Japanese Hirigana/Katakana text converter. Converts between Roman characters and Japanese characters

My CA-1 is a Japanese character converter from the Japanese Characters to Romanji which is the conversion of Japanese characters to English characters. Japanese is made out of there main sets of characters. Hirgana, Katakana and Kanji. In this project I used all and created tools to study all of these.

In this project it is mainly based around 1. The translation class created. So you can convert any characters perfectly both back and forth from Romanji(Japanese in our English Alphabet) to Hirgana(Japanese in the Japanese native alphabet) and 2. A Study tool to help learn these Japanese characters.

At the moment I am currently learning Japanese and I have a few random apps to help me learn these characters. But I wanted to make a centred place to make learning these characters and easy and personalized for me. This is why I made a Japanese character converter.

**WARNING: Need to change file directories for some code to work.**

Complexities in writing this code. When writing this code I came across one main problem. The first and main problem I had throughout the project was pythons inability to directly read Japanese characters from text files. To avoid this I had to use the **.encode(“utf-8”) & .decode(“utf-8”).** After I fixed this problem I had to spend a lot of time formatting the characters and making sure my data was all clean and usable. I did this using just a python IDLE tab. All the main snippets of code I used is at the end of this file, but rather out of context I admit. Once I figured this out the rest was much more enjoyable.

**OOP Concepts:**

* Firstly, I used **Encapsulation** a few times. I used it in the study and translation class as completely private fields with the sample sentences used for translation examples. There were accessible using the “gethiri()” & “getkana()” methods and the stats in the stats class. All my character sets were semi-private, so available to the study class and all its children.
* Secondly, I used **Abstraction** throughout my project, I used it with the translation class. Using those methods as “black boxes” to magically translate, I used it the Daily class to magically keep track of the characters you have learned. All hidden within the class.
* As you can see **Inheritance** was possibly overused in this project. All my classes were children of the parent study class. This proved useful though as the base methods in the Study class were used/useful.
* There were **Polymorphisms** present between the study class and the stats class. The methods named hiragana and katakana were both present in both classes with different functionality.

**My Code:**

There are several Classes and I will break it down into each.

1. **Study Class**

Here I setup all my functions that are related to my personal study.

This is the parent class. Everything transcends this class. In this class I had the formatted Hiragana, Katakana and Kanji. Here there were many study methods available. The study of all the character sets, and also the sentence tests.

1. **Translation Class**

Here I have all the methods to translate any characters.

Here I implemented private fields. Only accessible to that class. Getters and Setters are also included.

This was proberbly the hardest part of the whole project. Here I had to translate back and forth between Romanji and Hirigana/Katakana. To do this I had to account for all the grammatical cases shown in the lists of unique if statement for both directions of translation. Admittedly, Romanji to Hirigana was a lot harder than Hirigana to Romanji.

1. **Daily Class**

Here I have my daily study method implemented within the methods of this class. Here you can study all the three-character sets. This was awkward to implement due to the inability of python to be able to read Japanese characters from a text file. Never the less I got it working, not without some tears though. ; )

1. **Stats Class**

In this class I include some extra fun methods and also the means of returning all the stats needed from your study.

This class uses super like other classes here and also used the \_\_str\_\_ method as it was only useful here, I feel.

1. Finally in the play function all the game play works. Here I used a set of if statements in a while loop to simulate a UI and way of navigating the project. This probably could have been implemented in a more effective way in retrospect.

**Quick Run Through of how to use:**

Text

Description automatically generated

1. Initial Run Screen. This is the home run screen. You input where what option you want to select.
2. The next step will just be one route to take to see how it works.

Text

Description automatically generated

1. You simply navigate through the methods and options by selecting the number. When done here type “leave” and you can return to the home screen.

**Code Used to filter out text data.**

* Filtering out the spaces at the end of the text
  + for t in range(len(k)):
  + while k[t][-1] == ' ':
  + k[t] = k[t][:-1]
* Encoding all the data for python.
  + for y in i:
  + q.append(y.encode("utf-8"))
* Comparing Characters to Meaning
  + for i in range(len(k)):
  + print(k[i].decode("utf-8"),m[i])