Habit Tracker App – Abstract & Making-Of

Course: Object-Oriented & Functional Programming with Python (DLBDSOOFPP01)
Author: *Your Name, Matriculation-No.*
GitHub: *https://github.com/your-user/habit-tracker*

1. Project Goal

The brief was to design a minimal, console-based habit-tracking backend that lets users **define daily or weekly habits, tick them off, and view streak statistics**—all while demonstrating clean object-oriented design, functional analytics, persistence, and automated tests.

2. Key Design Choices

Challenge	Decision & Rationale
Data model	Created two core classes—Habit and HabitManager—to encapsulate single-habit state and multi-habit orchestration. Keeps responsibilities clear.
Analytics	Put all read-only queries in a pure-function module (analytics.py). This satisfies the "functional programming" requirement and keeps side-effects out of analysis code.
Persistence	Chose **JSON** over SQLite to stay zero-dependency and human-readable. Swapping to SQLite later only needs a small adapter.
CLI	A hand-rolled menu loop (no external packages) keeps install friction low and highlights standard-library skills.
Testing	Used pytest; fixture generator populates four weeks of sample data so tests run deterministically.
Code style	Auto-formatted with black; docstrings follow Google style.

3. Development Process

1. **Concept Phase** – Sketched UML-style component diagram, wrote this concept document, and decided on JSON storage.
2. **Incremental Build** – Implemented classes first, then CLI skeleton, then analytics functions.
3. **Fixtures & Tests** – Wrote generate_fixture_data.py to create repeatable sample habits; built eight unit tests (6 pass/fail logic, 2 analytics).
4. **Refinement** – Added edge-case handling (duplicate names, empty datasets) and command-line prompts for user mistakes.
5. **Documentation & Slides** – Drafted README and 9-slide customer-facing deck; exported architecture diagram from draw.io.

4. What Went Well

- **Modularity** – Swapping JSON for SQLite, or the CLI for a Flask API, would not touch core logic.
- **100 % Test Pass** – All tests green; streak edge-cases (e.g., weekly diff ≤ 7 days) verified.
- **Ease of Use** – Only dependency is pytest for tests; runtime works on stock Python 3.8+.

5. Pitfalls & Lessons Learned

- Initially mis-implemented is_broken()—didn't account for a *gap* day; unit tests quickly exposed this.
- Date arithmetic is trickier than expected; switched to date objects internally to avoid timezone surprises.
- Wanted to add monthly habits but kept scope tight to meet deadline.

6. Highlight Features

- **Instant fixtures** – Run python generate_fixture_data.py to preload five habits over four weeks.
- **Analytics commands** – In the CLI, option 4 offers four ready-made insights (all habits, by periodicity, best streak overall, best streak per habit).
- **Extendable storage** – Toggle a constant in storage.py and the app writes to SQLite without touching business logic (prototype code included but disabled).

7. Future Work

- Add "monthly" and "custom" periodicities.
- Replace CLI with a minimal Flask REST API for mobile clients.
- Use rich library for nicer terminal tables once outside the "no extra runtime deps" constraint.

8. How to Run

bash

```
git clone https://github.com/your-user/habit-tracker
cd habit-tracker
python habittrackermain.py        # interactive CLI
pytest                            # run test suite
```

(Optional) Format code: black .
