

Aufgabe 2: Twist

Team: Pied Piper

Team-ID: 00342

23. November 2018

Inhalt

Inhalt	1
Lösungsidee.....	1
Twister:	1
Enttwister:	1
Umsetzung	2
Twister:	2
Enttwister	2
Beispiele	3
Twister	3
Enttwister	4
Quellcode	6
Twister	6
Enttwister	7

Lösungsidee

Twister:

Für die erste Teilaufgabe sollen Wörter so chiffriert werden, dass der erste und der letzte Buchstabe gleichbleiben und alle anderen Buchstaben zufällig angeordnet werden. Die Herausforderung in der Aufgabe besteht darin, dass das Programm, trotz Sonderzeichen und Zahlen, Wörter erkennt, die getwistet werden müssen. Dazu soll zunächst das Programm Zeile für Zeile im jeweiligen Text durchgehen. In jeder Zeile soll dabei „Wort“ für „Wort“ getwistet werden. Ein „Wort“ bedeutet hierbei erstmal eine Folge von Zeichen, die durch ein Leerzeichen getrennt ist. Beim Twisten eines „Wortes“ wird also geschaut, welche richtigen Wörter sich darin befinden, die getwistet werden sollen. Dies geschieht, indem Buchstabenfolgen zwischen möglichen Sonderzeichen erkannt werden. Anschließend folgt ein Twist mit dem bereits erwähnten Chiffrierungsverfahren.

Enttwister:

In diesem Aufgabenteil sollen nun getwistete Wörter wieder enttwistet werden. Wie beim Twister, müssen auch hier zunächst „Wörter“, also Zeichenfolgen ohne Sonderzeichen, Zahlen oder Satzzeichen, erkannt werden. Nachdem dies geschehen ist, wird die Liste mit den deutschen Wörtern linear Wort für Wort durchgegangen. Dabei wird abgeglichen, ob die sortierte Zeichenfolge mit dem sortierten Wort in der Wörterliste übereinstimmt. Sortiert heißt dabei, dass in beiden Zeichenfolgen die Buchstaben so vertauscht werden, dass sie alphabetisch aufsteigend sortiert sind.

Umsetzung

Beide Lösungsideen der Teilaufgaben wurden mit Python umgesetzt. Es muss also lediglich die .py-Datei gestartet werden, es folgt ein kleines User Interface.

Twister:

In der Methode `start` wird zunächst der Text abgefragt, der getwistet werden soll. Man kann dabei zwischen einer manuellen Text-Eingabe und der Auslese einer Text-Datei wählen. Anschließend wird jede Zeichenfolge, die durch ein Leerzeichen getrennt ist der Methode `wort_twisten` übergeben, damit das in der Zeichenfolge potenzielle Wort getwistet wird.

Falls das zu twistende Wort weniger als vier Buchstaben hat, muss es gar nicht erst getwistet werden, da es nur eine Möglichkeit gibt, wie man das Wort twisten könnte. Um nun zu definieren, welcher Teil der Zeichenfolge ein Wort enthält, wurden zwei `for`-Schleifen implementiert. Die erste startet beim ersten Zeichen und bricht ab, wenn sie auf den ersten Buchstaben stößt oder das Ende erreicht worden ist. Die Variable `erster_buchstabe_pos` speichert die Position des ersten Buchstabens in der Zeichenfolge und nimmt den Wert -1 an, falls kein Buchstabe gefunden worden ist. Ist dies der Fall, wird die Zeichenfolge wieder zurückgegeben, da sie nur aus Sonderzeichen besteht und letztendlich nur Wörter getwistet werden sollen. Die Variable `praeфик` speichert den Teil der Zeichenkette vom ersten Zeichen bis einschließlich des ersten Buchstabens, da der Teil nicht getwistet werden muss. Nun startet eine zweite Schleife nach der Position des ersten Buchstabens und stellt sicher, dass kein weiteres Sonderzeichen mehr folgt. Ist dies der Fall, werden nun die Buchstaben vertauscht, die nach dem ersten Buchstaben folgen und vor dem letzten Buchstaben enden. Die Vertauschung erfolgt in der Methode `buchstaben_twisten` und das Ergebnis wird in `vertauschte_buchstaben` gespeichert. Anschließend wird die Zeichenkette zurückgegeben, die sich aus `praeфик`, `vertauschte_buchstaben` und `suffix` zusammensetzt. `Suffix` ist nichts anderes als der letzte Buchstabe in dem Fall. Stößt die zweite Schleife jedoch auf ein Sonderzeichen, kommt es zu einem rekursiven Methodenaufruf. Alles, was nach dem Sonderzeichen folgt, wird in der Methode `wort_twisten` extra getwistet und in der Variable `naechstes_wort` gespeichert. Genauso wie beim Fall, dass kein weiteres Sonderzeichen mehr folgt wird nun die Zusammensetzung aus Präfix, getwisteter Buchstaben und das Suffix zurückgegeben mit dem Zusatz des nächsten Wortes.

Der wahre Twist findet also in der Methode `buchstabe_twisten` statt. Hierzu werden die zu twistenden Buchstaben in einer Liste gespeichert, die anschließend mit der Methode `random.shuffle()` zufällig angeordnet wird. Die Liste wird dann wieder in eine Zeichenkette umgewandelt mit `"".join(buchstaben_liste)` und anschließend zurückgegeben.

Anmerkungen:

1. Theoretisch könnte die Methode `wort_twisten` eine ganze Zeile richtig chiffrieren, jedoch würde die rekursive Tiefe bei langen Zeilen den Rahmen sprengen und es käme zu Performance-Verlusten.
2. Mit Sonderzeichen ist alles gemeint, was kein Buchstabe ist (Zahlen, Leerzeichen und reguläre Sonderzeichen)

Enttwister

Genauso wie beim Twister findet in der Methode `enttwiste_wort` die Worterkennung statt. Da die Umsetzung dieses Prozesses bereits erklärt wurde (in den ersten beiden Absätzen unter Umsetzung, Twister), wird an dieser Stelle auf eine wiederholte Erklärung verzichtet, da die Programmteile nahezu gleich sind.

Aufgabe 2: Twist

Der zweite Teil des Programms dagegen ist nur für diesen Aufgabenteil. Um das getwistete Wort zu enttwisten, muss es mit den Wörtern in der Liste abgeglichen werden. Dies geschieht in der Methode `wort_in_liste_finden`. Dazu wird mithilfe der `codecs`-Bibliothek, die auch deutsche Umlaute berücksichtigt, die Wörterliste geöffnet und alle darin enthaltenden Wörter in der Liste `alle_deutschen_woerter` gespeichert. Nun wird in einer `for`-Schleife jedes einzelne Wort für Wort durchgegangen. Dabei gibt es zwei Bedingungen, die das Wort zunächst erfüllen muss.:

1. Das Wort in der Liste muss denselben Anfangs- und Endbuchstaben haben
2. Die Länge der beiden Wörter muss gleich sein

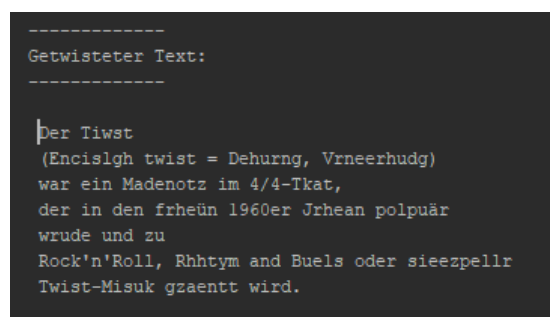
Dabei kann man diese Bedingungen auch als Abbruchbedingungen verstehen, denn sobald eine Bedingung nicht erfüllt wird, geht die `for`-Schleife zum nächsten Wort über. Wurde keine der beiden `if`-Verzweigungen, die als Abbruchbedingung fungieren, nicht betreten, so muss die dritte Bedingung erfüllt werden. Mit einer `if`-Verzweigung wird überprüft, ob die sortierte Zeichenfolge des zu twistenden Wortes der sortierten Zeichenfolge in der Wörterliste entspricht. Ist dies der Fall, so handelt es sich höchst wahrscheinlich, um das ursprüngliche Wort vor dem Twisten. Immerhin besteht bei manchen Wörtern die Möglichkeit, dass einem getwisteten Wort mehrere Wörter zugeordnet werden könnten. Wird kein passendes Wort gefunden, so wird das getwistete Wort wieder zurückgegeben.

An dieser Stelle wären wir auch bei den Grenzen des Programms. Zwar kann es Wortzusammensetzung wie „Test-Wort“ ohne Probleme enttwisten, jedoch stellen Komposita, die sich nicht in der Wortliste befinden, ein Problem dar. Genauso wie jedes andere Wort, dass nicht in der Liste vorhanden ist. Ansonsten kann man generell sagen, dass das Programm in sehr vielen Fällen einen Text enttwisten kann, was gleich an den Beispielen veranschaulicht wird. Weiterhin besteht die Möglichkeit zur Optimierung der Effizienz in der Suche eines passenden Wortes in der Wörterliste. Anstatt dass jedes Wort linear durchgegangen wird, könnte man mit einer binären Suche sicherlich bessere Performance-Zeiten erzielen, wobei die aktuellen Zeiten noch absolut befriedigend sind.

Beispiele

Twister

Beispiel 1) `twist1.txt` - Screenshot



```
-----  
Getwisteter Text:  
-----  
  
Der Tiwst  
(Encislgh twist = Dehurng, Vrneerhudg)  
war ein Madenotz im 4/4-Tkat,  
der in den frheün 1960er Jrhean polpuär  
wruide und zu  
Rock'n'Roll, Rhhtym and Buels oder sieezpellr  
Twist-Misuk gzaentt wird.
```

Wie man sieht, kann das Programm einwandfrei twisten. Zahlen und Sonderzeichen stellen kein Problem dar, da das Programm erkennt, welche Teile eines Wortes getwistet werden müssen.

Aufgabe 2: Twist

Beispiel 2) twist2.txt – Ausgabe

```
Hat der atle Heximseteenr sich doch eamnil wgeeegebbn! Und nun slolen senie
Geeitsr acuh ncah mienem Wileln leben.
Sneie Wrot und Wrkee mrket ich und den Bacruh, und mit Gstksäteirese tu ich
Wndeur auch.
```

Beispiel 3) twist3.txt – Ausgabe

```
Ein Ruseanatrt, wlcehes a la ctrae ateriebt, btieet sien Abngeot onhe enie
vhorer fleteegtse Mheüonegrnfeile an.
Dudacrh heabn die Gstäe zawr mher Sparlieum bei der Wahl iehrr Spiseen, für
das Raerantsut enhestetn jodceh zzcäuelshtir
Aufwnad, da wegnier Prcnsnhhgelseaiuit veodarnhn ist.
```

Beispiel 4) Eigenes Beispiel (Python Wikipedia Eintrag)

Eingabe

```
Python
(['paɪθən], ['paɪθən], auf Deutsch auch ['py:tən])
ist eine universelle,
üblicherweise interpretierte höhere Programmiersprache.
Sie hat den Anspruch, einen gut lesbaren,
knappen Programmierstil zu fördern.
So werden beispielsweise Blöcke nicht durch geschweifte Klammern,
sondern durch Einrückungen strukturiert.
Wegen seiner klaren und übersichtlichen
Syntax gilt Python als einfach zu erlernen.
```

Ausgabe

```
Pthoyñ
(['aɪθəñ], ['pɪaθən], auf Dcsueth acuh ['tpɔɪ:n])
ist eine unevrslilee,
üelheiciswrbe ieetertnitrpe hhröee Prpsiohcaemrmgrare.
Sie hat den Asrpcunh, einen gut lrseeban,
kaepnpñ Prtrsoaeriigmml zu föredrn.
So wrdeen bisiepsiseelwe Bcökle nchit dcurh giefchswtee Kmmralen,
snerdon dcurh Eunürceinkgn srtteukirrut.
Wegen sneier klearn und ühieirlescctbhn
Sytnax gilt Phyoñ als eianfch zu errlneen.
```

Enttwister

Beispiel 1) enttwist.txt – Ausgabe

```
-----
Enttwisteter Text:
-----

Der Twsit
(Englisch tiwst = Drehung, Verdrehung)
war ein Mdaotenz im 4/4-Takt,
der in den frühen 1960er Jahren populär
wurde und zu
```

Aufgabe 2: Twist

```
Rock'n'Roll, Ryhthm and Blues oder spezieller  
Twisit-Musik getanzt wird.
```

Wie man sehen kann, konnten Fremdwörter wie „Twist“ oder „Rhythm“ nicht in der Wortliste gefunden werden, weshalb diese Wörter getwistet geblieben sind. Zudem konnte das Kompositum „Modetanz“ ebenfalls nicht in der Liste gefunden werden. Ansonsten wurden die restlichen Wörter einwandfrei entwistet. Sonderzeichen sowie durch Bindestrich verbundene Wörter wurden erfolgreich enttwistet.

Beispiel 2) Eigenes Beispiel (Wikipedia Eintrag zu Python) – Ausgabe

```
-----  
Enttwisteter Text:  
-----
```

```
Pthoyn  
(['aɪθpɪ], ['pɪaθɔn], auf Deutsch auch ['tpɔɪ:n])  
ist eine universelle,  
üblicherweise interpretierte höhere Programmiersprache.  
Sie hat den Anspruch, einen gut lesbaren,  
knappen Programmierstil zu fördern.  
So werden beispielsweise Blöcke nicht durch geschweifte Klammern,  
sondern durch Einrückungen strukturiert.  
Wegen seiner klaren und übersichtlichen  
Syntax gilt Phyotn als einfach zu erlernen.
```

In diesem Beispiel konnte Python als einziges Wort nicht erkannt werden, genauso wie die griechischen Übersetzungen, da sie nicht in der Wörterliste vorhanden sind. Bis auf diese Ausnahmen, konnte das eigene Beispiel ohne weitere Fehler enttwistet werden.

Quellcode

Twister

```
@staticmethod
def wort_twisten(zu_twistendes_wort):
    # Falls das Wort drei oder weniger Buchstaben hat, muss es nicht mehr getwistet werden
    if len(zu_twistendes_wort) < 4:
        return zu_twistendes_wort

    # Falls kein Buchstabe gefunden wird, dient -1 als Platzhalter
    erster_buchstabe_pos = -1

    # Schleife geht solange durch jeden Buchstaben, bis an Position i ein Sonderzeichen vorhanden ist
    for i in range(len(zu_twistendes_wort)):
        # Sofern an Position i ein Buchstabe vorhanden ist...
        if zu_twistendes_wort[i].isalpha():
            erster_buchstabe_pos = i
            break

    # Sofern bei der vorherigen Schleife kein Buchstabe gefunden worden ist...
    if erster_buchstabe_pos == -1:
        # ...werden die Sonderzeichen oder Zahlen zurückgegeben
        return zu_twistendes_wort

    # Sonderzeichen oder Zahlen vor dem zu twistenden Wort, falls vorhanden
    praefix = zu_twistendes_wort[: (erster_buchstabe_pos + 1)]
    linke_grenze = (erster_buchstabe_pos + 1)

    # Zweite Schleife startet beim ersten Buchstaben und endet wenn an Position des Zählers j ein Sonderzeichen
    # vorhanden ist, oder wenn das letzte Zeichen erreicht worden ist
    for j in range(linke_grenze, len(zu_twistendes_wort)):
        # Wenn die zweite Schleife auf ein Sonderzeichen stoest
        if not zu_twistendes_wort[j].isalpha():
            # Die Position des letzten Buchstaben wird gespeichert (eine Position vor dem Sonderzeichen)
            rechte_grenze = j-1
            # Rekursiver Methodenaufruf, damit alles, was nach dem Sonderzeichen folgt, extra getwistet wird
            next_wort = Twister.wort_twisten(zu_twistendes_wort[j + 1:len(zu_twistendes_wort)])

            # Wenn nur maximal drei Buchstaben vorhanden sind...
            if (rechte_grenze - linke_grenze) < 2:
                # muss die aktuelle Buchstabenfolge nicht extra getwistet werden
                return praefix + zu_twistendes_wort[erster_buchstabe_pos+1:j+1] + next_wort

            # Letzter Buchstabe und das darauffolgende Sonderzeichen
            suffix = zu_twistendes_wort[j-1:j+1]
            # Buchstaben zwischen dem Anfangs- und Endbuchstaben werden zufällig angeordnet
            vertauschte_buchstaben = Twister.buchstaben_twisten(zu_twistendes_wort[linke_grenze:rechte_grenze])
            # Getwistetes Wort wird zurückgegeben
            return praefix + vertauschte_buchstaben + suffix + next_wort

    rechte_grenze = len(zu_twistendes_wort) - 1
    vertauschte_buchstaben = Twister.buchstaben_twisten(zu_twistendes_wort[linke_grenze:rechte_grenze])
    suffix = zu_twistendes_wort[len(zu_twistendes_wort)-1]
    return praefix + vertauschte_buchstaben + suffix
```

Die Methode `wort_twisten` gibt ein getwistetes Wort zurück, wobei in der Methode selbst zunächst die Buchstaben erkannt werden, die getwistet werden müssen.

Aufgabe 2: Twist

```
@staticmethod
def buchstaben_twisten(zu_twistende_buchstaben):
    # Die Buchstaben des zu twistendes Wortes in einer Liste
    buchstaben_liste = list(zu_twistende_buchstaben)
    # Zufälliges Sortieren der Items in der Buchstaben-Liste
    random.shuffle(buchstaben_liste)
    # Aus Buchstaben-Liste wieder einen String machen
    getwistete_buchstaben = ''.join(buchstaben_liste)
    return getwistete_buchstaben
```

In der Methode buchstabe_twisten findet der eigentliche Twist der Buchstaben statt.

```
with codecs.open('beispieldaten/Enttwister/' + optionen[int(benutzer_auswahl)], 'r', 'utf-8') as datei:
    # Alle Zeilen werden in einem Array gespeichert
    zeilen = datei.read().splitlines()
    # Schleife durchläuft Zeile für Zeile
    for zeile in zeilen:
        # Wörter einer Zeile werden in einem Array gespeichert
        woerter = zeile.split()
        # Ausgabe für die jeweilige Zeile wird initialisiert
        ausgabe = ""
        # Schleife durchläuft Wort für Wort
        for wort in woerter:
            # Wörter werden enttwistet und mit einem Leerzeichen getrennt
            ausgabe += " " + Enttwister.enttwiste_wort(wort)
        # Ausgabe einer enttwisteten Zeile
        print(ausgabe)
```

Teil des Programmcodes, dass die Datei einliest und alle Zeichenfolgen, die durch ein Leerzeichen getrennt sind, in einer Liste speichert.

Enttwister

```
# Ausgewählte Text-Datei wird geöffnet
with codecs.open('beispieldaten/Enttwister/' + optionen[int(benutzer_auswahl)], 'r', 'utf-8') as datei:
    # Alle Zeilen werden in einem Array gespeichert
    zeilen = datei.read().splitlines()
    # Schleife durchläuft Zeile für Zeile
    for zeile in zeilen:
        # Wörter einer Zeile werden in einem Array gespeichert
        woerter = zeile.split()
        # Ausgabe für die jeweilige Zeile wird initialisiert
        ausgabe = ""
        # Schleife durchläuft Wort für Wort
        for wort in woerter:
            # Wörter werden enttwistet und mit einem Leerzeichen getrennt
            ausgabe += " " + Enttwister.enttwiste_wort(wort)
        # Ausgabe einer enttwisteten Zeile
        print(ausgabe)
```

Teil des Programmcodes, dass die Datei einliest und alle Zeichenfolgen, die durch ein Leerzeichen getrennt sind, in einer Liste speichert.

Aufgabe 2: Twist

```
@staticmethod
def enttwiste_wort(zu_enttwistendes_wort):
    # Falls das zu enttwistende Wort drei oder weniger Buchstaben hat, muss es nicht mehr enttwistet werden
    if len(zu_enttwistendes_wort) < 4:
        return zu_enttwistendes_wort

    # Falls kein Buchstabe gefunden wird, dient -1 als Platzhalter
    erster_buchstabe_pos = -1

    # Schleife geht solange durch jeden Buchstaben, bis an Position i ein Sonderzeichen vorhanden ist
    for i in range(len(zu_enttwistendes_wort)):
        if zu_enttwistendes_wort[i].isalpha():
            erster_buchstabe_pos = i
            break

    # Sofern beim vorherigen Schleifenvorgang kein Buchstabe gefunden worden ist...
    if erster_buchstabe_pos == -1:
        # ... werden die Sonderzeichen oder Zahlen zurückgegeben
        return zu_enttwistendes_wort

    # Wenn das Wort nach dem Sonderzeichen maximal 3 Buchstaben hat...
    if len(zu_enttwistendes_wort) - erster_buchstabe_pos < 4:
        # ... wird die Zeichenkette auch zurückgegeben
        return zu_enttwistendes_wort

    # Sonderzeichen oder Zahlen vor dem zu enttwistenden Wort, falls vorhanden
    praefix = zu_enttwistendes_wort[:erster_buchstabe_pos]

    # Zweite Schleife startet beim ersten Buchstaben und endet wenn an Position des Zählers j ein Sonderzeichen
    # vorhanden ist, oder wenn das letzte Zeichen erreicht worden ist
    for j in range(erster_buchstabe_pos, len(zu_enttwistendes_wort)):

        # Wenn zweiter Zaehler auf Sonderzeichen stoesst
        if not zu_enttwistendes_wort[j].isalpha():

            # Rekursiver Methodenaufruf, damit alles, was nach dem Sonderzeichen folgt, extra enttwistet wird
            folgendes_wort = Enttwister.enttwiste_wort(zu_enttwistendes_wort[j + 1:len(zu_enttwistendes_wort)])

            # Falls es sich nur um drei Buchstaben handelt, muss nicht enttwistet werden
            if (j - erster_buchstabe_pos) < 3:
                return praefix + zu_enttwistendes_wort[erster_buchstabe_pos:j+1] + folgendes_wort

            # Sonderzeichen an Position j
            suffix = zu_enttwistendes_wort[j]

            # Das getwistete Wort ab dem ersten Buchstaben bis zum Buchstaben vor dem Sonderzeichen an Position j
            getwistetes_wort = zu_enttwistendes_wort[erster_buchstabe_pos:j]

            # Das getwistete Wort wird mithilfe der Methode wort_in_liste_finden enttwistet
            enttwistetes_wort = Enttwister.wort_in_liste_finden(getwistetes_wort)

            # Rückgabe des enttwisteten Wortes mit Sonderzeichen
            return praefix + enttwistetes_wort + suffix + folgendes_wort

    getwistetes_wort = zu_enttwistendes_wort[erster_buchstabe_pos:len(zu_enttwistendes_wort)]
    # Das getwistete Wort wird mithilfe der Methode wort_in_liste_finden enttwistet
    enttwistetes_wort = Enttwister.wort_in_liste_finden(getwistetes_wort)
    return praefix + enttwistetes_wort
```

Die Methode `wort_enttwisten` gibt ein enttwistetes Wort zurück, wobei in der Methode selbst zunächst die Buchstaben erkannt werden, die enttwistet werden müssen.

Aufgabe 2: Twist

```
@staticmethod
def wort_in_liste_finden(getwistetes_wort):
    # Text-Datei mit deutschen Wörtern wird geöffnet
    with codecs.open('beispieldaten/woerterliste.txt', 'r', 'utf-8') as sortierte_woerterliste:
        # Deutsche Wörterliste wird in einem Array gespeichert
        alle_deutschen_woerter = sortierte_woerterliste.read().splitlines()
        # Es wird Wort für Wort die Wörterliste durchgegangen
        for wort in alle_deutschen_woerter:
            # 1. Bedingung: Wort in der Liste muss denselben Anfangs- und Endbuchstaben
            # wie das zu enttwistende Wort haben
            # Die Groß- und Kleinschreibung wird erstmal vernachlässigt, da die Liste unvollständig ist
            if not (wort.lower().startswith(getwistetes_wort[0].lower()) and wort.endswith(getwistetes_wort[-1])):
                continue

            # 2. Bedingung: Länge der beiden Wörter muss gleich sein
            if not len(getwistetes_wort) == len(wort):
                continue

            # 3. Bedingung: Gleiche Buchstaben müssen in beiden Wörtern gleich häufig vorkommen
            # Alphabetisch sortierte Buchstaben in der Mitte des Wortes in der Liste
            sortierte_buchstaben_1 = ''.join(sorted(wort[1:-1]))
            # Alphabetisch sortierte Buchstaben in der Mitte des Wortes, das enttwistet werden soll
            sortierte_buchstaben_2 = ''.join(sorted(getwistetes_wort[1:-1]))
            # Wenn die selben Buchstaben in der selben Anzahl in beiden Wörtern vorkommen,
            # handelt es sich mit hoher Wahrscheinlichkeit um das enttwistete Wort
            if sortierte_buchstaben_1 == sortierte_buchstaben_2:
                # Es kann sein, dass ein anderes Wort dieselben Bedingungen erfüllt
                # An dieser Stelle wird also die erste Übereinstimmung zurückgegeben
                # Der Anfangsbuchstabe wird vom getwisteten Wort übernommen, da zuvor Groß- und Kleinschreibung
                # vernachlässigt wurde
                return getwistetes_wort[0] + wort[1:]

    # Wenn kein passendes Wort gefunden wurde
    return getwistetes_wort
```

Methode, die zu einem getwisteten Wort das jeweilige enttwistete Wort in der Wörterliste findet.