Page:
Date: / /

Experiment : 01

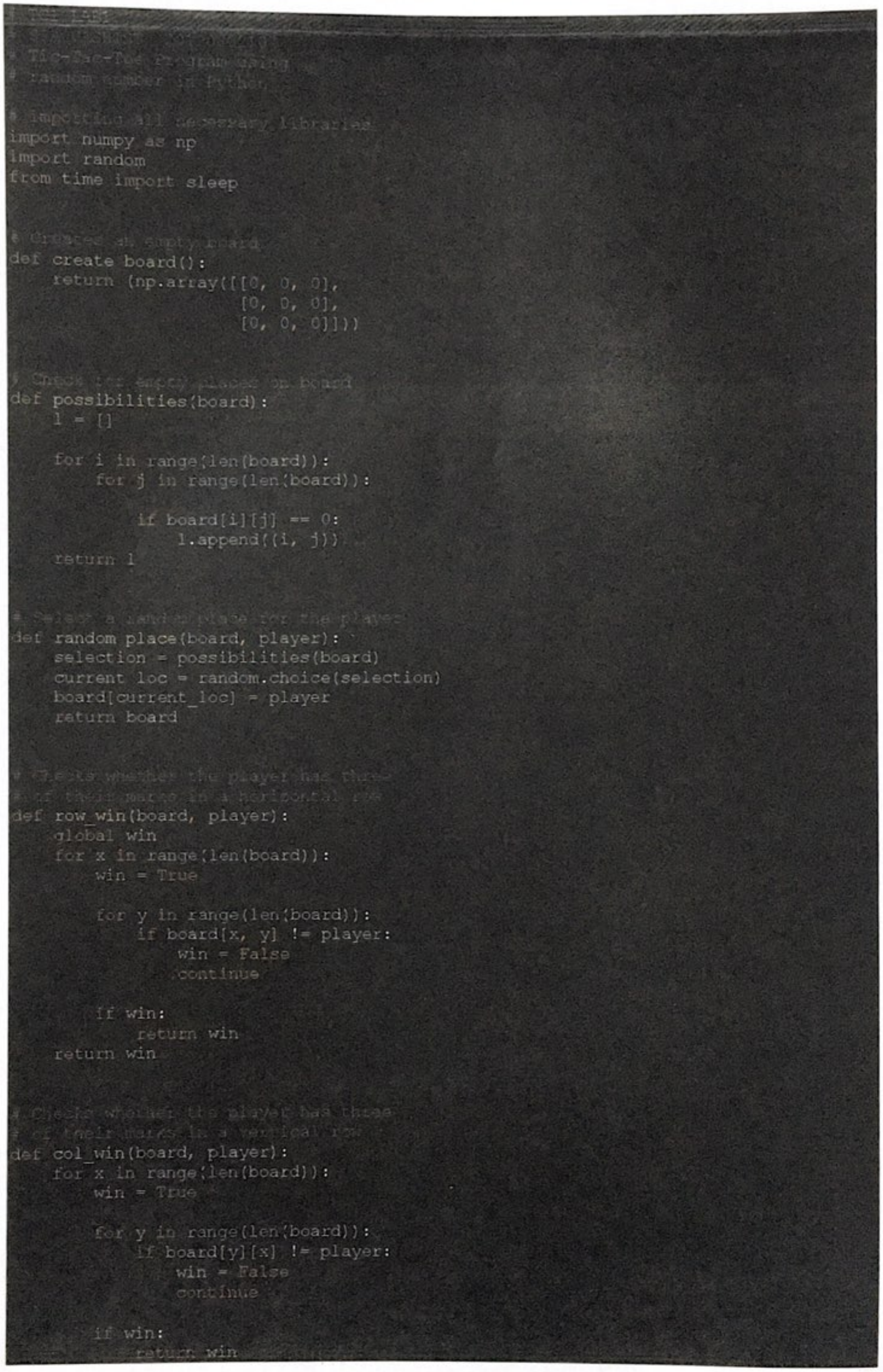
- Aim :- Python Implementation of automatic Tic Tac Toe game using random number.

- Libraries used:- Numpy, Random

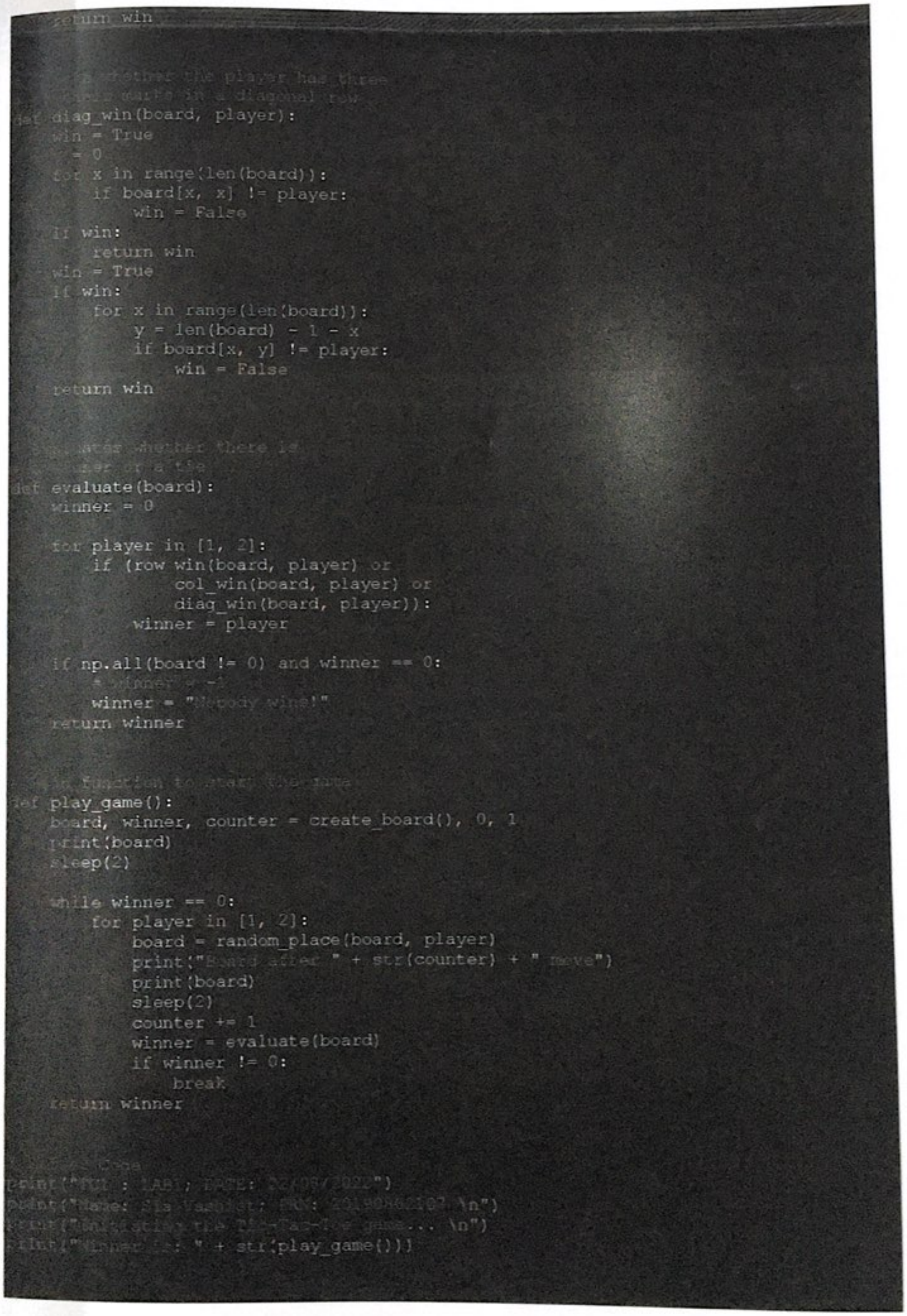
- Theory:-
- Algorithm:-

1. Create a board using 2-dimensional arrays and initialize each element as empty.
~~2~~ ↳ Creates a 9 x 9 board and initializes with 0.
2. For each player (1 or 2) calls the random place function to randomly choose a location on board and marks that location with the player number, alternatively.
3 Print the board after each move.
4 Evaluate the board after each move to check whether a row or column or diagonal has the same player number (ie. X or O).
If so, display the winner name. If after 9 moves, there are no players then display -1.

* Functions:-
1) shows the board multiple times while they are playing.
2) start game:
↳ shows the user to select the spot for next move.
↳ Asks the user to input the move.

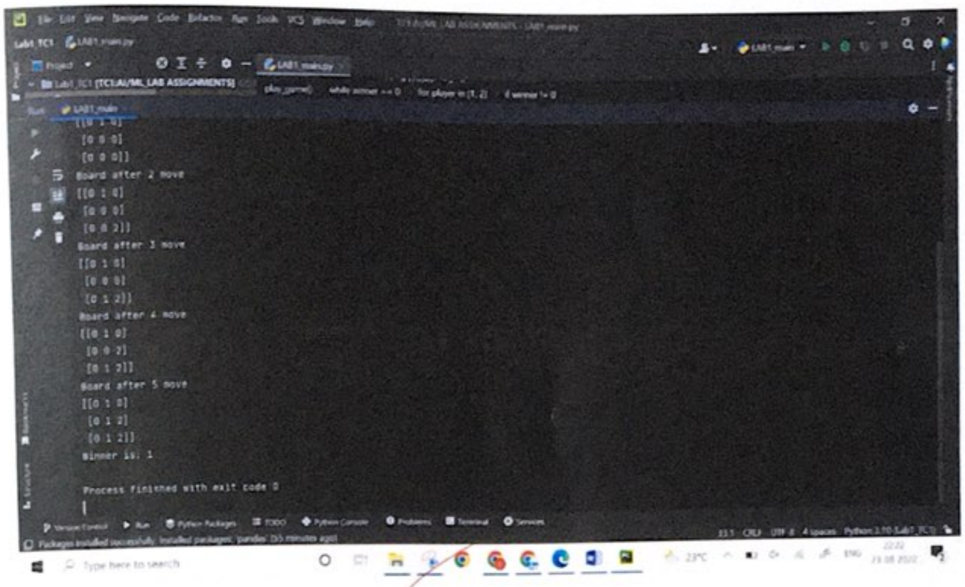


```
# TC1 LAB1
# [illegible]
# Tic-Tac-Toe Program using
# random number in Python

# importing all necessary libraries
import numpy as np
import random
from time import sleep


# Creates an empty board
def create board():
    return (np.array([[0, 0, 0],
                      [0, 0, 0],
                      [0, 0, 0]]))


# Check for empty places on board
def possibilities(board):
    l = []

    for i in range(len(board)):
        for j in range(len(board)):

            if board[i][j] == 0:
                l.append((i, j))
    return l


# Select a random place for the player
def random place(board, player):
    selection = possibilities(board)
    current loc = random.choice(selection)
    board[current_loc] = player
    return board


# Checks whether the player has three
# of their marks in a horizontal row
def row_win(board, player):
    global win
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[x, y] != player:
                win = False
                continue

        if win:
            return win
    return win


# Checks whether the player has three
# of their marks in a vertical row
def col_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue

        if win:
            return win
```



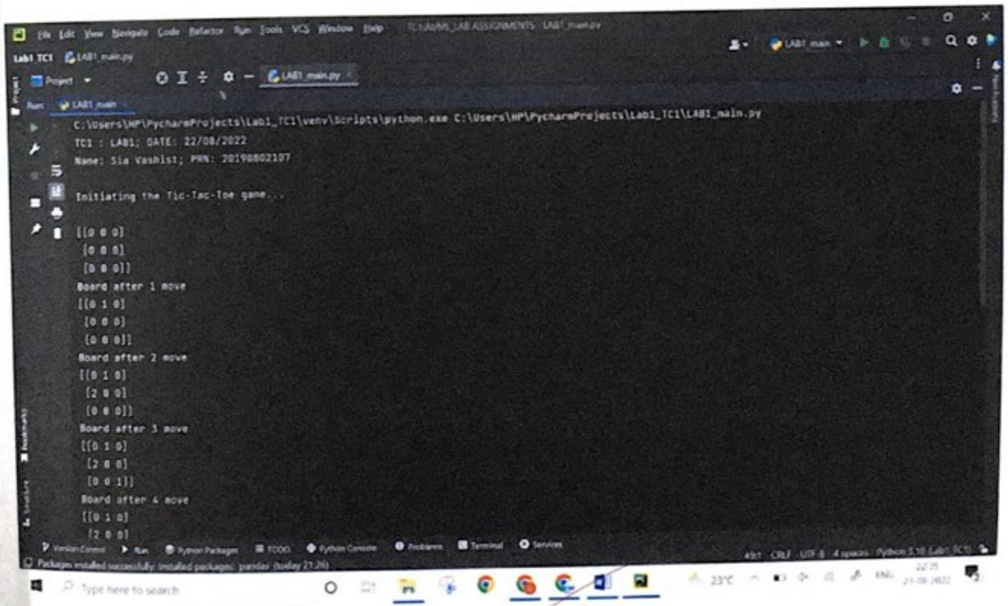
```
    return win


# Checks whether the player has three
# of their marks in a diagonal row
def diag_win(board, player):
    win = True
    y = 0
    for x in range(len(board)):
        if board[x, x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win


# Evaluates whether there is
# a winner or a tie
def evaluate(board):
    winner = 0

    for player in [1, 2]:
        if (row win(board, player) or
                col_win(board, player) or
                diag_win(board, player)):
            winner = player

    if np.all(board != 0) and winner == 0:
        # winner = -1
        winner = "Nobody wins!"
    return winner


# Main function to start the game
def play_game():
    board, winner, counter = create_board(), 0, 1
    print(board)
    sleep(2)

    while winner == 0:
        for player in [1, 2]:
            board = random_place(board, player)
            print("Board after " + str(counter) + " move")
            print(board)
            sleep(2)
            counter += 1
            winner = evaluate(board)
            if winner != 0:
                break
    return winner


# Driver Code
print("TC1 : LAB1 ; DATE: 22/08/2022")
print("Name: Sia Vashist; PRN: 20190802107 \n")
print("Initiating the Tic-Tac-Toe game... \n")
print("Winner is: " + str(play_game()))
```

- **Output Screenshots:**
    - **<u>Case 1</u>:** Either 1 or 2 will be a winner



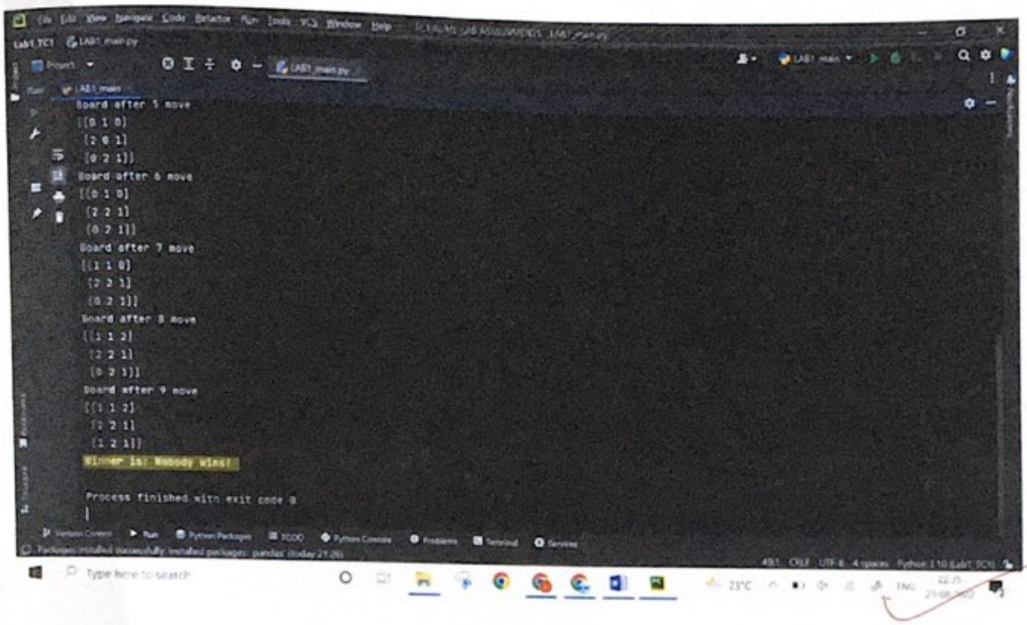
```
[[0 1 0]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 1 0]
 [0 0 0]
 [0 0 2]]
Board after 3 move
[[0 1 0]
 [0 0 0]
 [0 1 2]]
Board after 4 move
[[0 1 0]
 [0 0 2]
 [0 1 2]]
Board after 5 move
[[0 1 0]
 [0 1 2]
 [0 1 2]]
Winner is: 1

Process finished with exit code 0
```

- **<u>Case 2:</u>** If after 9 moves, there are no winner. Nobody would win, it'll be a tie.



```
C:\Users\HP\PycharmProjects\Lab1_TC1\venv\Scripts\python.exe C:\Users\HP\PycharmProjects\Lab1_TC1\LAB1_main.py
TC1 : LAB1; DATE: 22/08/2022
Name: Sia Vashist; PRN: 20190802107

Initiating the Tic-Tac-Toe game...

[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 1 0]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 1 0]
 [2 0 0]
 [0 0 0]]
Board after 3 move
[[0 1 0]
 [2 0 0]
 [0 0 1]]
Board after 4 move
[[0 1 0]
 [2 0 0]
```

Lab report: TC1- AI/ML



```
Board after 5 move
[[0 1 0]
 [2 0 1]
 [0 2 1]]
Board after 6 move
[[0 1 0]
 [2 2 1]
 [0 2 1]]
Board after 7 move
[[1 1 0]
 [2 2 1]
 [0 2 1]]
Board after 8 move
[[1 1 2]
 [2 2 1]
 [0 2 1]]
Board after 9 move
[[1 1 2]
 [2 2 1]
 [1 2 1]]
Winner is: Nobody wins!

Process finished with exit code 0
```

Packages installed successfully: Installed packages: 'pandas' (today 21:26)

28/09/22

↳ Updates the spot with respective player.

↳ Checks if the current player has won or not.

↳ If one of the player has won, it returns a winning message and breaks the infinite loop.

↳ It then checks ~~the~~, if the board is filled or not.

↳ If the board is filled -with no winners then it prints the draw message and breaks the infinite loop.

3) Finally, shows the user the final view of the board.

* Conclusion :- Intelligence, can be a property of any purpose- driven decision maker. An algorithm of playing Tic Tac Toe game has been presented and tested that works in an efficient way. Overall the program runs without any errors.

28/09/22

Experiment : 03

- Aim :- Write a program to implement Water Jug Problem

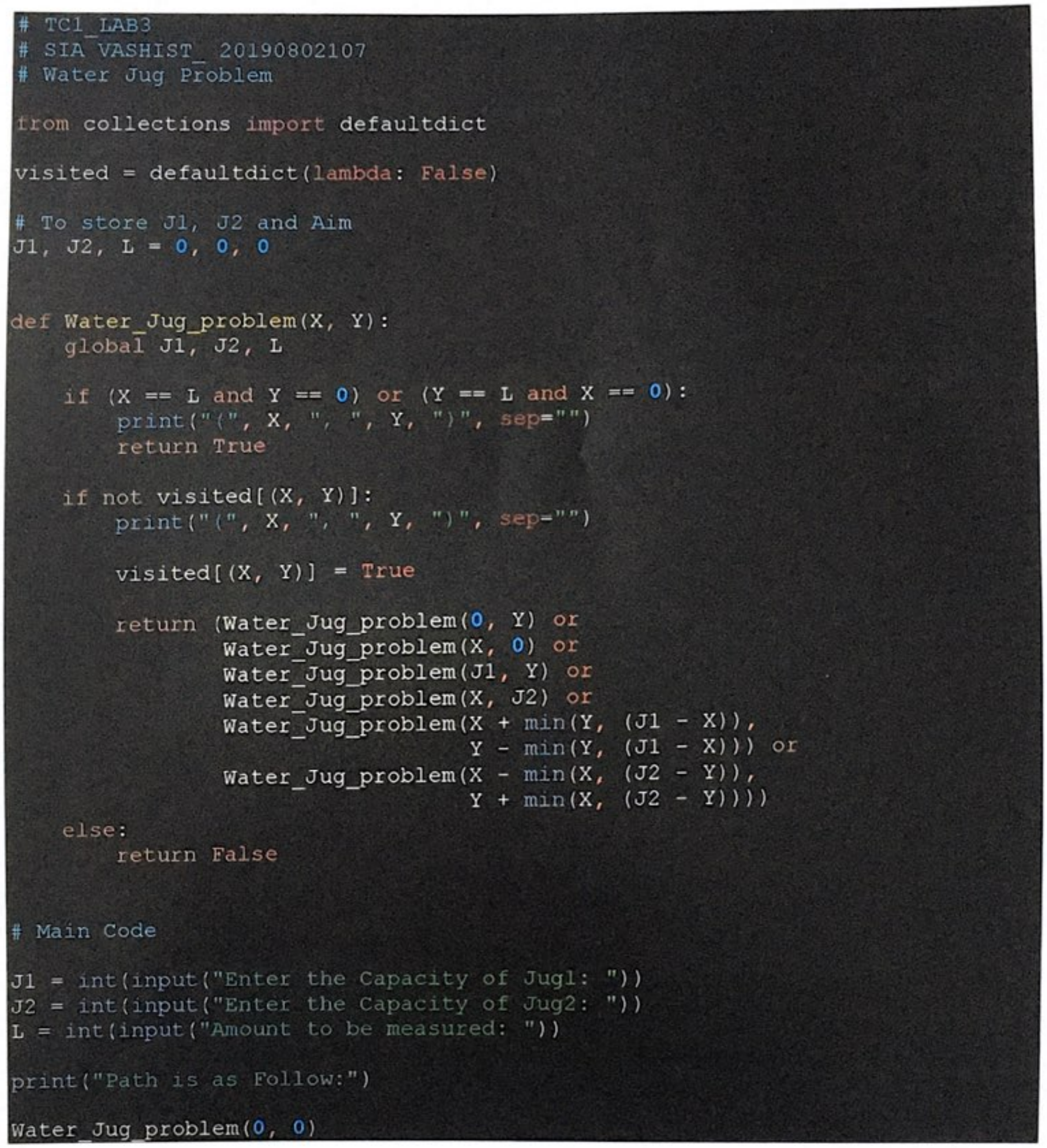
- Problem Description: You are given two jugs, a 4-litre & a 3-litre one. Neither has any measuring marker on it. Write a program to get 2 liters of water into either one of them.

↳ Operations that can be performed:-
- Fill any of the jugs completely with water.
- Pour water from one - jug to another until one of jugs is either empty / full.
  $(X,Y) \rightarrow (X-d, Y+d)$.
- Empty any of the jugs.

- For example, $X=4$, $Y=3$, $Z=2$
  output = {(0,0), (0,3), (3,0), (3,3), (4,2), (0,2)}

Explanation:-
- Fill the 4 litre jug completely.
- Empty water from 4L jug to 3L (4= 1L & 3L=full)
- Empty water from 3L jug.
- Pour water from 4L jug to 3L jug (4L= empty & 3L=1Litre)
- Fill 4L ~~with~~ completely again.
- Transfer from 4L to 3L jug, (2litre water in 4L jug)

- **Program:**



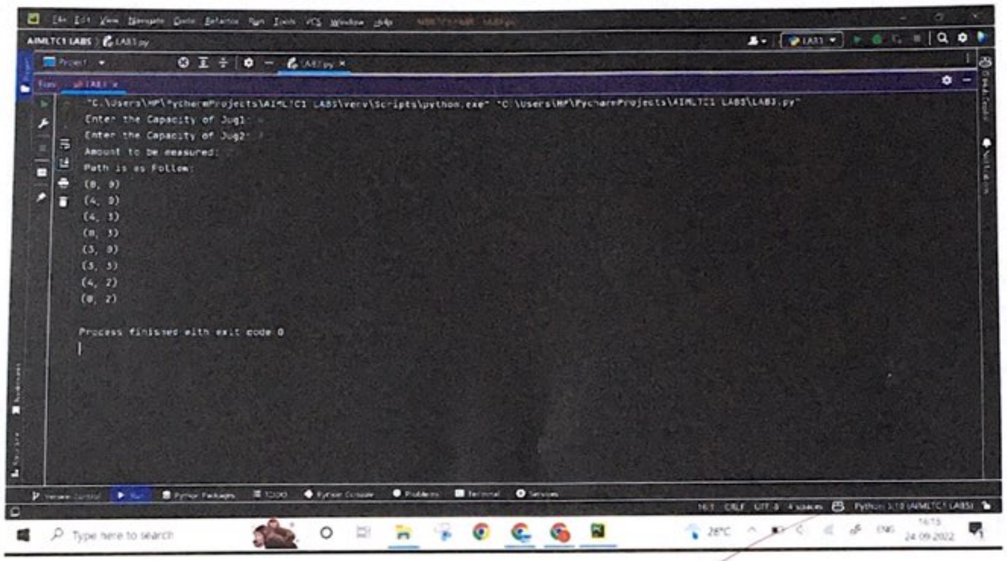
```
# TC1_LAB3
# SIA VASHIST_ 20190802107
# Water Jug Problem

from collections import defaultdict

visited = defaultdict(lambda: False)

# To store J1, J2 and Aim
J1, J2, L = 0, 0, 0


def Water_Jug_problem(X, Y):
    global J1, J2, L

    if (X == L and Y == 0) or (Y == L and X == 0):
        print("(", X, ", ", Y, ")", sep="")
        return True

    if not visited[(X, Y)]:
        print("(", X, ", ", Y, ")", sep="")

        visited[(X, Y)] = True

        return (Water_Jug_problem(0, Y) or
                Water_Jug_problem(X, 0) or
                Water_Jug_problem(J1, Y) or
                Water_Jug_problem(X, J2) or
                Water_Jug_problem(X + min(Y, (J1 - X)),
                                  Y - min(Y, (J1 - X))) or
                Water_Jug_problem(X - min(X, (J2 - Y)),
                                  Y + min(X, (J2 - Y))))
    else:
        return False


# Main Code

J1 = int(input("Enter the Capacity of Jug1: "))
J2 = int(input("Enter the Capacity of Jug2: "))
L = int(input("Amount to be measured: "))

print("Path is as Follow:")

Water_Jug_problem(0, 0)
```

* Algorithm:-

1. Initialises a queue to implement BFS.
2. Since both jugs are empty insert state {0,0} into the queue.
3. Till the queue is empty:

- Pop out the first element of the queue.
- If popped element is equal to target, return True.
- Let X-left & Y-left be the amount of water left in jugs, respectively.
- Fill water operation:-
  - ↳ If value of X-left < X, insert (X-left, Y) into hashmap, since some water can still be poured.
  - ↳ If value of Y-left < Y, insert (Y-left, Y) into hashmap, since some water can still be poured.
- Empty operation:-
  - ↳ If state {0, Y-left} isn't visited, insert it into hashmap since we can empty any of the jugs.
  - ↳ If state {X-left, 0} isn't visited, insert it into hashmap since we can empty any of the jugs.
- Water transfer operation:-
  - ↳ min({X - X-Left, Y) can be poured from second jug to first. Hence, {X + min({X-X-Left, Y)}, Y - min({X-X-left, Y}) isn't visited, put it into hashmap.
  - ↳ min ({X-left, Y-Y-left}) can be poured from first jug to second jug. Hence, {X-left, -min ({X-Left, Y-X-left}), Y+min({X-left, Y-Y-left}) isn't visited, put it into hashmap.
- Return False, if it is not possible to measure Target litres.

- **Output:**



```
"C:\Users\HP\PycharmProjects\AIMLTC1 LABS\venv\Scripts\python.exe" "C:\Users\HP\PycharmProjects\AIMLTC1 LABS\LAB3.py"
Enter the Capacity of Jug1: 
Enter the Capacity of Jug2: 
Amount to be measured: 
Path is as Follow:
(0, 0)
(4, 0)
(4, 3)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)

Process finished with exit code 0
```

* Cases

X	Y	Rule
0	0	—
4	0	1
4	3	5
0	3	8
3	0	10
3	3	5
4	2	10
0	2	8

* Conclusion :-

Hence, we have performed & solved the water Jug problem , successfully.

28/09/22

Experiment no :- 04

- Aim :- Write a program to implement monkey-ladder problem.

- Description :- Given a staircase of N steps and you a monkey can either climb either step 1 or 2 steps at a time. The task is to return the count of distinct ways to climb to the top.

Example :-

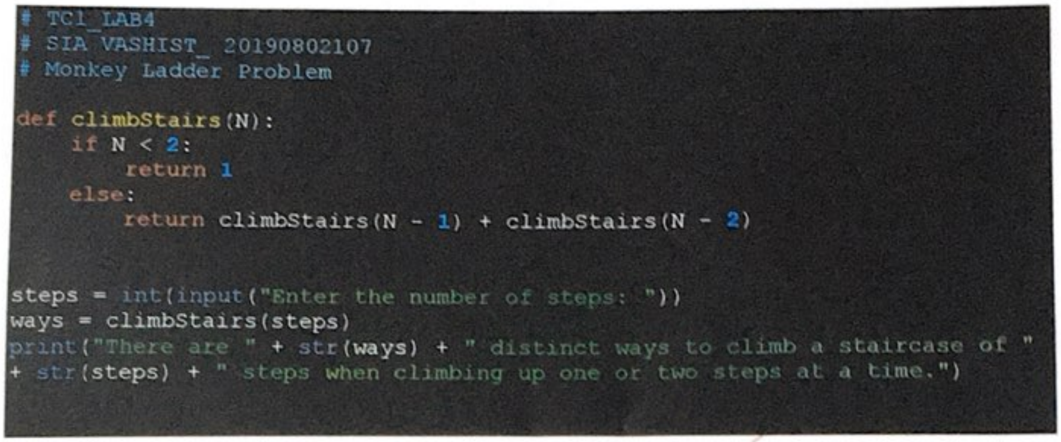
Input :- N = 4
Output :- 5

Explanation :- If n = 4, we can reach n^{th} step in 5 ways, w.r.t the given conditions :
i) 1 step at a time ii) 1+1 = 2 steps.
i.e. [1+1+1+1] ; [1+1+2] ; [2+1+1]; [1+2+1]; [2+2].

* Algorithm :- The algorithm for this problem is almost similar to the fibonacci series.

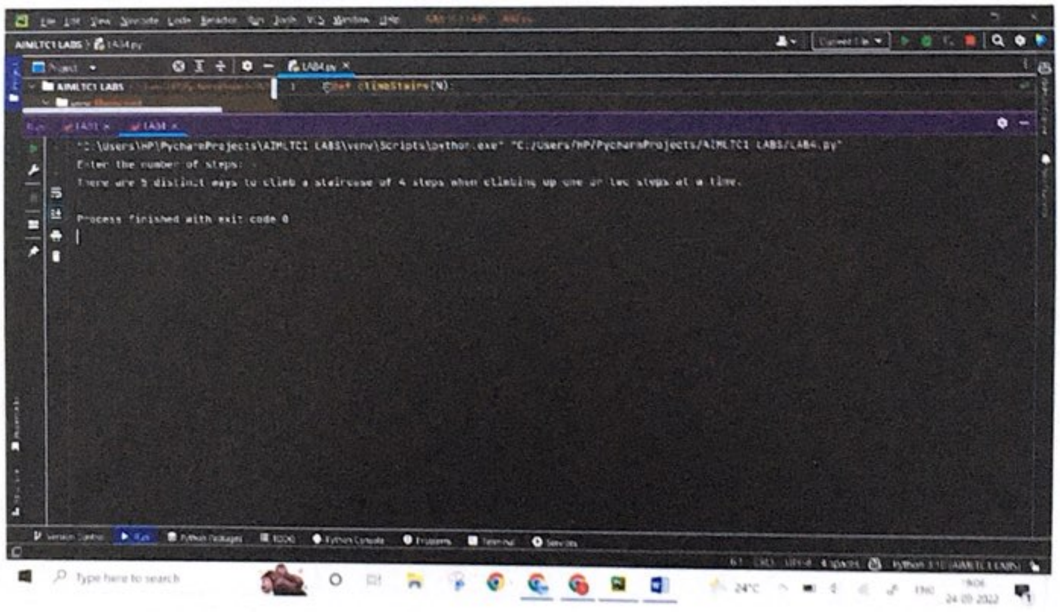
1. Let n be the number of stairs. If n <= 0, then the number of ways to climb the stairs should be zero.

- **Monkey Ladder**
  - **Code:**



```
# TC1_LAB4
# SIA VASHIST_ 20190802107
# Monkey Ladder Problem

def climbStairs(N):
    if N < 2:
        return 1
    else:
        return climbStairs(N - 1) + climbStairs(N - 2)


steps = int(input("Enter the number of steps: "))
ways = climbStairs(steps)
print("There are " + str(ways) + " distinct ways to climb a staircase of "
+ str(steps) + " steps when climbing up one or two steps at a time.")
```

- **Output:**



```
"C:\Users\HP\PycharmProjects\AIMLTC1 LABS\venv\Scripts\python.exe" "C:/Users/HP/PycharmProjects/AIMLTC1 LABS/LAB4.py"
Enter the number of steps: 
There are 5 distinct ways to climb a staircase of 4 steps when climbing up one or two steps at a time.

Process finished with exit code 0
```

2. If n==1, then there is only one way to climb the stair.
3. For the number greater than 1, we can simply find the solution by adding previous steps i.e. Steps(N) = Steps(N-1) + Stairs(N-2). Create a new array and store the output at each step.
4. Then finally return the last value in the array as the output.

* Time complexity :- $O(2^n)$ .. Exponential time
Space complexity :- O(1) ..(Constant space)

* Conclusion :- Hence, we have executed & solved the monkey ladder problem using recursion in python.

28/09/22