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

Program:

```
import heapq

class PuzzleState:
    def __init__(self, board, goal, moves=0):
        self.board = board
        self.goal = goal
        self.moves = moves
        self.zero = board.index(0)

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

    def f(self):
        return self.moves + self.heuristic()

    def heuristic(self):
        return sum(abs(b % 3 - g % 3) + abs(b // 3 - g // 3)
                   for b, g in ((self.board.index(i), self.goal.index(i)) for i in range(1, 9)))

    def get_neighbors(self):
        neighbors = []
        x, y = divmod(self.zero, 3)
        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:
                nz = nx * 3 + ny
                new_board = list(self.board)
```

```
                new_board[self.zero], new_board[nz] = new_board[nz], new_board[self.zero]
                neighbors.append(PuzzleState(tuple(new_board), self.goal, self.moves + 1))
        return neighbors

def solve(start, goal):
    start_state = PuzzleState(start, goal)
    frontier = []
    heapq.heappush(frontier, start_state)
    visited = set()

    while frontier:
        state = heapq.heappop(frontier)
        if state.board == goal:
            return state.moves
        visited.add(state.board)
        for neighbor in state.get_neighbors():
            if neighbor.board not in visited:
                heapq.heappush(frontier, neighbor)
    return -1

start = (1, 2, 3, 4, 0, 5, 6, 7, 8)
goal = (1, 2, 3, 4, 5, 6, 7, 8, 0)
print("Moves to solve:", solve(start, goal))
```

Output: