Das Tagging-Projekt in Python

```
#-*- coding: iso-8859-1 -*-

##########################################################################
####################### TAGGING  UND CHUNKPARSING ########################
##########################################################################

####################       TEIL 1: TAGGING     ########################

#Importierte Module:
import nltk
from nltk.corpus import brown
from nltk.tokenize import *
from nltk.chunk import *
from nltk.chunk.regexp import *
from re import *
from re.show import *
import datei
from datei import *

#Sprachdaten für die Analyse:
text = """the dog barked.
the boy sang beautifully.
the likelihood for that result is unexpectedly low.
John knows that she extracted the information carefully.
the neighbourhood evolved slowly.
Mary thinks that Bill took her car.
the guy from the garage said that the car is unrepairable.
that book is abolutely readable.
the students guessed that the task involves tagging.
"""
#Methoden zur Aufbereitung der Sprachdaten
def tokenisiere_satz(Satz):
    return nltk.WordPunctTokenizer().tokenize(Satz)

def splitte_text(Text):
    return Text.split('\n')

def tokenisiere_text(Text):
       Satzlistenliste = []
       for element in splitte_text(Text):
              Satzlistenliste.append(tokenisiere_satz(element))
       return Satzlistenliste

#Methoden zur Erzeugung eines Trainingsdatensatz mit angenehmen Tags

#Dictionary mit Zuweisung Brownscher Tags zu neuen Tags
tag_table={'AB':'predet','AP':'det','AT':'det','BE':'cop','CD':'card','CS':'conj','CC':'c
onj','DO':'verb','DT':'det','EX':'pro','HV':'verb', 'IN':'prep','JJ':'adj',
'MD':'modal','NN':'noun','NP':'name','NR':'name','OD':'ord','PN':'pro', 'PP':'pro',
'RB':'adv','RP':'adv','TO':'inf','VB':'verb','WD':'det','WP':'pro','WQ':'adv',
'WR':'adv','--':'--'}

#Methode zum Abbilden eines 'alten' Tags auf ein 'neues' Tag (Bezug: tag_table)
def tag2tag(tag):
    if tag in '()*,.:':return tag
    if tag == 'PP$':return 'det'
    temp=tag[:2]
    if tag_table.has_key(temp):
        return tag_table[temp]
    else:
        return '??'
#Methode zum Erzeugen eines 'Token/Tag'-Tupels mit neuem Tag auf Grundlage eines
#bestehenden Tupels (Bezug: tag2tag())
def replace_tag(tupel):
    return (tupel[0],tag2tag(tupel[1]))
#Methode zum Ersetzen der Tags in einer Liste von'Token/Tag'- Tupeln
(Bezug: replace_tag())
def replace_tags(Satzliste):
    return [replace_tag(tupel) for tupel in Satzliste]
#Erzeugen zweier Datensätze aus dem Brown-Corpus zum Tagger-Training
#brown_train: mit Brownschen Tags
#brown_train2: die gleichen Daten mit neuen Tags
brown_train = brown.tagged_sents(categories='k')
brown_train2 = [replace_tags(Tupel) for Tupel in brown_train]
```

```
#Diverse Tagger

#1. Tagger vom Typ _neu: verwendet die neuen Tags
regexp_tagger_neu = nltk.RegexpTagger([
    (r'^-?[0-9]+(.[0-9]+)?$', 'card'),  # Kardinalzahlen
    (r'([Tt]he|[Aa]n?)$', 'det'),       # Dets
    (r'.*([aiu]ble)$', 'adj'),          # Adjective
    (r'.*(ness|hood)$', 'noun'),        # de-adjektivische Nomina
    (r'.*ly$', 'adv'),                  # Adverbien
    (r'.*s$', 'noun'),                  # Pluralnomina
    (r'.*(ing|ed)$', 'verb'),           # Verben
    (r'.*','??')
     ])
unireg_tagger_neu = nltk.UnigramTagger(brown_train2, backoff = regexp_tagger_neu)
kombi_tagger_neu = nltk.BigramTagger(brown_train2, backoff = unireg_tagger_neu)

#2. Tagger vom Typ _brown: verwendet die Tags aus dem Brown-Corpus
regexp_tagger_brown = nltk.RegexpTagger([
    (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),  # cardinal numbers
    (r'([Tt]he|[Aa]n?)$', 'AT'),      # article
    (r'.*([aiu]ble)$', 'JJ'),         # adjective
    (r'.*(ness|hood)$', 'NN'),        # de-adjectival nouns singular
    (r'.*ly$', 'RB'),                 # adverbs
    (r'.*s$', 'NNS'),                 # plural nouns
    (r'.*(ed)$', 'VBD'),              # tensed verbs simple past, past participle
    (r'.*(ing)$', 'VBG'),             # tensed verbs present particple
    (r'.*','??')
     ])
unireg_tagger_brown = nltk.UnigramTagger(brown_train, backoff = regexp_tagger_brown)
kombi_tagger_brown = nltk.BigramTagger(brown_train, backoff = unireg_tagger_brown)

#Funktion zum Taggen von Texten
def tagge_text(Text,Tagger):
        Taglistenliste = []
        for element in tokenisiere_text(Text):
            Taglistenliste.append(Tagger.tag(element))
        return Taglistenliste

######################       TEIL : CHUNKING    ########################

#Funktion re_show(RegEx, Zeichenkette), die in Zeichenkette die über RegEx beschriebenen
#Muster erkennt und durch Schweifklammern markiert:
def re_show(pat,s):
    print re.compile(pat,re.M).sub("{\g<0>}",s.rstrip()),'\n'

#Funktion, die das zweite Element eines Tupels zurückgibt:
def hole_tag(Tupel):
    return Tupel[1]

#Funktion, die aus einer Listen von 'Token/Tag'-Tupeln die Tags herauszieht und diese,
durch Schrägstrich getrennt, zu einer Zeichenkette #zusammenfügt. Das letzte Tag schließt
mit Schrägstrich ab, die Satzende-markierung (der Punkt) wird ausgefiltert.
def mache_tagkette(Tupelliste):
    Tagliste = [hole_tag(Tupel) for Tupel in Tupelliste if Tupel[1]!= '.']
    return ('/').join(Tagliste) + '/'

#Alternative Realisierung:
#def mache_tagkette(Tupelliste):
#    return ''.join([Tag+'/' for (Token,Tag) in Tupelliste if Tag !='.'])

#Funktion zum Taggen einzelner, tokenisierter Texte
def tagge_satz(Tokensatz,Tagger):
      return Tagger.tag(Tokensatz)

#Ein RA für NP
NP = r'(name/)|(det/((adv/)*(adj/))*noun/)'

#Funktion zum Ersetzen eines Patterns durch Kategorialsymbol
def ersetze_pattern(Pattern,Ersetzung,Kette):
    return re.sub(Pattern,Ersetzung,Kette)

#Eine Chunk-Parsing-Shift-Reduce Grammatik
Grammatik1 = {'np':'(name/)|(pro/)|(det/(ap/)*noun/)',
              'ap':'(adv/)*adj/',
              'pp':'prep/np/',
              'vp':'verb/(np/|pp/)*',
              's':'np/vp/'}
```

```
#Funktion zum Aufbereiten eines Satzes für das Chunken. Nimmt Satz als Argument,
#normalisiert ihn und gibt Tagkette von Satz zurück
def bereite_auf(Satz):
    Tokensatz = tokenisiere_satz(Satz.lower())
    Tagsatz = tagge_satz(Tokensatz,bi_tagger_neu)
    Tagkette = mache_tagkette(Tagsatz)
    return Tagkette

#Funktion zum Parsen eines Satzes per Chunking durch RA
def parse_satz(Satz):
    Tagkette = bereite_auf(Satz)
    Protokoll = [Tagkette]
    for element in ['ap','np','pp','vp','s']:
        Tagkette = ersetze_pattern(Grammatik1[element],element+'/',Tagkette)
        if Tagkette not in Protokoll:
            Protokoll.append(Tagkette)
    return Protokoll

#Funktion zum Parsen eines vom User eingegebenen Satzes per Chunking durch RA
def parse():
    Satz = raw_input("Bitte einen Satz 'normal'eingeben:\n")
    Reduktionen = parse_satz(Satz)
    for Element in Reduktionen:
        print [Element]
```

'datei.py'

```
#-*- coding: iso-8859-1 -*-

#Methode zum Schreiben eines Objektes als Zeichenkette in eine Datei
def schreibe_datei(Dateiname,Objekt):
    Datei = file(Dateiname,'w')
    Datei.write(repr(Objekt))
    Datei.close()

#Methode zum Einlesen eines Dateiinhaltes.
def hole_datei(Dateiname):
    try:
        Datei = file(Dateiname,'r')
        Objekt = eval(Datei.read())
        Datei.close()
        return Objekt
    except:
        print 'Das Objekt hat keinen Inhalt\n'
```