Habit Tracker

The Habit Tracker is used to create and manage your habits. You can create new habits with different periods and check off them anytime. Everytime you will manage a habit task you will get a streak. The programme tries to be the backend code for further developments, but is having a simplified GUI to try out the different functions.

Python Version & libraries:

Python 3.7 or higher is necessary.

Libraries:

- sqlite3
- datetime
- tkinter
- pytest

Overview of the programme and python files:

- Installation.py
- main.py
- Test.py
- Analyse_Development.py
- Start_Data.py

Installation

To set up the database for the habit tracker, the python file **Installation.py** has to be run. It will create the Habit.db with two Tables Habits and User.

After the Installation the habit tracker can be used.

If you want to have some dummy values in the two tables you can run the python file **Start_Data.py**. The two dummy-users are Andi (no password) and Magi (no password).

Application Habit Tracker

To use the Habit Tracker the python file **main.py** has to run. Then a GUI will pop up to guide the user through the Habit Tracker. If you need a detailed description, you are welcome to use the Habit Tracker Presentation.

Backend

There are different classes in the programme, feel free to use the functions of the habit tracker.
You can find further descriptions in the code.

Class Habit

Here functions for creating, deleting, checking off, and getting Habits can be found.

Creating Habits:

The function create_habit needs the input “Name”, “Period” and “User”.
Name = Name of the habit

Period = which period the habit should be checked off. It is important to know that the input period must be “Daily”, “Weekly”, or “Monthly”, otherwise the function will return “Wrong Period”.

User = Which User belongs to this habit

After that, the function creates an entry in the table habits and returns “Habit created”.

Class Users
In this section the register and login functions are implemented

Analyse Habits
In the programme are two functions, which are programmed functionally:

- selected_items
- highest_value

They are both used in the Analyse_Habit class to analyse different things from the database.

End Of Day To-Dos

It is important to know, that the EoD class must be computed every day before midnight. (23:59) There, **Open** habit tasks will set to failed and the streak to 0. Further, all active habit tasks in the table Habits in the database habit tasks are copied and set to the new date (tomorrow).

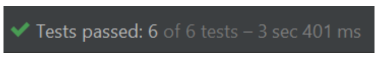
To test or compute the function manually a EoD Button is implemented. This button is only visible if the admin user is logged in.
User: admin
PW: PW1234!

Analysis – Development

If you want to see all registered users and habits an own python file **Analysis_Development.py** exists to allow further data analysis.

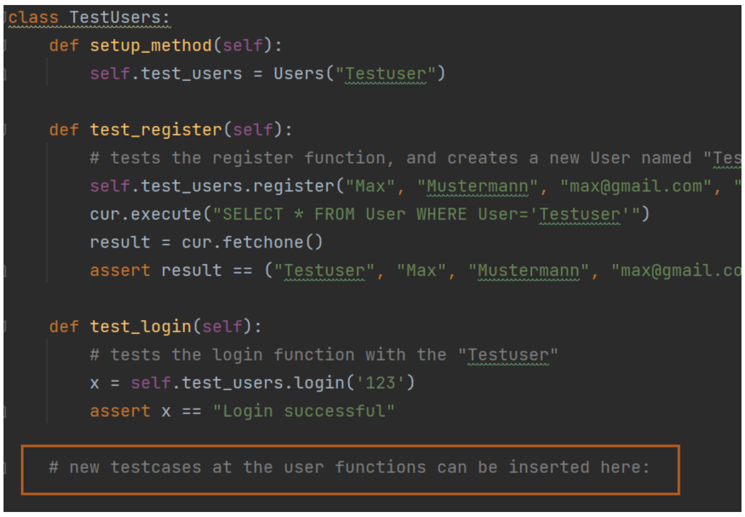
Testing

Description:

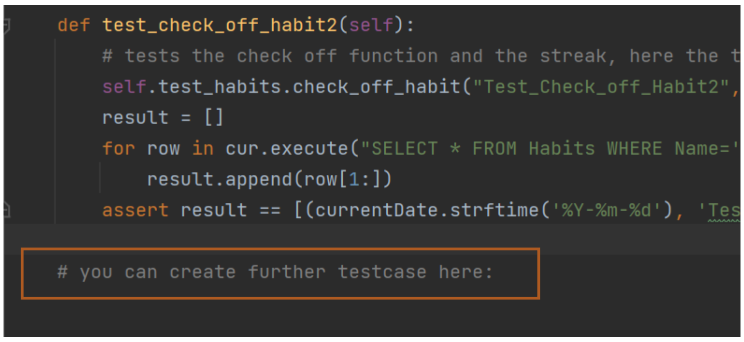
1. Open the Python-Module **Tests.py**
2. Run the python code:
   In case that some GUI pops up, you can ignore it and close the windows
3. If all things are fine, it will return “Tests passed”: 
   Tests passed: 6 of 6 tests – 3 sec 401 ms
4. In case of an error the programme will return the failed testcase. If the test fails there could be a bug in the code or the testcase wasn’t defined correctly. Please check the testcases first to avoid unnecessary search in the programme code.

New testcases:

If you want to create new testcases you are welcome to do that in the classes TestUsers and TestHabits. It is commented, where the new testcases should be inserted.

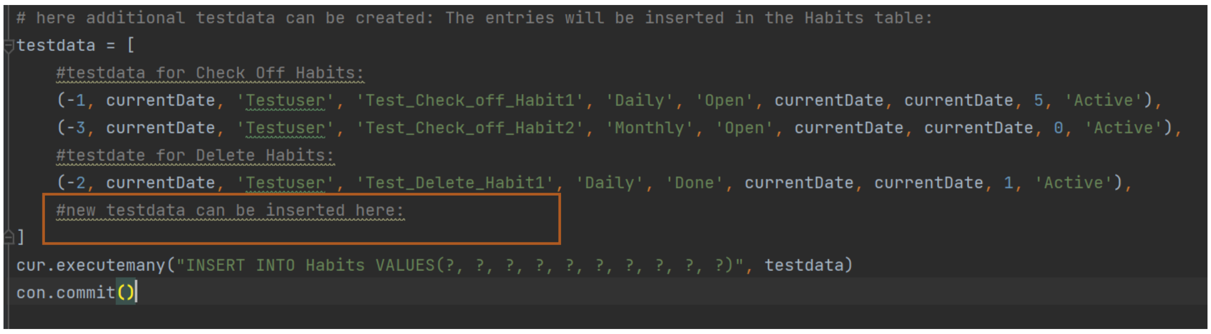


```
class TestUsers:
    def setup_method(self):
        self.test_users = Users("Testuser")

    def test_register(self):
        # tests the register function, and creates a new User named "Tes
        self.test_users.register("Max", "Mustermann", "max@gmail.com", "
        cur.execute("SELECT * FROM User WHERE User='Testuser'")
        result = cur.fetchone()
        assert result == ("Testuser", "Max", "Mustermann", "max@gmail.co

    def test_login(self):
        # tests the login function with the "Testuser"
        x = self.test_users.login('123')
        assert x == "Login successful"

    # new testcases at the user functions can be inserted here:
```



```
    def test_check_off_habit2(self):
        # tests the check off function and the streak, here the t
        self.test_habits.check_off_habit("Test_Check_off_Habit2",
        result = []
        for row in cur.execute("SELECT * FROM Habits WHERE Name='
            result.append(row[1:])
        assert result == [(currentDate.strftime('%Y-%m-%d'), 'Tes

    # you can create further testcase here:
```

If you are in need of new testdata for the testcases you can create the entries at the defined section:



```
# here additional testdata can be created: The entries will be inserted in the Habits table:
testdata = [
    #testdata for Check Off Habits:
    (-1, currentDate, 'Testuser', 'Test_Check_off_Habit1', 'Daily', 'Open', currentDate, currentDate, 5, 'Active'),
    (-3, currentDate, 'Testuser', 'Test_Check_off_Habit2', 'Monthly', 'Open', currentDate, currentDate, 0, 'Active'),
    #testdate for Delete Habits:
    (-2, currentDate, 'Testuser', 'Test_Delete_Habit1', 'Daily', 'Done', currentDate, currentDate, 1, 'Active'),
    #new testdata can be inserted here:
]
cur.executemany("INSERT INTO Habits VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", testdata)
con.commit()
```

Please pay attention to create a new unique ID (first entry) and choose a negative number, fe -10, to avoid problems.