**1.**

```
from collections import deque

# Define the goal state
GOAL_STATE = ((0, 7, 2),
              (1, 4, 5),
              (5, 8, 3))

# Function to check if a state is the goal state
def is_goal(state):
    return state == GOAL_STATE

# Function to find the position of the blank (0) tile
def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

# Function to generate successors of the current state
def successors(state):
    x, y = find_blank(state)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    result = []
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [list(row) for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            result.append(tuple(tuple(row) for row in new_state))
    return result

# BFS algorithm
def bfs(initial_state):
    queue = deque([(initial_state, [])])
    visited = set()
    while queue:
        state, path = queue.popleft()
        if state in visited:
            continue
        visited.add(state)
        if is_goal(state):
            return path + [state]
        for successor in successors(state):
            queue.append((successor, path + [state]))
    return None

# Initial state
initial_state = ((5, 3, 2),
```

```
            (7, 1, 6),
            (8, 4, 0))

# Solve the puzzle
solution = bfs(initial_state)

# Print the solution
if solution:
    print("Solution found in {} steps:".format(len(solution) - 1))
    for step in solution:
        for row in step:
            print(row)
        print()
else:
    print("No solution found.")
```

```
┌──(bee㉿Bee)-[~/Desktop]
└─$ python3 bfs.py
No solution found.
```

**2.**

```
def solve_puzzle():
    for T in range(1, 10):  # T cannot be 0 (as it's a leading digit)
        for W in range(10):
            if W == T:
                continue
            for O in range(10):
                if O == T or O == W:
                    continue

                two_two_T_W_O = 2 * (100 * T + 10 * W + O)

                if two_two_T_W_O >= 1000 and two_two_T_W_O <= 9999:
                    F = two_two_T_W_O // 1000
                    rest = two_two_T_W_O % 1000
                    O_ = rest // 100
                    U = (rest % 100) // 10
                    R = rest % 10

                    if (F != O and F != U and F != R and
                        O_ != F and O_ != U and O_ != R and
                        U != F and U != O and U != R and
                        R != F and R != O and R != U):

                        if (T != F and T != O_ and T != U and T != R and
                            W != F and W != O_ and W != U and W != R):

                            if O == O_:  # Check if O equals O_
                                print(f"T = {T}, W = {W}, O = {O}, F = {F}, U = {U}, R = {R}")

# Call the function to solve the puzzle
solve_puzzle()
```

```
┌──(bee㉿Bee)-[~/Desktop]
└─$ python3 twotwofour.py
T = 7, W = 3, O = 4, F = 1, U = 6, R = 8
T = 7, W = 6, O = 5, F = 1, U = 3, R = 0
T = 8, W = 3, O = 6, F = 1, U = 7, R = 2
T = 8, W = 4, O = 6, F = 1, U = 9, R = 2
T = 8, W = 6, O = 7, F = 1, U = 3, R = 4
T = 9, W = 2, O = 8, F = 1, U = 5, R = 6
T = 9, W = 3, O = 8, F = 1, U = 7, R = 6
```

**3.**

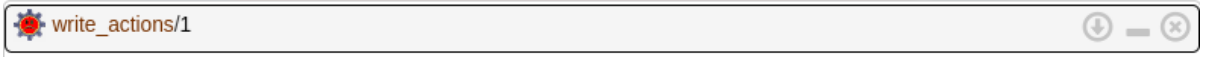
```
% Fact: Meena is a student.
% This fact states that Meena is a student.
student(meena).

% Fact: Meena is a hard worker.
% This fact states that Meena is a hard worker.
hard_worker(meena).

% Rule: Every student is sincere.
% This rule states that if someone is a student, then they are sincere.
sincere(X) :- student(X).

% Rule: All who are sincere and hard workers will succeed in their career.
% This rule states that if someone is both sincere and a hard worker,
% then they will succeed in their career.
succeed_in_career(X) :- sincere(X), hard_worker(X).

% Main rule to check and write the result
% The main rule checks if Meena will succeed in her career.
% If the condition succeed_in_career(meena) is true, it writes
% 'Meena will succeed in her career.' to the console.
% If the condition is false, it writes 'Meena will not succeed in her career.'
main :-
    ( succeed_in_career(meena) ->
        write('Meena will succeed in her career.');
        write('Meena will not succeed in her career.')
    ).
% Run the main rule
% This directive tells Prolog to run the main rule when the program starts.
:- main.
```




Meena will succeed in her career.