Habit Tracker App

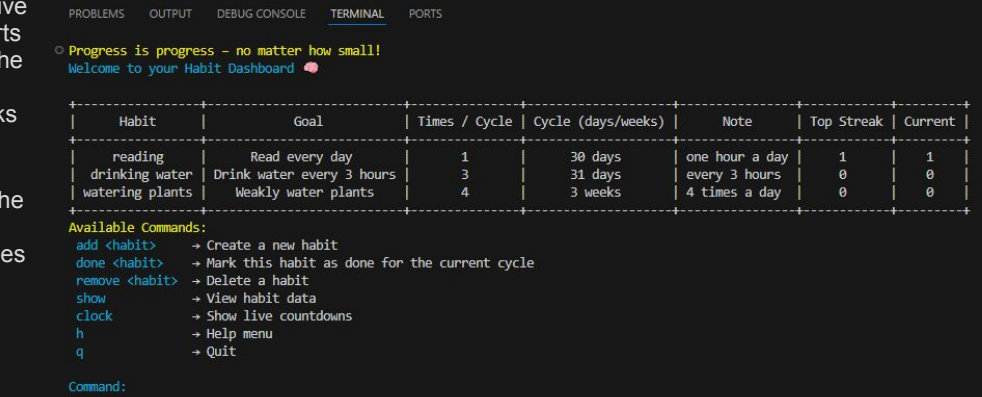
Timafei Bielawski – 102203482

Project Overview

This project is a console-based habit tracker application developed in Python. Its primary goal is to assist users in establishing, developing, and maintaining positive daily or weekly routines by providing a simple and effective tool. The app supports creating, managing, and analyzing multiple habits simultaneously, giving users the flexibility to specify how often each habit should be completed within a specified period, whether daily or weekly. Additionally, it enables users to track their streaks over time to monitor their progress and consistency.

During the development phase, particular focus was placed on implementing the core backend logic using an object-oriented approach, which helps organize the code efficiently and makes it easier to extend. Functional programming techniques were also employed for analytical features to process and evaluate user data effectively. A JSON-based persistence layer was integrated to store habit data persistently across sessions.

All interactions with the application are conducted through a user-friendly command-line interface (CLI). This interface guides users step-by-step through various operations, making it accessible even for those with minimal technical experience.



Key Functionalities

The following core features were successfully implemented:

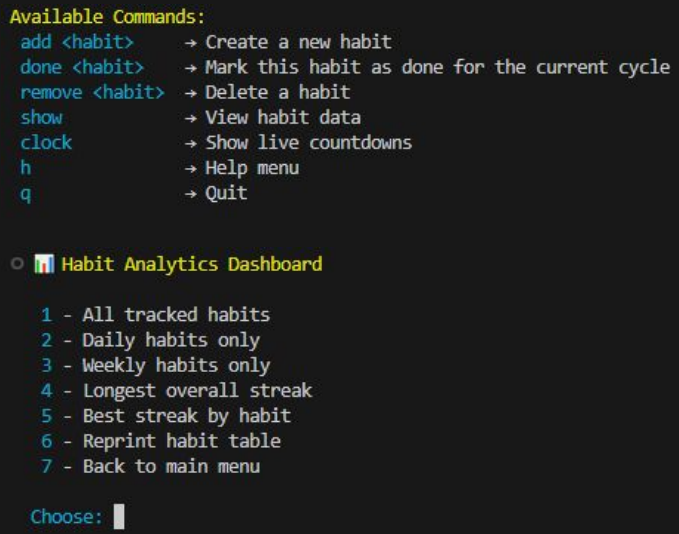
Habit Creation:
Users input the name, goal, cycle type (daily/weekly), time frame, and target frequency. This flexibility supports both lightweight and complex routines.

Marking as Done:
Each completion updates a live counter and streak calculation. If a cycle is missed, the counter resets.

Habit Deletion & Recovery:
Removing a habit deletes its entry. If no habits are left, a placeholder habit auto-restores to preserve structure.

Real-Time Countdown:
A unique feature – users can view a live countdown for each habit's remaining cycle duration. This motivates timely completion.

Habit Table View:
Dynamic, auto-scaling tables display all habit data in clean visual rows adjusted to the user's screen size.



Architecture & File Structure

The project is organized into three main components:

- Main.py: The entry point. It drives the user interface, parses commands, and interacts with the tracker.
- HabitsTrackerApp.py: Contains the Tracker class with all business logic: adding habits, checking them off, tracking progress, and analytics.
- Data.json: Stores all habit information persistently using a structured format. This includes frequency, goals, timestamps, and streaks.

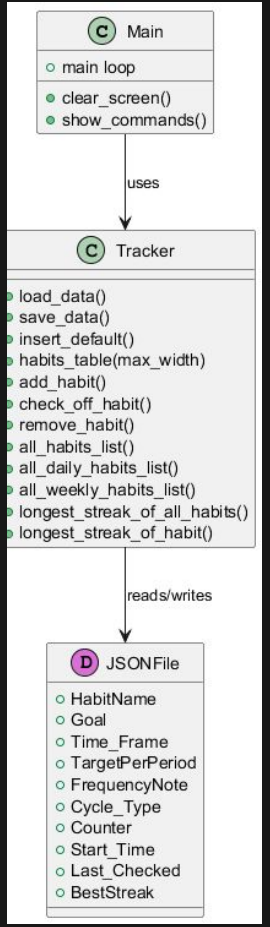
This modular architecture separates concerns cleanly:

- Logic is encapsulated.
- Data is decoupled.
- UI is responsive to terminal size.

The design follows key OOP principles:

- **Encapsulation**: All data and logic reside inside the Tracker class.
- **Abstraction**: The user doesn't see internal mechanics – they interact through intuitive commands.
- **Modularity**: Every action is its own method, allowing easy testing and future reuse.
- **Scalability**: Adding new features (like categories, reminders, etc.) is straightforward.

The class is self-contained and responsible for managing the full life cycle of each habit from creation to analysis.



Functional Programming in Analytics

Analytics operations are handled using **functional paradigms**: mapping, filtering, and aggregation, ensuring stateless and efficient queries:

- **List All Habits**: A full overview of tracked habits.
- **List by Cycle Type**: Filter habits by days or weeks.
- **Longest Overall Streak**: Returns the habit with the best streak using max() and lambda functions.
- **Best Streak Per Habit**: Looks up the peak streak for a specific habit name.

These functions are pure – they don't mutate global state – aligning with functional programming goals.

```python
# Display all habits in a dynamic-width table
def habits_table(self, max_width=None):
    if not self.data:
        print("You haven't added any habits yet.")
        return

    table = PrettyTable()
    table.field_names = [
        "Habit", "Goal", "Times / Cycle", "Cycle (days/weeks)",
        "Note", "Top Streak", "Current"
    ]

    for habit, info in self.data.items():
        duration = info.get("Time_Frame", 0)
        unit = info.get("Cycle_Type", "days")
        table.add_row([
            habit,
            info.get("Goal", ""),
            info.get("TargetPerPeriod", 0),
            f"{duration} {unit}",
            info.get("FrequencyNote", ""),
            info.get("BestStreak", 0),
            info.get("Counter", 0)
        ])
```

```python
# Print a list of all habit names
def all_habits_list(self, show=False):
    if show:
        print("📋 Current Habits:")
        for habit in self.data:
            print(f" - {habit}")

# Print daily habits only; return True if found
def all_daily_habits_list(self, show=False):
    found = False
    if show:
        print("🗓️ Daily Habits:")
        for name, info in self.data.items():
            if info.get("Cycle_Type") == "days":
                print(f" - {name}")
                found = True
    return found
```

```python
# Print weekly habits only; return True if found
def all_weekly_habits_list(self, show=False):
    found = False
    if show:
        print("🗓️ Weekly Habits:")
        for name, info in self.data.items():
            if info.get("Cycle_Type") == "weeks":
                print(f" - {name}")
                found = True
    return found

# Find and display the longest streak among all habits
def longest_streak_of_all_habits(self, show=False):
    if not self.data:
        print("No habits to evaluate.")
        return

    best = max(self.data.items(), key=lambda item: item[1].get("BestStreak", 0))
    if show:
        print(f"🏆 Longest overall streak is '{best[0]}' with {best[1]['BestStreak']}.")
```

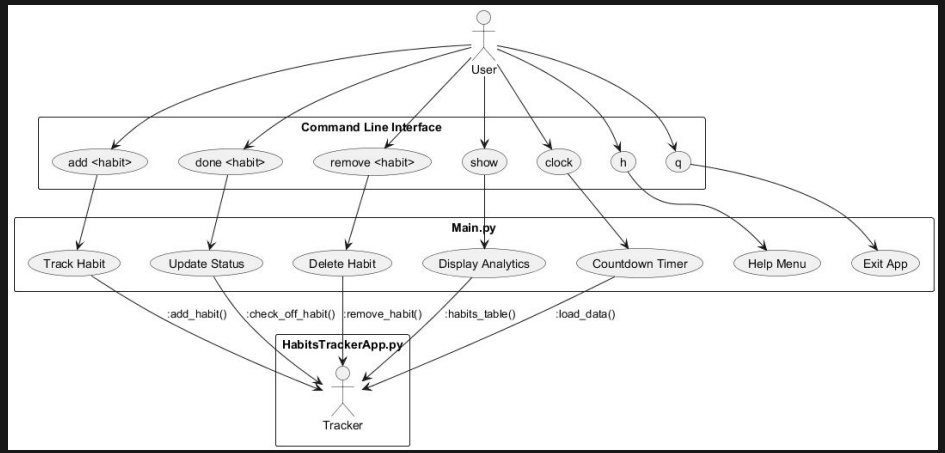
```python
# Display best streak for a specific habit
def longest_streak_of_habit(self, name, show=False):
    if name not in self.data:
        print(f"Habit '{name}' does not exist.")
        return
    if show:
        print(f"🔥 Best streak for '{name}': {self.data[name]['BestStreak']}")
```

Command-Line User Interface

The CLI was designed with accessibility and friendliness in mind. It includes:

- **Command List View** (help prompt): add, done, remove, show, clock, h, q
- **Interactive Prompts**: Users are guided through habit creation with clear questions and validation.
- **Tabular Output**: Uses prettytable to neatly format habit data. Columns adjust based on screen width via shutil.
- **Live Clock Mode**: The clock command visually shows a per-habit countdown timer to motivate users.

This ensures a smooth user experience with immediate feedback.




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Progress is progress – no matter how small!
Welcome to your Habit Dashboard 🧠

+----------------+-------------------------+-------------+-------------------+----------------+------------+---------+
|     Habit      |          Goal           | Times / Cycle | Cycle (days/weeks) |      Note      | Top Streak | Current |
+----------------+-------------------------+-------------+-------------------+----------------+------------+---------+
|    reading     |      Read every day     |      1      |      30 days      | one hour a day |     1      |    1    |
| drinking water | Drink water every 3 hours |      3      |      31 days      | every 3 hours  |     0      |    0    |
| watering plants |    Weakly water plants  |      4      |      3 weeks      | 4 times a day  |     0      |    0    |
+----------------+-------------------------+-------------+-------------------+----------------+------------+---------+
Available Commands:
 add <habit>      → Create a new habit
 done <habit>     → Mark this habit as done for the current cycle
 remove <habit>   → Delete a habit
 show             → View habit data
 clock            → Show live countdowns
 h                → Help menu
 q                → Quit

Command:
```

Data Storage and Structure

The app uses a lightweight **file-based storage** model via Data.json, avoiding the overhead of a full SQL DB but preserving persistent data.

Each habit entry includes:

- **Goal:** Description
- **Time_Frame:** Duration of a habit cycle (in days or weeks)
- **TargetPerPeriod:** Required completions per cycle
- **FrequencyNote:** Optional user note (e.g., “before bed”)
- **Cycle_Type:** "days" or "weeks"
- **Start_Time:** Timestamp when cycle started
- **Last_Checked:** Last time habit was marked as done
- **Counter:** How many times marked during the cycle
- **BestStreak:** Longest successful streak

This format ensures fast lookups and human-readable backups.

Enhancements Beyond Requirements

The project includes several **creative and unique enhancements** beyond the basic criteria:

- **Live countdown timers** with cycle-based clock logic (using time, datetime, timedelta)
- **Cycle type selector (days/weeks)** during habit creation, making the tool adaptable to different goals
- **Placeholder habit logic**: If all habits are removed, a default dummy habit is restored
- **Auto-formatted, screen-responsive table views** based on terminal size

These features enhance usability and demonstrate a deeper thought into how real users might use the app daily.

Code Showcase

HabitsTrackerApp.py

```python
import os
import time
import json
import shutil
from prettytable import PrettyTable

class Tracker:
    def __init__(self):
        self.filename = "Data.json"
        self.load_data()

    # Load JSON data or initialize default
    def load_data(self):
        if not os.path.exists(self.filename):
            with open(self.filename, "w") as f:
                json.dump({}, f)
        with open(self.filename, "r") as f:
            self.data = json.load(f)
        if not self.data:
            self.insert_default()

    # Save current habit data to JSON
    def save_data(self):
        with open(self.filename, "w") as f:
            json.dump(self.data, f, indent=4)

    # Insert a placeholder habit when starting fresh
    def insert_default(self):
        self.data["DefaultHabit"] = {
            "Goal": "",
            "Time_Frame": 1,
            "TargetPerPeriod": 1,
            "FrequencyNote": "",
            "Cycle_Type": "days",
            "Counter": 0,
            "Start_Time": time.time(),
            "Last_Checked": None,
            "BestStreak": 0
        }
        self.save_data()

    # Display all habits in a dynamic-width table
    def habits_table(self, max_width=None):
        if not self.data:
            print("You haven't added any habits yet.")
            return

        table = PrettyTable()
        table.field_names = [
            "Habit", "Goal", "Times / Cycle", "Cycle (days/weeks)",
            "Note", "Top Streak", "Current"
        ]

        for habit, info in self.data.items():
            duration = info.get("Time_Frame", 0)
            unit = info.get("Cycle_Type", "days")
            table.add_row([
                habit,
                info.get("Goal", ""),
                info.get("TargetPerPeriod", 0),
                f"{duration} {unit}",
                info.get("FrequencyNote", ""),
                info.get("BestStreak", 0),
                info.get("Counter", 0)
            ])

        # Fit to screen width if needed
        if max_width:
            table.max_width = max_width
        else:
            columns, _ = shutil.get_terminal_size()
            table.max_width = columns

        print(table)

    # Add a new habit (or replace placeholder)
    def add_habit(self, name, goal, time_frame, target, note="", cycle_type="days"):
        if len(self.data) == 1 and "default" in list(self.data.keys())[0].lower():
            self.data.clear()

        if name in self.data:
            print(f"The habit '{name}' already exists.")
            return

        now = time.time()
        self.data[name] = {
            "Goal": goal,
            "Time_Frame": time_frame,
            "TargetPerPeriod": target,
            "FrequencyNote": note,
            "Cycle_Type": "weeks" if cycle_type == "w" else "days",
            "Counter": 0,
            "Start_Time": now,
            "Last_Checked": None,
            "BestStreak": 0
        }
        self.save_data()
        print(f"New habit '{name}' added!")

    # Mark a habit as completed
    def check_off_habit(self, name):
        if name not in self.data:
            print(f"Habit '{name}' doesn't exist.")
            return

        now = time.time()
        habit = self.data[name]
        length = habit.get("Time_Frame", 1)
        unit = habit.get("Cycle_Type", "days")
        cycle_seconds = length * (7 if unit == "weeks" else 1) * 86400
        last_checked = habit.get("Last_Checked")

        if last_checked and now - last_checked < cycle_seconds:
            habit["Counter"] += 1
        else:
            habit["Counter"] = 1
            habit["Start_Time"] = now

        habit["Last_Checked"] = now

        if habit["Counter"] > habit["BestStreak"]:
            habit["BestStreak"] = habit["Counter"]

        self.save_data()
        print(f"✅ You've logged progress for '{name}'.")

    # Remove a habit from the tracker
    def remove_habit(self, name):
        if name not in self.data:
            print(f"Habit '{name}' not found.")
            return

        del self.data[name]
        print(f"'{name}' has been removed.")

        if not self.data:
            self.insert_default()
            print("All habits removed. Default placeholder added.")

        self.save_data()

    # Print a list of all habit names
    def all_habits_list(self, show=False):
        if show:
            print("📋 Current Habits:")
            for habit in self.data:
                print(f" - {habit}")

    # Print daily habits only; return True if found
    def all_daily_habits_list(self, show=False):
        found = False
        if show:
            print("📅 Daily Habits:")
            for name, info in self.data.items():
                if info.get("Cycle_Type") == "days":
                    print(f" - {name}")
                    found = True
        return found

    # Print weekly habits only; return True if found
    def all_weekly_habits_list(self, show=False):
        found = False
        if show:
            print("🗓️ Weekly Habits:")
            for name, info in self.data.items():
                if info.get("Cycle_Type") == "weeks":
                    print(f" - {name}")
                    found = True
        return found

    # Find and display the longest streak among all habits
    def longest_streak_of_all_habits(self, show=False):
        if not self.data:
            print("No habits to evaluate.")
            return

        best = max(self.data.items(), key=lambda item: item[1].get("BestStreak", 0))
        if show:
            print(f"🏆 Longest overall streak is '{best[0]}' with {best[1]['BestStreak']}.")

    # Display best streak for a specific habit
    def longest_streak_of_habit(self, name, show=False):
        if name not in self.data:
            print(f"Habit '{name}' does not exist.")
            return
        if show:
            print(f"🔥 Best streak for '{name}': {self.data[name]['BestStreak']}")
```

Main.py

```python
import os
import subprocess
import sys
import time
from datetime import timedelta
import shutil  # To fetch terminal width dynamically

# ------------------------------------------------
# Function to auto-install required packages
# ------------------------------------------------
def ensure_package(pkg):
    try:
        __import__(pkg)
    except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", pkg])

# Ensure necessary libraries are installed
ensure_package("colorama")
ensure_package("prettytable")

import colorama
from colorama import Fore, Style
from HabitsTrackerApp import Tracker

# ------------------------------------------------
# Color Setup
# ------------------------------------------------
colorama.init()
MainColor, Highlight, ResetColor = Fore.CYAN, Fore.YELLOW, Style.RESET_ALL

# ------------------------------------------------
# Clears the terminal screen
# ------------------------------------------------
def clear_screen():
    os.system('cls' if os.name == 'nt' else 'clear')

# ------------------------------------------------
# Displays user instructions and available commands
# ------------------------------------------------
def show_commands():
    print(f"{Highlight}Available Commands:{ResetColor}")
    print(f"{MainColor} add <habit>     {ResetColor}→ Create a new habit")
    print(f"{MainColor} done <habit>    {ResetColor}→ Mark this habit as done for the current cycle")
    print(f"{MainColor} remove <habit>  {ResetColor}→ Delete a habit")
    print(f"{MainColor} show            {ResetColor}→ View habit data")
    print(f"{MainColor} clock           {ResetColor}→ Show live countdowns")
    print(f"{MainColor} h               {ResetColor}→ Help menu")
    print(f"{MainColor} q               {ResetColor}→ Quit")

# ------------------------------------------------
# Initialize the habit tracker instance
# ------------------------------------------------
habits = Tracker()

# ------------------------------------------------
# Main application loop
# ------------------------------------------------
while True:
    clear_screen()
    print(f"{Highlight}Progress is progress 🔄 no matter how small!{ResetColor}")
    print(f"{MainColor}Welcome to your Habit Dashboard 🧠{ResetColor}\n")

    # Auto-resize the table to match terminal width
    columns, _ = shutil.get_terminal_size()
    habits.habits_table(max_width=columns)

    show_commands()
    user_input = input(f"\n{MainColor}Command: {ResetColor}").strip().lower()
    if not user_input:
        continue

    parts = user_input.split()
    command = parts[0]
    param = " ".join(parts[1:]) if len(parts) > 1 else ""

    # ---------------- ADD HABIT ----------------
    if command == "add":
        if not param:
            continue
        goal = input(f"{MainColor}Goal: {ResetColor}")
        while True:
            cycle_type = input(f"{MainColor}Cycle unit - (d)ays or (w)eeks? {ResetColor}").lower()
            if cycle_type in ["d", "w"]: break
            print("Please enter 'd' or 'w'")
        while True:
            length = input(f"{MainColor}Cycle length in {'days' if cycle_type == 'd' else 'weeks'}: {ResetColor}")
            if length.isdigit():
                length = int(length)
                break
            print("Enter a valid number.")
        while True:
            freq = input(f"{MainColor}How many times per cycle (number): {ResetColor}")
            if freq.isdigit():
                freq = int(freq)
                break
            print("Enter a valid number.")
        note = input(f"{MainColor}Optional note (e.g. 'before bed'): {ResetColor}")
        habits.add_habit(param, goal, length, freq, note, cycle_type)

    # ---------------- MARK AS DONE ----------------
    elif command == "done":
        if param:
            habits.all_habits_list(True)
            habits.check_off_habit(param)

    # ---------------- REMOVE HABIT ----------------
    elif command == "remove":
        if param:
            habits.all_habits_list(True)
            habits.remove_habit(param)

    # ---------------- SHOW ANALYTICS ----------------
    elif command == "show":
        while True:
            clear_screen()
            print(f"{Highlight}📊 Habit Analytics Dashboard{ResetColor}\n")
            print(f"{MainColor} 1 {ResetColor}- All tracked habits")
            print(f"{MainColor} 2 {ResetColor}- Daily habits only")
            print(f"{MainColor} 3 {ResetColor}- Weekly habits only")
            print(f"{MainColor} 4 {ResetColor}- Longest overall streak")
            print(f"{MainColor} 5 {ResetColor}- Best streak by habit")
            print(f"{MainColor} 6 {ResetColor}- Reprint habit table")
            print(f"{MainColor} 7 {ResetColor}- Back to main menu")

            opt = input(f"\n{MainColor}Choose: {ResetColor}").strip()

            if opt == "1":
                habits.all_habits_list(True)
            elif opt == "2":
                if not habits.all_daily_habits_list(True):
                    print("⚠️ No daily habits found.")
            elif opt == "3":
                if not habits.all_weekly_habits_list(True):
                    print("⚠️ No weekly habits found.")
            elif opt == "4":
                habits.longest_streak_of_all_habits(True)
            elif opt == "5":
                habits.all_habits_list(True)
                name = input(f"{MainColor}Habit name: {ResetColor}")
                habits.longest_streak_of_habit(name, True)
            elif opt == "6":
                columns, _ = shutil.get_terminal_size()
                habits.habits_table(max_width=columns)
            elif opt == "7":
                break
            else:
                print(f"{Fore.RED}Invalid input. Try again.{ResetColor}")
                continue

            input(f"\n{Highlight}Press Enter to continue...{ResetColor}")

    # ---------------- CLOCK COUNTDOWN ----------------
    elif command == "clock":
        for _ in range(10):  # Simulate for 10 seconds
            clear_screen()
            print("\n⏳ Live Countdown for Habit Cycles:\n")
            for name, info in habits.data.items():
                start = info.get("Start_time", time.time())
                cycle_type = info.get("Cycle_Type", "days")
                length = info.get("Time_Frame", 1)
                seconds = length * 7 * 86400 if cycle_type == "weeks" else length * 86400
                remaining = max(0, int(start + seconds - time.time()))
                print(f"⌛ {name}: {str(timedelta(seconds=remaining))} left")
            time.sleep(1)

    # ---------------- HELP ----------------
    elif command == "h":
        show_commands()

    # ---------------- QUIT ----------------
    elif command == "q":
        print(f"{MainColor}Thanks for using the Habit Tracker. Stay consistent!{ResetColor}")
        break

    # ---------------- UNKNOWN ----------------
    else:
        print(f"{Fore.RED}Unknown command. Type 'h' for help.{ResetColor}")
```