Python Project 07: To Do CLI

1. Project Overview

A **To Do CLI** is a command-line application for managing daily tasks directly from the terminal.

Users can:

- Add tasks
- View tasks
- Mark tasks as completed
- Delete tasks
- Clear completed tasks

The project uses Python's built-in `argparse` module to create CLI commands and arguments. `argparse` is designed for building command-line interfaces and automatically provides help and argument validation.

Tasks are stored locally in a JSON file, so they remain available when the application is run again.

---

2. Features

- Add a task
- List all tasks
- Mark a task as completed
- Delete a task
- Clear completed tasks
- Persistent JSON storage
- Command-line interface
- Automatic `--help`

- Input validation
- No external packages

---

3. Technologies

- Python 3
- argparse
- json
- pathlib
- Object Oriented Programming

All modules used in this project are part of Python's standard library.

---

4. Folder Structure

```
to_do_cli/
□□□ main.py
□□□ todo.py
□□□ storage.py
□□□ tasks.json
```

File Responsibilities

File	Responsibility
main.py	CLI commands and application entry point

`todo.py`	Task management logic
`storage.py`	Reading and saving tasks
`tasks.json`	Local task data

---

5. How the Project Works

`User enters CLI command`
↓
`argparse reads command`
↓
`main.py`
↓
`TodoManager`
↓
`Storage`
↓
`tasks.json`

Example:

```
python main.py add "Learn Python"
```

The command is parsed by `argparse`, sent to the task manager, and saved to `tasks.json`.

---

6. Complete Backend Code

storage.py

```
import json
from pathlib import Path


class TaskStorage:

    def __init__(self, filename="tasks.json"):
        self.file_path = Path(filename)

    def load_tasks(self):
        if not self.file_path.exists():
            return []

        try:
            with self.file_path.open("r", encoding="utf-8") as file:
                data = json.load(file)

            if not isinstance(data, list):
                return []

            return data

        except (json.JSONDecodeError, OSError):
            return []

    def save_tasks(self, tasks):
        with self.file_path.open("w", encoding="utf-8") as file:
            json.dump(tasks, file, indent=4)
```

---

todo.py

```
from storage import TaskStorage


class TodoManager:

    def __init__(self):
        self.storage = TaskStorage()
        self.tasks = self.storage.load_tasks()

    def add_task(self, title):
        title = title.strip()

        if not title:
            raise ValueError("Task cannot be empty.")

        task = {
            "id": self.get_next_id(),
            "title": title,
            "completed": False
        }

        self.tasks.append(task)
        self.save()

        return task

    def get_next_id(self):
        if not self.tasks:
            return 1

        return max(task["id"] for task in self.tasks) + 1
```

```
  def get_tasks(self):
      return self.tasks

  def complete_task(self, task_id):
      task = self.find_task(task_id)

      if task is None:
          raise ValueError("Task not found.")

      if task["completed"]:
          return False

      task["completed"] = True
      self.save()

      return True

  def delete_task(self, task_id):
      task = self.find_task(task_id)

      if task is None:
          raise ValueError("Task not found.")

      self.tasks.remove(task)
      self.save()

  def clear_completed(self):
      original_count = len(self.tasks)

      self.tasks = [
          task
          for task in self.tasks
          if not task["completed"]
      ]
```

```
        self.save()

        return original_count - len(self.tasks)

    def find_task(self, task_id):
        for task in self.tasks:
            if task["id"] == task_id:
                return task

        return None

    def save(self):
        self.storage.save_tasks(self.tasks)
```

---

7. Complete Frontend / CLI Code

main.py

```
import argparse

from todo import TodoManager


def display_tasks(tasks):

    if not tasks:
        print("\nNo tasks found.")
        return

    print("\nYour Tasks")
    print("-" * 45)
```

```
    for task in tasks:

        status = "□" if task["completed"] else " "

        print(
            f'{task["id"]:>2}. [{status}] {task["title"]}'
        )

    print("-" * 45)


def create_parser():

    parser = argparse.ArgumentParser(
        description="A simple To Do CLI application."
    )

    subparsers = parser.add_subparsers(
        dest="command"
    )

    # Add command

    add_parser = subparsers.add_parser(
        "add",
        help="Add a new task."
    )

    add_parser.add_argument(
        "title",
        help="Task description."
    )

    # List command
```

```
    subparsers.add_parser(
        "list",
        help="Show all tasks."
    )

    # Complete command

    complete_parser = subparsers.add_parser(
        "complete",
        help="Mark a task as completed."
    )

    complete_parser.add_argument(
        "id",
        type=int,
        help="Task ID."
    )

    # Delete command

    delete_parser = subparsers.add_parser(
        "delete",
        help="Delete a task."
    )

    delete_parser.add_argument(
        "id",
        type=int,
        help="Task ID."
    )

    # Clear command
```

```
    subparsers.add_parser(
        "clear",
        help="Delete all completed tasks."
    )

    return parser


def main():

    parser = create_parser()
    args = parser.parse_args()

    manager = TodoManager()

    try:

        if args.command == "add":

            task = manager.add_task(args.title)

            print(
                f'Added task #{task["id"]}: {task["title"]}'
            )

        elif args.command == "list":

            display_tasks(manager.get_tasks())

        elif args.command == "complete":

            changed = manager.complete_task(args.id)

            if changed:
```

```
            print(
                f"Task #{args.id} marked as completed."
            )
        else:
            print(
                f"Task #{args.id} is already completed."
            )

    elif args.command == "delete":

        manager.delete_task(args.id)

        print(
            f"Task #{args.id} deleted."
        )

    elif args.command == "clear":

        count = manager.clear_completed()

        print(
            f"Removed {count} completed task(s)."
        )

    else:

        parser.print_help()

except ValueError as error:

    print(f"Error: {error}")


if __name__ == "__main__":
```

```
    main()
```

---

8. How Frontend and Backend Connect

The CLI frontend sends commands to the backend.

For example:

```
python main.py add "Learn Python"
```

The process is:

```
Terminal Command
      
argparse
      
main.py
      
TodoManager
      
TaskStorage
      
tasks.json
```

For listing:

```
python main.py list
```

The process becomes:

```
tasks.json
```

- TaskStorage
- TodoManager
- main.py
- Terminal

---

9. How to Run

Open the terminal inside the project folder:

```
cd to_do_cli
```

Run:

```
python main.py
```

You will see the available commands.

You can also directly use:

```
python main.py --help
```

`argparse` automatically generates help and usage information for the CLI.

---

10. Using the Application

Add a task

`python main.py add "Learn Python"`

Output:

```
Added task #1: Learn Python
```

Add another:

`python main.py add "Build a Python project"`

View tasks

`python main.py list`

Example:

```
Your Tasks
-----------------------------------------------
 1. [ ] Learn Python
 2. [ ] Build a Python project
-----------------------------------------------
```

Complete a task

`python main.py complete 1`

Output:

```
Task #1 marked as completed.
```

Now:

```
python main.py list
```

Output:

```
Your Tasks
------------------------------------------------
 1. [□] Learn Python
 2. [ ] Build a Python project
------------------------------------------------
```

Delete a task

```
python main.py delete 2
```

Output:

```
Task #2 deleted.
```

Clear completed tasks

```
python main.py clear
```

Output:

```
Removed 1 completed task(s).
```

---

11. Basic Testing

Test the following commands:

`python main.py add "Test task"`

`python main.py list`

`python main.py complete 1`

`python main.py list`

`python main.py delete 1`

`python main.py clear`

Also test an invalid task ID:

`python main.py complete 999`

Expected result:

`Error: Task not found.`

Test an empty task:

`python main.py add ""`

Expected result:

`Error: Task cannot be empty.`

---

12. Small Improvements

The project can later be extended with:

- Task priorities
- Due dates
- Categories
- Search
- Task editing
- Sorting
- Colored terminal output
- Recurring tasks
- SQLite database
- Export to CSV
- Interactive menu mode
- Task statistics

---

13. Project Architecture

```
main.py
  │
  ▼
CLI / argparse
  │
  ▼
TodoManager
  │
  ▼
TaskStorage
  │
  ▼
tasks.json
```

The project demonstrates a simple but important application pattern:

CLI Input Argument Parsing Business Logic Storage Output

The same architecture can later be expanded into a web application by replacing the CLI layer with a web frontend while keeping much of the task-management logic.

Visit haas.dev for more resources and guides.