Aufgabenblatt 1 – 24.04.2025

Programmieren II: Fortgeschrittene (Python) Abgabe bis 01.05.2025

Aufgabe 1) Indexing **Punkte: 2**

- Implementiere die Funktion `slice_string` so, dass du *ausschließlich mithilfe von Slicing* den Text `nemgP` erhältst, wenn du der Funktion `Programmieren` übergibst.

  (1 Punkt)

- Erkläre, was passiert, wenn du folgendes Programm ausführst:

```
s = "Programmieren"
print(s[len(s)])
```

  (0.5 Punkte)

- Erkläre, was passiert, wenn du das unten beschriebene Programm ausführst. Was ist vielleicht irritierend?

```
s = "Programmieren\n2"
s_raw = r"Programmieren\n2"
print(len(s))
print(len(s_raw))
```

  (0.5 Punkte)

Aufgabe 2) Korpusanalyse **Punkte: 5**

Analysiere die Datei `grail.txt` hinsichtlich der vorkommenden Wörter, ohne dabei andere Bibliotheken zu importieren. Ein Wort ist in unserem Fall eine Zeichenfolge, die durch Leerzeichen von anderen Zeichenfolgen getrennt ist.

- Lies die Datei `grail.txt` in `read_corpus` ein. *(0.5 Punkte)*

- Berechne die Type-Token-Relation in `compute_ttr`. Die Type-Token-Relation gibt das Verhältnis der Anzahl unterschiedlicher Wörter (Types) zur Gesamtzahl der Wörter (Tokens) in einem Text an.[1] *(1 Punkt)*

- Erstelle in `get_unigrams` ein Dictionary, das für jedes Wort zählt, wie häufig es

[1] *Tokens* sind die Gesamtzahl aller Wörter in einem Text, wobei jedes Vorkommen eines Wortes gezählt wird. *Types* sind die Anzahl der unterschiedlichen Wörter in einem Text. Jedes Wort wird dabei nur einmal gezählt, egal wie oft es vorkommt.

im Korpus vorkommt. *(1.5 Punkte)*

- Invertiere das Dictionary in `invert_dict`, sodass der *Schlüssel* nun die Häufigkeit ist, der *Wert* das dazugehörige Wort. Welches Problem gibt es? Wie lässt sich dieses für unseren Anwendungsfall lösen? *(1 Punkt)*

- Erstelle eine Klasse `Type` mit den Attributen `type` und `freq`. `Type` soll dieselben Informationen speichern wie ein Key-Value-Paar aus `get_unigrams`. Konvertiere das Dictionary aus `get_unigrams` in eine Liste aus Type-Instanzen in einer Funktion `dict_to_type_list`. *(1 Punkt)*

Aufgabe 3) Matrizen **Punkte: 3**

Eine Matrix kann als verschachtelte Liste dargestellt werden. Beispielsweise kann man die Matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ folgendermaßen in Python repräsentieren:

```
a = [[1, 2, 3], [4, 5, 6]]
```

- Schreibe eine Funktion `scale_matrix`, die ein Skalar und eine beliebige Matrix nimmt und jeden Wert in der Matrix mit dem Skalar multipliziert, z. B.:

$$5 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 10 & 15 \\ 20 & 25 & 30 \end{bmatrix}$$

(1 Punkt)

- Schreibe eine Funktion `sum_matrices`, die zwei beliebige Matrizen derselben Form addiert und das Ergebnis zurückgibt. *(1 Punkt)*

- Die transponierte Matrix $A^\top = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$ entsteht, wenn man bei der Matrix A Zeilen und Spalten vertauscht. Schreibe eine Funktion, die eine beliebige Matrix nimmt und die zugehörige transponierte Matrix zurückgibt. *(1 Punkt)*

Abgabemodalitäten
Reiche die Lösung pünktlich bis 13:15 am genannten Abgabetermin auf Moodle ein.
Valide Abgaben sind Dateien, die gemäß gesundem Menschenverstand eine sinnvolle Möglichkeit sind, den Tutor:innen eine Lösung zu präsentieren (wenn Code abzugeben ist: .py-Datei, .txt-Datei: oft sinnvoll, .pdf-Datei: oft sinnvoll, .txt-Datei auf Arabisch + verschlüsselt: nicht sinnvoll).
Die Abgabe muss in Zweiergruppen erfolgen. Solltest du noch keinen Gruppenpartner haben dann schreibe in die Partnerbörse auf moodle.

Füge zur Abgabe ein Dokument "Gruppenmitglieder.txt" hinzu, das die Vor- und Nachnamen aller Gruppenmitgliedern auflistet.
Bei offensichtlichem Abschreiben von anderen Gruppen werden alle beteiligten Abgaben mit 0 Punkten bewertet.