Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel
Dr. Christian Becker-Asano, Dr. Stefan Wölfl
Wintersemester 2014/2015

Universität Freiburg
Institut für Informatik

Übungsblatt 3

Abgabe: Freitag, 14. November 2014, 18:00 Uhr

Wichtige Hinweise: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet03` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann entsprechend dem ersten Übungsblatt 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/ws1415/info1/hinweise_uebungen.txt`

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/svn/infoI1415/$RZLOGIN`

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

Aufgabe 3.1 (Operieren mit Tupeln; Dateien: `polynomial.py`, `doctest_polynomial.txt`; Punkte: 3+1+3+1)

Im Folgenden repräsentieren wir ein Polynom n-ten Grades

$$p(X) = a_n X^n + \cdots + a_2 X^2 + a_1 X + a_0$$

mit rationalen Koeffizienten ($a_n \neq 0$) durch ein Python-Tupel von floats

`(a_n, ..., a_2, a_1, a_0)`.

Die Tupel-Repräsentation eines Polynoms n-ten Grades ist also immer ein Tupel der Länge $n + 1$ und der Eintrag für den höchsten Koeffizient des Polynoms `a_n` ist von 0 verschieden.

(a) Definieren Sie in der Datei `polynomial.py` eine Funktion `first_deriv(t)`, die bei Eingabe eines Tupels `t`, das ein Polynom $p(X)$ repräsentiert, eine Tupel-Repräsentation der 1. Ableitung dieses Polynoms zurückgibt.

(b) Definieren Sie eine Funktion `deriv(t, k)`, die bei Eingabe einer Tupel-Repräsentation `t` eines Polynoms $p(X)$ eine entsprechende Repräsentation der k-ten Ableitung des Polynoms zurückgibt (für negatives k soll die Funktion `None` zurückgeben und für $k = 0$ das eingegebene Tupel selbst).

(c) Dokumentieren Sie die Datei `polynomial.py` wie in der Beispiel-Datei

`http://gki.informatik.uni-freiburg.de/teaching/ws1415/info1/docs/example.py`

exemplarisch vorgeführt. Beachten Sie die generellen Formatierungshinweise für Python-Dateien unter:

`http://gki.informatik.uni-freiburg.de/teaching/ws1415/info1/hinweise_uebungen.txt`

Achten Sie auf eine Dokumentation auf Modul-Ebene und eine Dokumentation der Funktionen mit jeweils mindestens 4 Beispielen. Ein solches Beispiel im Doc-String der ersten Funktion könnte z.B. die folgende Gestalt haben:

```
>>> first_deriv((4, 3, 2))
(8, 3)
```

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

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

aus: Voraussetzung hierfür ist, dass im Doc-String der Funktionen Beispiele angeführt wurden. Kopieren Sie die Ausgabe auf der Shell in eine Datei `doctest_polynomial.txt` und committen Sie auch diese Datei zum SVN-Repository.

Aufgabe 3.2 (Binärbäume; Datei: `addition.py`; Punkte: 3+2)

Unter einem *additiven Term* verstehen wir im Folgenden Zeichenketten, die rekursiv wie folgt definiert sind:

(1) Jeder der Kleinbuchstaben x, y, z ist ein additiver Term.

(2) Sind a und b additive Terme, so ist auch $(a + b)$ ein additiver Term.

(3) Additive Terme sind nur Zeichenketten, die sich nach den Regeln (1) und (2) bilden lassen.

Zu den Aufgaben:

(a) Definieren Sie eine Funktion `read(term)`, die angewendet auf einen String `term` den *Binärbaum* des Terms `term` erzeugt und zurückgibt: In diesem Baum entspricht jeder Nachfolgerknoten einem additiven Term, der zum Aufbau von `term` benutzt wurde. Die Blätter des Baumes sind gerade die additiven Terme, die nicht mittels Addition aus anderen additiven Termen zusammengesetzt sind.

Der Binärbaum soll in der Funktion als verschachtelte Liste repräsentiert werden (siehe Vorlesung). Das heißt, der Baum für einen additiven Term der Form

($a+b$) wird durch eine Liste `['+', left_tree, right_tree]` dargestellt, wobei der Binärbaum von a durch `left_tree` und der von b durch `right_tree` repräsentiert wird. Blattknoten werden in der Form `[char, None, None]` repräsentiert, wobei `char` einen der Kleinbuchstaben x, y oder z darstellt.

Sie dürfen bei der Aufgabe annehmen, dass `read` nur aufgerufen wird, wenn der als Argument übergebene String ein additiver Term ist. Insbesondere kommen in diesen Strings also nur die folgenden Zeichen vor: die Buchstaben x, y, z, die Klammern (und) sowie das Additionszeichen +.

(b) Definieren Sie eine Funktion `print_tree(tree)`, die den Binärbaum eines additiven Terms in *Preorder* auf der Konsole ausgibt (siehe Vorlesung). Die Ausgabe eines solchen Binärbaumes soll hierbei in einer Zeile und ohne Leerzeichen zwischen den Markierungen der Knoten erfolgen. Am Ende der Ausgabe eines Baumes soll eine neue Zeile ausgegeben werden.

Hinweis: Teilaufgabe (b) kann auch unabhängig von (a) gelöst werden. Überlegen Sie sich dazu zunächst wie die Listen-Repräsentation eines additiven Terms aussehen muss.

Aufgabe 3.3 (Tower of Hanoi; Datei: `hanoi.py`; Punkte: 1+1+3)

Das *Tower of Hanoi*-Problem ist ein bekanntes Knobelspiel, welches sich durch einen rekursiven Algorithmus lösen lässt. Dabei sind initial eine Anzahl n von unterschiedlich großen Scheiben der Größe nach (von unten nach oben) kleiner werdend auf auf einem ersten Stab *Quelle* gestapelt. Zwei weitere, leere Stäbe *Hilf* und *Ziel* befinden sich rechts von diesem ersten. Das Ziel des Spielers ist es nun, die Scheiben von der *Quelle* unter Verwendung des *Hilf*sstabes auf das *Ziel* umzustapeln. Dabei darf in jedem Schritt nur eine Scheibe von einem Stab auf einen anderen bewegt werden und niemals darf eine größere Scheibe auf einer kleineren liegen (siehe auch `http://de.wikipedia.org/wiki/Türme_von_Hanoi` für weitere Informationen).
Das folgende Python Programm löst Türme von Hanoi Instanzen rekursiv. Dabei wird jeder Stab als eine Liste bestehend aus einem Namen für den Stab und einer Liste der Scheiben, die auf ihm liegen, repräsentiert.

```
def tower_hanoi(n, source, helper, target):
    if n > 0:
        # (1) was geschieht in der naechsten Zeile?
        tower_hanoi(n - 1, source, target, helper)
        # (2) was geschieht im naechsten Anweisungsblock?
        if source[1]:
            disk = source[1][-1]
            print("Bewege", disk, "von", source[0], "nach", target[0])
            del source[1][-1]
            target[1] += [disk]
        # (3) was geschieht in der naechsten Zeile?
        tower_hanoi(n - 1, helper, source, target)

tower_hanoi(4, ["Quelle", [4, 3, 2, 1]], ["Hilf", []], ["Ziel", []])
```

Im Aufruf in der letzten Zeile werden erstens die Anzahl der Scheiben auf `source` als Integer und zweitens die jeweiligen Inhalte und Bezeichner der drei Stäbe in Form zwei-elementiger Listen als Argumente übergeben. Es wird also beispielhaft eine Türme von Hanoi-Instanz mit vier Scheiben gelöst.

(a) Kopieren Sie den Programmcode in das Eingabefenster der Webseite

`pythontutor.com/visualize.html#mode=edit`.

Visualisieren Sie anschließend die Ausführung des Programms.

- In welchem Ausführungsschritt wird zum ersten Mal eine Scheibe von einem Stab entfernt und warum erst dann?
- In welcher Konfiguration befinden sich die Stäbe nach exakt 100 der insgesamt 200 Ausführungsschritte und was passiert dann in den folgenden vier Schritten?

(b) Ersetzen Sie die Kommentare (1), (2) und (3) durch eigene Kommentare, die die jeweiligen Fragen kurz beantworten.

Aufgabe 3.4 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet03` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Interessantes, etc.).