Work Sheet 7b: Minor Project (Lab assignment) based on Python Programming

Project 1: Lab Task: Tic-Tac-Toe Game in Python

Objective:
The goal of this lab task is to develop a Tic-Tac-Toe game in Python, reinforcing the students' understanding of:

- Control flow (if-else statements, loops)
- Functions
- Lists and indexing
- Input validation
- Game logic

Task Details:
You will create a Tic-Tac-Toe game where two players (Player X and Player O) take turns to mark a 3x3 grid with their respective symbols ("X" or "O"). The game ends when one player wins by aligning their symbols in a row, column, or diagonal, or when the game results in a tie.

Instructions:

1. **Create the Tic-Tac-Toe board:**
    - Use a list of 9 elements (initialized with empty spaces) to represent the 3x3 grid.
    - Write a function print_board() to display the current state of the board.
2. **Handle Player Input:**
    - Write a function that asks the current player to choose a position (1-9) on the board.
    - Validate that the input is a number between 1 and 9 and that the chosen position is not already taken.
3. **Game Logic:**
    - Implement a function to check if a player has won. A player wins if they have filled any one of the following combinations with their symbol:
        - Rows: [0, 1, 2], [3, 4, 5], [6, 7, 8]
        - Columns: [0, 3, 6], [1, 4, 7], [2, 5, 8]
        - Diagonals: [0, 4, 8], [2, 4, 6]
    - Implement a function to check if the game has resulted in a tie (i.e., all positions are filled, and no player has won).
4. **Switch Turns:**
    - Write logic to alternate between the two players ("X" and "O") after each valid move.
5. **Determine the End of the Game:**
    - After each move, check if a player has won or if the game ends in a tie.
    - If a player wins, display a congratulatory message.
    - If it's a tie, display a message indicating the draw.
6. **Replay Option:**
    - At the end of the game, prompt the players with the option to play again or exit.

Guidelines:

1. **You must write the following functions:**
    - **print_board(board):** Prints the Tic-Tac-Toe board.
    - **check_winner(board, player):** Checks if the current player has won the game.
    - **check_tie(board):** Checks if the game ends in a tie.
    - **get_player_input(player, board):** Prompts the current player for a valid move.

- **play_game():** The main game loop that controls the flow of the game.

2. **Validate all user inputs, ensuring they enter correct positions and avoid overwriting previously marked spots.**

Lab Submission:

- Submit your Python code file (tic_tac_toe.py).
- Ensure your code is properly commented, explaining the functionality of each section.

Example Output:

```
Welcome to Tic-Tac-Toe!

   |   |
--+---+--
   |   |
--+---+--
   |   |

Player X, choose a position (1-9): 5

   |   |
--+---+--
   | X |
--+---+--
   |   |

Player O, choose a position (1-9): 1

O  |   |
--+---+--
   | X |
--+---+--
   |   |

...
Player X wins!
```

Project 2: To-Do List Application

Objective: The goal of this project is to create a simple command-line To-Do list application using Python. This project will allow users to add, view, and delete tasks from their to-do list. By completing this project, students will practice working with lists, functions, and loops.

Instructions

1. Project Overview:

You are required to develop a Python program that acts as a basic to-do list system. The program should continuously prompt the user to either add new tasks, view current tasks, or delete existing tasks until the user decides to exit the application.

---

2. Functionalities to Implement:

1. Add Task (add_task):
   - Function to add a new task to the to-do list.
   - Each task will be a string, and it will be stored in a list.
2. View Tasks (view_tasks):
   - Display the current tasks in the to-do list, showing the task along with its index.
3. Delete Task (delete_task):

- Function to delete a task from the list based on its index.
- If the user provides an invalid index, print an error message.

4. Exit the Program:
   - A command to allow users to exit the to-do list application.

Project 3: Project Question: Robot Path Planning Application

Problem Statement:
You are tasked with developing a Python-based command-line application that simulates a robot navigating through a grid-based environment. The robot should find the shortest path from a specified starting point to a destination while avoiding obstacles placed within the environment. You will implement the A* pathfinding algorithm to compute the shortest path and visualize the result.

Instructions:

1. **Environment Representation**:
   - Use a 2D NumPy array to represent the environment (grid).
   - Free spaces will be represented by 0s and obstacles by 1s.
   - The user will input the grid size (rows and columns) and the number of obstacles with their locations.
2. **User Input**:
   - Prompt the user to input the following:
     - Grid size (number of rows and columns).
     - Number of obstacles and their positions in the grid.
     - Starting point (coordinates) for the robot.
     - Destination point (coordinates) for the robot.
3. *A Pathfinding Algorithm**:
   - Implement the A* algorithm to find the shortest path from the start to the destination.
   - The algorithm should calculate both:
     - g: The actual distance traveled from the start point to the current point.
     - h: The heuristic (Manhattan distance) from the current point to the destination.
     - f = g + h: The total cost.
   - If no path is found, the program should display a message indicating that there is no valid path.
4. **Visualization**:
   - After calculating the path, visualize the grid using matplotlib:
     - Show the robot's path from the starting point to the destination.
     - Mark the obstacles, starting point, and destination clearly on the grid.
     - Obstacles should be shown as black cells, the robot's path as red dots, the starting point as green, and the destination as blue.
5. **Data Management**:
   - Use Pandas to display and manage the grid data before and after pathfinding.
6. **Bonus**:
   - Provide the option for the user to retry with different start, destination, and obstacle positions without restarting the program.