Pro Attendance Tracker

A Modern Python Desktop Application for Student Management

Built with CustomTkinter & JSON

Presentation Agenda

- **Part 1:** Project Overview & Objectives
- **Part 2:** Technology Stack
- **Part 3:** System Architecture
- **Part 4:** Core Functionality
- **Part 5:** Data Management (JSON)
- **Part 6:** Algorithm & Logic
- **Part 7:** User Interface Design
- **Part 8:** Future Scope & Conclusion

The Problem

Manual attendance tracking is inefficient and error-prone.

- Difficult to calculate percentages in real-time.
- Hard to know how many classes can be safely missed.
- Paper records get lost or damaged.

The Solution

Pro Attendance Tracker

A digital, desktop-based solution that automates the calculation of attendance metrics.

It provides instant feedback on whether a student meets the required attendance criteria (e.g., 75%).

Key Features

Multi-Student Support

Manage multiple student profiles within a single app instance.

Auto-Calculation

Real-time calculation of attendance percentages and "safe to miss" counts.

Persistence

Automatically saves all data to a local JSON file.

Tech Stack

- **Python 3:** Core logic and scripting.
- **Tkinter:** Standard GUI library for Python.
- **CustomTkinter:** Modern, rounded, high-DPI UI library.
- **JSON:** Lightweight data interchange format for storage.

Why Python?

Python was chosen for its rapid development capabilities and extensive standard library.

```
        import json import os import tkinter as
tk
```

- **Readability:** Clean syntax makes the code easy to maintain.
- **Libraries:** Rich ecosystem for GUI (Tkinter) and File I/O.
- **Cross-Platform:** Runs on Windows, macOS, and Linux.

Tkinter Base

Tkinter provides the fundamental windowing system.

- We use `tkinter.messagebox` for alerts.
- We use `tkinter.simpledialog` for input popups.
- It serves as the foundation upon which CustomTkinter builds.

Modernizing with CustomTkinter

Standard Tkinter looks outdated. **CustomTkinter** provides:

- Rounded corners
- Dark mode support out-of-the-box
- High DPI scaling
- Modern color themes

```
import customtkinter as ctk
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("dark-blue")
```

Data Persistence: JSON

We use a `attendance_pro_data.json` file to store application state.

- Human-readable format.
- Easy to debug and edit manually if needed.
- Native support in Python via the `json` module.

App Configuration

Global constants allow for easy theming and maintenance.

```
    DATA_FILE = "attendance_pro_data.json" APP_TITLE = "Pro Attendance Tracker (Students)" PEACH =
"#FFB88C" # primary accent BG_DARK = "#1E1F22" # background CARD_BG = "#2A2C2F" # card background
```

Color Palette

Background

#1E1F22

Dark, modern canvas that reduces eye strain.

Peach Accent

#FFB88C

Used for primary actions, highlights, and titles.

Card Surface

#2A2C2F

Slightly lighter dark for distinct UI elements.

File Structure

The project consists of a single script and a generated data file.

📁 Project Root

├── 📄 attendance_custom_students.py (Main Application)

└── 📄 attendance_pro_data.json (Generated Data)

This simple structure makes the application portable and easy to distribute.

Loading Data

The `load_data()` function ensures robustness.

```
    def load_data(): if not os.path.exists(DATA_FILE): return {"students": {...}, "settings": {...}}
try: with open(DATA_FILE, "r") as f: data = json.load(f) #  ...  validation logic  ...  return data
```

Data Migration

The code includes a smart migration system for older data formats.

- **Old Format:** Top-level "subjects" key.
- **New Format:** Nested under "students" -> "Default Student".
- This prevents data loss when users upgrade the application.

Saving Data

The `save_data()` function writes the current state to the disk.

It uses `indent=4` for pretty-printing, making the JSON file human-readable.

```
      def save_data(data): with
open(DATA_FILE, "w") as f: json.dump(data, f,
indent=4)
```

The AttendanceApp Class

The entire GUI logic is encapsulated in the `AttendanceApp` class.

- **`__init__`:** Sets up the window and loads data.
- **State Variables:** Tracks current student and goal settings.
- **UI Construction:** Builds frames, buttons, and labels.

Window Setup

```
    self.root.title(APP_TITLE) self.root.geometry("900×700") self.root.minsize(760, 560)
ctk.set_appearance_mode("dark")
```

We define a responsive geometry, ensuring the app looks good on different screen sizes while enforcing a minimum usable size.

Header Component

The header contains:

- App Title
- Student Selector (OptionMenu)
- Global Actions (Add Student, Set Goal)

Top Navigation Bar

Student Management

The app isn't limited to one user.

Add Student

Creates a new key in the JSON dictionary.

Delete Student

Removes the student and all associated subject data.

Switching

Dropdown menu instantly swaps the data context.

Add Student Popup

```
      popup = tk.Toplevel(self.root)
popup.title("Add Student") #  ...  inputs for
Name, Class, Roll  ...
```

We use `tk.Toplevel` to create a modal dialog window.

This keeps the main interface clean while gathering necessary details like Name, Class, and Roll Number.

Input Validation

Robust validation prevents data corruption.

- **Empty Names:** Rejected.
- **Duplicates:** Checks if student name already exists.
- **Numeric Inputs:** Ensures attendance counts are integers.

Deletion Safety

Critical actions require confirmation.

```
    if messagebox.askyesno("Confirm Delete", f"Delete student '{self.current_student}'?"): del
self.data["students"][self.current_student]
```

The app ensures at least one student (or a default placeholder) always remains to prevent crashes.

Subject Management

The core of the app revolves around "Subjects".

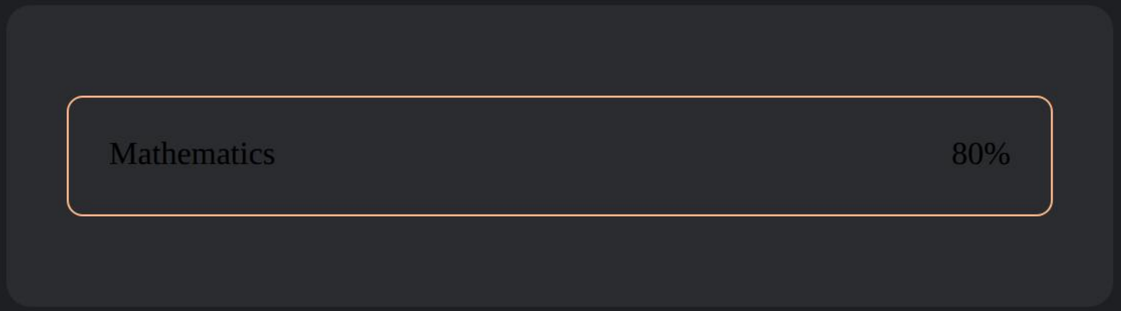
- Stored as a dictionary: `{ "Maths": { "attended": 10, "total": 12 } }`
- Users can add unlimited subjects per student.
- Each subject maintains its own counters.

The "Subject Card" UI

Instead of a boring list, we use dynamic "Cards".

Left Side: Subject Name + Status Message.

Right Side: Stats + Quick Action Buttons.



Rendering Logic

The `render_subject_cards` function creates the UI dynamically.

1. Clears existing widgets from the scrollable frame.
2. Iterates through the student's subjects.
3. Calculates the current percentage.
4. Instantiates a new `ctk.CTkFrame` for each subject.

Scrollable Frames

```
self.scrollable = ctk.CTkScrollableFrame( list_area_frame, corner_radius=8 )
```

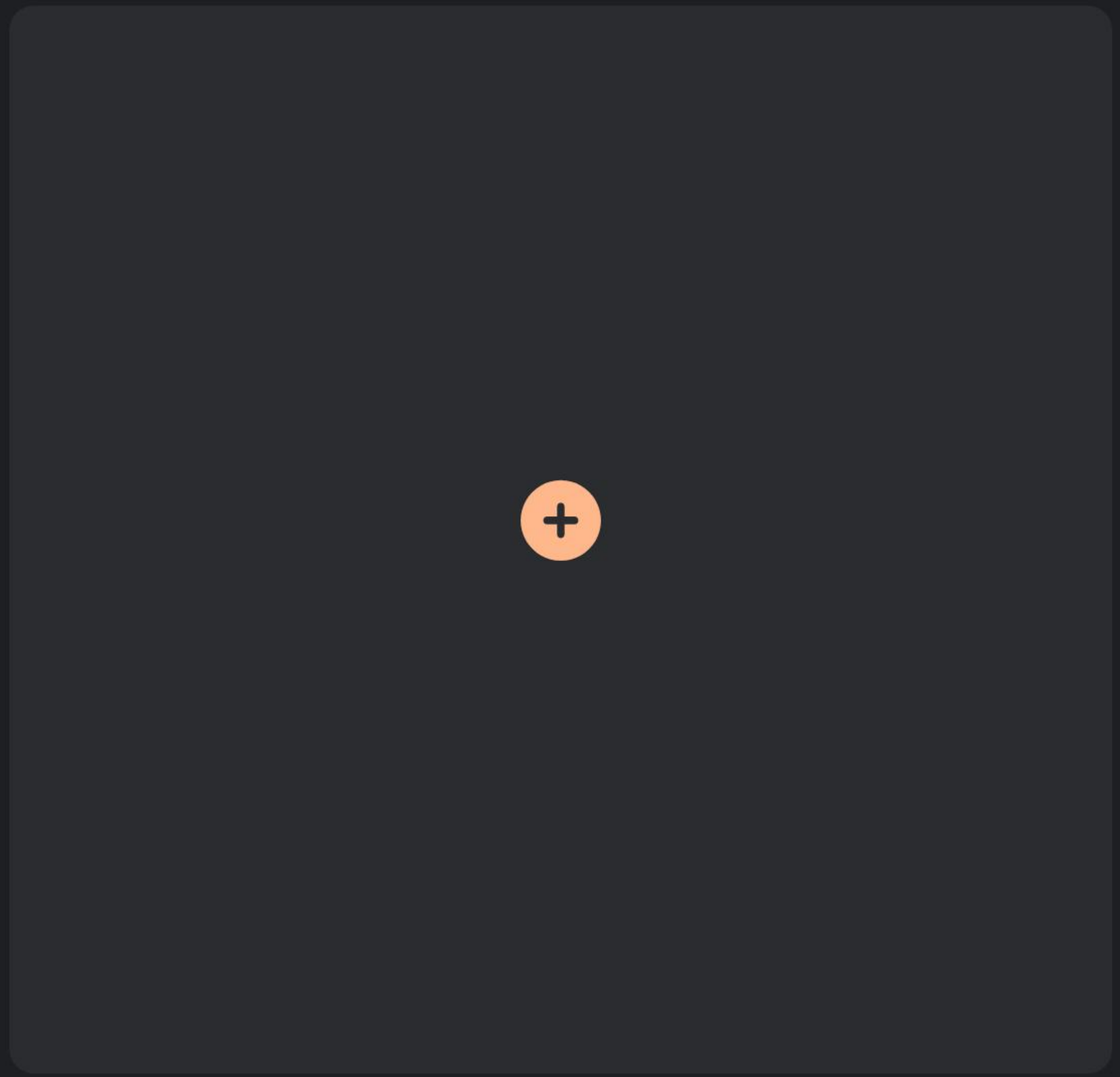
Using `CTkScrollableFrame` allows the list of subjects to grow infinitely without breaking the window layout.

Adding a Subject

User enters a name in the entry box.

The system initializes counters to zero:

```
{"attended": 0, "total": 0}
```



Selection Logic

The UI relies on a "Selection" state.

- Clicking a card sets `self.selected_subject`.
- The selected card changes color (visual feedback).
- Action buttons (Mark Attended/Missed) apply to the *selected* item.

Marking Attendance

Attended (Present)

Increments BOTH **Attended** and **Total**.

+1 / +1

Missed (Absent)

Increments ONLY **Total**.

+0 / +1

Result

Percentage recalculates instantly.

Calculation Logic

$$\text{Percentage} = \frac{\text{Attended}}{\text{Total}} \times 100$$

Handled gracefully when Total is 0 to avoid DivisionByZero errors.

Quick Actions

Each card has mini-buttons for rapid data entry.

```
    # Lambda function captures the specific subject 's' command=lambda s=subj: self._quick_attend(s)
```

This bypasses the need to "Select" -> "Click Button", making the workflow twice as fast.

Goal Setting

Users can define a target percentage (default 75%).

This variable drives the logic for the status indicators.

Algorithm: "Safe to Miss"

If current % > Goal, how many classes can I skip?

```
can_miss = int((attended / goal_percent) - total)
```

This tells the student exactly how much "buffer" they have before dropping below the threshold.

Algorithm: "Must Attend"

If current % < Goal, how many consecutive classes must I attend?

```
needed = ceil(((goal * total) - attended) / (1 - goal))
```

This solves for x where `(attended + x) / (total + x) = goal`.

Visual Feedback

Safe Zone

Displays a "Safe" message telling the user they have a buffer.

Danger Zone

Displays a "Warning" message indicating recovery steps needed.

Status Text

e.g., "Attend next 3 classes."

Manual Editing

Mistakes happen. The "Edit Selected" button opens a dialog to manually override numbers.

- Useful if the user forgot to log attendance for a week.
- Validates that `Attended <= Total`.

System Reset

The "Reset All" button wipes the JSON data clean.

- Restores the "Default Student".
- Resets goal to 75%.
- **Safety:** Requires a confirmation popup to prevent accidental data loss.

The Summary Footer

A persistent footer provides a high-level overview.

```
Subjects: 5 Overall: 45/50 (90.00%) At-risk: 0
```

Aggregates data from all subjects to give a "Big Picture" view of academic performance.

Event Binding

To improve UX, we bind click events to the entire card frame.

```
card.bind("
 ", make_onclick()) lbl_name.bind("

  ", make_onclick())
```

This ensures the user doesn't have to click a tiny pixel; clicking anywhere on the subject row selects it.

Robustness

The app handles edge cases gracefully:

- **Corrupt JSON:** Resets to default.
- **Zero Division:** Handled in percentage calc.
- **Invalid Inputs:** Caught with Try/Except blocks.

UX Design Choices

- **Dark Mode:** Default setting for modern aesthetic.
- **Minimalism:** Hidden complexity; only shows what's needed.
- **Color Coding:** Green for good, Red for bad (universal signals).

Code Modularity

The code is split into logical sections:

Section	Purpose
Config	Constants and settings
Helpers	Pure functions like `load_data`
Class	GUI and State logic

Project Advantages

Fast

Lightweight script, instant startup.

Private

Data stored locally, no cloud dependency.

Flexible

Works for any number of subjects or students.

Current Limitations

- **Local Only:** Data doesn't sync across devices.
- **No Timestamps:** Doesn't record *when* a class was attended, only the count.
- **UI Scaling:** Might require adjustments on very small monitors.

Future Scope: Database

Migrating from JSON to **SQLite**.

This would allow for complex queries, history tracking, and reporting (e.g., "Attendance by Month").

Future Scope: Export

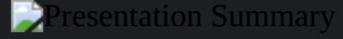
Adding an "Export to Excel/PDF" feature.

This would allow students to generate a report card to share with parents or teachers directly from the app.

Learning Outcomes

Building this project demonstrated proficiency in:

- Python OOP (Object Oriented Programming).
- GUI Development with Tkinter/CustomTkinter.
- Data Serialization (JSON).
- Algorithm design for real-world problems.



Summary

The Pro Attendance Tracker is a complete, user-friendly tool for students.

It solves the anxiety of attendance management through automation and clear visual feedback.

Thank You

Questions?

Image Sources

https://img.freepik.com/free-vector/time-attendance-tracking-system-abstract-concept-illustration_335657-3762.jpg

Source: www.freepik.com

https://d1uxiwmpc9j4yg.cloudfront.net/images/all/zastavka_1706179980.jpg

Source: inveritasoft.com

https://uxwing.com/wp-content/themes/uxwing/download/file-and-folder-type/json-file-icon.svg

Source: uxwing.com

https://images.stockcake.com/public/b/7/9/b79a5ef6-4a89-4431-b086-88ab9d40ec47_large/team-meeting-discussion-stockcake.jpg

Source: stockcake.com