Data Structures 2023-2

Lab 02

1. Task-1: Matrix Data Type

Develop a Matrix Data Type that has the following functionalities

__init__(row, col) : it initializes a matrix of size rows x cols with random values

__str__() : It returns the string representation of matrix

__repr__() : It is representation of Matrix object

__add__() : it multiplies two matrices and returns a resultant matrix

__sub__() : it subtracts two matrices and returns a resultant matrix

__mul__() : it multiplies two matrices and returns a resultant matrix

transpose() :It returns the transpose of the matrix

Lab02 Code

```python
import random
class Matrix:
    rnb = random.Random()
    def __init__(self, rows, cols, f='r'):
        self.M = []
        if f == 'r':
            self.rMatrix(rows, cols)
        else :
            self.zMatrix(rows, cols)


    def rMatrix(self, rows, cols):
        while len(self.M) < rows:
            self.M.append([])
            while len(self.M[-1]) < cols:
                self.M[-1].append(Matrix.rnb.randint(1, 10

    def zMatrix(self, rows, cols):
        while len(self.M) < rows:
            self.M.append([])
            while len(self.M[-1]) < cols:
                self.M[-1].append(0)

    def mPrint(self):
        for rows in self.M:
            print([x for x in rows])

    def __str__(self):
        return self.M

    def __repr__(self):
        print("Mymatrix is that")
        return self.M
```

```python
    def __add__(self, other):
        rowsA = len(self.M)
        colsA = len(self.M[0])
        C = Matrix(rowsA, colsA, 'z')

        for row in range(rowsA):
            for col in range(colsA):
                C.M[row][col] = self.M[row][col] + other.M[row][col]
        print("add two Matrix!")
        return C
    def __sub__(self, other):
        rowsA = len(self.M)
        colsA = len(self.M[0])
        C = Matrix(rowsA, colsA, 'z')

        for row in range(rowsA):
            for col in range(colsA):
                C.M[row][col] = self.M[row][col] - other.M[row][col]
        print("sub two Matrix!")
        return C

    def __mul__(self, other):
        rowsA = len(self.M)
        colsA = len(self.M[0])
        C = Matrix(rowsA, colsA, 'z')

        for X in range(rowsA):
            for Y in range(colsA):
                for Z in range(rowsA):
                    C.M[X][Y] = self.M[X][Z] * other.M[Z][Y]
        print("mul two Matrix!")
        return C


    def transpose(self):
        rowsA = len(self.M)
        colsA = len(self.M[0])
        T = [[0 for _ in range(rowsA)] for _ in range(colsA)]

        for X in range(rowsA):
            for Y in range(colsA):
                T[Y][X] = self.M[X][Y]
        return T
```

Lab02Test Code

```
from Lab02 import Matrix, EightQueens, TicTacToe
def useMatrix():
    print("Matrix initializer and string __str__")
    print(*Matrix(3, 3, 'r' ).__str__(), sep="\n")
    print("Matrix initializer and representaion __repr__()")
    print(*Matrix(3, 3, 'r').__repr__(), sep="\n")
    print("m1 Matrix")
    m1 = Matrix(5,5, 'r')
    m1.mPrint()
    print("\nm2 Matrix")
    m2 = Matrix(5,5,'r')
    m2.mPrint()
    # print(m1)
    print("\nm1 add m2 Matrix")
    m = m1 + m2 # (m1).__add()__(m2)
    m.mPrint()
    print("\nm1 sub m2 Matrix")
    m = m1 - m2# (m1).__sub()__(m2)
    m.mPrint()
    print("\nm1 mul m2 Matrix")
    m = m1 * m2# (m1).__mul()__(m2)
    m.mPrint()
    print("\nm Transepose")
    print(*m.transpose(), sep="\n")
def useEightQueens():...
def useTicTacToe():...

def main():
    useMatrix()
    #useEightQueens()
    #useTicTacToe()
if __name__ == '__main__':
    main()
```

Results/Output

Insert pictures for the output of the programs written for this task

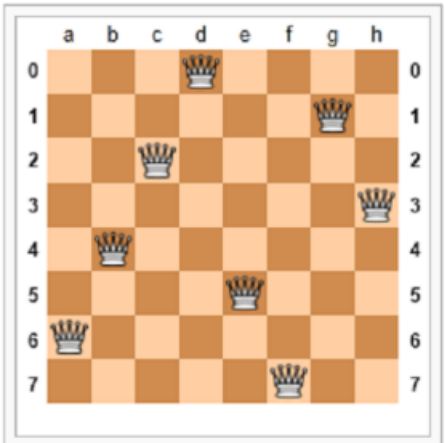
Lab02Test

```
Matrix initializer and string __str__
[7, 3, 4]
[4, 8, 10]
[5, 3, 1]
Matrix initializer and representaion __repr__()
Mymatrix is that
[1, 7, 2]
[6, 5, 2]
[8, 9, 6]
m1 Matrix
[5, 7, 1, 3, 6]
[4, 7, 2, 4, 6]
[5, 3, 1, 7, 10]
[9, 7, 5, 1, 1]
[10, 9, 7, 10, 5]

m2 Matrix
[10, 6, 6, 5, 6]
[6, 1, 9, 3, 6]
[4, 7, 9, 6, 5]
[8, 3, 4, 10, 10]
[3, 8, 1, 7, 6]

m1 add m2 Matrix
add two Matrix!
[15, 13, 7, 8, 12]
[10, 8, 11, 7, 12]
[9, 10, 10, 13, 15]
[17, 10, 9, 11, 11]
[13, 17, 8, 17, 11]

m1 sub m2 Matrix
sub two Matrix!
[-5, 1, -5, -2, 0]
[-2, 6, -7, 1, 0]
[1, -4, -8, 1, 5]
[1, 4, 1, -9, -9]
[7, 1, 6, 3, -1]
```

```
m1 mul m2 Matrix
mul two Matrix!
[18, 48, 6, 42, 36]
[18, 48, 6, 42, 36]
[30, 80, 10, 70, 60]
[3, 8, 1, 7, 6]
[15, 40, 5, 35, 30]

m Transepose
[18, 18, 30, 3, 15]
[48, 48, 80, 8, 40]
[6, 6, 10, 1, 5]
[42, 42, 70, 7, 35]
[36, 36, 60, 6, 30]
```

2. Task-2: EightQueens Data Type

- The eight-queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.
- A basic iterative algorithm starts by initially place the eight queens at random on the board subject to the constraint that there is only one queen on each row and column



One solution to the eight queens puzzle

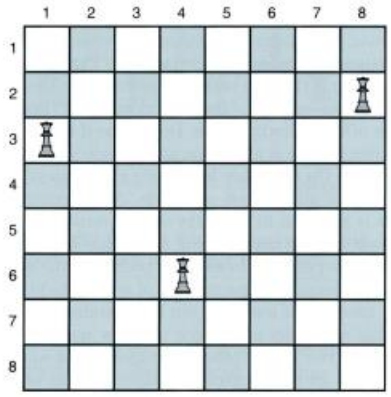
Check! if two queens are attacking

– Condition-1 (check two queens are in the same column). let col(i) be the column where the queen in the i_{th} row is located, then to check whether the queen in the k_{th} row is in the same column

$$col\,[i] = col\,[k]$$

– Condition-2 (check two queens are in the same diagonal)

$$|col\,(i) - col\,(k)| = |i - k|$$



Write a class/Type/Data Type to solve this puzzle

EightQueens
- rnb : random.Random() - int bd : list
+ def __init__() : void + def runEQ(nos) : void + def has_clashes() : bool + def col_clashes(k) : bool + def dclashes(x0, y0, x1, y1) : bool

Lab02 Code

```
78  class EightQueens:
79      rnb = random.Random()
80      def __init__(self):
81          self.bd = list(range(8))
82      def runEQ(self, nos):
83          found = 0
84          tries = 0
85          while found<nos:
86              while found < nos:
87                  EightQueens.rnb.shuffle(self.bd)
88                  if not self.has_clashes():
89                      found +=1
90                      print(f"solution {found}. {self.bd}, {tries}")
91                  tries+=1
92      def has_clashes(self):
93          for col in range(1, len(self.bd)):
94              if self.col_clashes(col):
95                  return True
96              else:
97                  return False
98      def col_clashes(self, k):
99          for i in range(k):
100             if self.dclashes(i, self.bd[i], k, self.bd[k]):
101                 return True
102         return False
103
104     def dclashes(self,x0, y0, x1, y1):
105         d1 = abs(x0-y0)
106         d2 = abs(x1-y1)
107         return d1 == d2
108
```

Lab02Test Code

```
1   from Lab02 import Matrix, EightQueens, TicTacToe
2   def useMatrix():...
25  def useEightQueens():
26      e1 = EightQueens()
27      e1.runEQ(10)
28  def useTicTacToe():...
30
31  def main():
32      #useMatrix()
33      useEightQueens()
34      #useTicTacToe()
35  if __name__ == '__main__':
36      main()
```

Results/Output

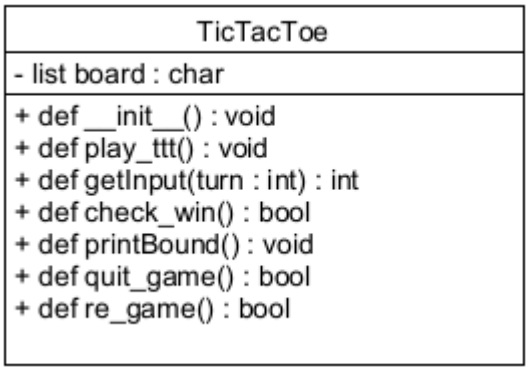
Insert pictures for the output of the programs written for this task

```
Lab02Test
C:\Users\iqeq1\anaconda3\envs\datamining\python.exe "C:\4-2\Data Structure\Lab02\Lab02Test.py"
solution 1. [5, 3, 6, 4, 2, 0, 1, 7], 0
solution 2. [3, 1, 7, 0, 5, 6, 4, 2], 1
solution 3. [5, 2, 6, 4, 7, 3, 0, 1], 3
solution 4. [1, 7, 3, 6, 0, 5, 2, 4], 4
solution 5. [3, 5, 4, 7, 6, 2, 0, 1], 5
solution 6. [2, 6, 7, 0, 3, 1, 5, 4], 6
solution 7. [4, 2, 5, 0, 7, 3, 1, 6], 7
solution 8. [5, 4, 1, 7, 3, 0, 2, 6], 8
solution 9. [0, 2, 6, 7, 4, 5, 3, 1], 9
solution 10. [0, 4, 3, 1, 7, 2, 6, 5], 10

종료 코드 0(으)로 완료된 프로세스
```

3. **Task-3: TicTacToe Data Type**

   Tic-tac-toe is a paper-and-pencil game for two players who take turns marking the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

   Write a class/Type/Data Type to play this game



Lab02 Code

```
class TicTacToe:
    def __init__(self):
        self.board = []
        for i in range(9):
            self.board.append('-')

    def play_ttt(self):
        win = False
        move = 0
        while not win:
            self.printBound()
            if move % 2 == 0:
                turn = 'X'
            else:
                turn = 'O'
            print(f"Turn for player '{turn}'")

            user = self.getInput(turn)
            while self.board[user] != '-':
                print("Invalid Input!")
                user = self.getInput(turn)

            self.board[user] = 'O' if turn == 'O' else 'X'
            move +=1
            if move > 3:
                winner = self.check_win()
                if winner:
                    print(f"winner is '{'X' if winner == 'X' else 'O'}'👏👏👏")
                    win = True
            if self.quit_game():
                break
        self.printBound()
        if win:
            print(f"Retry?(Y or n)")
            self.re_game()
        else:
            print(f"Oh~ Draw~!\nRetry?(Y or n)")
            self.re_game()

```

```
    def getInput(self, turn):
        n = int(input(f"{turn}'s turn! Input Your Number"))
        return n

    def check_win(self):
        win_cord = ((1,2,3), (4,5,6),(7,8,9), (1,4,7), (2,5,8), (3,6,9), (1,5,9), (3,5,7) )
        for each in win_cord:
            if self.board[each[0]-1] == self.board[each[1]-1] == self.board[each[2]-1] and self.board[each[0]-1] != '-':
                return self.board[each[0]-1]
        return False

    def printBound(self):
        print(self.board[:3], self.board[3:6], self.board[6:], sep="\n")

    def quit_game(self):
        return '-' not in self.board
    def re_game(self):
        while True:
            n = input()
            if n == 'Y':
                print("Okay! retry~!!")
                for i in range(9):
                    self.board[i]='-'
                self.play_ttt()
                break
            if n =='n':
                print("Okay! See You Later~!!")
                break
            print("Your input is wrong!!\nRetry?(Y or n)")
```

Lab02Test Code

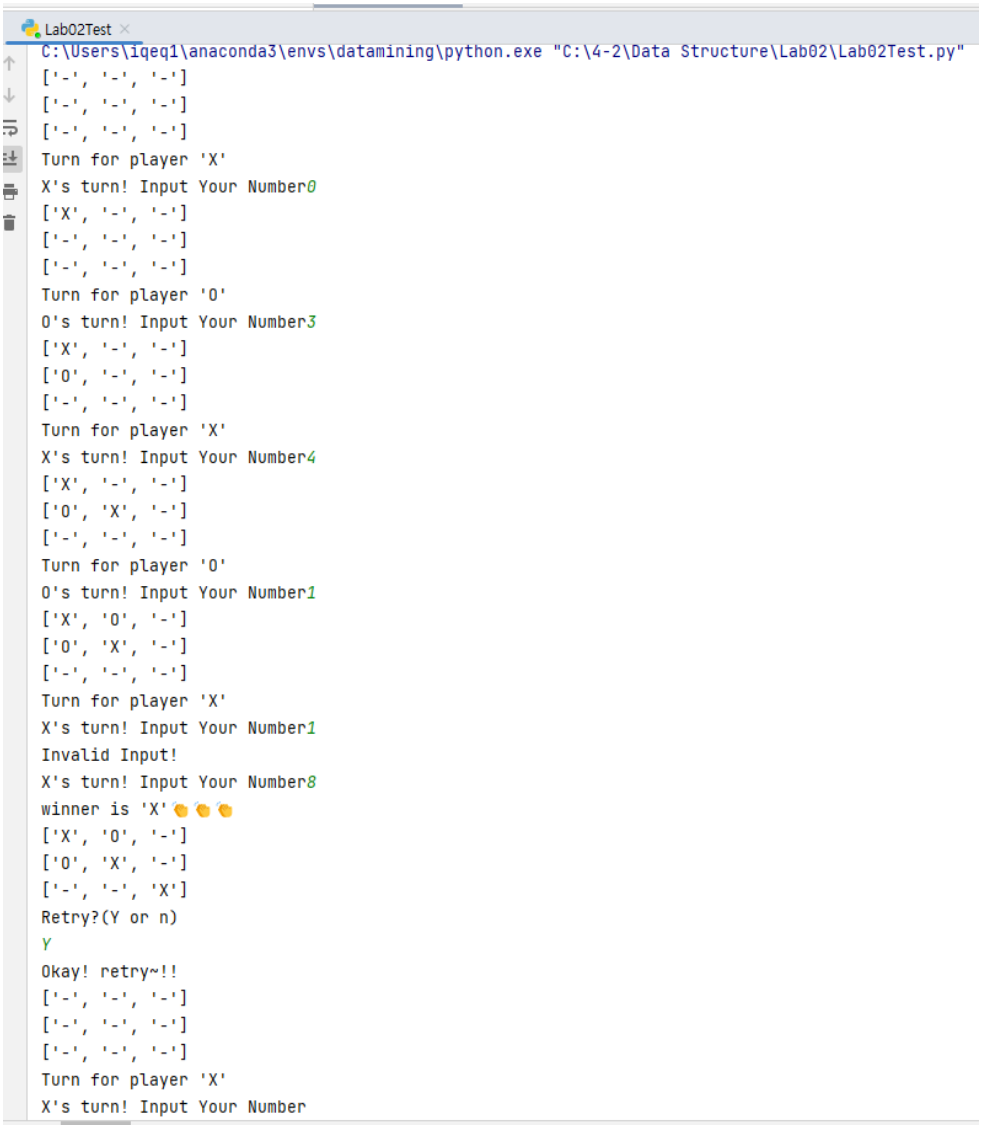
Lab02Test.py | Lab02.py

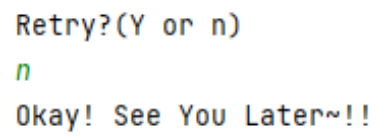
```
1   from Lab02 import Matrix, EightQueens, TicTacToe
2   def useMatrix():...
25  def useEightQueens():...
28  def useTicTacToe():
29      t1 = TicTacToe().play_ttt()
30
31  def main():
32      #useMatrix()
33      #useEightQueens()
34      useTicTacToe()
35  if __name__ == '__main__':
36      main()
```

Results/Output

Input range : 0 ~ 8

End game => Can select Retry by 'Y' or 'n'





And if Draw, print like this

```
['X', 'O', 'X']
['X', 'X', 'O']
['O', 'X', 'O']
Oh~ Draw~!
Retry?(Y or n)
```

4. Conclusion

Conclude the Lab. Write your views about it , i.e. what have you learnt from this lab? It was helpful or difficult etc