Ignacio Streuly
Natural Language Processing
Adam Meyers
HMM POS Tagger

System is written and Python and is run simply by running:

```
$ python system.py
```

When in the project directory. The script reads from a file training_corpus.pos which is a merged version of the development and training corpora originally used for this assignment. The output file, ijs253.pos is the output from the POS tagger. On the console will print the amount of lines read from the file training_corpus.pos and the total amount of POS tags added to the output file.

The program works by using a few different tables to handle different edge cases. First, it runs through and creates a likelihood table for words, so, all words in the corpus are added and the likelihood for which POS tag they should appear as is calculated. Next, to handle cases in which the word does not exist in the final_run_system.words file, originally WSJ_23.words, I created another likelihood table to assess the frequency of all POS tags in the training corpus relative to the total amount of POS tags in the training corpus. This comes in handy later when actually deciding the output.

Next, I made a table assessing prior probabilities, since we were to use bigrams, the table takes into account the frequency of each 2 sequence combination of POS tags, or $\frac{freq(prevPos, currentPos)}{freq(prevPos)}$ in a floating point number.

Then the program runs through the final_run_system.words file word by word. For each POS tag that a word comes up for, it then multiplies it by the prior probability of the last tag, returning the max probability, the product of likelihood and prior probability, and adds it to our output sequence. This is if it's in our original word-likelihood table.

If our final_run_system.words has a word that fails the lookup on our word-likelihood table, we use the overall likelihood relative to POS tags in our training document to sub in. Thus, for each POS tag we recorded while training, we multiply its probability by our prior probability table and once again return the maximum.

Therefore, each part of the sequence relies on the last POS tag that we assigned, and each POS tag is initially assigned with the best possible probability, relative to our training corpus.