

Aufgabe: Dein erstes Python-CLI-Projekt

Du baust ein kleines Kommandozeilen-Tool (CLI = Command Line Interface) – ein Programm, das man über das Terminal mit Befehlen und Optionen bedient, z.B. so wie `git status` oder `git commit -m "..."`.

Ziel-Tool: ein Aufgaben-Manager, den du später so aufrufst:

```
python cli.py add "Wäsche waschen"
```

```
python c
```

Teil 1: Projekt-Setup im Terminal (Schritt für Schritt)

Schritt 1 – Projektordner anlegen

```
mkdir aufgaben-cli
```

```
cd aufg
```

Schritt 2 – Virtuelle Umgebung anlegen & aktivieren

```
python3 -m venv venv
```

```
source v
```

Du erkennst die Aktivierung am `(venv)` vor deiner Terminal-Zeile.

Schritt 3 – Projektdateien anlegen

```
touch cli.py
```

```
touch a
```

- `cli.py` → dein eigentliches Programm

- `aufgab`

In `aufgaben.json` schreibst du am Anfang manuell eine leere Liste rein:

```
[]
```

Schritt 4 – Git initialisieren (optional, aber gute Übung)

```
git init
```

```
echo "v
```

Schritt 5 – Grundgerüst testen

Schreib testweise in `cli.py`:

```
print("CLI läuft!")
```

Dann im Terminal ausführen:

```
python cli.py
```

Wenn "CLI läuft!" erscheint, ist dein Setup korrekt.

Teil 2: Was du für die Aufgabe brauchst

Das eingebaute Modul `argparse` – damit liest Python die Befehle/Argumente ein, die du im Terminal nach `python cli.py` eintippst. Kein `pip` install nötig, ist Teil der Standardbibliothek.

Minimal-Grundgerüst, um zu verstehen wie `argparse` funktioniert (das ist noch NICHT die Lösung deiner Aufgabe, nur ein Beispiel zum Verstehen des Prinzips):

```
import argparse
```

Aufruf im Terminal:

```
python cli.py Nalan
```

Ausga

Für mehrere Befehle (`add`, `list`, `done`) brauchst du Subparsers – das Prinzip dahinter:

```
import argparse
```

Aufruf:

```
python cli.py add "Wäsche waschen"
```

Ausga

Das ist bewusst nur ein Mini-Beispiel zum Prinzip verstehen – deine eigentliche Aufgabe ist umfangreicher (siehe unten).

Teil 3: Deine eigentliche Aufgabe

Baue in `cli.py` einen vollständigen Aufgaben-Manager mit folgenden Anforderungen:

Muss-Anforderungen

1. `python cli.py add "Titel"`

Fügt e

2. `python cli.py list`

Zeigt a

```
```text
```

[1] Wä

3. `python cli.py done 1`

Markie

4. Daten müssen persistent sein — nach Neustart des Programms müssen die Aufgaben noch da sein (deshalb `aufgaben.json`).

Zusatz-Anforderungen (wenn die Basis läuft)

5. `python cli.py remove 1` — löscht eine Aufgabe anhand der ID.

6. pytho

## Teil 4: Wie du es angehst (Struktur-Tipp, ohne fertigen Code)

Bau dir das Programm in kleinen, testbaren Schritten auf – nach jedem Schritt im Terminal testen, bevor du weitermachst:

1. Zuerst: Funktionen `aufgaben_laden()` und `aufgaben_speichern(liste)`, die die JSON-Datei lesen/schreiben — teste sie isoliert, bevor irgendwas mit `argparse` passiert.

2. Dann

Fragen, die dir beim Strukturieren helfen:

- Welche Funktion ist für "Datei lesen/schreiben" zuständig, welche für "Argumente verarbeiten"? (Trenne diese zwei Verantwortlichkeiten!)

- Woher

Wenn du feststeckst

Sag mir, an welchem der 4 Basis-Schritte (`add` / `list` / `done` / `speichern`) du gerade bist und was genau nicht funktioniert (Fehlermeldung oder unerwartetes Verhalten) — dann gehen wir das gemeinsam Schritt für Schritt durch, ohne dass ich dir gleich den kompletten Code gebe.