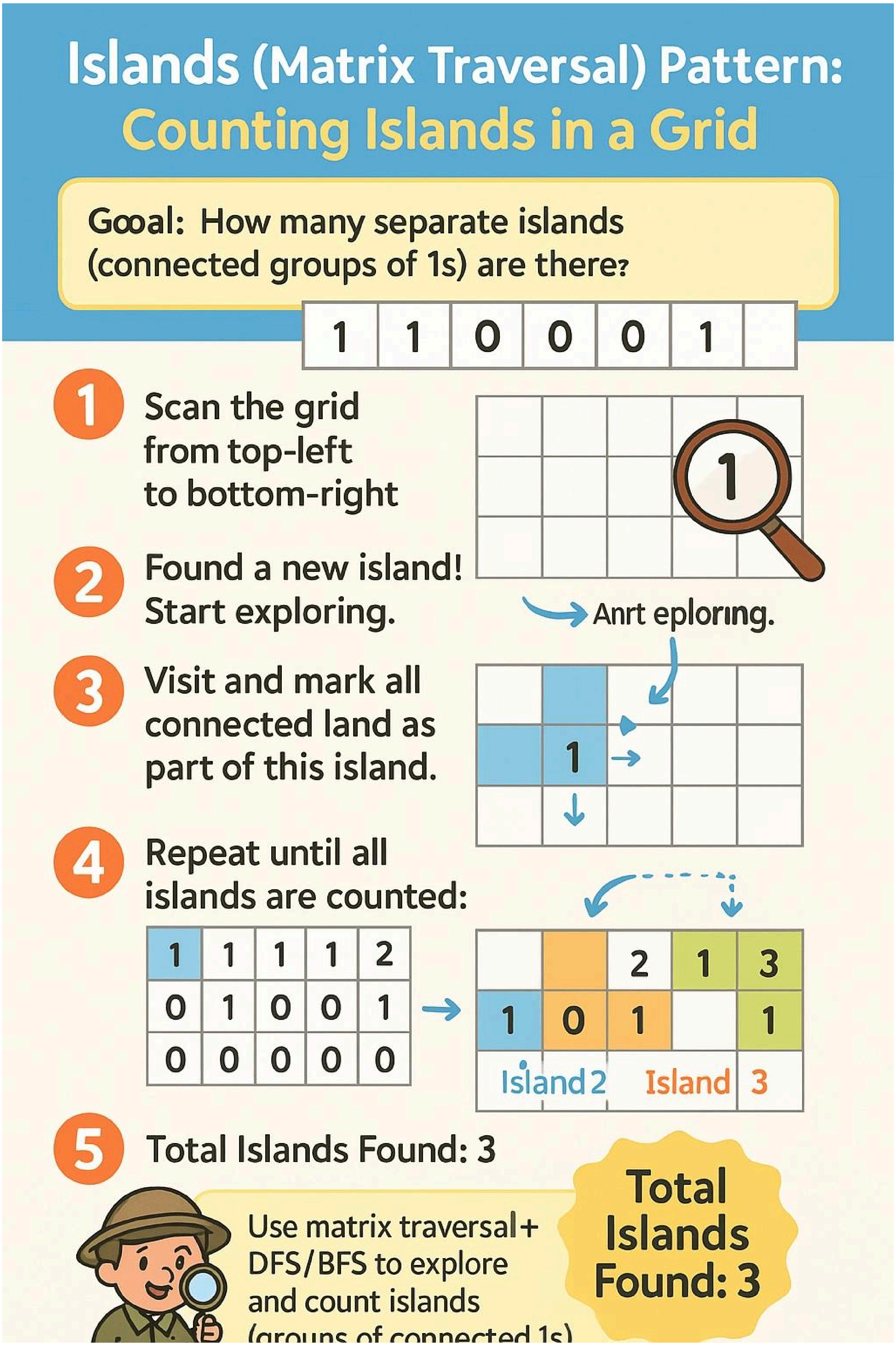
Islands (Matrix Traversal) Pattern

The **Islands Pattern** (also called Matrix Traversal or Flood Fill) involves efficiently exploring or traversing a matrix (2D array/grid) to find or process groups of connected elements, such as "islands" of 1s in a sea of 0s.

- **Classic Use:** Count the number of "islands" (groups of connected 1s) in a binary matrix.
- **Other Uses:** Coloring/filling an area (flood fill), searching paths, finding cycles, etc.



Islands (Matrix Traversal) Pattern:

Counting Islands in a Grid

Gooal: How many separate islands (connected groups of 1s) are there?

1	1	0	0	0	1	

1 Scan the grid from top-left to bottom-right

2 Found a new island! Start exploring.

3 Visit and mark all connected land as part of this island.

4 Repeat until all islands are counted:

1	1	1	1	2
0	1	0	0	1
0	0	0	0	0

5 Total Islands Found: 3

Use matrix traversal+ DFS/BFS to explore and count islands (groups of connected 1s)

Total Islands Found: 3

Classic Problem: Number of Islands

Problem Statement:

Given a matrix with 0s (water) and 1s (land), count how many islands are there.

An island is a group of adjacent 1s connected **up, down, left, or right** (not diagonally).

Step-by-step Approach

1. **Scan the matrix.**
   When you find a 1 (land), you've found a new island.
2. **Explore (traverse) all connected 1s** using DFS (Depth First Find) or BFS (Breadth First Find) to mark them as visited (e.g., change to 0 or use a visited array).
3. **Increase the island count.**
   Continue until you've visited the whole matrix.

C Code Example (DFS Approach)

```
#include <stdio.h>

#define ROWS 4
#define COLS 5

// Directions: Up, Down, Left, Right
int dirRow[] = {-1, 1, 0, 0};
int dirCol[] = {0, 0, -1, 1};

// Helper: Check if a cell is safe to visit
int isSafe(int matrix[ROWS][COLS], int row, int col) {
    return (row >= 0 && row < ROWS && col >= 0 && col < COLS && matrix[row][col] == 1);
}

// DFS to mark connected land as visited
void dfs(int matrix[ROWS][COLS], int row, int col) {
    matrix[row][col] = 0; // Mark as visited (sink the land)

    // Explore all 4 directions
    for (int i = 0; i < 4; i++) {
        int newRow = row + dirRow[i];
        int newCol = col + dirCol[i];
        if (isSafe(matrix, newRow, newCol)) {
            dfs(matrix, newRow, newCol);
        }
    }
}

// Main function to count islands
int countIslands(int matrix[ROWS][COLS]) {
    int count = 0;

    for (int row = 0; row < ROWS; row++) {
        for (int col = 0; col < COLS; col++) {
            if (matrix[row][col] == 1) {
                count++;
                dfs(matrix, row, col); // Visit all land in this island
            }
        }
    }
    return count;
}

int main() {
    int matrix[ROWS][COLS] = {
        {1, 1, 0, 0, 0},
        {0, 1, 0, 0, 1},
        {1, 0, 0, 1, 1},
        {0, 0, 0, 0, 0}
    };

    int result = countIslands(matrix);
    printf("Number of islands: %d\n", result);

    return 0;
}
```

Explanation

1. Scan the Grid

- Go through every cell in the matrix (row by row, column by column).

2. Find a New Island

- When you hit a `1`, it's part of an island you haven't visited yet.
- Increase the island count.

3. Mark the Whole Island

- Use DFS (recursive function) to visit all connected `1`s from the starting cell (up, down, left, right).
- As you visit each cell, mark it as `0` (water) so you don’t count it again.

4. Keep Going

- Repeat until all cells are checked.

5. Output

- The island count is your answer!

How the Example Works

For the given matrix:

```
1 1 0 0 0
0 1 0 0 1
1 0 0 1 1
0 0 0 0 0
```

There are **3 islands** (connected groups of 1s):

1. Top-left (3 connected 1s)
2. Top-right and middle-right (connected)
3. Bottom-left (single 1)

When to Use This Pattern

- Any problem where you need to process or count groups/regions in a 2D grid (e.g., minesweeper, flood fill, regions in an image, maze traversal, etc.).
- Whenever you need to visit each cell in a ”region” only once.

Practice Challenge

- **Modify the code to count the size (number of cells) of the largest island.**
- **Try the BFS approach (using a queue) instead of DFS (recursion).**

Key Takeaways

- Traverse all cells; use DFS/BFS to mark connected groups as visited.
- Use for "counting groups/regions/connected components" in grids.