Greetings! This is Mohammed Kamil Khan and I will be explaining to you the contributions of each function towards getting a suitable output for a basic Tic-Tac-Toe game, with a few more changes/additions to be made in the following days/weeks. Just another point that majority of the logic behind the game comprises of the concept of lists. Let’s get started!

* In the first cell, we are creating lists t7,t8,t9,t4,t5,t6,t1,t2,t3 with only a single element in them, mainly for the proper functioning of the display function which will be shown in a while. A point to note is that the foundational representation of the game board that i have chosen is as follows :

7|8|9

4|5|6

1|2|3

* The second cell comprises of the creation several other lists. I would like to call these lists as “DEFAULT LISTS”. Emphasis on these default lists will be laid later in the program.
* The 3rd cell comprises of a list called posi with elements 1-9. This list indicates the positions on the board which are available. For example, if the first user inserts his/her symbol at the position ‘7’, the ‘7’ from the posi list will be popped out. This cell is essential for the sole purpose of serving one functionality of the game : When a user inserts a position that has already been taken over by the other user or himself/herself earlier in the game, like the above example, after 7 has been popped out after a single insertion in that position, if either of the two users, later on, select the position ‘7’, a message will be displayed that “This position is already taken, please select another one’. We use the global keyword to make this variable, to wish a list is defined, accessible to other functions.
* In the 4th cell, we create a list called posi1. This list comprises of the elements that have been popped out from the list ‘posi1’ as mentioned in the previous point. Taking the above example: after 7 has been selected by a user to place his marker on, 7 will be popped out of posi and appended(added) into the list ‘posi1’.
* The 5th cell comprises of the display function which displays the table and the positions/markers it contains after each turn. For example, we have initialised lists t1,t2,….t9 to certain single element lists. Before we start the game, the table will be displayed as shown in the diagrammatic representation of the table as shown in the 1st point. After each turn played, the initial elements will be replaced by the respective markers : ‘X’ or ‘O’.
* The 6th cell comprises of the check\_true() function whose sole purpose is to check whether a game has been won or not. **Strongly recommend you to read this point thoroughly AFTER** **the next point**. We know that there are 8 ways in which a tic-tac-toe game can be one. If we consider the diagrammatic representation of the table, the winning combinations are as follows: 7-8-9, 4-5-6,1-2-3,7-4-1,8-5-2,9-6-3,7-5-3,9-5-1. For each condition, we display the message that the person has won. ‘playagain’ is an input statement which asks whether the persons wish to play again. If ‘Y’(Yes) is selected, we are popping out all the elements of the lists t1,t2,t3….t9, after the completion of the game, with the elements of the DEFAULT LIST as mentioned in the 2nd point. This is done so that in the next game, the display will be back to the diagrammatic representation mentioned above and will not contain the markers or left-over positions of the previous game. I would recommend giving a thorough read through this point right after the next point. The for loop towards the end of every winning condition is for appending/adding the elements of the posi1 list into the posi list. To give you a clear idea, let’s consider the following example:

Let’s say that the resultant table of a game won is as follows:

X|8|O

O|X|6

1|2|X

In this case, the lists posi and posi1 will contain the following elements:

posi = [1,2,6,8]; posi1 = [7,9,5,4,3]

We perform the for loop operation to append the elements of posi1 list into posi list, so that in the next game, though the display aspect of the board has been reset to its original state, the problem of us selecting an available position (visibly) and a statement like “This position is already taken”, will be avoided. In case this is confusing, let’s take the continuation of the above example:

Following the success of us replacing the markers or positions with the single-elements of the table (after a game) with the elements of the default lists, if we do not consider the for loop portion, the following would happen:

Take the same end-result of a game as show above. The display aspect will be refreshed, but, not performing the for loop operation, the lists posi and posi1 will remain the same as above, i.e., posi = [1,2,6,8] and posi1 = [7,9,5,4,3], and if, in the next game, a user selects a position which is NOT in the posi list, a statement will appear that the position is already taken. The else block at the end of each win condition consists of the pass keyword, which basically continues the flow of the program if that particular win condition has not been satisfied.

* The 7th cell comprises of the ask() function which initialises the game, with function calls to different functions for different purposes.

The first question asked is which symbol the player wishes to choose, either ‘X’ or ‘O’. A list called mylist is created which is meant to contain 2 elements : ‘X’ and ‘O’. If the player chooses ‘X’, ‘X’ is appended into the mylist list, and automatically ‘O’ is appended into mylist list as the second element. Hence, mylist = [‘X’,’O’], and vice versa, i.e., mylist = [‘O’,’X’].

The for loop is meant for switching turns between the 2 players. We know that the maximum number of turns in this game (which leads to a draw) is 9, with the 1st player getting the turns 1,3,5,7,9 and the second player getting 2,4,6,8. However, in the range of for loop, we have included 10, this is because when ‘i’ is 10, it represents a draw as maximum number of turns in this game is 10. The “if i%2 != 0” depicts the turns taken by the first player, and the “if i%2 == 0 and i!= 10” depicts the turns of the second player. The ‘pos’ statement is an input statement whose data type is a string. The “if int(pos) in posi” is to check whether the inputted position is available or not. If it is, we give a function call to calling1(), if not, we give a function call to receiving1. Same goes for the second player with function calls to calling2 and receiving2 functions as per the respective conditions. The purposes of the functions will be explained in the next point.

* To fully understand the purpose of the calling1() function, let’s briefly study one case:

Suppose the user inputs 7, posi (which contains the available positions of the board) will remove(the ‘remove’ keyword is used for deleting a particular element as ‘pop’ is used to either delete the last element i.e., pop(), or delete an element at a particular index i.e., pop(index)) ‘7’ from its list and ‘7’ will be added into posi1(the list which contains the positions taken by a user). The “t7.pop()” removes the 7 from its list. The “t7.append(mylist[0])” appends/adds into t7(which is now an empty set due to removal of 7) the symbol chosen by the first player i.e., mylist[0]. All the if statements in this function are for different positions which the user might input. As you may notice, the calling2() function uses exactly the same logic as calling1() function, the only difference being that we append mylist[1] i.e., for the second player.

* The functions receiving1() and receiving2() are called when the user enters a position that is already taken i.e., if the position is not in the ‘posi’ list. In this block, using the while block, we repeated ask about inserting another position until the user enters a position that is not taken, as we can see, after else(when the ‘pos’ input is in posi(list of available positions)) we give a function call to calling1() function to carry out the positioning of the marker of that particular player.