Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel — Universität Freiburg
Dr. Christian Becker-Asano, Dr. Stefan Wölfl — Institut für Informatik
Wintersemester 2013/2014

Übungsblatt 3

Abgabe: Freitag, 15. November 2013, 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 des ersten Übungsblatts in Dateien in diesem Unterverzeichnis erwartet. Für Lösungen, bei denen eine Python-Datei eingereicht werden soll, geben wir den Dateinamen an. Generell gilt, dass in Dateinamen von selbständigen Python-Dateien (sog. *Modulen*) stets ein kurzer, nur aus Kleinbuchstaben bestehender Name zu verwenden ist (Unterstriche sind i.d.R. erlaubt, Bindestriche nicht).

Beachten Sie bei allen Aufgaben in der Python-Shell die Formatierungshinweise, die in Aufgabe 1.6 auf Blatt 1 angegeben wurden.

Für Python-Dateien beachten Sie ferner folgende Formatierungshinweise:

(a) Am Anfang jeder Python-Datei steht ein Doc-String der kurz den Inhalt der Datei beschreibt (falls Sie einen *Sha-Bang* wie etwa `#!/bin/python3` verwenden, muss dieser vor dem Doc-String stehen). Dieser Doc-String soll die folgende Gestalt haben:

```
"""<Eine Überschrift>

<Eine kurze Beschreibung, welche Funktionen, etc., in dieser Datei
definiert werden>

<Eine Liste der Quellen, die Sie bei der  Bearbeitung der
Übungsaufgaben (Bücher, Online-Quellen, etc.) verwendet haben.
Zum Beispiel:>

References:
 * http://www.python.org/dev/peps/pep-0008
 * etc.

"""
```

(b) Nach diesem Doc-String folgen zwei leere Zeilen und darauf die folgende Zeile:

```
__author__ = "<Ihr Name> (<Ihre Email Adresse>)"
```

Dieser Zeile folgen zwei Leerzeilen.

(c) Eine Einrückungsebene in Python entspricht genau 4 Leerzeichen (keine Tabulatorzeichen). Falls Sie die Tabulatortaste zum Einrücken verwenden, achten Sie darauf, dass Ihr Editor diese zu 4 Leerzeichen expandiert.

(d) Jede Zeile soll aus höchstens 79 Zeichen bestehen.

(e) Funktionen, die nicht innerhalb anderer Funktionen definiert werden, sollen durch zwei Leerzeilen getrennt werden.

(f) Funktionsnamen bestehen ausschließlich aus Kleinbuchstaben (Unterstriche sind ggf. erlaubt, wenn dies die Lesbarkeit des Codes verbessert).

(g) Jede definierte Funktion soll mit einem Doc-String versehen werden. Wir empfehlen dazu ein Schema wie in PEP 257 beschrieben. Ein Beispiel (analog zu dem in PEP 257):

```
def complex(real, imag):
    """Form a complex number.

    Arguments:
    real -- the real part (a float)
    imag -- the imaginary part (a float)

    """
    pass    # <Ihr Code beginnt hier>
```

Beachten Sie in dem Beispiel die leeren Zeilen.

Wie immer gilt: 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/infoI1314/$RZLOGIN`

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

Aufgabe 3.1 (Formatierung von Python-Dateien; 2 Punkte)

Legen Sie eine Python-Datei `first.py` entsprechend den oben angegebenen Formatierungshinweisen im Unterverzeichnis `sheet03` an. Definieren Sie in dieser Datei eine einfache Funktion Ihrer Wahl mit genau zwei Argumenten. Öffnen Sie die Datei in IDLE (Version 3.3) und führen Sie im Fenster der Python-Shell die folgenden beiden Befehle aus:

```
>>> import first
>>> help(first)
```

Speichern Sie Ihre interaktive Sitzung in der Python-Shell in der Datei `ex03-1.txt`.

Als Lösung reichen Sie bitte die beiden Dateien `first.py` und `ex03-1.txt` ein.

Aufgabe 3.2 (Operieren mit Tupeln; 3 + 1)

Im Folgenden repräsentieren wir ein Polynom n-ten Grades $f(X) = a_n X^n + \cdots + a_2 X^2 + a_1 X + a_0$ mit rationalen Koeffizienten a_i ($a_n \neq 0$) durch ein Python-Tupel von floats `(a_0, a_1, ..., a_n)`.

(a) Definieren Sie eine Funktion `fst_differentiate(t)`, die bei Eingabe einer Tupel-Repräsentation eines Polynoms $f(X)$ eine Tupel-Repräsentation der 1. Ableitung dieses Polynoms zurückgibt.

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

Bearbeiten Sie diese Aufgaben in IDLE. Legen Sie dazu eine neue Datei `differentiation.py` an, in der Sie Ihre Funktionen definieren. Beachten Sie dabei die generellen Formatierungshinweise. Testen Sie ferner Ihre Funktionen an geeigneten Beispielen in der Python-Shell von IDLE. Speichern Sie Ihre Tests (also die interaktive Sitzung in der Python-Shell) in der Datei `ex03-2.txt`. Als Lösung reichen Sie bitte die beiden Dateien `differentiation.py` und `ex03-2.txt` ein.

Aufgabe 3.3 (Aufrufbäume; 2 + 2 Punkte)

Betrachten Sie das folgende Python-Programm.

```
from math import log, floor

max = 4

def foo(n, s):
    return bar(n ** 2, s)

def bar(n, s):
    if n == 0:
        return 0
    elif s < max:
        s = s + 1
        return foo(floor(log(n, 2)), s)
    else:
        return n
```

Geben Sie die Aufrufbäume (analog zur Folie 5-26) für die folgenden Funktionsaufrufe an:

(a) `foo(1000, 0)`

(b) `bar(1000, 0)`

Aufgabe 3.4 (Aufrufbaum als Tupel; 4 Punkte)

Definieren Sie analog zur in der Vorlesung vorgestellten Fibonacci-Funktion `fib(n)` eine Funktion `fib_call(n)`, die den Aufrufbaum von `fib(n)` als ein 4-Tupel `(n, left, right, res)` zurückgibt (siehe die graphische Darstellung auf Folie 5-27). Dabei steht `left` für den Aufrufbaum von `fib(n-1)`, `right` für den Aufrufbaum von `fib(n-2)` und `res` für den Wert `fib(n)`. Für Aufrufbäume, die keine „Blätter“ unter sich haben, verwenden Sie für `left` und `right` den Wert `None`.
Bearbeiten Sie diese Aufgaben in IDLE. Legen Sie dazu eine neue Datei `fibonacci.py` an, in der Sie Ihre Funktionen definieren. Beachten Sie dabei die generellen Formatierungshinweise. Testen Sie ferner Ihre Funktionen an geeigneten Beispielen in der Python-Shell von IDLE. Speichern Sie Ihre Tests (also die interaktive Sitzung in der Python-Shell) in der Datei `ex03-4.txt`. Als Lösung reichen Sie bitte die beiden Dateien `fibonacci.py` und `ex03-4.txt` ein.

Aufgabe 3.5 (Tower of Hanoi; 2 + 2 Punkte)

Das *Tower of Hanoi*-Problem[1] 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 einem ersten Stab A gestapelt. Zwei weitere, leere Stäbe B und C befinden sich rechts von diesem ersten. Das Ziel des Spielers ist es nun, die Scheiben von Stab A unter Verwendung des Stabs B auf den Stab C 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.

```
def hanoi(n, src, hlp, trg):
    # (a) print <Infos ueber Aufruf>
    if n == 0:
        return src, hlp, trg
    # (1) was geschieht in der naechsten Zeile?
    src, trg, hlp = hanoi(n-1, src, trg, hlp)
    # (2) was geschieht im naechsten Anweisungsblock?
    sid, slst = src
    if slst:
        dsk, slst = slst[-1], slst[:-1]
        src = sid, slst
        tid, tlst = trg
        # (b) print <Anweisung für menschlichen Spieler>
        trg = tid, tlst + [dsk]
    # (3) was geschieht in der naechsten Zeile?
    hlp, src, trg = hanoi(n-1, hlp, src, trg)
    return src, hlp, trg
```

Das obige Python-Programm löst „Türme von Hanoi“-Instanzen rekursiv.

[1] Siehe `http://de.wikipedia.org/wiki/Türme_von_Hanoi`.

Dabei wird jeder Stab als ein Tupel bestehend aus einem Namen für den Stab und einer Liste der Scheiben, die auf ihm liegen, repräsentiert.
Mit dem Aufruf

```
hanoi(4, ("A", [4,3,2,1]), ("B", []), ("C", []))
```

wird beispielsweise eine Instanz mit vier Scheiben gelöst.

- Ersetzen Sie als erstes die Kommentare (a) und (b) durch geeigente `print` Anweisungen. Insbesondere sollte an der Stelle (b) die Aktionssequenz für einen menschlichen Spieler ersichtlich werden.
- Ersetzen Sie die Kommentare (1), (2) und (3) durch eigene Kommentare, die die jeweiligen Fragen kurz beantworten.

Als Lösung reichen Sie bitte die Datei `hanoi.py` mit dem entsprechend modifizierten und lauffähigen Quellcode ein.

Aufgabe 3.6 (Erfahrungen; 2 Punkte)

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