}) m.run() 	

	Explanation:
	1. **Importing models;**
	`from pyamaze import maze, agent, COLOR`
	
	`maze`, `agent`, and `COLOR` are imported from the `pyamaze` library.
	
	• `maze`: A class to create and manage the maze.
	• `agent`: A class to create and manage the agent navigating the maze.
	• `COLOR`: A module to define various colors for visualizing the maze.
	2. **Creating the Maze -** This line initializes a maze object with 7 rows and 10 columns. The maze is an instance of the `maze` class
	3. **Generating the Maze:**
	
	m.CreateMaze(pattern='v', loopPercent=40, theme=COLOR.dark)
	`CreateMaze`: A method to generate the maze with specific parameters.
	
	• `pattern='v'`: This specifies the pattern of the maze generation. The 'v' pattern stands for a vertical pattern with a variation of the maze layout.
	• `loopPercent=40`: This determines the percentage of the maze with loops, which makes the maze more complex.
	• `theme=COLOR.dark`: This sets the color theme of the maze to a dark color palette.
	
	4. Adding the Agent:
	`a = agent(m, filled=True, shape='arrow', footprints=True, color='green')`
	
	This line initializes an agent object that will navigate the maze.	

	• `m`: The maze object to which the agent is added. • `filled=True`: Indicates that the agent will be represented as a filled shape. • `shape='arrow'`: Sets the shape of the agent to an arrow. • `footprints=True`: The agent will leave a trail of footprints as it moves. • `color='green'`: Sets the color of the agent to green. 5. Tracing the Path; m.tracePath({a:m.path}) `tracePath`: A method to visualize the path taken by the agent. • `{a:m.path}`: This maps the agent `a` to its path `m.path`. The path is a sequence of steps that the agent has taken.	

6.	Implementation of N-Queens Problem.
	
	def is_safe(board, row, col):
	    # Check the left side of the current row
	    for i in range(col):
	        if board[row][i] == 1:
	            return False
	    # Check upper diagonal on the left
	    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
	        if board[i][j] == 1:
	            return False
	    # Check lower diagonal on the left
	    for i, j in zip(range(row, len(board), 1), range(col, -1, -1)):
	        if board[i][j] == 1:
	            return False
	    return True
	
	def solve_n_queens(board, col):
	    if col >= len(board):
	        return True
		16/9/24

```
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens(board, col + 1):
                return True
            board[i][col] = 0
    return False

def n_queens(n):
    board = [[0] * n for _ in range(n)]
    if not solve_n_queens(board, 0):
        print("No solution found.")
        return
    for row in board:
        print(" ".join(["Q" if cell == 1 else "." for cell in row]))
if __name__ == "__main__":
    n = 8  # Change this to the desired board size
    n_queens(n)
```

Output

```
====================== RESTART: F:/Punam Part1/n-queen.py =====
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

7.	Implementation of constraint satisfaction problems using Prolog.	

		T	O
+		G	O
	O	U	T

			I	S
+	T	H	I	S
	H	E	R	E

	M	A	T	H
+	M	Y	T	H
	H	A	R	D

		T	W	O
+		T	W	O
	F	O	U	R

```
% Define the constraint that ensures no two variables have the
same digit.
all_different([]).
all_different([H|T]) :- \+ member(H, T), all_different(T).

% Define the main predicate for solving the puzzle.
send_more_money([S, E, N, D, M, O, R, Y]) :- Digits = [0, 1, 2,
3, 4, 5, 6, 7, 8, 9],
member(S, Digits), S > 0,  member(E, Digits),member(N,
Digits),member(D, Digits),
member(M, Digits), M > 0, member(O, Digits),member(R,
Digits),member(Y, Digits),
all_different([S, E, N, D, M, O, R, Y]), 1000 * S + 100 * E + 10 *
N + D +
```

<table>
<tr>
<td></td>
<td>
1000 * M + 100 * O + 10 * R + E =:= 10000 * M + 1000 * O + 100 * N + 10 * E + Y.

Output -
<pre>
| ?- [send_money].
compiling F:/Punam Part1/send_money.pl for byte code...
F:/Punam Part1/send_money.pl compiled, 9 lines read - 4427 bytes written, 6 ms

yes
| ?- send_more_money([S, E, N, D, M, O, R, Y]).

D = 7
E = 5
M = 1
N = 6
O = 0
R = 8
S = 9
Y = 2 ?
</pre>
</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>
Implementation of logic programming using Prolog.

1. Define apple, orange, banana, grapes etc… as fruits
Eg- fruits(apple).
2. Define tomato, chilli, potato, capsicum etc… as veg
Eg- veg(tomato).
3. Define some fruits as sweet and some fruits as sour.
Eg- sweet(apple).
Eg- sour(grapes).
4. Write two rules for stating ‘I like sweet fruits’ and ‘i don't like sour fruits’
Eg. like(X) :- fruit(X), sweet(X)
dont_like(X):-fruit(X), sour(X)
5. Query- which fruit you like or don't like??
SOLUTION
fruits(apple).
fruits(orange).
fruits(grapes).
fruits(banana).
sweet(banana).
sweet(apple).
sour(grapes).

i_likes(X):- fruits(X),sweet(X).
</td>
<td>Done</td>
</tr>
</table>

<table>
<tr><td></td><td>
Execute the file
[fruit].

Query-

?- i_likes(banana).

yes

| ?- i_likes(orange).

no

| ?- i_likes(X).

X = apple ? ;

X = banana

yes
| ?- i_likes(ladoo).

no
| ?- i_likes(banana).

yes
| ?- i_likes(apple).

yes
| ?- i_likes(orange).

no
| ?- i_likes(grapes).

no
| ?-
</td><td></td></tr>
<tr><td>9.</td><td>Implementation of a fuzzy-based application using Python / R.

pip install skfuzzy
pip install networkx</td><td>16/9/24</td></tr>
</table>

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Create input and output variables
temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

# Define fuzzy sets for temperature
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature['warm'] = fuzz.trimf(temperature.universe, [0, 50, 100])
temperature['hot'] = fuzz.trimf(temperature.universe, [50, 100, 100])

# Define fuzzy sets for fan_speed
fan_speed['low'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [0, 50, 100])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Define rules
rule1 = ctrl.Rule(temperature['cold'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['warm'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['hot'], fan_speed['high'])

# Create the control system
fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

# Create a simulation
fan_sim = ctrl.ControlSystemSimulation(fan_ctrl)

# Input temperature value
temperature_input = 15

# Set the input temperature
```

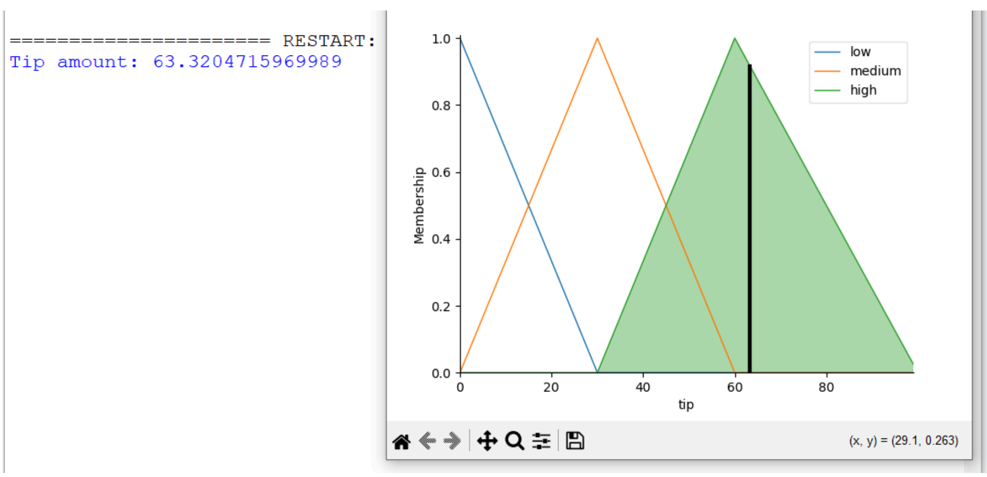
```
fan_sim.input['temperature'] = temperature_input

# Compute the fan speed
fan_sim.compute()

# Get the fan speed value
fan_speed_output = fan_sim.output['fan_speed']

print(f"For a temperature of {temperature_input} degrees:")
print(f"Fan Speed: {fan_speed_output:.2f}%")
```

Example 2-

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Define the variables
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 100, 1), 'tip')

# Auto membership functions
quality.automf(3)
service.automf(3)

# Tip membership functions
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 30])
tip['medium'] = fuzz.trimf(tip.universe, [0, 30, 60])
tip['high'] = fuzz.trimf(tip.universe, [30, 60, 100])

# Define fuzzy rules
rules = [
    ctrl.Rule(quality['good'] & service['good'], tip['high']),
    ctrl.Rule(quality['good'] & service['average'], tip['medium']),
    ctrl.Rule(quality['average'] & service['good'], tip['medium']),
```

```
    ctrl.Rule(quality['average'] & service['average'], tip['medium']),
    ctrl.Rule(quality['poor'] & service['poor'], tip['low']),
    ctrl.Rule(quality['poor'] & service['average'], tip['low']),
    ctrl.Rule(quality['average'] & service['poor'], tip['low']),
]

# Create control system
tip_ctrl = ctrl.ControlSystem(rules)
tip_simulation = ctrl.ControlSystemSimulation(tip_ctrl)

# Example input
tip_simulation.input['quality'] = 10  # Good
tip_simulation.input['service'] = 10   # Good

# Compute the tip
tip_simulation.compute()

# Output
print(f'Tip amount: {tip_simulation.output["tip"]}')
tip.view(sim=tip_simulation)
```

OUTPUT



Prolog video-
https://www.youtube.com/watch?v=bU1vbhdFFPc
Download-

http://www.gprolog.org/#download
Simple Questions
https://www.tutorialspoint.com/prolog/prolog_relations.htm
https://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html
https://athena.ecs.csus.edu/~mei/logicp/prolog/programming-examples.htm
l
Example prolog -
-https://drive.google.com/file/d/1VwdO8Y7Aqxrz8MDtWaAJBvuqdhPBRoQ
U/view?usp=sharing

```
has_symptom(flu, fever).
has_symptom(flu, headache).
has_symptom(flu, body_aches).
has_symptom(flu, cough).
has_symptom(flu, sore_throat).
has_symptom(flu, runny_nose).
has_symptom(allergy, sneezing).
has_symptom(allergy, watery_eyes).
has_symptom(allergy, runny_nose).
has_symptom(allergy, itchy_eyes).
has_symptom(cold, sneezing).
has_symptom(cold, watery_eyes).
has_symptom(cold, runny_nose).
has_symptom(cold, cough).
has_symptom(cold, sore_throat).
% Rule
has_condition(X,C) :- has_symptom(C,X).
% Query
has_condition(sneezing, X)?
```

In this example, the program defines a set of facts that describe the symptoms of three different medical conditions: the flu, allergies and the common cold. The program also defines a rule using the `has_condition/2` predicate, which states that if a patient has a certain symptom, then they have the medical condition that is associated with that symptom.

Finally, the program includes a query that asks the interpreter to determine which medical condition a patient has based on their reported symptoms. In this case, the query specifies that the patient has the symptom of sneezing, and it asks the interpreter to determine which medical condition the patient has.

The interpreter will use the `has_condition/2` rule and the `has_symptom/2` facts to deduce that the patient has either the flu, allergies or the common cold, and it will return one of these conditions as the solution to the query.

This simple Prolog program demonstrates how the language can be used to develop an AI application that can diagnose medical conditions based on symptoms. Of course, in a real-world application, the program would need to be much more comprehensive and sophisticated, with a larger set of rules and facts and the ability to handle a wider range of symptoms and conditions

https://www.youtube.com/watch?v=SykxWpFwMGs&t=543s

Prolog Practice Questions:

1.	Consider the following facts- 1. Perry is a Cat. 2. Perry has white spots. 3. Jerry is a Dog. 4. Perry has black spots. Rule - 1. Mary owns a Pet if it is a cat and it has White spots. 2. If someone owns something, he/she loves it. Perform the following queries in prolog- 1. Who is a cat? 2. Who has black spots? 3. Who owns a pet? 4. Whom does Mary love?
2.	Consider the following facts- 1. John likes Jane 2. Jane likes John 3. Jack likes Jane Rule - if X likes Y and Y likes X, X and Y are friends. Perform the following queries in prolog- 1. Is Jane a friend of Jack? 2. Whom does Jane like?

3. Is John a friend of Jane?
4. Who is the friend of Jack?

3.

Consider the following facts-

1. Burger is a food.
2. Pasta is a food.
3. Pizza is a food.
4. Pizza is a lunch
5. Pasta is a dinner.

Rule- Every food is a meal OR anything is a meal if it is a food.

Perform the following queries in prolog-

1. Is pizza a food?
2. Which food is a meal and dinner?
3. Is pasta a dinner?
4. Is pizza a dinner?

4.

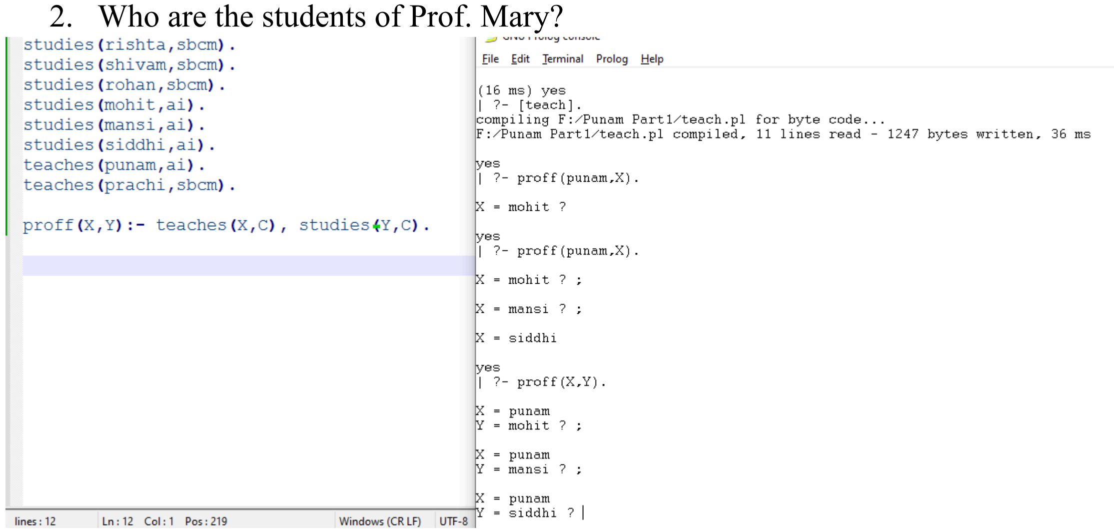
Consider the following facts-

1. Rahul studies Java.
2. Sneha Studies Jave.
3. Jiya studies Statistics.
4. Prof Mary teaches Java.
5. Prof Jary teaches Statistics.

Rule- X is a professor of Y if X teaches C and Y studies C.

Perform the following queries in prolog-

1. Rahul studies what?
2. Who are the students of Prof. Mary?



5.

Consider the following facts-

	1. Jack owns bmw car. 2. John owns chevy car. 3. Olivia owns civic car. 4. Jane owns chevy car. 5. bmw car is sedan. 6. civic car is sedan. 7. chevy car is truck. Perform the following queries in prolog- 1. What does john own? 2. Does john own something? 3. Who owns car chevy? 4. Does jane own sedan? 5. Does jane own truck?
6.	Consider the following predicates - 1. Sudip is the father of Piyus 2. Sudip is the father of Raj. 3. Piyush is male 4. Raj is male Rules- 1. If A and B both are male and both have the same mother and father and A is not equal to B then A and B are brothers. Perform the following queries in prolog- 1. Is Sudip brother of Piyush? 2. Who is the brother of Raj? 3. What is the relation of Piyush and Raj? 4. List all the males.
7.	Consider the following predicates - 1. Pam is the parent of Bob 2. Tom is the parent of Bob. 3. Bob is the parent of Ann. 4. Bob is the parent of Pat. 5. Pam is a female 6. Tom is a male. 7. Bob is a male. 8. Ann is a female. 9. Pat is a female. Rules- 1. X is the mother of Y if X is the parent of Y and X is a female. mother(X,Y):-parent(X,Y),female(X) 2. X is the father of Y if X is the parent of Y and X is a male.

	Perform the following Query in Prolog- 1. Who is the mother of Bob? 2. What is the relation of Tom and Bob? 3. What is the relation of Bob and Pat? 4. Is Bob the mother of Ann?
8	Consider the following facts- 1. Jack owns bmw car. 2. John owns chevy car. 3. Olivia owns civic car. 4. Jane owns chevy car. 5. bmw car is sedan. 6. civic car is sedan. 7. chevy car is truck. Prolog Queries- 1. What does john own? 2. Does john own something? 3. Who owns car chevy? 4. Does jane own sedan? 5. Does jane own truck?
9	Consider the following predicates - 1. Sudip is the father of Piyus 2. Sudip is the father of Raj. 3. Piyush is male 4. Raj is male Rules- 1. If A and B both are male and both have the same mother and father and A is not equal to B then A and B are brothers. Perform the following queries in prolog- 1. Is Sudip brother of Piyush? 2. Who is the brother of Raj? 3. What is the relation of Piyush and Raj? 4. List all the males.
10	bigger(elephant, horse). bigger(elephant, horse). bigger(horse, donkey). bigger(donkey, dog). bigger(donkey, monkey).

	is_bigger(X, Y) :- bigger(X, Y). is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y). Query: Is monkey bigger than elephant?
11	Consider the following predicates - 1. Randhir is the parent of Kareena. 2. Babita is the parent of Kareena. 3. Randhir is the parent of Karishma. 4. Babita is the parent of Karishma. 5. Kareena is the parent of Taimur 6. Kareena is the parent of Jahangir 7. Babita is a female. 8. Randhir is a male. 9. Karina is a female 10. Karishma is a female. 11. Taimur is a male. 12. Jahangir is a male. 13. Saif is a male. 14. Saif is the parent of Taimur. 15. Saif is the parent of Jahangir. Rules- 1. Define mother_of relation as : X is the mother of Y if X is the parent of Y and X is a female. - mother_of(X,Y):-parent(X,Y),female(X) 2. Define father_of relation as: X is the father of Y if X is the parent of Y and X is a male. 3. Define Spouse_of relation 4. Define sibling_of relation as: X is the sibling of Y if Z is the parent of X and Z is the parent of Y. 5. Define aunt_of relation using sibling_of relation - aunt_of(X, Z) :- sister_of(X, Y), parent(Y, Z). 6. Define grand_parent relation as: X is the grand parent of Y if X is the parent of parent of Y.