Phase 3: Abstract

Emre Ertürk, Matriculation: 92113596, 21st March 2022

HTTPS://GITHUB.COM/MR-EMREERTURK/HABIT-TRACKER.GIT

Synapsis

The objective of the habit tracker app is to build a non-GUI-based habit tracker app that allows the user to track habits on a daily and weekly basis. Therefore, the initial idea to visualize the habits through a data frame has been shown to be quite effective as the functionality of this design exceeded expectations. The use of the object-oriented programming (OOP) approach exceeded expectations as well and has been used to its full potential.

The execution

As for the main.py, the concept was simple. Use classes and assign objects in this file to keep the code readable. Doing so, allowed the programmer to use an advanced while loop to catch details and add conveniences for the user. E.g., the user can add multiple habits repetitively without restarting the entire program, or the user gets messages as to why their command has failed to execute. This is due to the many **try** & **except** statements in the code.

The **HabitTracker** class is the backbone of the program. Within the **__init__** call at the start, the programmer included a sample habit tracker data frame which gets displayed if the habit tracker is empty. For the convenience of the user, this sample deletes itself and isn't stored locally so that no interference between the sample and actual habits can occur. Within this class, you will find the methods that add, delete, and manage entries from the user, but also a method that prints out the data frame to the terminal and a method to update the data frame every time the app restarts or the user finishes one of the 3 major tasks - edit, manage, or analyze.

Speaking of analysis, the **Analytics** class is a child-class to the HabitTracker parent-class. This structure allowed the programmer to inherit all attributes from the parent class and compute the supplemental Analytics without creating new attributes.

The wins

The biggest win is the use of the data frame structure. Through the usage of a table, the user can see all relevant information for the habits all at once without it being overwhelming, making the Analytics class supplemental and for additional information only. Streaks, in days and weeks, and the record per habit is clearly shown in the table. The **combination of Pandas, DateTime** and **Tabulate modules was** crucial in this process. Through **DateTime**, the computation of the current streaks was easy. **Pandas** was the backbone of the entire table and helped in supplemental tasks such as sorting habits by streaks. Lastly, the **Tabulate** module allows a unified look of the table, independent of the program used.

Next, as stated in the conception phase, the substantial effect of JSON file format compared to the CSV format, crystallized early in the development of the app. JSON simply allows the use of nested dictionaries where the habit names operate as parent keys and the habit features such as "Descriptions" or "Date started" are nested as a dictionary within the parent dictionary. Combining the power of JSON's nested dictionary with the Pandas modules, the user gets shown a data frame where the habits are the indexes, and the habit features are the columns. Furthermore, adding, editing, and deleting habits is easier in JSON files as well.

Another, rather a small win, is the count separation of daily and weekly habits. The computation of such seemed to be the most difficult part of the concept. How does the app count a streak? By day or successful period. For the latter, the comparison of the "Record" column would seem impossible. A successful weekly habit would score +1 whereas a daily successful habit for a week would score +7. How would I compare these scores in a simple way? I decided to give the user an option to track a habit daily or weekly but no matter how the user tracks, the results are displayed in days and weeks, no matter what. This way the "Record" column is relatable, and the user can interpret the data.
Lastly, maybe the biggest win is the added convenience for the user. As mentioned earlier, I went over different scenarios on how the user may cause errors within the code. E.g., the user may have added habits and created the "habits.json" file already but has accidentally deleted all records. That would leave "{}" in the JSON file. Next time the user adds a habit to the app, it will cause an Error. The try & except statement at the beginning of the main.py file catches exactly that scenario and deletes the JSON file. Small details like this are what make me proud of the code I have written.

The pitfalls

I forgot to stop the loop. Using a while loop is great, and as a developer exiting the program is quite logical. I did it hundreds of times. My mistake was to imply that the user knows that as well. As shown in the conception, my flow chart did not have a feature to stop the app. At the very end, I realized that the user would never know how to stop the app. That's why I have added a stop call to stop the while loop and let the user terminate the app.
Another issue that I have faced was the nested methods within **Habitstracker**'s **add_entry()** and **manage_habits()** method. The computation of the current streaks and the record took the most time. I was overfocusing on using the analytics class to calculate these columns that I forgot about using nested methods. Although, I must admit that this is not the most elegant way to do it but using more time, energy, and resources seems irresponsible. A possible solution for the simplification of this code is either decorators or the use of an additional class. A Senior Developer might find a solution to it.

Conclusion

The app works fluently. In my tests, there are no errors that the user can see. The results match the expectations of the project.

- The user can add, delete, manage habits.
- The user can "*tick-off*" habits. The app automatically restarts the habit from the day it stopped and saves the maximum number of streaks in the "Record" column until that high score is broken.
- The asked Analytics are shown in the main table and come sorted so that the same periodicities are shown together

Resources

https://www.roelpeters.be/pandas-read-json-orient/