Advance Programming Practice

Python

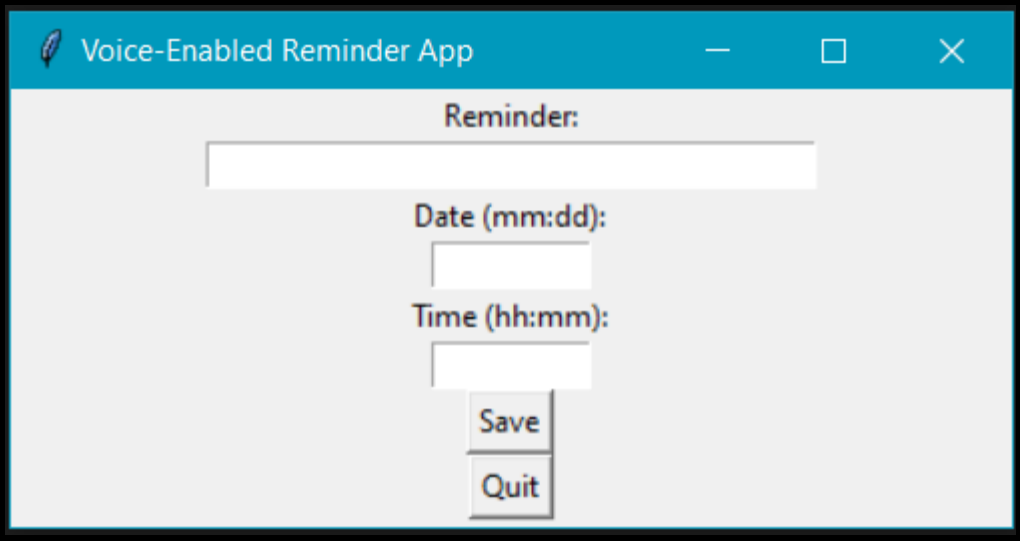
based

Reminder App

By:
Shashanka Niraula (RA221102601028)
ANKARBOINA SURABHI (RA2211026010527)
BHASWAT LENKA (RA2211026010522)

About:

Welcome to our Python Reminder App – a versatile and user-friendly solution to help you stay organized and on top of your tasks! Designed with efficiency and simplicity in mind, our app leverages the power of Tkinter GUI and SpeechRecognition to provide a seamless reminder experience using both manual input and voice commands. The app transcribes spoken reminders into text, offering a hands-free alternative for users seeking efficiency.



With robust manual management options users can modify, delete, and list reminders at their convenience, tailoring the application to their dynamic schedules. Furthermore, the app operates continuously in the background, diligently checking for reminder conditions. Once a reminder's time and date align, the app triggers a customizable sound notification and a pop-up window, ensuring users stay informed and on top of their tasks.

Our Python Reminder App is more than just a tool; it's a reliable companion designed to simplify your life by keeping you organized and on track.

Features:

GUI with Tkinter:

The app has a clean and user-friendly graphical interface created using Tkinter, making it easy for users to navigate and interact with the reminder functionalities effortlessly.

Voice Reminder Input:

With the help of SpeechRecognition API, the app allows you to set reminders using your voice. Simply speak your reminder, and let the app transcribe it into a text format, saving time and effort.

Manual Reminder Management:

With Convenience of manual management options Add, Delete, Change, or List it gives more functions to the user. Also allows the user to use the app even in Noisy environment

Background Continuity:

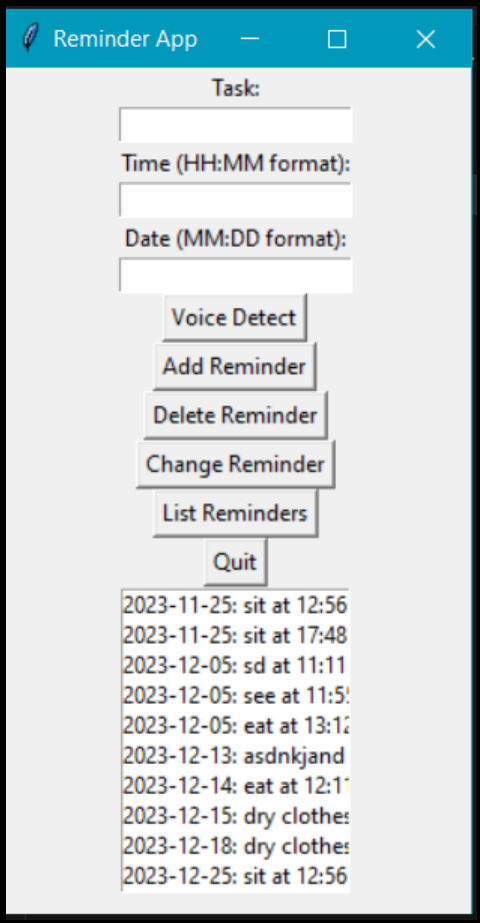
The app runs continuously in the background, checking for reminder times and dates. Once set reminder conditions are met, the app springs into action, playing a alarm with a displaying pop-up window to ensure user doesn't miss their reminder

Learning Curve:

Libraries:

Tkinter:

Mastering Tkinter in Python marks an essential step for developing straightforward and effective graphical user interfaces (GUIs). Tkinter, serving as Python's standard GUI toolkit, offers a straightforward approach to GUI development that seamlessly integrates into Python, simplifying the process of building interfaces. Through its widgets, layout managers, and event handling mechanisms, Tkinter provides a pragmatic solution for crafting visually pleasing applications, whether they're basic desktop tools or more complex software endeavors.Proficiency in Tkinter lays the groundwork for developing practical and user-friendly applications that bridge the gap between code and end-users.



Sub-Processes:

Diving into the intricacies of using subprocess in Python unveils a powerful tool for managing external processes and executing system commands seamlessly. The subprocess module offers a robust and flexible interface, enabling developers to interact with system commands, scripts, and executables directly from their Python scripts. Learning to use subprocess involves understanding its various functions, such as run(), Popen(), and check_output(), each catering to different scenarios. This module proves invaluable when integration with external programs or handling parallel execution is necessary. By mastering subprocess, Python developers gain a versatile skill set, enhancing their ability to orchestrate complex workflows, automate tasks, and harness the capabilities of the underlying operating system.

Pygame:

The application of Pygame for playing music in Python is a straightforward yet dynamic avenue for audio integration in software projects. Pygame, a versatile library originally designed for game development, extends its capabilities to include sound and music functionalities. Learning to utilize Pygame's music module allows developers to seamlessly incorporate background music, sound effects, and interactive audio elements into their Python applications. By understanding Pygame's music playback functions, such as `pygame.mixer.init()`, `pygame.mixer.music.load()`, and `pygame.mixer.music.play()`, developers can easily integrate and control audio playback within their projects.

Other Library:

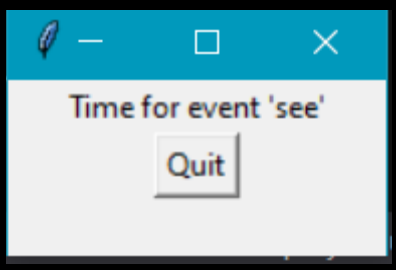
Learning to use the os, datetime, and threading modules in Python provides a versatile toolkit for various tasks. The os module facilitates interaction with the operating system, enabling file management and process control. The datetime module is crucial for handling date and time operations, useful in scheduling and logging. Finally, threading allows concurrent execution, enhancing program efficiency. Together, these modules empower developers to create efficient, time-sensitive applications with seamless system interactions.

Implementation:

The implementation of this reminder app required an approach involving five distinct Python programs working in tandem to ensure seamless functionality. The first program is dedicated to voice recognition, leveraging the SpeechRecognition library to decode user input. This program employs Tkinter for GUI elements and Subprocess to seamlessly allow the user to add as many voice reminders as they want or guide the user towards the second program .

Program 2 takes charge of fundamental reminder functionalities, managing tasks like editing, deleting, listing, and adding reminders manually. Each button triggers subprocesses working in the background to keep reminders updated and maintain the timing for the ringtone. This program acts as the control center, directing the flow of information to the subsequent components.

Program 3, essential for the auditory aspect, utilizes pygame to manage the ringtone. Continuously monitoring the database, this program triggers a pop-up window and plays the ringtone when the specified date and time conditions are met. It operates persistently in the background to ensure timely reminders.



The 4th and 5th programs may seem modest but play critical roles. They handle the sequencing of program execution and the essential task of sorting and deleting reminder files based on their designated date and time. Together, these five programs collaboratively bring the reminder app to life, offering a robust and user-friendly solution for efficient task management.

Code:

(As the names of program are crucial for subprocess the names aren't conventional)

Program 1: finalspeech_recog

```
import tkinter as tk
from tkinter import messagebox
import re
import subprocess
import speech_recognition as sr
from dateutil import parser
import threading
from datetime import datetime

recognizer = sr.Recognizer()

# Initialize variables
reminder_text = ""
date_text = ""
time_text = ""

# Function to open other applications and close the main app
def open_applications_and_quit():
    app_1_path = "D:\\pythonProject\\reminder_app\\Final_work_tkinter\\final_reminder_play_sound_part.py"
    app_2_path = "D:\\pythonProject\\reminder_app\\Final_work_tkinter\\final_reminder_app.py"
    app_3_path = "D:\\pythonProject\\reminder_app\\Final_work_tkinter\\final_sorting_reminder_part.py"

    subprocess.Popen(["python", app_3_path])
    subprocess.Popen(["python", app_2_path])
    subprocess.Popen(["python", app_1_path])
    app.quit()
```

```
# Function to save the reminder to a file
def save_reminder(reminder_text, date_text, time_text):
    try:
        # Get the current date and time
        now = datetime.now()
        current_year = now.year
        formatted_date = f"{current_year}-{date_text}"

        # Create the reminder string in the specified format
        reminder_str = f"{formatted_date},{reminder_text},{time_text}\n"
        with open("reminders.txt", "a") as file:
            file.write(reminder_str)

        messagebox.showinfo("Success", "Reminder saved successfully!")
        reminder_entry.delete(0, tk.END)
        date_entry.delete(0, tk.END)
        time_entry.delete(0, tk.END)
    except Exception as e:
        print(f"Error saving reminder: {e}")

# Function to continuously listen for voice input
def listen_for_voice_input():
    global reminder_text, date_text, time_text

    while True:
        with sr.Microphone() as source:
            recognizer.adjust_for_ambient_noise(source)
            print("Listening...")
            audio = recognizer.listen(source)
            print("Recognizing...")

        try:
            voice_input = recognizer.recognize_google(audio)
            print("You said:", voice_input)

            # First-level checks
            reminder_match = re.search(r'remind me to (.+) on', voice_input)
```

```
            time_match = re.search(r'at (\d+:\d+(?: [ampm]*))', voice_input)
            date_match = re.search(r'on (\d+:\d+)', voice_input)

            if reminder_match:
                reminder_text = reminder_match.group(1)
                reminder_entry.delete(0, tk.END)
                reminder_entry.insert(0, reminder_text)
            if time_match:
                # Extract the time string and convert it to 24-hour format
                time_str = time_match.group(1) + 'm'  # Add 'm' to make sure
it's a complete time string
                time_obj = datetime.strptime(time_str, '%I:%M %p')
                time_text = time_obj.strftime('%H:%M')
                time_entry.delete(0, tk.END)
                time_entry.insert(0, time_text)
            if date_match:
                date_text = date_match.group(1)
                date_entry.delete(0, tk.END)
                date_entry.insert(0, date_text)

            if reminder_text and date_text and time_text:
                save_reminder(reminder_text, date_text, time_text)
                open_applications_and_quit()

        except sr.UnknownValueError:
            print("Could not understand audio")
        except sr.RequestError as e:
            print(f"Error recognizing audio: {e}")

            # Second-level checks (if the first checks didn't catch it)
            if not reminder_text and "remind me to" in voice_input:
                reminder_text = voice_input.split("remind me to", 1)[1].strip()
                reminder_entry.delete(0, tk.END)
                reminder_entry.insert(0, reminder_text)
            if not time_text and "at" in voice_input:
                time_match_second = re.search(r'at (\d+:\d+)',
voice_input.split("at", 1)[1])
                if time_match_second:
```

```
                    time_text = time_match_second.group(1)
                    time_entry.delete(0, tk.END)
                    time_entry.insert(0, time_text)
            if not date_text and "on" in voice_input:
                date_match_second = re.search(r'on (\d+:\d+)',
voice_input.split("on", 1)[1])
                if date_match_second:
                    date_text = date_match_second.group(1)
                    date_entry.delete(0, tk.END)
                    date_entry.insert(0, date_text)

            if reminder_text and date_text and time_text:
                save_reminder(reminder_text, date_text, time_text)
                open_applications_and_quit()

        except sr.UnknownValueError:
            print("Could not understand audio")
        except sr.RequestError as e:
            print(f"Error recognizing audio: {e}")

def save_button_callback():
    global reminder_text, date_text, time_text
    reminder_text = reminder_entry.get()
    date_text = date_entry.get()
    time_text = time_entry.get()
    save_reminder(reminder_text, date_text, time_text)

# Create the main window
app = tk.Tk()
app.geometry("400x175")
app.title("Voice-Enabled Reminder App")

# Reminder Entry
reminder_label = tk.Label(app, text="Reminder:")
reminder_label.pack()
reminder_entry = tk.Entry(app, width=40)
reminder_entry.pack()
```

```
# Date Entry
date_label = tk.Label(app, text="Date (mm:dd):")
date_label.pack()
date_entry = tk.Entry(app, width=10)
date_entry.pack()

# Time Entry
time_label = tk.Label(app, text="Time (hh:mm):")
time_label.pack()
time_entry = tk.Entry(app, width=10)
time_entry.pack()

#Save Button
save_button = tk.Button(app, text="Save", command=save_button_callback)
save_button.pack()

# Quit Button
quit_button = tk.Button(app, text="Quit", command=open_applications_and_quit)
quit_button.pack()

# Start a thread for listening to voice input
voice_thread = threading.Thread(target=listen_for_voice_input)
voice_thread.daemon = True
voice_thread.start()

app.mainloop()
```

Program 2: final_reminder_app

```
import tkinter as tk
from tkinter import messagebox
import datetime
import subprocess

REMINDERS_FILE = 'reminders.txt'

def load_reminders():
    reminders = {}
    try:
        with open(REMINDERS_FILE, 'r') as file:
            lines = file.readlines()
            for line in lines:
                parts = line.strip().split(',')
                if len(parts) == 3:
                    date_str, task, time_str = parts
                    date = datetime.datetime.strptime(date_str, "%Y-%m:%d")
                    time_obj = datetime.datetime.strptime(time_str, "%H:%M")

                    # Check if the date already exists in reminders
                    if date not in reminders:
                        reminders[date] = []

                    reminders[date].append({'task': task, 'time':
time_obj.strftime("%H:%M")})
    except FileNotFoundError:
        pass
    return reminders

def save_reminders(reminders):
    with open(REMINDERS_FILE, 'w') as file:
        for date, reminder_list in reminders.items():
            date_str = date.strftime("%Y-%m:%d")
            for reminder in reminder_list:
                time = reminder['time']
```

```
                task = reminder['task']
                file.write(f"{date_str},{task},{time}\n")


def add_reminder():
    task = task_entry.get()
    time_str = time_entry.get()
    date_str = date_entry.get()

    try:
        current_year = datetime.datetime.now().year
        reminder_date =
datetime.datetime.strptime(f"{current_year}-{date_str}", "%Y-%m:%d")
        time_obj = datetime.datetime.strptime(time_str, "%H:%M")

        # Check if the date already exists in reminders
        if reminder_date not in reminders:
            reminders[reminder_date] = []

        reminders[reminder_date].append({'task': task, 'time':
time_obj.strftime("%H:%M")})
        save_reminders(reminders)
        messagebox.showinfo("Success", "Reminder added successfully!")
        task_entry.delete(0, 'end')
        time_entry.delete(0, 'end')
        date_entry.delete(0, 'end')
    except ValueError:
        messagebox.showerror("Error", "Invalid time or date format. Use HH:MM
format for time and MM-DD format for date.")

def delete_reminder():
    date_str = date_entry.get()

    try:
        current_year = datetime.datetime.now().year
        reminder_date =
datetime.datetime.strptime(f"{current_year}-{date_str}", "%Y-%m:%d")

        if reminder_date in reminders:
            del reminders[reminder_date]
```

```
            save_reminders(reminders)
            messagebox.showinfo("Success", "Reminder deleted successfully!")
            date_entry.delete(0, 'end')
        else:
            messagebox.showerror("Error", "Reminder not found for the given
date.")
    except ValueError:
        messagebox.showerror("Error", "Invalid date format. Use MM-DD format.")

def change_reminder():
    date_str = date_entry.get()

    try:
        current_year = datetime.datetime.now().year
        reminder_date =
datetime.datetime.strptime(f"{current_year}-{date_str}", "%Y-%m:%d")
        if reminder_date in reminders:
            new_text = task_entry.get()
            time_str = time_entry.get()
            try:
                time_obj = datetime.datetime.strptime(time_str, "%H:%M")
                reminders[reminder_date] = {'task': new_text, 'time':
time_obj.strftime("%H:%M")}
                save_reminders(reminders)
                messagebox.showinfo("Success", "Reminder changed
successfully!")
                task_entry.delete(0, 'end')
                time_entry.delete(0, 'end')
                date_entry.delete(0, 'end')
            except ValueError:
                messagebox.showerror("Error", "Invalid time format. Use HH:MM
format.")
        else:
            messagebox.showerror("Error", "Reminder not found for the given
date.")
    except ValueError:
        messagebox.showerror("Error", "Invalid date format. Use MM-DD format.")

def list_reminders():
    reminder_list.delete(0, 'end')
```

```
    if reminders:
        for date, reminder_list_for_date in sorted(reminders.items()):
            for reminder in reminder_list_for_date:
                reminder_list.insert('end', f"{date.strftime('%Y-%m-%d')}:
{reminder['task']} at {reminder['time']}")
    else:
        reminder_list.insert('end', "No reminders found.")


def open_applications():
    app_4_path =
"D:\\pythonProject\\reminder_app\\Final_work_tkinter\\close_sound.py"
    subprocess.Popen(["python", app_4_path])
    app_2_path =
"D:\\pythonProject\\reminder_app\\Final_work_tkinter\\final_sorting_reminder_pa
rt.py"
    subprocess.Popen(["python", app_2_path])


def go_back():
    app_1_path =
"D:\\pythonProject\\reminder_app\\Final_work_tkinter\\finalspeech_recog.py"
    subprocess.Popen(["python", app_1_path])



def add_reminder_combined():
    open_applications()
    add_reminder()

def delete_reminder_combined():
    open_applications()
    delete_reminder()

def list_reminders_combined():
    open_applications()
    list_reminders()

def change_reminder_combined():
    change_reminder()
    list_reminders_combined()
```

```
def quit_app():
    root.destroy()

def quit_app_combined():
    open_applications()
    quit_app()

def go_back_combined():
    go_back()
    quit_app()

reminders = load_reminders()

root = tk.Tk()
root.geometry("250x450")
root.title("Reminder App")

# Create and configure the GUI elements
task_label = tk.Label(root, text="Task:")
task_label.pack()
task_entry = tk.Entry(root)
task_entry.pack()

time_label = tk.Label(root, text="Time (HH:MM format):")
time_label.pack()
time_entry = tk.Entry(root)
time_entry.pack()

date_label = tk.Label(root, text="Date (MM:DD format):")
date_label.pack()
date_entry = tk.Entry(root)
date_entry.pack()

go_back_button = tk.Button(root, text="Voice Detect", command=go_back_combined)
go_back_button.pack()

add_button = tk.Button(root, text="Add Reminder",
command=add_reminder_combined)
```

```
add_button.pack()


delete_button = tk.Button(root, text="Delete Reminder",
command=delete_reminder_combined)
delete_button.pack()


change_button = tk.Button(root, text="Change Reminder",
command=change_reminder_combined)
change_button.pack()


list_button = tk.Button(root, text="List Reminders",
command=list_reminders_combined)
list_button.pack()


quit_button = tk.Button(root, text="Quit", command=quit_app_combined)
quit_button.pack()


reminder_list = tk.Listbox(root)
reminder_list.pack()


root.mainloop()
```

Program 3: final_reminder_play_sound_part

```
from datetime import datetime, timedelta
import time
import threading
import pygame
import tkinter as tk
import subprocess
import os
import signal

# Initialize Pygame mixer once at the beginning
pygame.mixer.init()

# Global variable to indicate if the app should stop
stop_signal = False

# Function to play background music
def play_song():
    pygame.mixer.music.load('sunflower_street.mp3')
    pygame.mixer.music.play(-1)

# Function to stop background music
def stop_song():
    pygame.mixer.music.stop()

# Function to close the popup
def close_pop_up(top):
    top.destroy()
```

```
# Function to open another application
def open_applications():
    app_path = os.path.join("D:\\pythonProject\\reminder_app\\Final_work_tkinter\\final_sorting_reminder_part.py")
    subprocess.Popen(["python", app_path])

# Function to check reminders
def check_reminders():
    try:
        with open("reminders.txt", "r") as file:
            lines = file.readlines()

        reminders = []
        current_time = datetime.now()

        for line in lines:
            parts = line.strip().split(',')
            if len(parts) != 3:
                print(f"Invalid line in the remind.txt file: {line}")
                continue

            date_str, reminder, time_str = parts
            try:
                desired_time = datetime.strptime(f"{date_str} {time_str}", "%Y-%m:%d %H:%M")
                reminders.append((desired_time, reminder))
            except ValueError:
                print(f"Invalid date or time format in the remind.txt file: {line}")
                continue
```

```
        if not reminders:
            print("No future reminders found in remind.txt.")
            return

        time_to_sleep = max(0, (reminders[0][0] -
current_time).total_seconds())

        print(f"Waiting for {time_to_sleep} seconds for the next event
'{reminders[0][1]}' at {reminders[0][0]}...")
        time.sleep(time_to_sleep)

        threading.Thread(target=show_gui, args=(reminders[0][1],)).start()

    except FileNotFoundError:
        print("Failed to read remind.txt. Please make sure the file exists.")
        exit(1)

# Function to quit the combined actions
def quit_combined(top):
    stop_song()
    close_pop_up(top)
    open_applications()
    # Add any additional cleanup code here

# Function to display a reminder popup
def show_gui(reminder):
    root = tk.Tk()
    root.withdraw()

    # Custom toplevel window with Quit button
```

```
    top = tk.Toplevel(root)
    top.geometry("150x70")
    top.title("Reminder")
    tk.Label(top, text=f"Time for event '{reminder}'").pack()

    play_song()  # Play the song when the popup shows up

    quit_button = tk.Button(top, text="Quit", command=lambda:
quit_combined(top))
    quit_button.pack()

    root.mainloop()

# Set up a signal handler for SIGTERM (terminate signal)
def handle_stop_signal(signum, frame):
    global stop_signal
    stop_signal = True

signal.signal(signal.SIGTERM, handle_stop_signal)

# Main loop: Check for reminders every 10 seconds
while not stop_signal:
    check_reminders()
    time.sleep(10)
```

Program 4: final_sorting_reminder_part

```
from datetime import datetime

def read_reminders(file_path):
    reminders = []

    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()

        for line in lines:
            parts = line.strip().split(',')
            if len(parts) != 3:
                print(f"Invalid line in the reminder.txt file: {line}")
                continue

            date_str, text, time_str = parts
            try:
                reminder_time = datetime.strptime(f"{date_str} {time_str}",
"%Y-%m:%d %H:%M")
                reminders.append((reminder_time, text))
            except ValueError:
                print(f"Invalid date or time format in the reminder.txt file:
{line}")
                continue

    except FileNotFoundError:
        print("Failed to read reminder.txt. Please make sure the file exists.")
        exit(1)

    return reminders

def delete_passed_reminders(file_path):
```

```
    current_datetime = datetime.now()
    reminders = read_reminders(file_path)

    # Filter out passed reminders for today
    remaining_reminders = [
        (time, text) for time, text in reminders
        if time.date() > current_datetime.date() or
        (time.date() == current_datetime.date() and time.time() >
current_datetime.time())
    ]

    # Sort reminders by the time difference (closest reminders first)
    remaining_reminders = sorted(
        [(time, text) for time, text in reminders if time > current_datetime],
        key=lambda x: x[0] - current_datetime
    )

    # Write the updated reminders back to the file
    with open(file_path, 'w') as file:
        for time, text in remaining_reminders:
file.write(f"{time.strftime('%Y-%m:%d')},{text},{time.strftime('%H:%M')}\n")

if __name__ == "__main__":
    reminder_file = 'reminders.txt'

    # Delete passed reminders and arrange the remaining ones
    delete_passed_reminders(reminder_file)

    print("Reminders processed and updated.")
```

Program 5: close_sound

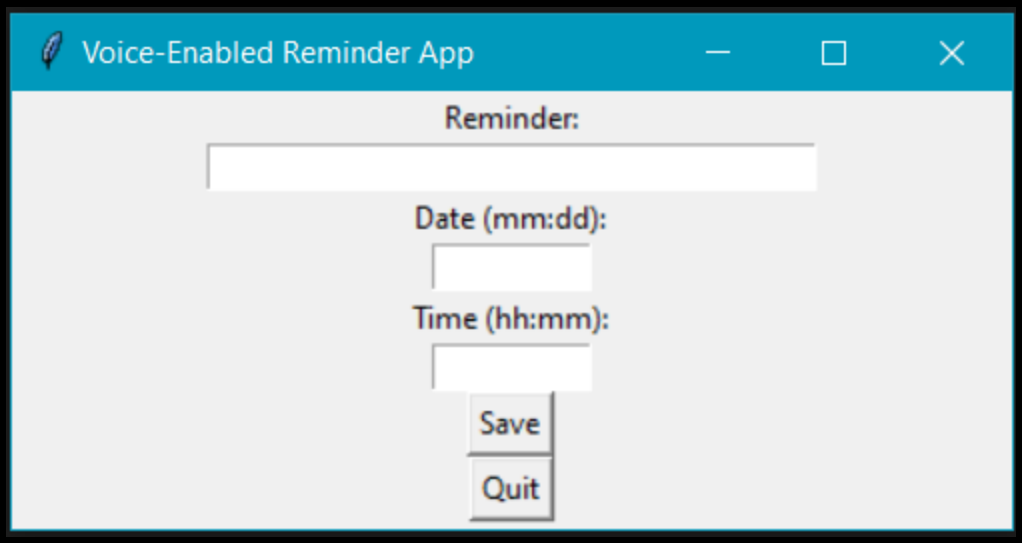
```
import subprocess


def stop_first_app():
    # Write to a file to signal the first app to stop
    with open("stop_signal.txt", "w") as stop_file:
        stop_file.write("stop")


# After stopping, you can restart the first app
def restart_first_app():
    subprocess.Popen(["python", "your_first_app.py"])


print("Stopped and reopened.")
```

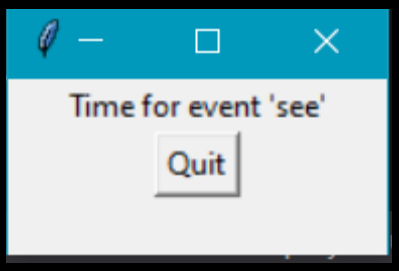
Other needed components:

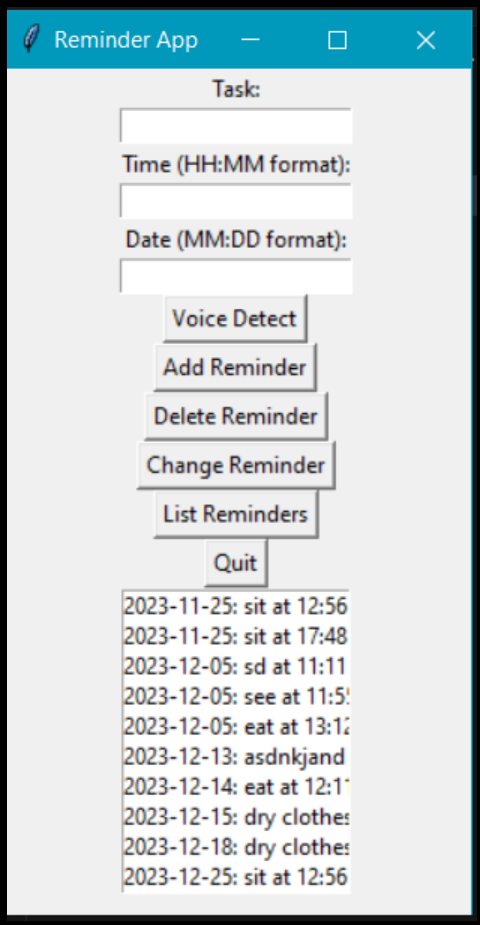
Music(mp3): “sunflower_street”

Txt file: “reminders.txt”

GUI ELEMENTS:







Thankyou:

We extend our heartfelt thanks for the opportunity to embark on the journey of creating a Python project. This experience has been incredibly enriching, allowing us to dive deep into the intricacies of programming, problem-solving, and project management. Our understanding of Python and our skills have been honed in crafting a meaningful project. This endeavor has not only expanded our technical knowledge but has also instilled in us a newfound confidence in our ability to tackle complex challenges. Thank you for fostering an environment that encourages exploration, learning, and the joy of bringing ideas to life through code. We are truly grateful for this empowering experience and look forward to applying the skills and insights gained in future endeavors.

:)