Rhyming Dictionary

2 submission (100 total points)
Scala Due: Thursday, February 14 @ 11:59pm
Python Due: Sunday, February 17 @ 11:59pm

Updates and Pitfalls

- Notice the deadline changes
- The dictionary filename on the server is "data/cmudict-0.7b" and you should use this exact name in your Test files. You can create a data directory and save your copy of cmudict-0.7b there to use the same test files on your laptop
- Reading files is slow. If you're program reads cmudict-0.7b ~133,000 times (the number of words in the dictionary) when findRhyme is called it will take a very long time to run (equivalent of reading about half a TeraByte). You should implement findRhyme in a way that limits the number of times the file is read, ideally only reading it once
- Remove Deja from the dictionary. It's the last d word and contains accents that can cause encoding errors. It will be removed from the file on AutoLab
- [Scala] If you have an Array and you need a List: .toList works on an Array just like the other type conversions we've seen in class
- [Python] In IntelliJ if you setup your Python project just like our Scala projects and are having trouble importing your own files: right click src -> mark directory as -> sources root. Each package must have an "__init__.py" file in it to mark it as a python package. Now you should be able import RhymingDictionary from your testing files
- [Python] Follow the same class naming convention as Scala for your testing files (ie. Class name matches filename)
- [Python] Do not add any of the graded functions to a class. They should be defined at the top level of the file
- [Python] IntelliJ runs python files from the directory of the file. If you are running a file from the tests package and looking a file in the data directory it will look for "tests/data/<filename>" to avoid file not found errors but still use the same filename as the server you can move your data folder to the tests package

Objective

Gain experience writing and testing functions in Scala and Python

Description

Using a pronunciation dictionary write an application in Scala and Python that that finds rhyming words.

All of the criteria must be completed in both Scala and Python which are submitted separately. These will count as 2 separate homework assignments each worth 50 points for a total of 100 points.

Data

Download: cmudict-0.7b
Documentation: http://www.speech.cs.cmu.edu/cgi-bin/cmudict

Grading Note: The cmudict-0.7b file will be in the AutoLab exactly as-is (With the word Deja removed). Do not modify your copy of the file since those modification won't be made on the grading server. The filename may also be different in AutoLab so be sure to use the filename parameter in your functions and use the filename "data/cmudict-0.7b" in your unit testing code.

To determine if two words rhyme we will first need a way of finding the pronunciation of words. For this purpose we will use the dictionary given above. Please visit the source for documentation on the data and study the format used so you can properly parse and interpret it in you code.

Define Rhyme

The pronunciation dictionary provides the sounds for each word and each sound has a type. For the types we are only concerned if the sound is a vowel or not. All vowel sounds end with an integer that determines the stress given to that vowel and other sounds do not have this stress so a vowel can be identified by checking if a sound ends with an integer. We will say that two words rhyme if all the sounds from the last vowel and after are the same. Words will rhyme even if the last vowel sounds have different stress. By defining a rhyme in this way we effectively consider two words to rhyme if their last syllable have the same sound, ignoring stress.
For example all the following words from the dictionary will be considered rhymes since the last vowel is the same and all sounds after the last vowel are the same:
HALF HH AE1 F
PHOTOGRAPH F OW1 T AH0 G R AE2 F
STAFF S T AE1 F
LAUGH L AE1 F

These words also all rhyme with each other:
THOUSAND TH AW1 Z AH0 N D
DIAMOND D AY1 M AH0 N D
FUND F AH1 N D
AND AH0 N D
SECOND S EH1 K AH0 N D
ISLAND AY1 L AH0 N D
LEGEND L EH1 JH AH0 N D

These words do not rhyme the sounds after the last vowel are different:
DEFINE D IH0 F AY1 N
RHYME R AY1 M

These words do not rhyme since the sounds after the last vowel are not all identical:
DIAMOND D AY1 M AH0 N D
DIAMONDS D AY1 M AH0 N D Z

Project Structure

1. Create a new project in IntelliJ
2. In the src folder create a package named rhymes
3. In the rhymes package create a new {Scala object, Python file} named RhymingDictionary
4. In the src folder create a package named tests

Primary Objective (35 points)

In RhymingDictionary write a method/function named findRhymes that takes two Strings as parameters and returns a List of Strings. The first input String represents the filename for the pronunciation dictionary. The second input String is a word for which you will find rhymes. Return a list containing all the words in the pronunciation dictionary that rhyme with the input work, including the input word itself. If the input word is not in the dictionary, return an empty list.

This findRhymes function is the primary goal of the assignments. The rest of the assignment is designed to guide you as you word towards this goal.

Testing Objectives (45 points)

*The total points of the homework add to more than 50, but your total points will be maxed out at 50 for each language. This gives you some options on how you approach and complete the assignment, though for full credit you must complete the primary objective and

at least 1 testing part. If you don't complete the primary object but want partial credit then you must complete testing objectives.

To earn the 15 points for a testing objective you must:

1. Write unit tests for the given function
    - In AutoLab there will be both correct and incorrect solutions. Your test suite will be marked correct only if it passes all of the correct solutions and fails on all incorrect solutions
2. Implement the given function and pass your unit tests

The 15 points for each testing objective are all-or-nothing. There will also be no useful feedback given by AutoLab. The objective is to give you practice testing your own code without the assistance of AutoLab to prepare you for developing your own projects outside of class, for your team project, in future courses, and throughout your career.

Testing Objective 1

In RhymingDictionary write a method/function named isRhymeSounds that takes two Lists of Strings representing the sounds for two different words and returns a Boolean indicating whether or not the two words rhyme (ie. Return true if the words rhyme, false otherwise).

In the tests package create an object/file named TestIsRhymeSounds that is set up as a test suite that tests the isRhymeSounds method/function.

Testing Objective 2

In RhymingDictionary write a method/function named getSounds that takes 2 Strings, the pronunciation dictionary filename, then a word and returns a List of Strings that are the sounds for that word from the pronunciation dictionary. If the word is not in the dictionary, return an empty List.

In the tests package create an object/file named TestGetSounds that is set up as a test suite that tests the getSounds method/function.

Testing Objective 3

In RhymingDictionary write a method/function named isRhyme that takes 3 Strings, the pronunciation dictionary filename, then two words (Strings) and returns a Boolean indicating whether or not the two words rhyme (ie. Return true if the words rhyme, false otherwise). If either word is not in the dictionary, return false.

In the tests package create an object/file named TestIsRhyme that is set up as a test suite that tests the isRhyme method/function.