A List/Set/Dictionary Comprehension - Theorie

```
li = [1, 2, 3, 4, 5]
[ x*x for x in li ]  # --> [1, 4, 9, 16, 25]
```

Das kann man auch mit einer Bedingung kombinieren

```
li = [1, 2, 3, 4, 5]
[ x**3 for x in li if x > 3 ]  # --> [64, 125]
```

oder mehrere Schleifen

```
[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
#-> [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

# anders formatiert = lesbarer
[ (x, y)
  for x in [1,2,3]
  for y in [3,1,4]
  if x != y
]
```

Anmerkung: das Kombinieren von zwei oder mehr Listen gibt es fertig als **zip()** (denke an Reißverschluss).

```
 all( x >= y
      for x,y in zip(liste, liste[1:])
```

Comprehensions für Dictionaries:

```
 >>> { x:x*x for x in li } #--> {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Oder für ein Set:

```
>>> { x*x for x in li } #--> {16, 1, 4, 25, 9}
```

- Details siehe https://www.datacamp.com/tutorial/python-list-comprehension
- Überblicksvideo https://www.youtube.com/watch?v=lyDLAutA88s

Hinweis: aus so etwas

```
    res = []  # oder {} o.ä. -- leer
    for x in ....:
        res.add(...x...)
```

wird

```
    res = [ ...x...
            for x in ...
        ]
```

Wichtig: das ist *dubios*:

```
    res = []
    [ res.add(...x...)
      for x in ...
    ]
```

Hier wird mit den Ergebnissen von **add()** eine Liste erzeugt und diese Liste dann verworfen. Nur als *Nebeneffekt* wird es auch zur Liste hinzugefügt.

B Test auf empty():

Fast alle Python-Objekte können als boolean getestet werden dh. bei einem **if** direkt eingesetzt werden: entspricht einem Test auf ist nicht leer.

```
  if x:
    do

# nicht
  if len(x) > 0:
    do
```

C Aufgabe: Rechtschreibkorrektur

Datei: **spellcheck.py**

Finde zu einem möglicherweise falsch geschriebenen Wort die beste Korrektur (Vereinfachung: alles wird klein geschrieben).

Vorgehen: wir simulieren die möglichen Fehler: ein Buchstabe fehlt oder ist zu viel, sowie Buchstabendreher. Und zwar nicht für alle Wörter im Wörterbuch (die Liste bzw. ein Dictionary aller falsch geschrieben Wörter inkl. richtiger Schreibweise wäre viel zu groß), sondern umgekehrt: Wir untersuchen, ob Änderungen (=Rück gängigmachen des Fehlers) am gegebenen Wort (das der Benutzer möglicherweise mit Tippfehlern eingegeben hat) zu einem Eintrag im Wörterbuch führen. Wir beschränken die Suche auf maximal zwei Fehler.

Möglichst alle Unterprogramme sollten mit List- oder Set-Comprehension gelöst werden. Selbstverständlich brauchen alle Unterprogramme einen docstring und sollten mit Typehints definiert werden.

C.1 Unterprogramm python read_all_words(filename:str)-> Set[str]

Gegeben: eine Datei mit vielen (allen?) Wörtern. Speichere dieses Wörterbuch in einem Set, aber **nicht** in deinem git-Repository.

- https://wortschatz.uni-leipzig.de/de/download
- http://sourceforge.net/projects/germandict/
- Linux/Mac(?): /usr/share/dict/*

Hinweis: verwende zum Einlesen/Öffnen der Datei ein **with** (context manager) – damit braucht man kein close():

```
with open(...) as f:
   for line in f:
      ...
```

Ein set ist ähnlich einer Liste (aber ohne definierte Reihenfolge, ohne doppelte Einträge) bzw. einem Dictionary (aber nur die Keys, keine **Values**). Und man kann sets recht einfach kombinieren2: **&** | ^

C.2 Unterprogramm split_word(wort:str)-> List[Tuple[str, str]]

Bestimmt eine Liste aller Aufteilungen des Wortes. Die Listenelemente bestehen aus Tupel mit den Elementen head und tail.

```
split_word("abc") #--> [ ("", "abc"), ("a", "bc"), ("ab", "c"), ("abc", "") ]
```

Tipp: s[3:] bzw. s[:3]

C.3 Unterprogramm edit1(wort:str)-> Set[str]

Finde alle Wörter mit Edit-Distanz eins (= ein Tippfehler)

1. Bestimme mit dem Ergebnis von **split_word()** alle *möglichen* Wörter mit einem Fehler *weniger*, dh. wobei jeweils

a) ein Buchstabe fehlt
b) zwei Buchstaben verdreht sind
c) ein Buchstabe durch einen anderen Buchstaben ersetzt wurde
d) ein Buchstabe eingefügt wurde

Kombiniere dazu die zwei Wortteile (head und tail) von split_word() - der Fehler tritt an der Trennstelle bzw. zu Beginn des zweiten Wortteils (dem tail) auf. Beispiel:

```
("a", "bc")-->
 "ac" # ein Buchstabe fehlt
 "acb" # zwei Buchstaben verdreht
 "aac", "abc", "acc", "adc", ... # ein Buchstabe ersetzt
 "aabc", "abbc", "acbc", ... # ein Buchstabe eingefüg
```

Hier bietet sich `doctest` an– es gibt doch einige Sonderfälle, die man berücksichtigen sollte.

2. Hinweis: benutze List Comprehension (zur Not kann man auch `for`-Schleifen verwenden).

C.4 Unterprogrammedit1_good(wort:str, alle_woerter:List[str])-> Set[str]

ruft edit1(wort) auf und filtert, d.h. liefert aber nur die richtigen Wörter (aus dem Wörterbuch).

Tipps / Wiederholung aus der Theorie:

- Mengenkann mandirekt verknüpfen: Durchschnitt, Vereinigung etc. Dafür gibt es Methoden und überladene Operatoren (&, |, ^)

```
# return {x for x in edit1(word.lower()) if x in alle_woerter}
# besser:
return edit1(word.lower()) & alle_woerter
```

- Achtung: and bzw. or bei Mengen:

```
a = {1,2,3}; b={2,3,4}; c=set()
a or b: # liefert a (falls true) sonst b:--> {1,2,3}
a and b: # liefert b (falls a true ist) sonst set()--> {2,3,4}
c and b: #--> set()
```

C.5 Unterprogrammedit2_good(wort:str, alle_woerter:List[str])-> Set[str]

Bestimme Wörter mit Edit-Distanz zwei: edit1(wort1) für alle Möglichkeiten mit einem Fehler (= Ergebnis von edit1(wort)).

Man braucht hier nur richtige Wörter– gleich filtern spart viel Platz.

C.6 Unterprogramm correct(word:str, alle_woerter:List[str])-> Set[str]

Finde die Korrektur(en) für word als "Liste":

- entweder ist das Wort im Wörterbuch (Ergebnis: eine Liste mit einem Eintrag word)
- oder (mindestens) ein Wort mit Edit-Distanz eins ist im Wörterbuch (Ergebnis: Liste dieser Wörter)
- oder (mindestens) ein Wort mit Edit-Distanz zwei ist im Wörterbuch (Ergebnis: Liste dieser Wörter)
- oder wir haben keine Idee (zu viele Fehler oder unbekanntes Wort): liefere eine Liste mit dem ursprünglichen Wort

Das kann man in einer Zeile/mit einem Befehl machen– oder übersichtlicher mit if

Tipp:

- Listen/Sets/Strings/etc. haben eine Methode boolean()– bei einem if entspricht das dem Test auf nicht leer.

-Was liefert a or b zurück? (siehe oben)

C.7 Verbesserungen für Experten:

- Wenn die Wortliste ein normaler Text ist (z.B. ein Buch) kann man auch Worthäufigkeiten bestimmen. Die Liste der Korrekturen sollte dann entsprechend sortiert werden

C.8 Test

Ein paar docstrings und doctests runden das Projekt ab.

```
>>> woerter = read_file("de-en.txt")
>>> correct("Aalsuppe", woerter)
{'aalsuppe'}
>>> correct("Alsuppe", woerter)
{'aalsuppe'}
>>> sorted(correct("Alsupe", woerter))
['aalsuppe', 'absude', 'alse', 'lupe', 'staupe']
```