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 5

Abgabe: Freitag, 28. November 2014, 18:00 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 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`

Insbesondere müssen alle Python-Dateien und die darin definierten Funktionen entsprechend diesen Hinweisen dokumentiert werden.

Unterbinden Sie alle nicht erforderlichen `print`-Anweisungen in Ihren Python-Dateien. Falls Sie zum Debuggen `print`-Anweisungen verwenden, benutzen Sie eine globale Variable oder Konstante, mit der Sie Ausgaben auf die Konsole ausschalten können.

Überprüfen Sie, dass Sie alle Lösungen ins Repository hochgeladen haben. Bewertet wird bei allen Aufgaben die letzte Version, die zur Deadline des Übungsblattes auf dem SVN-Server eingereicht ist.

Aufgabe 5.1 (PEP8; Punkte: 3)

Die Abgaben zu diesem Übungsblatt werden daraufhin geprüft, ob Ihr Code PEP8-konform ist. Benutzen Sie daher einen Style-Checker, um alle von Ihnen eingereichten Python-Dateien daraufhin zu prüfen.

Hinweis: Das in der Vorlesung angegebene Tool pep8 berücksichtigt nicht die in PEP8 beschriebenen Namenskonventionen für Funktionen, etc. Hierzu kann (falls nicht bereits vorhanden) das Tool flake8 und die Erweiterung pep8-naming in den meisten Python-Installationen einfach nachinstalliert werden (z.B. mit pip3 install flake8).

Aufgabe 5.2 (Mengen-Operationen; Punkte: 3+2; Datei: `sets.py`)

(a) Definieren Sie eine Funktion `subsets(s, k)`, die angewendet auf eine Menge `s` und eine natürliche Zahl `k` ≥ 0, die Menge gerade jener Teilmengen von `s` zurückgibt, die genau `k` Elemente enthalten.

(b) Definieren Sie eine Funktion `permutations(s, k)`, die angewendet auf eine Menge `s` und eine natürliche Zahl `k` ≥ 0, die Menge aller Permutationen von `s` der Länge k zurückgibt.

Unter einer *Permutation* von s der Länge k verstehen wir hierbei ein Tupel der Länge k, in dem nur Elemente aus s vorkommen, kein Element aber mehr als einmal.

Schreiben Sie für die beiden Funktionen in (a) und (b) jeweils vier Testfunktionen (`test_subsets_xyz()` und `test_permutations_xyz()`; `xzy` ist dabei durch Namen Ihrer Wahl zu ersetzen).

Hinweis: Verwenden Sie zur Lösung von Aufgabe (a) keine Listen, sondern Mengen. Das Importieren von anderen Modulen ist bei der Bearbeitung der Teilaufgaben nicht erlaubt. Achten Sie ferner darauf, dass die an diese Funktionen übergebenen Mengen nach Ausführung der Funktion noch dieselben Mengen sind!

Zur Motivation von Teilaufgabe (b): Vergleichen Sie die Größe der von den Funktionen `subsets` und `permutations` zurückgegebenen Mengen.

Aufgabe 5.3 (Dictionaries; Punkte: 1+3; Datei: `actors.py`)

In dieser Aufgabe beschäftigen wir uns mit dem Datentyp `dictionary`. Zur Übung sollen Sie Daten, die in einer Liste abgelegt sind, in ein Dictionary umwandeln. Laden Sie die Datei `actors.py` von der Webseite der Übungen herunter. Die darin enthaltenen Daten haben die folgende Form:

```
actors = [[("salutation", "Mr."), ("givenName", "Peter"),
               ("familyName", "Dinklage"), ("birthday", "06/11/1969"),
               ("GoT", "Lannister", "Tyrion")],
              [("salutation", "Mrs."), ("givenName", "Michelle"),
               ("familyName", "Fairley"), ("birthday", "01/17/1964"),
               ("GoT", "Stark", "Catelyn")],
              [("salutation", "Mr."), ("givenName", "Nicolaj"),
               ("familyName", "Coster-Waldau"), ("birthday", "07/27/1970"),
               ("GoT", "Lannister", "Jamie")],
              [("salutation", "Ms."), ("givenName", "Maisie"),
               ("familyName", "Williams"), ("birthday", "04/17/1997"),
               ("GoT", "Stark", "Arya")],
              [("salutation", "Mrs."), ("givenName", "Emilia"),
               ("familyName", "Clarke"), ("birthday", "05/01/1987"),
               ("GoT", "Targaryen", "Daenerys")]]
```

Diese Beispielliste besteht aus fünf Listen mit jeweils fünf Tupeln unterschiedlicher Länge. Jedes dieser Tupel enthält Einträge vom Typ String.

(a) Machen Sie sich mit der Datenstruktur und den Daten in `actors` vertraut, indem Sie eine Funktion `actors_to_string(actors)` schreiben. Diese Funktion bekommt eine Liste in der angegebenen Form als Argument übergeben und gibt einen String zurück, der wie folgt aufgebaut ist: für jede Person werden die in `actors` abgelegten Dateneinträge der jeweiligen Person zeilenweise in der Form `<typ>: <wert>` an den String angefügt. Beachten Sie dabei, dass der Name des Charakters, den ein Schauspieler in einer Fernsehserie (im Beispiel GoT) spielt, in der Form

    `<Serie>: <Vorname> <Nachname>`

    wiederzugeben ist, also z.B. “GoT: Tyrion Lannister” im Falle des Schauspielers Peter Dinklage.

    In der Python-Konsole könnte ein Funktionsaufruf also wie folgt aussehen:

```
>>> res = actors_to_string(actors)
>>> res
salutation: Mr.
givenName:  Peter
...
```

(b) Implementieren Sie nun eine Funktion

```
actors_to_gotfamilies(actors)
```

die eine Liste der angegebenen Form als Argument übergeben bekommt. Diese Funktion soll ein `dictionary` wie folgt zurückgeben: In der Serie GoT gehören die fünf Charaktere, die von den Schauspielern in der Liste `actors` gespielt werden, drei verschiedenen Familien an. Diese Familiennamen dienen als "Schlüssel" eines äußeren Dictionary. Der Wert zu jedem Schlüssel ist wiederum ein Dictionary. In diesem (inneren) Dictionary werden nun die Vornamen der Charaktere als Schlüssel verwendet. Die Werte in den inneren Dictionaries sind dann die Daten des Schauspielers des jeweiligen Charakters. Das resultierende Dictionary hat also die folgende beispielhafte Struktur:

```
{'Lannister': {'Jamie': {'birthday': '07/27/1970',
                         'givenName': 'Nicolaj',
                         ...},
               'Tyrion': {'birthday': '06/11/1969',
                          ...} },
 ...}
```

Testen Sie Ihre Funktion mittels einer selbst definierten Testfunktion: diese soll zunächst die Funktion `actors_to_gotfamilies` auf die Liste `actors` anwenden. Formulieren Sie dann mindestens 4 geeignete Assertions über das zurückgegebene Dictionary und dessen Einträge.

Ergänzen Sie in der Datei `actors.py` an der markierten Stelle ferner eine `print`-Anweisung, die ausgibt, wieviele Mitglieder die fiktionale Familie "Stark" hat. Nutzen Sie dabei das in dieser Teilaufgabe erstellte Dictionary.

Aufgabe 5.4 (Erste Schritte mit Dateien; Punkte: 3+3; Datei: `zenfile.py`)

In den folgenden beiden Teilaufgaben werden wir Dateien lesen und schreiben. Legen Sie daher ein Unterverzeichnis `testing` im Verzeichnis `sheet05` an und vergewissern Sie sich, dass beim Testen der folgenden Funktionen nur auf Dateien in diesem Unterverzeichnis zugegriffen wird.

(a) Führen Sie in der Shell/Eingabeaufforderung den Befehl

```
python3 -m this
```

aus (es wird hier angenommen, dass Sie Python über den Befehl `python3` aufrufen können). Implementieren Sie nun in der Datei `zenfile.py` eine Funktion

`add_zen(fname)`, die die Datei `fname` im Verzeichnis `testing` öffnet und an das Ende dieser Datei die Ausgabe des Shell-Befehls `python3 -m this` schreibt. Falls die Datei `fname` noch nicht in diesem Verzeichnis existiert, soll sie von Ihrer Funktion erzeugt werden. Falls die Datei bereits existiert, darf sie nicht überschrieben (gelöscht) werden.

Testen Sie die Funktionsweise, indem Sie mindestens zweimal in dieselbe Datei schreiben.

Hinweis: Pipes werden am Ende von Foliensatz 13 eingeführt. Beim Aufruf der Pipe mittels `os.popen` muss ggfs. auch der Modus `'r'` mit übergeben werden.

(b) Implementieren Sie eine Funktion `readchars(fname, lower, upper)`, die eine Textdatei `fname` im Verzeichnis `testing` lesend öffnet und jenen Teilstring zurückgibt, der sich in der Datei ab dem Zeichen an der Stelle `lower` bis einschließlich des Zeichens an der Stelle `upper` befindet. Die Funktion wird nur aufgerufen mit Integer-Werten `lower` und `upper`, wobei beide ≥ 0 sind und `lower` $\leq$ `upper`. Ihre Funktion soll dabei maximal `upper` Zeichen aus der Datei lesen.

Falls die Datei `fname` weniger als `upper` Zeichen enthält, so soll Ihre Funktion anstelle einer Rückgabe den Fehler `ValueError` werfen. Genauer verwenden Sie hierzu den Aufruf

```
raise ValueError(msg)
```

wo `msg` ein String ist, der den Fehler erklärt.

Schreiben Sie eine oder mehrere Testfunktionen, die die Funktionalität von `readchars` mit insgesamt 4 Aufrufen der Funktion testet. Die Testfunktionen müssen dazu erst entsprechende Dateien im Verzeichnis `testing` erzeugen.

Aufgabe 5.5 (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.).