Data Transformation

CompSci 101

February 5, 2013

1 Introduction/Goals

The goal of this assignment is to write a program that transforms text. This is done by reading a text input file, performing the transformations on each word and then writing the transformed words to an output file.

In the context of this assignment you must:

- Write 2 different functions that transform a word. These functions are described in section 3.
- Modify an existing program so a user selects a text file that he wants to transform.

In order to do so, a source code file is provided in the form of a *python module* and students will build all the additional functionalities on top of that module.

It is strongly recommended that before continuing with section 2 you download the python module that you will need to modify so you can actually see the relevant source code. The code can be found here: `http://www.cs.duke.edu/courses/fall12/compsci101/assign/pig/code/FileTransform.py`

2 Code Revision

In this section are given instruction on how to tweak the original code found here `http://www.cs.duke.edu/courses/fall12/compsci101/assign/pig/code/FileTransform.py` in order to expand its utility.

2.1 Overview

You need to download the original source file and make modfications to it in order to have the desired funcitonality. After all the modifications are made the program should do the following:

1 Ask the user choose a file to read.

2 Ask the user which transformation to apply to every word in the file read.

3 Apply the transformation.

4 Save the result to a file specified by the user.

In Fig. 1 you see what the program is supposed to do given that the user has chosen to apply the pig-latin transformation.

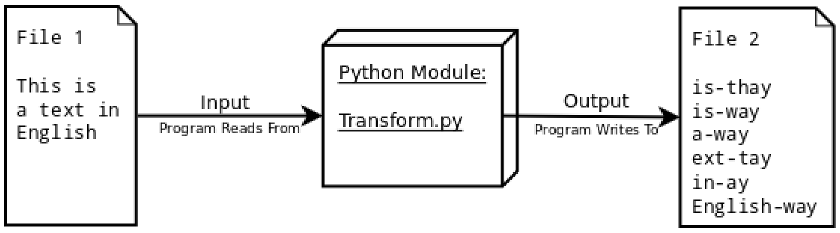

Figure 1: Program Abstraction

2.2 Modifications

The structure of the code is given to you and you need to the following modifications:

- Write new code that implements two new word transformations (Rot13 and pig-latin which are described in section 3).
- Modify the existing code to support these new transformations.
- Modify the existing code to implement the save to file functionality.

The main functions that you will have to change are the following:

> **transform_***(word):** there are 3 functions of this type, each of which takes a string as an argument and performs a type of transformation.
>
> *The functions that you will write on section 3 fall into this category of functions.*
>
> **choose_transform():** this function opens a menu so the user can choose which transformation he wants to perform. It **returns a string** relevant to the transformation chosen.
>
> *You have to tweak this function so it takes into account the new transformation that you will add (Rot13).*
>
> **transform(lines, func):** this is the heart of our program, it takes as an argument a list of lists that contains all the original words and a string (which is returned by choose_transform()) dictating which transformation to apply on these words.
>
> *You actually need to write the code that parses all items of the input list and then given the 'func' argument performs the correct transformation - by choosing the transform_*** function.*
>
> **write_words(fd, words):** this function writes all the words found in the 'words' double nested list in the Eclipse terminal.
>
> *You need to tweak it so it writes instead in the file that corresponds to the 'fd' file descriptor and not just the output*

The final program should initially read a text file and save that text in a nested list. Then the program should transform each word of the text, this transformation will be done based on specific rules. Lastly, the program should save the transformed text on a new file.

3 Transforms

In this section it is needed to implement two different functions that perform a transformation over a word. Moreover, these functions should:

> Take as an argument (input) a **string** which corresponds to the original word.
>
> Return a **string** (that's the output of the function) which corresponds to the transformed word

In the context of these functions each character should be treated as a *vowel* or a *non-vowel* - i.e. "-" is a non-vowel.

In the following subsections it the specifications for each transformation are described thoroughly.

3.1 Pig Latin

Create a python function named "**transform_pig**" that accept a string as an argument and returns a transformed string according to the following rules:

- Any word that begins with a vowel ('a', 'e', 'i', 'o', 'u') is transformed by simply appending the string "-way" to its end. Some examples:

  anchor to **anchor-way**

  elephant to **elephant-way**

  it to **it-way**

- Words that begin with a non-vowel are transformed by removing the non-vowels from the beginning and adding them to the end of the word with a hyphen before and the "ay" string after. "Y" is considered a non-vowel when it is the first character of a word.

  The following examples illustrate these rules:

  superbowl to **uperbowl-say**

  strike to **ike-stray**

  yesterday to **esterday-yay**

  symbiotic to **ymbiotic-say**

- Any word starting with "qu" should be treated as the 'u' is a non-vowel. Some examples:

  quite to **ite-quay**

  queue to **eue-quay**

  quote to **ote-quay**

It is obvious that the Pig Latin transformation does not produce unique matchings - i.e. *it* and *wit* both produce *it-way.*

3.2 Rot13

Create a python function named "**transform_rot13**" that accept a string as an argument and returns a transformed string according to the following simple rule of the Rot13 transformation.

Rot13 stands for "*Rotate 13 places*" and is a simple letter substitution transformation where each letter is replaced by the letter 13 letters after it in the alphabet.

An example of this transformation is shown in the image below:

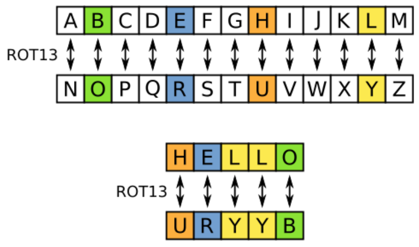

Figure 2: Replacement table

It is obvious that the Rot13 transformation is a bidirectional transformation meaning that the Rot13 transformation of the Rot13 transformation of any word is the word itself - i.e. HELLO $\Rightarrow^{Rot13}$ URYYB $\Rightarrow^{Rot13}$ HELLO.

Fig. 2 is property of wikimedia.com and was found here on Feb. 4 2013: `http://upload.wikimedia.org/wikipedia/commons/3/33/ROT13_table_with_example.svg`

4 Deliverables

The following are expected:

- The revised python module as described in section 2 - this module should contain the code you wrote in section 3.
- A README plain text file that contains valuable information like additional comments, the resources you used to finish the assignment etc.

Working together is permitted, what is not permitted is working together without stating so in the README file. Plagiarism will not be tolerated and we expect everybody to comply with the Duke Community Standard.

Important Dates

- Turn in by Feb. 21: Full points
- Turn in by Feb. 24: 10% down
- Turn in by March 3: 50% down

All deadlines are at 11:55pm on that day. **Anything turned in after 11:55 on Feb. 14 will not be graded**.

How to submit: Either via Eclipse Ambient or the web submit system. Links to these two methods can be found on the Large Assignment tab on the course website.

5 Appendix

New Concepts With this assignment you are introduced to the following common programming concepts:

- **Nested lists:** lists whose elements are lists themselves. An example would be this segment of code:

```
teams = [["San Francisco", "49'ers"], ["Baltimore", "Ravens"], ...]
print teams[1][1]
```

  Which prints "Ravens" in the terminal.

- **Code revision:** one of the most common approaches in the software industry is to revise the code that other people have written - i.e. add more functionalities, fix bugs, etc. Part of this assignment is to revise a python module provided to you.

- **File I/O:** reading from and writing to files. Another common concept in the software industry is to write programs whose input is so large that is saved in another file, and similarly the output needs to be saved in a file as well. In this assignment you will need to do both!

Input Files In this url `http://www.cs.duke.edu/courses/fall12/compsci101/assign/pig/data/` you will find some plain-text files which you can use as an input file for your program - i.e. the program will extract words out of these files.

Functions found in Module

get_words(file): this function reads from file and returns a list of lists where each element of the outer list corresponds to a line of the file and each element of the inner lists corresponds to a word found on that particular line of the file.

Hints When in programming we encounter a data structure with multiple dimensions and we need to parse all elements of that data structure we usually need multiple iteration statements.

An example is a table which resembles a grade-book where each line corresponds to a student and each row corresponds to grades of that student. Then in order to revise a student's grades we need to actually find which line corresponds to that student and then at each row of that particular line add new grades. The only way to do that is to iterate through each line until we find our target student, and then iterate through each row of that line to update his grades.

Whenever you see a 'TODO' string in the comments it means that you actually need to do something there.