Review - 2

Date : 20-02-24

Language Translator using Google API

CELEBRATING 25 YEARS

CMRIT

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.

ACCREDITED WITH A+ GRADE BY NAAC

1CR21CS151	Rohan N Karnkoti
1CR21CS214	Vismitha S

Under the guidance of:

Mrs. Manjula Subramaniam

Department of Computer Science & Engineering

TITLE

Language Translator using Google API

PROBLEM STATEMENT

Develop a python project which does language translator using Google API. Use any native language

DETAILED REQUIREMENTS

- Knowledge of Python
- Understanding of Google API
- Understanding of tkinter library to take in voice input and display output.
- Understanding of googletrans and speech_recognition libraries.

SOFTWARE TOOLS USED

- Visual Studio Code
- Git
- GitHub
- Python 3.11
- Tkinter library
- Googletrans library
- Speech_recognition library
- Gtts library

PROPOSED SOLUTION

- **Graphical User Interface (GUI)**: Create a basic GUI using tkinter with a single button (e.g., microphone symbol) to initiate voice input.
- **Voice Input**: Implement voice input using the *speech_recognition* library when the user clicks the microphone button.
- **Translation**: Utilize the googletrans library to translate the captured voice input into a target language.
- **Text-to-Speech Output**: Use the gTTS library to convert the translated text into speech for output.

PROPOSED SOLUTION

- **Asynchronous Processing**: Implement threading to ensure the GUI remains responsive during voice capture and translation.
- **Error Handling**: Handle potential errors during voice recognition and translation to provide a smooth user experience.
- **Dynamic Language Selection**: Allow users to dynamically select the target language (e.g., through a dropdown or input).
- **User Feedback**: Provide clear feedback to the user, such as displaying the translated text on the GUI and potentially using visual indicators for the translation process.

TOTAL HOUR & WORKS

This project will require around 40 days.
20 days for one person.

DESIGN

User Interface:

- Microphone Button: Allows users to trigger voice input.
- Language Dropdown: Enables users to select the target language for translation.
- Output Label: Displays the translated text to the user.

Translation Process:

- Speech Recognition: Utilizes the *speech_recognition* library to convert speech input into text.
- Language Detection: Determines the language of the input text using the *googletrans* library.
- Translation: Translates the input text into the selected target language using the *googletrans* library.
- Text-to-Speech Conversion: Converts the translated text into speech using the *gtts* library.

IMPLEMENTATION

Libraries Used:

- tkinter: for building the graphical user interface.
- googletrans: for language detection and translation.
- speech_recognition: for converting speech input to text.
- gtts: for converting text to speech.

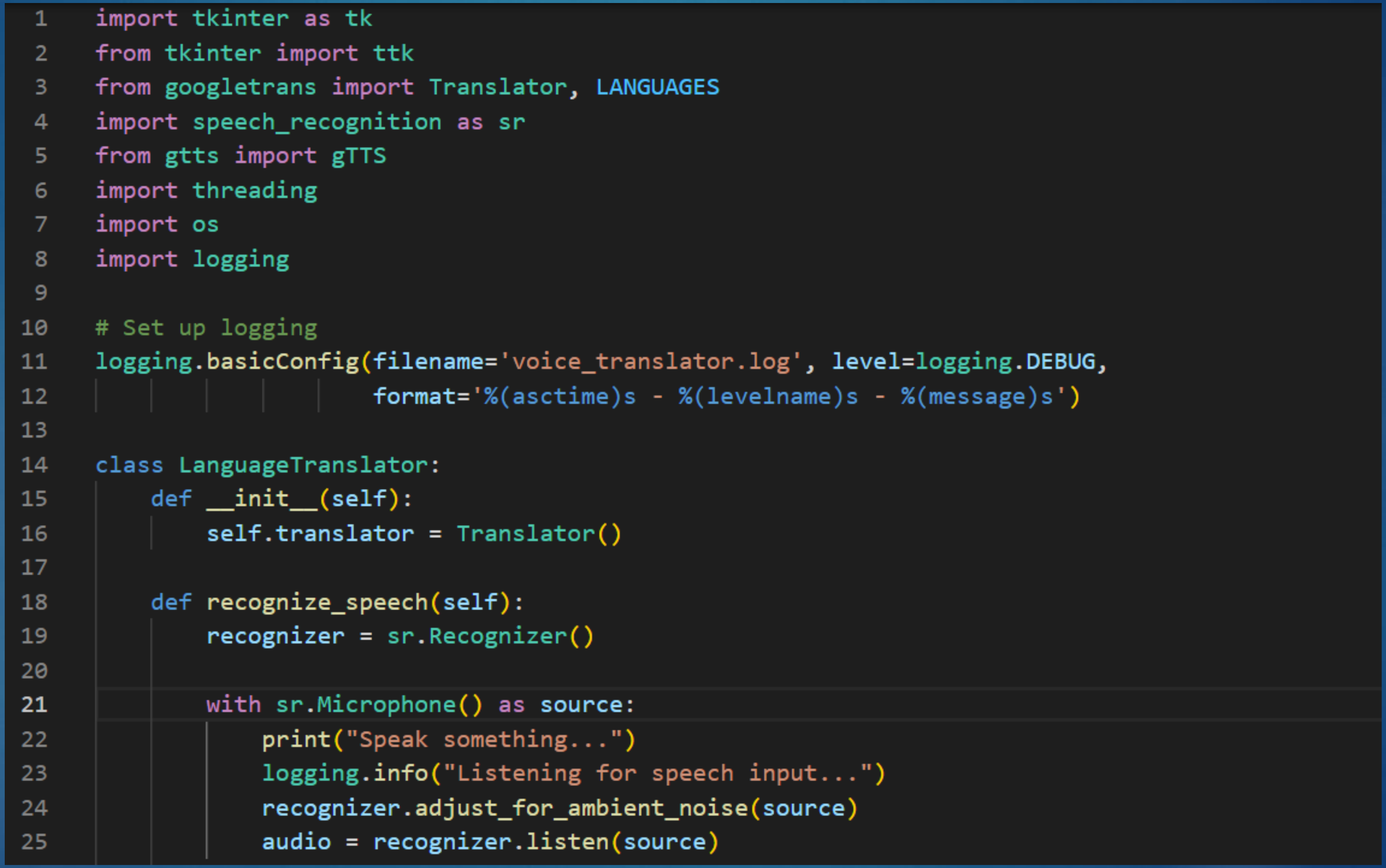
Classes and Functions:

- LanguageTranslator Class: Contains methods for speech recognition, translation, and text-to-speech conversion.
- TranslatorApp Class: Constructs the GUI and handles translation operations.
- Main Function: Initializes the application and starts the main event loop.

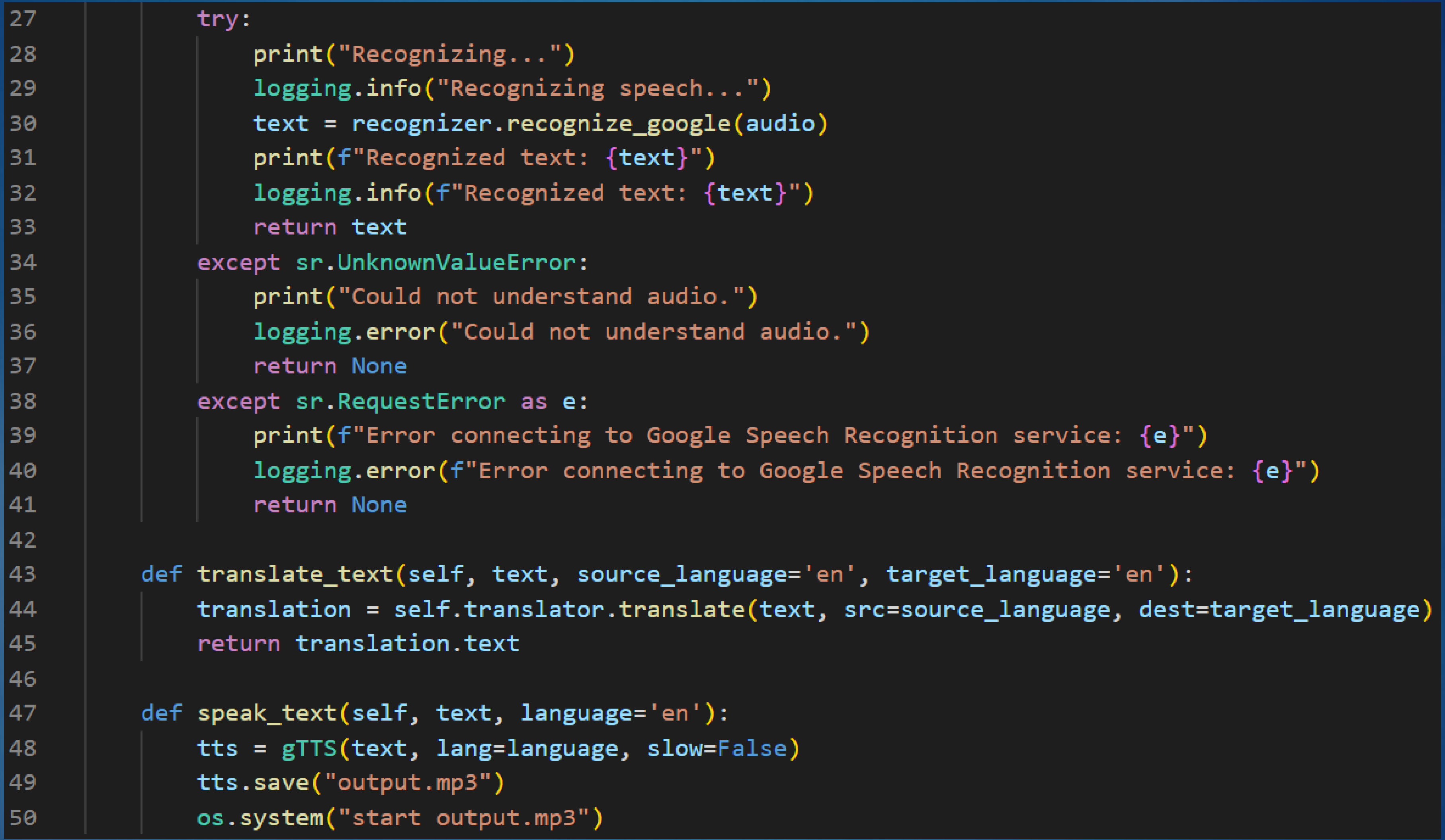
Workflow:

- The user clicks the microphone button to start voice input.
- The application listens for speech input using the *speech_recognition* library.
- The recognized text is sent for language detection to determine the source language.
- The detected language is used to translate the text into the selected target language.
- The translated text is displayed on the GUI and converted into speech for the user.

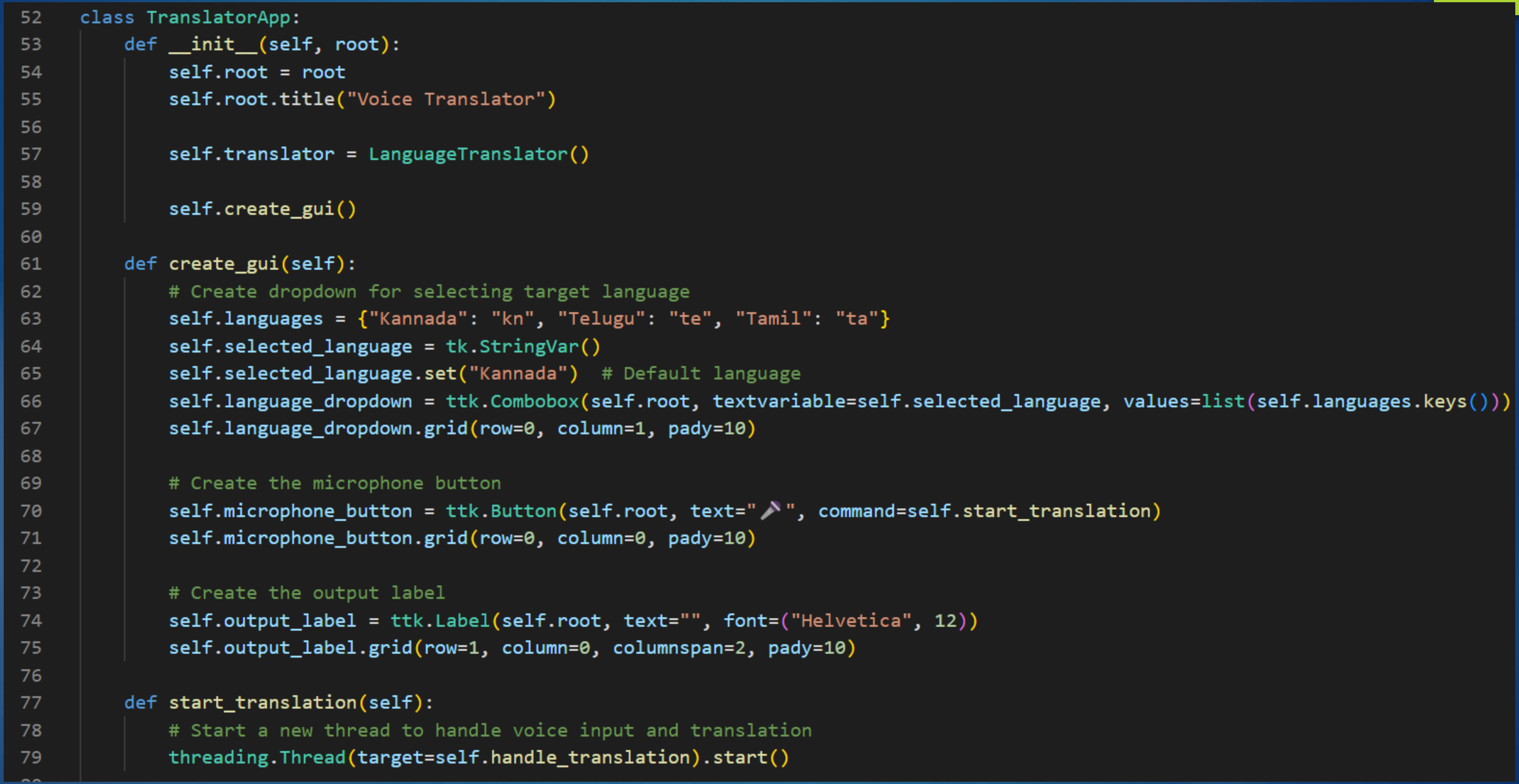
SOURCE CODE



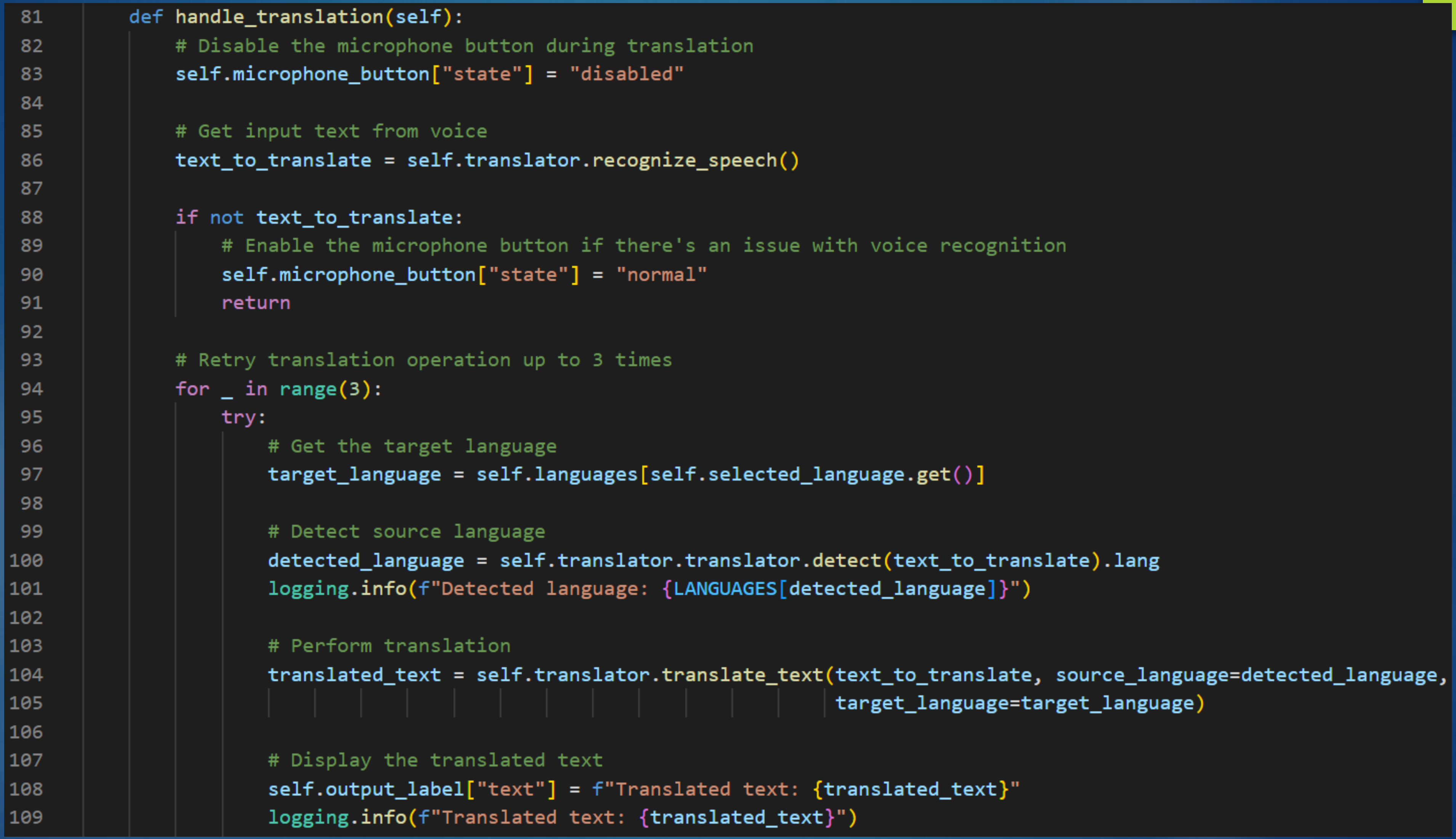
```python
import tkinter as tk
from tkinter import ttk
from googletrans import Translator, LANGUAGES
import speech_recognition as sr
from gtts import gTTS
import threading
import os
import logging

# Set up logging
logging.basicConfig(filename='voice_translator.log', level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %(message)s')

class LanguageTranslator:
    def __init__(self):
        self.translator = Translator()

    def recognize_speech(self):
        recognizer = sr.Recognizer()

        with sr.Microphone() as source:
            print("Speak something...")
            logging.info("Listening for speech input...")
            recognizer.adjust_for_ambient_noise(source)
            audio = recognizer.listen(source)
```



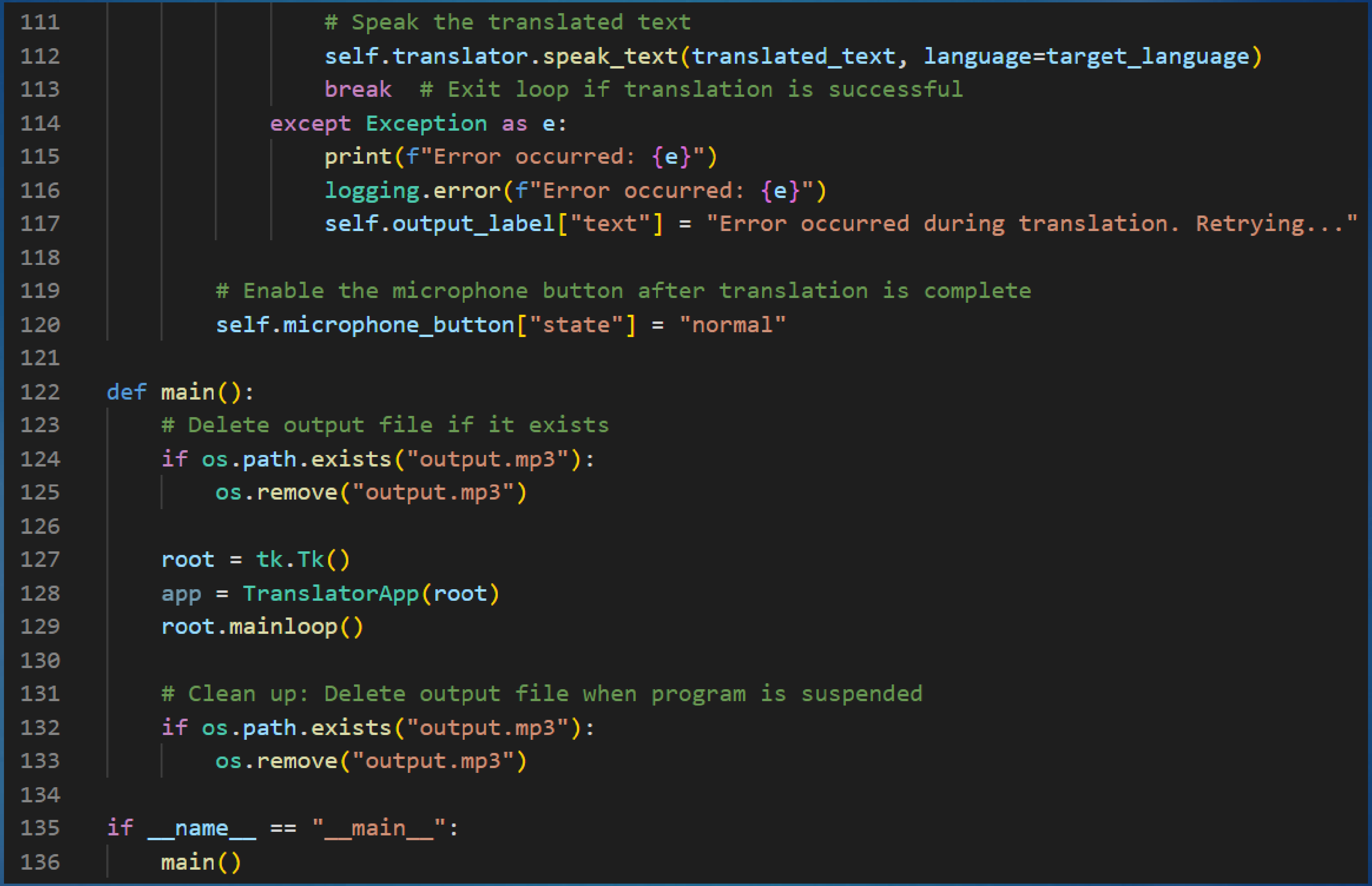
```python
        try:
            print("Recognizing...")
            logging.info("Recognizing speech...")
            text = recognizer.recognize_google(audio)
            print(f"Recognized text: {text}")
            logging.info(f"Recognized text: {text}")
            return text
        except sr.UnknownValueError:
            print("Could not understand audio.")
            logging.error("Could not understand audio.")
            return None
        except sr.RequestError as e:
            print(f"Error connecting to Google Speech Recognition service: {e}")
            logging.error(f"Error connecting to Google Speech Recognition service: {e}")
            return None

    def translate_text(self, text, source_language='en', target_language='en'):
        translation = self.translator.translate(text, src=source_language, dest=target_language)
        return translation.text

    def speak_text(self, text, language='en'):
        tts = gTTS(text, lang=language, slow=False)
        tts.save("output.mp3")
        os.system("start output.mp3")
```



```python
class TranslatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Voice Translator")

        self.translator = LanguageTranslator()

        self.create_gui()

    def create_gui(self):
        # Create dropdown for selecting target language
        self.languages = {"Kannada": "kn", "Telugu": "te", "Tamil": "ta"}
        self.selected_language = tk.StringVar()
        self.selected_language.set("Kannada")  # Default language
        self.language_dropdown = ttk.Combobox(self.root, textvariable=self.selected_language, values=list(self.languages.keys()))
        self.language_dropdown.grid(row=0, column=1, pady=10)

        # Create the microphone button
        self.microphone_button = ttk.Button(self.root, text="🎤", command=self.start_translation)
        self.microphone_button.grid(row=0, column=0, pady=10)

        # Create the output label
        self.output_label = ttk.Label(self.root, text="", font=("Helvetica", 12))
        self.output_label.grid(row=1, column=0, columnspan=2, pady=10)

    def start_translation(self):
        # Start a new thread to handle voice input and translation
        threading.Thread(target=self.handle_translation).start()
```

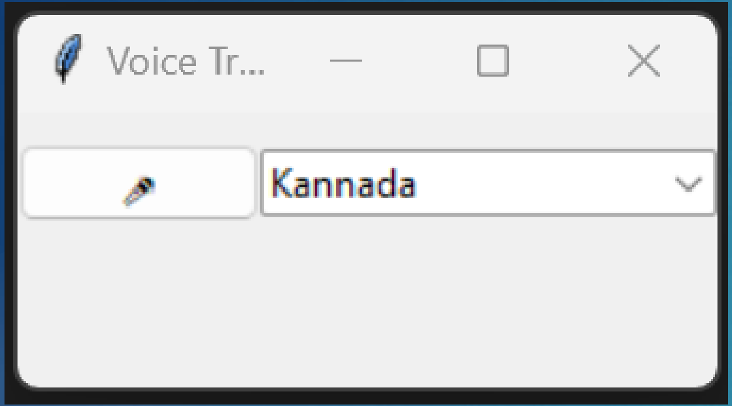


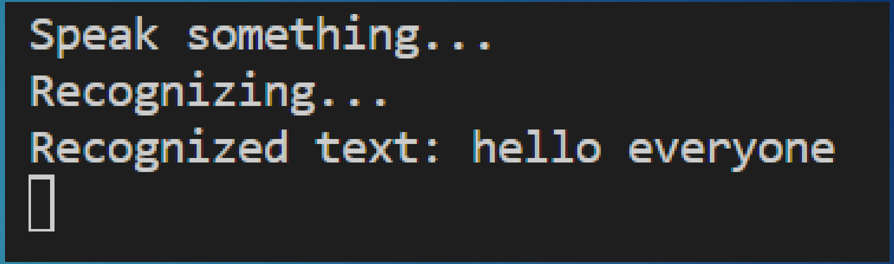
```
    def handle_translation(self):
        # Disable the microphone button during translation
        self.microphone_button["state"] = "disabled"

        # Get input text from voice
        text_to_translate = self.translator.recognize_speech()

        if not text_to_translate:
            # Enable the microphone button if there's an issue with voice recognition
            self.microphone_button["state"] = "normal"
            return

        # Retry translation operation up to 3 times
        for _ in range(3):
            try:
                # Get the target language
                target_language = self.languages[self.selected_language.get()]

                # Detect source language
                detected_language = self.translator.translator.detect(text_to_translate).lang
                logging.info(f"Detected language: {LANGUAGES[detected_language]}")

                # Perform translation
                translated_text = self.translator.translate_text(text_to_translate, source_language=detected_language,
                                                                 target_language=target_language)

                # Display the translated text
                self.output_label["text"] = f"Translated text: {translated_text}"
                logging.info(f"Translated text: {translated_text}")
```

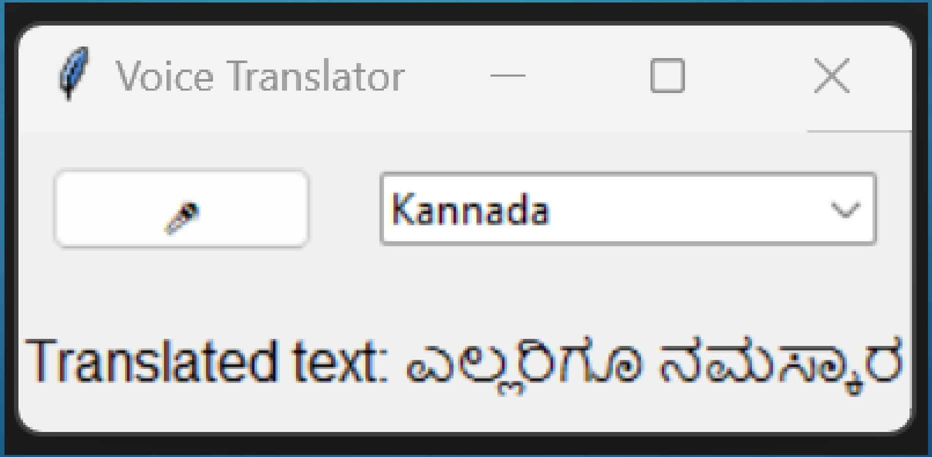


```
                    # Speak the translated text
                    self.translator.speak_text(translated_text, language=target_language)
                    break  # Exit loop if translation is successful
                except Exception as e:
                    print(f"Error occurred: {e}")
                    logging.error(f"Error occurred: {e}")
                    self.output_label["text"] = "Error occurred during translation. Retrying..."

            # Enable the microphone button after translation is complete
            self.microphone_button["state"] = "normal"

def main():
    # Delete output file if it exists
    if os.path.exists("output.mp3"):
        os.remove("output.mp3")

    root = tk.Tk()
    app = TranslatorApp(root)
    root.mainloop()

    # Clean up: Delete output file when program is suspended
    if os.path.exists("output.mp3"):
        os.remove("output.mp3")

if __name__ == "__main__":
    main()
```

OUTPUT




```
Speak something...
Recognizing...
Recognized text: hello everyone
```



Translated text: ಎಲ್ಲರಿಗೂ ನಮಸ್ಕಾರ

Thank you