**Lab 04(N\_Queen)**

****

**Name Syed M Haider Jafir**

**Roll no SU92-BSAIM-S24-004**

**Section BSAI-4A**

**Subject PF (AILab)**

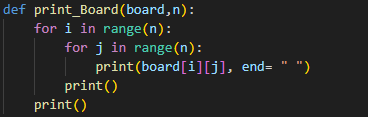
**1. Quick overview**  
It uses backtracking: try a column, recurse to the next row, and if that path fails, remove the queen and try the next column. The program prints the **first** valid arrangement it finds (or a message if none exists).

**2. Board representation**

****

* board is an n x n list of lists.
* "." means empty, "Q" means a queen.
* Rows and columns are zero-indexed: top-left is (row=0, col=0).

**3. Function-by-function explanation**

****

* Loops each row i and prints each column j on the same line (end=" " keeps printing on one line).
* print() after the inner loop moves to the next row.
* Final print() adds a blank line after the entire board.

**is\_safe(board, row, col, n)**

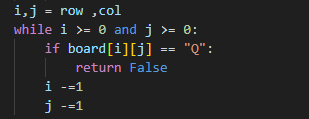
This checks whether placing a queen at (row, col) would be attacked by any previously placed queen.

1. **Column check**



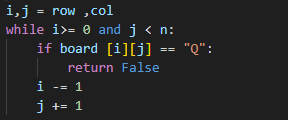
* Only checks rows 0..row-1 because queens are placed one-per-row from top to bottom. There are no queens yet on lower rows.

2.**Upper-left diagonal**

****

* Moves up-left (decreasing both i and j) and returns False if it finds a Q.

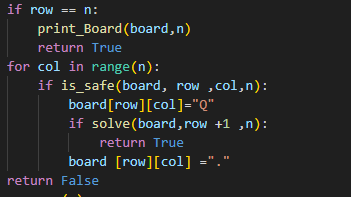
**3.Upper-right diagonal**

****

* Moves up-right (decrease i, increase j). The loop condition j < n prevents going past the right edge — this is the correct boundary check that avoids IndexError.

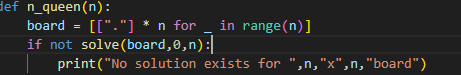
4.If none found: return True.

**Why only check previous rows and these diagonals?**  
Because we place queens row-by-row. When we are trying to place a queen in row, there are no queens yet in rows row..n-1, so only rows 0..row-1 can contain conflicts.



* **Base case:** if row == n → all n queens placed → print board and return True.
* **Try each column** in the current row:
  + If is\_safe, place a queen and recurse to next row.
  + If recursive call returns True, propagate True to stop searching (this is how you ensure **only one** solution is printed).
  + If recursion fails, backtrack by removing the queen (".") and continue.
* If no column leads to a solution, return False.

**n\_queen(n)**



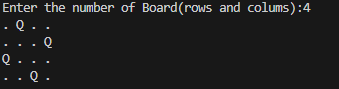
* Prepares the board and starts recursion at row 0.
* If solve returns False, prints a message (note: the print uses commas so it will add spaces; you could use an f-string for a cleaner message).

**Input and run**

****

**4. Example output**

**For n = 4, your program prints one valid solution, for example:**

****