Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel
Dr. Stefan Wölfl, Thorsten Engesser
Wintersemester 2015/2016

Universität Freiburg
Institut für Informatik

Übungsblatt 5

Abgabe: Freitag, 27. November 2015, 20:00 Uhr

Wichtige Hinweise: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet05` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann in Dateien in diesem Unterverzeichnis erwartet.

Beachten Sie bitte bei allen Aufgaben die *Hinweise zur Bearbeitung der Übungsaufgaben* unter der folgenden URL:

`http://gki.informatik.uni-freiburg.de/teaching/ws1516/info1/wiki/hinweise.html`

Überprüfen Sie, dass Sie alle Lösungen ins Repository hochgeladen haben (z.B. mit dem Befehl `svn status`). Überprüfen Sie auch die Webseite Ihres SVN-Unterverzeichnisses:

`https://daphne.informatik.uni-freiburg.de/ws1516/InformatikI/svn/$RZLOGIN`

Bewertet wird bei allen Aufgaben die letzte Version, die zur Deadline des Übungsblattes auf dem SVN-Server eingereicht ist.

Erläuterungen zu den Aufgaben. In den folgenden Aufgaben geht es darum, eine Zeichenfolge (einen String) einzulesen und dabei eine Datenstruktur anzulegen, die es später erlaubt für ein gegebenes Wort zu entscheiden, ob und wie oft dieses Wort in der Zeichenkette vorkommt. Unter einem *Wort* verstehen wir im Folgenden jede endliche Folge von Buchstaben des deutschen Alphabets (also den Zeichen a, b, c, ..., z, A, B,..., Z, ä, Ä, ö, Ö, ü, Ü, ß) der Länge ≥ 1. Je zwei Wörter in der Zeichenfolge werden durch eine nicht-leere, endliche Folge von Zeichen, die nicht zu diesen Buchstaben gehören (z.B., Leerzeichen, Satzzeichen, Zeilenumbrüche), getrennt. Somit sind echte Teilfolgen eines Wortes im Folgenden nicht als Wörter zu zählen.

Aufgabe 5.1 (Nächstes Wort; Dateien: `words.py`, `doctest_words.txt`; Punkte: 3+2+1)

Im folgenden Python-Code wird eine Funktion `next_word(s)` definiert, die angewendet auf einen String `s` ein Tupel `(word, rest)` zurückgibt, wobei `word` das erste Wort (im Sinne der Erläuterung) 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 Definitionen enthalten einen syntaktischen, 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.

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

```
LETTERS = "abcdefghijklmnopqrstuvwxyz" + \
          "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + \
          "ÄÖÜäöüß"

def _next_word_helper(s):
    """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):
    """Return the first word of an input string s and the rest of it."""
    # eat leading punctuation marks, white spaces, etc.
    while s[0] not in LETTERS
        s = s[1:]
    return _next_word_helper(s)
```

(a) Finden und korrigieren Sie die Fehler! Versuchen Sie die Fehlersuche zunächst auf dem Papier. Kopieren Sie dann den Code in die angegebene Datei, kommentieren Sie die fehlerhaften Zeilen aus (ergänzen Sie jeweils auch einen Kommentar, der den Fehler benennt und erläutert, warum er auftritt) und fügen Sie eine Korrektur der jeweils fehlerhaften Zeile nach dem Kommentarblock ein.

(b) Ergänzen Sie anschließend in der Funktion `next_word` Doc-Tests, die deren Funktionsweise dokumentieren. Geben Sie dabei auch Testfälle an, die im nichtkorrigierten Code zu semantischen oder Laufzeitfehlern führen würden (pro Fehler zwei Tests).

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

```
python3 -m doctest -v words.py
```

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

Aufgabe 5.2 (Wort-Baum; Datei: `words.py`; Punkte: 3+2+2+2+3)

Mit Hilfe der korrigierten Funktion `next_word(s)` aus Aufgabe 5.1 soll nun ein Suchbaum der in einem String `s` vorkommenden Wörter erzeugt werden: Jeder Knoten des Suchbaumes wird durch eine Liste

```
[word, ltree, rtree, n]
```

repräsentiert. Dabei ist `word` ein Wort, `ltree` der linke Teilbaum, `rtree` der rechte Teilbaum und `n` die Anzahl der Vorkommnisse von `word` in `s`. Als Ordnungsrelation

verwenden wir Python's lexikographische Ordnung von Strings. Das heißt, ein Wort `w` wird im linken Teilbaum eines Knotens `[word, ltree, rtree, n]` eingefügt bzw. gesucht, falls der Vergleich `w < word` den Wert `True` zurückgibt, etc. In Blattknoten sind `ltree` und `rtree` jeweils der Wert `None`.

(a) Definieren Sie eine Funktion `word_tree(s)`, die bei Eingabe eines Strings `s` diesen Suchbaum erzeugt und zurückgibt. Natürlich kann Ihre Funktion eine selbst-definierte Hilfsfunktion verwenden.

(b) Definieren Sie eine Funktion `word_freq(tree, word)`, die für einen solchen Suchbaum `tree` und ein Wort `word`, die in `tree` hinterlegte Anzahl der Wortvorkommnisse von `word` zurückgibt. Falls das Wort in dem Baum nicht vorkommt, soll die Funktion den Wert `0` zurückgeben.

(c) Definieren Sie eine Funktion `print_tree(tree)`, die alle in `tree` abgelegten Wörter und die in `tree` jeweils hinterlegte Anzahl der jeweiligen Wortvorkommnisse zeilenweise (pro Zeile ein Wort und dessen Anzahl) ausgibt. Dabei soll der Baum in anti-symmetrischer Reihenfolge (*Reverse In-Order*) traversiert werden.

Die Ausgabe könnte in etwa wie folgt aussehen:

```
>>> s = "spam eggs spam eggs ham spam hamham Spam eggs hamham"
>>> tr = word_tree(s)
>>> print_tree(tr)
spam    : 3
hamham  : 2
ham     : 1
eggs    : 3
Spam    : 1
```

(d) Definieren Sie eine Funktion `freq_words(tree)`, die ein Dictionary für die Worthäufigkeiten der Wörter aus `tree` erzeugt und zurückgibt. Jeder vorkommenden Worthäufigkeit soll dabei die nicht-leere Menge aller Wörter dieser Häufigkeit zugeordnet werden. Ein Beispiel könnte wie folgt aussehen:

```
>>> freq_words(word_tree('spam eggs ham ham eggs spam eggs eggs'))
{2: {'spam', 'ham'}, 4: {'eggs'}}
```

(e) Schreiben Sie für die Funktionen in (a) und (b) und (d) auch geeignete Testfunktionen (`test_word_tree()`, `test_word_freq()` und `test_freq_words()`), die die Funktionen an jeweils vier geeigneten und selbst gewählten Beispielen als *Assertions* testen.

Benutzen Sie in Ihren Tests zur Funktion `word_freq` zum Testen auch längere Texte (mehr als 5000 Wörter). In der auf der Webseite der Vorlesung verfügbaren Datei `words_data.py` finden Sie kleine Beispieltexte, die Sie für Ihre Tests verwenden können. Fügen Sie diese Datei bitte zu Ihrem SVN-Unterverzeichnis hinzu und ergänzen Sie diese um weitere Texte Ihrer Wahl. Eine Quelle ist z.B. `http://www.gutenberg.org/`.

Hinweis: In der Datei `words.py` können Sie die Datendatei importieren, z.B. mit:

```
from words_data import loremipsum
```

Aufgabe 5.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet05` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Bezug zur Vorlesung, Interessantes, etc.).