

ECEE 2140: Computing Fundamentals for Engineers  
Self-Tutor Translator

Git Project: <https://github.com/KabatoBurka/Translation-Tutor>

Kabato Burka

## **Introduction**

The purpose of the Self-Tutor Translation system was for a user to learn a multitude of languages by simply saying any phrase in English and requesting a language they would like to not only read but also hear. With a friendly user interface, the user has three options: look at the 106 languages offered to translate English into, conduct a translation only using the user's voice, and continue the system until they stop translation.

In order to run the code, one must have a Python code editor, such as Visual Studio Code, as that is what the system was written using. Listed in the GitHub project link are the libraries and tools necessary to run the system, which are Playsound, PyAudio, SpeechRecognition, googletrans, and gTTS. To install them, one can access the Visual Studio Code Command Prompt and type “pip install” following a space and then the name of the library or tool. Once everything is downloaded, the user can run the code in the “main” file. Figure 1 shows what the user should be able to do once they run the program.

```
-----  
Self-Tutor Translator  
-----  
1. Language List  
2. Translate  
3. Done  
-----  
Option: 1  
- Afrikaans  
- Albanian  
- Amharic  
- Arabic  
- Armenian  
- Azerbaijani  
- Basque  
- Belarusian  
-----  
Self-Tutor Translator  
-----  
1. Language List  
2. Translate  
3. Done  
-----  
Option: 2  
Speak...  
Wait...  
Heard: I hope I pass this class  
Say desired language:  
Speak...  
Wait...  
Heard: traditional Chinese  
Translation: 我希望我能通過這門課  
-----  
Self-Tutor Translator  
-----  
1. Language List  
2. Translate  
3. Done  
-----  
Option: 3  
Bye Bye <3
```

**Figure 1:** The three presented options for the user to interact with by simply typing the number for the corresponding action

## Application Design

There are four main classes within the system that are essential for the most significant part of the code, which is imputing the voice and translating it. These classes are called Mic, Translation, UseTranslation, and GenerateLang.

The Mic class contains the attribute *speech*, which is used for whatever the user will say in the method *to\_listen*. The *to\_listen* method then turns speech if it is recognized by the PC microphone. The recognition is a class called Recognizer imported from Speech Recognition that can detect if what is said is within its comprehension of words. However, this Recognizer depends on the Microphone class that uses the PC's default audio as the source.

The Translation class contains the method *find\_language* containing the object *mic*. Within the method, it checks if there is anything said that is within the dictionary of language option; if there is, it will return that language said in the variable *switch\_lang*.

The UseTranslation class initializes its own attribute *switch\_lang* and contains the method *take\_dic* containing the object *mic*. The method uses the googletrans *translate* method to what was said from the *mic.speech*. It then prints the text and uses the gTTs class to say that translated text into speech.

The GenerateLang class brings the classes together in a more formulaic way when the user chooses to start translating. In the class' *all* method, a phrase variable is created as a new Mic object for the phrase to be translated, which is connected to *speech* whatever is said by the user through the *to\_listen* method. A new Mic object is recreated with the *language* variable connected to the *speech* but equal to "None." The *switch\_lang* is then used as the out of the *find\_langauge* method, which listens to the user, calls *to\_listen*, and shows validity with either the language (valid) or "None" (invalid). From there, in the *find\_language*, if "None," it returns to the menu; if it is not "None", the speech will be translated. The translation is then played through *take\_dic* if applicable.

Of course, there are more classes, such as a class containing the loop to list all the languages neatly (LongList class), the simplistic interface (MainMenu class), and the interface view for the user (TitleScreen class), but they do not conduct the necessary tasks in which the system is made for. Figure 2 contains the class diagram of the system.

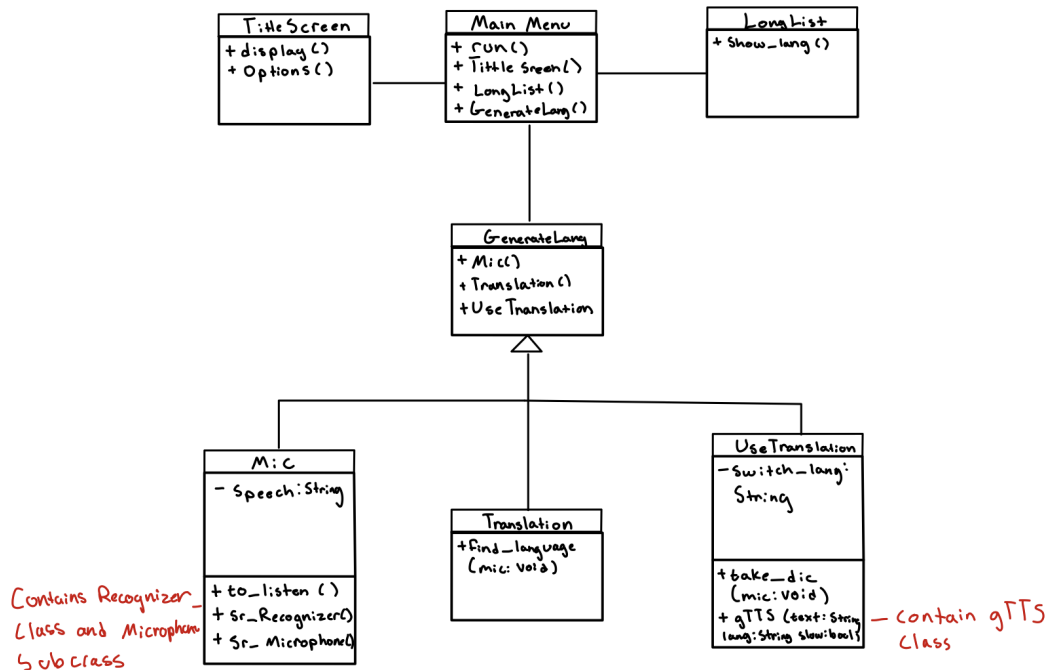


Figure 2: Self-Tutor Translator class diagram

## Libraries and Tools

As aforementioned, the necessary libraries and tools for this project included Playsound, PyAudio, SpeechRecognition, googletans, and gTTS.

The Playsound<sup>4</sup> is a simple Python module that plays any audio files. This means in order for the user to do anything, an MP3 file must be created. This is done by creating an mp3 with the OS module to be deleted after playing. The PyAudio<sup>5</sup> library allows a user to record and play any sound. With the Pyaudio, the SpeechRecognition<sup>6</sup> library contains multiple APIs that can identify words in speech. Specifically, the Google speech recognition API module was used. The googletans<sup>1</sup> library uses 106 available languages. Finally, the gTTS<sup>2</sup> is another Google library API that can speak any 106 languages from text to aloud. To get the entire dictionary, the Google Cloud Language Support<sup>3</sup> was used.

All these libraries and tools in tandon allow the second option within the system to function.

## Testing

From the project's inception, there were a multitude of troubles faced to get the code working. To start, the googletans import refused to work with its newest update and somehow only functioned in the previous update (this is mentioned in the GitHub directions). In addition to the import problems, there were and still are troubles importing different files within the code. These

imports only worked in a certain folder on my personal computer, so unit tests could not be successfully conducted without errors on the imports themselves. The only way tests were conducted was through that working folder.

Outside of trivial problems not consisting of the actual code, there was a bug that the program would not continue the system until the user said a valid language. To avoid the user from being stuck for not knowing any languages to say, the response for a language was limited to one attempt to say something valid or taking too long to respond before returning to the main menu. However, this brought another bug, in which if the user said something invalid, an error would occur. This was avoided by using “None” as the return if something within the language dictionary was not said.

There are no overtly evident problems remaining in the code, but the only unfriendly feature within the code is that a user must say something in English before continuing to the following prompt, rather than waiting for the user to say something in a certain amount of time similar to the language request prompt.

Despite the problem with running import and a very minor bug for the user response, making a working system was still possible with the time given.

## **Conclusion**

In the Self-Tutor Translator project, I learned how to use libraries and APIs from Google properly. These tools made it significantly more straightforward to write the code. On top of this, I learned how to troubleshoot problems by using the debugging feature.

If given more time in the project, I would like to add more features. For example, It would be convenient for the user to type an input in English and get the same output if they chose to say it aloud instead. Having all the translations saved onto a file for later use could also be helpful. Additionally, using Tkinter to make a more friendly interface would be much more appealing than using code in the terminal.

To future EECE 2140 students, I suggest they start as soon as their time allows them to because they will run into several problems that may not concern their code but other things surrounding it. More time should be spent on the code than fixing other things, so they should be wary of those problems.

## **References**

1. “Googletrans,” *PyPI*. [Online]. Available: <https://pypi.org/project/googletrans/>. [Accessed: 10-Nov-2022].
2. “GTTS,” *PyPI*. [Online]. Available: <https://pypi.org/project/gTTS/>. [Accessed: 10-Nov-2022].
3. “Language support | cloud translation | google cloud,” *Google*. [Online]. Available: <https://cloud.google.com/translate/docs/languages>. [Accessed: 10-Nov-2022].
4. “Playsound,” *PyPI*. [Online]. Available: <https://pypi.org/project/playsound/>. [Accessed: 10-Nov-2022].
5. “Pyaudio,” *PyPI*. [Online]. Available: <https://pypi.org/project/PyAudio/>. [Accessed: 10-Nov-2022].
6. “Speechrecognition,” *PyPI*. [Online]. Available: <https://pypi.org/project/SpeechRecognition/>. [Accessed: 10-Nov-2022].