APPLIED ARTIFICIAL INTELLIGENCE PRACTICAL JOURNAL

Gayatri Kulkarni
MSCIT- III
53004230002

INDEX

Practical 1a

Module 1:-Expert System its Applications, Probability theory

Aim:- Design an Expert system using AIML.

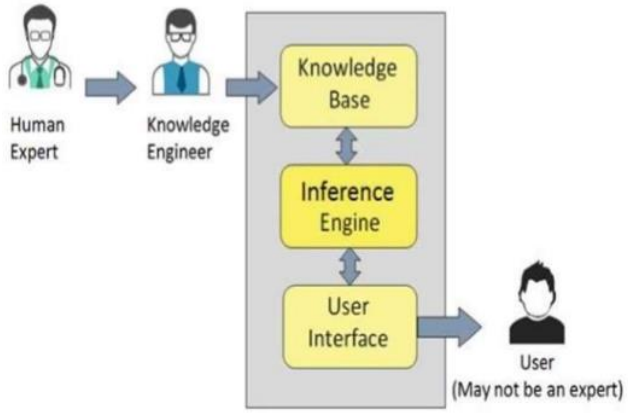
Theory:-

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

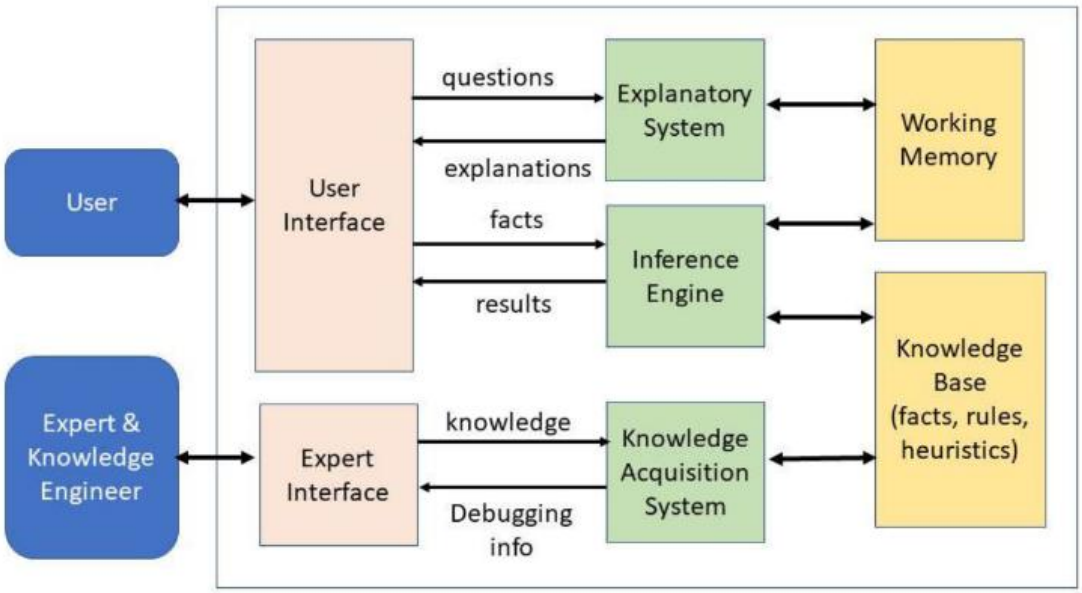
Concept: Artificial Intelligence Markup Language is referred to as AIML. It was the Alicebot that created AIML. It's a community for free software. Dr. Richard S. Wallace from 1995 to the present. An application called a chat box is made with AIML. In order to create an expert system using AIML, the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

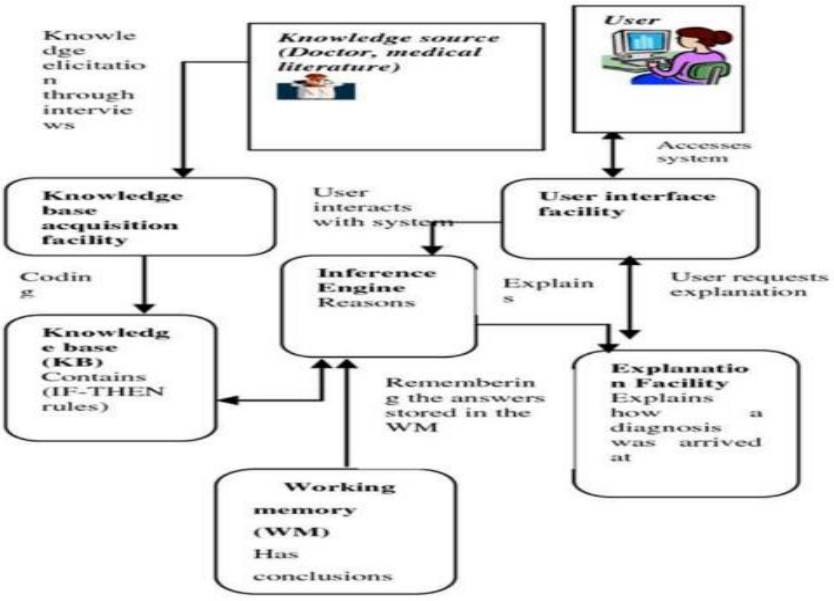
• In an expert system there are three main components: User Interface, Inference Engine and Knowledge Base

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.





Code:-

Flu.aiml(Text file)

```
<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

```
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('Gayatri Kulkarni-53004230002')
```

Output:-

```
Loading flu.aiml...done (0.00 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:   I have fever and cough
System: These symptoms could be associated with the flu. However, I recommend visiting a healthcare professional for an accurate diagnosis.
You:  How can I prevent flu
System: 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 pe
ople, and maintain a healthy lifestyle.
You:  bye
System: Goodbye! Stay healthy.
Gayatri Kulkarni-53004230002
```

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:-

Output:-

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:-

```
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 will be for Species (rows) and Island (columns), and we'll compute frequencies
pivot_table = pd.crosstab(df['species'], df['island'], normalize=True)

print("\nJoint Probability (Pivot Table):")
print(pivot_table)

# Example: Conditional Probability of Species given Island
# We can 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)

# To calculate Joint Probability, we already have it in the pivot table, normalized=True gives joint probabilities
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}")
print('Gayatri Kulkarni-53004230002')
```

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  year
0       3750.0    male  2007
1       3800.0  female  2007
2       3250.0  female  2007
3          NaN     NaN  2007
4       3450.0  female  2007

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
Gayatri Kulkarni-53004230002
```

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:-

```
%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).
%Gayatri kulkarni -53004230002
```

Output:-

rulebased.pl

File Edit Browse Compile Prolog Pce Help

rulebased.pl

```
%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).

%Gayatri kulkarni -53004230002
```

SWI-Prolog (AMD64, Multi-threaded, version 9.1.15)

File Edit Settings Run Debug Help

```
% c:/users/admin/documents/prolog/rulebased compiled 0.00 sec, -2 clauses
?- diagnose(Condition).
Condition = flu ;

?- diagnose(Condition).
Condition = flu
```

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:-

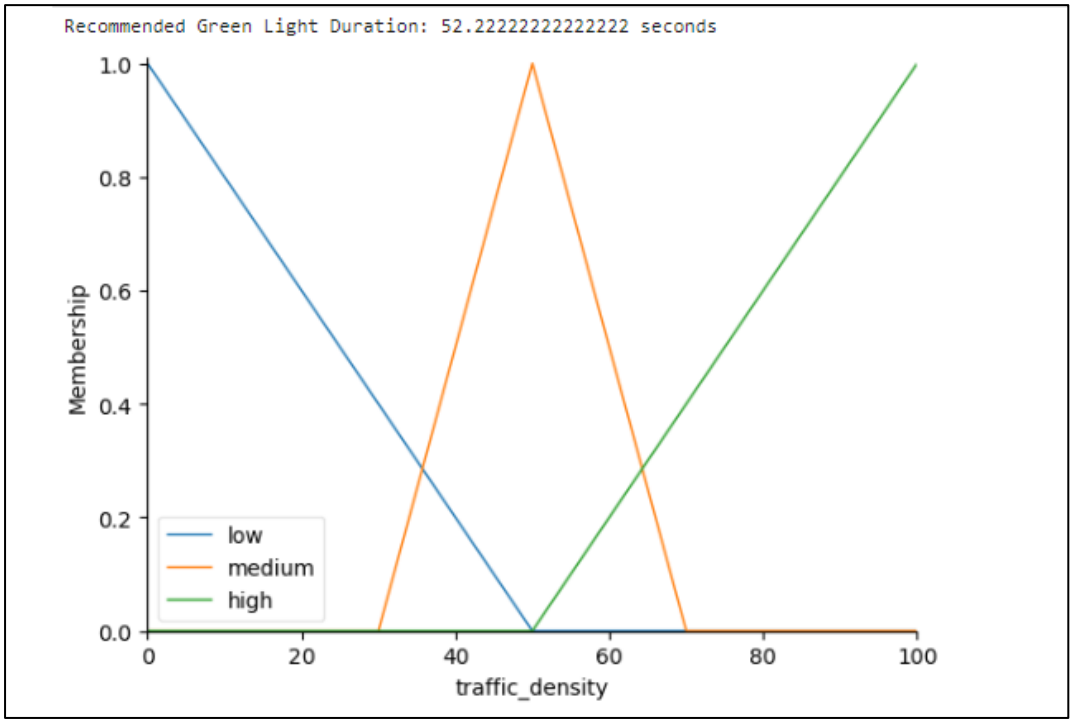
```
import numpy as np

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')
```

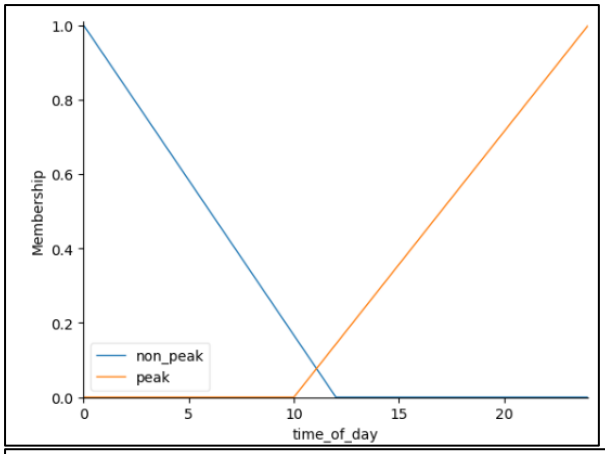
```
# Define membership functions for traffic density (low, medium, high)
traffic_density['low'] = fuzz.trimf(traffic_density.universe, [0, 0, 50])
traffic_density['medium'] = fuzz.trimf(traffic_density.universe, [30, 50, 70])
traffic_density['high'] = fuzz.trimf(traffic_density.universe, [50, 100, 100])

# 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()
```

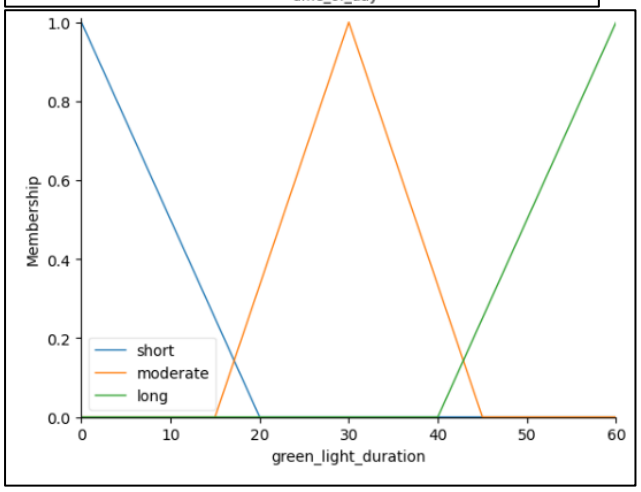
```
# Define the rules for the fuzzy system
rule1 = ctrl.Rule(traffic_density['low'] & time_of_day['non_peak'], green_light_duration['short'])
rule2 = ctrl.Rule(traffic_density['low'] & time_of_day['peak'], green_light_duration['moderate'])
rule3 = ctrl.Rule(traffic_density['medium'] & time_of_day['non_peak'],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

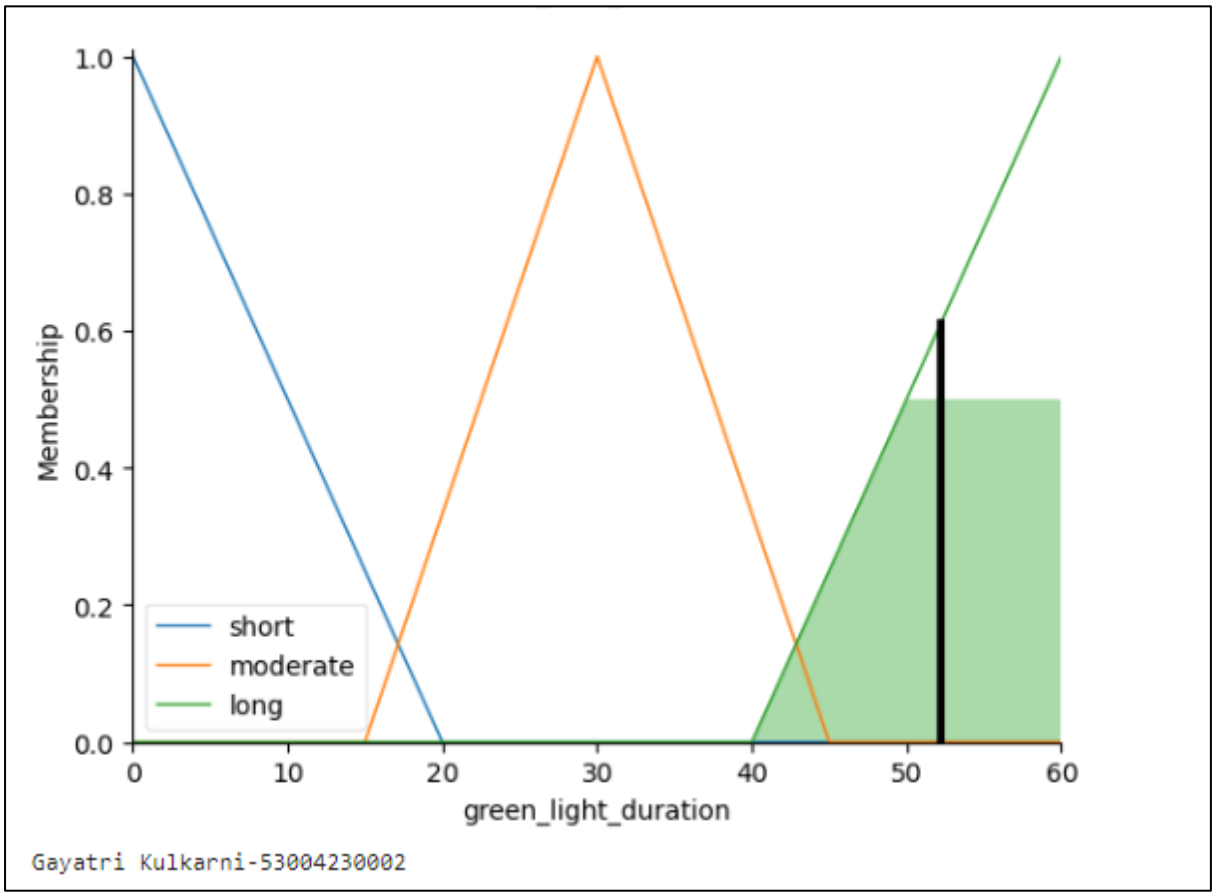
```
# Print and visualize the output
print(f"Recommended Green Light Duration:
{green_light_sim.output['green_light_duration']} seconds")
green_light_duration.view(sim=green_light_sim)


# Show the plots
plt.show()
print('Gayatri Kulkarni-53004230002')
```

Output:-




Membership
time_of_day
non_peak
peak


Membership
green_light_duration
short
moderate
long


Membership
green_light_duration
short
moderate
long
Gayatri Kulkarni-53004230002

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.

Key Components of an ANN:

1. Input Layer: Receives the input data.

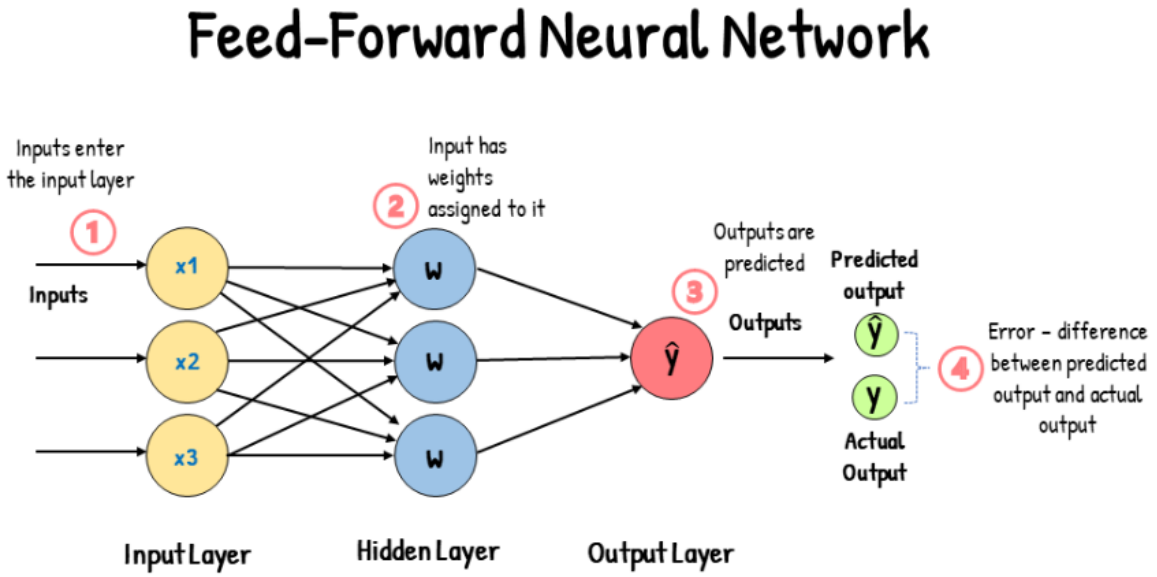
2. Hidden Layer(s): Intermediate layers that process the inputs through weighted connections and activation functions.

3. Output Layer: Produces the final result or prediction.

Feedforward Neural Network:

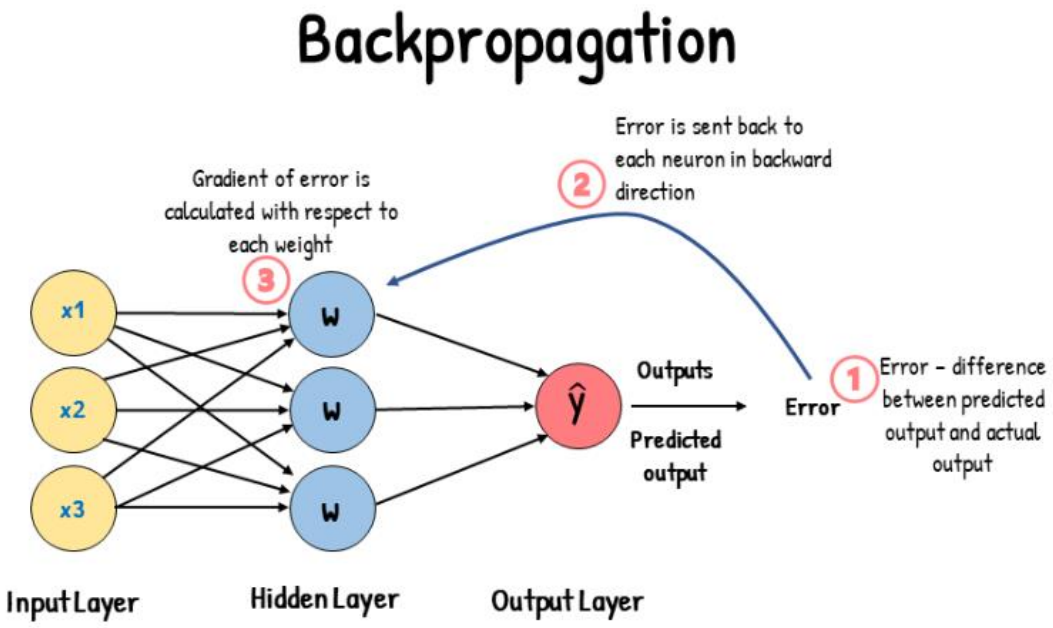
In a Feedforward Neural Network, information moves in one direction only: from the input layer through the hidden layers to the output layer. There are no loops or cycles in the network.

- How it works: Input data is passed through the network, with each neuron applying a weighted sum of its inputs and an activation function to produce an output. This continues layer by layer until the final output is obtained.



Example Flow:

- Input → Hidden Layer → Output (e.g., predicting house prices based on various features).



Backpropagation:

Backpropagation is the learning algorithm used to train an ANN. It adjusts the weights of the network to minimize the error between the predicted output and the actual target.

Steps in Backpropagation:

1. Forward Pass: Input data passes through the network (feedforward).

2. Error Calculation: The difference between the predicted output and the actual output (target) is computed using a loss function.

3. Backward Pass: The error is propagated backward through the network, and the weights are updated using gradient descent to minimize the loss function.

4. Repeat: This process is repeated iteratively across multiple training examples (epochs) until the network learns and the error is minimized.

- Feedforward: Data moves only forward through the network, layer by layer, to make predictions.

- Backpropagation: An error correction technique that adjusts weights in the network by moving backwards to minimize prediction errors, allowing the ANN to learn.

These mechanisms are key to how ANNs are trained and used in various AI applications, including image recognition, language processing, and more.

Code:-

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

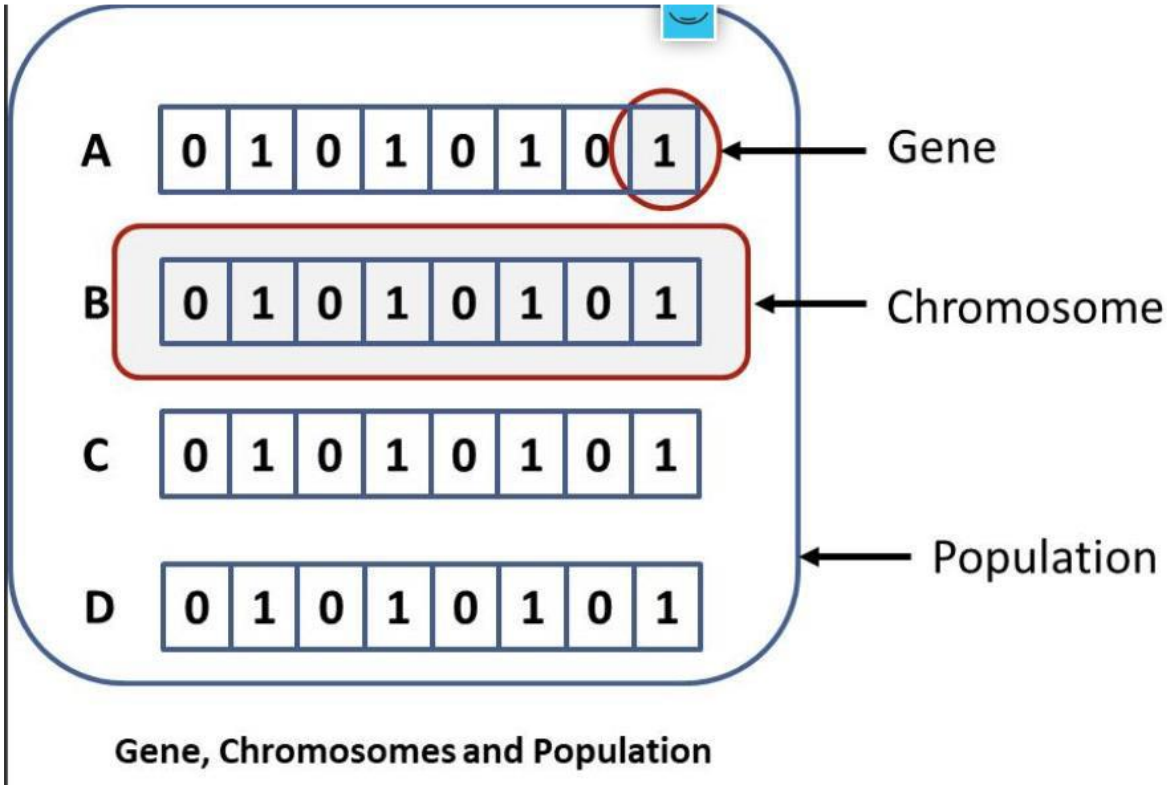
3. Genes: A chromosome is composed of genes, which represent specific features of the solution.

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 a suboptimal solution.



Gene, Chromosomes and Population

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:-

```
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('Gayatri Kulkarni-53004230002')
```

Output:-

```
Generation 0: Best individual: QKLIR, Fitness: 1
Generation 1: Best individual: HINLI, Fitness: 2
Generation 2: Best individual: HNLLC, Fitness: 3
Generation 3: Best individual: HNLLR, Fitness: 3
Generation 4: Best individual: HNLLC, Fitness: 3
Generation 5: Best individual: HELLC, Fitness: 4
Generation 6: Best individual: HELLR, Fitness: 4
Generation 7: Best individual: HELLC, Fitness: 4
Generation 8: Best individual: HELLC, Fitness: 4
Generation 9: Best individual: HELLC, Fitness: 4
Generation 10: Best individual: HELLC, Fitness: 4
Generation 11: Best individual: HELLC, Fitness: 4
Generation 12: Best individual: HELLC, Fitness: 4
Generation 13: Best individual: HELLC, Fitness: 4
Generation 14: Best individual: HELLC, Fitness: 4
Generation 15: Best individual: HELLR, Fitness: 4
Generation 16: Best individual: HELLC, Fitness: 4
Generation 17: Best individual: HELLC, Fitness: 4
Generation 18: Best individual: HELLC, Fitness: 4
Generation 19: Best individual: HELLC, Fitness: 4
Generation 20: Best individual: HELLR, Fitness: 4
Generation 21: Best individual: HELLC, Fitness: 4
Generation 22: Best individual: HELLR, Fitness: 4
Generation 23: Best individual: HELLR, Fitness: 4
Generation 24: Best individual: HELLR, Fitness: 4
Generation 25: Best individual: HELLC, Fitness: 4
Generation 26: Best individual: HELLR, Fitness: 4
Generation 27: Best individual: HELLI, Fitness: 4
Generation 28: Best individual: HELLI, Fitness: 4
Generation 29: Best individual: HELLC, Fitness: 4
Generation 30: Best individual: HELLC, Fitness: 4
Generation 31: Best individual: HELLC, Fitness: 4
Generation 32: Best individual: HELLR, Fitness: 4
Generation 33: Best individual: HELLC, Fitness: 4
Generation 34: Best individual: HELLC, Fitness: 4
Generation 35: Best individual: HELLU, Fitness: 4
Generation 36: Best individual: HELLR, Fitness: 4
Generation 37: Best individual: HELLR, Fitness: 4
Generation 38: Best individual: HELLU, Fitness: 4
Generation 39: Best individual: HELLC, Fitness: 4
Generation 40: Best individual: HELLC, Fitness: 4
Generation 41: Best individual: HELLR, Fitness: 4
Generation 42: Best individual: HELLU, Fitness: 4
Generation 43: Best individual: HELLI, Fitness: 4
Generation 44: Best individual: HELLI, Fitness: 4
Generation 45: Best individual: HELLU, Fitness: 4
Generation 46: Best individual: HELLP, Fitness: 4
Generation 47: Best individual: HELLP, Fitness: 4
Generation 48: Best individual: HELLC, Fitness: 4
Generation 49: Best individual: HELLC, Fitness: 4
Generation 50: Best individual: HELLP, Fitness: 4
Generation 50: Best individual: HELLP, Fitness: 4
Generation 51: Best individual: HELLP, Fitness: 4
Generation 52: Best individual: HELLP, Fitness: 4
Generation 53: Best individual: HELLY, Fitness: 4
Generation 54: Best individual: HELLY, Fitness: 4
Generation 55: Best individual: HELLP, Fitness: 4
Generation 56: Best individual: HELLU, Fitness: 4
Generation 57: Best individual: HELLU, Fitness: 4
Generation 58: Best individual: HELLP, Fitness: 4
Generation 59: Best individual: HELLP, Fitness: 4
Generation 60: Best individual: HELLY, Fitness: 4
Generation 61: Best individual: HELLY, Fitness: 4
Generation 62: Best individual: HELLY, Fitness: 4
Generation 63: Best individual: HELLP, Fitness: 4
Generation 64: Best individual: HELLY, Fitness: 4
Generation 65: Best individual: HELLP, Fitness: 4
Generation 66: Best individual: HELLY, Fitness: 4
Generation 67: Best individual: HELLP, Fitness: 4
Generation 68: Best individual: HELLB, Fitness: 4
Generation 69: Best individual: HELLP, Fitness: 4
Generation 70: Best individual: HELLP, Fitness: 4
Generation 71: Best individual: HELLP, Fitness: 4
Generation 72: Best individual: HELLY, Fitness: 4
Generation 73: Best individual: HELLP, Fitness: 4
Generation 74: Best individual: HELLH, Fitness: 4
Generation 75: Best individual: HELLN, Fitness: 4
Generation 76: Best individual: HELLP, Fitness: 4
Generation 77: Best individual: HELLP, Fitness: 4
Generation 78: Best individual: HELLP, Fitness: 4
Generation 79: Best individual: HELLP, Fitness: 4
Generation 80: Best individual: HELLP, Fitness: 4
Generation 81: Best individual: HELLP, Fitness: 4
Generation 82: Best individual: HELLP, Fitness: 4
Generation 83: Best individual: HELLP, Fitness: 4
Generation 84: Best individual: HELLP, Fitness: 4
Generation 85: Best individual: HELLP, Fitness: 4
Generation 86: Best individual: HELLP, Fitness: 4
Generation 87: Best individual: HELLP, Fitness: 4
Generation 88: Best individual: HELLY, Fitness: 4
Generation 89: Best individual: HELLV, Fitness: 4
Generation 90: Best individual: HELLY, Fitness: 4
Generation 91: Best individual: HELLP, Fitness: 4
Generation 92: Best individual: HELLP, Fitness: 4
Generation 93: Best individual: HELLP, Fitness: 4
Generation 94: Best individual: HELLP, Fitness: 4
Generation 95: Best individual: HELLP, Fitness: 4
Generation 96: Best individual: HELLY, Fitness: 4
Generation 97: Best individual: HELLY, Fitness: 4
Generation 98: Best individual: HELLY, Fitness: 4
Generation 99: Best individual: HELLO, Fitness: 5
Best individual: HELLO, Fitness: 5
Gayatri Kulkarni-53004230002
```

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:-

```
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('Gayatri Kulkarni-53004230002')
```

Output:-

```
Addition adds two numbers together.
Result of 10 + 5 = 15
Gayatri Kulkarni-53004230002
```

```
Subtraction subtracts the second number from the first.
Result of 10 - 5 = 5
Gayatri Kulkarni-53004230002
```

```
Multiplication gives the product of two numbers.
Result of 10 * 5 = 50
Gayatri Kulkarni-53004230002
```

```
Division divides the first number by the second.
Result of 10 / 5 = 2.0
Gayatri Kulkarni-53004230002
```

Practical 4a

Module:- 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.

Theory:-

Code:-

```
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 = "(3 + 5) * 2"
    parser = SimpleParser(expr)
    result = parser.parse()
    print(f"Result of '{expr}' is {result}")
print('Gayatri Kulkarni-53004230002')
```

Output:-

```
Result of '(3 + 5) * 2' is 16
Gayatri Kulkarni-53004230002

Result of '(8 - 9) * 2' is -2
Gayatri Kulkarni-53004230002
```

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:-

```
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('Gayatri Kulkarni-53004230002')
```

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
Gayatri Kulkarni-53004230002
```