Danny Rehl, Programmieren I - Python

Übungsblatt 07

Abgabefrist: Montag, den 09.12.2024, um 23:59 Uhr

Sie finden für dieses Übungsblatt eine Vorlage für Ihre Abgabe zum Herunterladen im Moodle.

Für die Bewertung Ihrer Abgabe zählen die von Ihnen vervollständigten Funktionen im oberen Teil der Vorlage. Schreiben Sie hier nur Ihre Lösungen hin, keine weiteren Befehle außerhalb der Funktionen.

Im unteren Teil finden Sie einen `if __name__ == "__main__"`-Abschnitt, welcher bereits einige Beispielaufrufe zu den Funktionen beinhaltet. Hier können Sie die Befehle beliebig ergänzen oder ersetzen, um Ihre Funktionen zu testen.

Hinweis: Trotz aller Tests muss die hochgeladene `.py`-Datei natürlich immer noch ausführbar bleiben. Beispielsweise wird ein `SyntaxError` im `if __name__ == "__main__"`-Teil in jedem Fall zur Pytest-Bewertung von 0 % führen.

Bitte achten Sie darauf, dass Ihr Code PEP 8 konform geschrieben ist. Das ist Voraussetzung für die abschließende Bewertung. Nutzen Sie die Hinweise des Pylint-Scorers, um die verbleibenden Style-Fehler zu finden.

Aufgabe 1 (2 Punkte)

Es gibt in `re`, dem Python-Modul für Reguläre Ausdrücke, mehrere Funktionen zum Suchen in Strings. Drei davon sind `re.findall`, `re.match` und `re.search`. Diese verhalten sich alle leicht unterschiedlich.

In der Vorlage finden Sie eine Funktion `complicated_search`, welche zwei Parameter hat: `pattern`, `string` und `mode`. Dabei bestimmt `mode` die Funktion, die benutzt wird.

Schauen Sie sich die folgenden Befehle der Python-Shell an:

```
>>> complicated_search(r"Python", text, 1)
'Python'
>>> complicated_search(r"Python", text, 2)
['Python']
>>> complicated_search(r"Python", text, 3)
'Python'
>>> complicated_search(r"Programmiersprache", text, 1)
>>> complicated_search(r"Programmiersprache", text, 2)
['Programmiersprache']
>>> complicated_search(r"Programmiersprache", text, 3)
'Programmiersprache'
>>> complicated_search(r"\w*mm\w+", text, 1)
>>> complicated_search(r"\w*mm\w+", text, 2)
['Programmiersprache', 'Programmierstil', 'Klammern']
>>> complicated_search(r"\w*mm\w+", text, 3)
'Programmiersprache'
```

Ohne zu wissen, wie der Textinhalt der Variablen `text` genau aussieht, können Sie bestimmen, welche der drei Funktionen `re.findall`, `re.match` und `re.search` zu den Modi 1, 2 und 3 gehören. Passen Sie die Variable `functions` in `complicated_search` entsprechend an. (Sie müssen nichts anderes an der Funktion ändern.)

Zur Erinnerung: Gibt eine Funktion `None` zurück, dann erhalten Sie in der Python-Shell keine Ausgabe - wie bei einigen Zeilen oben zu sehen.

Aufgabe 2 (1 Punkt)

`check_vowels` überprüft, ob ein als Parameter gegebener String Vokale enthält. Die Funktion sollte den Wert `True` zurückgeben, wenn der gegebene String mindestens einen der folgenden fünf Charaktere enthält: `a`, `o`, `e`, `i` oder `u`. Wir gehen dabei davon aus, dass der String nur aus Kleinbuchstaben besteht. Enthält der gegebene String keinen der genannten Charaktere, soll `check_vowels` den Wert `False` zurückgeben.

Benutzen Sie die Methode `re.search`, um diese Aufgabe zu lösen.

Beispiel:

```
>>> check_vowels("hello")
True
>>> check_vowels("xyz")
False
```

Aufgabe 3 (1 Punkt)

`count_adverbs` zählt in einem gegebenen String Wörter, die mit dem Suffix `-ly` enden, wie zum Beispiel “strongly” oder “greatly”, aber nicht “flying”. Die Funktion sollte die Anzahl dieser Wörter als Integer zurückgegeben. Ein Wort bezeichnen wir hier als eine Zeichenreihe, die zwischen zwei Leerzeichen oder Satzzeichen steht.

Benutzen Sie die Methode `re.findall`, um diese Aufgabe zu lösen.

Beispiel:

```
>>> count_adverbs("He was carefully disguised but captured quickly by police")
2
```

Aufgabe 4 (1 Punkt)

`remove_numbers` ersetzt in einem gegebenen Text jegliche Vorkommen von Zahlen und Prozenten durch den Platzhalter `"[DD]"`. Ergänzen Sie die Funktion so, dass sie eine Version des Textes zurückgibt, in der alle Zahlen und Prozentangaben (inklusive "%") durch den Platzhalter ersetzt wurden. Schauen Sie sich hierfür auch die verschiedenen Zahlen in `CORPUS_TEXT` an: Zahlen größer als 1000 enthalten ein Komma und einige Dezimalzahlen enthalten einen Punkt vor den Nachkommastellen. Überlegen Sie sich einen regulären Ausdruck, der diese unterschiedlichen Formate erkennt.

Benutzen Sie die Methode `re.sub`, um diese Aufgabe zu lösen.

Beispiel:

```
>>> remove_numbers("The S&P climbed 1.4% as the tech sector outperformed.")
The S&P climbed [DD] as the tech sector outperformed.
>>> remove_numbers("The stock index broke above 26,000 for the first time")
The stock index broke above [DD] for the first time
>>> remove_numbers("Market expectations rose to 27.5% from 16.7%.")
Market expectations rose to [DD] from [DD].
```

Aufgabe 5 (2 Punkte)

`begin_x` gibt alle Zeilen zurück, die mit einem "x" beginnen.

`begin_end_x` gibt alle Zeilen zurück, die mit einem "x" beginnen und/oder mit einem "x" enden. (Also ein inklusives "oder".)

Benutzen Sie die Methode `re.findall`, um diese Aufgabe zu lösen.

Hinweis: Sie werden für diese Aufgabe "regex flags" benutzen müssen, zum Beispiel über das `flags` Argument. Es kann sich also lohnen, die Dokumentation zu `re` nochmal anzuschauen.

Beispiel:

```
>>> EXAMPLE_LINES = """
... abc
... xyz
... 123
... x456x
... 789x
... """
>>> begin_x(EXAMPLE_LINES)
['xyz', 'x456x']
>>> begin_end_x(EXAMPLE_LINES)
['xyz', 'x456x', '789x']
>>>
```

Aufgabe 6 (3 Punkte)

Sie möchten mit der Funktion `logging_examples` das `logging` Modul von Python näher kennenlernen. Konfigurieren Sie dafür mit einem Aufruf von `logging.basicConfig` das Logging-Basissystem. Überlegen Sie sich dabei sinnvolle Werte für die möglichen Parameter. Sie finden Beispiele für den Aufruf auf den Vorlesungsfolien oder auch in der Python-Dokumentation zum `logging` Modul. Anschließend erstellen Sie einen neuen Logger mit dem Namen `Beispiel`.

An diesen Logger sollen Sie nun in der angegebenen Reihenfolge die folgenden Mitteilungen senden:

- Mit Loglevel `20` folgende Nachricht: `Es geht los`
- Mit Loglevel `10` folgende Nachricht: `Vielleicht ein Problem`
- Mit Loglevel `ERROR` folgende Nachricht: `Leider ein Fehler`

Die Definitionen der einzelnen Loglevels sind ebenfalls in der offiziellen Python-Dokumentation zum `logging` Modul zu finden. Sie können frei wählen, ob Sie die passende Methode zum Loglevel Ihrer `logging.Logger` Instanz raussuchen oder die allgemeine Methode `logging.Logger.log` benutzen und das Loglevel als Parameter übergeben.
