Exercise 4

Explanation:

The code implements a solution to determine if a knight on a chessboard of size R x C can visit all squares exactly once, starting from a given position (posx, posy). It uses backtracking to explore all possible knight moves, defined by the moves list, and recursively attempts to visit all squares. The is_valid_move function ensures that moves stay within the board and avoid revisiting squares. The move function marks squares as visited with a move count, recursively explores valid moves, and backtracks if a path fails. If all squares are visited (move_count == R * C), the function returns True along with the board showing the knight's path. If no solution exists, it returns False.

Code:

```
def max_knight_moves(R, C, posy, posx):
    # Create a board to keep track of visited squares
    board = [[0 for _ in range(C)] for _ in range(R)]

    # Possible moves of a knight
    moves = [(2, 1), (2, -1), (-2, 1), (-2, -1), (1, 2), (1, -2), (-1, 2), (-1, -2)]

    # Function to check if the move is valid
    def is_valid_move(x, y):
        return 0 <= x < R and 0 <= y < C and board[x][y] == 0

    # Function to perform backtracking
    def move(x, y, move_count):
        if move_count == R * C: # If all squares are visited
            return True

        for dx, dy in moves: # Try all possible moves
            new_x = x + dx
            new_y = y + dy

            if is_valid_move(new_x, new_y): # If the move is valid
                board[new_x][new_y] = move_count + 1 # Mark the square as visited with the move count
                if move(new_x, new_y, move_count + 1): # Recursion to continue the path
                    return True
                board[new_x][new_y] = 0 # Backtrack: unmark the square, as smth went wrong

        return False # If no valid moves are found, return False

    board[posx][posy] = 1 # Mark the starting position as visited
    if move(posx, posy, 1): # Start backtracking from the starting position
        return True, board # Return True and the board with the path
    else:
        return False, board
```

```
# Example usage
R = 7
C = 3
posx = 0 #the board is 0-indexed
posy = 2

""" Example indexes board:
(0,0),(0,1),(0,2)
(1,0),(1,1),(1,2)
(2,0),(2,1),(2,2)
(3,0),(3,1),(3,2)
(4,0),(4,1),(4,2)
(5,0),(5,1),(5,2)
(6,0),(6,1),(6,2)
"""

print("Is it possible to visit all squares?")

result, board = max_knight_moves(R, C, posy, posx)

if result:
    print("Yes, it is possible.")

    print("Path:")
    # Print the board with the path in a nice format
    for row in board:
        print(" ".join(f"{cell:2}" for cell in row)) # Print each cell with a width of 2

else:
    print("No, it is not possible.")
```

Test case:

R = 7
C = 3
posx = 0 #the board is 0-indexed
posy = 2

Is it possible to visit all squares?
Yes, it is possible.
Path:
3 6 1
8 21 4
5 2 7
20 9 18
17 12 15
14 19 10
11 16 13

Exercise 6

Explanation:

Code:

Test case: