A assignment of word processing and guessing.

The writing here about hangman and NLTK should not be used as factual information, but as an expression of my opinions.

To obtain a pdf of this page, click here. Please note that the pdf is not a 1 to 1 representation of this page and is rendered using `pandoc` and `texlive`.

The code is located here. Executing this file is done via the command line using the command `python3.11 nlpwordguess.py <textfile>` and **requires** python 3.11 due to type hints. If you are not using python 3.11, remove them from the function signatures.

Word ingestion

During the writing of the assignment, I found working with NLTK to be relatively easy when I knew what I was looking for. It's stable, and provides output in a string based format. Some improvements that could be done include output of enumerated types where parts of speech are concerned, for example replacing "NN" with a POS.Noun type.

Hangman

Hangman is weird. It's easy to program, just create a shadow list of unguessed vars, then compare the guess across the word. The main problem is trying to look at the guess per loop without creating ugly code. With python, checking for a false-ish value per loop is very easy, then checking for a corresponding guess in the word is quite easy.

Whython

Python has some interesting rules regarding sets and dicts. You can't slice them, you can't index a set, you can't immediately index a dict without checking if it even has the key, etc. One thing that python does that particularly messed with with project output was ordering done with set/dict constructors. Python doesn't guarantee ordering with sets and dicts, only that they exist in the output. This means that when processing with pos_tag, which uses word ordering and proximity to understand language, it will actually lose words after processing.