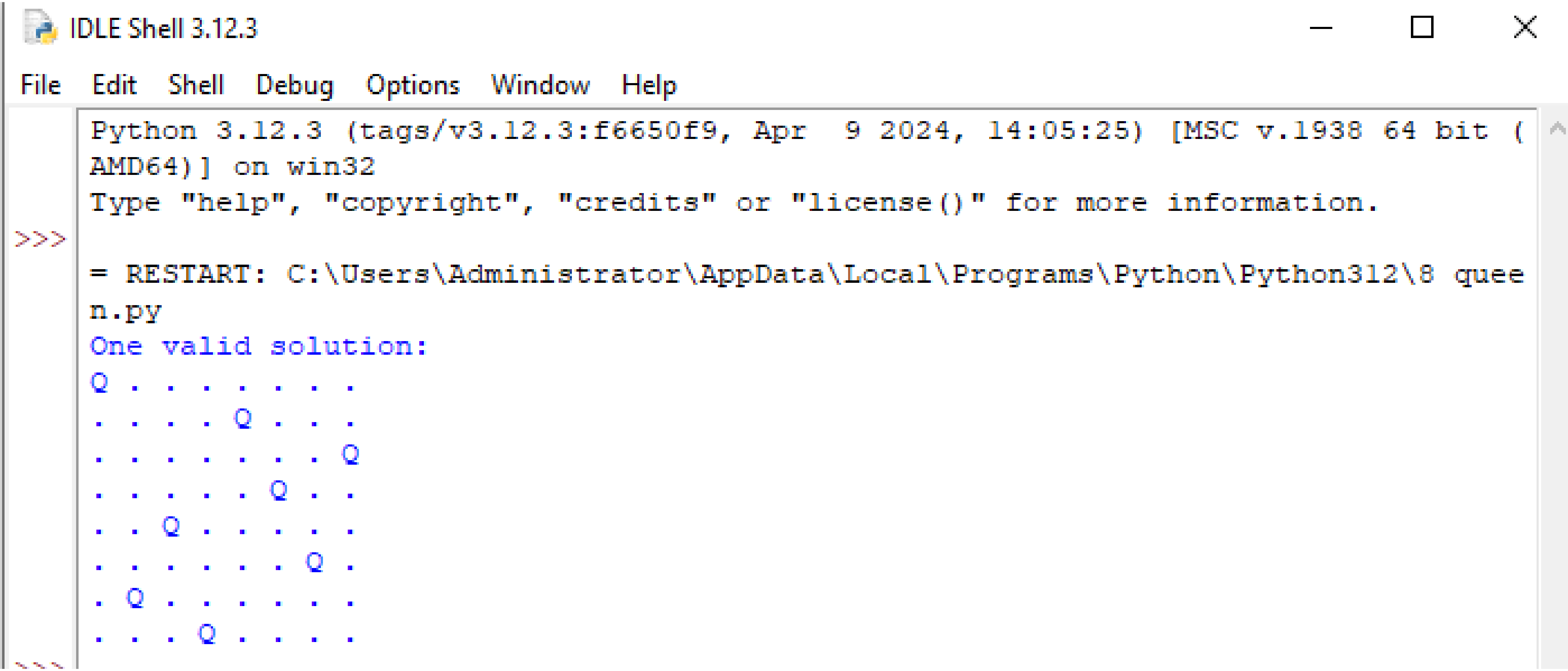
8 queen.py - C:\Users\Administrator\AppData\Local\Programs\Python\Python312\8 queen.py (3.12.3)

File Edit Format Run Options Window Help

```
N = 8

def printSolution(board):
    for row in board:
        for val in row:
            print("Q" if val else ".", end=" ")
        print()

def isSafe(board, row, col):
    for i in range(row):
        if board[i][col]:
            return False
    i, j = row - 1, col - 1
    while i >= 0 and j >= 0:
        if board[i][j]:
            return False
        i -= 1
        j -= 1
    i, j = row - 1, col + 1
    while i >= 0 and j < N:
        if board[i][j]:
            return False
        i -= 1
        j += 1
    return True
```

```python
def solve(board, row):
    if row == N:
        printSolution(board)
        return True  # Stop after first solution
    for col in range(N):
        if isSafe(board, row, col):
            board[row][col] = 1
            if solve(board, row + 1):
                return True
            board[row][col] = 0
    return False

def eightQueens():
    board = [[0 for _ in range(N)] for _ in range(N)]
    if not solve(board, 0):
        print("No solution found.")

eightQueens()
```


IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Administrator\AppData\Local\Programs\Python\Python312\8 queen.py
One valid solution:
Q . . . .
. . . . Q . . .
. . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .

a star.py - C:/Users/Administrator/AppData/Local/Programs/Python/Python312/a star.py (3.12.3)

File Edit Format Run Options Window Help

```python
import heapq

class Node:
    def __init__(self, position, parent=None, g=0, h=0):
        self.position = position
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])A
def a_star(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_list = []
    heapq.heappush(open_list, Node(start, None, 0, heuristic(start, goal)))
    closed_set = set()

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.position == goal:
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]

        closed_set.add(current_node.position)

        for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            new_pos = (current_node.position[0] + dr, current_node.position[1] + dc)

            if (0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols and
                grid[new_pos[0]][new_pos[1]] == 0 and new_pos not in closed_set):
```

```
                new_node = Node(new_pos, current_node, current_node.g + 1, heuristic(new_pos, goal))
                heapq.heappush(open_list, new_node)

    return None
warehouse_grid = [
    [0, 0, 0, 0, 1],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
start_position = (0, 0)
goal_position = (4, 4)

path = a_star(warehouse_grid, start_position, goal_position)
print("Optimal Path:", path)
```

IDLE Shell 3.12.3

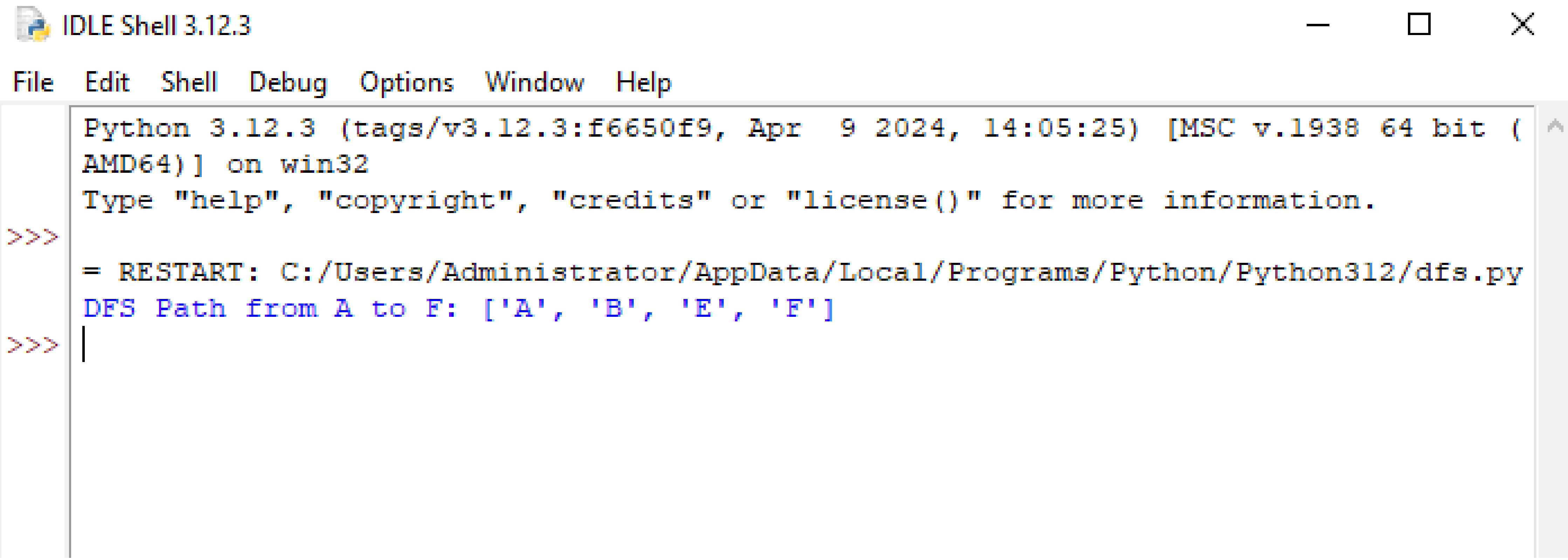
File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python312/a star.py
Optimal Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4)]
```

dfs.py - C:/Users/Administrator/AppData/Local/Programs/Python/Python312/dfs.py (3.12.3)

File Edit Format Run Options Window Help

```python
warehouse_graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
def dfs(graph, start, goal, visited=None, path=None):
    if visited is None:
        visited = set()
    if path is None:
        path = []

    visited.add(start)
    path.append(start)

    if start == goal:
        return path

    for neighbor in graph[start]:
        if neighbor not in visited:
            result = dfs(graph, neighbor, goal, visited, path[:])
            if result:
                return result

    return None

start_node = 'A'
goal_node = 'F'
path_found = dfs(warehouse_graph, start_node, goal_node)
print(f"DFS Path from {start_node} to {goal_node}: {path_found}")
```


IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python312/dfs.py
DFS Path from A to F: ['A', 'B', 'E', 'F']
>>>

min max algorithm.py - C:/Users/Administrator/AppData/Local/Programs/Python/Python312/min max algorithm.py (3.12.3)

File Edit Format Run Options Window Help

```
PLAYER_X = 1
PLAYER_O = -1
EMPTY = 0

def evaluate(board):
    for row in range(3):
        if board[row][0] == board[row][1] == board[row][2] != EMPTY:
            return board[row][0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != EMPTY:
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != EMPTY:
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != EMPTY:
        return board[0][2]
    return 0

def isMovesLeft(board):
    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                return True
    return False

def minimax(board, isMax):
    score = evaluate(board)

    if score == PLAYER_X:
        return score

    if score == PLAYER_O:
        return score

    if not isMovesLeft(board):
        return 0
```

min max algorithm.py - C:/Users/Administrator/AppData/Local/Programs/Python/Python312/min max algorithm.py (3.12.3)

File Edit Format Run Options Window Help

```
    if isMax:
        best = -float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = PLAYER_X
                    best = max(best, minimax(board, not isMax))
                    board[row][col] = EMPTY
        return best
    else:
        best = float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = PLAYER_O
                    best = min(best, minimax(board, not isMax))
                    board[row][col] = EMPTY
        return best

def findBestMove(board):
    bestVal = -float('inf')
    bestMove = (-1, -1)

    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER_X
                moveVal = minimax(board, False)
                board[row][col] = EMPTY
                if moveVal > bestVal:
                    bestMove = (row, col)
                    bestVal = moveVal

    return bestMove
```

```
    return bestMove

def printBoard(board):
    for row in board:
        print(" ".join(["X" if x == PLAYER_X else "O" if x == PLAYER_O else "." for x in row]))
board = [
    [PLAYER_X, PLAYER_O, PLAYER_X],
    [PLAYER_O, PLAYER_X, EMPTY],
    [EMPTY, PLAYER_O, PLAYER_X]
]

print("Current Board:")
printBoard(board)

move = findBestMove(board)
print(f"Best Move: {move}")
board[move[0]][move[1]] = PLAYER_X
print("\nBoard after best move:")
printBoard(board)
```

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python312/min ma
x algorithm.py
Current Board:
X O X
O X .
. O X
Best Move: (1, 2)

Board after best move:
X O X
O X X
. O X
>>>
```

Execute | Source Code | Share | Help

```prolog
minimum(X,Y,X) :- X =< Y. % If X is less than or equal to Y, X is the minimum.
minimum(X,Y,Y) :- X > Y.  % If X is greater than Y, Y is the minimum.

maximum(X,Y,X) :- X >= Y. % If X is greater than or equal to Y, X is the maximum.
maximum(X,Y,Y) :- X < Y.  % If X is less than Y, Y is the maximum.

:- initialization(main).

main :-
    minimum(3, 5, Min),
    write(Min), nl.
```

```
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/680cb07615db0/main.pg for byte code...
/home/cg/root/680cb07615db0/main.pg compiled, 10 lines read - 1003 bytes written, 4 ms
3
| ?-
```

unification and resolution algorithm.py - C:/Users/kavit/AppData/Local/Programs/Python/Python313/unification and resolution algorithm.py (3.13.2)

File Edit Format Run Options Window Help

```python
# Function to unify a variable with a term
def unify_var(var, x, theta):
    if var in theta:
        return unify(theta[var], x, theta)
    elif x in theta:
        return unify(var, theta[x], theta)
    else:
        theta[var] = x
        return theta

# Function to apply resolution rule using known facts and implications
def resolution(kb, facts, query):
    # First, check if query matches any known fact
    for fact in facts:
        if unify(fact, query, {}) is not None:
            return True

    # Then, use implications
    for clause in kb:
        premise, conclusion = clause
        theta = unify(conclusion, query, {})
        if theta is not None:
            # Try to prove the premise using known facts
            new_premise = [theta.get(arg, arg) for arg in premise]
            if resolution(kb, facts, new_premise):
                return True
    return False

# Implications in the form: [premise, conclusion]
knowledge_base = [
    [["Human", "x"], ["Mortal", "x"]],  # Human(x) => Mortal(x)
]

# Known facts
facts = [
    ["Human", "John"]
]

# Query
query = ["Mortal", "John"]

# Apply resolution
if resolution(knowledge_base, facts, query):
    print("Query is resolved: John is Mortal")
else:
    print("Query could not be resolved")
```

IDLE Shell 3.13.2

File Edit Shell Debug Options Window Help

```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======================================== RESTART: C:/Users/kavit/AppData/Local/Programs/Python/Python313/unification and resolution algorithm.py
Query is resolved: John is Mortal
>>> =
```

backward chaining.py - C:/Users/dell/AppData/Local/Programs/Python/Python313/backward chaining.py (3.13.2)

File Edit Format Run Options Window Help

```python
# Knowledge Base (Rules in IF-THEN format)
knowledge_base = {
    "flu": [["cough", "fever"]],
    "fever": [["sore_throat"]],
}

# Known facts
facts = {"sore_throat", "cough"}

# Backward chaining function
def backward_chaining(goal):
    if goal in facts:  # If the goal is a known fact, return True
        return True
    if goal in knowledge_base:  # If the goal has rules in KB
        for conditions in knowledge_base[goal]:  # Check each rule
            if all(backward_chaining(cond) for cond in conditions):  # Recursively verify
                return True
    return False  # If no rule or fact supports the goal, return False

# Query: Does the patient have flu?
query = "flu"
if backward_chaining(query):
    print(f"The patient is diagnosed with {query}.")
else:
    print(f"The patient does NOT have {query}.")
```

IDLE Shell 3.13.2

File Edit Shell Debug Options Window Help

```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================================================== RESTART: C:/Users/dell/AppData/Local/Programs/Python/Python313/backward chaining.py ===
The patient is diagnosed with flu.
>>>
```

forward chaining.py - C:/Users/kavit/AppData/Local/Programs/Python/Python313/forward chaining.py (3.13.2)

File Edit Format Run Options Window Help

```python
# Knowledge Base: Rules in IF-THEN format
knowledge_base = [
    (["cough", "fever"], "flu"),
    (["sore_throat", "runny_nose"], "cold"),
    (["sore_throat"], "fever")  # Sore throat can lead to fever
]

# Given initial facts
facts = {"cough", "sore_throat"}

# Forward Chaining Function
def forward_chaining():
    inferred = True  # Keep looping as long as new facts are added
    while inferred:
        inferred = False  # Stop if no new fact is added in an iteration

        for conditions, conclusion in knowledge_base:
            if all(condition in facts for condition in conditions) and conclusion not in facts:
                facts.add(conclusion)  # Add the inferred fact
                inferred = True  # Mark that we inferred a new fact

# Run forward chaining
forward_chaining()

# Check if flu or cold is inferred
if "flu" in facts:
    print("The patient is diagnosed with flu.")
elif "cold" in facts:
    print("The patient is diagnosed with cold.")
else:
    print("No conclusive diagnosis could be made.")
```

Ln: 32 Col: 0

31°C Search ENG 19:24

IDLE Shell 3.13.2

File Edit Shell Debug Options Window Help

```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================================================== RESTART: C:/Users/kavit/AppData/Local/Programs/Python/Python313/forward chaining.py
The patient is diagnosed with flu.
>>> |
```

blocks world program.py - C:/Users/dell/AppData/Local/Programs/Python/Python313/blocks world program.py (3.13.2)

File Edit Format Run Options Window Help

```python
class BlocksWorld:
    def __init__(self):
        # Initial block configuration
        self.state = {
            "A": "B",      # A is on B
            "B": "table", # B is on table
            "C": "table"  # C is on table
        }

        # Goal block configuration
        self.goal = {
            "A": "B",      # A should be on B
            "B": "C",      # B should be on C
            "C": "table"  # C should be on table
        }

    def is_goal_state(self):
        return self.state == self.goal

    def move(self, block, destination):
        if block in self.state and self.state[block] != destination:
            print(f"Moving {block} from {self.state[block]} to {destination}")
            self.state[block] = destination

    def plan_moves(self):
        print("\nInitial State:", self.state)
        max_iterations = 10  # Prevent infinite loops in simple logic
        iteration = 0
        while not self.is_goal_state() and iteration < max_iterations:
            iteration += 1
            for block, target in self.goal.items():
                if self.state[block] != target:
                    # Ensure destination is clear (basic check)
                    if target != "table":
                        # Can't move to a block that is not free
                        occupied_blocks = [k for k, v in self.state.items() if v == target]
                        if occupied_blocks:
                            continue
                    self.move(block, target)
        print("\nFinal State:", self.state)
        if self.is_goal_state():
            print("Goal state reached!")
        else:
            print("Goal state not fully achieved within iteration limit.")

# Run the Blocks World Solver
bw = BlocksWorld()
bw.plan_moves()
```

IDLE Shell 3.13.2

File Edit Shell Debug Options Window Help

```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================================================= RESTART: C:/Users/dell/AppData/Local/Programs/Python/Python313/blocks world program.py ==

Initial State: {'A': 'B', 'B': 'table', 'C': 'table'}
Moving B from table to C

Final State: {'A': 'B', 'B': 'C', 'C': 'table'}
Goal state reached!
>>>
```

```python
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

experience = ctrl.Antecedent(np.arange(0, 21, 1), 'experience')
success_rate = ctrl.Antecedent(np.arange(0, 101, 1), 'success_rate')
performance = ctrl.Consequent(np.arange(0, 101, 1), 'performance')


experience['low'] = fuzz.trimf(experience.universe, [0, 0, 10])
experience['medium'] = fuzz.trimf(experience.universe, [5, 10, 15])
experience['high'] = fuzz.trimf(experience.universe, [10, 20, 20])

success_rate['low'] = fuzz.trimf(success_rate.universe, [0, 0, 50])
success_rate['medium'] = fuzz.trimf(success_rate.universe, [25, 50, 75])
success_rate['high'] = fuzz.trimf(success_rate.universe, [50, 100, 100])

performance['poor'] = fuzz.trimf(performance.universe, [0, 0, 50])
performance['average'] = fuzz.trimf(performance.universe, [25, 50, 75])
performance['excellent'] = fuzz.trimf(performance.universe, [50, 100, 100])

rule1 = ctrl.Rule(experience['low'] & success_rate['low'], performance['poor'])
rule2 = ctrl.Rule(experience['medium'] | success_rate['medium'], performance['average']
rule3 = ctrl.Rule(experience['high'] & success_rate['high'], performance['excellent'])


performance_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
performance_sim = ctrl.ControlSystemSimulation(performance_ctrl)


performance_sim.input['experience'] = 12
performance_sim.input['success_rate'] = 70


performance_sim.compute()


print(f"Predicted Performance Score: {performance_sim.output['performance']:.2f}")
```

```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:/Users/DSEPY05118/AppData/Local/Programs/Python/Python312/POAI PROBLEM-3.py
Predicted Performance Score: 57.52
```