Northeastern University, Khoury College of Computer Science

CS 6120 — Assignment 0

Due: January 16, 2025 (100 points)

YOUR NAME + LDAP

In this assignment, we will develop a *word completion* autocomplete system that assists creative writing by suggesting words, phrases, or even entire sentences based on the style, genre, or theme of Shakespeare. To do so, we'll need the complete works of Shakespeare from Project Gutenberg, which is available at our website with slight modifications. When you've finished with this assignment, turn in your code and its artifacts on Gradescope. Your checklist:

- Python file named `autocomplete.py`, with the following functions
  - ☐ Your solution in Q1: `read_vocabulary`
  - ☐ Your final solution in Q2: `autocomplete`

Question 1: Setup and Preprocessing

In this question, we will do all the text processing that will enable us to work on the common problem of auto-completing words.

Download the data. If you're working in Google Colab, you can do so using the `!` magick - `!<command>`, which allows you to use Linux commands in the cell. In this case, the linux command is `wget --no-check-certificate`. In an execution cell, the following command will download the data to a local folder in the workspace:

```
!wget --no-check-certificate https://course.ccs.neu.edu/cs6120s25/data/shakespeare/shakespeare-edit.txt
```

Look for the download in your workspace from Project Gutenberg that is called `shakespeare-edit.txt`, the complete works, including sonnets, of William Shakespeare.

Write a function that:

- reads in the data through UTF-8
- removes all punctuation

- lower-cases every letter
- splits by space to create all words
- removes numerical digits
- returns a list of unique words ordered by their frequency (most frequent to least)

The function signature should look like this:

```
def read_vocabulary(filename):
  """
  Reads in a given file specified by "filename" and processes it
  by removing punctuation, forcing lowercase, splits into
  individual words, and removes the numbers that might appear in
  the text.

  Args:
    filename: the name of the file to be processed

  Returns:
    A list of words in the order in which they appeared in the
    text.
  """
  % <YOUR-CODE_HERE>
  return []
```

The result of the above should produce a list with words in the order by which they appeared in the text.

```
['the',
 'and',
 'i',
 '...']
```

Question 2: Autocomplete

In this question, we will write the function `autocomplete`, which takes a string prefix and the `word_list` that you have created in the last function. Because this course is about its applications in practice, can you make your solution efficient? The function signature is as follows:

```
def autocomplete(prefix, word_list):
  """
  Returns a list of words starting with the given prefix.

  Args:
    prefix: The prefix to search for.
    word_list: A list of words sorted by their frequency of occurrence.

  Returns:
```

```
    A list of ten most-common words starting with the prefix.
  """

  return []
```

Testing Out Your Code

Test your functions out in real-time. You can do that in a number of ways. For example, if you are in Google Colab, the following code will create a text box that updates everytime you enter a character:

```
import ipywidgets as widgets
from IPython.display import display

def on_value_change(change):
  prefix = change['new']
  suggestions = autocomplete(prefix, word_list)
  if suggestions:
    with output:
      output.clear_output()
      print("Suggestions:")
      for word in suggestions:
        print(word)
  else:
    with output:
      output.clear_output()
      print("No suggestions found.")

text = widgets.Text()
output = widgets.Output()

display(text, output)
text.observe(on_value_change, names='value')
```

Or, if you would like to get a head start on making an app, the following code will create a real-time streamlit app:

```
import streamlit as st
from st_keyup import st_keyup

def main():
    # Notice value updates after every key press
    st.title("Autocomplete App")
    query = st_keyup("Enter a value", debounce=500, key="0")
    suggestions = autocomplete_help(query)
    st.write(suggestions)
```

You would need the `streamlit` and `st_keyup` libraries. You can install with `pip`.