Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Tim Schulte, Christoph-Simon Senjak
Wintersemester 2018/2019

Universität Freiburg
Institut für Informatik

Übungsblatt 8

Abgabe: Dienstag, 11.12.2018, 20:00 Uhr

Aufgabe 8.1 (Testen; Dateien: `words.py`, `test.txt`; Punkte: 3+4+1)

Unter einem Wort verstehen wir im Folgenden jede endliche Folge von Buchstaben des deutschen Alphabets der Länge ≥ 1. Je zwei Wörter in einer Zeichenfolge werden durch eine nicht-leere, endliche Folge von Zeichen, die nicht zu diesen Buchstaben gehören getrennt. Die Funktion `next_word(s)` soll, angewendet auf einen String `s`, ein Tupel `(word, rest)` zurückgeben, wobei `word` das erste Wort in `s` ist und `rest` die Zeichenfolge ist, die in `s` auf `word` folgt. Falls `s` kein Wort enthält, gibt die Funktion das Tupel `(None, "")` zurück. Die folgenden Definitionen enthalten einen syntaktischen Fehler, einen semantischen Fehler und einen Fehler, der bei vielen (auch kurzen) Eingaben zur Laufzeit auftritt[1]. Fehler sind also in genau 3 Zeilen zu finden.

```
LETTERS = ("abcdefghijklmnopqrstuvwxyz"
           "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜäöüß")

def _next_word_helper(s: str) -> tuple:
    """Helper function for next_word."""
    if not s:
        return None, s
    if s[0] not in LETTERS:
        return None, s
    word = s[0]
    word_rest, s_rest = _next_word_helper(s[1:])
    if word_rest:
        word = word_rest
    return word, s_rest

def next_word(s: str) -> tuple:
    """Return the first word of an input string s and the rest of it."""
    while s[0] not in LETTERS
        s = s[1:]
    return _next_word_helper(s)
```

(a) Finden und korrigieren Sie die Fehler! Laden Sie hierzu das Template `words.py` von der Website oder kopieren Sie den Code in die angegebene Datei. Kommentieren Sie die fehlerhaften Zeilen aus und fügen Sie eine Korrektur der jeweils fehlerhaften Zeile nach dem Kommentarblock ein. Ergänzen Sie jeweils auch einen Kommentar, der den Fehler benennt und erläutert, warum er auftritt. Überarbeiten Sie außerdem die Docstrings, sodass diese unseren Richtlinien entsprechen (beschreiben Sie die Argumente und Rückgabewerte).

[1] Bei Eingaben mit sehr langen Wörtern gibt es einen weiteren Laufzeitfehler, der hier aber nicht behandelt werden soll.

(b) Implementieren Sie eine Funktion `test_next_word()`, welche mindestens vier Testfälle (`assert`-Statements) enthält. Die Testfälle sollten im nicht-korrigierten Code zu semantischen oder Laufzeitfehlern führen (pro Fehler zwei unterschiedliche, sinvolle Tests).

(c) Führen Sie in der Shell bzw. Eingabeaufforderung das Kommando

`pytest -v words.py` (bzw. `python -m pytest -v words.py`)

zum Durchführen der Tests aus. Kopieren Sie die Shell-Ausgabe in eine Datei `test.txt` und committen Sie auch diese Datei zum SVN-Repository.

Aufgabe 8.2 (Liste glätten; Datei: `flatten.py`; Punkte: 6+4)

Wir nennen eine Liste *verschachtelt*, wenn sie mindestens eine weitere Liste als Element enthält. Die geglättete Version einer (verschachtelten oder nicht-verschachtelten) Liste ergibt sich wie folgt: jedes Element L, das eine Liste ist, wird (unter Beibehaltung der Reihenfolge) durch die Elemente der geglätteten Version von L ersetzt; jedes andere Element wird (unter Beibehaltung der Reihenfolge) übernommen. *Hinweis:* Bei der Bearbeitung der folgenden Aufgaben sollen nur Hilfsmittel verwendet werden, die bisher in der Vorlesung eingeführt wurden. Lösungen, die zusätzliche Module importieren, werden nicht bewertet.

(a) Definieren Sie eine rekursive Funktion `flatten(ls: list) -> list`, die die geglättete Version der übergebenen Liste `ls` berechnet und zurückgibt. Ihre Funktion soll dabei eine komplett neue Liste zurückgeben und darf die übergebene Liste bzw. alle betrachteten Teillisten nicht verändern. Starten Sie mit dem Gerüst für Funktionen, die Listen verarbeiten, und überlegen Sie dann, wann ein Listenelement geglättet werden muss.

```
>>> a = [3, 4, [[5]]]
>>> b = [[[1, 2, a], (6, [7]), 8], 9, False]
>>> c = flatten(b)
>>> a == [3, 4, [[5]]]
True
>>> b == [[[1, 2, [3, 4, [[5]]]], (6, [7]), 8], 9, False]
True
>>> c == [1, 2, 3, 4, 5, (6, [7]), 8, 9, False]
True
```

(b) Implementieren Sie eine Funktion `test_flatten()`, welche mindestens vier eigene, sinnvolle und voneinander verschiedene Testfälle (`assert`-Statements) enthält.

Aufgabe 8.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet08` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).