Problem 2: Tic-Tac-Toe + N

This problem asks you to write eight small functions—all very closely related—that operate on Python 2D data, that is, on lists-of-lists.

The application we have in mind is a *gameboard* in which your program will determine whether there are three-in-a-row of a single character.

To get started, make a copy of this trinket. Then try out the code!

Representation

Notice that all of the 2d data in this problem will be lists-of-lists-of-single-characters:

- The overall structure, typically called `A`, is a list of rows
- Each row is a list of data elements
- Each data element is a single-character string
- In fact, we will stick with only three strings:
  - `'X'`, a capital X
  - `'O'`, a capital O
  - and `' '`, the space character (this is *not* the empty string)

Checking for Three in a Row

The first four functions to write check whether a three-in-a-row occurs:

- In a specific direction (noted in the function name)
- For a specific checker `ch`
- At a specific starting location row and column: `r_start` and `c_start`
- Within a given 2d data array, `A`

Each one should return `False`:

- If there is **no room** for a three-in-a-row starting at `r_start` and `c_start`, or
- If `r_start` or `c_start` is out of bounds of `A`, or
- (even if there is room in bounds), if there is NOT a three-in-a-row pattern within `A` entirely matching the element `ch` in the specified direction starting at the `r_start` or `c_start` location.

On the other hand, each function should return `True` **only** if there **is** a three-in-a-row pattern within `A` entirely

matching the element `ch` in the specified direction starting at the `r_start` or `c_start` location.

Here are the signatures of the four functions to write:

1. `def inarow_3east( ch, r_start, c_start, A ):`
    - This should start from `r_start` and `c_start` and check for three-in-a-row **eastward** of element `ch`, returning `True` or `False`, as appropriate
2. `def inarow_3south( ch, r_start, c_start, A ):`
    - This should start from `r_start` and `c_start` and check for three-in-a-row **southward** of element `ch`, returning `True` or `False`, as appropriate
3. `def inarow_3se( ch, r_start, c_start, A ):`
    - This should start from `r_start` and `c_start` and check for three-in-a-row **southeastward** of element `ch`, returning `True` or `False`, as appropriate
4. `def inarow_3ne( ch, r_start, c_start, A ):`
    - This should start from `r_start` and `c_start` and check for three-in-a-row **northeastward** of element `ch`, returning `True` or `False`, as appropriate

You might notice that these four possibilities encompass **all** possible in-a-row combinations through the entire board (you'll leverage this in the next two weeks).

Here are four tests for each one—please do paste these into your file and make sure they work!.

```
# tests of inarow_3east
A = createA( 3, 4, 'XXOXXXOOOOOO')
# print2d(A)
print "inarow_3east('X',0,0,A): False ==", inarow_3east('X',0,0,A)
print "inarow_3east('O',2,1,A):  True ==", inarow_3east('O',2,1,A)
print "inarow_3east('X',2,1,A): False ==", inarow_3east('X',2,1,A)
print "inarow_3east('O',2,2,A): False ==", inarow_3east('O',2,2,A)

# tests of inarow_3south
A = createA( 4, 4, 'XXOXXXOXXOO OOOX')
# print2d(A)
print "inarow_3south('X',0,0,A):    True ==", inarow_3south('X',0,0,A)
print "inarow_3south('O',2,2,A):   False ==", inarow_3south('O',2,2,A)
print "inarow_3south('X',1,3,A):   False ==", inarow_3south('X',1,3,A)
print "inarow_3south('O',42,42,A): False ==",
inarow_3south('O',42,42,A)

# tests of inarow_3se
A = createA( 4, 4, 'XOOXXXOXX XOOOOX')
# print2d(A)
print "inarow_3se('X',1,1,A):  True ==", inarow_3se('X',1,1,A)
print "inarow_3se('X',1,0,A): False ==", inarow_3se('X',1,0,A)
print "inarow_3se('O',0,1,A):  True ==", inarow_3se('O',0,1,A)
print "inarow_3se('X',2,2,A): False ==", inarow_3se('X',2,2,A)

# tests of inarow_3ne
A = createA( 4, 4, 'XOXXXXOXXOXOOOOX')
# print2d(A)
print "inarow_3ne('X',2,0,A):  True ==", inarow_3ne('X',2,0,A)
print "inarow_3ne('O',3,0,A):  True ==", inarow_3ne('O',3,0,A)
print "inarow_3ne('O',3,1,A): False ==", inarow_3ne('O',3,1,A)
print "inarow_3ne('X',3,3,A): False ==", inarow_3ne('X',3,3,A)
```

Checking for N in a Row

Tic-tac-toe is a solved game! Let's handle *arbitrary* gameboards.

To that end, you'll generalize your three-in-a-row functions to N-in-a-row functions.

Each one will have one more input at the end, an integer `N`, which represents the number of identical elements (equal to `ch`) need to be found to return `True`

- If the starting location is out of bounds—or even in bounds, but counting `N` would reach out of bounds—your functions should return `False`.
- Of course, your functions should also return `False` even if the N checkers are entirely in-bounds, but there aren't N-in-a-row!

Here are the signatures of the four N-in-a-row functions to write:

1. `def inarow_Neast( ch, r_start, c_start, A, N ):`
    - this should start from `r_start` and `c_start` and check for N-in-a-row **eastward** of element `ch`,

returning `True` or `False`, as appropriate

2. `def inarow_Nsouth( ch, r_start, c_start, A, N ):`
    - this should start from `r_start` and `c_start` and check for N-in-a-row **southward** of element `ch`, returning `True` or `False`, as appropriate
3. `def inarow_Nse( ch, r_start, c_start, A, N ):`
    - this should start from `r_start` and `c_start` and check for N-in-a-row **southeastward** of element `ch`, returning `True` or `False`, as appropriate
4. `def inarow_Nne( ch, r_start, c_start, A, N ):`
    - this should start from `r_start` and `c_start` and check for N-in-a-row **northeastward** of element `ch`, returning `True` or `False`, as appropriate

Again, here are four tests for each one—please do paste these into your file and make sure they work!.

```
# tests of inarow_Neast
A = createA( 5, 5, 'XXOXXXOOOOOOXXXX XXXOOOOO')
# print2d(A)
print "inarow_Neast('O',1,1,A,4):  True ==", inarow_Neast('O',1,1,A,4)
print "inarow_Neast('O',1,3,A,2):  True ==", inarow_Neast('O',1,3,A,2)
print "inarow_Neast('X',3,2,A,4): False ==", inarow_Neast('X',3,2,A,4)
print "inarow_Neast('O',4,0,A,5):  True ==", inarow_Neast('O',4,0,A,5)


# tests of inarow_Nsouth
A = createA( 5, 5, 'XXOXXXOOOOOOXXXXOXXXOOOXO')
# print2d(A)
print "inarow_Nsouth('X',0,0,A,5): False ==",
inarow_Nsouth('X',0,0,A,5)
print "inarow_Nsouth('O',1,1,A,4):  True ==",
inarow_Nsouth('O',1,1,A,4)
print "inarow_Nsouth('O',0,1,A,6): False ==",
inarow_Nsouth('O',0,1,A,6)
print "inarow_Nsouth('X',4,3,A,1):  True ==",
inarow_Nsouth('X',4,3,A,1)


# tests of inarow_Nse
A = createA( 5, 5, 'XOO XXXOXOOOXXXXOXXXOOOXX' )
# print2d(A)
print "inarow_Nse('X',1,1,A,4):  True ==", inarow_Nse('X',1,1,A,4)
print "inarow_Nse('O',0,1,A,3): False ==", inarow_Nse('O',0,1,A,3)
print "inarow_Nse('O',0,1,A,2):  True ==", inarow_Nse('O',0,1,A,2)
print "inarow_Nse('X',3,0,A,2): False ==", inarow_Nse('X',3,0,A,2)


# tests of inarow_Nne
A = createA( 5, 5, 'XOO XXXOXOOOXOXXXOXXXOOXX' )
# print2d(A)
print "inarow_Nne('X',4,0,A,5):  True ==", inarow_Nne('X',4,0,A,5)
print "inarow_Nne('O',4,1,A,4):  True ==", inarow_Nne('O',4,1,A,4)
print "inarow_Nne('O',2,0,A,2): False ==", inarow_Nne('O',2,0,A,2)
print "inarow_Nne('X',0,3,A,1): False ==", inarow_Nne('X',0,3,A,1)
```

You'll be using these inarow functions over the next two weeks as you implement a version of Connect-Four!

Submit Homework 9, Problem 2

25.0/25.0 points (graded)

To submit your Homework 9, Problem 2 code, you'll need to copy it from your trinket or file and paste it into the box below. After you've pasted your code below, click the "Check" button.

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

1

```
2
3
4
5
6
7
8
9
10
11
def print2d( A
):
12
    """ print2d prints a 2d array,
A
```

```
13
        as rows and
columns
14
        input: A, a 2d list of
lists
15
        output: None (no return
value)
16
"""
17
    NR =
len(A)
18
    NC =
len(A[0])
19
20
    for r in range(NR):
21
        for c in range(NC):
22
```

```
            print A[r]
[c],
```

23

```
print('')
```

24

25

```
    return None
```

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 1 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.