

# Object Oriented and functional programming with python

---

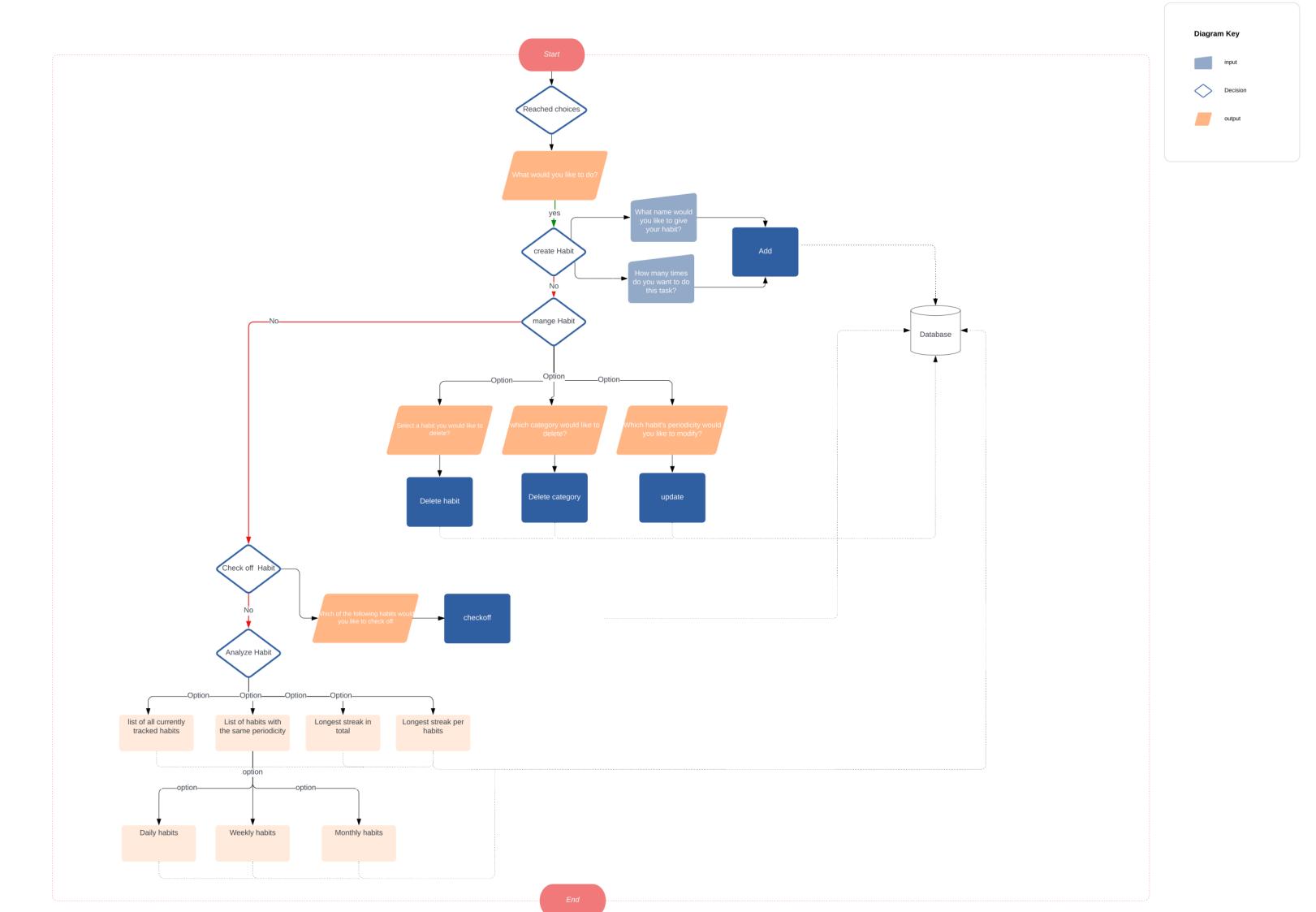
[https://github.com/Oumaymabamoh/Habit\\_Tracker\\_Backend](https://github.com/Oumaymabamoh/Habit_Tracker_Backend)

Habit tracker backend

3 June 2023

## The habit tracker app flowchart:

1. The program started
2. Ask the user to
  - o create habit
  - Manage habit
  - checkoff habit
  - analyze habit
3. The program performs actions based on the user's choices.
4. The program will perform actions until the user wants to end the operation.
5. User can exit the program at anytime



# Main class

- `db = connect()`: This function establishes a connection to the database. It is responsible for connecting the application to a data storage system where habit-related data is stored.
  - `stop = False`: This variable is used as a flag to control the main loop of the application. It determines whether the application should continue running or exit based on user input.
  - `cli()`: This function represents the command-line interface (CLI) of the application. It serves as the main interaction point with the user, displaying options and handling user input to perform various actions.
  - `Exit`: This option within the CLI allows the user to exit the application. It terminates the program and displays a farewell message.
  - `sys.exit()`: This function from the `sys` module is called when the user selects the "Exit" option. It terminates the program execution and exits the application.

The screenshot shows the PyCharm IDE interface with the 'HabitTracker' project open. The left sidebar displays the project structure with files like main.py, database.py, habit.py, get.py, test.py, and testanalytics.py. The main editor window shows the content of main.py, which contains Python code for a command-line habit tracker application. The code includes imports for Habit, analytics, sys, database, and PyQt5 modules. It defines a cli() function that connects to a database, loops until a user exits, and handles five main choices: creating a habit, managing a habit, checking off a habit, viewing analytics, or exiting. Each choice leads to further sub-interactions, such as prompting for habit name, category, and periodicity.

```
from habit import Habit
from analytics import habits_list, habits_by_period
import analytics
import sys
from database import *
import questionnaire as qt
import get

# welcome message
print("Welcome to the {}!".format("Habit Tracker app"))

# Command-line interface
def cli():
    """
    using the command line interface in order to interact with the user.
    """

    # Connect to the database by calling the connect function
    db = connect()

    stop = False
    while not stop:
        choice = qt.select("What would you like to do?", [
            "1. Create habit",
            "2. Manage habit",
            "3. Check off habit",
            "4. Show my analytics",
            "5. Exit"
        ]).ask() # Prompt the user for a choice of action from a list

        if choice == "1. Create habit":
            habit_name = qt.text("What name would you like to give your habit?").ask() # Prompt for habit name
            habit_category = qt.text("What category would you like to give to your habit?").ask() # Prompt for habit category
            habit_periodicity = qt.select("How many times do you want to do this habit?", [
                "Daily", "Weekly", "Monthly"
            ]).ask().lower() # Prompt for habit periodicity
            habit = Habit(habit_name, habit_category, habit_periodicity) # Create an instance of the Habit class
            habit.add() # Add the habit to the database

        elif choice == "2. Manage habit":
            print("What would you like to do?")
            while True:
                print("1. Delete habit")
                print("2. Delete category")
                print("3. Modify periodicity")
                print("4. Go Back to the Menu")

                sub_choice = input("Enter your choice (1-4): ") # Prompt for a sub-choice
```

```
... database.py habit.py analytics.py get.py test.py testanalytics.py

HabitTracker ~/Pycharm
venv
analytics.py
database.py
get.py
habit.py
habits.db
main.py
test.py
testanalytics.py
External Libraries
Scratches and Consoles

main.py
110     "4. The longest streak you've had for a specific habit.",
111 ]
112     analytics_choice = qt.select("Which analytics do you want to see?", choices=options).ask() # Prompt for an analytics choice
113
114     if analytics_choice == options[0]:
115         habits_list(db) # Call the habits.list() function to display all current habits
116     elif analytics_choice == options[1]:
117         time_period = qt.select("View",
118             choices=[
119                 "Daily Habits",
120                 "Weekly Habits",
121                 "Monthly Habits",
122                 "Back to Main Menu"
123             ]).ask() # Prompt for a time period choice
124
125     if time_period == "Daily Habits":
126         habits_by_period(db, "daily") # Call the habits_by_period() function with "daily" as the period
127     elif time_period == "Weekly Habits":
128         habits_by_period(db, "weekly") # Call the habits_by_period() function with "weekly" as the period
129     elif time_period == "Monthly Habits":
130         habits_by_period(db, "monthly") # Call the habits_by_period() function with "monthly" as the period
131
132     elif analytics_choice == options[2]:
133         longest_streak_all_habits = analytics.longest_streak_all_habits(db) # Get the longest streak for all habits
134         print(f"\nLongest streak you have out of all your current habits is {longest_streak_all_habits}.")
135
136     elif analytics_choice == options[3]:
137         habit_name = get.habit_from_db(db) # Prompt for a habit name
138         if habit_name is not None:
139             longest_streak = analytics.habit_longest_streak(db, habit_name) # Get the longest streak for the specified habit
140             if longest_streak is not None:
141                 print(f"\nThe longest streak for habit {habit_name} is {longest_streak}.")
142             else:
143                 print(f"No streaks found for habit {habit_name}.")
144         else:
145             print("No habits found in the database.")
146
147     elif choice == "5. Exit":
148         print("\nYou have successfully exited the program. Goodbye!")
149         sys.exit() # Exit the program
150
151
152 stop = True
153
154 if __name__ == '__main__':
155     cli() # Call the cli() function if the script is executed directly
```

## Database Class

- SQLite3 is used as the database system.
- `connect(name="habits.db")`: Establishes a connection to the SQLite database. It returns a database connection object that allows for data storage and retrieval. The optional parameter name specifies the name of the database file.
- `create_tables(db)`: Creates the necessary tables in the database if they don't already exist. It uses a cursor object to execute SQL statements that create the "habit" and "habit\_checks" tables. The tables are created using the CREATE TABLE IF NOT EXISTS statement, ensuring they are only created if they don't exist. The changes are committed using `db.commit()`.
- `db.cursor()`: Creates a cursor object associated with the database connection. The cursor object allows for executing SQL statements and fetching data from the database.
- `c.execute(sql_statement, parameters)`: Executes an SQL statement using the cursor object. The `sql_statement` parameter contains the SQL statement to be executed, and the `parameters` parameter (optional) contains the values to be substituted in the SQL statement.
- `db.commit()`: Commits the changes made to the database. After executing SQL statements that modify the database, the `commit()` method is called to save those changes permanently.

```

import sqlite3
from datetime import datetime

def connect(name="habits.db"):
    db = sqlite3.connect(name)
    create_tables(db)
    return db

def create_tables():
    c = db.cursor()

    c.execute("""CREATE TABLE IF NOT EXISTS habit (
        name TEXT PRIMARY KEY,
        category TEXT,
        periodicity TEXT,
        creation_time TEXT,
        completion_time TEXT,
        streak INT
    )""")

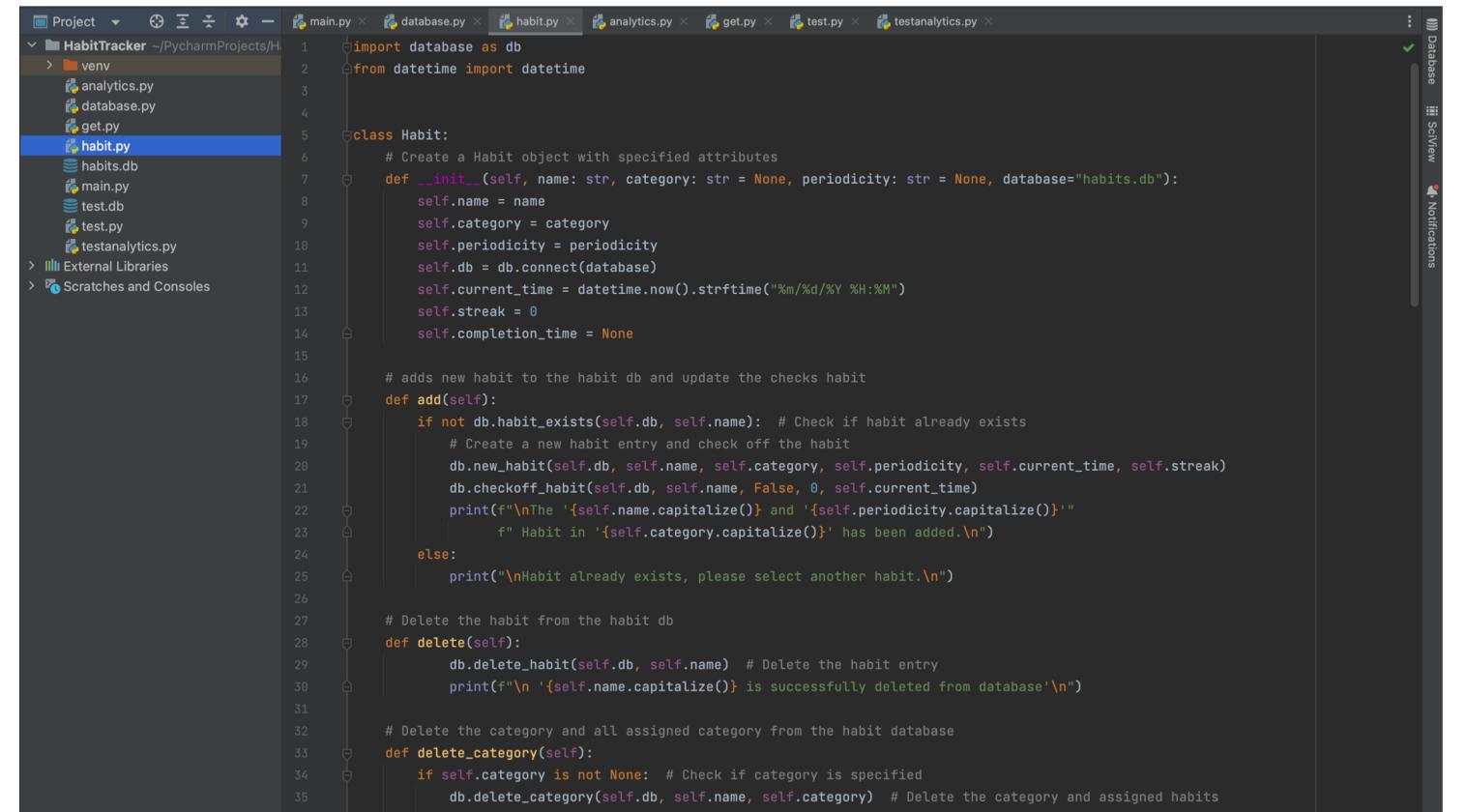
    c.execute("""CREATE TABLE IF NOT EXISTS habit_checks(
        name TEXT,
        checkoff INT,
        streak INT DEFAULT 0,
        completion_time TEXT,
        FOREIGN KEY (name) REFERENCES habit(name)
    )""")

    db.commit()

```

## Habit class

- Classes provide modular organization and code reusability.
- They encapsulate data and behavior, promoting abstraction and code maintainability.
- The Habit class encapsulates the attributes and behaviors related to
  - Adding habits
  - managing habits
  - tracking habit progress
- It interacts with the database module to perform database operations and provides an interface for manipulating and tracking individual habits.



The screenshot shows the PyCharm IDE interface with the 'habit.py' file selected in the project tree. The code defines a 'Habit' class with methods for adding, deleting, and deleting categories from a database. The code uses Python's built-in datetime module for timestamping and the database module for interacting with a SQLite database named 'habits.db'.

```

import database as db
from datetime import datetime

class Habit:
    # Create a Habit object with specified attributes
    def __init__(self, name: str, category: str = None, periodicity: str = None, database="habits.db"):
        self.name = name
        self.category = category
        self.periodicity = periodicity
        self.db = db.connect(database)
        self.current_time = datetime.now().strftime("%m/%d/%Y %H:%M")
        self.streak = 0
        self.completion_time = None

    # adds new habit to the habit db and update the checks habit
    def add(self):
        if not db.habit_exists(self.db, self.name): # Check if habit already exists
            # Create a new habit entry and check off the habit
            db.new_habit(self.db, self.name, self.category, self.periodicity, self.current_time, self.streak)
            db.checkoff_habit(self.db, self.name, False, 0, self.current_time)
            print(f"\nThe '{self.name.capitalize()}' and '{self.periodicity.capitalize()}'"
                  f" Habit in '{self.category.capitalize()}' has been added.\n")
        else:
            print("\nHabit already exists, please select another habit.\n")

    # Delete the habit from the habit db
    def delete(self):
        db.delete_habit(self.db, self.name) # Delete the habit entry
        print(f"\n'{self.name.capitalize()}' is successfully deleted from database!\n")

    # Delete the category and all assigned category from the habit database
    def delete_category(self):
        if self.category is not None: # Check if category is specified
            db.delete_category(self.db, self.name, self.category) # Delete the category and assigned habits

```

Habit tracker app |

## Create habit

- Program start asking user to enter
  - The user is prompted to enter the habit name.
  - The user is prompted to enter the habit category.
  - The user is prompted to choose the habit's periodicity.
- User manages the program by answering questions in the Terminal
  - Questions in white
  - Answers in orange
- Once the entries are provided by user:
  - a successful message shows up and
  - the data is automatically saved in the database

The screenshot shows the PyCharm IDE interface with the main.py file open. The code implements a menu system with options for creating, managing, checking off, and viewing analytics for habits. It uses the `sqlite3` module to interact with a database named 'habits.db'. The terminal below shows the execution of the program, where the user creates a habit named 'cooking' under the 'health' category with a daily periodicity. A success message indicates the habit has been added to the database.

```
# Connect to the database by calling the connect function
db = connect()

stop = False
while not stop:
    choice = qt.select("What would you like to do?", choices=[1, 2, 3, 4, 5])
    if choice == "1. Create habit":
        habit_name = qt.text("What name would you like to give your habit?").ask() # Prompt for habit name
        habit_category = qt.text("What category would you like to give to your habit?").ask() # Prompt for habit category
        habit_periodicity = qt.select("How many times do you want to do this task", choices=["Daily", "weekly", "Monthly"]).ask().lower() # Prompt for habit periodicity
        habit = Habit(habit_name, habit_category, habit_periodicity) # Create an instance of the Habit class
        habit.add() # Add the habit to the database
```

The screenshot shows the PyCharm IDE interface with the database.py file open, displaying the contents of the 'habits' table in the database. The table has columns: name, category, periodicity, creation\_time, completion\_time, and streak. Two rows are present: one for 'tennis' (sport, monthly) and one for 'cooking' (health, daily). The terminal below shows the same interaction as the first screenshot, confirming the addition of the 'cooking' habit.

	name	category	periodicity	creation_time	completion_time	streak
1	tennis	sport	monthly	05/24/2023 18:36	0	0
2	cooking	health	daily	05/26/2023 13:38	0	0

```
? What would you like to do? 1. Create habit
? What name would you like to give your habit? cooking
? What category would you like to give to your habit: health
? How many times do you want to do this task Daily

The 'Cooking and 'Daily' Habit in 'Health' has been added.

? What would you like to do? (Use arrow keys)
```

# Manage habit

- Deleting a Habit:
    - Prompt the user to select the habit they want to delete.
    - Create a Habit object for the selected habit and call the `delete()` method to delete it from the database.
    - Inform the user if the habit was successfully deleted or if any error occurred.
  - Deleting a Category:
    - Prompt the user to select the category they want to delete.
    - Create a Habit object with the selected category and call the `delete_category()` method to delete it from the database.
    - Inform the user if the category was successfully deleted or if any error occurred.
  - Modifying Periodicity:
    - Prompt the user to select the habit they want to modify the periodicity for.
    - Display the current habit name and prompt the user to enter the new periodicity (Daily, Weekly, or Monthly).
    - Confirm with the user if they want to proceed with the change.
    - Create a Habit object with the selected habit and new periodicity, and call the `update_periodicity()` method to update the periodicity in the database.
    - Inform the user if the periodicity was successfully changed or if any error occurred.

The screenshot shows the PyCharm IDE interface with the 'HabitTracker' project open. The left sidebar displays the project structure with files like analytics.py, database.py, habit.py, habits.db, main.py, test.py, and testanalytics.py. The main editor window shows the content of main.py, which handles user interaction for managing habits and categories. The status bar at the bottom indicates the file is saved.

```
elif choice == "2. Manage habit":
    print("What would you like to do?")
    while True:
        print("1. Delete habit")
        print("2. Delete category")
        print("3. Modify periodicity")
        print("4. Go Back to the Menu")

        sub_choice = input("Enter your choice (1-4): ") # Prompt for a sub-choice

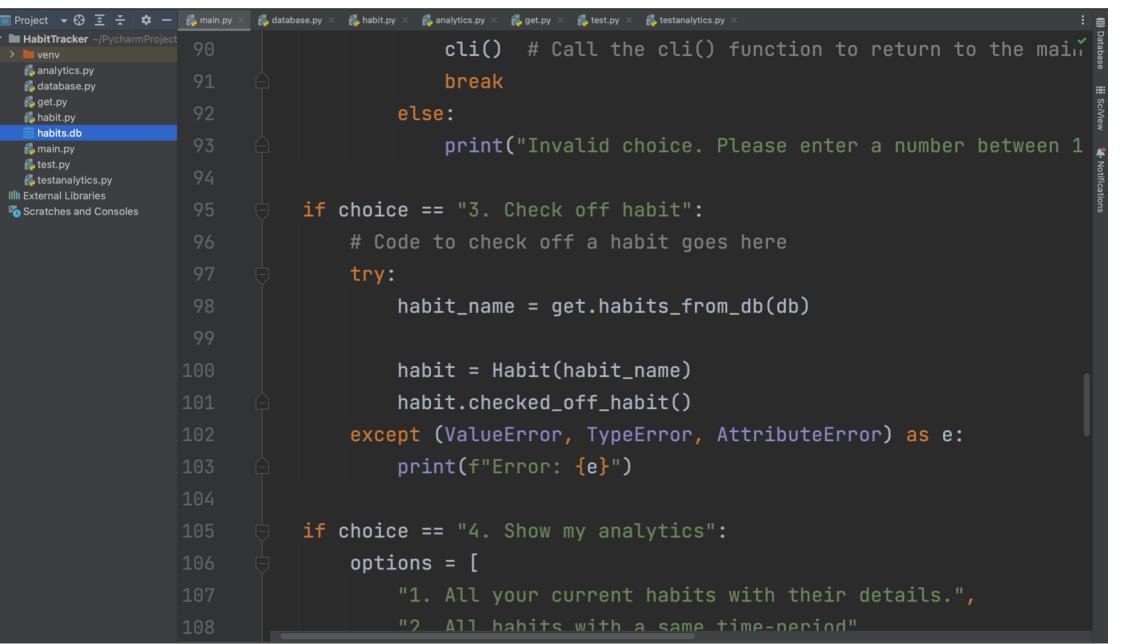
        if sub_choice == "1":
            try:
                habit_name = get.habits_from_db(db)
                habit = Habit(habit_name)
                habit.delete()
                # Code to delete the habit goes here
            except ValueError:
                print("habit not deleted ")
        elif sub_choice == "2":
            try:
                habit_category = get.defined_categories()
                category_to_delete = Habit(db, habit_category)
                category_to_delete.delete_category()
                # Code to delete the category goes here
                print(f"The category '{habit_category}' has been deleted.")
            except ValueError:
                print("category not deleted ")

        elif sub_choice == "3":
            habit_name = get.habits_from_db(db)
            # Code to modify periodicity goes here
            if not habit_name: # Check if database is empty
                print("\nDatabase empty; Please add a habit first.\n")
            else:
                print(f"Selected habit name: {habit_name}")
                new_periodicity = qt.select("What would you like to change the periodicity to?", choices=["Daily", "Weekly", "Monthly"]).ask()
                if qt.confirm(
                    f"Are you sure you want to change the periodicity of {habit_name} to {new_periodicity}!").ask():
                    habit = Habit(db, habit_name)
                    habit.update_periodicity()
                    print(f"\nPeriodicity of {habit_name} changed to {new_periodicity}!\n")
                else:
                    print(f"\nPeriodicity of {habit_name} remains unchanged! Thanks for confirming.\n")

```

## Check off habit

- The streak by 1 when a habit is checked off and resetting the streak to 0 when it is not complete
- The checked\_off\_habit method is called, which checks if the habit exists and retrieves its periodicity.
- Based on the periodicity, it checks if the habit has already been completed within the specified time period (daily, weekly, or monthly).
- If the habit has not been completed, the update\_streak method is called to increment the streak count by 1 and update the streak value in the database.
- The checkoff\_habit method is called to update the completion status, streak count, and completion time in the database.
- If the habit has already been completed within the time period, the reset method is called to reset the streak count to 0.



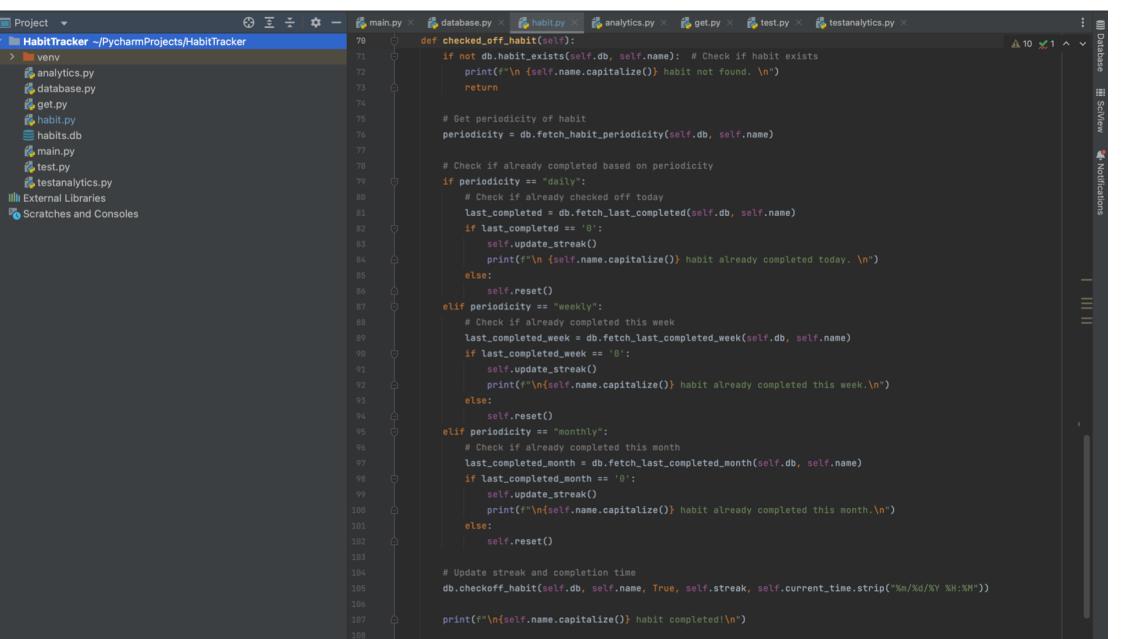
```

    cli() # Call the cli() function to return to the main menu
    break
else:
    print("Invalid choice. Please enter a number between 1 and 4")

if choice == "3. Check off habit":
    # Code to check off a habit goes here
    try:
        habit_name = get.habits_from_db(db)
        habit = Habit(habit_name)
        habit.checked_off_habit()
    except (ValueError, TypeError, AttributeError) as e:
        print(f"Error: {e}")

if choice == "4. Show my analytics":
    options = [
        "1. All your current habits with their details.",
        "2. All habits with a same time-period"
    ]

```



```

def checked_off_habit(self):
    if not db.habit_exists(self.db, self.name): # Check if habit exists
        print(f"\n{self.name.capitalize()} habit not found.\n")
        return

    # Get periodicity of habit
    periodicity = db.fetch_habit_periodicity(self.db, self.name)

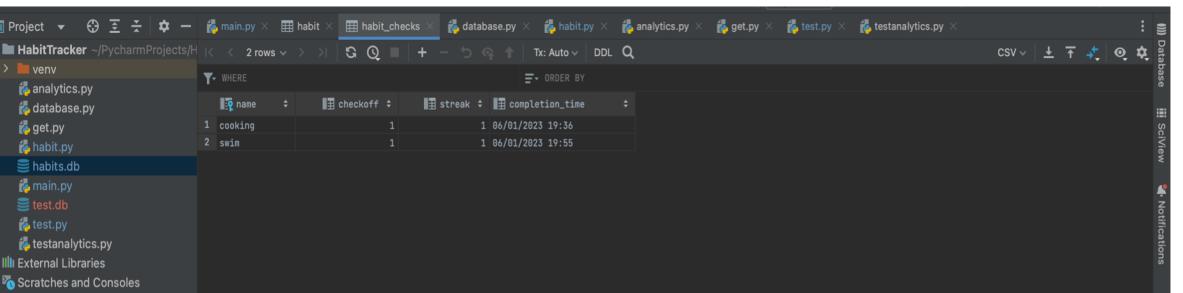
    # Check if already completed based on periodicity
    if periodicity == "daily":
        # Check if already checked off today
        last_completed = db.fetch_last_completed(self.db, self.name)
        if last_completed == '':
            self.update_streak()
            print(f"\n{self.name.capitalize()} habit already completed today.\n")
        else:
            self.reset()

    elif periodicity == "weekly":
        # Check if already completed this week
        last_completed_week = db.fetch_last_completed_week(self.db, self.name)
        if last_completed_week == '':
            self.update_streak()
            print(f"\n{self.name.capitalize()} habit already completed this week.\n")
        else:
            self.reset()

    elif periodicity == "monthly":
        # Check if already completed this month
        last_completed_month = db.fetch_last_completed_month(self.db, self.name)
        if last_completed_month == '':
            self.update_streak()
            print(f"\n{self.name.capitalize()} habit already completed this month.\n")
        else:
            self.reset()

    # Update streak and completion time
    db.checkoff_habit(self.db, self.name, True, self.streak, self.current_time.strftime("%d/%m %H:%M"))
    print(f"\n{self.name.capitalize()} habit completed!\n")

```

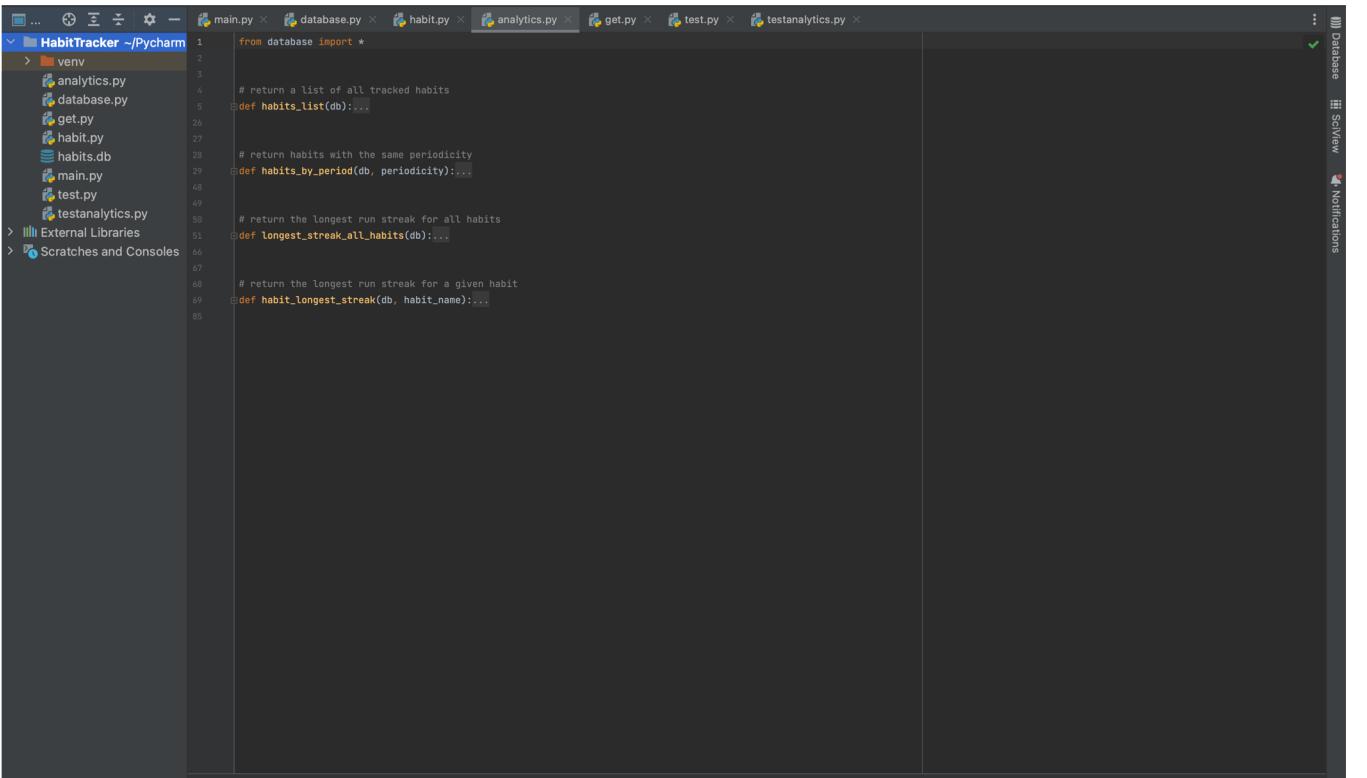


name	checkoff	streak	completion_time
cooking	1	1	06/01/2023 19:36
swim	1	1	06/01/2023 19:55

## Habit analytics

The analysis module contains a concise overview of notable habit-related information:

- habits\_list(db):
  - Retrieves all tracked habits from the database and displays them in a table format.
  - The table includes columns for the habit name, category, periodicity, and completion date/time.
- habits\_by\_period(db, periodicity):
  - Retrieves all habits with a specified periodicity from the database.
  - Displays the habits in a table format, including columns for the habit name, periodicity, streak, and completion time.
- longest\_streak\_all\_habits():
  - Calculates and returns the longest streak among all habits.
  - Iterates over all habits in the database and finds the maximum streak value.
- habit\_longest\_streak(db, habit\_name):
  - Retrieves the longest streak for a given habit.
  - Executes a SQL query to select the maximum streak value from the habit\_checks table for the specified habit name.
  - Returns the longest streak value, or 0 if no streak is found or an error occurs.



A screenshot of the PyCharm IDE interface. The left sidebar shows a project structure with a 'venv' folder containing 'analytics.py', 'database.py', 'get.py', 'habit.py', 'habits.db', 'main.py', 'test.py', and 'testanalytics.py'. Below this are 'External Libraries' and 'Scratches and Consoles'. The main editor window displays the 'analytics.py' file with the following code:

```
from database import *

# return a list of all tracked habits
def habits_list(db):...
# return habits with the same periodicity
def habits_by_period(db, periodicity):...
# return the longest run streak for all habits
def longest_streak_all_habits(db):...
# return the longest run streak for a given habit
def habit_longest_streak(db, habit_name):...
```

**Thank you**