```
OXO
XXO
OOX
```

```
OOO
OXX
XXO
```

```
XXX
OOX
XOO
```

```
XXO
XOO
OXX
```

```
X0X
XX0
00X
```

```
OOX
XOX
OXX
```

```python
"""
functions related to creating, printing,
and evaluating tic-tac-toe boards

:author: Ian Sulley
:note: I affirm that I have carried out the attached academic endeavors with full academic honesty,
in accordance with the Union College Honor Code and the course syllabus
"""


def remove_blank_lines(list_of_strings):
    """
    Given a list of strings, return a copy
    with all empty strings removed
    :param list_of_strings: list of strings, some of which may be ''; this list is unchanged
    :return: list identical to list_of_strings, but all empty strings removed
    """
    result = list()
    for s in list_of_strings:
        if s != '':
            result.append(s)
    return result


def get_board_from_file(filename):
    """
    Reads board, returns a list of rows.
    :param filename: text file with a tic-tac-toe board such as
    X X X
    O X O
    X O O
    where each line is one row
    :return: list of strings where each string is a
    row from filename; any blank lines in the file are removed
    Example: ["X X X", "O X O", "X O O"]
    """
    board_list = []
    board_file = open(filename, "r")
    for line in board_file:
        board_list.append(line.strip())
    board_file.close()
    board_list = remove_blank_lines(board_list)
    return board_list


def print_row(row):
    """
    Nicely prints a row of the board.
    :param row: string of Xs and Os
    """
    nice_row = ''
    for i in range(0, len(row)):
        nice_row += row[i]
        if i != len(row) - 1:
            nice_row += ' | '
    print(nice_row)


def print_board(board):
    """
    prints the tic-tac-toe board
    :param board: list of rows
    """
    for i in range(0, len(board)):
        row = board[i]
        print_row(row)
        if i != len(board) - 1:
            print('----------')


def three_in_row(board, player, start_x, start_y, dx, dy):
    """
    Determines if a player has three in a row, starting
    from a starting position (start_x, start_y) and going
    in the direction indicated by (dx, dy). Example:
    (start_x, start_y) = (2,2) means we start at the lower
    right (row 2, col 2). (dx, dy) = (-1, 0) means the next
    square we check is (2+dx, 2+dy) = (1,2).  And the last
    square we check is (1+dx, 2+dy) = (0,2).  So we've just
    checked the rightmost column - (2,2), (1,2), and (0,2).
    :param board: list of rows
    :param player: string -- either "X" or "O"
    :param start_x: row to start checking at; first row is
```

```
 row 0
    :param start_y: col to start checking at; first col is col 0
    :param dx: 1 if checking downward, -1 if checking upward, 0 if checking this row
    :param dy: 1 if checking rightward, -1 if checking leftward, 0 if checking this col
    """
    x = start_x
    y = start_y
    for i in range(0, 3):
        if board[x][y] != player:
            return False
        x += dx
        y += dy
    return True


def is_winner(board, player):
    """
    Returns True if and only if the given player has won.
    :param board: list of row strings
    :param player: string - "X" or "O"
    :return: True if player won; False if player lost or tied
    """
    if (three_in_row(board, player, 0, 0, 1, 1)
            or three_in_row(board, player, 0, 2, 1, -1)):
        return True
    else:
        for i in range(0, 3):
            if (three_in_row(board, player, 0, i, 1, 0)
                    or three_in_row(board, player, i, 0, 0, 1)):
                return True
        return False


def get_winner(board):
    """
    Returns the name of the winner, or None if there is no winner
    :param board: list of row strings
    :return: "X" if X is winner, "O" if O is winner, None if tie
    """
    if is_winner(board, 'X'):
        return 'X'
    elif is_winner(board, 'O'):
        return 'O'
    else:
        return None

def confirm_result(board, expected_winner):
    """
    Checks that the computed result matches the expected result.
    :param board:list of row strings
    :param expected_winner: Correct winner that should occur
    :return: "PASS" if computed matches expected result and "FAIL" and the correct winner, if the result does nat match.
    """

    if (get_winner(board) == expected_winner
        or get_winner(board) == None):
        print("PASS")
    else:
        print("FAIL")
        print("Should have returned " + expected_winner + " wins")

def test_all(board_files):
    """
    Iterates through all boards and computes their solutions.
    Calls print_board(), get_winner() and confirm_result for each.
    :param board_files: list of txt files or lists of lists constaining tic tak toe board
    :return: calls confirm_result to state if it is a PASS or Fail
    """

    if isinstance(board_files[0][0], str):
        i = 0
        for file in board_files:

            board = get_board_from_file(file[0])
            """
            I commented this section out for if you want
            to print out the boards and calculated results.
            By default it only tells you if you PASS or FAIL confirm_result()


            print_board(board)
            winner = get_winner(board)
            print("Result: %s wins" % (str(winner)))
            """
            confirm_result(board, board_files[i][1])
            i += 1

    else:
        i = 0
        for board in board_files:

            """
            I commented this section out for if you want to print out the boards and calculated results.
            By default it only tells you if you PASS or FAIL confirm_result()
            print_board(board[0])
            winner = get_winner(board[0])
            print("Result: %s wins" % (str(winner)))
            """
            confirm_result(board[0], board_files[i][1])
            i += 1

def main():
    """
    contains list of tuples which each contain
    (board_file, expected result)
    :return: calls test_all(board_files)
    """

    board_files = [
        ("X_wins.txt" , "X"),
        ("X_wins2.txt" , "X"),
        ("X_wins3.txt" , "X"),
        ("O_wins.txt" , "O"),
        ("O_wins2.txt" , "O"),
        ("Tie1.txt" , None)
    ]

    test_all(board_files)

```

```python
def main2():
    """
    constains list of tuples with each tuple containing
    (hardcoded_board, expected result)
    :return: calls test_all(hardcoded_boards)
    """

    Xwins_board = [
        "XXX",
        "OOX",
        "XXO"
    ]

    Xwins2_board = [
        "XOX",
        "XXO",
        "OOX"
    ]

    Xwins3_board = [
        "OOX",
        "XOX",
        "OXX"
    ]

    Owins_board = [
        "OOO",
        "OXX",
        "XXO"
    ]

    Owins2_board = [
        "XXO",
        "XOO",
        "OXX"
    ]

    Tie_board = [
        "OXO",
        "XXO",
        "OOX"
    ]

    hardcoded_boards = [
        (Xwins_board , "X"),
        (Xwins2_board, "X"),
            (Xwins3_board , "X"),
            (Owins_board , "O"),
            (Owins2_board , "O"),
            (Tie_board , None)
        ]

        test_all(hardcoded_boards)

    if __name__ == "__main__":
        main()

```