CS 120 (Spring 22): Introduction to Computer Programming II

Long Project #9 - Rhymes

due at 5pm, Tue 29 Mar 2022

REMEMBER: The `itertools` and `copy` libraries in Python are **banned.**

1 Overview

You will write a program named `rhymes.py` . It reads in a long "pronunciation dictionary," which is a text file (**not** a Python dictionary) that indicates how different words in English are pronounced. You will store this data into an internal data structure (probably a Python dictionary).

You will then read words from the user, one word per line. For each one, you will look up the word in the dictionary. You will then search the rest of the dictionary to find all **other** words which rhyme with it. You will print out all of the rhymes that you find. Keep reading words from the input, until you read the end of the line.

2 Background

Most of us have, at some point in our lives, struggled to find just the right rhyme for some word. Figuring out good rhymes isn't always easy, but it might help if we could easily come up with a list of possible rhyming words to choose from. This program is a step in this direction.

This problem involves writing a program to find all the words that rhyme with some given word. To do this, it will be necessary to reason about the pronunciation of different words; this information will be read from a file (the "pronunciation dictioanry") that gives information about the pronunciation of a wide variety of words. For this assignment, we will use pieces of the file `PronunciationDictionary.txt` as pronunciation dictionaries. In particular, for testing and debugging your code you can use smaller subsets of this file. (Several such subsets are available on the class website, and you can also create your own.)

(spec continues on the next page)

2.1 Definition of a Rhyme

It turns out that there are many different kinds of rhymes (see Wikiepedia for a discussion). For the purposes of this assignment, we will focus on perfect rhymes. Wikipedia defines this kind of rhyme as satisfying the following two conditions:

- The stressed vowel sound (i.e., the primary stress) in both words must be identical, as well as any subsequent sounds. For example, "sky" and "high," or "skylight" and "highlight."
- The phoneme that preceds the stressed vowel must **not** be the same. For example, "bean" and "green" is a perfect rhyme, but "leave" and "believe" is not.[1]
- If a word does not have a primary strees, or if the primary stress is on the first phoneme, then (for the purposes of this program), we will say that it has no rhymes. (I'm not sure what linguists would say about this, but it's good enough for our project.)

In the rest of this document, "rhyme" refers to perfect rhymes unless otherwise specified.

2.2 How Phonemes Work

The pronunciation dictionaries used in this assignment are based on the system used in the Carnegie Mellon Pronunciation Dictionary. Basically, they have built a list of all of the sounds used in English, and have built a very long list of words, along with the pronunciation(s) for each word. (Some words have multiple pronunciations.)

The pronunciation is broken into "phonemes," which are basic units of sound; each possible phoneme is represented by a string of letters. For instance, the "uh" sound in "hut" is represented by `AH`. In the pronunciation dictionary, the entry for "hut" looks like this:

```
HUT  HH AH T
```

which means that the word is made up of three phonemes.

(Note that phonemes are **not** the same as syllables: a syllable can contain many phonemes.)

[1]**Linguist Note:** This definition is not strictly correct, because syllables have stress, not phonemes. And rhymes are defined by the start of the syllable being different, **not** the single phoneme. However, our dataset only tells us about phonemes, not syllables, and so we are limited: our programs can only make decisions based on the phonemes.

As such, our rhyming program will sometimes reject rhymes which it should find. But it's the best we have for now.

2.3 Accents

In most (not all) words, some of the phonemes have "stress" - meaning that they are emphasized more than the rest. There are three possible types of stress; each is represented by a number attached to the phoneme:

- 0 : no stress
- 1 : primary stress
- 2 : secondary stress

For instance, consider the words "weekly" and "computerize." In "weekly," the primary stress is on the second phoneme, and "no stress" is on the last. In "computerize," the primary stress is on the sixth phoneme, secondary stress is on the ninth, and "no stress" is on the second and eighth phonemes.

```
WEEKLY  W IY1 K L IY0
COMPUTERIZE  K AH0 M P Y UW1 T ER0 AY2 Z
```

3 Program Requirements

Write a program, named `rhymes.py` . Prompt the user for a filename; (don't print out any prompt text). Don't worry about having a loop here; we won't test this program with invalid input for the first line. Open the requested file, and read the contents; it will be a Pronunciation Dictionary, made up of many lines like the entries we've seen above. (We will give you a few examples on the class website.)

Store the information from the Pronunciation Dictioanry into a variable (I would recommend a Python dictionary).

Once you have read the contents of the pronunciation dictionary (and stored it into some internal data structure), read each additional line of input, all the way to EOF (end-of-file). Do not print any prompts. For each line of input, sanity-check that the line is reasonable:

- If the line is blank (that is, only has whitespace), print

  ```
  No word given.
  ```

  followed by a blank line, and then immediately read the next line of input.
- If the line has more than 1 word on it, report the error

  ```
  Multiple words entered, please enter only one word at a time.
  ```

  followed by a blank line, and then immediately read the next word of input.
- Otherwise, do the rhyme search described below. After printing the output described below, print a blank line and then read the next line of input.

3.1 The Search

Each time that you read a valid line of input, you will search the pronunciation dictionary (or rather, your copy of it in memory), looking for rhymes of that word. You will print out all of the rhymes you find. Print out the rhymes with one rhyme per line, sorted alphabetically, like this:

The user types "silly" as the first line. You print:

```
Rhymes for: SILLY
  CHILE
  CHILI
  LILLY
```

The user types "Sour" as the first line. You print:

```
Rhymes for: SOUR
  DOUR
  OVERPOWER
  SHOWER
  THREE-HOUR
  TOWER
```

The user types "FOUR" as the first line. You print:

```
Rhymes for: FOUR
  BOER
  DOOR
  STORE
```

If, during any search, you find the same word twice, only print it once.

If a word doesn't have any rhymes, then print the line `-- none found --` in place of any words.

The user types "difficult" as the first line. You print:

```
Rhymes for: DIFFICULT
  -- none found --
```

3.2 Case Sensitivity

Throughout this program, all data (read from the pronunciation dictionary, or words-to-search provided by the user) should be treated as **case insensitive.**

However, in order to make it possible to auto-grade this assignment, you must print out your results in ALL CAPS (except as shown in the examples above).

(spec continues on the next page)

3.3 Multiple Pronunciations

Some words have multiple pronunciations in the dictionary. Some of these are simply different pronunciations of the same word, like "necessity:"

```
NECESSITY  N AH0 S EH1 S AH0 T IY0
NECESSITY  N AH0 S EH1 S IH0 T IY0
```

Others are homonyms, like "read:"

```
READ  R EH1 D
READ  R IY1 D
```

When you encounter this, you must record **all** versions of each such word, and your search must consider all of them. That is, if the user types "read" at the input, then you must list both "EMBED" and "NEED" as rhymes, since you don't know which pronunciation of "read" the user intended.

Likewise, if the user types "need", you must print out "READ" (since "need" rhymes with one of the pronunciations of "read"); you must also print out "READ" if they ask for "embed."

4 Turning in Your Solution

You must turn in your code using GradeScope.

5 Acknowledgements

Thanks to Saumya Debray and Janalee O'Bagy for many resources that I used and adapted for this class.