Name:	**Ms. Momin Sameera Salim**
Roll No.:	**34**
Student ID:	**IT-7769**
Class:	T.Y. B.Sc.(I.T.)
Semester:	**V**
Academic Year:	2023-2024
Subject:	USIT P : Artificial Intelligence
Professor In-Charge:	
Department:	Department of Information Technology
Incharge:	Ms. Misbah Momin

K. M. E. Society's

G. M. Momin Women's College

(Affiliated to the University of Mumbai, Re- Accredited by NAAC with 'B++' Grade, Recipient of Best College Award, ISO 9001:2018 Certified, Selected for Star College Scheme of DBT, Ministry ofScience & Technology, Supported under RUSA 2.0, Recipient of FIST 'O' Level Grant from DST, Govt. of India, Winner of BEQET Award)

CERTIFICAE

This is to certify that Ms. Momin Sameera Salim ,Student ID 7769 of T.Y. B.Sc.(I.T.) has completed the practicalwork in the subject of **"Artificial Intelligence "**in **Semester v** during the academic year 2023-2024 under the guidance of **Prof. Mis Naba Momin Maam**, being the partial requirement for the fulfillment of the curriculum of Degree of Bachelor of Science in Information Technology, University of Mumbai.

Date: _____

Signature of Internal Examiner	Signature of I/C	Signature of External Examiner

Practical 0: Prolog Basics and Installations

Features:

1. It is a Logic Programming Language
2. It is a Declarative Programming Language
3. Database consist of facts and rules consist of relations
4. Computing is carried out by running query over therelations

Facts:

football(ronaldo).

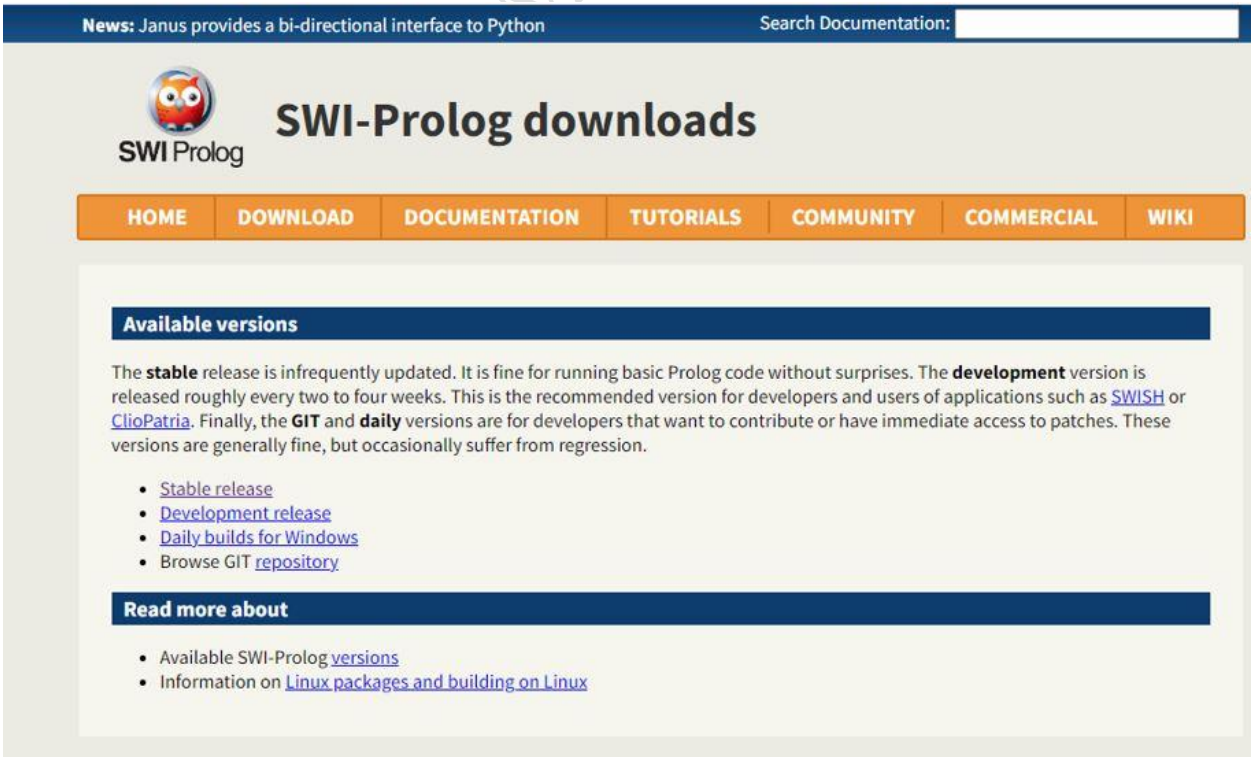
red(rose).

father(john,bob).

Rules:

mother(X,Y):-parent(X,Y),female(X)

Installation:



Steps:

1. First open the SWI prolog and go to the file menu and take file and save it as filename.p1.
2. New window opened write your code on that window and click on consult option to make.
3. Then go to your first window and click on file and select option consult and select your filename and click open.
4. Lastly pass your queries on that window.

Practical 1:Simple Predicates

Facts:

batsman(Williamson).

bowler(boult).

Keeper(marsh).

Rules:

Cricketer(X):-batsman(X);bowler(X);Keeper(X).

Queries:

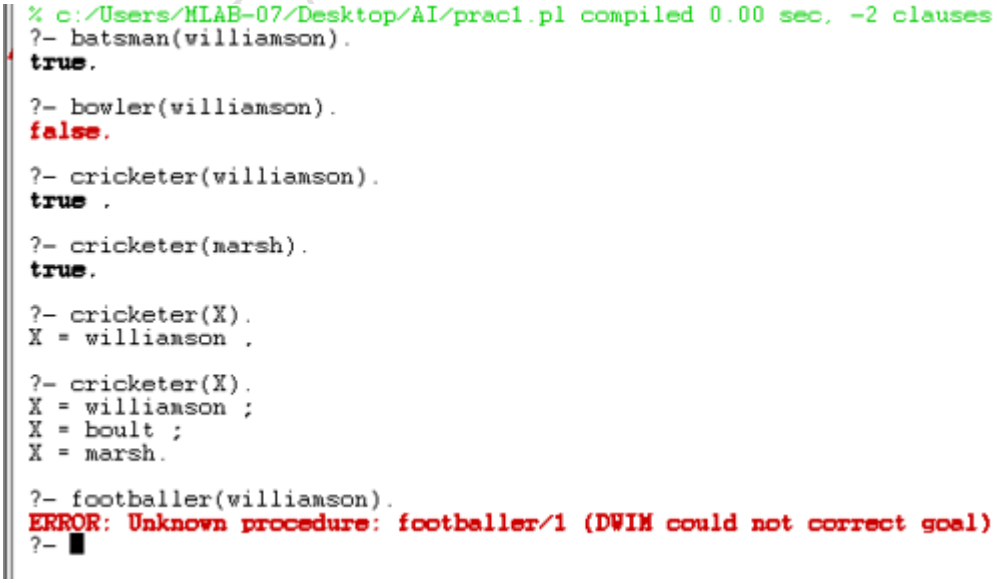
batsman(X).

bowler(X).

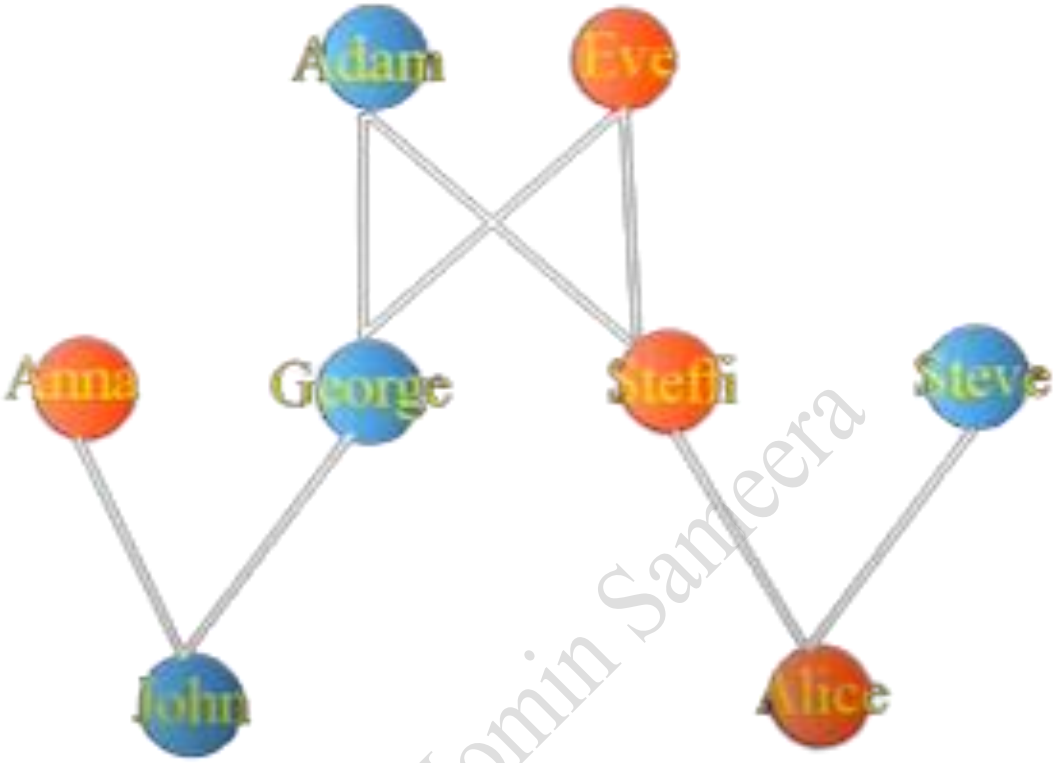
Keeper(X).

Cricketer(X).

Output:



Practical 2:Family Tree



Facts:

male(adam).

male(george).

male(steve).

male(john).

female(alice).

female(anna).

female(eve).

female(steffi).

```
parent(adam,george).
parent(eve,george).
parent(adam,steffi).
parent(eve,steffi).
parent(george,john).
parent(anna,john).
parent(steffi,alice).
parent(steve,alice).
```

Rules:

```
mother(X,Y) :- parent(X,Y), female(X).
father(X,Y) :- parent(X,Y), male(X).

sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X),X\==Y.
brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X),X\==Y.

grandfather(X,Z) :- father(X,Y), parent(Y,Z).
grandmother(X,Z) :- mother(X,Y), parent(Y,Z).

siblings(X,Y) :- (brother(X,Y);sister(X,Y)),X\==Y.

uncle(X,Y) :- parent(Z,Y),brother(X,Z).
aunty(X,Y) :- parent(Z,Y),sister(X,Z).

cousin(X,Y) :- parent(A,X),parent(B,Y),siblings(A,B).
```

Queries:

1.grandfather(X,Y).

2.grandmother(X,Y).

3.father(X,Y).

4.cousin(X,Y).

5.uncle(X,Y).

6.father(adam,_).

7.father(adam,X).

Output:

SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/lenovo/Documents/Prolog/family-tree.pl compiled 0.00 sec, -2 clauses
?- grandfather(X,Y).
X = adam,
Y = john ,

?- grandmother(X,Y).
X = eve,
Y = john ,

?- father(X,Y).
X = adam,
Y = george ,

?- father(X,Y).
X = adam,
Y = george ;
X = adam,
Y = steffi ;
X = george,
Y = john ;
X = steve,
Y = alice.

?- cousin(X,Y).
X = john,
Y = alice ;
X = john,
Y = alice ;
X = alice,
Y = john ;
X = alice,
Y = john
```

Practical 3:Tower of Hanoi Problem

Facts:

```
move(1,X,Y,_):-
    write("Move top disk from"), write(X), write("to"), write(Y), nl.
move(N,X,Y,Z):-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Query:

Move(3,source,destination,intermediate).

Output:

SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/lenovo/Documents/Prolog/3.pl compiled 0.00 sec, 2 clauses
?- move(3,source,destination,intermediate).
Move top disk fromsourcetodestination
Move top disk fromsourcetointermediate
Move top disk fromdestinationtointermediate
Move top disk fromsourcetodestination
Move top disk fromintermediatetosource
Move top disk fromintermediatetodestination
Move top disk fromsourcetodestination
true
```

Practical:4 Constraint and Satisfaction Problem

Facts:

```
Coloring(WA,SA,NT,QU,NSW,VI) :-
different(WA,SA),
different(WA,NT),
different(NT,SA),
different(NT,QU),
different(SA,QU),
different(SA,NSW),
different(SA,VI),
different(QU,NSW),
different(NSW,VI).

different(red,blue).
different(blue,red).
different(blue,green).
different(green,red).
different(green, blue).
different(red,green).
```


Query:

```
coloring(WA,SA,NT,QU,NSW,VI) .
```

Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/lenovo/Documents/Prolog/4.pl compiled 0.00 sec, 7 clauses
?- coloring(WA,SA,NT,QU,NSW,VI) .
WA = QU, QU = VI, VI = red,
SA = blue,
NT = NSW, NSW = green
```

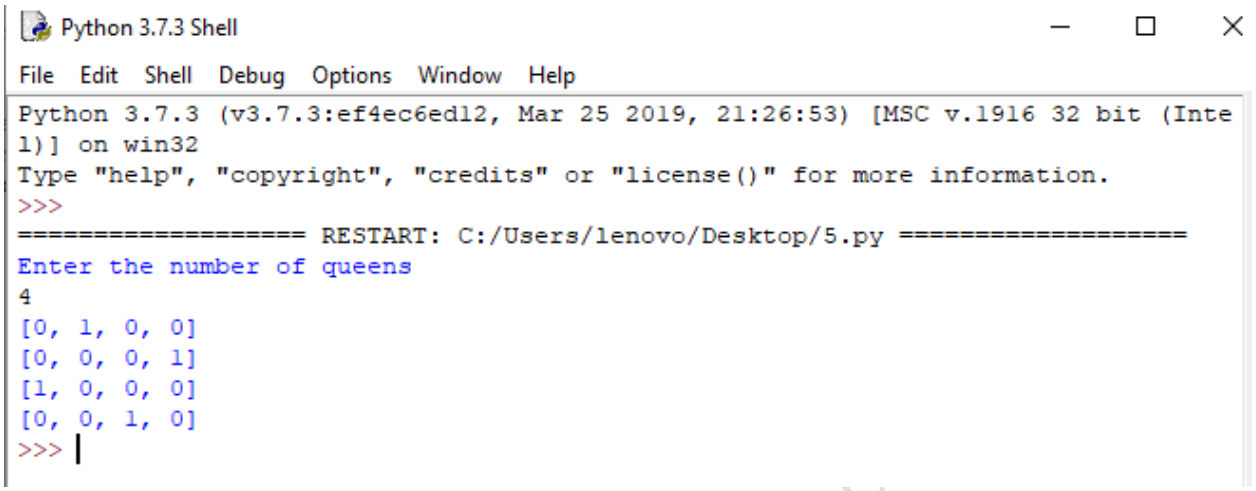
Practical :5 Four Queen Problem

Code:

```
#Number of queens
print ("Enter the number of queens")
N = int(input())


#chessboard
#NxN matrix with all elements 0
board = [[0]*N for _ in range(N)]


def is_attack(i, j):
    #checking if there is a queen in row or column
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonals
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False


def N_queen(n):
    #if n is 0, solution found
```

```
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            '''checking if we can place a queen here or not
            queen will not be placed if the place is being attacked
            or already occupied'''
            if (not(is_attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                #recursion
                #wether we can put the next queen with this arrangment or not
                if N_queen(n-1)==True:
                    return True
                board[i][j] = 0

    return False

N_queen(N)
for i in board:
    print (i)
```

Output:



Practical:6 Tic Tac Toe

Code:

```
#!/usr/bin/env python3
from math import inf as infinity
from random import choice
import platform
import time
from os import system


HUMAN = -1
COMP = +1
board = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
]


def evaluate(state):
    """
    Function to heuristic evaluation of state.
    :param state: the state of the current board
    :return: +1 if the computer wins; -1 if the human wins; 0 draw
```

```
    """
    if wins(state, COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0

    return score


def wins(state, player):
    """
    This function tests if a specific player wins. Possibilities:
    * Three rows    [X X X] or [O O O]
    * Three cols    [X X X] or [O O O]
    * Two diagonals [X X X] or [O O O]
    :param state: the state of the current board
    :param player: a human or a computer
    :return: True if the player wins
    """
    win_state = [
        [state[0][0], state[0][1], state[0][2]],
        [state[1][0], state[1][1], state[1][2]],
        [state[2][0], state[2][1], state[2][2]],
```

```
        [state[0][0], state[1][0], state[2][0]],
        [state[0][1], state[1][1], state[2][1]],
        [state[0][2], state[1][2], state[2][2]],
        [state[0][0], state[1][1], state[2][2]],
        [state[2][0], state[1][1], state[0][2]],
    ]
    if [player, player, player] in win_state:
        return True
    else:
        return False


def game_over(state):
    """
    This function test if the human or computer wins
    :param state: the state of the current board
    :return: True if the human or computer wins
    """
    return wins(state, HUMAN) or wins(state, COMP)


def empty_cells(state):
    """
    Each empty cell will be added into cells' list
    :param state: the state of the current board
```

```
    :return: a list of empty cells
    """
    cells = []

    for x, row in enumerate(state):
        for y, cell in enumerate(row):
            if cell == 0:
                cells.append([x, y])

    return cells



def valid_move(x, y):
    """
    A move is valid if the chosen cell is empty
    :param x: X coordinate
    :param y: Y coordinate
    :return: True if the board[x][y] is empty
    """
    if [x, y] in empty_cells(board):
        return True
    else:
        return False
```

```
def set_move(x, y, player):
    """
    Set the move on board, if the coordinates are valid
    :param x: X coordinate
    :param y: Y coordinate
    :param player: the current player
    """
    if valid_move(x, y):
        board[x][y] = player
        return True
    else:
        return False



def minimax(state, depth, player):
    """
    AI function that choice the best move
    :param state: current state of the board
    :param depth: node index in the tree (0 <= depth <= 9),
    but never nine in this case (see iaturn() function)
    :param player: an human or a computer
    :return: a list with [the best row, best col, best score]
    """
    if player == COMP:
        best = [-1, -1, -infinity]
```

```
    else:
        best = [-1, -1, +infinity]

    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]

    for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y

        if player == COMP:
            if score[2] > best[2]:
                best = score  # max value
        else:
            if score[2] < best[2]:
                best = score  # min value

    return best


def clean():
```

```
    """
    Clears the console
    """
    os_name = platform.system().lower()
    if 'windows' in os_name:
        system('cls')
    else:
        system('clear')


def render(state, c_choice, h_choice):
    """
    Print the board on console
    :param state: current state of the board
    """

    chars = {
        -1: h_choice,
        +1: c_choice,
        0: ' '
    }
    str_line = '---------------'

    print('\n' + str_line)
    for row in state:
```

```
        for cell in row:
            symbol = chars[cell]
            print(f'| {symbol} |', end='')
        print('\n' + str_line)


def ai_turn(c_choice, h_choice):
    """
    It calls the minimax function if the depth < 9,
    else it choices a random coordinate.
    :param c_choice: computer's choice X or O
    :param h_choice: human's choice X or O
    :return:
    """
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    clean()
    print(f'Computer turn [{c_choice}]')
    render(board, c_choice, h_choice)

    if depth == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
```

```
    else:
        move = minimax(board, depth, COMP)
        x, y = move[0], move[1]

    set_move(x, y, COMP)
    time.sleep(1)


def human_turn(c_choice, h_choice):
    """
    The Human plays choosing a valid move.
    :param c_choice: computer's choice X or O
    :param h_choice: human's choice X or O
    :return:
    """
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    # Dictionary of valid moves
    move = -1
    moves = {
        1: [0, 0], 2: [0, 1], 3: [0, 2],
        4: [1, 0], 5: [1, 1], 6: [1, 2],
        7: [2, 0], 8: [2, 1], 9: [2, 2],
```

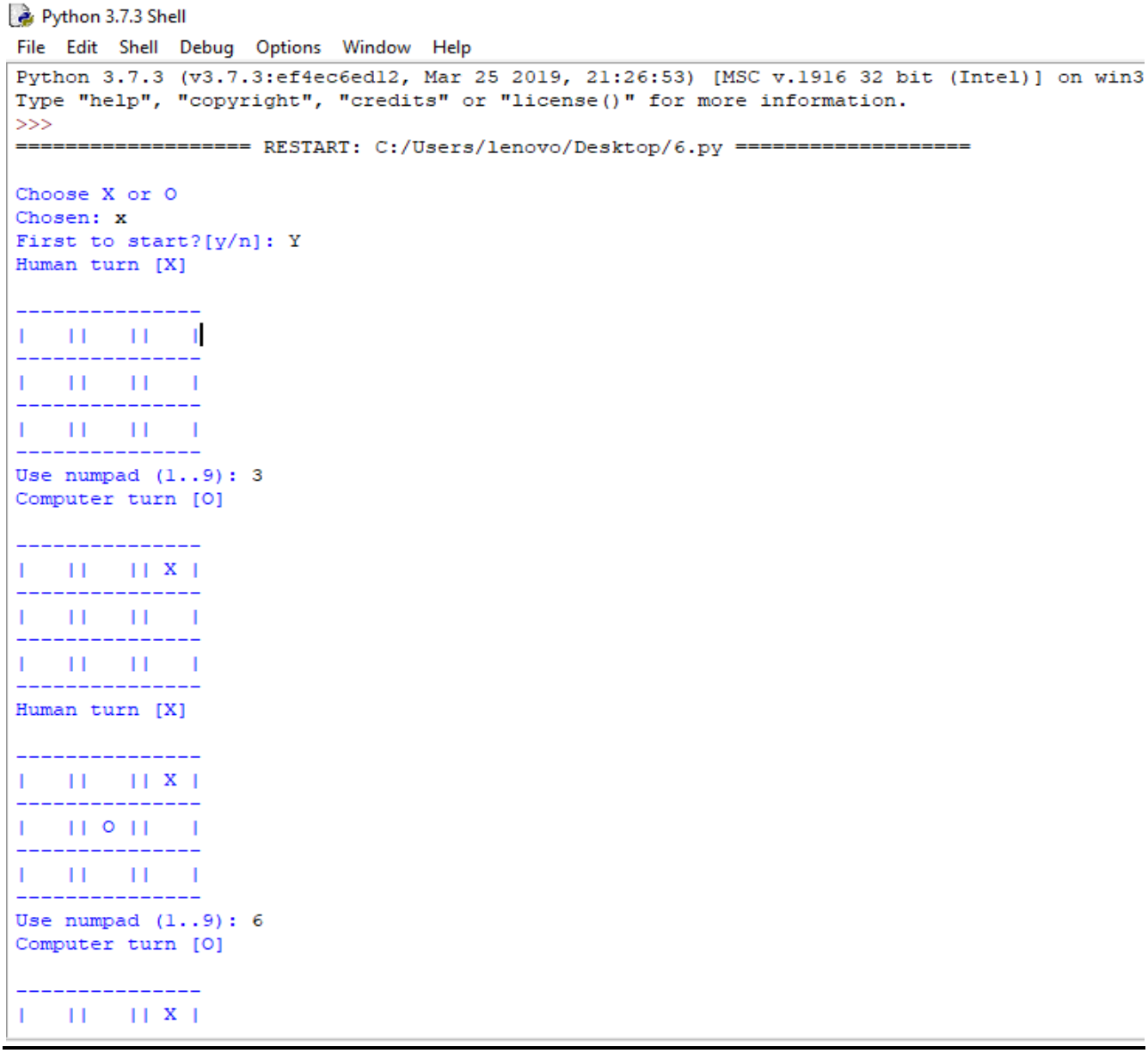
```
    }

    clean()
    print(f'Human turn [{h_choice}]')
    render(board, c_choice, h_choice)

    while move < 1 or move > 9:
        try:
            move = int(input('Use numpad (1..9): '))
            coord = moves[move]
            can_move = set_move(coord[0], coord[1], HUMAN)

            if not can_move:
                print('Bad move')
                move = -1
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')


def main():
    """
    Main function that calls all functions
```

```
"""
clean()
h_choice = ''  # X or O
c_choice = ''  # X or O
first = ''  # if human is the first

# Human chooses X or O to play
while h_choice != 'O' and h_choice != 'X':
    try:
        print('')
        h_choice = input('Choose X or O\nChosen: ').upper()
    except (EOFError, KeyboardInterrupt):
        print('Bye')
        exit()
    except (KeyError, ValueError):
        print('Bad choice')

# Setting computer's choice
if h_choice == 'X':
    c_choice = 'O'
else:
    c_choice = 'X'

# Human may starts first
clean()
```

```
while first != 'Y' and first != 'N':
    try:
        first = input('First to start?[y/n]: ').upper()
    except (EOFError, KeyboardInterrupt):
        print('Bye')
        exit()
    except (KeyError, ValueError):
        print('Bad choice')

# Main loop of this game
while len(empty_cells(board)) > 0 and not game_over(board):
    if first == 'N':
        ai_turn(c_choice, h_choice)
        first = ''

    human_turn(c_choice, h_choice)
    ai_turn(c_choice, h_choice)

# Game over message
if wins(board, HUMAN):
    clean()
    print(f'Human turn [{h_choice}]')
    render(board, c_choice, h_choice)
    print('YOU WIN!')
elif wins(board, COMP):
```

```
            clean()
            print(f'Computer turn [{c_choice}]')
            render(board, c_choice, h_choice)
            print('YOU LOSE!')
        else:
            clean()
            render(board, c_choice, h_choice)
            print('DRAW!')

        exit()


if __name__ == '__main__':
    main()
```

Output:


Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/lenovo/Desktop/6.py ==========
Choose X or O
Chosen: x
First to start?[y/n]: Y
Human turn [X]
Use numpad (1..9): 3
Computer turn [O]
				X
Human turn [X]
				X
		O		
Use numpad (1..9): 6
Computer turn [O]
				X

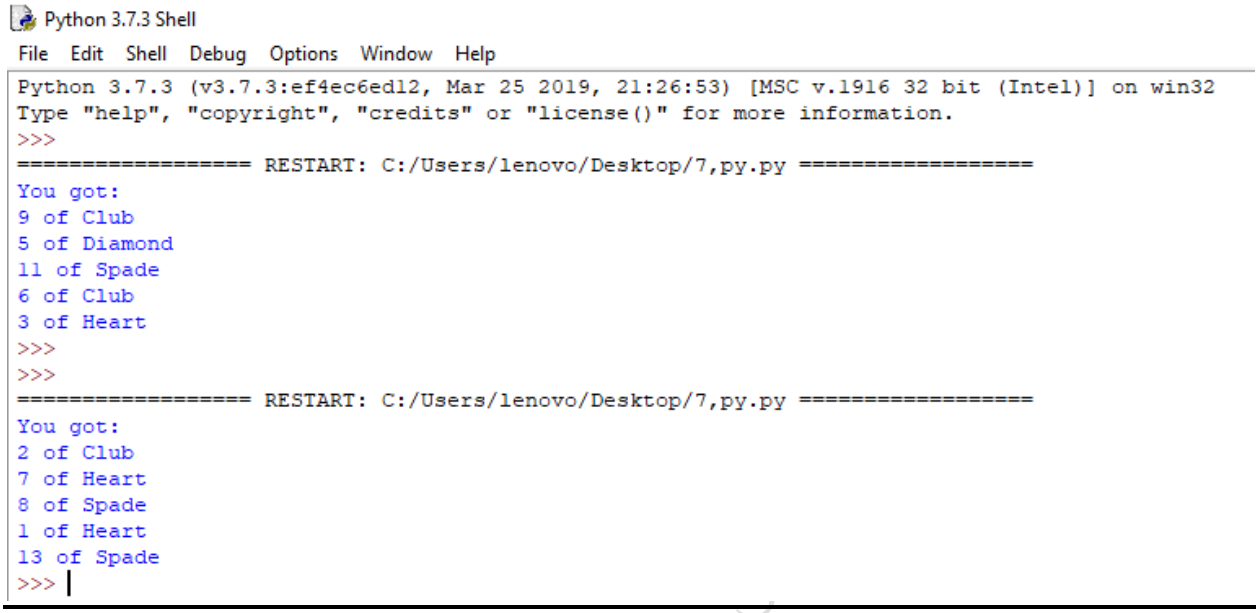
```
|    ||    ||    |
---------------
Use numpad (1..9): 6
Computer turn [O]

---------------
|    ||    || X |
---------------
|    || O || X |
---------------
|    ||    ||    |
---------------
Human turn [X]

---------------
|    ||    || X |
---------------
|    || O || X |
---------------
|    ||    || O |
---------------
Use numpad (1..9): 4
Computer turn [O]

---------------
|    ||    || X |
---------------
| X || O || X |
---------------
|    ||    || O |
---------------
Computer turn [O]

---------------
| O ||    || X |
---------------
| X || O || X |
---------------
|    ||    || O |
---------------
YOU LOSE!
>>>
```

Practical:7 Program To Shuffle Deck of Cards

Code:

```
# Python program to shuffle a deck of card

# importing modules
import itertools, random

# make a deck of cards
deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))

# shuffle the cards
random.shuffle(deck)

# draw five cards
print("You got:")
for i in range(5):
   print(deck[i][0], "of", deck[i][1])
```

Output:



Practical:8 Depth First Search(DFS) algorithm

Code:

```
def dfs(graph,start,visited=None):
    if visited is None:
        visited = set()
    visited.add(start)


    print(start)
    for next in graph[start] - visited:
        dfs(graph,next,visited)
    return visited


graph = {'0':set(['1', '2']),
         '1':set(['0','3','4']),
         '2':set(['0']),
         '3':set(['1']),
         '4':set(['2','3'])}
dfs(graph,'0')
```

Output: