Final Project

Group: 002

School clubs' management program

Managed & Presented by:

- Khov Mony
- Lou Leakhena
- Thay Kakada Tepprapey

COCS 111 Class of Spring, 2025

1. Introduction

- This project is a basic computer program with a graphical user interface (GUI) to help manage school clubs.
- It is made using Python and a tool called Tkinter to create windows, buttons, and input boxes.
- It allows users to create clubs, delete clubs, add activities, delete activities, join clubs, see clubs' details, and see member lists.
- All the data is saved automatically in a file using JSON format, so nothing gets lost when the program is closed.

2. Objective

- To provide an easy-to-use interface for both students and club leaders.
- Simplify the process of creating, joining, and managing school clubs.
- Allow leaders to maintain records of members and activities digitally.
- Learn how to use Tkinter, JSON, and apply Python logic practically.

3. Methodology and Concepts Used:

- Python for the main logic.
- Tkinter for GUI components like buttons, frames, and entry fields.
- JSON for saving and loading data persistently.
- To keep the code clean and reusable, we created substitute functions that some of them we can call whenever needed**:**

    - load_data() – loads saved club data from a JSON file
    - save_data() – saves all club information to a file
    - custom_ask_string() – shows a custom pop-up to get user input
    - show_frame(frame) – switches between pages or screens

- Functions and Conditions: To validate inputs and manage workflows.
- Loops and Python's collection data types: To handle multiple clubs, members, clubs' information and activities.

Main Functionalities:

In '**Create New Club**':

- def create(): create a club with input requirements and validations.
- def delete_club(): delete an existing club. (for leader and co leader)
- def add_activity(): add club activities. (for leader and co-leader)

- def delete_activity(): delete a club activity. (for leader and co-leader)

In '**Join Club**':

- def refresh_club_list(): refresh the latest version of clubs that exist
- def show_details(): show club's details
- def join_selected_club(): join the club
- def view_members(): views all members in the club (for leader and co-leader)

4. Demo/flowcharts:

Before explaining the main features of the app, we'll first talk about the initialization and helper functions that are important for keeping the app running smoothly and safely.

We start by importing necessary libraries.

```
import tkinter as tk
from tkinter import messagebox
import json
import os
```

'

Explanation:

- tkinter: Used to build the GUI (Graphical User Interface).
- messagebox: Used to show pop-up alerts (like warnings or messages).
- json: Used to save and load data (like club details) in a readable file format.
- os: Helps check if a file exists before loading it.

Then we initialize with an empty dictionary.

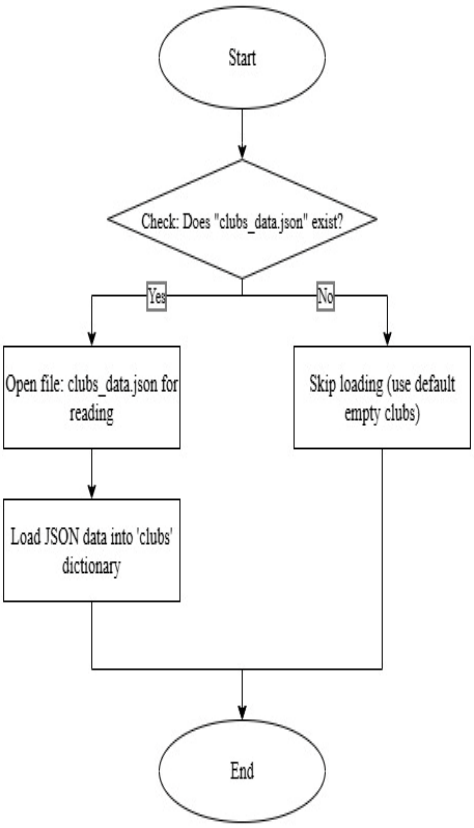
```
clubs = {}
```

Explanation:

- This empty dictionary will store all club information like names, leaders, members, and activities.
- It will be filled later when the user creates or loads clubs.

Next, loading and saving data.

```
def load_data():
    global clubs
    if os.path.exists("clubs_data.json"):
        with open("clubs_data.json", "r") as f:
            clubs = json.load(f)
```



Explanation:

def load_data():
Defines a reusable function to load saved club data.

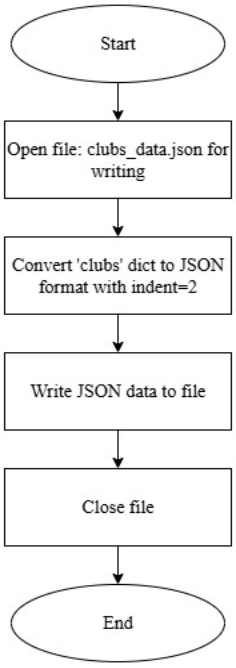
global clubs:
Refers to the global clubs dictionary to update it directly, and changes affect the dictionary used throughout the program.

if os.path.exists("clubs_data.json"):
Checks if the data file exists to avoid errors on first use.

with open("clubs_data.json", "r") as f:
Opens the file in read mode, and with the "with" statement, ensures it closes properly after reading.

clubs = json.load(f):
Loads JSON data from the file and updates the clubs dictionary.

```
def save_data():
    with open("clubs_data.json", "w") as f:
        json.dump(clubs, f, indent=2)
```



Explanation:

def save_data():
Defines a function called save_data() used to save club data to a file.

with open("clubs_data.json", "w") as f:
Opens (or creates) the file in write mode and ensures it closes safely after writing.

json.dump(clubs, f, indent=2)
Converts the clubs dictionary to JSON format and writes it to the file **f** with indentation for easy reading.

Overall, this part:

1. load_data() = Memory Recall
   - When the app starts, load_data() checks if there's a saved file (clubs_data.json) and loads all club data into the program.
   - Why it's useful: Without this, the app would forget everything (members, clubs, activities) every time you close it.
2. save_data() = Auto-Save function
   - Every time someone creates a club, joins a club, or edits anything, save_data() saves the changes to the file.
   - We call this function everywhere to ensure no data is lost if the app crashes or closes. Like saving progress in our app, but automatic and constant.

Let's move on to the next part, which creates a custom input dialog for gathering user input (e.g., names, IDs, reasons) throughout the app.

```
def custom_ask_string(title, prompt):
    dialog = tk.Toplevel(root, bg="#d1d39d")
    dialog.title(title)
    dialog.geometry("400x150+550+280")
    dialog.resizable(True, True)
    tk.Label(dialog, text=prompt, padx=10, pady=5, bg="#d1d39d").pack(anchor="w")

    entry = tk.Entry(dialog, width=50)
    entry.pack(padx=10, pady=5, fill=tk.X)
    result = [None]

    def on_ok():
        result[0] = entry.get()
        dialog.destroy()

    def on_cancel():
        result[0] = None
        dialog.destroy()

    btn_frame = tk.Frame(dialog, bg="#d1d39d")
    btn_frame.pack(pady=10)
    tk.Button(btn_frame, text="OK", width=10, command=on_ok, bg="#63a5e8").pack(side=tk.LEFT, padx=5)
    tk.Button(btn_frame, text="Cancel", width=10, command=on_cancel, bg="#d76a6c").pack(side=tk.LEFT, padx=5)
    dialog.protocol("WM_DELETE_WINDOW", on_cancel)
    dialog.wait_window()
    return result[0]
```

- **def custom_ask_string(title, prompt):** Defines a reusable function that creates a custom input dialog using a title and prompt to standardize input collection.
- **dialog = tk.Toplevel(root, bg):** Creates a popup window with a specific background color, separate from the main app.
- **dialog.title(title)** Sets the window title to match the purpose of the input (e.g., "Name").
- **dialog.geometry:** Positions and sizes the dialog window for a centered, user-friendly view.
- **dialog.resizable(True, True)**: Allows users to resize the dialog, improving flexibility on different screens.
- **tk.Label(dialog, text=prompt, ...).pack(anchor="w")**: Displays a prompt label to guide the user on what to enter.
- **entry = tk.Entry(dialog, width=50)**: Creates a text field for user input.
- **entry.pack(padx=10, pady=5, fill=tk.X)**: Adds spacing and stretches the entry box for clean layout.

- **result = [None]**: Initializes a list to hold the user's input for later retrieval.
- **def on_ok():** Defines a function to save the input and close the dialog when OK is clicked.
- **def on_cancel():** Defines a function to cancel the input and close the dialog cleanly.
- **btn_frame = tk.Frame(dialog, bg)** Creates a frame to hold the OK and Cancel buttons.
- **btn_frame.pack(pady=10)**: Adds vertical padding to position the button frame neatly.
- **tk.Button(..., text="OK", command=on_ok, ...).pack(...)**: Creates an OK button that triggers input saving when clicked.
- **tk.Button(..., text="Cancel", command=on_cancel, ...).pack(...)**: Creates a Cancel button to discard input when clicked.
- **dialog.protocol("WM_DELETE_WINDOW", on_cancel)**: Makes the window's close (X) button behave like Cancel to avoid errors.
- **dialog.wait_window()**: Freezes the app until the user finishes and closes the dialog.
- **return result[0]**: Returns the user's input value or None if canceled.

Why this function is useful

Example feature:

Unified input system

- Replaces Python's default askstring() with a custom, styled dialog.
- Ensures consistent UI across the app (same fonts, colors, and layout).

User control

- Handles cancellation (Cancel/X button) and empty input gracefully.
- Returns None when canceled, allowing functions to stop safely.

Reusability

- It's reusable and used throughout the app whenever we need quick text input from the user.

Finally, the show_frame function takes a frame (a section or screen of the app) as input.

```
def show_frame(frame):
    frame.tkraise()
```

It uses frame.tkraise() to bring that frame to the front, making it visible on the window.

This small function controls which screen is currently shown by raising the chosen frame above the others. It allows our app to switch smoothly between different pages without opening new windows

Now we're getting to the feature part of the app.

First, creates the main application window.

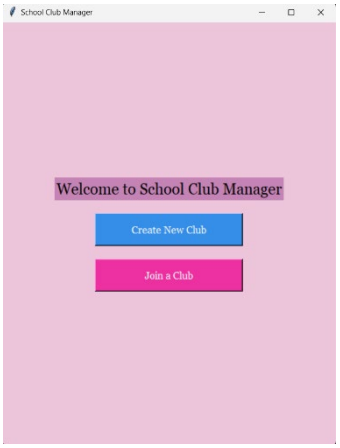
```
root = tk.Tk()
root.title("School Club Manager")
root.geometry("530x650")
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
root.configure(bg="#edc5db")
```

This block sets up the main window of the Tkinter app by creating the root window, setting its title (shown on the window bar), size, and background color, and configuring it to resize smoothly.

It builds the GUI foundation, allowing creators to add and display all other widgets, like buttons and text, just like building the frame of a house before adding rooms.

Then, creates a **main menu frame** centered with a welcome label and two navigation buttons.

```
main_menu = tk.Frame(root, bg="#edc5db")
main_menu.grid(row=0, column=0, sticky="nsew")
main_center = tk.Frame(main_menu, bg="#edc5db")
main_center.place(relx=0.5, rely=0.5, anchor="center")

tk.Label(main_center, text="Welcome to School Club Manager", font=("Georgia", 18), bg="#c584b6").pack(pady=10)

tk.Button(main_center, text="Create New Club", height=2, width=25, font=("Georgia", 12), bg="#358ee8", fg="white",
          command=lambda: show_frame(create_club)).pack(pady=10)

tk.Button(main_center, text="Join a Club", height=2, width=25, font=("Georgia", 12), bg="#ed30a1", fg="white",
          command=lambda: [refresh_club_list(), show_frame(join_club)]).pack(pady=10)
```



Explanation:

1. Create Main Menu Frame

Creates a full-screen frame (main_menu) with a pink background. Uses grid() to place it in row 0, column 0 of the main window. sticky="nsew" makes it expand to fill the whole window.

2. Center Content Frame

Creates a smaller frame (main_center) inside main_menu for the content.
Uses place() to center it:

- relx=0.5, rely=0.5 → center of the parent frame
- anchor="center" → aligns the frame exactly to the center

3. Welcome Message

Adds a large title label at the top of the screen. Uses a soft purple background to highlight it.
pack(pady=10) adds space above and below the text.

4. "Create New Club" & "Join a Club" Buttons

- Blue "Create New Club" button → Takes the user to the club creation frame.

- Pink "Join Club" button → Updates the club list and opens the join club screen.

Overall, this part creates the home screen of the app. It's the first thing users see. It welcomes them and shows what the app is for. There are two main choices: create a new club or join one. The layout uses grid, place, and pack to keep everything neat and in the center. The buttons link to other parts of the app using show_frame(), like turning pages in a book.

I. Create_club

Now let's move to our first option, which is the Create Club screen. This is the page we navigate to, and it includes entry fields and contains four important functions:

- def create()
- def delete_club()
- def add_activity()
- def delete_activity()

```
create_club = tk.Frame(root, bg="#98bf92")
create_club.grid(row=0, column=0, sticky="nsew")
create_club.columnconfigure(0, weight=1)
create_club.rowconfigure(0, weight=1)

tk.Label(create_club, text="Create a Club", font=("Georgia", 16), bg="#f9fbf8").pack(pady=10)
tk.Label(create_club, text="* Do not add 'club' at the end.", fg="gray", bg="#a2be9d").pack()

entry_name = tk.Entry(create_club, width=55)
entry_desc = tk.Entry(create_club, width=55)
entry_leader = tk.Entry(create_club, width=55)
entry_leader_id = tk.Entry(create_club, width=55)
entry_co_leader = tk.Entry(create_club, width=55)
entry_co_leader_id = tk.Entry(create_club, width=55)
entry_max = tk.Entry(create_club, width=40)

for label, entry in [
    ("Club Name", entry_name),
    ("Description", entry_desc),
    ("Leader Name", entry_leader),
    ("Leader ID", entry_leader_id),
    ("Co-Leader Name", entry_co_leader),
    ("Co-Leader ID", entry_co_leader_id),
    ("Max Members", entry_max)
]:
    tk.Label(create_club, text=label, height=1, width=14, bg="#079E52", fg="#F6F9F7").pack()
    entry.pack()
```

1. Create the create_club Frame

- Creates a new full-screen frame for the "Create Club" section inside the same app window.
- Uses grid() to position it in the main window, same as main_menu.
- columnconfigure and rowconfigure make the frame expandable so content resizes smoothly.

2. Title Label & Naming Tip Label

- Adds a header label at the top of the screen.
- Adds a small note below the title, telling users not to include "club" in the name.

3. Entry Fields

Defines input fields for:

- Club name
- Description
- Leader name & ID
- Co-leader name & ID
- Max members

Each field is assigned to a variable for later access.

5. Add Labels and Entry Fields

- Use a for loop to create labels and pack entries.
- This method keeps code short and clean by handling all labels and entries together.
- Each label is shown above its corresponding text box.
- We also use this loop to prepare this data for use when the user clicks ‘Create Club’ later on.
- pack() adds each widget vertically with default alignment.

This page also contains five buttons, each one calling a specific function when clicked:

1. "Create Club" – Runs the def create() function to add a new club using the data from the entry fields.
2. "Delete Club" – Calls def delete_club() to remove an existing club from the system.
3. "Add Activity to Club" – Runs def add_activity() to add a new activity under the selected club.
4. "Delete Activity" – Calls def delete_activity() to remove an activity from a club.
5. "Back to Menu" – Returns the user to the main menu screen by calling show_frame(main_menu).

1. Create() function

This function handles club creation by collecting input from the GUI fields, validating the data (checking required fields, naming rules, and max members), adding the club to the clubs dictionary, and saving the updated data.

Key Concepts & Flow Alignment

1. Input Collection

Retrieves the club's name from the GUI input field.

2. Validation Logic

- Empty name check: Ensures the club's name isn't blank.
- Unique club check:
  - Creates a unique key ("club name club" in lowercase) to prevent duplicates.
  - Checks if the club already exists.
- Naming rule check: Prevents names ending with "club".
- Required fields check: Validates all mandatory fields (description, leaders, co-leaders, IDs, max-members).
- Max Members Validation: Ensures max_members is an integer ≥10.

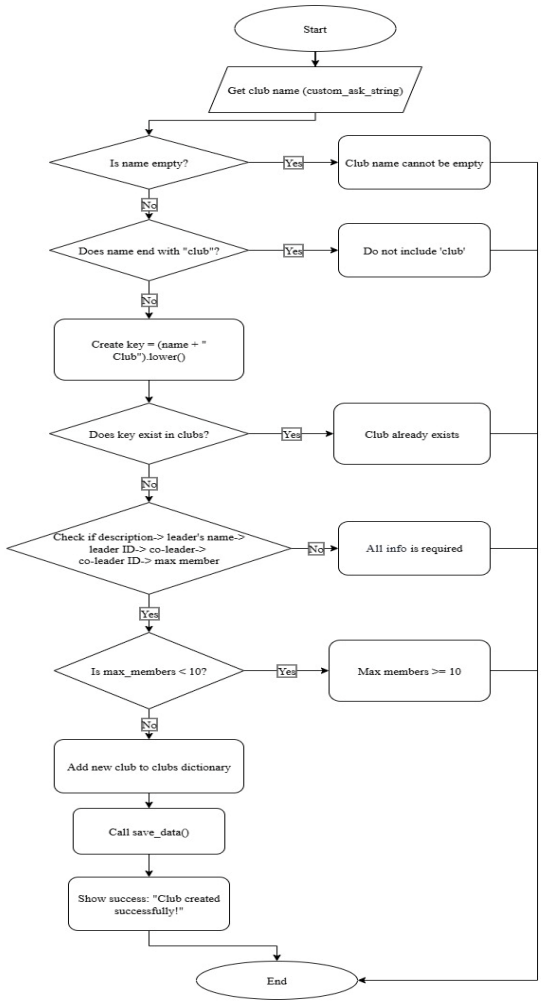
3. Data Storage

- Stores club data in the clubs dictionary.
- Calls save_data() to persist changes.

4. User Feedback

- Shows a success message.
- Clears input fields for the next operation.

Flowchart




```
def create():
    name = entry_name.get().strip()
    key = (name + " Club").lower()
    if not name:
        messagebox.showwarning("Missing", "Club name cannot be empty.")
        return
    if key in clubs:
        messagebox.showwarning("Exists", "Club already exists.")
        return
    if name.lower().endswith("club"):
        messagebox.showwarning("Naming Tip", "Please do not include the word 'club' in the name.")
        return
    if not entry_desc.get().strip():
        messagebox.showwarning("Missing Info", "Please enter a description.")
        return
    if not entry_leader.get().strip():
        messagebox.showwarning("Missing Info", "Please enter the leader's name.")
        return
    if not entry_leader_id.get().strip():
        messagebox.showwarning("Missing Info", "Please enter the leader's ID.")
        return
    if not entry_co_leader.get().strip():
        messagebox.showwarning("Missing Info", "Please enter the co-leader's name.")
        return
    if not entry_co_leader_id.get().strip():
        messagebox.showwarning("Missing Info", "Please enter the co-leader's ID.")
        return
    try:
        max_members = int(entry_max.get().strip())
        if max_members < 10:
            raise ValueError
    except ValueError:
        messagebox.showerror("Invalid", "Please enter a number >= 10 for max members.")
        return

    clubs[key] = {
        "name": name + " Club",
        "description": entry_desc.get().strip(),
        "leader": entry_leader.get().strip(),
        "leader_id": entry_leader_id.get().strip(),
        "co_leader": entry_co_leader.get().strip(),
        "co_leader_id": entry_co_leader_id.get().strip(),
        "max_members": max_members,
        "members": [],
        "activities": []
    }
    save_data()
    messagebox.showinfo("Created", f"'{name}' club created successfully!")
    for e in [entry_name, entry_desc, entry_leader, entry_leader_id, entry_co_leader, entry_co_leader_id, entry_max]:
        e.delete(0, tk.END)
```

2. delete_club() function

This function handles club deletion by collecting input from the user (club name and ID), validating access to ensure only leaders or co-leaders can delete, confirming the deletion, removing the club from the clubs dictionary, and then updating the UI and saving the changes.

Key Concepts & Flow Alignment

1. Input collection:

- Retrieves the club's name from the user.

2. Validation logic:

- Empty Name Check: Skips further steps if no name is entered
- Unique club check:
  - ➢ Creates a unique key ("club name club" in lowercase) to prevent confusion.
  - ➢ Checks if the club already exists.
- UID Collection: Gets the user's ID for access validation.
- Access Validation: Checks if the user is authorized to delete the club.

3. Deletion confirmation

- Asks the user to confirm deletion.

4. Data deletion & persistence

- Removes the club from the clubs dictionary.
- Saves the updated data to clubs dictionary.

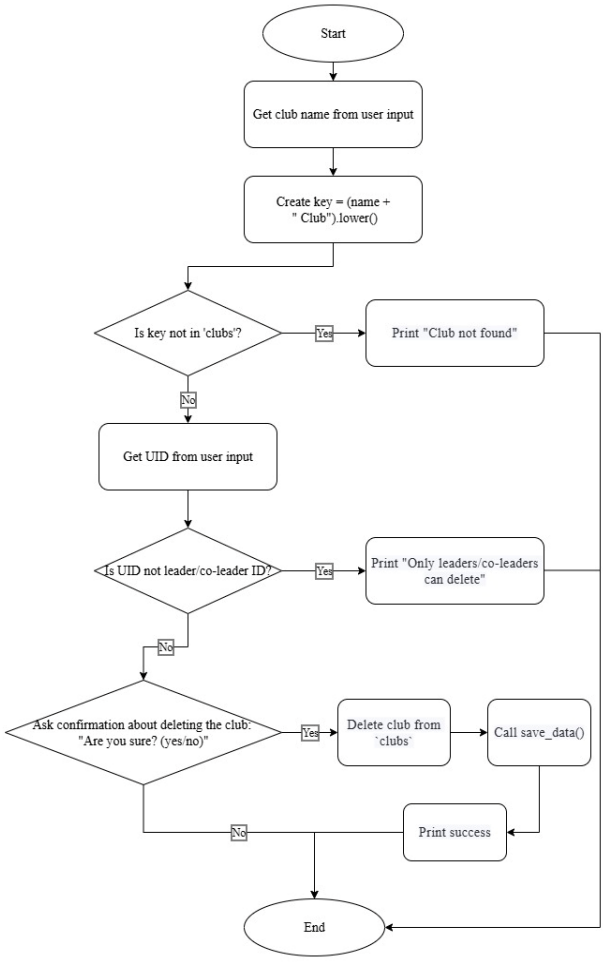
5. Ui updates

- Refreshes the club list.
- Clears the details text box.
- Shows a success message.

Key Python Concepts:

- "del" Statement: Used to remove club and deletes a key-value pair from dictionary.
- "if" Statements: Used to control program flow based on conditions.

Flowchart




```python
def delete_club():
    club_name = custom_ask_string("Delete Club", "Enter club name to delete:")
    if not club_name:
        return
    key = (club_name.strip() + " Club").lower()
    if key not in clubs:
        messagebox.showerror("Error", "Club not found.")
        return
    uid = custom_ask_string("Your ID", "Enter your leader/co-leader ID:")
    if not uid:
        return
    if uid not in (clubs[key]['leader_id'], clubs[key]['co_leader_id']):
        messagebox.showerror("Access Denied", "Only leaders/co-leaders can delete the club.")
        return
    if messagebox.askyesno("Confirm", f"Are you sure you want to delete '{club_name}' club?"):
        del clubs[key]
        save_data()
        refresh_club_list()
        details_text.config(state="normal")
        details_text.delete(1.0, tk.END)
        details_text.config(state="disabled")
        messagebox.showinfo("Deleted", f"'{club_name}' has been deleted.")
```

3. add_activity() function

This function allows club leaders or co-leaders to add activities by first checking that the club exists, confirming the user is authorized, ensuring all required activity details are provided, and then saving the activity persistently.

1. Input collection:

- Retrieves the club's name from the user.

2. Validation logic:

- Empty Name Check: Skips further steps if no name is entered
- Unique club check:
  - ➢ Creates a unique key ("club name club" in lowercase) to prevent confusion.
  - ➢ Checks if the club already exists.
- UID Collection: Gets the user's ID for access validation.
- Access Validation: Checks if the user is authorized to add activity to club.

3. Activity Details Validation

- Collects and validates activity details (activity title, date & place, description).

4. Add Activity to Dictionary

- Adds the activity to the club's activities list.

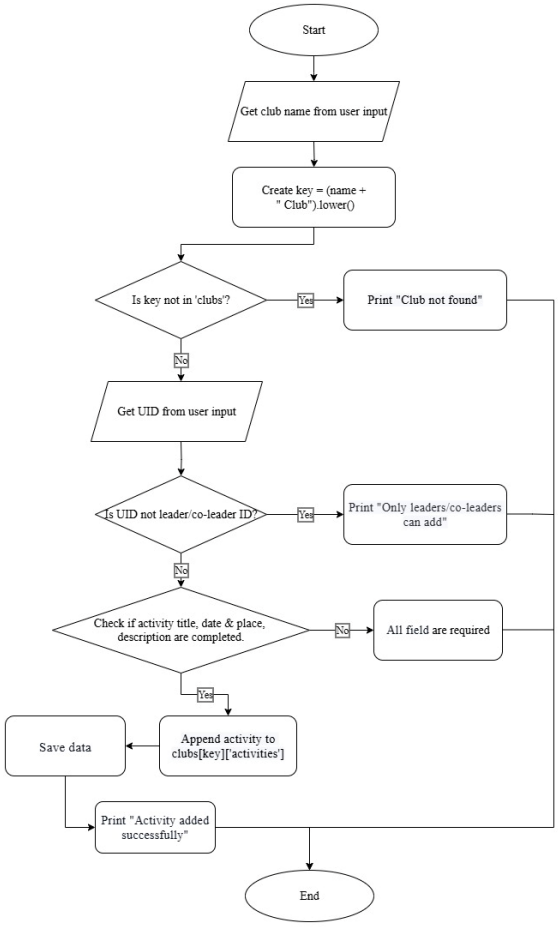
5. Persistence & Feedback

- Saves changes to disk and informs the user.

Python Concepts Used

- Dictionaries: Store and organize club data, including nested activities.
- String Manipulation: Use strip(), lower(), and concatenation to create unique keys.
- Conditionals: Check input validity and user access with 'if' statements.
- List Operations: Use append() to add activities to a club's list.
- Functions: Use custom_ask_string() and save_data() for reusable tasks.

Flowchart




```python
def add_activity():
    club_name = custom_ask_string("Club Name", "Enter the name of your club (no 'club' at the end):")
    if not club_name:
        return
    key = (club_name.strip() + " Club").lower()
    if key not in clubs:
        messagebox.showerror("Not Found", "Club not found.")
        return
    user_id = custom_ask_string("Leader/Co-Leader ID", "Enter your student ID:")
    if not user_id:
        return
    if user_id not in (clubs[key]['leader_id'], clubs[key]['co_leader_id']):
        messagebox.showerror("Access Denied", "Only the leader or co-leader can add activities.")
        return
    title = custom_ask_string("Activity Title", "Enter activity title:").strip()
    date = custom_ask_string("Activity Date and place", "Enter date and place (e.g., 2025-08-01(Room 5)):").strip()
    desc = custom_ask_string("Activity Description", "Enter activity description:").strip()
    if not title or not date or not desc:
        messagebox.showwarning("Missing Info", "All fields (title, date/place, description) are required.")
        return

    clubs[key]["activities"].append({
        "title": title,
        "date & place": date,
        "description": desc})
    save_data()
    messagebox.showinfo("Added", f"Activity added to {clubs[key]['name']}.")
```

4. delete_activity() function

This function allows club leaders or co-leaders to delete activities by checking that the club exists, verifying the user's authority, confirming the activity exists, and then saving the changes persistently.

1. Input collection:

Retrieves the club's name from the user.

2. Validation logic:

- Empty Name Check: Skips further steps if no name is entered
- Unique club check:
  - ➢ Creates a unique key ("club name club" in lowercase) to prevent confusion.
  - ➢ Checks if the club already exists.
- UID Collection: Gets the user's ID for access validation.
- Access Validation: Checks if the user is authorized to delete activity from club.

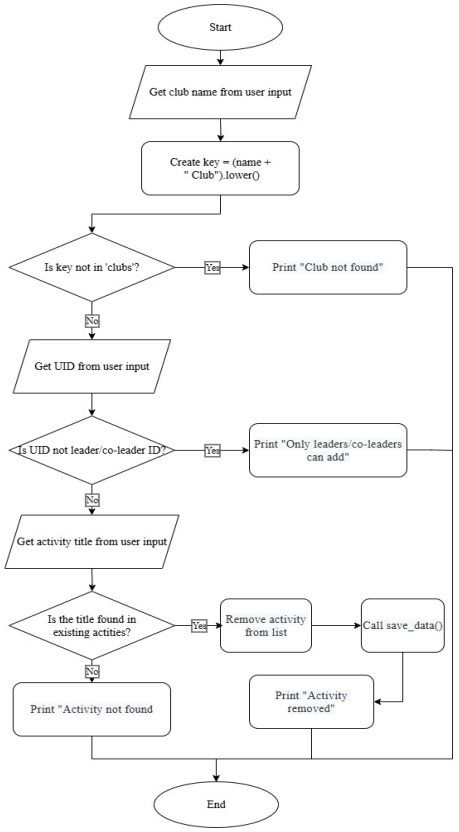
3. Activity title input, activity existence check & deletion

- Collects the title of the activity to delete.
- Searches for the activity by title (case-insensitive).
- Deletes it if found and saves changes.

Python Concepts Used

- Dictionaries: Store club data and access activities using keys.
- String Manipulation: Use strip() and lower() for consistent comparisons.
- for loop & Conditionals (if): Validates input, checks access, and ensures activity exists.
- List Operations: Use remove() to delete an activity from a club.
- Functions: Reuse logic with custom_ask_string() and save_data().

Flowchart




```python
def delete_activity():
    club_name = custom_ask_string("Club Name", "Enter your club name:")
    if not club_name:
        return
    key = (club_name.strip() + " Club").lower()
    if key not in clubs:
        messagebox.showerror("Not Found", "Club not found.")
        return
    uid = custom_ask_string("Your ID", "Enter your leader/co-leader ID:")
    if not uid:
        return
    if uid not in (clubs[key]['leader_id'], clubs[key]['co_leader_id']):
        messagebox.showerror("Access Denied", "Only leaders/co-leaders can delete activities.")
        return
    title = custom_ask_string("Activity Title", "Enter the title of activity to delete:")
    c = clubs[key]
    for a in c['activities']:
        if a['title'].lower() == title.lower():
            c['activities'].remove(a)
            save_data()
            messagebox.showinfo("Deleted", f"Activity '{title}' removed.")
            return
    messagebox.showinfo("Not Found", "Activity not found.")
```

II. join_club

Now let's move to our second option, which is the Join Club screen. This is the page we next navigate to, and it contains four significant functions:

- def refresh_club_list()
- def show_details()
- def join_selected_club()
- def view_members()

User Interface (Tkinter)

- Frame, Listbox, Text, Button, and Label build the visual layout.
- Scrollbars and layout tools (pack, place, grid) keep it organized.

1. Create the create_club Frame

- Creates a new full-screen frame for the "Join Club" section inside the main root window.
- Uses grid() to position it in the main window, same as main_menu.
- columnconfigure and rowconfigure make the frame expandable so content resizes smoothly.

2. Title Label

- Adds a header label (Join Club) at the top of the screen.

3. Top Section: Clubs List & Button

- Left : A teal-labeled section for "Available Clubs".
- Right : A button for leaders/co-leaders to view members (calls view_members()).

4. Listbox for Available Clubs

- Displays a scrollable list of clubs.
- Vertical/horizontal scrollbars allow viewing long lists or wide club names.

5. Club Details Text Area

- Shows club details (description, leaders, members, activities) when a club is selected and "View club details" is clicked.
- Read-only text area with vertical scrolling.

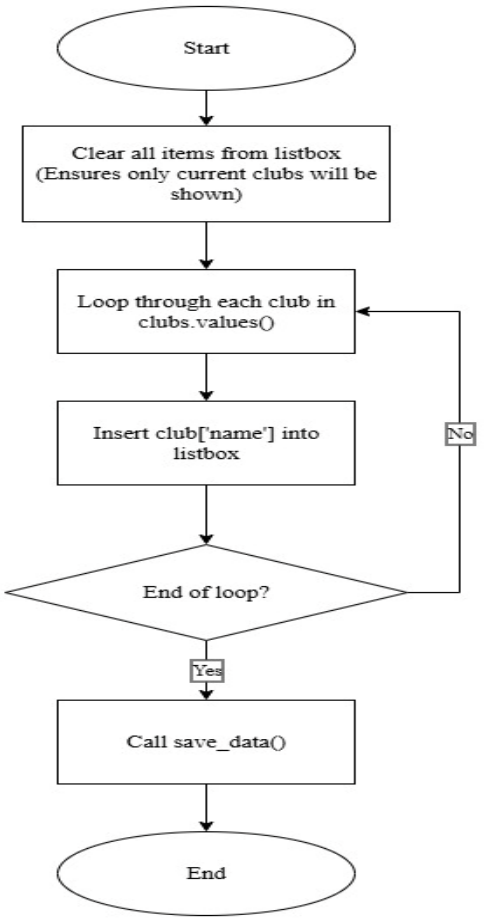
6. Action Buttons

Select a club in the listbox.

- Click "Refresh List" → Updates the listbox with current available clubs.
- Click "View club details" → show_details() fills the text area with club info.
- Click "Join Club" → join_selected_club() prompts for name, ID, and reason (if valid, adds to club).
- Click "Back to Menu" → Returns to the main menu.

1. refresh_club_list() function

Flowchart



This function updates the club listbox to show the latest club names from the clubs dictionary. It ensures the UI reflects real-time data after actions like creating or deleting clubs.

Key Concepts & Flow Alignment

1. Clear Listbox

- Removes all existing items from the listbox.
- 0 means the first item; tk.END means the last item.

2. Loop Through Clubs

- Loops through all the club dictionaries stored in the global clubs dictionary.
- Inserts each club's name into the listbox.

3. Call save_data()

- Saves the current state of clubs to a file.

Code concepts used

Listbox Operations

- delete(0, tk.END): Clears all items.
- insert(tk.END, ...): Adds new items to the end.

Dictionary Iteration

- clubs.values(): Loops through all club entries in the clubs dictionary.

Data Persistence

- save_data(): Ensures changes are saved to disk.

```
def refresh_club_list():
    listbox.delete(0, tk.END)
    for club in clubs.values():
        listbox.insert(tk.END, club['name'])
    save_data()
```

2. show_details() function

This function displays detailed information about a selected club, including its description, leaders, current and maximum member count, and any listed activities.

Key Concepts & Flow Alignment

1. Check if a Club is Selected

- Checks if a club is selected in the listbox.
- Clears the details text box if no selection.

2. Get Club Name & Create Key

- Retrieves the selected club's name.
- Creates a lowercase key ("name" → "name club").

3. Look Up the Club in clubs Dictionary

- Ensures the club exists in the clubs dictionary.
- Fetches the club's data.

4. Display Club Info

- Prepares the text box for writing.
- Displays core club details (description, leaders, member count).

5. Check for Activities

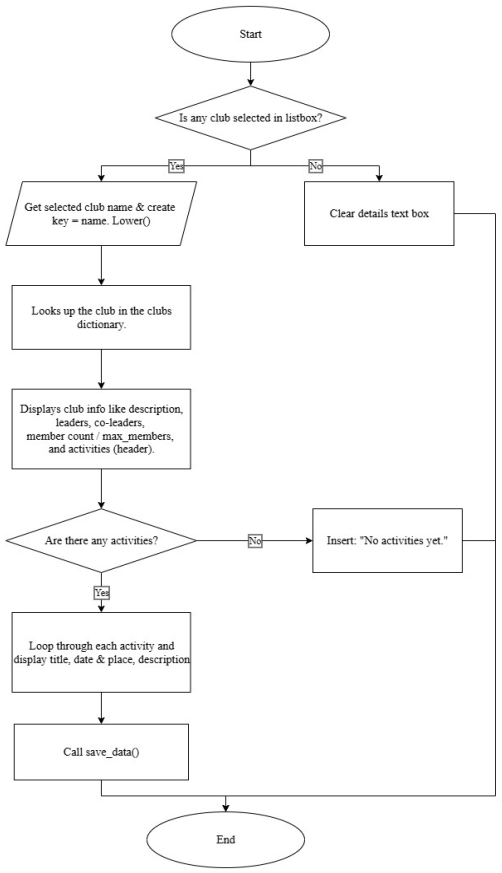
- Checks if the club has activities.
- Loops through and displays each activity if present.

6. Save Data: Disables the text box after updating. Saves changes to disk

Python Concepts Used

- String Manipulation: strip(), lower() for consistent key formatting.
- Dictionary Access: c = clubs[key] retrieves club data.
- Conditional Logic (if): Checks for selection, club existence, and activities.
- Text Widget Operations: config(state="normal/disabled"), delete(...), insert(...) to manage the details text box.

Flowchart




```python
def show_details():
    sel = listbox.curselection()
    if not sel:
        details_text.config(state="normal")
        details_text.delete(1.0, tk.END)
        details_text.config(state="disabled")
        return

    name = listbox.get(sel[0]).strip()
    key = name.lower()
    if key not in clubs:
        messagebox.showerror("Error", f"Club '{name}' not found.\nKey tried: '{key}'")
        return

    c = clubs[key]
    details_text.config(state="normal")
    details_text.delete(1.0, tk.END)

    details_text.insert(tk.END, f"Description: {c['description']}\n")
    details_text.insert(tk.END, f"Leader: {c['leader']}\n")
    details_text.insert(tk.END, f"Co-Leader: {c['co_leader']}\n")
    details_text.insert(tk.END, f"Members: {len(c['members'])}/{c['max_members']}\n")
    details_text.insert(tk.END, "Activities:\n")

    if c['activities']:
        for activity in c['activities']:
            details_text.insert(tk.END, f"Title: {activity['title']} | Date & Place: {activity['date & place']} | Description: {activity['description']}\n")
    else:
        details_text.insert(tk.END, "No activities yet.\n")

    details_text.config(state="disabled")
    save_data()
```

3. join_selected_club() function

This function allows users to join a selected club by collecting their name, student ID (SID), and reason for joining. It ensures a club is selected, checks that the club isn't full, validates the inputs, avoids duplicate SIDs, and saves the changes using save_data().

Key Concepts & Flow Alignment

1. Check Club Selection: Ensures a club is selected before proceeding.

2. Get Club Key & Check Fullness

- Retrieves the club's key and data.
- Checks if the club has reached its member limit.

3. Collect Name

- Prompts for the user's name with retry logic.
- Handles cancellation (None) and empty input.

4. Collect SID

- Prompts for the user's SID with retry logic.
- Checks for SID duplicates and handles cancellation, empty input.

5. Collect Reason

- Prompts for the user's reason with retry logic.
- Handles cancellation and empty input.

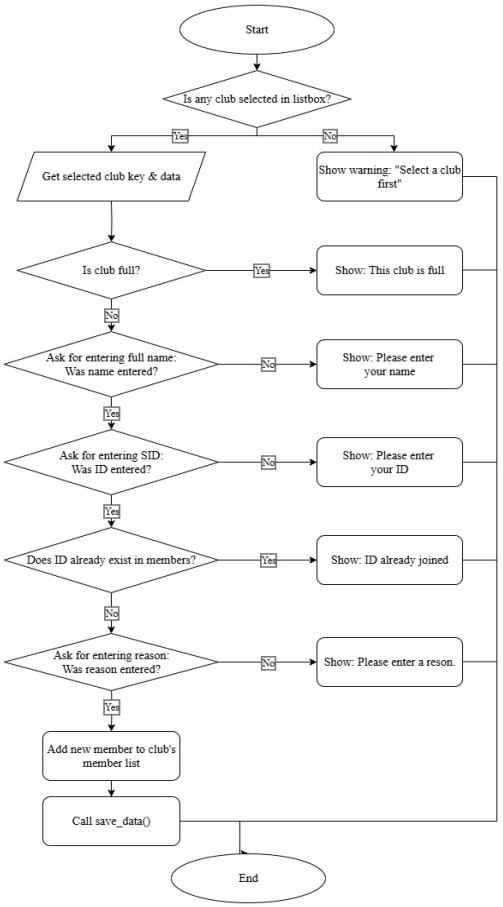
6. Add Member & Save Data

- Adds the new member to the club's list.
- Shows a success message and saves changes.

Python Concepts Used

- Loops with Conditions: while True loops ensure valid input before proceeding.
- List Comprehensions: any(m['id'] == sid for m in c['members']) checks for duplicate IDs.
- Error Handling: messagebox provides feedback for invalid inputs or cancellation

Flowchart

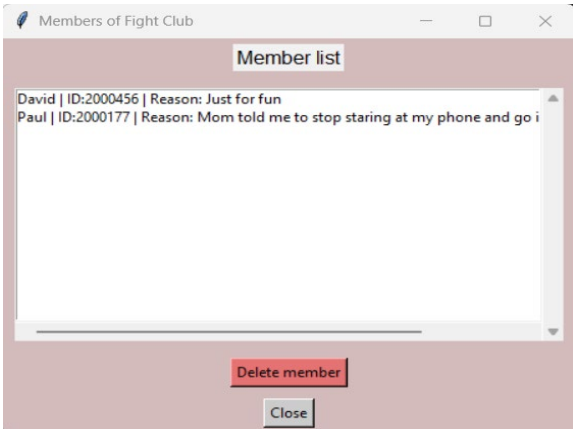



```python
def join_selected_club():
    sel = listbox.curselection()
    if not sel:
        messagebox.showwarning("Select", "Select a club first.")
        return

    key = listbox.get(sel[0]).lower()
    c = clubs[key]
    if len(c['members']) >= c['max_members']:
        messagebox.showinfo("Full", "This club is full.")
        return

    while True:
        name = custom_ask_string("Name", "Enter your full name:")
        if name is None:
            messagebox.showinfo("Canceled", "Joining canceled.")
            return
        if name.strip() == "":
            messagebox.showwarning("Input Required", "Please enter your name to continue.")
        else:
            break

    while True:
        sid = custom_ask_string("ID", "Enter your student ID:")
        if sid is None:
            messagebox.showinfo("Canceled", "Joining canceled.")
            return
        if sid.strip() == "":
            messagebox.showwarning("Input Required", "Please enter your student ID.")
        elif any(m['id'] == sid for m in c['members']):
            messagebox.showinfo("Exists", "This student ID already joined.")
        else:
            break

    while True:
        reason = custom_ask_string("Purpose", "Why join this club?")
        if reason is None:
            messagebox.showinfo("Canceled", "Joining canceled.")
            return
        if reason.strip() == "":
            messagebox.showwarning("Input Required", "Please enter a reason.")
        else:
            break
    c['members'].append({"name": name, "id": sid, "purpose": reason})
    messagebox.showinfo("Joined", f"{name} joined {c['name']}!")
    save_data()
```

4. view_members() function

This function allows leaders/co-leaders to view and manage members of a club. It includes:

- Input validation (club existence, access control).
- Displaying members in a scrollable list.
- A nested delete_selected() function for removing members.



1. Input Collection & Validation: Retrieves the club's name and creates a unique key, and checks if the club exists.

2. Access Control: Validates the user's ID against the club's leaders.

3. Check for Members: Checks if the club has any members.

4. Display Members in a New Window

- Creates a new window (Toplevel) to display members.
- Uses scrollbars for navigation.
- Delete member button: Triggers delete_selected() to remove a selected member.
- Close button: Closes the window.

5. Nested Function: (delete_selected())

This function is defined inside view_members() so it can access the mems list (current club members), update the member_list display, and call save_data() to save any changes made. How It Works:

1. Check for Selection: Ensures a member is selected before proceeding.

2. Get Member Data: Retrieves the index of the selected member and its data.

3. Confirmation Dialog: Asks the user to confirm deletion.

4. Delete Member, save changes and show message:

- Remove Member: Uses Python's del to remove the member from the list at the selected index.
- Save Changes: Ensures deletion is saved to the file (clubs_data.json)

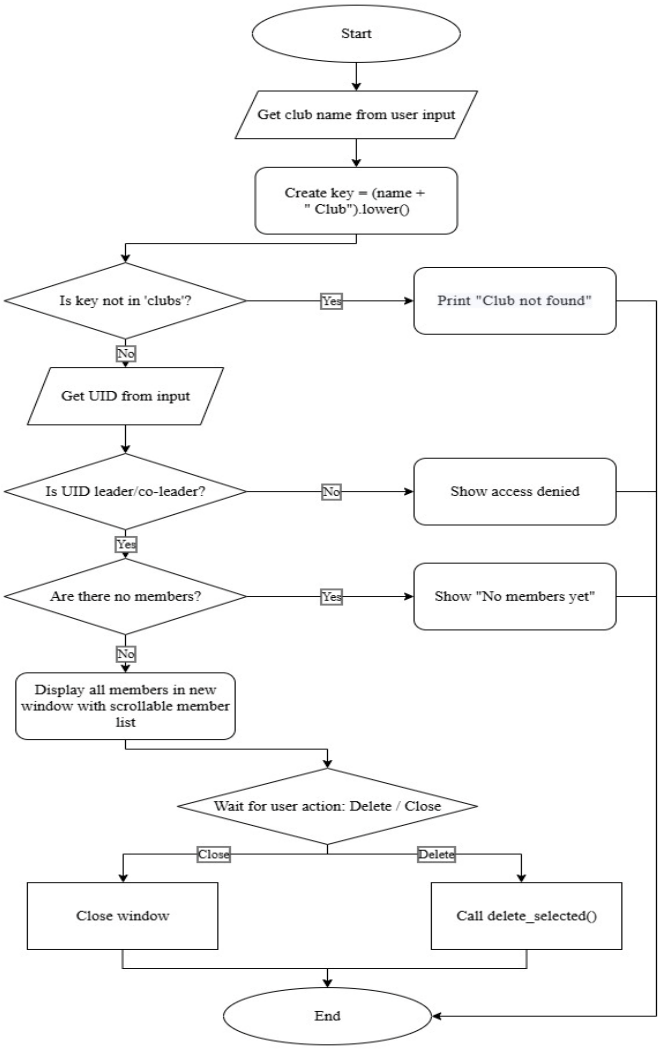
```python
def view_members():
    club_name = custom_ask_string("Club Name", "Enter your club name (no 'club' at the end):")
    if not club_name:
        return
    key = (club_name.strip() + " Club").lower()
    if key not in clubs:
        messagebox.showerror("Error", "Club not found.")
        return

    uid = custom_ask_string("ID", "Enter your leader/co-leader ID:")
    if not uid:
        return
    if uid not in (clubs[key]['leader_id'], clubs[key]['co_leader_id']):
        messagebox.showerror("Access Denied", "Only the leader or co-leader can view members.")
        return

    mems = clubs[key]['members']
    if not mems:
        messagebox.showinfo("Members", "No members yet.")
        return

    win = tk.Toplevel(root)
    win.title(f"Members of {clubs[key]['name']}")
    win.geometry("450x350+550+250")
    win.configure(bg="#d3bbbb")
    tk.Label(win, text="Member list", bg="#f2f2f2", font=("Arial", 12)).pack(pady=5)

    list_frame = tk.Frame(win)
    list_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
    v_scrollbar = tk.Scrollbar(list_frame, orient=tk.VERTICAL)
    v_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    h_scrollbar = tk.Scrollbar(list_frame, orient=tk.HORIZONTAL)
    h_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)
    member_list = tk.Listbox(list_frame, width=60, height=12, yscrollcommand=v_scrollbar.set, xscrollcommand=h_scrollbar.set)
    member_list.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
    v_scrollbar.config(command=member_list.yview)
    h_scrollbar.config(command=member_list.xview)

    for m in mems:
        member_list.insert(tk.END, f"{m['name']} | ID:{m['id']} | Reason: {m['purpose']}")

    def delete_selected():
        sel = member_list.curselection()
        if not sel:
            messagebox.showwarning("No Selection", "Please select a member to remove.")
            return
        index = sel[0]
        member = mems[index]
        confirm = messagebox.askyesno("Confirm", f"Are you sure you want to delete {member['name']}?")
        if confirm:
            del mems[index]
            save_data()
            member_list.delete(index)
            messagebox.showinfo("Deleted", f"{member['name']} removed.")
```

- Update UI Listbox: Removes the deleted member from the displayed list.
- Show Confirmation: Notifies the user of successful deletion.

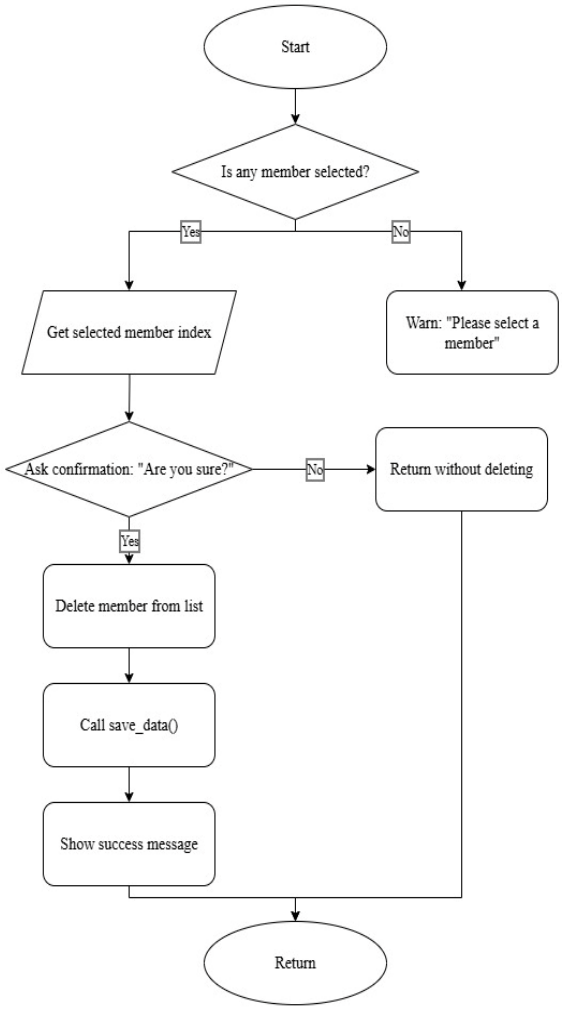
Key Concepts Used

- Nested Functions: delete_selected() is defined inside view_members() to access mems, member_list, and save_data().
- Toplevel Windows: tk.Toplevel(root) creates a secondary window for member management.
- Listbox & Scrollbars: Listbox, Scrollbar, and pack() organize the member list with scrolling.
- Conditional Logic (if): Ensures only valid users can view/delete members.

Main function's flowchart (view_members())



Sub-function's flowchart (delete_selected())



In conclusion, this project presents a practical and interactive School Club Management System developed using Python and Tkinter, designed to simplify the organization of clubs, members, and activities in a school environment. The system allows users to create and delete clubs, manage members and activities, view club details, and securely store all data between sessions.

Core Python Concepts Applied

1. Collection data type like dictionary and lists are used to store and manage nested club data effectively.
2. Conditionals and Loops control logic flow, validate input, and handle multiple entries.
3. Data Persistence with json.dump() and json.load() ensures long-term storage and retrieval of club information.
4. Defined Functions are each used for specific tasks, keeping the code modular and organized.

GUI Design with Tkinter

Tkinter handles the visual interface, using frames, labels, entry fields, buttons, and custom dialogs. Components like Listbox, Scrollbars, and messageboxes enhance interactivity and user experience.