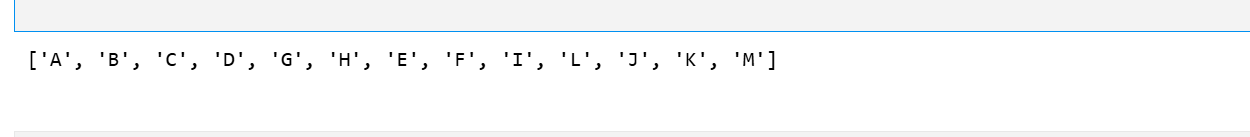
**Task # 6 Description**

**Question # 1:**

This program performs a **Breadth-First Search (BFS)** using a **queue** to explore all nodes level by level

1. **Function bfs\_queue(graph, start)**
   * **visit** → a set to store visited nodes (so no node is visited twice).
   * **queue** → starts with the start node (used for level-by-level traversal).
   * **result** → list to record the visiting order of nodes.
2. **BFS Logic (Loop)**
   * While the queue isn’t empty:
     + Remove the **first element** (pop(0) → FIFO order).
     + If the node isn’t visited:
       - Add it to result and mark it visited.
       - Add all its connected neighbors to the **end of the queue** (so they’re visited next).
3. **Graph Example**
   * 'A' connects to 'B', 'C'
   * 'B' → 'D', 'G', 'H'
   * 'C' → 'E', 'F'
   * 'F' → 'J', 'K', etc.
4. **Traversal Order**
   * BFS explores **breadth-wise (level by level)**, it visits all immediate neighbors of a node before moving deeper.
   * Starting from 'A', the visiting order becomes something like:  
     ['A', 'B', 'C', 'D', 'G', 'H', 'E', 'F', 'I', 'L', 'J', 'K', 'M']

**Output Screen Shots:**

****

**Question # 2:**

This program performs **Breadth-First Search (BFS)** but in a **recursive way**, meaning it doesn’t use a queue explicitly, yet still explores nodes **level by level**

1. **Function bfs\_recursive(graph, current\_level, seen, order)**
   * **current\_level** → list of nodes currently being visited (same as a BFS queue’s “front”).
   * **seen** → keeps track of visited nodes to avoid repetition.
   * **order** → stores the traversal order.
   * If current\_level is empty → recursion stops (base case).
   * For every node in the current level:
     + If not already seen, add to order and mark as visited.
     + Collect all its connected neighbors in next\_level.
   * Then recursively call bfs\_recursive() on next\_level to process the next layer of the graph.
2. **Wrapper Function bfs(graph, start)**
   * Initializes empty seen and order sets/lists.
   * Starts recursion with the starting node in a list: [start].
   * Returns the full traversal order once recursion ends.
3. **Graph Example**
   * 'A' → 'B', 'C'
   * 'B' → 'D', 'G', 'H'
   * 'C' → 'E', 'F', and so on.
4. **Traversal Order**
   * BFS (level-wise) output:  
     ['A', 'B', 'C', 'D', 'G', 'H', 'E', 'F', 'I', 'L', 'J', 'K', 'M']

**Output Screen Shots:**

