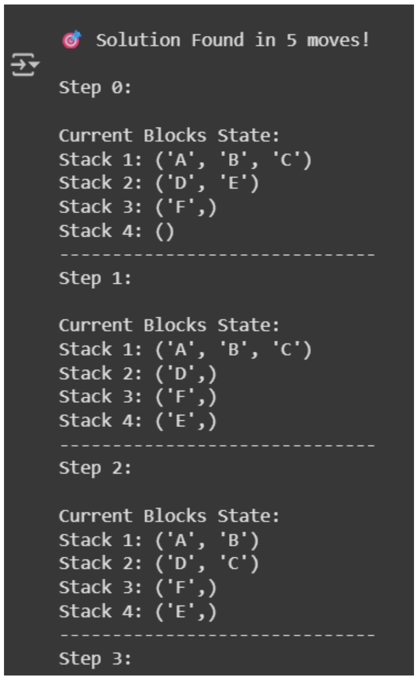
Practical: 4	**4) Write a program to implement Single Player Game (Using Heuristic Function)**
15/07/2025	

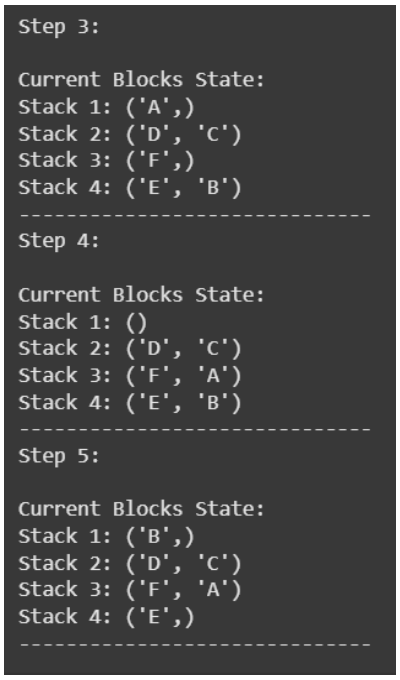
PROGRAM-

```
import heapq

def print_state(state):
   print("\nCurrent Blocks State:")
   for i, stack in enumerate(state):
      print(f"Stack {i + 1}: {stack}")
   print("-" * 30)

def is_goal(state, goal):
   return state == goal

def heuristic(state, goal):
   # Heuristic: number of misplaced blocks (stack-wise)
   misplaced = 0
   for s_stack, g_stack in zip(state, goal):
      for i in range(min(len(s_stack), len(g_stack))):
         if s_stack[i] != g_stack[i]:
            misplaced += 1
      # Count remaining unmatched blocks
      misplaced += abs(len(s_stack) - len(g_stack))
   return misplaced

def get_successors(state):
   successors = []
   for i, stack in enumerate(state):
      if not stack:
         continue
      block = stack[-1]
      for j in range(len(state)):
         if i == j:
                continue
            new_state = [list(s) for s in state]
            new_state[i].pop()
            new_state[j].append(block)
            successors.append(tuple(tuple(s) for s in new_state))
    return successors

def a_star_search(initial_state, goal_state):
    initial = tuple(tuple(s) for s in initial_state)
    goal = tuple(tuple(s) for s in goal_state)

    frontier = []
    heapq.heappush(frontier, (heuristic(initial, goal), 0, initial, []))
    visited = set()

    while frontier:
        est_total_cost, cost_so_far, current, path = heapq.heappop(frontier)

        if current in visited:
            continue
        visited.add(current)

        if current == goal:
            return path + [current]

        for successor in get_successors(current):
            if successor in visited:
                continue
            new_path = path + [current]
            g = cost_so_far + 1
            h = heuristic(successor, goal)
            heapq.heappush(frontier, (g + h, g, successor, new_path))

    return None
initial_state = [
    ['A', 'B', 'C'],
    ['D', 'E'],
    ['F'],
    []
]

goal_state = [
    ['B'],
    ['D', 'C'],
    ['F', 'A'],
    ['E']
]

solution_path = a_star_search(initial_state, goal_state)

if solution_path:
    print("\n🎯 Solution Found in", len(solution_path) - 1, "moves!\n")
    for i, state in enumerate(solution_path):
        print(f"Step {i}:")
        print_state(state)
else:
    print("No solution found.")
```

OUTPUT-



```
🎯 Solution Found in 5 moves!

Step 0:

Current Blocks State:
Stack 1: ('A', 'B', 'C')
Stack 2: ('D', 'E')
Stack 3: ('F',)
Stack 4: ()
------------------------------
Step 1:

Current Blocks State:
Stack 1: ('A', 'B', 'C')
Stack 2: ('D',)
Stack 3: ('F',)
Stack 4: ('E',)
------------------------------
Step 2:

Current Blocks State:
Stack 1: ('A', 'B')
Stack 2: ('D', 'C')
Stack 3: ('F',)
Stack 4: ('E',)
------------------------------
Step 3:
```



```
Step 3:

Current Blocks State:
Stack 1: ('A',)
Stack 2: ('D', 'C')
Stack 3: ('F',)
Stack 4: ('E', 'B')
------------------------------
Step 4:

Current Blocks State:
Stack 1: ()
Stack 2: ('D', 'C')
Stack 3: ('F', 'A')
Stack 4: ('E', 'B')
------------------------------
Step 5:

Current Blocks State:
Stack 1: ('B',)
Stack 2: ('D', 'C')
Stack 3: ('F', 'A')
Stack 4: ('E',)
------------------------------
```