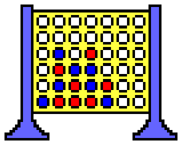
Problem 2, part 1: Connect 4 Board [25 points; individual or pair]

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/Connect4BoardBlack on 3/13/17

Overview: You'll write two classes for Problem 2:
- a `Board` class (this part of the problem) and
- a `Player` class (the next part of the problem)

You'll place both of these classes into a file named `hw11pr2.py` file and then submit it in the usual way to the NewSubmissionSite2014
The `Player` class is described in the second part of this problem.

<strong>Note:</strong> The CS 5 "Black" version of this problem is not identical to the CS 5 "Gold" version. Please do the version for the section in which you are enrolled!
Connect Four is a variation of tic-tac-toe played on a rectangular board. Typically there are 6 rows and 7 columns, although your code will work for any number of rows and columns.



The game is played by two players, alternating turns, with each trying to place four checkers in a row vertically, horizontally, or diagonally. Because the board stands vertically and the checkers are subject to gravity, a checker may only be placed at the top of one of the currently existing columns (or start a new column).

The `Board` class—a preview

In this problem, you will need to create a class named `Board` that implements some of the features of the Connect Four game. The `Board` class will have three data members: a two-dimensonal list (a list of lists) containing characters to represent the game area, and a pair of variables holding the number of rows and columns on the board (6 rows and 7 columns is standard, but your `Board` datatype will be able to handle boards of any size). The details of the `Board` class appear below.

The `Board` class

You will probably want to store the representation of the board as a two-dimensional list/array of *characters*. You should represent an empty slot by ' ', the space character. You should represent player X's checkers with an 'X' (the capital x character) and player O's checkers with an 'O' (the capital o character).

Methods required for the `Board` class

- `__init__(self, width=7, height=6)`: This is a constructor for `Board` objects that (in addition to `self`) takes two **named** arguments, one for the number of rows and one for the number of columns. It uses the default number of columns and rows (7 and 6, respectively) in the event that the user does not specify those arguments. Inside the constructor, you should set the values of the data members of the object, including initializinng the two-dimensional array of characters to contain all ' 's (space characters). **Note:** It is tempting to initialize the board using the multiplication operator. For example, a 2 by 3 array of 0's could be constructed this way: `[[' '] * 3] * 2`. Unfortunately, this looks nice but doesn't work because Python actually creates multiple copies of the *same* row this way. Thus, changing an element in one row will change the corresponding entries in all of the rows! Instead, use a strategy analogous to the way we constructed a blank board in the game of Life.

- `__repr__(self)`: This method **returns** a string (it does not `print` a string) representing the `Board` object that calls it. Basically, each "checker" takes up one space, and all columns are separated by vertical bars (|). The columns are labeled at the bottom. *Note that on some computers, the default font is a "variable width" font that makes the "X" and "O" symbols wider than the space symbol. This will make your board look messy. To fix this, choose a "Courier" or "Courier New" font in your preferences (usually found in the "Edit" menu).*

Here is an example of how your board should look:

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |O| |O| | |
| |X|X|X|O| | |
---------------
 0 1 2 3 4 5 6
```

See the sample run below for more examples of what a board should look like as a game is played. Remember that `__repr__` returns a string but doesn't actually print anything! The symbol `\n` can be placed in a string to cause a newline (return to beginning of next line). Here's an example:

```
In [1]: foo = "I\nlike\nspam"

In [2]: foo
Out[2]: 'I\nlike\nspam'

In [3]: print foo
I
like
spam
```

- `allowsMove(self, col)`: This method should return `True` if the calling `Board` object can allow a move into column `c` (because there is space available). It returns `False` if `c` does not have space available **or if it is not a valid column**. Thus, this method should check to be sure that `c` is within the range from 0 to the last column *and* make sure that there is still room left in the column!
- `isFull(self)`: This method should return `True` if the calling `Board` object has no more moves left available at all. Otherwise, it should return `False`. Note that you can use `allowsMove` as a helper to this one!
- `addMove(self, col, ox)`: This method should add an `ox` checker, where `ox` is a variable holding a string that is either `"X"` or `"O"`, into column `col`. Note that the code will have to find the highest row number available in the column `col` and put the checker in that row. The highest row number available is the highest index with a space character ' ' in the column `c`. Notice that the *highest row number* corresponds to the *lowest* physical row on the board.

- setBoard(self, move_string) : This method helps you (and us!) to test your Connect-Four Board class. Code is provided below, if you'd like to use it—or adapt it to suit your representation of the game. But be sure to include a method that has this functionality in your class!
- delMove(self,col) : This method should do the "opposite" of addMove. That is, it should remove the top checker from the column col. If the column is empty, then delMove should do nothing. This function may not seem crucial right away, but it is *very* useful in the next problem in which you implement your own Connect Four AI. It's also useful if you implement "undo."
- winsFor(self, ox) : This method should return True if the given checker, 'X' or 'O', held in ox, has won the calling Board. It should return False othwerwise. **Important Note:** you need to check if the player has won horizontally, vertically, or diagonally (and there are two different directions for a diagonal win).
- hostGame(self) This is a method that, when called from a Connect Four board object, will run a loop allowing the user(s) to play a game. See below for an example user interface.

Here is our code for setBoard—please use this or something equivalent that works with your class:

```
    def setBoard(self, moveString):
        """ Accepts a string of columns and places
            alternating checkers in those columns,
            starting with 'X'.

            For example, call b.setBoard('012345')
            to see 'X's and 'O's alternate on the
            bottom row, or b.setBoard('000000') to
            see them alternate in the left column.

            moveString must be a string of integers
        """
        nextCh = 'X'   # start by playing 'X'
        for colDigit in moveString:
            col = int(colDigit)
            if 0 <= col < self.width:
                self.addMove(col, nextCh)
            if nextCh == 'X':
                nextCh = 'O'
            else:
                nextCh = 'X'
```

Continue with this file for the next part of the problem (the `Player` class)

You'll continue developing this Connect-Four application in the second part of this problem. There, you'll implement an AI for the game in the `Player` class.

Sample run of `hostGame`

Sample run:

```
In [1]: b = Board(7, 6)

In [2]: b.hostGame()


Welcome to Connect Four!
```

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
---------------
 0 1 2 3 4 5 6

X's choice:  3

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
---------------
 0 1 2 3 4 5 6

O's choice:  4

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X|O| | |
---------------
 0 1 2 3 4 5 6

X's choice:  2

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |X|X|O| | |
---------------
 0 1 2 3 4 5 6

O's choice: 4

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | |O| | |
| | |X|X|O| | |
---------------
 0 1 2 3 4 5 6

X's choice: 1

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | |O| | |
| |X|X|X|O| | |
---------------
 0 1 2 3 4 5 6

O's choice: 2

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |O| |O| | |
| |X|X|X|O| | |
---------------
 0 1 2 3 4 5 6

X's choice:  0


X wins -- Congratulations!

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |O| |O| | |
|X|X|X|X|O| | |
---------------
 0 1 2 3 4 5 6
```