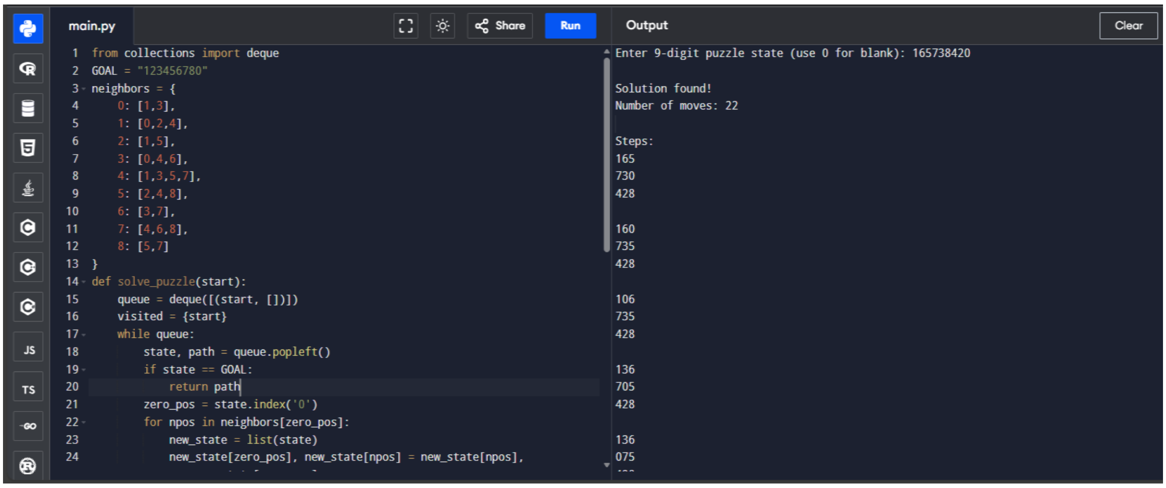
MLA0109 – Artificial Intelligence and Expert System

Python Programs – Day 1

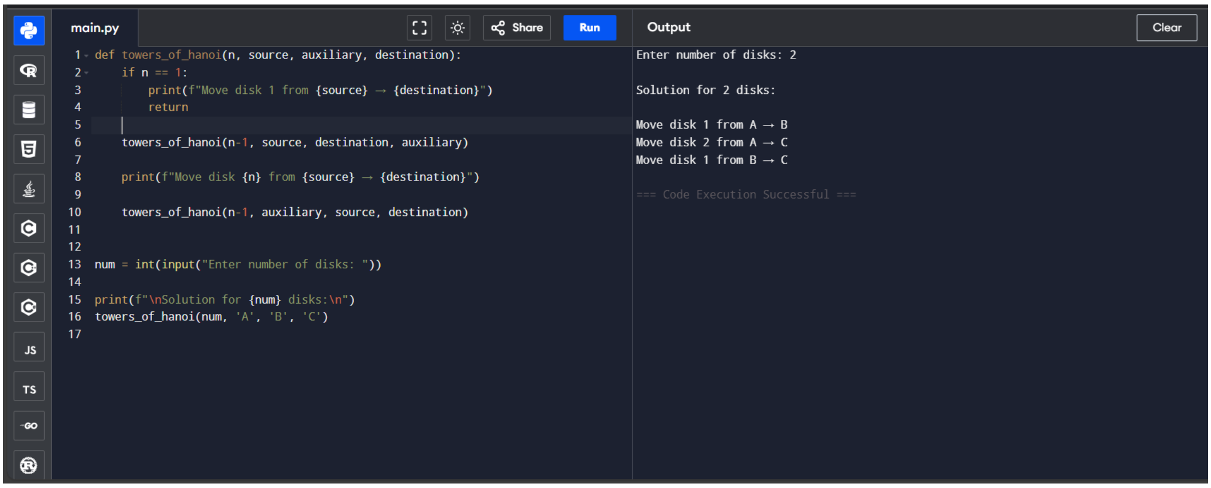
1. In sliding puzzle, it consists of a 3x3 grid with eight numbered tiles and one empty space. The goal of the puzzle is to rearrange the tiles from their initial, scrambled state to a goal state where the numbers are ordered from 1 to 8, with the empty space in the bottom-right corner. Implement the same using python.



```python
from collections import deque
GOAL = "123456780"
neighbors = {
    0: [1,3],
    1: [0,2,4],
    2: [1,5],
    3: [0,4,6],
    4: [1,3,5,7],
    5: [2,4,8],
    6: [3,7],
    7: [4,6,8],
    8: [5,7]
}
def solve_puzzle(start):
    queue = deque([(start, [])])
    visited = {start}
    while queue:
        state, path = queue.popleft()
        if state == GOAL:
            return path
        zero_pos = state.index('0')
        for npos in neighbors[zero_pos]:
            new_state = list(state)
            new_state[zero_pos], new_state[npos] = new_state[npos],
```

```
Enter 9-digit puzzle state (use 0 for blank): 165738420

Solution found!
Number of moves: 22

Steps:
165
730
428

160
735
428

106
735
428

136
705
428

136
075
```

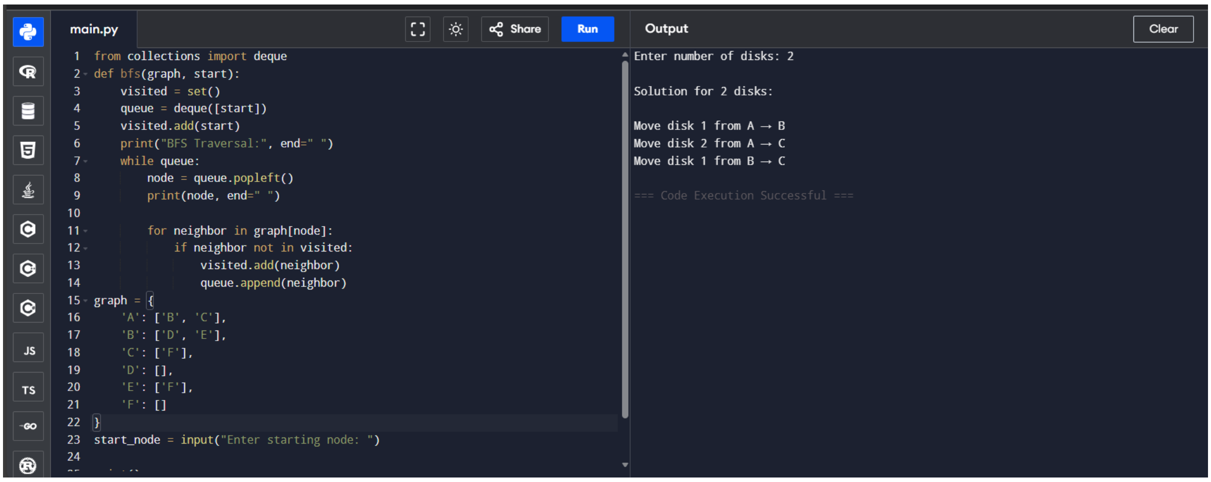
2. The Towers of Hanoi problem involves three pegs (A, B, C) and a number of disks of different sizes that can slide onto any peg. The puzzle starts with all the disks stacked on one peg in order of decreasing size, with the largest at the bottom. The objective of the problem is to move the entire stack to another peg, following these rules:
 a) Only one disk can be moved at a time.
 b) Each move consists of taking the upper disk from one of the stacks and placing it on top of another tack.
 c) No disk may be placed on top of a smaller disk.
 Implement the above Towers of Hanoi problem using Python.



```python
def towers_of_hanoi(n, source, auxiliary, destination):
    if n == 1:
        print(f"Move disk 1 from {source} → {destination}")
        return
    
    towers_of_hanoi(n-1, source, destination, auxiliary)

    print(f"Move disk {n} from {source} → {destination}")

    towers_of_hanoi(n-1, auxiliary, source, destination)


num = int(input("Enter number of disks: "))

print(f"\nSolution for {num} disks:\n")
towers_of_hanoi(num, 'A', 'B', 'C')

```

```
Enter number of disks: 2

Solution for 2 disks:

Move disk 1 from A → B
Move disk 2 from A → C
Move disk 1 from B → C

=== Code Execution Successful ===
```

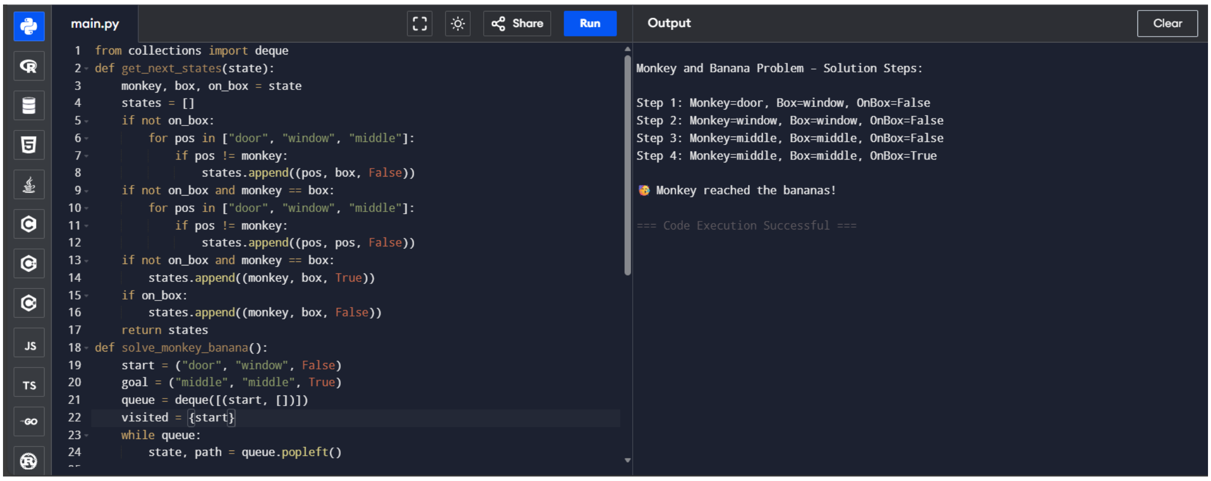
3. Implement a Python program to perform Breadth-First Search (BFS) on a graph. The program should:
   a) Represent the graph using an adjacency list.
   b) Start the traversal from a given source node.
   c) Print the order in which the nodes are visited.



```python
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)
    print("BFS Traversal:", end=" ")
    while queue:
        node = queue.popleft()
        print(node, end=" ")

        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
start_node = input("Enter starting node: ")

```

```
Enter number of disks: 2

Solution for 2 disks:

Move disk 1 from A → B
Move disk 2 from A → C
Move disk 1 from B → C

=== Code Execution Successful ===
```

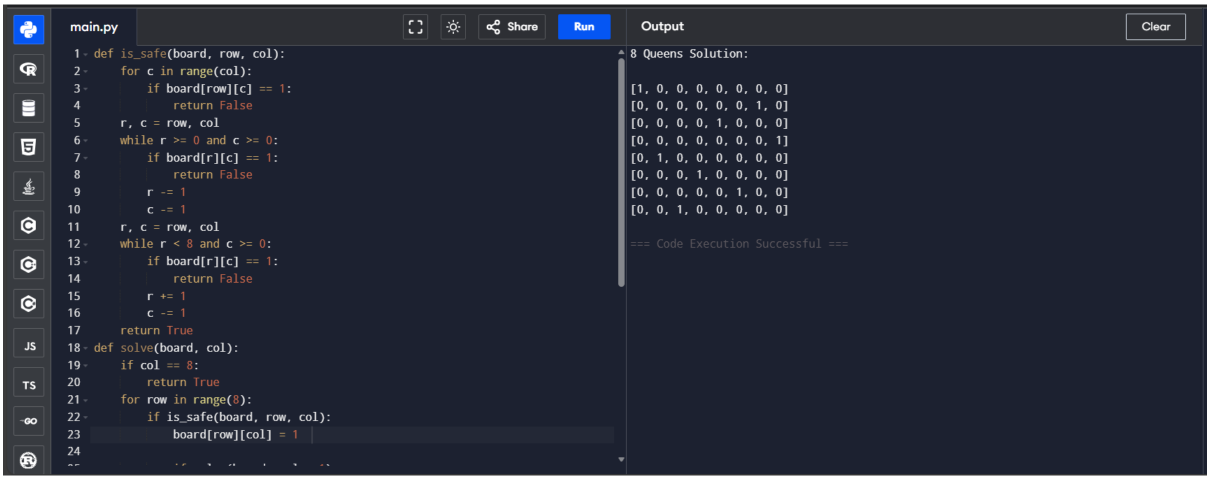
4. The Monkey and Banana problem is a classic artificial intelligence problem where a monkey needs to navigate through a room to reach a bunch of bananas hanging from the ceiling. The monkey has to move a box to stand on it to reach the bananas. Implement the above scenario using python.



```python
from collections import deque
def get_next_states(state):
    monkey, box, on_box = state
    states = []
    if not on_box:
        for pos in ["door", "window", "middle"]:
            if pos != monkey:
                states.append((pos, box, False))
    if not on_box and monkey == box:
        for pos in ["door", "window", "middle"]:
            if pos != monkey:
                states.append((pos, pos, False))
    if not on_box and monkey == box:
        states.append((monkey, box, True))
    if on_box:
        states.append((monkey, box, False))
    return states
def solve_monkey_banana():
    start = ("door", "window", False)
    goal = ("middle", "middle", True)
    queue = deque([(start, [])])
    visited = {start}
    while queue:
        state, path = queue.popleft()
```

```
Monkey and Banana Problem - Solution Steps:

Step 1: Monkey=door, Box=window, OnBox=False
Step 2: Monkey=window, Box=window, OnBox=False
Step 3: Monkey=middle, Box=middle, OnBox=False
Step 4: Monkey=middle, Box=middle, OnBox=True

🥳 Monkey reached the bananas!

=== Code Execution Successful ===
```

5. Write the python program to place eight queens on an 8 x 8 chessboard in such a way that no two queens threaten each other. In other words, no two queens can share the same row, column, or diagonal. Use backtracking to explore different possibilities and ensures that no two queens threaten each other.



```python
def is_safe(board, row, col):
    for c in range(col):
        if board[row][c] == 1:
            return False
    r, c = row, col
    while r >= 0 and c >= 0:
        if board[r][c] == 1:
            return False
        r -= 1
        c -= 1
    r, c = row, col
    while r < 8 and c >= 0:
        if board[r][c] == 1:
            return False
        r += 1
        c -= 1
    return True
def solve(board, col):
    if col == 8:
        return True
    for row in range(8):
        if is_safe(board, row, col):
            board[row][col] = 1

```

```
8 Queens Solution:

[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]

=== Code Execution Successful ===
```