CSCI 111, Lab 6
Connect Four!

Due date: Midnight, Tuesday, October 25, on Canvas. No late work accepted.

File names: Names of files, functions, and variables, when specified, must be EXACTLY as specified. This includes simple mistakes such as capitalization.

Individual work: All work must be your own. Do not share code with anyone other than the instructor and teaching assistants. This includes looking over shoulders at screens with the code open. You may discuss ideas, algorithms, approaches, *etc.* with other students but NEVER actual code.

Connect Four: I'm going to assume no one needs an explanation of the game. It is a simple variation on tic-tac-toe played on a 6 row by 7 column board, you need to get four in a row to win, and each play occupies the lowest unoccupied square in its column.

Turn in: Write a single module, `connect4.py`, for this assignment. Zip it in a folder titled `lab06`.

GUI Implementation: You will make this with a graphical interface in Tkinter. Everything you need to get started is provided with the `tictactoe.py` module in this lab's folder. You should follow my strategy exactly for representing places on the board with buttons, states with dictionaries, *etc.*, but generalize and modify it as follows.

Board size: Before the game starts, ask the user for the number of rows and columns. Default values should be 6 rows and 7 columns. You can use two `simpledialog`'s for this, as in my tic tac toe game, or make a fancier dialog with tkinter.

Gravity: The user can click on any unoccupied square, but the mark will appear in the lowest empty square.

You do not have to animate the falling mark! That's for graphics or games classes.

Check for winner: In my tic tac toe game I use a brute force algorithm to see if there's a winner. I check all possible winning moves to see if one player has won. With connect four, on a reasonable size board, this would be really slow. (How many possible wins would you have to check?) Instead, if we assume that we check for a win after each move, we need check only four possible wins:

- After a move, the only possible win is by the player that just moved.
- After a move, the only possible win must involve the position just moved.
- There are only four possible directions a win can go: vertical, horizontal, northwest-southeast, and southwest-northeast.

Check winner algorithm: Follow the following procedure. You will have to solve several programming problems to translate this informal description into code.

1. For each of the possible winning directions, start in the topmost, leftmost, northwestmost, or southwestmost position on the board that is aligned with the last move position.

   Do *not* try to start just 3 squares away from the last move. Start at the very edge of the board. You will have to figure out how to find this square.
2. Step along the winning direction one square at a time. If you started in the right spot, you should hit the square for the most recent move along the way.
3. When you hit any square with the same color of the last move, start counting.
4. Keep counting until you hit a square not the last mover's color.
5. If you counted to 4 or beyond, winner!
6. Otherwise, keep stepping along the direction looking for streaks of 4 or more until you're off the other side of the board.
7. Checking for a cat's game (no winner) is really easy if you keep track of the number of moves so far.

You may recognize the search pattern involved here from previous labs!

Notes:

- You will probably need a reverse dictionary (mapping buttons to positions), not implemented in my tic tac toe game.
- You may want to make the number of rows and columns global variables.