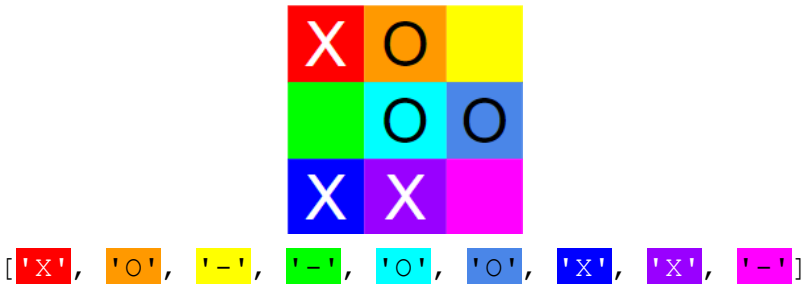
CSCI 1133
Lab 06

Lists

Most of this lab will be spent on making one large program with many interconnected functions, in order to play a game of tic-tac-toe.

Tic-Tac-Toe is a very simple game involving a 3x3 grid, where players take turns, attempting to place 3 of their marks in a row, either vertically, horizontally, or diagonally (see the link for details if you are unclear on the rules). You will be creating a program that plays tic-tac-toe by choosing an open spot at random, in order to determine how often 'X' wins, how often 'O' wins, and how often the game results in a draw when both players use that strategy.

We'll be representing the board as a single list of 9 one-character strings in this problem. The first 3 elements of the list represent the top row of the board, the second 3 elements represent the middle row, and the last 3 elements represent the bottom row. Each string can either be 'X', 'O', or '-' (which represents an empty space). See the diagram below for an example of how to represent a tic-tac-toe board as a 9 element list.



Warm-up

1). **Display the Board**

At various points throughout the other functions, we're going to want to see the state of the board not as a list of 9 elements, but in the more traditional 3x3 grid. Write a function `print_board(board)` that takes in a 9-element list as explained above, and prints out a graphical representation of the board. It doesn't have to exactly match the examples below so long as you can clearly distinguish the rows and columns. (Hint: It's probably going to be faster to just manually get the three list elements that make up each row and concatenate them together than to do something fancy with a loop in this case)

Examples:

```
>>> print_board(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'])
-XO
OX-
-XO

>>> print_board(['-', '-', '-', '-', '-', '-', '-', '-', '-'])
---
---
---

>>> print_board(['X', 'X', 'O', '-', 'X', '-', 'X', 'O', 'O'])
XXO
-X-
XOO

>>> print_board(['O', 'X', 'O', 'X', 'X', 'O', 'X', 'O', 'X'])
OXO
XXO
XOX
```

2). **Get the Open Spots**

It will be also important to determine which spots on the board are still available. To do this, write a function `open_spots(board)` which takes in the 9-element board list and returns a list of all of the indexes within the list that are still open (these should be numbers between 0 and 8, inclusive).

Hint: Loop through the possible indexes, and check whether the element at each index is `'-'`. If so, add it to the list of open indexes. Remember that you can add a value to a list using the .append method.

Examples:

```
>>> open_spots(['-', '-', '-', '-', '-', '-', '-', '-', '-'])
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> open_spots(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'])
[0, 5, 6]
>>> open_spots(['O', 'X', 'O', 'X', 'X', 'O', 'X', 'O', 'X'])
[]
>>> open_spots(['X', 'X', 'O', '-', 'X', '-', 'X', 'O', 'O'])
[3, 5]
```

Stretch

Remember that while it's a good idea to have one person primarily responsible for typing (the driver) and the other reviewing each line of code and making suggestions (the navigator) **you must switch roles after every problem**.

1). **Random Move**

Next, we're going to make a function that has the computer take a turn, by placing their symbol in one of the open spots at random. Write a function `random_move(board, symbol)` which takes in the 9-element board list and a symbol (either 'X' or 'O') representing whose turn it is. The function should pick one of the open spots in the board at random, and change the list at that index to that symbol. Then, it should print out the new board state.

You must call both `open_spots` and `print_board` as part of this function. You can assume that the board passed into the function has at least one open spot to choose from.

Hint: You can use `random.choice` to choose an element from the open spots list at random. Remember that you will need to import random.

Example (Note: which of the two open slots is chosen first is random, but the second call to random_move should always choose the other spot)**:**

```
>>> board = ['X', 'X', 'O', '-', 'X', '-', 'X', 'O', 'O']
>>> random_move(board, 'O')
XXO
OX-
XOO
>>> board
['X', 'X', 'O', 'O', 'X', '-', 'X', 'O', 'O']
>>> random_move(board, 'X')
XXO
OXX
XOO
>>> board
['X', 'X', 'O', 'O', 'X', 'X', 'X', 'O', 'O']
```

Workout

1). **Three of a Kind**

Determining whether the game is over is a bit complicated, so we're going to break it into two functions. The first function is just going to take in a board and three indexes, and determine whether all of them are 'X', all of them are 'O', or neither. Don't worry about whether those indexes are actually in a row yet, we'll handle that in the next function.

Write a function `check_three(board, idx1, idx2, idx3)` that takes in the 9-element board list, along with three indexes (these should be numbers between 0 and 8, inclusive). If the elements at all three indexes are `'X'`, then the function should return `'X'`. If the elements at all three indexes are `'O'`, then the function should return `'O'`. Otherwise, the function should return `'-'`.

Examples (indexes being checked are highlighted for clarity)**:**

```
>>> check_three(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'], 1, 4,
7)
'X'
>>> check_three(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'], 2, 5,
8)
'-'
>>> check_three(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'], 0, 5,
6)
'-'
>>> check_three(['-', 'X', 'O', 'O', 'X', '-', '-', 'X', 'O'], 2, 3,
8)
'O'
```

2). **Determine the Winner**

Now, we'll use the previous function in order to check the board for a winner.

Write a function `winner(board)` that takes in a 9-element board list, and returns a single character string that describes whether a player has won the game. In particular, it should return:

- `'X'`, if X won the game (there are three X's in a row, horizontally, vertically or diagonally)
- `'O'`, if O won the game
- `'D'`, if the game ended in a draw (all spots are filled but neither player won)
- `'-'`, if the game is not over (there is at least one empty spot and neither player won)

You must call both `check_three` and `open_spots` as part of this function. You can ignore the possibility of getting a board where both X and O have achieved three in a row, since this should not happen in a real game.

Here are the 8 combinations of three indexes that represent three in a row horizontally, vertically, or diagonally:

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8],
[0, 3, 6], [1, 4, 7], [2, 5, 8],
[0, 4, 8], [2, 4, 6]]
```

Hints:

- You could write 8 different if statements for the 8 combinations, but it's much more efficient to just loop through the list of lists above and call the `check_three` function inside of the loop. If you're not sure how that would work, try the following code, to see what it prints out:

```
combos = [[0, 1, 2], [3, 4, 5], [6, 7, 8],
[0, 3, 6], [1, 4, 7], [2, 5, 8],
[0, 4, 8], [2, 4, 6]]
for triple in combos:
    print(triple[0], triple[1], triple[2])
```

- Once you have checked whether X or O has won for all eight possible combinations listed above, checking for a draw is simple: just see if there are any open spots. If neither player won, and there are no open spots left, it's a draw.

```
>>> winner(['X', 'X', 'O', 'O', 'X', 'X', 'O', 'X', 'O'])
'X'
>>> winner(['X', '-', 'O', 'X', 'O', '-', 'O', '-', 'X'])
'O'
>>> winner(['O', 'X', 'O', 'X', 'X', 'O', 'X', 'O', 'X'])
'D'
>>> winner(['X', 'X', 'O', '-', 'X', '-', 'X', 'O', 'O'])
'-'
>>> winner(['-', '-', '-', 'X', 'X', 'X', 'O', 'O', '-'])
'X'
```

Challenge

1). **Play Game**

Finally, we can combine the functions in the Stretch and the Workout to make the computer play a game of tic-tac-toe against itself.

Write a function `tic_tac_toe()` that takes in no arguments, and simulates a single game of tic-tac-toe in which the computer plays randomly against itself. The function returns a one character string that signifies the winner ('X', 'O', or 'D' for a draw). This simulation needs to follow the rules of tic-tac-toe: X and O alternate turns, starting with X; each player must choose an empty space on their turn; and the game ends as soon as someone wins or all spaces are filled.

You must call both `winner` and `random_move` as part of this function.

Hint: Keep looping until the result of `winner` is something other than `'-'`. You will need some way to keep track of whose turn it is (X or O): one strategy is to just keep a turn counter and check whether that counter is currently even or odd.

Examples (text in italics is printed, text in bold is returned)
Note that since the moves are random, your output will be different.

```
>>> tic_tac_toe()
---
---
--X

O--
---
--X

O--
---
X-X

O--
--O
X-X

O--
--O
XXX
```

```
'X'

>>> tic_tac_toe()
---
---
-X-

-0-
---
-X-

-0-
---
XX-

-0-
---
XX0

-0-
X--
XX0

-00
X--
XX0

-00
XX-
XX0

-00
XX0
XX0

'O'
```

2). **Play 100 Games**

Write a program to run 100 games of tic-tac-toe, and display the number of wins, losses, and draws. You may want to comment out the calls to `print_board` in `random_move` to avoid a ridiculous amount of output printed to the console. Probability says that you should see X win roughly 58% of the time, O win about 29% of the time, and a draw about 13% of the time: how close does your simulation get to the prediction?

Example:

```
X wins: 48
O wins: 41
Draws: 11
```