

****

**Midterm Project Report**

**Advanced Computer Programming**

**Web Scraping with Python**

**Student Name : Igor Evtushenko**

**Student ID**  **: 112021179**

**Teacher : DINH-TRUNG VU**

**2024-04**

# Chapter 1 Introduction

## GitHub

1. **Personal GitHub Account**: https://github.com/112021179
2. **Group Project Repository**: https://github.com/112021179/1
3. **My Project Repository** https://github.com/112021179/1/tree/main/midterm/112021179
4. **List of** **submitted file**s:
   * **main**.py
   * **try.csv**

## Topic

Build a web scraper

## Project Overview

This project was done on a whim because my friend asked me to help him. The idea is quite simple. There’s an app ‘Re:Word Chinese’, which can help you learn new words. Unfortunately, basic capabilities are quite poor, the apps’ words repository is small and is poorly organized. My friend wants to get ready for TOCFL. All the words for different levels were found, the only thing that is left is to go to <https://www.trainchinese.com/v2/index.php>, it’s one of the best sites for Chinese translation, and take translation, pinyin, and the examples for the words themselves. Doing this manually turned out to be a hassle and took too much time. So, a webscraper would be of such a help in this task. You take a table of words for each level as an input and get the csv file as an output.

Main requirements: find the Chinese word, pinyin and translation for it, if the word cannot be found, just skip it; examples should be phrases, at least 4 char long, if there’re no examples, just skip them.

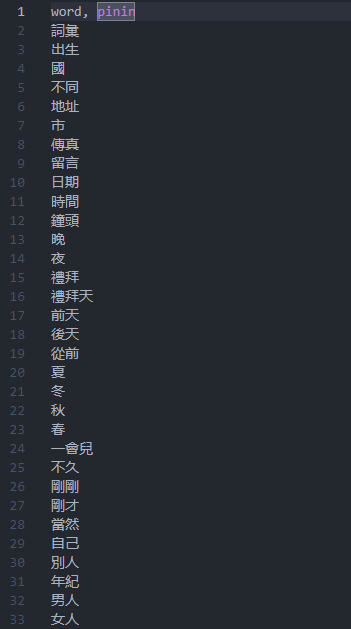
I used these libraries: csv, os, re, pandas, requests, bs4, threading, queue and time. Basically, time and pandas are unnecessary, but I still want to see how much time it takes for each word to be scraped and pandas was implemented early into the project and works like a charm. Queue is needed to queue all the data. Otherwise, writing the data in a csv file, once the thread finishes, can corrupt the data, since multiple threads might be trying to write to the file at the same time.

The “advanced language features,” would probably be implementation of multithreading and data queueing. For the most part it’s a simple webscraper, that tries to follow the websites’ logic and hierarchy.

# Implementation

## Input

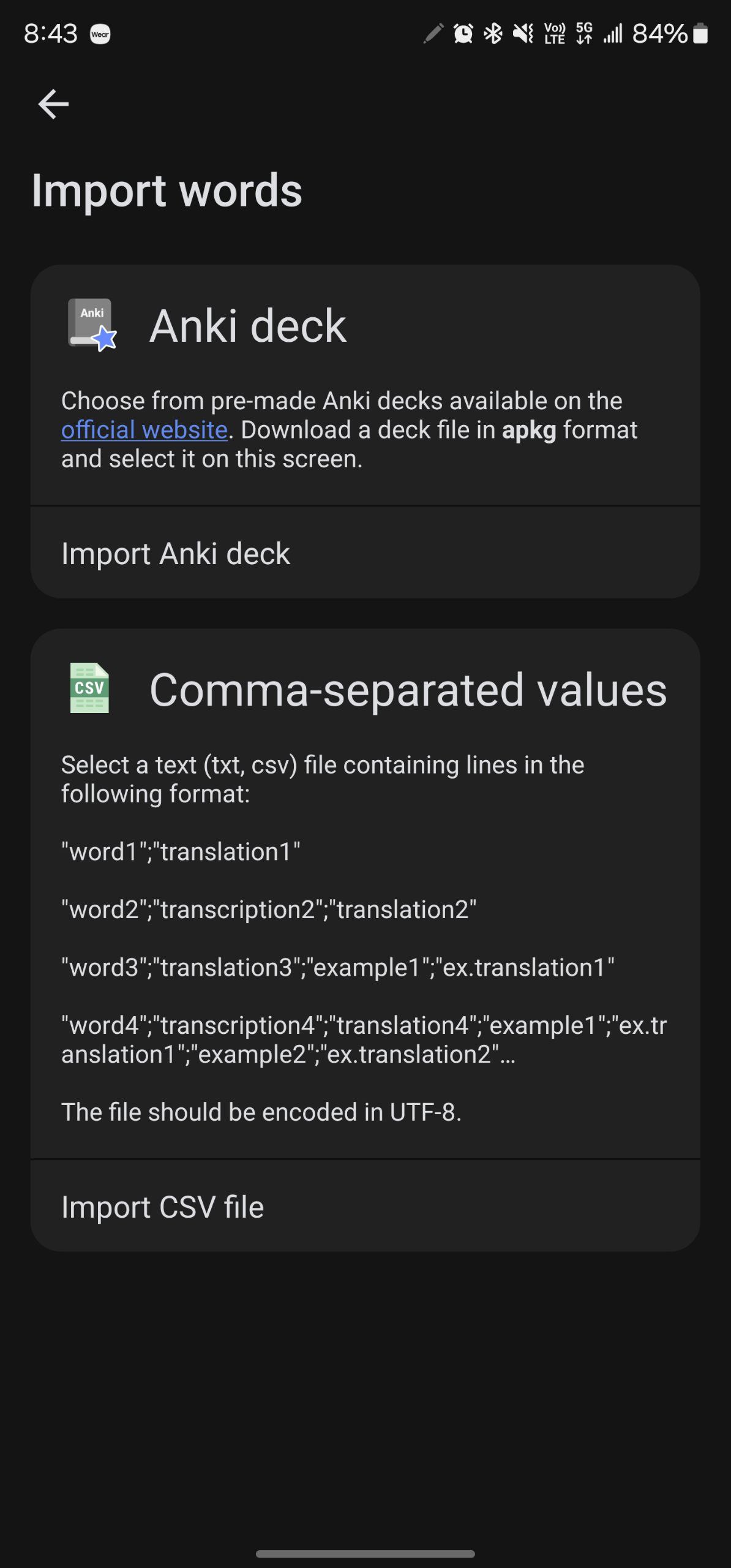
As an input you get a csv file that looks something like this

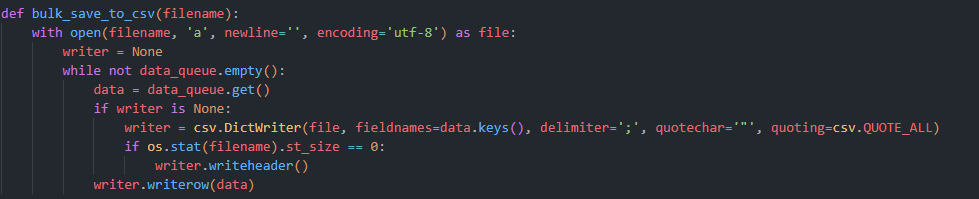


I read it with pandas. For the most part words look like this ‘市,’ but they can appear with added spacing ‘市 ‘, which means we need a second meaning from the site. There’re also some words that look like this ‘一點/一點兒’, the slash means that there’re are the same, so I just strip the word, and take the first one. Otherwise, the website is unable to find the word.

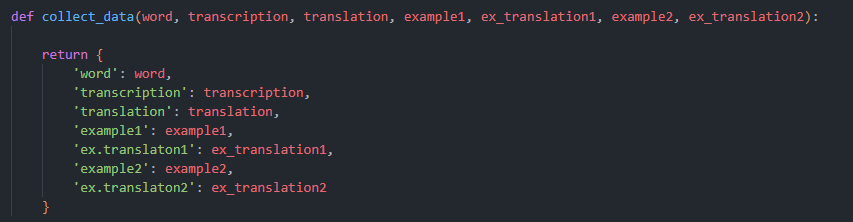
## Output

Here’s the required format for the csv file we want to import





That’s how I implement writing to the file, specifying the delimiter, and quoting all the values as the format specifies (otherwise the imported words will be a mess).



And that is what my data looks like

## Main Logic

It is actually a bunch of functions, no classes, nothing fancy. I didn’t feel the need to implement classes here. There’s nothing that the user itself specifies. However, that should be fairly simple to implement with a GUI.

The main hiccup is that we have to make 2 requests to the site per word, first request

[https://www.trainchinese.com/v2/search.php?searchWord=chicken&rAp=0&height=0&width=0&tcLanguage=en], that’s the link to the website, we can see that we can search for words, just by specifying parameter ‘searchWord’. The first request gives us the understanding, whether we have found the word, if we did - get its translation, pinyin and the word\_id. What’s the last one for? Well, we need 2 examples, we can’t find them on the main page, which only displays the meanings, we must go deeper and click on the word we need. The problem here is that the second page requires us to have a word\_id to access it,

[https://www.trainchinese.com/v2/wordDetails.php?rAp=0&wordId=2546&tcLanguage=en]

and I have no idea how to get it, except to make a first request to find it, otherwise I would be doing only one request to the site to scrape the data.

I will only go through the crucial functions or else this document will be way too long.

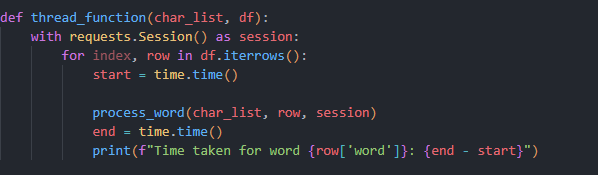
**2.3.1 def main**

****

It’s rather straightforward, specify the file name, read the file, specify the number of threads you want to run, calculate the chunk for each thread and we’re ready to go. Starting the loop, starting the threads and that’s it.

Char list is just a list of characters that the website uses in the examples.

**2.3.2 def thread function**



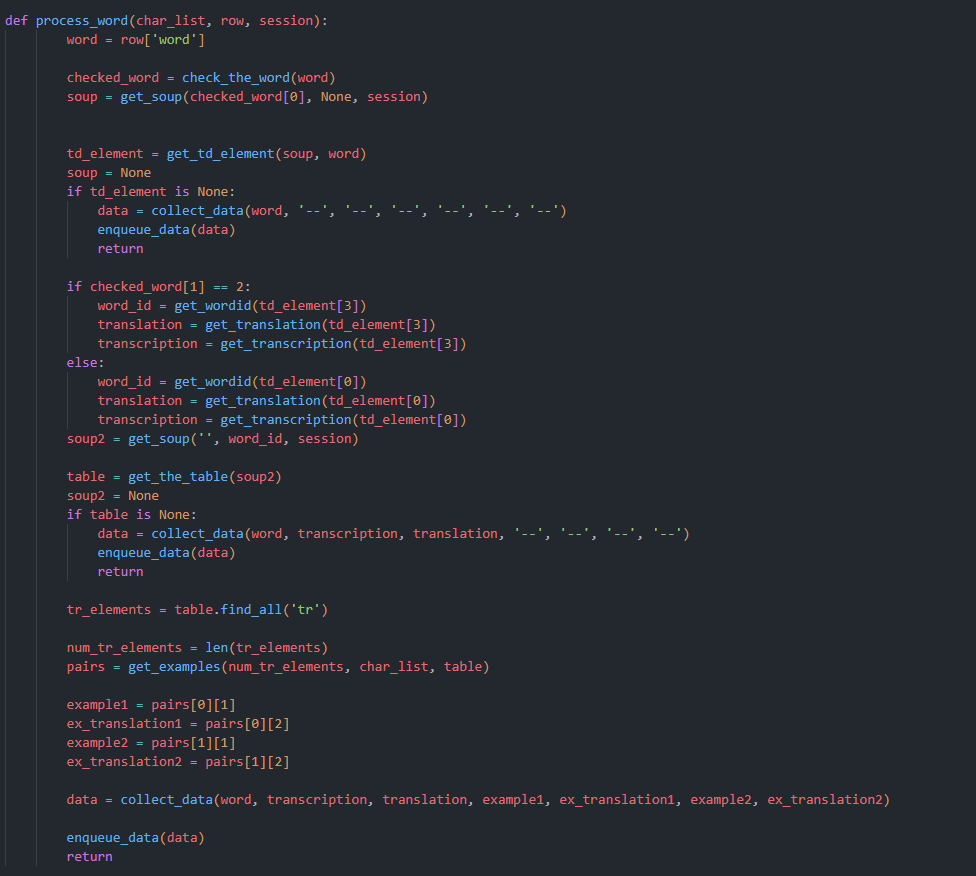
This loops over all the words that our thread was assigned, and we continue to our next function.

Also, I create a separate session for each thread

requests.Session() as session

as it’s not time efficient to open the session every time you make a request. This actually saves quite some time(on 30 threads on average 8 seconds per word, and that improves to ~2 seconds per word).

**2.3.3 def process word**

****

This one is quite chunky.

First, I get the word from the row, then I have to check it for added spacing or if there’s a slash in the word.

After this, we get the soup, and get the td element, which is a table of different meanings/other instances.

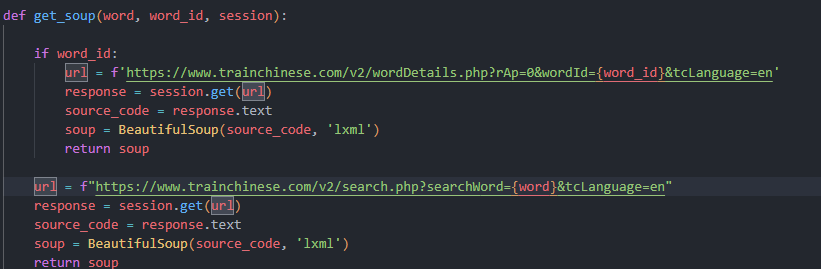
If it’s None, that means we didn’t find anything, and we can return an empty result.

checked\_word is a tuple; the second value tells us whether we need a second meaning or the first one. We check for it and take the right element.

Later, we get the second soup, that is the second request we are making to get the examples. Get the table from the page

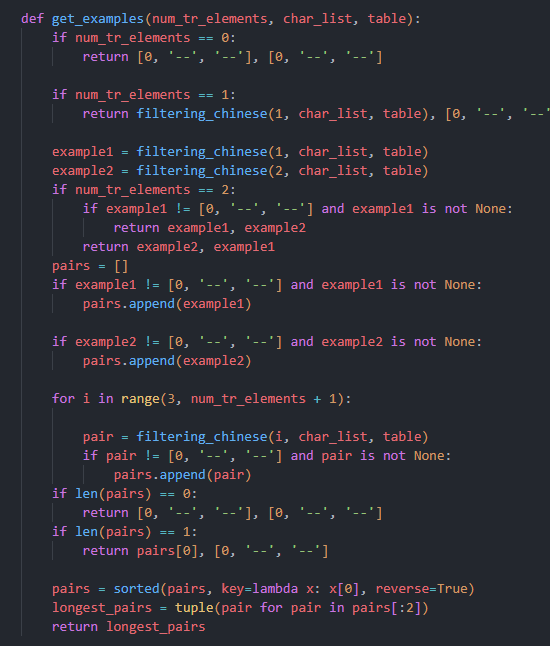
If there’s no table that means we have no examples, and we can return only the pinyin and translation. Otherwise, we find all the tr elements(examples) and proceed to filter them in get\_examples and save them.

**2.3.4 def get\_soup**



We get 3 values to work with: word, word\_id, and session. If we pass the word\_id, that means we need soup2, otherwise it’s our first request and we need soup1. Making the request through session, parse it with lxml and return it.

**2.3.5 def get\_examples**



Again, we need 2 examples, they must be phrases longer than 4 characters.

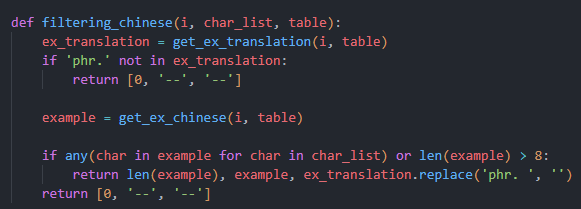
If we got into this function that means, there’s a table. Just in case I check if the number of examples is zero.

If we get only 1 example, I filter it (we need it to be a phrase of 4 char long) and return it, with the second example being empty.

If we get only two examples that’s simple to deal with, filter them, and if we get an empty example, that means we need to return the second one first and then the first one.

If we have more than 2 examples, create a list and append all the examples in it, filter them, and take the longest. Getting more than 2, doesn’t guarantee that we have the examples that we need. That means we run some checks before returning anything.

**2.3.6 def filtering\_chinese**

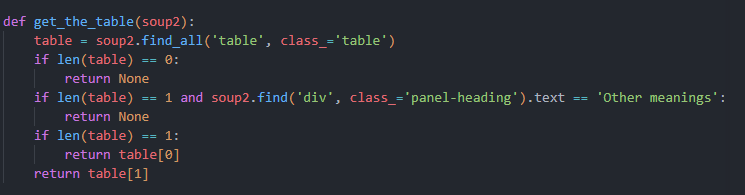


As an input we get, the number of an example (we can have 1 – 100 of them)

We get the translation of an example first, if the ‘phr.’ not in the translation we just return an empty example. Like here

Then we get the example in Chinese. Check to see if there’s a full stop, question mark or an exclamation point, and if it’s longer than 4 characters (Chinese character counts as two as it turns out).

**2.3.7 def get\_the\_table**



The main issue for me was figuring out what was wrong. The site says that the div element ‘examples’ has the class ‘table’, other divs have different classes. However, it turns out I was wrong, sometimes the word has a div element ‘Other meanings’. And ‘Other meanings' has the class ‘table’ as well.

After I figured this out the fix was simple.

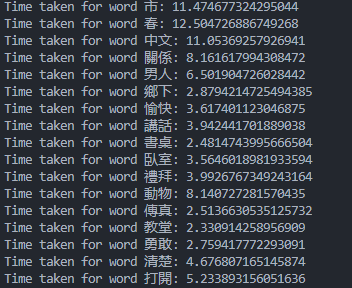
Basically, this function just checks whether we have the class ‘table’, if we don’t just return None. If we have just one table and it’s ‘Other meanings’ we return None, we’re not interested in that. And if we have two tables, we are guaranteed to have the ‘Examples’, which means we return the second one.

# Results

## Result 1

As for the results, I can’t say it’s the most efficient piece of software. We’re wasting quite some time when making requests. The performance really depends on the number of threads you’re running and obviously on your machine.

If the laptop is running of battery, I’m getting this



If it’s plugged in, I’m getting this



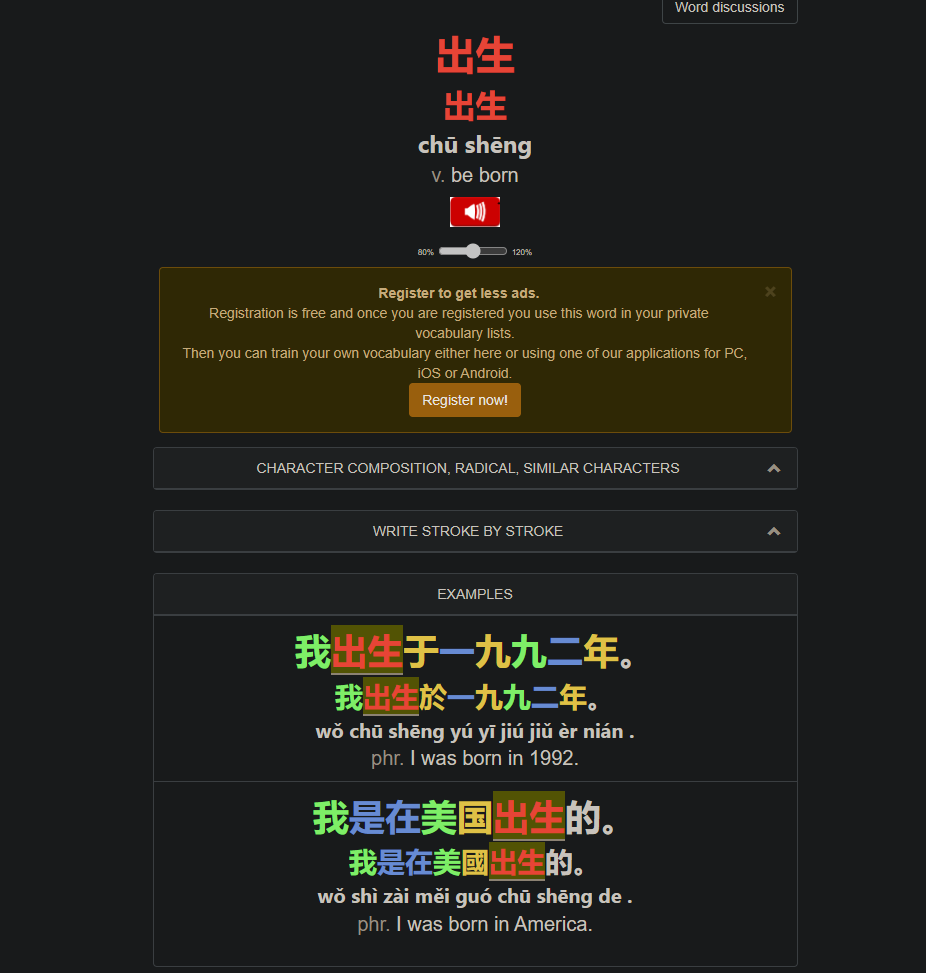
The difference is vast.

Results for (20 threads)

When it’s done running, we get a csv file

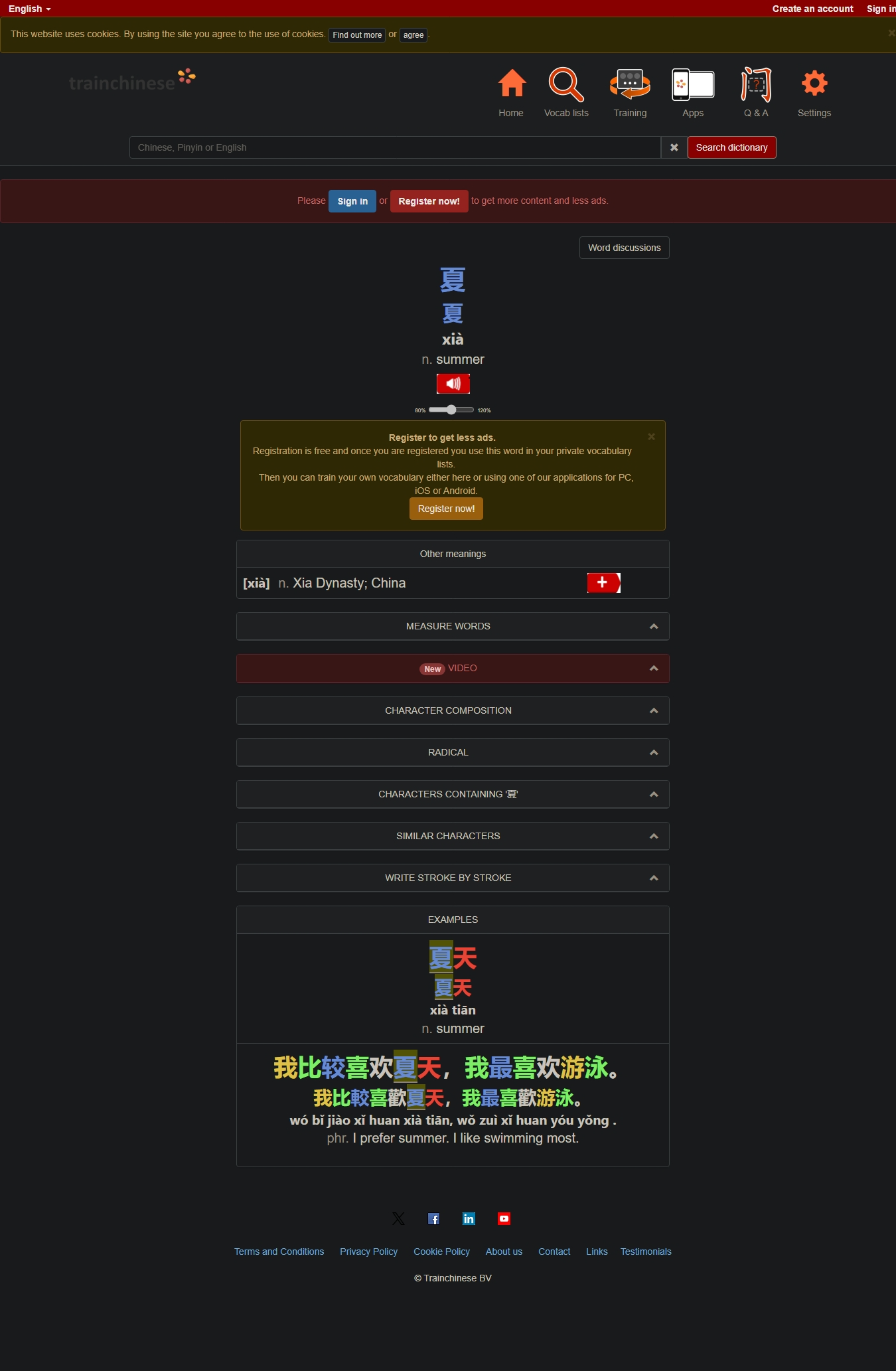
We can look closer and see: word with 2 examples;





Word with just 1 example;

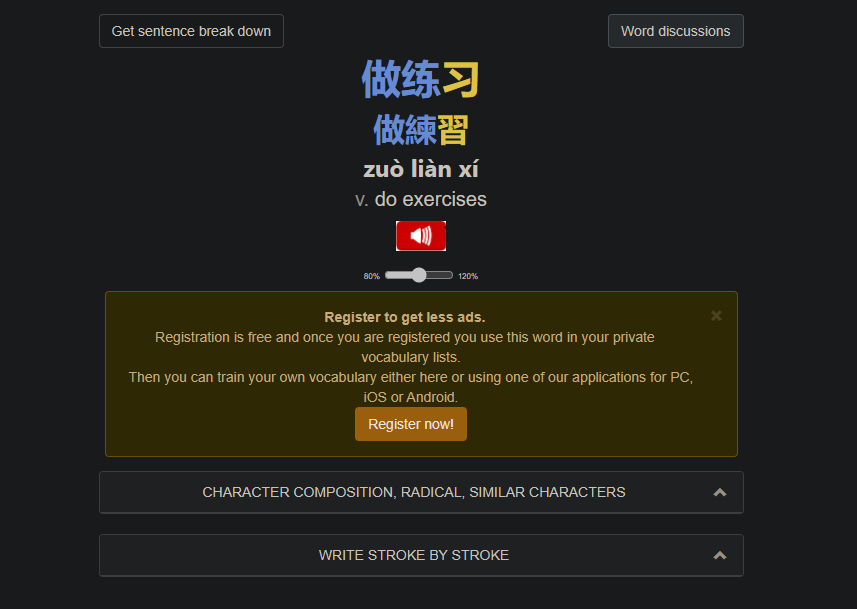




On the page we can also see an extra ‘Other meanings’ table, and only 2 examples, first one is a noun and the second is a phrase, which means our script works.

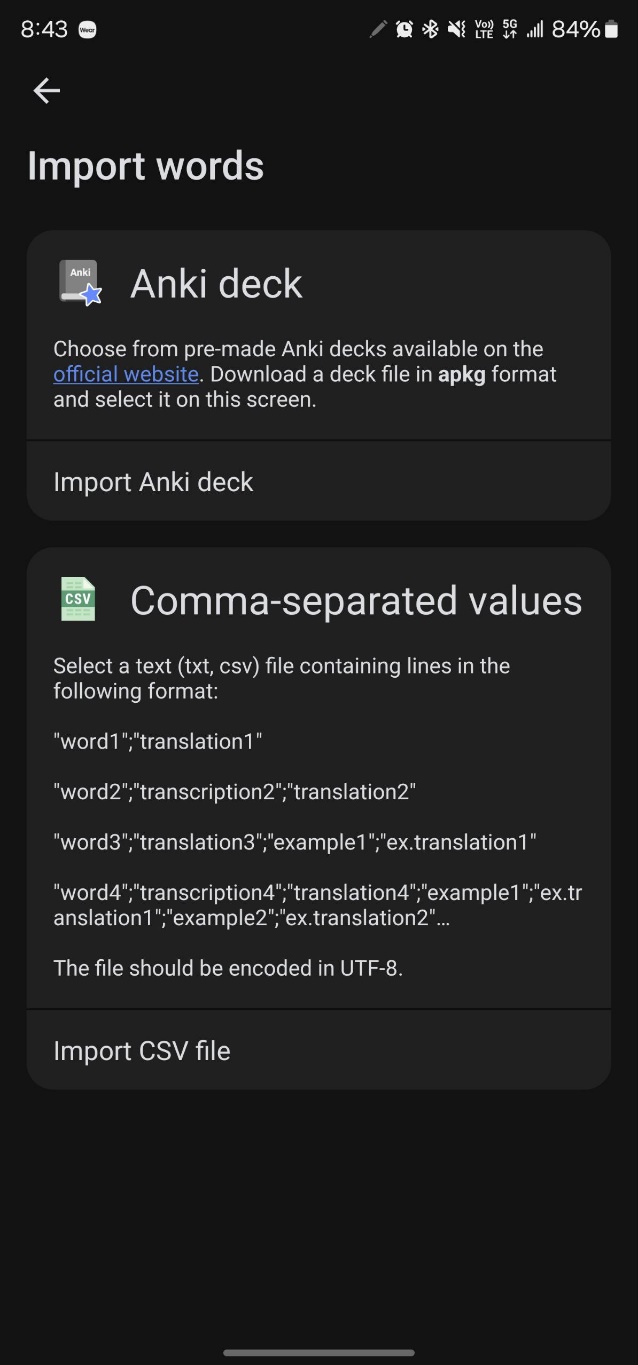
Word with no examples.

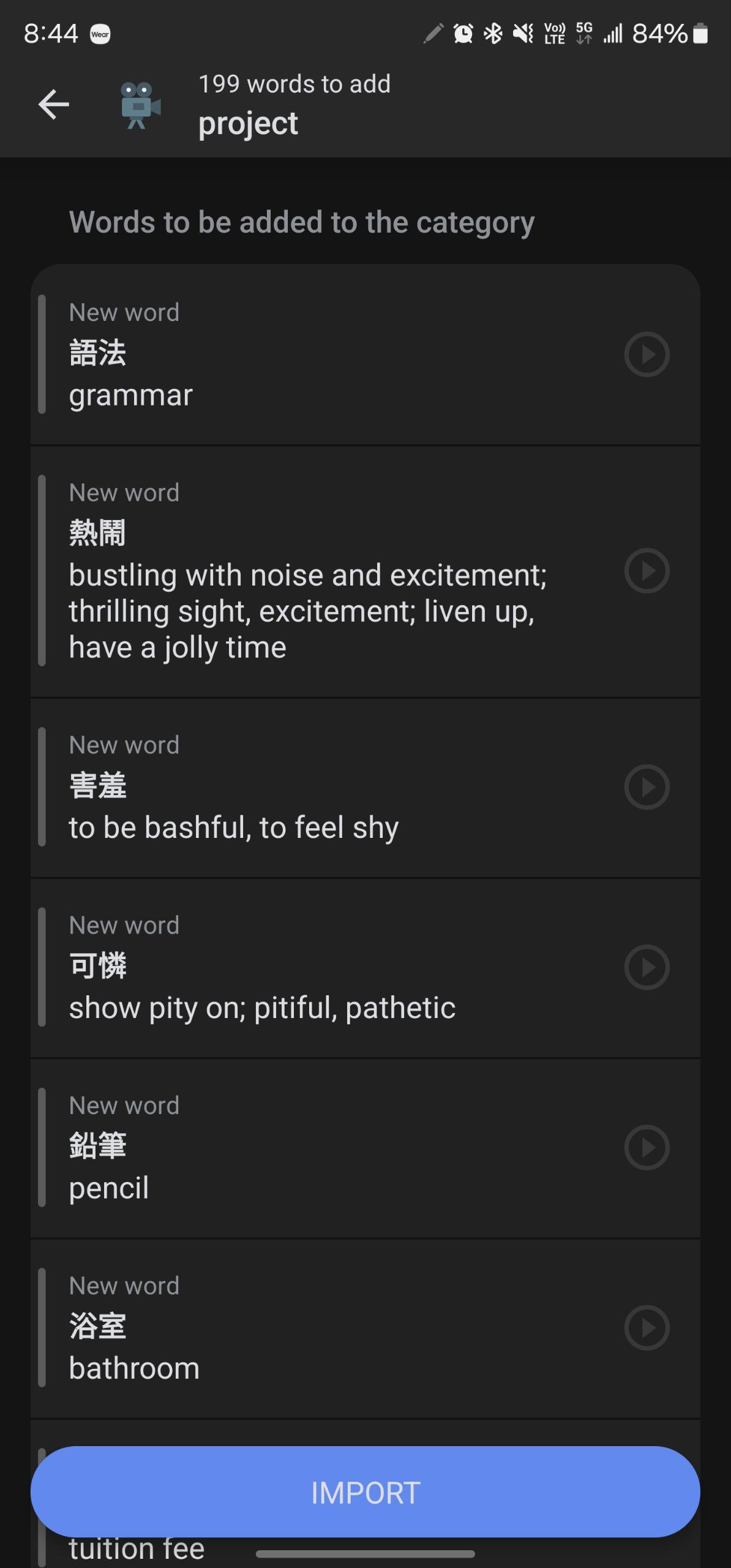


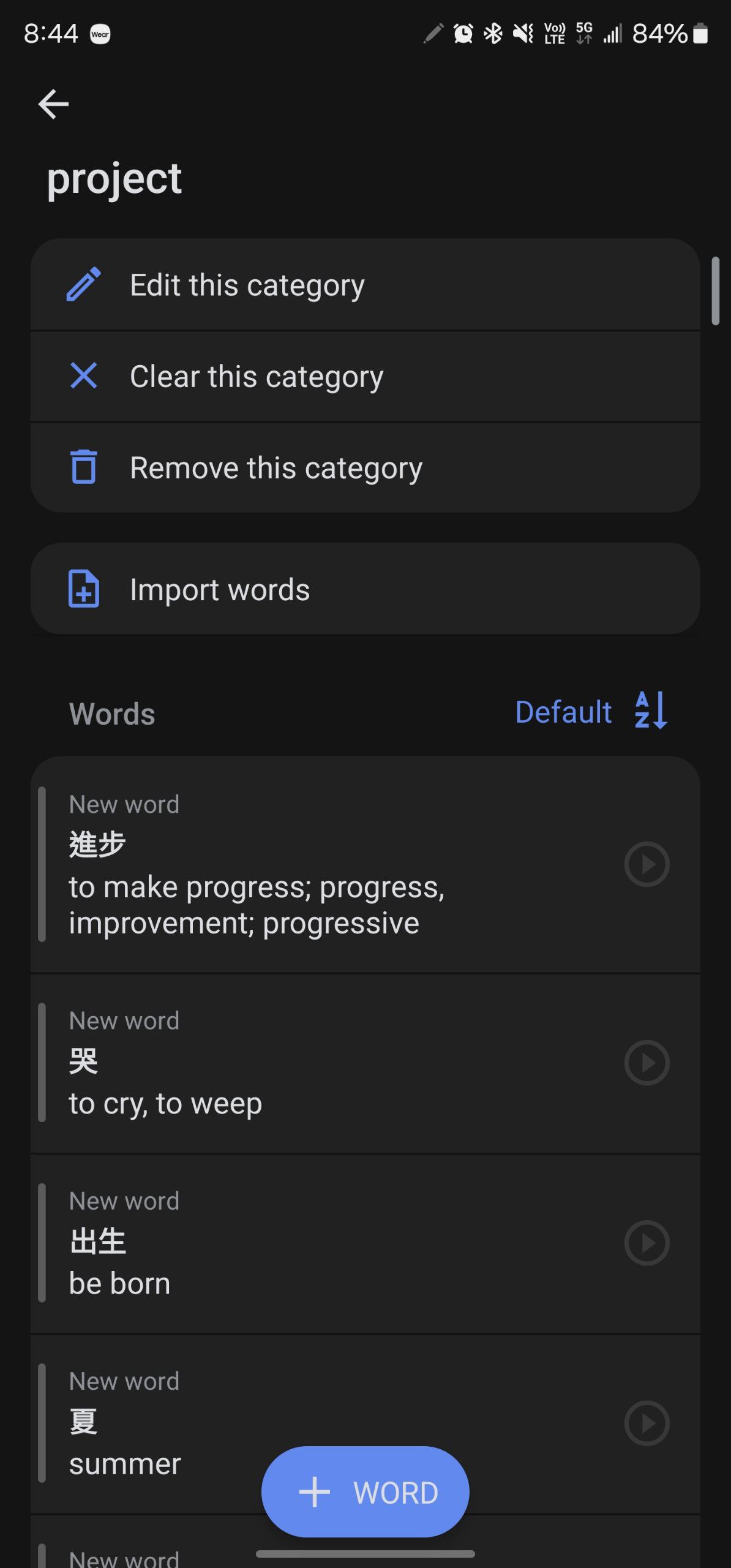


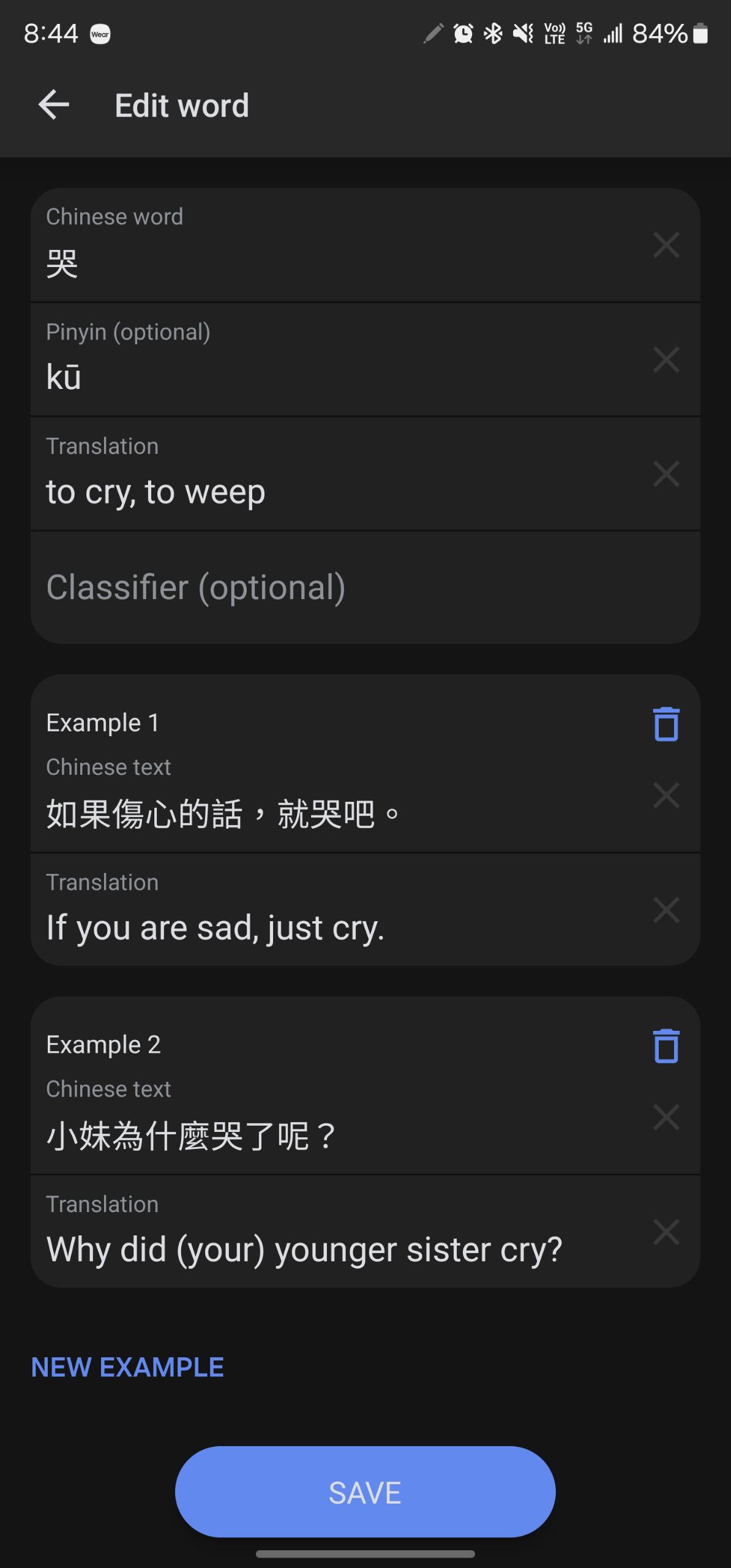
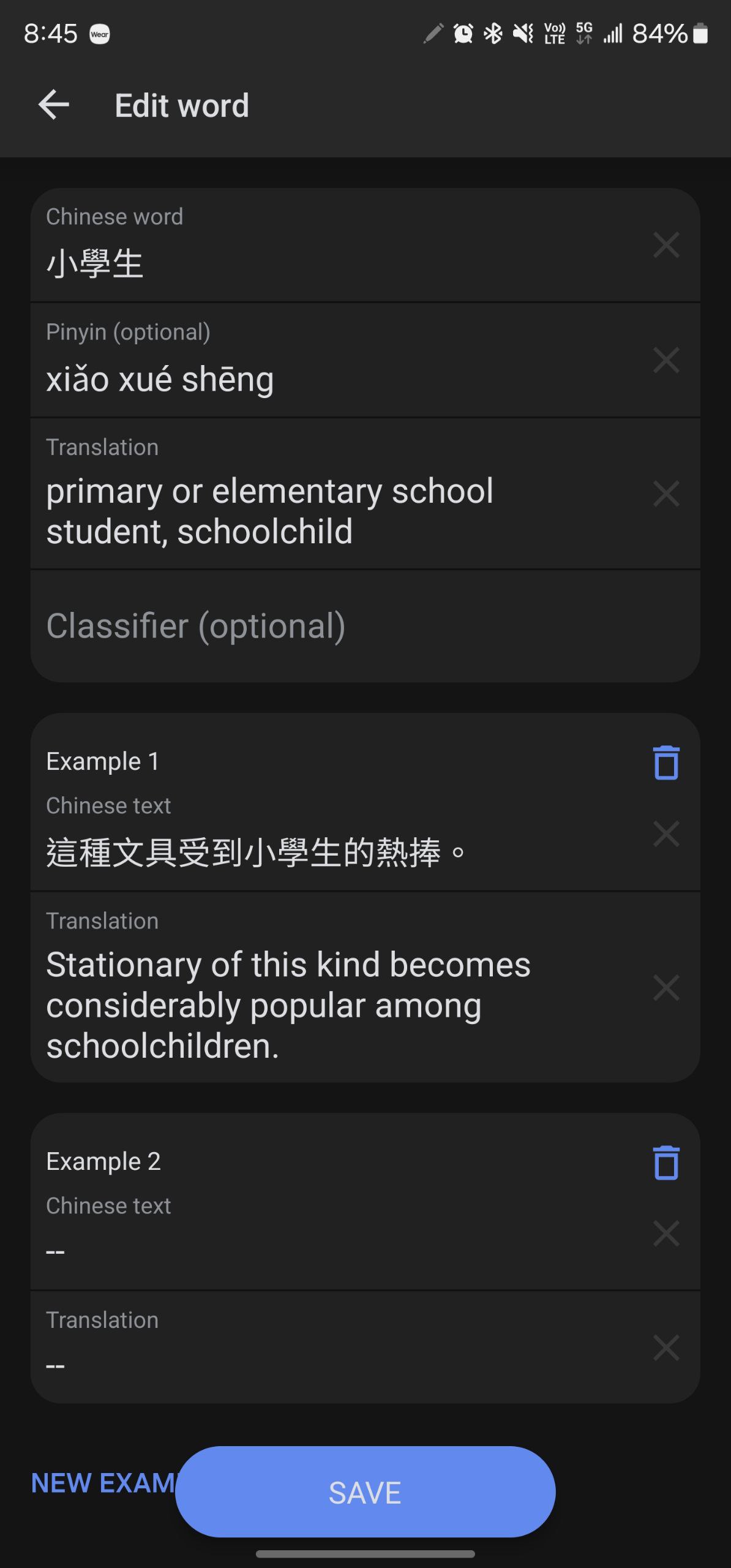
## Result 2

Does the word import in the app work? Let’s see.

 Go inside the app, choose our file and click import.

 So far so good, all the words are displayed properly. Let’s click ’import’

 Okay, it’s all in here, now let’s check the words for examples.

Perfect, it works, I think we can count this one as a success.

# Conclusions

Oof, this took more time than I expected. The hardest part was just figuring out how the website works and performs and fixing various bugs. This project taught me that if you do a webscraping project, first you must become familiar with the website. Second, understanding what you want to scrape. Third, how you’re going to organize the data. Finally, how you’re gonna implement everything. The last one is the hardest part. However, without understanding what you want and how you want it, you will spend a lot of time going back and forth.

Overall, I’m satisfied with the results. It works and this thing saves quite some time.