In this lab, you will write a program that lets you play Tic-Tac-Toe against the computer. See `https://en.wikipedia.org/wiki/Tic-tac-toe` for details on the game. In this lab, you should try to write functions that are as concise as possible. In many cases, this means using loops even if a more tedious solution that doesn't use loops is possible.

## Problem 1.

Your first task is to enable two users to play against each other. Download `ttt.py`, and understand the functions for creating an empty board and to print the board and the legend. Run `ttt.py` and observe how `print_board_and_legend(board)` prints the list of lists `board`, which represents the board.

The goal is to be able to produce a game that goes, for example, as follows:

```
   |   |         1 | 2 | 3
---+---+---      ---+---+---
   |   |         4 | 5 | 6
---+---+---      ---+---+---
   |   |         7 | 8 | 9
Enter your move: 5


   |   |         1 | 2 | 3
---+---+---      ---+---+---
   | X |         4 | 5 | 6
---+---+---      ---+---+---
   |   |         7 | 8 | 9
Enter your move: 1


 O |   |         1 | 2 | 3
---+---+---      ---+---+---
   | X |         4 | 5 | 6
---+---+---      ---+---+---
   |   |         7 | 8 | 9
Enter your move: 3
```

### Part (a)

Write a function that takes in an integer `square_num` between 1 and 9, and returns a list `coord` such that `board[coord[0]][coord[1]] = "X"` would put an `"X"` in square `square_num` (an integer from 1 to 9). Hint: the row number is `((square_num - 1) // 3)`.

### Part (b)

Write a function `put_in_board(board, mark, square_num)` that modifies the contents of `board` such that the string `mark` (`"X"` or `"O"`) is put in the coordinates in `board` that correspond to `square_num`.

**Part (c)**

Write a loop that asks for the user to alternately enter coordinates for "X"s and "O"s such that two users can play against each other as shown in the example above. Reminder: there is an example of repeatedly asking the user for input here:
`http://www.cs.toronto.edu/~guerzhoy/180/lectures/W04/lec2/WhileInput.html`

## Problem 2.

The goal now is to write a simple function that would have the computer play against the user.

**Part (a)**

Write a function with the signature `get_free_squares(board)` which creates and returns a new list which contains a list of the coordinates of the free squares in the board. For example, if the board is represented as follows

```
 O |   | X
---+---+---
   | X |
---+---+---
 O |   |
```

, the function should return `[[0, 1], [1, 0], [1, 2], [2, 1], [2, 2]]`.

**Part (b)**

Now write a function `make_random_move(board, mark)` that finds a random free square in `board`, and puts the string `mark` in the free square. Hint: you can print a random number between `0` and `n-1` as follows:

```
import random
print(n * random.random())
```

**Part (c)**

Now use `make_random_move()` in order to have the computer play against the user.

## Problem 3.

Now, the goal is to automatically figure out if the game is over. The game is over if there is a line of 3 "X"s or a line of 3 "O"s.

**Part (a)**

Write a function with the signature `is_row_all_marks(board, row_i, mark)` which returns `True` iff the row with index `row_i` in `board` contains 3 marks equal to `mark`.

**Part (b)**

Write a function with the signature `is_col_all_marks(board, col_i, mark)` which returns `True` iff the column with index `row_i` in `board` contains 3 marks equal to `mark`.

**Part (c)**

Using the functions above, and also checking the diagonals, write a function with the signature `is_win(board, mark)` that returns `True` iff the mark `mark` won on the board `board` (i.e., there is a line of 3 `mark`s somewhere in `board`).

**Part (d)**

Incorporate `is_win()` into the program such that the game stops when either the user or the computer win, and prints the result of the game.

# Problem 4.

Your job now is to improve the the function that makes the computer's move.

**Part (a)**

Write a function which tries to put the computer's mark in every free square on the board, and checks whether `is_win()` returns `True` for the new board, returns if it does, and removes the mark and tries to place it in another square otherwise. If there is no square such that putting a mark in it leads to an immediate win, the function should put `mark` in a random free square.

**Part (b)**

Improve the algorithm that plays for the computer as much as you can.