Title: Development Phase of Habit Tracking App

Subtitle: Implementation and User Interaction

Author: Joe Kariuki

Date: 26/06/2024

Introduction

Overview of the Habit Tracking Benefits

1. Improved productivity and Focus
   - Helps in prioritizing tasks
   - Reduces distractions effectively
2. Builds Discipline and Consistency
   - Establishes a routine
   - Encourages regular behavior
3. Tracks Progress and Motivates
   - Provides visual feedback on achievements
   - Boosts motivation through streaks and goals

Technical Foundation

1. Python 3.7+

Language Choice for Robustness

- Widely supported and stable
- Rich ecosystem of libraries

2. SQLite3

Data Storage for Reliability

- Lightweight, embedded database
- Transactional and ACID-compliant

3. ‘click’

CLI Tool for User Interaction

- Simplifies command-line interface creation
- Supports parameter handling.

4. ‘pytest’

Testing Framework for Reliability

- Fixture-based setup for tests
- Extensible with plugins

5. ‘datetime’

Handling Dates and Times

- Provides classes for manipulating dates and times
- Supports time zone handling

6. ‘json’

Serialization for Data Storage

- Lightweight data interchange format
- Easy to read and write by humans

Core Components

Habit Class

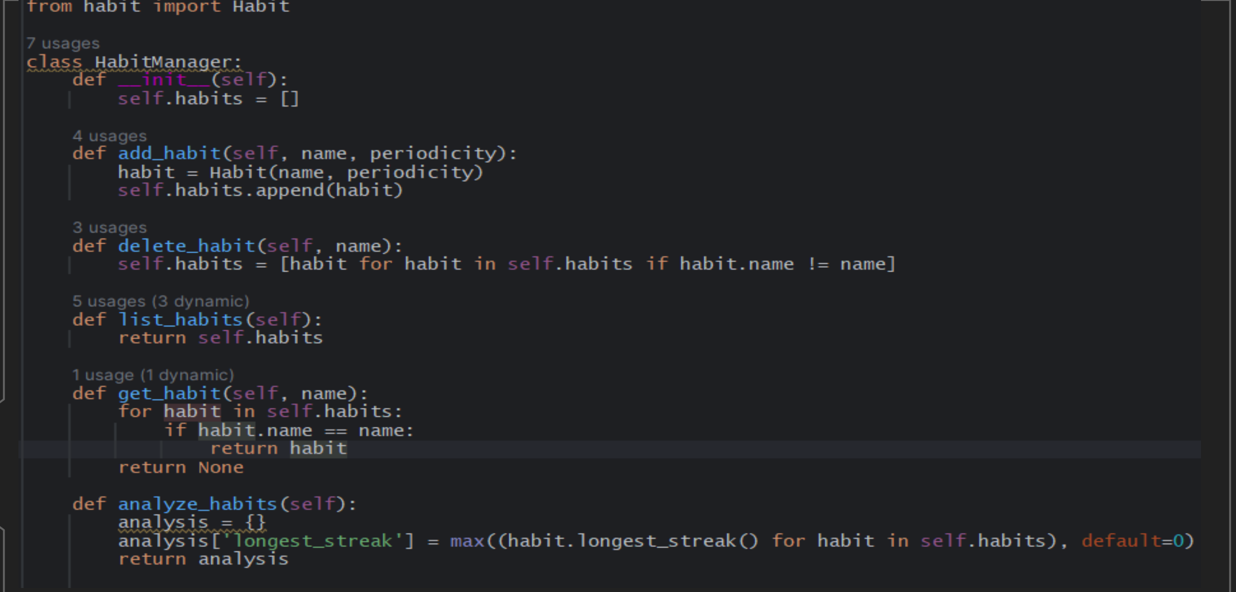
Attributes and Methods

- ‘name’, ‘periodicity’, ‘creation_date’, ‘completion_dates’
- Methods: ‘check_off’, ‘is_broken’, ‘streak’, ‘longest_streak’

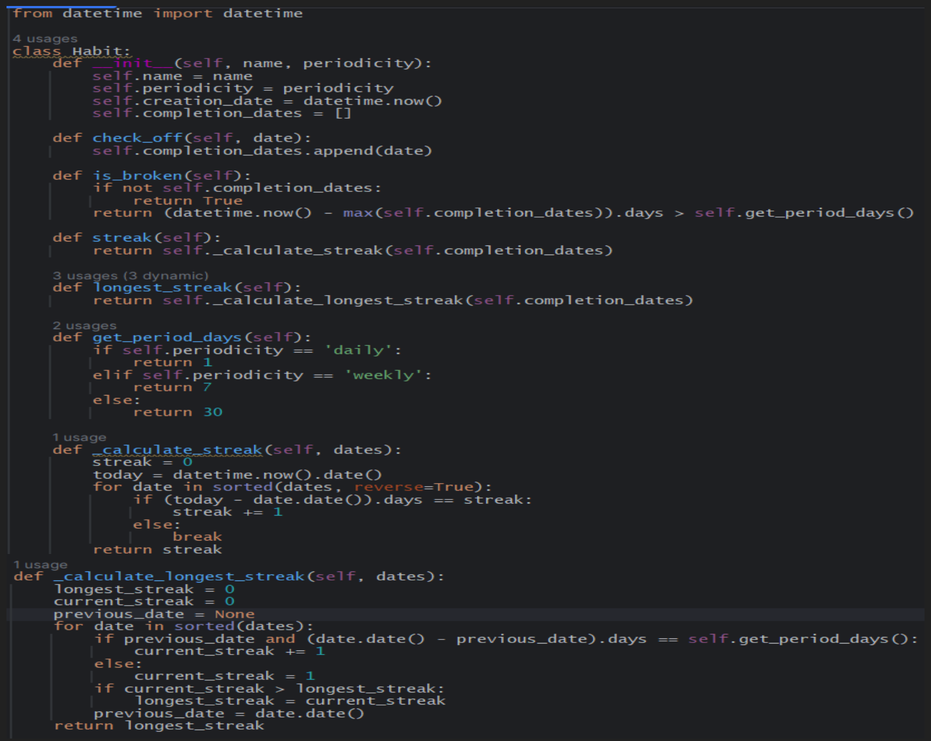
HabitManager Class

Methods for Habit Management

- ‘add_habit’, ‘delete_habit’, ‘list_habits’, ‘get_habit’ , ‘analyze_habits’



```
from habit import Habit

7 usages
class HabitManager:
    def __init__(self):
        self.habits = []

    4 usages
    def add_habit(self, name, periodicity):
        habit = Habit(name, periodicity)
        self.habits.append(habit)

    3 usages
    def delete_habit(self, name):
        self.habits = [habit for habit in self.habits if habit.name != name]

    5 usages (3 dynamic)
    def list_habits(self):
        return self.habits

    1 usage (1 dynamic)
    def get_habit(self, name):
        for habit in self.habits:
            if habit.name == name:
                return habit
        return None

    def analyze_habits(self):
        analysis = {}
        analysis['longest_streak'] = max((habit.longest_streak() for habit in self.habits), default=0)
        return analysis
```



```
from datetime import datetime

4 usages
class Habit:
    def __init__(self, name, periodicity):
        self.name = name
        self.periodicity = periodicity
        self.creation_date = datetime.now()
        self.completion_dates = []

    def check_off(self, date):
        self.completion_dates.append(date)

    def is_broken(self):
        if not self.completion_dates:
            return True
        return (datetime.now() - max(self.completion_dates)).days > self.get_period_days()

    def streak(self):
        return self._calculate_streak(self.completion_dates)

    3 usages (3 dynamic)
    def longest_streak(self):
        return self._calculate_longest_streak(self.completion_dates)

    2 usages
    def get_period_days(self):
        if self.periodicity == 'daily':
            return 1
        elif self.periodicity == 'weekly':
            return 7
        else:
            return 30

    1 usage
    def _calculate_streak(self, dates):
        streak = 0
        today = datetime.now().date()
        for date in sorted(dates, reverse=True):
            if (today - date.date()).days == streak:
                streak += 1
            else:
                break
        return streak
1 usage
def _calculate_longest_streak(self, dates):
    longest_streak = 0
    current_streak = 0
    previous_date = None
    for date in sorted(dates):
        if previous_date and (date.date() - previous_date).days == self.get_period_days():
            current_streak += 1
        else:
            current_streak = 1
        if current_streak > longest_streak:
            longest_streak = current_streak
        previous_date = date.date()
    return longest_streak
```

Analytics Module

Functionality Programming Approach

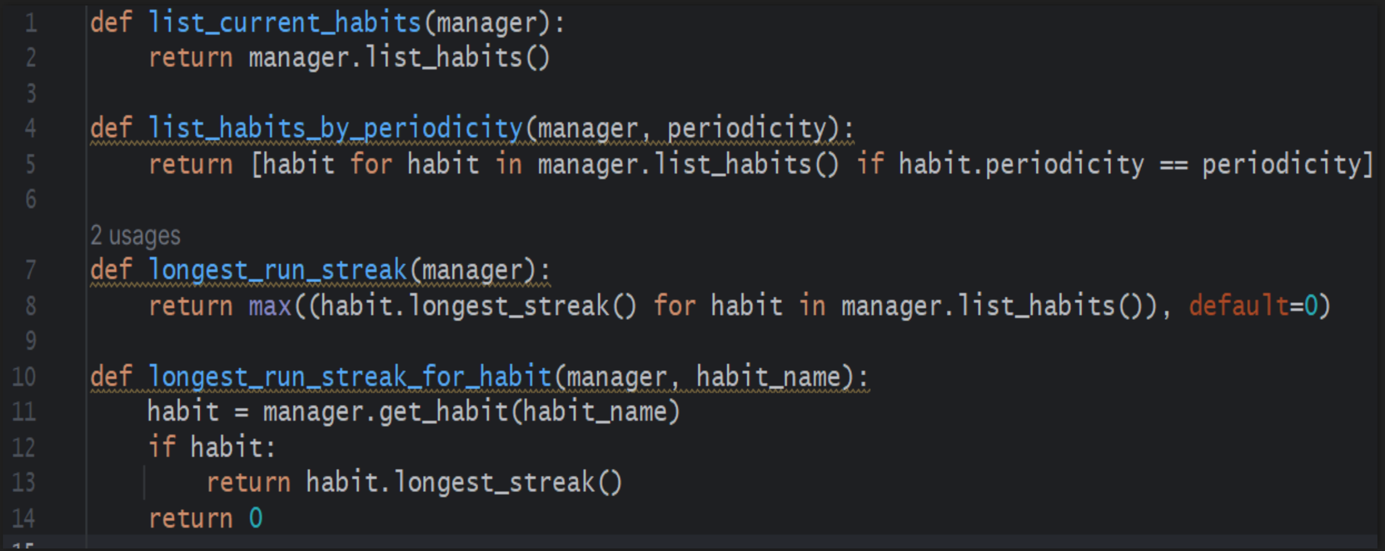
Functions for Analysis

- ‘list_current_habits’, ‘list_habits_by_periodicity’.
- ‘longest_run_streak’, ‘longest_run_streak_for _habit’

‘analytics.py’

Functions overview

- Utilize HabitManager to analyze habits
- Return relevant insights based on user queries



```
def list_current_habits(manager):
    return manager.list_habits()

def list_habits_by_periodicity(manager, periodicity):
    return [habit for habit in manager.list_habits() if habit.periodicity == periodicity]

2 usages
def longest_run_streak(manager):
    return max((habit.longest_streak() for habit in manager.list_habits()), default=0)

def longest_run_streak_for_habit(manager, habit_name):
    habit = manager.get_habit(habit_name)
    if habit:
        return habit.longest_streak()
    return 0
```

Data Storage

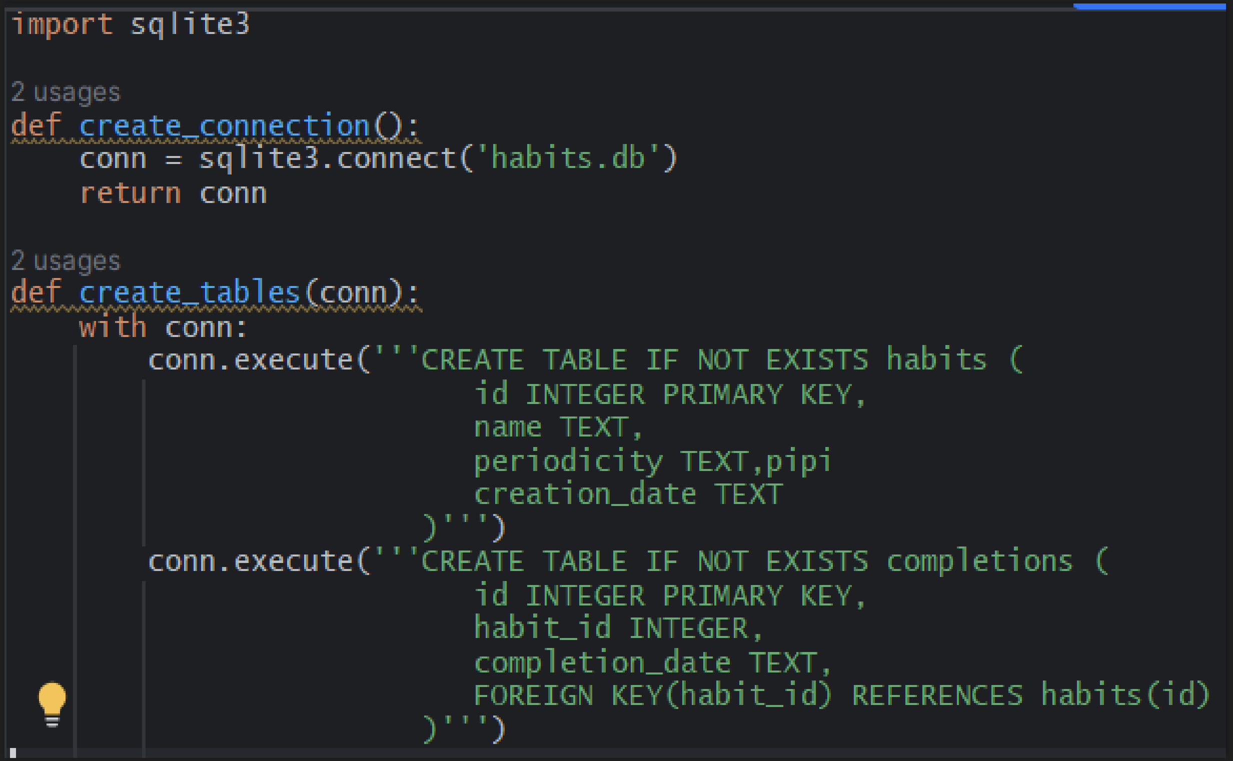
SQLite Database Structure

Tables: Habits, Completions

- Habits Table: Stores habit details (name, periodicity, creation date) with unique IDs.
- Completions Table: Tracks completion dates linked to habits via foreign keys.

Importance for Reliability and Scalability

- Ensures Data Integrity by maintaining data consistency.
- Supports Growth since scalable architecture accommodates increasing habit tracking data.



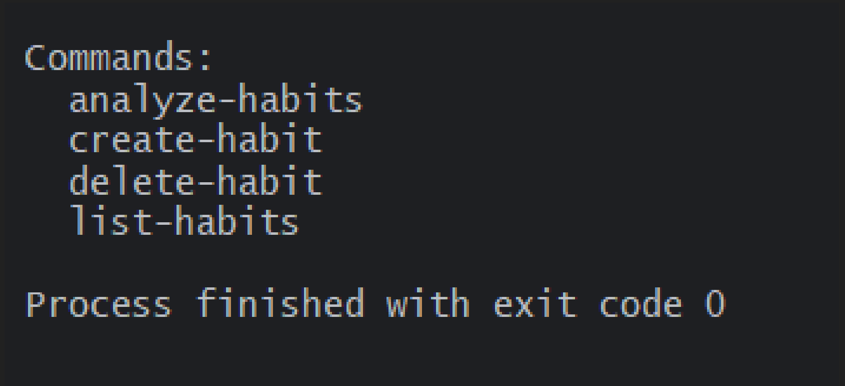
```
import sqlite3

2 usages
def create_connection():
    conn = sqlite3.connect('habits.db')
    return conn

2 usages
def create_tables(conn):
    with conn:
        conn.execute('''CREATE TABLE IF NOT EXISTS habits (
                            id INTEGER PRIMARY KEY,
                            name TEXT,
                            periodicity TEXT,pipi
                            creation_date TEXT
                        )''')
        conn.execute('''CREATE TABLE IF NOT EXISTS completions (
                            id INTEGER PRIMARY KEY,
                            habit_id INTEGER,
                            completion_date TEXT,
                            FOREIGN KEY(habit_id) REFERENCES habits(id)
                        )''')
```

User Interaction

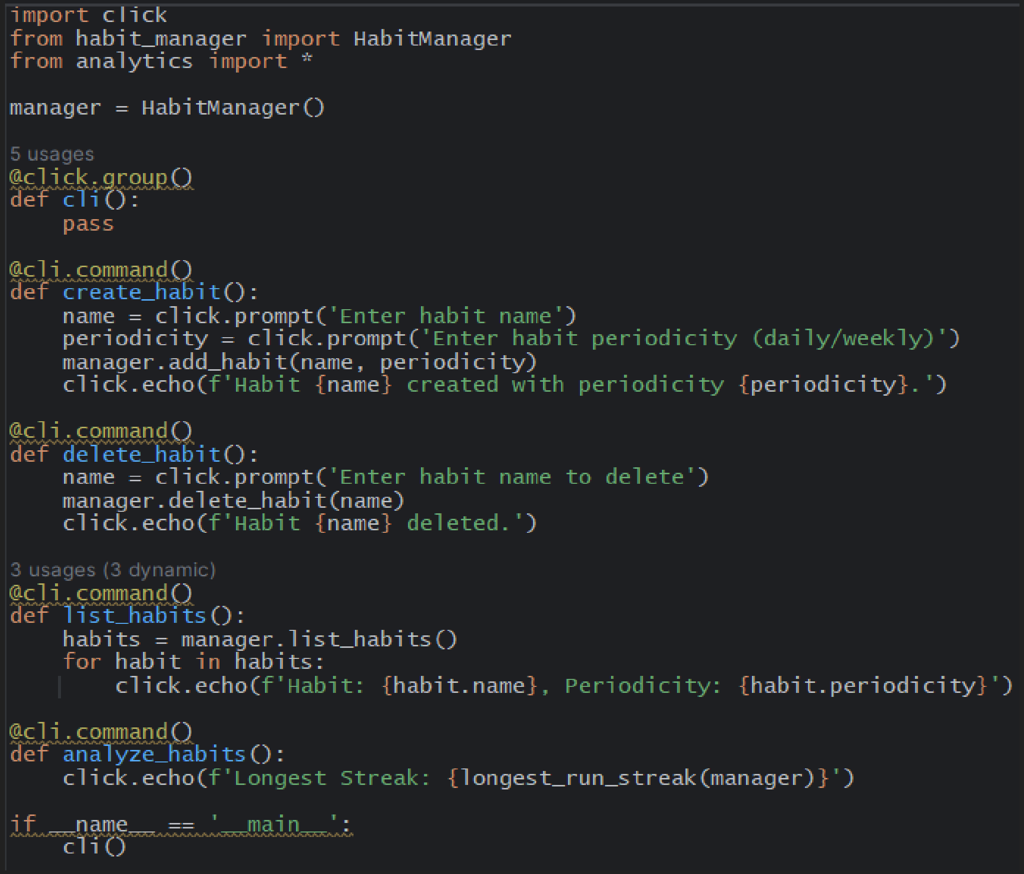
Command-Line Interface (CLI) Overview

Commands: create_habit, delete_habit, list_habits

- **create_habit:** Adds a new habit with user-specified name and periodicity (daily/weekly).
- **delete_habit:** removes an existing habit based on the habit name provided.
- **list_habits:** displays a list of all currently tracked habits with their names and periodicities.



```
Commands:
  analyze-habits
  create-habit
  delete-habit
  list-habits

Process finished with exit code 0
```



```
import click
from habit_manager import HabitManager
from analytics import *

manager = HabitManager()

5 usages
@click.group()
def cli():
    pass

@cli.command()
def create_habit():
    name = click.prompt('Enter habit name')
    periodicity = click.prompt('Enter habit periodicity (daily/weekly)')
    manager.add_habit(name, periodicity)
    click.echo(f'Habit {name} created with periodicity {periodicity}.')

@cli.command()
def delete_habit():
    name = click.prompt('Enter habit name to delete')
    manager.delete_habit(name)
    click.echo(f'Habit {name} deleted.')

3 usages (3 dynamic)
@cli.command()
def list_habits():
    habits = manager.list_habits()
    for habit in habits:
        click.echo(f'Habit: {habit.name}, Periodicity: {habit.periodicity}')

@cli.command()
def analyze_habits():
    click.echo(f'Longest Streak: {longest_run_streak(manager)}')

if __name__ == '__main__':
    cli()
```

User Flow

<u>**Steps to use the App**</u>

1. **Initialize:**

- Install and run the habit tracking app.

2. **Create Habits:**

- Use ‘create_habit’ command to define new habits. (e.g. ‘Drink Water’, ‘Exercise’)

3. **Complete Tasks:**

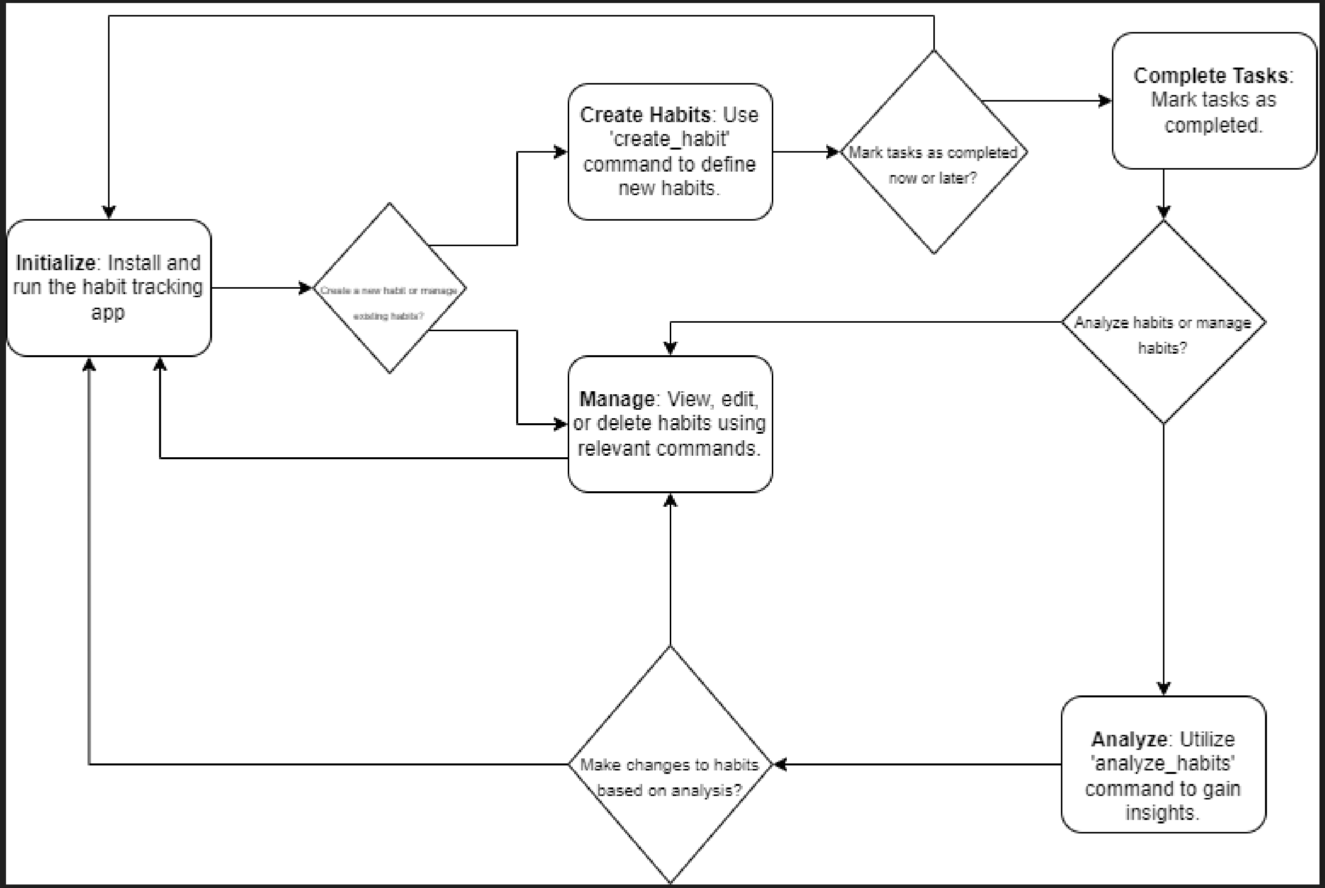
- Mark tasks as completed using the app’s functionality.

4. **Analyze:**

- Utilize ‘analyze_habits’ command to gain insights into habit performance (e.g. longest streak).

5. **Manage:**

- View, edit, or delete habits as needed with ‘list_habits’ and ‘delete_habit’ commands.


Initialize: Install and run the habit tracking app
Create a new habit or manage existing habits?
Create Habits: Use 'create_habit' command to define new habits.
Mark tasks as completed now or later?
Complete Tasks: Mark tasks as completed.
Analyze habits or manage habits?
Manage: View, edit, or delete habits using relevant commands.
Analyze: Utilize 'analyze_habits' command to gain insights.
Make changes to habits based on analysis?

Example Data

Predefined Habits

Examples

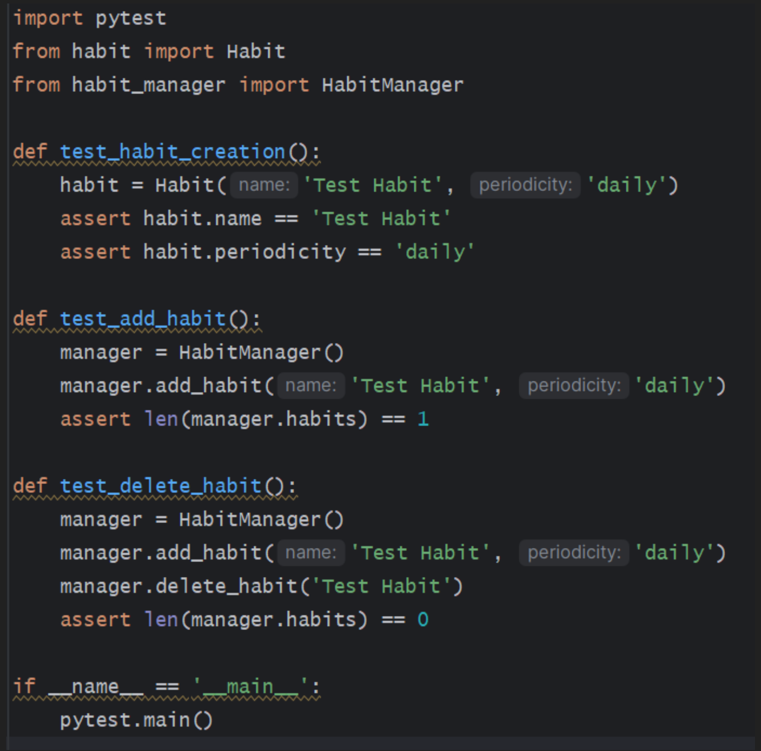
- ‘Drink Water’
- ‘Exercise’

Early Implementation and Testing

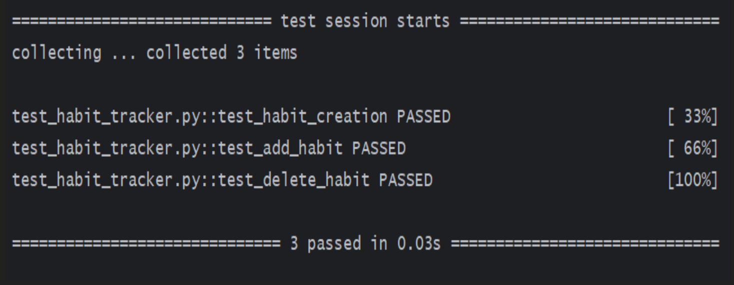
Influence on App Development:

- Predefined habits used for initial testing and functionality validation.
- Helped refine user experience and optimize app performance based on real-use scenarios.

Test_habit_tracker.py file

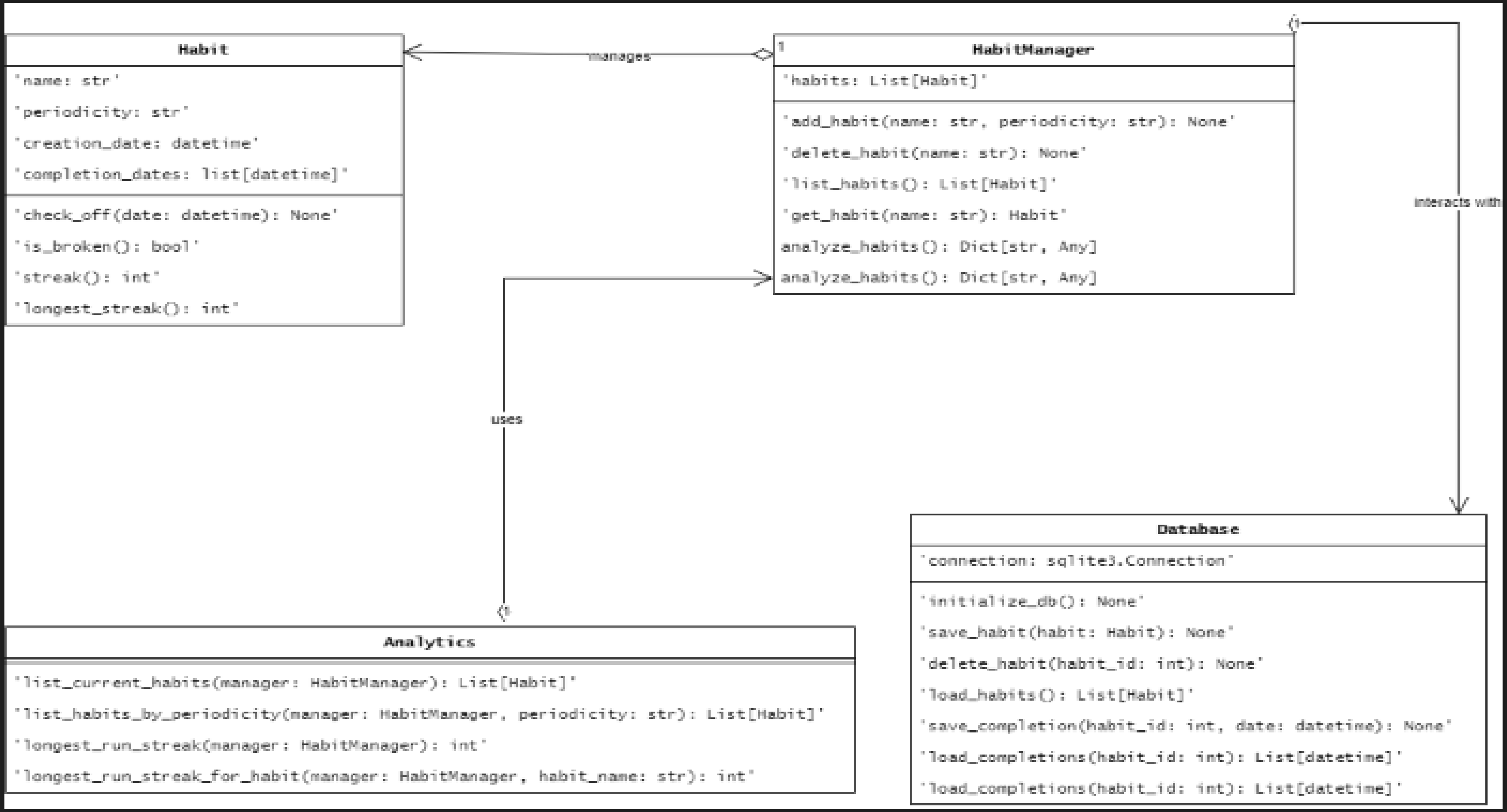


```
import pytest
from habit import Habit
from habit_manager import HabitManager

def test_habit_creation():
    habit = Habit( name: 'Test Habit', periodicity: 'daily')
    assert habit.name == 'Test Habit'
    assert habit.periodicity == 'daily'

def test_add_habit():
    manager = HabitManager()
    manager.add_habit( name: 'Test Habit', periodicity: 'daily')
    assert len(manager.habits) == 1

def test_delete_habit():
    manager = HabitManager()
    manager.add_habit( name: 'Test Habit', periodicity: 'daily')
    manager.delete_habit('Test Habit')
    assert len(manager.habits) == 0

if __name__ == '__main__':
    pytest.main()
```



```
============================ test session starts ============================
collecting ... collected 3 items

test_habit_tracker.py::test_habit_creation PASSED                      [ 33%]
test_habit_tracker.py::test_add_habit PASSED                           [ 66%]
test_habit_tracker.py::test_delete_habit PASSED                        [100%]

============================ 3 passed in 0.03s ============================
```

UML Diagram


Habit
'name: str'
'periodicity: str'
'creation_date: datetime'
'completion_dates: list[datetime]'
'check_off(date: datetime): None'
'is_broken(): bool'
'streak(): int'
'longest_streak(): int'
manages
1
HabitManager
'habits: List[Habit]'
'add_habit(name: str, periodicity: str): None'
'delete_habit(name: str): None'
'list_habits(): List[Habit]'
'get_habit(name: str): Habit'
analyze_habits(): Dict[str, Any]
analyze_habits(): Dict[str, Any]
1
interacts with
uses
1
Analytics
'list_current_habits(manager: HabitManager): List[Habit]'
'list_habits_by_periodicity(manager: HabitManager, periodicity: str): List[Habit]'
'longest_run_streak(manager: HabitManager): int'
'longest_run_streak_for_habit(manager: HabitManager, habit_name: str): int'
Database
'connection: sqlite3.Connection'
'initialize_db(): None'
'save_habit(habit: Habit): None'
'delete_habit(habit_id: int): None'
'load_habits(): List[Habit]'
'save_completion(habit_id: int, date: datetime): None'
'load_completions(habit_id: int): List[datetime]'
'load_completions(habit_id: int): List[datetime]'

Conclusion

The Habit Tracking App enhances productivity and consistency by allowing users to easily create, manage, and analyze their habits. With a user-friendly command-line interface, reliable SQLite data storage, and detailed analytics, users can effectively track their progress and stay motivated. Start using the app today to build positive habits and achieve your goals.