Anto Felix
192111089

1. **Write the python program to solve 8-Puzzle problem**

PROGRAM

```
def is_safe(board, row, col):    for i in range(col):        if
board[row][i] == 1:            return False    for i, j in
zip(range(row, -1, -1), range(col, -1, -1)):        if board[i][j]
== 1:            return False    for i, j in zip(range(row,
len(board), 1), range(col, -1, -1)):        if board[i][j] == 1:
return False    return True def solve_queens(board, col):
if col >= len(board):
      return True    for i in
range(len(board)):        if
is_safe(board, i, col):
         board[i][col] = 1           if
solve_queens(board, col + 1):
            return True
board[i][col] = 0    return
False def
print_solution(board):
for row in board:
      print(" ".join(["Q" if x else "." for x in row])) def
solve_8_queens():
  board = [[0] * 8 for _ in range(8)]
if solve_queens(board, 0):
print_solution(board)    else:
      print("No solution exists.") if
__name__ == "__main__":
  solve_8_queens()
```

OUTPUT:

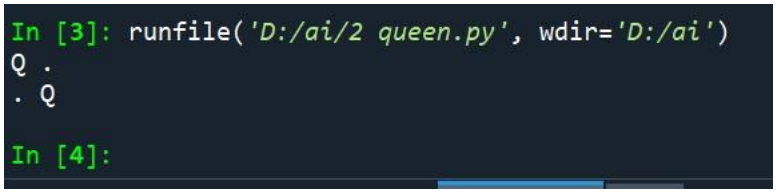
2.**Write the python program to solve 8-Queen problem**

PROGRAM: def is_safe(board, row, col): for i in range(row):
    if board[i][col] == 'Q' or board[i][row - i] == 'Q':
      return False
return True def

Anto Felix
192111089

```
solve_queens(n):
if n != 2:
      print("The 2x2 Queens Problem is not solvable.")
return    board = [['.' for _ in range(n)] for _ in
range(n)]    if solve(board, 0, n):       for row in
board:          print(' '.join(row))    else:
      print("No solution exists.")
def solve(board, row, n):    if
row == n:       return True    for
col in range(n):       if
is_safe(board, row, col):
board[row][col] = 'Q'          if
solve(board, row + 1, n):
            return True
board[row][col] = '.'
return False if __name__ ==
"__main__":
  solve_queens(2)
```

OUTPUT:



3.Write the python program for Water Jug Problem **PROGRAM:**

```
from collections import deque def
water_jug_problem(capacity_x, capacity_y, target):
visited = set()    initial_state = (0, 0)    queue =
deque([initial_state])    while queue:
      current_state = queue.popleft()
x, y = current_state       if x ==
target or y == target:          return
current_state       actions = [
(capacity_x, y),
          (x, capacity_y),
          (0, y),
          (x, 0),
          (min(x + y, capacity_x), max(0, x + y - capacity_x)),
          (max(0, x + y - capacity_y), min(x + y, capacity_y))
      ]
      for action in actions:
if action not in visited:
queue.append(action)
visited.add(action)
   return None   def print_solution(solution,
capacity_x, capacity_y):    if solution:
      x, y = solution
print("Solution:")       print(f"Jug X:
{x}/{capacity_x}")       print(f"Jug Y:
{y}/{capacity_y}")    else:
```

Anto Felix
192111089

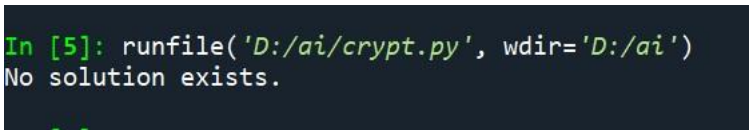
```
        print("No solution exists.")
  if __name__ == "__main__":
  capacity_x = 4      capacity_y =
  3      target = 2
     solution = water_jug_problem(capacity_x, capacity_y, target)
  print_solution(solution, capacity_x, capacity_y)
```

OUTPUT:

4.**Write the python program for Cript-Arithmetic problem** **PROGRAM:**

```
import itertools def is_valid_solution(words, mapping):     values = []     for word in words:
value = 0        for letter in word:            value = value * 10 + mapping[letter]
values.append(value)     return all(values[0] + values[1] == values[2] for values in
itertools.permutations(values)) def solve_cryptarithmetic(words):     unique_letters =
set("".join(words))     if len(unique_letters) > 10 or len(words[0]) < len(words[-1]):
return None     for perm in itertools.permutations("0123456789", len(unique_letters)):
      mapping = dict(zip(unique_letters, perm))
if is_valid_solution(words, mapping):
         return mapping
return None if __name__ ==
"__main__":
   word1 = "SEND"     word2 = "MORE"
result_word = "MONEY"     words =
[word1, word2, result_word]     solution =
solve_cryptarithmetic(words)     if
solution:
      print("Solution found:")        for word in words:
print("".join(str(solution[letter]) for letter in word))
else:
      print("No solution exists.")
```

OUTPUT:



5.**Write the python program for Missionaries Cannibal problem**

PROGRAM:

```
from collections import deque initial_state = (3, 3, 1)   goal_state = (0, 0, 0)
valid_actions = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)] def is_valid(state):     m, c, b = state
if m < 0 or c < 0 or m > 3 or c > 3 or (m != 0 and m < c) or (m != 3 and 3 - m < 3 - c):
      return False     return
True def
get_neighbors(state):
neighbors = []     for action
in valid_actions:        if
state[2] == 1:
         new_state = tuple(state[i] - action[i] if i < 2 else 1 - state[i] for i in range(3))
else:
```

Anto Felix
192111089

```
        new_state = tuple(state[i] + action[i] if i < 2 else 1 - state[i] for i in range(3))
if is_valid(new_state) and new_state != state:
neighbors.append(new_state)    return neighbors def
solve_missionaries_and_cannibals():
  visited = set()    queue = deque([(initial_state, [])])    while queue:       state, path =
queue.popleft()       if state == goal_state:          return path       for neighbor in
get_neighbors(state):          if neighbor not in visited:              visited.add(neighbor)
new_path = path + [state, neighbor]              queue.append((neighbor, new_path))    return None
def print_solution(solution):    if solution:       print("Solution:")       for state in solution:
print(f"Missionaries: {state[0]}, Cannibals: {state[1]}, Boat: {'Left' if state[2] == 1 else 'Right'}")
else:
        print("No solution exists.") if
__name__ == "__main__":
  solution = solve_missionaries_and_cannibals()
  print_solution(solution)
```

OUTPUT:

```
In [6]: runfile('D:/ai/missionary cannibals.py', wdir='D:/ai')
Solution:
Missionaries: 3, Cannibals: 3, Boat: Left
Missionaries: 3, Cannibals: 1, Boat: Right
Missionaries: 3, Cannibals: 1, Boat: Right
Missionaries: 3, Cannibals: 2, Boat: Left
Missionaries: 3, Cannibals: 2, Boat: Left
Missionaries: 3, Cannibals: 0, Boat: Right
Missionaries: 3, Cannibals: 0, Boat: Right
Missionaries: 3, Cannibals: 1, Boat: Left
Missionaries: 3, Cannibals: 1, Boat: Left
Missionaries: 1, Cannibals: 1, Boat: Right
Missionaries: 1, Cannibals: 1, Boat: Right
Missionaries: 2, Cannibals: 2, Boat: Left
Missionaries: 2, Cannibals: 2, Boat: Left
Missionaries: 0, Cannibals: 2, Boat: Right
Missionaries: 0, Cannibals: 2, Boat: Right
Missionaries: 0, Cannibals: 3, Boat: Left
Missionaries: 0, Cannibals: 3, Boat: Left
Missionaries: 0, Cannibals: 1, Boat: Right
Missionaries: 0, Cannibals: 1, Boat: Right
Missionaries: 1, Cannibals: 1, Boat: Left
Missionaries: 1, Cannibals: 1, Boat: Left
Missionaries: 0, Cannibals: 0, Boat: Right
```

6.**Write the python program for Vacuum Cleaner problem**

PROGRAM: import random

```
class VacuumCleaner:    def
__init__(self, position=0):
      self.position = position

  def move(self):
      self.position = random.choice([-1, 1])  # Move left or right

  def clean(self):
      print(f"Cleaning at position {self.position}")
```

Anto Felix
192111089

```
def simulate_vacuum_cleaner(steps):
    cleaner = VacuumCleaner()

    for step in range(steps):
        cleaner.move()
        cleaner.clean()

if __name__ == "__main__":
    simulation_steps = 10
simulate_vacuum_cleaner(simulation_steps)
```

OUTPUT:

```
In [2]: runfile('D:/ai/untitled2.py', wdir='D:/ai')
Cleaning at position -1
Cleaning at position 1
Cleaning at position -1
Cleaning at position 1
Cleaning at position -1
Cleaning at position 1
Cleaning at position -1
Cleaning at position 1
Cleaning at position 1
Cleaning at position -1
```

7.**Write the python program to implement BFS. PROGRAM:**

```
from collections import defaultdict, deque
class Graph:    def __init__(self):
self.graph = defaultdict(list)    def
add_edge(self, vertex, neighbor):
self.graph[vertex].append(neighbor)    def
bfs(self, start_vertex):
      visited = set()       queue = deque()
visited.add(start_vertex)
queue.append(start_vertex)       while queue:
current_vertex = queue.popleft()
print(current_vertex, end=" ")          for
neighbor in self.graph[current_vertex]:
if neighbor not in visited:
visited.add(neighbor)
queue.append(neighbor) if __name__ ==
"__main__":    g = Graph()
   g.add_edge(0, 1)
   g.add_edge(0, 2)
   g.add_edge(1, 2)
   g.add_edge(2, 0)
   g.add_edge(2, 3)
   g.add_edge(3, 3)    start_vertex = 2  #
Starting vertex for BFS
   print("Breadth-First Traversal (starting from vertex", start_vertex, "):")
   g.bfs(start_vertex)
```

OUTPUT:

Anto Felix
192111089

```
In [3]: runfile('D:/ai/bfs.py', wdir='D:/ai')
Breadth-First Traversal (starting from vertex 2 ):
2 0 3 1
```

8.**Write the python program to implement DFS. PROGRAM:**

```
from collections import defaultdict class
Graph:
   def __init__(self):
      self.graph = defaultdict(list)
   def add_edge(self, vertex, neighbor):
self.graph[vertex].append(neighbor)    def
dfs(self, start_vertex, visited):
visited.add(start_vertex)
print(start_vertex, end=" ")       for
neighbor in self.graph[start_vertex]:
if neighbor not in visited:
self.dfs(neighbor, visited) if __name__ ==
"__main__":    g = Graph()
   g.add_edge(0, 1)
   g.add_edge(0, 2)
   g.add_edge(1, 2)
   g.add_edge(2, 0)
   g.add_edge(2, 3)
   g.add_edge(3, 3)    start_vertex = 2  #
Starting vertex for DFS
   visited = set()
   print("Depth-First Traversal (starting from vertex", start_vertex, "):")
g.dfs(start_vertex, visited)
```

OUTPUT:

```
In [4]: runfile('D:/ai/dfs.py', wdir='D:/ai')
Depth-First Traversal (starting from vertex 2 ):
2 0 1 3
```

9.**Write the python to implement Travelling Salesman Problem**

PROGRAM:

```
import itertools def
calculate_total_distance(path, distances):
   total_distance = 0    for i in range(len(path) - 1):
total_distance += distances[path[i]][path[i+1]]
   total_distance += distances[path[-1]][path[0]]  # Return to the starting city
return total_distance def traveling_salesman_bruteforce(distances):
   num_cities = len(distances)    cities =
list(range(num_cities))    shortest_path = None
shortest_distance = float('inf')    for path in
itertools.permutations(cities):       distance =
calculate_total_distance(path, distances)       if distance
< shortest_distance:           shortest_path = path
shortest_distance = distance    return shortest_path,
shortest_distance if __name__ == "__main__":
   distances = [
[0, 29, 20, 21],
      [29, 0, 15, 22],
```

```
        [20, 15, 0, 24],
        [21, 22, 24, 0]
    ]
    shortest_path, shortest_distance = traveling_salesman_bruteforce(distances)
print("Shortest path:", shortest_path)     print("Shortest distance:",
shortest_distance)
```

OUTPUT:

```
In [5]: runfile('D:/ai/travelling sales man.py', wdir='D:/ai')
Shortest path: (0, 2, 1, 3)
Shortest distance: 78
```

10. **Write the python program to implement A* algorithm PROGRAM:**

```
import heapq class Node:     def
__init__(self, x, y, parent=None):
self.x = x        self.y = y        self.parent
= parent
        self.g = 0        self.h
= 0     def __lt__(self,
other):
        return (self.g + self.h) < (other.g + other.h) def
heuristic(node, goal):
    return abs(node.x - goal.x) + abs(node.y - goal.y) def astar(grid, start,
goal):     open_set = []     closed_set = set()     start_node = Node(start[0],
start[1])     goal_node = Node(goal[0], goal[1])
heapq.heappush(open_set, start_node)     while open_set:
current_node = heapq.heappop(open_set)        if current_node.x ==
goal_node.x and current_node.y == goal_node.y:            path = []
while current_node:                path.append((current_node.x,
current_node.y))               current_node = current_node.parent
          return path[::-1]
closed_set.add((current_node.x, current_node.y))
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
          new_x, new_y = current_node.x + dx, current_node.y + dy           if 0 <= new_x <
len(grid) and 0 <= new_y < len(grid[0]) and grid[new_x][new_y] != 1:               if (new_x,
new_y) not in closed_set:                  child_node = Node(new_x, new_y,
parent=current_node)
                child_node.g = current_node.g + 1
                child_node.h = heuristic(child_node, goal_node)
heapq.heappush(open_set, child_node)     return None if
__name__ == "__main__":
    # Example grid (0 represents empty, 1 represents an obstacle)
grid = [
        [0, 0, 0, 0, 0],

        [0, 1, 1, 0, 0],

        [0, 1, 0, 0, 0],

        [0, 1, 0, 1, 0],
```

```
        [0, 0, 0, 0, 0]
    ]
    start = (0, 0)     goal = (4, 4)
path = astar(grid, start, goal)
if path:
        print("Path found:", path)
else:
        print("No path found.")
```

OUTPUT:

11. **Write the python program for Map Coloring to implement CSP**

PROGRAM:

```
import numpy as np

import pandas as pd

from sklearn.metrics import confusion_matrix

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report def
create_artificial_data(size=100):
np.random.seed(42)
    features = np.random.rand(size, 4)

    labels = np.random.choice(['L', 'B', 'R'], size=size)

    artificial_data = pd.DataFrame(data=np.column_stack((labels, features)), columns=['Class', 'F1',
'F2', 'F3', 'F4'])

    return artificial_data def
splitdataset(balance_data):

X = balance_data.values[:, 1:5]

Y = balance_data.values[:, 0]



    X_train, X_test, y_train, y_test = train_test_split(

        X, Y, test_size=0.3, random_state=100)
```

Anto Felix
192111089

```
    return X, Y, X_train, X_test, y_train, y_test
  def train_using_gini(X_train, X_test, y_train):    clf_gini =
  DecisionTreeClassifier(criterion="gini", random_state=100, max_depth=3,
min_samples_leaf=5)
    clf_gini.fit(X_train, y_train)    return clf_gini
  def train_using_entropy(X_train, X_test, y_train):
  clf_entropy = DecisionTreeClassifier(
      criterion="entropy", random_state=100, max_depth=3, min_samples_leaf=5)
  clf_entropy.fit(X_train, y_train)    return clf_entropy def prediction(X_test,
  clf_object):    y_pred = clf_object.predict(X_test)    print("Predicted values:")
  print(y_pred)    return y_pred def cal_accuracy(y_test, y_pred):
  print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))    print("Accuracy :
  ", accuracy_score(y_test, y_pred) * 100)    print("Report : ",
  classification_report(y_test, y_pred)) def main():
    data = create_artificial_data(size=100)  # Use the artificial dataset
  X, Y, X_train, X_test, y_train, y_test = splitdataset(data)    clf_gini =
  train_using_gini(X_train, X_test, y_train)    clf_entropy =
  train_using_entropy(X_train, X_test, y_train)    print("Results Using
  Gini Index:")    y_pred_gini = prediction(X_test, clf_gini)
  cal_accuracy(y_test, y_pred_gini)    print("Results Using Entropy:")
    y_pred_entropy = prediction(X_test, clf_entropy)
  cal_accuracy(y_test, y_pred_entropy) if __name__
  == "__main__":
    main()
```

OUTPUT:

```
Results Using Gini Index:
Predicted values:
['B' 'B' 'B' 'L' 'R' 'B' 'R' 'B' 'L' 'R' 'B' 'B' 'L' 'R' 'L' 'R' 'L' 'B'
 'R' 'B' 'B' 'L' 'B' 'R' 'R' 'L' 'R' 'R' 'R' 'L']
Confusion Matrix:  [[2 2 4]
 [6 1 2]
 [3 5 5]]
Accuracy :  26.666666666666668
Report :               precision    recall  f1-score   support

           B       0.18      0.25      0.21         8
           L       0.12      0.11      0.12         9
           R       0.45      0.38      0.42        13

    accuracy                           0.27        30
   macro avg       0.25      0.25      0.25        30
weighted avg       0.28      0.27      0.27        30

Results Using Entropy:
Predicted values:
['B' 'B' 'R' 'R' 'R' 'R' 'R' 'B' 'L' 'R' 'B' 'B' 'L' 'R' 'B' 'R' 'L' 'B'
 'R' 'R' 'B' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'L' 'L']
Confusion Matrix:  [[1 3 4]
 [4 1 4]
 [3 3 7]]
Accuracy :  30.0
```

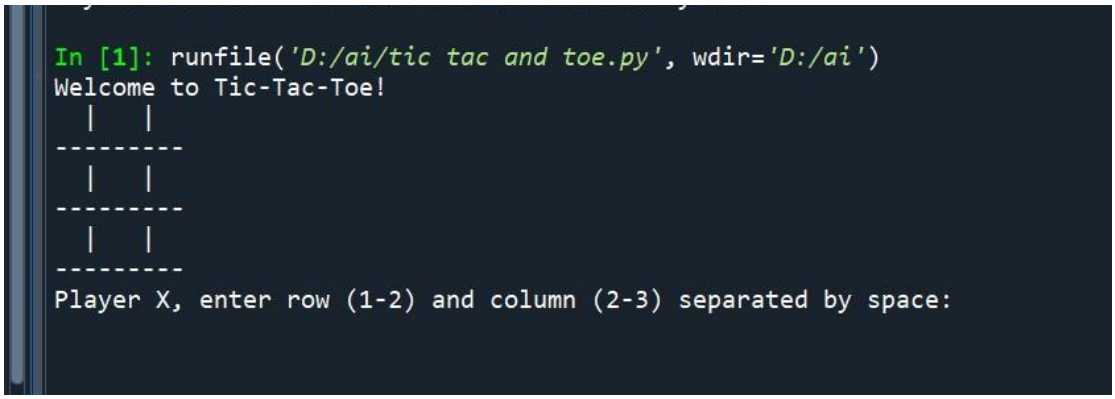
Anto Felix
192111089

12.**Write the python program for Tic Tac Toe game**

PROGRAM:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(row[col] == player for row in board):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
        return True
    return False
def is_board_full(board):
    return all(cell != " " for row in board for cell in row)
def play_tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"
    while True:
        print_board(board)
        try:
            while True:
                row, col = map(int, input(f"Player {current_player}, enter row (0-2) and column (0-2) separated by space: ").split())
                if row not in [0, 1, 2] or col not in [0, 1, 2]:
                    print("Invalid input. Please enter numbers between 0 and 2.")
                    continue
                if board[row][col] == " ":
                    break
                else:
                    print("That cell is already taken. Try again.")
        except ValueError:
            print("Invalid input. Please enter numbers.")
            continue
        except Exception as e:
            print(f"An error occurred: {str(e)}")
            continue
        board[row][col] = current_player
        if check_winner(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break
        if is_board_full(board):
            print_board(board)
            print("It's a tie!")
            break
        current_player = "O" if current_player == "X" else "X"
if __name__ == "__main__":
    print("Welcome to Tic-Tac-Toe!")
    play_tic_tac_toe()
```

OUTPUT:



13.**Write the python program to implement Minimax algorithm for gaming**

PROGRAM:

```
def is_valid_assignment(graph, assignment, node, color):
    for neighbor in graph[node]:
        if neighbor in assignment and
```

Anto Felix
192111089

```
assignment[neighbor] == color:            return False     return True def
backtracking(graph, colors, assignment, node):
   if node not in assignment:        for color in colors:            if
is_valid_assignment(graph, assignment, node, color):
assignment[node] = color                if len(assignment) ==
len(graph):
                return True
            next_node = get_unassigned_node(graph, assignment)
if backtracking(graph, colors, assignment, next_node):
return True                assignment.pop(node)     return False def
get_unassigned_node(graph, assignment):
   for node in graph:        if
node not in assignment:
         return node def
map_coloring(graph, colors):
   assignment = {}     start_node =
get_unassigned_node(graph, assignment)     if
backtracking(graph, colors, assignment, start_node):
return assignment     else:
      return None if
__name__ == "__main__":

   graph = {

      "WA": ["NT", "SA"],

      "NT": ["WA", "SA", "Q"],

      "SA": ["WA", "NT", "Q", "NSW", "V"],

      "Q": ["NT", "SA", "NSW"],

      "NSW": ["Q", "SA", "V"],

      "V": ["SA", "NSW"]

   }
   colors = ["Red", "Green", "Blue"]
solution = map_coloring(graph, colors)     if
solution:
      print("Map coloring solution:")
for node, color in solution.items():
print(f"{node}: {color}")     else:
      print("No solution exists.")
```

OUTPUT:

Anto Felix
192111089

```
In [1]: runfile('D:/ai/min max.py', wdir='D:/ai')
Map coloring solution:
WA: Red
NT: Green
SA: Blue
Q: Red
NSW: Green
V: Red
```

14.**Write the python program to implement Apha & Beta pruning algorithm for gaming PROGRAM:**

```
MAX, MIN = 1000, -1000 def minimax(depth,
nodeIndex, maximizingPlayer,          values,
alpha, beta):    if depth == 3:
    return values[nodeIndex]
if maximizingPlayer:
    best = MIN      for
i in range(0, 2):
      val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
best = max(best, val)
      if beta <= alpha:
        break
return best    else:
    best = MAX      for
i in range(0, 2):
      val = minimax(depth + 1, nodeIndex * 2 + i,True, values, alpha, beta)
best = min(best, val)          beta = min(beta, best)          if beta <= alpha:
        break      return
best  if __name__ ==
"__main__":
  values = [3, 5, 6, 9, 1, 2, 0, -1]
  print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

OUTPUT:

```
In [2]: runfile('D:/ai/alphabeta.py', wdir='D:/ai')
The optimal value is : 5

In [3]:
```

15 .Write the python program to implement Decision Tree PROGRAM:

```
import numpy as np import pandas as pd from
sklearn.metrics import confusion_matrix from
sklearn.model_selection import train_test_split from
sklearn.tree import DecisionTreeClassifier from
sklearn.metrics import accuracy_score from
sklearn.metrics import classification_report def
create_artificial_data(size=100):
np.random.seed(42)    features =
np.random.rand(size, 4)
  labels = np.random.choice(['L', 'B', 'R'], size=size)
  artificial_data = pd.DataFrame(data=np.column_stack((labels, features)), columns=['Class', 'F1', 'F2',
'F3', 'F4'])    return
artificial_data def
splitdataset(balance_data):

X = balance_data.values[:, 1:5]
Y = balance_data.values[:, 0]
  X_train, X_test, y_train, y_test = train_test_split(      X, Y, test_size=0.3,
random_state=100)    return X, Y, X_train, X_test, y_train, y_test def
```

Anto Felix
192111089

```
train_using_gini(X_train, X_test, y_train):     clf_gini =
DecisionTreeClassifier(criterion="gini", random_state=100, max_depth=3,
min_samples_leaf=5)
   clf_gini.fit(X_train, y_train)     return clf_gini
def train_using_entropy(X_train, X_test, y_train):
   clf_entropy = DecisionTreeClassifier(
      criterion="entropy", random_state=100, max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)     return clf_entropy def prediction(X_test,
clf_object):     y_pred = clf_object.predict(X_test)     print("Predicted values:")
print(y_pred)     return y_pred def cal_accuracy(y_test, y_pred):
   print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
   print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
print("Report : ", classification_report(y_test, y_pred)) def main():
data = create_artificial_data(size=100)  # Use the artificial dataset
   X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
clf_gini = train_using_gini(X_train, X_test, y_train)
clf_entropy = train_using_entropy(X_train, X_test, y_train)
print("Results Using Gini Index:")     y_pred_gini =
prediction(X_test, clf_gini)     cal_accuracy(y_test,
y_pred_gini)     print("Results Using Entropy:")
   y_pred_entropy = prediction(X_test, clf_entropy)
cal_accuracy(y_test, y_pred_entropy) if __name__
== "__main__":
   main()
```

OUTPUT:

Anto Felix
192111089

```
In [3]: runfile('D:/ai/decision tree.py', wdir='D:/ai')
Results Using Gini Index:
Predicted values:
['B' 'B' 'B' 'L' 'R' 'B' 'R' 'B' 'L' 'R' 'B' 'B' 'L' 'R' 'L' 'R' 'L' 'B'
 'R' 'B' 'B' 'L' 'B' 'R' 'R' 'L' 'R' 'R' 'R' 'L']
Confusion Matrix:  [[2 2 4]
 [6 1 2]
 [3 5 5]]
Accuracy :  26.666666666666668
Report :               precision    recall  f1-score   support

           B       0.18      0.25      0.21         8
           L       0.12      0.11      0.12         9
           R       0.45      0.38      0.42        13

    accuracy                           0.27        30
   macro avg       0.25      0.25      0.25        30
weighted avg       0.28      0.27      0.27        30

Results Using Entropy:
Predicted values:
['B' 'B' 'R' 'R' 'R' 'R' 'R' 'B' 'L' 'R' 'B' 'B' 'L' 'R' 'B' 'R' 'L' 'B'
 'R' 'R' 'B' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'L' 'L']
Confusion Matrix:  [[1 3 4]
 'R' 'R' 'B' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'L' 'L']
Confusion Matrix:  [[1 3 4]
 [4 1 4]
 [3 3 7]]
Accuracy :  30.0
Report :               precision    recall  f1-score   support

           B       0.12      0.12      0.12         8
           L       0.14      0.11      0.12         9
           R       0.47      0.54      0.50        13

    accuracy                           0.30        30
   macro avg       0.24      0.26      0.25        30
weighted avg       0.28      0.30      0.29        30
```

16 .**Write the python program to implement Feed forward neural Network**

PROGRAM:

```
import numpy as np def
sigmoid(x):
   return 1 / (1 + np.exp(-x)) def
sigmoid_derivative(x):    return
x * (1 - x) class NeuralNetwork:
def __init__(self, input_size,
hidden_size, output_size):
self.input_size = input_size
self.hidden_size = hidden_size
self.output_size = output_size
self.weights_input_hidden =
np.random.uniform(size=(input
_size, hidden_size))
self.weights_hidden_output =
np.random.uniform(size=(hidd
en_size, output_size))    def
feedforward(self, X):
self.hidden_layer_input =
```

Anto Felix
192111089

```
np.dot(X,
self.weights_input_hidden)
self.hidden_layer_output =
sigmoid(self.hidden_layer_inpu
t)
      self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output)
self.output_layer_output = sigmoid(self.output_layer_input)        return
self.output_layer_output     def train(self, X, y, learning_rate):
      output_error = y - self.feedforward(X)        output_delta = output_error *
sigmoid_derivative(self.output_layer_output)        hidden_layer_error =
output_delta.dot(self.weights_hidden_output.T)
      hidden_layer_delta = hidden_layer_error * sigmoid_derivative(self.hidden_layer_output)
self.weights_hidden_output += self.hidden_layer_output.T.dot(output_delta) * learning_rate
self.weights_input_hidden += X.T.dot(hidden_layer_delta) * learning_rate if __name__ ==
"__main__":     input_size = 2     hidden_size = 4     output_size = 1
   neural_network = NeuralNetwork(input_size, hidden_size, output_size)
   X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
   epochs = 10000     learning_rate = 0.1     for
epoch in range(epochs):
neural_network.train(X, y, learning_rate)
for i in range(len(X)):
      output = neural_network.feedforward(X[i])
print(f"Input: {X[i]} Output: {output}")
```

OUTPUT:

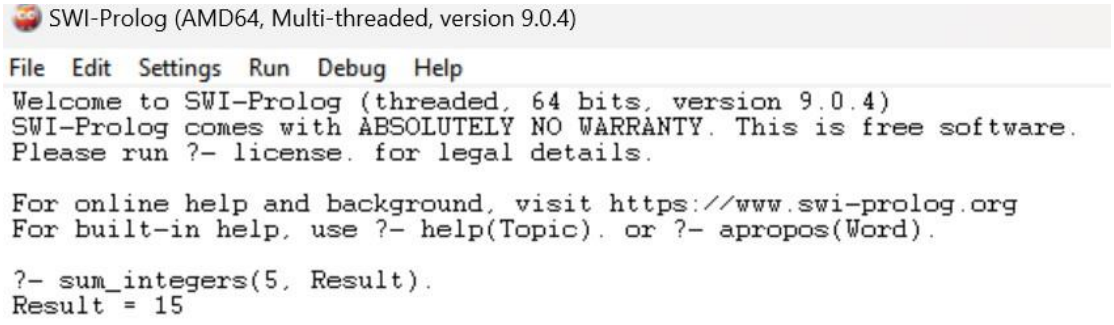
```
In [4]: runfile('D:/ai/neural.py', wdir='D:/ai')
Input: [0 0] Output: [0.09697139]
Input: [0 1] Output: [0.9173407]
Input: [1 0] Output: [0.91591576]
Input: [1 1] Output: [0.07566714]
```

17.**Write a Prolog Program to Sum the Integers from 1 to n. PROGRAM:**

```
% Base case: sum of integers from 1 to 0 is 0 sum_integers(0,
0).

% Recursive case: sum of integers from 1 to n is Sum = n + sum_integers(n-1)
sum_integers(N, Sum) :-
  N > 0,
  N1 is N - 1,
  sum_integers(N1, SubSum),
Sum is N + SubSum.
```

OUTPUT:

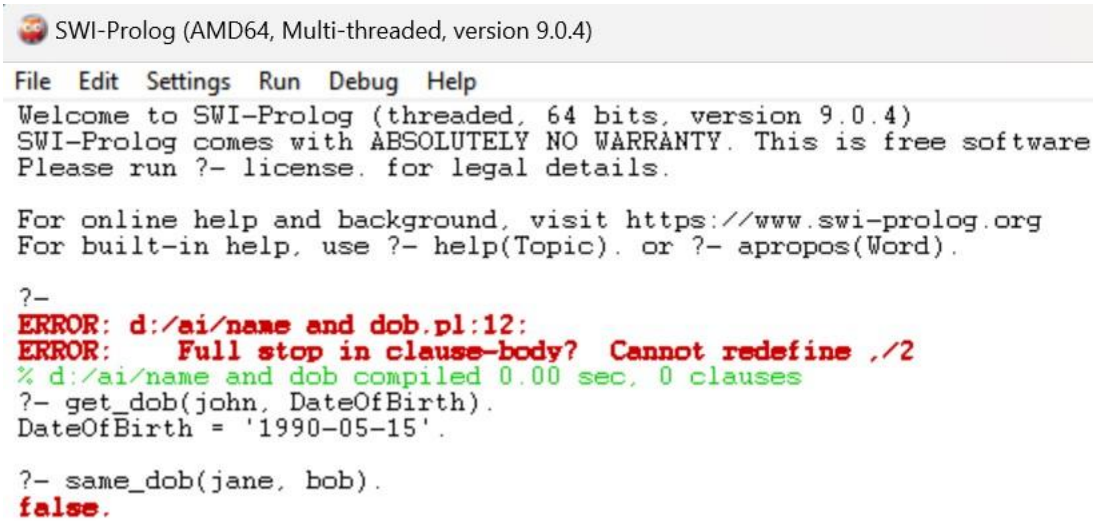
Anto Felix
192111089



18.**Write a Prolog Program for A DB WITH NAME, DOB.**
PROGRAM:

```
% Facts representing the database
dob(john, '1990-05-15').
dob(jane, '1985-12-20'). dob(bob,
'1995-08-10'). dob(alice, '1980-
03-25').

% Query to retrieve the date of birth for a given person
get_dob(Person, DateOfBirth) :-  dob(Person,
DateOfBirth).

% Query to check if two persons have the same date of birth
same_dob(Person1, Person2) :-  dob(Person1,
DateOfBirth),  dob(Person2, DateOfBirth),  Person1 \=
Person2.
```

OUTPUT:



19.**Write a Prolog Program for STUDENT-TEACHER-SUB-CODE. PROGRAM:**

```
% Facts representing relationships between students, teachers, and subjects
teaches(teacher_alice, math101). teaches(teacher_bob, physics201).
teaches(teacher_charlie, english301).
```

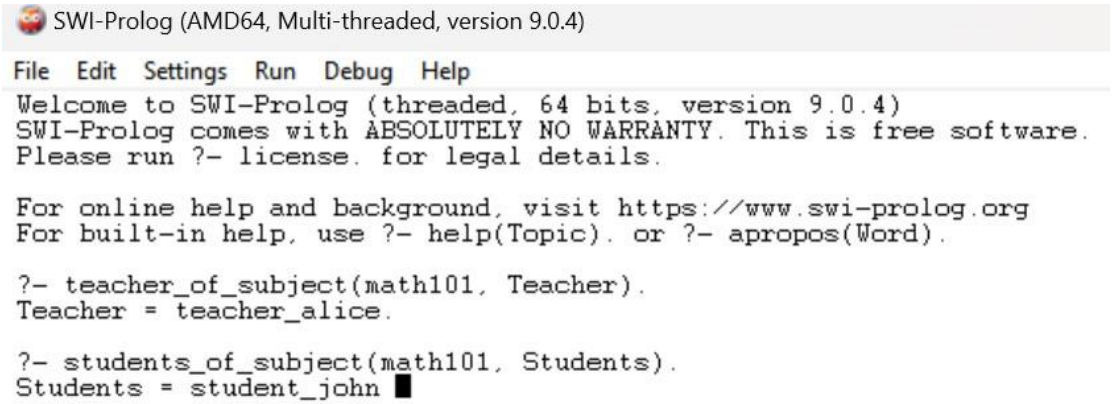
Anto Felix
192111089

enrolled(student_john, math101). enrolled(student_jane, physics201). enrolled(student_joe, english301). enrolled(student_jane, math101).

% Query to find the teacher of a subject
teacher_of_subject(Subject, Teacher) :-
teaches(Teacher, Subject).

% Query to find students enrolled in a subject
students_of_subject(Subject, Students) :-
enrolled(Students, Subject).

% Query to find subjects taught by a teacher
subjects_taught_by_teacher(Teacher, Subjects) :-
teaches(Teacher, Subjects). **OUTPUT:**

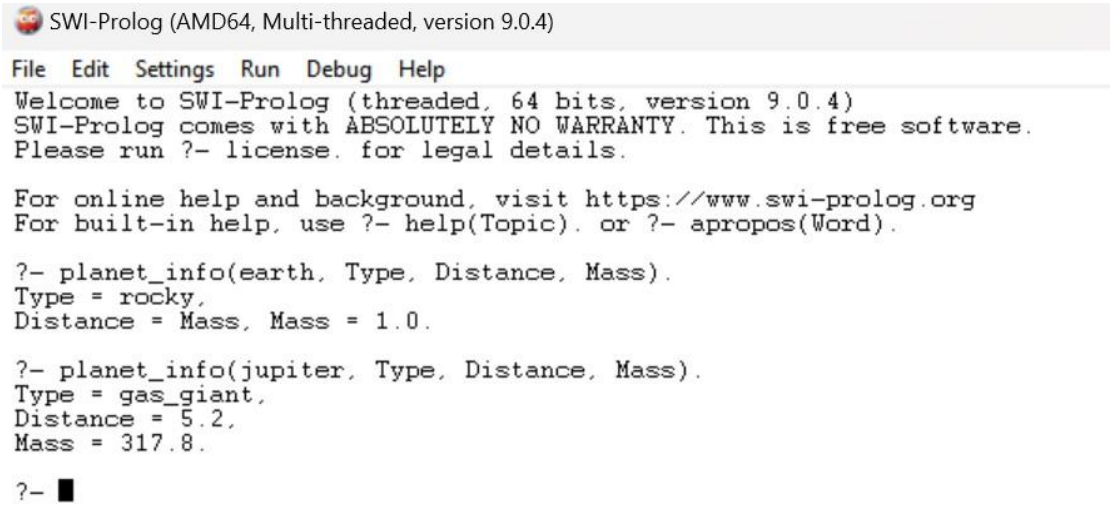


20.**Write a Prolog Program for PLANETS DB.**
PROGRAM:
% Facts representing information about planets
planet(mercury, rocky, 0.39, 0.24).
planet(venus, rocky, 0.72, 0.62). planet(earth, rocky, 1.0, 1.0). planet(mars, rocky, 1.52, 0.11).
planet(jupiter, gas_giant, 5.2, 317.8).
planet(saturn, gas_giant, 9.58, 95.2).
planet(uranus, ice_giant, 19.22, 14.5).
planet(neptune, ice_giant, 30.05, 17.2).

% Query to get information about a specific planet
planet_info(Name, Type, DistanceFromSun, Mass) :-
planet(Name, Type, DistanceFromSun, Mass).

OUTPUT:

Anto Felix
192111089

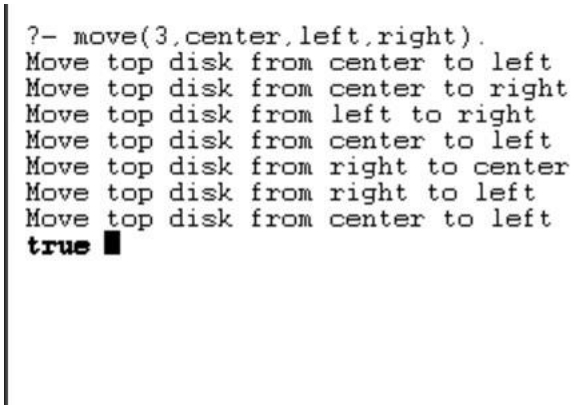


21.**Write a Prolog Program to implement Towers of Hanoi.**

PROGRAM:

```
move(1, X, Y, _) :-
write('Move top disk from '),
   write(X),
write(' to '),
write(Y),
nl.
move(N, X, Y, Z) :-
N > 1,     M is N-1,
move(M, X, Z, Y),
move(1, X, Y, _),
move(M, Z, Y, X).
```

OUTPUT:



22.**Write a Prolog Program to print particular bird can fly or not. Incorporate required queries.**

PROGRAM: can_fly(crow). can_fly(sparrow). can_fly(eagle).

```
cannot_fly(penguin).
cannot_fly(ostrich).
fly(X) :- can_fly(X). fly(X)
:- \+ cannot_fly(X).
```

Output:

Anto Felix
192111089

23.**Write the prolog program to implement family tree.**

Program:

```
female(ann).
female(pat).
male(tom).
male(bob).
male(jim).

% Define the parent relations
parent(pam, liz).
parent(pam, ann).
parent(pam, pat).
parent(tom, liz). parent(tom,
ann). parent(tom, pat).
parent(liz, bob). parent(liz,
jim).

% Define the mother relation mother(M,
C) :- female(M), parent(M, C).

% Define the father relation father(F,
C) :- male(F), parent(F, C).

% Define the grandfather relation grandfather(G, C) :-
male(G), parent(G, X), parent(X, C).

% Define the grandmother relation grandmother(G, C) :-
female(G), parent(G, X), parent(X, C).

% Define the sister relation sister(S, P) :-
female(S), parent(X, S), parent(X, P).

% Define the brother relation brother(B, P) :-
male(B), parent(X, B), parent(X, P).
```

Output:

```
?- sister(ann, pat).
true .

?- father(tom, pat).
true.
```

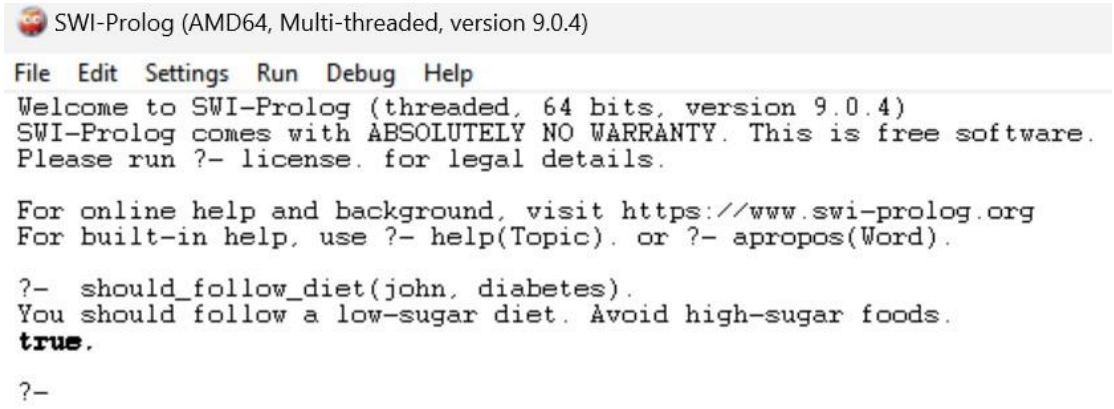
24.**Write a Prolog Program to suggest Dieting System based on Disease.**

Program:

Anto Felix
192111089

```
% Define facts about foods and their sugar content
sugar_content(apple, 10). sugar_content(banana,
15). sugar_content(carrot, 5).
sugar_content(chocolate, 30). sugar_content(cake,
40).

% Define a rule to check if a person should follow a low-sugar diet based on a disease
should_follow_diet(Person, Disease) :-    diabetic(Person),    % Rule: Person is
diabetic    disease_diet(Disease, Diet),  % Rule: Diet for the specific disease
write('You should follow a '), write(Diet), write(' diet. Avoid high-sugar foods.').

% Define specific diets for diseases disease_diet(diabetes,
'low-sugar').

% Define individuals with health conditions
diabetic(john).
```

OUTPUT:



25.**Write a Prolog program to implement Monkey Banana Problem**

PROGRAM:

```
% Initial state: monkey is on the left side without banana, boat is on the left
state(left, left, left).

% Final state: monkey, banana, and boat are all on the right side state(right,
right, right).

% Valid moves
move(state(left, left, left), grab, state(left, left, in_boat)). move(state(left,
left, in_boat), row, state(right, right, right)). move(state(right, right,
right), drop, state(right, right, in_boat)).
move(state(right, right, in_boat), row, state(left, left, left)).

% Helper predicate to check if a state is valid
valid_state(state(Monkey, Banana, Boat)) :-
member(Monkey, [left, right]),
member(Banana, [left, right]),
member(Boat, [left, right]).
```

Anto Felix
192111089

```
% Predicate to perform a sequence of moves
perform_moves([], State, State). perform_moves([Move |
Moves], CurrentState, FinalState) :-        move(CurrentState,
Move, NewState),
   valid_state(NewState), perform_moves(Moves,
    NewState, FinalState).
```

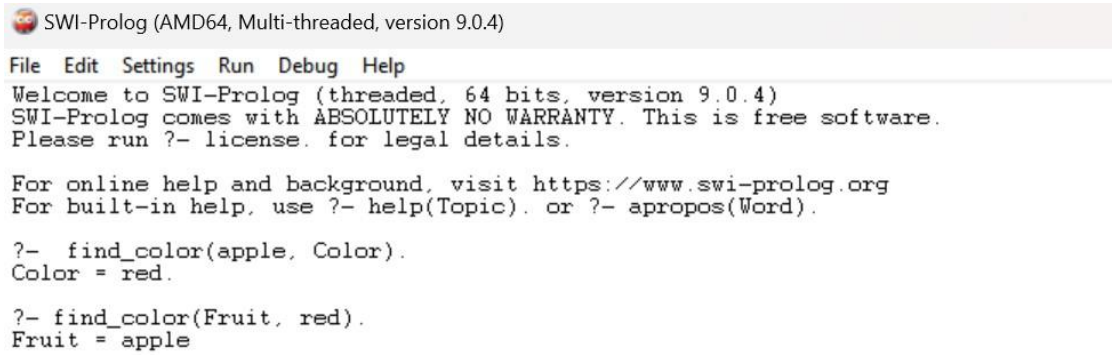
OUTPUT:

26.**Write a Prolog Program for fruit and its color using Back Tracking.**

PROGRAM:

```
% Facts about fruits and their colors fruit_color(apple,
red).
fruit_color(banana, yellow).
fruit_color(grape, purple). fruit_color(orange,
orange). fruit_color(strawberry, red).

% Backtracking rule to find the color of a fruit
find_color(Fruit, Color) :-     fruit_color(Fruit,
Color).
```

OUTPUT:



27.**Write a Prolog Program to implement Best First Search algorithm**

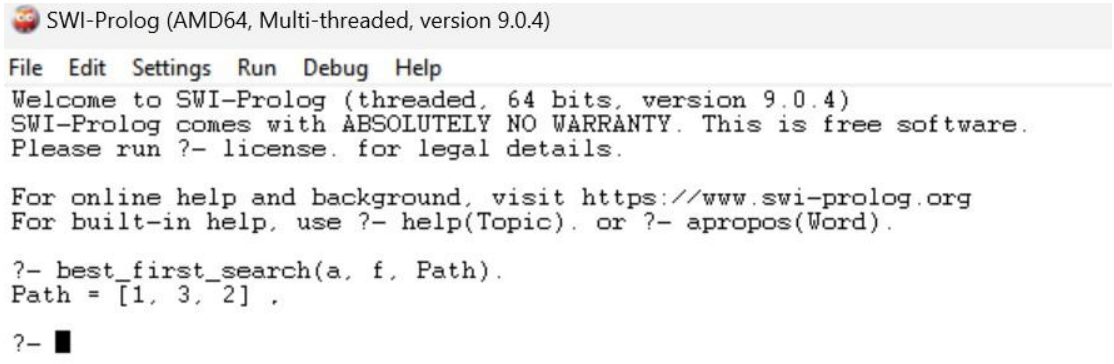
PROGRAM:

```
% best_first_search(Start, Goal, Path) best_first_search(Start,
Goal, Path) :-     best_first_search_helper([node(Start, [])],
Goal, Path).

% best_first_search_helper(Frontier, Goal, Path)
best_first_search_helper([node(Current, Path) | _], Goal, Path) :-
Current = Goal. best_first_search_helper(Frontier, Goal, Path) :-
choose_best(Frontier, Chosen),     expand(Chosen, Children),
append(Children, Frontier, NewFrontier),
best_first_search_helper(NewFrontier, Goal, Path).

% choose_best(Frontier, Best) choose_best([Node
| Rest], Best) :-     best_node(Rest, Node, Best).
```

Anto Felix
192111089

```
% best_node(Rest, CurrentBest, Best) best_node([],
Best, Best). best_node([Node | Rest], CurrentBest,
Best) :-    heuristic(Node, H1),
heuristic(CurrentBest, H2),    H1 < H2,
best_node(Rest, Node, Best). best_node([_ | Rest],
CurrentBest, Best) :-    best_node(Rest,
CurrentBest, Best).

% expand(Node, Children) expand(node(State,
Path), Children) :-    findall(node(Child, [Move
| Path]),
        (move(State, Child, Move), not(member(Child, Path))),
        Children).
```

OUTPUT:



28.**Write the prolog program for Medical Diagnosis**

PROGRAM:

```
% Facts about symptoms and possible diseases
symptom(john, fever). symptom(john, cough).
symptom(jane, headache). symptom(jane,
fever). symptom(bob, cough). symptom(bob,
fatigue).

disease(fever, flu). disease(cough,
cold).
disease(headache, stress). disease(fatigue,
anemia).

% Rules for diagnosis diagnose(Patient,
Disease) :-    symptom(Patient,
Symptom),    disease(Symptom,
Disease).
```

OUTPUT:

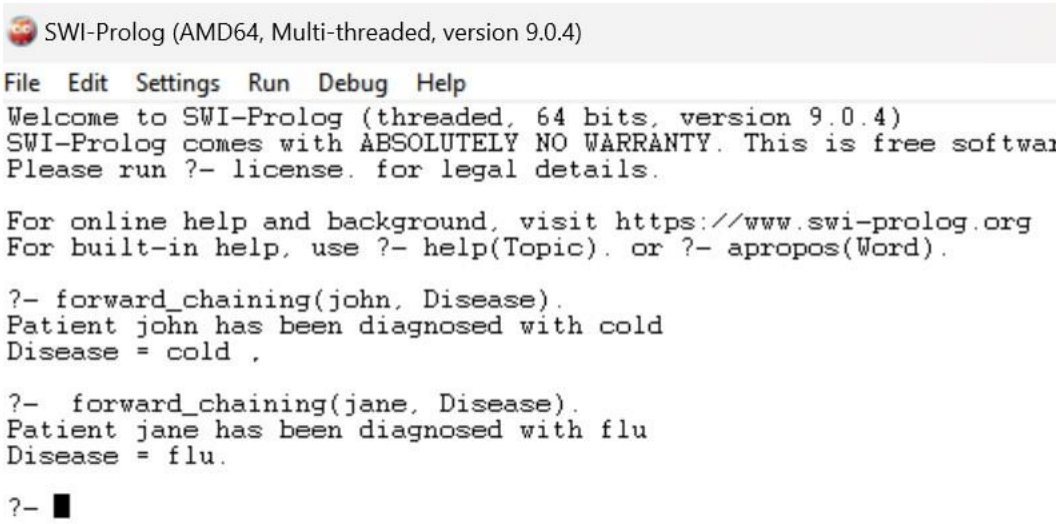
Anto Felix
192111089

29.**Write a Prolog Program for forward Chaining. Incorporate required queries.**

PROGRAM:

```
% Rules for logical inference
diagnose(Patient, cold) :-
symptom(Patient, cough),
symptom(Patient, sneezing).

diagnose(Patient, flu) :-
symptom(Patient, fever),
symptom(Patient, body_aches).

% Facts about symptoms
symptom(john, cough).
symptom(john, sneezing).
symptom(jane, fever).
symptom(jane, body_aches).

% Forward Chaining Inference Rule forward_chaining(Patient, Disease) :-
diagnose(Patient, Disease),     write('Patient '), write(Patient), write(' has been
diagnosed with '), write(Disease), nl.
```

OUTPUT:



30.**Write a Prolog Program for backward Chaining. Incorporate required queries.**

Anto Felix
192111089

PROGRAM:

```
% Rules for logical inference
diagnose(Patient, cold) :-
symptom(Patient, cough),
symptom(Patient, sneezing).

diagnose(Patient, flu) :-
symptom(Patient, fever),
symptom(Patient, body_aches).

% Facts about symptoms
symptom(john, cough).
symptom(john, sneezing).
symptom(jane, fever). symptom(jane,
body_aches).

% Backward Chaining Inference Rule backward_chaining(Patient, Disease) :-
diagnose(Patient, Disease),     write('Patient '), write(Patient), write(' has been
diagnosed with '), write(Disease), nl.
```

OUTPUT: