To-Do List Application

Project Documentation

1. Project Title

Console-Based To-Do List Manager in Python

2. Introduction

This project implements a **simple console-based To-Do List application** written in **Python**. It allows users to manage their tasks by providing basic functionalities like adding new tasks, viewing pending tasks, marking tasks as completed, and deleting tasks. The application ensures **data persistence** by storing the tasks in a local text file named tasks.txt.

3. OBJECTIVE

The primary objectives of this project are:

- To create a **functional and user-friendly** command-line interface (CLI) To-Do List application.
- To demonstrate basic **file handling** in Python for data persistence (reading from and writing to a file).
- To practice using **Python data structures** (lists and dictionaries) to manage application data.
- To provide a clear, modular, and easy-to-understand codebase.

4. Technologies Used

The entire application is built using the following core technology:

- **Programming Language:** Python 3.x
- **Libraries/Modules:**
    - **os module:** Used to check for the existence of the task file (tasks.txt).

5. Task Data Structure

Each task is represented as a **Python dictionary** with two key-value pairs:

- "task": A string containing the description of the task.
- "status": A string indicating the task's state, either "pending" or "done".

The complete list of tasks is managed as a **Python list of these dictionaries**.

6. File Structure

Tasks are stored in a plain text file named **tasks.txt**. Each line in the file represents one task and uses a simple delimiter (|) to separate the task description and its status.

- **Format:** [Task Description] | [Status]
- **Example File Content:**
- Buy groceries | pending
- Finish report | done
- Call plumber | pending

7. Module Breakdown and Functionality

The application is structured into several modular functions, each responsible for a specific operation.

Function Name	Purpose
load_tasks()	Reads tasks.txt, handles file creation if it doesn't exist, and returns the tasks as a list of dictionaries.
save_tasks(tasks)	Takes the current list of task dictionaries and overwrites the tasks.txt file with the updated data.
add_task()	Prompts the user for a new task, defaults its status to "pending", loads existing tasks, appends the new one, and saves the list.
view_pending_tasks()	Loads all tasks and prints only those where the status is "pending", displaying them with an index.
mark_task_completed()	Shows pending tasks, prompts for an index, and changes the status of the selected task to "done".
delete_task()	Shows all tasks, prompts for an index, removes the corresponding task from the list, and saves the changes.
main()	The core function that displays the main menu, handles user input, and calls the appropriate function in a loop until the user chooses to exit.

8. ALGORITHM

The main execution follows these steps:

1. **Start.**
2. **Enter Loop:** a. Display the **TO-DO LIST MENU** with options (Add, View Pending, Mark Complete, Delete, Exit). b. Prompt the user to **Enter choice (1-5)**. c. **Process Choice:** * If **1**: Call add_task(). * If **2**: Call view_pending_tasks(). * If **3**: Call mark_task_completed(). * If **4**: Call delete_task(). * If **5**: Print exit message and **Break Loop**. * If **Invalid**: Print an error message. d. **Continue Loop.**
3. **End.**

9. PSEUDOCODE

1. FUNCTION add_task():
2. INPUT task_description
3. tasks = load_tasks()
4. APPEND {"task": task_description, "status": "pending"} to tasks
5. save_tasks(tasks)
6. PRINT "Task added successfully!"
7. **mark_task_completed()**
8. FUNCTION mark_task_completed():
9. tasks = load_tasks()
10. view_pending_tasks() // To show indices
11. INPUT index_to_complete (as task number)
12. index = index_to_complete - 1
13. IF 0 <= index < length(tasks) AND tasks[index]["status"] == "pending":

14. SET tasks[index]["status"] = "done"

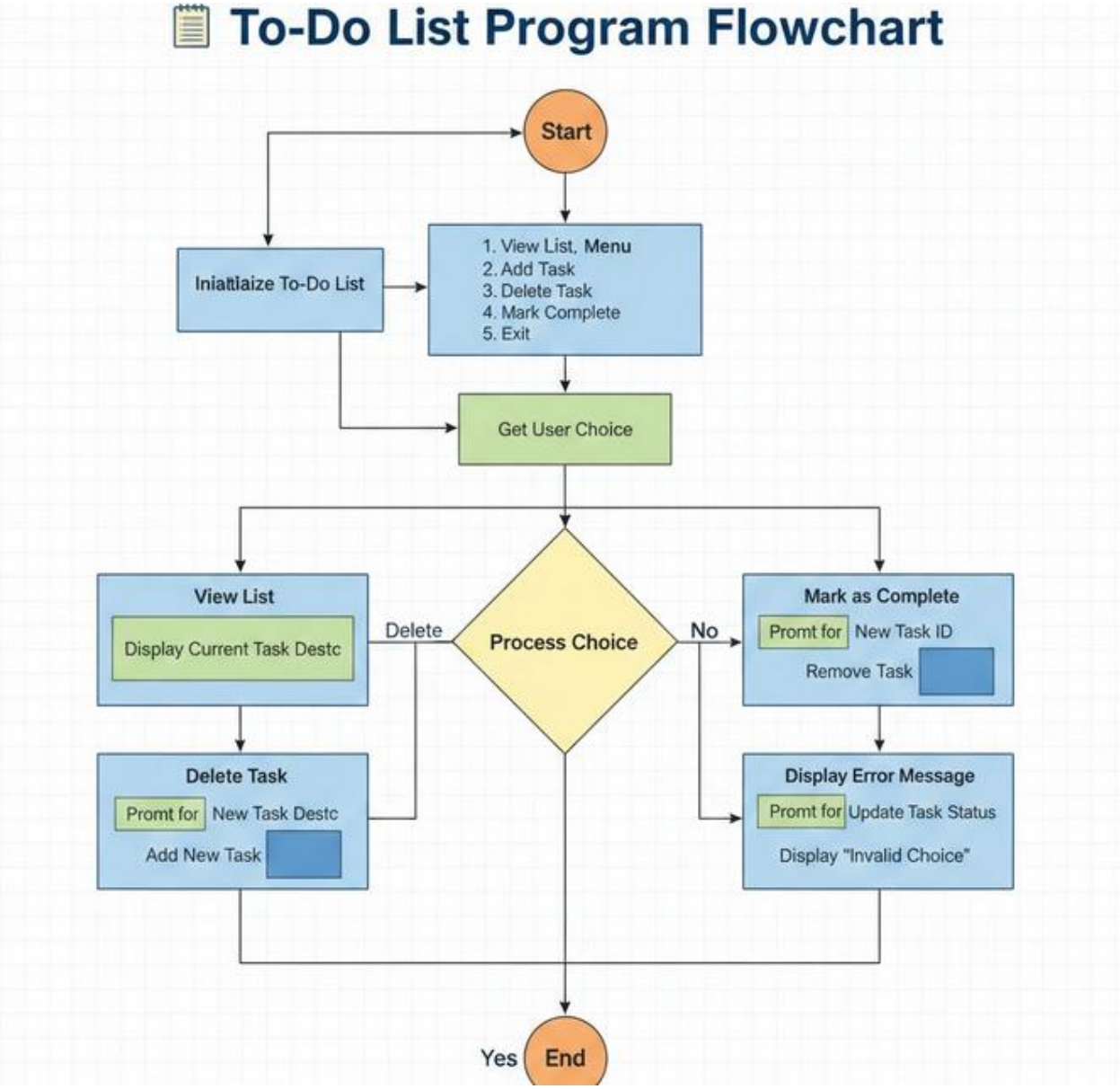
17. save_tasks(tasks)

18. PRINT "Task marked as completed!"

19. ELSE:

15. PRINT "Invalid task number!"

10. FLOWCHART



11. Potential Improvements and Enhancements

While functional, the application could be enhanced with the following features:

- **Error Handling:** Implement try-except blocks to handle non-integer input when asking for task numbers (e.g., in mark_task_completed or delete_task).
- **View All Tasks:** Add an option to view both pending and completed tasks simultaneously.
- **Edit Task:** Add a function to allow the user to modify the description of an existing task.
- **Search/Filter:** Implement searching for tasks by keywords.
- **Data Format Upgrade:** Consider using a more robust data format like **JSON** instead of a custom delimited text file for easier complex data management.

12. Conclusion

The Python To-Do List application successfully achieves its goal of providing a persistent, simple task management system via the command line. It serves as an excellent foundational example of **Python programming**, demonstrating practical concepts like functions, file I/O, error checking, and list/dictionary manipulation.