To-Do List Application Documentation

Overview

This application is a **To-Do List** manager with the added functionality of setting reminders. Built with **Tkinter** (Python's standard GUI toolkit), it provides a simple and dark-themed interface where users can add, delete, mark tasks as done, set reminders, and sort tasks. Reminders are displayed as pop-up messages when the scheduled time is reached.

Key Features

- **Add Task**: Allows users to add new tasks to their list.
- **Delete Task**: Removes a selected task.
- **Mark Done**: Marks a task as completed.
- **Clear Completed**: Removes all completed tasks.
- **Sort Tasks**: Sorts tasks, putting completed ones at the end.
- **Set Reminder**: Users can set reminders for tasks, with notifications.

Getting Started

Prerequisites

- **Python 3.x**: Download and install from python.org.
- **Tkinter**: Included with Python. Verify installation by running `python -m tkinter` in your terminal. If a Tkinter window opens, it's installed.

Installation

1. **Download the Code**:

    - Clone the repository or download the project files.

2. **Set Up the Environment (Optional)**:

    - (Optional) Create a virtual environment to isolate dependencies:

    ```
    python -m venv todo-env
    source todo-env/bin/activate  # On Windows: todo-env\Scripts\activate
    ```

3. **Install Requirements (Optional)**

    All required packages are part of the Python standard library:

    - tkinter (included in standard library for GUI)
    - ttk, messagebox (part of tkinter)
    - datetime (included in standard library for date and time operations)
    - threading (included in standard library for threading)

- time (included in standard library for time operations)

Running the Application

1. **Start the Application**:

    - In the terminal, navigate to the project folder and run:

      ```
      python todo_list.py
      ```

2. **Using the App**:

    - Add tasks, set reminders, and manage your list directly in the app window.

Code Structure

Global Variables

Two primary global variables are used to store tasks and manage reminders:

- tasks: A list that stores each task as a string.
- reminders: A dictionary associating each task with a reminder datetime.

```
tasks = []
reminders = {}
```

File Handling Functions

These functions handle reading from and writing to a file (tasks.txt), which stores tasks persistently. This ensures that tasks are not lost when the application is closed.

- **load_tasks():** Reads tasks from tasks.txt (if available) and loads them into the tasks list, displaying each task in the listbox.

```
def load_tasks():
    try:
        with open("tasks.txt", "r") as file:
            for line in file:
                tasks.append(line.strip())
    except FileNotFoundError:
        pass
```

- **save_tasks():** Writes the current list of tasks to tasks.txt, ensuring that data persists between sessions.

```
def save_tasks():
    with open("tasks.txt", "w") as file:
```

```
        for task in tasks:
            file.write(task + "\n")
```

Task Management Functions

These functions control the primary task management operations in the app. They allow users to add, delete, mark as done, sort, and clear tasks.

- **add_task():** Adds a new task from the entry field to the tasks list, updates the listbox, and saves to file.

```
def add_task():
    task = entry.get()
    if task:
        tasks.append(task)
        listbox.insert(tk.END, task)
        entry.delete(0, tk.END)
        save_tasks()
    else:
        messagebox.showwarning("Warning", "You must enter a task.")
```

- **delete_task():** Deletes the selected task from both the tasks list and the listbox, and updates the saved file.

```
def delete_task():
    try:
        task_index = listbox.curselection()[0]
        tasks.pop(task_index)
        listbox.delete(task_index)
        save_tasks()
    except IndexError:
        messagebox.showwarning("Warning", "You must select a task to delete.")
```

- **mark_done():** Toggles a checkmark (✔) for the selected task to indicate completion. Completed tasks are marked visually in the listbox.

```
def mark_done():
    try:
        task_index = listbox.curselection()[0]
        task = tasks[task_index]
        tasks[task_index] = task + " ✔" if "✔" not in task else task.replace("
✔", "")
        listbox.delete(task_index)
        listbox.insert(task_index, tasks[task_index])
        save_tasks()
    except IndexError:
        messagebox.showwarning("Warning", "You must select a task to mark as
done.")
```

- **clear_competed():** Removes all tasks marked as completed (those containing "✔") from the tasks list and updates the listbox.

```
def clear_completed():
    global tasks
    tasks = [task for task in tasks if "✔" not in task]
    listbox.delete(0, tk.END)
    for task in tasks:
        listbox.insert(tk.END, task)
    save_tasks()
```

- **sort_tasks():** Sorts tasks, moving completed tasks (marked ✔) to the end of the list, then updates the listbox and saved file.

```
def sort_tasks():
    tasks.sort(key=lambda x: "✔" in x)
    listbox.delete(0, tk.END)
    for task in tasks:
        listbox.insert(tk.END, task)
    save_tasks()
```

Reminder Functionality

The reminder functionality allows users to set specific date and time reminders for tasks. This feature runs in the background and displays pop-up notifications when a reminder time is reached.

- **set_reminder():** Allows users to set a reminder for a selected task by specifying a date and time. The date and time are entered in "YYYY-MM-DD" and "HH:MM" formats, respectively. A reminder datetime is stored in the reminders dictionary if the input is valid.

```
def set_reminder():
    try:
        task_index = listbox.curselection()[0]
        task = tasks[task_index]
        date_str = date_entry.get()
        time_str = time_entry.get()

        # Convert the date and time to a datetime object
        try:
            reminder_dt = datetime.strptime(f"{date_str} {time_str}", "%Y-%m-%d 
%H:%M")
            reminders[task] = reminder_dt
            date_entry.delete(0, tk.END)
            time_entry.delete(0, tk.END)
            messagebox.showinfo("Reminder Set", f"Reminder set for {task} at 
{reminder_dt}")
```

```
        except ValueError:
            messagebox.showwarning("Invalid Format", "Enter date as YYYY-MM-DD and
time as HH:MM")
    except IndexError:
        messagebox.showwarning("Warning", "You must select a task to set a
reminder.")
```

- **check_reminders():** Runs continuously to check if any tasks have reached their reminder time. If a reminder is due, a notification pop-up appears, and the reminder is removed from the dictionary. This function runs on a daemon thread to avoid blocking the main interface.

```
def check_reminders():
    while True:
        now = datetime.now()
        for task, reminder_time in list(reminders.items()):
            if now >= reminder_time:
                messagebox.showinfo("Reminder", f"Reminder: {task}")
                del reminders[task]
        time.sleep(60)

# Start the reminder thread
reminder_thread = threading.Thread(target=check_reminders, daemon=True)
reminder_thread.start()
```

User Interface Setup

The user interface includes several key elements, each serving a specific function within the application.

1. **Main Window:** Creates the main app window with a dark theme.

```
root = tk.Tk()
root.title("To-Do List")
root.geometry("400x600")
root.configure(bg="#1E1E1E")  # Dark background
```

2. **Listbox:** Displays tasks. Users can select tasks for editing or deleting.

```
listbox = tk.Listbox(frame, width=40, height=15, font=("Arial", 14), bg="#252526",
fg="white", selectbackground="#3B3B3B", bd=0)
listbox.pack(pady=10)
```

3. **Task Entry Field:** Allows users to type new tasks for addition to the list.

```
entry = tk.Entry(root, width=35, font=("Arial", 14), bg="#252526", fg="white",
insertbackground="white", bd=0)
```

```
entry.pack(pady=5)
```

4. **Date and Time Entry Fields:** Users enter the reminder date and time in these fields.

```
date_label = tk.Label(root, text="Reminder Date (YYYY-MM-DD)", font=("Arial", 12),
bg="#1E1E1E", fg="white")
date_label.pack(pady=(10, 0))
date_entry = tk.Entry(root, width=35, font=("Arial", 14), bg="#252526",
fg="white", insertbackground="white", bd=0)
date_entry.pack(pady=5)

time_label = tk.Label(root, text="Reminder Time (HH:MM)", font=("Arial", 12),
bg="#1E1E1E", fg="white")
time_label.pack(pady=(10, 0))
time_entry = tk.Entry(root, width=35, font=("Arial", 14), bg="#252526",
fg="white", insertbackground="white", bd=0)
time_entry.pack(pady=5)
```

5. **Buttons:**

   - **Add Task:** Adds a new task.

   ```
   add_button = ttk.Button(button_frame, text="Add Task", style="Add.TButton",
   command=add_task)
   add_button.grid(row=0, column=0, padx=5)
   ```

   - **Mark Done:** Marks the selected task as completed.

   ```
   done_button = ttk.Button(button_frame, text="Mark Done",
   style="Done.TButton", command=mark_done)
   done_button.grid(row=0, column=1, padx=5)
   ```

   - **Delete Task:** Deletes the selected task.

   ```
   delete_button = ttk.Button(button_frame, text="Delete Task",
   style="Delete.TButton", command=delete_task)
   delete_button.grid(row=0, column=2, padx=5)
   ```

   - **Clear Completed:** Clears all completed tasks.

   ```
   clear_button = ttk.Button(bottom_button_frame, text="Clear Completed",
   style="TButton", command=clear_completed)
   clear_button.grid(row=0, column=0, padx=10)
   ```

- **Sort Tasks:** Sorts tasks, with completed ones at the bottom.

```
sort_button = ttk.Button(bottom_button_frame, text="Sort Tasks",
style="TButton", command=sort_tasks)
sort_button.grid(row=0, column=1, padx=10)
```

- **Set Reminder:** Sets a reminder for the selected task.

```
reminder_button = ttk.Button(bottom_button_frame, text="Set Reminder",
style="TButton", command=set_reminder)
reminder_button.grid(row=0, column=2, padx=10)
```

Initial Setup

Loads existing tasks into the listbox when the app starts.

```
load_tasks()
for task in tasks:
    listbox.insert(tk.END, task)

root.mainloop()
```

Usage Examples

Here are some examples of common tasks you can perform within the To-Do List application:

1. Adding a Task

1. Type the task description into the entry field at the bottom of the app.
2. Click the **Add Task** button.
3. The task will appear in the list above.

2. Deleting a Task

1. Select the task you want to delete from the list by clicking on it.
2. Click the **Delete Task** button.
3. The selected task will be removed from the list.

3. Marking a Task as Completed

1. Select the task you want to mark as completed.
2. Click the **Mark Done** button.
3. A checkmark (✔) will appear next to the task, indicating that it’s completed.

4. Clearing Completed Tasks

1. Click the **Clear Completed** button.
2. All tasks marked with a checkmark (✔) will be removed from the list.

5. Setting a Reminder

1. Select the task for which you want to set a reminder.
2. Enter the reminder date in the **Reminder Date** field (format: YYYY-MM-DD).
3. Enter the reminder time in the **Reminder Time** field (format: HH:MM).
4. Click the **Set Reminder** button.
5. A pop-up notification will appear when the scheduled time arrives.

6. Sorting Tasks

1. Click the **Sort Tasks** button.
2. The tasks will be reordered, moving completed tasks to the bottom of the list.

Known Issues & Troubleshooting

Here are some common issues users may encounter while using the To-Do List application, along with suggested solutions:

- **Tasks Not Saving**:
  - Ensure that the `tasks.txt` file exists in the same directory as the script. If it doesn't, create a blank `tasks.txt` file manually.
  - Check file permissions to ensure the app has permission to read from and write to `tasks.txt`.
- **Reminder Notifications Not Working**:
  - Make sure the date and time for the reminder are entered in the correct format (`YYYY-MM-DD` for date and `HH:MM` for time).
  - If notifications still do not appear, check if the thread running reminders is being blocked by other processes on your system.
- **UI Freezes or Lags**:
  - This can sometimes happen if the application is running on an older system. Try closing other applications to free up system resources.
  - Avoid setting too many reminders simultaneously, as the app's performance can be affected.
- **App Won't Start**:
  - Verify that Python and Tkinter are installed correctly. Run `python -m tkinter` to check if Tkinter is installed.
  - Ensure that you are using Python 3.x, as the app may not work with older versions of Python.

Future Improvements

Here are some planned features and enhancements that could improve the To-Do List application:

- **Recurring Reminders**: Adding support for recurring reminders (e.g., daily, weekly) to help manage repeating tasks.
- **Task Categories**: Allowing users to categorize tasks (e.g., Work, Personal) for better organization.
- **Customizable Themes**: Offering multiple themes so users can personalize the app’s appearance.
- **Priority Levels**: Allowing users to set priority levels (e.g., High, Medium, Low) to help prioritize tasks.
- **Search Functionality**: Adding a search bar to quickly find tasks in a large list.
- **Improved Notification System**: Enhancing reminders with sound alerts or integration with system notifications.

These improvements are planned to make the application more versatile and user-friendly, helping users better manage their tasks.