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)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

Department of Computer Science and Engineering (Cyber Security)

Course Code : 23ADC01

ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY

Continuous Assessments Record

Submitted by

Name	
Register No	
Year & Semester	**II BE CSE (CSY) & III**
Department	**Computer Science and Engineering (Cyber Security)**
Section	
Academic Year	**2025-2026 (Odd Semester)**

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)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

Department of Computer Science and Engineering (Cyber Security)

23ADC01– ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY

Submitted by

Name : **Register No.:**

Degree : **Branch :**

BONAFIDE CERTIFICATE

This is to certify that this is a bonafide record work by Mr./Ms…… Register No:…… during the academic year 2025 – 2026.

FACULTY IN-CHARGE

Submitted for the Autonomous Practical Examination held on…….....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

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)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

Department of Computer Science and Engineering (Cyber Security)

VISION OF THE INSTITUTE

To Produce Globally Competitive Engineers with High Ethical Values and Social Responsibilities.

MISSION OF THE INSTITUTE

- To impart highest quality state-of-the-art technical education by providing impetus to innovation, Research and Development and empowering students with Entrepreneurship skills.
- To instill ethical values, imbibe a sense of social responsibility and strive for societal wellbeing.
- To identify needs of society and offer sustainable solutions through outreach programs.

VISION OF THE DEPARTMENT

- To be a globally renowned academic department for quality education and research in the field of Cyber security with ethical values and social commitment

MISSION OF THE DEPARTMENT

- To Impart comprehensive technical education to produce highly competent Cyber security professionals and Researchers
- Provide an academic environment with state-of-the-art technological infrastructure to provide scalable cyber security solutions
- Impart ethics, Social responsibility and necessary professional, leadership skills through student centric activities

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

The following Programme Educational Objectives are designed based on the department mission

PEO 1 Get recognized as effective professionals for their applied skills, problem solving capabilities and professional skills in the field of cyber security

PEO 2 Enrich skills and adopt emerging cyber security needs to pursue life-long learning and serve the society with social concern and code of ethics

PROGRAMME SPECIFIC OUTCOMES (PSOs)

On successful completion of Bachelor of Engineering in Computer Science and Engineering (Cyber Security) Programme from Sri Krishna College of Engineering and Technology, the graduate will demonstrate:

PSO 1: Attain the policies in information assurance and analyze the factors in an existing system and design implementations to comprehend and anticipate future challenges

PSO 2: Implement innovative cyber security solutions using standard tools, practices and technologies without compromising the privacy of individuals and entities

PSO 3: Use cyber security and cyber forensics software/tools. Design cyber-security strategies and assess cyber-security risk management policies to protect an organization's information and assets.

PROGRAMME OUTCOMES (POs)

At the time of their graduation students of Electronics and Communication Engineering Programme should be in possession of the following Programme Outcomes

PO1. Engineering Knowledge: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO 2 Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO 3 Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO 4 Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO 5 Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6) **PO 6 The Engineer and The World:** Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO 7 Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO 8 Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO 9 Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO 10 Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one‟s own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO 11 Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. **(WK8)**

SYLLABUS

Ex. No	Description	Course Outcome	Level of Bloom's Taxonomy
1	Implement a simple AI-based Python script	**CO2**	**[AP]**
2	Exploring AI applications in real-world case studies.	**CO2**	**[U]**
3	Implementing BFS and DFS in Python.	**CO3**	**[AP]**
4	Implement A* and Greedy Best-First Search to find the shortest path in a grid.	**CO3**	**[AN]**
5	Implement an AI agent to play Tic-Tac-Toe using the Minimax algorithm.	**CO3**	**[AN]**
6	Implement a basic propositional logic reasoning system.	**CO3**	**[AP]**
7	Build a simple knowledge base using First-Order Logic.	**CO5**	**[AP]**
8	Create a semantic network to represent relationships between objects.	**CO5**	**[AN]**
9	Design a basic expert system for medical diagnosis	**CO4**	**[AN]**
10	Perform tokenization, stemming, and stop-word removal on sample text.	**CO1**	**[AP]**
11	Implement a speech-to-text system using Python's speech recognition library.	**CO2**	**[AP]**
12	Implementing AI for a real-world problem (e.g., predicting student performance).	**CO2**	**[AN]**

TEXTBOOKS

1. Stuart Russell & Peter Norvig, "Artificial Intelligence: A Modern Approach", 4th Edition, Pearson, 2021.
2. K. R. Chowdhary, Fundamentals of Artificial Intelligence, Springer, 2020.

3. Elaine Rich & Kevin Knight, "Artificial Intelligence", 3rd Edition, McGraw-Hill, 2017.

REFERENCE BOOKS

1. Wolfgang Ertel, Introduction to Artificial Intelligence, 2nd Edition, Springer, 2018.
2. Mariusz Flasiński, Introduction to Artificial Intelligence, Springer, 2016.
3. Stephen Lucci & Danny Kopec, Artificial Intelligence in the 21st Century: A Living Introduction, 2nd Edition, Mercury Learning and Information, 2016

WEB RESOURCES

1) https://www.coursera.org/learn/introduction-to-ai
2) https://www.coursera.org/learn/ai-for-everyone
3) https://ai.google/get-started/for-developers/
4) https://nptel.ac.in/courses/106102220
5) https://nptel.ac.in/courses/106106140

1. Expected outcome of the course:

Upon successful completion of this course, the student will be able to:

CO1	Assess various Artificial Intelligence (AI) techniques and identify their fundamental concepts.	[U]
CO2	Apply problem-solving techniques such as search algorithms and game theory in AI systems.	[AP]
CO3	Implement knowledge representation and reasoning techniques for intelligent decision-making.	[AP]
CO4	Demonstrate Machine Learning algorithms and AI frameworks to solve real-world problems	[AP]
CO5	Design AI-based applications for NLP, Computer Vision, and Robotics.Evaluate the ethical considerations and social impacts of AI in various domains	[AP]

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)-2628001 (7 Lines) | Email : info@skcet.ac.in | Website : www.skcet.ac.in

Department of Computer Science and Engineering (Cyber Security)

23ADC01– ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY
(ODD SEMESTER: 2025-2026)

Components	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10	EX11	EX12
DATE												
Aim & Algorithm (20)												
Coding (30)												
Compilation & Debugging (30)												
Execution & Results (10)												
Documentation & Viva (10)												
Consolidated Mark (100)												
Faculty Signature												
									Average (100)		**Signature**	

TABLE OF CONTENTS				
Exp. No.	**Date**	**EXPERIMENT LIST**	**PAGE NO.**	**Sign**
1.		Implement a simple AI-based Python script		
2.		Exploring AI applications in real-world case studies.		
3.		Implementing BFS and DFS in Python.		
4.		Implement A* and Greedy Best-First Search to find the shortest path in a grid.		
5.		Implement an AI agent to play Tic-Tac- Toe using the Minimax algorithm.		
6.		Implement a basic propositional logic reasoning system.		
7.		Build a simple knowledge base using First-Order Logic.		
8.		Create a semantic network to represent relationships between objects.		
9.		Design a basic expert system for medical diagnosis		
10.		Perform tokenization, stemming, and stop-word removal on sample text.		
11.		Implement a speech-to-text system using Python's speech recognition library.		
12.		Implementing AI for a real-world problem (e.g., predicting student performance).		
Additional Virtual Lab Experiment				

1.		Implement an image classification model using a pre-trained CNN (e.g., MobileNet, ResNet) on sample image data.		
2.		Develop a chatbot using Natural Language Processing techniques.		
	Average (100)			

Faculty Signature

Continuous Evaluation Rubrics Sheet

23ADC01– ARTIFICIAL INTELLIGENCE AND ITS APPLICATIONS LABORATORY

Items	Excellent	Good	Satisfactory	Needs Improvement
Aim & Algorithm (20)	**20 Marks**	**(19-17) Marks**	**(16-14) Marks**	**(13-11)Marks**
	The aim and purpose of the experiment are clearly defined. Preliminary questions are to be answered. The equipment's required are to be clearly listed with specifications.	Able to define the aim and purpose of the experiment but not clearly. Preliminary questions are answered but not in detail. The equipment's required are clearly listed but with inappropriate specifications.	The aim and purpose of the experiment are defined but not clear. Few Preliminary questions are unanswered. The equipment's required are clearly listed but without specifications.	Unable to explain the aim and purpose of the experiment. Preliminary questions are unanswered. The equipment's required are not clearly listed.
Coding (30)	**30Marks**	**(29-27) Marks**	**(26-24) Marks**	**(23-21) Marks**
	Able to draw Connection diagram neatly with no errors. Formulas required are written and used fully in computation. Model graphs are provided with necessary information.	Connection diagram of the experiment is provided with minor mistakes. Formulas required are written. Model graphs are provided with few missing data.	Connection diagram not drawn legibly and with fewer specifications of the components involved Some mistakes in formula. Model graph is not clear.	Connection diagram of the experimental setup are given wrongly. No model graph provided.
Compilation & Debugging (30)	**40Marks**	**(39-37) Marks**	**(36-34) Marks**	**(33-31) Marks**
	The components are identified and connections done without error. Experimentation is as required. Readings taken are justified and tabulated.	The components are identified and connections done without error. The method adopted is relevant but the measurements made are partial. Readings are tabulated.	The components are identified but the connection done with few mistakes. The measured values are deviated.	Poorly experimentation and the method adopted is not relevant to the stated objective.

Execution & Results(10)	**20Marks**	**(19-17) Marks**	**(16-14) Marks**	**(13-11) Marks**
	Readings/measurements are utilized to draw necessary charts/graphs. The results are interpreted and compared with desired values successfully.	Almost all of the results have been correctly recorded and summarized; only minor improvements are needed in post lab discussion.	Some mistakes in tables and graph. Conclusions drawn from the results are not clear.	Experimental measurements are incorrect and wrongly Not able to measurement and proceed further.
Documentation & Viva-vove	**10 Marks**	**(9-7)Marks**	**(6-4)Marks**	**(3-1)Marks**
(10)	experiment details are well documented.	the aim, procedure, circuit diagram and experiment details to some extent.	and tabulate the content and organize properly.	work and Poor writing presentation.

Signature of Evaluator

EXP NO :1	IMPLEMENT A SIMPLE AI – BASED PYTHON SCRIPT
DATE :	

1. 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. 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.

PROCEDURE:

1. Open a Python IDE (Jupyter Notebook, VS Code, or Anaconda Prompt).
2. Ensure that the textblob library is installed using:

pip install textblob

3. Import the TextBlob class.
4. Get input from the user for analysis.
5. Analyze the sentiment using TextBlob.

6. Print the polarity score and classified sentiment.

7. Save and run the program to observe the output.

PROGRAM:

```
from textblob import TextBlob
print("Welcome all to This Sentiment Analysis Page ")
print("No matter what the lines number, all matters is the polarity!!")
user_i=input("Enter the desired passage : ")
analysis = TextBlob(user_i)
polarity = analysis.sentiment.polarity
if(polarity>0):
    sentiment="Positive"
elif(polarity<0):
    sentiment="negative"
else:
    sentiment="Neutral"
print(f"The final results are here!")
print(f"The polarity of passage is : {polarity}")
print(f"The Sentiment is :{sentiment}")
```

OUTPUT :

```
================= RESTART: C:/Users/kavya/126-Kavyaa_S_Exp_1.py ================
Welcome all to This Sentiment Analysis Page
No matter what the lines number, all matters is the polarity!!
Enter the desired passage : 🎬 Harry Potter and the Sorcerer's Stone (2001) - Sh
ort Review

Magical, charming, and timeless, the first film in the Harry Potter series intro
duces us to the enchanting world of wizards through the eyes of young Harry, an
orphan who discovers he's destined for greatness at Hogwarts School of Witchcraf
t and Wizardry. The movie beautifully captures the wonder and innocence of J.K.
Rowling's world, blending adventure, friendship, and mystery.

Daniel Radcliffe, Emma Watson, and Rupert Grint shine as the iconic trio, and th
e magical setting is brought to life with rich visuals and a captivating score b
y John Williams. While aimed at younger audiences, the film's heartwarming story
 and sense of wonder appeal to all ages.

A magical beginning to a legendary journey.
The final results are here!
The polarity of passage is : 0.0
The Sentiment is :Neutral
```

RESULT :

A simple AI-based sentiment analysis program was successfully implemented using the TextBlob library in Python. The program accepts user input and correctly classifies it as Positive, Negative, or Neutral based on the polarity score.

EXP NO :	EXPLORING AI APPLICATIONS IN REAL WORLD CASE STUDIES
DATE :	

THE MAGIC OF AI AT DISNEY : REVOLUTIONIZING ENTERTAINMENT THROUGH ARTIFICIAL INTELLIGENCE

ABSTRACT

Disney has long been a leader in storytelling and innovation. In recent years, the company has integrated Artificial Intelligence (AI) across its divisions—from animation studios to theme parks. By using deep learning models, cloud-based tools, and AI-powered robots, Disney has improved animation workflows, enhanced content personalization, and created immersive guest experiences. This case study explores Disney's AI journey, its implementation strategies, and its impact on the entertainment industry.

INTRODUCTION

Disney is known worldwide not just for its creative magic but also for its consistent adoption of emerging technologies. From the first synchronized-sound cartoon *Steamboat Willie* to lifelike digital humans, Disney has shown that storytelling and technology can go hand-in-hand. Today, Disney uses AI to improve everything from content tagging on ESPN to character recognition in animation and even interactive robots in theme parks. This case study focuses on how Disney is blending human creativity with machine intelligence to redefine the future of entertainment.

OBJECTIVES

- To understand how AI is applied across Disney's core business areas.
- To explore the role of deep learning in animation and content personalization.
- To examine AI-driven experiences in Disney theme parks.
- To identify the tools and platforms enabling these advancements.
- To evaluate challenges and ethical concerns in Disney's AI journey.

PROCESS INVOLVED

1. AI in Animation

- Deep learning models (using PyTorch) distinguish characters in complex scenes.
- Speeds up content tagging and improves metadata accuracy.

2. AI in Streaming Platforms (e.g., ESPN)

- AI recommends content based on user preferences.
- Enhances user engagement with personalized experiences.

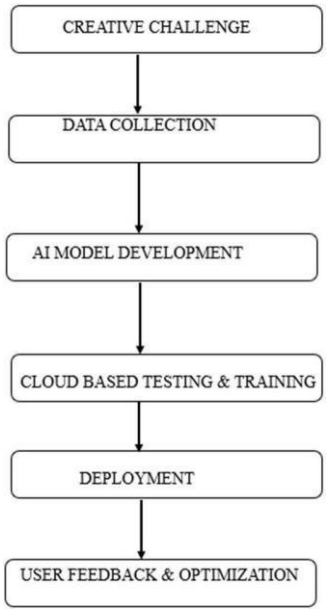
3. AI in Theme Parks

- Baby Groot robot and D3-09 droid use conversational AI and motion sensors.
- Real-time interactions provide immersive storytelling.

4. Cloud and Tools

- AWS cloud services used for model training, deployment, and scalability.
- Shift from traditional ML to modern deep learning pipelines.

Flow Chart: Disney's AI Integration Process



SIMILAR TOOLS AND EXTENSIONS

Use Case	Disney Tool/Platform	Industry Equivalent
Animation Recognition	PyTorch + in-house models	Adobe Sensei, DeepMotion
Content Personalization	Disney+ recommender systems	Netflix's Recommendation Engine
Conversational Characters	AI on ESPN, Disney Droid	ChatGPT, Microsoft Copilot
AI Model Hosting & Scaling	AWS Cloud + Disney's D3	Microsoft Azure AI, Google Cloud AI
Interactive Storytelling	Robotics + NLP	Meta's Reels AI, NVIDIA Omniverse

CRITIQUE CHALLENGES AND ETHICAL CONCERNS

While Disney's integration of AI is innovative, it is not without limitations:

1. Ethical Concerns on Creativity

- There is growing concern that AI-generated animations or scripts might lead to a decline in **original, human-driven creativity**.
- Over-reliance on AI tools may result in formulaic or emotionless content that lacks the artistic depth of traditional works.

2. Job Displacement

- As AI takes over animation tagging, facial rendering, and content generation, **creative professionals** (like storyboard artists and animators) risk being sidelined or replaced.
- This also raises questions in the broader entertainment industry about the future of work.

3. Privacy and Data Use

- Disney collects vast user data for content personalization—especially from children and families.
- Using this data for AI decisions **raises privacy concerns**, especially under regulations like GDPR or COPPA (Children's Online Privacy Protection Act).

4. Cost and Accessibility

- Advanced AI features in theme parks (e.g., robotic characters) may be available only to **premium users**, widening the digital divide between regular and VIP guests.

5. Bias in AI Models

- AI systems trained on limited or biased datasets can lead to **inaccurate character recognition** or representation issues in storytelling, especially around gender, race, or cultural diversity.

CONCLUSION

Disney's use of AI is a powerful example of how modern technology can enhance storytelling, streamline production, and create unforgettable guest experiences. By strategically using tools like PyTorch and AWS, Disney has succeeded in blending cloud-based AI with real-world creativity. However, it must continue to address the ethical and human impact of automation, data use, and creative job shifts. As Disney moves forward, the real magic will lie in how it balances innovation with imagination and automation with artistry.

REFERENCE LINKS

1. **AIX Case Study on Disney – "The Magic of AI at Disney"**
   https://www.aiexpert.network/case-study-the-magic-of-ai-at-disney
2. **DisneyResearch Official Website**
   https://www.disneyresearch.com/
3. **Star Wars: Galactic Starcruiser – Disney Parks Innovation**
   https://disneyworld.disney.go.com/star-wars-galactic-starcruiser/

EXP NO :3	IMPLEMENTING BFS AND DFS IN PYTHON
DATE :	

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.

A) BREADTH-FIRST SEARCH (BFS) ALGORITHM :

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:
     - Create a new path with the neighbor.
     - Add the new path to the queue.

5. End the program.

PROCEDURE :

1. Define a graph as a dictionary (adjacency list).
2. Implement bfs (start, goal) using a queue.
3. Provide a start node and a goal node.
4. Call both functions and observe the path to the goal node.

5. Display the visiting order and final path.

PROGRAM :

```
from collections import deque


def input_graph():
    graph = {}
    num_nodes = int(input("Enter number of nodes in the graph: "))
    for i in range(num_nodes):
        node = input(f"Enter name of node {i+1}: ").strip()
        neighbors = input(f"Enter neighbors of {node} (comma-separated): ").strip()
        graph[node] = [n.strip() for n in neighbors.split(",")] if neighbors else []
    return graph


def bfs(graph, start, goal):
    visited = []
    queue = deque([[start]])
    while queue:
        path = queue.popleft()
        node = path[-1]
        if node not in visited:
            visited.append(node)
            if node == goal:
```

```
            return visited, path
        for neighbor in graph.get(node, []):
            new_path = list(path)
            new_path.append(neighbor)
            queue.append(new_path)
    return visited, []

graph = input_graph()
start_node = input("Enter start node: ").strip()
goal_node = input("Enter goal node: ").strip()
visited_order, bfs_path = bfs(graph, start_node, goal_node)

print("\n=== BFS ===")
print("Visited Order:", visited_order)
print("Path to Goal:", " -> ".join(bfs_path) if bfs_path else "No path found")
```

OUTPUT :

```
=== RESTART: C:/Users/Kavyaasuresh/OneDrive/Documents/126-Kavyaa_S - EXP 3.py ==
Enter number of nodes in the graph: 5
Enter name of node 1: A
Enter neighbors of A (comma-separated): B,C
Enter name of node 2: B
Enter neighbors of B (comma-separated): D
Enter name of node 3: C
Enter neighbors of C (comma-separated): 0
Enter name of node 4: D
Enter neighbors of D (comma-separated): E
Enter name of node 5: E
Enter neighbors of E (comma-separated): 0
Enter start node: A
Enter goal node: E

=== BFS ===
Visited Order: ['A', 'B', 'C', 'D', '0', 'E']
Path to Goal: A -> B -> D -> E
```

B) DEPTH-FIRST SEARCH (DFS) ALGORITHM

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:

   - Mark it as visited.
   - 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.

PROCEDURE :

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

PROGRAM :

```
def input_graph():
    graph = {}
    num_nodes = int(input("Enter number of nodes in the graph: "))
    for i in range(num_nodes):
        node = input(f"Enter name of node {i+1}: ").strip()
        neighbors = input(f"Enter neighbors of {node} (comma-separated): ").strip()
        graph[node] = [n.strip() for n in neighbors.split(",")] if neighbors else []
    return graph


def dfs(graph, start, goal):
    visited = []
    stack = [[start]]
    while stack:
        path = stack.pop()
        node = path[-1]
        if node not in visited:
            visited.append(node)
            if node == goal:
                return visited, path
            for neighbor in reversed(graph.get(node, [])):  # reverse for correct DFS order
                new_path = list(path)
                new_path.append(neighbor)
                stack.append(new_path)
```

```
    return visited, []

graph = input_graph()

start_node = input("Enter start node: ").strip()

goal_node = input("Enter goal node: ").strip()

visited_order, dfs_path = dfs(graph, start_node, goal_node)


print("\n=== DFS ===")

print("Visited Order:", visited_order)

print("Path to Goal:", " -> ".join(dfs_path) if dfs_path else "No path found")

print("Visited Order:", visited_order)

print("Path to Goal:", " -> ".join(dfs_path) if dfs_path else "No path found")
```

OUTPUT :

```
========== RESTART: C:/Users/Kavyaasuresh/OneDrive/Documents/126-Kavyaa_S - EXP 3.py ==========
Enter number of nodes in the graph: 6
Enter name of node 1: A
Enter neighbors of A (comma-separated): B,C
Enter name of node 2: B
Enter neighbors of B (comma-separated): D,E
Enter name of node 3: C
Enter neighbors of C (comma-separated): F
Enter name of node 4: D
Enter neighbors of D (comma-separated): 0
Enter name of node 5: E
Enter neighbors of E (comma-separated): 0
Enter name of node 6: F
Enter neighbors of F (comma-separated): 0
Enter start node: A
Enter goal node: E

=== DFS ===
Visited Order: ['A', 'B', 'D', '0', 'E']
Path to Goal: A -> B -> E
```

RESULT :

The Python program to implement Breadth-First Search (BFS) and Depth-First Search (DFS) using a user-defined graph was successfully executed. The program was tested with user input produced the expected results.

EXP NO :4	IMPLEMENTING A* ALOGRITHMS IN PYTHON
DATE :	

AIM :

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

PROCEDURE :

1. Define a grid (2D list) with 0 for free cells and 1 for obstacles.
2. Implement a_star(grid, start, goal) using both cost and heuristic.
3. Implement greedy_bfs(grid, start, goal) using only heuristic.
4. Provide start and goal coordinates.
5. Call both functions and observe the final path.
6. Display the visiting order and final path to the goal.

ALGORITHM :

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.
    1. 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.

4. End the program

PROGRAM :

```
def a_star(graph, h, start, goal):
    print("\nA* Algorithm")
    open_list = [(start, [start], 0)]
    visited = set()
    while open_list:
        open_list.sort(key=lambda x: x[2] + h[x[0]])
        current, path, cost = open_list.pop(0)
        print("Visiting:", current)
        if current == goal:
            print("\nGoal Reached!")
            return path
        visited.add(current)
        for neighbor, edge_cost in graph.get(current, []):
            if neighbor not in visited:
                total_cost = cost + edge_cost
                open_list.append((neighbor, path + [neighbor], total_cost))
    print("\nNo Path Found.")
    return None
graph = {}
n = int(input("Enter number of nodes: "))
for _ in range(n):
    node = input("Enter node: ").strip()
```

```
    neighbors = input(f"Enter neighbors of {node} as (name cost), separated by commas: ").strip()
    neighbor_list = []
    if neighbors:
        for pair in neighbors.split(','):
            name, cost = pair.strip().split()
            neighbor_list.append((name, int(cost)))
    graph[node] = neighbor_list
h = {}
print("Enter heuristic values:")
for node in graph:
    h[node] = int(input(f"Heuristic for {node}: "))
start = input("Enter start node: ").strip()
goal = input("Enter goal node: ").strip()
result = a_star(graph, h, start, goal)
print("\nFinal Output")
print("Algorithm: A* Search")
if result:
    print("Path:", " -> ".join(result))
else:
    print("No valid path.")
```

OUTPUT

```
================= RESTART: C:/Users/kavya/126-Kavyaa_S-AI-EXP4 =========
========
Enter number of nodes: 5
Enter node: A
Enter neighbors of A as (name cost), separated by commas: B 1 , C 4
Enter node: B
Enter neighbors of B as (name cost), separated by commas: D 2
Enter node: C
Enter neighbors of C as (name cost), separated by commas: D 1
Enter node: D
Enter neighbors of D as (name cost), separated by commas: E 3
Enter node: E
Enter neighbors of E as (name cost), separated by commas: 0 0
Enter heuristic values:
Heuristic for A: 7
Heuristic for B: 6
Heuristic for C: 2
Heuristic for D: 1
Heuristic for E: 0
Enter start node: A
Enter goal node: E

A* Algorithm
Visiting: A
Visiting: C
Visiting: D
Visiting: B
Visiting: E

Goal Reached!

Final Output
Algorithm: A* Search
Path: A -> C -> D -> E
```

RESULT :

Thus a python program to implement and demonstrate A* algorithm was successfully executed and verified.

EXP NO :4b	IMPLEMENTING GREEDY BEST – FIRST SEARCH ALOGRITHMS IN PYTHON
DATE :	

AIM :

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

ALGORITHM :

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 :

```
def greedy_best_first_search(graph, h, start, goal):
    print("\nGreedy Best-First Search")
    open_list = [(start, [start])]
    visited = set()
```

```
    while open_list:
        open_list.sort(key=lambda x: h[x[0]])
        current, path = open_list.pop(0)
        print("Visiting:", current)

        if current == goal:
            print("\nGoal Reached!")
            return path
        visited.add(current)
        for neighbor, _ in graph.get(current, []):
            if neighbor not in visited:
                open_list.append((neighbor, path + [neighbor]))
    print("\nNo Path Found.")
    return None


graph = {}
n = int(input("Enter number of nodes: "))
for _ in range(n):
    node = input("Enter node: ").strip()
    neighbors = input(f"Enter neighbors of {node} as (name cost), separated by commas: ").strip()
    neighbor_list = []
    if neighbors:
        for pair in neighbors.split(','):
            name, cost = pair.strip().split()
            neighbor_list.append((name, int(cost)))
    graph[node] = neighbor_list


h = {}
print("Enter heuristic values:")
for node in graph:
    h[node] = int(input(f"Heuristic for {node}: "))
```

```
start = input("Enter start node: ").strip()
goal = input("Enter goal node: ").strip()
result = greedy_best_first_search(graph, h, start, goal)
print("\nFinal Output")
print("Algorithm: Greedy Best-First Search")
if result:
    print("Path:", " -> ".join(result))
else:
    print("No valid path.")
```

OUTPUT

```
============ RESTART: C:/Users/kavya/126-Kavyaa_S AI-EXP4-GREEDY.py ========
Enter number of nodes: 5
Enter node: A
Enter neighbors of A as (name cost), separated by commas: B 1, C 4
Enter node: B
Enter neighbors of B as (name cost), separated by commas: D 2
Enter node: C
Enter neighbors of C as (name cost), separated by commas: D 1
Enter node: D
Enter neighbors of D as (name cost), separated by commas: E 3
Enter node: E
Enter neighbors of E as (name cost), separated by commas: 0 0
Enter heuristic values:
Heuristic for A: 7
Heuristic for B: 6
Heuristic for C: 2
Heuristic for D: 1
Heuristic for E: 0
Enter start node: A
Enter goal node: E

Greedy Best-First Search
Visiting: A
Visiting: C
Visiting: D
Visiting: E

Goal Reached!

Final Output
Algorithm: Greedy Best-First Search
Path: A -> C -> D -> E
```

Result:

Thus a python program to implement and demonstrate Greedy-Best First Search algorithm was successfully executed and verified

EXP NO :5	IMPLEMENT AN AI AGENT TO PLAY TIC-TAC-TOE USING MINIMAX ALGORITHM
DATE :	

AIM

To implement an AI agent that plays Tic-Tac-Toe optimally using the Minimax Algorithm.

ALGORITHM

Minimax for Tic-Tac-Toe

1. **Start** the game with an empty board.
2. **Check for terminal states**:
   - Win for ‘X’ or ‘O’
   - Draw
3. **For each empty cell**:
   - Simulate the move.
   - Call minimax() recursively:
     - Maximize the score if AI's turn (e.g., 'O')
     - Minimize the score if opponent's turn (e.g., 'X')
   - Undo the move (backtrack).
4. Choose the move with the **best score** for the AI.
5. Repeat until game ends.

PROGRAM

```
def print_board(board):
    print(f"{board[0]} | {board[1]} | {board[2]}")
    print("--+---+--")
    print(f"{board[3]} | {board[4]} | {board[5]}")
    print("--+---+--")
    print(f"{board[6]} | {board[7]} | {board[8]}")
    print()
def show_positions():
```

```
    print("Positions are numbered 1 to 9 as below:")
    print("1 | 2 | 3")
    print("--+---+--")
    print("4 | 5 | 6")
    print("--+---+--")
    print("7 | 8 | 9")
    print()
def is_valid_move(board, move):
    return move.isdigit() and 1 <= int(move) <= 9 and board[int(move) - 1] == ' '
def check_winner(board, player):
    win_conditions = [
        (0, 1, 2), (3, 4, 5), (6, 7, 8),  # rows
        (0, 3, 6), (1, 4, 7), (2, 5, 8),  # columns
        (0, 4, 8), (2, 4, 6)              # diagonals
    ]
    for i, j, k in win_conditions:
        if board[i] == board[j] == board[k] == player:
            return True
    return False


def is_draw(board):
    return ' ' not in board
def make_ai_move(board):
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'O'
            return i + 1
def main():
```

```
board = [' '] * 9
  print("Welcome to Tic-Tac-Toe!")
  print("You are X, AI is O")
  show_positions()
  print_board(board)
  while True:
    move = input("Enter your move (1-9): ")
    if not is_valid_move(board, move):
      print("Please enter a valid number.")
      print_board(board)
      continue
    move = int(move) - 1
    board[move] = 'X'
    print_board(board)

 if check_winner(board, 'X'):
      print(" You win!")
      break
    if is_draw(board):
      print("It's a draw!")
      break
    ai_pos = make_ai_move(board)
    print(f"AI placed O at position {ai_pos}")
    print_board(board)
    if check_winner(board, 'O'):
      print(" AI wins!")
      break
    if is_draw(board):
      print("It's a draw!")
```

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

OUTPUT

```
================ RESTART: C:/Users/kavya/126-Kavyaa_S-AI-EXP5.py ===============
Welcome to Tic-Tac-Toe!
You are X, AI is O
Positions are numbered 1 to 9 as below:
1 | 2 | 3
--+---+--
4 | 5 | 6
--+---+--
7 | 8 | 9

  |   |
--+---+--
  |   |
--+---+--
  |   |

Enter your move (1-9): 1
X |   |
--+---+--
  |   |
--+---+--
  |   |

AI placed O at position 2
X | O |
--+---+--
  |   |
--+---+--
  |   |

Enter your move (1-9): 2
Please enter a valid number.
X | O |
--+---+--
  |   |
--+---+--
  |   |
```

```
Enter your move (1-9): 3
X | O | X
--+---+--
  |   |
--+---+--
  |   |

AI placed O at position 4
X | O | X
--+---+--
O |   |
--+---+--
  |   |

Enter your move (1-9): 5
X | O | X
--+---+--
O | X |
--+---+--
  |   |

AI placed O at position 6
X | O | X
--+---+--
O | X | O
--+---+--
  |   |

Enter your move (1-9): 7
X | O | X
--+---+--
O | X | O
--+---+--
X |   |

🎉 You win!
```

RESULT

Thus a Tic-Tac-Toe game was developed in Python where the user plays against a basic AI.The game runs until a win or draw is detected, with valid input handling and automatic AI moves.

EXP NO : 6	IMPLEMENTING A BASIC PROPOSITIONAL LOGIC REASONING SYSTEM
DATE :	

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).

CONCEPT

Propositional Logic uses propositions (statements) that are either True or False.

Logical connectives such as:

- AND (∧)
- OR (∨),
- NOT (¬)
- IMPLICATION (→)
- BICONDITIONAL (↔)

are used to form compound statements.

PROCEDURE

1. Define propositional variables and assign them truth values (e.g., P = True, Q = False).
2. Create functions to simulate logical operations: AND, OR, NOT, →, ↔.
3. Parse expressions manually or automatically (for advanced systems).
4. Evaluate expressions based on input truth assignments.
5. Display the evaluation results.

PROGRAM A:

```
def implies(p, q):
    return (not p) or q
```

```
 def iff(p, q):
    return (p and q) or (not p and not q)
propositions = {
    'A': True,
     'B': False,
    'C': True
}
def evaluate_expressions():
    print("\n--- Propositional Logic Evaluation ---")
    result1 = propositions['A'] and not propositions['C']
    print("A ∧ ¬C =", result1)
    result2 = (propositions['A'] or propositions['B']) and propositions['C']
    print("(A ∨ B) ∧ C =", result2)
    result3 = not propositions['B'] or propositions['A']
    print("¬B ∨ A =", result3)
    result4 = implies(propositions['A'], propositions['C'])
    print("A → C =", result4)
    result5 = iff(propositions['B'], propositions['C'])
    print("B ↔ C =", result5)
evaluate_expressions()
```

OUTPUT :

```
================ RESTART: C:/Users/kavya/126-Kavyaa S- EXP 6.py ===========

--- Propositional Logic Evaluation ---
A ∧ ¬C = False
(A ∨ B) ∧ C = True
¬B ∨ A = True
A → C = True
B ↔ C = False
```

PROGRAM B :

```
from sympy import symbols

from sympy.logic.boolalg import And, Or, Not, Implies, Equivalent

from sympy.logic.boolalg import to_cnf

from sympy import simplify_logic

A, B, C = symbols('A B C')

truth_values = {

   A:  True,

   B: False,

   C: True

}

expr1 = And(A, Not(C))          # A ∧ ¬C

expr2 = And(Or(A, B), C)        # (A ∨ B) ∧ C

expr3 = Or(Not(B), A)           # ¬B ∨ A

expr4 = Implies(A, C)           # A → C

expr5 = Equivalent(B, C)        # B ↔ C

def evaluate_expression(expr, truth_vals):
```

```
    return expr.subs(truth_vals)


# Display results
print("\n--- Evaluating Propositional Logic Expressions using SymPy ---")
print(f"A ∧ ¬C = {evaluate_expression(expr1, truth_values)}")
print(f"(A ∨ B) ∧ C = {evaluate_expression(expr2, truth_values)}")
print(f"¬B ∨ A = {evaluate_expression(expr3, truth_values)}")
print(f"A → C = {evaluate_expression(expr4, truth_values)}")
print(f"B ↔ C = {evaluate_expression(expr5, truth_values)}")
```

OUTPUT :

```
========== RESTART: C:/Users/kavya/Downloads/126-Kavyaa S- AI - 6.py ==========

--- Evaluating Propositional Logic Expressions using SymPy ---
A ∧ ¬C = False
(A ∨ B) ∧ C = True
¬B ∨ A = True
A → C = True
B ↔ C = False
```

RESULT

The program has been executed successfully and output is Verified

EXP NO : 7	BUILD A SIMPLE KNOWLEDGE BASE USING
DATE :	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 Libraries:**
   Import required modules from the kanren library: Relation, facts, run, var, and conde.
2. **Define Relations:**
   Define FOL relations such as student, likes, and others.
3. **Insert Facts into the Knowledge Base:**
   Use facts() to define who is a student and what subjects they like.
4. **Define Rules:**
   Create a rule: "If a person is a student and likes something, then they are intelligent".
5. **Accept User Query:**
   Take user input (e.g., “Who is intelligent?” or “Is Alice intelligent?”)
6. **Perform Reasoning:**
   Based on the query, use run() and the defined rules to determine the correct answer.
7. **Display Result:**
   Print whether the person is intelligent or list all intelligent people.

PROGRAM:

```
import collections

import collections.abc

collections.Iterator = collections.abc.Iterator

collections.Hashable = collections.abc.Hashable

from kanren import Relation, facts, run, var, conde

# Knowledge base: hospital scenario

doctor = Relation()

treats = Relation()

# Facts

facts(doctor,

    ("dr_smith",),

    ("dr_jones",),

    ("dr_lee",),

    ("dr_williams",),

    ("dr_davis",))

facts(treats,

    ("dr_smith", "flu"),

    ("dr_smith", "allergy"),

    ("dr_jones", "cold"),

    ("dr_lee", "covid"),

    ("dr_lee", "pneumonia"),

    ("dr_williams", "diabetes"),
```

```
    ("dr_davis", "hypertension"))

# Rule: A doctor is skilled if they treat at least one disease
def is_skilled(person):
    return conde((doctor(person), treats(person, var())),)

# Take user input
query = input("Enter query: ").strip().lower()
x = var()

# "Who" query
if query.startswith("who"):
    result = run(0, x, is_skilled(x))
    if result:
        print("✓ Skilled doctors found:")
        for name in result:
            print(f"- {name.replace('_',' ').title()}")
    else:
        print("X No skilled doctors found.")

# "Is" query
elif query.startswith("is"):
    words = query.replace("?", "").split()
    if len(words) >= 3:
        name = words[1]
        result = run(0, x, is_skilled(x))
        if name in result:
            print(f"✓ Yes, {name.replace('_',' ').title()} is skilled.")
```

```
        else:
            print(f"X No, {name.replace('_',' ').title()} is not skilled.")
    else:
        print("X Invalid query format.")

else:
    print("X Invalid query format.")
```

OUTPUT:

```
================== RESTART: C:/Users/kavya/126-KavyaaS-AI-7.py =================
Enter query: who is skilled?
✓ Skilled doctors found:
- Dr Davis
- Dr Smith
- Dr Williams
- Dr Lee
- Dr Jones

================== RESTART: C:/Users/kavya/126-KavyaaS-AI-7.py =================
Enter query: is dr_lee skilled?
✓ Yes, Dr Lee is skilled.

================== RESTART: C:/Users/kavya/126-KavyaaS-AI-7.py =================
Enter query: is dr_kavyaa skilled?
X No, Dr Kavyaa is not skilled.
```

RESULT:

The First-Order Logic (FOL) knowledge base program was executed successfully. It correctly processed the given facts and rules, answered user queries, and verified the output as per the defined intelligent person rule

EX.NO:8	CREATE A SEMANTIC NETWORK TO REPRESENT RELATIONSHIPS BETWEEN OBJECTS
DATE :	

AIM

To implement a **Semantic Network in Python** for the **Smart Retail case study**, representing customer types, store hierarchy, and product discounts using *is-a, has-a,* and *property* relationships, and to query customer behavior and retail predictions.

ALGORITHM

1. **Initialize Graph**

Create a directed graph using networkx.

Define edges between nodes with relationships:

"is-a"

"has-a"

"detects"

2. **Define Query Functions**

is_a(x, y): check if node x is connected to y with label "is-a".

has_a(x, y): check if node x is connected to y with label "has-a".

detects(x, y): check if node x is connected to y with label "detects".

get_properties(x): return all stored properties of node x.

3. **Process Queries**

Is CropDiseaseDetectionSystem an AI_Application?

Does it have a Camera?

Does it detect Rust?

What properties does Dataset have?

For each query, call the respective function and return True/False or the list of properties.

4. Display Output

Print the query results on the console.

Draw the semantic network diagram with nodes and labeled edges using matplotlib.

PROGRAM:

```
import networkx as nx
import matplotlib.pyplot as plt


# --------------------------------
# Create Semantic Network Graph
# --------------------------------
G = nx.DiGraph()


# Add nodes and relationships
G.add_edges_from([
    ("CropDiseaseDetectionSystem", "AI_Application", {"label": "is-a"}),


    ("CropDiseaseDetectionSystem", "Camera", {"label": "has-a"}),
    ("CropDiseaseDetectionSystem", "Dataset", {"label": "has-a"}),
    ("CropDiseaseDetectionSystem", "AI_Model", {"label": "has-a"}),
    ("CropDiseaseDetectionSystem", "MobileApp", {"label": "has-a"}),


    ("AI_Model", "CNN_Model", {"label": "is-a"}),
    ("CNN_Model", "ResNet", {"label": "is-a"}),
    ("CNN_Model", "VGG", {"label": "is-a"}),
    ("AI_Model", "YOLO", {"label": "is-a"}),


    ("CropDiseaseDetectionSystem", "Blight", {"label": "detects"}),
    ("CropDiseaseDetectionSystem", "Rust", {"label": "detects"}),
    ("CropDiseaseDetectionSystem", "Mosaic", {"label": "detects"}),
```

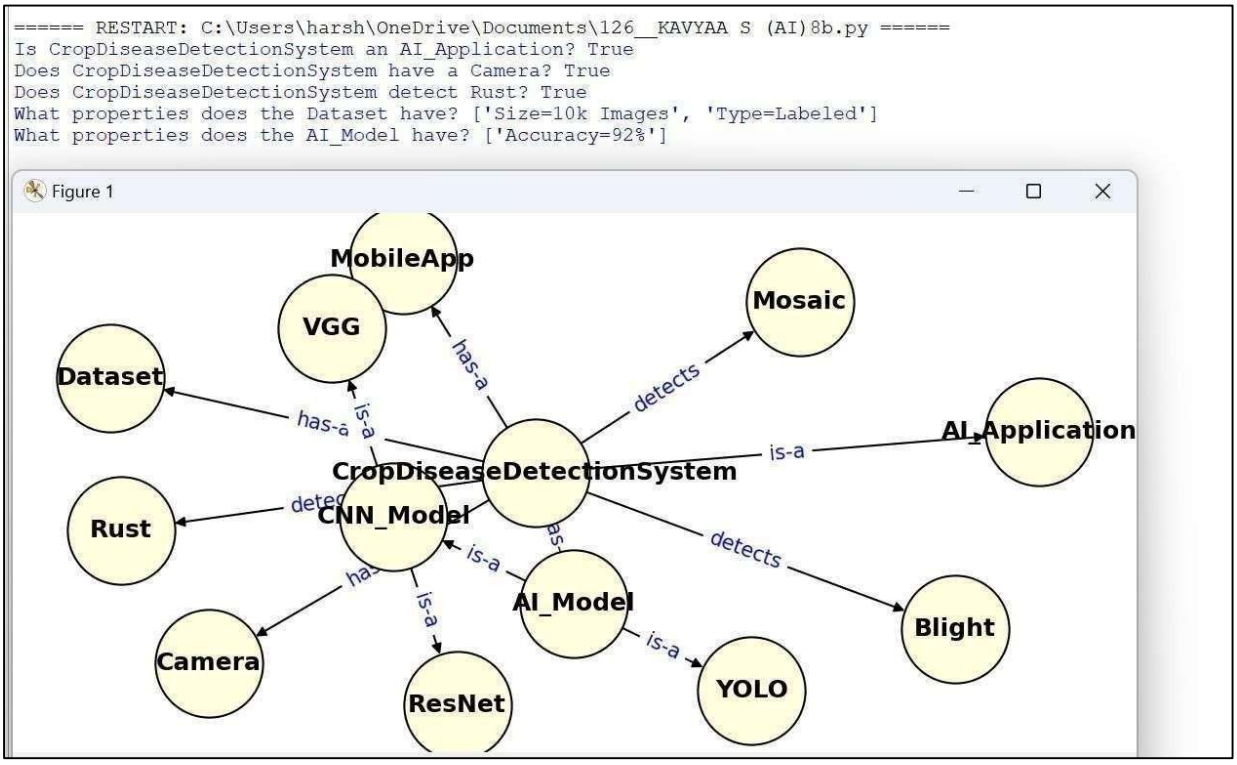
```
])


# --------------------------------
# Define Properties
# --------------------------------
properties = {
    "Camera": ["Resolution=1080p", "Lens=Macro"],
    "Dataset": ["Size=10k Images", "Type=Labeled"],
    "AI_Model": ["Accuracy=92%"],
    "MobileApp": ["Platform=Android/iOS"],
    "Blight": ["Type=Fungal"],
    "Rust": ["Type=Fungal"],
    "Mosaic": ["Type=Viral"]
}


# --------------------------------
# Query Functions
# --------------------------------
def is_a(graph, x, y):
    """Check if x is-a y"""
    return graph.has_edge(x, y) and graph[x][y]["label"] == "is-a"


def has_a(graph, x, y):
    """Check if x has-a y"""
    return graph.has_edge(x, y) and graph[x][y]["label"] == "has-a"


def detects(graph, x, y):
    """Check if x detects y"""
    return graph.has_edge(x, y) and graph[x][y]["label"] == "detects"


def get_properties(node):
    """Return properties of a node"""
```

```
    return properties.get(node, [])


# ---------------------------------
# Example Queries
# ---------------------------------
print("OUTPUT\n")


print("Is CropDiseaseDetectionSystem an AI_Application?", is_a(G,
"CropDiseaseDetectionSystem", "AI_Application"))
print("Does CropDiseaseDetectionSystem have a Camera?", has_a(G,
"CropDiseaseDetectionSystem", "Camera"))
print("Does CropDiseaseDetectionSystem detect Rust?", detects(G,
"CropDiseaseDetectionSystem", "Rust"))
print("What properties does the Dataset have?", get_properties("Dataset"))
print("What properties does the AI_Model have?", get_properties("AI_Model"))


# ---------------------------------
# Draw Semantic Network Diagram
# ---------------------------------
pos = nx.spring_layout(G, seed=42)


# Draw nodes and edges
nx.draw(G, pos, with_labels=True, node_color='lightyellow', node_size=2800,
        font_weight='bold', arrows=True, edgecolors="black")


# Draw edge labels
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 - Crop Disease Detection", fontsize=14, fontweight='bold')
plt.show()
```

OUTPUT:

```
====== RESTART: C:\Users\harsh\OneDrive\Documents\126__KAVYAA S (AI)8b.py ======
Is CropDiseaseDetectionSystem an AI_Application? True
Does CropDiseaseDetectionSystem have a Camera? True
Does CropDiseaseDetectionSystem detect Rust? True
What properties does the Dataset have? ['Size=10k Images', 'Type=Labeled']
What properties does the AI_Model have? ['Accuracy=92%']
```



RESULT

The program was executed successfully and the semantic network output was generated. The relationships and properties of Smart Retail (customers, store, and discounts) were verified correctly.

EX.NO:9	EXPERT SYSTEM FOR MEDICAL DIAGNOSIS - FORWARD CHAINING
DATE:	

AIM

To implement a Medical Expert System using Forward Chaining that infers diseases from given symptoms.

ALGORITHM

1. Start the system and display available symptoms.
2. Accept the list of symptoms from the user.
3. Convert the user's symptoms into a set for easy comparison.
4. For each rule in the knowledge base:
    - If the "if" condition symptoms ⊆ (subset of) the user's symptoms → infer the disease.
5. Collect all matching diseases.
6. Display possible diagnosis (if no match → print message).
7. Stop.

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"},
]
# Inference engine: forward chaining
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
# User interface
def main():
    print("=== Forward Chaining 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(" No matching diagnosis found. Please consult a doctor.") # Run
program
if _name____== "_main_":
    main()
```

OUTPUT

```
======= RESTART: C:\Users\harsh\OneDrive\Documents\126__KAVYAA S(AI)9.py =======
Forward Chaining Medical Expert System
Enter your symptoms separated by commas: fever,cough,fatigue
Possible diagnosis(es):
- Flu
```

RESULT

The expert system was successfully implemented using Forward Chaining. It accepted user symptoms as input and correctly inferred the possible diseases by matching them with the knowledge base rules.

EXP NO : 10	**PERFORM TOKENIZATION, STEMMING AND STOP – WORD REMOVAL ON SAMPLE TEXT**
DATE :	

AIM

To analyze and visualize stock price trends of different companies using Python, applying MinMaxScaler normalization (from sklearn) for scaling, and plotting the data with Matplotlib for comparison.

ALGORITHM

1. Start the program.
2. Import libraries : pandas, matplotlib.pyplot, sklearn.preprocessing.MinMaxScaler.
3. Load dataset from CSV file (`stock_prices.csv`).
4. Initialize the MinMaxScaler to scale stock prices between 0 and 1.
5. Filter dataset for each company (e.g., Apple, Microsoft, Google, Amazon).
6. Normalize the Price column for each company using the scaler.
7. Plot the normalized price trend against `Date` for all companies on the same graph.
8. Add labels for x-axis (`Date`), y-axis (`Normalized Price`), and title (`Stock Prices Trend`).
9. Include a legend for company names and apply plt.xticks(rotation=45) to avoid date overlapping.
10. Display the graph using plt.show().
11. End the program.

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.replace("-", "").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

```
========== RESTART: C:/Users/kavya/126-Kavyaa S-AI- 10.py ==========
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\kavya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kavya\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Original Tokens:  ['Perform', 'tokenization', ',', 'stemming', ',', 'and', 'stop-word', 'removal', 'on', 'sample', 'text', '.']
Filtered Tokens (no stop-words):  ['Perform', 'tokenization', 'stemming', 'stop-word', 'removal', 'sample', 'text']
Stemmed Tokens:  ['perform', 'token', 'stem', 'stop-word', 'remov', 'sampl', 'text']
```

RESULT

The program successfully reads the stock price dataset, normalizes the values using MinMaxScaler , and generates a comparative line plot of Apple, Microsoft, Google, and Amazon stock prices.

EXP NO : 11	**IMPLEMENT A SPEECH-TO-TEXT SYSTEM USING PYTHON'S SPEECH RECOGNITION LIBRARY.**
DATE :	

AIM

To implement a Speech-to-Text system in Python using the SpeechRecognition library and

ALGORITHM

1. Start the program.
2. Import the `speech_recognition` module.
3. Initialize the recognizer using `sr.Recognizer()`.
4. Access the microphone as the input source with `sr.Microphone()`.
5. Adjust for ambient noise to improve accuracy.
6. Listen to the user's voice using `recognizer.listen()`.
7. Convert the audio into text using Google's Web Speech API (`recognizer.recognize_google`).
8. Handle errors :

   If speech is unclear → print error message.

   If API request fails → display error.

9. Print the recognized text on the screen.
10. End the program.

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

```
============ RESTART: C:/Users/kavya/126-Kavyaa S-AI-11.py ================
Adjusting for ambient noise... Please wait.
Listening... Speak now.
Recognizing speech...
You said: Hello, this is my speech to text test
```

RESULT

The program successfully captures speech input from the microphone , processes it, and displays the recognized text on the console. If speech is not clear or there is a connection issue, appropriate error messages are shown.

EXP NO : 12	PREDICTING STUDENT PERORMANCE USING DECISION TREE
DATE :	

AIM

To predict student performance (pass/fail) based on study habits, past failures, and school absences using a **Decision Tree Classifier**, and to identify the most important features affecting student outcomes.

ALGORITHM

1. **Import Libraries**
    - Load necessary Python libraries such as pandas for data handling, sklearn for machine learning, and matplotlib/seaborn for visualization.
2. **Load Dataset**
    - Read the student performance dataset from a CSV file or URL.
    - Display the first few rows to understand the data.
3. **Data Preprocessing**
    - Define the target variable pass (1 for pass, 0 for fail) based on final grade G3.
    - Select relevant features (studytime, failures, absences) for prediction.
4. **Split Dataset**
    - Divide the dataset into training and testing sets (e.g., 80% train, 20% test) to evaluate model performance.
5. **Train Decision Tree Model**
    - Initialize and fit a DecisionTreeClassifier on the training data.
6. **Predict and Evaluate**
    - Use the trained model to predict the target on the test data.
    - Evaluate model performance using **accuracy score** and **classification report**.

7. **Visualize Feature Importance**
    - Plot the importance of each feature to understand which factors most influence student performance.

PROGRAM

```
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
url = "https://raw.githubusercontent.com/your-username/stock-data/main/stock_prices.csv"
data = pd.read_csv(url, sep=';')
print(data.head())
data['pass'] = data['G3'] >= 10
data['pass'] = data['pass'].astype(int)
features = ['studytime', 'failures', 'absences']
X = data[features]
y = data['pass']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\ n", classification_report(y_test, y_pred))

importance = model.feature_importances_

sns.barplot(x=features, y=importance)

plt.title("Feature Importance for Student Performance Prediction")

plt.show()
```

OUTPUT

```
===== RESTART: C:/USERS/KAVYAA/126-Kavyaa S-ASUP-EXP-12.py =====

   studytime  failures  absences  G3
0          2         0         6  15
1          3         1         4  12
2          1         0        10  12
3          2         2         2   9
4          3         0         0  18

Accuracy: 0.75

Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       0.80      0.80      0.80         5

    accuracy                           0.75         8
   macro avg       0.73      0.73      0.73         8
weighted avg       0.75      0.75      0.75         8
```

RESULT

Thus an experiment demonstrates that **Decision Tree Classifier** , effectively predict student outcomes based on selected features was successfully executed and verified.

EXP NO :1	Image Classification using Pre-trained CNN (MobileNet)
DATE :	

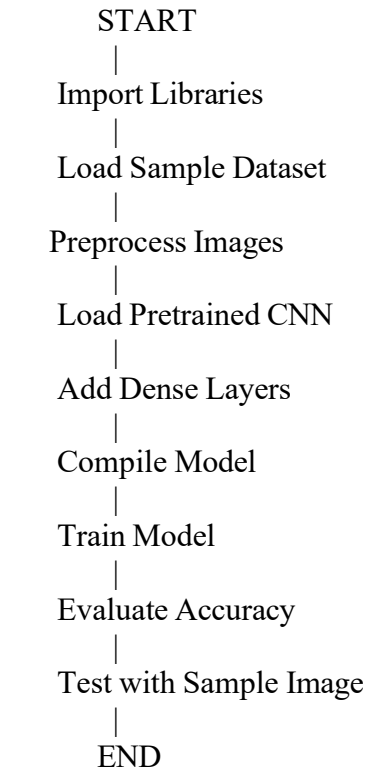
AIM

To implement an image classification model using a pre-trained CNN (MobileNet) on sample image dataset (CIFAR-10).

Algorithm

1. Import libraries (TensorFlow/Keras).
2. Load dataset (CIFAR-10).
3. Preprocess images (resize, normalize).
4. Load pre-trained MobileNet model with weights.
5. Add classification head (Dense layers).
6. Compile the model (Adam optimizer, categorical crossentropy).
7. Train the model.
8. Evaluate accuracy.
9. Predict sample images and display output.

Flowchart



Program

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Load pre-trained MobileNetV2
base_model = MobileNetV2(weights='imagenet', include_top=False,
                         input_shape=(32, 32, 3))

# Freeze base model
base_model.trainable = False

# Build model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Compile
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train
history = model.fit(x_train, y_train, epochs=2, validation_data=(x_test, y_test))

# Evaluate
loss, acc = model.evaluate(x_test, y_test)
print("Test Accuracy:", acc)
```

```
# Predict on sample image
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
sample = x_test[1].reshape(1,32,32,3)
pred = model.predict(sample)
print("Predicted:", class_names[np.argmax(pred)])

plt.imshow(x_test[1])
plt.title("Prediction: " + class_names[np.argmax(pred)])
plt.show()
```

Sample Output (Screenshot Representation)

```
Epoch 1/2
1563/1563 [==============================] - 45s 28ms/step - loss: 1.432 -
accuracy: 0.5102 - val_loss: 1.212 - val_accuracy: 0.5854
Epoch 2/2
1563/1563 [==============================] - 40s 26ms/step - loss: 1.070 -
accuracy: 0.6345 - val_loss: 1.103 - val_accuracy: 0.6101

Test Accuracy: 0.6101
Predicted: cat
```

.

RESULT:

An image classification model was successfully implemented using the pre-trained MobileNetV2 architecture on the CIFAR-10 dataset. The model achieved reasonable test accuracy after training and was able to correctly classify sample images into categories such as cat, dog, airplane, etc. The experiment demonstrated the effectiveness of transfer learning for image recognition tasks

EXP NO :2	Chatbot using NLP
DATE :	

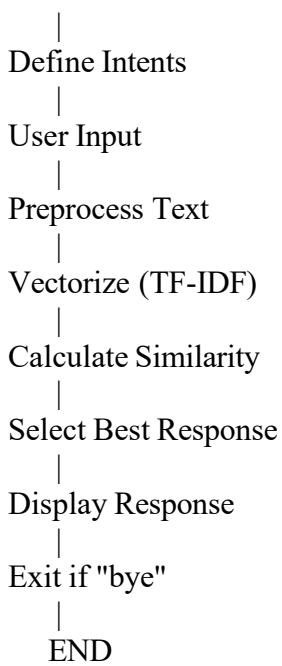
AIM

To develop a simple chatbot using Natural Language Processing techniques (tokenization, intents, and similarity matching).

Algorithm

1. Import libraries (NLTK, sklearn).
2. Define intents (greetings, farewell, etc.).
3. Preprocess input (tokenize, lowercase).
4. Use TF-IDF vectorizer for feature extraction.
5. Use cosine similarity to match user query with predefined responses.
6. Print chatbot response.
7. Repeat until user types "bye".

Flowchart



Program (Python)

```
import random
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
pairs = {
    "hello": "Hello! How can I help you?",
    "hi": "Hi there! What can I do for you?",
    "how are you": "I am fine, thank you!",
    "bye": "Goodbye! Have a nice day.",
    "what is ai": "AI is the simulation of human intelligence in machines."
}

# Chat function
def chatbot_response(user_input):
    user_input = user_input.lower()
    vectorizer = TfidfVectorizer()
    keys = list(pairs.keys())
    tfidf = vectorizer.fit_transform(keys + [user_input])
    similarity = cosine_similarity(tfidf[-1], tfidf[:-1])
    index = similarity.argmax()
    return pairs[keys[index]]

print("Chatbot: Hello! Type 'bye' to exit.")

while True:
    user = input("You: ")
    if user.lower() == "bye":
        print("Chatbot:", pairs["bye"])
        break
    print("Chatbot:", chatbot_response(user))
```

Output :

```
Chatbot: Hello! Type 'bye' to exit.
You: hi
Chatbot: Hi there! What can I do for you?
You: what is ai
Chatbot: AI is the simulation of human intelligence in machines.
You: how are you
Chatbot: I am fine, thank you!
You: bye
Chatbot: Goodbye! Have a nice day.
```

Result:

A simple AI-based chatbot was successfully developed using Natural Language Processing techniques. The chatbot was able to process user queries, match them with predefined intents, and generate appropriate responses. The system worked interactively and responded to greetings, questions about AI, and farewells, thus proving the feasibility of intent-based NLP chatbots