Abstract of a CLI-Based Habit Tracking Application with Predefined Habits and Analytics

GitHub Repository: https://github.com/toukoalvine/OOFPP_Habit_Tracker

1. Introduction

A habit tracking application works like a digital coach, helping people stay focused in today's busy and self-improvement-driven world. As more individuals look for ways to build self-discipline, the need for easy-to-use digital tools has grown. This project presents a command-line interface (CLI) habit tracker built with modular, object-oriented Python. It also includes basic analytics features. Users can track predefined habits, monitor their consistency, and analyze behavior patterns. My application is designed to be simple, flexible, and easy to test.

2. Main features of the habit tracker

This CLI-based habit tracking application helps users build and maintain consistent routines through structured tracking and analysis. Users can select a habit from a predefined list and specify the periodicity (daily or weekly) along with an end date (default is 4 weeks). It allows users to:

- View all active habits in a clear list format
- Update the periodicity and duration of existing habits
- Delete habits that are no longer relevant
- Mark habits as completed, with automatic check-off tracking and streak maintenance
- Analyse progress by reviewing active habits, filtering by periodicity, and identifying the longest maintained streaks

3. System architecture

The system follows modular architecture and is developed using Python, applying both object-oriented and functional programming principles. The following are the python modules of the application:

- **functions_main.py**: provides functions used in the main menu
- **main.py**: control the Command Line Interface (CLI) and user interaction loop
- **habit.py**: contains the Habit class with attributes (habit_id, task, periodicity, start-date and end_date) and associated methods for managing habits.
- **db.py**: manage SQLite-based persistent storage, including table creation for habits, checked-offs, and streaks.
- **Analyse.py**: displays all habits, filters the habits based on periodicity and the longest streak for a habit.

This ensures the application is well-organized, scalable, and easy to maintain

Alvine Gadeu, Student number: UPS10745343

4. Data storage

Initially I wanted to use a JSON based system, and I later switched to SQLite3, which allows efficient storage of checked-off events and easy calculation of the streaks. The database (db.py) stores all habits completion data, making it possible to dynamically calculate the streaks. It contains the following tables:

- Habits table (with user_id and habits attributes)
- Checked_offs table (timestamped user actions)
- Streaks table (to support personalized tracking)

The streak is calculated based on the time-difference logic aligned with the periodicity (i.e., daily = 1-day intervals, weekly = 7-day intervals). If a checkoff is missed in the current period, the streak ends. The checked_offs are then stored then as raw event data, which allows retrospective analysis and derivation of other metrics like current streaks, maximum streaks.

5. Testing and validation

Unit testing was implemented using pytest and unittest.mock. CLI interaction paths were tested via simulated user input calls patched with mock.patch for quality assurance. Test coverage includes:

- Menu logic (for options 1–6 and invalid inputs).
- Integration testing between user interface layer and analytics functions.
- Validation of analytics output against known test data.
- Habit modul testing with valid and invalid habit_id, periodicity, end_date and invalid end_date format

Multiple anomalies were detected during this testing phase and corrected.

6. Challenges

As a beginner with no programming experience, one of the biggest challenges for me was designing a user-friendly command-line interface. I had to make sure the menu, options, and prompts were clear and easy to follow. I also found testing and debugging quite overwhelming at first. However, this process helped me better understand how users interact with software and improved my knowledge of test-driven development.

7. Conclusion

This CLI-based habit tracker has been cleanly designed, and test approved for personal use. It contains structured data storage and menu, and user-driven habit management. But compared to what is nowadays commercially available, this is a good base with many extensions' possibilities like notifications to support users' behaviors, GUI integration, visualizations, user authentication, new habit creation and potential deployment as a web application using flask or similar.

Object Oriented and Functional Programming with Python

Alvine Gadeu, Student number: UPS10745343