**Rayat Shikshan Sanstha's**
**KARMAVEER BHAURAO PATIL COLLEGE,VASHI**
**(Empowered Autonomous)**

Reaccredited NAAC with Grade 'A++' (CGPA3.51)|ISO 9001:2015 Certified Institute 'Best College' Award by University of Mumbai

**[DEPARTMENT OF INFORMATION TECHNOLOGY]**

# CERTIFICATE

This is to certify that **Mr./Miss:** SHUBHAM VIKAS NAVALE student of Class M S C I T from **Karmaveer Bhaurao Patil College, Vashi, Navi Mumbai** has satisfactorily completed the practical course in subject **Computer Forensics.** As per the syllabus laid by the college during the Academic. Year **2024-25.**

**ROLL NO.: 240609**

**EXAM NO.: 240609**

**Date:__/__/2025**

**MANOJ CHOUDHARY**
Course Coordinator

**MADHURI GABHANE**
Head, Department of IT

**External Examiner**

# INDEX

| SR.NO. | PRACTICAL NAME | DATE | SIGN |
|---|---|---|---|
| 01 | Implement a python program to show the working of a simple chatbot. | | |
| 02 | Implement a python program for a simple URL shortener. | | |
| 03 | Implement operations of fuzzy set(union, intersection, complement and difference) | | |
| 04 | Design an expert system for responding to patient queries for identifying the flu. | | |
| 05 | Implement Breadth First Search Algorithm. | | |
| 06 | Implement Depth First Search Algorithm. | | |
| 07 | Design an e-commerce chatbot using AIML. | | |
| 08 | Write a program to implement Automatic Sprinkler RBS. | | |
| 09 | Implement A* algorithm. | | |
| 10 | Design a gamebot(rock, paper, scissors) using AIML. | | |
| 11 | Design an expert system AIML for a restaurant recommendation. | | |

# Practical No: 1

**Aim: Implement a python program to show the working of a simple chatbot.**

**Code:**

```
import random
responses={
      "Hello":["Hi","Hey Whatsup"],
      "Bye":["See you soon","Have a nice day"],
      "How are you":["I'm Fine","What about you"],
      "What is your name":["Chatbot"],
      "Default":["I don't understand","I'm a simple chatbot kindly explain in detail"]
   }

def chatbot():
   while True:
      user_input=input("You: ")
      if user_input=="exit":
         print("Goodbye")
         break
      response=responses.get(user_input,responses["Default"])
      result="".join(random.choices(response))
      print("Bot: ",result)

chatbot()
```

**Output:**

```
========================= RESTART: K:/rr/msc1/aai/pras.py =
You: Hello
Bot:  Hey Whatsup
You: How are you
Bot:  I'm Fine
You: What is your name
Bot:  Chatbot
You: Goodbye
Bot:  I don't understand
You:
```

# Practical No: 2

## Aim: Implement a python program for a  simple URL shortener.

## Code:

```
import pyshorteners
def shortener_url(old_url):
    s = pyshorteners.Shortener()
    new_url = s.tinyurl.short(old_url)
    return new_url

old_url = input("Enter the url:")
result = shortener_url(old_url)
print(f"result url=",{result})
print("result url = ",result)
```

## Output:

```
>>>
========================= RESTART: K:/rr/practical2.py =========================
Enter the url:https://www.google.com/search?q=bajaj&rlz=1C1CHBF_enIN1127IN1127&oq=bajaj&gs_lcrp=EgZjaHJvbWUyBggAEE
UYOTINCAEQABiDARixAxiABDITCAIQLhiDARjHARixAxjRAxiABDIQCAMQLhjHARixAxjRAxiABDIQCAQQABiDARixAxiABBiKBTIKCAUQABixAxiA
BDIKCAYQABixAxiABDIHCAcQABiABDIQCAgQLhjHARixAxjRAxiABDIKCAkQLhixAxiABNIBCTUzNjczajBqN6gCALACAA&sourceid=chrome&ie=
UTF-8
result url= {'https://tinyurl.com/26ge73mo'}
result url =  https://tinyurl.com/26ge73mo
>>>
```

# Practical No: 3

**Aim: Implement operations of fuzzy set(union,intersection,complement and difference)**

## Code:

```
# Example to Demonstrate the Union, Intersection, Complement, and Difference Between Two Fuzzy Sets

# Initialize fuzzy sets
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 given fuzzy sets
print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)

# 1. Union of Two Fuzzy Sets (Maximum of values)
Y_union = {}
for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

    # Union takes the maximum of values
    if A_value > B_value:
        Y_union[A_key] = A_value
    else:
        Y_union[B_key] = B_value

print('Fuzzy Set Union is :', Y_union)


# 2. Intersection of Two Fuzzy Sets (Minimum of values)
Y_intersection = {}
for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]
```

```
    # Intersection takes the minimum of values
    if A_value < B_value:
        Y_intersection[A_key] = A_value
    else:
        Y_intersection[B_key] = B_value

print('Fuzzy Set Intersection is :', Y_intersection)



# 3. Complement of Fuzzy Set A (1 - value of each element)
Y_complement = {}
for A_key in A:
    Y_complement[A_key] = 1 - A[A_key]

print('Fuzzy Set Complement of A is :', Y_complement)



# 4. Difference Between Two Fuzzy Sets (A - B)
Y_difference = {}
for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]
    B_value = 1 - B_value  # Complement of B for difference operation

    # Difference takes the minimum between A_value and complement of B_value
    if A_value < B_value:
        Y_difference[A_key] = A_value
    else:
        Y_difference[B_key] = B_value

print('Fuzzy Set Difference (A - B) is :', Y_difference)
```

## Output:

```
========================================= RESTART: K:/rr/msc1/aai/pras.py ==================================
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}
Fuzzy Set Intersection is : {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
Fuzzy Set Complement of A is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
Fuzzy Set Difference (A - B) is : {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.6, 'd': 0.5}
>
```

# Practical no: 4

**Aim: Design an expert system for responding to patient queries for identifying the flu.**

**Code:**

```
def ask_question(question):
    """Ask the user a question and get a yes/no response."""
    response = input(question + " (yes/no): ").strip().lower()
    while response not in ["yes", "no"]:
        print("Please answer with 'yes' or 'no'.")
        response = input(question + " (yes/no): ").strip().lower()
    return response == "yes"

def collect_symptoms():
    """Collect symptoms from the user."""
    print("Please answer the following questions about your symptoms:")
    symptoms = {}
    symptoms["fever"] = ask_question("Do you have a fever or chills?")
    symptoms["cough"] = ask_question("Do you have a cough?")
    symptoms["sore_throat"] = ask_question("Do you have a sore throat?")
    symptoms["runny_nose"] = ask_question("Do you have a runny or stuffy
nose?")
    symptoms["body_aches"] = ask_question("Do you have muscle or body
aches?")
    symptoms["fatigue"] = ask_question("Do you feel fatigued or very tired?")
    symptoms["loss_of_taste_smell"] = ask_question("Have you lost your sense of
taste or smell?")
    return symptoms

def evaluate(symptoms):
    """Evaluate the symptoms and provide a diagnosis."""
    # Rule-based evaluation
    if symptoms["fever"] and symptoms["cough"] and symptoms["fatigue"]:
        if symptoms["loss_of_taste_smell"]:
            return "Your symptoms suggest COVID-19. Please get tested and consult a
healthcare provider."
```

```
        return "Your symptoms suggest the flu. Rest, hydrate, and consult a doctor if
symptoms worsen."
    elif symptoms["sore_throat"] and not symptoms["fever"]:
        return "Your symptoms suggest a common cold. Rest and stay hydrated."
    else:
        return "Your symptoms are inconclusive or may indicate another condition.
Please consult a healthcare provider."

def main():
    """Run the expert system."""
    print("Welcome to the Flu Expert System! Let's assess your symptoms.")
    symptoms = collect_symptoms()
    diagnosis = evaluate(symptoms)
    print("\nDiagnosis:")
    print(diagnosis)

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

**Output:**

```
>>>
= RESTART: C:/Users/KBP/AppData/Local/Programs/Python/Python311/new1.py
Welcome to the Flu Expert System! Let's assess your symptoms.
Please answer the following questions about your symptoms:
Do you have a fever or chills? (yes/no): yes
Do you have a cough? (yes/no): no
Do you have a sore throat? (yes/no): yes
Do you have a runny or stuffy nose? (yes/no): no
Do you have muscle or body aches? (yes/no): yes
Do you feel fatigued or very tired? (yes/no): no
Have you lost your sense of taste or smell? (yes/no): no

Diagnosis:
Your symptoms are inconclusive or may indicate another condition. Please consult a healthcare
provider.
>>>
```

# Practical No: 5

**Aim: Implement Breadth First Search Algorithm.**

**Code:**

```
from collections import defaultdict,deque
class Graph:
    def _init_(self):
        self.graph=defaultdict(list)
    def add_edge(self,u,v):
        self.graph[u].append(v)

    def bfs(self,start):
        visited =set()
        queue=deque([start])
        visited.add(start)

        while queue:
            vertex = queue.popleft()
            print(vertex)
            for neighbour in self.graph[vertex]:
                if neighbour is not visited:
                    visited.add(neighbour)
                    queue.append(neighbour)

g=Graph()
g.add_edge(0,1)
g.add_edge(0,2)
g.add_edge(1,4)
g.add_edge(2,3)

print("BFS:")
g.bfs(0)
```

## Output:

```
>>>
========================= RESTART: K:/rr/msc1/aai/pras.py =========================
BFS:
0
1
2
4
3
>>>
```

# Practical No: 6

## Aim: Implement Depth First Search Algorithm.

## Code:

```
from collections import defaultdict,deque
class Graph:
    def __init__(self):
        self.graph=defaultdict(list)
    def add_edge(self,u,v):
        self.graph[u].append(v)

    def dfs(self,start,visited=None):
        if visited is None:
            visited=set()
        visited.add(start)
        print(start)
        for neighbour in self.graph[start]:
            if neighbour not in visited:
                self.dfs(neighbour,visited)

g=Graph()
g.add_edge(0,1)
g.add_edge(0,2)
g.add_edge(1,4)
g.add_edge(2,3)

print("DFS:")
g.dfs(0)
```

**Output:**

```
========================= RESTART: K:/rr/msc1/aai/pras.py ==
DFS:
0
1
4
2
3
```

# Practical No: 7

## Aim: Design an e-commerce chatbot using AIML.

## Code:

```
import aiml
import os

def main():
    # Create AIML Kernel
    kernel = aiml.Kernel()
    kernel.learn("p3aiml.aiml")

    print("E-Commerce Chatbot is ready! Type 'exit' to end the chat.")

    while True:
        user_input = input("You: ")
        if user_input.upper() == "EXIT":
            print("Chatbot: Thank you for visiting! Goodbye!")
            break
        response = kernel.respond(user_input)
        print(f"Chatbot: {response}")

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

## AIML:

```
<aiml version="2.0">
    <category>
        <pattern>HELLO</pattern>
        <template>Hello! Welcome to our e-commerce store. How can I assist you
today?</template>
    </category>

    <category>
        <pattern>I WANT TO BUY *</pattern>
        <template>Great choice! We have a wide range of <star/>. Would you like some
recommendations?</template>
    </category>
```

```
  <category>
    <pattern>YES</pattern>
    <template>Can you specify your preferences? For example, color, brand, or
budget?</template>
  </category>

  <category>
    <pattern>NO</pattern>
    <template>No problem! Let me know if there's anything else I can help you
with.</template>
  </category>

  <category>
    <pattern>WHAT ARE YOUR OFFERS</pattern>
    <template>We currently have discounts on electronics and fashion items. Would you
like to explore these categories?</template>
  </category>

  <category>
    <pattern>EXIT</pattern>
    <template>Thank you for visiting our store. Have a great day!</template>
  </category>
</aiml>
```

## Output:

```
= RESTART: D:\prac3\P3.py
Loading p3aiml.aiml...done (0.59 seconds)
E-Commerce Chatbot is ready! Type 'exit' to end the chat.
You: hello
Chatbot: Hello! Welcome to our e-commerce store. How can I assist you today?
You: what are your offers
Chatbot: We currently have discounts on electronics and fashion items. Would you like to ex
plore these categories?
You: yes
Chatbot: Can you specify your preferences? For example, color, brand, or budget?
You: exit
Chatbot: Thank you for visiting! Goodbye!
```

# Practical no: 8

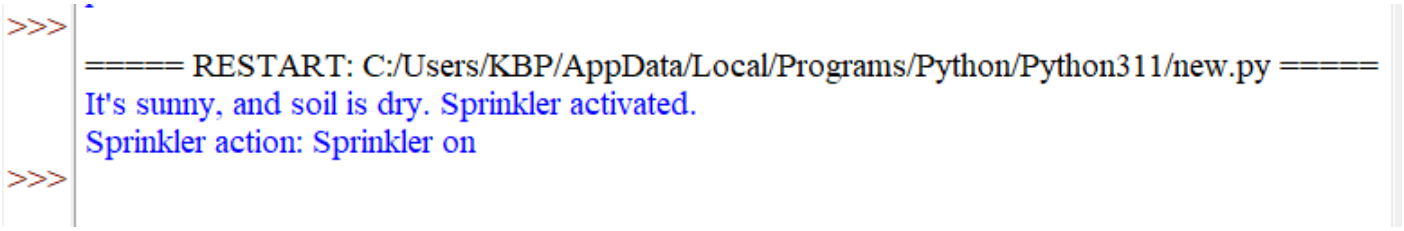
## Aim: Write a program to implement Automatic Sprinkler RBS.

## Code:

```
class AutomaticSprinkler:
    def __init__(self):
        self.soil_moisture = 0  # Soil moisture level (0 - dry, 100 - saturated)
        self.weather_condition = "sunny"  # weather condition (sunny, cloudy, rainy)
        self.time_of_day = "day"  # time of day (day or night)

    def set_soil_moisture(self, moisture_level):
        """Sets the current soil moisture level."""
        self.soil_moisture = moisture_level

    def set_weather_condition(self, condition):
        """Sets the current weather condition."""
        self.weather_condition = condition

    def set_time_of_day(self, time_of_day):
        """Sets the current time of day."""
        self.time_of_day = time_of_day

    def decide_sprinkler_action(self):
        """Decides whether to activate the sprinkler based on RBS rules."""
        if self.weather_condition == "rainy":
            print("It's raining. No need to water the plants.")
            return "No action"

        if self.weather_condition == "cloudy":
            if self.soil_moisture < 50:
                print("It's cloudy, but soil is dry. Sprinkler activated.")
                return "Sprinkler on"
            else:
                print("It's cloudy, and soil moisture is sufficient. No need to water.")
                return "No action"

        if self.weather_condition == "sunny":
            if self.soil_moisture < 30 and self.time_of_day == "day":
                print("It's sunny, and soil is dry. Sprinkler activated.")
```

```
            return "Sprinkler on"
        elif self.soil_moisture >= 30 and self.time_of_day == "day":
            print("It's sunny, but soil moisture is sufficient. No need to water.")
            return "No action"
        elif self.time_of_day == "night":
            print("It's night, no watering needed.")
            return "No action"

    return "No action"

# Example usage
sprinkler = AutomaticSprinkler()

# Set conditions
sprinkler.set_soil_moisture(25)  # Soil moisture level
sprinkler.set_weather_condition("sunny")  # Weather condition
sprinkler.set_time_of_day("day")  # Time of day

# Check if the sprinkler needs to be activated
sprinkler_action = sprinkler.decide_sprinkler_action()
print(f"Sprinkler action: {sprinkler_action}")
```

## Output:



# Practical No: 9

## Aim: Implement A* algorithm.

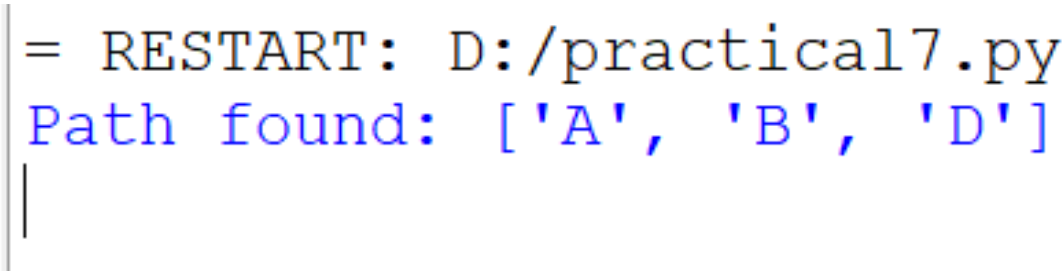
## Code:

```
import heapq

class Node:
    def __init__(self, name, parent=None, g=0, h=0):
        self.name = name
        self.parent = parent
        self.g = g  # cost from start to node
        self.h = h  # heuristic estimated cost from node to goal
        self.f = g + h  # total cost (f = g + h)

    def __lt__(self, other):
        # This ensures that heapq can compare nodes based on their f value
        return self.f < other.f

def a_star_algorithm(start, goal, graph, heuristic):
    open_list = []
    closed_list = set()

    # Create the start node
    start_node = Node(start, None, 0, heuristic[start])
    heapq.heappush(open_list, start_node)
    i=0
    while open_list:
        current_node = heapq.heappop(open_list)
        #print(current_node.name)
        # If we reach the goal, reconstruct the path
        if current_node.name == goal:
            path = []
            while current_node:
                path.append(current_node.name)
                current_node = current_node.parent
            return path[::-1]  # Reverse the path to get the correct order

        closed_list.add(current_node.name)
```

```
        # Explore the neighbors
        for neighbor, cost in graph[current_node.name]:
            #print(neighbor,cost)
            if neighbor in closed_list:
                continue
            #print(current_node.g,"  ",cost)
            g_cost = current_node.g + cost
            #print(g_cost)
            h_cost = heuristic[neighbor]
            #print(h_cost)
            neighbor_node = Node(neighbor, current_node, g_cost, h_cost)

            # Check if the neighbor is already in the open list
            if not any(open_node.name == neighbor and open_node.f <= neighbor_node.f for
open_node in open_list):
                heapq.heappush(open_list, neighbor_node)

    return None  # No path found

# Example graph with costs (adjacency list representation)
graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 2)],
    'C': [('A', 3), ('D', 1)],
    'D': [('B', 2), ('C', 1)],
}

# Heuristic values (straight-line distance to goal)
heuristic = {
    'A': 4,
    'B': 2,
    'C': 1,
    'D': 0,
}

start = 'A'
goal = 'D'

# Find the path from start to goal
path = a_star_algorithm(start, goal, graph, heuristic)
```

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

**Output:**



# Practical No: 10

**Aim: Design a gamebot(rock, paper, scissors) using AIML.**

**Code:**

```
import aiml

def gamePlay():
    # Create a Kernel instance
    kernel = aiml.Kernel()

    # Load the AIML file
    kernel.learn("p10aiml.aiml")

    print("Gamebot: Hi there! Type 'Rock', 'Paper', or 'Scissors' to play. Type 
'Goodbye' to exit.")

    # Start the game loop
    while True:
        user_input = input("You: ").strip().upper()
        if user_input == "GOODBYE":
            print("Gamebot: Goodbye! Thanks for playing.")
            break
        elif user_input in ["ROCK", "PAPER", "SCISSORS"]:
            response = kernel.respond(user_input)
            print(f"Gamebot: {response}")
        else:
            print("Gamebot: I didn't understand that. Please type 'Rock', 'Paper', or 
'Scissors'.")

gamePlay()
```

## Output:

```
Loading p10aiml.aiml...done (0.00 seconds)
Gamebot: Hi there! Type 'Rock', 'Paper', or 'Scissors' to play. Type 'Goodbye' to exit.
You: Rock
Gamebot: Paper! I win! Let's play again. Type Rock, Paper, or Scissors.
You: Paper
Gamebot: Scissors! I win! Let's play again. Type Rock, Paper, or Scissors.
You: Scissors
Gamebot: Rock! I win! Let's play again. Type Rock, Paper, or Scissors.
You: Rock
Gamebot: Paper! I win! Let's play again. Type Rock, Paper, or Scissors.
You: Paper
Gamebot: Scissors! I win! Let's play again. Type Rock, Paper, or Scissors.
You: Scissors
Gamebot: Scissors! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Paper! I win! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Rock! It's a tie! Let's play again. Type Rock, Paper, or Scissors.
You: rock
Gamebot: Scissors! You win! Let's play again. Type Rock, Paper, or Scissors.
```

# Practical No: 11

## Aim: Design an expert system AIML for a restaurant recommendation.

## Code:

```
import aiml
import os

# Initialize AIML kernel
kernel = aiml.Kernel()

# Load specific AIML files
kernel.learn('path_to_aiml_files/restaurant_recommendation.aiml')  # Replace with your specific AIML file path
kernel.learn('p11aiml.aiml')  # Replace with your specific AIML file path

# Start the conversation
print("Welcome to the restaurant recommendation bot! Type 'goodbye' to exit.")

while True:
    # Get user input
    user_input = input("You: ").strip()

    # Handle exit commands
    if user_input.lower() in ['exit', 'quit', 'goodbye']:
        print("Bot: Goodbye! Enjoy your meal!")
        break

    # Get and print bot's response
    response = kernel.respond(user_input)

    # If the bot doesn't have a response, prompt for more details
    if not response:
        response = "Sorry, I didn't understand that. Can you tell me what kind of cuisine you're interested in?"

    print(f"Bot: {response}")
```

**AIML:**

```
<aiml version="2.0">
   <!-- Basic greeting -->
   <category>
      <pattern>HELLO</pattern>
      <template>Hi there! I'm your restaurant guide. How can I help you
today?</template>
   </category>

   <!-- Ask for help -->
   <category>
      <pattern>HELP</pattern>
      <template>If you need a restaurant suggestion, just tell me what type of cuisine
you're craving!</template>
   </category>

   <!-- Goodbye -->
   <category>
      <pattern>GOODBYE</pattern>
      <template>Goodbye! Have a great meal!</template>
   </category>
</aiml>
```

**Botaiml.aiml**

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8">
   <category>
      <pattern>RECOMMEND A RESTAURANT</pattern>
      <template>What type of cuisine do you prefer?</template>
   </category>

   <category>
      <pattern>I LIKE * FOOD</pattern>
      <template>
         <think><set name="cuisine"><star/></set></think>
         Great choice! Do you prefer casual or fine dining?
      </template>
   </category>

   <category>
```

```
    <pattern>* DINING</pattern>
    <template>
      Based on your preference for <get name="cuisine"/> food and <star/> dining, I recommend trying a place like:
      <condition name="cuisine">
        <li value="Italian">Luigi's Trattoria</li>
        <li value="Chinese">Golden Dragon</li>
        <li value="Mexican">El Rancho</li>
        <li value="Indian">Spice House</li>
        <li value="Japanese">Sakura Sushi</li>
        <li value="American">Burger Haven</li>
        <li value="French">Le Petit Bistro</li>
        <li value="Thai">Thai Orchid</li>
        <li value="Mediterranean">Olive Grove</li>
        <li value="Korean">Seoul BBQ</li>
        <li value="Greek">Santorini Grill</li>
        <li value="Vegan">Green Earth Café</li>
      </condition>
    </template>
  </category>
</aiml>
```

## Output:

```
>>>
========================== RESTART: K:\rr\aai\p2.py ==========================
Loading botaiml.aiml...done (0.04 seconds)
Loading p11aiml.aiml...done (0.00 seconds)
Welcome to the restaurant recommendation bot! Type 'goodbye' to exit.
You: hello
Bot: Hi there! I'm your restaurant guide. How can I help you today?
You: help
Bot: If you need a restaurant suggestion, just tell me what type of cuisine you're craving!
You: recommend a restaurant
Bot: What type of cuisine do you prefer?
You: i like indian food
Bot: Great choice! Do you prefer casual or fine dining?
You:
```