Part A: Project: CBT Thought Diary (CLI)

A specialized mental health tool designed to help users navigate negative thought patterns by identifying cognitive distortions and reframing them through evidence-based challenges.

Project Overview

The **CBT Thought Diary** is an interactive command-line interface (CLI) application that guides users through the "Thought Record" process. By documenting feelings, identifying logical fallacies in thinking, and creating balanced alternative perspectives, users can actively improve their emotional well-being.

Key Features

- **Mood Tracking:** Users log their emotional state both before and after the exercise to track the effectiveness of the reframing.
- **Distortion Identification:** A comprehensive library of **15 cognitive distortions** (such as All-or-Nothing Thinking, Catastrophizing, and Emotional Reasoning) to help users label irrational thoughts.
- **Structured Reframing:** Interactive prompts guide the user to provide evidence against their negative thoughts and develop a more balanced view.
- **Persistent Storage:** Uses a JSON-based local database to save, retrieve, and manage past entries.
- **Dynamic UI:** Implements clear screen transitions and text wrapping for a polished, readable terminal experience.

Technical Stack

- **Language:** Python 3.12
- **Data Persistence:** JSON (Local File System)
- **Standard Libraries:** * `datetime`: For timestamping entries.
    - `textwrap`: To ensure examples and thoughts are formatted cleanly regardless of terminal width.
    - `os` & `sys`: For environment-specific commands and system exits.

Cognitive Distortions Supported

The software includes a robust catalog of distortions that users can select from when analyzing their thoughts:

Distortion	Description / Example
All-or-Nothing	Seeing things in black-and-white (e.g., "If I'm not perfect, I'm a failure").

Distortion	Description / Example
Fortune Telling	Predicting things will turn out badly as if it were an established fact.
Emotional Reasoning	Assuming negative emotions reflect reality (e.g., "I feel guilty, so I must have done something wrong").
Labeling	Extreme overgeneralization, such as calling yourself a "loser" instead of acknowledging a mistake.
Should Statements	Using "musts" and "shoulds" to motivate, which often leads to unnecessary guilt.

System Architecture & Logic

The application follows a modular functional approach:

1. **Input Layer:** Captures multi-line user responses for thoughts and challenges.
2. **Logic Layer:** Processes the identification of distortions and calculates mood shifts.
3. **Storage Layer:** Handles the serialization of entries into `cbt_diary_data.json` for long-term use.
4. **Navigation:** Includes a menu-driven interface to read through historical entries or delete them as needed.

Project Outcome: This tool successfully creates a bridge between therapeutic practices and software, providing a private, local-first way for individuals to manage stress and anxiety through structured logic.

Part B: Project Title: CBT Thought Diary (Python GUI)

Overview

A desktop application designed to facilitate Cognitive Behavioral Therapy (CBT) exercises. Originally conceived as a command-line interface (CLI) script, I refactored and expanded the project into a fully interactive GUI application using Python and Tkinter.

The application guides users through a structured 5-step process to identify negative thoughts, recognize cognitive distortions, and reframe thinking patterns.

Key Features

- **5-Step CBT Wizard:** A guided interface taking the user through Mood Check-in, Situation Analysis, Distortion Identification, Reframing, and Final Reflection.
- **Modern UI/UX Design:** Built with a custom design system mimicking modern web aesthetics (Tailwind CSS "Indigo" & "Slate" palettes) within a legacy desktop framework.

- **Interactive Elements:** Custom-built interactive "cards" for selecting cognitive distortions, replacing standard native checkboxes for better usability.
- **Data Persistence:** Automatically saves and loads entries via local JSON storage, allowing users to review their history.
- **Zero Dependencies:** Runs on standard Python libraries (`tkinter`, `json`, `datetime`) requiring no `pip install` for the end user.

Technical Highlights

1. Modernizing Legacy GUI Frameworks

One of the primary challenges was the visual limitation of Python's `tkinter` library. To overcome the "Windows 95" aesthetic, I implemented a custom styling engine:

- **Design Tokens:** I defined a dictionary of constants for colors (`#4f46e5` for primary, `#f8fafc` for backgrounds) to ensure visual consistency.
- **Custom Widgets:** I wrapped standard Tkinter widgets to create "Cards," "Badges," and "Ghost Buttons," overriding default borders and padding to create a flat, modern look.
- **Responsive Layout:** Utilized Tkinter's `pack` and `grid` geometry managers with scrollable canvases to ensure the app works on various screen sizes and OS environments (macOS/Windows).

2. State Management

The application manages the state of a multi-step wizard. I implemented a dictionary-based state container (`self.current_entry_data`) that accumulates data across the 5 steps. This allows the user to navigate back and forth between steps without losing their input, only committing the data to the JSON database upon final save.

3. Cross-Platform Compatibility

I addressed specific OS-level rendering bugs (specifically MacOS button highlighting issues) by building custom click-event handlers on `Frame` and `Label` widgets, ensuring the UI behaves consistently regardless of the operating system.

Code Snippet: Custom UI Component

Example of creating a selectable "Card" UI element in Python/Tkinter:

```
# Creating a clickable "Distortion Card" with hover effects and state
toggling
def create_distortion_card(self, parent, title, desc, is_selected):
    bg_col = COLORS["selected_bg"] if is_selected else "white"
    border_col = COLORS["selected_border"] if is_selected else "#dddddd"

    # Custom Frame acting as a button
    card = tk.Frame(parent, bg=bg_col, pady=10, padx=10)
    card.configure(highlightbackground=border_col, highlightthickness=2 if
is_selected else 1)
```

```
    # Binding click events to the frame and all children elements
    card.bind("<Button-1>", lambda e: self.toggle_selection(title))

    return card
```

Technologies Used

- **Language:** Python 3.x
- **GUI:** Tkinter (Tcl/Tk)
- **Data:** JSON
- **Version Control:** Git