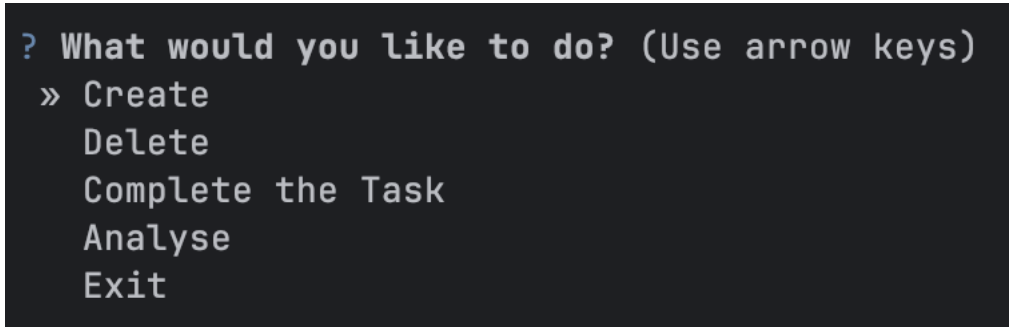
Habit Tracking App

- Ernesta Ippolitova. Matriculation: 92208665
- Object Oriented and Functional Programming with Python (OOFPP01)
- B.Sc. Computer Science
- Tutor: Prof. Dr. Max Pumperla
- GitHub repository link: https://github.com/Ernesta-Ip/Tracker_App

Introducing Habit Tracker

The application is designed to track on personal habits and achievements of goals. Basic principles:

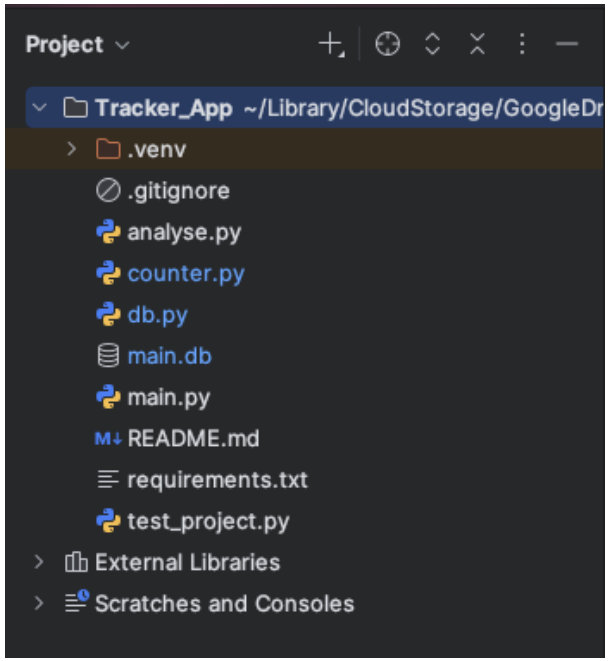
- CLI-based app for speed and automation. User interacts with the app via CLI (using the library *questionary*)
- Flexible periodicity of habits: three type of periods are available (daily, weekly, monthly)
- Analytics: counts, streaks, grouping



Callouts:

- Python 3.13
- DB created with SQLite
- Questionary for interactive prompts
- Tests using *pytest*

Architecture and Components

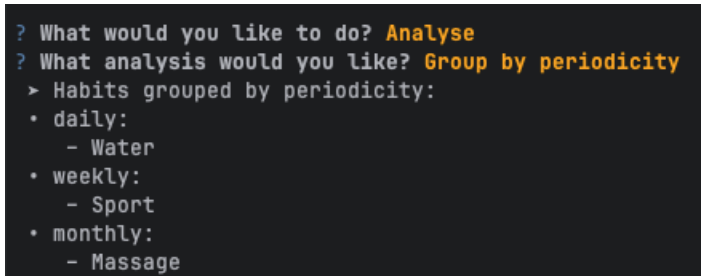


The application consists of the following modules:

- **main.py**
  Implements the CLI interface, connects to the database, and invokes functions from the other modules.
- **db.py**
  Manages data storage and all database operations.
- **counter.py**
  Defines the *Counter* class (which represents a habit) and provides functions to add or delete events.
- **analyse.py**
  Offers analytical routines for counting events, calculating the current streak for a given habit, and finding the longest streak across all habits.
- **test_project.py**
  Contains unit tests to verify the correctness of the application's functions.
- **README.md**
  A brief overview of the application plus installation instructions.
- **requirements.txt**
  A list of the Python libraries required for the application to run.

main.py

main.py is the interactive front end: it handles all user prompts and display logic, validates inputs, and orchestrates calls into the database, counter, and analyse layers to perform every CRUD and reporting operation.

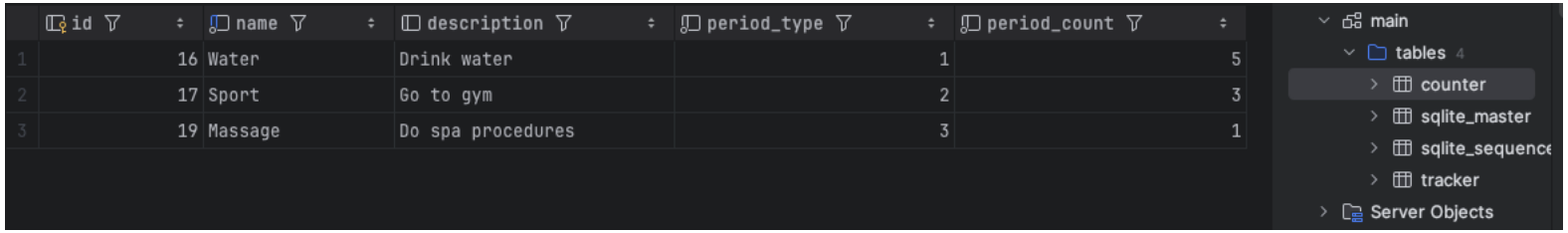


main.py is the CLI entry point that:

- **Initializes** the SQLite DB.
- **Loops** a menu (Create, Delete, Complete, Analyse, Exit) via questionary.
- **Create** prompts for name/description/periodicity/target, then saves with *add_counter().*
- **Delete**: confirms deletion of records and calls *delete_event().*
- **Complete:** records a habit check-off (with optional manual timestamp) via *add_event().*
- **Analyse:** dispatches to *analyse.py* for counts, listings, groupings, and streaks.
- **Exit:** ends the loop.

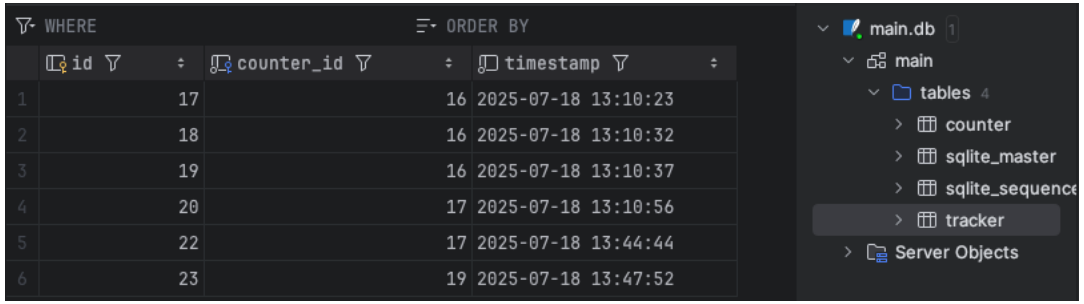
db.py

Database and Storage
Implemented in db.py module: creation of two tables and functions to manipulate with data in tables (*create, insert, select, delete*)



	id	name	description	period_type	period_count
1	16	Water	Drink water	1	5
2	17	Sport	Go to gym	2	3
3	19	Massage	Do spa procedures	3	1

Table *counter*: keeps information on habits.
id, name, description of habit, period_type (daily, weekly or monthly), period_count (how many time per period type the task should be checked-off



WHERE ORDER BY

	id	counter_id	timestamp
1	17	16	2025-07-18 13:10:23
2	18	16	2025-07-18 13:10:32
3	19	16	2025-07-18 13:10:37
4	20	17	2025-07-18 13:10:56
5	22	17	2025-07-18 13:44:44
6	23	19	2025-07-18 13:47:52

Table *tracker*: keeps information on the events for the given habit (checking-off the tasks)
id, counter id (foreign key) and timestamp of the task completed (datetime format).

counter.py

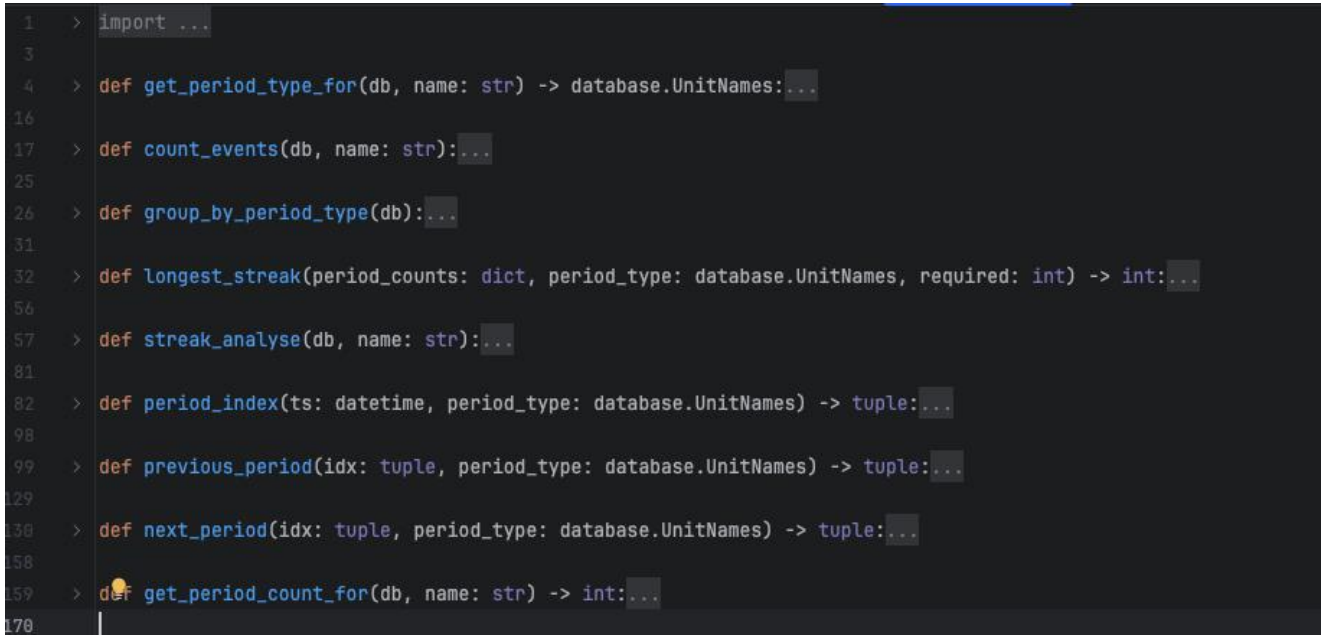
```python
class Counter:  2 usages  Ernesta Ippolitova

    def __init__(self, name: str, description: str, period_type: UnitNames, period_count: int):  Ernesta Ippolitova
        """..."""
        self.name = name
        self.description = description
        self.period_type = UnitNames(period_type)
        self.period_count = period_count
        self.count = 0

    def __str__(self):  Ernesta Ippolitova
        """..."""
        return f"{self.name}: {self.count} – {self.period_count}× per {self.period_type.label}"
```

The **counter** module encapsulates the core “habit” entity (class Counter) and the operations the user can perform on it.
Class Counter

- holds habit’s metadata (name, description, periodicity)
- tracks in-memory count
- implements __str__() to render user-frendly summary

In this module *add_event()* and *delete_event()* functions look up a habit by name and add event to the existing habit or remove the habit and all associated tracker entries.

analyse.py



```
import ...

def get_period_type_for(db, name: str) -> database.UnitNames:...

def count_events(db, name: str):...

def group_by_period_type(db):...

def longest_streak(period_counts: dict, period_type: database.UnitNames, required: int) -> int:...

def streak_analyse(db, name: str):...

def period_index(ts: datetime, period_type: database.UnitNames) -> tuple:...

def previous_period(idx: tuple, period_type: database.UnitNames) -> tuple:...

def next_period(idx: tuple, period_type: database.UnitNames) -> tuple:...

def get_period_count_for(db, name: str) -> int:...
```

The analyse module is responsible for turning raw habit-completion data into meaningful metrics and streak insights:

- **Configuration lookups -** fetches habit's periodicity and its required count per period from the database.
- **Basic summaries -** counts the total number of completion of events recorded for a habit, delegates to the DB to list which habits fall under each periodicity.

- **Period bucketing -** maps each timestamp into a tuple key representing its day, week or month, computes adjacent period tuples.
- **Streak computation:**
  - *longest_streak:* finds the longest run of consecutive periods where the count meets or exceeds the required threshold.
  - *streak_analyse*: fetches the raw timestamps for a habit, buckets them, builds the per-period count dict, then calls *longest_streak()* to return both the streak length and the habit's period type.

test_project.py

Implemented via **pytest + Temp DBs:** each test uses a fresh SQLite file with foreign keys enabled.

For the tests 4 habits were created (run, yoga, water, gym) and events were added to the habit tracker.

Schema & CRUD:

- Verifies table creation (counter, tracker)
- Tests adding, finding, listing, and deleting habits

Data Retrieval:

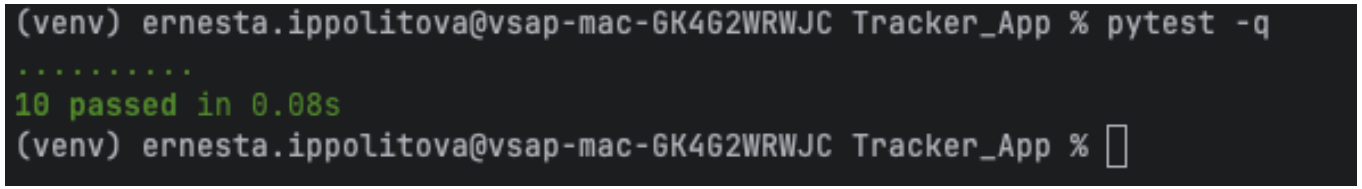
- Confirms event insertion and fetching via increment_counter/get_counter_data
- Checks total counts and grouping by period type

Period Logic:

- Ensures period_index, previous_period, next_period handle day/week/month correctly

Streak Analysis:

- Unit-tests for longest_streak on synthetic data
- End-to-end check with streak_analyse for real timestamps



Thank you for your time!