Junioraufgabe 1: Reimerei

Team-ID: 00944

Team: Dominik

Bearbeiter/-innen dieser Aufgabe:
Dominik Korolko

18. November 2022

Inhaltsverzeichnis

Lösungsidee

Für die Lösung werden immer zwei verschiedene Wörter miteinander verglichen und von beiden Wörtern wird der “Reimteil” bestimmt, also die maßgebliche Vokalgruppe und alles, was danach kommt. Wenn der Reimteil bei beiden Wörtern gleich ist, gelten diese unter folgenden Einschränkungen als Reimpaar.

Bei der Bestimmung des Reimteils ist zu beachten, dass die Wörter verschiedene Groß- und Kleinschreibung haben können. Der Vergleich muss also unabhängig von Groß-/Kleinschreibung funktionieren.

Eine weitere Einschränkung ist, dass das eine Wort im Reimpaar nicht mit dem kompletten anderen Wort enden darf. Auch hier darf Groß-/Kleinschreibung nicht beachtet werden.

Außerdem ist zu beachten, dass der Reimteil mindestens die Hälfte des gesamten Wortes umfassen muss. Wenn dies für ein Wort nicht gilt, kann dieses kein Reimpaar mit einem anderen Wort bilden.

Ich habe mich dafür entschieden, die Buchstaben "a", "e", "i", "o" und "u" als Vokale zu behandeln, ebenso die Umlaute "ä", "ö" und "ü". Der Buchstabe “y” gilt als Vokal, wenn keine anderen Vokale im Wort sind, da “y” meistens die Funktion eines Vokals annimmt, wenn es ohne andere Vokale im Wort steht. Dies kommt besonders oft im Englischen vor. Ansonsten gilt “y” als Konsonant, wie alle anderen Buchstaben und Sonderzeichen.

Darüber hinaus habe ich mich gegen die Betrachtung von “ß” als “ss” entschieden, da Wörter, die sich im Reimteil durch “ß” und “ss” unterscheiden, meistens keine Reime sind. Beispiel: Maße und Masse.

Der Reimteil ergibt sich aus der vorletzten Vokalgruppe, wenn das Wort zwei oder mehr Vokalgruppen hat, ansonsten die letzte Vokalgruppe. Als Vokalgruppe gilt jede längstmögliche Folge von Vokalen im Wort. Zwischen zwei Vokalgruppen muss immer mindestens ein Nicht-Vokal sein.

Umsetzung

Für die Lösung dieses Problems habe ich die Programmiersprache Python verwendet. Nachdem die Wörter einzeln eingelesen wurden, wird jedes Wort mit jedem anderen verglichen, wofür ich eine doppelte for-loop benutzt habe.

Um Performance zu sparen und keine doppelten Reimpaare in umgekehrter Reihenfolge zu haben (gehen:stehen und stehen:gehen sollen nicht als zwei verschiedene Paare gelten) und damit Wörter nicht mit sich selber verglichen werden, beginnt die zweite for-loop mit dem Wort nach dem Wort der ersten for-loop und überspringt somit sich selber und alle Wörter vor sich selber.

Als Erstes werden beide Wörter auch in komplett kleingeschriebener Form gespeichert, damit die Groß- und Kleinschreibung keine Rolle spielt. Mit dieser kleingeschriebenen Form findet der ganze Vergleich im Programm statt, am Ende wird aber die Originalform des Wortes zurückgegeben und gespeichert. Danach wird überprüft, ob eins der Wörter mit dem kompletten anderen Wort endet. Wenn dies der Fall ist, wird die Iteration übersprungen, da diese Wörter kein Reimpaar sein können.

Für die Bestimmung des Reimteils eines Wortes iteriert die get_rhyme_part Funktion rückwärts durch das Wort. Dabei wird es in einer Variable gespeichert, wenn die letzte Vokalgruppe des Wortes erreicht wurde, wenn die Konsonantengruppe vor der letzten Vokalgruppe erreicht wurde und schließlich, wenn die vorletzte Vokalgruppe, also die maßgebliche Vokalgruppe, erreicht wurde. Für jeden iterierten Buchstaben wird überprüft, ob er ein Vokal ist. Mit dieser Information und dem Stand der drei Variablen werden die Variablen entsprechend verändert, bis die vorletzte Vokalgruppe durchlaufen ist.

Die Überprüfung, ob ein Buchstabe ein Vokal ist, findet der is_vowel Funktion statt. Wenn der Buchstabe “y” ist, wird überprüft, ob im Wort noch andere Vokale sind. Wenn nicht, wird True zurückgegeben, sonst False. Ansonsten gibt die Funktion zurück, ob der Buchstabe in der Konstante mit den Vokalen “aeiouäöü” vorhanden ist.

Während der Iteration wird außerdem immer überprüft, ob vom Iterationsort bis zum Anfang des Wortes nur noch Konsonanten vorhanden sind. Wenn dies der Fall ist, wird der ganze Wortteil vom Iterationsort bis zum Ende des Wortes gespeichert. Dies ist wichtig, damit bei Wörtern mit nur einer Vokalgruppe, die dann die maßgebliche Vokalgruppe ist, der korrekte Teil zurückgegeben wird.

Schließlich überprüft die Funktion, ob der Reimteil aus mindestens der Hälfte des Wortes besteht. Wenn dies nicht der Fall ist, gibt die Funktion None zurück, ansonsten wird der Reimteil zurückgegeben.

Die übergeordnete Funktion mit der doppelten for-loop fügt das Reimpaar letztendlich zu einer Liste hinzu, wenn die Ergebnisse nicht None sind und beide Reimteile gleich sind. Die Liste der Reimpaare beinhaltet nach dem Beenden der beiden for-loops alle Reimpaare und diese werden in der Konsole ausgegeben.

Beispiele

Beispiel 1 - reimerei0.txt

Der Reimteil beim Wort “breitschlagen” ist kürzer als die Hälfte des Wortes, also bildet das Wort kein Reimpaar mit einem anderen.

Eingabe:
bemühen
Biene
breitschlagen
glühen
hersagen
Hygiene
Knecht
Recht
Schiene
Schlank
Schwank

Ausgabe:
Alle Reimpaare:
bemühen : glühen
Biene : Hygiene
Biene : Schiene
Hygiene : Schiene
Knecht : Recht

Beispiel 2 - reimerei1.txt

Eingabe:
Absorption
Bildnis
Brote
Geständnis
Konsumption
Note
Paprikaschote
Wildnis
XOR-Funktion

Ausgabe:
Alle Reimpaare:
Bildnis : Wildnis
Brote : Note

Beispiel 3 - reimerei2.txt

Eingabe:
Epsilon
Foto
Passfoto
Poststempel
Tempel
Ypsilon

Ausgabe:
Alle Reimpaare:
Epsilon : Ypsilon

Beispiel 4 - reimerei3.txt

Eingabe:
[zu lang für die Dokumentation; die Wortliste befindet sich in der Datei reimerei3.txt]

Ausgabe:

Alle Reimpaare:
Absender : Kalender
Ansage : Frage
Ansage : Garage
Bahn : Zahn
Bank : Dank
Baum : Raum
Bein : Wein
Bier : Tier
Bild : Schild
Bitte : Mitte
Butter : Großmutter
Butter : Mutter
Dame : Name
Dezember : November
Dezember : September
Drucker : Zucker
Durst : Wurst
Ermäßigung : Kündigung
Ermäßigung : Reinigung
Fest : Test
Feuer : Steuer
Fisch : Tisch
Flasche : Tasche
Frage : Garage
Fuß : Gruß
Gas : Glas
Glück : Stück
Gleis : Kreis
Gleis : Preis
Gleis : Reis
Gruppe : Suppe
Hand : Land
Hand : Strand
Hose : Rose
Hund : Mund
Kündigung : Reinigung
Kanne : Panne
Kasse : Klasse
Kasse : Tasse
Kassette : Kette
Kassette : Toilette
Keller : Teller
Kette : Toilette
Kind : Rind
Kind : Wind
Klasse : Tasse
Kopf : Topf
Kreis : Preis
Kunde : Stunde
Land : Strand
Lohn : Sohn
Magen : Wagen
Nachmittag : Vormittag
Rock : Stock
S-Bahn : Zahn
Sache : Sprache
Sekunde : Stunde
Kassette : Toilette
Keller : Teller
Kette : Toilette
Kind : Rind
Kind : Wind
Klasse : Tasse
Kopf : Topf
Kreis : Preis
Kunde : Stunde
Land : Strand
Lohn : Sohn
Magen : Wagen
Nachmittag : Vormittag
November : September
Platz : Satz
Rind : Wind
Rock : Stock
S-Bahn : Zahn
Sache : Sprache
Sekunde : Stunde
November : September
Platz : Satz
Rind : Wind

Eigene Beispiele

Eigenes Beispiel 1 - extra_reimerei0.txt

Bei einer leeren Eingabe stürzt das Programm nicht ab, findet aber auch keine Reimpaare.

Eingabe:
[zwei leere Zeilen]

Ausgabe:
Keine Reimpaare gefunden.

Eigenes Beispiel 2 - extra_reimerei1.txt

Groß-/Kleinschreibung spielt beim Vergleich der Reimteile keine Rolle. Selbst verschieden großgeschriebene Wörter können ein Reimpaar bilden.

Eingabe:
SEE
Fee

Ausgabe:
Alle Reimpaare:
SEE : Fee

Eigenes Beispiel 3 - extra_reimerei2.txt

Wenn das gleiche Wort zweimal in der Eingabe vorhanden ist, gelten diese nicht als Reimpaar.

Eingabe:
gehen
gehen

Ausgabe:
Keine Reimpaare gefunden.

Eigenes Beispiel 4 - extra_reimerei3.txt

Wie beschrieben gilt der Buchstabe “y” als Vokal, wenn das Wort keine anderen Vokale hat. Das ist besonders im Englischen sinnvoll.

Eingabe:
by
my

Ausgabe:
Alle Reimpaare:
by : my

Quellcode

```
VOWELS = "aeiouäöü"


def find_rhyme_pairs(word_list: List[str]) -> List[Tuple[str, str]]:
    rhyme_pairs: List[Tuple[str, str]] = []

    for idx, word_1 in enumerate(word_list):
        lower_word_1 = word_1.lower()
        for word_2 in word_list[idx + 1:]:
            lower_word_2 = word_2.lower()
            if lower_word_1.endswith(lower_word_2) or
lower_word_2.endswith(lower_word_1):
                continue

            word_1_rhyme_end = get_rhyme_part(lower_word_1)
            word_2_rhyme_end = get_rhyme_part(lower_word_2)

            if word_1_rhyme_end and word_2_rhyme_end and
word_1_rhyme_end == word_2_rhyme_end:
                rhyme_pairs.append((word_1, word_2))

    return rhyme_pairs
```

```
def is_vowel(char: str, word: str) -> bool:
    if char == "y" and not any(v in word for v in VOWELS):
        return True
    return char in VOWELS
def get_rhyme_part(word: str) -> Optional[str]:
    at_last_vowel_group = False
    between_vowel_groups = False
    at_penultimate_vowel_group = False
    result = None
    for idx, i in enumerate(reversed(word)):
        if is_vowel(i, word):
            if not any([is_vowel(j, word) for j in word[:-(idx + 1)]]):
                result = word[-(idx + 1):]
                break
            if at_last_vowel_group and between_vowel_groups:
                at_penultimate_vowel_group = True
            at_last_vowel_group = True
        elif at_penultimate_vowel_group:
            result = word[-idx:]
            break
        elif at_last_vowel_group and not between_vowel_groups:
            between_vowel_groups = True

    if not result or (len(word) / 2) > len(result):
        return None
    return result
```