SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

III B TECH – INFORMATION TECHNOLOGY

23ADC01 - ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY

PRACTICAL RECORD

Submitted by

Name : ……

Reg. No : ……

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

23ADC01 - ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY

PRACTICAL RECORD

Name : …… **Reg.no :……..……**

Class : …… **Semester :**

BONAFIDE CERTIFICATE

Certified Bonafide record of work done by Mr./Ms. ----------

Reg,No. ---------- during the ODD Semester of Academic Year 2025-2026.

Staff-In Charge HOD

Submitted for the end semester practical examination held on ----------

Internal Examiner External Examiner

INDEX

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
An Autonomous Institution | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with A++ Grade
Kuniamuthur, Coimbatore – 641008
Phone : (0422)-2678001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

Register Number : **Name of the Student :**

Name of the lab : 23ADC01- Artificial Intelligence and its Applications Laboratory

Components	Exp No and Date												Average Score
	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7	Ex8	Ex9	Ex10	Ex11	Ex12	
Aim & Algorithm 20 Marks													
Coding 30 Marks													
Compilation & Debugging 30 Marks													
Execution & Results 10 Marks													
Documentation & Viva 10 Marks													
Total													

Staff In-charge

Exp. No.: 1

Date:

Implement a simple AI -based Python script

Sentiment Analysis using TextBlob

Aim:

To implement a simple AI-based sentiment analysis program using the TextBlob library in Python to classify a sentence or paragraph as Positive, Negative, or Neutral based on its polarity.

Algorithm:

1. 1 Start the program.
2. Import the required library: TextBlob.
3. Accept user input (a sentence or paragraph).
4. Create a TextBlob object using the input text.
5. Use the .sentiment.polarity property to compute the sentiment score.
6. Determine the sentiment:
    a. If polarity > 0 → Positive
    b. If polarity < 0 → Negative
    c. If polarity = 0 → Neutral
7. Display the sentiment and polarity score.
8. End the program.
    a. If polarity > 0 → Positive
    b. If polarity < 0 → Negative
    c. If polarity = 0 → Neutral
7. Display the sentiment and polarity score.
8. End the program

Program:

```
# Importing the required library
from textblob import TextBlob
# Welcome message
print("Welcome to the AI Sentiment Analysis Lab!")
print("This program analyzes the sentiment of your text (positive, neutral, or negative).")
# Input from the user
user_input = input("\nEnter a sentence or paragraph to analyze: ")
# Create a TextBlob object
analysis = TextBlob(user_input)
# Analyze sentiment polarity
# Polarity ranges from -1 (negative) to 1 (positive)
polarity = analysis.sentiment.polarity
# Determine sentiment category
if polarity > 0:
    sentiment = "Positive"
elif polarity < 0:
    sentiment = "Negative"
else:
    sentiment = "Neutral"
# Display the result
print("\nSentiment Analysis Result:")
print(f"Polarity Score: {polarity}")
print(f"Sentiment: {sentiment}")
# Goodbye message
print("\nThank you for using the AI Sentiment Analysis Lab!")
```

Output:

```
Welcome to the AI Sentiment Analysis Lab!
This program analyzes the sentiment of your text (positive, neutral, or negative).

Enter a sentence or paragraph to analyze: I love programming, it makes me happy!

Sentiment Analysis Result:
Polarity Score: 0.75
Sentiment: Positive

Thank you for using the AI Sentiment Analysis Lab!
```

Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 2

Date:

Exploring AI Applications in Real-World Case Studies

Aim:

To explore the role of Artificial Intelligence (specifically ChatGPT) in supporting students during real-world physics experiments, focusing on its effectiveness in problem-solving, data interpretation, and conceptual understanding under teacher supervision.

Algorithm:

1. Start
2. Form student teams (7 groups).
3. Setup the acoustic levitation apparatus for measuring the speed of sound.
4. Integrate AI (ChatGPT) to assist students with:
   - Formula guidance
   - Step-by-step procedures
   - Error correction
   - Data analysis
5. Teachers monitor AI-student interaction, noting accuracy and effectiveness.
6. Conduct a reflection session: students fill surveys and discuss AI's role.
7. End

Program:

```
from textblob import TextBlob
def analyze_feedback(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0:
        return "Positive feedback on AI assistance"
    elif polarity < 0:
```

```
        return "Negative feedback on AI assistance"
    else:
        return "Neutral feedback on AI assistance"
feedback = input("Enter your experience with AI in the experiment: \n")
print(analyze_feedback(feedback))
```

Output:

```
Enter your experience with AI in the experiment:
AI helped me understand the formulas clearly and corrected my mistakes.
Positive feedback on AI assistance
```

Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 3

Date:

Implementing BFS and DFS in Python

Aim:

To implement and demonstrate Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms in Python for solving AI search problems using graph traversal.

Procedure:

1. Define a graph as a dictionary (adjacency list).
2. Implement bfs (start, goal) using a queue.
3. Implement dfs (start, goal) using a stack.
4. Provide a start node and a goal node.
5. Call both functions and observe the path to the goal node.
6. Display the visiting order and final path.

Algorithm:

i) Breadth-First Search (BFS):

1. Start the program.
2. Create a queue and add the start node as a path (in a list).
3. Initialize an empty set to store visited nodes.
4. Repeat until the queue is empty:

a. Remove the first path from the queue.

b. Get the last node from the path.

c. If the node is the goal, display the path and exit.

d. If the node is not visited:

- Mark it as visited.
- For each neighbor of the node:

o Create a new path with the neighbor.

o Add the new path to the queue.

5. End the program.

ii) Depth-First Search (DFS):

1. Start the program.
2. Create a stack and add the start node as a path (in a list).
3. Initialize an empty set to store visited nodes.
4. Repeat until the stack is empty:

a. Pop the top path from the stack.

b. Get the last node from the path.

c. If the node is the goal, display the path and exit.

d. If the node is not visited:

o Mark it as visited.

o For each neighbor of the node (in reverse order):

- Create a new path with the neighbor.
- Add the new path to the stack.

5. End the program.

Program:

```
from collections import deque
graph = {
    'S': ['A', 'B'],
    'A': ['C', 'D'],
    'B': ['E'],
    'C': [],
    'D': ['G'],
    'E': ['G'],
    'G': []
}
def bfs(start, goal):
    visited = set()
```

```
    queue = deque([[start]])
    print("BFS Path:")
    while queue:
        path = queue.popleft()
        node = path[-1]
        if node not in visited:
            print("Visiting:", node)
            visited.add(node)
            if node == goal:
                print("Goal found! Path:", ' -> '.join(path))
                return
            for neighbor in graph[node]:
                new_path = list(path)
                new_path.append(neighbor)
                queue.append(new_path)
    print("Goal not found in BFS.")
def dfs(start, goal):
    visited = set()
    stack = [[start]]
    print("\nDFS Path:")
    while stack:
        path = stack.pop()
        node = path[-1]
        if node not in visited:
            print("Visiting:", node)
            visited.add(node)
            if node == goal:
                print("Goal found! Path:", ' -> '.join(path))
                return
            for neighbor in reversed(graph[node]):
```

```
            new_path = list(path)
            new_path.append(neighbor)
            stack.append(new_path)
    print("Goal not found in DFS.")
start_node = 'S'
goal_node = 'G'
print("AI Graph Search using BFS and DFS")
bfs(start_node, goal_node)
dfs(start_node, goal_node)
```

Output:

```
AI Graph Search using BFS and DFS
BFS Path:
Visiting: S
Visiting: A
Visiting: B
Visiting: C
Visiting: D
Visiting: E
Visiting: G
Goal found! Path: S -> A -> D -> G

DFS Path:
Visiting: S
Visiting: A
Visiting: C
Visiting: D
Visiting: G
Goal found! Path: S -> A -> D -> G
```

Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 4

Date:

Implementing A* and Greedy Best-First Search in Python

Aim:

To implement and demonstrate A* and Greedy Best-First Search algorithms in Python to find the shortest path in a grid using heuristic search.

Algorithm:

i) A* Search:

1. Start the program.
2. Create a **priority queue** and add the start node with cost f(n) = g(n) + h(n).
3. Initialize g(start) = 0 and an empty set for visited nodes.
4. Repeat until the queue is empty:
   a. Remove the node with the **lowest f(n)**.
   b. If it is the goal node, display the path and exit.
   c. If not visited:
      - Mark as visited.
      - For each neighbor:
        - Calculate g(neighbor) = g(current) + 1.
        - Calculate f(neighbor) = g(neighbor) + h(neighbor).
        - Add neighbor to the queue.
5. End the program

ii) Greedy Best-First Search:

1. Start the program.
2. Create a **priority queue** and add the start node with cost f(n) = h(n).
3. Initialize an empty set for visited nodes.
4. Repeat until the queue is empty:
   a. Remove the node with the **lowest h(n)**.

b. If it is the goal node, display the path and exit.

c. If not visited:

- Mark as visited.
- For each neighbor:
  - Add neighbor to the queue with priority h(neighbor).

5. End the program.

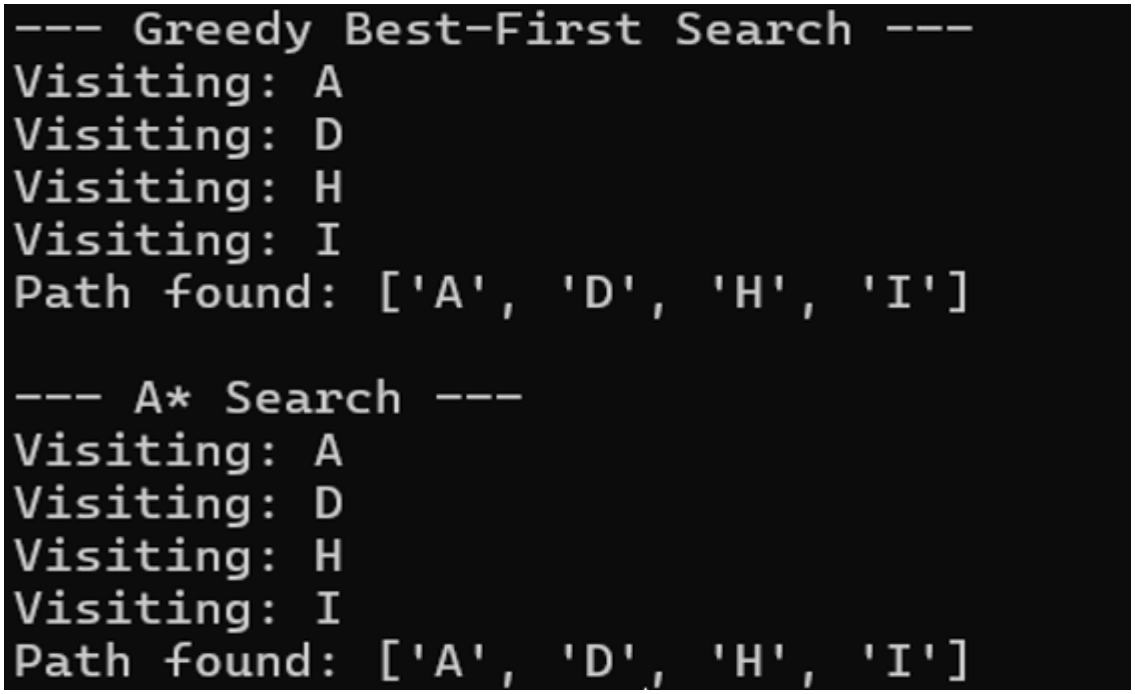
Program:

```
import heapq
# Define the graph (adjacency list)
graph = {
    'A': ['B', 'D'],
    'B': ['E', 'G'],
    'D': ['G', 'H'],
    'G': ['I'],
    'H': ['I'],
    'E': [],
    'I': []
}
# Define heuristic values (h(n))
heuristic = {
    'A': 999,
    'B': 5,
    'D': 2,
    'E': 4,
    'G': 6,
    'H': 1,
    'I': 0
}
# Uniform edge cost for A*
```

```
edge_cost = 1
# --- Greedy Best-First Search ---
def greedy_bfs(start, goal):
    print("--- Greedy Best-First Search ---")
    visited = set()
    pq = [(heuristic[start], start)]
    path = []
    while pq:
        _, current = heapq.heappop(pq)
        if current in visited:
            continue
        print("Visiting:", current)
        visited.add(current)
        path.append(current)
        if current == goal:
            print("Path found:", path)
            return
        for neighbor in graph[current]:
            if neighbor not in visited:
                heapq.heappush(pq, (heuristic[neighbor], neighbor))
    print("Goal not found using Greedy BFS.")
# --- A* Search ---
def a_star(start, goal):
    print("\n--- A* Search ---")
    visited = set()
    pq = [(heuristic[start], 0, start, [])]  # (f(n), g(n), node, path)
    while pq:
        f, g, current, path = heapq.heappop(pq)
        if current in visited:
            continue
```

```
        print("Visiting:", current)
        visited.add(current)
        path = path + [current]
        if current == goal:
            print("Path found:", path)
            return
        for neighbor in graph[current]:
            if neighbor not in visited:
                new_g = g + edge_cost
                new_f = new_g + heuristic[neighbor]
                heapq.heappush(pq, (new_f, new_g, neighbor, path))
    print("Goal not found using A*.")
# Function Calls
start_node = 'A'
goal_node = 'I'
greedy_bfs(start_node, goal_node)
a_star(start_node, goal_node)
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 5

Date:

Implementation of a Simple Tic-Tac-Toe Game in Python

Aim:

To design and implement a simple two-player Tic-Tac-Toe game in Python using lists and functions.

Algorithm:

1. Start the program.
2. Initialize the game board as a list with 9 positions marked as "-".
3. Define a function print_board() and print the board in a 3×3 grid format.
4. Define a function take_turn(player):
   a. Display the current player's turn.
   b. Ask the player to choose a position from 1–9.
   c. Validate the input until a correct number is entered.
   d. If the position is already filled, ask again.
   e. Place the player's symbol ("X" or "O") in the chosen position.
   f. Display the updated board.
5. Define a function check_game_over():
   a. Check all rows, columns, and diagonals for a win.
   b. If a win condition is found → return "win".
   c. Else, if no "-" remains on the board → return "tie".
   d. Otherwise, return "play".
6. Define the main function play_game():
   a. Display the initial empty board.
   b. Set the current_player as "X".
   c. Repeat until the game is over:
      i. Call take_turn(current_player).
      ii. Call check_game_over() to check the game status.

iii. If "win", announce winner and end the loop.

iv. If "tie", display tie message and end the loop.

v. Otherwise, switch player ("X" ↔ "O") and continue.

7. Call play_game() to start the game.

8. End the program.

Program:

```
# Set up the game board
board = ["-", "-", "-",
         "-", "-", "-",
         "-", "-", "-"]
# Print the game board
def print_board():
    print(board[0] + " | " + board[1] + " | " + board[2])
    print(board[3] + " | " + board[4] + " | " + board[5])
    print(board[6] + " | " + board[7] + " | " + board[8])
# Handle a player's turn
def take_turn(player):
    print(player + "'s turn.")
    position = input("Choose a position from 1-9: ")
    while position not in ["1","2","3","4","5","6","7","8","9"]:
        position = input("Invalid input. Choose a position from 1-9: ")
    position = int(position) - 1
    while board[position] != "-":
        position = int(input("Position already taken. Choose a different position: ")) - 1
    board[position] = player
    print_board()
# Check if the game is over
def check_game_over():
    if (board[0] == board[1] == board[2] != "-") or \
```

```
        (board[3] == board[4] == board[5] != "-") or \
        (board[6] == board[7] == board[8] != "-") or \
        (board[0] == board[3] == board[6] != "-") or \
        (board[1] == board[4] == board[7] != "-") or \
        (board[2] == board[5] == board[8] != "-") or \
        (board[0] == board[4] == board[8] != "-") or \
        (board[2] == board[4] == board[6] != "-"):
        return "win"
    elif "-" not in board:
        return "tie"
    else:
        return "play"
# Main game loop
def play_game():
    print_board()
    current_player = "X"
    game_over = False
    while not game_over:
        take_turn(current_player)
        game_result = check_game_over()
        if game_result == "win":
            print(current_player + " wins!")
            game_over = True
        elif game_result == "tie":
            print("It's a tie!")
            game_over = True
        else:
            current_player = "O" if current_player == "X" else "X"
# Start the game
play_game()
```

Output:

```
- | - | -
- | - | -
- | - | -
X's turn.
Choose a position from 1-9: 1
X | - | -
- | - | -
- | - | -
O's turn.
Choose a position from 1-9: 5
X | - | -
- | O | -
- | - | -
X's turn.
Choose a position from 1-9: 2
X | X | -
- | O | -
- | - | -
O's turn.
Choose a position from 1-9: 9
X | X | -
- | O | -
- | - | O
X's turn.
Choose a position from 1-9: 3
X | X | X
- | O | -
- | - | O
X wins!
```

Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 6

Date:

Implementing a Basic Propositional Logic Reasoning System

Aim:

To implement a basic propositional logic reasoning system in Python that can evaluate logical expressions and determine their truth values based on given facts (truth assignments).

Algorithm:

i) Manual Evaluation:

1. Start the program.
2. Define propositional variables and assign them truth values (e.g., P = True, Q = False, R = True).
3. Define functions for logical operations:
   a. Implication (→) using (not P) or Q.
   b. Biconditional (↔) using (P and Q) or (¬P and ¬Q).
4. Evaluate the following logical expressions:
   a. $P \land Q$
   b. $P \lor Q$
   c. $\neg Q$
   d. $P \rightarrow Q$
   e. $P \leftrightarrow R$
5. Display the results of each expression.
6. End the program.

ii) Using sympy library:

1. Start the program.
2. Import the required library: sympy.
3. Define propositional symbols (e.g., P, Q).
4. Create a knowledge base (KB) using logical expressions (e.g., $(P \rightarrow Q) \land P$).
5. Define the query (e.g., Q).
6. Check entailment by verifying if $KB \land \neg Q$ is unsatisfiable using satisfiable().

7. Display whether KB entails the query.
8. End the program.

Program:

i) Manual Evaluation:

```
# Define implication and biconditional functions
def implies(p, q):
    return (not p) or q
def iff(p, q):
    return (p and q) or (not p and not q)
# Step 1: Define truth values of propositions
propositions = {
    'P': True,
    'Q': False,
    'R': True
}
# Step 2: Evaluate various logical expressions
def evaluate_expressions():
    print("\n--- Propositional Logic Evaluation ---")
    # Expression 1: P ∧ Q
    result1 = propositions['P'] and propositions['Q']
    print("P ∧ Q =", result1)
    # Expression 2: P ∨ Q
    result2 = propositions['P'] or propositions['Q']
    print("P ∨ Q =", result2)
    # Expression 3: ¬Q
    result3 = not propositions['Q']
    print("¬Q =", result3)
    # Expression 4: P → Q
    result4 = implies(propositions['P'], propositions['Q'])
```

```
    print("P → Q =", result4)
    # Expression 5: P ↔ R
    result5 = iff(propositions['P'], propositions['R'])
    print("P ↔ R =", result5)
# Step 3: Call the evaluator
evaluate_expressions()
```

ii) Using sympy library:

```
from sympy import symbols
from sympy.logic.boolalg import Implies, And, Not, satisfiable
# Step 1: Define propositional symbols
P, Q = symbols('P Q')
# Step 2: Knowledge base (KB): (P → Q) ∧ P
kb = And(Implies(P, Q), P)
# Step 3: Query: Q
query = Q
# Step 4: Check if KB ⊨ Q
result = not satisfiable(And(kb, Not(query)))
# Step 5: Output
print(f"Does KB entail Q? {result}")
```

Output:

i) Manual Evaluation:

```
--- Propositional Logic Evaluation ---
P ∧ Q = False
P ∨ Q = True
¬Q = True
P → Q = False
P ↔ R = True
```

ii) Using sympy library:

```
Does KB entail Q? True
```

Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 7

Date:

Build a Simple Knowledge Base Using First-Order Logic

Aim:

To build a simple knowledge base using First-Order Logic (FOL), store facts and rules, and process user queries to derive intelligent conclusions using Python.

Algorithm:

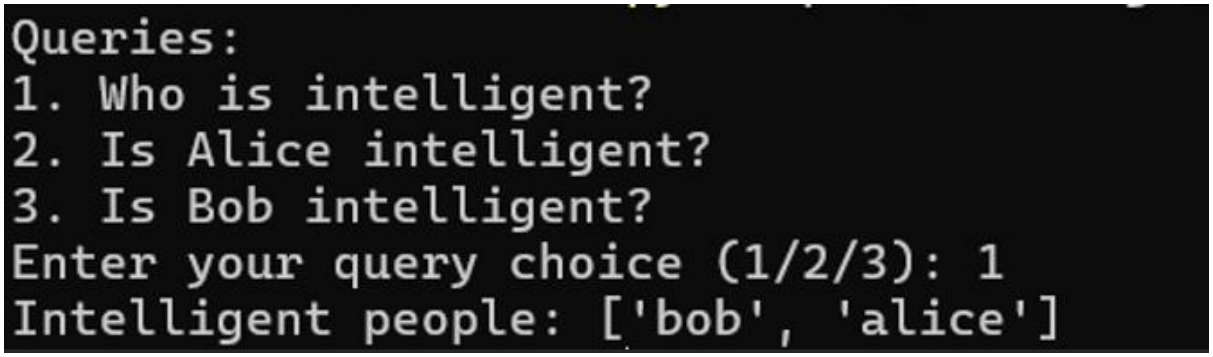
1. Import required modules from the **kanren** library: Relation, facts, run, var, and conde.
2. Define relations such as student and likes.
3. Insert facts into the knowledge base using facts().
    - Example: Alice and Bob are students; Alice likes Math, Bob likes Science.
4. Define a rule:
    - *If a person is a student and likes something, then they are intelligent.*
5. Accept user queries such as:
    - "Who is intelligent?"
    - "Is Alice intelligent?"
    - "Is Bob intelligent?"
6. Use the run() function and the defined rules to reason over the knowledge base.
7. Display the results according to the query.

Program:

```
from kanren import Relation, facts, run, var, conde
student = Relation()
likes = Relation()
facts(student, ("alice",), ("bob",))
facts(likes, ("alice", "math"), ("bob", "science"))
def is_intelligent(person):
    return conde((student(person), likes(person, var())),)
```

```
x = var()
print("Queries:")
print("1. Who is intelligent?")
print("2. Is Alice intelligent?")
print("3. Is Bob intelligent?")
choice = input("Enter your query choice (1/2/3): ")
if choice == "1":
    result = run(0, x, is_intelligent(x))
    print("Intelligent people:", list(result))
elif choice == "2":
    result = run(0, x, is_intelligent("alice"))
    if result:
        print("Yes, Alice is intelligent.")
    else:
        print("No, Alice is not intelligent.")
elif choice == "3":
    result = run(0, x, is_intelligent("bob"))
    if result:
        print("Yes, Bob is intelligent.")
    else:
        print("No, Bob is not intelligent.")
else:
    print("Invalid query!")
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 8

Date:

Create a Semantic Network to Represent Relationships Between Objects

Aim:

To construct a semantic network using Python and visualize the relationships (is-a, has-a) between objects such as Computer, CPU, Monitor, etc.

Algorithm:

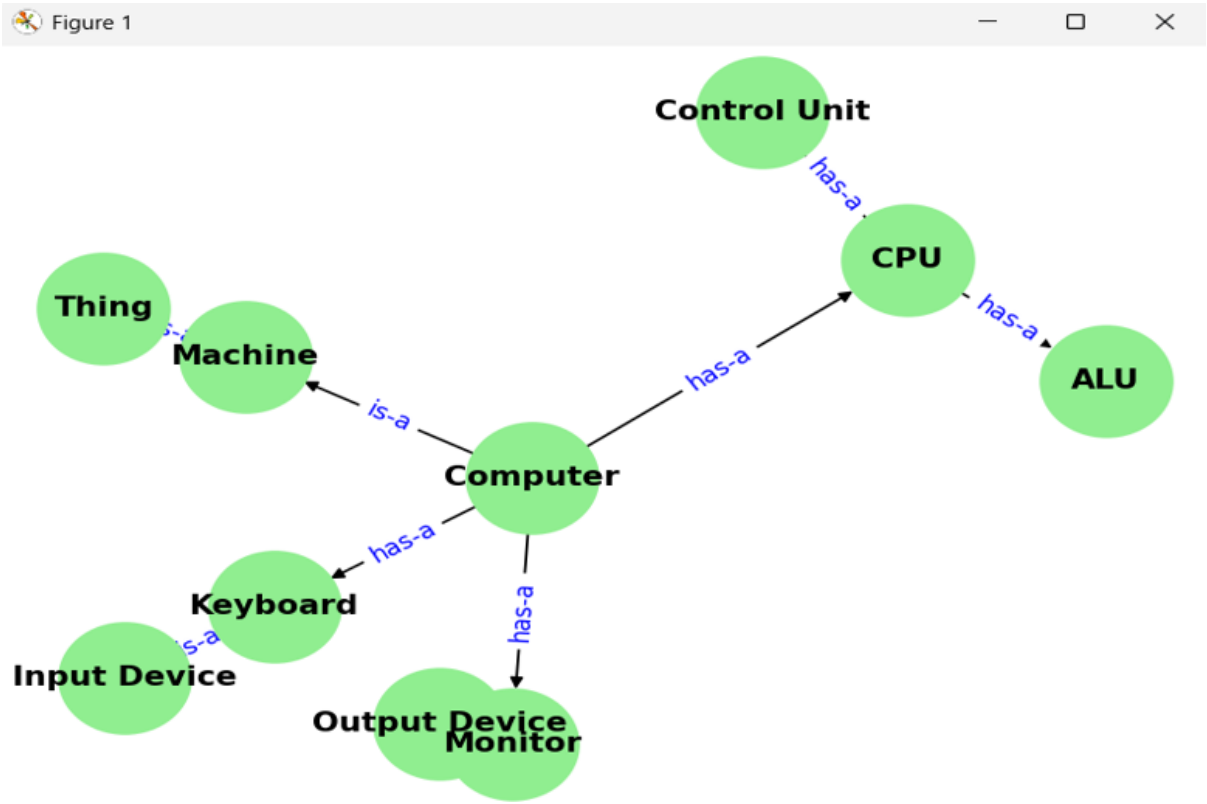
1. Import required libraries:
    - networkx for graph creation.
    - matplotlib.pyplot for visualization.
2. Create a directed graph using nx.DiGraph().
3. Add nodes and edges representing relationships between objects.
    - Example: “Computer is-a Machine”, “Computer has-a CPU”.
4. Define edge labels such as “is-a” and “has-a”.
5. Use spring_layout() to position the graph nodes.
6. Draw the graph with nodes, edges, and labels using nx.draw() and nx.draw_networkx_edge_labels().
7. Add a title, hide axes, and display the semantic network using plt.show().

Program:

```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
G.add_edges_from([
    ("Computer", "Machine", {"label": "is-a"}),
    ("Computer", "CPU", {"label": "has-a"}),
    ("CPU", "ALU", {"label": "has-a"}),
    ("CPU", "Control Unit", {"label": "has-a"}),
```

```
    ("Computer", "Monitor", {"label": "has-a"}),
    ("Monitor", "Output Device", {"label": "is-a"}),
    ("Computer", "Keyboard", {"label": "has-a"}),
    ("Keyboard", "Input Device", {"label": "is-a"}),
    ("Machine", "Thing", {"label": "is-a"})
])
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightgreen', node_size=2500,
        font_weight='bold', arrows=True)
nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attributes(G, 'label'),
                             font_color='blue')
plt.axis('off')
plt.title("Semantic Network")
plt.show()
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 9

Date:

Knowledge-Based Expert System for Medical Diagnosis

Aim:

To build a simple knowledge-based expert system in Python that uses rules for diagnosis and provides possible diseases based on user symptoms.

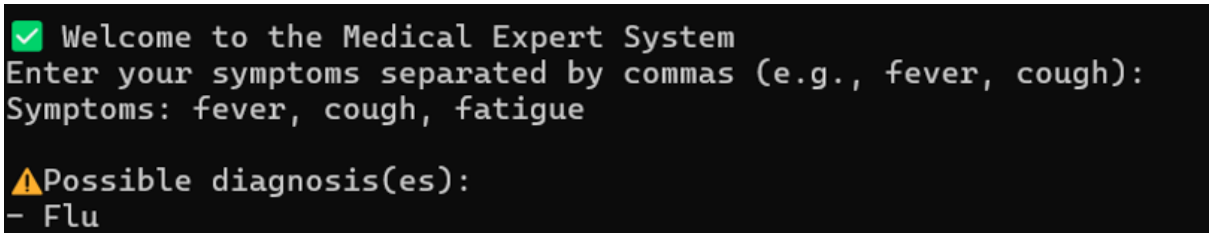
Algorithm:

1. Define the Knowledge Base:
   - Store medical rules in the format {if: conditions, then: diagnosis}.
   - Each rule contains a set of symptoms that indicate a specific disease.
2. Design the Inference Engine:
   - Accept symptoms as input from the user.
   - Convert the symptoms list into a set for easy comparison.
   - Check each rule in the knowledge base.
   - If all symptoms in a rule's "if" part are present in the user's symptoms, add the corresponding disease to the matched results.
3. User Interface:
   - Prompt the user to enter symptoms separated by commas.
   - Pass the input to the inference engine for processing.
   - Collect the list of possible diagnoses.
4. Display Results:
   - If one or more diseases match, print the possible diagnoses.
   - Otherwise, display a message advising the user to consult a doctor.
5. End Execution.

Program:

```
rules = [
    {"if": {"fever", "cough", "fatigue"}, "then": "Flu"},
    {"if": {"fever", "cough", "shortness of breath"}, "then": "COVID-19"},
    {"if": {"headache", "nausea", "sensitivity to light"}, "then": "Migraine"},
    {"if": {"sore throat", "runny nose", "sneezing"}, "then": "Common Cold"},
    {"if": {"chest pain", "shortness of breath", "dizziness"}, "then": "Heart Attack"},
]
def diagnose(symptoms):
    matched_diseases = []
    symptoms_set = set(symptoms)
    for rule in rules:
        if rule["if"].issubset(symptoms_set):
            matched_diseases.append(rule["then"])
    return matched_diseases
def main():
    print("\nWelcome to the Medical Expert System")
    print("Enter your symptoms separated by commas (e.g., fever, cough):")
    user_input = input("Symptoms: ").lower()
    symptoms = [s.strip() for s in user_input.split(",")]
    diagnoses = diagnose(symptoms)
    if diagnoses:
        print("\n⚠ Possible diagnosis(es):")
        for disease in diagnoses:
            print(f"- {disease}")
    else:
        print("\nNo matching diagnosis found. Please consult a doctor.")
if __name__ == "__main__":
    main()
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 10

Date:

Tokenization, Stemming, and Stop-word Removal using NLTK

Aim:

To implement natural language preprocessing techniques such as tokenization, stop-word removal, and stemming on sample text using Python NLTK library.

Algorithm:

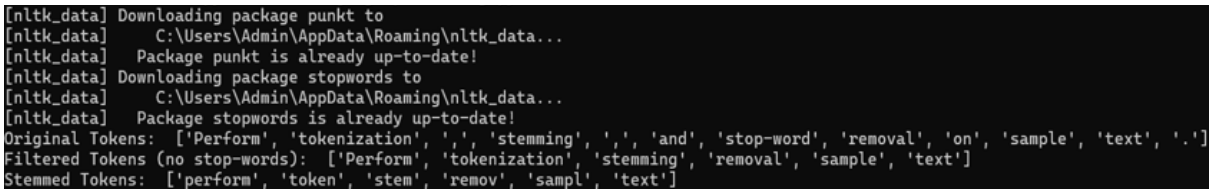
1. Import the required libraries from NLTK.
2. Download the necessary resources (punkt and stopwords).
3. Define a sample text for preprocessing.
4. Perform **Tokenization** using word_tokenize() to split text into words.
5. Remove **Stop-words** and non-alphabetic tokens from the tokenized list.
6. Apply **Stemming** using PorterStemmer to reduce words to their root form.
7. Print the original tokens, filtered tokens, and stemmed tokens.

Program:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')
text = """Perform tokenization, stemming, and stop-word removal on sample text."""
tokens = word_tokenize(text)
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words and word.isalpha()]
stemmer = PorterStemmer()
```

```
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
print("Original Tokens: ", tokens)
print("Filtered Tokens (no stop-words): ", filtered_tokens)
print("Stemmed Tokens: ", stemmed_tokens)
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 11

Date:

Implementation of Speech-to-Text System using Python Speech Recognition Library

Aim:

To implement a speech-to-text system in Python that captures audio from a microphone and converts it into text using the SpeechRecognition library.

Algorithm:

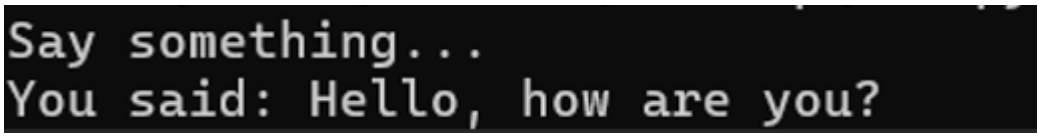
1. Import the speech_recognition library.
2. Create a Recognizer object.
3. Access the default microphone as the input source.
4. Adjust for ambient noise to improve accuracy.
5. Listen to the user's speech through the microphone.
6. Send the recorded audio to Google's Speech Recognition API.
7. If recognized successfully, display the converted text.
8. If speech is not clear or API fails, show an error message.

Program:

```
import speech_recognition as sr
def recognize_speech_from_mic():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Adjusting for ambient noise... Please wait.")
        recognizer.adjust_for_ambient_noise(source, duration=1)
        print("Listening... Speak now.")
        audio = recognizer.listen(source)
    try:
        print("Recognizing speech...")
```

```
        text = recognizer.recognize_google(audio)
        print("You said:", text)
    except sr.UnknownValueError:
        print("Sorry, I could not understand the audio.")
    except sr.RequestError as e:
        print(f"Could not request results; {e}")
if __name__ == "__main__":
    recognize_speech_from_mic()
```

Output:



Result:

Thus the above program was implemented and successfully completed.

Exp. No.: 12

Date:

Implementing AI for a Real-world Problem (Predicting Student Performance)

Aim:

To implement an AI-based machine learning model using Decision Tree Classifier to predict student performance (pass/fail) based on study time, failures, and absences.

Algorithm:

1. Import required libraries (pandas, scikit-learn, matplotlib, seaborn).
2. Load the student performance dataset.
3. Preprocess the data:
   a. Create a binary target variable "pass" based on final grade (G3 ≥ 10 → pass).
   b. Select relevant features (studytime, failures, absences).
4. Split the dataset into training and testing sets.
5. Train a Decision Tree Classifier on the training data.
6. Predict student performance on the testing data.
7. Evaluate the model using accuracy and classification report.
8. Visualize feature importance using a bar plot.

Program:

```
# Install dependencies
# pip install pandas scikit-learn matplotlib seaborn
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
```

```
import seaborn as sns
# Load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/student-mat.csv'
data = pd.read_csv(url, sep=';')
# Create binary target (pass/fail)
data['pass'] = (data['G3'] >= 10).astype(int)
# Select features
features = ['studytime', 'failures', 'absences']
X = data[features]
y = data['pass']
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
# Prediction
y_pred = model.predict(X_test)
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
# Feature Importance Plot
importance = model.feature_importances_
sns.barplot(x=features, y=importance)
plt.title("Feature Importance")
plt.show()
```

Output:

```
Loading dataset...
Training Decision Tree model...
Evaluating...

Accuracy: 0.75
              precision    recall  f1-score   support

           0       0.70      0.65      0.67        40
           1       0.77      0.81      0.79        60

    accuracy                           0.75       100
   macro avg       0.74      0.73      0.73       100
weighted avg       0.75      0.75      0.75       100
```

Result:

Thus the above program was implemented and successfully completed.