Prüfung Programmiertechniken in der Computerlinguistik I
HS 2014

Aufgabenstellung: Simon Clematide/Martin Volk

Prüfung vom 8. Januar 2015
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)

☐ Anderer Studiengang: _____

Aufgabe Nr.:	1	2	3	4	5	6	7	8	9	10	**Summe**
Punktzahl:	18	15	12	4	5	7	6	8	5	10	90
Davon erreicht:											

Note SU: _____ Note SP: _____

Endnote: _____ Bestanden: ☐ Ja ☐ Nein

Wichtige Hinweise

- Maximale Punktzahl: 90
- Pro Punkt sollte ungefähr 1 Minute gebraucht werden.
- Bitte schreiben Sie in einem knappen, aber verbalen Stil (keine Stichwortsammlungen). Bei inhaltlichen Auswahlsendungen, wo einfach alles spontan hingeschrieben und Falsches wie Korrektes munter vermischt wird, behalten wir uns Abzüge vor.
- Erlaubtes Hilfsmittel ist die Referenzkarte zu Python.
- Die Antworten können **deutsch** (bevorzugt) oder **englisch** sein. Bei sprachlichen Verständnisfragen melden Sie sich bei der Aufsichtsperson.

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

1. Auf der Kommandozeile mit dem Text+Berg-Korpus arbeiten (18 Punkte)

Gegeben seien 48 Jahrbücher des Text+Berg-Korpus von 1900 bis 1949 wie in der Übung 0 von PCL I. Es gibt Tokenzeilen und Strukturzeilen. Tokenzeilen enthalten in 3 tabulator-separierten Spalten das Token, das POS-Tag und das Lemma. Strukturzeilen markieren den **Anfang** von Zeitschriftenartikeln und Sätzen und beginnen mit "`<article `" bzw. "`<s `".

```
<article n="1">
<s lang="de" n="1-1">
Vorwort           NN              Vorwort
<s lang="de" n="1-2">
Das               ART             d
Jahrbuch          NN              Jahrbuch
XXXVI             CARD            @card@
weicht            VVFIN           ab+weichen
ab                PTKVZ           ab
.                 $.              .
<s lang="de" n="1-3">
Dementsprechend   ADJD            dementsprechend
```

Alle Dateien sind gemäss Beispiel "`SAC-Jahrbuch_1900_mul_column.txt`" benannt. Die folgenden Fragen gehen davon aus, dass die Befehle im Verzeichnis abgesetzt werden, wo die Jahrbücher liegen. Bei grep können Sie die Option "-E" oder "-P" hinschreiben, um anzudeuten, dass Sie die erweiterten regulären Ausdrücke verwenden, welche Sie auch von Python kennen.

(a) Wie bestimmt man im Terminal mit Hilfe von UNIX-Kommandos die Anzahl Zeitschriftenartikel im Jahrbuch von 1900? [2]

```
$ grep -c '^<article' SAC-Jahrbuch_1900_mul_column.txt
$ grep -c '<article ' SAC-Jahrbuch_1900_mul_column.txt
```

(b) Wie bestimmt man im Terminal **die Gesamtanzahl aller** Zeitschriftenartikel in allen Jahrbüchern? [3]

```
$ cat *.txt | grep -c '<article '
$ grep -c '<article ' *.txt gibt nur einen Punkt, da die counts pro Datei ausgegeben werden und nicht die Gesamtzahl.
```

(c) Wie kann man jedes (deutsche) Partizip Perfekt (VVPP POS-Tag) im Terminal ausgeben, das mit ‚ge' beginnt und auf ‚t' endet, wobei die Grossschreibung keine Rolle spielen soll? [4]

```
$ grep -Pio '^(ge\S*t)\tVVPP' *.txt
$ cat *.txt | grep -Pio '^(ge\S*t)\tVVPP'
```

(d) Wie kann man im Terminal eine Häufigkeitsstatistik aller POS-Tags erstellen? D.h. einen Output ausgeben, wo für jedes POS-Tag ersichtlich ist, wie oft es insgesamt in allen Jahrbüchern zusammen vorkommt? [4]

```
$ cat *.txt | grep -Po '\t(.+)\t' | sort | uniq -c
$ cat *.txt | grep -Po '\t(.+?)\t' | sort | uniq -c
Solange es nur 3 tabseparierte Spalten gibt pro Zeile (d.h. höchstens 2 Tabulatorzeichen, muss nicht zwingend nicht-gierig gematcht werden.
```

(e) Wie kann man im Terminal alle Zeilen ausgeben, bei denen sich die Wortform nur durch eine s-Endung oder es-Endung vom Lemma unterscheidet? [5]

```
$ cat *.txt | grep -P '(\S+)e?s\t.*\t\1$'
$ grep -P '(\S+)e?s\t.*\t\1$' *.txt
Wenn $ vergessen wurde, 3 Punkte Abzug.
```

2. Python-Datenstruktur für das Text+Berg-Korpus (15 Punkte)

Wie könnte eine geeignete und einfache Repräsentation (=Datenstruktur) in Python aussehen für die Korpusdaten eines Jahrbuchs von Aufgabe 1 dieser Prüfung? Folgende Bedingungen müssen erfüllt sein:

- Die Artikelstruktur wird ignoriert.
- Für jedes Token muss das POS-Tag und das Lemma abgespeichert sein.
- Für jeden Satz muss die Sprache bestimmbar sein.
- Man soll mit normaler Python-Indexierung, d.h. mit der `[]`-Notation auf die Elemente zugreifen können, z.B. die Wortart vom letzten Token in einem Satz berechnen.

(a) Beschreiben Sie in natürlicher Sprache, wie Sie die Korpusdaten in Python repräsentieren. Begründen Sie kurz und erwähnen Sie allfällige Vor- bzw. Nachteile. 5

- Ein satzsegmentiertes Korpus ist grundsätzlich eine Sequenz von Sequenzen von Elementen. Ob man als Sequenz-Datenstruktur Listen oder Tupel verwendet, hängt davon ab, ob man das Korpus in irgendeiner Form modifizieren können will. Im Folgenden gehen wir davon aus, dass man das nur auf der Ebene von ganzen Sätzen machen können möchte.
- Pro Satz braucht es eine Sequenz (Liste) mit Informationen zu jedem Token (TOKENINFO) sowie eine Sprachangabe (SPRACHINFO). Die beiden Informationen können in einem 2-er-Tupel gespeichert werden: SATZINFO = (SPRACHINFO, TOKENINFO-LISTE)
- Jedes Token muss mit seinen linguistischen Informationen (POS-Tag und Lemma) abgespeichert sein. TOKENINFO = (WORTFORM,POS,LEMMA)
- Ein Jahrbuch ist dann eine Liste von SATZINFOs.
- Grundsätzlich könnte man die Satznummer auch explizit speichern in den Sätzen, was für gewisse Anwendungen, wo man ausgehend von einem Satz seine Umgebung absuchen will, von Vorteil sein kann.
- Man könnte für jede TOKENINFO auch einen kleine Dictionary machen oder auch für jeden Satz, damit man mit benannten Merkmalen auf die Werte zugreifen kann. Das benötigt dann doch mehr Speicher, weil ja immer auch die Merkmal mitgespeichert werden müssen zu den Werten (das Dictionary kann nicht davon ausgehen, dass nichts anderes drin abgespeichert werden soll.)

(b) Schreiben Sie einen Python-Ausdruck hin, der die Beispielzeilen von Aufgabe 1 in der von Ihnen gewählten Datenstruktur repräsentiert. 5

```
sample = [
   ("de", [('Vorwort','NN','Vorwort')]),           # Satz 1
   ("de", [('Das','ART','d'),('Jahrbuch','NN','Jahrbuch'),
             ('XXXVI','CARD','@card@'), ('weicht','VVFIN','ab+weichen'),
             ('ab','PTKVZ','ab'), ('.','$.','.')]),    # Satz 2
   ("de", [('Dementsprechend','ADJD','dementsprechend'),...]), # Satz 3
   ...
  ]
```

Wenn man alle Sequenzen als Listen nimmt:

```
sample = [
   ["de", [['Vorwort','NN','Vorwort']]],        # Satz 1
   ["de", [['Das','ART','d'],['Jahrbuch','NN','Jahrbuch'],
             ['XXXVI','CARD','@card@'], ['weicht','VVFIN','ab+weichen'],
             ['ab','PTKVZ','ab'], ['.','$.','.']]], # Satz 2
   ["de", [['Dementsprechend','ADJD','dementsprechend'],...]], # Satz 3
   ...
  ]
```

Falls man eine völlig andere Datenstruktur hat mit eigenen Objekten, müssen die entsprechenden Konstrukturaufrufe notiert sein.

(c) Angenommen das Jahrbuch 1900 wäre in Ihrem Format in der Variable `corpus1900` abgelegt. Schreiben Sie den Python-Kode auf, um herauszufinden, wie viele deutsche Sätze im Jahrbuch weniger als 3 Token haben. 5

```
de_short_sents_count_1900 = len([s for s in corpus1900
                                 if s[0] == "de" and len(s[1]) < 3])
```

3. Python-Theorie (12 Punkte)

Kreuzen Sie **alle** korrekten Aussagen an. Jede richtige Antwort ergibt 0.5 Punkte, jede **falsche** Antwort ergibt 0.5 Punkte **Abzug**. Man kann minimal 0 Punkte machen pro Teilaufgabe. Es können beliebig viele Antworten richtig oder falsch sein.

(a) Funktionen [2]

- ■ **Funktionen müssen nicht zwingend ein return-Statement enthalten.**
- ■ **Wenn im Rumpf einer Funktionsdefinition zuerst ein Stringliteral steht, wird dies als Online-Hilfe der Funktion verwendet.**
- □ Wenn eine Funktion mit 2 Argumenten definiert wurde, kann man sie auch nur mit 1 Argument aufrufen. Das 2. Argument wird dann implizit als None gesetzt.
- □ Die Namensräume für Funktionen und andere Objekte sind unterschiedlich. Folgender Code gibt "Test" aus:

```
def foo():
   print "Test"
foo = 1
foo()
```

(b) Klassen [2]

- ■ **Methoden sind auf der Ebene von Klassen definiert.**
- ■ **Vererbung bezweckt, dass nicht jede Klasse alle ihre Methoden definieren muss.**
- ■ **Alle Objekte sind Instanzen einer Klasse.**
- ■ **Die grundlegende Klasse in Python heisst `object`.**

(c) Objekte [2]

- ■ **Jedes Attribut ist ein Objekt.**
- ■ **Attribute werden standardmässig in den Instanzen gespeichert.**
- □ Listen sind in Python keine Objekte, sondern spezielle Datenstrukturen.
- □ Dictionaries sind in Python keine Objekte, sondern spezielle Datenstrukturen.

(d) Zeichenketten [2]

- ■ **Der String `r"\t"` hat eine Länge von 2 Buchstaben.**
- □ Zeichenketten, die mit einfachem Apostroph `'` begrenzt sind, dürfen sich über mehrere Zeilen im Quellkode erstrecken.
- ■ **UNICODE-Escape-Sequenzen der Form `\uNNNN` werden in einer Zeichenkette wie `ur'Ein\u0020A'` zu einem Buchstaben (hier Leerzeichen) aufgelöst.**
- ■ **Eine Zeichenkette `s`, welche UTF-8-kodierte Bytes enthält, kann mittels `s.decode('utf-8')` in den entsprechenden `unicode`-String verwandelt werden.**

(e) Veränderliche Datenstrukturen [2]

- □ Zeichenketten vom Typ `str` sind in Python veränderlich.
- □ Ein Python-Ausdruck für ein Tupel wie `('der',[103,1014])` darf als Schlüssel in einem Dictionary verwendet werden.
- ■ **Mengen (`set`) können nur unveränderliche Datenstrukturen als Elemente haben.**
- □ Tupel dürfen keine Listen als Element enthalten.

(f) Identität und Wertgleichheit [1]

- □ Wenn 2 Objekte denselben Wert haben (`==`), dann sind sie auch identisch (`is`).
- □ Wenn 2 Objekte identisch (`is`) sind, dann haben sie auch denselben Wert (`==`).

(g) Listenkomprehensionsausdrücke [1]

- ■ **Auf Listenausdrücke lassen sich Methoden anwenden mit der Punktnotation, z.B. so:**
  `[x.lower() for x in "ABC"].index("b")`
- □ Listenkomprehensionsausdrücke können im Bedingungsteil Print-Anweisungen enthalten.

4. Erklären Sie den Unterschied von Ausdrücken und Anweisungen in Python (4 Punkte) 4

- Ausdrücke evaluieren zu Objekten (falls nicht eine Ausnahme die Berechnung unterbricht) und können keine Anweisungen enthalten.
- Anweisungen werden ausgeführt, aber evaluieren nicht zu einem Wert. Anweisungen können Ausdrücke enthalten.

5. Lokale Namensräume (5 Punkte) 5

Was versteht man in Python unter lokalen Namensräumen? Geben Sie 2 unterschiedliche Beispiele, wie lokale Namen entstehen können.

Der globale Namensraum ist nur auf der Ebene eines Moduls vorhanden. Lokale Namensräume sind alle Namensräume, die nicht global sind, z.B. in Funktionen, Methoden. Diese Namen existieren nur solange, wie das Konstrukt, in denen sie definiert wurden.

- Die Parameter einer Funktion werden beim Funktionsaufruf zu lokalen Namen.
- Jede Variable, die in einer Funktion gebunden wird und nicht global ist, wird zu einem lokalen Namen.

6. Reguläre Ausdrücke in Python (7 Punkte) 7

Im Gutenberg-Projekt werden in der Textversion von Büchern manchmal Notationen im Markdown-Stil verwendet. Wie in der Tabelle unten ersichtlich wird mit Tilde *kursiver Text* und mit Dollar **Fettdruck** markiert.[1]

Markdown-Notation	Notation mit HTML-Tags (=markup)
`Kursiver Text ist ~als kursiv~ und Fettdruck $als fett$ markiert.`	`Kursiver Text ist <i>als kursiv</i> und Fettdruck <b>als fett</b> markiert.`

Schreiben Sie in Python eine Funktion `markdown2html(string)`, welche die Markdown-Notation (wie links) in eine Zeichenkette mit HTML-Tags (wie rechts) überführt und als Funktionswert zurück gibt.

Am einfachsten geht es mit Hilfe von regulären Ausdrücken mit nicht-gierigen Operatoren.

```
import re
def markdown2html(string):
      result = re.sub(r'~(.+?)~',r'<i>\1</i>',string)
      result = re.sub(r'\$(.+?)\$',r'<b>\1</b>',result)
      return result
```

- Falls gierig gematcht wird, gibt es 2 Punkte Abzug.
- Falls die Funktion nur für obigen Fall mit nur einer Markdown-Markierung funktioniert, 4 Punkte Abzug.
- Falls `$` nicht geschützt ist, 1 Punkt Abzug.
- Falls keine raw-Strings benützt werden, muss der Backslash im Ersetzungsausdruck geschützt werden: `result = re.sub('\$(.+?)\$',r'<b>\1</b>',result)` oder
  `result = re.sub('\$(.+?)\$','<b>\\1</b>',result)`
- Eine Lösung, welche davon ausgeht, dass die Formatierungszeichen, welche eine Formatierung einleiten, jeweils links Whitespace haben, und die rechte Begrenzung jeweils rechts ein Whitespace hat, ist auch ok.

[1] `http://www.gutenberg.org/files/42489/42489-0.txt`

```
def markdown2html2(string):
      result = string.replace(' ~',' <i>')
      result = result.replace(' $',' <b>')
      result = result.replace('~ ','</i> ')
      result = result.replace('$ ','</b> ')
      return result
```

7. Manipulation von Listen (6 Punkte) 6

Das Folgende sind Eingaben im Python-Interpreter, welche nacheinander ausgeführt werden. Schreiben Sie die Ausgabe der Print-Statements auf die entsprechenden Linien.

```
>>> names = ['Xenia', 'Carla', 'Rosa']
>>> print [n[0] for n in names]          Ausgabe: ['X','C','R']
>>> names2 = names
>>> names.sort()
>>> print names[1]                       Ausgabe: Rosa

>>> print names2[1]                      Ausgabe: Rosa
>>> names = ['Xenia', 'Carla', 'Rosa']
>>> print names.append('Daniela')        Ausgabe: None

>>> print names[-1]                      Ausgabe: Daniela
>>> names2 = names
>>> del names[0]
>>> print names2 is names                Ausgabe: True
```

Benotungshinweis: 1 Punkt pro Antwort, bei der ersten Aufgabe 0.5 Punkte Abzug, wenn Anführungszeichen fehlen in der Liste.

8. Übersetzen (8 Punkte) 8

Ein Hacker hat Teile eines einfachen Übersetzungsprogramms böswillig gelöscht. Mit Hilfe des folgenden Beispieloutputs sollte es Ihnen möglich sein, das Programm wieder lauffähig zu machen.

```
Input: Tante Polly sah den Mann mit dem Fernglas
Translation: aunt <<<Polly>>> saw the man with the telescope
```

Schreiben Sie den fehlenden Kode direkt auf die leeren Linien. Für die Übersichtlichkeit ist das Lexikon alphabetisch geordnet auf den Keys.

```
my_lexicon = {
      'Fernglas':'telescope',
      'Mann':'man',
      'Tante':'aunt',
      'dem':'the',
      'den':'the'
      'der':'the',
      'im':'in,
      'mit':'with',
      'sah':'saw'}

test_sentence = 'Tante Polly sah den Mann mit dem Fernglas'
print 'Input:', test_sentence

test_list = test_sentence.split()

print 'Translation:',

for word in test_list:
```

```
if word in my_lexicon:
    print my_lexicon[word],
else:
    print '<<<'+word+'>>>',
```

Benotungshinweis: Pro Zeile 2 Punkte. Wenn das Komma am Ende des Print-Statements fehlt, dann gibts jeweils 0.5 Punkt Abzug.

9. Init-Methode (5 Punkte) Wozu dient die `__init__()`-Methode bei selbstdefinierten Klassen? Wann wird sie aufgerufen? **5**

- Die `__init__()`-Methode ist die **Konstruktordefinition**, die man in Python bei einer selbstdefinierten Klasse normalerweise explizit definiert. (2 Punkte)
- Die Initialisierung wird genau dann aufgerufen, wenn ein Objekt von einer Klasse instantiiert wird (**Objektinstantiierung** von einer Klasse). (2 Punkte)
- Sie dient dazu, die bei der Instantiierung allfällig **mitgegebenen Parameter** zu verarbeiten und **allfällige Instanzvariablen** des Objekts festzulegen. (2 Punkte)
- Diese Methode wird normalerweise nur **einmal aufgerufen** für ein Objekt und zwar implizit via **Konstruktur**, der dem **Klassennamen** entspricht. (2 Punkte)

Man kann maximal 5 Punkte erhalten.

10. Kontextabhängig Übersetzen (10 Punkte) **10**

Die Firma "troogle" bietet ein gleichnamiges Python-Package an, mit dem man kontextabhängig die wahrscheinlichste deutsche Übersetzung für ein einzelnes englisches Wort nachschlagen kann. Die Funktion `lookup(word, preceding, succeeding, translated_preceding)` liefert die beste deutsche Übersetzung für Wort `word`, wobei gewisse Kontextinformation berücksichtigt wird. Je **mehr Information** mitgegeben wird, umso **besser** wird die Übersetzung.

- Das Argument `preceding` ist das vorangehende englische Token oder `None`, falls es keines gibt.
- Das Argument `succeeding` ist das nachfolgende englische Token oder `None`, falls es keines gibt.
- Falls die deutsche Übersetzung des vorangegangenen Worts bereits bekannt ist, wird es als Argument `translated_preceding` mitgegeben, ansonsten `None`.

Der Beispielaufruf, um die Übersetzung von "the" in einem Satz wie "the man with a telescope…" nachzuschlagen lautet: `lookup('the',None,'man',None)` und würde idealerweise zu "der" evaluieren. Im Satz "the woman with a telescope" würde man die Zeichenkette "die" als Resultat des Lookups erwarten.

Definieren Sie eine Funktion `lookup_sentence(sent)`, welche einen englischen Satz (=Liste von Tokens) von links nach rechts mit Hilfe von `lookup` möglichst gut in einen deutschen Satz (=Liste von Tokens) übersetzt und ihn als Funktionswert zurückliefert. Bitte auch Sätze, die nur aus 1 Wort bestehen, korrekt behandeln. Achten Sie auf die korrekte Behandlung von Wörtern am Anfang und Ende des Satzes.

```
from troogle import lookup
def lookup_sentence(sent):
      sentlen = len(sent)
      if sentlen == 1:
            return [lookup(sent[0],None,None,None)]
      translation = []
      lastindex = sentlen-1
      for i,w in enumerate(sent):
```

```
        if i == 0:
              translation.append(lookup(w,None,sent[i+1],None))
        elsif i == lastindex:
              translation.append(lookup(w,sent[i-1],None,translation[i-1])
        else:
              translation.append(lookup(w,sent[i-1],sent[i+1],translation[i-1])
  return translation
```

- Je 3 Punkte Abzug, wenn 1-Wort-Sätze, oder Anfang und Ende nicht korrekt behandelt werden.