***Advanced Alarm Clock Python Project***

## 📅 Project Overview

This project is a fully functional alarm clock application built using Python and Tkinter. It supports multiple alarms, snoozing, customizable alarm tones, a digital and analog clock interface, and saves alarm history to a file. It is designed with a clean, dark-themed GUI.

## 🌐 Features

- ⏰ Set multiple alarms

- 🔊 Use custom audio files as alarm tones (MP3/WAV)

- ⏲️ Snooze support with customizable snooze duration

- 🌡 Analog and digital clock in real-time

- 🔒 Dark mode GUI

- 🕙 Save alarm history to a text file with timestamps

- ❌ Ability to remove alarms

## 👨‍💻 Technologies Used

- Python 3

- Tkinter for GUI

- Pygame for audio playback

- Threading for background alarm checking

- Math for analog clock rendering

## 📖 How It Works

1. User Interface:

- Built using Tkinter widgets

- Shows real-time digital and analog clocks

- Allows the user to set alarms, choose tones, and snooze

1. Alarm Management:

- Stored as objects in a list

- Checked in a background thread every 10 seconds

- When time matches, alarm rings and prompts user action

1. Alarm History:

- Logged to alarm\_history.txt with timestamps

1. Analog Clock Drawing:

- Uses Canvas widget and trigonometry to draw hands

## 📄 Code Explanation

1. Alarm Class:

class Alarm:

def \_\_init\_\_(self, time\_str, snooze, tone):

self.time\_str = time\_str

self.snooze = snooze

self.tone = tone

self.time\_obj = datetime.datetime.strptime(time\_str, "%H:%M").time()

self.active = True

**Purpose**: Holds data for each alarm: time, snooze duration, tone file path, and status.

1. add\_alarm():

def add\_alarm(self):

time\_str = self.time\_entry.get()

snooze = int(self.snooze\_entry.get())

alarm = Alarm(time\_str, snooze, self.selected\_tone)

self.alarms.append(alarm)

**Purpose**: Reads user input and adds a new alarm to the list.

1. check\_alarms():

def check\_alarms(self):

while True:

now = datetime.datetime.now().time()

for alarm in self.alarms:

if alarm.active and ...:

self.trigger\_alarm(alarm)

time.sleep(10)

**Purpose**: Runs continuously in a thread to check if any alarm should ring.

1. trigger\_alarm():

def trigger\_alarm(self, alarm):

play\_alarm(alarm.tone)

...

while True:

choice = messagebox.askquestion("Alarm", "Snooze or Stop?")

if choice == 'yes':

... # Snooze logic

else:

... # Stop logic

**Purpose**: Handles what happens when alarm time is matched: snoozing or stopping.

## 📊 Sample alarm\_history.txt

2025-05-27 07:00:00 - ALARM RANG at 07:00

2025-05-27 07:00:10 - SNOOZED 07:00 for 5 mins

2025-05-27 07:05:10 - STOPPED 07:00

## 🔧 How to Run the Project

1. Install Python and pip

2. Install pygame:

pip install pygame

3. Save the Python script (e.g., alarm\_clock.py)

4. Run the file:

python alarm\_clock.py

## 🔄 Optional Improvements

- Convert to desktop `.exe` using pyinstaller

- Add limit to snooze attempts

- Set alarm for future dates

- GUI volume control

- Add calendar view

## 🎉 Conclusion

This alarm clock project is a great combination of GUI design, real-time updates, sound control, and file handling in Python. It can be expanded into a full productivity or reminder app.

Happy Coding! 🚀