To Do List Manager Project Report

By- Nehal Jindal

Reg.no-25BAC10003

Overview

This project details the creation of a Python command-line To-Do List Manager program. With basic permanent file storage, the application lets users add, view, and delete tasks. It illustrates basic Python ideas such as data structures, file I/O, and menu-driven interface design.

1. Introduction

1.1 Project Overview

The To-Do List Manager is a simple command-line application that helps users organize and track their daily tasks through basic add, view, and delete operations.

1.2 Project Objectives

- Allow users to add new tasks
- Display all existing tasks with numbering
- Enable task deletion by task number
- Store tasks persistently in a text file
- Provide a simple menu-driven interface

1.3 Scope

Included Features:

- Add tasks to the list
- View all tasks with numbers
- Delete tasks by number
- File-based data storage
- Menu navigation (View, Add, Remove, Exit)

2. Requirements

2.1 Functional Requirements

The application must:

1. Allow users to add tasks through menu option

2. Display all tasks with numbered list format
3. Allow deletion of tasks by selecting task number
4. Save tasks to a text file (to_do_list.txt)
5. Load tasks from file when program starts
6. Display a menu with View, Add, Remove, and Exit options

2.2 Non-Functional Requirements

1. **Performance:** Application must load and respond to user input in less than 100 milliseconds.
2. **Usability:** Interface must be simple to use and require little training
3. **Compatibility:** Must work on all modern browsers (Chrome, Firefox, Safari, Edge)
4. **Reliability:** Data persistence must have 99.9% reliability
5. **Scalability:** Architecture must be able to handle more than 1000 tasks without experiencing a drop in performance.

3. Architecture

3.1 System Design

The application consists of three main functions:

- `read()`: Reads tasks from the text file
- `write()`: Saves tasks to the text file
- `display()`: Shows the task list with numbering

Data is stored as a Python list during execution and persisted to a text file.

3.2 Technology Used

- Language: Python 3.13
- Storage: Text file (.txt)
- Data Structure: Python List

4. Implementation

4.1 How It Works

1. **Program Start:** Load tasks from to_do_list.txt file

2. **Main Menu:** Display options and get user choice
3. **View Tasks:** Display all tasks with numbers
4. **Add Task:** Accept task name and add to list
5. **Remove Task:** Delete selected task by number
6. **Exit:** Close the program

4.2 Code Functions

read() function:
Opens the file and loads all existing tasks into a list.

write() function:
Saves all tasks from the list back to the text file.

display() function:
Prints all tasks with 1-based numbering for easy selection.

Main Loop:
Infinite while loop that displays menu and processes user choices.

5. Testing

5.1 Test Cases

Test	Action	Result
TC1	Add a task	Task appears in list
TC2	View tasks	All tasks display with numbers
TC3	Delete a task	Task is removed from list and file
TC4	Exit program	Program closes
TC5	Restart program	Previous tasks are still there

6. Results

6.1 Completed Features

- Add tasks functionality
- View tasks functionality
- Delete tasks functionality

- File persistence working
- Menu navigation complete

7. Challenges Faced

1. **File Synchronization:** Needed to ensure tasks stay synchronized between file and memory
2. **Input Validation:** Program needs better error handling for invalid inputs
3. **File Management:** Proper handling of file reading and writing operations

8. Future Improvements

- Add task priority levels
- Add due dates for tasks
- Implement task editing capability
- Create a graphical user interface (GUI)
- Add task categories or projects
- Implement search functionality

9. Conclusion

The To-Do List Manager successfully implements all basic requirements for a command-line task management tool. The application works reliably for adding, viewing, and deleting tasks with persistent file storage. It serves as a practical learning project demonstrating Python fundamentals and can be extended with additional features in the future.

10. Challenges Faced

10.1 Index Management

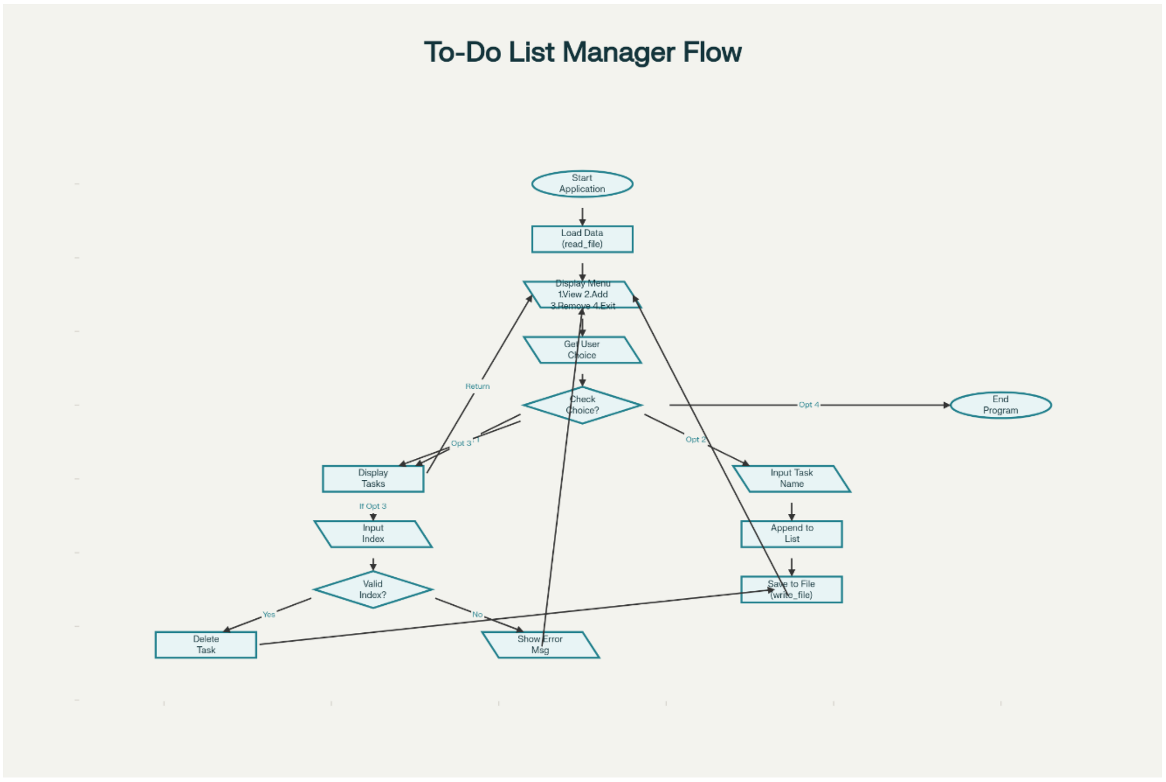
Initially, users would enter "1" to delete the first task, but Python deletes index 1 (the second task). I resolved this by subtracting 1 from the user's input.

10.2 File Creation

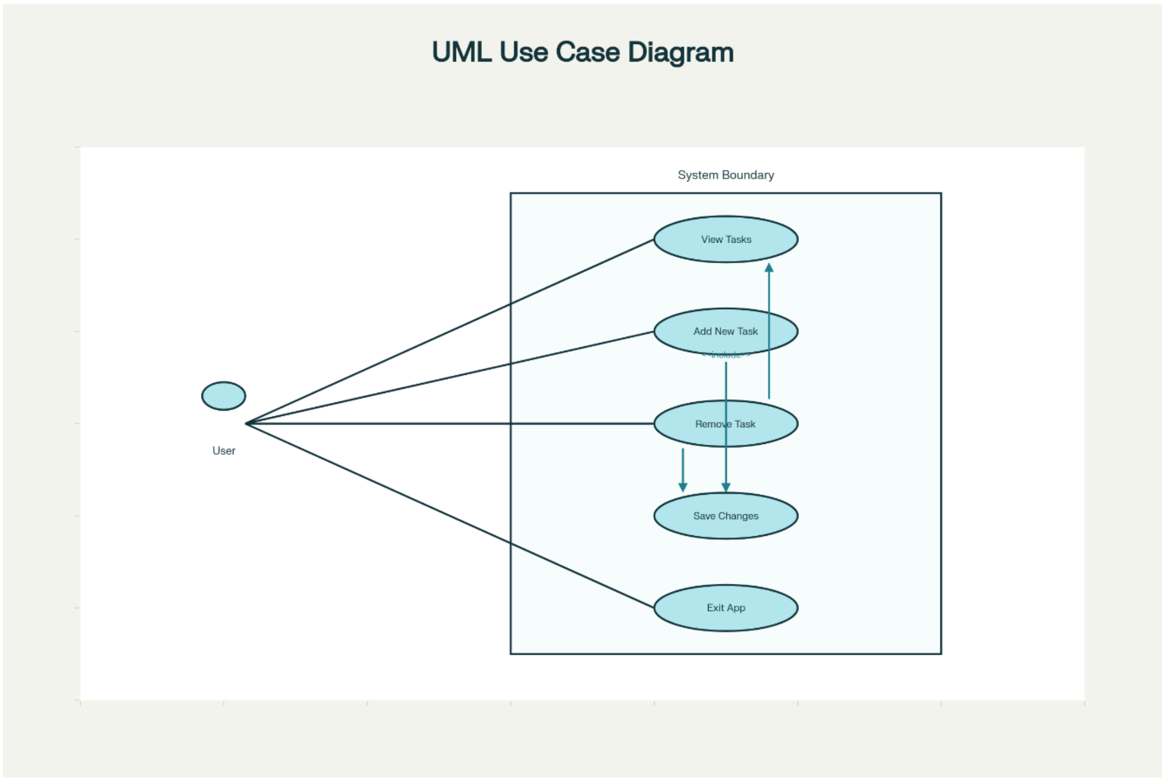
The program initially crashed if the text file didn't exist. I added an os.path.exists check to create a new list gracefully.

11. Design Diagrams

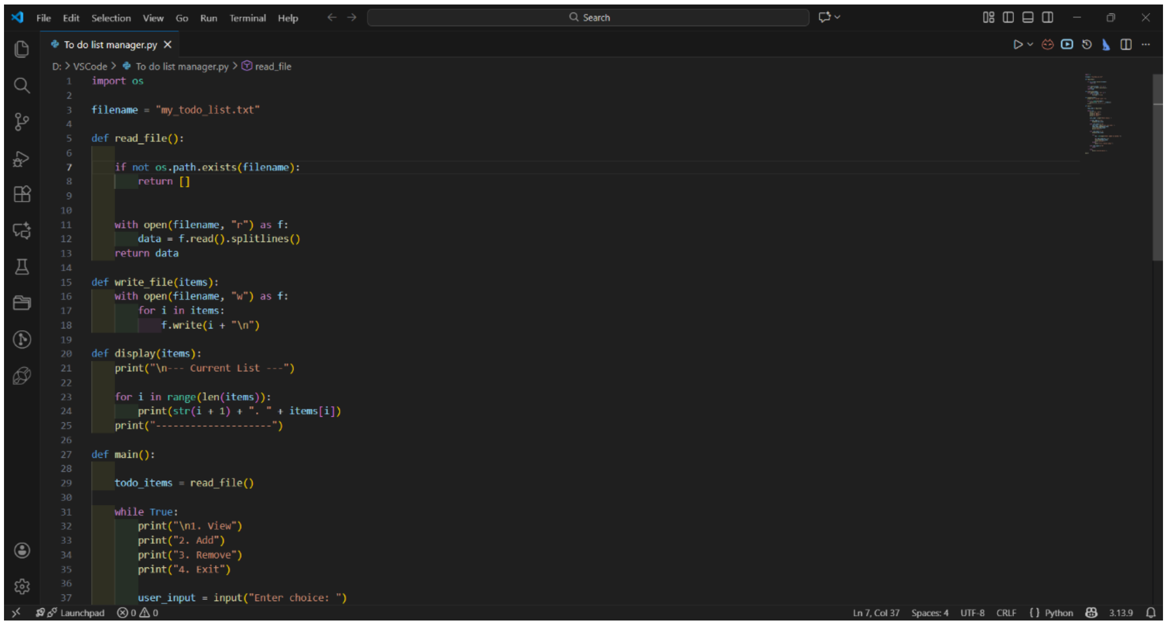
11.1 Workflow



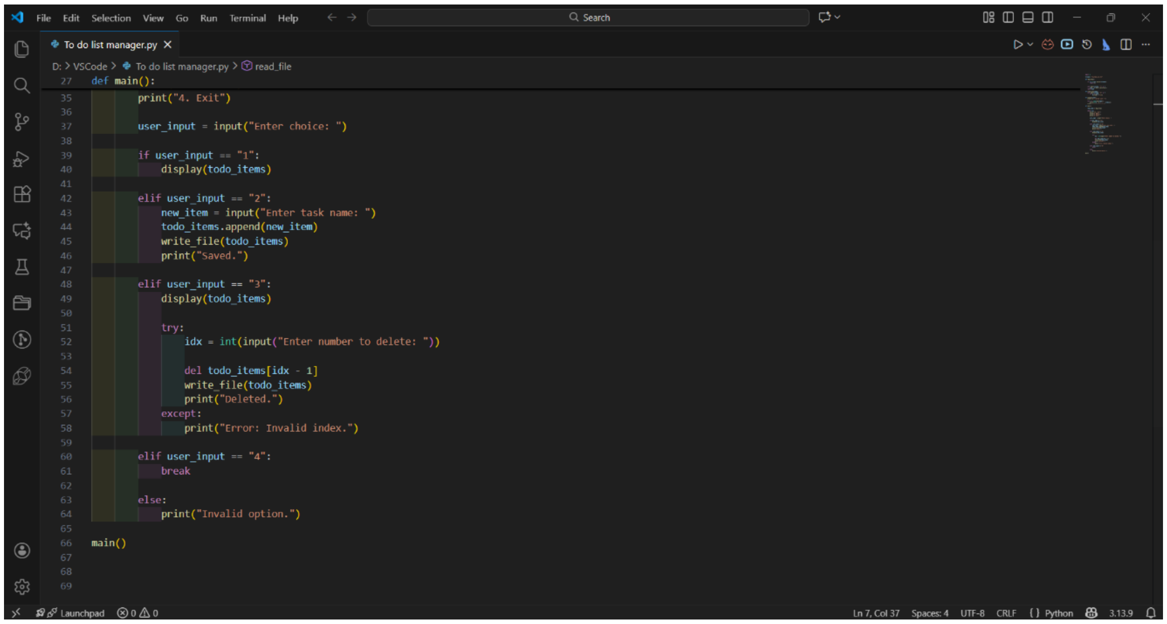
11.2 Use Case



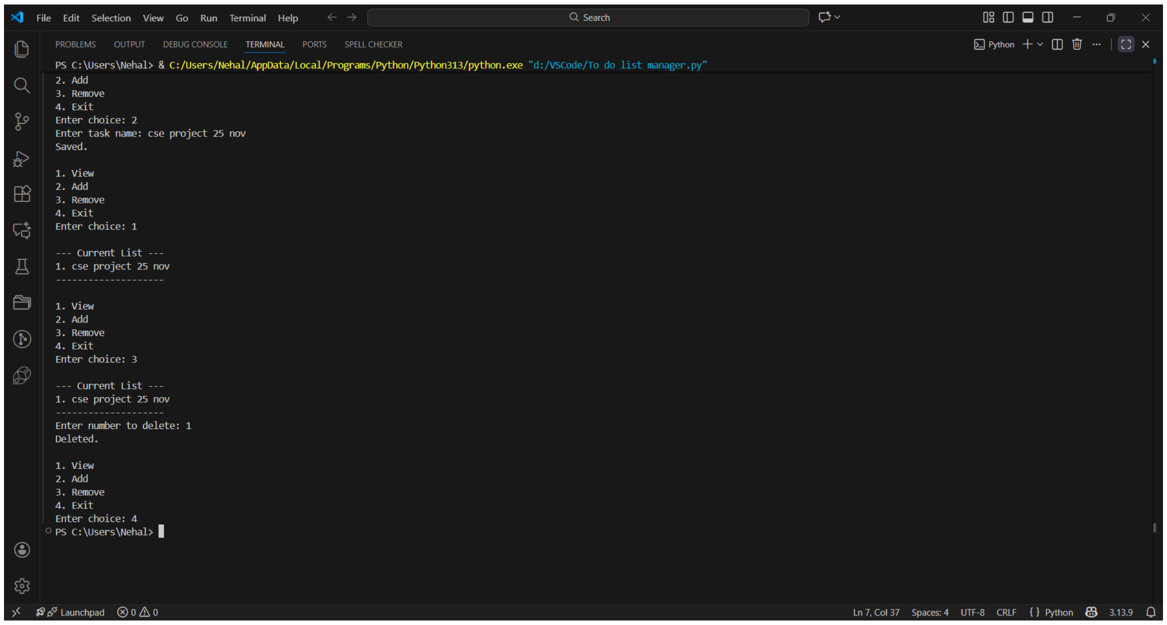
12.Screesnshots


File Edit Selection View Go Run Terminal Help
Search
To do list manager.py
D: > VSCode > To do list manager.py > read_file
Launchpad
Ln 7, Col 37 Spaces: 4 UTF-8 CRLF { } Python 3.13.9

```python
import os

filename = "my_todo_list.txt"

def read_file():

    if not os.path.exists(filename):
        return []


    with open(filename, "r") as f:
        data = f.read().splitlines()
    return data

def write_file(items):
    with open(filename, "w") as f:
        for i in items:
            f.write(i + "\n")

def display(items):
    print("\n--- Current List ---")

    for i in range(len(items)):
        print(str(i + 1) + ". " + items[i])
    print("-------------------")

def main():

    todo_items = read_file()

    while True:
        print("\n1. View")
        print("2. Add")
        print("3. Remove")
        print("4. Exit")

        user_input = input("Enter choice: ")
```


File Edit Selection View Go Run Terminal Help
To do list manager.py
D: > VSCode > To do list manager.py > read_file
Ln 7, Col 37 Spaces: 4 UTF-8 CRLF { } Python 3.13.9

```python
def main():
    print("4. Exit")

    user_input = input("Enter choice: ")

    if user_input == "1":
        display(todo_items)

    elif user_input == "2":
        new_item = input("Enter task name: ")
        todo_items.append(new_item)
        write_file(todo_items)
        print("Saved.")

    elif user_input == "3":
        display(todo_items)

        try:
            idx = int(input("Enter number to delete: "))

            del todo_items[idx - 1]
            write_file(todo_items)
            print("Deleted.")
        except:
            print("Error: Invalid index.")

    elif user_input == "4":
        break

    else:
        print("Invalid option.")

main()
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER
Ln 7, Col 37 Spaces: 4 UTF-8 CRLF { } Python 3.13.9

```
PS C:\Users\Nehal> & C:/Users/Nehal/AppData/Local/Programs/Python/Python313/python.exe "d:/VSCode/To do list manager.py"
2. Add
3. Remove
4. Exit
Enter choice: 2
Enter task name: cse project 25 nov
Saved.

1. View
2. Add
3. Remove
4. Exit
Enter choice: 1

--- Current List ---
1. cse project 25 nov
--------------------

1. View
2. Add
3. Remove
4. Exit
Enter choice: 3

--- Current List ---
1. cse project 25 nov
--------------------
Enter number to delete: 1
Deleted.

1. View
2. Add
3. Remove
4. Exit
Enter choice: 4
PS C:\Users\Nehal>
```

13. References

[1] Python Software Foundation. (2024). Python File I/O.
https://docs.python.org/3/tutorial/inputoutput.html