Table of Contents

- Aim
- Modules
- Implementation
- Video Production
- Tree Structure

Aim

In this project, I will propose a program that simulates a dictionary. Specifically, I will web scrap an online dictionary Merriam-Webster - and extracts data from it, printing out the definition(s) as the user inputs a word. Moreover, I also retrieve the pronunciation from the dictionary and play it if the user asks for.

Modules

Throughout the project, I will use six Python modules, two of which will require the user to download in order to run the program.

1. time - its sleep() function gives a short break (0.5 second) between each major part of the program.
2. urllib.request – defines functions and classes which help in opening URLs.
    - Its urlretrieve() function is used to retrieve the content of a URL directly into a local location on disk.
3. bs4 – its BeautifulSoup() function pulls data out of HTML files of the website.
4. pygame – its mixer() function plays the mp3 file (pronunciation file).
    - This is what I use to play the pronunciation file downloaded by using the urlretrieve() function.
5. requests - its get() function allows for the exchange of HTTP requests.
6. functions - this is a user-defined module that contains a set of three functions which I separate from the main program to improve code legibility and code reuse.

Implementation

To make the program more like a dictionary, before getting into any definition, I make a welcome statement and show the top lookup today. I get the URL of the website, then use the requests and bs4 module to extract the HTML data from the page. Then, I search through the HTML data to find the *a* tag which contains the information /word-of-the-day. Since the one I want to find is the fourth *a* tag containing the information /word-of-the-day, I use the *find_next()* function three times to find it. Lastly, I use the *get_text()* function to get the actual information without any HTML – which is the word of the day I am looking for.

I use a similar procedure to find the definition. Some slight differences are:

- Ask the user to enter a word, then utilise the order of the URL, which is https://www.merriam-webster.com/dictionary/{word}, I just need to add the inputted word into the last curly brackets. Then I can get the URL to the entry of that word.

- I rule out word that is not in the dictionary by checking whether the "false" message "isn't in the dictionary" is in the HTML data extracted from the URL. If yes, I use a while loop to keep inviting the user to re-enter a word until the "false" message is not in the extracted HTML data.
- After having a valid word, I will determine how many definitions the word has by counting *dtText* in the HTML data.
    - If there is only 1 definition, print it out in the main program.
    - Otherwise, use the defined function *find_all_definition()* to find every definitions.

Lastly, I use the define function called *mp3()* to return the URL of the mp3 of the word's pronunciation. I simply assign the argument text, which is the HTML data of the word in the dictionary, to the *mp3()* function and locate the text "contentURL". After collecting the line containing the URL of the mp3, which is in the type of a list, I convert it into a string using the *join()* method. After receiving the URL, I will ask the user whether they want to hear the pronunciation. If yes, I will use the *urlretrieve()* function from the urllib.request module to download the content, which is the recording, to the same folder as this program and name it word_pronounce.mp3. After that, I will use the *mixer.init()*, *mixer.music.load()*, *mixer.music.play()* function from the pygame library to play the *word_pronounce.mp3* in the local folder. I then use a while loop to keep asking whether the user wants to re-play the pronunciation. The loop stops if the user inputs 'n' or 'N'. The try block is used to test the mp3 URL, and the except block is used to handle the error where the recording is not available – which means there is not any pronunciation available for this word. This situation usually happens with phrases and abbreviations. In this case, I will just send a message saying sorry to the user.

The dictionary program culminates with a little thank you note.

Video Production

Execute the **main** module

Tree Structure

```
.
├── README.md
├── functions.py
├── main.py
├── requirements.txt
└── video_production.mp4

0 directories, 5 files
```