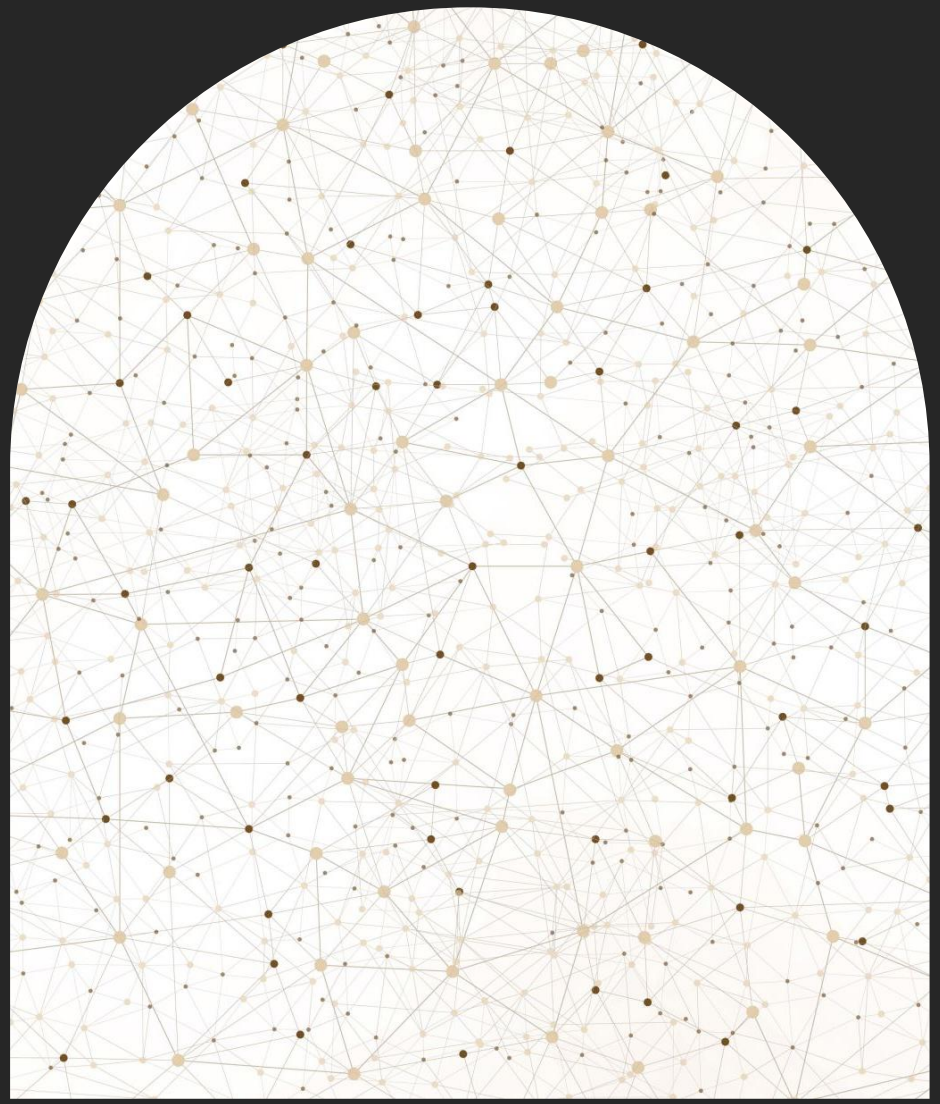
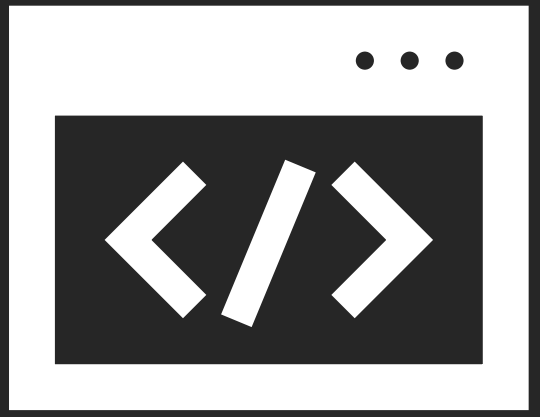
Object Oriented and Functional Programming with Python (Habit Tracker)



CREATED BY:FAHRİ EFE YANİKLAR

Introduction



This application must be a good application for those who want to be planned and methodical. In this application, everything must be easily accessible, user-friendly, and not complicated. Application designed and created for this. This presentation explains the technical background and development stages.

```python
import sys
import sqlite3
from PySide6.QtWidgets import (
    QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel,
    QLineEdit, QPushButton, QMessageBox, QTextEdit
)

def init_db():  1 usage
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS monthly_habit_tracker
                 ( id INTEGER PRIMARY KEY AUTOINCREMENT,
                  week INTEGER NOT NULL,
                  date_str TEXT NOT NULL,
                  healthy_eating TEXT NOT NULL,
                  daily_exercise TEXT NOT NULL,
                  no_smoke TEXT NOT NULL,
                  time_outdoors TEXT NOT NULL,
                  blogging TEXT NOT NULL
                  )''')
    conn.commit()
    conn.close()
```

1.Import Process and Defining Functions

In this section firstly we have to import tools for our application and we need to create table for user can fill it. In import section we need to import sys(for python interact with system), sqlite3(for create and type query) and PySide6.Qtwidgets (for GUI (Graphical User Interface)). In the def_init function, we are creating a table with sqlite3 for users can fill in his/her information.

```python
def add_or_update_table(week, date_str, healthy_eating, daily_exercise, no_smoke, time_outdoors, blogging):  1 usage
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    c.execute( sql: '''SELECT id FROM monthly_habit_tracker
                  WHERE week = ?
                  AND date_str = ?''',
               parameters: (week, date_str))
    result = c.fetchone()

    if result:
        c.execute( sql: '''UPDATE monthly_habit_tracker
                      SET healthy_eating = ?,
                          daily_exercise = ?,
                          no_smoke = ?,
                          time_outdoors = ?,
                          blogging = ?
                          WHERE id = ?''',
                   parameters: (healthy_eating, daily_exercise, no_smoke, time_outdoors, blogging, result[0]))
        print(f"Updated habits on {date_str} in {week}.")
    else:
        c.execute( sql: '''INSERT INTO monthly_habit_tracker(week,date_str,healthy_eating,daily_exercise,no_smoke,time_outdoors,blogging)
                      VALUES(?,?,?,?,?,?,?)''',
                   parameters: (week, date_str, healthy_eating, daily_exercise, no_smoke, time_outdoors, blogging))
        print(f"Added habits on {date_str} in {week}.")
    conn.commit()
    conn.close()
```

> Next, defined the add_or_update_table function this function is used for adding a row or updating a row in the table. Then defined get_record and delete_row functions, get_record is used for see which you want to record, and delete_row function is used for deleting which you want row.

```python
def get_record(week: int, date_str: str):  1 usage
    try:
        week = int(week)
    except ValueError:
        return None
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    c.execute( sql: 'SELECT * FROM monthly_habit_tracker WHERE week =
    row = c.fetchone()
    conn.close()
    return row

def delete_row(week, date_str):  1 usage
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    c.execute( sql: '''SELECT * FROM monthly_habit_tracker
                  WHERE week = ? AND date_str = ?''',
               parameters: (week, date_str))
    row = c.fetchone()
    if row:
        c.execute( sql: '''DELETE FROM monthly_habit_tracker
                      WHERE week = ? AND date_str = ?''',
                   parameters: (week, date_str))
        conn.commit()
        print(f'Deleted habits on {date_str} in {week}.')
    else:
        print(f"No habits found for {date_str} in {week}.")
    conn.close()
```

```python
def best_streak():  1 usage
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    c.execute('''
        SELECT healthy_eating, daily_exercise, no_smoke, time_outdoors, blogging
        FROM monthly_habit_tracker
        ORDER BY id ASC
    ''')
    rows = c.fetchall()
    conn.close()
    streak = 0
    for row in rows:
        if all(col == "checked-off" for col in row):
            streak += 1
        else:
            break
    print(f"You have a {streak} day streak!")
    return streak

def best_habit_streak():  1 usage
    conn = sqlite3.connect('monthly_habit_tracker.db')
    c = conn.cursor()
    habits = ["healthy_eating", "daily_exercise", "no_smoke", "time_outdoor
    results = {}
    for habit in habits:
        c.execute(f'''SELECT COUNT(*) FROM monthly_habit_tracker
                      WHERE {habit} = "checked-off"''')
        count = c.fetchone()[0]
        results[habit] = count
    conn.close()
    if results:
        best_habit = max(results, key=results.get)
        best_count = results[best_habit]
        return best_habit, best_count
    else:
        return None
```

Next, defined best_streak and best_habit_streak. In this section, best_streak is used to see the perfect days (every habit checked-off), and best_habit_streak is used to see the most checked-off habit.

```python
class HabitTrackerApp(QWidget):  1 usage
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Habit Tracker")
        self.setGeometry(200, 200, 600, 400)

        init_db()

        layout = QVBoxLayout()

        week_date_layout = QHBoxLayout()
        self.week_input = QLineEdit()
        self.week_input.setPlaceholderText("Week number")
        self.date_input = QLineEdit()
        self.date_input.setPlaceholderText("Date (e.g. 06.06.2025)")
        week_date_layout.addWidget(QLabel("Week:"))
        week_date_layout.addWidget(self.week_input)
        week_date_layout.addWidget(QLabel("Date:"))
        week_date_layout.addWidget(self.date_input)
        layout.addLayout(week_date_layout)
```

2.Creating GUI (Graphical User Interface)

In this section, we defined a class for GUI (Graphical User Interface). Firstly, created a GUI tab and week and date input space.

```
        self.habit_inputs = {}
        habits = ["healthy_eating", "daily_exercise", "no_smoke", "time_outdoors", "blogging"]
        for habit in habits:
            h_layout = QHBoxLayout()
            label = QLabel(habit.replace( __old: "_",  __new: " ").title() + ":")
            edit = QLineEdit()
            edit.setPlaceholderText("checked-off / not checked-off")
            h_layout.addWidget(label)
            h_layout.addWidget(edit)
            self.habit_inputs[habit] = edit
            layout.addLayout(h_layout)

        btn_layout = QHBoxLayout()
        self.view_btn = QPushButton("View Record")
        self.view_btn.clicked.connect(self.view_record)
        self.save_btn = QPushButton("Add/Update Record")
        self.save_btn.clicked.connect(self.save_record)
        self.delete_btn = QPushButton("Delete Record")
        self.delete_btn.clicked.connect(self.delete_record)
        self.streak_btn = QPushButton("Show Streak")
        self.streak_btn.clicked.connect(self.show_streak)
        self.best_habit_btn = QPushButton("Show Best Habit Streak")
        self.best_habit_btn.clicked.connect(self.show_best_habit_streak)
```

```
    def view_record(self):  2 usages
        week = self.week_input.text().strip()
        date = self.date_input.text().strip()
        if not week or not date:
            QMessageBox.warning(self,  title: "Input Error",  text: "Please enter both week and date.")
            return
        record = get_record(week, date)
        if record:
            text = (
                f"Week: {record[1]}, Date: {record[2]}\n"
                f"Healthy Eating: {record[3]}\n"
                f"Daily Exercise: {record[4]}\n"
                f"No Smoke: {record[5]}\n"
                f"Time Outdoors: {record[6]}\n"
                f"Blogging: {record[7]}"
            )
            self.result_display.setPlainText(text)
            for idx, habit in enumerate(["healthy_eating", "daily_exercise", "no_smoke", "time_outdoors", "blogging"], start=3):
                self.habit_inputs[habit].setText(record[idx])
        else:
            self.result_display.setPlainText("No record found for this week and date.")
            for field in self.habit_inputs.values():
                field.clear()
```

```
        btn_layout.addWidget(self.view_btn)
        btn_layout.addWidget(self.save_btn)
        btn_layout.addWidget(self.delete_btn)
        btn_layout.addWidget(self.streak_btn)
        btn_layout.addWidget(self.best_habit_btn)
        layout.addLayout(btn_layout)

        self.result_display = QTextEdit()
        self.result_display.setReadOnly(True)
        layout.addWidget(self.result_display)
```

Next, we created the main form and buttons for interacting with the database and GUI. As you can see created ‘View Record’ button’s a technical background.

```
    def delete_record(self):  1 usage
        week = self.week_input.text().strip()
        date = self.date_input.text().strip()
        if not week or not date:
            QMessageBox.warning(self,  title: "Input Error",  text: "Please enter both w
            return

        confirmed = QMessageBox.question(
            self,  title: "Confirm Delete",
             text: f"Are you sure you want to delete the record for week {week} and d
            QMessageBox.Yes | QMessageBox.No
        )

        if confirmed == QMessageBox.Yes:
            delete_row(week, date)
            self.result_display.setPlainText(f"Record for week {week}, date {date}
            for field in self.habit_inputs.values():
                field.clear()
    def save_record(self):  1 usage
        week = self.week_input.text().strip()
        date = self.date_input.text().strip()
        if not week or not date:
            QMessageBox.warning(self,  title: "Input Error",  text: "Please enter both week a
            return

        values = []
        valid_values = {"checked-off", "not checked-off"}
        for habit in ["healthy_eating", "daily_exercise", "no_smoke", "time_outdoors", '
            val = self.habit_inputs[habit].text().strip()
            if val not in valid_values:
                QMessageBox.warning(self,  title: "Input Error",
                                    text: f"'{habit.replace( _old: '_',  _new: ' ').title()
                return
            values.append(val)

        add_or_update_table(week, date, *values)
        QMessageBox.information(self,  title: "Saved",  text: "Record added/updated successfu
        self.view_record()
```

Then, built the ‘Add/Update Record’ and ‘Delete Record’ technical background.

```
239         def show_streak(self):  1 usage
240             streak = best_streak()
241             self.result_display.setPlainText(f"Current streak (all habits checked-off consecutively): {streak} day(s)")
242
243         def show_best_habit_streak(self):  1 usage
244             result = best_habit_streak()
245             if result:
246                 habit_name, count = result
247                 formatted_name = habit_name.replace("_", " ").title()
248                 self.result_display.setPlainText(
249                     f"Best habit streak is '{formatted_name}' with {count} day(s) checked-off.")
250             else:
251                 self.result_display.setPlainText("No habit streak data found.")
252
253 if __name__ == "__main__":
254     app = QApplication(sys.argv)
255     window = HabitTrackerApp()
256     window.show()
257     sys.exit(app.exec())
```

Finally, ‘Show Streak’, ‘Show Best Habit Streak’ technical infrastructure was created to ensure that the system opens and closes properly.

Thanks for reading my presentation.

CREATED BY: FAHRI EFE YANIKLAR