Practical 1a

Aim:- Design an Expert system using AIML.

Code:-

Defining Flu.aiml

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

<category>

<pattern>WHAT ARE THE SYMPTOMS OF FLU </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>

Jupyter Code:-

import aiml

kernel = aiml.Kernel()

kernel.learn("flu.aiml")

print("Expert System for Identifying Flu Symptoms")

print("Type 'bye' to exit the conversation.")

while True:

user\_input = input("You: ")

if user\_input.lower() == "bye":

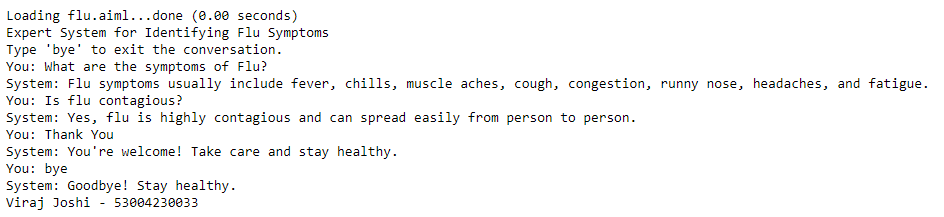
print("System: Goodbye! Stay healthy.")

break

response = kernel.respond(user\_input.upper())

print(f"System: {response}")

Output:-



Practical 1b

Code:-

import pandas as pd

def bayes\_theorem(prior\_A, likelihood\_B\_given\_A, marginal\_B):

"""

Calculate the posterior probability using Bayes' Theorem.

:param prior\_A: P(A) - Prior probability of A

:param likelihood\_B\_given\_A: P(B|A) - Likelihood of B given A

:param marginal\_B: P(B) - Marginal probability of B

:return: P(A|B) - Posterior probability of A given 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)

file\_path = 'iris.csv' # Path to the iris dataset file

class\_col = 'species' # The column representing the class (A)

class\_value = 'virginica' # 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()

print('Viraj Joshi - 53004230033')

Output:-



Practical 1c

Aim:-Implement Conditional Probability and joint probability using Python.

import pandas as pd

df = pd.read\_csv('penguins.csv')

print('Viraj Joshi - 53004230033')

print("Data Preview:")

print(df.head())

pivot\_table = pd.crosstab(df['species'], df['island'], normalize=True)

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

print(pivot\_table)

p\_adelie\_given\_dream = pivot\_table.loc['Adelie', 'Dream']

print()

print('The joint probability of an Adelie penguin being found on Dream Island.')

print(f"\nP(Adelie | Dream) = {p\_adelie\_given\_dream:.4f}")

conditional\_probability = pivot\_table.div(pivot\_table.sum(axis=0), axis=1)

print("\nConditional Probability of Species given Island:")

print(conditional\_probability)

normalized=True

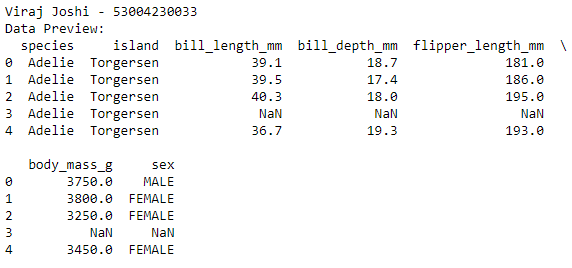
p\_adelie\_given\_dream = conditional\_probability.loc['Adelie', 'Dream']

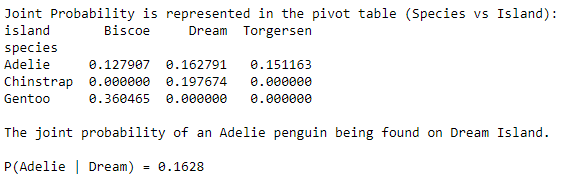
print()

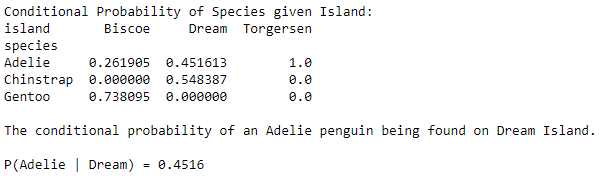
print('The conditional probability of an Adelie penguin being found on Dream Island.')

print(f"\nP(Adelie | Dream) = {p\_adelie\_given\_dream:.4f}")

Output:-







Practical 2a (Prolog Code)

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

%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),

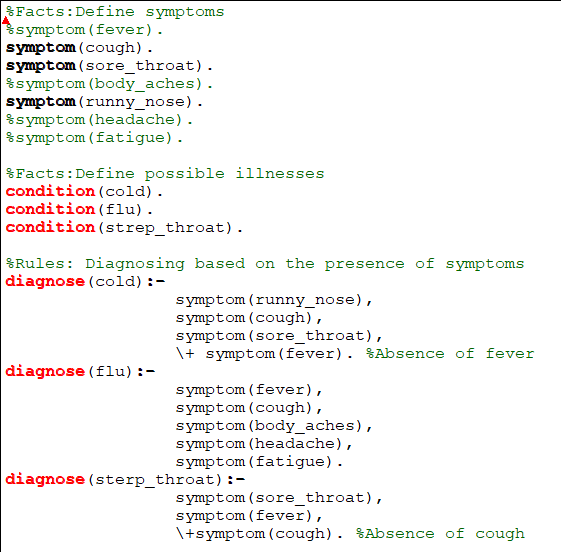
\+diagnose(strep\_throat).

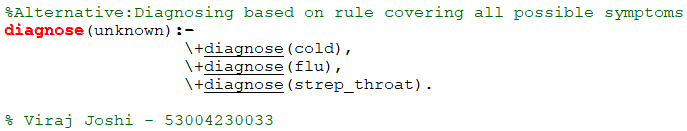
%You can ask Prolog:

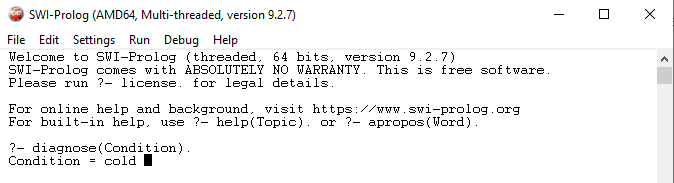
?-diagnose(Condition).

% Viraj Joshi – 53004230033

Output:-







Practical 2b

Aim:- Design a Fuzzy based application using Python.

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

import matplotlib.pyplot as plt

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')

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])

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])

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])

traffic\_density.view()

time\_of\_day.view()

green\_light\_duration.view()

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'])

green\_light\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6])

green\_light\_sim = ctrl.ControlSystemSimulation(green\_light\_ctrl)

green\_light\_sim.input['traffic\_density'] = 75 # High traffic

green\_light\_sim.input['time\_of\_day'] = 18 # Peak hours

green\_light\_sim.compute()

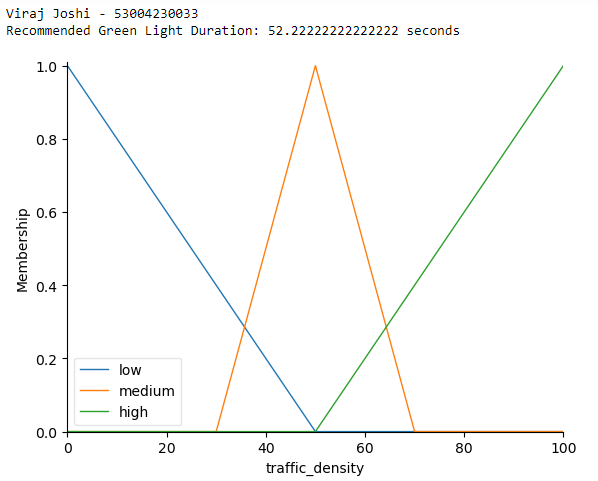
print('Viraj Joshi - 53004230033')

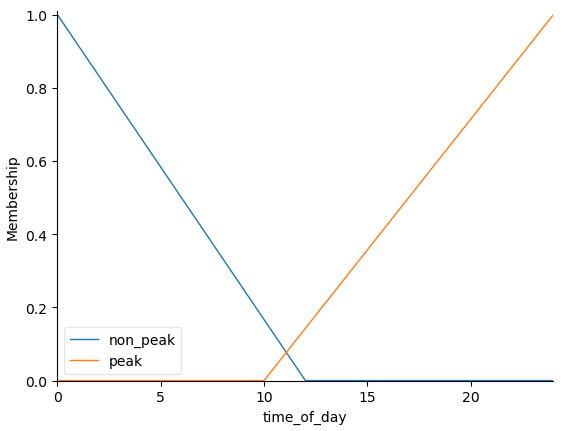
print(f"Recommended Green Light Duration: {green\_light\_sim.output['green\_light\_duration']} seconds")

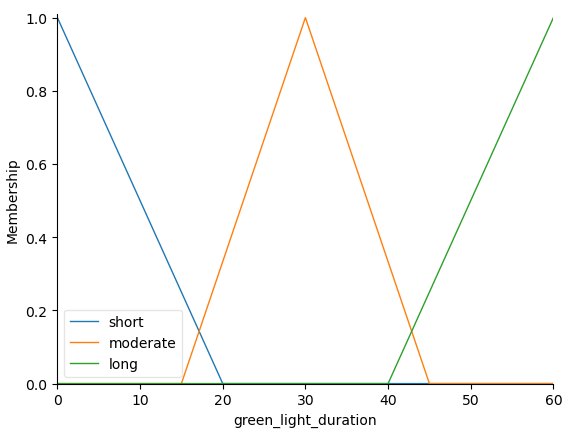
green\_light\_duration.view(sim=green\_light\_sim)

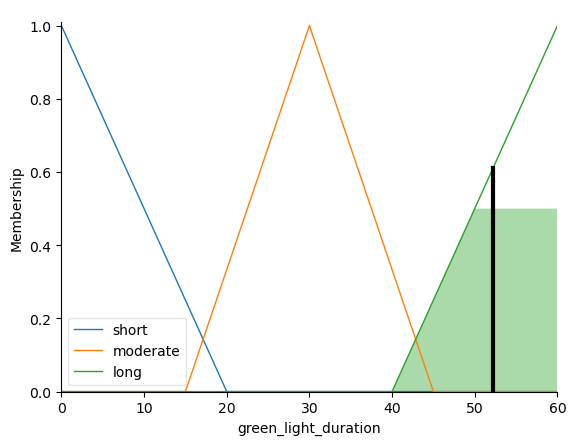
plt.show()

Output:-









Practical 2c

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

Code:-

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

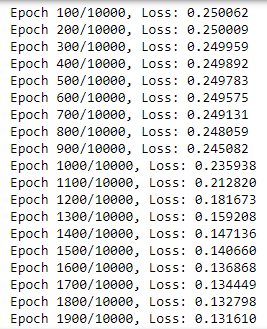
output = ann.feedforward(X)

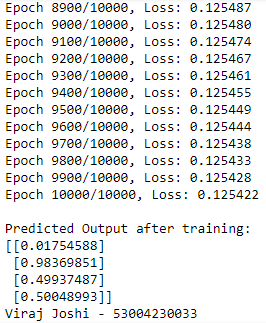
print("\nPredicted Output after training:")

print(output)

print('Viraj Joshi - 53004230033')

Output:-





Practical 3a

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

import random

import string

target\_string = "HELLO"

population\_size = 50

mutation\_rate = 0.01

generations = 200

def fitness(individual):

return sum(1 for a, b in zip(individual, target\_string) if a == b)

def create\_population(size):

return [''.join(random.choices(string.ascii\_uppercase, k=len(target\_string))) for \_ in range(size)]

def select\_parents(population):

tournament = random.sample(population, 5)

return max(tournament, key=fitness)

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1) - 1)

return parent1[:crossover\_point] + parent2[crossover\_point:]

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)

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):

break

new\_population = []

for \_ in range(population\_size):

parent1 = select\_parents(population)

parent2 = select\_parents(population)

child = crossover(parent1, parent2)

child = mutate(child)

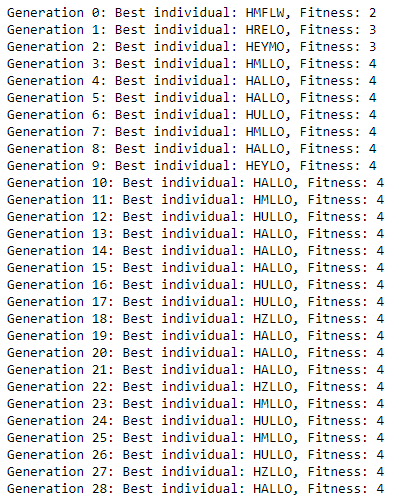
new\_population.append(child)

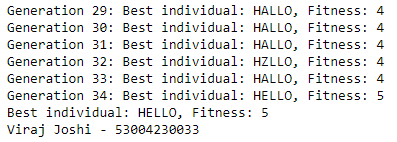
population = new\_population

best\_individual = max(population, key=fitness)

print(f"Best individual: {best\_individual}, Fitness: {fitness(best\_individual)}")

Output:-





Practical 3b

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

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 = 24, 8

print(tutor.explain\_operation(operator))

result = tutor.perform\_operation(operator, a, b)

print(f"Result of {a} {operator} {b} = {result}")

print('Viraj Joshi - 53004230033')

Output:-









Practical 4a

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 = "(10 + 5) \* 2"

parser = SimpleParser(expr)

result = parser.parse()

print(f"Result of '{expr}' is {result}")

print('Viraj Joshi - 53004230033')

Output:-





Practical 4b

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()

print('Viraj Joshi - 53004230033')

Output:-

