Übung 06: Selbstdefinierte Python-Klassen und fortgeschrittenes Matchen mit Unicode-Buchstabenklassen

Programmiertechniken in der Computerlinguistik I, HS 15

26. November 2015

Hinweise zur Abgabe

- Bitte gib jedes Python-Programm in einer eigenen Datei ab, die die Dateiendung `.py` hat und *ausführbaren* Python-Code enthält.
- Geize nicht mit Kommentaren direkt im Programm-Code, wo Erläuterungen angebracht sind. Umfangreiche Erklärungen werden hingegen besser in einer separaten README-Datei mitgeliefert (vorzugsweise Plain-Text oder PDF).
- Um das Hochladen der Abgabe auf OLAT zu erleichtern, kannst du die Dateien mit `zip` oder `tar` (oder einem anderen verbreiteten Format) archivieren / komprimieren.

1 Python-Klassen und Datenstrukturen

Literatur

Lies diese Einführung ins objektorientierte Programmieren mit Python und diese Erklärung zu magischen Methoden durch.

1.1 Text + Berg

Der berühmte schwedische Computerlinguist Professor **Torin Kvalm** ist von seinem Urlaub in den Bahamas nie zurückgekehrt. Vor seinem Verschwinden arbeitete der Professor an einem System, von dem er behauptete, damit die letzten Fragen der Computerlinguistik lösen zu können. Ein wichtiger Teil dieses Systems beruht auf der Analyse des Text + Berg-Korpus. Es fehlt nur noch eine Datenstruktur, bis das System einsatzbereit ist. Glücklicherweise gibt ein gut dokumentiertes Mockup, das in der Datei `aufgabe1_1.py` bereitliegt.

a) Die Klasse SACTriple repräsentiert ein getaggtes Wort aus dem Text + Berg-Korpus. Implementiere die Klasse, indem du die leeren Methoden im Mockup nach den Vorgaben ausfüllst.

b) Teste deine Klasse mit dem SAC-Test-Artikel. Welche Vorteile bietet eine solche Klasse gegenüber einem einfachen 3-Tuple? Gibt es Nachteile?

1.2 Normalisierte Strings

Wir möchten Unicode-Strings ohne Rücksicht auf Diakritika wie Akzente oder Umlaut vergleichen. Zu diesem Zweck sollst du eine Klasse `Normalstring` schreiben, die sich wie ein Unicode-String verhält, aber Diakritika ignoriert. Sie soll folgende Methoden definieren, von denen du die meisten von normalen Strings her kennst:

a) ___init___: Initialisierung der Klasse.

b) ___eq___: Implementiert den Operator `==`

c) ___getitem___: Implementiert den Operator `[]`

d) ___contains___: Implementiert den Operator `in`

e) ___str___: Wird von print aufgerufen

f) index: Gibt die erste Position zurück, an der ein Substring gefunden wurde.

Ausserdem soll sie ein Attribut `unnormal` definieren, das den ursprünglichen String enthält, mit dem der Normalstring initialisiert wurde.

Deine Klasse soll folgendes Verhalten zeigen:

```
>>> ns1 = Normalstring(u'môr')
>>> ns2 = Normalstring(u'mor')
>>> ns3 = Normalstring(u'Mor')
>>> ns1 == ns2
True
>>> ns2 == ns3
False
>>> u'ô' in ns2
True
>>> ns1[1] == ns2[1]
True
>>> print ns1
mor
>>> ns2.index(u'ô')
1
>>> ns1.unnormal
u'môr'
```

Beim Normalisieren hilft das Modul unicodedata. Verwende dazu diese Funktion:

```
import unicodedata
def normalize(input_str):
    nfkd_form = unicodedata.normalize('NFKD', input_str)
    return u"".join(c for c in nfkd_form if not unicodedata.combining(c))
```

2 Externe Bibliotheken und fortgeschrittene Regexes

Die Europäischen Parlamentsdienste haben uns eine Liste, `NAME.tsv`, verfügbar gemacht, in der auf einer Zeile jeweils der Vor- und Nachname eines Mitglieds stehen. Der Nachname ist durchgehend grossgeschrieben, der/die Vorname(n) beginnen jeweils mit einem Grossbuchstaben gefolgt von Kleinbuchstaben. Leider sind sowohl einzelne Namensbestandteile als auch der Vor- und Nachname nur durch ein Leerzeichen getrennt, sodass wir nicht wissen, wo der Nachname endet und der Vorname beginnt.

Die Liste ist in UTF-8 gespeichert und enthält Unicode-Zeichen, die im ASCII nicht vorkommen. Du solltest für diese Aufgabe das Standard-Modul `codecs` verwenden.

Um richtig mit regulären Ausdrücken auf Unicode-Strings arbeiten zu können, solltest du ausserdem die externe Bibliothek `regex` anstatt des eingebauten `re` verwenden. Du kannst dieses Paket von der Kommandozeile mithilfe von `$ sudo pip install regex` installieren. Wichtig aus dieser Bibliothek sind hauptsächlich die Zeichenklassen `\p{Lu}`, für lateinische Grossbuchstaben, und `\p{Ll}` für die Kleinbuchstaben. Zum Beispiel:

```
>>> regex.sub(ur'\p{Lu}', '_', u'DUKA-ZÁLYOMI Árpád')
u'____-_______ _rpád'
```

Ansonsten funktioniert sie wie `re`.

a) Schreibe nun ein Python-Programm, das die Nachnamensbestandteile und Vornamensbestandteile erkennt, in Kleinbuchstaben umwandelt, und daraus eine neue Datei erstellt, wo der normalisierte Vorname, Nachname, und der ursprüngliche Name jeweils von einem Tabulator getrennt auf einer Zeile stehen.

b) Teste deine Datei gegen den Goldstandard. Du kannst dazu das Kommandozeilentool diff benutzen:

```
$ diff --suppress-common-lines -y GOLD.tsv out.tsv | wc
```

Dein Programm sollte nicht mehr als 100 Fehler machen.

Reflexion/Feedback

a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen zusammen.

b) Wie viel Zeit hast du in diese Übungen investiert?