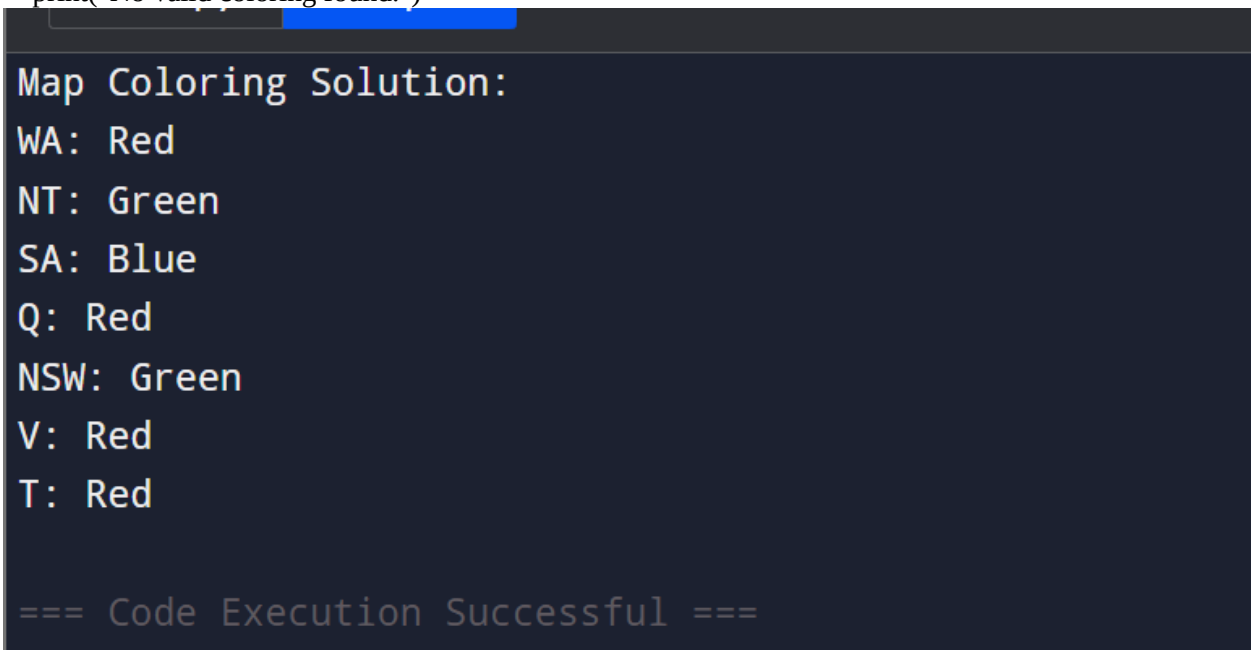
11.MAP COLOURING TO IMPLEMENT CSP

```
# Map Coloring using Constraint Satisfaction Problem (CSP) and Backtracking

# Regions (Variables)
regions = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T']

# Domains (Available Colors)
colors = ['Red', 'Green', 'Blue']

# Constraints (Adjacency map)
adjacency = {
    'WA': ['NT', 'SA'],
    'NT': ['WA', 'SA', 'Q'],
    'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],
    'Q': ['NT', 'SA', 'NSW'],
    'NSW': ['SA', 'Q', 'V'],
    'V': ['SA', 'NSW'],
    'T': []  # Tasmania has no neighbors
}

# Function to check if the current color assignment is valid
def is_valid(region, color, assignment):
    for neighbor in adjacency[region]:
        if neighbor in assignment and assignment[neighbor] == color:
            return False
    return True

# Backtracking CSP Solver
def backtrack(assignment):
    # If all regions are assigned, return the assignment
    if len(assignment) == len(regions):
        return assignment

    # Select an unassigned region
    unassigned = [r for r in regions if r not in assignment]
    current = unassigned[0]

    for color in colors:
        if is_valid(current, color, assignment):
            assignment[current] = color
            result = backtrack(assignment)
            if result:
                return result
            # Backtrack
            del assignment[current]

    return None

# Solve and print the solution
```

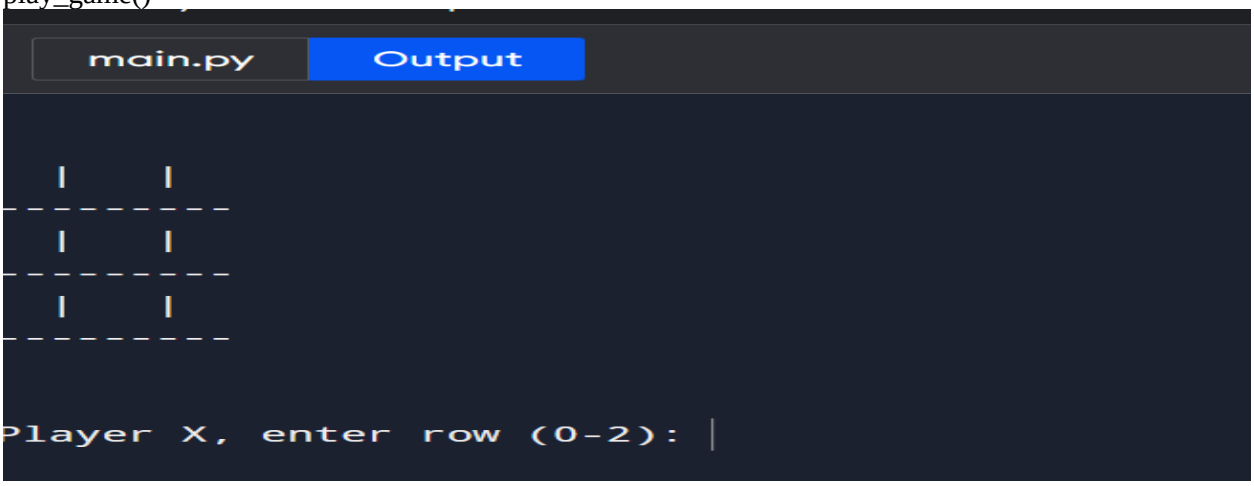
```
solution = backtrack({})
if solution:
    print("Map Coloring Solution:")
    for region in regions:
        print(f"{region}: {solution[region]}")
else:
    print("No valid coloring found.")
```



12.TIC TAC TOE PROGRAM

```
# Tic Tac Toe Game - Two Players (Terminal-based)

def print_board(board):
    print("\n")
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
    print("\n")
def check_winner(board, player):
    # Check rows, columns and diagonals
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or \
       all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_draw(board):
```

```
    return all(cell in ['X', 'O'] for row in board for cell in row)

def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "X"

    while True:
        print_board(board)
        try:
            row = int(input(f"Player {current_player}, enter row (0-2): "))
            col = int(input(f"Player {current_player}, enter column (0-2): "))
        except ValueError:
            print("Please enter valid numbers.")
            continue

        if row not in range(3) or col not in range(3):
            print("Invalid position! Try again.")
            continue

        if board[row][col] != " ":
            print("Cell already taken! Try again.")
            continue
        board[row][col] = current_player
        if check_winner(board, current_player):
            print_board(board)
            print(f"🎉 Player {current_player} wins!")
            break
        elif is_draw(board):
            print_board(board)
            print("It's a draw!")
            break

        # Switch players
        current_player = "O" if current_player == "X" else "X"

# Run the game
play_game()
```

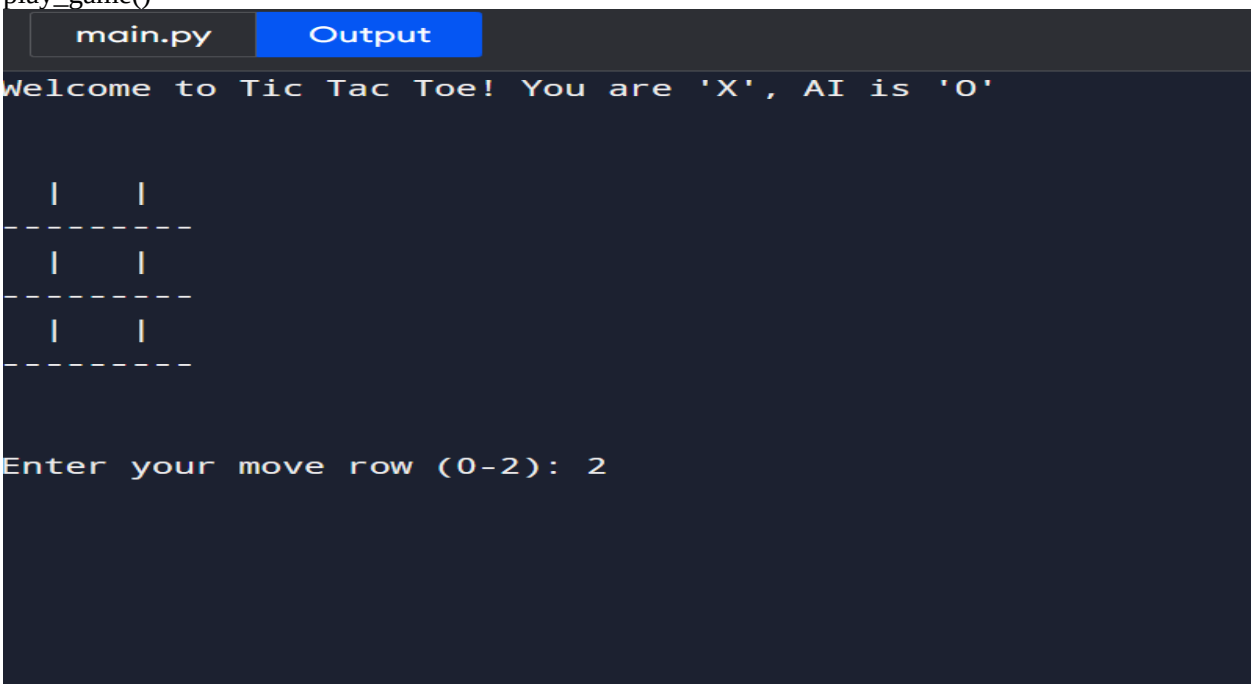


13.TO IMPLEMENT THE MINMAX ALGORITHM FOR GAMING

```
import math

# Initialize the board
board = [[" " for _ in range(3)] for _ in range(3)]

def print_board(board):
    print("\n")
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
    print("\n")

def check_winner(b, player):
    for row in b:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(b[row][col] == player for row in range(3)):
            return True
    if all(b[i][i] == player for i in range(3)) or \
       all(b[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_draw(b):
    return all(cell in ['X', 'O'] for row in b for cell in row)

def minimax(board, depth, is_maximizing):
    if check_winner(board, 'O'):
        return 1
    if check_winner(board, 'X'):
        return -1
    if is_draw(board):
        return 0

    if is_maximizing:
        best_score = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = 'O'
                    score = minimax(board, depth + 1, False)
                    board[i][j] = " "
                    best_score = max(best_score, score)
        return best_score
    else:
        best_score = math.inf
        for i in range(3):
            for j in range(3):
```

```
            if board[i][j] == " ":
                board[i][j] = 'X'
                score = minimax(board, depth + 1, True)
                board[i][j] = " "
                best_score = min(best_score, score)
        return best_score

def best_move():
    best_score = -math.inf
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = 'O'
                score = minimax(board, 0, False)
                board[i][j] = " "
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move

def play_game():
    print("Welcome to Tic Tac Toe! You are 'X', AI is 'O'")
    print_board(board)

    while True:
        # Human turn
        while True:
            try:
                row = int(input("Enter your move row (0-2): "))
                col = int(input("Enter your move column (0-2): "))
                if board[row][col] == " ":
                    board[row][col] = 'X'
                    break
                else:
                    print("Cell already occupied.")
            except:
                print("Invalid input. Try again.")

        print_board(board)

        if check_winner(board, 'X'):
            print(" 🎉 You win!")
            break
        if is_draw(board):
            print("It's a draw!")
            break

        # AI turn
        ai_move = best_move()
        board[ai_move[0]][ai_move[1]] = 'O'
```

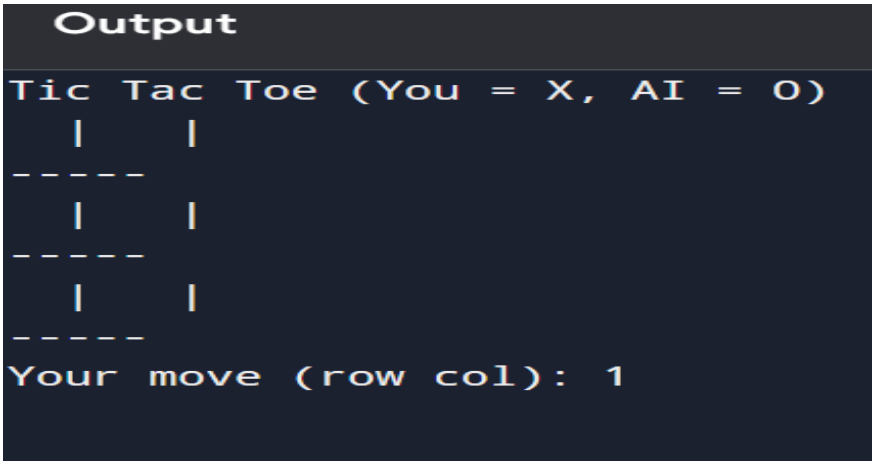
```
        print("AI made its move:")
        print_board(board)

        if check_winner(board, 'O'):
            print("😈 AI wins!")
            break
        if is_draw(board):
            print("It's a draw!")
            break

# Run the game
play_game()
```



14.TO IMPLEMENT THE ALPHA BETA PURNING ALGORITHM

```
import math

board = [[" "]*3 for _ in range(3)]

def print_board(): [print(" | ".join(r)) or print("-"*5) for r in board]

def win(p): return any(all(c==p for c in r) for r in board) or any(all(board[r][c]==p for r in range(3)) for c in range(3)) or all(board[i][i]==p for i in range(3)) or all(board[i][2-i]==p for i in range(3))

def draw(): return all(c in "XO" for r in board for c in r)

def minimax(alpha, beta, max_turn):
    if win('O'): return 1
    if win('X'): return -1
    if draw(): return 0
    best = -math.inf if max_turn else math.inf
```

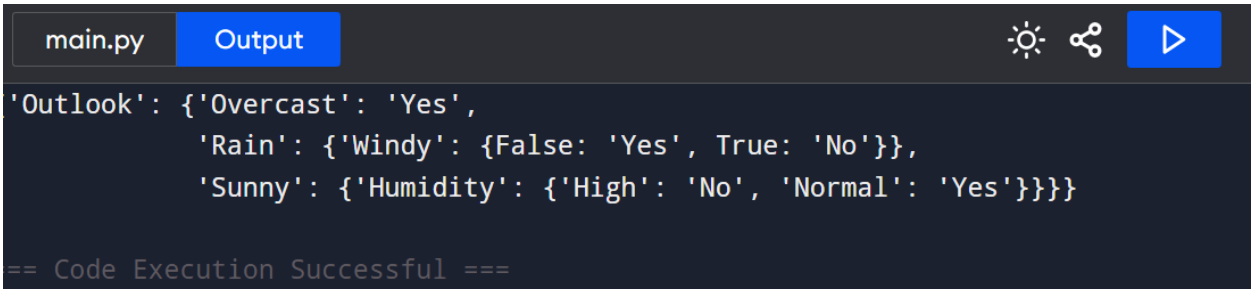
```
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = 'O' if max_turn else 'X'
                score = minimax(alpha, beta, not max_turn)
                board[i][j] = " "
                if max_turn:
                    best = max(best, score)
                    alpha = max(alpha, score)
                else:
                    best = min(best, score)
                    beta = min(beta, score)
                if beta <= alpha: return best
    return best

def best_move():
    move, best = (0, 0), -math.inf
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = 'O'
                score = minimax(-math.inf, math.inf, False)
                board[i][j] = " "
                if score > best:
                    best, move = score, (i, j)
    return move

def game():
    print("Tic Tac Toe (You = X, AI = O)")
    while True:
        print_board()
        if win('O'): print("AI wins!"); break
        if win('X'): print("You win!"); break
        if draw(): print("Draw!"); break
        try:
            r, c = map(int, input("Your move (row col): ").split())
            if board[r][c] != " ": raise
            board[r][c] = 'X'
        except: print("Invalid! Try again."); continue
        if not draw() and not win('X'):
            ai_r, ai_c = best_move()
            board[ai_r][ai_c] = 'O'

game()
```



15.TO IMPLEMENT THE DECISION TREE ALGORITHM

```
import math
from collections import Counter

# Sample dataset (Outlook, Temperature, Humidity, Windy, PlayTennis)
data = [
    ['Sunny', 'Hot', 'High', False, 'No'],
    ['Sunny', 'Hot', 'High', True, 'No'],
    ['Overcast', 'Hot', 'High', False, 'Yes'],
    ['Rain', 'Mild', 'High', False, 'Yes'],
    ['Rain', 'Cool', 'Normal', False, 'Yes'],
    ['Rain', 'Cool', 'Normal', True, 'No'],
    ['Overcast', 'Cool', 'Normal', True, 'Yes'],
    ['Sunny', 'Mild', 'High', False, 'No'],
    ['Sunny', 'Cool', 'Normal', False, 'Yes'],
    ['Rain', 'Mild', 'Normal', False, 'Yes'],
    ['Sunny', 'Mild', 'Normal', True, 'Yes'],
    ['Overcast', 'Mild', 'High', True, 'Yes'],
    ['Overcast', 'Hot', 'Normal', False, 'Yes'],
    ['Rain', 'Mild', 'High', True, 'No']
]

features = ['Outlook', 'Temperature', 'Humidity', 'Windy']

# Entropy calculation
def entropy(data):
    labels = [row[-1] for row in data]
    counts = Counter(labels)
    total = len(labels)
    return -sum((count/total) * math.log2(count/total) for count in counts.values())

# Info gain
def info_gain(data, index):
```

```
    total_entropy = entropy(data)
    values = set(row[index] for row in data)
    subset_entropy = 0
    for v in values:
        subset = [row for row in data if row[index] == v]
        subset_entropy += len(subset)/len(data) * entropy(subset)
    return total_entropy - subset_entropy

# ID3 algorithm
def build_tree(data, features):
    labels = [row[-1] for row in data]
    if labels.count(labels[0]) == len(labels):
        return labels[0]
    if not features:
        return Counter(labels).most_common(1)[0][0]

    gains = [info_gain(data, i) for i in range(len(features))]
    best_idx = gains.index(max(gains))
    best_feature = features[best_idx]

    tree = {best_feature: {}}
    values = set(row[best_idx] for row in data)

    for val in values:
        subset = [row[:best_idx] + row[best_idx+1:] for row in data if row[best_idx] == val]
        sub_features = features[:best_idx] + features[best_idx+1:]
        subtree = build_tree(subset, sub_features)
        tree[best_feature][val] = subtree

    return tree

# Train decision tree
decision_tree = build_tree(data, features)

# Print the tree
import pprint
pprint.pprint(decision_tree)
```



16.WRITE THE PROLOG PROGRAM FOR THE STUDENT TEACHER AND SUB CODE]

```
teacher(mr_smith, math101).
```

```
teacher(ms_jones, phy101).
teacher(mr_brown, chem101).

student(alice, math101).
student(bob, phy101).
student(charlie, math101).
student(diana, chem101).
student(eric, phy101).

teaches(Teacher, Student) :-
    student(Student, Code),
    teacher(Teacher, Code).

:- initialization(main).

main :-
    teaches(Teacher, Student),
    format('~w teaches ~w~n', [Teacher, Student]),
    fail.
main.
```

Output:

```
mr_smith teaches alice
ms_jones teaches bob
mr_smith teaches charlie
mr_brown teaches diana
ms_jones teaches eric
```

17. TO IMPLEMENT FEED FORWARD NEAURAL NETWORK PYHTON CODE

```
import numpy as np

# Sigmoid activation and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Training dataset (X: inputs, y: outputs)
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],
```

```
            [1],
            [1],
            [0]])  # XOR problem

# Initialize weights randomly
input_layer_neurons = X.shape[1]
hidden_layer_neurons = 2
output_neurons = 1

np.random.seed(42)
weights_input_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_neurons))

# Learning rate
lr = 0.5
epochs = 10000

for epoch in range(epochs):
    # Forward Propagation
    hidden_layer_input = np.dot(X, weights_input_hidden)
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

    # Updating weights
    weights_hidden_output += hidden_layer_output.T.dot(d_predicted_output) * lr
    weights_input_hidden += X.T.dot(d_hidden_layer) * lr

    # Print error every 1000 epochs
    if (epoch % 1000) == 0:
        print(f"Epoch {epoch} Error: {np.mean(np.abs(error))}")

print("\nFinal predicted output:")
print(np.round(predicted_output, 3))
```

main.py | Output

```
ERROR!
Epoch 0 Error: 0.4994268057883715
ERROR!
Epoch 1000 Error: 0.395381114673746 26
ERROR!
Epoch 2000 Error: 0.268896548955760 54
ERROR!
Epoch 3000 Error: 0.208730899776009 23
ERROR!
Epoch 4000 Error: 0.173858732869423 33
ERROR!
Epoch 5000 Error: 0.150934180346030 46
ERROR!
Epoch 6000 Error: 0.134587745107061 98
ERROR!
Epoch 7000 Error: 0.122261298878379 94
ERROR!
Epoch 8000 Error: 0.112581327790487 24
ERROR!
Epoch 9000 Error: 0.104742920852059 08

Final predicted output:
[[0.054]
 [0.898]
 [0.898]
 [0.135]]
```

18. PROLOG CODE FOR A DB WITH NAME DOB

```
person(alice, dob(1990, 5, 21)).
person(bob, dob(1985, 12, 1)).
person(charlie, dob(2000, 7, 15)).
person(diana, dob(1995, 10, 30)).

dob_of(Name, dob(Y,M,D)) :-
    person(Name, dob(Y,M,D)).

born_in_year(Year, Name) :-
    person(Name, dob(Year, _, _)).

born_before(Year, Name) :-
    person(Name, dob(Y,_,_)),
    Y < Year.

:- initialization(main).

main :-
    person(Name, dob(Y,M,D)),
    format('~w was born on ~w-~w-~w~n', [Name, Y, M, D]),
    fail.
main.
```

Output:

```
alice was born on 1990-5-21
bob was born on 1985-12-1
charlie was born on 2000-7-15
diana was born on 1995-10-30
```

19. SUM THE INTEGERS FROM 1 TO N

```
sum_to_n(0, 0).
sum_to_n(N, Sum) :-
    N > 0,
    N1 is N - 1,
    sum_to_n(N1, Sum1),
    Sum is Sum1 + N.

:- initialization(main).

main :-
```

```
    N = 10,
    sum_to_n(N, Sum),
    format("Sum of integers from 1 to ~w is ~w~n", [N, Sum]),
    halt.
```

Output:

```
Sum of integers from 1 to 10 is 55
```

20. PROLOG CODE FOR PLANETS DB

```
% Your planets database facts
planet(mercury, 1, terrestrial, no).
planet(venus, 2, terrestrial, no).
planet(earth, 3, terrestrial, no).
planet(mars, 4, terrestrial, no).
planet(jupiter, 5, gas_giant, yes).
planet(saturn, 6, gas_giant, yes).
planet(uranus, 7, gas_giant, yes).
planet(neptune, 8, gas_giant, yes).

% Query examples as rules
planet_info(Name, Order, Type, Rings) :-
    planet(Name, Order, Type, Rings).

planets_with_rings(Name) :-
    planet(Name, _, _, yes).

terrestrial_planets(Name) :-
    planet(Name, _, terrestrial, _).

% Initialization directive to print all planets on startup
:- initialization(main).

main :-
    format("Planets database:~n"),
    planet(Name, Order, Type, Rings),
    format("~w: Order ~w, Type ~w, Rings: ~w~n", [Name, Order, Type, Rings]),
    fail.
main.
```

```
?- planet_info(earth, Order, Type, Rings).
% Order = 3, Type = terrestrial, Rings = no.

?- planets_with_rings(Name).
% Name = jupiter ;
% Name = saturn ;
% Name = uranus ;
% Name = neptune.

?- terrestrial_planets(Name).
% Name = mercury ;
% Name = venus ;
% Name = earth ;
% Name = mars.
```


21. PROLOG PROGRAM FOR TOWERS OF HANOI

```
towers_of_hanoi(0, _, _, _, []) :- !.
towers_of_hanoi(N, Source, Target, Auxiliary, Moves) :-
    N > 0,
    N1 is N - 1,
    towers_of_hanoi(N1, Source, Auxiliary, Target, Moves1),
    Move = move(Source, Target),
    towers_of_hanoi(N1, Auxiliary, Target, Source, Moves2),
    append(Moves1, [Move|Moves2], Moves).

print_hanoi_moves(N) :-
    towers_of_hanoi(N, left, right, center, Moves),
    print_moves(Moves).

print_moves([]).
print_moves([move(From, To)|Rest]) :-
    format('Move disk from ~w to ~w~n', [From, To]),
    print_moves(Rest).

% Initialization directive to run automatically
:- initialization(main).

main :-
```

```
    print_hanoi_moves(3),
    halt.
```

Output:

```
Move disk from left to right
Move disk from left to center
Move disk from right to center
Move disk from left to right
Move disk from center to left
Move disk from center to right
Move disk from left to right
```

22. PROLOG CODE FOR PARTICULAR BIRD CAN FLY OR NOT

```
can_fly(eagle).
can_fly(parrot).
can_fly(sparrow).

cannot_fly(ostrich).
cannot_fly(penguin).
cannot_fly(kiwi).

bird_can_fly(Bird) :-
    can_fly(Bird),
    write(Bird), write(' can fly.'), !.

bird_can_fly(Bird) :-
    cannot_fly(Bird),
    write(Bird), write(' cannot fly.'), !.

bird_can_fly(Bird) :-
    write('I do not know if '), write(Bird), write(' can fly or not.').

:- initialization(main).

main :-
    bird_can_fly(eagle), nl,
    bird_can_fly(penguin), nl,
    bird_can_fly(duck), nl,
    halt.
```

Output:

```
eagle can fly.
penguin cannot fly.
I do not know if duck can fly or not.
```

23. PROLOG CODE TO IMPLEMENT THE FAMILY TREE

```
parent(john, mary).
parent(john, david).
parent(mary, susan).
parent(mary, james).
parent(david, lisa).

sibling(X, Y) :-
   parent(P, X),
   parent(P, Y),
   X \= Y.

grandparent(GP, GC) :-
   parent(GP, P),
   parent(P, GC).

ancestor(A, D) :-
   parent(A, D).
ancestor(A, D) :-
   parent(A, X),
   ancestor(X, D).

:- initialization(main).

main :-
   ( sibling(mary, david) -> writeln('Mary and David are siblings.') ; writeln('Not siblings.') ),
   ( grandparent(john, susan) -> writeln('John is grandparent of Susan.') ; writeln('No grandparent
relation.') ),
   ( ancestor(john, lisa) -> writeln('John is ancestor of Lisa.') ; writeln('No ancestor relation.') ),
   halt.
```

Output:

```
/usr/bin/ld: /tmp/gplcGuMvNh.o: in function `predicate(ma
(.text+0x468): undefined reference to `predicate(writeln/
/usr/bin/ld: /tmp/gplcGuMvNh.o: in function `Lpred6_1':
(.text+0x486): undefined reference to `predicate(writeln/
/usr/bin/ld: /tmp/gplcGuMvNh.o: in function `predicate(ma
(.text+0x528): undefined reference to `predicate(writeln/
/usr/bin/ld: /tmp/gplcGuMvNh.o: in function `Lpred7_1':
(.text+0x546): undefined reference to `predicate(writeln/
/usr/bin/ld: /tmp/gplcGuMvNh.o: in function `predicate(ma
(.text+0x5e9): undefined reference to `predicate(writeln/
/usr/bin/ld: /tmp/gplcGuMvNh.o:(.text+0x607): more undefi
collect2: error: ld returned 1 exit status
compilation failed
```

24. TO SUGGEST DIETING SYSTEM BASED ON DISEASE

```
diet_for_disease(diabetes, 'Low sugar, high fiber, balanced carbs').
diet_for_disease(hypertension, 'Low sodium, high potassium, DASH diet').
diet_for_disease(celiac, 'Gluten-free diet').
diet_for_disease(obesity, 'Low calorie, balanced nutrition, exercise').
diet_for_disease(anemia, 'Iron-rich foods like spinach, red meat, beans').

suggest_diet(Disease) :-
    diet_for_disease(Disease, Diet),
    format('For ~w, the recommended diet is: ~w.~n', [Disease, Diet]), !.

suggest_diet(Disease) :-
    format('Sorry, no diet information available for ~w.~n', [Disease]).

:- initialization(main).

main :-
    suggest_diet(diabetes),
    suggest_diet(hypertension),
    suggest_diet(asthma),
    halt.
```

Output:

```
For diabetes, the recommended diet is: Low sugar, high fiber, balanced carbs.
For hypertension, the recommended diet is: Low sodium, high potassium, DASH diet.
Sorry, no diet information available for asthma.
```

25. PROLOG CODE FOR TO IMPLEMEMT THE MONKEY BANANA PROBLRM

```
% monkey_state(MonkeyLocation, BoxLocation, MonkeyStatus)
% MonkeyStatus: on_floor or on_box

initial_state(state(door, door, on_floor)).

goal(state(_, _, has_bananas)).

move(state(From, Box, on_floor), To, state(To, Box, on_floor)) :-
   From \= To.

push_box(state(From, From, on_floor), To, state(To, To, on_floor)) :-
   From \= To.

climb_up(state(Location, Location, on_floor), state(Location, Location, on_box)).

climb_down(state(Location, Location, on_box), state(Location, Location, on_floor)).

grab_banana(state(ceiling, ceiling, on_box), state(ceiling, ceiling, has_bananas)).

location(door).
location(window).
location(ceiling).

plan(State, _, []) :-
   goal(State).

plan(State, Visited, [Action|Rest]) :-
   action(State, NextState, Action),
   \+ member(NextState, Visited),
   plan(NextState, [NextState|Visited], Rest).

action(State, NextState, move(To)) :-
   State = state(MonkeyLoc, BoxLoc, on_floor),
   location(To),
   move(State, To, NextState).

action(State, NextState, push_box(To)) :-
   State = state(MonkeyLoc, BoxLoc, on_floor),
   location(To),
   push_box(State, To, NextState).

action(State, NextState, climb_up) :-
   State = state(MonkeyLoc, BoxLoc, on_floor),
```

```
    climb_up(State, NextState).

action(State, NextState, climb_down) :-
    State = state(MonkeyLoc, BoxLoc, on_box),
    climb_down(State, NextState).

action(State, NextState, grab_banana) :-
    grab_banana(State, NextState).

solve :-
    initial_state(S),
    plan(S, [S], Actions),
    write('Actions to get bananas:'), nl,
    print_actions(Actions).

print_actions([]).
print_actions([H|T]) :-
    write(H), nl,
    print_actions(T).

:- initialization(main).

main :-
    solve,
    halt.
```

Output:

```
Actions to get bananas:
push_box(window)
push_box(ceiling)
climb_up
grab_banana
```

26. PROLOG CODE FOR FRUITS AND ITS COLOUR USING BACK TRACKING

```
fruit_color(apple, red).
fruit_color(banana, yellow).
fruit_color(grape, purple).
fruit_color(orange, orange).
fruit_color(lemon, yellow).
fruit_color(cherry, red).
```

```
fruits_of_color(Color, Fruit) :-
    fruit_color(Fruit, Color).

color_of_fruit(Fruit, Color) :-
    fruit_color(Fruit, Color).

:- initialization(main).

main :-
    writeln('Fruits with color red:'),
    fruits_of_color(red, Fruit1), writeln(Fruit1), fail; true,

    writeln('Colors of banana:'),
    color_of_fruit(banana, Color), writeln(Color), fail; true,

    halt.
```

```
Fruits with color red:
apple
cherry
Colors of banana:
yellow
```

27.PROLOG PROGRAM TO IMPLEMENT BEST FIRST SERCH ALGORITHM

```
% Example graph edges: edge(Node, Neighbor, Cost)
edge(a, b, 1).
edge(a, c, 4).
edge(b, d, 2).
edge(c, d, 5).
edge(b, e, 7).
edge(d, goal, 3).
edge(e, goal, 1).

% Heuristic estimate for each node to goal (lower is better)
heuristic(a, 7).
heuristic(b, 6).
heuristic(c, 2).
heuristic(d, 1).
heuristic(e, 0).
heuristic(goal, 0).

% Best-First Search algorithm:
% best_first(Start, Goal, Path)
best_first(Start, Goal, Path) :-
    heuristic(Start, H),
    best_first_search([node(Start, [Start], H)], Goal, RevPath),
```

```
    reverse(RevPath, Path).

% best_first_search(OpenList, Goal, Path)
best_first_search([node(Goal, Path, _)|_], Goal, Path).

best_first_search(OpenList, Goal, FinalPath) :-
    % Select node with lowest heuristic value (best node)
    select_best(OpenList, node(Current, Path, _), RestOpen),
    % Find successors
    findall(node(Next, [Next|Path], H),
            (edge(Current, Next, _), \+ member(Next, Path), heuristic(Next, H)),
            Successors),
    % Add successors to open list
    append(RestOpen, Successors, NewOpen),
    % Sort open list by heuristic value ascending
    sort_open_list(NewOpen, SortedOpen),
    best_first_search(SortedOpen, Goal, FinalPath).

% select_best(OpenList, BestNode, RestOpenList)
select_best([Best|Rest], Best, Rest).

% sort_open_list(OpenList, Sorted)
sort_open_list(OpenList, Sorted) :-
    predsort(compare_nodes, OpenList, Sorted).

compare_nodes(Order, node(_, _, H1), node(_, _, H2)) :-
    (H1 < H2 -> Order = (<)
    ; H1 > H2 -> Order = (>)
    ; Order = (=)).
```

```
?- best_first(a, goal, Path).
Path = [a, c, d, goal].
```

28.PROLOG PROGRAM FOR MEDICAL DAIGNOSIS

```
% Medical Diagnosis Program

% Facts
symptom(fever).
symptom(cough).
symptom(sore_throat).
symptom(headache).
symptom(fatigue).
symptom(nausea).

% Rules
diagnosis(flu) :- symptom(fever), symptom(cough), symptom(sore_throat).
diagnosis(cold) :- symptom(cough), symptom(sore_throat), symptom(headache).
diagnosis(migraine) :- symptom(headache), symptom(fatigue).
diagnosis(gastroenteritis) :- symptom(nausea), symptom(fever).
```

% Query
% To diagnose, use the following query format:
% ?- diagnosis(X).

```
Possible diagnosis:
flu
```

29. PROLOG PROGRSM FOR FORWARD CHAINING INCORPARATE REQUIRED QURIES

```
% Initial facts (known symptoms of the patient)
symptom(fever).
symptom(cough).
symptom(sore_throat).

% Rules to derive new facts
rule(flu) :- symptom(fever), symptom(cough), symptom(sore_throat).
rule(cold) :- symptom(cough), symptom(sore_throat), symptom(headache).
rule(migraine) :- symptom(headache), symptom(fatigue).
rule(gastroenteritis) :- symptom(nausea), symptom(fever).

% Forward chaining engine
forward :-
    rule(Diagnosis),
    \+ known(Diagnosis),    % If not already known
    assertz(known(Diagnosis)),
    write('Inferred: '), writeln(Diagnosis),
    fail.

forward :- writeln('Forward chaining complete.').

% Start point
:- dynamic known/1.
:- initialization(main).

main :-
    writeln('Running Forward Chaining...'),
    forward,
    findall(D, known(D), Diagnoses),
    (Diagnoses = [] ->
        writeln('No diagnosis inferred.')
    ;
        writeln('Final Diagnoses:'),
        print_list(Diagnoses)
    ),
    halt.

% Helper to print a list
print_list([]).
```

```
print_list([H|T]) :-
    writeln(H),
    print_list(T).
```

```
cd /home/cg/root/682c97df2537e
gprolog --consult-file main.pg
Running Forward Chaining...
Inferred: flu
Forward chaining complete.
Final Diagnoses:
flu
```

30. PROLOG PROGRSM FOR BACKWARD CHAINING INCORPARATE REQUIRED QURIES

```
% Patient's known symptoms (facts)
has_symptom(fever).
has_symptom(cough).
has_symptom(sore_throat).

% Diagnosis rules (goals)
diagnosis(flu) :-
    has_symptom(fever),
    has_symptom(cough),
    has_symptom(sore_throat).

diagnosis(cold) :-
    has_symptom(cough),
    has_symptom(sore_throat),
    has_symptom(headache).

diagnosis(migraine) :-
    has_symptom(headache),
    has_symptom(fatigue).

diagnosis(gastroenteritis) :-
    has_symptom(nausea),
    has_symptom(fever).

% Program entry
:- initialization(main).

main :-
    writeln('Backward Chaining Diagnosis System'),
```

```
    findall(D, diagnosis(D), Diagnoses),
    ( Diagnoses = [] ->
        writeln('No diagnosis found.')
    ;
        writeln('Possible diagnosis:'),
        print_list(Diagnoses)
    ),
    halt.

% Helper to print list
print_list([]).
print_list([H|T]) :-
    writeln(H),
    print_list(T).
```

```
GNU Prolog 1.5.0 (64 bits)
Compiled Dec 17 2024, 14:00:19 with gcc
Copyright (C) 1999-2024 Daniel Diaz

compiling /home/cg/root/682c97df2537e/main.pg for byte code...
/home/cg/root/682c97df2537e/main.pg compiled, 42 lines read - 3122 bytes written, 3 ms
warning: /home/cg/root/682c97df2537e/main.pg:26: user directive caused exception: error
    (existence_error(procedure,writeln/1),main/0)
| ?-
has_symptom(fever).
has_symptom(cough).
has_symptom(sore_throat).
Backward Chaining Diagnosis System
Possible diagnosis:
flu
```

31. CREATE A WEB BLOG USING WORD PRESS TO DEMONSTRASTE ANCHOR TAG,TITLE TAG,ETC

```
% Define some basic HTML tag facts
html_tag(a, 'Defines a hyperlink').
html_tag(title, 'Defines the title of the document shown in the browser tab').
html_tag(h1, 'Defines a top-level heading').
html_tag(p, 'Defines a paragraph').
html_tag(br, 'Inserts a line break').

% Define examples for tags
html_example(a, '<a href="https://openai.com">Visit OpenAI</a>').
html_example(title, '<title>My Blog</title>').
html_example(h1, '<h1>Welcome to My Blog</h1>').
html_example(p, '<p>This is a sample paragraph.</p>').
html_example(br, 'First Line<br>Second Line').
```

```
% Rule to display info about an HTML tag
describe_tag(Tag) :-
    html_tag(Tag, Description),
    html_example(Tag, Example),
    format('Tag: <~w>~n', [Tag]),
    format('Description: ~w~n', [Description]),
    format('Example Usage: ~w~n~n', [Example]).

% Run all tags
show_all_tags :-
    html_tag(Tag, _),
    describe_tag(Tag),
    fail.

show_all_tags.

% Initialization
:- initialization(main).

main :-
    writeln('HTML Tags Explained in Prolog:'),
    show_all_tags,
    halt.
```

```
| ?- HTML Tags Explained in Prolog:
Tag: <a>
Description: Defines a hyperlink
Example Usage: <a href="https://openai.com">Visit OpenAI</a>

Tag: <title>
Description: Defines the title of the document shown in the browser tab
Example Usage: <title>My Blog</title>

Tag: <h1>
Description: Defines a top-level heading
Example Usage: <h1>Welcome to My Blog</h1>
```

32. Write a Prolog program to implement pattern matching

```
% Base case: empty pattern always matches
match_pattern([], _).

% Pattern and list start with same element, check rest
match_pattern([P|PT], [P|LT]) :-
    match_pattern(PT, LT).
```

```
% Pattern doesn't match first element of list, try rest of list
match_pattern(Pattern, [_|LT]) :-
    match_pattern(Pattern, LT).

% Sample query:
% ?- match_pattern([b, c], [a, b, c, d]).
% true.
```

```
GNU Prolog 1.5.0 (64 bits)
Compiled Dec 17 2024, 14:00:19 with gcc
Copyright (C) 1999-2024 Daniel Diaz

compiling /home/cg/root/682c97df2537e/main.pg for byte code...
/home/cg/root/682c97df2537e/main.pg compiled, 13 lines read - 648 bytes written, 3 ms
| ?- ?- match_pattern([b, c], [a, b, c, d]).
true.

?- match_pattern([b, d], [a, b, c, d]).
true.

?- match_pattern([b, e], [a, b, c, d]).
false.
```

33. Write a Prolog program to find the number of vowels

```
% Base case: empty list has 0 vowels
count_vowels([], 0).

% Head is a vowel: increment count
count_vowels([H|T], Count) :-
    is_vowel(H),
    count_vowels(T, Count1),
    Count is Count1 + 1.

% Head is not a vowel: continue without increment
count_vowels([H|T], Count) :-
    \+ is_vowel(H),
    count_vowels(T, Count).

% Define vowels
is_vowel(a).
is_vowel(e).
is_vowel(i).
is_vowel(o).
is_vowel(u).
is_vowel(A) :- downcase_atom(A, L), is_vowel(L).  % Handle uppercase
```

```
GNU Prolog 1.5.0 (64 bits)
Compiled Dec 17 2024, 14:00:19 with gcc
Copyright (C) 1999-2024 Daniel Diaz

compiling /home/cg/root/682c97df2537e/main.pg for byte code...
/home/cg/root/682c97df2537e/main.pg compiled, 20 lines read - 1641 bytes written, 3 ms
| ?- ?- count_vowels([h, e, l, l, o], Count).
Count = 2.

?- count_vowels([P, r, O, L, o, G], Count).
Count = 2.
```