Wir programmieren eine Rechtschreibkorrektur und üben dabei einige *pythoneske* Konzepte wie *Slicing*, *Comprehensions* und *Mengenoperationen*. Der Begriff *pythonesk* bezieht sich auf den absurden, surrealen, von schwarzem Humor und Gesellschaftskritik geprägten Stil der Comedy-Gruppe *Monty Python*, deren Name auch die Inspiration für die Programmiersprache war. In der Programmierwelt hat sich dieser Begriff weiterentwickelt und beschreibt den Stil und die Philosophie hinter gutem Python-Code.

A Theorie (Wiederholung)

A.1 List/Set/Dictionary Comprehension

```
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
]
```

Das *Kombinieren* von zwei oder mehr Listen gibt es fertig als `zip()` (denke an *Reißverschluss*).

```
list(zip(li, li[1:]))         # --> [(1, 2), (2, 3), (3, 4), (4, 5)]
```

Mit Comprehensions kann man auch Dictionaries

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

oder ein Set erzeugen.

```
>>> { 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: Statt

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

schreibt man *pythonesk*:

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

A.2 Python-Datenstrukturen als boolean:

Alle Python-Typen können *als boolean getestet* werden:

Datenstruktur	`False`	`True`
`bool`	`False`	`True`
`int` / `float` / `complex`	`0` / `0.0` / `0j`	Jede andere Zahl
`str` / `bytes`	`""` / `b""`	Jeder nicht-leere String
`list` / `tuple` / `dict` / `set`	`[]` / `()` / `{}` / `set()`	Jeder nicht-leere Container
`None`	`None`	*gibt es nicht*

Daher kann man bei einem `if` direkt eine Variable verwenden. Das entspricht bei den meisten Datentypen sinngemäß einem Test auf *ist nicht leer*:

```
if x:                          # pythonesk = guter Programmierstil
  do

if len(x) > 0:                 # nicht pytonesk, weil schlechter lesbar und häufig langsamer
  do
```

B Aufgabe: Rechtschreibkorrektur

Datei: `spellcheck.py`

Finde zu einem möglicherweise falsch geschriebenen Wort die beste Korrektur. Wir nehmen an, dass alles klein geschrieben wird.

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 (=Rückgängigmachen 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[1].

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

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

Gegeben: eine Datei mit vielen (allen?) Wörtern. Speichere die Datei **nicht** in deinem `git`-Repository sonder lies sie lokal in einem `set` ein.

- Datei `de-en.zip` im Anhang oder alternativ von hier herunterladen.
- 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 `set`s recht einfach kombinieren[2]: `& | ^`

[1] etwa wie in https://de.wikipedia.org/wiki/Levenshtein-Distanz beschrieben
[2] siehe Theoriestunde und https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset

B.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]`

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

Findet 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[3]

   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"                          # a) ein Buchstabe fehlt
       "acb"                         # b) zwei Buchstaben verdreht
       "aac", "acc", "adc", "aec",... # c) ein Buchstabe ersetzt
       "aabc", "abbc", "acbc", ... )  # d) ein Buchstabe eingefügt
   ```

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

   Hinweis: benutze *List Comprehension* – nur zur Not sollte man `for`-Schleifen verwenden.

2) liefere alle Möglichkeiten als `set` zurück – damit gibt es keine doppelten Einträge.

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

Ruft `edit1(wort)` auf und filtert, d.h. liefert nur die *richtigen* Wörter (aus dem Wörterbuch). Beispiel:

```
>>> sorted(edit1_good("Pyton", read_file("de-en.txt"))
['pylon', 'python']
```

Tipps / Wiederholung aus der Theorie:

- Auch wenn sich hier eine Comprehension anbietet – es ist eigentlich eine *Mengenoperation*, und die sind viel schneller:
- Mengen kann man direkt verknüpfen: Durchschnitt, Vereinigung etc. Dafür gibt es Methoden und überladene Operatoren (`&`, `|`, `^`), also:

  ```
  return edit1(word.lower()) & alle_woerter # Schnittmenge
  # oder (aber nicht so übersichtlich):
  return {x for x in edit1(word.lower()) if x in alle_woerter}
  ```

- Achtung: `and` bzw. `or` bei Mengen haben eine andere Bedeutung:

  ```
  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: # liefert c (weil false) --> set()
  ```

[3] `string.ascii_lowercase` oder besser: alle in der Wortliste vorkommenden Zeichen

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

Bestimmt 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.

B.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 leere Liste

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

- Tipp: Was liefert (bei Python-sets) `a or b` zurück und was `a | b`? (siehe oben)

B.7 Test

Ein paar `docstrings` und `doctests` runden das Projekt ab.

```
>>> woerter = read_file("de-en.txt")
>>> sorted(correct("Aalsuppe", woerter))
['aalquappe', 'aalsuppe', 'aalsuppen']
>>> sorted(correct("Alsuppe", woerter))
['aalsuppe', 'aalsuppen', 'suppe', 'ursuppe']
>>> sorted(correct("Alsupe", woerter))
['aalsuppe', 'absude', 'alse', 'lupe']
```

B.8 Bonus: Verbesserungen für Experten

- Man kann mit einem umfangreichen Buch die Worthäufigkeiten bestimmen. Die Liste der Korrekturvorschläge sollte nach Worthäufigkeit sortiert werden.