**1st program- 8-puzzle**

import heapq

# Define the goal state

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Directions: up, down, left, right

moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Heuristic function: Manhattan distance

def manhattan\_distance(state):

distance = 0

for i in range(3):

for j in range(3):

val = state[i][j]

if val != 0:

goal\_x, goal\_y = divmod(val - 1, 3)

distance += abs(goal\_x - i) + abs(goal\_y - j)

return distance

# Convert list to tuple for hashing

def to\_tuple(state):

return tuple(tuple(row) for row in state)

# Find position of zero (blank tile)

def find\_zero(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

# Generate neighbors

def get\_neighbors(state):

neighbors = []

x, y = find\_zero(state)

for dx, dy in moves:

nx, ny = x + dx, y + dy

if 0 <= nx < 3 and 0 <= ny < 3:

new\_state = [row[:] for row in state]

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

neighbors.append(new\_state)

return neighbors

# A\* Search Algorithm

def solve\_puzzle(start\_state):

visited = set()

pq = []

heapq.heappush(pq, (manhattan\_distance(start\_state), 0, start\_state, []))

while pq:

est\_cost, cost\_so\_far, current\_state, path = heapq.heappop(pq)

if current\_state == goal\_state:

return path + [current\_state]

state\_key = to\_tuple(current\_state)

if state\_key in visited:

continue

visited.add(state\_key)

for neighbor in get\_neighbors(current\_state):

if to\_tuple(neighbor) not in visited:

heapq.heappush(pq, (

cost\_so\_far + 1 + manhattan\_distance(neighbor),

cost\_so\_far + 1,

neighbor,

path + [current\_state]

))

return None

# Example usage

start = [[1, 2, 3],

[4, 0, 6],

[7, 5, 8]]

solution\_path = solve\_puzzle(start)

if solution\_path:

print("Steps to solve the puzzle:")

for step in solution\_path:

for row in step:

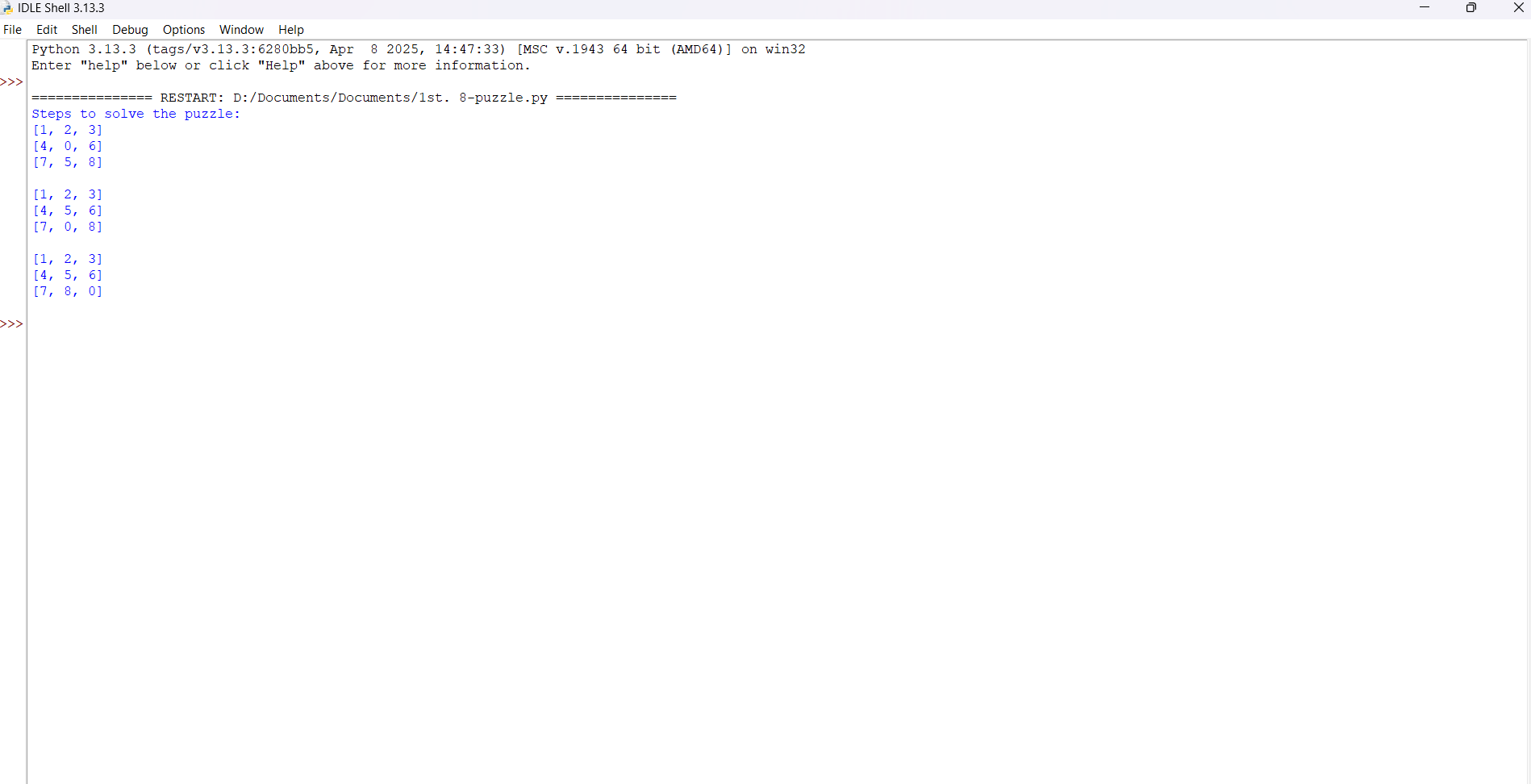
print(row)

print()

else:

print("No solution found.")

**Output:**

****