Introduction

Task management is an essential activity for students, professionals, and individuals who manage multiple daily responsibilities. In most cases, people rely on memory or handwritten notes, which may lead to missed deadlines or disorganized work.

PyTask Scheduler is a simple, command-line-based Python application designed to help users manage their daily tasks efficiently. The system allows users to add, view, update, delete, and prioritize tasks without requiring any external libraries or GUI. The project demonstrates the use of Python fundamentals such as lists, functions, loops, conditionals, and menu-driven logic.

Problem Statement

People often struggle to keep track of their tasks, due dates, and priorities, especially when managing multiple activities. Traditional methods like notes or mental lists are unreliable and may lead to missed deadlines.
There is a need for a simple, structured system that allows users to organize and monitor their tasks easily through a basic and accessible interface.

Functional Requirements

These are the features the system **must** provide:

1. **Add Task**
    - User can enter title, priority, and due date.
2. **View Tasks**
    - Display all tasks with their details.
3. **Mark as Completed**
    - Update the status of a selected task.
4. **Delete a Task**
    - Remove a task from the list.
5. **Sort Tasks**
    - Sort by priority (High → Medium → Low)
6. **Exit Program**
    - Close the application safely.

Non-functional Requirements

1. **Usability**
    - Easy-to-understand menu structure.

2. **Efficiency**
    - Fast execution with minimal memory usage.
3. **Reliability**
    - Inputs validated to avoid errors.
4. **Portability**
    - Runs on any system with Python installed.
5. **Maintainability**
    - Code is modular and well-structured.

System Architecture

The system follows a **linear modular architecture**:

- **Main Module**
  Controls application flow and menu.
- **Task Management Module**
  Functions for adding, viewing, deleting, marking, and sorting tasks.
- **Data Storage Module**
  Uses Python lists to temporarily store tasks in memory.

Architecture Flow:

User → Menu → Function Call → Task List → Output → Back to Menu

Design Diagrams

a. Use Case Diagram

Actors: User
Use Cases: Add Task, View Tasks, Delete Task, Mark Completed, Sort Tasks, Exit

b. Workflow Diagram

1. Start
2. Display Menu
3. User selects option
4. System executes function
5. Display result
6. Return to menu
7. Exit

c. Sequence Diagram

User → Menu → Selected Function → Task List → Output → Menu

d. Class / Component Diagram

(For simple projects without OOP, treat each function as a component)

Components:

- AddTask()
- ViewTasks()
- MarkCompleted()
- DeleteTask()
- SortByPriority()

e. ER Diagram

Not applicable (no database used).
(Write: *"Tasks are stored temporarily in memory using Python lists. Hence, no ERD is required."*)

Design Decisions & Rationale

- **Python chosen** due to simplicity and readability.
- **CLI interface**—easy for beginners, platform independent.
- **List data structure**—efficient for dynamic task storage.
- **Modular functions**—makes the code cleaner and maintainable.
- **No external libraries**—keeps the project simple and portable.

Implementation Details

- Program is developed in Python 3.x.
- Tasks stored as nested lists:
  ["Title", "Priority", "Due Date", "Status"]
- Menu-driven interface within an infinite loop.
- Sorting implemented using manual comparison logic (without libraries).
- Program ends only when user chooses Exit.

Include Code Snippets (your main program file).

Testing Approach

Tested using **manual testing** with the following cases:

1. Valid task input
2. Empty task list
3. Invalid menu options
4. Invalid task number
5. Sorting with same priorities
6. Sorting with different priorities
7. Due date sorting
8. Delete task functionality
9. Completion marking

All features tested and validated.

Challenges Faced

- Designing sorting logic without using libraries
- Ensuring menu loops run correctly
- Handling invalid user inputs
- Maintaining clean code structure for readability
- Ensuring all features work together without errors

Learnings & Key Takeaways

- Improved understanding of Python fundamentals
- Learned modular programming and function design
- Experience with debugging and testing
- Better understanding of user-interaction program flow
- Learned how to document and structure a project

Future Enhancements

- Add file storage or database to save tasks
- Add editing feature for tasks
- Convert to GUI (Tkinter/PyQt)
- Create a mobile or web version
- Add reminders or notifications

- Implement color-coded terminal output

References

- Python Official Documentation (python.org)