Schlussprüfung Programmiertechniken in der Computerlinguistik I HS 12

Aufgabenstellung: Martin Volk/Simon Clematide

Prüfung vom 10. Januar 2013
Institut für Computerlinguistik, Universität Zürich

Vorname _____ Matrikelnummer _____

Nachname _____

Für Studierende der folgenden Studiengänge:

☐ BA - Studiengang Computerlinguistik (Phil. Fakultät)

☐ BA - Studiengang Computerlinguistik und Sprachtechnologie (Phil. Fakultät)

☐ BA-Studierende (Wirtschaftswissenschaftliche Fakultät)

☐ Studierende des Nebenfachs Informatik mit Studienbeginn ab WS 04/05

☐ Multidisziplinfach (ETH)

☐ Andere:

Nur für Lizenziatsstudierende der Computerlinguistik als ein Fach aus der Phil. Fakultät:

Strasse: _____ Hauptfach: _____

PLZ/Ort: _____ E-Mail: _____

Die Prüfungsresultate von ECL und PCL von den Lizentiatsstudierenden werden (zusammen) per Post verschickt. *Bitte Adresse nicht vergessen!*

Aufgabe Nr.:	1	2	3	4	5	6	7	8	**Summe**
Punktzahl:	11	12	4	10	5	14	16	18	90
Davon erreicht:									

Note SU: _____ Note SP: _____

Endnote: _____ Bestanden: ☐ Ja ☐ Nein

Auf jedes separate Blatt mit Lösungen den Nachnamen schreiben! Viel Erfolg!

Wichtige Hinweise

Maximale Punktzahl: 90 Pro Punkt sollte ungefähr 1 Minute gebraucht werden.
Hinweise: Bitte schreibe in einem überlegten und knappen, aber verbalen Stil (keine Stichwortsammlungen). Bei inhaltlichen Auswahlsendungen, wo einfach mal alles spontan hingeschrieben wird und Falsches wie Korrektes munter vermischt sind, behalten wir uns Abzüge vor. Erlaubtes Hilfsmittel ist die Referenzkarte zu Python. Nutze die Hinterseite der Blätter, falls du mehr Platz brauchst. Die Antworten können auf **deutsch** (bevorzugt) oder **englisch** erfolgen.

1. Greppen im Text+Berg-Korpus (11 Punkte)

Das multilinguale Korpus Text+Berg liegt in XML-Dateien vor, welche für jedes Jahrbuch (1864–2011) den tokenisierten, getaggten und lemmatisierten Text in identischer Formatierung enthalten, wie im folgenden Auschnitt gezeigt:

```
<s lang="de" n="1-5">
  <w lemma="Rein" n="1-5-1" pos="NN">Rein</w>
  <w lemma="rechnerisch" n="1-5-2" pos="ADJD">rechnerisch</w>
  <w lemma="haben" n="1-5-3" pos="VAFIN">haben</w>
```

Die Dateien sind für eine Sprache bis auf die vierstellige Jahreszahl immer identisch benannt. Der Name fürs deutsche Jahrbuch für 2010 ist `SAC-Jahrbuch_2010_de.xml`

4 (a) Beschreibe knapp, aber möglichst genau, was der folgende Befehl genau ausgibt, wenn er in dem Verzeichnis ausgeführt wird, wo alle XML-Dateien von Text und Berg liegen:

```
$ grep -B 1 'Gipfel' *191*de.xml | grep 'ADJA'
```

.....

.....

.....

.....

1 (b) Wie lautet der Terminalbefehl, wenn man die Anzahl aller Tokens von einem **einzelnen** deutschen Jahrbuch mit UNIX-Tools berechnen will?

.....

3 (c) Wie lautet Terminalbefehl, wenn man die **Gesamtzahl** der Tokens aus allen deutschen Jahrbüchern mit UNIX-Tools berechnen will?

.....

3 (d) Um das Tagging von Bergnamen zu verbessern, soll überprüft werden, welche Tokens mit typischen Bergnamenendungen **fälschlicherweise als normale Nomen** (`NN`) getaggt sind. Wie lautet der Befehl, um vom Text+Berg-Korpus jede Zeilen auszugeben, deren Token auf `horn`, `berg` oder `spitz` **endet** und mit `NN` getaggt ist.

.....

.....

2. Theoriefragen (12 Punkte)

2 (a) Inwiefern sind Generatorausdrücke effizienter als Listenkomprehension? Wann ist das relevant?

3 (b) Was ist eine univariate Häufigkeitsverteilung? Gib ein Beispiel.

4 (c) Was sind die Unterschiede bzw. Gemeinsamkeiten von Klassen/Typen und Objekten in Python? Wie werden sie erzeugt?

3 (d) Wozu sind Funktionen gut und was zeichnet gute Funktionen aus?

4 **3. Modifizierbare Datenstrukturen (4 Punkte)** Hans behauptet, dass Zeichenketten in Python veränderliche Datenstrukturen sind, da man ja folgenden Kode schreiben kann:

```
a = "Ein Satz."
a += "Und ein anderer Satz."
```

Was meinst du zu seinem Beispiel? Wie kann man diese Frage am Interpreter besser prüfen?

.....
.....
.....
.....
.....
.....
.....

10 **4. Reguläre Ausdrücke (10 Punkte)** Gegeben sei der folgende reguläre Ausdruck in Python:

```
pattern = r'(\d{1,2}[./-]){2}([12]\d(\d\d)?)
```

Kreuze das Kästchen an, falls obiger Ausdruck mittels `re.search(pattern,string)` etwas in den folgenden Strings matcht und schreibe die erste gematchte Zeichenkette auf die freie Linie. (Die Hochkommata begrenzen den String und gehören nicht zu den zu matchenden Zeichen.)

☐ `'03.2003'` _____

☐ `'around 2012/9/21'` _____

☐ `'@1.66/444$$'` _____

☐ `'ev. am 19.11. oder 21.12.2012'` _____

☐ `'1/1/1988'` _____

☐ `'1.12-22.12.2013'` _____

5 **5. Programm kommentieren (5 Punkte)** Was macht die folgende Python-Funktion? Kommentiere die Zeilen und beschreibe damit die Funktionalität.

```
def foo(w, s):
    try:
        return w.index(s)
    except ValueError:
        return None
```

6. Listenoperationen in Python (14 Punkte) Gegeben sei folgende Liste[1] mit den Wörtern eines einfachen Liedes:

```
song = ['I', 'scream', 'you', 'scream', 'we', 'all', 'scream', 'for', 'ice', 'Cream']
```

[1] (a) Wie kann man die Anzahl **verschiedener** Wörter in `song` berechnen?

.....

[1] (b) Ersetze das **letzte** Wort von `song` durch seine kleingeschriebene Variante.

.....

[1] (c) Programmiere so, dass die Variable `song` den Liedtext dreimal hintereinander enthält.

.....

[4] (d) Wie kann man die vorgängig modifizierte Liste `song` so modifizieren, dass alle `"I"` mit `"you"` vertauscht sind?

.....

.....

.....

.....

[3] (e) Wie kann man jedes 2. Wort von `song` mit `print` ausgeben? Benutze den arithmetischen Operator `%` (ausgesprochen modulo), der den Rest einer Ganzzahl-Division berechnet:

```
print 12 % 2
>>> 0
print 11 % 2
>>> 1
```

.....

.....

.....

[2] (f) Schreibe einen Listenkomprehensionsausdruck, welcher den Text von `song` in LAUT GESCHRIEHENER GROSSSCHREIBUNG enthält.

.....

.....

[2] (g) Schreibe einen Listenkomprehensionsausdruck, welche jedes Wort und seine Wortlänge als Paar in einer Liste ergibt. Z.B. `[('I',1),('scream',6), ..., ('cream',5)]`

.....

.....

[1] Inspiriert durch den Jarmusch-Film "Down by Law" (1986).

7. Getaggter Text und Dictionaries (16 Punkte) Gegeben sei eine in UTF-8 kodierte Datei 'es.tts' mit vertikalisiertem getaggtem Text, wo jeder Wortform eine Wortart zugeordnet ist:

```
Estos DM
30 CARD
símbolos NC
representan VLfin
un ART
sistema NC
fonológico ADJ
de PREP
5 CARD
fonemas NC
vocálicos ADJ
y CC
25 CARD
fonemas NC
consonanticos ADJ
. FS
```

5 (a) Schreibe die Anweisungen, welche nötig sind, um den Inhalt dieser Datei in einer Liste namens `es_tts` abzuspeichern, welche aus Paaren der Form (Wort, Wortart) besteht.

4 (b) Erzeuge ein Dictionary/Hash `pos_stats`, das zählt, wie oft jede Wortart vorkommt.

2 (c) Gebe die Anzahl unterschiedlicher Wortarten in diesem Korpus aus mit `print`.

5 (d) Erzeuge ein Dictionary, das als Wortartenlexikon dient. D.h. es enthält als Schlüssel eine Wortform und als dazugehöriger Wert die Liste aller möglichen Wortarten-Tags.

18 **8. Verwandte maskuline und feminine Wortformen finden (18 Punkte)**

Der Hilfsassistent Claudio und die Hilfsassistentin Claudia haben ihren ersten Auftrag erhalten: Extrahiert im deutschen Teil des Text+Berg-Korpus morphologisch verwandte Wortpaare, die aus einer männlichen Form wie "Hilfsassistent" und einer entsprechenden weiblichen Form auf "-in" wie "Hilfsassistentin" bestehen. Das Resultat soll in folgendem Format mit `print` ausgegeben werden, aber jedes Wortpaar soll nur einmal erscheinen:

```
Hilfsassistent Hilfsassistentin
Kinderarzt Kinderärztin
Bergbauer Bergbäuerin
```

Das Programm soll auch einfache Umlautung (nur Wechsel von "a" zu "ä") behandeln. Implementiere die Umlautung so, dass immer das letzte "a" in einem Wort zu einem "ä" werden kann in der femininen Form. Tipps: Die Methode `str.rindex()` funktioniert gleich wie `str.index()`, sucht aber von rechts nach links. Löse die Aufgabe zuerst ohne die Umlautung, falls sie dir schwierig vorkommt.

Claudia und Claudio sind soweit gekommen, dass sie alle 9'308'358 Tokens aus dem Korpus in einer langen Python-Liste namens `bttext` abgespeichert haben. Leider wissen Sie nicht mehr weiter. Sie sind froh, wenn du ihnen den fehlenden Kode programmierst und dabei auf eine effiziente Verarbeitung achtest.

```
print bttext[:6]
>>> ['Zum','Geleit','Mit','dieser','ersten','Seite' ]
```