***LAB-07: Minimax Search***

***Objective:***

Apply and implementMinimax algorithm.

***Activity Outcomes:***

An implementation of Minimax AI Algorithm on Tic-Tac-Toe.

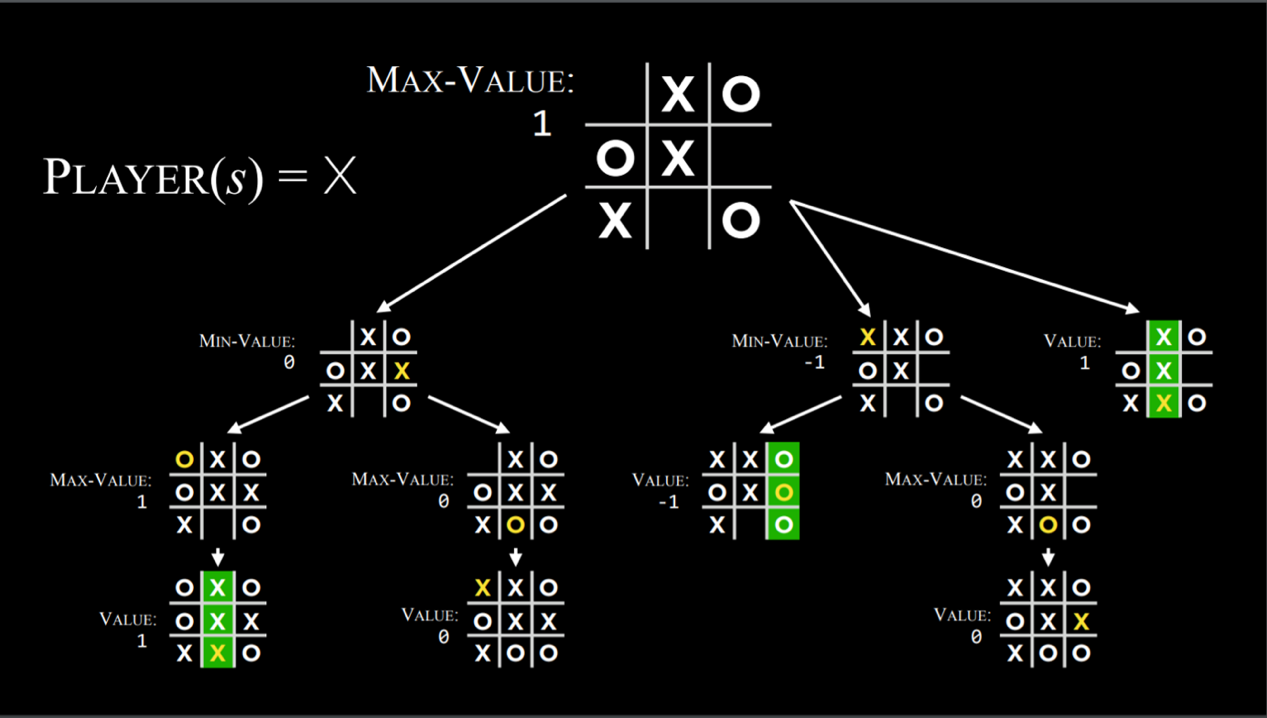
***Lab Requirements:***

Jupyter notebook (Anaconda 3)

***Lab Description:***

**Minimax algorithm:**

Minimax is a artificial intelligence applied in two player games, such as tic-tac-toe, checkers, chess and go. This games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of anyone not to win (0).



**The game flow:**

Let’s say the computer is represented by the character ‘O’ and the user is represented by ‘X’.

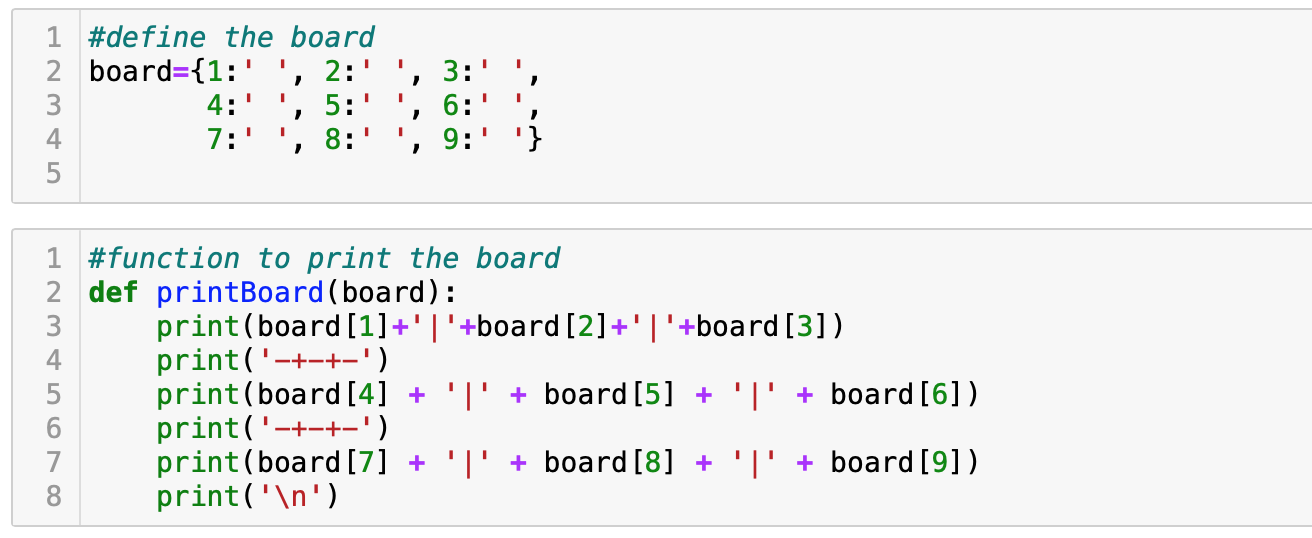
1. Print the empty board.
2. The computer makes the first move and the board is printed with an ‘O’ added to the slot that the algorithm picked.
3. The user is prompted to pick a slot represented by a number.
4. Once the user inputs a number, that slot is checked whether it’s empty or not, if it’s taken, an error is displayed and the user is prompted to pick another slot if the slot is empty, an ‘X’ is inserted there.
5. The algorithm then plays with itself in the background, checking the score yielded by picking every available slot and finding the best move. Once the best move is found, it makes that move, prints out the board, and back to step 3.

**Building a bot to play Tic-Tac-Toe**

Tic-Tac-Toe (Noughts and Crosses) is probably one of the most famous games. Let's see how to build a game where the computer can play against the user.

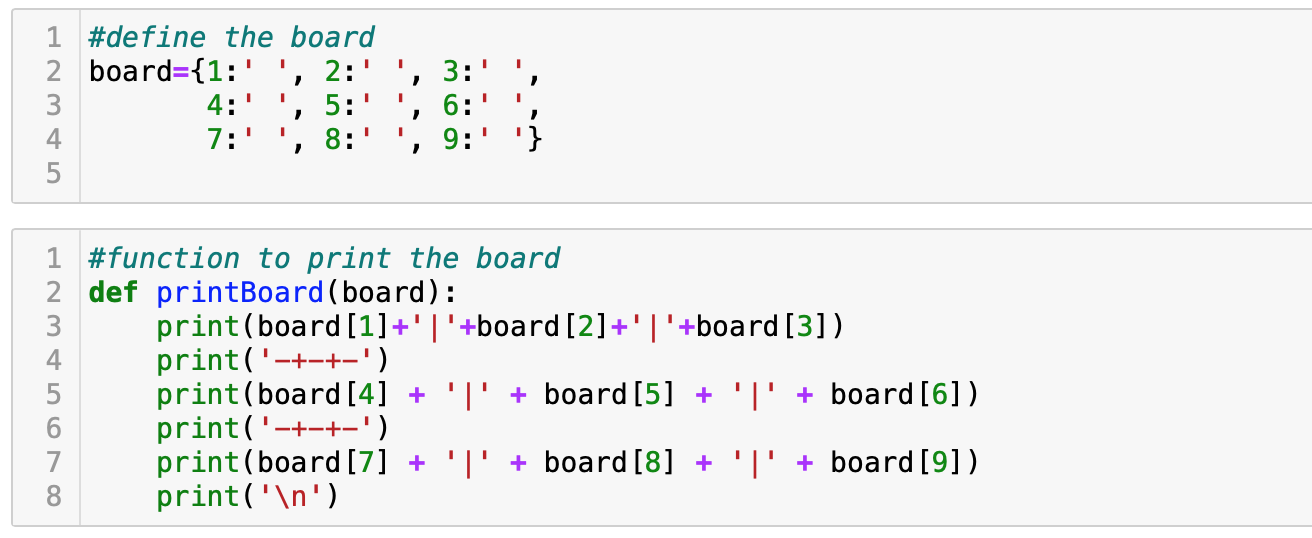
**Defining the game board:**

We can think of the board as a dictionary. In tic tac toe, the board has a total of 9 slots, and each slot is represented by a number which will be the key in our dictionary as shown in the figure below. In the beginning, all the slots are empty so all the keys hold a value of ‘ ’, basically empty space.



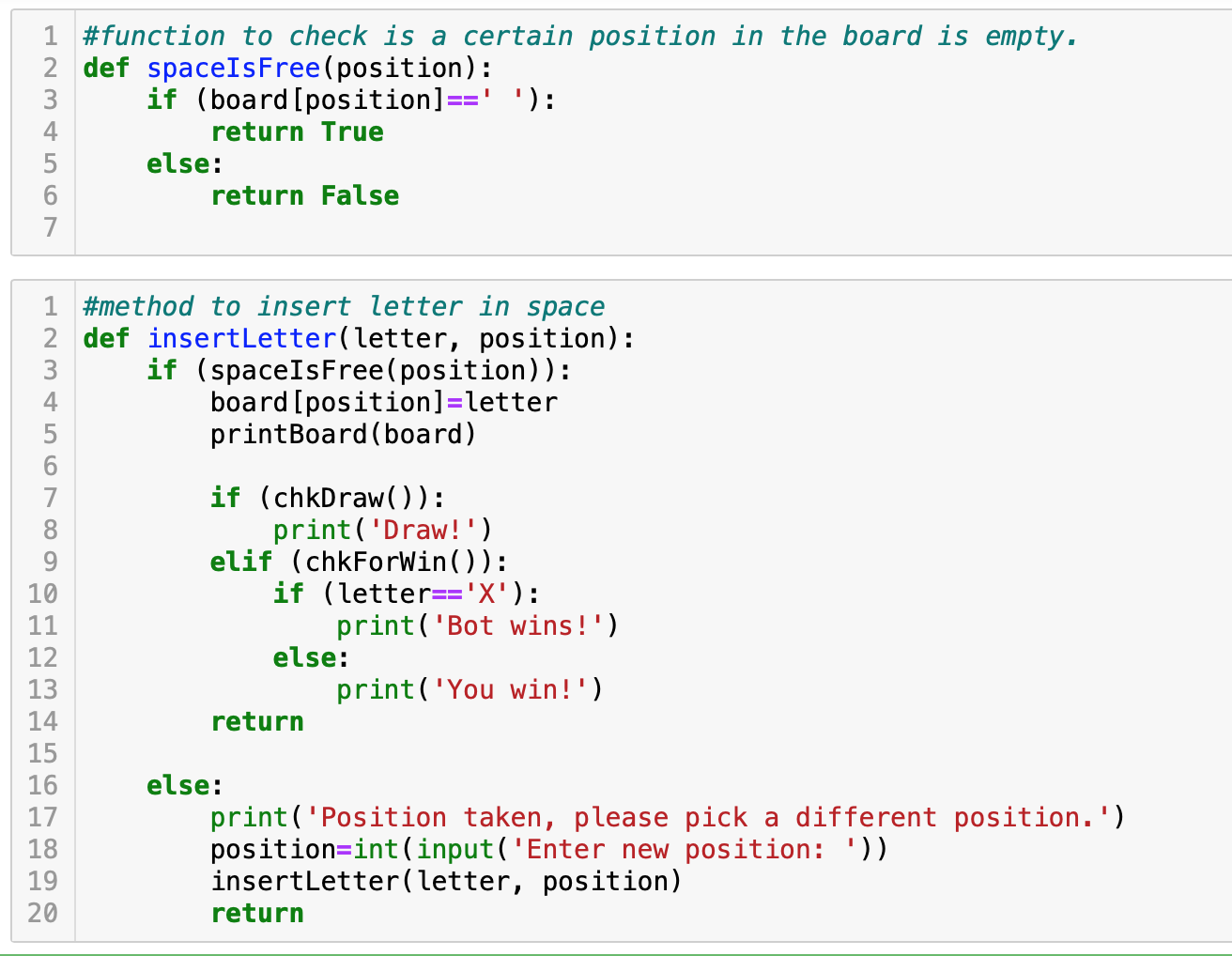
**Printing the board:**

The following function will be used to print the board:



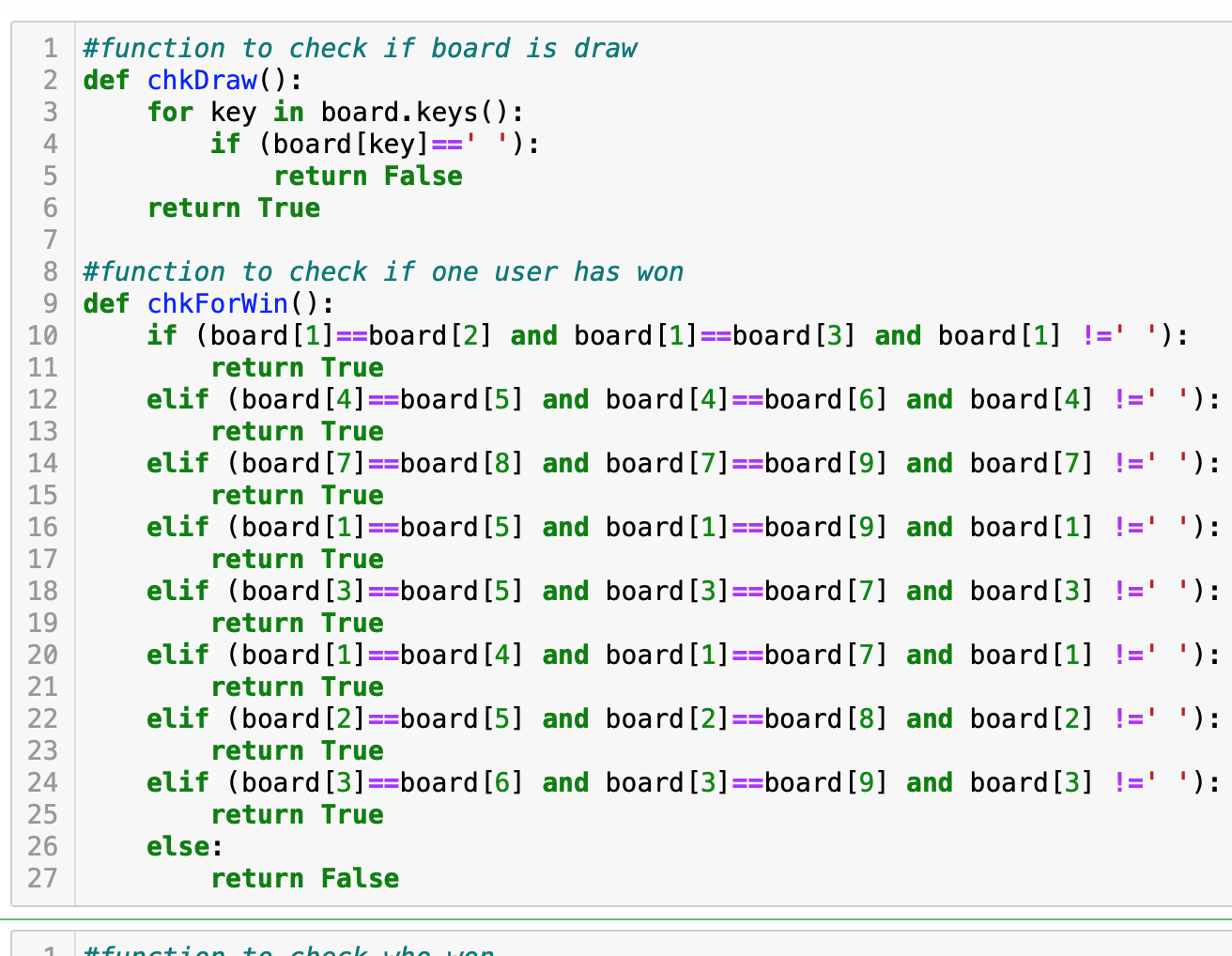
**Function to check whether a slot is free or not:**

Every time a user wants to make a move, the following function will check whether the user selected slot is available or taken. It returns true if the slot is free and false if the slot is taken.



**Checking if the game has been won:**

The board is in a won state when 3 consecutive slots carry the same symbol/mark. This could be horizontal, vertical, or diagonal. So using code, we define all the possible combinations of various slots that would lead to a user winning.



**Check if the game is a draw:**

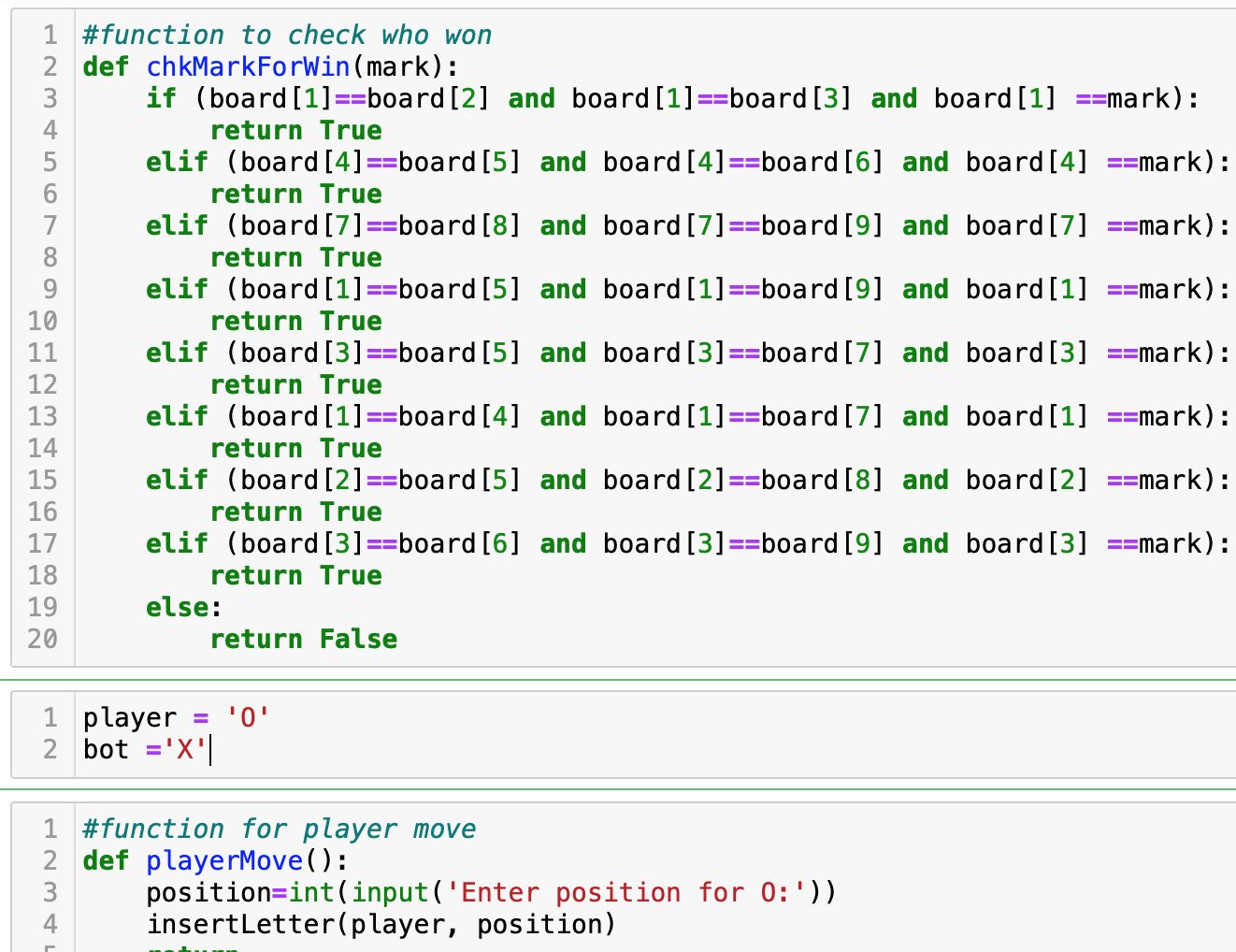
Assuming the game is not won and all the slots are filled and there is no move that can be made, the game is considered a draw. Thus we write a function that checks all the slots and if they are all full, the game is considered a draw.

Graphical user interface, text, application

Description automatically generated

**Check to see which player won:**

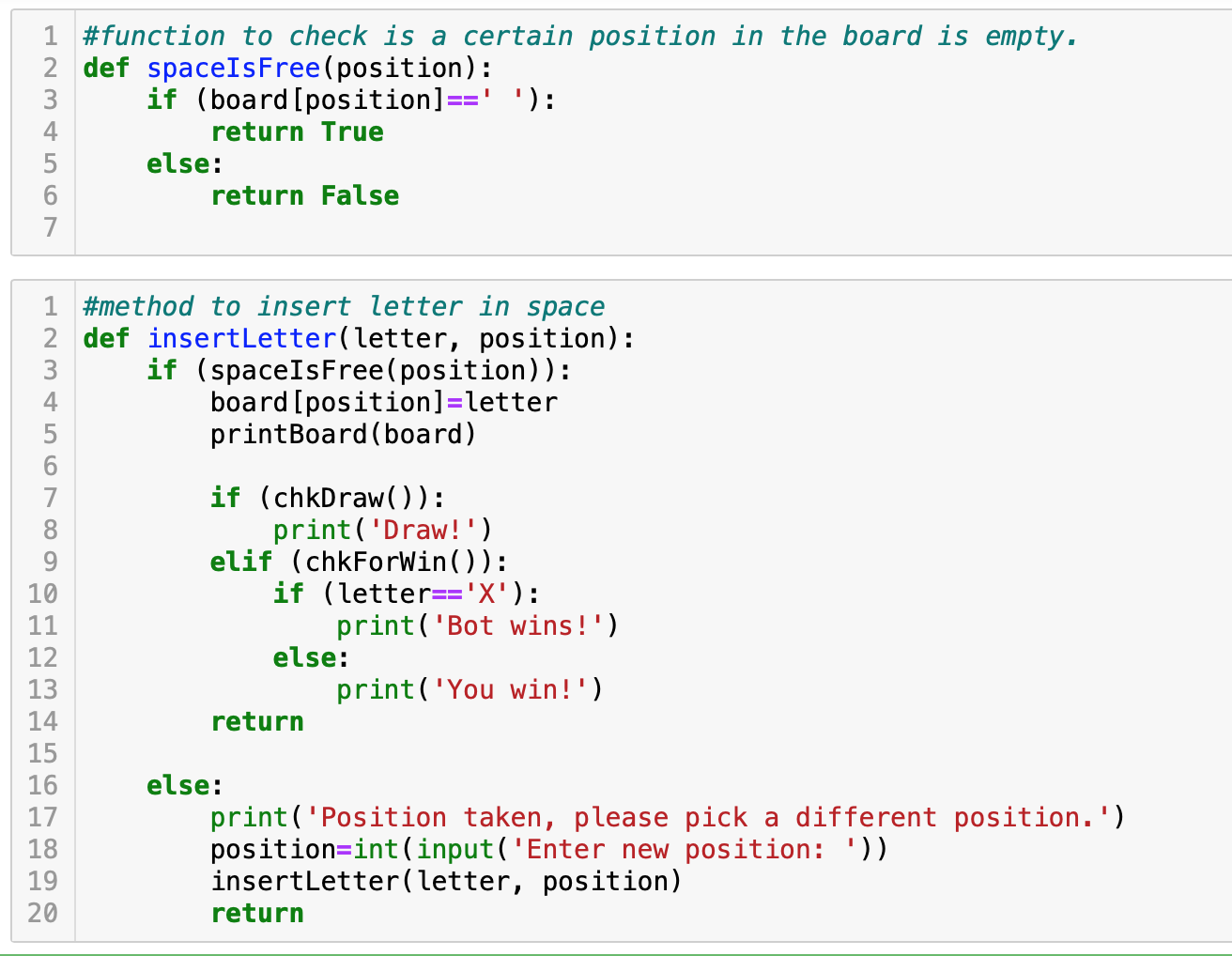
The following function will accept the mark/symbol and check if the board is in a won state for the given symbol. If the given symbol has won, it will return true, otherwise, it will return false.



**Inserting a symbol at a certain slot:**

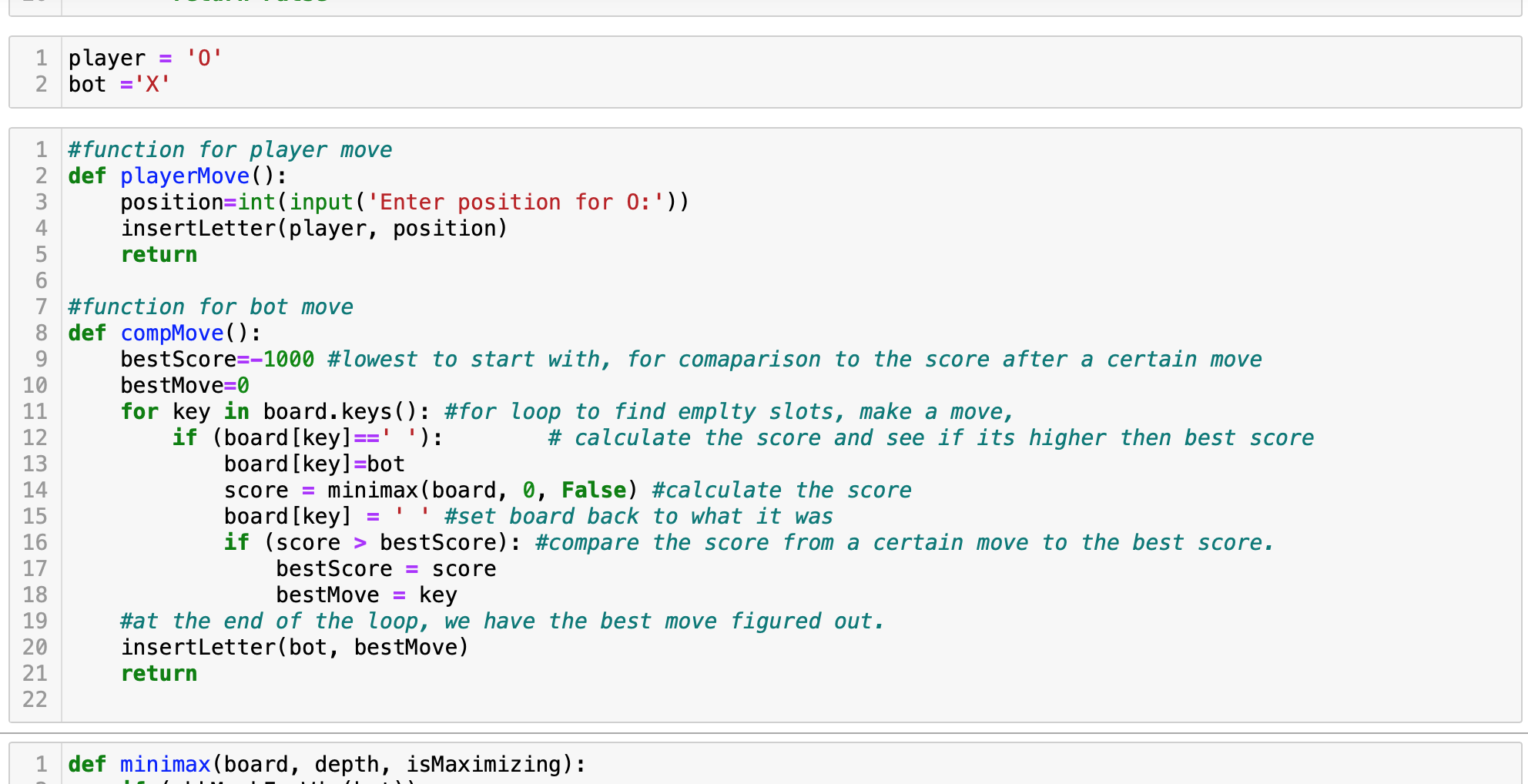
Now that we have the various checks on the board defined, let’s write the function that will insert a symbol/mark at a certain slot.

The user provides a number representing the slot. The function then first makes sure that the slot is free. If it is, it inserts the mark there. After that, it checks if the game is a draw. If it’s not a draw, then it checks if the game has been won.



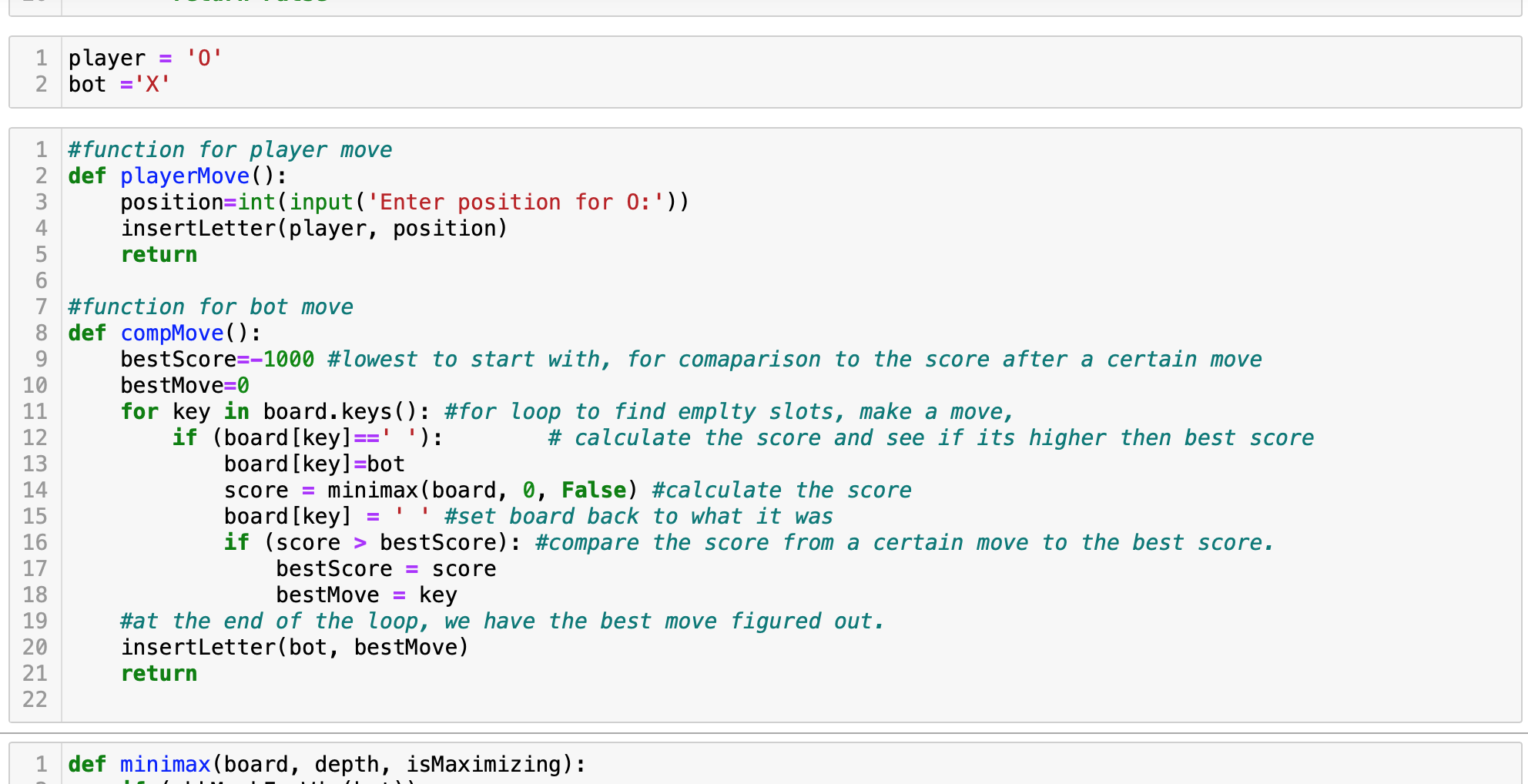
**Assigning marks/symbols to player and computer/algorithm:**

The player is represented by ‘O’ and the computer by ‘X’.



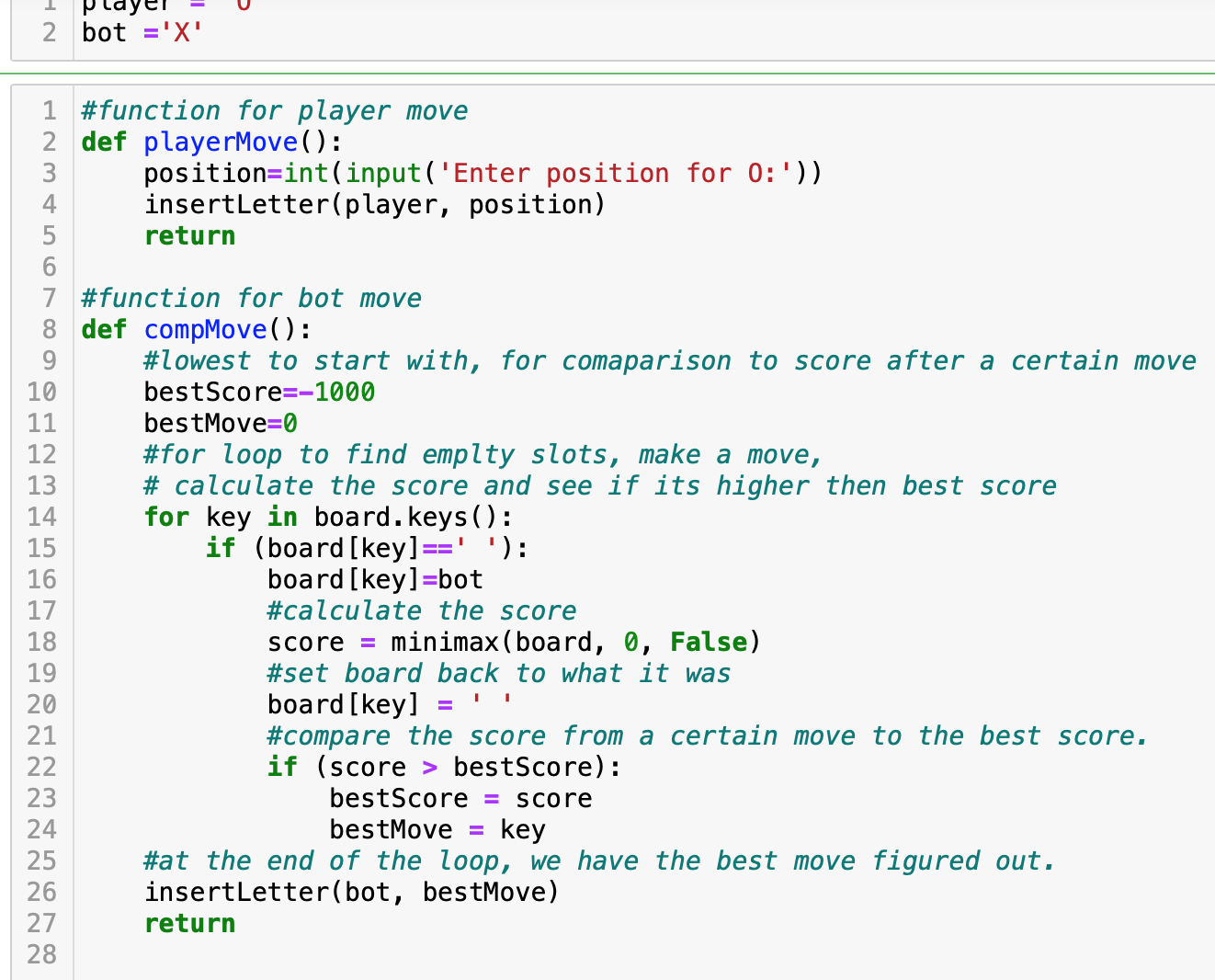
**Making the move:**

Now we define two functions. One that is called when the player makes a move, and the second when the computer makes a move.

****

**The function that determines the next move by the computer:**

This function analyzes every available move and returns the best possible move. It iterates over every slot, if it is available, it calls the minimax function to determine if that move will yield the best score or not. If it does, it assigns that key to the bestMove variable and inserts a mark for the computer in that slot.



**The minimax function:**

* The minimax function receives the board, checks the status to see who won, if the move led to the computer winning, it returns 1 to the compMove function.
* If minimax checks the board status and finds out that the user has won, it returns -1 to compMove. -1 is less than the best score so nothing happens, the compMove function moves to the next key.
* If the board turns out to be a draw, as determined by chkDraw function, 0 is returned to the compMove function, the iteration continues to the next key.



Thus all the keys are iterated and possible combinations of the user vs computer game are simulated. In the end, compMove will have the winning move figured out and uses that.

**Starting the game:**

**Graphical user interface, text, application

Description automatically generated**

**Assessment:**

Implement the Minimax AI Algorithm on Tic-Tac-Toe.