Applied Artificial Intelligence Practical's
Academic Year: 2024-2025
Index

Practical 1a

Module 1:

Expert System its Applications, Probability theory

Aim:

Design an Expert system using AIML.

Theory:

Design an Expert system using AIML E.g: An expert system for responding the patient query for identifying the flu.

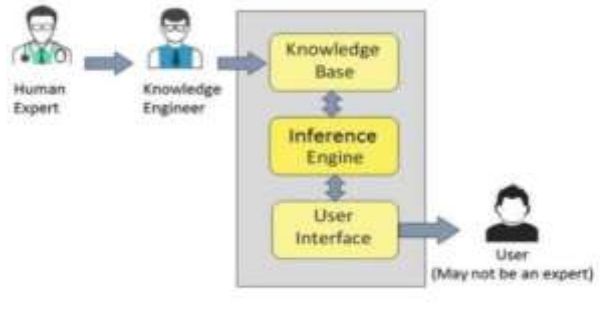
Concept: Artificial Intelligence Markup Language is referred to as AIML. It was the Alicebot that created AIML. It's a community for free software. Dr. Richard S. Wallace from 1995 to the present. An application called a chat box is made with AIML. In order to create an expert system using AIML, the following points should be taken into account.

• In an expert system there are three main components: User Interface, Inference Engine and Knowledge Base

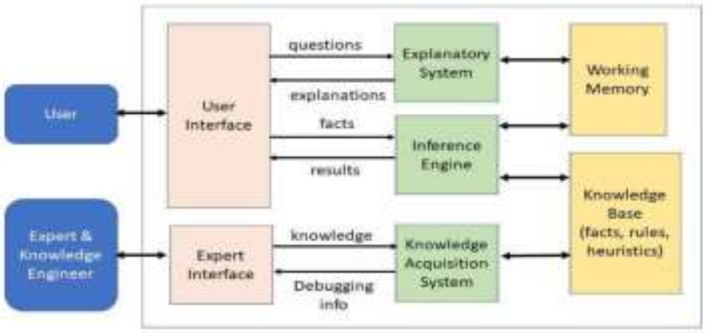
User Interface: Uses various user interfaces, such as menus, graphics, and dashboards, or Natural Language Processing (NLP) to communicate with the user.

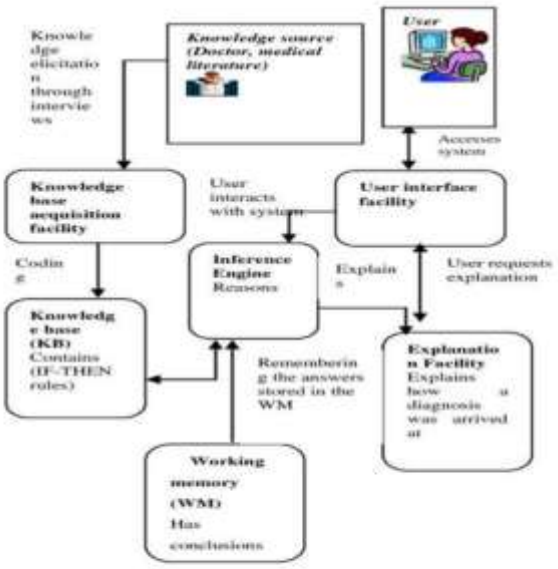
Expert System: A software program that makes decisions or provides advice based on databases of expert knowledge in various contexts, such as medical diagnosis. An expert system is a computer program that solves problems in a specialized field that typically calls for human expertise using techniques from artificial intelligence. A well-organized collection of data about the system's domain is called a knowledge base.

The knowledge base's facts are interpreted and assessed by the inference engine, which then outputs the desired results or an answer.



- The expert system communicates with the user through a readable user interface, receives queries as input, and outputs results.




Knowledge elicitation through interviews
Knowledge source (Doctor, medical literature)
User
Accesses system
Knowledge base acquisition facility
User interacts with system
User interface facility
Coding
Inference Engine Reasons
Explains
User requests explanation
Knowledge base (KB) Contains (IF-THEN rules)
Remembering the answers stored in the WM
Explanation Facility Explains how a diagnosis was arrived at
Working memory (WM) Has conclusions

<u>Code:</u>

<u>Flu.aiml(Text file)</u>

```
<aiml version="1.0.1" encoding="UTF-8">

  <category>

    <pattern>WHAT ARE FLU SYMPTOMS</pattern>

    <template>

      Flu symptoms usually include fever, chills, muscle aches, cough, congestion, runny nose, 
headaches, and fatigue.

    </template>

  </category>
```

```
 <category>
  <pattern>I HAVE FEVER AND COUGH</pattern>
  <template>
    These symptoms could be associated with the flu. However, I recommend visiting a
healthcare professional for an accurate diagnosis.
  </template>
 </category>

 <category>
  <pattern>IS FLU CONTAGIOUS</pattern>
  <template>
    Yes, flu is highly contagious and can spread easily from person to person.
  </template>
 </category>

 <category>
  <pattern>HOW CAN I PREVENT FLU</pattern>
  <template>
    The best way to prevent the flu is by getting a flu vaccine each year. Additionally, wash your
hands frequently, avoid close contact with sick people, and maintain a healthy lifestyle.
  </template>
 </category>

 <category>
  <pattern>THANK YOU</pattern>
  <template>
    You're welcome! Take care and stay healthy.
  </template>
 </category>
```

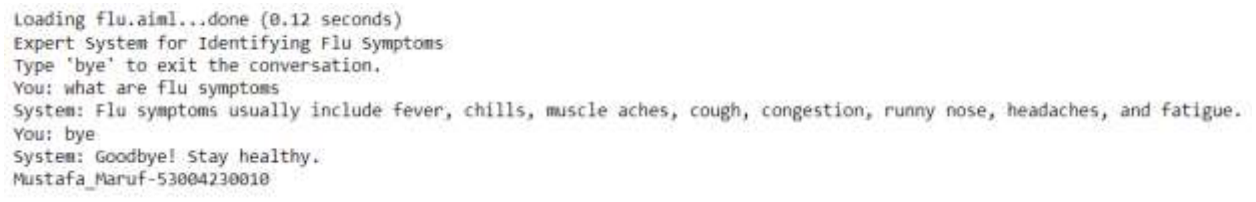
```
  <category>
    <pattern>BYE</pattern>
    <template>
      Goodbye! Feel free to reach out if you have more questions.
    </template>
  </category>

  <category>
    <pattern>FLU*</pattern>
    <template>
      Could you please provide more details about your symptoms so that I can assist you better?
    </template>
  </category>
</aiml>
```

Juypyer code

```
#Prac1a
import aiml

# Create a Kernel instance
kernel = aiml.Kernel()

# Load AIML files
kernel.learn("flu.aiml")

# Loop to interact with the expert system
print("Expert System for Identifying Flu Symptoms")
print("Type 'bye' to exit the conversation.")
```

```
while True:
    user_input = input("You: ")

    # Exit the conversation if the user types 'bye'
    if user_input.lower() == "bye":
        print("System: Goodbye! Stay healthy.")
        break

    # Get the system's response
    response = kernel.respond(user_input.upper())

    # Print the system's response
    print(f"System: {response}")

print('Mustafa_Maruf-53004230010')
```

Output:



```
Loading flu.aiml...done (0.12 seconds)
Expert System for Identifying Flu Symptoms
Type 'bye' to exit the conversation.
You: what are flu symptoms
System: Flu symptoms usually include fever, chills, muscle aches, cough, congestion, runny nose, headaches, and fatigue.
You: bye
System: Goodbye! Stay healthy.
Mustafa_Maruf-53004230010
```

Practical 1b

Aim:

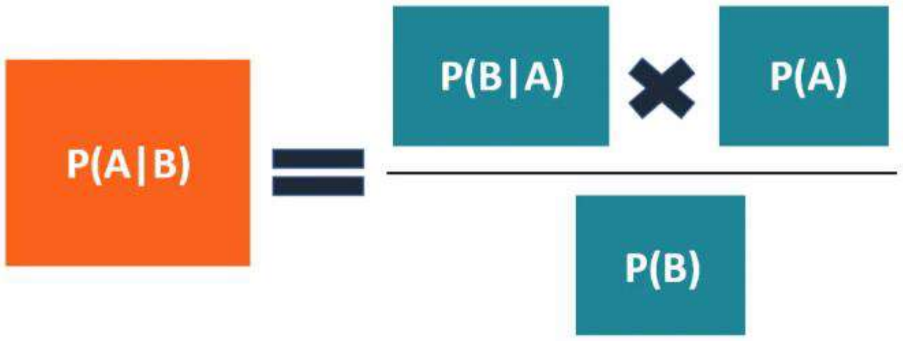
Implement Bayes Theorem using Python.

Theory:

Bayes' Theorem is a fundamental concept in probability theory that describes how to update the probability of a hypothesis based on new evidence. It's widely used in various fields such as medicine, finance, and machine learning.

Bayes' Theorem Formula

P(A|B) – the probability of event A occurring, given event B has occurred. P(B|A) – the probability of event B occurring, given event A has occurred. P(A) – the probability of event A.



Bayes' Theorem Formula

Where:

- (P(A | B)): The probability of event (A) (the hypothesis) occurring given that event (B) (the evidence) is true. This is called the posterior probability.

- (P(B | A)): The probability of observing event (B) given that (A) is true. This is called the likelihood.

- (P(A)): The probability of event (A) occurring independently of

(B). This is called the prior probability.

- (P(B)): The total probability of event (B) occurring. This is called the marginal likelihood or evidence.

Intuition Behind Bayes' Theorem: Bayes' Theorem allows you to update your belief about an event based on new evidence. In this case, although the test is highly accurate, the fact that the disease is rare lowers the probability of actually having the disease even after testing positive.

Applications of Bayes' Theorem:

- Medical diagnosis: Estimating the likelihood of a disease given test results.

- Spam filtering: Calculating the probability that an email is spam given certain keywords.

- Machine learning: Algorithms like Naive Bayes classifiers use Bayes' Theorem to classify data.

- Finance: Estimating the likelihood of an investment succeeding based on prior market performance.

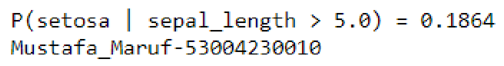
Bayes' Theorem is a powerful tool for decision-making under uncertainty, allowing us to revise our predictions as more data becomes available.

Code:

```
# Prac1b
import pandas as pd


def bayes_theorem(prior_A, likelihood_B_given_A, marginal_B):
    return (likelihood_B_given_A * prior_A) / marginal_B


# Load the Iris dataset
def load_iris_dataset(file_path):
    return pd.read_csv(file_path)


# Calculate prior probability P(A)
def calculate_prior(data, class_col, class_value):
    return len(data[data[class_col] == class_value]) / len(data)


# Calculate likelihood P(B|A)
def calculate_likelihood(data, class_col, class_value, feature_col, feature_condition):
    subset = data[data[class_col] == class_value]
    return len(subset[subset[feature_col] > feature_condition]) / len(subset)

# Calculate marginal probability P(B)
def calculate_marginal(data, feature_col, feature_condition):
    return len(data[data[feature_col] > feature_condition]) / len(data)

# Apply Bayes' Theorem on the Iris dataset
def apply_bayes_to_iris(file_path, class_col, class_value, feature_col, feature_condition):
    # Load dataset
    data = load_iris_dataset(file_path)

    # Calculate prior P(A)
    prior_A = calculate_prior(data, class_col, class_value)

    # Calculate likelihood P(B|A)
    likelihood_B_given_A = calculate_likelihood(data, class_col, class_value, feature_col,
feature_condition)

    # Calculate marginal probability P(B)
    marginal_B = calculate_marginal(data, feature_col, feature_condition)

    # Apply Bayes' Theorem
    posterior_A_given_B = bayes_theorem(prior_A, likelihood_B_given_A, marginal_B)

    return posterior_A_given_B

# Example usage:
# Assume we want to calculate the probability P(Class='setosa' | SepalLength > 5.0)
```

```python
file_path = 'iris.csv'  # Path to the iris dataset file
class_col = 'species'  # The column representing the class (A)
class_value = 'setosa'  # The class value we're interested in (A)
feature_col = 'sepal_length'  # The feature we're using (B)
feature_condition = 5.0  # The condition on the feature (B > 5.0)


# Calculate posterior probability P(setosa | sepal_length > 5.0)
posterior_probability = apply_bayes_to_iris(file_path, class_col, class_value, feature_col,
feature_condition)
print(f"P({class_value} | {feature_col} > {feature_condition}) = {posterior_probability:.4f}")
print('Mustafa_Maruf-53004230010')
```

<u>Output:</u>



```
P(setosa | sepal_length > 5.0) = 0.1864
Mustafa_Maruf-53004230010
```

Practical 1c

Aim:

Implement Conditional Probability and joint probability using Python.

Theory:

1. Joint Probability: Joint probability refers to the probability of two events happening at the same time. It represents the likelihood of both events occurring simultaneously. In probability theory, the joint probability of two events (A) and (B) is denoted as (P(A, B)).

For example, if you roll two dice, the joint probability of getting a 3 on the first die and a 4 on the second die would be the product of the probabilities of getting a 3 on the first die and a 4 on the second die.

Mathematical Definition:

The joint probability of two events (A) and (B) occurring together is defined as:

P(A and B) = P(A) * P(B)

If the two events are independent (i.e., one event does not affect the other), the joint probability is the product of the probabilities of the two events:

Conditional Probability:

In the case where events A and B are independent (where event A has no effect on the probability of event B), the conditional probability of event B given event A is simply the probability of event B, that is P(B)

P(A and B) = P(A)P(B|A)

2. Conditional Probability:

Conditional probability refers to the probability of an event occurring given that another event has already occurred. It quantifies the probability of one event happening under the assumption that another

event is true. It is denoted by (P(A | B)), which reads as "the probability of (A) given (B)."

Mathematical Definition: The conditional probability of event (A) given that event (B) has occurred is defined as:

Events A and B are independent(i.e., events whose probability of occurring together is the product of their individual probabilities). if

$P(A\cap B)=P(A)\cdot P(B)P(A\cap B)=P(A)\cdot P(B)$

If A and B are not independent then they are dependent.

Conditional probability adjusts the probability of an event based on new information. For example, if you already know that it is raining (event (B)), the probability of you carrying an umbrella (event (A)) may increase.

Example:

Suppose:

A single fair die is rolled. Let $A=\{3\}A=\{3\}$ and $B=\{1,3,5\}B=\{1,3,5\}$. Are A and B independent

Solution

In this example we can compute all three probabilities $P(A)=16P(A)=16$, $P(B)=12P(B)=12$, and $P(A\cap B)=P(\{3\})=16P(A\cap B)=P(\{3\})=16$. Since the product $P(A)\cdot P(B)=(16)(12)=112P(A)\cdot P(B)=(16)(12)=112$ is not the same number as $P(A\cap B)=16P(A\cap B)=16$, the events A and B are not independent

Relationship Between Joint and Conditional Probability:

Joint and conditional probability are closely related through Bayes' Theorem. You can rearrange the formula for conditional probability to express the joint probability in terms of conditional probability:

The joint probability $P(A\cap B)$ can be expressed in terms of the conditional probability: $P(A\cap B)=P(A|B)\cdot P(B)$

This shows that the joint probability is the product of the conditional probability of A given Band the probability of B. Similarly, you could also express the joint probability as: $P(A\cap B)=P(B|A)\cdot P(A)$

In summary, joint probability is essentially the "unconditional" version of the overlap between two events, while conditional probability looks at how likely one event is given that the other has already happened. These two concepts are tightly linked through the formulas above.

The joint probability $P(A\cap B)$ represents the probability that it rains and your friend carries an umbrella.

The conditional probability $P(A|B)$ represents the probability that it rains, given that your friend carries an umbrella.

The relationship between them is $P(A\cap B)=P(A|B)\cdot P(B)$

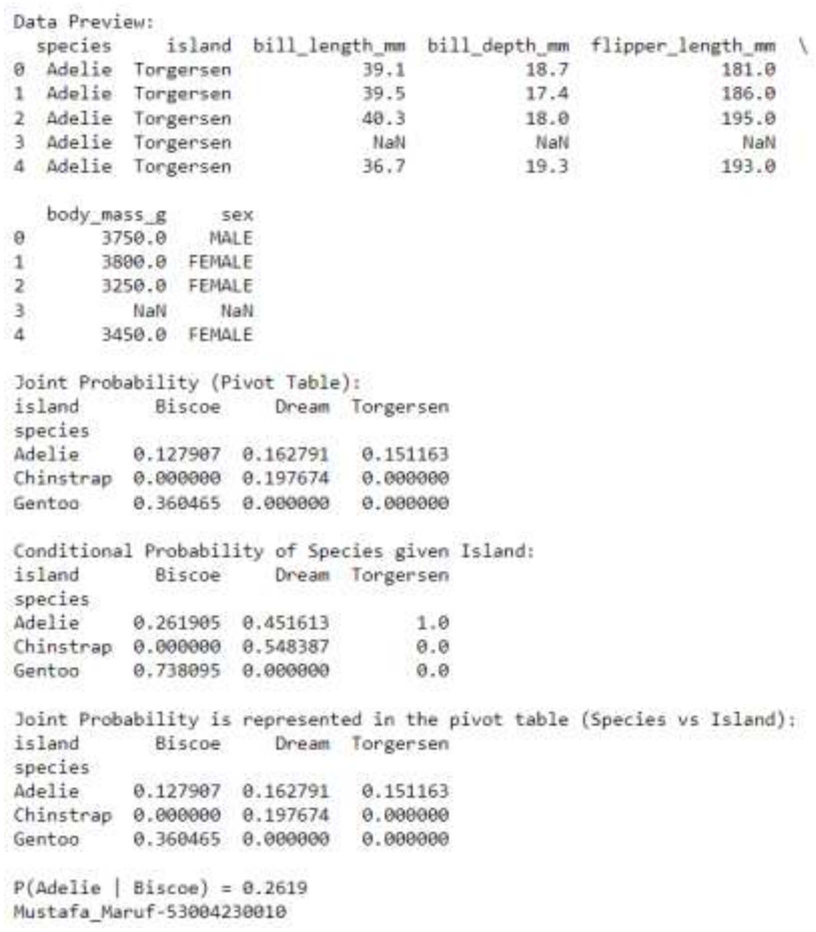
Both concepts are fundamental in understanding how events relate to each other in probability theory and are used in various applications such as decision-making, risk assessment, and machine learning.

Code:

```
# Prac1c
import pandas as pd

# Load the penguins dataset from a CSV file
df = pd.read_csv('penguins.csv')

# Preview the data
print("Data Preview:")
print(df.head())

# Create a pivot table for joint probability
# Pivot table for Species (rows) and Island (columns), computing frequencies
pivot_table = pd.crosstab(df['species'], df['island'], normalize=True)

# Display the joint probability pivot table
print("\nJoint Probability (Pivot Table):")
print(pivot_table)

# Calculate Conditional Probability of Species given Island
# Normalize along columns to get conditional probabilities
conditional_probability = pivot_table.div(pivot_table.sum(axis=0), axis=1)
print("\nConditional Probability of Species given Island:")
print(conditional_probability)
```

```
# Joint Probability is represented in the pivot table (Species vs Island)
print("\nJoint Probability is represented in the pivot table (Species vs Island):")
print(pivot_table)

# Example: Calculating P(Species = Adelie | Island = Biscoe)
p_adelie_given_biscoe = conditional_probability.loc['Adelie', 'Biscoe']
print(f"\nP(Adelie | Biscoe) = {p_adelie_given_biscoe:.4f}")

# Identifier
print('Mustafa_Maruf-53004230010')
```

Output:



```
Data Preview:
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0

   body_mass_g     sex
0       3750.0    MALE
1       3800.0  FEMALE
2       3250.0  FEMALE
3          NaN     NaN
4       3450.0  FEMALE

Joint Probability (Pivot Table):
island       Biscoe     Dream  Torgersen
species
Adelie     0.127907  0.162791   0.151163
Chinstrap  0.000000  0.197674   0.000000
Gentoo     0.360465  0.000000   0.000000

Conditional Probability of Species given Island:
island       Biscoe     Dream  Torgersen
species
Adelie     0.261905  0.451613        1.0
Chinstrap  0.000000  0.548387        0.0
Gentoo     0.738095  0.000000        0.0

Joint Probability is represented in the pivot table (Species vs Island):
island       Biscoe     Dream  Torgersen
species
Adelie     0.127907  0.162791   0.151163
Chinstrap  0.000000  0.197674   0.000000
Gentoo     0.360465  0.000000   0.000000

P(Adelie | Biscoe) = 0.2619
Mustafa_Maruf-53004230010
```

Practical 2a

Module 2:

Fuzzy Sets, Fuzzy Logic, Artificial Neural Networks

Aim:

Create a simple rule-based system in Prolog for diagnosing a common illness based on symptoms.

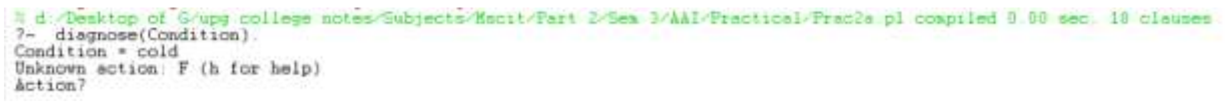
Theory:

Prolog expressions are comprised of the following truth-functional symbols, which have the same interpretation as in the predicate calculus.

Code:

```
#Prac2a

%Facts:Define symptoms symptom(fever)

symptom(cough).

symptom(sore_throat).

symptom(body_aches).

symptom(runny_nose).

symptom(headache).

symptom(fatigue).


%Facts:Define possible illnesses

condition(cold).

condition(flu).

condition(strep_throat).


%Rules: Diagnosing based on the presence of symptoms

diagnose(cold):-

        symptom(runny_nose),
        symptom(cough),
        symptom(sore_throat),
        \+ symptom(fever). %Absence of fever

diagnose(flu):-
        symptom(fever),
        symptom(cough),
        symptom(body_aches),
        symptom(headache),
        symptom(fatigue).

diagnose(sterp_throat):-
        symptom(sore_throat),
        symptom(fever),
        \+symptom(cough). %Absence of cough

%Alternative:Diagnosing based on rule covering all possible symptoms
diagnose(unknown):-
        \+diagnose(cold),
        \+diagnose(flu),
        \+diagnose(strep_throat).

%Quries: Example of how to diagnose
%?-diagnose(Condition).
%Output:Condition = flu.(if the symptoms match the flu criteria)

%Assuming the patient has the following
symptoms: symptom(fever).
symptom(cough).
symptom(body_aches).
symptom(headaches).
symptom(fatigue).

%You can ask Prolog:
?-diagnose(Condition).
%Mustafa_Maruf-53004230010
```

<u>Output:</u>



```
% d:/Desktop of G/upg college notes/Subjects/Mscit/Part 2/Sem 3/AAI/Practical/Prac2a.pl compiled 0.00 sec, 18 clauses
?-  diagnose(Condition).
Condition = cold
Unknown action: F (h for help)
Action?
```

Practical 2b

Aim:

Design a Fuzzy based application using Python.

Theory:

You have to designing fuzzy logic system to control traffic lights at an intersection. The goal is to adjust the duration of the green light based on the current traffic density and the time of day (peak and non-peak hours).

Traffic Density: You can classify density as low,medium, or high.

Time of Day: The time can be either peak hours or non-peak hours.

Green Light Duration: The duration of the green light will be determined based on the traffic density and time of day, classified as short, moderate, or long.

A fuzzy logic system is a mathematical framework that handles reasoning with uncertainty and imprecision. Unlike traditional binary logic (where variables are either true or false, i.e., 0 or 1), fuzzy logic allows for degrees of truth, where values can range between 0 and 1. This makes fuzzy logic particularly useful in systems that involve human-like reasoning, where decisions are not strictly binary but involve some level of vagueness.

Key Components of a Fuzzy Logic System:

1. Fuzzification: Converts crisp input values (precise, like temperature = 70°F) into fuzzy sets using membership functions. For example, a temperature of 70°F might be "partially warm" and "partially hot" with certain degrees of membership.

2. Inference Engine: Applies a set of fuzzy rules (if-then conditions) to the fuzzy inputs. These rules are based on expert knowledge and help the system reason under uncertainty.

- Example rule: "If temperature is warm, then fan speed is medium."

3. Defuzzification: Converts the fuzzy output back into a crisp value to give a final, real-world answer or control action (e.g., fan speed = 60%).

Example: Consider an air conditioning system:

- Inputs: Temperature and humidity (both can be fuzzy).

- Outputs: Fan speed (also fuzzy).

- Fuzzy rules might say: "If the temperature is high and the humidity is low, then set fan speed to high."

Applications:

- Control systems: Air conditioning, washing machines, and automatic transmission in cars.

- Decision-making systems: Medical diagnosis, stock market prediction, etc.

Fuzzy logic is well-suited for systems where precise data is unavailable, and human-like reasoning is required to make decisions under uncertainty.

Code:

```
#Prac2b
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# Define fuzzy variables for traffic density, time of day, and green light duration
traffic_density = ctrl.Antecedent(np.arange(0, 101, 1), 'traffic_density')
time_of_day = ctrl.Antecedent(np.arange(0, 25, 1), 'time_of_day')
green_light_duration = ctrl.Consequent(np.arange(0, 61, 1), 'green_light_duration')

# Define membership functions for traffic density (low, medium, high)
traffic_density['low'] = fuzz.trimf(traffic_density.universe, [0, 0, 50])
traffic_density['medium'] = fuzz.trimf(traffic_density.universe, [30, 50, 70])
traffic_density['high'] = fuzz.trimf(traffic_density.universe, [50, 100, 100])

# Define membership functions for time of day (non-peak, peak)
time_of_day['non_peak'] = fuzz.trimf(time_of_day.universe, [0, 0, 12])
time_of_day['peak'] = fuzz.trimf(time_of_day.universe, [10, 24, 24])
```

```
# Define membership functions for green light duration (short, moderate, long)
green_light_duration['short'] = fuzz.trimf(green_light_duration.universe, [0, 0, 20])
green_light_duration['moderate'] = fuzz.trimf(green_light_duration.universe, [15, 30, 45])
green_light_duration['long'] = fuzz.trimf(green_light_duration.universe, [40, 60, 60])

# Visualize the membership functions
traffic_density.view()
time_of_day.view()
green_light_duration.view()

# Define the rules for the fuzzy system
rule1 = ctrl.Rule(traffic_density['low'] & time_of_day['non_peak'], green_light_duration['short'])
rule2 = ctrl.Rule(traffic_density['low'] & time_of_day['peak'], green_light_duration['moderate'])
rule3 = ctrl.Rule(traffic_density['medium'] & time_of_day['non_peak'],
green_light_duration['moderate'])
rule4 = ctrl.Rule(traffic_density['medium'] & time_of_day['peak'], green_light_duration['long'])
rule5 = ctrl.Rule(traffic_density['high'] & time_of_day['non_peak'], green_light_duration['long'])
rule6 = ctrl.Rule(traffic_density['high'] & time_of_day['peak'], green_light_duration['long'])

# Control system
green_light_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6])
green_light_sim = ctrl.ControlSystemSimulation(green_light_ctrl)

# Simulate the system for some input values (traffic density and time of day)
green_light_sim.input['traffic_density'] = 75  # High traffic
green_light_sim.input['time_of_day'] = 18      # Peak hours

# Compute the output based on the input values
```

green_light_sim.compute()

# Print and visualize the output

print(f"Recommended Green Light Duration: {green_light_sim.output['green_light_duration']} seconds")

green_light_duration.view(sim=green_light_sim)

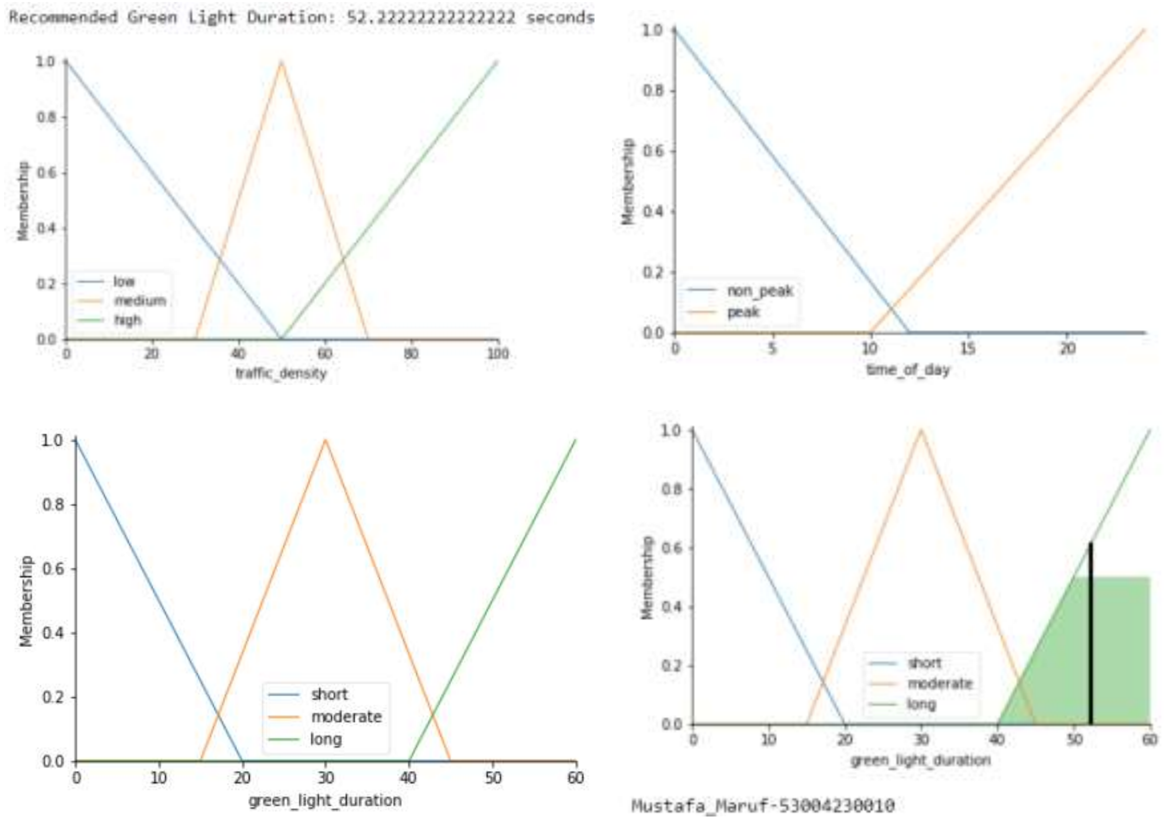
# Show the plots

plt.show()

# End note

print('Mustafa_Maruf-53004230010')

<u>Output:</u>



```
Recommended Green Light Duration: 52.22222222222222 seconds
```

```
Mustafa_Maruf-53004230010
```

Practical 2c

Aim:

Simulate artificial neural network model with both feedforward and backpropagation approach.

Theory:

An Artificial Neural Network (ANN) is a computational model inspired by the way biological neural networks (like the brain) process information. ANNs consist of layers of interconnected "neurons" (nodes), which process and transmit signals. They are used to solve complex problems in machine learning, such as classification, regression, and pattern recognition.

Components of the ANN:

1. Feedforward: Compute the output of the network based on input values and current weights.

2. Backpropagation: Adjust the weights based on the error between the predicted output and the true output (gradient descent).

Basic 3-layer neural network:

- Input layer
- Hidden layer
- Output layer

Explanation:

1. Activation Function:

•We use the Sigmoid activation function because it outputs values between 0 and 1, which is ideal for binary classification problems like XOR.

•Sigmoid derivative is used for backpropagation to compute the gradient during the weight update step.

2. Feedforward:

•In the feedforward step, we calculate the activations at the hidden and output layers. This is done by performing a dot product between the input values and the weights, adding a bias term, and applying the activation function.

3. Backpropagation:

•During backpropagation, we calculate the error at the output layer and propagate it backward through the network to update the weights using the gradient descent method.

$$\text{new weight} = \text{old weight} + \Delta\text{weight}$$

4. Training:

•The network is trained for a certain number of epochs. In each epoch, the feedforward and backpropagation processes are executed, and the loss is minimized by adjusting the weights.

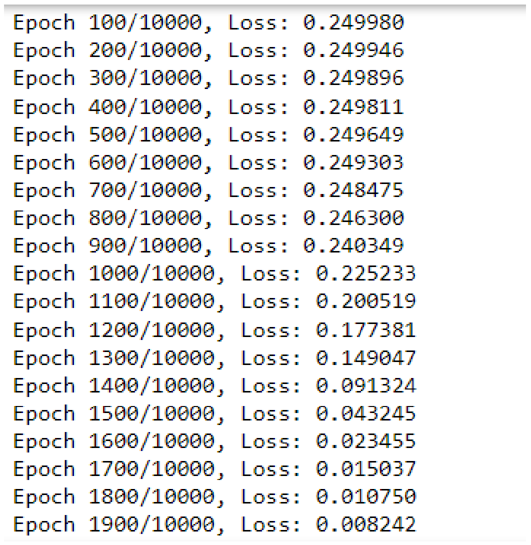
5. Example (XOR Problem):

•In the example, we're using the XOR problem, which is a classic problem to demonstrate the capabilities of a neural network. The XOR function is not linearly separable, meaning a simple linear classifier can't solve it, but a neural network can.

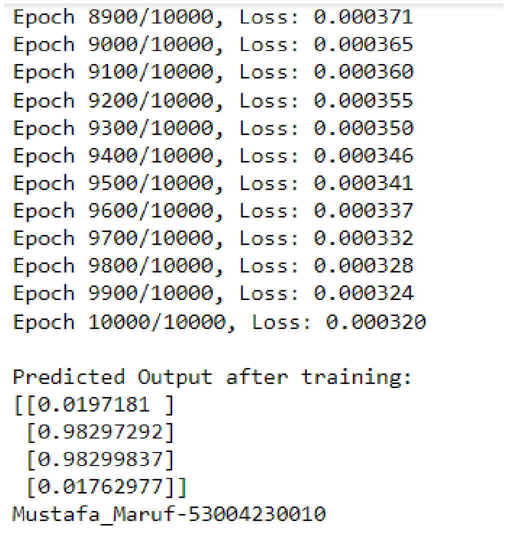
Code:

```
# Prac2c
import numpy as np


# Sigmoid Activation Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivative of the Sigmoid Function for backpropagation
def sigmoid_derivative(x):
    return x * (1 - x)


# ANN class to simulate feedforward and backpropagation
class ArtificialNeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.5):
        # Initialize weights randomly
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)

        # Initialize biases randomly
        self.bias_hidden = np.random.rand(1, hidden_size)
        self.bias_output = np.random.rand(1, output_size)

        # Set the learning rate
        self.learning_rate = learning_rate

    # Feedforward process
    def feedforward(self, X):
        # Hidden layer activation
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)

        # Output layer activation
        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output) +
self.bias_output
        self.output = sigmoid(self.output_input)

        return self.output

    # Backpropagation process
    def backpropagation(self, X, y):
        # Error at the output layer
        output_error = y - self.output
        output_delta = output_error * sigmoid_derivative(self.output)
        # Error at the hidden layer
        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error * sigmoid_derivative(self.hidden_output)

        # Update the weights and biases using the deltas
        self.weights_hidden_output += self.hidden_output.T.dot(output_delta) * self.learning_rate
        self.weights_input_hidden += X.T.dot(hidden_delta) * self.learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * self.learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * self.learning_rate

    # Train the neural network
    def train(self, X, y, epochs):
        for epoch in range(epochs):
            # Feedforward
            self.feedforward(X)
            # Backpropagation
            self.backpropagation(X, y)
            # Print loss every 100 epochs
            if (epoch + 1) % 100 == 0:
                loss = np.mean(np.square(y - self.output))
                print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss:.6f}')

# Example usage
if __name__ == "__main__":
    # Input dataset (XOR problem)
    X = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
    # Output dataset (XOR output)
    y = np.array([[0],
                  [1],
                  [1],
                  [0]])

    # Parameters
    input_size = X.shape[1]  # 2 features in input
    hidden_size = 2          # 2 neurons in hidden layer
    output_size = 1          # 1 output neuron (binary classification)

    # Create the neural network
    ann = ArtificialNeuralNetwork(input_size, hidden_size, output_size, learning_rate=0.5)

    # Train the neural network
    ann.train(X, y, epochs=10000)

    # Test the neural network
    output = ann.feedforward(X)
    print("\nPredicted Output after training:")
    print(output)

# Identifier
print('Mustafa_Maruf-53004230010')
```

Output:



```
Epoch 100/10000, Loss: 0.249980
Epoch 200/10000, Loss: 0.249946
Epoch 300/10000, Loss: 0.249896
Epoch 400/10000, Loss: 0.249811
Epoch 500/10000, Loss: 0.249649
Epoch 600/10000, Loss: 0.249303
Epoch 700/10000, Loss: 0.248475
Epoch 800/10000, Loss: 0.246300
Epoch 900/10000, Loss: 0.240349
Epoch 1000/10000, Loss: 0.225233
Epoch 1100/10000, Loss: 0.200519
Epoch 1200/10000, Loss: 0.177381
Epoch 1300/10000, Loss: 0.149047
Epoch 1400/10000, Loss: 0.091324
Epoch 1500/10000, Loss: 0.043245
Epoch 1600/10000, Loss: 0.023455
Epoch 1700/10000, Loss: 0.015037
Epoch 1800/10000, Loss: 0.010750
Epoch 1900/10000, Loss: 0.008242
```



```
Epoch 8900/10000, Loss: 0.000371
Epoch 9000/10000, Loss: 0.000365
Epoch 9100/10000, Loss: 0.000360
Epoch 9200/10000, Loss: 0.000355
Epoch 9300/10000, Loss: 0.000350
Epoch 9400/10000, Loss: 0.000346
Epoch 9500/10000, Loss: 0.000341
Epoch 9600/10000, Loss: 0.000337
Epoch 9700/10000, Loss: 0.000332
Epoch 9800/10000, Loss: 0.000328
Epoch 9900/10000, Loss: 0.000324
Epoch 10000/10000, Loss: 0.000320

Predicted Output after training:
[[0.0197181 ]
 [0.98297292]
 [0.98299837]
 [0.01762977]]
Mustafa_Maruf-53004230010
```

Practical 3a

Module 3:

Evolutionary Computation & Intelligent Agents

Aim:

Simulate genetic algorithm with suitable example using Python any other platform.

To create a Python program that functions as a basic math tutor, capable of explaining and performing basic arithmetic operations (addition, subtraction, multiplication, and division). The program should help users understand the operations and perform calculations based on user input.

Theory:

- Genetic Algorithms are search heuristics inspired by Charles Darwin's theory of natural evolution. They are used to find approximate solutions to optimization and search problems by mimicking the process of natural selection.

Key Concepts:

1. Population: A set of potential solutions to the problem.

2. Chromosomes: Everyone in the population is represented by a chromosome, which is a string of data (usually binary) encoding a potential solution.

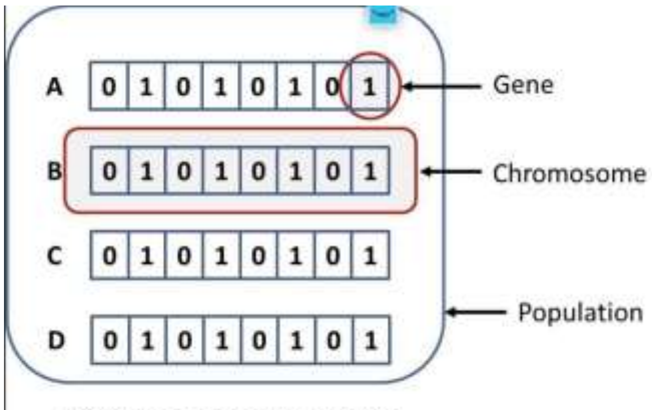
3. Genes: A chromosome is composed of genes, which represent specific features of the solution.

4. Fitness Function: A function that evaluates how close a given solution is to the optimal solution. The higher the fitness, the better the solution.

5. Selection: The process of choosing individuals from the current population based on their fitness to create the next generation.

6. Crossover (Recombination): Combining two parent chromosomes to produce offspring. It helps to explore new areas of the solution space.

7. Mutation: Randomly altering genes in a chromosome to maintain diversity in the population and prevent premature convergence to asuboptimal solution.



Gene, Chromosomes and Population

In a Feedforward Neural Network, information moves in one direction only: from the input layer through the hidden layers to the output layer. There are no loops or cycles in the network.

Step-by-Step Explanation:

1. Initialization: Start with a randomly generated population of chromosomes.

2. Evaluation: Calculate the fitness of each chromosome using the fitness function.

3. Selection: Select the fittest individuals to become parents for the next generation.

4. Crossover: Perform crossover between pairs of parents to produce offspring (new chromosomes).

5. Mutation: Apply mutation to some of the offspring to introduce new genetic material.

6. Replacement: Replace the old population with the new generation.

7. Repeat: Continue the process for several generations until a termination condition is met (e.g., a solution with sufficient fitness is found or a maximum number of generations is reached).

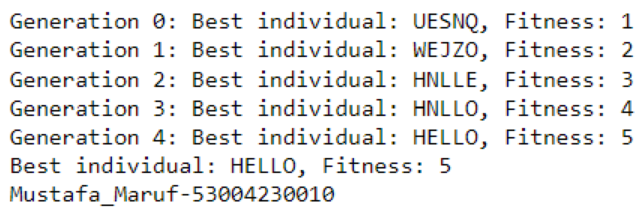
Code:

```
# Prac3a
import random
import string


# Genetic Algorithm parameters
```

```
target_string = "HELLO"

population_size = 50  # Increased population size

mutation_rate = 0.01

generations = 200  # Increased generations for more evolution


# Fitness function: number of characters matching the target

def fitness(individual):

    return sum(1 for a, b in zip(individual, target_string) if a == b)


# Create initial population (random strings)

def create_population(size):

    return [''.join(random.choices(string.ascii_uppercase, k=len(target_string))) for _ in
range(size)]

# Select parents (tournament selection)

def select_parents(population):

    tournament = random.sample(population, 5)  # Select 5 individuals instead of 3 for better
diversity

    return max(tournament, key=fitness)

# Crossover (single-point crossover)

def crossover(parent1, parent2):

    crossover_point = random.randint(1, len(parent1) - 1)

    return parent1[:crossover_point] + parent2[crossover_point:]

# Mutation (random character mutation)

def mutate(individual):

    individual = list(individual)

    for i in range(len(individual)):

        if random.random() < mutation_rate:

            individual[i] = random.choice(string.ascii_uppercase)

    return ''.join(individual)
```

```
# Main genetic algorithm loop
population = create_population(population_size)
for generation in range(generations):
    best_individual = max(population, key=fitness)
    print(f"Generation {generation}: Best individual: {best_individual}, Fitness: {fitness(best_individual)}")
    if fitness(best_individual) == len(target_string):  # Stop early if the optimal solution is found
        break
    # Create new generation
    new_population = []
    for _ in range(population_size):
        parent1 = select_parents(population)
        parent2 = select_parents(population)
        child = crossover(parent1, parent2)
        child = mutate(child)
        new_population.append(child)
    population = new_population
# Best individual in the final population
best_individual = max(population, key=fitness)
print(f"Best individual: {best_individual}, Fitness: {fitness(best_individual)}")
print('Mustafa_Maruf-53004230010')
```

<u>Output:</u>



```
Generation 0: Best individual: UESNQ, Fitness: 1
Generation 1: Best individual: WEJZO, Fitness: 2
Generation 2: Best individual: HNLLE, Fitness: 3
Generation 3: Best individual: HNLLO, Fitness: 4
Generation 4: Best individual: HELLO, Fitness: 5
Best individual: HELLO, Fitness: 5
Mustafa_Maruf-53004230010
```

Practical 3b

Aim:

Design intelligent agent using any AI algorithm. design expert tutoring system.

Genetic Algorithm to Solve a Simple String-Matching Problem. This example demonstrates a genetic algorithm that evolves a population of strings to match a target string.

Theory:

An intelligent agent is an autonomous entity that observes and acts upon an environment and directs its activity towards achieving goals. It can be a software program or a robotic system.

Characteristics of Intelligent Agents

- Autonomy: Operates without human intervention.
- Reactive: Responds to changes in the environment.
- Proactive: Takes initiative to achieve goals.
- Social: Communicates and cooperates with other agents or humans.

Example of an Intelligent Agent: Self-Driving Car

Explanation:

What Happened Here? Early Convergence: In this case, the algorithm reached the optimal solution "HELLO" in just 3 generations. The best individual in Generation 2 matches the target string exactly, achieving a fitness score of 5 (perfect match).

Parameters: The increased population size (50 instead of 10) and the ability to run for 200 generations gave the algorithm more opportunities to explore different combinations. Additionally, a larger tournament size for selecting parents (5 individuals) helps maintain better diversity, leading to quicker convergence.

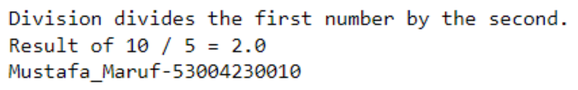
When you increase the population size and allow more generations, the genetic algorithm has more time and diversity to evolve toward the optimal solution. This demonstrates the importance of balancing the parameters of the genetic algorithm to improve its performance. The success of the algorithm depends on finding the right balance between exploration (mutation, diversity) and exploitation (selection of the best individuals.

Code:

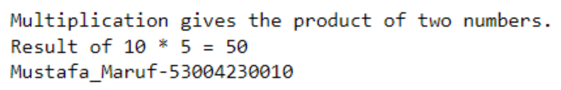
```
# Prac3b
class MathTutor:
    def __init__(self):
        self.operations = {
            '+': lambda a, b: a + b,
            '-': lambda a, b: a - b,
            '*': lambda a, b: a * b,
            '/': lambda a, b: a / b,
        }

    def explain_operation(self, operator):
        explanation = {
            '+': "Addition adds two numbers together.",
            '-': "Subtraction subtracts the second number from the first.",
            '*': "Multiplication gives the product of two numbers.",
            '/': "Division divides the first number by the second.",
        }
        return explanation.get(operator, "Invalid operation.")

    def perform_operation(self, operator, a, b):
        if operator in self.operations:
            return self.operations[operator](a, b)
        else:
            return None

if __name__ == "__main__":
    tutor = MathTutor()
```

```
# Example usage:
operator = '/'
a, b = 10, 5


print(tutor.explain_operation(operator))
result = tutor.perform_operation(operator, a, b)
print(f"Result of {a} {operator} {b} = {result}")


print('Mustafa_Maruf-53004230010')
```

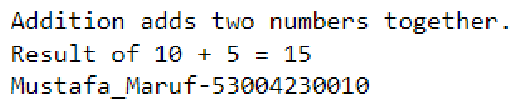
Output:



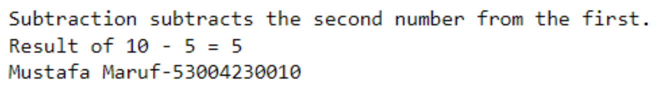
```
Division divides the first number by the second.
Result of 10 / 5 = 2.0
Mustafa_Maruf-53004230010
```



```
Multiplication gives the product of two numbers.
Result of 10 * 5 = 50
Mustafa_Maruf-53004230010
```



```
Addition adds two numbers together.
Result of 10 + 5 = 15
Mustafa_Maruf-53004230010
```



```
Subtraction subtracts the second number from the first.
Result of 10 - 5 = 5
Mustafa Maruf-53004230010
```

Practical 4a

Module 4:

Knowledge Representation and Language Decoding

Aim:

Design an application to simulate language parser.

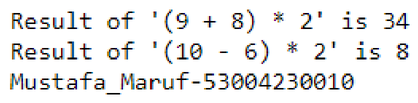
To design and implement a simple language parser that can evaluate basic arithmetic expressions involving addition, subtraction, multiplication, and division.

Code:

```
# Prac4a
class SimpleParser:
  def __init__(self, expr):
    self.tokens = expr.replace('(', ' ( ').replace(')', ' ) ').split()
    self.pos = 0

  def parse(self):
    return self.expr()

  def advance(self):
    self.pos += 1

  def current_token(self):
    return self.tokens[self.pos] if self.pos < len(self.tokens) else None

  def expr(self):
    result = self.term()
    while self.current_token() in ('+', '-'):
      if self.current_token() == '+':
            self.advance()
            result += self.term()
        elif self.current_token() == '-':
            self.advance()
            result -= self.term()
    return result

def term(self):
    result = self.factor()
    while self.current_token() in ('*', '/'):
        if self.current_token() == '*':
            self.advance()
            result *= self.factor()
        elif self.current_token() == '/':
            self.advance()
            result /= self.factor()
    return result

def factor(self):
    token = self.current_token()
    if token.isdigit():
        self.advance()
        return int(token)
    elif token == '(':
        self.advance()
        result = self.expr()
        self.advance()  # skip ')'
        return result
    raise ValueError("Invalid syntax")
```

```
if __name__ == "__main__":
    expr = "(9 + 8) * 2"
    expr1 = "(10 - 6) * 2"
    parser = SimpleParser(expr)
    result = parser.parse()
    print(f"Result of '{expr}' is {result}")

    parser = SimpleParser(expr1)
    result1 = parser.parse()
    print(f"Result of '{expr1}' is {result1}")

print('Mustafa_Maruf-53004230010')
```

<u>Output:</u>



```
Result of '(9 + 8) * 2' is 34
Result of '(10 - 6) * 2' is 8
Mustafa_Maruf-53004230010
```

Practical 4b

Aim:

Develop the semantic net using python.

To design and implement a semantic network in Python that models relationships between concepts in a domain.

Theory:

Semantic Network:

A semantic network is a data structure used to represent knowledge in the form of concepts (nodes) and their interrelationships (edges or links). It is a graphical model that depicts how different concepts in a particular domain are connected and how they relate to each other semantically.

Key Components:

1. Nodes (Concepts): These represent entities, objects, or concepts in the domain (e.g., "Dog", "Animal", "Barks").

2. Edges (Relationships): These represent the relationships or associations between the concepts (e.g., "is a", "has", "can do").

Example: Consider a semantic network for animals:

- Concepts (Nodes): "Dog", "Cat", "Animal", "Mammal".

- Relationships (Edges): "Dog is a Mammal", "Mammal is a Animal", "Dog has Fur", "Dog can Bark".

In this network:

- Dog is connected to Mammal by an "is a" relationship.

- Dog is connected to Bark by a "can" relationship.

Purpose: Semantic networks are used in AI and cognitive science to model knowledge in a structured and meaningful way. They help in understanding how concepts are interrelated and are commonly used in applications like natural language processing, expert systems, and knowledge representation.

Modeling Relationships:

Semantic networks model various types of relationships, such as:

- Hierarchical relationships: "is a" (e.g., Dog is a Mammal).

- Part-whole relationships: "has a" (e.g., Car has Wheels).

- Cause-effect relationships**: (e.g., Fire causes Smoke).

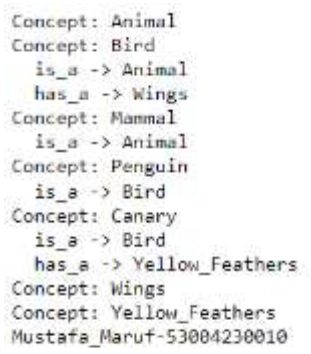
These networks are powerful tools for visualizing and reasoning about the structure of knowledge within a specific domain.

Code:

```
#Prac4b
class SemanticNetwork:
    def __init__(self):
        self.network = {}
    def add_concept(self, concept):
        if concept not in self.network:
            self.network[concept] = {'is_a': [], 'has_a': []}
    def add_relation(self, relation, concept1, concept2):
        self.add_concept(concept1)
        self.add_concept(concept2)
        self.network[concept1][relation].append(concept2)
    def get_relations(self, concept):
        return self.network.get(concept, {})
    def display_network(self):
        for concept, relations in self.network.items():
            print(f"Concept: {concept}")
            for relation, related_concepts in relations.items():
                for related_concept in related_concepts:
                    print(f"  {relation} -> {related_concept}")
if __name__ == "__main__":
    sn = SemanticNetwork()

    # Adding concepts and relations
```

```
   sn.add_concept("Animal")
   sn.add_concept("Bird")
   sn.add_concept("Mammal")
   sn.add_concept("Penguin")
   sn.add_concept("Canary")
   sn.add_relation("is_a", "Bird", "Animal")
   sn.add_relation("is_a", "Mammal", "Animal")
   sn.add_relation("is_a", "Penguin", "Bird")
   sn.add_relation("is_a", "Canary", "Bird")
   sn.add_relation("has_a", "Bird", "Wings")
   sn.add_relation("has_a", "Canary", "Yellow_Feathers")

   # Displaying the network
   sn.display_network()
print('Mustafa_Maruf-53004230010')
```

Output:



```
Concept: Animal
Concept: Bird
  is_a -> Animal
  has_a -> Wings
Concept: Mammal
  is_a -> Animal
Concept: Penguin
  is_a -> Bird
Concept: Canary
  is_a -> Bird
  has_a -> Yellow_Feathers
Concept: Wings
Concept: Yellow_Feathers
Mustafa_Maruf-53004230010
```