Aufgabenblatt 6

Aufgabe 20: checkSolution()

Schreiben Sie eine Funktion `checkSolution(problem, solution) -> bool`, die zwei Strings `problem` und `solution` gegeneinander abgleicht. Dabei sei `problem` eine Art Lückentext, der durch Ersetzen bestimmter Buchstaben durch einen _ entstanden ist. `solution` sei eine potentielle Lösung für den Lückentext.

Die Funktion soll überprüfen, ob es sich dabei um eine gültige Lösung handelt. Eine Lösung sei gültig, wenn `solution` aus `problem` erzeugt werden kann, indem nur an den mit _ markierten Stellen Ersetzungen vorgenommen werden. Alle Zeichen an anderen Positionen im String müssen übereinstimmen; auch dürfen die Strings nicht in ihrer Länge abweichen.

Beispiel:

```
>>> checkSolution('St__l___en', 'Stallhosen')
True
>>> checkSolution('St__l___en', 'Stuhlboden')
True
>>> checkSolution('St__l___en', 'Stilleben')
False
```

Aufgabe 21: Kryptografie

Ein einfacher, seit dem Altertum verwendeter Verschlüsselungsalgorithmus ist das Ersetzen jedes Zeichens durch ein anderes. Dies geschieht mit einer Ersetzungstabelle, die beiden Kommunikationspartnern bekannt sein muss. Es handelt sich also um symmetrische Verschlüsselung.

Eine Spielart dieser Methode ist der ALBaM-Algoritmus. Dabei wird (im lateinischen Alphabet) jeder Buchstabe durch seinen 13. Nachfolger ersetzt. Am Ende des Alphabets wird wieder vorne angefangen. Also: Aus „a“ wird „n“, aus „b“ wird „o“, aus „v“ wird „i“.

Schreiben Sie ein Modul, das eine Funktion enthält, die zunächst nur für Kleinbuchstaben diese Verschlüsselung anwendet. Sie können dazu die eingebauten Python-Funktionen `ord(char) -> num` und `chr(num) -> char` verwenden. Diese geben für einen String der Länge 1 die laufende Nummer in der Ascii Tabelle zurück und umgekehrt.

Das Ergebnis sollte so aussehen:

```
>>> import krypto
>>> krypto.albam('hallo')
'unyyb'
>>> krypto.albam('unyyb')
```



```
'hallo'
```

Diskutieren Sie Limitierungen, Schwächen und mögliche Verbesserungen Ihres Algorithmus. Was können Sie tun, um eine verschlüsselte Nachricht zu knacken, wenn Sie nicht wissen, um wie viel Stellen die Buchstaben verschoben wurden?

Zusatzaufgabe: Erweitern Sie Ihre Funktion so, dass sie auch mit Großbuchstaben funktioniert.

Aufgabe 22: n-gramme

In dieser Aufgabe beschäftigen Sie sich mit n-grammen, die Sie bereits in der ECL-Vorlesung kennengelernt haben. Schreiben Sie ein Modul `ngrams`, das folgende Funktion enthält:

`makeNGrams(filename, n) -> list`

Diese Funktion soll den Inhalt einer Textdatei in n-gramme zerlegen.

Beispiel (und – wie alle anderen Beispiele – doctest!):

```
>>> import ngrams
>>> ngrams.makeNGrams("ngrams.txt", 2)
[['Das', 'ist'], ['ist', 'eine'], ['eine', 'kleine'], ['kleine', 'Datei'],
['Datei', 'zum'], ['zum', 'Testen']]
```

Die Argumente dieser Funktion sind der Dateiname und ein numerischer Wert, der n angibt. Im Beispiel oben war $n = 2$, wir haben also Bigramme erzeugt. Analog für Trigramme:

```
>>> ngrams.makeNGrams("ngrams.txt", 3)
[['Das', 'ist', 'eine'], ['ist', 'eine', 'kleine'], ['eine', 'kleine', 'Datei'],
['kleine', 'Datei', 'zum'], ['Datei', 'zum', 'Testen']]
```

Hinweise:

- Machen Sie sich nochmals mit *Slices* vertraut. Sie sind für diese Aufgabe äußerst hilfreich.
- Um den Dateiinhalt zu tokenisieren, können Sie weiterhin die `split`-Methode in ihrer simpelsten Form verwenden.