ALARM CLOCK USING PYTHON AND TKINTER

A Project Report

In partial fulfillment of the requirements for the degree of

Bachelor of Computer Applications (BCA)

Under the guidance

of

LAKHAN MAHTO

And

UGCPL (Unati Global Connect Private Limited)

Excel Beyond Boundaries

Submitted by

Roushan Kumar, Durga Sharma, Anjani Kumari, Priya Kumari

SARALA BIRLA UNVERSITY

Unati Global Connect

Excel Beyond Boundaries

CERTIFICATE

This is to certify that the project entitled:

"ALARM CLOCK USING PYTHON AND TKINTER"

has been successfully completed by:

Roushan Kumar, Durga Sharma, Anjani Kumari, Priya Kumari

under the internship program conducted by **UGCPL (Unati Global Connect Private Limited)** in partial fulfillment of the requirements of the **Bachelor of Computer Applications (BCA), Semester IV** of **SARALA BIRLA UNIVERSITY.**

The project demonstrates the application of core concepts of Python programming in the development of a fully functional desktop application. It reflects the team's ability to analyze a problem, design a solution, implement it using appropriate tools, and test it for real-time use cases. The students have put great effort into developing an interactive alarm system with advanced features like snooze, database storage, repeat alarms, and audio alerts.

We appreciate the hard work, creativity, and team collaboration shown during the development process. The project work stands as a valuable outcome of their internship training.

Date:

Signature of Mentor:

UGCPL Internship Coordinator

ACKNOWLEDGMENT

We take this opportunity to express our deepest gratitude to our project mentor **LAKHAN MAHTO** and **UGCPL (Unati Global Connect Private Limited)** for organizing the internship program on **Python for Industrial Applications** and for giving us the chance to work on this hands-on project.

We are especially thankful to our mentor and guides at UGCPL, whose constant support, expert advice, and motivation kept us on the right track. They helped us understand real-world software development and encouraged us to explore programming beyond just theory.

We also extend our thanks to our college faculty members for building our base knowledge, and to our families and friends for encouraging us throughout this journey.

Special thanks to each of our team members — **Roushan Kumar, Durga Sharma, Anjani Kumari, and Priya Kumari** — for their active participation, teamwork, and responsibility in contributing to every aspect of the project. The experience we gained from this internship and project will always help us in our future learning and career.

INDEX

ABSTRACT

The **Alarm Clock** project is a real-time, GUI-based desktop application built using **Python programming language**, with **Tkinter** for the graphical interface and **SQLite** as the backend database. It is designed to offer a customizable, easy-to-use solution for setting and managing alarms.

Unlike traditional alarm clocks, our software allows users to set alarms with descriptive event names, choose between 12-hour or 24-hour formats, and select repeat modes such as Daily or Weekly. The use of threading enables the program to continuously check alarm times in the background without freezing the GUI. An integrated popup window appears when an alarm is triggered, and sound is played using the **pygame** module.

This project was created as part of an internship at UGCPL and aims to combine theoretical learning with practical implementation. It is a complete example of a real-world Python application that integrates GUI, databases, sound, and multitasking logic.

INTRODUCTION

An alarm clock is a basic but powerful tool in daily life. In today’s fast-paced environment, having an alarm system that is customizable, reliable, and easy to use is very helpful. Instead of relying on mobile or physical alarm clocks, we created a **Python-based desktop alarm clock** that offers more control and flexibility.

This project was developed during our internship to help us understand how Python can be used to build useful software. It also helped us learn about how real-time processes work, how to manage databases, and how to make user-friendly graphical interfaces. The system supports alarm creation with optional event names, a repeat mechanism, and snooze functionality.

We learned how to combine different Python modules to create an application that not only looks good but also works efficiently. It is a small but complete project that helped us gain confidence in software development.

OBJECTIVE

The main goal of this project is to develop a simple yet effective desktop-based alarm system that users can use to manage their time better. Here are the detailed objectives:

- To build a graphical user interface (GUI) using Tkinter that allows easy interaction with the user.
- To allow users to set alarms with time, event name, repeat pattern, and days.
- To store alarms in a permanent way using SQLite, so that data is not lost after closing the application.
- To add a snooze function to delay alarms by 5 minutes when needed.
- To play a sound alert using pygame and show a popup window to get the user's attention.
- To manage multiple alarms and allow operations like turning them ON/OFF or deleting them.

TOOLS AND TECHNOLOGY USED

To build the project, we used the following tools and technologies:

- **Python:** Our primary programming language, which is easy to learn and powerful enough to create full applications.

- **Tkinter:** This is the default Python library for building GUIs. We used it to create the interface, input forms, buttons, tables, and popups.

- **SQLite (DB Browser):** We used QLite to store the alarms in a structured format. It is a lightweight, server-less database.

- **pygame:** This module is used to play alarm sounds when the alarm is triggered.

- **Date and time:** These libraries were used to handle time-related functions and check the current time.

- **Threading:** This was used to check for alarms in the background without interrupting the GUI.

METHODOLOGY

The **methodology** used for developing the Alarm Clock project followed a systematic approach to software development. This process helped us go from idea to implementation in a structured and efficient way. We divided the work into different stages and completed each one with proper planning, coding, and testing.

Each stage of the methodology is explained below:

❖ Requirement Analysis

At the beginning of the project, we conducted a **requirement analysis** session where we discussed what features the alarm clock should have and how users would interact with it.

We identified that our alarm clock must:

- Allow setting alarms with event names.
- Offer 12-hour and 24-hour format options.
- Store alarm data permanently using a database.
- Provide options for repeat (Daily/Weekly)
- Show a popup window with sound alert.
- Include a snooze function.
- Display all alarms in a user-friendly interface.

This stage was very important because it gave us a **clear vision** of what we wanted to build.

❖ User Interface (UI) Design

Once we had our requirements, we started working on the **Graphical User Interface (GUI)** using **Tkinter**, Python’s built-in GUI toolkit. We chose a **dark theme layout** to make the app look modern and visually appealing.

The interface design included:

- Input fields for time and event name.
- Dropdown menus for repeat options and AM/PM.
- Checkboxes for selecting weekdays
- A treeview (table) for displaying alarms.
- Buttons for Set, Delete, Toggle, Snooze, and Stop actions.

We made sure that the interface was **simple, clean, and easy to use**, even for people with little technical knowledge.

❖ <u>Database Design and Integration</u>

To make sure that alarm data is **not lost** when the app is closed, we used an **SQLite database**. SQLite is lightweight, serverless, and easy to integrate with Python.

We designed a table called alarms with the following fields

- id (primary key)
- time (formatted alarm time)
- event (user-defined label)
- status (ON/OFF)
- repeat (No, Daily, Weekly)
- days (for storing selected weekdays)

This allowed us to **store, retrieve, update, and delete** alarms directly from the database.

❖ <u>Core Logic Development</u>

This is where we implemented the **actual alarm functionality**.

- We created functions to set new alarms, validate inputs, and insert them into the database.
- We added logic to detect duplicates and prevent users from adding the same alarm more than once.
- The program converts time input (12-hour or 24-hour) into a standard format using Python's date-time module.

We made sure that each function was **modular**, meaning it did one task only, so that the code was easy to understand and maintain.

❖ <u>Real-Time Alarm Checking Using Threading</u>

Checking alarm time continuously in a loop would freeze the GUI. So we used the threading **module** to run a background thread.

This thread:

- Runs every second.

- Checks the current time against stored alarm times.
- Verifies repeat conditions (Daily, Weekly, or No Repeat).
- Triggers an alarm only once per day per alarm.
- Plays a sound using pygame and shows a popup.

Using threading made the application **responsive** and smooth to use.

❖ Sound Playback and Popup Alerts

- When the alarm time is reached:
- A **popup window** appears with the alarm event name.
- The window has two buttons: **Snooze (5 minutes)** and **Stop**.
- Alarm sound plays in a loop using the pygame.mixer module.

The snooze button resets the alarm temporarily for 5 minutes, while the stop button ends the sound and closes the popup.

We made sure the popup is **always on top** so the user cannot miss it.

❖ Alarm Deletion and Toggle Controls

We added options to:

- **Delete alarms** permanently from the database.
- **Toggle ON/OFF** without removing them.

This is done through the Treeview. The user can select any alarm and click the respective button to control it. This gave users full control over their alarms.

❖ Error Handling and Input Validation

To make the app robust, we added checks for:

- Invalid time input (like 25:80).
- Empty event names.

- Incorrect formats.
- Duplicate alarms.

Whenever a user makes an error, a **message box appears** showing a clear error message.

This ensured the app does not crash and guides the user to correct mistakes.

Testing the Application

We tested the application for:

- Different time formats (AM/PM and 24-hour).
- Repeat options and weekday selections.
- Functionality of snooze and stop.
- Database persistence across sessions.
- Proper alarm triggering at the exact time.

Each team member tested on their own computer systems. We also checked behavior when no internet was available (since SQLite is offline).

Finalization and Deployment

After testing, we:

- Cleaned up the code.
- Removed unused variables and comments.
- Created a README for setup instructions.
- Placed alarm.mp3 in the folder with the Python file.

Now the application can be run by any user who has Python and pygame installed. It is a **ready-to-use desktop tool**.

PROJECT FEATURES

The **Alarm Clock Application** we developed as part of our internship is packed with useful and interactive features. Each feature has been thoughtfully implemented to ensure ease of use, flexibility, and functional utility. Below is a detailed list of the core features offered by our project:

❖ Multiple Alarm Support

The system allows users to create and manage **multiple alarms** at the same time. Each alarm can have its own time, event description, and configuration settings. This means users can set different alarms for various tasks throughout the day — such as waking up, taking medicine, attending meetings, or completing assignments — all within one application.

❖ 12-Hour and 24-Hour Time Format

To suit different user preferences, we have included both **12-hour format (with AM/PM)** and **24-hour format (military time)**. Users can select their desired time format before entering the alarm time, making the system flexible and easy for all users, regardless of which format they are comfortable with.

❖ Event Name or Description

Each alarm can be assigned a short **event name or description**, such as “Study Time”, “Drink Water”, or “Team Meeting”. This label appears when the alarm goes off and helps users remember why the alarm was set in the first place. This feature makes the alarm clock more than just a timer — it becomes a smart reminder tool.

❖ Repeat Alarm Functionality

The system supports **three types of repeat options** for alarms:

- **No Repeat** – alarm rings only once.
- **Daily Repeat** – alarm rings every day at the same time.
- **Weekly Repeat** – alarm rings on selected weekdays (like Monday, Wednesday, Friday).

This repeat feature ensures that users do not have to manually set the same alarm again and again. Weekly repeat includes checkboxes for each weekday, giving fine control over the alarm schedule.

❖ Weekday Selection for Weekly Repeat

When a user chooses the "Weekly" repeat option, they can select specific days from **Monday to Sunday** on which the alarm should ring. This is especially useful for people who have different schedules on different days — like college students or working professionals.

❖ Alarm Sound Integration

A major highlight of our application is its ability to play a **custom alarm sound** when the set time is reached. The sound is played using the **pygame** module in Python, which allows smooth audio playback. The sound file alarm.mp3 is played in a loop until the user chooses to snooze or stop the alarm.

❖ Popup Alert Window

At the time of an alarm, a **popup window** appears automatically. This window contains the event name and two action buttons — **Snooze** and **Stop**. It ensures the alarm doesn't go unnoticed and gives the user options to delay or dismiss the alert. The popup also brings the application to the front to catch the user's attention.

❖ Snooze Functionality (5 Minutes)

The **Snooze** button allows the user to delay the alarm by 5 minutes. This feature is useful when the user needs a short break or isn't ready to dismiss the alarm yet. After snoozing, the alarm is temporarily disabled and will ring again after 5 minutes.

❖ Toggle Alarm ON/OFF

Each alarm can be **enabled or disabled** using a simple ON/OFF toggle. This helps users keep their alarms in the list without deleting them. For example, a weekend alarm can be turned OFF during holidays and then reactivated later without needing to re-enter the details.

❖ Delete Alarm Feature

Users can **delete alarms** they no longer need. This removes them permanently from the database. The feature ensures the interface stays clean and organized, especially when the user has many alarms set.

❖ Alarm Overview Table (Treeview)

All alarms set by the user are displayed in a structured **table format** using Tkinter's Treeview widget. The table shows the time, event name, repeat type, days selected, and status (ON/OFF). This makes it easy for users to view and manage all alarms in one place.

❖ Persistent Storage with SQLite Database

All alarm details are stored in an **SQLite database** (alarms.db) to ensure they are not lost when the application is closed. This means users can set alarms once and trust that they will still be active the next time they launch the program.

❖ Modern and Clean GUI Design

The graphical user interface is designed with a **dark mode theme**, consistent font styling, and clean button layouts. We used custom colors and spacing to make the application look professional, modern, and easy to navigate. Buttons and input fields respond well, and the overall experience is smooth.

❖ Real-Time Alarm Monitoring Using Threading

To ensure the GUI stays responsive and alarms are triggered in real-time, the alarm-checking logic runs in a **separate background thread**. This avoids freezing the application and ensures alarms are checked every second, even if the user is interacting with the interface.

❖ Simple Folder-Based Setup

The application is self-contained. Users only need to keep the Python script, the database file, and the sound file (alarm.mp3) in the same folder. There is no need for any installation or external server setup.

❖ Input Validation and Error Handling

To ensure that users do not make mistakes while entering the time or selecting options, we implemented several **input validations**. For example:

- Time must be in proper format (HH:MM).
- Event name should not be blank.
- Duplicate alarms are not allowed.
- Error messages are shown when inputs are invalid.

STEP-BY-STEP WORKING

- User opens the application and the main GUI window appears.
- The user selects either **12-hour** or **24-hour** format.
- User enters the time, event name, and selects repeat type.
- If Weekly is selected, they can check the weekdays.
- On clicking **“Set Alarm”**, the alarm is saved in the database.
- The alarm is displayed in a table showing all its details.
- A separate thread checks current time in the background.
- When time matches, a **popup appears** and sound is played
- The user can either click **“Snooze”** (adds 5 minutes) or **“Stop”** (stops sound).
- The user can also delete alarms or toggle their status from the table.

PROJECT LIMITATIONS

- Despite its many features, the project has the following limitations:
- It runs only on **desktop systems** (Windows/Linux).
- The user must have **Python installed** to run the project.
- The alarm sound file **alarm.mp3** must be present in the project folder.
- There is **no login or user authentication**.
- The interface currently supports **only English**.
- The app does not run in the background when closed.

FUTURE SCOPE

- There are many ways to improve and expand the project in the future:
- Add **user accounts** to allow saving alarms for multiple users.
- Create a **mobile app version** using Kivy or Flutter.
- Add **voice alerts** using Text-to-Speech (TTS).
- Send **email or push notifications** as alarms.
- Allow **cloud storage and sync** using Firebase or Google Drive.
- Add **calendar view** and time-based scheduling.
- Enable **background operation** with system tray integration.

CONCLUSION

Working on this project has been a very educational and fulfilling experience for all of us. It taught us not only about how to use Python in real-world projects but also about **teamwork**, **logic building**, **problem-solving**, and **time management**.

We gained practical experience in working with GUIs, databases, and threads. We also learned how to manage a complete software project from scratch. This Alarm Clock application is a simple but powerful example of what can be achieved using the core concepts of Python programming.

BIBLIOGRAPHY

- Chat GPT
- Python Official Documentation – https://docs.python.org/3
- GeeksforGeeks Python GUI – https://www.geeksforgeeks.org/python-gui-tkinter
- SQLite Documentation – https://www.sqlite.org/docs.html
- pygame Module – https://www.pygame.org/docs
- StackOverflow
- YouTube

APPENDIX: PROJECT CODE

```
import tkinter as tk
from tkinter import ttk, messagebox
from datetime import datetime
import sqlite3
import threading
import time
import os
import pygame

pygame.mixer.init()
temp_snooze = {}
triggered_today = {}

class AlarmClock:
def __init__(self, root):
self.root = root
self.root.title("Alarm Clock")
self.root.geometry("600x500")
self.root.configure(bg="#1e1e2f")  # Dark background

self.conn = sqlite3.connect("alarms.db", check_same_thread=False)
self.cursor = self.conn.cursor()
self.cursor.execute('''
CREATE TABLE IF NOT EXISTS alarms (
id INTEGER PRIMARY KEY AUTOINCREMENT,
time TEXT NOT NULL,
event TEXT,
status TEXT,
repeat TEXT,
days TEXT
)
''')
self.conn.commit()

self.event_var = tk.StringVar()
self.repeat_var = tk.StringVar(value="No")
self.status_var = tk.StringVar(value="ON")
self.time_format_var = tk.StringVar(value="12")
self.ampm_var = tk.StringVar(value="AM")
self.days_vars = {day: tk.BooleanVar() for day in ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]}

self.time_24_var = tk.StringVar()
self.hour_12_var = tk.StringVar()
self.min_12_var = tk.StringVar()

self.popup_window = None

self.build_gui()
self.load_alarms()
threading.Thread(target=self.check_alarms, daemon=True).start()

def build_gui(self):
style = ttk.Style()
style.theme_use("default")
style.configure("Treeview", background="#2e2e3e", foreground="white", fieldbackground="#2e2e3e", rowheight=25)
style.configure("Treeview.Heading", background="#3a3a4f", foreground="white", font=('Helvetica', 10, 'bold'))
style.map("Treeview", background=[('selected', '#5c5cff')])
```

```
entry_frame = tk.Frame(self.root, bg="#1e1e2f")
entry_frame.pack(pady=10)

tk.Label(entry_frame, text="Choose Time Format", bg="#1e1e2f", fg="white").grid(row=0, column=0, sticky='w')
format_frame = tk.Frame(entry_frame, bg="#1e1e2f")
format_frame.grid(row=0, column=1, sticky='w')
tk.Radiobutton(format_frame, text="12-hour (AM/PM)", variable=self.time_format_var, value="12", command=self.update_time_input, bg="#1e1e2f", fg="white", selectcolor="#444").pack(side=tk.LEFT)
tk.Radiobutton(format_frame, text="24-hour", variable=self.time_format_var, value="24", command=self.update_time_input, bg="#1e1e2f", fg="white", selectcolor="#444").pack(side=tk.LEFT)

self.time_input_frame = tk.Frame(entry_frame, bg="#1e1e2f")
self.time_input_frame.grid(row=1, column=1)
self.update_time_input()

tk.Label(entry_frame, text="Time", bg="#1e1e2f", fg="white").grid(row=1, column=0)
tk.Label(entry_frame, text="Event", bg="#1e1e2f", fg="white").grid(row=2, column=0)
tk.Entry(entry_frame, textvariable=self.event_var, width=20, bg="#2e2e3e", fg="white", insertbackground="white").grid(row=2, column=1)

tk.Label(entry_frame, text="Repeat", bg="#1e1e2f", fg="white").grid(row=3, column=0)
self.repeat_box = ttk.Combobox(entry_frame, textvariable=self.repeat_var, values=["No", "Daily", "Weekly"])
self.repeat_box.grid(row=3, column=1)
self.repeat_box.current(0)

tk.Label(entry_frame, text="Days (for Weekly)", bg="#1e1e2f", fg="white").grid(row=4, column=0)
days_frame = tk.Frame(entry_frame, bg="#1e1e2f")
days_frame.grid(row=4, column=1)
for i, (day, var) in enumerate(self.days_vars.items()):
tk.Checkbutton(days_frame, text=day, variable=var, bg="#1e1e2f", fg="white", selectcolor="#444").grid(row=0, column=i)

tk.Button(entry_frame, text="Set Alarm", command=self.set_alarm, bg="#444", fg="white", activebackground="#666").grid(row=5, columnspan=2, pady=5)

self.tree = ttk.Treeview(self.root, columns=("time", "event", "status", "repeat", "days"), show='headings')
for col in ("time", "event", "status", "repeat", "days"):
self.tree.heading(col, text=col.capitalize())
self.tree.pack(expand=True, fill='both', padx=10, pady=10)

btn_frame = tk.Frame(self.root, bg="#1e1e2f")
btn_frame.pack(pady=5)
tk.Button(btn_frame, text="Delete", command=self.delete_alarm, bg="#880808", fg="white", activebackground="#aa0000").pack(side=tk.LEFT, padx=5)
tk.Button(btn_frame, text="Toggle ON/OFF", command=self.toggle_alarm, bg="#005f5f", fg="white", activebackground="#007f7f").pack(side=tk.LEFT, padx=5)

self.status_label = tk.Label(self.root, text="", fg="lightgreen", bg="#1e1e2f")
self.status_label.pack()

self.root.protocol("WM_DELETE_WINDOW", self.on_close)

def update_time_input(self):
for widget in self.time_input_frame.winfo_children():
widget.destroy()
if self.time_format_var.get() == "12":
tk.Entry(self.time_input_frame, textvariable=self.hour_12_var, width=5, bg="#2e2e3e", fg="white", insertbackground="white").pack(side=tk.LEFT)
tk.Label(self.time_input_frame, text=":", bg="#1e1e2f", fg="white").pack(side=tk.LEFT)
tk.Entry(self.time_input_frame, textvariable=self.min_12_var, width=5, bg="#2e2e3e", fg="white", insertbackground="white").pack(side=tk.LEFT)
ampm_menu = ttk.Combobox(self.time_input_frame, textvariable=self.ampm_var, values=["AM", "PM"], width=5, state="readonly")
ampm_menu.pack(side=tk.LEFT)
ampm_menu.current(0)
```

```
else:
tk.Entry(self.time_input_frame, textvariable=self.time_24_var, width=10, bg="#2e2e3e", fg="white", insertbackground="white").pack(side=tk.LEFT)
tk.Label(self.time_input_frame, text="(HH:MM)", bg="#1e1e2f", fg="white").pack(side=tk.LEFT)

def set_alarm(self):
event = self.event_var.get().strip()
repeat = self.repeat_var.get().strip()
status = self.status_var.get().strip()
days_selected = ",".join([d for d, v in self.days_vars.items() if v.get()])
format_selected = self.time_format_var.get()

if format_selected == "12":
hour = self.hour_12_var.get().zfill(2)
minute = self.min_12_var.get().zfill(2)
ampm = self.ampm_var.get()
time_input = f"{hour}:{minute} {ampm}"
try:
parsed_time = datetime.strptime(time_input, "%I:%M %p")
except ValueError:
messagebox.showerror("Error", "Invalid 12-hour format. Use HH MM and AM/PM.")
return
else:
time_input = self.time_24_var.get()
try:
parsed_time = datetime.strptime(time_input, "%H:%M")
except ValueError:
messagebox.showerror("Error", "Invalid 24-hour format. Use HH:MM.")
return

formatted_time = parsed_time.strftime("%I:%M %p")

self.cursor.execute("SELECT * FROM alarms WHERE time=? AND event=?", (formatted_time, event))
if self.cursor.fetchone():
messagebox.showerror("Error", "Duplicate alarm")
return

self.cursor.execute("INSERT INTO alarms (time, event, status, repeat, days) VALUES (?, ?, ?, ?, ?)",
(formatted_time, event, status, repeat, days_selected))
self.conn.commit()
self.load_alarms()

self.event_var.set("")
self.hour_12_var.set("")
self.min_12_var.set("")
self.ampm_var.set("AM")
self.time_24_var.set("")
for v in self.days_vars.values():
v.set(False)

def load_alarms(self):
for row in self.tree.get_children():
self.tree.delete(row)
self.cursor.execute("SELECT * FROM alarms ORDER BY time")
for row in self.cursor.fetchall():
self.tree.insert('', 'end', iid=row[0], values=row[1:])

def delete_alarm(self):
selected = self.tree.selection()
for i in selected:
self.cursor.execute("DELETE FROM alarms WHERE id=?", (i,))
self.conn.commit()
self.load_alarms()
```

```
def toggle_alarm(self):
selected = self.tree.selection()
for i in selected:
current = self.tree.item(i)['values']
new_status = "OFF" if current[2] == "ON" else "ON"
self.cursor.execute("UPDATE alarms SET status=? WHERE id=?", (new_status, i))
self.conn.commit()
self.load_alarms()

def check_alarms(self):
thread_conn = sqlite3.connect("alarms.db", check_same_thread=False)
thread_cursor = thread_conn.cursor()

while True:
now = datetime.now()
current_time = now.strftime("%I:%M %p")
weekday = now.strftime("%a")[:3]
key_time = now.strftime("%Y-%m-%d %I:%M %p")

thread_cursor.execute("SELECT * FROM alarms WHERE status='ON'")
for row in thread_cursor.fetchall():
alarm_id, time_str, event, status, repeat, days = row

if temp_snooze.get(alarm_id) and time.time() < temp_snooze[alarm_id]:
continue

if time_str == current_time:
if repeat == "No" or repeat == "Daily" or (repeat == "Weekly" and weekday in days.split(",")):
if triggered_today.get(alarm_id) != key_time:
triggered_today[alarm_id] = key_time
self.status_label.config(text=f"Triggered: {event} at {current_time}")
if repeat == "No":
thread_cursor.execute("UPDATE alarms SET status='OFF' WHERE id=?", (alarm_id,))
thread_conn.commit()
self.trigger_alarm(alarm_id, event)
time.sleep(1)

def trigger_alarm(self, alarm_id, event):
if self.popup_window and self.popup_window.winfo_exists():
return

def alarm_popup():
self.popup_window = tk.Toplevel(self.root)
self.popup_window.title("Alarm")
self.popup_window.geometry("250x150")
tk.Label(self.popup_window, text=f"Event: {event}").pack(pady=10)
tk.Button(self.popup_window, text="Snooze 5 Min", command=lambda: self.snooze_alarm(alarm_id)).pack()
tk.Button(self.popup_window, text="Stop", command=self.stop_sound).pack()

mp3_path = os.path.join(os.getcwd(), "alarm.mp3")
if os.path.exists(mp3_path):
pygame.mixer.music.load(mp3_path)
pygame.mixer.music.play(-1)
else:
print("Alarm sound not found:", mp3_path)

self.root.after(0, alarm_popup)

def stop_sound(self):
pygame.mixer.music.stop()
if self.popup_window:
self.popup_window.destroy()

def snooze_alarm(self, alarm_id):
```

```
temp_snooze[alarm_id] = time.time() + 300
self.stop_sound()

def on_close(self):
pygame.mixer.music.stop()
self.conn.close()
self.root.destroy()

if __name__ == '__main__':
root = tk.Tk()
app = AlarmClock(root)
root.mainloop()
```