The habit tracking app will allow users to create, manage, and analyze their habits. The core functionality revolves around creating habits, tracking their completion, and providing insights into habit performance and consistency.
To create new habits and complete them following attributes and methods are required:

Class: Habit

Attributes:

name: string representing the name of the habit (5 predefined habit).
period: string indicating the periodicity of the habit ('daily' and 'weekly').
created_at: timestamp indicating when the habit was created.
completed_at: list of timestamps indicating when the habit was completed.
streak: integer tracking the current streak of the habit.
longest_streak: integer tracking the longest streak ever achieved for the habit.
missed_periods: integer tracking the number of periods the habit was missed.

Methods:

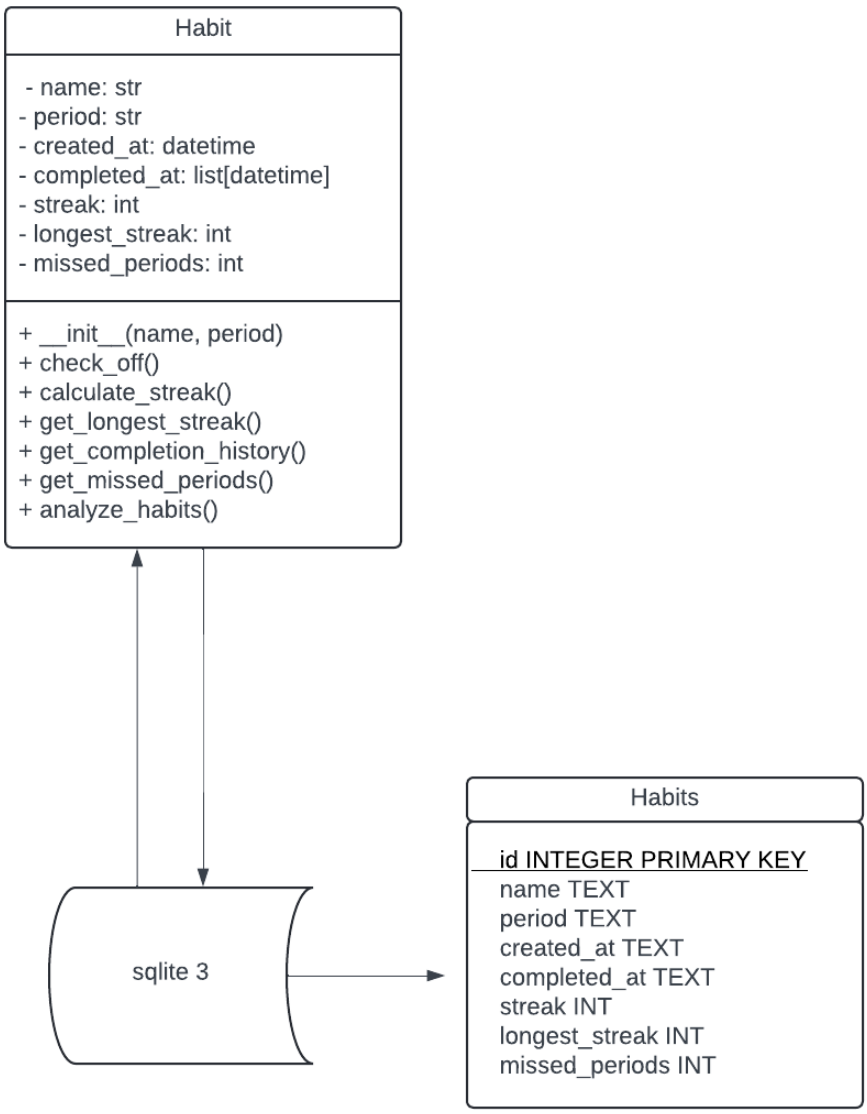
init: Initializes a new habit with a name, period, and creation timestamp.
check_off: Marks the habit as completed for the current period.
calculate_streak: Calculates the current streak based on the completion history.
get_longest_streak: Returns the longest streak achieved for the habit.
get_completion_history: Returns the history of habit completion.
get_missed_periods: Returns the number of periods the habit was missed.
analyze_habits: Provides analysis and insights on the habit.

The SQLite table to store habit data should have columns for each attribute of the Habit class that needs to be persisted. The following schema aligns with the previous Habit class:

name: The name of the habit (text)
period: The period of the habit (text)
created_at: The creation timestamp of the habit (text)
completed_at: A list of completion timestamps (text)
streak: The current streak count (integer)
longest_streak: The longest streak count (integer)
missed_periods: The count of missed periods (integer)

User Flow:

1. Create a new habit: The user creates a new habit by specifying its name and period (daily or weekly). The habit is initialized with the current date and time, and the completed_at list, streak, longest_streak, and missed_periods attributes are set to their initial values.
2. Check off the habit: The user checks off the habit once they have completed it. This increments the streak and updates the completed_at list. The calculate_streak method is called to determine the new streak and longest_streak values.
3. View habit analysis: The user can view an analysis of their habit, which includes the name, period, creation date, completion history, current streak, longest_streak, and the number of missed_periods.



Introduction

Key features:

•Habit creation with customizable periods

•Progress tracking with streaks

•Detailed habit analysis

•Data persistence using SQLite

Aim of the work:

The following introduced habit app allows users to create, manage and analyze habits in a time-resolved manner to stay consistent
and reach their personal habit goals.

System Architecture

System overview:

The application is built using a Python-based Habit class that interacts with an SQLite database to store, load and manage habit data.

- **Habit Class:** Core for habit management
- **SQLite Database:** Persistent storage for habit data
- **User Interaction:** Command-line interface for habit operations

```
habit_tracker/
│
├── habit.py          # Contains the Habit class with diverse methods
├── setup.py          # Handles the setup of predefined habits
├── interface.py      # Contains the user interface logic
├── main.py           # Entry point of the application
└── habits.db         # Stores all the habits
```

Class ‘Habit’

Purpose:

The Habit class is the central component of the application, responsible for handling all habit-related operations.

__init__(self, name, period, created_at, completed_at, streak, longest_streak, missed_periods)
•**Purpose:** Initializes a new Habit instance with the specified name and periodicity (daily/weekly). It also sets up initial attributes like creation time and counters for streaks.
create_table()
•**Purpose:** A static method that ensures the SQLite database has the necessary table (habits) to store habit data. It's called upon initializing the class to set up the database structure.
save_to_db(self)
•**Purpose:** Saves or updates the current state of the habit instance to the database. This includes details like completion times, streaks, and missed periods.
load_from_db(self)
•**Purpose:** Loads the most recent data of the habit from the database into the instance. This is essential for updating or analyzing the habit's progress.
check_off(self)
•**Purpose:** Marks the habit as completed for the current period. It records the completion time, recalculates streaks, and saves the updated state to the database.
calculate_streak(self)
•**Purpose:** Calculates the current streak based on completion history. It determines if the user has maintained the habit without missing periods and updates the longest streak accordingly.
check_habit_exists(name, period)
•**Purpose:** A static method that checks whether a habit with the given name and periodicity already exists in the database. This prevents duplicate entries.
get_all_habits()
•**Purpose:** Retrieves a list of all unique habit names stored in the database. Useful for displaying all tracked habits to the user.
get_habits_by_period(period)
•**Purpose:** Fetches habits filtered by their periodicity (daily or weekly). Helps users view habits based on their frequency.
get_longest_run_streak_all()
•**Purpose:** Retrieves the longest streaks for all habits. Useful for comparative analysis across different habits.
get_longest_run_streak(name, period)
•**Purpose:** Fetches the longest streak for a specific habit based on its name and periodicity.

SQLite Database Integration

Purpose:

SQLite is used to store habit data persistently, allowing users to maintain their progress at any session.

id	name	period	created_at	completed_at	streak	longest_streak	missed_periods
1	Exercise	Daily	2022-01-01	2022-01-01	3	5	0
2	Reading	Weekly	2022-01-03	2022-01-10	2	3	1

Persistence:

All habit data (streaks and completion history) is stored in the SQLite database, ensuring that progress is never lost.

Command Line User Interface

Menu Options:

- Add a new habit
- Check off a habit
- View habit analysis
- Exit

Add Habit:

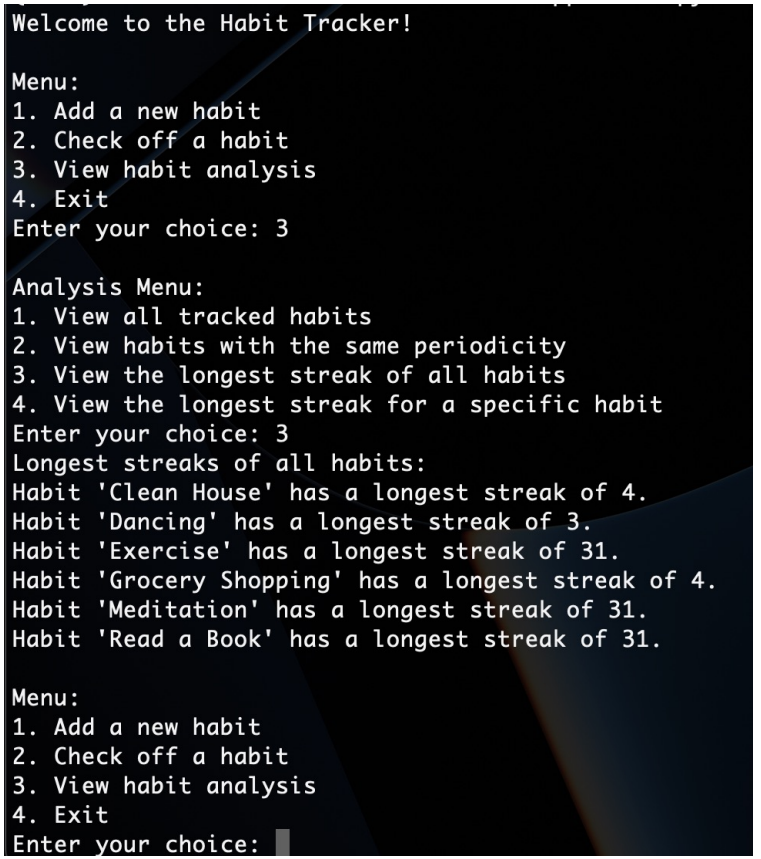
Prompt user for habit name and period
Check if habit already exists in database
If not, create new habit and save to database

Check Off Habit:

Prompt user for habit name and period
Load habit from database and check off

View Habit Analysis Options:

- View all tracked habits
- View habits with same periodicity
- View longest streak of all habits
- View longest streak for specific habit



Updating the Streak

```python
def calculate_streak(self):

    if not self.completed_at:
        self.streak = 0
        return

    created_at = self.created_at
    now = self.completed_at[-1]  # The current check-off time

    if self.period == 'daily':
        # Calculate the difference in days between the check-off and the last habit creation/check-off
        delta = (now.date() - created_at.date()).days

        if delta <= 1:
            self.streak += 1
        elif delta > 1:
            self.missed_periods += delta - 1
            self.streak = 1  # Reset streak since a day was missed

    elif self.period == 'weekly':
        # Calculate the difference in weeks between the check-off and the last habit creation/check-off
        delta = (now.date() - created_at.date()).days // 7

        if delta <= 1:
            self.streak += 1
        elif delta > 1:
            self.missed_periods += delta - 1
            self.streak = 1  # Reset streak since a week was missed

    # Update the longest streak if the current streak is the highest
    if self.streak > self.longest_streak:
        self.longest_streak = self.streak
```

•**Check-off**: When a habit is checked off, the current date and time are added to the completed_at list.

•**Calculate Streak**: The calculate_streak method is called to update the streak based on the period (daily or weekly).

•**Daily Period**:

- Calculate the difference in days between the check-off and the last habit creation/check-off.
- If the difference is 1 day or less, increment the streak.
- If the difference is more than 1 day, reset the streak and increment the missed periods.

•**Weekly Period**:

- Calculate the difference in weeks between the check-off and the last habit creation/check-off.
- If the difference is 1 week or less, increment the streak.
- If the difference is more than 1 week, reset the streak and increment the missed periods.

•**Update Longest Streak**: If the current streak is greater than the longest streak, update the longest streak.

Project Overview

This project involves creating a Habit Tracking application aimed at assisting users in establishing, tracking, and managing their habits over time. The core components include the Habit class, which manages individual habits, and several supporting modules that handle the setup, user interface, and data persistence. This report provides a concise technical overview of the project, highlighting the development journey, notable achievements, challenges faced, and areas where future enhancements could be made.

Technical Approach

At the heart of the application lies the Habit class, which encapsulates the essential features and operations of a habit. This class includes attributes such as the habit's name, period, creation and completion dates, streaks, the longest streak, and missed periods. These elements enable comprehensive tracking of a habit's lifecycle, from its inception to its successful completion or eventual lapse. The class also provides methods to update these attributes, including logging completions, calculating streaks, and managing missed periods, establishing it as the backbone of the application’s habit management capabilities.

Supporting modules include setup.py for initializing predefined habits, interface.py for managing the user interface, and main.py, which serves as the application’s entry point. User data is stored in an SQLite database (habits.db), ensuring interactions are preserved and accessible across sessions.

Successes

The project saw several significant successes. Implementing the Habit class created a flexible and scalable system for managing diverse habits, allowing users to monitor various behaviors over different timeframes. The class's design, incorporating attributes for streaks and missed periods, provided immediate feedback on progress, crucial for maintaining motivation during habit formation.

The modular architecture of the project was another achievement. By segregating the habit management logic (habit.py), setup procedures (setup.py), and user interface (interface.py), the codebase remained organized and maintainable. This modularity facilitated isolated testing of components, contributing to a more robust final product.

Integrating an SQLite database was another success. By storing habit data locally, the application allowed users to continue tracking habits seamlessly, even after closing and reopening the app. This data persistence was vital for the user experience, enabling consistent and uninterrupted habit tracking.

Challenges and Pitfalls

Despite its successes, the project faced several challenges. One major challenge was ensuring accurate streak tracking. The logic required to handle various edge cases, such as missed periods

and updating the longest streak, proved more complex than initially expected. Although these issues were ultimately resolved, they demanded more time and effort than originally anticipated.
Designing a user interface that was both intuitive and comprehensive posed another challenge. Striking a balance between simplicity and functionality was difficult, especially when incorporating feedback mechanisms to keep users informed about their progress. An interface with too much information might overwhelm some users, while others might find it lacking if feedback is too sparse.
Additionally, while SQLite was effective for data storage, it presented limitations regarding scalability. Though suitable for the project's current scope, future iterations might need to explore more robust database solutions to accommodate a growing user base or data volume.

Key Features and Value Addition

A standout feature is the streak tracking mechanism. By calculating and displaying streaks, the application gives users a tangible measure of their progress, which can be highly motivating. This feature significantly enhances the product's value by aligning with the application's primary goal of helping users build and maintain habits.

The application’s modular design is another noteworthy feature. By organizing the code into distinct modules, the project remained maintainable and opened avenues for future expansion. For instance, new features or habit types could be added with minimal disruption to the existing codebase.
Incorporating the SQLite database is another valuable addition. It ensures the application can handle real-world scenarios where users need their habit data to persist across sessions. This greatly enhances the application's usability and reliability.

Conclusion

Developing the Habit Tracking application was a successful endeavor, resulting in a product that effectively aids users in tracking and managing their habits. The project benefited from a well-structured codebase, a flexible habit management system, and a robust data persistence mechanism. However, it also encountered challenges, particularly in streak tracking logic and user interface design. Despite these challenges, the project delivered a valuable tool that aligns with its intended purpose. Future development could focus on improving scalability, refining the user interface, and adding new features to enhance the user experience further.