Object Oriented and Functional Programming with Python

Professor Dr. Pumperla, Max

Bruno Silva
UPS10620925

INTERNATIONALE HOCHSCHULE

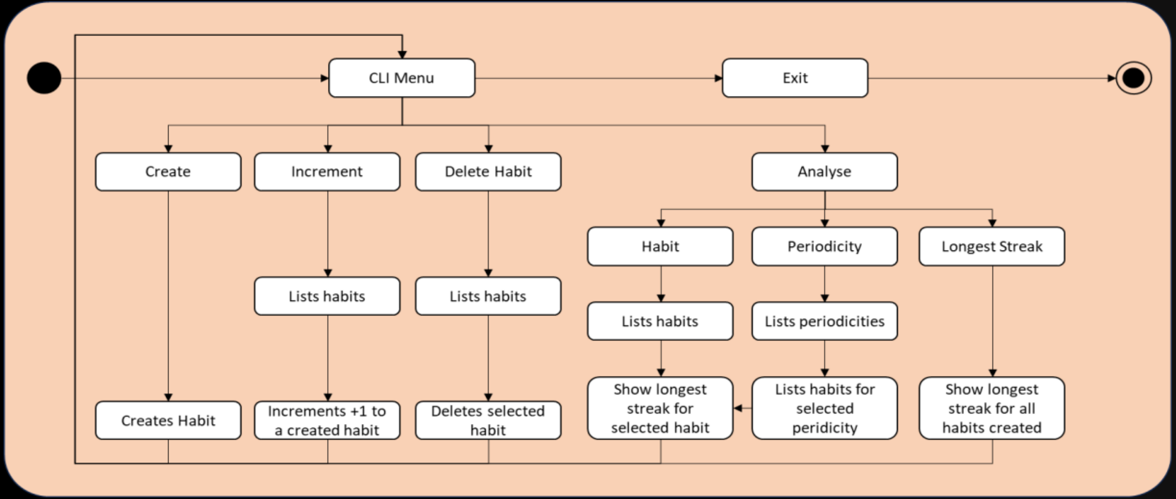
Habit Tracking App Project

Development phase

Quick review of the Habit Tracking App

This is an app created within the "Object Oriented and Functional Programming with Python" module by IU – International University. It is designed to keep track of certain habits, meaning it is possible to create new habits, delete them or increment them every time you accomplished that habit. Alongside with the app functionality, a test module and a database loader (creates a database with 5 habits and a month worth of dates) will be provided.

Habit Tracking App flowchart



Used Liraries

questionary – used to create a simplified text-based menu (command line interface). It also allows to chain a sequence of prompts which will be used.

sqlite3 – used to create databases and its tables, data manipulation (deleting counters) and SQL queries.

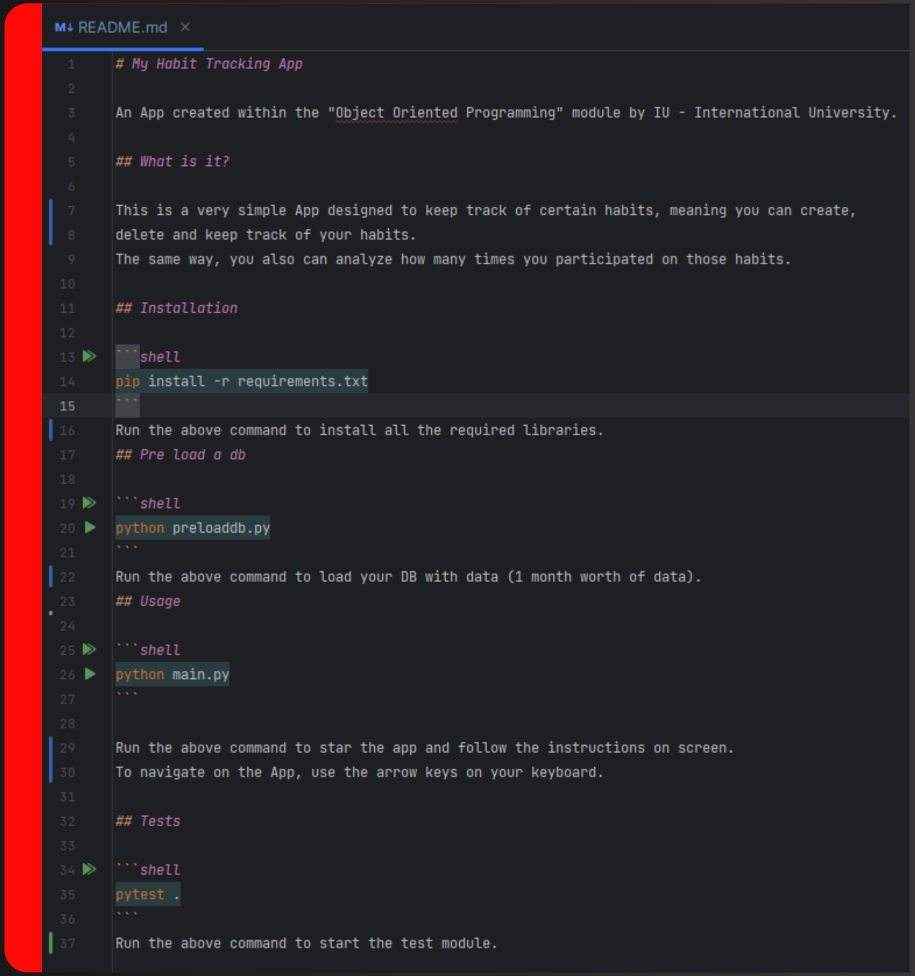
datetime – used to get the current time date, format dates and to parse strings into datetime objects.

pandas – used for data manipulation (grouping data acording to the defined criteria) to obtain the streak count of the habits.

pytest – used to test functionalities and expected results in the projects's functions.

README.md		Run Command
1	Install Requirements	pip install –r requirements.txt
2	Pre Loading a DB Instructions	python preloaddb.py
3	Starting the APP	python main.py
4	Starting the Test Module	pytest .

Readme Code



~~~
# My Habit Tracking App

An App created within the "Object Oriented Programming" module by IU - International University.

## What is it?

This is a very simple App designed to keep track of certain habits, meaning you can create,
delete and keep track of your habits.
The same way, you also can analyze how many times you participated on those habits.

## Installation

```shell
pip install -r requirements.txt
```
Run the above command to install all the required libraries.
## Pre load a db

```shell
python preloaddb.py
```
Run the above command to load your DB with data (1 month worth of data).
## Usage

```shell
python main.py
```


Run the above command to star the app and follow the instructions on screen.
To navigate on the App, use the arrow keys on your keyboard.

## Tests

```shell
pytest .
```
Run the above command to start the test module.
~~~

Counter Class

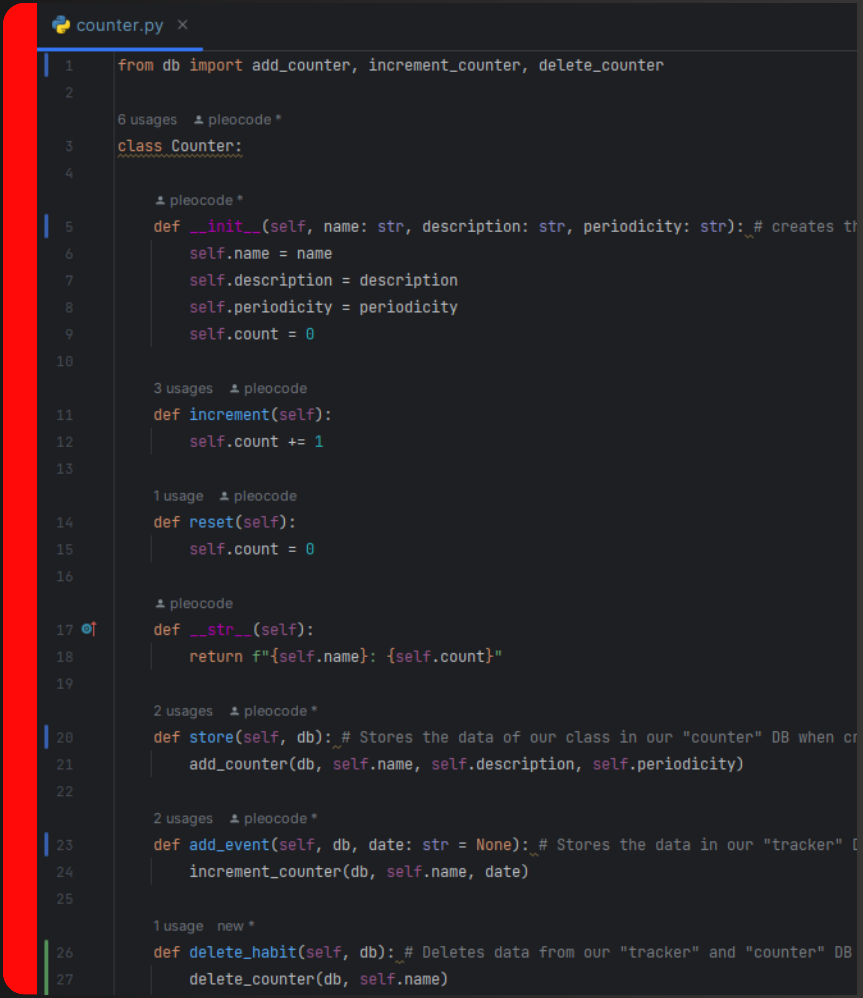
Defining the initializer for the “Counter” class. The constructor is called when we create a new instance (object) of the class with 3 parameters, name, description and periodicity. “self” refers to the instance being created and allows us to set attributes for that instance.

store function:
Used to add a new habit to the database’s “counter” table by using the “add_counter” function defined in DB.py.

add_event function:
Used to add an event (name and date) to the database’s “tracker” table by using the “increment_counter” function defined in DB.py.

delete_habit function:
Used to delete data from our database’s “tracker” and “counter” tables by using the “delete_counter” function defined in DB.py.

Counter Class code



```python
from db import add_counter, increment_counter, delete_counter

6 usages  pleocode *
class Counter:

    pleocode *
    def __init__(self, name: str, description: str, periodicity: str): # creates th
        self.name = name
        self.description = description
        self.periodicity = periodicity
        self.count = 0

    3 usages  pleocode
    def increment(self):
        self.count += 1

    1 usage  pleocode
    def reset(self):
        self.count = 0

    pleocode
    def __str__(self):
        return f"{self.name}: {self.count}"

    2 usages  pleocode *
    def store(self, db): # Stores the data of our class in our "counter" DB when cr
        add_counter(db, self.name, self.description, self.periodicity)

    2 usages  pleocode *
    def add_event(self, db, date: str = None): # Stores the data in our "tracker" D
        increment_counter(db, self.name, date)

    1 usage  new *
    def delete_habit(self, db): # Deletes data from our "tracker" and "counter" DB
        delete_counter(db, self.name)
```

Data Base 1/3

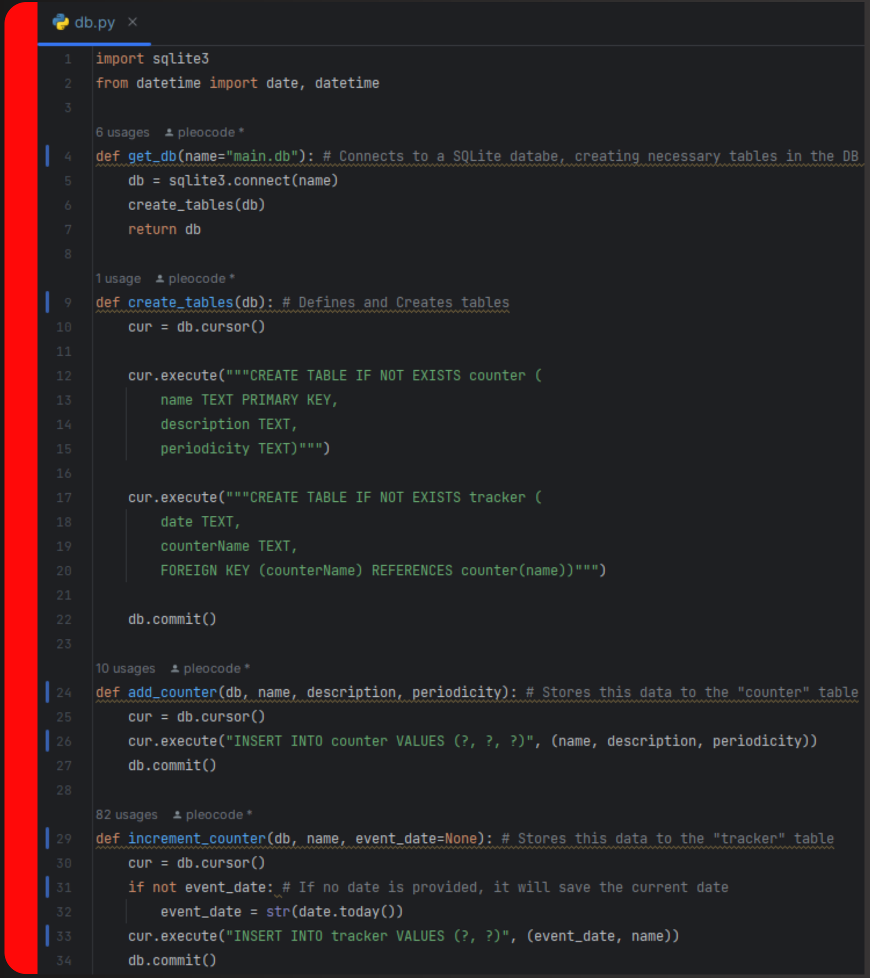
create_tables function:
Responsible for creating two talbes in a SQLite database:
counter table – table with name as primary key, description and periodity.
tracker table – table with counterName as foreign key and date.

get_db function:
Connects to a SQLite databe, creating the necessary tables in the DB defined in the "create_tables" function and then returns the database connection. The function simplifies the process of connecting to the database and prepares it for further use.

add_counter function:
This function is used to store information such as the name of the habit, its description and its priodicity in the database's "counter" table.

increment_counter function:
This function is used to store the date (current date if none is provided) of the increment event and its name in the database's "tracker" table.

DB code


db.py

```python
import sqlite3
from datetime import date, datetime

6 usages  pleocode *
def get_db(name="main.db"): # Connects to a SQLite databe, creating necessary tables in the DB
    db = sqlite3.connect(name)
    create_tables(db)
    return db

1 usage  pleocode *
def create_tables(db): # Defines and Creates tables
    cur = db.cursor()

    cur.execute("""CREATE TABLE IF NOT EXISTS counter (
        name TEXT PRIMARY KEY,
        description TEXT,
        periodicity TEXT)""")

    cur.execute("""CREATE TABLE IF NOT EXISTS tracker (
        date TEXT,
        counterName TEXT,
        FOREIGN KEY (counterName) REFERENCES counter(name))""")

    db.commit()

10 usages  pleocode *
def add_counter(db, name, description, periodicity): # Stores this data to the "counter" table
    cur = db.cursor()
    cur.execute("INSERT INTO counter VALUES (?, ?, ?)", (name, description, periodicity))
    db.commit()

82 usages  pleocode *
def increment_counter(db, name, event_date=None): # Stores this data to the "tracker" table
    cur = db.cursor()
    if not event_date: # If no date is provided, it will save the current date
        event_date = str(date.today())
    cur.execute("INSERT INTO tracker VALUES (?, ?)", (event_date, name))
    db.commit()
```

Data Base 2/3

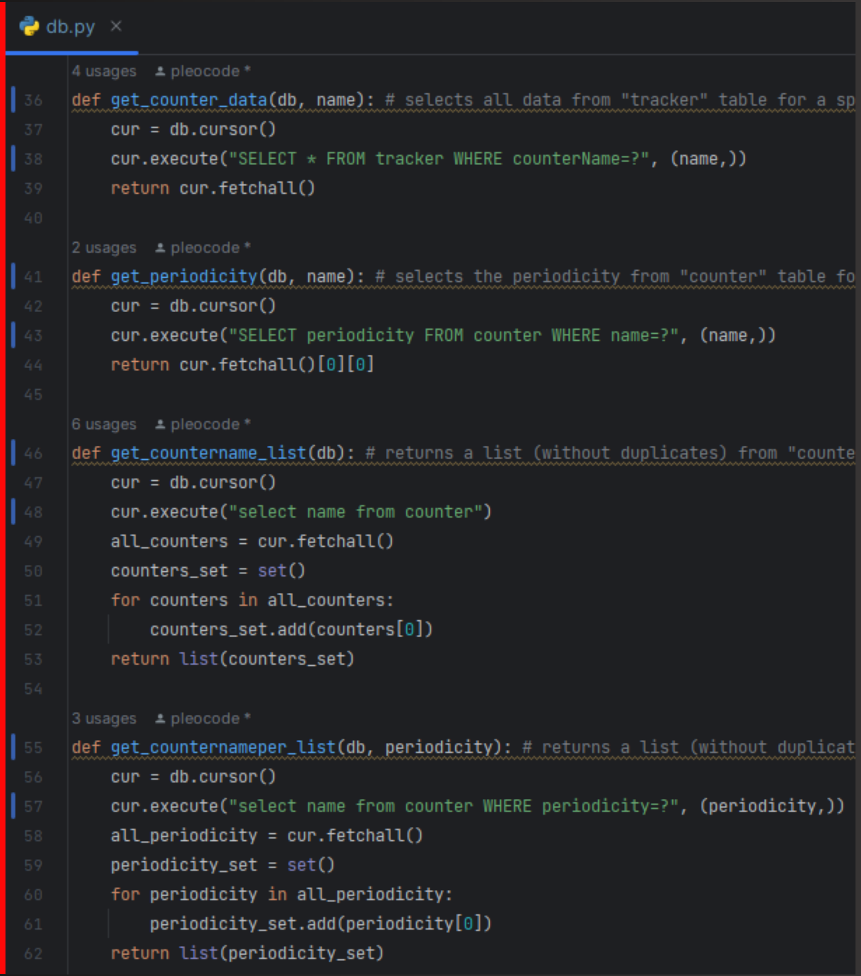
DB code

get_counter_data function:
This function is used to fetch information about all the incremented events (“tracker” table) associated with the selected habit. The function returns a list of tuples, where each represents a row of data including the date and the name of the habit.

get_periodicity function:
Retrieves the peirodicity from “counter” table for a specific habit. The function returns a string representing the periodicity of the habit.

get_countername_list function:
Retrieves a list of habits from the “counter” table. The function returns a list without any duplicate habit names.

get_counternameper_list function:
Retrieves a list of habits from the “counter” table for a specific periodicity. The function returns a list without any duplicate habit names.



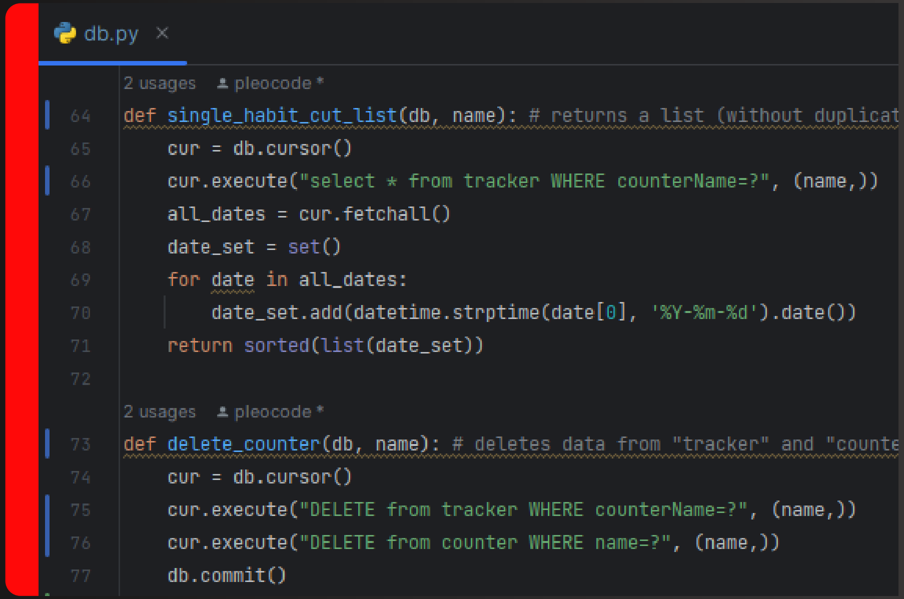
```
4 usages  pleocode *
def get_counter_data(db, name): # selects all data from "tracker" table for a sp
    cur = db.cursor()
    cur.execute("SELECT * FROM tracker WHERE counterName=?", (name,))
    return cur.fetchall()

2 usages  pleocode *
def get_periodicity(db, name): # selects the periodicity from "counter" table fo
    cur = db.cursor()
    cur.execute("SELECT periodicity FROM counter WHERE name=?", (name,))
    return cur.fetchall()[0][0]

6 usages  pleocode *
def get_countername_list(db): # returns a list (without duplicates) from "counte
    cur = db.cursor()
    cur.execute("select name from counter")
    all_counters = cur.fetchall()
    counters_set = set()
    for counters in all_counters:
        counters_set.add(counters[0])
    return list(counters_set)

3 usages  pleocode *
def get_counternameper_list(db, periodicity): # returns a list (without duplicat
    cur = db.cursor()
    cur.execute("select name from counter WHERE periodicity=?", (periodicity,))
    all_periodicity = cur.fetchall()
    periodicity_set = set()
    for periodicity in all_periodicity:
        periodicity_set.add(periodicity[0])
    return list(periodicity_set)
```

Data Base 3/3

DB code

single_habit_cut_list function:
Retrieves a list of unique incrementation dates for a specific habit from “tracker” table. The function returns a sorted list of dates without duplicates.

delete_counter function:
Deletes data related to a specific habit from both the “tracker” and “counter” tables. This can be useful to remove a habit and all its tracking history.



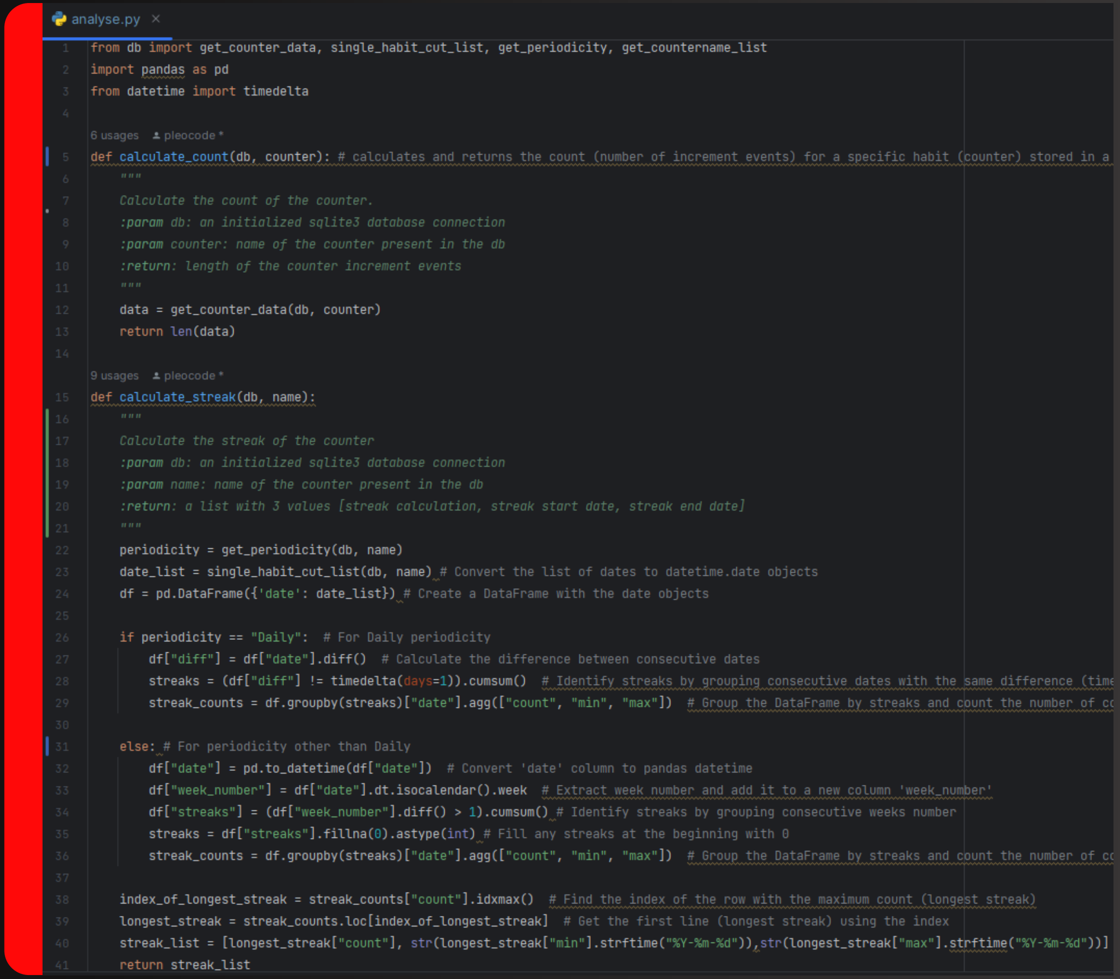
```
2 usages  pleocode *
def single_habit_cut_list(db, name): # returns a list (without duplicat
    cur = db.cursor()
    cur.execute("select * from tracker WHERE counterName=?", (name,))
    all_dates = cur.fetchall()
    date_set = set()
    for date in all_dates:
        date_set.add(datetime.strptime(date[0], '%Y-%m-%d').date())
    return sorted(list(date_set))

2 usages  pleocode *
def delete_counter(db, name): # deletes data from "tracker" and "counte
    cur = db.cursor()
    cur.execute("DELETE from tracker WHERE counterName=?", (name,))
    cur.execute("DELETE from counter WHERE name=?", (name,))
    db.commit()
```

analyse 1/2

Analyse code

calculate_count function:
Calculates and returns the count (number of incremented events) for a specific habit (counter) stored in “counter” table. This count represents the number of times the habit has been performed.

calculate_streak function:
Calculates and returns information about the longest streak of incremented events for a specific habit stored in the database. The calculation method differs based on the periodicity of the habit. The “get_periodicity” and “single_habit_cut_list” are defined in DB.py to fetch the necessary data for the calculation. Additionally, the code uses the pandas library for data manipulation. The returned information will be in a form of list with [streak calculation, streak start date, streak end date].



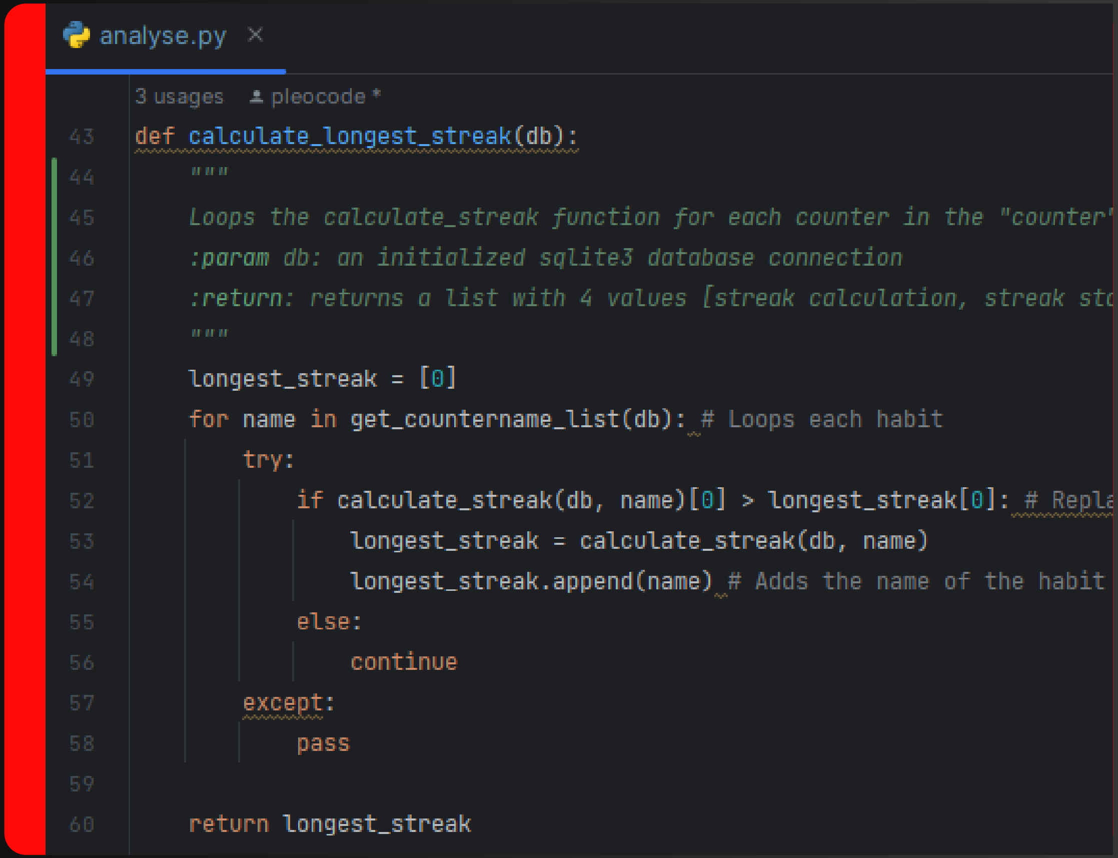
analyse.py ×

```
from db import get_counter_data, single_habit_cut_list, get_periodicity, get_countername_list
import pandas as pd
from datetime import timedelta

6 usages  pleocode *
def calculate_count(db, counter): # calculates and returns the count (number of increment events) for a specific habit (counter) stored in a
    """
    Calculate the count of the counter.
    :param db: an initialized sqlite3 database connection
    :param counter: name of the counter present in the db
    :return: length of the counter increment events
    """
    data = get_counter_data(db, counter)
    return len(data)

9 usages  pleocode *
def calculate_streak(db, name):
    """
    Calculate the streak of the counter
    :param db: an initialized sqlite3 database connection
    :param name: name of the counter present in the db
    :return: a list with 3 values [streak calculation, streak start date, streak end date]
    """
    periodicity = get_periodicity(db, name)
    date_list = single_habit_cut_list(db, name) # Convert the list of dates to datetime.date objects
    df = pd.DataFrame({'date': date_list}) # Create a DataFrame with the date objects

    if periodicity == "Daily":  # For Daily periodicity
        df["diff"] = df["date"].diff()  # Calculate the difference between consecutive dates
        streaks = (df["diff"] != timedelta(days=1)).cumsum()  # Identify streaks by grouping consecutive dates with the same difference (time
        streak_counts = df.groupby(streaks)["date"].agg(["count", "min", "max"])  # Group the DataFrame by streaks and count the number of co

    else: # For periodicity other than Daily
        df["date"] = pd.to_datetime(df["date"])  # Convert 'date' column to pandas datetime
        df["week_number"] = df["date"].dt.isocalendar().week  # Extract week number and add it to a new column 'week_number'
        df["streaks"] = (df["week_number"].diff() > 1).cumsum() # Identify streaks by grouping consecutive weeks number
        streaks = df["streaks"].fillna(0).astype(int) # Fill any streaks at the beginning with 0
        streak_counts = df.groupby(streaks)["date"].agg(["count", "min", "max"])  # Group the DataFrame by streaks and count the number of co

    index_of_longest_streak = streak_counts["count"].idxmax()  # Find the index of the row with the maximum count (longest streak)
    longest_streak = streak_counts.loc[index_of_longest_streak]  # Get the first line (longest streak) using the index
    streak_list = [longest_streak["count"], str(longest_streak["min"].strftime("%Y-%m-%d")),str(longest_streak["max"].strftime("%Y-%m-%d"))]
    return streak_list
```

analyse 2/2

Analyse code

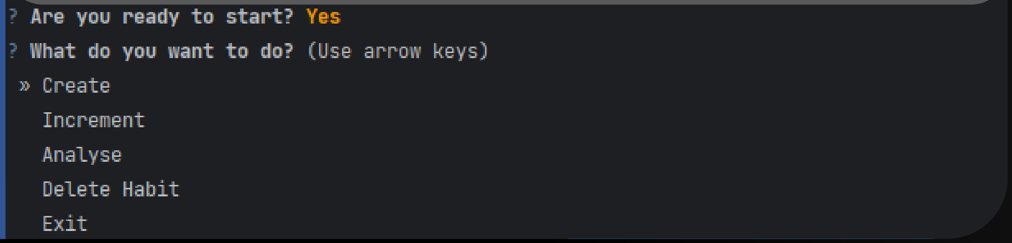
calculate_longest_streak function:
Loops through each counter present int the “counter” table, calculates the longest streak for each habit using the “calculate_streak” function, and returns information about the longest streak amongst all habits. The returned information will be in a form of list with [streak calculation, streak start date, streak end date, name of the counter].



```python
def calculate_longest_streak(db):
    """
    Loops the calculate_streak function for each counter in the "counter"
    :param db: an initialized sqlite3 database connection
    :return: returns a list with 4 values [streak calculation, streak sta
    """
    longest_streak = [0]
    for name in get_countername_list(db): # Loops each habit
        try:
            if calculate_streak(db, name)[0] > longest_streak[0]: # Repla
                longest_streak = calculate_streak(db, name)
                longest_streak.append(name) # Adds the name of the habit
            else:
                continue
        except:
            pass

    return longest_streak
```

main 1/5 – Create Command

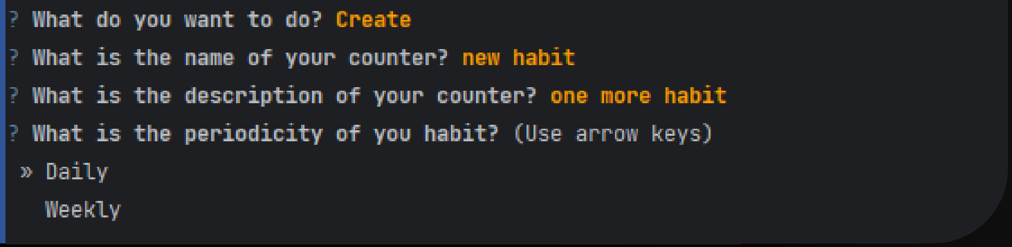
Create an interactive command-line menu where the user can select one of the listed actions, “Create”, “Increment”, “Analyse”, “Delete Habit”, “Exit”. The loop will keep presenting the menu to the user until they choose to exit by selecting “Exit”. The “questionary” library simplifies the process of creating this interactive command-line interface.



```
? Are you ready to start? Yes
? What do you want to do? (Use arrow keys)
 » Create
   Increment
   Analyse
   Delete Habit
   Exit
```

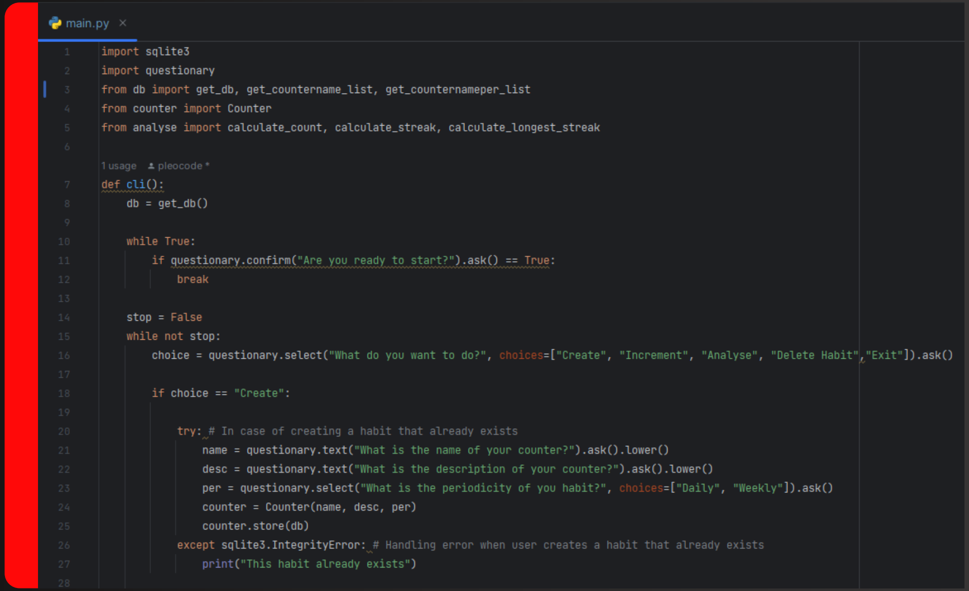
Create command:

Command used when the user wants to create a new habit. The user will be needed to manually input the name and description of the habit and to select its periodicity (daily or weekly). I wanted to avoid mistakes by reducing the human manual input as much as possible. The variables are converted and stored in lower cases to avoid duplicates (example: “Study” and “study”). This information will be stores in the “counter” table.



```
? What do you want to do? Create
? What is the name of your counter? new habit
? What is the description of your counter? one more habit
? What is the periodicity of you habit? (Use arrow keys)
 » Daily
   Weekly
```

main code



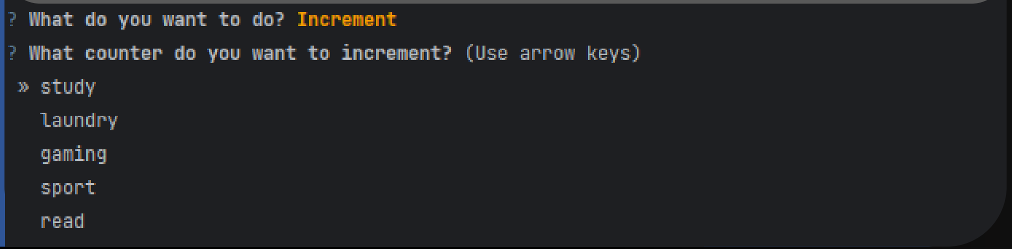
```python
import sqlite3
import questionary
from db import get_db, get_countername_list, get_counternameper_list
from counter import Counter
from analyse import calculate_count, calculate_streak, calculate_longest_streak

def cli():
    db = get_db()

    while True:
        if questionary.confirm("Are you ready to start?").ask() == True:
            break

    stop = False
    while not stop:
        choice = questionary.select("What do you want to do?", choices=["Create", "Increment", "Analyse", "Delete Habit","Exit"]).ask()

        if choice == "Create":

            try: # In case of creating a habit that already exists
                name = questionary.text("What is the name of your counter?").ask().lower()
                desc = questionary.text("What is the description of your counter?").ask().lower()
                per = questionary.select("What is the periodicity of you habit?", choices=["Daily", "Weekly"]).ask()
                counter = Counter(name, desc, per)
                counter.store(db)
            except sqlite3.IntegrityError: # Handling error when user creates a habit that already exists
                print("This habit already exists")
```

The code has an error handling logic and helps improve the user experience by providing informative feedback when trying to create a habit that already exists in the DB.

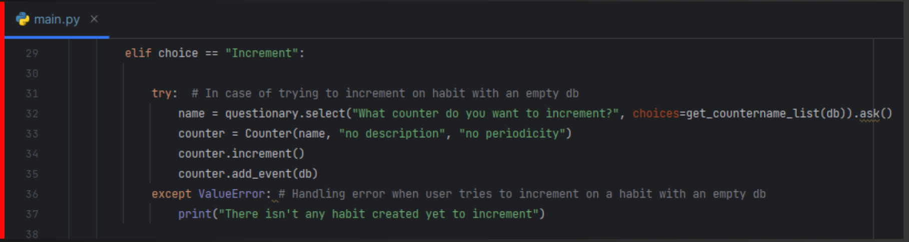
main 2/5 – Increment Command

main code

Increment command:
Command used when the user wants to increment a previously created counter. When incrementing, the date and the name of the counter, will be stored into the database's "tracker" table.



```
? What do you want to do? Increment
? What counter do you want to increment? (Use arrow keys)
 » study
   laundry
   gaming
   sport
   read
```



main.py ×

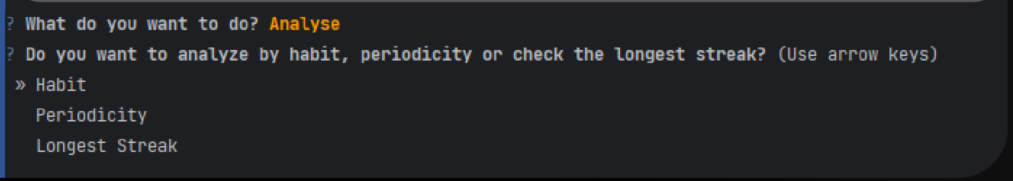
```python
        elif choice == "Increment":

            try:  # In case of trying to increment on habit with an empty db
                name = questionary.select("What counter do you want to increment?", choices=get_countername_list(db)).ask()
                counter = Counter(name, "no description", "no periodicity")
                counter.increment()
                counter.add_event(db)
            except ValueError: # Handling error when user tries to increment on a habit with an empty db
                print("There isn't any habit created yet to increment")
```

The code has an error handling logic and helps improve the user experience by providing informative feedback when attempting retrieve the habit list of an empty database.

main 3/5 – Analyse Command

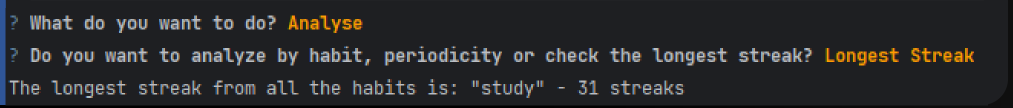
main code

Analyse command:
When selecting analyse, the user will be prompted to select one option from the desired actions, analyze by "Habit", analyze by "Periodicity" or analyze the "Longest Streak".

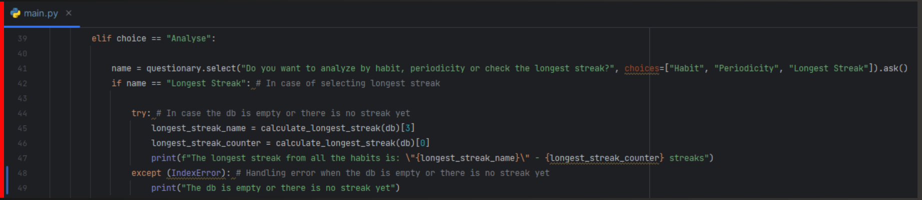


```
? What do you want to do? Analyse
? Do you want to analyze by habit, periodicity or check the longest streak? (Use arrow keys)
 » Habit
   Periodicity
   Longest Streak
```

Longest Streak command:
Provides the user an informative string about the longest streak from all the habits stored in the database



```
? What do you want to do? Analyse
? Do you want to analyze by habit, periodicity or check the longest streak? Longest Streak
The longest streak from all the habits is: "study" - 31 streaks
```



main.py

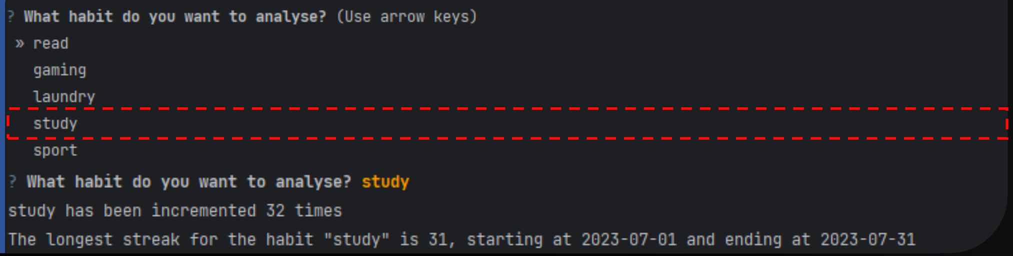
```python
        elif choice == "Analyse":

            name = questionary.select("Do you want to analyze by habit, periodicity or check the longest streak?", choices=["Habit", "Periodicity", "Longest Streak"]).ask()
            if name == "Longest Streak": # In case of selecting longest streak

                try: # In case the db is empty or there is no streak yet
                    longest_streak_name = calculate_longest_streak(db)[3]
                    longest_streak_counter = calculate_longest_streak(db)[0]
                    print(f"The longest streak from all the habits is: \"{longest_streak_name}\" - {longest_streak_counter} streaks")
                except (IndexError): # Handling error when the db is empty or there is no streak yet
                    print("The db is empty or there is no streak yet")
```

The code has an error handling logic and helps improve the user experience by providing informative feedback when attempting retrieve the longest streak with an empty database.

main 4/5 – Analyse Command

Habit command:

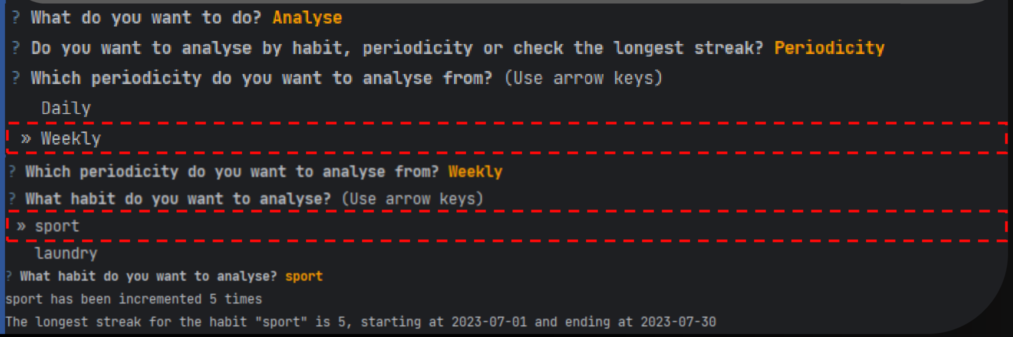
When selecting the habit command, a list will show with all the habits stored in the DB. Upon selecting the desired habit, the icrementation times and the longest streak for that habit will be shown.



```
? What habit do you want to analyse? (Use arrow keys)
 » read
   gaming
   laundry
   study
   sport
? What habit do you want to analyse? study
study has been incremented 32 times
The longest streak for the habit "study" is 31, starting at 2023-07-01 and ending at 2023-07-31
```

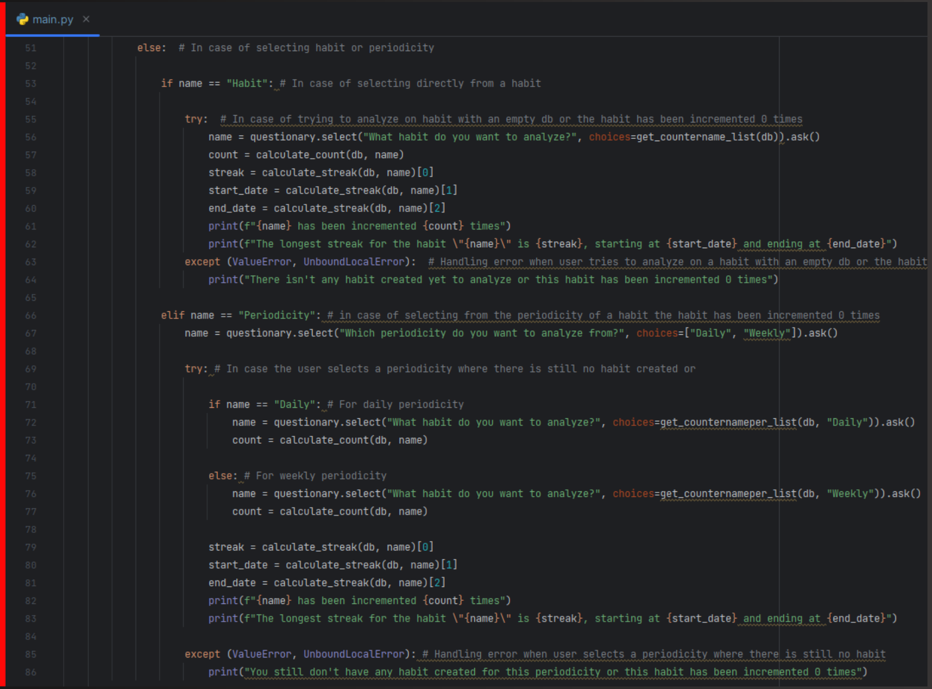
Periodicity command:

When selected, the user will be asked to pick one from the two periodicities (daily or weekly). Upon selecting the desired periodicity, only the habits with that peridicity will be shown. After that, the user selects a habit and the same information as before will be shown, icrementation times and the longest streak.



```
? What do you want to do? Analyse
? Do you want to analyse by habit, periodicity or check the longest streak? Periodicity
? Which periodicity do you want to analyse from? (Use arrow keys)
   Daily
 » Weekly
? Which periodicity do you want to analyse from? Weekly
? What habit do you want to analyse? (Use arrow keys)
 » sport
   laundry
? What habit do you want to analyse? sport
sport has been incremented 5 times
The longest streak for the habit "sport" is 5, starting at 2023-07-01 and ending at 2023-07-30
```

main code



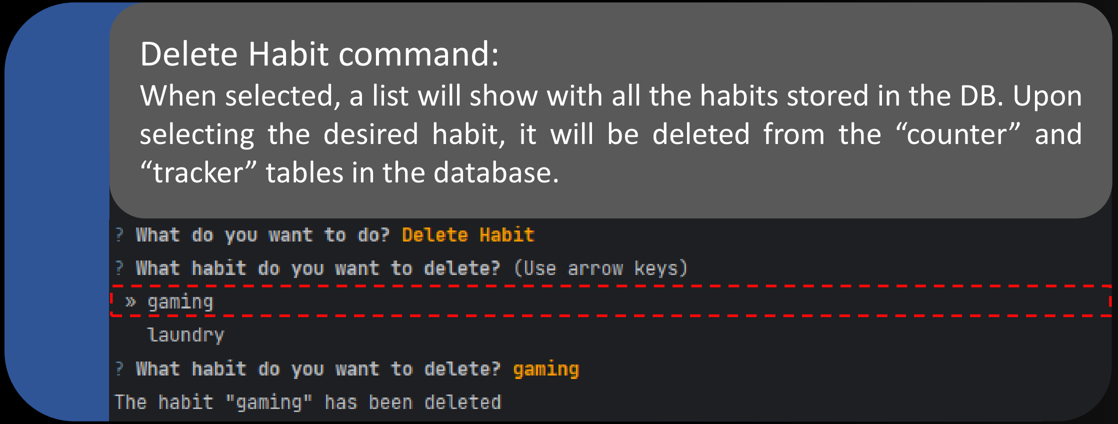
main.py

```python
else:  # In case of selecting habit or periodicity

    if name == "Habit": # In case of selecting directly from a habit

        try:  # In case of trying to analyze on habit with an empty db or the habit has been incremented 0 times
            name = questionary.select("What habit do you want to analyze?", choices=get_countername_list(db)).ask()
            count = calculate_count(db, name)
            streak = calculate_streak(db, name)[0]
            start_date = calculate_streak(db, name)[1]
            end_date = calculate_streak(db, name)[2]
            print(f"{name} has been incremented {count} times")
            print(f"The longest streak for the habit \"{name}\" is {streak}, starting at {start_date} and ending at {end_date}")
        except (ValueError, UnboundLocalError):  # Handling error when user tries to analyze on a habit with an empty db or the habit
            print("There isn't any habit created yet to analyze or this habit has been incremented 0 times")

    elif name == "Periodicity": # in case of selecting from the periodicity of a habit the habit has been incremented 0 times
        name = questionary.select("Which periodicity do you want to analyze from?", choices=["Daily", "Weekly"]).ask()

        try: # In case the user selects a periodicity where there is still no habit created or

            if name == "Daily": # For daily periodicity
                name = questionary.select("What habit do you want to analyze?", choices=get_counternameper_list(db, "Daily")).ask()
                count = calculate_count(db, name)

            else: # For weekly periodicity
                name = questionary.select("What habit do you want to analyze?", choices=get_counternameper_list(db, "Weekly")).ask()
                count = calculate_count(db, name)

            streak = calculate_streak(db, name)[0]
            start_date = calculate_streak(db, name)[1]
            end_date = calculate_streak(db, name)[2]
            print(f"{name} has been incremented {count} times")
            print(f"The longest streak for the habit \"{name}\" is {streak}, starting at {start_date} and ending at {end_date}")

        except (ValueError, UnboundLocalError): # Handling error when user selects a periodicity where there is still no habit
            print("You still don't have any habit created for this periodicity or this habit has been incremented 0 times")
```

The code has an error handling logic and helps improve the user experience by providing informative feedback when attempting retrieve information about a habit that has been created but was not yet incremented or the database itself doesn't have any habits stored.

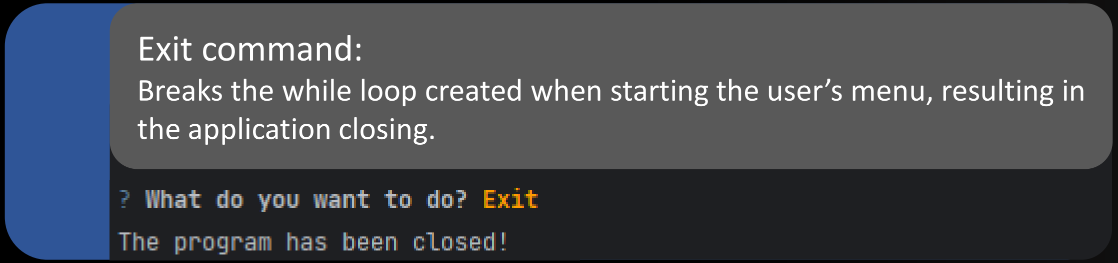
main 5/5 – Delete Habit Command and Exit

main code



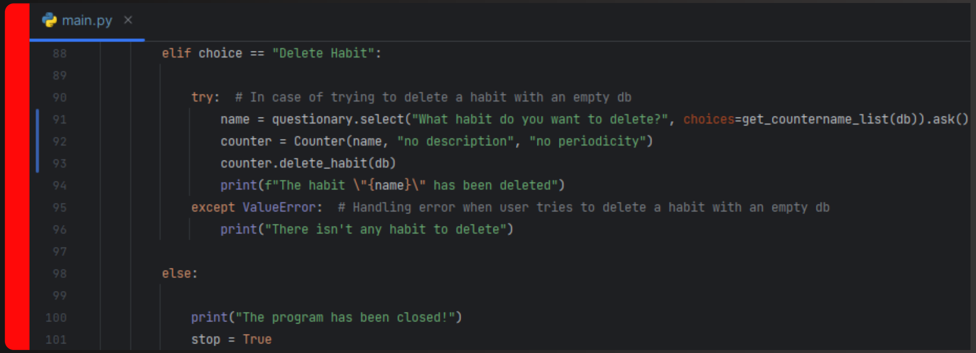
Delete Habit command:
When selected, a list will show with all the habits stored in the DB. Upon selecting the desired habit, it will be deleted from the “counter” and “tracker” tables in the database.

```
? What do you want to do? Delete Habit
? What habit do you want to delete? (Use arrow keys)
 » gaming
   laundry
? What habit do you want to delete? gaming
The habit "gaming" has been deleted
```



Exit command:
Breaks the while loop created when starting the user’s menu, resulting in the application closing.

```
? What do you want to do? Exit
The program has been closed!
```



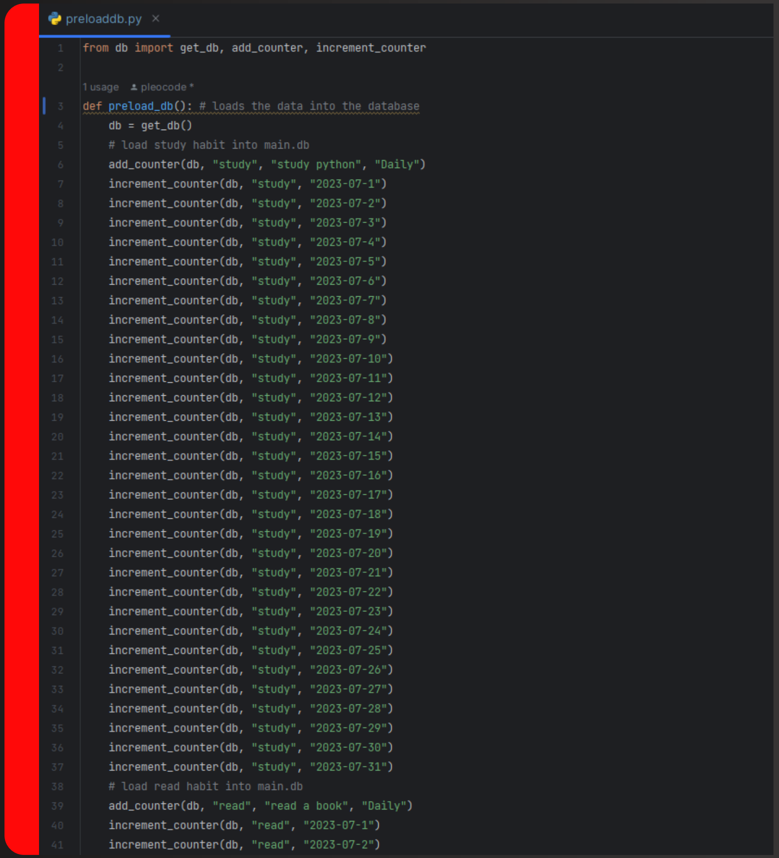
```
main.py ×
        elif choice == "Delete Habit":

            try:  # In case of trying to delete a habit with an empty db
                name = questionary.select("What habit do you want to delete?", choices=get_countername_list(db)).ask()
                counter = Counter(name, "no description", "no periodicity")
                counter.delete_habit(db)
                print(f"The habit \"{name}\" has been deleted")
            except ValueError:  # Handling error when user tries to delete a habit with an empty db
                print("There isn't any habit to delete")

        else:

            print("The program has been closed!")
            stop = True
```

The code has an error handling logic and helps improve the user experience by providing informative feedback when attempting retrieve the habit list (to select a habit to delete from the DB) with an empty database.

preloaddb

preloaddb code

preload_db function:
Adds 5 habits to the database, “study”, “read”, “gaming”, “sport”, “laundry” each with its own set of incrementations.

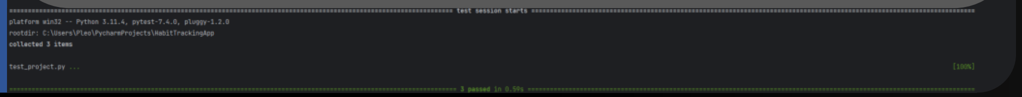

preloaddb.py

```python
from db import get_db, add_counter, increment_counter

1 usage  pleocode *
def preload_db(): # loads the data into the database
    db = get_db()
    # load study habit into main.db
    add_counter(db, "study", "study python", "Daily")
    increment_counter(db, "study", "2023-07-1")
    increment_counter(db, "study", "2023-07-2")
    increment_counter(db, "study", "2023-07-3")
    increment_counter(db, "study", "2023-07-4")
    increment_counter(db, "study", "2023-07-5")
    increment_counter(db, "study", "2023-07-6")
    increment_counter(db, "study", "2023-07-7")
    increment_counter(db, "study", "2023-07-8")
    increment_counter(db, "study", "2023-07-9")
    increment_counter(db, "study", "2023-07-10")
    increment_counter(db, "study", "2023-07-11")
    increment_counter(db, "study", "2023-07-12")
    increment_counter(db, "study", "2023-07-13")
    increment_counter(db, "study", "2023-07-14")
    increment_counter(db, "study", "2023-07-15")
    increment_counter(db, "study", "2023-07-16")
    increment_counter(db, "study", "2023-07-17")
    increment_counter(db, "study", "2023-07-18")
    increment_counter(db, "study", "2023-07-19")
    increment_counter(db, "study", "2023-07-20")
    increment_counter(db, "study", "2023-07-21")
    increment_counter(db, "study", "2023-07-22")
    increment_counter(db, "study", "2023-07-23")
    increment_counter(db, "study", "2023-07-24")
    increment_counter(db, "study", "2023-07-25")
    increment_counter(db, "study", "2023-07-26")
    increment_counter(db, "study", "2023-07-27")
    increment_counter(db, "study", "2023-07-28")
    increment_counter(db, "study", "2023-07-29")
    increment_counter(db, "study", "2023-07-30")
    increment_counter(db, "study", "2023-07-31")
    # load read habit into main.db
    add_counter(db, "read", "read a book", "Daily")
    increment_counter(db, "read", "2023-07-1")
    increment_counter(db, "read", "2023-07-2")
```

test_project

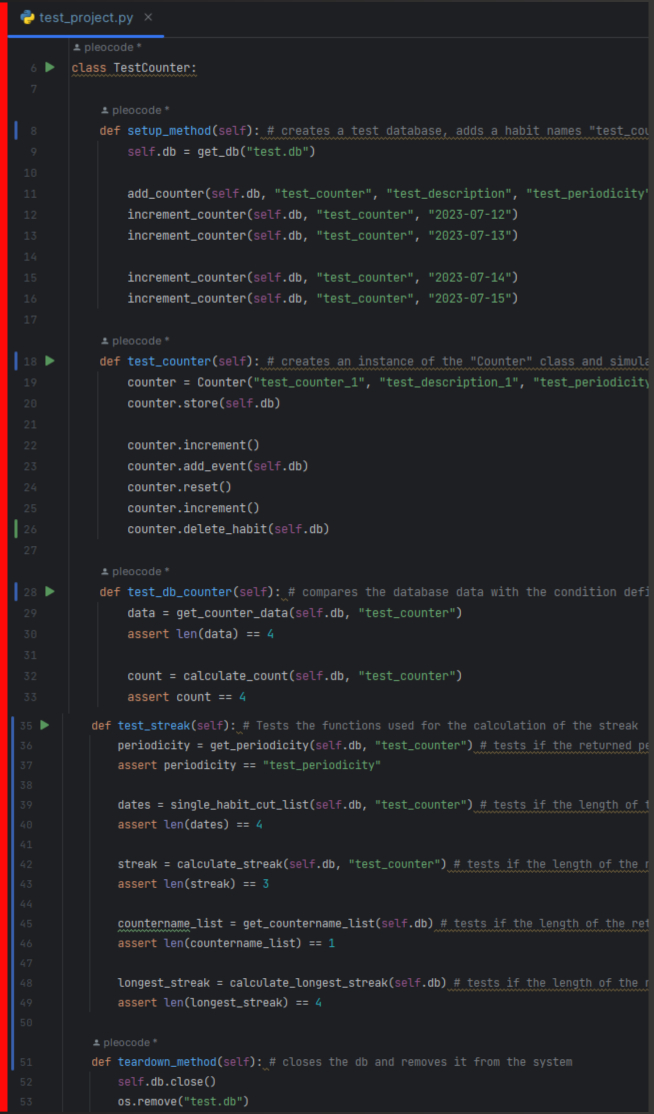
On the test environement, we are testing the following functionalities:

- Create tables in the “test.db”;
- Create a counter;
- Increment a counter;
- Delete a counter;
- Retrieving information from the DB (testing if the result corresponds the expected result);
- Testing the functionality and the expected results for the calculations of “streak” functions.



```
================================ test session starts ================================
platform win32 -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: C:\Users\Pleo\PycharmProjects\HabitTrackingApp
collected 3 items

test_project.py ...                                                          [100%]

================================ 3 passed in 0.59s ================================
```

test_project code



test_project.py

```
pleocode *
class TestCounter:

    pleocode *
    def setup_method(self): # creates a test database, adds a habit names "test_cou
        self.db = get_db("test.db")

        add_counter(self.db, "test_counter", "test_description", "test_periodicity'
        increment_counter(self.db, "test_counter", "2023-07-12")
        increment_counter(self.db, "test_counter", "2023-07-13")

        increment_counter(self.db, "test_counter", "2023-07-14")
        increment_counter(self.db, "test_counter", "2023-07-15")

    pleocode *
    def test_counter(self): # creates an instance of the "Counter" class and simula
        counter = Counter("test_counter_1", "test_description_1", "test_periodicity
        counter.store(self.db)

        counter.increment()
        counter.add_event(self.db)
        counter.reset()
        counter.increment()
        counter.delete_habit(self.db)

    pleocode *
    def test_db_counter(self): # compares the database data with the condition defi
        data = get_counter_data(self.db, "test_counter")
        assert len(data) == 4

        count = calculate_count(self.db, "test_counter")
        assert count == 4
    def test_streak(self): # Tests the functions used for the calculation of the streak
        periodicity = get_periodicity(self.db, "test_counter") # tests if the returned pe
        assert periodicity == "test_periodicity"

        dates = single_habit_cut_list(self.db, "test_counter") # tests if the length of t
        assert len(dates) == 4

        streak = calculate_streak(self.db, "test_counter") # tests if the length of the r
        assert len(streak) == 3

        countername_list = get_countername_list(self.db) # tests if the length of the ret
        assert len(countername_list) == 1

        longest_streak = calculate_longest_streak(self.db) # tests if the length of the r
        assert len(longest_streak) == 4

    pleocode *
    def teardown_method(self): # closes the db and removes it from the system
        self.db.close()
        os.remove("test.db")
```