Project: Generator of paraphrasing

Author: Aleksandra Buczma

1. General information

   The aim of this project was to create a program in Python which would enable the user to paraphrase the text (f. e. poetry). Since this project doesn't use NLP algorithms, the paraphrasing is based on 3 things:
   a) Switching synonyms (good → right)
   b) Adding related adjective to the word (road → main road)
   c) Switching rhyming words, at the end of the line (road → railroad)

2. How is generator made?

The whole process has mainly 3 steps: getting the text of the poem with an API, extracting all the words from it, and paraphrasing (what is it- explained in point 1.)

The project is divided into 5 files:

- Config.json

  This is the file when all urls, file paths and o non-changeable words are stored

- Saved_lyrics.txt
  This file is empty at the beginning, then stores the original text of the poem

- Defs.py- get_defs, all_urls, words_not_to_change, saved_lyrics

  In this file the data from config.json is read. We first load all the data using json.load(), and then extract specific data: f. e. all_urls() returns the dictionary with needed urls

- Getting_lyrics.py- Class Poem, write_to_lyrics, read_from_lyrics

  Class Poem: gets the text of the poem for given title and author

- Paraphrasing_tools.py- Class Paraphrasing Tool, get_paraphrasing_tool

  Class Paraphrasing Tool – does the three paraphrasing thing (point 1) for a given word

- Paraphrase.py- Class Paraphrase

  For given text (which is the data from read_from_lyrics()) first "qualifies" the words that are suitable for paraphrasing, and then chooses from it as many elements as is required for given percentage. Then does the main process of paraphrasing: the same what Paraphrasing Tool does, but for the whole text.

- Main.py
  Asks user for title, author and percentage, then displays original and later paraphrased text

3. How to use the generator?

The Usage of this program is quite simple. The program is run in terminal. After running the program, first this communicate is displayed:

“Welcome to paraphrasing, here you can paraphrase any poem that can be find on the internet. What would you like to paraphrase?”

1. The user has to type in the title of the poem
2. Then he has to type the author
3. Then he has to type what percentage of the text he would like to be paraphrased (a number in range (0,100))
4. Then the original text is displayed
5. And later on (usually after ca. 1 minute) the paraphrased text is displayed

4. “To be or not to be?” – what could be done better?

In conclusion, looking now at the project, the logic behind it may seem quite simple. At first it looked really complicated, but most of the complex work can be done by APIs (looking for synonyms etc. and getting lyrics).

Before actually writing any code I studied deeply given APIs, which immediately eliminated one of my ideas. The lyrics.ovh (API for song lyrics) wasn’t working, when I looked up, I found out that I wasn’t the only one. And since I couldn’t find any public API, which would do the same (or just similar). So I narrowed down my project to just Poetry, but that doesn’t really change the complexity of the problem.

The other thing I didn’t manage to do, was to implement GUI interface. I didn’t give it much thought as the project is all based on text. When the suggestion came, I had already finished my project and didn’t manage changing it before deadline.

Overall, my first concept wasn’t much different from what I actually managed to do. The only really unexpected aspect was how long it takes to run this program. Single request to API takes ~ 5 seconds, but to paraphrase the whole text 2-3 minutes are needed.