Task Management Application

Complete Project Documentation

Python Programming Assignment
Command-Line Task Manager with Persistent Storage

Date: December 22, 2025

Submission Deadline: January 26, 2026 - 4:30 PM

Table of Contents

1. 1. Executive Summary
2. 2. Project Overview
3. 3. Features and Functionality
4. 4. Technical Specifications
5. 5. System Architecture
6. 6. Installation and Setup
7. 7. User Guide
8. 8. Code Documentation
9. 9. Error Handling
10. 10. Testing and Validation
11. 11. Version Control
12. 12. Requirements Compliance
13. 13. Future Enhancements
14. 14. Appendices

1. Executive Summary

The Task Management Application is a sophisticated command-line interface (CLI) program developed in Python that enables users to efficiently manage their daily tasks. The application provides a complete CRUD (Create, Read, Update, Delete) interface with persistent data storage using JSON format. Built with object-oriented programming principles and following PEP 8 standards, this application demonstrates best practices in Python development.

Key Highlights

- Fully functional menu-driven CLI interface
- Persistent data storage using JSON format
- Comprehensive error handling and input validation
- Object-oriented design with Task and TaskManager classes
- Complete test suite with automated testing
- Professional documentation and code comments
- Git version control with meaningful commit history

2. Project Overview

2.1 Purpose

This application was developed to fulfill the requirements of a Python programming assignment, demonstrating proficiency in core Python concepts including data structures, file I/O, error handling, object-oriented programming, and software development best practices.

2.2 Scope

The application provides a complete task management solution that allows users to:

- Add new tasks with titles and optional descriptions
- View all tasks organized by completion status
- Mark tasks as completed with visual indicators
- Delete unwanted tasks with confirmation prompts
- Automatically save and load tasks from persistent storage

2.3 Technology Stack

Programming Language	**Python 3.10.5**
Data Format	JSON
Architecture	Object-Oriented Programming
Version Control	Git
Dependencies	Standard Library Only

3. Features and Functionality

3.1 Core Features

3.1.1 Add Task

Users can create new tasks by providing a title (required) and an optional description. Each task is automatically assigned a unique ID and timestamp.

3.1.2 View Tasks

Displays all tasks in an organized format, separating pending and completed tasks. Shows task ID, title, description, completion status, and creation timestamp.

3.1.3 Mark as Completed

Allows users to mark tasks as completed by entering the task ID. Visual indicators (✓) show completion status.

3.1.4 Delete Task

Enables users to remove tasks from the system. Includes a confirmation prompt to prevent accidental deletions.

3.2 Additional Features

- Task descriptions for detailed information
- Automatic timestamp tracking
- Visual indicators (emoji) for better UX
- Empty list detection with helpful messages
- Input validation for all user entries
- Confirmation prompts for destructive operations
- Automatic data persistence on every change
- Graceful error recovery with rollback mechanism

4. Technical Specifications

4.1 System Requirements

Minimum Requirements:

- Python 3.6 or higher
- Operating System: Windows, macOS, or Linux
- Storage: Minimal (< 1 MB)
- Memory: < 50 MB RAM
- No external dependencies required

4.2 File Structure

```
project/
│
├── task_manager.py          # Main application file (354 lines)
├── test_task_manager.py     # Automated test suite (150 lines)
├── tasks.json               # Data storage (auto-generated)
├── README.md                # Project documentation
├── .gitignore               # Git ignore rules
└── SUBMISSION_CHECKLIST.txt # Requirements verification
```

4.3 Data Structure

Tasks are stored in JSON format with the following structure:

```
{
  "task_id": 1,
  "title": "Complete Python assignment",
  "description": "Build a task manager",
  "completed": false,
  "created_at": "2025-12-22T14:30:00.123456"
}
```

5. System Architecture

5.1 Class Diagram

The application uses two main classes:

1. Task Class

Represents a single task with the following attributes:

- task_id: Unique identifier (integer)
- title: Task title (string)
- description: Optional description (string)
- completed: Completion status (boolean)
- created_at: ISO format timestamp (string)

2. TaskManager Class

Manages all task operations with methods for:

- load_tasks(): Load tasks from JSON file
- save_tasks(): Save tasks to JSON file
- add_task(): Create a new task
- view_tasks(): Display all tasks
- mark_completed(): Mark task as done
- delete_task(): Remove a task
- _find_task(): Helper to locate task by ID

5.2 Program Flow

1. Initialize application and load existing tasks from JSON

2. Display main menu with 5 options

3. Accept user input and validate

4. Execute selected operation (add, view, complete, delete)

5. Save changes to JSON file automatically

6. Display result and return to menu

7. Repeat until user chooses to exit

6. Installation and Setup

6.1 Prerequisites

Ensure Python 3.6+ is installed on your system.

```
python --version
```

6.2 Installation Steps

1. Clone or download the repository

2. Navigate to the project directory

3. Verify all files are present (task_manager.py, README.md, etc.)

4. Run the application using: python task_manager.py

6.3 First Run

On the first run, the application will create a tasks.json file automatically. This file stores all your tasks and is loaded every time you start the application.

7. User Guide

7.1 Starting the Application

Run the following command in your terminal:

```
python task_manager.py
```

7.2 Menu Options

The main menu displays 5 options:

15. 1. Add a new task - Create a task with title and description
16. 2. View all tasks - Display all tasks organized by status
17. 3. Mark task as completed - Update task status to completed
18. 4. Delete a task - Remove a task permanently
19. 5. Exit - Close the application

7.3 Usage Examples

Example 1: Adding a Task

```
Enter your choice (1-5): 1

➕ Add New Task
------------------------------------------------------------
Task title: Complete Python assignment
Task description (optional): Build a task manager
✓ Task added successfully: 'Complete Python assignment'
```

Example 2: Viewing Tasks

```
Enter your choice (1-5): 2

📋 Your Tasks (2 total):
============================================================

⏳ Pending Tasks:

[X] 1. Complete Python assignment
   Description: Build a task manager
   Created: 2025-12-22 14:30
```

Example 3: Marking Task as Completed

```
Enter your choice (1-5): 3

Enter task ID to mark as completed: 1
✓ Task marked as completed: 'Complete Python assignment'
```

8. Code Documentation

8.1 Code Quality Standards

- PEP 8 compliant naming conventions (snake_case for functions, CamelCase for classes)
- Comprehensive docstrings for all classes and functions
- Type hints documented in function docstrings
- Inline comments explaining complex logic
- Proper indentation and spacing (4 spaces)
- Line length limited to 100 characters where practical
- Meaningful variable and function names

8.2 Key Functions

main()

The main application loop that displays the menu, handles user input, and coordinates all operations.

get_user_input()

Validates and returns user input with type conversion and empty value handling.

display_menu()

Displays the main menu with formatted options and visual indicators.

9. Error Handling

9.1 Error Types Handled

Error Type	Handling Mechanism
Invalid menu choice	Input validation with range check (1-5)
Empty task list	Display helpful message instead of error
Invalid task ID	ValueError exception with clear message
File not found	Create new file automatically
JSON corruption	Fallback to empty list with warning
Empty input	Prompt user to enter valid data
Save failure	Rollback changes and show error

9.2 Error Recovery

The application implements a rollback mechanism for critical operations. If saving fails after adding or deleting a task, the operation is reversed to maintain data consistency.

10. Testing and Validation

10.1 Automated Test Suite

A comprehensive test suite (test_task_manager.py) validates all functionality:

- TEST 1: Initialize Task Manager
- TEST 2: Add New Tasks
- TEST 3: View All Tasks
- TEST 4: Mark Task as Completed
- TEST 5: View Tasks (with completed items)
- TEST 6: Delete a Task
- TEST 7: Data Persistence
- TEST 8: Error Handling - Invalid Task ID
- TEST 9: Error Handling - Empty Task Title
- TEST 10: Add More Tasks and View Final State
- TEST 11: Verify JSON File Structure

10.2 Test Results

All tests passed successfully:

✓ Total tasks created: 5

✓ Current tasks in system: 4

✓ Completed tasks: 2

✓ Pending tasks: 2

✓ Data persistence: Verified

✓ Error handling: Verified

10.3 Manual Testing Checklist

✓ Add task with title only

✓ Add task with title and description

✓ View empty task list

✓ View task list with multiple items

✓ Mark task as completed

✓ Try to mark already completed task

✓ Delete task with confirmation

✓ Cancel delete operation

✓ Restart app and verify data loads

✓ Enter invalid menu choice

✓ Enter invalid task ID

✓ Try to add task with empty title

11. Version Control

11.1 Git Repository

The project uses Git for version control with a well-structured commit history.

11.2 Commit History

1. ba1f6c5 - Initial commit: Add project documentation and gitignore

2. 0a8d851 - Add Task and TaskManager classes with CRUD operations

3. 4ddecef - Add comprehensive test suite for all features

4. 2feea2e - Add submission checklist documenting all requirements

11.3 Repository Structure

The repository includes a .gitignore file to exclude Python cache files, virtual environments, and other non-essential files from version control.

12. Requirements Compliance

12.1 Core Features Checklist

Requirement	Status
Add a new task	✓ Implemented
View all tasks	✓ Implemented
Mark task as completed	✓ Implemented
Delete a task	✓ Implemented
Menu-based CLI	✓ Implemented

12.2 Technical Requirements Checklist

Requirement	Status
Uses lists/dictionaries	✓ Implemented
Persistent storage (JSON)	✓ Implemented
Data reloads on restart	✓ Implemented
Functions and classes	✓ Implemented
PEP 8 compliant	✓ Implemented
Comprehensive error handling	✓ Implemented
Complete documentation	✓ Implemented
Git version control	✓ Implemented

13. Future Enhancements

Potential improvements for future versions:

- Task priority levels (High, Medium, Low)
- Due dates with reminder notifications
- Task categories or tags for organization
- Search and filter functionality
- Task editing capabilities
- Export to CSV or PDF formats
- Multi-user support with authentication
- Web-based interface using Flask or Django
- Database backend (SQLite, PostgreSQL)
- Cloud synchronization
- Mobile app companion
- Statistics and productivity reports

14. Appendices

Appendix A: Sample Code Snippets

Task Class Implementation (Excerpt):

```
class Task:
    """Represents a single task."""

    def __init__(self, task_id, title, description="",
                 completed=False, created_at=None):
        self.task_id = task_id
        self.title = title
        self.description = description
        self.completed = completed
        self.created_at = created_at or datetime.now().isoformat()
```

Appendix B: JSON Data Format Example

```
[
  {
    "task_id": 1,
    "title": "Complete Python assignment",
    "description": "Build a task manager",
    "completed": true,
    "created_at": "2025-12-22T14:22:59.584997"
  },
  {
    "task_id": 2,
    "title": "Study for exams",
    "description": "Focus on chapters 5-8",
    "completed": false,
    "created_at": "2025-12-22T14:23:15.123456"
  }
]
```

Appendix C: Contact Information

Project Repository: GitHub (URL to be added)

Submission Date: January 26, 2026 - 4:30 PM

Python Version: 3.10.5

Development Date: December 22, 2025

Conclusion

This Task Management Application successfully demonstrates proficiency in Python programming, object-oriented design, error handling, file I/O, and software development best practices. The application meets and exceeds all assignment requirements, providing a robust, user-friendly solution for task management with comprehensive documentation and testing.

The implementation showcases clean code architecture, proper error handling, persistent data storage, and professional documentation standards. The project is production-ready and fully prepared for submission.
