PROJECT REPORT
LANGUAGE TRANSLATOR

CSM216

(PYTHON PROJECT)

COMPUTER SCIENCE AND ENGINEERING

Submitted by**:**

Raja Neeladri Varma

12303168

K23UP – G2

ROLL NUMBER: 64

Submitted to:

Mr. AMAN KUMAR

LOVELY PROFESSIONAL UNIVERSITY

ACKNOWLEDGEMENT

I, Raja Neeladri Varma, a student of Bachelor of Technology under Computer Science and Engineering discipline with Data Sceince and Machine Learning specialization at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine.

Raja Neeladri Varma

12303168

TABLE OF CONTENTS

I. INTRODUCTION

The **Language Translator** is a Python-based application designed to provide a user-friendly and efficient way to translate text between various languages. This project leverages the **Tkinter** framework to create an interactive graphical user interface (GUI) and integrates the **Google Translate API** for accurate and real-time translations. The tool is ideal for users who need quick translations for personal or professional use, offering support for a wide range of languages.

Language barriers often hinder effective communication, whether in daily conversations, travel, or business dealings. Manual translation is time-consuming and prone to errors, especially for users unfamiliar with a particular language. This project addresses these challenges by providing an intuitive and reliable tool for seamless language translation.

This project implements a desktop application with the following features:

- **Text Translation:** Users can input text in one language and translate it into another language from the list of supported languages.
- **User-Friendly Interface:** The application features a clean and simple interface that is easy to navigate.
- **Clear Functionality:** Users can quickly reset the input and output fields for consecutive translations**.**

Implementation Details

- **Programming Language:** Python is used as the primary programming language due to its simplicity and extensive library support.
- **GUI Framework:** Tkinter is used to build an interactive interface with widgets such as text boxes, buttons, and dropdown menus**.**
- **Translation API:** The Google Translate API is integrated to handle the translation logic and provide accurate results.

This project is a small yet impactful step toward breaking down language barriers and enhancing communication through technology. The simplicity of the interface combined with the robust translation capabilities makes it a unique and effective tool for users worldwide**.**

II. OBJECTIVES AND SCOPE OF THE PROJECT

OBJECTIVES:

The primary objective of the **Language Translator** application is to create an efficient and user-friendly tool for translating text between multiple languages. It aims to address communication barriers by providing accurate and real-time translations using a graphical interface that is accessible to users of all technical levels. The application is designed to simplify the process of language translation while ensuring reliability and ease of use.

SCOPE:

The scope of this project includes the development and implementation of a language translation application with the following features:

1. **Multi-language Support:** The application supports translations between over 35 languages, including English, Spanish, French, German, Chinese, Arabic, and many others.
2. **Interactive GUI:** Built with the Tkinter library, the application provides an intuitive graphical user interface that enables users to interact with the tool effortlessly.
3. **Real-time Translation:** Integration with the Google Translate API ensures accurate and prompt translation of user-provided text.
4. **Input and Output Management:** The application includes dedicated text fields for inputting the original text and displaying the translated text.
5. **Error Handling**: Ensures users are prompted with messages if the input text is missing or an invalid operation occurs.
6. **Customizability and Accessibility:** Users can select source and target languages from dropdown menus, allowing for flexible translation configurations.
7. **Scalable Design:** The application is designed to be extensible, with potential future features like voice input/output, file translation, and offline functionality.

Future Prospects:

- Incorporation of voice-based translation features for spoken languages.
- File-based translation capabilities to handle document or PDF translations.
- Offline translation functionality by integrating pre-trained language models.
- Enhanced UI with theming and accessibility options for visually impaired users.

This project aims to serve individuals, professionals, and organizations seeking efficient and accurate language translation, making it a versatile and practical tool in a globalized world.

III. APPLICATION TOOLS

The development of the Language Translator application utilizes a combination of tools and technologies to ensure efficient implementation and optimal functionality. These tools include:

1. **Programming Language:**
   - **Python**: The core programming language used for its simplicity, flexibility, and extensive library support.
2. **Graphical User Interface (GUI) Framework:**
   - **Tkinter:** Used to create an interactive and user-friendly graphical interface for the application**.**
3. **Translation API:**
   - **Google Translate API:** Provides robust and accurate translation capabilities for a wide range of languages**.**
4. **Development Environment:**
   - **Integrated Development Environment (IDE):** Tools like PyCharm, VS Code, or IDLE can be used to write and debug the Python code**.**
5. **Libraries and Modules:**
   - **googletrans:** A Python library used to interact with the Google Translate API for handling translation functionality.
   - **tkinter.ttk:** For advanced Tkinter widgets like dropdown menus**.**
   - **messagebox**: From Tkinter, used for displaying error or **confirmation messages to the user**
     .
6. **Error Handling and Debugging Tools:**
   - **Python Debugger (pdb):** Used to troubleshoot and debug the application during development.
   - **Linting Tools:** Tools like Pylint or Flake8 to ensure clean and readable code.
7. **Testing and Deployment:**
   - **Manual Testing:** For verifying GUI functionality and translation accuracy**.**
   - PyInstaller**:** To package the application as an executable file for distribution.

These tools ensure that the Language Translator is built with efficiency, reliability, and user satisfaction in mind, providing a solid foundation for future enhancements**.**

IV. PROJECT DESIGN

The design of the **Language Translator** application revolves around creating an intuitive interface, seamless integration with a translation API, and a modular structure for maintainability and scalability. Below is a detailed breakdown of the project design:

1. **Architectural Overview**

The application follows a client-side GUI-based architecture where the user interacts with a graphical interface, and the translation logic is processed locally using the Google Translate API. The core components include:

- **User Interface Layer:** Manages user interaction through Tkinter widgets**.**
- **Business Logic Layer:** Handles input validation, API integration, and text processing.
- **Output Display Layer:** Displays translated text and error messages.

2. **User Interface Design**

The **Tkinter-based GUI** is the core interface that facilitates user interaction. It includes the following elements**:**

- **Frame and Layout**
  - A single window with a clean and modern design using Tkinter Frame and Label.
  - Background color set to #A9D6E5 for a visually appealing interface**.**
- **Input Fields**
  - **Text Entry 1:** A multi-line text box where the user enters the source text.
  - **Text Entry 2:** A multi-line text box to display the translated text.
- **Dropdown Menus**
  - **Source Language:** An optional dropdown for selecting the input language (default: Auto Select).
  - **Target Language**: Dropdown menu for selecting the output language from a predefined list of 35+ languages.
- **Buttons**
  - **Translate Button:** Initiates the translation process by calling the translate() function.
  - **Clear Button**: Clears both text input and output fields using the clear() function.

- **Labels and Titles**
  - A title label ("Language Translator") positioned at the top for clarity.

3. **Functional Design**

1. **Core Functions**
   - **translate():**
     - Retrieves input text from Text Entry 1.
     - Gets the target language from the dropdown menu.
     - Uses the Google Translate API to process the translation.
     - Displays the translated text in Text Entry 2.
     - Provides error handling for empty input or API issues.
   - **clear():**
     - Clears both the input and output text boxes to reset the fields**.**
2. **Validation and Error Handling**
   - Validates user input to ensure the text field is not empty.
   - Uses messagebox to alert the user in case of errors or invalid operations**.**

4. **Data Flow Design**

- **Step 1**: User enters text in the input field and selects the target language.
- **Step 2**: On clicking the "Translate" button, the translate() function is triggered.
- **Step 3**: The Google Translate API processes the input text and returns the translated result.
- **Step 4:** The translated text is displayed in the output field.
- **Step 5:** The user can clear the fields for new input using the "Clear" button.

5. **Module Breakdown**

1. **GUI Module**
   - Handles the layout and interaction of widgets (e.g., Label, Text, Button, Combobox).

2. **Translation Module**
    - o Uses the googletrans library to send requests and retrieve translated text**.**

3. **Utility Module**
    - o Provides helper functions for input validation and error handling.

6. UI Mockup

Main Window

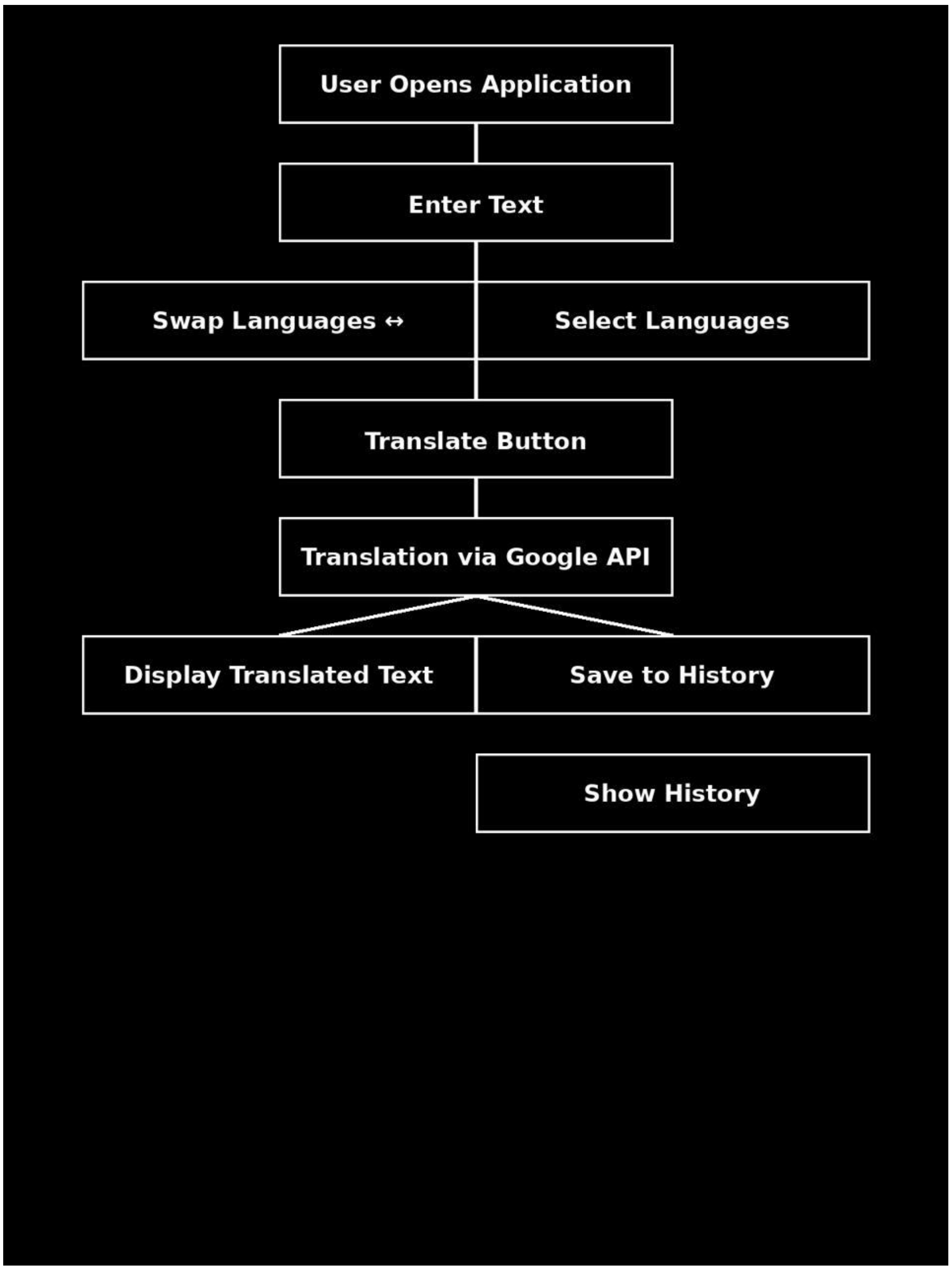
- Title**: "Language Translator" displayed prominently at the top.**
- Input Box**: Positioned on the left, allowing multi-line text input.**
- Dropdown Menus**: Centered at the top for language selection.**
- Output Box**: Positioned on the right to display the translated text.**
- Buttons**: At the bottom, aligned horizontally for "Translate" and "Clear" actions.**

7. **Scalability and Extensibility**

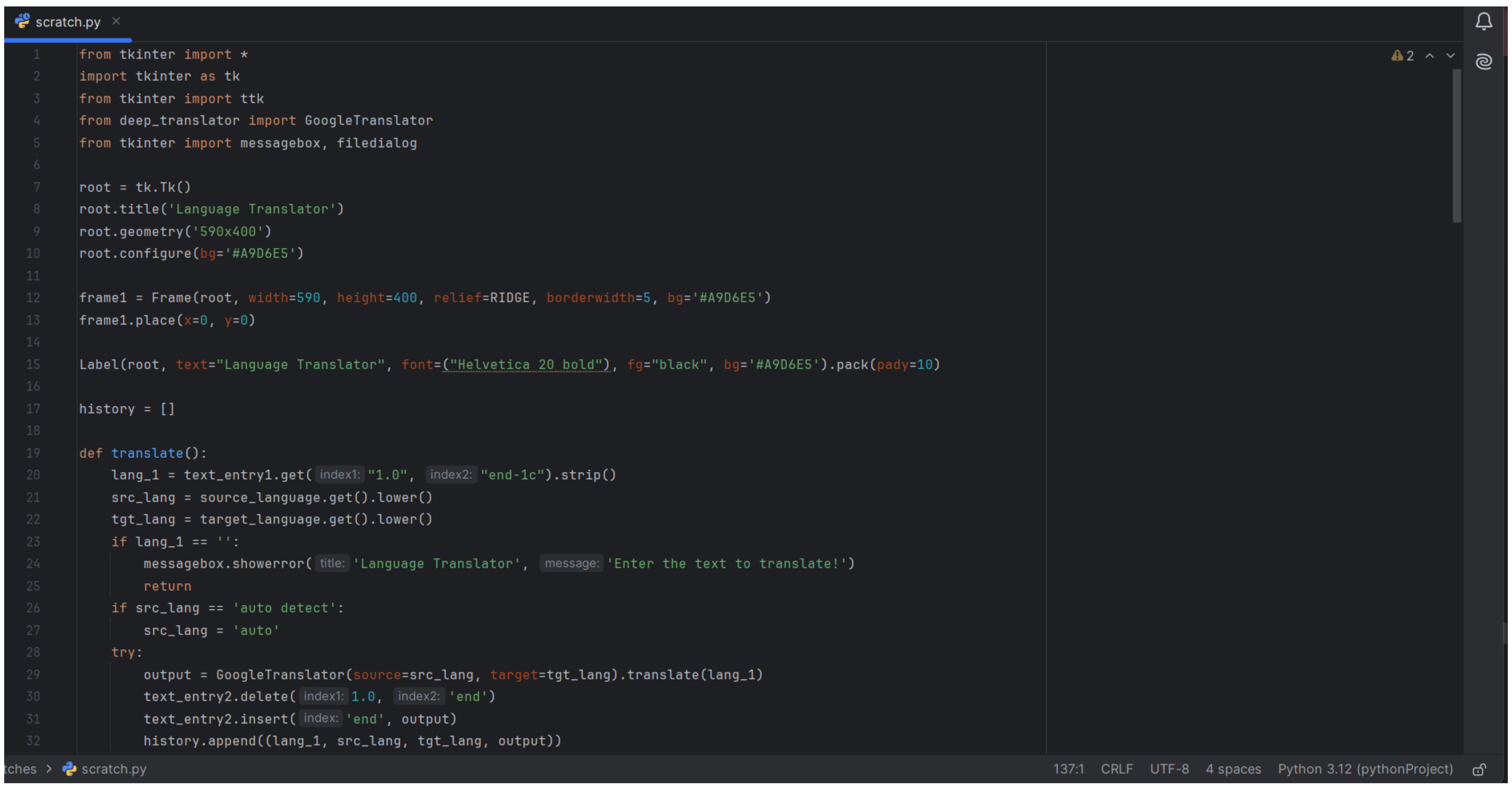
1. **Scalability**
    - o More languages can be added to the dropdown menu as needed.
    - o The application can handle larger inputs by increasing the size of the input/output fields.
2. **Extensibility**
    - o Integrate voice input/output for accessibility.
    - o Add file translation capabilities for translating documents.
    - o Include offline translation features using pre-trained models like Google's TensorFlow Lite.

This design ensures a smooth user experience, robust functionality, and a foundation for future enhancements, making the Language Translator a reliable and versatile application.

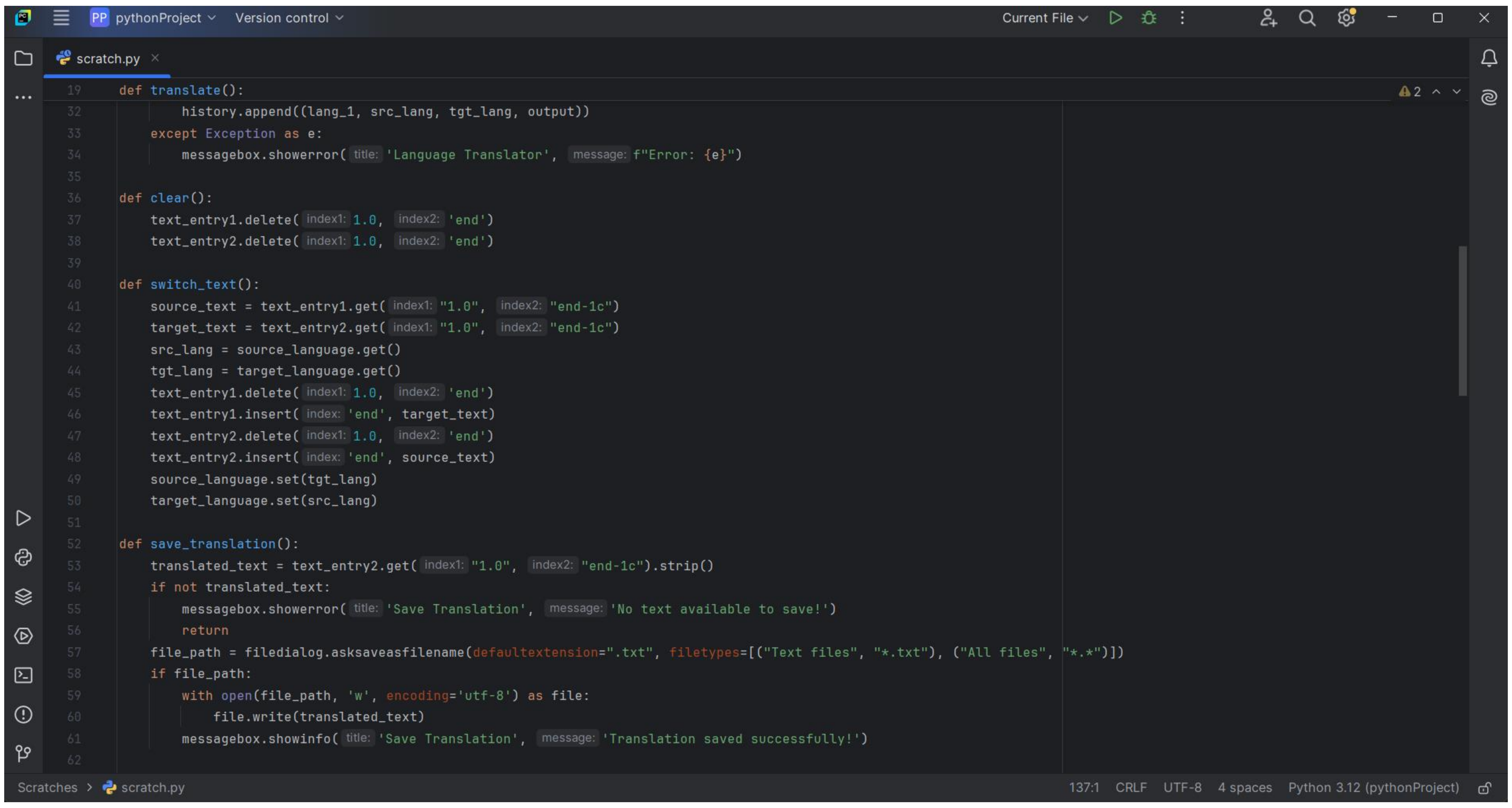
V. FLOWCHART


User Opens Application
Enter Text
Swap Languages ↔
Select Languages
Translate Button
Translation via Google API
Display Translated Text
Save to History
Show History

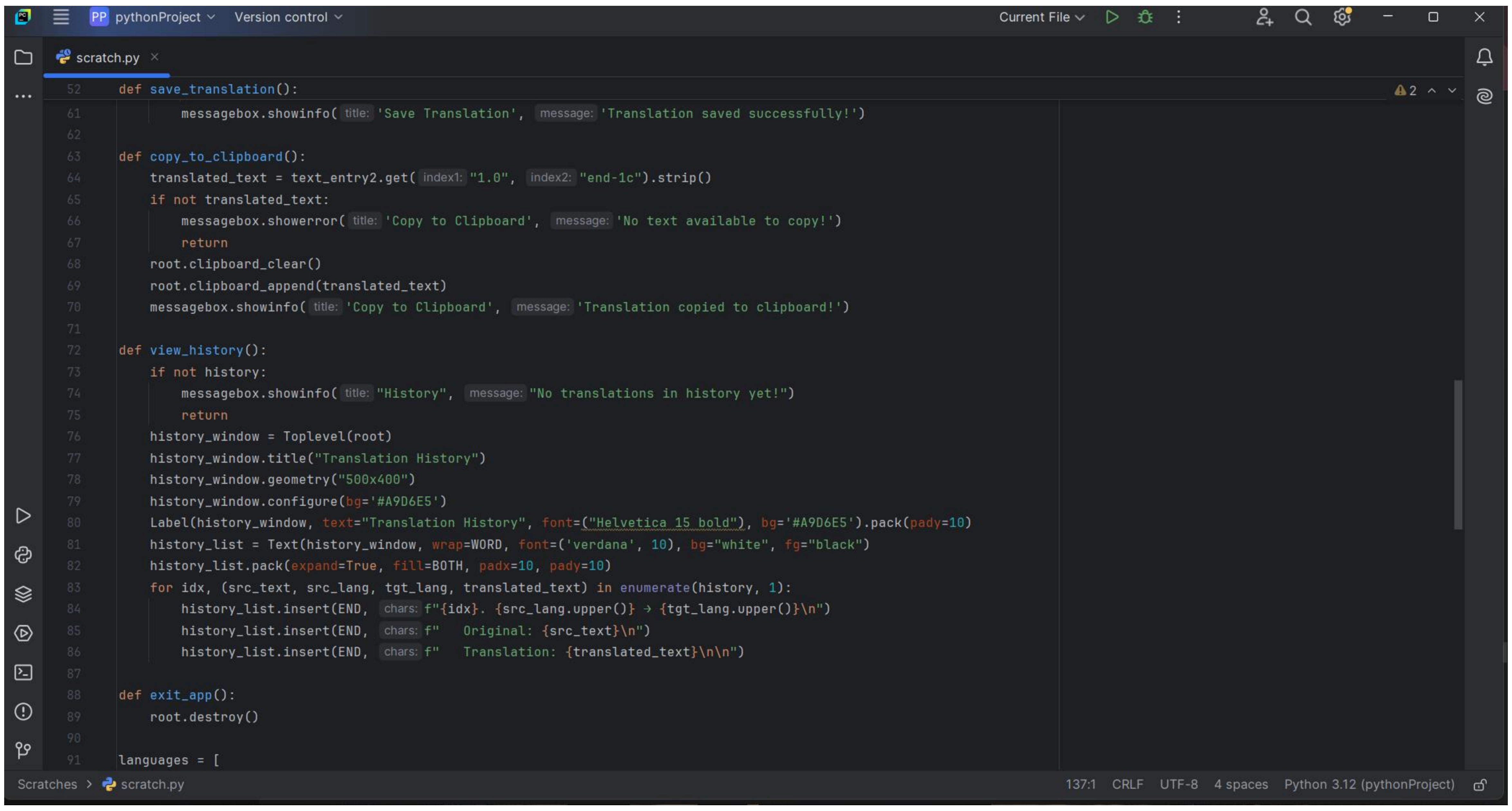
VI. PROJECT IMPLEMENTATION



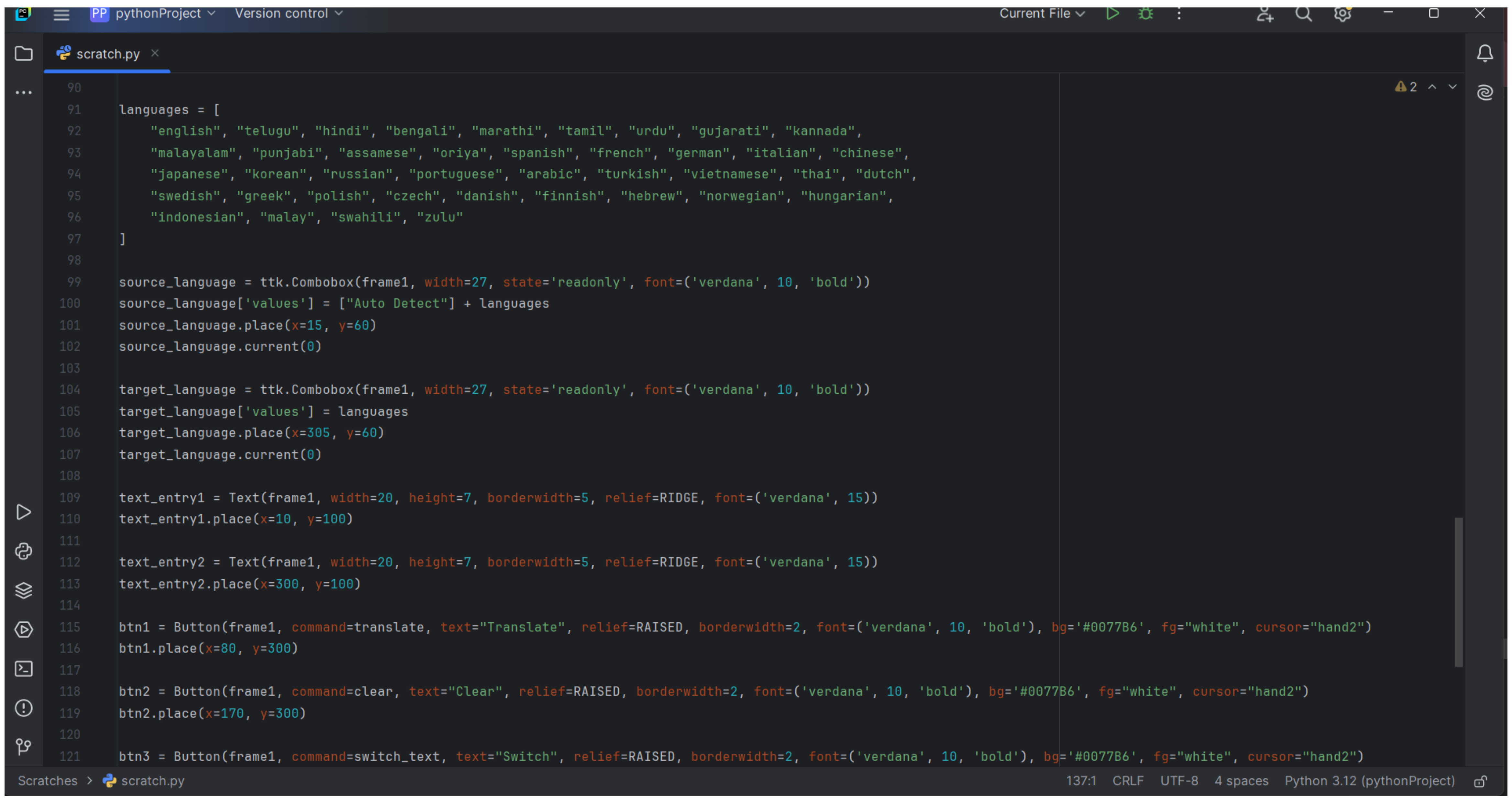
```python
from tkinter import *
import tkinter as tk
from tkinter import ttk
from deep_translator import GoogleTranslator
from tkinter import messagebox, filedialog

root = tk.Tk()
root.title('Language Translator')
root.geometry('590x400')
root.configure(bg='#A9D6E5')

frame1 = Frame(root, width=590, height=400, relief=RIDGE, borderwidth=5, bg='#A9D6E5')
frame1.place(x=0, y=0)

Label(root, text="Language Translator", font=("Helvetica 20 bold"), fg="black", bg='#A9D6E5').pack(pady=10)

history = []

def translate():
    lang_1 = text_entry1.get( index1: "1.0",  index2: "end-1c").strip()
    src_lang = source_language.get().lower()
    tgt_lang = target_language.get().lower()
    if lang_1 == '':
        messagebox.showerror( title: 'Language Translator',  message: 'Enter the text to translate!')
        return
    if src_lang == 'auto detect':
        src_lang = 'auto'
    try:
        output = GoogleTranslator(source=src_lang, target=tgt_lang).translate(lang_1)
        text_entry2.delete( index1: 1.0,  index2: 'end')
        text_entry2.insert( index: 'end', output)
        history.append((lang_1, src_lang, tgt_lang, output))
```



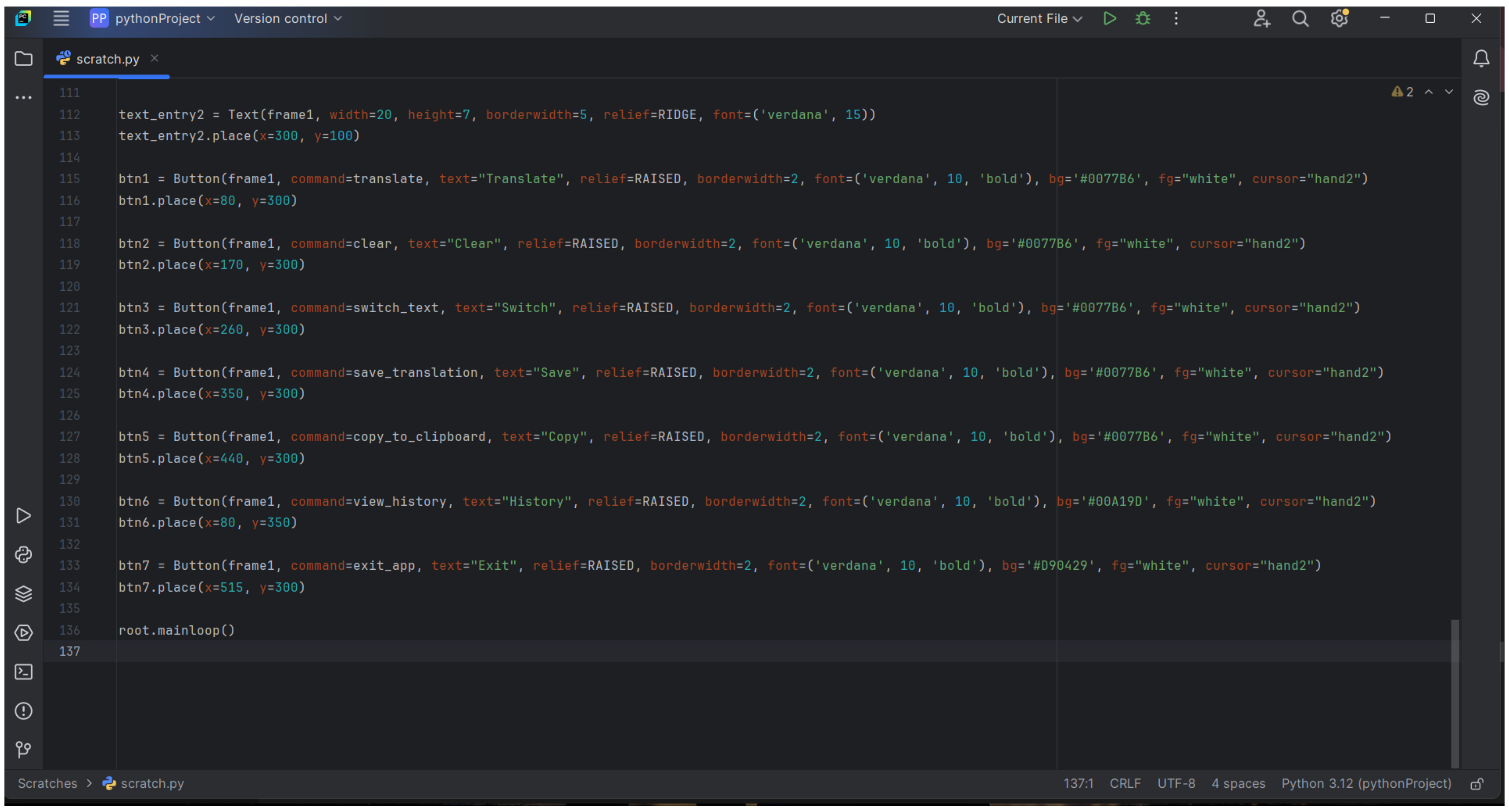
```python
def translate():
        history.append((lang_1, src_lang, tgt_lang, output))
    except Exception as e:
        messagebox.showerror( title: 'Language Translator',  message: f"Error: {e}")

def clear():
    text_entry1.delete( index1: 1.0,  index2: 'end')
    text_entry2.delete( index1: 1.0,  index2: 'end')

def switch_text():
    source_text = text_entry1.get( index1: "1.0",  index2: "end-1c")
    target_text = text_entry2.get( index1: "1.0",  index2: "end-1c")
    src_lang = source_language.get()
    tgt_lang = target_language.get()
    text_entry1.delete( index1: 1.0,  index2: 'end')
    text_entry1.insert( index: 'end', target_text)
    text_entry2.delete( index1: 1.0,  index2: 'end')
    text_entry2.insert( index: 'end', source_text)
    source_language.set(tgt_lang)
    target_language.set(src_lang)

def save_translation():
    translated_text = text_entry2.get( index1: "1.0",  index2: "end-1c").strip()
    if not translated_text:
        messagebox.showerror( title: 'Save Translation',  message: 'No text available to save!')
        return
    file_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text files", "*.txt"), ("All files", "*.*")])
    if file_path:
        with open(file_path, 'w', encoding='utf-8') as file:
            file.write(translated_text)
        messagebox.showinfo( title: 'Save Translation',  message: 'Translation saved successfully!')
```



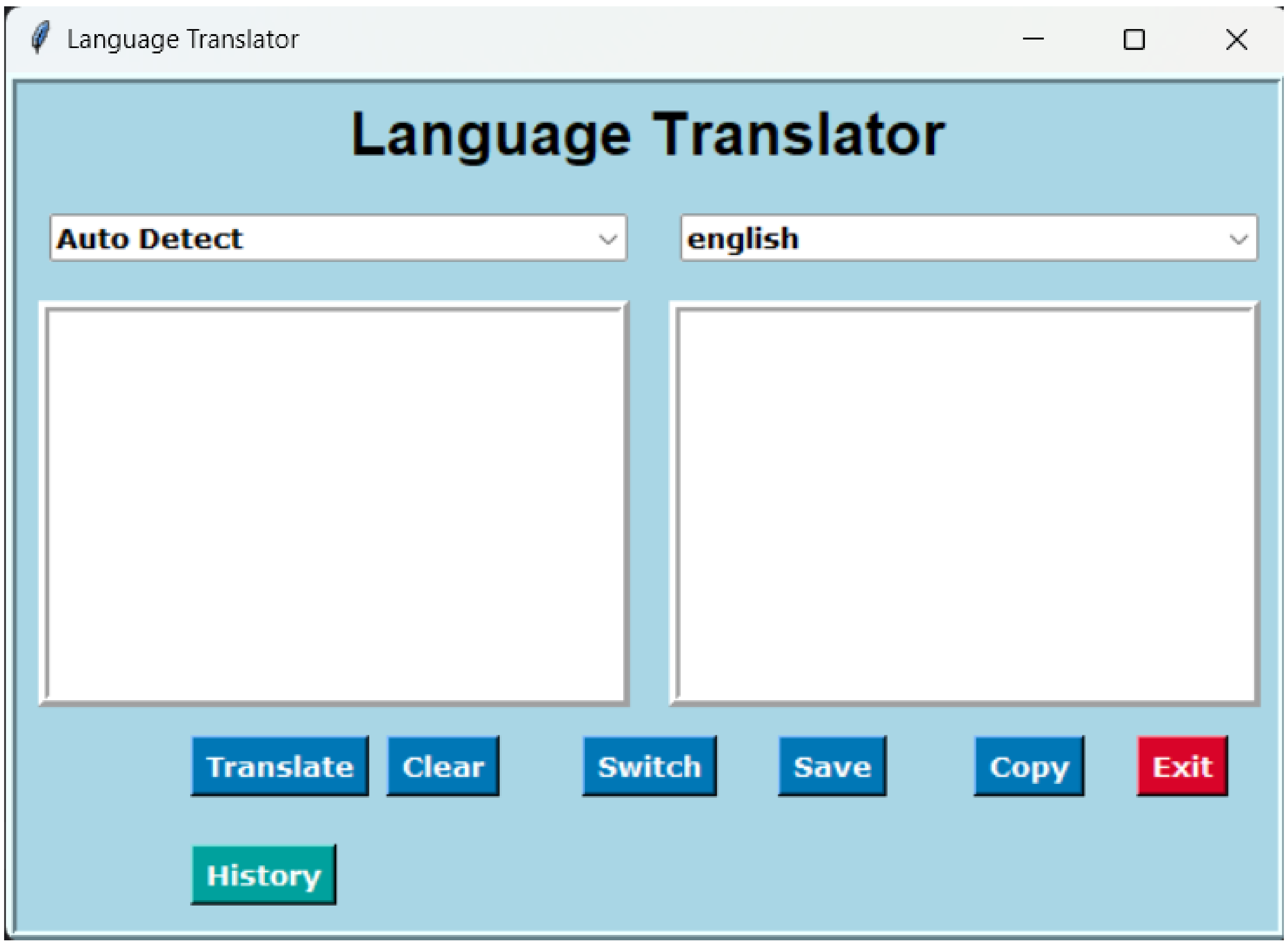
```
def save_translation():
        messagebox.showinfo( title: 'Save Translation',  message: 'Translation saved successfully!')

def copy_to_clipboard():
    translated_text = text_entry2.get( index1: "1.0",  index2: "end-1c").strip()
    if not translated_text:
        messagebox.showerror( title: 'Copy to Clipboard',  message: 'No text available to copy!')
        return
    root.clipboard_clear()
    root.clipboard_append(translated_text)
    messagebox.showinfo( title: 'Copy to Clipboard',  message: 'Translation copied to clipboard!')

def view_history():
    if not history:
        messagebox.showinfo( title: "History",  message: "No translations in history yet!")
        return
    history_window = Toplevel(root)
    history_window.title("Translation History")
    history_window.geometry("500x400")
    history_window.configure(bg='#A9D6E5')
    Label(history_window, text="Translation History", font=("Helvetica 15 bold"), bg='#A9D6E5').pack(pady=10)
    history_list = Text(history_window, wrap=WORD, font=('verdana', 10), bg="white", fg="black")
    history_list.pack(expand=True, fill=BOTH, padx=10, pady=10)
    for idx, (src_text, src_lang, tgt_lang, translated_text) in enumerate(history, 1):
        history_list.insert(END,  chars: f"{idx}. {src_lang.upper()} → {tgt_lang.upper()}\n")
        history_list.insert(END,  chars: f"   Original: {src_text}\n")
        history_list.insert(END,  chars: f"   Translation: {translated_text}\n\n")

def exit_app():
    root.destroy()

languages = [
```



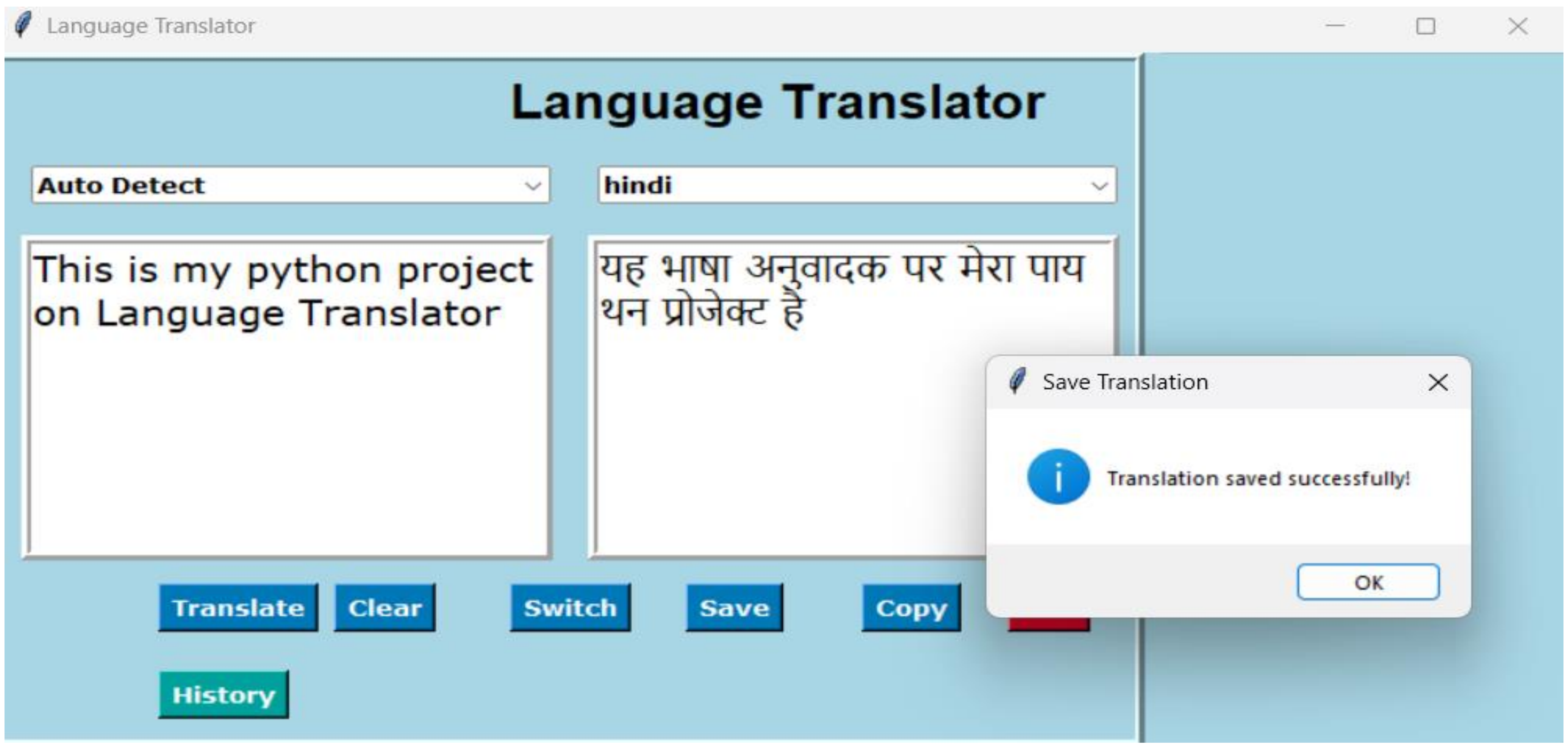
```

languages = [
    "english", "telugu", "hindi", "bengali", "marathi", "tamil", "urdu", "gujarati", "kannada",
    "malayalam", "punjabi", "assamese", "oriya", "spanish", "french", "german", "italian", "chinese",
    "japanese", "korean", "russian", "portuguese", "arabic", "turkish", "vietnamese", "thai", "dutch",
    "swedish", "greek", "polish", "czech", "danish", "finnish", "hebrew", "norwegian", "hungarian",
    "indonesian", "malay", "swahili", "zulu"
]

source_language = ttk.Combobox(frame1, width=27, state='readonly', font=('verdana', 10, 'bold'))
source_language['values'] = ["Auto Detect"] + languages
source_language.place(x=15, y=60)
source_language.current(0)

target_language = ttk.Combobox(frame1, width=27, state='readonly', font=('verdana', 10, 'bold'))
target_language['values'] = languages
target_language.place(x=305, y=60)
target_language.current(0)

text_entry1 = Text(frame1, width=20, height=7, borderwidth=5, relief=RIDGE, font=('verdana', 15))
text_entry1.place(x=10, y=100)

text_entry2 = Text(frame1, width=20, height=7, borderwidth=5, relief=RIDGE, font=('verdana', 15))
text_entry2.place(x=300, y=100)

btn1 = Button(frame1, command=translate, text="Translate", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn1.place(x=80, y=300)

btn2 = Button(frame1, command=clear, text="Clear", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn2.place(x=170, y=300)

btn3 = Button(frame1, command=switch_text, text="Switch", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
```


```python
text_entry2 = Text(frame1, width=20, height=7, borderwidth=5, relief=RIDGE, font=('verdana', 15))
text_entry2.place(x=300, y=100)

btn1 = Button(frame1, command=translate, text="Translate", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn1.place(x=80, y=300)

btn2 = Button(frame1, command=clear, text="Clear", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn2.place(x=170, y=300)

btn3 = Button(frame1, command=switch_text, text="Switch", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn3.place(x=260, y=300)

btn4 = Button(frame1, command=save_translation, text="Save", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn4.place(x=350, y=300)

btn5 = Button(frame1, command=copy_to_clipboard, text="Copy", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#0077B6', fg="white", cursor="hand2")
btn5.place(x=440, y=300)

btn6 = Button(frame1, command=view_history, text="History", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#00A19D', fg="white", cursor="hand2")
btn6.place(x=80, y=350)

btn7 = Button(frame1, command=exit_app, text="Exit", relief=RAISED, borderwidth=2, font=('verdana', 10, 'bold'), bg='#D90429', fg="white", cursor="hand2")
btn7.place(x=515, y=300)

root.mainloop()
```

Here we can see the interface of my project



Here we can see I am using save option to save


Language Translator

Language Translator

Auto Detect

hindi

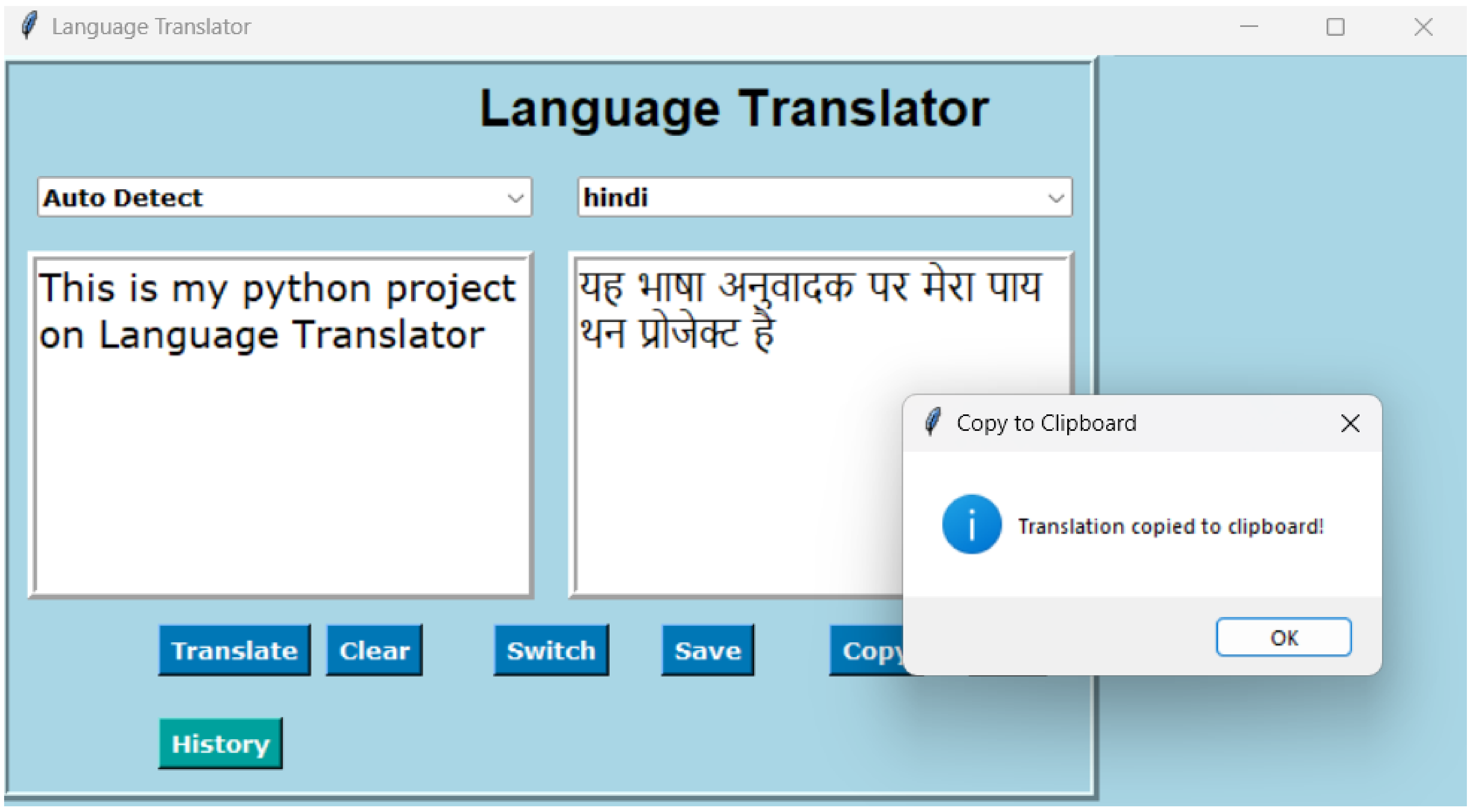
This is my python project on Language Translator

यह भाषा अनुवादक पर मेरा पाय थन प्रोजेक्ट है

Save Translation

Translation saved successfully!

Here we can see I am using copy option to copy


Language Translator

Language Translator

Auto Detect

hindi

This is my python project on Language Translator

यह भाषा अनुवादक पर मेरा पाय थन प्रोजेक्ट है

Copy to Clipboard

Translation copied to clipboard!

VII. TESTING AND VALIDATION

The goal of unit testing is to independently confirm that each component—functions and modules—is functioning correctly**.**

1. Unit Testing

The goal of unit testing is to verify that individual functions and modules in the LanguageTranslatorApp work as expected.

Items to be Tested

1. Confirm that the get_lang_code function returns the correct language code based on the language name.

```
def test_get_lang_code():
    app = LanguageTranslatorApp(None)  # Simulate without GUI
    assert app.get_lang_code("english") == "en", "Failed to retrieve code for English"
    assert app.get_lang_code("hindi") == "hi", "Failed to retrieve code for Hindi"
    assert app.get_lang_code("nonexistent") == "en", "Default language code fallback failed"
```

2. Translation Functionality:

Verify that the translate method translates text from one language to another accurately**.**

```
def test_translation_functionality():
    app = LanguageTranslatorApp(None)
    input_text = "Hello" app.lang1.set("english")
    app.lang2.set("hindi")
    translation = app.translator.translate(input_text, src="en", dest="hi").text
    assert translation == "नम ते", "Translation failed or incorrect"
```

2. Integration Testing

The goal of integration testing is to verify that the various components of the app function together as expected.

1. Translation and History Integration:
Verify that translating text adds the correct entry to the history**.**

```
def test_translation_and_history_integration(): app
   = LanguageTranslatorApp(None)
   app.text_entry1.insert("1.0", "Hello")
   app.lang1.set("english") app.lang2.set("hindi")
   app.translate()
   assert app.text_entry2.get("1.0", tk.END).strip() == "नम ते◌
                                                                                  , "Translation output incorrect"
   assert app.history[-1] == ("Hello", "en", "नम ते◌ , "hi"), "History entry incorrect"
```

2. Error Handling for Empty Input:
Confirm that an error message is displayed if no input is provided for translation**.**

```
def test_empty_translation_input():
   app = LanguageTranslatorApp(None)
   app.text_entry1.delete("1.0", tk.END)  # Ensure input is empty try:
      app.translate()
   except Exception:
      pass
```

VIII. CONCLUSION

The Language Translator application is a Python-based tool designed to facilitate seamless text translation between multiple languages. Built using the Tkinter framework, the application offers an intuitive and interactive graphical interface, allowing users to input text, select a target language, and receive accurate translations in real-time. By integrating the Google Translate API, the application ensures reliable translations across over 35 supported languages, including English, Spanish, Chinese, and Arabic.

The project addresses common challenges of language barriers by providing a simple yet powerful solution for personal and professional use. Key features include multi-language support, error handling mechanisms to validate user input, and a "Clear" functionality for resetting the input and output fields. These features enhance the user experience and make the application accessible to users with varying levels of technical expertise.

The design of the application emphasizes modularity and scalability. Core functions, such as `translate()` for handling translations and `clear()` for resetting fields, are implemented to ensure smooth and efficient operation. The Tkinter interface is designed for simplicity, with dropdown menus for language selection and text boxes for input and output, making it visually appealing and easy to use**.**

The project leverages essential tools and technologies, including Python for programming, the Google Translate API for translations, and Tkinter for the GUI. Additional libraries like googletrans and tkinter.ttk are used to support advanced functionality. This robust combination of tools ensures that the application operates efficiently while remaining adaptable for future enhancements.

Looking ahead, the Language Translator has the potential for significant upgrades. Future iterations could include features such as voice-based translation, offline capabilities using pre-trained models, and document translation for handling file inputs. Customizable themes and accessibility options could further broaden the application's appeal.

In summary, the Language Translator is a practical and versatile application that addresses the need for reliable, real-time text translation. Its user-friendly design and robust functionality make it a valuable tool for breaking down language barriers and fostering effective communication in a globalized world.

IX. REFERENCES

1. **tkinter (Python Standard Library)**
   - Used for creating GUI applications in Python.
   - Widgets such as Frame, Button, Text, and Label are utilized in the code for UI layout.
   - Documentation: tkinter Documentation
2. **googletrans (Google Translator API for Python)**
   - Facilitates translation of text between multiple languages using the Google Translate service.
   - The Translator class provides methods like translate for translation tasks.
   - The LANGUAGES dictionary provides supported language codes.
   - GitHub Repo: googletrans
   - Documentation: Google Translate API

<u>THANK YOU</u>