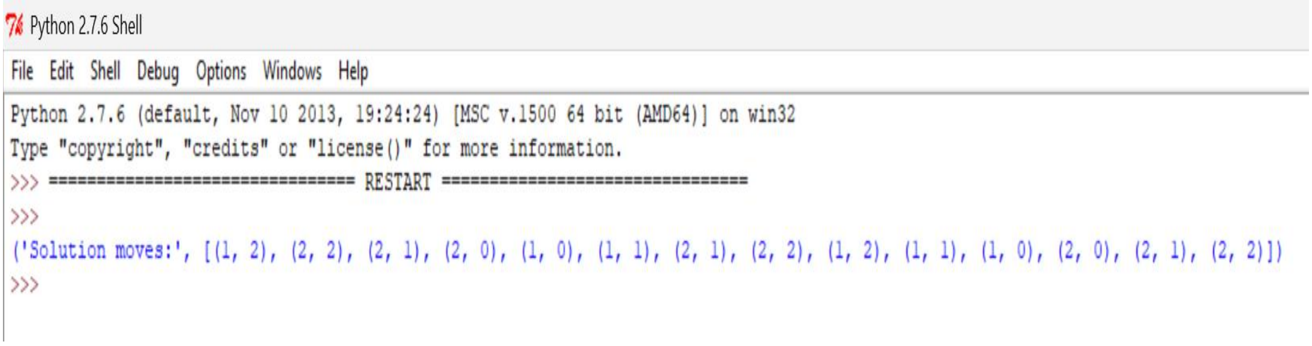
PROGRAM 1

```
import heapq

class PuzzleState:
    def __init__(self, board, empty_pos, moves, cost):
        self.board = board
        self.empty_pos = empty_pos
        self.moves = moves
        self.cost = cost

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

def get_neighbors(state):
    neighbors = []
    x, y = state.empty_pos
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_board = [row[:] for row in state.board]
            new_board[x][y], new_board[nx][ny] = new_board[nx][ny], new_board[x][y]
            neighbors.append(PuzzleState(new_board, (nx, ny), state.moves + [(nx, ny)], 0))
    return neighbors

def heuristic(state, goal):
    cost = 0
    for i in range(3):
        for j in range(3):
            val = state.board[i][j]
            if val != 0:
                gx, gy = goal[val]
                cost += abs(gx - i) + abs(gy - j)
    return cost

def solve_puzzle(start_board, goal_board):
    goal_positions = {val: (i, j) for i, row in enumerate(goal_board) for j, val in enumerate(row)}
    empty_pos = next((i, j) for i, row in enumerate(start_board) for j, val in enumerate(row) if val == 0)
    start_state = PuzzleState(start_board, empty_pos, [], heuristic(PuzzleState(start_board, empty_pos,
[], 0), goal_positions))

    open_list = []
    heapq.heappush(open_list, start_state)
    visited = set()

    while open_list:
        current_state = heapq.heappop(open_list)
        if tuple(map(tuple, current_state.board)) in visited:
            continue

        visited.add(tuple(map(tuple, current_state.board)))
        if current_state.board == goal_board:
            return current_state.moves

        for neighbor in get_neighbors(current_state):
            neighbor.cost = len(neighbor.moves) + heuristic(neighbor, goal_positions)
            heapq.heappush(open_list, neighbor)
    return None

# Example usage:
start_board = [
    [1, 2, 3],
    [4, 0, 5],
    [6, 7, 8]
]

goal_board = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

solution = solve_puzzle(start_board, goal_board)
if solution:
    print("Solution moves:", solution)
else:
    print("No solution found.")
```

OUTPUT:



```
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
('Solution moves:', [(1, 2), (2, 2), (2, 1), (2, 0), (1, 0), (1, 1), (2, 1), (2, 2), (1, 2), (1, 1), (1, 0), (2, 0), (2, 1), (2, 2)])
>>>
```