Experiment 1

Aim: Study Of Prolog

Theory:

Prolog (short for Programming in Logic) is a high-level programming language primarily associated with artificial intelligence (AI) and computational linguistics. It is based on formal logic and is particularly well-suited for tasks involving symbolic reasoning, pattern matching, and rule-based systems.

Key Features of Prolog:

1. **Declarative Programming:** We define facts, rules, and queries instead of writing explicit instructions.
2. **Logical Inference:** Prolog uses backward chaining (goal-driven reasoning) to derive answers from facts and rules.
3. **Symbolic Reasoning:** Useful for applications requiring symbolic rather than numeric computation.
4. **Pattern Matching:** Efficiently matches patterns and assigns variable bindings.
5. **Recursion:** Commonly used to solve complex problems.

Basic Components:

1. **Facts:** Statements about objects or relationships.

```
parent(john, mary).  % John is a parent of Mary
```

2. **Rules:** Conditional statements defining relationships.

```
ancestor(X, Y) :- parent(X, Y).

ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

3. **Queries:** Questions to the system to find answers.

```
?- ancestor(john, mary).
```

Use Cases:

- **Artificial Intelligence:** Expert systems, game AI, and knowledge representation
- **Natural Language Processing:** Parsing and understanding languages
- **Theorem Proving:** Automated proof systems
- **Database Querying:** Logical data retrieval

Strengths:

- Powerful for solving complex logical problems
- Built-in backtracking for efficient searching
- Elegant handling of recursive problems

Limitations:

- Performance can be slow for large datasets
- Challenging for developers accustomed to procedural languages
- Limited libraries compared to mainstream languages

Installation and usage on macOS:

Installation:

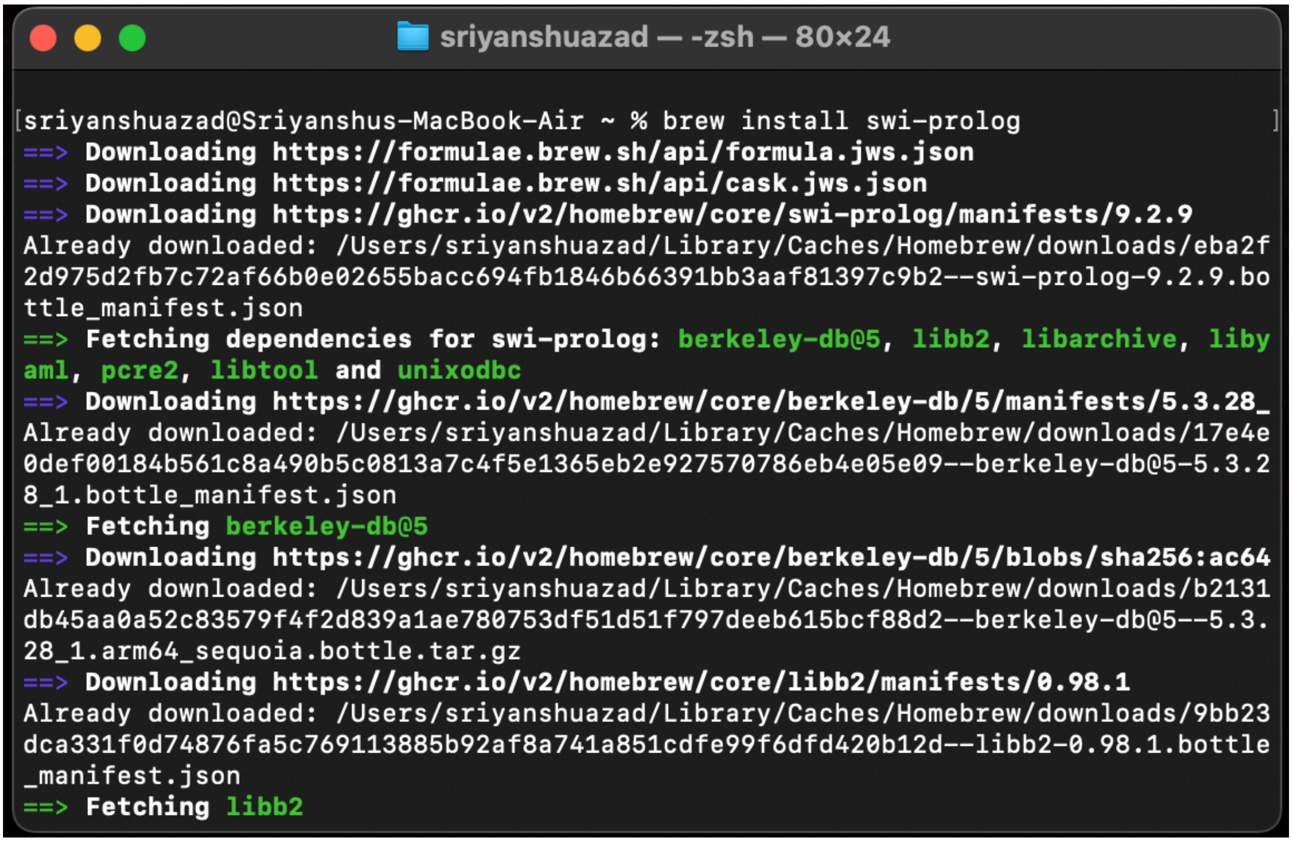
Using Homebrew

1. Open the terminal
2. Install Swi-Prolog by running:

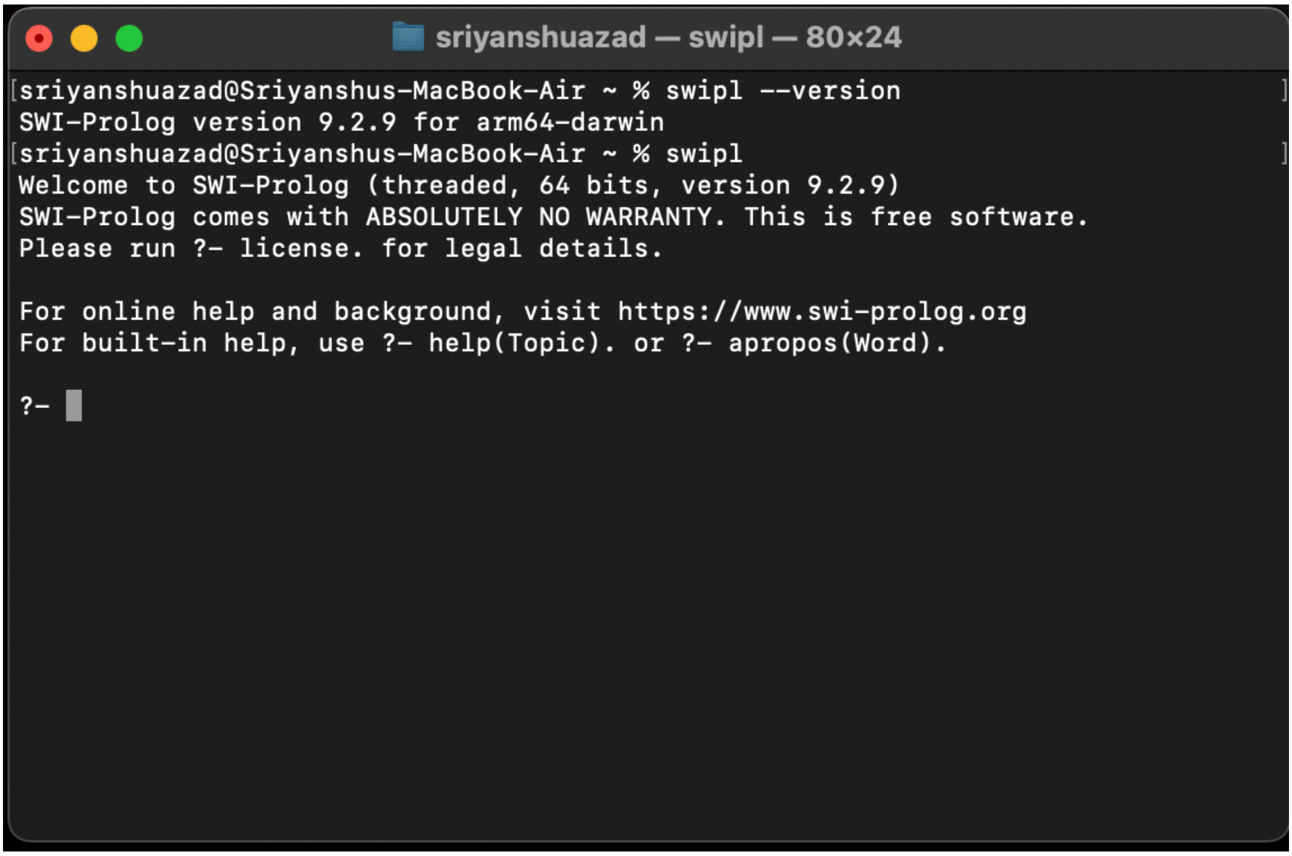
```
brew install sci-prolog
```

3. Verify the installation:

```
swipl —version
```



```
sriyanshuazad — -zsh — 80×24
[sriyanshuazad@Sriyanshus-MacBook-Air ~ % brew install swi-prolog                ]
==> Downloading https://formulae.brew.sh/api/formula.jws.json
==> Downloading https://formulae.brew.sh/api/cask.jws.json
==> Downloading https://ghcr.io/v2/homebrew/core/swi-prolog/manifests/9.2.9
Already downloaded: /Users/sriyanshuazad/Library/Caches/Homebrew/downloads/eba2f
2d975d2fb7c72af66b0e02655bacc694fb1846b66391bb3aaf81397c9b2--swi-prolog-9.2.9.bo
ttle_manifest.json
==> Fetching dependencies for swi-prolog: berkeley-db@5, libb2, libarchive, liby
aml, pcre2, libtool and unixodbc
==> Downloading https://ghcr.io/v2/homebrew/core/berkeley-db/5/manifests/5.3.28_
Already downloaded: /Users/sriyanshuazad/Library/Caches/Homebrew/downloads/17e4e
0def00184b561c8a490b5c0813a7c4f5e1365eb2e927570786eb4e05e09--berkeley-db@5-5.3.2
8_1.bottle_manifest.json
==> Fetching berkeley-db@5
==> Downloading https://ghcr.io/v2/homebrew/core/berkeley-db/5/blobs/sha256:ac64
Already downloaded: /Users/sriyanshuazad/Library/Caches/Homebrew/downloads/b2131
db45aa0a52c83579f4f2d839a1ae780753df51d51f797deeb615bcf88d2--berkeley-db@5--5.3.
28_1.arm64_sequoia.bottle.tar.gz
==> Downloading https://ghcr.io/v2/homebrew/core/libb2/manifests/0.98.1
Already downloaded: /Users/sriyanshuazad/Library/Caches/Homebrew/downloads/9bb23
dca331f0d74876fa5c769113885b92af8a741a851cdfe99f6dfd420b12d--libb2-0.98.1.bottle
_manifest.json
==> Fetching libb2
```



Usage:

- To launch Swi-Prolog type:

  `swipl`

We will see the Swi-Prolog prompt (`?-`), indicating it's ready to accept Prolog queries.

- To exit Swi-Prolog type:

  `halt.`

- To load a file type:

  `[filename].`

- To clear terminal:

  `Ctrl + L`

- Use a text editor like `nano`, `vim`, or any code editor to write your Prolog code. Save it with the `.pl` extension.

Example:

```
ai — nano prac1.pl — 80×24
  UW PICO 5.09                    File: prac1.pl                       Modified

%Facts
parent(john, mary).
parent(mary, susan).

%Rules
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z), ancestor(Z,Y).

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

```
ai — swipl — 80×24
sriyanshuazad@Sriyanshus-MacBook-Air ai % swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [prac1].
true.

?- ancestor(john, susan).
true
```

Learning Outcomes:

Experiment 2

Aim: Write simple fact for the statements using Prolog

a. **Ram likes mango.**

b. **Seema is a girl.**

c. **Bill likes Cindy.**

d. **Rose is red.**

e. **John owns gold.**

Theory:

A **fact** in Prolog asserts that something is unconditionally true in the knowledge base. Facts are composed of **predicates** and **arguments.**

```
predicate(argument1, argument2, …).
```

- The **predicate** is the relationship or property we are defining.
- The **arguments** are constants, variables, or structured terms associated with that predicate.

Rules for Writing Facts in Prolog

A. **Use Meaningful Predicates**: Always choose predicates that clearly describe the relationship.
B. **Avoid Redundancy**: Keep facts concise and avoid redundant information.
C. **Use Descriptive Argument Names:** Structure facts in a way that makes their purpose obvious.
D. **Be Consistent with Predicate Arity:** The **arity** refers to the number of arguments a predicate takes. Facts with the same predicate name should always have the same arity.
E. **Use Constants for Fixed Values:** In Prolog, constants start with lowercase letters, while variables start with uppercase.
F. **Maintain Logical Consistency**: Facts should not contradict each other.
G. **Create General Predicates for Scalability:** Instead of writing multiple predicates for similar facts, use a generalised predicate.

Code:

```
ai — nano prac2.pl — 80×24
  UW PICO 5.09                   File: prac2.pl                   Modified

likes(ram, mango).
likes(bill, cindy).
girl(seema).
color(rose, red).
owns(john, gold).

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
             Y Yes
^C Cancel    N No
```

Output:

```
ai — swipl — 80×26
sriyanshuazad@Sriyanshus-MacBook-Air ai % nano prac2.pl
sriyanshuazad@Sriyanshus-MacBook-Air ai % swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [prac2].
true.

?- likes(ram, mango).
true.

?- girl(seema).
true.

?- likes(bill, cindy).
true.

?- color(rose, red).
true.

?- owns(john, gold).
true.
```

Learning Outcomes:

Experiment 3

Aim: Write predicates, one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing using PROLOG.

Theory:

In PROLOG, **predicates** are fundamental building blocks that define logical relationships between terms. Predicates are defined by clauses, which consist of a **head** and an optional **body**. The predicate's purpose is to express a logical fact or a rule that PROLOG can use to infer information or verify conditions.

Declarative Nature of PROLOG

- Predicates in PROLOG define **what** needs to be true rather than **how** it should be computed.
- This declarative approach allows PROLOG to derive conclusions by applying rules and facts without explicitly specifying a computational procedure.
-

Structure of a Predicate

A predicate generally has the following form:

```
predicate_name(Arguments) :- Conditions.
```

- **Predicate Name:** Identifies the logical relationship.
- **Arguments:** The values or variables involved in the relationship.
- **Conditions (Body):** A sequence of goals that must be satisfied for the predicate to hold. If no conditions are specified, it is a fact.

Predicate Types

1. **Facts:**
   A fact represents a basic statement that is always true.

```
freezing_point(32).
```

   This declares that the freezing point is 32°F.

2. **Rules:**
   A rule defines a relationship that holds if certain conditions are met.

```
below_freezing(F) :- F < 32.
```

   This rule states that a temperature F is below freezing if it is less than 32°F.

3. **Queries:**
   Queries ask whether specific facts or rules hold.

```
?- below_freezing(31).
```

   This query checks whether 31°F is below freezing.

Code:

```
ai — nano prac3.pl — 80×24
  UW PICO 5.09                     File: prac3.pl

celsius_to_fahrenheit(C, F):-
F is (9/5) * C + 32.

below_freezing(C):-
C < 0.

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

Output:

```
ai — -zsh — 80×24
cd: no such file or directory: Desktop/prog/VI sem/AI\n
sriyanshuazad@Sriyanshus-MacBook-Air ~ % cd "Desktop/prog/VI sem/ai"
sriyanshuazad@Sriyanshus-MacBook-Air ai % swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('prac3.pl').
true.

?- celsius_to_fahrenheit(30,F).
F = 86.0.

?- below_freezing(5).
false.

?- below_freezing(-5).
true.

?- halt.
sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 4

Aim: Write a program to implement Breath First Search Traversal.

Theory:

Breadth-First Search (BFS) is a graph traversal algorithm used to explore nodes and edges of a graph systematically. It is particularly well-suited for unweighted graphs to find the shortest path from a starting node to all reachable nodes.

Characteristics of BFS

1. **Traversal Strategy:** BFS explores all nodes at the present depth level before moving on to nodes at the next depth level.
2. **Data Structure:** It uses a **queue** (First-In-First-Out, FIFO) to keep track of nodes to be explored.
3. **Path Finding:** BFS guarantees the shortest path in an unweighted graph.
4. **Graph Representation**: BFS can operate on:
    - **Adjacency Matrix**
    - **Adjacency List** (more memory-efficient for sparse graphs)

Time Complexity of BFS Algorithm: O(V + E)

BFS explores all the vertices and edges in the graph. In the worst case, it visits every vertex and edge once. Therefore, the time complexity of BFS is **O(V + E)**, where V and E are the number of vertices and edges in the given graph.

Auxiliary Space of BFS Algorithm: O(V)

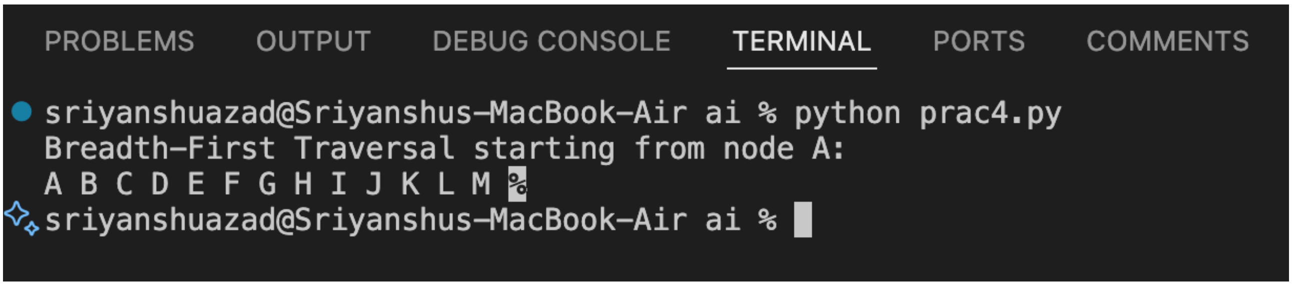
BFS uses a queue to keep track of the vertices that need to be visited. In the worst case, the queue can contain all the vertices in the graph. Therefore, the space complexity of BFS is O(V).

Algorithm:

Code:

```
from collections import deque

def bfs(tree, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()

        if node not in visited:
            print(node, end=" ")
            visited.add(node)

            for neighbor in tree[node]:
                if neighbor not in visited:
                    queue.append(neighbor)


tree = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F','G'],
    'D': ['H','I'],
    'E': ['J','K'],
    'F': ['L','M'],
    'G':[],'H':[],'I':[],'J':[],
    'K':[],'L':[],'M':[],
}

print("Breadth-First Traversal starting from node A:")
bfs(tree, 'A')
```

Output:




```
sriyanshuazad@Sriyanshus-MacBook-Air ai % python prac4.py
Breadth-First Traversal starting from node A:
A B C D E F G H I J K L M %
sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 5

Aim: Write a program to implement Water Jug problem.

Theory:

The **Water Jug Problem** is a classic puzzle in **artificial intelligence** and **theory of computation**, often used to demonstrate **state-space search algorithms** such as **Breadth-First Search (BFS)** and **Depth-First Search (DFS)**.

Problem Statement

Given two jugs with fixed capacities (say **X liters** and **Y liters**) and an **unlimited water supply**, determine if it is possible to measure exactly **T liters** using the jugs. The operations allowed are:

1. **Fill a Jug** completely.
2. **Empty a Jug** entirely.
3. **Pour water from one jug to another**, transferring as much as possible without overflowing.

State Representation

Each state in the problem is represented as **(x, y)** where:

- **x** → Amount of water in Jug X
- **y** → Amount of water in Jug Y

The **initial state** is **(0,0)** (both jugs are empty).

Possible State Transitions

From any state **(x, y)**, the following moves are possible:

1. **Fill Jug X** → (capacity_x, y)
2. **Fill Jug Y** → (x, capacity_y)
3. **Empty Jug X** → (0, y)
4. **Empty Jug Y** → (x, 0)
5. **Pour water from X → Y**
   A. If X has more water than Y can take: (x - (capacity_y - y), capacity_y)
   B. Else: (0, x + y)
6. **Pour water from Y → X**
   A. If Y has more water than X can take: (capacity_x, y - (capacity_x - x))
   B. Else: (x + y, 0)

Applications of Water Jug Problem

- **Artificial Intelligence & Robotics** – Used in **state-space search problems**
- **Cryptography** – Number theory applications using **GCD**
- **Computer Science Algorithms** – Problem-solving using **graph traversal**
- **Mathematical Puzzles** – Variants appear in **game theory**

Algorithm:

Code:

```
from collections import deque

def water_jug_shortest_path(capacity_x, capacity_y, target):
    """Finds the shortest path to solve the Water Jug Problem 
using BFS."""
    visited = set()
    queue = deque([(0, 0)])  # Initial state: both jugs empty
    parent = {}  # To store the previous state for backtracking

    while queue:
        x, y = queue.popleft()

        if (x, y) in visited:
            continue

        visited.add((x, y))

        if x == target or y == target:
            print("Solution found!\n")
            path = []
            while (x, y) in parent:
                path.append((x, y))
                x, y = parent[(x, y)]
            path.append((0, 0))  # Start state
            path.reverse()

            print("Steps to solution:")
            for step in path:
                print(f"Jug X: {step[0]}L, Jug Y: {step[1]}L")
            return

        # Define optimized moves (Prioritizing pouring)
        possible_moves = [
            (capacity_x, y),  # Fill Jug X
            (x, capacity_y),  # Fill Jug Y
            (0, y),  # Empty Jug X
            (x, 0),   # Empty Jug Y
            (max(0, x - (capacity_y - y)), min(capacity_y, y + 
x)),  # Pour X → Y
            (min(capacity_x, x + y), max(0, y - (capacity_x - x))) 
# Pour Y → X
        ]

        for move in possible_moves:
            if move not in visited:
                queue.append(move)
                parent[move] = (x, y)  # Store the move that led 
here
```

```
    print("No solution possible.")

# Example Usage
jug_x = 4
jug_y = 3
target_amount = 2
water_jug_shortest_path(jug_x, jug_y, target_amount)
```

Output:

```
python -u "/Users/sriyanshuazad/Desktop/prog/VI se
sriyanshuazad@Sriyanshus-MacBook-Air VI sem % pyth
Solution found!

Steps to solution:
Jug X: 0L, Jug Y: 0L
Jug X: 0L, Jug Y: 3L
Jug X: 3L, Jug Y: 0L
Jug X: 3L, Jug Y: 3L
Jug X: 4L, Jug Y: 2L
sriyanshuazad@Sriyanshus-MacBook-Air VI sem %
```

Learning Outcomes:

Experiment 6

Aim: Write a program to remove punctuations from the given string.

Theory:

Removing punctuation is an **important preprocessing step** in many **Artificial Intelligence (AI)** and **Machine Learning (ML)** tasks, especially in **Natural Language Processing (NLP)**.

1. Standardizes Text Data

- In AI, especially when analyzing large amounts of text (emails, tweets, reviews, etc.), **clean and standardized data** is essential.
- Removing punctuation reduces variations in the data and makes words easier to compare or process.

  Example:

  "hello", "hello!" and "hello." will be treated as **different tokens** if punctuation is not removed — but they all mean the same.

2. Improves Text Tokenization

- Tokenization means breaking a sentence into words or tokens.
- Punctuation can interfere with this process by **splitting meaningful words or phrases incorrectly**.
- Removing punctuation before tokenization ensures that tokens are clean and meaningful.

3. Enhances Performance of Text-Based Models

- AI models like **sentiment analysis, spam detection, language translation**, etc., work better when input data is consistent.
- Unnecessary punctuation adds noise to data, which can **confuse the model and reduce accuracy**.

4. Essential for Word Matching or Search

- In applications like **chatbots, search engines, or recommendation systems**, punctuation can prevent proper matching of keywords or phrases.
- Cleaning punctuation ensures better **pattern matching and intent detection**.

Therefore, removing punctuation helps reduce noise, improve consistency, and make AI models more accurate and efficient.

Algorithm:

Code:

```
def remove_punctuation(text):
    punctuations = '''!()-[]{};:'"\\,<>./?@#$%^&*_~'''

    for punc in text:
        if punc in punctuations:
            text=text.replace(punc,"")
    return text

text = input("Enter a string to remove punctuations: ")
print("String without punctuations: \n", remove_punctuation(text))
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    COMMENTS

sriyanshuazad@Sriyanshus-MacBook-Air VI sem % cd ai
sriyanshuazad@Sriyanshus-MacBook-Air ai % python prac6.py
Enter a string to remove punctuations: @@Yayy I did\ this&$ program!!!.
String without punctuations:
 Yayy I did this program
sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 7

Aim: Write a program to sort the sentence in alphabetical order.

Theory:

Sorting the words in a sentence, although not a core AI task on its own, plays an important role in various **preprocessing** and **text normalization** techniques in the field of **Artificial Intelligence (AI)** and **Natural Language Processing (NLP)**.

1. Text Normalization

- In AI systems, especially in NLP, text data needs to be cleaned and standardized before it is processed.
- Sorting words helps **normalize text** so that similar content can be compared more easily.

2. Duplicate Detection / Document Matching

- In tasks like **plagiarism detection**, **duplicate content detection**, or **document clustering**, sorting helps identify whether two texts contain the **same set of words**, even if the word order differs.
- Sorting makes it easier to compare word lists from two documents.

3. Feature Extraction in NLP

- When creating features like **Bag of Words (BoW)** or **TF-IDF**, the words are typically stored in **alphabetical order** to maintain **consistency in vector representations**.
- Sorting simplifies the indexing of features and improves processing efficiency.

4. Anagram and Pattern Matching

- In AI applications like **language games**, **anagram solvers**, or **cryptography-based models**, sorting helps quickly identify **word patterns** or **rearrangements**.

5. Search Optimization

- Sorted tokens or keywords are often used in **search engine indexing**, **autocomplete systems**, or **chatbots** to optimize lookup speed and improve accuracy in AI-driven information retrieval systems.

Algorithm:

Code:

```
def sorted_str(text):
    text=' '.join(sorted(text.split()))
    return text

text=input("Enter a string: ").strip()
print("Sorted String is:", sorted_str(text))
```

Output:

```
sriyanshuazad@Sriyanshus-MacBook-Air ai % python prac7.py
Enter a string: wow what a beautiful program
Sorted String is: a beautiful program what wow
sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 8

Aim: Write a program to implement Hangman game using python.

Theory:

Hangman is a classic word-guessing game typically played between two players. One player thinks of a word, and the other player tries to guess it by suggesting letters within a fixed number of guesses.

Mathematical Theory:

Hangman is fundamentally a **probabilistic problem**. The challenge lies in selecting letters that maximize the chance of discovering the word while minimizing mistakes.

Key concepts involved include:

- **Frequency Analysis:** Choosing letters that are more commonly found in words (like vowels `A`, `E`, `I`, `O`, `U`).
- **Pattern Recognition:** Guessing based on revealed letters and common word patterns.
- **Probability:** Estimating the most likely letters based on partially revealed words.

Challenges and Limitations:

- **Word Ambiguity:** Some patterns may fit multiple words (e.g., `_ A _ E` could be "GAME" or "NAME").
- **Rare Words:** Uncommon words can reduce AI accuracy, as the guessing strategy is based on common word structures.
- **Efficiency:** Searching through a large word list and calculating probabilities can be computationally expensive.

Real-World Applications:

While the AI Hangman itself is a game, the techniques used have broader applications:

- **Word Prediction:** Similar to how AI suggests words while typing.
- **Speech Recognition:** Predicting word sequences based on partial data.
- **Error Correction:** Identifying the most likely intended word based on input.

Algorithm:

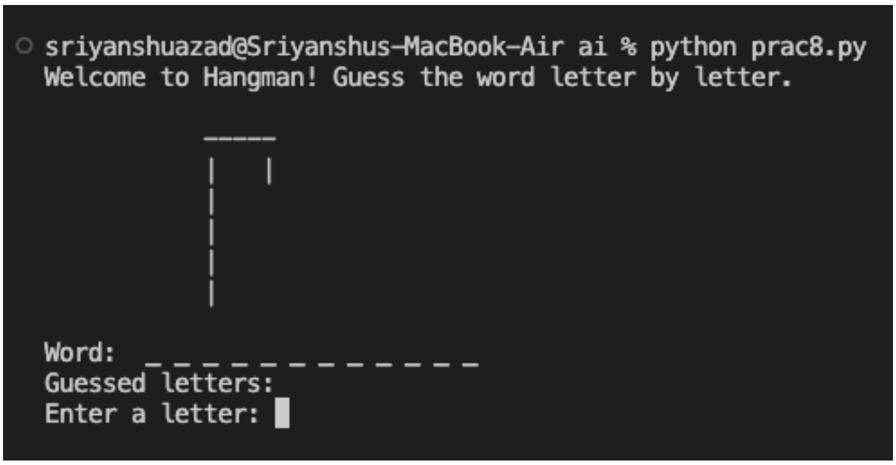
Code:

```
import random

def choose_word():
    words = ["python", "hangman", "programming", "developer",
"computer", "artificial", "intelligence"]
    return random.choice(words)

def display_hangman(attempts):
    stages = [
        """
           -----
           |   |
           |   O
           |  /|\\
           |  / \\
           |
        """,
        """
           -----
           |   |
```

```
          |   O
          |  /|\\
          |  /
          |
      """,
      """
         -----
          |   |
          |   O
          |  /|\\
          |
          |
      """,
      """
         -----
          |   |
          |   O
          |  /|
          |
          |
      """,
      """
         -----
          |   |
          |   O
          |   |
          |
          |
      """,
      """
         -----
          |   |
          |   O
          |
          |
          |
      """,
      """
         -----
          |   |
          |
          |
          |
          |
      """
    ]
    return stages[attempts]

def play_hangman():
    word = choose_word()
    word_letters = set(word)
    guessed_letters = set()
```

```
    attempts = 6

    print("Welcome to Hangman! Guess the word letter by letter.")

    while attempts > 0 and word_letters:
        print(display_hangman(attempts))
        print("Word: ", " ".join([letter if letter in
guessed_letters else "_" for letter in word]))
        print("Guessed letters: ", " ".join(guessed_letters))

        guess = input("Enter a letter: ").lower()

        if len(guess) != 1 or not guess.isalpha():
            print("Invalid input. Enter a single letter.")
            continue

        if guess in guessed_letters:
            print("You already guessed that letter. Try again.")
            continue

        guessed_letters.add(guess)

        if guess in word_letters:
            word_letters.remove(guess)
            print("Correct guess!")
        else:
            attempts -= 1
            print("Wrong guess! Attempts left:", attempts)

    if not word_letters:
        print("\nCongratulations! You guessed the word:", word)
    else:
        print(display_hangman(attempts))
        print("\nGame Over! The correct word was:", word)

# Start the game
play_hangman()
```

Output:



```
Enter a letter: c
Correct guess!

     -----
     |   |
     |
     |
     |
     |

Word:  _ _ _ _ _ _ _ _ _ _ c _
Guessed letters:  c
Enter a letter: e
Correct guess!

     -----
     |   |
     |
     |
     |
     |

Word:  _ _ _ e _ _ _ _ e _ c e
Guessed letters:  e c
Enter a letter: a
Wrong guess! Attempts left: 5

     -----
     |   |
     |   O
     |
     |
     |

Word:  _ _ _ e _ _ _ _ e _ c e
Guessed letters:  a e c
Enter a letter: t
Correct guess!

     -----
     |   |
     |   O
     |
     |
     |

Word:  _ _ t e _ _ _ _ e _ c e
Guessed letters:  a t e c
Enter a letter: g
Correct guess!
```

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

Word:  _ _ t e _ _ _ g e _ c e
Guessed letters:  g a t c e
Enter a letter: i
Correct guess!

     -----
     |   |
     |   O
     |
     |
     |

Word:  i _ t e _ _ i g e _ c e
Guessed letters:  g a t c i e
Enter a letter: v
Wrong guess! Attempts left: 4

     -----
     |   |
     |   O
     |   |
     |
     |

Word:  i _ t e _ _ i g e _ c e
Guessed letters:  g a t c i v e
Enter a letter: n
Correct guess!

     -----
     |   |
     |   O
     |   |
     |
     |

Word:  i n t e _ _ i g e n c e
Guessed letters:  n g a t c i v e
Enter a letter: l
Correct guess!

Congratulations! You guessed the word: intelligence
sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 9

Aim: Write a program to implement Tic-Tac-Toe game.

Theory:

Tic-Tac-Toe is a classic two-player game played on a **3x3 grid**. Players take turns marking a cell with their symbol.
Objective: Be the first to get **three of your symbols in a row** — horizontally, vertically, or diagonally.

The **Minimax algorithm** is a recursive decision-making strategy used in **game AI**, especially for **turn-based two-player games** like Tic-Tac-Toe, Chess, etc.
Goal: Simulate all possible future moves to choose the **optimal move** that maximizes your chance of winning, assuming your opponent plays perfectly.

Properties of Minimax:

- **Deterministic**: Always makes the best move.
- **Complete**: Explores the full game tree.
- **Unbeatable** in games like Tic-Tac-Toe.

Algorithm:

Code:

```
import math

# Initialize empty board
board = [[' ' for _ in range(3)] for _ in range(3)]

def print_board():
    print()  # Add spacing before board
    for i in range(3):
        print(" " + " | ".join(board[i]))
        if i < 2:
            print("---+---+---")
    print()  # Add spacing after board

def is_winner(player):
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or \
           all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or \
       all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_draw():
    return all(cell != ' ' for row in board for cell in row)

def minimax(is_maximizing):
    if is_winner('x'):
        return 1
    elif is_winner('o'):
        return -1
    elif is_draw():
        return 0

    if is_maximizing:
        best_score = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'x'
                    score = minimax(False)
                    board[i][j] = ' '
                    best_score = max(best_score, score)
        return best_score
    else:
        best_score = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
```

```
                    board[i][j] = 'o'
                    score = minimax(True)
                    board[i][j] = ' '
                    best_score = min(best_score, score)
        return best_score

def best_move():
    best_score = -math.inf
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'x'
                score = minimax(False)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move

def main():
    print("Welcome to Tic-Tac-Toe!")
    print("You are 'o'. Computer is 'x'.")
    print("Enter your move as row and column (e.g., 1 2)")
    print_board()

    while True:
        # Player move
        try:
            row, col = map(int, input("Your move (row col): 
").split())
            row -= 1
            col -= 1
            if not (0 <= row < 3 and 0 <= col < 3) or board[row]
[col] != ' ':
                print("Invalid move. Try again.")
                continue
        except:
            print("Invalid input. Enter two numbers between 1 and 
3.")
            continue

        board[row][col] = 'o'
        print_board()

        if is_winner('o'):
            print("You win!\n")
            break
        if is_draw():
            print("It's a draw!\n")
            break
```

```
        print("Computer is making a move...\n")
        row, col = best_move()
        board[row][col] = 'x'
        print_board()

        if is_winner('x'):
            print("Computer wins!\n")
            break
        if is_draw():
            print("It's a draw!\n")
            break

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

Output:

```
sriyanshuazad@Sriyanshus-MacBook-Air ai % pyth
Welcome to Tic-Tac-Toe!
You are 'o'. Computer is 'x'.
Enter your move as row and column (e.g., 1 2)

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

Your move (row col): 2 2

   |   |
---+---+---
   | o |
---+---+---
   |   |

Computer is making a move...


 x |   |
---+---+---
   | o |
---+---+---
   |   |

Your move (row col): 1 3

 x |   | o
---+---+---
   | o |
---+---+---
   |   |

Computer is making a move...


 x |   | o
---+---+---
   | o |
---+---+---
 x |   |
```

```
Your move (row col): 2 1

 x |   | o
---+---+---
 o | o |
---+---+---
 x |   |

Computer is making a move...


 x |   | o
---+---+---
 o | o | x
---+---+---
 x |   |

Your move (row col): 3 3

 x |   | o
---+---+---
 o | o | x
---+---+---
 x |   | o

Computer is making a move...


 x | x | o
---+---+---
 o | o | x
---+---+---
 x |   | o

Your move (row col): 3 2

 x | x | o
---+---+---
 o | o | x
---+---+---
 x | o | o

It's a draw!

sriyanshuazad@Sriyanshus-MacBook-Air ai %
```

Learning Outcomes:

Experiment 10

Aim: Write a program to remove stop words for a given passage from a text file using NLTK.

Theory:

Stop words are **common words in a language** (like "is", "the", "in", "on", "and", etc.) that **do not add much meaning** in tasks such as text analysis, classification, or information retrieval. They are usually removed during **text preprocessing** to focus on the **meaningful content** of the text.

NLTK (Natural Language Toolkit) is a powerful Python library used for:

- Text processing
- Tokenization
- Removing stop words
- Stemming, lemmatization
- Sentiment analysis, etc.

Components Used in the Program:

1. nltk.corpus.stopwords:

Provides a list of standard stop words for various languages.

2. nltk.tokenize.word_tokenize():

Breaks the passage into individual tokens (words and punctuation) using the **punkt tokenizer** model. This is more accurate than split().

3. string.punctuation:

A Python string containing common punctuation characters. Used to remove punctuation from the token list.

Use Case:

This kind of program is useful for:

- Search engines (e.g., indexing only meaningful words)
- Chatbots
- Text classification / NLP pipelines
- Removing noise before training a machine learning model

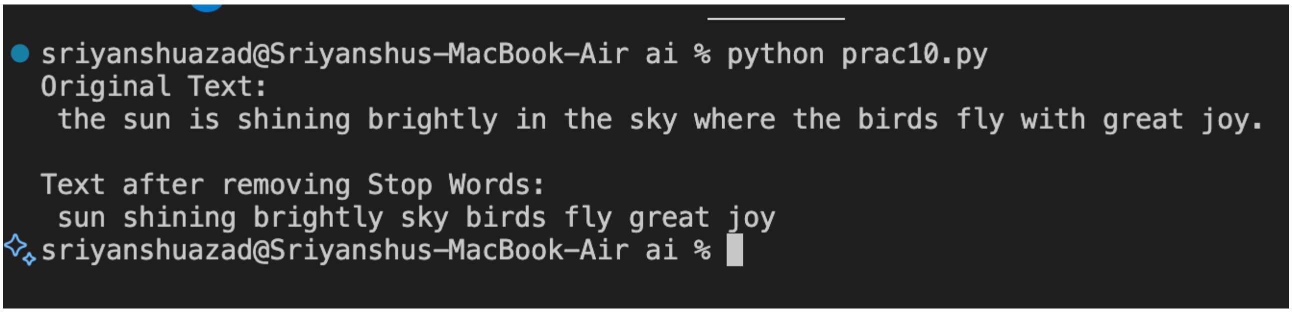
Algorithm:

Code:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

# Download resources (only once)
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')

# Read the text file
with open('prac10Text.txt', 'r') as file:
    text = file.read()

# Tokenize using NLTK's word_tokenize
words = word_tokenize(text)

# Load English stop words
stop_words = set(stopwords.words('english'))

# Filter out stop words and punctuation
```

```
filtered_words = [word for word in words if word.lower() not in
stop_words and word not in string.punctuation]

# Join the filtered words into a single string
filtered_text = ' '.join(filtered_words)

# Output
print("Original Text:\n", text)
print("\nText after removing Stop Words:\n", filtered_text)
```

Output:



Learning Outcomes:

Experiment 11

Aim: Write a program to POS (part of speech) tagging for the given sentence using NLTK.

Theory:

Part-of-Speech (POS) Tagging is the process of assigning a **part of speech (such as noun, verb, adjective, etc.)** to each word in a sentence based on its meaning and context.

- Helps in understanding **grammatical structure** of a sentence.
- Essential for tasks like:
    - **Named Entity Recognition (NER)**
    - **Information Retrieval**
    - **Text-to-Speech**
    - **Machine Translation**
    - **Chatbots and Question Answering**

Common POS Tags and Their Meanings:

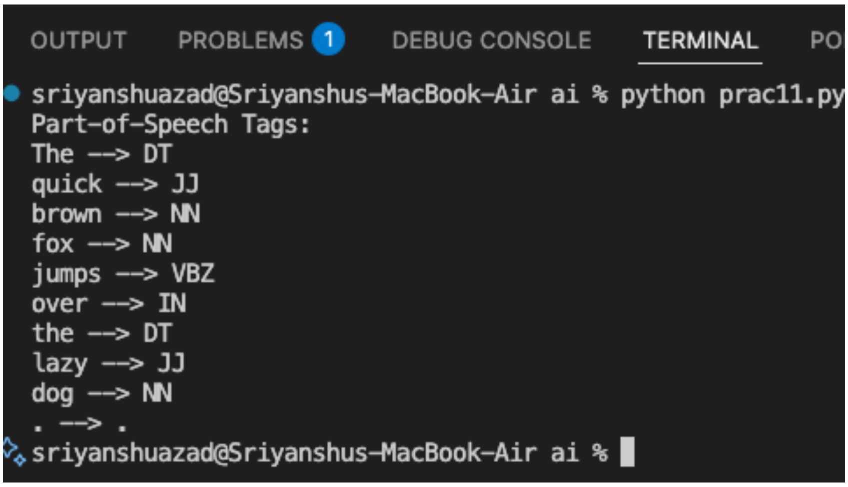
Tag	Meaning
NN	Noun (singular)
NNS	Noun (plural)
VB	Verb (base form)
VBZ	Verb (3rd person, sing)
JJ	Adjective
RB	Adverb
DT	Determiner
IN	Preposition
PRP	Personal Pronoun

Algorithm:

Code:

```
import nltk

# Download required NLTK resources (only need to run once)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger_eng')

# Input sentence
sentence = "The quick brown fox jumps over the lazy dog."

# Tokenize the sentence
tokens = nltk.word_tokenize(sentence)

# POS tagging
pos_tags = nltk.pos_tag(tokens)

# Display the POS tags
print("Part-of-Speech Tags:")
for word, tag in pos_tags:
    print(f"{word} --> {tag}")
```

Output:



Learning Outcomes:

Experiment 12

Aim: Write a program for Text Classification for the given sentence using NLTK.

Theory:

Text Classification is a Natural Language Processing (NLP) technique used to automatically assign a predefined category or label to a piece of text based on its content.

Examples:

- Classifying emails as "spam" or "not spam"
- Categorizing news articles into "sports", "technology", "politics", etc.
- Sentiment analysis: determining if a review is "positive" or "negative"

Naive Bayes Classifier

- It's fast and works well with text data.
- Based on Bayes' theorem with a strong assumption that features (words) are independent.
- NLTK provides a built-in `NaiveBayesClassifier` which simplifies implementation.

Applications of Text Classification:

- Sentiment Analysis
- Email Spam Filtering
- Document Categorization
- Chatbot Intent Recognition
- Topic Detection in News or Social Media

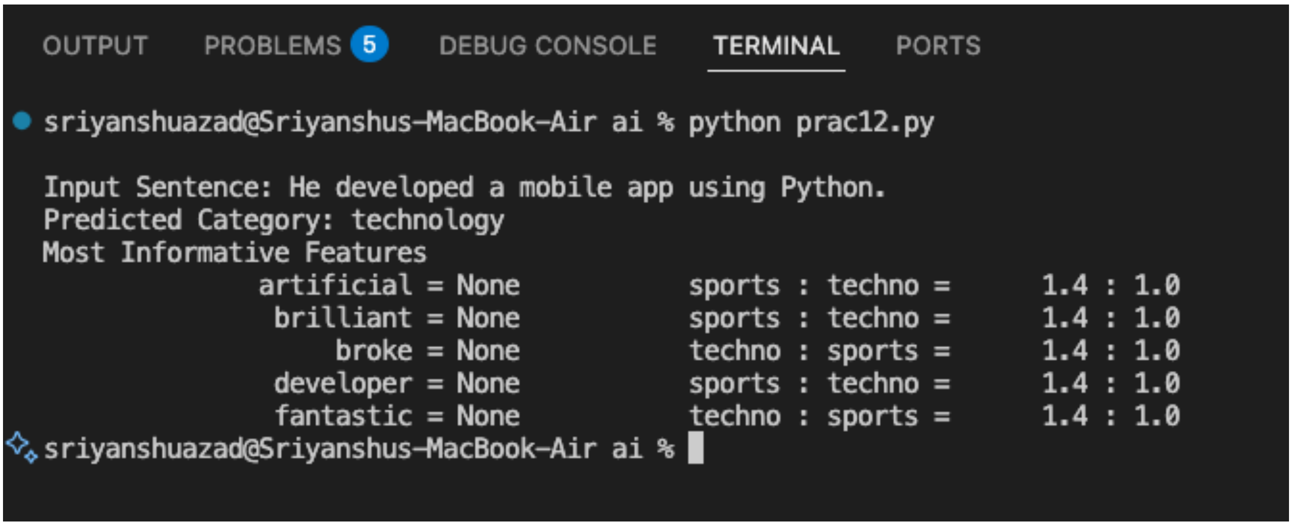
Algorithm:

Code:

```
import nltk
import random
from nltk import NaiveBayesClassifier
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download required NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Sample training data: (sentence, category)
training_data = [
    ("The team played a fantastic match and won the trophy",
"sports"),
    ("He scored a goal in the final minutes of the game",
"sports"),
    ("She is a brilliant software developer at Google",
"technology"),
    ("Python is a widely used programming language",
"technology"),
    ("The player broke the world record in sprinting", "sports"),
    ("Artificial Intelligence is transforming the world",
"technology"),
]

# Preprocessing: Remove stopwords and convert to features
stop_words = set(stopwords.words('english'))

def preprocess(sentence):
    words = word_tokenize(sentence.lower())
    return {word: True for word in words if word not in stop_words
and word.isalpha()}

# Create feature sets
feature_sets = [(preprocess(sentence), category) for sentence,
category in training_data]

# Shuffle and split data (not necessary here since data is small)
random.shuffle(feature_sets)

# Train the Naive Bayes Classifier
classifier = NaiveBayesClassifier.train(feature_sets)

# Input sentence to classify
test_sentence = "He developed a mobile app using Python."

# Preprocess and classify
features = preprocess(test_sentence)
predicted_category = classifier.classify(features)
```

```
print(f"\nInput Sentence: {test_sentence}")
print(f"Predicted Category: {predicted_category}")

# Show the most informative features
classifier.show_most_informative_features(5)
```

Output:



Learning Outcomes: