COSC-243: *Natural Language Processing* — Spring 2025
Instructor: Professor Shira Wein — Amherst College

Homework #2: Regular Expressions, Tokenization

Due: Friday, February 14 at 11:59 AM

———————————————————————————————————————

Download Instructions: Download the starter code (golf.py, tokenizer.py) and all of the .txt files from Moodle.

Submission Instructions: Submit three files to Moodle: your code from Part 1 the golf.py file, your code from Part 2 in the tokenizer.py file, and your written solutions to Part 2 in a file called "written_answers" in either PDF or .txt format.

Collaboration Policy: As with all the homework, you must work alone on these homework problems. You may not discuss solutions to these problems with other students or search for specific solutions online. You are encouraged to discuss general concepts with other students as well as to search online for reference material. You are also encouraged to discuss the homework with the TAs and the instructor. All course materials including the textbook and lecture notes may be referenced.

Credits: This work is adapted from assignments created by Carolyn Anderson.

———————————————————————————————————————

Part 1: RegEx Practice

Practice regular expressions by playing RegEx Golf: https://alf.nu/RegexGolf
You must complete the following tasks on the RegEx Golf site:

- Warmup
- Anchors
- It never ends
- Ranges
- Backrefs
- Abba
- A man, a plan

1.1 Your Solutions

In the file `golf.py`, fill in the empty strings with your best (that is, shortest) regular expression for each of the puzzles that you solved.

You'll get full points for each solution as long as your regular expression didn't avoid the challenge of writing rules that generalize properties of the words (e.g., `r"^(word1|word2|word3...)$"` would be ridiculously long and a little silly).

That said, if your length seems multiple times longer than what's on the leaderboards, challenge yourself to get one that's shorter!

1.2 Integrity Note
Since these puzzles are online, there are (undoubtedly) solutions posted somewhere. I trust you not to look for those solutions, but to submit the best solutions you can come up with on your own. If you need help, come to office hours.

Part 2: Tokenization and Word Frequencies
Put all of your code for this part in `tokenizer.py`.
Along with some other functions, the starter code has an empty main function that you should fill in with code to demonstrate how your functions work. You should call your main function using the following pattern (at the bottom of your file):

```
if __name__ == "__main__":
      main()
```

This pattern will be useful for you as you develop more complex python programs. It allows you to write functions that will can be imported into other programs while still having your `tokenizer.py` be runnable as a stand-alone program.

2.1 Project Gutenberg
We will be exploring files from Project Gutenberg. Start by reading in Lewis Carroll's Alice's Adventures in Wonderland, which is stored in the file `carroll-alice.txt`.

2.1.1 get_words function
Fill in the `get_words` function so that it takes in the full text (argument `s`) and returns a list of all words in the text, and—if `do_lower` is True—all words are lowercased.

2.1.2 words_by_frequency function
Use the existing `words_by_frequency` function (which also calls the existing `count_words` function) to explore the text. You should find that the five most frequent words in the text are:

```
the      1603
and       766
to        706
a         614
she       518
```

For this entire assignment, we will lowercase when getting a list of words and will use `words_by_frequency` to get counts.

The word 'alice' actually appears 398 times in the text, though this is not the answer you got for the previous question. Why? Examine the data to see if you can figure it out before continuing.

2.2 Punctuation
There is a deficiency in how we implemented the `get_words` function! When we calculate word frequencies, we probably don't care whether the word was adjacent to a punctuation mark. For example, the word 'hatter' appears in the text 57 times, but our count_words Counter says it appears only 24 times. Because it sometimes appears next to a punctuation mark, those instances are being counted separately.

Our `get_words` function would be better if it separated punctuation from words. We can accomplish this by using the `re.split` function. Be sure to import re at the top of your file to make `re.split()` work. Do not change the `get_words` function; we will be wiring a new function for this. Below is a small example that demonstrates how `str.split` works on a small text and compares it to using `re.split`:

```
>>> text = "‘Oh no, no,’ said the little Fly, ‘to ask me is in vain.’"
>>> text.split()
['"Oh', 'no,', 'no,"', 'said', 'the', 'little', 'Fly,', '"to', 'ask',
'me', 'is',
 'in', 'vain."']
>>> re.split(r'(\W)', text)
['', '"', 'Oh', ' ', 'no', ',', '', ' ', 'no', ',', '', '"', '', ' ',
'said', ' ',
 ' ', 'little', ' ', 'Fly', ',', '', ' ', '', '"', 'to', ' ', 'ask', '
', 'me', ' '
 ' ', 'in', ' ', 'vain', '.', '', '"', '']
```

Note that this is not exactly what we want, but it is a lot closer. Using the above example as a guide, write and test a function called `tokenize` that takes a string as an input and returns a list of words and punctuation, but not extraneous spaces and empty strings.

Like `get_words`, `tokenize` should take an optional argument `do_lower`, in this case which defaults to False, to determine whether the string should be case normalized before separating the words. You don't need to modify the `re.split()` line: just remove the empty strings and spaces from the list to be returned.

Checking In
Use your tokenize function in conjunction with your count_words function to list the top 5 most frequent words in `carroll-alice.txt`. You should get this:

```
’         2871   <-- single quote
,        2418   <-- comma
the      1642
.         988   <-- period
and       872
```

2.3 Getting words

Write a function called `filter_nonwords` that takes a list of strings as input and returns a new list of strings that excludes anything that isn't entirely alphabetic. Use the `str.isalpha()` method to determine if a string is comprised of only alphabetic characters. Here is an example of how it should behave:

```
>>> text = "‘Oh no, no,’ said the little Fly, ‘to ask me is in vain.’"
>>> tokens = tokenize(text, do_lower=True)
>>> filter_nonwords(tokens)
['oh', 'no', 'no', 'said', 'the', 'little', 'fly', 'to', 'ask', 'me',
'is', 'in', 'vain']
```

Checking In
Use this function to list the top 5 most frequent words in `carroll-alice.txt`. Confirm that you get the following before moving on:

```
the    1642
and    872
to     729
a     632
```

```
it      595
```

2.4 Most frequent words

Create a directory (folder) called “gutenberg_data” and put all of the .txt files in it. Iterate through all of the files in the gutenberg data directory and print out the top 5 words for each. To get a list of all the files in a directory, use the `os.listdir` function:

```
import os
directory = "gutenberg_data"
files = os.listdir(directory)
infile = open(os.path.join(directory, files[0]), ’r’,
encoding=’latin1’)
```

This example also uses the function `os.path.join` that you might want to read about. Here the “infile” variable is an IOTextWrapper reading in the contents of the first file in the directory. You’ll need to alter that line so that you can loop through all of the files.

Note: This open function uses the optional encoding argument to tell Python that the source file is encoded as `latin1`. Be sure to use this encoding flag to read the files in the Gutenberg corpus!

2.5 Written Answers

Answer the following questions in your assignment writeup, which should be submitted as a PDF or .txt file along with your Python files.

1) **Most Frequent Word**: Loop through all the files the gutenberg data directory that end in .txt. Is ‘the’ always the most common word? If not, what are some other words that show up as the most frequent word?
2) **Impact of Lowercasing**: If you don’t lowercase all the words before you count them, how does this result change, if at all?
3) **Frequency Graphs**: Generate frequency graphs of all of the text files using the `plot_frequency` function with an argument `n` of 100 (setting it to display only the 100 most frequent words). What trends do you see?
4) **Impact of Removing Punctuation**: Generate frequency graphs of all of the files with and without remove punctuation. How does removing punctuation affect the trends we discussed? For this you may want to investigate a specific file to be able to see the change in graph up close.