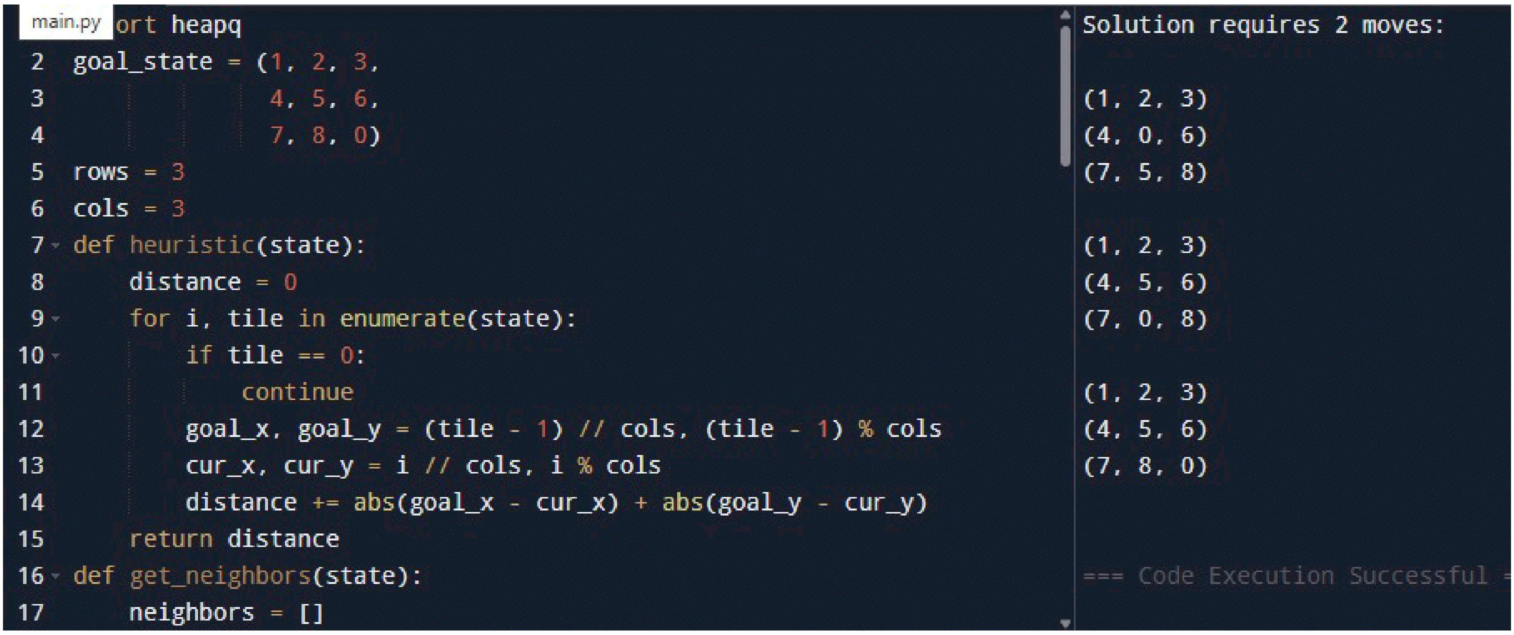
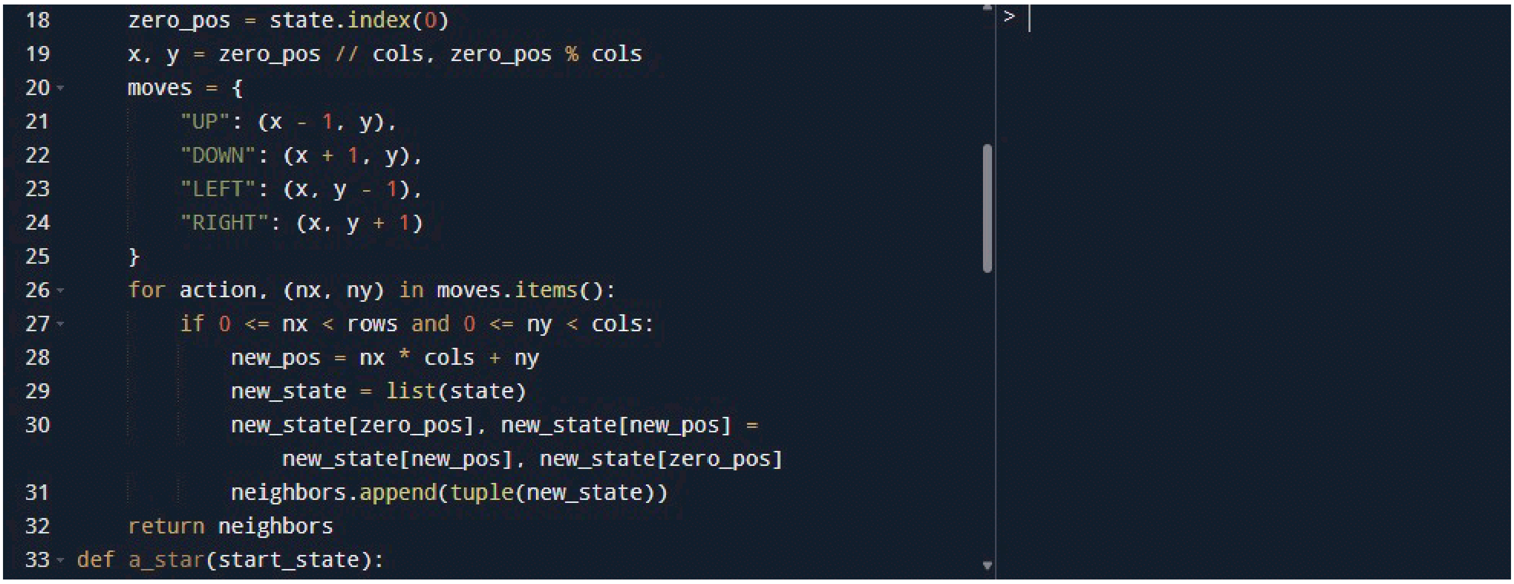


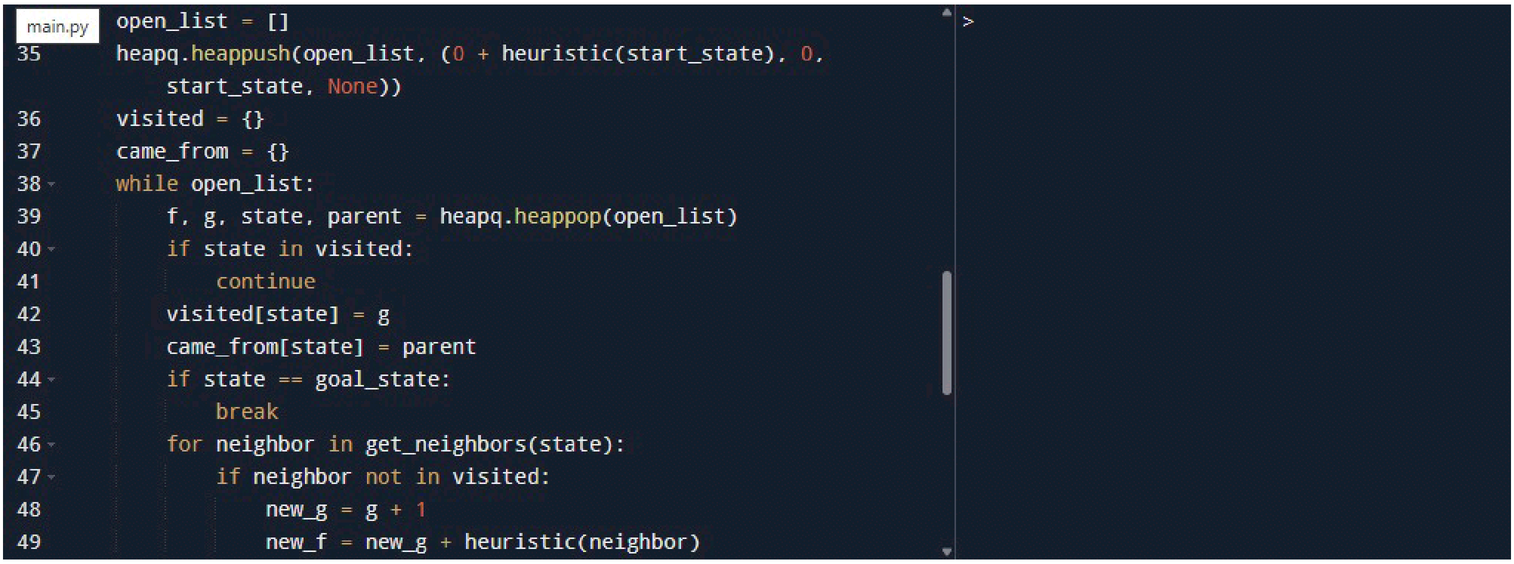
```python
ort heapq
goal_state = (1, 2, 3,
              4, 5, 6,
              7, 8, 0)
rows = 3
cols = 3
def heuristic(state):
    distance = 0
    for i, tile in enumerate(state):
        if tile == 0:
            continue
        goal_x, goal_y = (tile - 1) // cols, (tile - 1) % cols
        cur_x, cur_y = i // cols, i % cols
        distance += abs(goal_x - cur_x) + abs(goal_y - cur_y)
    return distance
def get_neighbors(state):
    neighbors = []
```

```
Solution requires 2 moves:

(1, 2, 3)
(4, 0, 6)
(7, 5, 8)

(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)


=== Code Execution Successful =
```

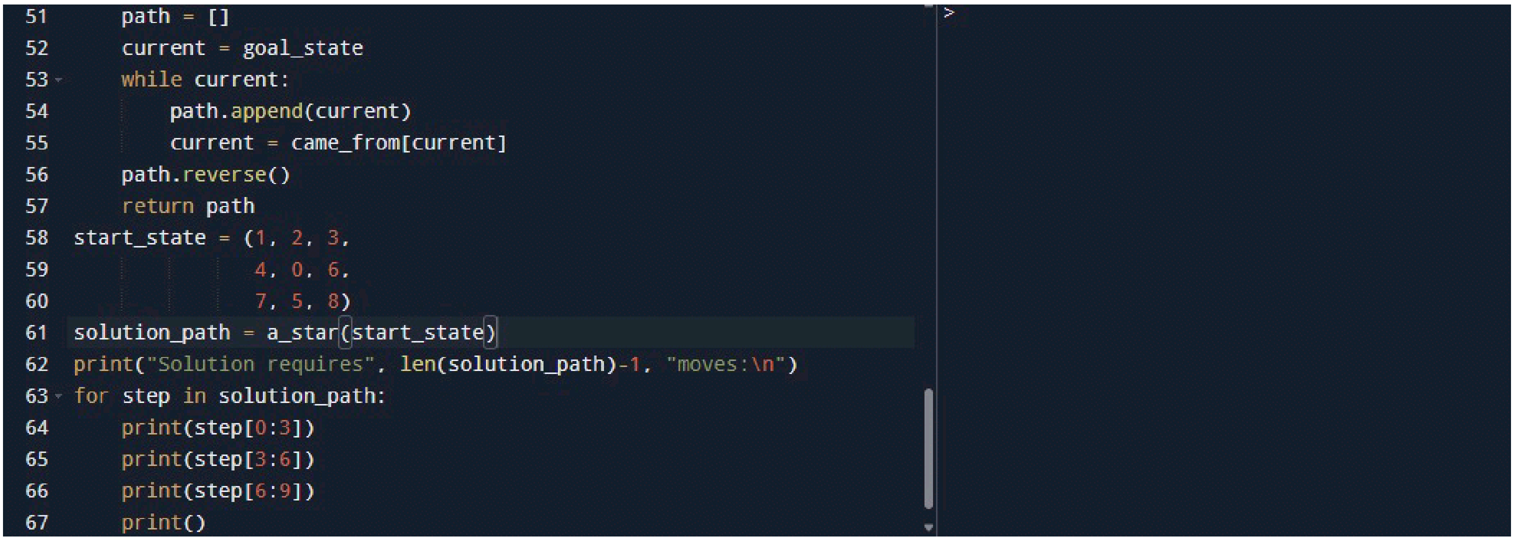


```
    zero_pos = state.index(0)
    x, y = zero_pos // cols, zero_pos % cols
    moves = {
        "UP": (x - 1, y),
        "DOWN": (x + 1, y),
        "LEFT": (x, y - 1),
        "RIGHT": (x, y + 1)
    }
    for action, (nx, ny) in moves.items():
        if 0 <= nx < rows and 0 <= ny < cols:
            new_pos = nx * cols + ny
            new_state = list(state)
            new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
            neighbors.append(tuple(new_state))
    return neighbors
def a_star(start_state):
```

```
>
```



```python
open_list = []
heapq.heappush(open_list, (0 + heuristic(start_state), 0,
    start_state, None))
visited = {}
came_from = {}
while open_list:
    f, g, state, parent = heapq.heappop(open_list)
    if state in visited:
        continue
    visited[state] = g
    came_from[state] = parent
    if state == goal_state:
        break
    for neighbor in get_neighbors(state):
        if neighbor not in visited:
            new_g = g + 1
            new_f = new_g + heuristic(neighbor)
```



```python
    path = []
    current = goal_state
    while current:
        path.append(current)
        current = came_from[current]
    path.reverse()
    return path
start_state = (1, 2, 3,
               4, 0, 6,
               7, 5, 8)
solution_path = a_star(start_state)
print("Solution requires", len(solution_path)-1, "moves:\n")
for step in solution_path:
    print(step[0:3])
    print(step[3:6])
    print(step[6:9])
    print()
```