Build a Command-Line Task Manager in Python

1. Objective

Develop a command-line application that allows users to manage their daily tasks. The application should enable users to:

- Add new tasks
- View all tasks
- Mark tasks as completed
- Delete tasks

This project reinforces fundamental Python concepts such as data structures, control flow, functions, and user input handling.

2. Problem Statement

Task management is a common requirement in both daily life and professional settings.
By building a simple task manager, learners will apply Python programming concepts to solve a real-world problem, enhancing their understanding and retention of core programming skills.

3. Inputs / Shared Artifacts

- Programming Language: Python 3.x
- Development Environment: Jupyter Notebook or any Python IDE
- Python Concepts Utilized:
    - Variables and Data Types
    - Lists and Dictionaries
    - Control Flow (if-else statements, loops)
    - Functions
    - User Input Handling

4. Expected Outcome

Upon completing this exercise, learners will:

- Apply Python fundamentals to build a functional application.
- Understand how to manage and manipulate data using lists and dictionaries.
- Gain experience writing clean, modular code using functions.
- Enhance problem-solving skills by implementing user interaction and input validation.

5. Concepts Covered

- Data structures for task management (list of dictionaries)
- Function design for modularity
- Input validation and error handling
- Loop control for continuous user interaction
- Basic command-line user interface design

6. Example: Step-by-Step Implementation Guide

Step 1: Define the Data Structure

- Use a list named `tasks` to hold all task dictionaries.
- Each task dictionary contains:

```
{
    "id": int,             # unique task ID
    "description": str,    # task description
    "completed": bool      # completion status
}
```

Step 2: Implement Core Functions

```
tasks = []

def add_task(description):
    task_id = tasks[-1]["id"] + 1 if tasks else 1
    task = {"id": task_id, "description": description, "completed":
False}
    tasks.append(task)
    print(f"Task '{description}' added with ID {task_id}.")
```

```
def view_tasks():
    if not tasks:
        print("No tasks available.")
        return
    for task in tasks:
        status = "Done" if task["completed"] else "Pending"
        print(f"{task['id']}: {task['description']} [{status}]")

def mark_completed(task_id):
    for task in tasks:
        if task["id"] == task_id:
            task["completed"] = True
            print(f"Task ID {task_id} marked as completed.")
            return
    print(f"No task found with ID {task_id}.")

def delete_task(task_id):
    global tasks
    tasks = [task for task in tasks if task["id"] != task_id]
    print(f"Task ID {task_id} deleted if it existed.")
```

Step 3: Create the User Interface Loop

```
def main():
    while True:
        print("\nOptions: add, view, complete, delete, exit")
        choice = input("Enter command: ").strip().lower()
        if choice == "add":
            desc = input("Enter task description: ").strip()
            add_task(desc)
        elif choice == "view":
            view_tasks()
        elif choice == "complete":
            try:
                task_id = int(input("Enter task ID to mark complete:
"))
                mark_completed(task_id)
            except ValueError:
                print("Invalid task ID.")
        elif choice == "delete":
            try:
                task_id = int(input("Enter task ID to delete: "))
                delete_task(task_id)
            except ValueError:
                print("Invalid task ID.")
        elif choice == "exit":
            print("Exiting Task Manager. Goodbye!")
```

```
                break
            else:
                print("Invalid command.")

if __name__ == "__main__":
    main()
```

Step 4: Run and Test

- Run your script and test adding, viewing, completing, and deleting tasks.
- Verify IDs increment properly, and tasks update as expected.

7. Final Submission Checklist

- **Source Code:** Complete Python script implementing all required features, clean and well-commented.
- **Documentation:** Short README or report including:
    - Explanation of your approach
    - Challenges faced and how you addressed them
    - Optional: Any enhancements or extensions implemented
- **Input Validation (Optional):** Robust handling of invalid or unexpected inputs in the user interface.