Lab 4

Classic game: Tic Tac Toe

The program allows two players to play **Tic Tac Toe** by taking turns to mark a 3x3 grid with either "X" or "O". The game continues until one player wins by forming a straight line with their marks (horizontally, vertically, or diagonally) or the game ends in a draw when all squares are filled without any player winning.

```
square = [0,1,2,3,4,5,6,7,8,9]

def main():
    player = 1
    status = -1

    while status== -1:
        board()

        if player%2 == 1:
            player = 1
        else:
            player = 2

        print('\nPlayer', player)
        choice = int(input('Enter a number:'))

        if player == 1:
            mark = 'X'
        else:
            mark = 'O'

        if choice == 1 and square[1] == 1:
            square[1] = mark
        elif choice == 2 and square[2] == 2:
            square[2] = mark
        elif choice == 3 and square[3] == 3:
            square[3] = mark
        elif choice == 4 and square[4] == 4:
            square[4] = mark
        elif choice == 5 and square[5] == 5:
            square[5] = mark
        elif choice == 6 and square[6] == 6:
            square[6] = mark
        elif choice == 7 and square[7] == 7:
            square[7] = mark
        elif choice == 8 and square[8] == 8:
            square[8] = mark
        elif choice == 9 and square[9] == 9:
            square[9] = mark
        else:
            print('Invalid move ')
            player -= 1

        status = game_status()
        player += 1

    print('RESULT')
    if status == 1:
        print('Player',player-1,'win')
    else:
        print('Game draw')


##############################################
#    FUNCTION TO RETURN GAME STATUS
```

```
#    1 FOR GAME IS OVER WITH RESULT
#    -1 FOR GAME IS IN PROGRESS
#    O GAME IS OVER AND NO RESULT
################################################

def game_status():
    if square[1] == square[2] and square[2] == square[3]:
        return 1
    elif square[4] == square[5] and square[5] == square[6]:
        return 1
    elif square[7] == square[8] and square[8] == square[9]:
        return 1
    elif square[1] == square[4] and square[4] == square[7]:
        return 1
    elif square[2] == square[5] and square[5] == square[8]:
        return 1
    elif square[3] == square[6] and square[6] == square[9]:
        return 1
    elif square[1] == square[5] and square[5] == square[9]:
        return 1
    elif square[3] == square[5] and square[5] == square[7]:
        return 1
    elif square[1] != 1 and square[2] != 2 and square[3] != 3 and square[4] != 4 and square[5]
!= 5 and square[6] != 6 and square[7] != 7 and square[8] != 8 and square[9] != 9:
        return 0
    else:
        return -1



################################################
#    FUNCTION TO DRAW BOARD
#    OF TIC TAC TOE WITH PLAYERS MARK
################################################


def board():
    print('\n\n\tTic Tac Toe\n\n')

    print('Player 1 (X)  -  Player 2 (O)' )
    print()

    print('     |     |     ' )
    print(' ' ,square[1] ,' | ' ,square[2] ,' |  ' ,square[3] )

    print('_____|_____|_____' )
    print('     |     |     ' )

    print(' ' ,square[4] ,' | ' ,square[5] ,' |  ' ,square[6] )

    print('_____|_____|_____' )
    print('     |     |     ' )

    print(' ' ,square[7] ,' | ' ,square[8] ,' |  ' ,square[9] )

    print('     |     |     ' )

main()
```

A quick explanation of the LUHN algorithm

The algorithm we're going to use to verify card numbers is called the Luhn algorithm, or Luhn formula. This algorithm is actually used in real-life applications to test credit or debit card numbers as well as SIM card serial numbers.

The purpose of the algorithm is to identify potentially mistyped numbers, because it can determine whether or not it's possible for a given number to be the number for a valid card.

The way we're going to use the algorithm is as follows:

Remove the rightmost digit from the card number. This number is called the checking digit, and it will be excluded from most of our calculations.

Reverse the order of the remaining digits.

For this sequence of reversed digits, take the digits at each of the even indices (0, 2, 4, 6, etc.) and double them. If any of the results are greater than 9, subtract 9 from those numbers.

Add together all of the results and add the checking digit.

If the result is divisible by 10, the number is a valid card number. If it's not, the card number is not valid.

Let's look at this step by step for a valid number so we can see this in action. The number we're going to use is **"5 8 9 3 8 0 4 1 1 5 4 5 7 2 8 9"**, which is a valid Master card number, but not one which is in use.

Number	Operation
5 8 9 3 8 0 4 1 1 5 4 5 7 2 8 9	Starting number
5 8 9 3 8 0 4 1 1 5 4 5 7 2 8 X	Remove the last digit
8 2 7 5 4 5 1 1 4 0 8 3 9 8 5 X	Reverse the remaining digits
16 2 14 5 8 5 2 1 8 0 16 3 18 8 10 X	Double digits at even indices
7 2 5 5 8 5 2 1 8 0 7 3 9 8 1 X	Subtract 9 if over 9

Now we sum these digits and add the checking digit:

7 + 2 + 5 + 5 + 8 + 5 + 2 + 1 + 8 + 0 + 7 + 3 + 9 + 8 + 1 + X

X = 9 (which we removed in the start)

7 + 2 + 5 + 5 + 8 + 5 + 2 + 1 + 8 + 0 + 7 + 3 + 9 + 8 + 1 + 9

If we perform this series if additions, we get **80**.

80 is divisible by **10**, so the card number is **valid**.

Lab 4 Task

1. **Code for LUHN Algorithm**
2. **Write a python program to remove punctuations from the given string?**
3. **Write a python program to sort the sentence in alphabetical order?**