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)]
```

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}
```

—

B Test auf empty()

```
if x:
    do

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

—

C Aufgabe: Rechtschreibkorrektur

Datei: `spellcheck.py`

Finde die beste Korrektur für ein falsch geschriebenes Wort, wobei alle Eingaben kleingeschrieben sind.

Vorgehen:

- Mögliche Tippfehler simulieren: fehlender Buchstabe, zusätzlicher Buchstabe, Buchstabendreher
- Prüfen, ob eine Korrektur im Wörterbuch existiert
- Fehler auf maximal zwei Tippfehler beschränken

Alle Unterprogramme sollten List- oder Set-Comprehensions verwenden.

—

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

Einlesen einer Datei mit Wörtern als Set.

```
with open(filename) as f:
    return {line.strip() for line in f}
```

—

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

Erstellt eine Liste mit allen möglichen Wortaufteilungen.

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

—

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

Finde alle Wörter mit Edit-Distanz 1.

```
("a", "bc") -->
"ac"  # Ein Buchstabe fehlt
"acb"  # Zwei Buchstaben verdreht
"aac", "abc", "acc", "adc"  # Ein Buchstabe ersetzt
"aabc", "abbc", "acbc"  # Ein Buchstabe eingefügt
```

Rückgabe als `set`, um doppelte Einträge zu vermeiden.

—

C.4 `edit1_good(wort:str, alle_woerter:Set[str]) -> Set[str]`

Filtert die Ergebnisse von `edit1()` auf gültige Wörter im Wörterbuch:

```
return edit1(word.lower()) & alle_woerter
```

—

C.5 `edit2_good(wort:str, alle_woerter:Set[str]) -> Set[str]`

Findet Wörter mit Edit-Distanz 2, indem `edit1()` rekursiv aufgerufen wird.

—

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

Findet die beste Korrektur:

- Falls das Wort existiert, Rückgabe dieses Wortes
- Falls `edit1()` Treffer hat, diese zurückgeben
- Falls `edit2()` Treffer hat, diese zurückgeben
- Sonst: Wort unverändert zurückgeben

```
return ({word.lower()} & alle_woerter) or edit1_good(word, alle_woerter) or \
       edit2_good(word, alle_woerter) or {word}
```

—