Habit tracker design and implementation procedure:

Following the flowchart diagram from the conception phase I implemented the code necessary for the habit tracker app. A selection of code for presentation purposes is included via screen capture in this document. The complete project can be found on my GitHub page under the link: https://github.com/dariuszarse/oop_course_habit_tracker

Used tools:

As my code editor I used Visual Studio Code. For code libraries I used the following:

```python
from datetime import datetime
from dateutil.relativedelta import relativedelta
import sqlite3
import re
import os
```

datetime and dateutil.relativedelta are great tools to capture timestamps and work with dates. sqlite3 is a tool for creating and managing databases in python code. This is key for data retention in between user sessions. I used re for matching and comparing user input and (string) variables to ensure correct input data and to prevent errors through bad input at runtime. os was used for simply clearing the terminal to make it look just that tiny bit cleaner.

Basic database structure:

For the app to save user data in between usages my first step was to implement a database that could store the habit data. The database gets created as a new file in the folder of main.py when the app starts up and will fill with some example habits for the user to experiment with. This should enable users to quickly understand the workings of the app without having to create their own habits.

Implementation of the habit table in the database:

```python
#creating the table if it doesn't exist already
cursor.execute("""CREATE TABLE IF NOT EXISTS HABITS
(   HABIT_NAME CHAR(50),
    HABIT_FREQUENCY CHAR(1),
    HABIT_START_DATE DATE,
    HABIT_CHECK_OFF_LAST DATE,
    HABIT_CHECK_OFF_NEXT DATE,
    HABIT_CURRENT_STREAK INT(5),
    HABIT_LONGEST_STREAK INT(5))""")
```

User interface:

For users to interact with the backend of the app they need some form of UI. I chose the simple method of user input within the terminal, so an IDE/code editor that provides a terminal or a command-line interface is essential for usage of the app. While this is nothing fancy looking, it does provide all the basics needed and is incredibly simple to use. User choices are just entered into the terminal whenever the program prompts to do so and that's it.

Class implementation habit:

Here is a screen capture of the class habit and its initiate function that is implemented to provide a predetermined structure all newly created habits.

```
class Habit:
    def __init__(self, habit_name, habit_frequency, habit_start_date, habit_check_off_last='', habit_check_off_next='', habit_current_streak=0, habit_longest_streak=0):
        self.habit_name = habit_name
        self.habit_frequency = self.is_valid_frequency()
        self.habit_start_date = habit_start_date
        self.habit_check_off_last = ''
        self.habit_check_off_next = self.calculate_check_off_next()
        self.habit_current_streak = 0
        self.habit_longest_streak = 0
```

App navigation:

Once that setup is taken care of a user can just start the app and will be greeted with this screen:

```
Welcome to this habit tracking app. I wish you all the best in chasing your goals!

You're in the main menu. What would you like to do?
You have the following options:
    [1] Display a selection of habits and/or streaks
    [2] Check off a habit for today
    [3] Create a new habit
    [4] Edit or delete an existing habit
    [5] Quit the app
Please enter the number belonging to the option of your choice here:
```

This is the main menu. From here all functions can be called. All interaction with the database will take place with this as a starting point as well. The main menu uses a simple user input validation system with re.match (imported from the re library) to ensure a correct selection of one of the menus options. The main menu function code (including the re.match validation) can be seen below:

```python
#defining navigation function
def main_menu():
    while True:
        user_choice = input(main_menu_message)
        regex = "[1-5]"
        if not re.match(regex, user_choice):
            print("\nThat is not a valid option. Try again!")
        elif user_choice == '1':
            display_habits()
            break
        elif user_choice == '2':
            check_off_habit()
            break
        elif user_choice == '3':
            create_new_habit()
            break
        elif user_choice == '4':
            edit_habit()
            break
        elif user_choice == '5':
            quit_app()
            break
```

Let's look at where the main menu options take us:

Main Menu Option [1] Display a selection of habits and/or streaks:

This section is used for inspecting and analyzing habits. Here users can choose what selection of habit data they want to have displayed. The options can be seen in the screen capture below:

```
What would you like to be displayed?
    [1] All tracked habits (ordered by name)
    [2] Your daily habits
    [3] Your weekly habits
    [4] Your monthly habits
    [5] Your longest active streak
    [6] Your longest ever streak
    [7] Your most urgent habit to check off
```

For a simple display of all tracked habits the user can enter 1 and gets presented with these example habits which have been created at the app startup:

```
Here are all your currently tracked habits:
[('Coding', 'D', '2023-03-13', '2023-05-16', '2023-05-17', 54, 54), ('Dancing', 'W', '2023-04-25', '2023-05-08', '2023-05-15', 0, 3), ('Exercise', 'D', '2023-04-20', '2023-05-15', '2023-05-16', 0, 15), ('Hairdresser', 'M', '2023-01-03', '2023-05-06', '2023-06-06', 2, 2), ('Swimming', 'W', '2023-05-12', '2023-05-12', '2023-05-19', 1, 1)]
```

For now, it displays all the habits (including their full set of data), but this can easily be adapted alongside any of the future queries shown down below. All it takes is a small change to one of the SQL SELECT * statements (one example below):

```
#retrieving data from database according to user input
elif user_choice == '1':
    cursor.execute("SELECT * FROM HABITS ORDER BY HABIT_NAME")
    print('\nHere are all your currently tracked habits:')
    break
```

Other options for display include all habits of a certain periodicity, the longest currently active streak, the longest ever streak and which habit needs to be checked off next (the most urgent one). In order to keep it compact I'll only show selected examples (for the purpose of showcasing other display formats):

Longest active streak (portrayed with habit name and active streak):

```
Here is your longest active streak:
[('Coding', 54)]
```

and most urgent habit to check off:

```
Here is your most urgent habit with its due date:
[('Dancing', '2023-05-15')]
```

Main Menu Option [2] Check off a habit for today:

In order to check off a habit it is important to select the correct habit first, which is done via user input into the terminal. This is what the user is presented with on choosing option 2:

```
Here are all your currently tracked habits:
[('Coding',), ('Dancing',), ('Exercise',), ('Hairdresser',), ('Swimming',)]

Which habit would you like to check off? Please enter the exact name of the habit here:
▌
```

Please note that the habit name must be spelled exactly as presented in order for the SQL query to find it in the database. Therefore, I employ another re.match validation statement to ensure that. After correctly selecting a habit, the user is presented with feedback from the app as follows:
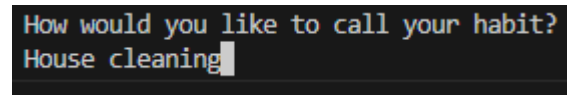
```
You have selected to check off your habit ('Dancing', 'W', '2023-04-25', '2023-05-08', '2023-05-15', 0, 3)
Your habit is being updated!
```

And this already concludes the check off section. I tried to keep it as simple as possible as according to my expectations this should be the most frequently used functions and shouldn't require too much input and effort on the user's side. In the background the program does increase the current streak by one for every successful habit check off and in order to keep the longest streak accurate it will set the longest streak equal to the current one if the current streak is bigger than the longest one.
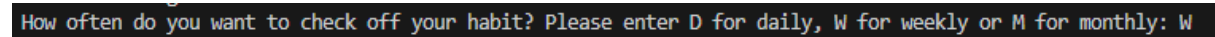
This does allow checking off the same habit multiple times a day and increasing the streak by +2 on a single day for example. While this should not be abused, I decided to not change it so that habits like 'exercise' can be checked off for a morning and evening workout or a habit like 'make new friends' can increase by +1 or +3 on a single day. Using this responsibly falls to the user.

Main Menu Option [3] Create a new habit:

In the class implementation of the Habit class most attributes are already automatically implemented on creation. The only ones that require user input are the name and frequency of the habit. The user is presented with these two questions:


```
How would you like to call your habit?
House cleaning
```
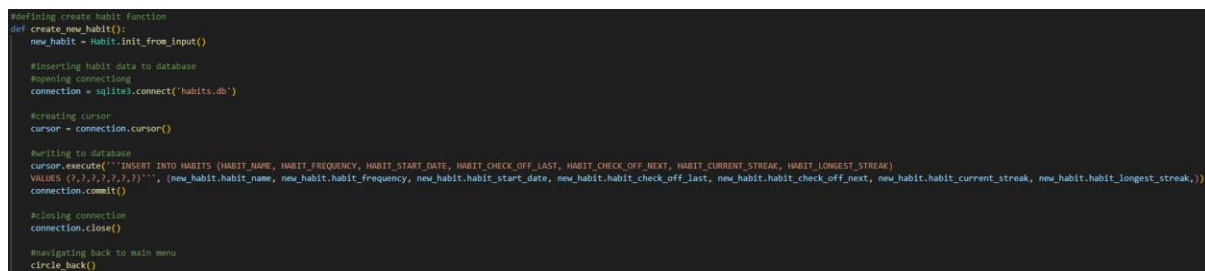


```
How often do you want to check off your habit? Please enter D for daily, W for weekly or M for monthly: W
```

Habit names are subject to the user's imagination and creativity, but frequency is restricted to daily, weekly and monthly. Therefore, I needed to implement another re.match on the respective letters for those frequencies. With these two inputs the rest of the attributes will be implemented (the start date of tracking the habit is taken from datetime, streaks start at 0 by default and the next check off is determined by today's date and the frequency). This data then gets written to the database. For illustration purposes, below you can see what the new weekly habit House cleaning would look like in the database:

HABIT_NAME	HABIT_FREQUENCY	HABIT_START_DATE	HABIT_CHECK_OFF_LAST	HABIT_CHECK_OFF_NEXT	HABIT_CURRENT_STREAK	HABIT_LONGEST_STREAK
Swimming	W	2023-05-12	2023-05-12	2023-05-19	1	1
Exercise	D	2023-04-20	2023-05-15	2023-05-16	0	15
Hairdresser	M	2023-01-03	2023-05-06	2023-06-06	2	2
Dancing	W	2023-04-25	2023-05-17	2023-05-24	1	3
Coding	D	2023-03-13	2023-05-16	2023-05-17	54	54
Sleeping	D	2023-05-17		2023-05-18	0	0
House cleaning	W	2023-05-17		2023-05-24	0	0

Thanks to implementing the innit function within the class itself I could keep the code for this section very compact. A major part of it is connecting to the database and committing the code to it:


```
#defining create habit function
def create_new_habit():
    new_habit = Habit.init_from_input()

    #inserting habit data to database
    #opening connection
    connection = sqlite3.connect('habits.db')

    #creating cursor
    cursor = connection.cursor()

    #writing to database
    cursor.execute('''INSERT INTO HABITS (HABIT_NAME, HABIT_FREQUENCY, HABIT_START_DATE, HABIT_CHECK_OFF_LAST, HABIT_CHECK_OFF_NEXT, HABIT_CURRENT_STREAK, HABIT_LONGEST_STREAK)
    VALUES (?,?,?,?,?,?,?)''', (new_habit.habit_name, new_habit.habit_frequency, new_habit.habit_start_date, new_habit.habit_check_off_last, new_habit.habit_check_off_next, new_habit.habit_current_streak, new_habit.habit_longest_streak,))
    connection.commit()

    #closing connection
    connection.close()

    #navigating back to main menu
    circle_back()
```

Main Menu Option [4] Edit or delete an existing habit:

This is the last major function regarding habits that can be called from the main menu. I grouped editing and deleting together because they both must access a specific habit in the database and then update that data to new values or delete it entirely. Therefore, the functionality seemed quite similar.

This function works by first selecting a habit through user input. Here I used the same validation function with re.match as is used for the habit check off. For the user this looks as follows:

```
Here are all your currently tracked habits:
[('Coding',), ('Dancing',), ('Exercise',), ('Hairdresser',), ('House cleaning',), ('Sleeping',), ('Swimming',)]

Which habit would you like to edit? Please enter the exact name of the habit here:
```

Let's say the user wants to delete the House cleaning habit again. This leads to the following screen:

```
You have selected to edit your habit House cleaning

You have the following options:
[1] Edit the name of the habit
[2] Edit the frequency of checking off the habit
[3] Delete the habit
Please enter what you would like to do here: []
```

Editing the name doesn't require any input check because all names are valid. Editing the frequency will make use of the same frequency input check that I used in the function for creating a habit. Both has been shown previously. Here we choose to delete the habit which drops it out of the database:

HABIT_NAME	HABIT_FREQUENCY	HABIT_START_DATE	HABIT_CHECK_OFF_LAST	HABIT_CHECK_OFF_NEXT	HABIT_CURRENT_STREAK	HABIT_LONGEST_STREAK
Swimming	W	2023-05-12	2023-05-12	2023-05-19	1	1
Exercise	D	2023-04-20	2023-05-15	2023-05-16	0	15
Hairdresser	M	2023-01-03	2023-05-06	2023-06-06	2	2
Dancing	W	2023-04-25	2023-05-17	2023-05-24	1	3
Coding	D	2023-03-13	2023-05-16	2023-05-17	54	54
Sleeping	D	2023-05-17		2023-05-18	0	0

Main Menu Option [5] Quit the app:

Quitting the app is quite straightforward. I chose to simply print a short goodbye message and that is it.

```
Have a great day! Good luck chasing your goals! :)
```

Circle back function:

After a function (other than quit the app) from the main menu is called and executed it calls the circle back function. This is to prevent having to reopen the app for every single function the user wants to employ. The circle back function is simply asking the user if they want to return to the main menu or quit the app:

```
Would you like to return to the main menu or quit the app?
[1] Main Menu
[2] Quit the App
```

Choosing 1 for the main menu will call the main menu function and allow the user to perform another action, quit the app will call the quit function and print the above shown goodbye message.

Streak updates:

So far, a streak does increase on successful check off but will never decrease. Habit streaks can't be broken yet. In order to keep it simple for the user I decided to implement this in the background on app startup, so it doesn't require any input on if habits have been checked off or not. *This also means that if a user forgets to update a habit on the day they complete it, the streak will break even if they have completed it and just forgot to check it off.*

On app startup all dates in the check off next column of the database are compared with today's date. If the date is further in the past than todays date the streak will be updated to 0.

Potential changes till finalization:

There is still some work to be done to make the format consistent when it comes to requiring user input on a specific line, new line characters need to be implemented, some extensive error checking (marginal cases of random user input, etc).