Junioraufgabe 1

Einleitung + Erklärung

Ich habe das Programm in Python geschrieben, da man keine Low-Level-Prozesse benötigt und hier sehr viel mit Listen gearbeitet wird, was in Python einfacher funktioniert.
Dieses Programm wurde in 5 Teile aufgeteilt (die weiter auf 8 Funktionen aufgeteilt wurde). Als erstes gibt es die 1. Regel, dann die 2. und die 3., dafür der Prozess zum Berechnen der maßgeblichen Vokalgruppe und die Ausführung, für entweder 2, oder mehr. Es sind mehrere Funktionen, damit es einfach übersichtlicher ist.
Bei den Funktionen habe ich mir ein bestimmtes System angewöhnt, dass ich auch hier angewendet habe.
Die Funktionen, die nur das Programm benötigt und nicht vom Benutzer benutzt werden sollten besitzen zwei Unterstriche davor.

Detaliertere Ansicht

class CheckWords

Dieser Import ist wichtig, deepcopy benutztz man um listen zu kopiern, um in diesem Fall die Liste nur im Stack-RAM, nicht die Eigentliche zu verändern.
Als erstes habe ich eine Klasse erstellt, was zwar nicht nötig ist, es aber übersichtlicher macht.
In dieser Klasse stehen alle funktionen als Classmethods, damit es nur optional ist ein Objekt zu erstellen (s. CheckWords().__init__())

```
from copy import deepcopy


class CheckWords:
```

vowels

CheckWords.vowels sind die erlaubten Vokale (falls unverändert: vowels = "aeiouäöü")

```
        vowels = ("aeiouäöü")
```

__vowelgroup()

Diese Funktion sucht die maßgebliche Vokalgruppe. Das passiert mit einer Schleife, die läuft, bis (inkl.) ein index i 0 erreicht. Dabei merkt er sich den letzten Vokal (damit falls es nur eine Vokalgruppe ist, man den Anfang hat) als lastvowel.
Das Programm merkt, wenn es bei der 2. Vokalgruppe von hinten ist, da wenn lastvowel schon definiert wurde, heißt es, dass es schon eine Gruppe gab.
Falls es die 2. Vokalgruppe ist, geht der index immer weiter nach unten, bis der Anfang der Vokalgruppe gefunden wurde, die dann zurückgegeben wird.

Falls aber nur eine Vokalgruppe da ist, kommt es nicht bis dahin, und deshalb wird am ender der Schleife lastvowel zurückgegeben.

```
        @classmethod
        def __vowelgroup(cls, word:str) -> int:
                if len(word) == 0:
                        raise ValueError("Not Enough Letters: No letters in
word!")
                lastvowel = None
                if word[len(word)-1] in cls.vowels:
                        lastvowel = len(word)-1
                i = len(word)-2
                while i > -1:
                        if word[i] in cls.vowels:
                                if not word[i+1] in cls.vowels and lastvowel !=
None:
                                        while i > 0 and word[i-1] in cls.vowels:
                                                i-=1
                                        return i
                                lastvowel = i
                        i-=1
                return lastvowel
```

rule1() / rule2() / __rule3()

'das Teilwort' (in der Dokomentation) ist der Teil zwischen dem Anfang der maßgeblichen Vokalgruppe und dem Ende des Wortes.
Die 1. Regel (__rule1), gibt zurück, ob das Teilwort des 1. Wortes word1 und das Teilwort vom 2. Wort word2 gleich ist.
Die 2. Regel, (__rule2) gibt zurück, ob das Teilwort des Wortes word größer oder gleich die hälfte der Anzahl der Buchstaben des Wortes word2 ist.
Die 3. Regel, (__rule3) gibt zurück, ob das 1. Wort word1 im 2. Wort word2 ist, oder umgekehrt.

```
        @classmethod
        def __rule1(cls, word1: str, word2: str, vowelgroupforword1:tuple,
vowelgroupforword2:tuple) -> bool:
                return word1[vowelgroupforword1:] == word2[vowelgroupforword2:]

        @classmethod
        def __rule2(cls, word: str, vowelgroup:int) -> bool:
                return len(word[vowelgroup:len(word)]) >= len(word)/2

        @classmethod
        def __rule3(cls, word1: str, word2: str) -> bool:
                return not word1 in word2 and not word2 in word1
```

Bei Regel 3, geht es zwar darum, dass ein Wort nicht in dem anderen endet, aber, wenn ein Wort in einem drinnen ist, ohne in dem zu Enden (z.B. Gier, Gierig), wird es an der 1. Regel scheitern, also ist es auch so ok. Pythons in ist optimizierter als z.B. eine for-Schleife.

__check()

Diese Funktion gibt zurück, ob 2 Wörter zusammenpassen.

```
        @classmethod
        def __check(cls, word1:str, word2:str, vowelgroupforword1:int,
vowelgroupforword2:int) -> bool:
                return cls.__rule1(word1, word2, vowelgroupforword1,
vowelgroupforword2) and cls.__rule2(word1, vowelgroupforword1) and
cls.__rule2(word2, vowelgroupforword2) and cls.__rule3(word1, word2)
```

check()/init()

This function goes through the list and in one go, it goes through it again. Than there are two indeces, with which we can compare every element in the list with every element in the list. That would be fine, but that's going to return the same thing twice (("Maus", "Haus"), ..., ("Haus", "Haus")). So everytime the inner loop finfisher, the first element of the list is being deleted (that's why the list is being copied at the beginning). That solves another problem: We don't need an index for the outer loop, beacuse the 0th element is always a new one.

Diese Funktion geht durch die Liste mit den Wörtern, und in allen Durchgang, noch einmal. Somit werden alle Elemente in der Liste mit allen Elementen verglichen. Das ist gut, aber 1. es geht effizienter und 2. man hat Wiederholungen im Ergebnis (("Maus", "Haus"), ..., ("Haus", "Haus")). Also macht es das Programm so, dass nach jedem durchgang der äußeren Schleife, das 0. Element gelöscht wird (deshalb wird die Liste am anfang kopiert). Das löst noch ein anderes Problem: Man benötigt keinen index für die äußere Schleife, da das 0. Element, immer das nächste ist.

```
        @classmethod
        def check(cls, wordsoriginal:list[str]) -> bool:
                words = deepcopy(wordsoriginal)
                correctconnections = []
                if len(words) < 2:
                        raise ValueError(f"Not Enough Words: There must be at
least two words. You have {len(words)} words.")
                while len(words) > 1:
                        word1 = words.pop(0).lower()
                        vowelgroupforword1 = cls.__vowelgroup(word1)
                        j = 0
                        while j < len(words):
                                word2 = words[j].lower()
                                vowelgroupforword2 = cls.__vowelgroup(word2)
                                if cls.__check(word1, word2, vowelgroupforword1,
vowelgroupforword2):
                                        correctconnections.append((word1, word2))
                                j+=1
                return tuple(correctconnections)
```

Die initalizing-Funktion der Klasse (__init__), besitzt die selben Parameter wie checkmany() und ruft auch diese Funktion auf.
Das Ergebnis speichert die Funktion in self.state ab.

```
        def __init__(self, *words) -> None:
                self.state = CheckWords.check(list(words))
```

Beispiele

__vowelgroup()

Wort -> alle Vokalgruppenanfänge -> maßgeblicher Vokalgruppenanfang (= __vowelgroup(word))
"Zwei" -> [2] -> 2
"Drei" -> [2] -> 2
"Bei" -> [1] -> 1
"Hai" -> [1] -> 1
"Heis" -> [1] -> 1
"Abtei" -> [3] -> 3
"Aufschrei" -> [0, 7] -> 0
"Ei" -> [0] -> 0
"Wei" -> [1] -> 1
"Agieren" -> [0, 2, 5] -> 2
"Frieren" -> [2, 5] -> 2
"Blockieren" -> [2, 5, 8] -> 5
"Gier" -> [1] -> 1
"John" -> [1] -> 1
"Arrangieren" -> [0, 2, 5, 8] -> 5
"Präsentieren" -> [2, 4, 7, 9] -> 7
"Gieren" -> [1, 4] -> 1
"Ieren" -> [1, 3] -> 1

__rule1()

Zwei, Drei, Bei -> True (bei allen gegenseitig und einzeln)
Zwei, Hai -> False
Zwei, Heis -> False
Agieren, Blockieren, Frieren -> True (bei allen gegenseitig und einzeln)
Agieren, Gier -> False
Agieren, John -> False

__rule2()

Zwei -> True
Drei -> True
Bei -> True
Abtei -> False
Aufschrei -> False
Agieren -> True
Frieren -> True
Blockieren -> True

Arrangieren -> False
Präsentieren -> False

__rule3()

Zwei, Drei, Bei -> True (bei allen gegenseitig und einzeln)
Zwei, Ei -> False
Zwei, Wei -> False
Agieren, Frieren, Blockieren -> True (bei allen gegenseitig und einzeln)
Agieren, Gieren -> False
Agieren, Ieren -> False

__check()

"Zwei", "Drei" -> True
"Zwei", "Bei" -> True
"Zwei", "Hai" -> False
"Zwei", "Abtei" -> False
"Zwei", "Ei" -> False
"Agieren", "Frieren" -> True
"Agieren", "Blockieren" -> True
"Agieren", "Gieren" -> False
"Agieren", "Arrangieren" -> False
"Agieren", "Gieren" -> False

check()

"Zwei", "Drei", "Bei" -> ("zwei", "drei"), ("zwei", "brei"), ("drei", "bei")
"Zwei", "Drei", "Bei", "Hai" -> ("zwei", "drei"), ("zwei", "brei"), ("drei", "bei")
"Zwei", "Drei", "Bei", "Abtei" -> ("zwei", "drei"), ("zwei", "brei"), ("drei", "bei")
"Zwei", "Drei", "Bei", "Ei" -> ("zwei", "drei"), ("zwei", "brei"), ("drei", "bei")
"Agieren", "Frieren", "Blockieren" -> ("agieren", "frieren"), ("agieren", "blockieren"), ("frieren", "blockieren")
"Agieren", "Frieren", "Blockieren", "Gier" -> ("agieren", "frieren"), ("agieren", "blockieren"), ("frieren", "blockieren")
"Agieren", "Frieren", "Blockieren", "Arrangieren" -> ("agieren", "frieren"), ("agieren", "blockieren"), ("frieren", "blockieren")
"Agieren", "Frieren", "Blockieren", "Gieren" -> ("agieren", "frieren"), ("agieren", "blockieren"), ("frieren", "blockieren"), ("frieren", "gieren"), ("blockieren", "gieren")

Junioraufgabe 2

Einleitung + Erklärung

Ich habe das Programm in Python geschrieben, da man keine Low-Level-Prozesse benötigt und hier sehr viel mit Listen gearbeitet wird, was in Python einfacher funktioniert.
Das Programm wurde in 3 Teile bzw. 4 Funktionen (inkl. __init__) aufgeteilt (für übersichtlichkeit). Es gibt eine Initializing-Funktion, was heißt, dass ich eine Klasse erstelt habe (auch fuer

übersichtlichkeit).
Das sind alles Classmethods, dass man kein Objekt erstellen muss, um es auszuführen (man kann aber).
Biggest().__init__() bzw. Biggest.process() sind die hauptfunktionen.
Das Programm funktioniert folgendermaßen:
Man schaut sich das größere Element vom ersten Vergleich an. Danach sucht man in den Vergleichen nach einem Vergleich, in dem das alte Größere, das Kleinere ist. Dann ist das Größere im Neuen, das Größere im Alten und dieser ganze Prozess wird wiederholt, bis es nicht mehr weitergeht. Dabei werden alle Elemente, die schonmal alt waren, aber ersetzt wurden asu der Liste gelöscht.
Dieser ganze Prozess wird mit verschiedenen Anfangs-Vergleichen durchgeführt. Falls alle Ergebnisse gleich sind, wird das Ergebnis zurückgegeben. Ansonsten kann kein klares Größtes gefunden werden --> Fehler.
bei den Vergleichen gilt: (größer, kleiner)

Detaliertere Ansicht

__go()

Die Funktion nimmt das gröste gefundene Element bisher als find und schaut in den Vergleichen comparisons nach, ob das irgendwo an der Kleiner-Stelle Steht.
Falls so etwas gefunden wird, wird dieser Vergleich sofort zurückgegeben.
Falls nichts gefunden wird, wird Nichts zurückgegeben.

```
    @classmethod
def __go(cls, find, comparisons:list):
    for comparison in comparisons:
        if comparison[1] == find:
            return comparison
    return None
```

__biggest()

Diese Funktion benötigt einen Start-Vergleich und die restlichen vergleiche.
Hier wird der Import deepcopy verwendet, der die Liste neu kopiert (für die Funktion, damit die Liste nur im Stack-Speicher verändert wird und die originelle, so bleibt).
Die Funktion geht so lange, bis Biggest.__go() etwas zurückgibt.
Das Ergebnis von Biggest.__go() wird abgespeichert und immerwieder in Biggest.__go() reingegeben, bis nichts mehr zurückgegeben wird. Dabei wird nach jedem Erfolgreichen Durchgang, das abgespeicherte Element aus der Liste gelöscht. Dann wird das letzte, was abgespeichert wurde zurückgegeben.

```
    @classmethod
def __biggest(cls, start, comparisonsoriginal:list):
    comparisons = deepcopy(comparisonsoriginal)
    result = None
    resulttmp = start
    while resulttmp:
        comparisons.remove(resulttmp)
```

```
            result = resulttmp[0]
            resulttmp = cls.__go(result, comparisons)
        return result
```

prozess() / init()

In dieser Funktion wird Biggest.__biggest() mit allen Vergleichen als Start-Vergleich aufgerufen. Falls alle Biggest.__biggest()-Aufrufe dasselbe zurückgeben, wird das Ergebnis zurückgegeben. Ansonsten ist es ein Fehler (man kann das größte nicht 100%ig herausfinden), und deshalb wird auch einer angeziegt.

```
        @classmethod
    def process(cls, comparisons:list):
        old = None
        for comparison in comparisons:
            new = cls.__biggest(comparison, comparisons)
            if new != old and old != None:
                raise ValueError(f"The biggest can't be told exactly! (According
to the calculations it could be: {new, old}, and more uncaclulated ones.)")
            else:
                old = new
        return old
```

Die Initalizing-Funktion ist nur dazu da, dass man ein Object dieser Klasse erstellen kann (und es auch etwas bringt).
Sie speichert das Ergebnis von Biggest.process in self.biggest ab.
Eine Möglichkeit __init__ zu benutzten ist so:
Biggest(comparisons).biggest

```
        @classmethod
    def __init__(self, *comparisons) -> None:
        self.biggest = Biggest.process(list(comparisons))
```

Beispiel

1. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (d, a)
Alt: ***d***

Alle Vergleiche: (d, b), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, b), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***

Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, b), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, b), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (d, ***c***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, b), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (b, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***

2. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (d, b)

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (d, ***c***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c), (b, a)
Alt: ***d***
Neu: (b, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***

3. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (c, a)

Alle Vergleiche: (d, a), (d, b), (c, b), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (d, c), (b, a)
Alt: ***c***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***c***)
Alt Groß =?= Alt Klein? => ***Ja*** -> ***Neu*** ((d, c)) ***wird zu Alt*** (c)

Alle Vergleiche: (d, a), (d, b), (c, b), (b, a)
Alt: ***d***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (b, a)
Alt: ***d***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (b, a)
Alt: ***d***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, b), (b, a)
Alt: ***d***
Neu: (b, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***

4. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (c, b)

Alle Vergleiche: (d, a), (d, b), (c, a), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (d, c), (b, a)
Alt: ***c***
Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (d, c), (b, a)
Alt: ***c***
Neu: (d, ***c***)
Alt Groß =?= Alt Klein? => ***Ja*** -> ***Neu*** ((d, c)) ***wird zu Alt*** (c)

Alle Vergleiche: (d, a), (d, b), (c, a), (b, a)
Alt: ***d***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (b, a)
Alt: ***d***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (b, a)
Alt: ***d***
Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (b, a)
Alt: ***d***
Neu: (b, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***

5. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (d, c)

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (b, a)
Alt: ***d***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (b, a)
Alt: ***d***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (b, a)
Alt: ***d***
Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (b, a)
Alt: ***d***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (b, a)
Alt: ***d***
Neu: (b, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***

6. Durchgang:

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c), (b, a)
Start-Vergleich: (b, a)

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c)
Alt: ***b***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (d, b), (c, a), (c, b), (d, c)
Alt: ***b***
Neu: (d, ***b***)
Alt Groß =?= Alt Klein? => ***Ja*** -> ***Neu*** ((d, b)) ***wird zu Alt*** (b)

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c)
Alt: ***d***
Neu: (d, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c)
Alt: ***d***

Neu: (c, ***a***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c)
Alt: ***d***
Neu: (c, ***b***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Alle Vergleiche: (d, a), (c, a), (c, b), (d, c)
Alt: ***d***
Neu: (d, ***c***)
Alt Groß =?= Alt Klein? => ***Nein*** -> ***Nichts Passiert***

Ende der Liste --> Rückgabe: ***d***