Conception Phase: Habit Tracking App

Introduction

This document outlines the concept and design for a habit tracking app, detailing the technical foundation, class structures, data storage, user interactions, and overall flow. The goal is to implement essential functionalities of a habit tracker using object-oriented and functional programming in Python (OOPP).

Technical Foundation

The app will be implemented in Python 3.7 or later. The following tools and libraries will be utilized:

- Data Storage: SQLite3 for persisting habit data.
- CLI Tools: ‘click’ for command-line interface
- Testing: ‘pytest’ for unit testing
- Date Handling: ‘datetime’ for managing dates and times.
- Date Serialization: ‘json’ for reading and writing data.

Core Components

1. Habit Class

- Attributes: ‘name’ – the name of the habit
  ‘periodicity’ – the frequency of the habit (e.g., daily, weekly).
  ‘creation_date’ – the date the habit was created
  ‘completion_dates’ – a list of dates when the habit was completed.
- Methods: ‘check_off(date)’ – marks a task as completed on the given date.
  ‘is_broken()’ – checks if the habit has been broken.
  ‘streak()’ – returns the current streak of consecutive completions.
  ‘longest_streak()’ – calculates the longest streak achieved.

2. HabitManager Class

- Attributes: ‘habits’ – a list of habit objects.
- Methods: ‘add_habit(name, periodicity)’ – creates and adds a new habit.
  ‘delete_habit(name)’ – removes a habit by name.

‘list_habits()’ – returns a list of all habits.

‘get_habit(name)’ – retrieves a habit by name.

‘analyze_habits()’ – provides analysis on habits (e.g., longest streak, habits by periodicity).

3. Analytics Module

Implemented using functional programming paradigms.

Functions

- ‘list_current_habits(manager)’: returns all tracked habits.
- ‘list_habits_by_periodicity(manager, periodicity)’: returns habits with the specified periodicity.
- ‘longest_run_streak(manager)’: returns the longest run streak among all habits.
- ‘longest_run_streak_for_habit(manager, habit_name)’: returns the longest run streak for a given habit.

Data Storage

Data will be stored using SQLite3 for reliability and scalability. The database will have tables for users, habits, and completions:

- Habits Table: Stores habit details (name, periodicity, creation date).
- Completions Table: Stores completion dates for each habit.

User Interaction

1. Command-Line Interface (CLI)
   Users can create, delete, and list habits, and mark tasks as completed via a CLI.
   Commands:
   - ‘create_habit’: Adds a new habit.
   - ‘delete_habit’: Removes an existing habit.
   - ‘lists_habits’: Lists all habits.
   - ‘complete_task’: Marks a task as completed.
   - ‘analyze_habits’: Analyzes habits and provides insights.

User Flow

1. **Initialize the App**: User installs and runs the app.
2. **Create Habits**: User creates new habits using the ‘create_habit’ command.
3. **Complete Tasks**: User marks tasks as completed with the ‘complete_task’ command.
4. **Analyze Habits**: User runs ‘analyze_habits’ to get insights into their habits.
5. **Manage Habits**: User lists or deletes habits as needed.

Example Data

Predefined habits for testing:

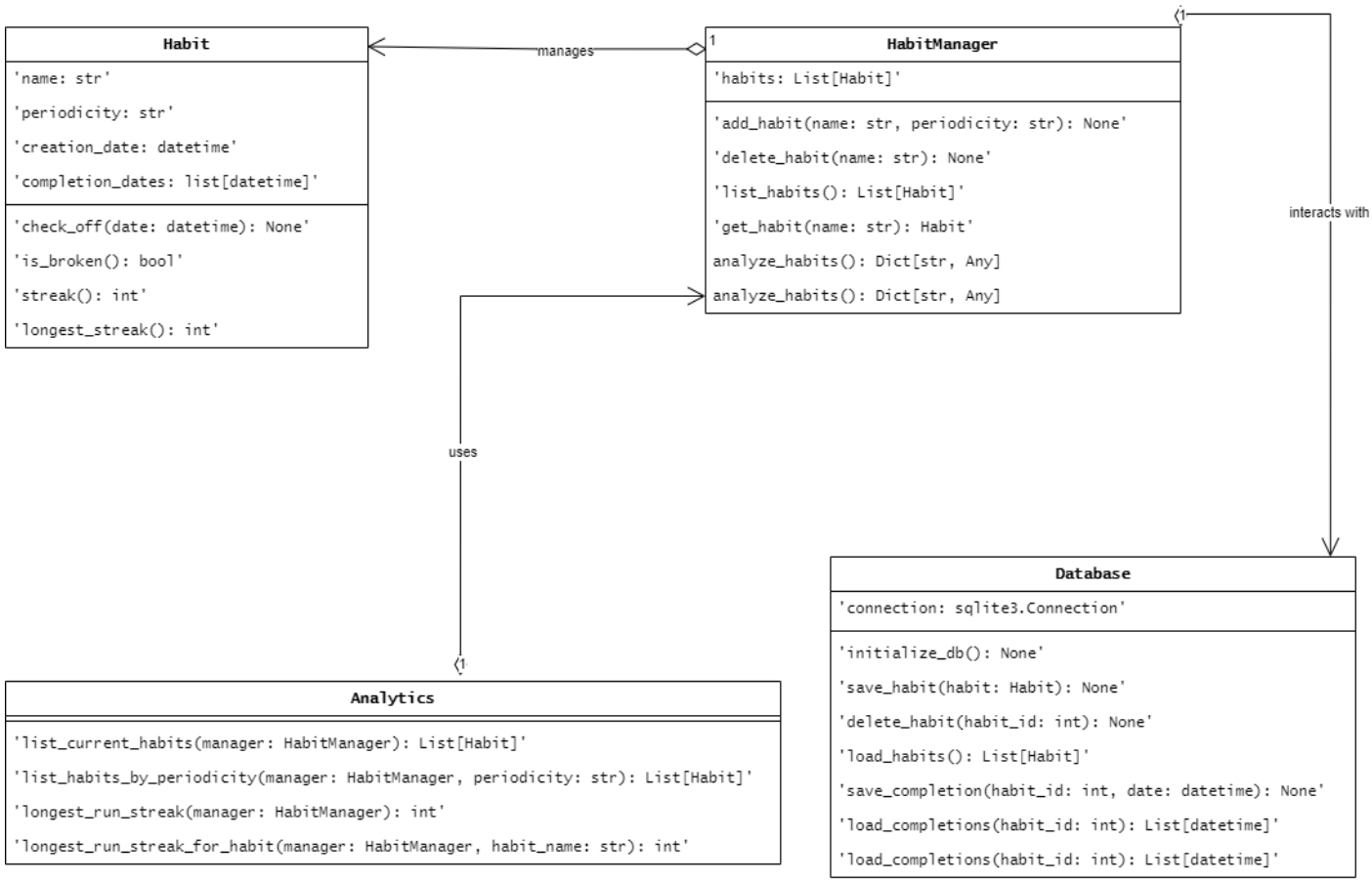
1. Daily Habit: ‘Drink Water’
2. Weekly Habit: ‘Exercise’

UML Diagram

The UML diagram visualizes the interaction of components, showing how the ‘Habit’, ‘HabitManager’, and analytics functions interact with each other and the database.

The diagram includes:

- Habit Class with its attributes and methods.
- HabitManager Class managing multiple ‘Habit’ objects.
- Analytics Functions operating on ‘HabitManager’ to analyze habits.
- Database storing and retrieving data for ‘Habit’ objects.



Conclusion

This conception document outlines the plan for developing a habit tracking app. The technical foundation, class structures, data storage, user interactions, and overall flow have been detailed to ensure a smooth implementation phase. The UML diagram provides a clear visualization of the components and their interactions, guiding the development phase.