Table of Contents

General Idea ……………………………………………………………………….. 1

Web Scraping ……………………………………………………………………… 2

Natural Language Processing …………………………………………………… 5

Interfacing ………………………………………………………………………….. 8

Notes ……………………………………………………………………………….. 10

General Idea

The general idea of the project is to create a device or an app that can give users, like doctors, patients, researchers, etc., accurate and reliable information about a disease. However, oftentimes, it is not as simple as looking up a disease. For example, a patient sometimes just knows a couple of symptoms. From there, our device is supposed to give the user related words to the previous input, until it ends up with what the user ultimately wants. The end result is a page article from medical websites; in this project, I used WebMD.

To achieve this, I used Python packages such as Beautiful Soup for web scraping, Spacy for Natural Language Processing, and Tkinter for interfacing. I have summarized in this document the work I have done with each package, the problems I ran into, and the improvements that need to be made in order for this project to be fully functional.

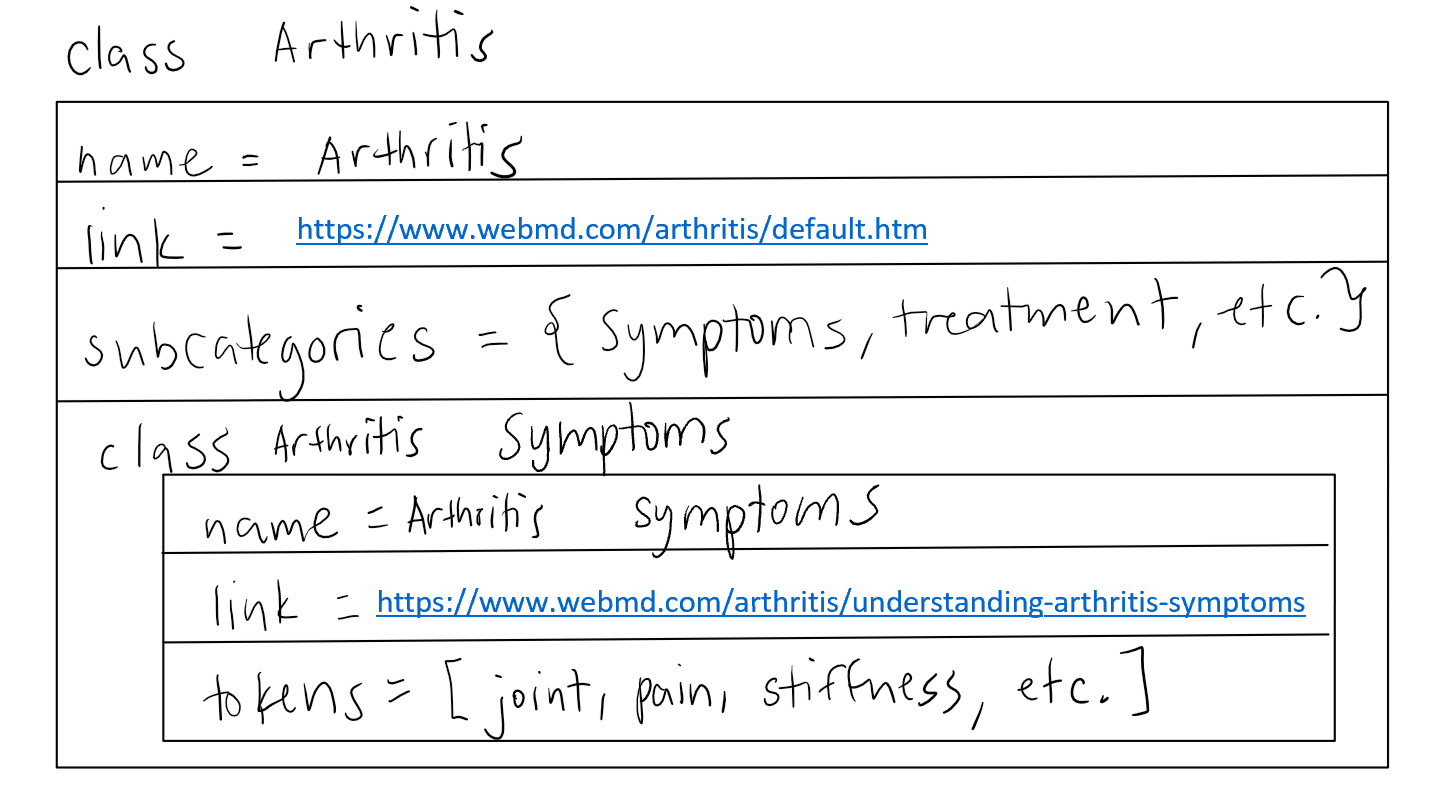
At the beginning of running the code, users will be asked whether they are a patient, doctor, or researcher/student. This would, ideally, lead to different end results for each one, as doctors would be more concerned with medical journals and other technical papers, while patients can be just content with news articles or even less technical pages like WebMD. This redirection would be hard to implement in code, as this would require more web scraping, and websites have different styles that often get modified by their respective web developers. A more dynamic code would be a great solution to this problem. Accessing online academic databases would be different and quite challenging too.

I used Visual Code Studio (<https://code.visualstudio.com/>) as my IDE and Anaconda for packet management (<https://www.anaconda.com/products/individual>).

Web Scraping

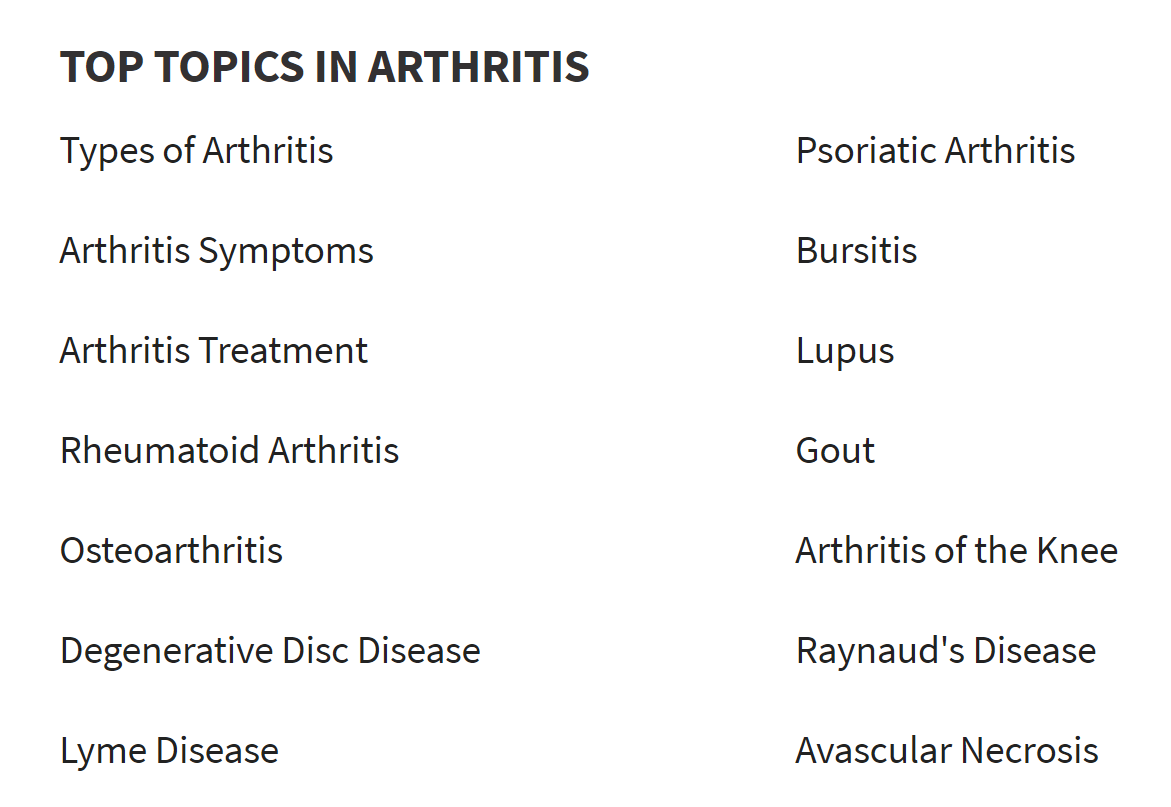
For web scraping WebMD, I used the Beautiful Soup package in Python: <https://pypi.org/project/beautifulsoup4/>. Beautiful Soup is a library that makes it easy to scrape information from web pages. It sits atop an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree.

Since web scraping happens in real time, I used a python class to store disease profiles. So in paper, here is what one disease profile looks like:



*Picture 1: Disease Profile Example*

As shown, each disease profile contains the above information. As a result, the class Arthritis would have the following nested classes:



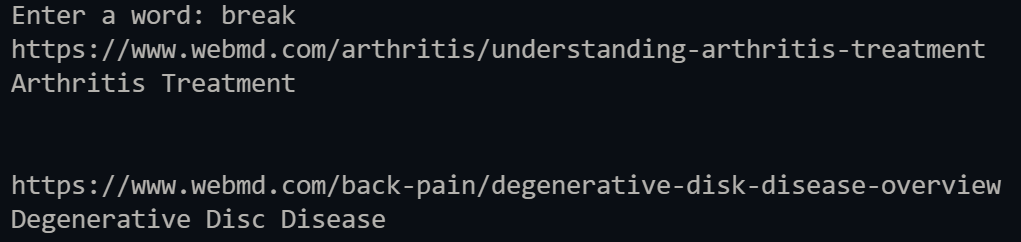
*Picture 2: Subcategories*

In code, this is how I implemented it:



*Picture 3: Code Implementation of a Disease Profile*

For example, in the python file “class\_test”, I have successfully proven the global accessibility of disease classes. In this example, you can see what links contain the word “break” in the context of arthritis (from WebMD):



*Picture 4: Word Search in Subcategories*

In Picture 4, it can be seen that the word “break” is found in Arthritis Treatment and in Degenerative Disc Disease (through the shown links).

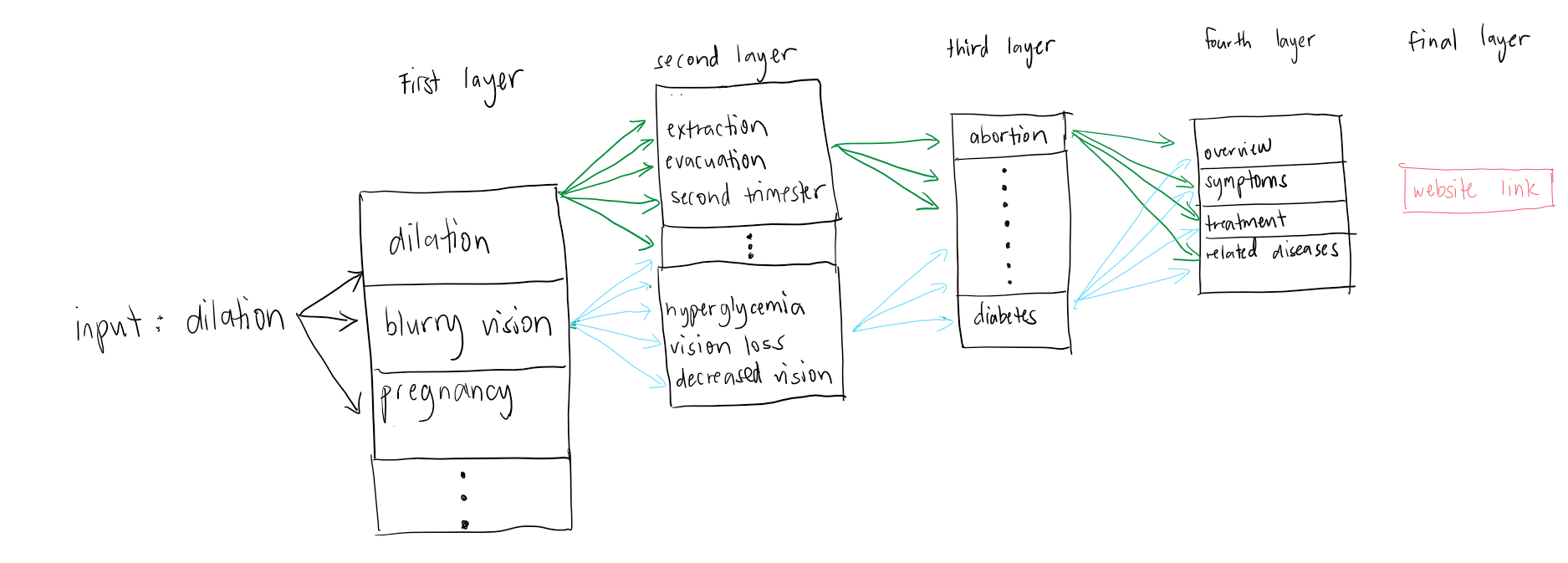
Doing web scraping through this method can have a really bad runtime. It often leads to your code crashing especially if you’re storing almost 250+ disease profiles, with 50-100 tokens of words each. So another solution would be storing said disease profiles locally after running the code for the first time. As long as the structure of the disease profile is maintained, such that tokens of words in a disease page can be associated with a link, then I believe this would yield faster runtime.

As mentioned in the General Idea section, a different code would be required for web scraping academic online databases. However, I have provided the basic template for web scraping common medical websites like WebMD. Although I already mentioned this in the code, web developers often change certain class handles like “article body” and “active page”. So, keep an eye on those occasional changes and modify the code accordingly.

Natural Language Processing (NLP)

For analyzing and grouping words, I used the Python package Spacy: <https://spacy.io/usage/models>. Spacy has a lot of language models that have built-in trained models for language processing. If you apply a pre-trained English model (en\_core\_web\_md) to a given text file, it already knows what words are related to what.

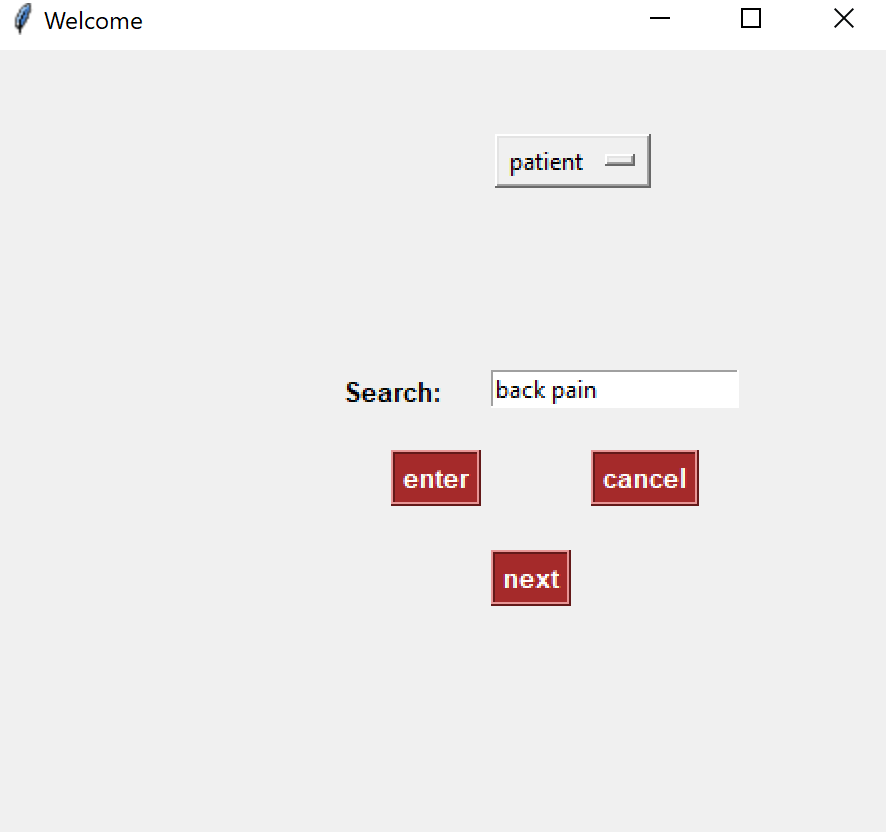
On a macro level, my interpretation of the implementation is shown:



*Picture 5: NLP Concept Drawing*

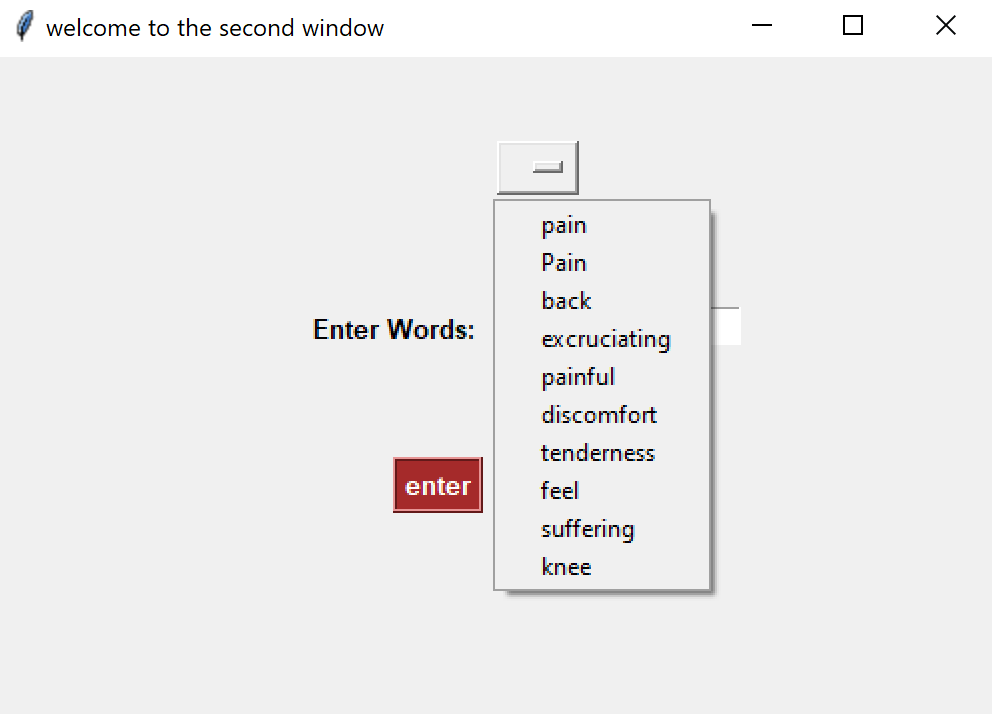
Now, the runtime can be ridiculous since we are traversing through a total of almost 500000 words in the first layer alone, considering only one word was inputted. Although Spacy tells you the words related to a certain word within a text file, it does not tell you where they are located. That’s one of the next challenges in this project. Ideally, as displayed in the image above, if the user inputs “dilation”, right off the bat, through the Spacy English model, it would tell you that “dilation”, “blurry vision”, and “pregnancy” are the related words in a given text file.

I also implemented a code that shows you closest words to two words you type. For example, I wanted to know what back pain can be related to:



*Picture 6: NLP Demonstration of Two Words*

Pressing the next button will take you to a new window where you can choose which the related word that makes sense the most from the dropdown, otherwise you can enter another set of words (not implemented yet):



*Picture 7: Results of NLP Demonstration*

Unfortunately, I have only achieved the first layer of this NLP process. Using the interface, I was only able to give the user the first layer of related words directly to the input. The next step would be to run the same model to each of the input’s related words, which in my example would lead to the second layer. For example, “dilation” is related to “blurry vision” which is related to “hyperglycemia”, “vision loss”, and “decreased vision”. Then you have to do the same process to these 3 words/strings until you end up with the final layer, which is the link of the specific page you want the contents of.

To understand the concept of Natural Language Processing, I read and implemented the following tutorials:

1. <https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469>

* This one was run on Jupyter Notebook; you can access Jupyter Notebook on Anaconda: <https://jupyter.org/install>

1. <https://rare-technologies.com/word2vec-tutorial/>

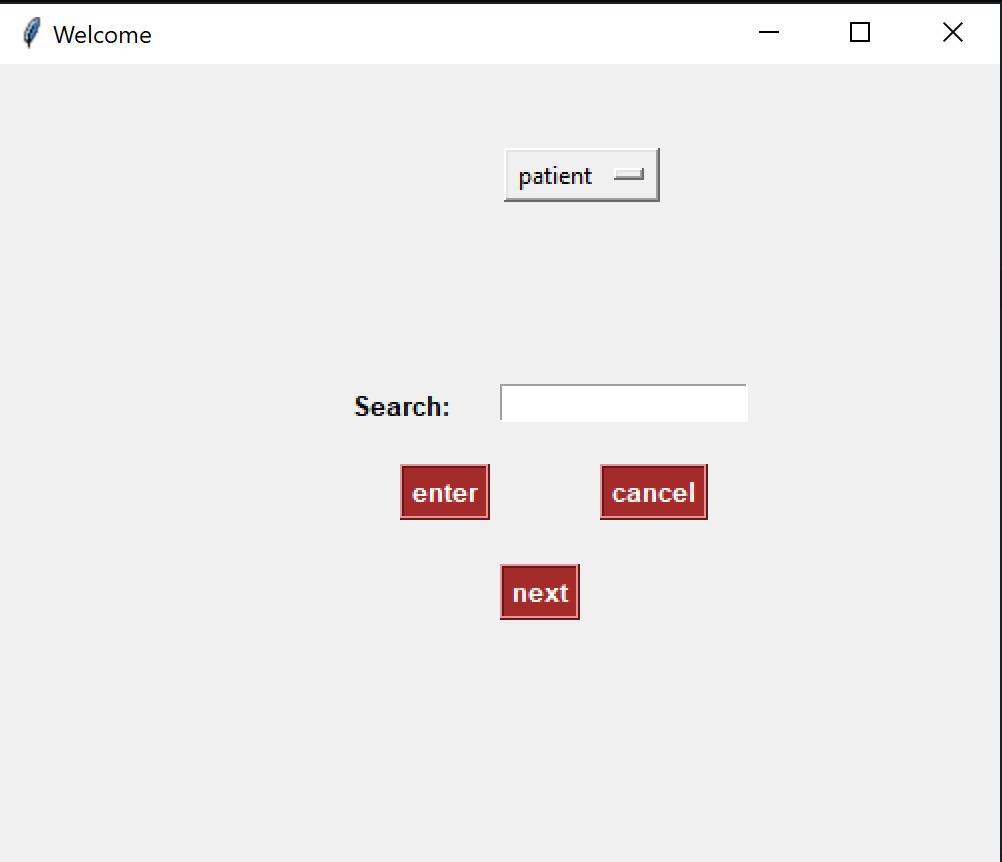
* This one used Word2Vec, not Spacy. In case, you want to try a different package.

1. <https://realpython.com/natural-language-processing-spacy-python/>

* This one is a very good tutorial for Spacy

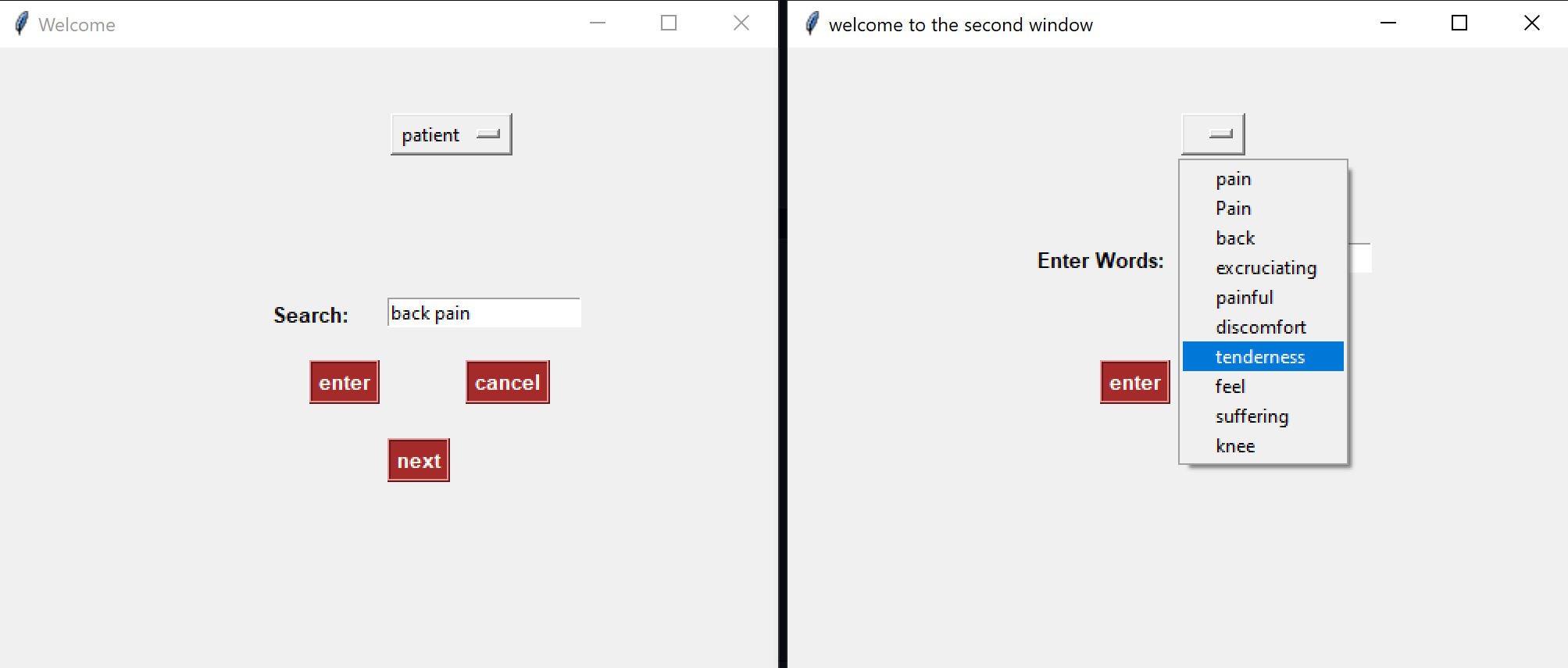
Interfacing

For interfacing, I used Tkinter: <https://docs.python.org/3/library/tkinter.html> and <https://docs.python.org/3/library/tk.html>. This is good for basic, simple interfacing. Here’s an example from my code:



*Picture 8: Tkinter Basic Interface Example*

A challenge that I have encountered with this is reusing only one window, instead of creating a new window. So far, I have only implemented opening a second window.



*Picture 9: Second Window Demonstration*

Another point of improvement is saving the history of choices and providing a “previous” button that would have access to the previous choices, in case the user was not happy about the current state of the search.

In learning Tkinter, I used the following tutorials and resources:

1. <https://realpython.com/python-gui-tkinter/>
2. <https://www.tutorialspoint.com/python/python_gui_programming.htm>
3. <https://www.youtube.com/watch?v=3E_fK5hCUnI> (dropdown menu)
4. <https://www.geeksforgeeks.org/python-tkinter-entry-widget/> (entry widget)
5. <https://www.python-course.eu/python_tkinter.php>

Notes

1. Make sure to install all packages and pipelines before running any of the Python files.
2. When installing packages, make sure they go in the same folder as your Python files.
3. If you are creating a virtual environment, make sure that the packages you are going to install are supported by the Python version in your virtual environment. For example, for this project, I initially was using a virtual environment, which runs on Python 3.5 because it would not update to the latest 3.9 version. Then, Spacy, all of a sudden, just stopped working. So I had to switch to the local Python environment, which runs on the latest Python 3.9 version, and it worked.