Aufgabe 1

Lösungsidee

Bei dieser Aufgabe hatte ich die Idee, reguläre Ausdrücke zu verwenden, um Regeln zum Finden der unvollständigen Zitate zu erstellen. Das ist aufgrund der Struktur der gegebenen Zitate hier vorteilhaft, da reguläre Ausdrücke ausgezeichnet mit Lückentexten klarkommen.

Umsetzung

Aus Gründen der Einheitlichkeit mit den anderen Aufgaben habe ich für diese Aufgabe Python gewählt und die Lösung objektorientiert gestaltet. Da der Großteil der tatsächlichen Lösung von der eingebauten `re`-Bibliothek übernommen wird, wäre man auch ohne Klassen und Objekte ausgekommen, ohne wirklich an Übersichtlichkeit, Lesbarkeit und Nachvollziehbarkeit zu verlieren. Meine Lösung enthält die Klasse `Buch` und die "main-Methode", oder zumindest das pythonische Pendant. `Buch` umfasst die Attribute `book_text` und `pattern`, welche den Buchtext bzw. das unvollständige Zitat, umgeformt zu einem regulären Ausdruck, speichern. Um letzteres zu erreichen, ist `pattern` eigentlich eine Methode, die mit dem Dekorator `@property` versehen ist, wodurch `pattern` wie ein normales Attribut aufgerufen werden kann. Die setter-Methode jedoch verarbeitet den überreichten Lückentext so, dass die "verschwundenen Wörter", welche durch einen Unterstrich dargestellt werden, durch `\w+` ersetzt werden. In regulären Ausdrücken bedeutet das Meta-Zeichen `\w` so viel wie "Wort-Zeichen", also Zeichen, die typischerweise in Worten natürlicher Sprache vorkommen. Diese sind alle Groß- und Kleinbuchstaben, alle Ziffern und Unterstriche, oder als Ausdruck: `[a-zA-Z0-9_]`. Dadurch, dass dahinter noch das Quantifizierungszeichen `+` steht, wird nicht nur genau ein Wortzeichen, sondern eins oder mehr für diese Lücke angenommen. Da Wörter eines Buches selten genau ein Zeichen umfassen, ist das notwendig. Nachdem das übergebene Zitat nun in ein gültiges und sinnvolles Regex-Pattern überführt wurde, kann der Buchtext nach (allen) passenden Stellen abgesucht werden, indem die `findall`-Methode der `re`-Bibliothek mit dem erstellten regulären Ausdruck und

dem Buchtext als Parameter aufgerufen wird. Diese gibt alle vervollständigten Zitate zurück, die auf das gesuchte Muster passen.

Beispiele

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung0.txt
Die passenden Stellen sind:

1. Das kommt mir gar nicht richtig vor
```

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung1.txt
Die passenden Stellen sind:

1. Ich muß in Clara verwandelt
2. Ich muß doch Clara sein
```

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung2.txt
Die passenden Stellen sind:

1. Fressen Katzen gern Spatzen
2. Fressen Spatzen gern Katzen
```

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung3.txt
Die passenden Stellen sind:

1. das Spiel fing an
2. Das Publikum fing an
```

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung4.txt
Die passenden Stellen sind:

1. ein sehr schöner Tag
```

```
$ python aufgabe1.py Alice_im_Wunderland.txt stoerung5.txt
Die passenden Stellen sind:

1. Wollen Sie so gut sein
```

Quelltext

```
from re import Pattern, compile, findall, IGNORECASE, MULTILINE

class Book:
    book_text: str
    _pattern: Pattern

    def find_passage(self) -> list[str]:
        matches = findall(self.pattern, self.book_text)
        return matches

    @property
    def pattern(self) -> Pattern:
        return self._pattern

    @pattern.setter
    def pattern(self, new_pattern: str):
        self._pattern = compile(new_pattern.replace("_", r"\w+"),
IGNORECASE | MULTILINE)
```