**PHILOSOPHISCHE FAKULTÄT**
**Seminar für Sprachwissenschaft**

**PS ‘Programming and Data Analysis’**
**Todd Snider/Michael Franke, WiSe 2023**
**Tutors: Boxin Li, Greta Hristozova, Weiting Wang, Yue Yu**

# Assignment 05: Processing Frequency Lists

December 05, 16:00 – December 12, 14:00

In this assignment, you will practice file input and output with a token frequency list as an example. A very useful collection of frequency lists was extracted by Hermit Dave from the OpenSubtitles corpus (`http://opus.nlpl.eu/OpenSubtitles.php`), a corpus of parallel subtitles for thousands of movies. Due to the high amount of dialogue in movies, this corpus comes much closer to actual every-day language usage than e.g. frequency lists extracted from the Wikipedia or newspaper texts. The frequency lists for dozens of languages are available in this public repository, which is also where the example frequency list for this exercise is from:

`https://github.com/hermitdave/FrequencyWords/`

## 1 Reading the Frequency List from a File [4 points]

The tokens in the frequency files are ranked by their absolute frequency in the corpus. Each token and its frequency (an integer) are written on a separate line, delimited by a single space. Your first task is to write a method `read_frequency_file(freq_file_name)` which reads in a frequency file with name `frequency_file_name` and stores the frequency information in a simple data structure. Every token is to be stored in a tuple (*token*, *count*), and the result of `read_frequency_file(freq_file_name)` should be a list of such tuples in the order in which they appear in the file. After a successful implementation, you should be able to extract the three most common words of English (and their counts) like this:

```
>>> frequencies = read_frequency_file("freq_en_50k.txt")
>>> frequencies[0:3]
[('you', 22484400), ('i', 19975318), ('the', 17594291)]
```

## 2 Determining the Decile Thresholds [4 points]

Next, we want a function `determine_decile_thresholds(frequencies)` that takes as input a data structure like the kind output by `read_frequency_file`, and then splits the language's tokens into ten parts of equal frequency. The **deciles**, or 10%-level thresholds, are the numbers of word forms that are needed to cover 10% of a text, 20%, 30%, and so on to 90%. (Obviously in order to understand 100% of a text, you'd need to understand 100% of the word forms within it!)
The correct result on the English data (at least as represented in our `freq_en_50k.txt` file) should imply that 62 word forms are enough for 50% of all tokens in an English text, whereas to reach 90% of all tokens (a threshold frequently quoted as the threshold to pleasant reading in the literature), passive knowledge of 2,617 English word forms is necessary:

```
>>> thresholds = determine_decile_thresholds(frequencies)
>>> thresholds[4:9]
[62, 115, 252, 662, 2617]
```

## 3 Computing Prefix Frequencies [4 points]

Another interesting thing one can do with a token frequency list is to compute the frequency of each prefixed substring. A prefixed substring of a string is any substring that starts at its first character, including the entire string itself. For example, the prefixes of `"house"` are `"h"`, `"ho"`, `"hou"`, `"hous"`, and `"house"`. (That these aren't all meaningful morphemes in English is not important to us at the moment.) Write a function `get_prefix_frequencies(frequencies)` which takes a list of (*token*, *count*) pairs as produced by Task 1, and creates a similar list containing all the prefix counts. If you did this correctly, you should be able to do this on your interactive console:

```
>>> prefix_frequencies = get_prefix_frequencies(frequencies)
>>> prefix_frequencies.sort()
>>> prefix_frequencies[500:504]
[('acc', 306281), ('acce', 104542), ('accel', 4894), ('accele', 4894
```

## 4 Save Prefix Frequencies in File [4 points]

Your final task is to write a function `store_frequencies_alphabetically(frequencies, freq_file_name)` which writes the data from any list of string-frequency pairs (`frequencies`) into a new alphabetically sorted file with the name `freq_file_name`, in the same format as the file you read the frequencies from. Only the prefixes which occurred 1000 times or more in the input data set should be written to the file. Use the function to save the result of your prefix frequency computations into a new file with the exact name `freq_en_prefixes.txt`. If you did everything correctly, the first few lines of your result file should look like this:

```
a 45546869
aa 55975
aaa 7478
aaaa 2175
aaaah 1092
aaah 3973
aah 35842
aar 9254
aaro 7657
aaron 7657
```

## 5 Tests [ungraded]

That's it! Before submitting, don't forget to test your methods one last time using `test_ex_05.py` and check whether the first lines of the output are identical to the desired result! You must submit TWO files: your version of `ex_05.py`, and the output file `freq_en_prefixes.txt` (with those precise names).