PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
#let the user enter two strings
a = raw_input()
b = raw_input()
#initialize a matrix for computing the Levenshtein distance
m = []
#go through every cell of the matrix starting with the 1st string
for i in range(0, len(a) + 1):
    #add an empty row to the matrix
    m += [[]]
    #go through the 2nd string
    for j in range(0, len(b) + 1):
        #add an empty cell to the current row
        m[i] += [0]
        if (i > 0 and j > 0):
            #min of: above+1, left+1, above_left+{0,1}
            m[i][j] = min(m[i - 1][j] + 1, m[i][j - 1] + 1,
                          m[i - 1][j - 1] +
                          (0 if (a[i - 1] == b[j - 1]) else 1))
#get the total Levenshtein distance from the bottom-right cell
print m[len(a)][len(b)]
```

</td><td>V1.1</td></tr>
<tr><td>

```
#----
class TmpClass2 (object):
    # i=5: creates 1 class variable!
    # until: self.i=number or x.i=number is called
    # then an additional instance variable is created!
    i = 5
x = TmpClass2()
print "1: ", x.i, TmpClass2.i #1:  5 5
y = TmpClass2()
print "2: ",y.i, TmpClass2.i #2:  5 5
x.i = 10 # change instance variable of x
print "5: ",y.i, TmpClass2.i #5:  5 5
print "6: ", x.i, TmpClass2.i #6:  10 5
TmpClass2.i = 11 # change class variable
print "7: ",y.i, TmpClass2.i #7:  11 11
print "8: ", x.i, TmpClass2.i #8:  10 11
y.i = 7 # change instance variable of y
print "9: ",y.i, TmpClass2.i #9:  7 11
print "10: ", x.i, TmpClass2.i #10:  10 11
print "--"
#----
class TmpClass (object):
    # i=5: creates 1 class variable!
    # until: self.i=number or x.i=number is called
    # then an additional instance variable is created!
    i = 5
    # to change the instance variable only
    def create(self,v):
        self.i = v
x = TmpClass()
print "1: ", x.i, TmpClass.i #1:  5 5
y = TmpClass()
print "2: ",y.i, TmpClass.i #2:  5 5
y.create(2) # change instance variable of y
print "3: ",y.i, TmpClass.i #3:  2 5
print "4: ", x.i, TmpClass.i #4:  5 5
x.i = 10 # change instance variable of x
print "5: ",y.i, TmpClass.i #5:  2 5
print "6: ", x.i, TmpClass.i #6:  10 5
TmpClass.i = 11 # change class variable
print "7: ",y.i, TmpClass.i #7:  2 11
print "8: ", x.i, TmpClass.i #8:  10 11
y.i = 7 # change instance variable of y
print "9: ",y.i, TmpClass.i #9:  7 11
print "10: ", x.i, TmpClass.i #10:  10 11
print "--"
#----
class TmpClass3 (object):
    # i=5: creates 1 class variable!
    # until: self.i=number or x.i=number is called
    # then an additional instance variable is created!
    i = 5
    # to change instance and class variables
    def create(self,v, w):
        self.i = v
        TmpClass3.i = w
        TmpClass.i=99
#print i #NameError 5
x = TmpClass3()
print "1: ", x.i, TmpClass3.i #1:  5 5
y = TmpClass3()
print "2: ",y.i, TmpClass3.i #2:  5 5
y.create(2,3) # change instance variable of y and class variables
print "3: ",y.i, TmpClass3.i, TmpClass.i #3:  2 3 99
print "4: ", x.i, TmpClass3.i, TmpClass.i #4:  3 3 99!!!
x.i = 10 # change instance variable of x
print "5: ",y.i, TmpClass3.i #5:  2 3
print "6: ", x.i, TmpClass3.i #6:  10 3
TmpClass3.i = 11 # change class variable
print "7: ",y.i, TmpClass3.i #7:  2 11
print "8: ", x.i, TmpClass3.i #8:  10 11
```

</td><td>V2.1</td></tr>
</table>

```
y.i = 7
print "9: ",y.i, TmpClass3.i #9:  7 11
print "10: ", x.i, TmpClass3.i #10:  10 11
y.create(2,3) # change instance variable of y and class variables
print "11: ",y.i, TmpClass3.i, TmpClass.i #11:  2 3 99
print "12: ", x.i, TmpClass3.i, TmpClass.i #12:  10 3 99!!!
print "--"
```

```
import nltk, nltk.stem
lem = nltk.stem.WordNetLemmatizer()
class SentenceInfo (object):
    #class variables SentenceInfo.var
    tokList = []
    posList = []
    lemmaList = []
    def create(self, rawSentence):
        # instance/object variable
        self.tokList = rawSentence.split()
        self.posList = nltk.pos_tag(self.tokList)
        self.lemmaList = [lem.lemmatize(tok) for tok in self.tokList]
        # class/static variable
        SentenceInfo.tokList=[["hi"],["there"]]
    # NB! needs to be self.tokList
    def getToken(self, tokIndex):
        return (self.tokList[tokIndex], self.posList[tokIndex],
self.lemmaList[tokIndex])
info = SentenceInfo()
info.create( "here are some tokens to process ." )
print info.getToken(3) #('tokens', ('tokens', 'NNS'), u'token')
print info.tokList #['here', 'are', 'some', 'tokens', 'to', 'process', '.']
print SentenceInfo.tokList #[['hi'], ['there']]
```

V2.2

```
import nltk, nltk.stem
lem = nltk.stem.WordNetLemmatizer()
#parent class
class SentenceInfo(object):
    # class variables SentenceInfo.var
    tokList_ = [], posList_ = [], lemmaList_ = []
    # constructor (some kind of)
    def __init__(self, rawSentence):
        self.tokList = rawSentence.split()
        self.posList = nltk.pos_tag(self.tokList)
        self.lemmaList = [lem.lemmatize(tok) for tok in self.tokList]
#child class
class MoreSentenceInfo(SentenceInfo):
    def __init__(self, raw_sentence, new_info):
        #call super class constructor
        SentenceInfo.__init__(self, raw_sentence)
        self.new_info = new_info
#calls
x = MoreSentenceInfo("here are some tokens to process .",
                     "they are already lower-cased and tokenized .")
SentenceInfo.tokList_="a v e"
#print
print SentenceInfo.tokList_ #a v e
print x.tokList #['here', 'are', 'some', 'tokens', 'to',...]
print x.posList #[('here', 'RB'), ('are', 'VBP'),..]
print x.lemmaList #['here', 'are', 'some', u'token',...]
print x.new_info #they are already lower-cased and tokenized
```

V2.3

```
class TmpClass(object):
    i = 5
    def add(self, arg):
        return self.i + arg
x = TmpClass()
y = TmpClass()
#print i # NameError
print x.i # 5
x.i = 10
print TmpClass.add(x, 2) # 12 = 10 + 2
print TmpClass.add(x, y.i) # 15 = 10 + 5
print y.add(2)# 7 = 5 + 2
print y.add(x.i) # 15 = 5 + 10
TmpClass.i=99 #affects calc
print y.add(2)# 101 = 99 + 2
print y.add(x.i) # 109 = 99 + 10
print TmpClass.i #99
y.i=20 #affects calc
print y.add(2)# 22 = 20 + 2
print y.add(x.i) # 30 = 20 + 10
print TmpClass.i #99
TmpClass.i=50 #no effect anymore!!!
print y.add(2)# 22 = 20 + 2
print y.add(x.i) # 30 = 99 + 10
print y.i #20
```

V2.4

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```python
def f(x):
    print(locals())
x = 20
print (globals()) # {'x': 20, 'f': <function ...>, ...}
f(10) # {'x': 10}
print (globals()) # {'x': 20, 'f': <function ...>, ...}
print x #20
print locals()['x'] #20
locals()['z'] = 3 #z=3
print z # 3
```

V2.5

```python
import os
path='./' #current dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
path='/' #root dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
path='/home/benzro/' #user home dir
for subPath in os.listdir(path):
    print subPath,
    if os.path.isdir(path+subPath):
        print '/'
```

V3.1

```python
import argparse
import sys
#Command line Parser
def parse_command_line():
    parser = argparse.ArgumentParser(
        description=" Prints the first 'num_lines' "
                    " of the file 'src' ")
    parser.add_argument('src',
        type=argparse.FileType('r'), #read file object
        metavar='FILE',
        help='source file');
    parser.add_argument('num_lines',
        help="number of lines to print",
        type=int) #integer object
    parser.add_argument('-o', '--out',
        type=argparse.FileType('w'), #write file object
        default=sys.stdout, #print console object
        metavar='FILE',
        help='out file');
    return parser.parse_args()
#Main
def main(args):
    #argument 2
    for i in range(args.num_lines):
        #argument 3 (args.out) and 1 (args.src)
        args.out.write(args.src.readline())
if __name__ == '__main__':
    #parse arguments
    args = parse_command_line()
    #call main
    main(args)
# Print three lines from one file into another
#$ python V3.2.py
#         '/home/benzro/TextFile.txt'
#         3
#        -o  '/home/benzro/TextFile1.txt'
# Print three lines from file onto console
#$ python V3.2.py
#         '/home/benzro/TextFile.txt'
#         3
```

V3.2

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
a_umlaut = 'ä'
print type(a_umlaut) #<type 'str'>
a_umlaut_unicode = u'\xe4'
print type(a_umlaut_unicode) #<type 'unicode'>
#UTF-8 Byte representation (hexadecimal)
a_umlaut = 'ä'
a_umlaut #'\xc3\xa4'
#Unicode Code point
a_umlaut_unicode = u'\xe4'
a_umlaut_unicode #ä
#map bytes to unicode code points
'\xc3\xa4'.decode('utf-8') # u'\xe4'
unicode('ä', 'utf-8') # u'\xe4'
unicode('\xc3\xa4', 'utf-8') # u'\xe4'
```

V3.3

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
#map unicode code points to bytes
u'\xe4'.encode('utf-8') # '\xc3\xa4'
re.findall("ö", u"Höhentraining")
```

</td><td></td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs
# open as UTF-8
f1 = open('/home/benzro/TextFile.txt', 'r')
lines1 = f1.readlines()
print "1: ", [lines1[0]] #\xc3\xa4\xc3\xb6\xc3\xbc\n
print "2: ", lines1[0] #äöü
line1 = "äöü\n"
print "3: ", [line1] #\xc3\xa4\xc3\xb6\xc3\xbc\n
print "4: ", line1 #äöü
if lines1[0] == line1:
    print "5: ", "true\n\n" #true
# open as Unicode (sandwich)
f2 = codecs.open('/home/benzro/TextFile.txt', 'r', 'UTF-8')
lines2 = f2.readlines()
print "1: ", [lines2[0]] #u'\xe4\xf6\xfc\n'
print "2: ", lines2[0] #äöü
line2 = u'äöü\n' #u'\xe4\xf6\xfc\n'
print "3: ", [line2]
print "4: ", line2 #äöü
if lines2[0] == line2:
    print "5: ", "true\n\n"
# print as UTF-8
print "6: ", [codecs.encode(lines2[0],'UTF-8')] #\xc3\xa4\xc3\xb6\xc3\xbc\n
print "7: ", codecs.encode(lines2[0],'UTF-8') #äöü
```

</td><td>V3.4</td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import codecs
g_in = codecs.open('/home/benzro/TextFile.txt', 'r', 'UTF-8')
g_lines = g_in.readlines()
print g_lines[0],[g_lines[0]] #äöü, [u'\xe4\xf6\xfc\n']
g_o = codecs.open('/home/benzro/TextFile1.txt', 'w', 'UTF-8')
g_o.write(' '.join(g_lines))
#UnicodeDecodeError: 'ascii' codec can't decode
# byte 0xc3 in position 0: ordinal not in range(128)
g_o.write(' '.join(codecs.encode(g_lines[0],'UTF-8')))
g_o.close()
```

</td><td>V3.5</td></tr>
<tr><td>

```
import nltk
b_a=[("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","Wagen"),("der","Wagen"),
     ("der","Wagen"),("der","schnelle"),("der","schnelle"),
     ("der", "Mensch"),("der", "Mensch"),("der", "Mensch"),
     ("schnelle", "Wagen"),("schnelle", "Wagen"),
     ("schnelle", "Wagen"),("schnelle", "Wagen"),
     ("schnelle", "Wagen")]
cfd=nltk.ConditionalFreqDist(b_a)
print cfd
print cfd["der"]
print cfd["der"]["Wagen"]
print sum(cfd["der"].values())
print cfd["der"]["Wagen"]/float(sum(cfd["der"].values()))
cfd.tabulate()
"""
<ConditionalFreqDist with 2 conditions>
<FreqDist with 3 samples and 15 outcomes>
10
15
0.666666666667
         Mensch Wagen schnelle
     der    3   10    2
schnelle    0    5    0
"""
cfd.plot()
```

</td><td>V5.1</td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
from collections import defaultdict
def main():
    text = "This is a sentence . Each sentence has a number of words . " \
           "The number of sentences is 3 ."
    #unigram
    absfreqs=count1_unigram(text)
    freqs=count2_unigram(text)
    # bigram
    f=count_bigrams(text)
    print absfreqs, "\n\n", freqs, "\n\n", f, "\n\n"
```

</td><td>V5.2</td></tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
    print freqs["a"], "\n\n", f["a"], "\n\n", f["a"]["number"]
def count1_unigram(text):
    absfreqs={}
    for word in text.split():
        absfreqs[word]=absfreqs.setdefault(word,0)+1
    return absfreqs
def count2_unigram(text):
    freqs=defaultdict(int)
    for word in text.split():
        freqs[word]+=1
    return freqs
def count_bigrams(text):
    f=defaultdict(lambda: defaultdict(int))
    hist=None
    for word in text.split():
        f[hist][word]+=1
        hist=word
    return f
if __name__ == "__main__":
    main()
```

</td><td></td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import nltk
import math
def entropy(labels):
    # frequency of labels
    freqdist = nltk.FreqDist(labels)
    # plot frequency of labels
    print "freq. distribution:"
    freqdist.tabulate()
    # probability of labels
    #probs = [freqdist.freq(l) for l in nltk.FreqDist(labels)]
    probs=[]
    for label in nltk.FreqDist(labels):
        print "label: ", label
        prob=freqdist.freq(label)#probability
        probs.append(prob)#list of probabilities
        print "probabilities: ", probs
    #entropy
    #entropy = -sum([p * math.log(p, 2) for p in probs])
    entropy=0
    for prob in probs:
        log = math.log(prob, 2)
        entropy_i = -prob * log
        entropy+=entropy_i
        print "prob, log, entropy", prob, log, entropy_i
    #return
    return entropy
print "no variety: low entropy (high discriminatory power)"
print entropy(['male', 'male', 'male', 'male'])#0.0
print
print "little variaty: medium entropy"
print entropy(['male', 'female', 'male', 'male'])#0.811
print
print "max variety: high entropy (no discriminatory power)"
print entropy(['female', 'male', 'female', 'male'])#1.0
print
print "little variaty: medium entropy"
print entropy(['female', 'female', 'male', 'female'])#0.811
print
print "no variety: low entropy (high discriminatory power)"
print entropy(['female', 'female', 'female', 'female'])#0.0
```

</td><td>V6.1</td></tr>
<tr><td>

```
import nltk
tokenList = \
    nltk.word_tokenize( "Fed raises interest rates 0.5 percent" )
print tokenList
# ['Fed', 'raises', 'interest', 'rates', '0.5', 'percent']
posResult = nltk.pos_tag(tokenList)
print posResult
# [('Fed', 'NNP'), ('raises', 'VBZ'), ('interest', 'NN'),
# ('rates', 'NNS'), ('0.5', 'CD'), ('percent', 'NN')]
print posResult[0], posResult[0][1]
# posResult[0] = ('Fed', 'NNP')
# posResult[0][1] = 'NNP'
```

</td><td>V7.1</td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import nltk.corpus
print nltk.corpus.brown.tagged_words()
# [('The', 'AT'), ('Fulton', 'NP-TL'), ...]
print nltk.corpus.brown.tagged_words(tagset='universal')
# [('The', 'DET'), ('Fulton', 'NOUN'), ...]
print nltk.corpus.treebank.tagged_words()
# [('Pierre', 'NNP'), ('Vinken', 'NNP'), ...]
print nltk.corpus.treebank.tagged_words(tagset= 'universal')
```

</td><td>V7.2</td></tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td><pre># [('Pierre', 'NOUN'), ('Vinken', 'NOUN'), ...]</pre></td><td></td></tr>
<tr><td><pre>import nltk
from nltk.tag import *
train_sents=nltk.corpus.brown.tagged_sents()[:2000]
sents=nltk.corpus.brown.tagged_sents()
default_tagger = nltk.DefaultTagger("NN")
unigram_tagger = nltk.UnigramTagger(train_sents,
                            backoff=default_tagger)
bigram_tagger = nltk.BigramTagger(train_sents,
                            backoff=unigram_tagger)
print bigram_tagger.evaluate(sents[2000:2100])
print bigram_tagger.tag(untag(sents[2007]))
print bigram_tagger.tag(untag(sents[4203]))</pre></td><td>V7.3</td></tr>
<tr><td><pre>import nltk
from nltk.tag import *
from nltk.tag.hmm import HiddenMarkovModelTagger
train_sents=nltk.corpus.brown.tagged_sents()[:2000]
sents=nltk.corpus.brown.tagged_sents()
hmm_tagger = HiddenMarkovModelTagger.train(train_sents)
print hmm_tagger.tag(untag(sents[4203]))</pre></td><td>V7.4</td></tr>
<tr><td><pre>import nltk
from nltk.tag import *
regexp_tagger = nltk.RegexpTagger(
 [(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),# cardinal numbers
 (r'(The|the|A|a|An|an)$', 'AT'),# articles
 (r'.*able$', 'JJ'),# adjectives
 (r'.*ness$', 'NN'),# nouns formed from adjectives
 (r'.*ly$', 'RB'),# adverbs
 (r'.*s$', 'NNS'),# plural nouns
 (r'.*ing$', 'VBG'),# gerunds
 (r'.*ed$', 'VBD'),# past tense verbs
 (r'.*', 'NN')])# nouns (default)
sents=nltk.corpus.brown.tagged_sents()
print regexp_tagger.tag(untag(sents[4203]))</pre></td><td>V7.5</td></tr>
<tr><td><pre>#!/usr/bin/env python
# -*- coding: UTF-8 -*-
import nltk
from nltk.tag import *
from nltk.tag.hmm import HiddenMarkovModelTagger
def tagList(sents):
    '''remove tokens and leave only tags'''
    return [tag for sent in sents for word, tag in sent]
def applyTagger(tagger, corpus):
    '''apply a tagger to a corpus'''
    return [tagger.tag(untag(sent)) for sent in corpus]
train_sents=nltk.corpus.brown.tagged_sents()[:2000]
test_sents=nltk.corpus.brown.tagged_sents()[2000:2100]
hmm_tagger = HiddenMarkovModelTagger.train(train_sents)
goldTags = tagList(test_sents)
testTags = tagList(applyTagger(hmm_tagger, test_sents))
cm = nltk.ConfusionMatrix(goldTags, testTags)
print cm.pretty_format(sort_by_count= True,
                       show_percents= True, truncate=9)</pre></td><td>V7.6</td></tr>
<tr><td><pre>from nltk.metrics import ConfusionMatrix
ref   = 'DET NN VB DET JJ NN NN IN DET NN'.split()
tagged = 'DET VB VB DET NN NN NN IN DET NN'.split()
cm = ConfusionMatrix(ref, tagged)
print cm.pretty_format(show_percents= True, truncate=9)</pre></td><td>V7.7</td></tr>
<tr><td><pre>import subprocess
Update the access and modification times
of each FILE to the current time.
subprocess.call(["touch",
                 "/home/benzro/PycharmProjects/Lectures_PCL2/"
                 "TextFile.txt"])</pre></td><td>V8.1</td></tr>
<tr><td><pre>import subprocess
# prints a list of files with mod. dates, sizes,etc.
out=subprocess.check_output(["ls", "-l", "/home/benzro/"])
print out</pre></td><td>V8.2</td></tr>
</table>

```python
import nltk
grammar1 = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "Mary saw Bob".split()
parser = nltk.parse.chart.BottomUpChartParser(grammar1)
for tree in parser.parse(sent):
    print tree #(S (NP Mary) (VP (V saw) (NP Bob)))
```

V12.1

```python
import nltk
grammar2 = nltk.PCFG.fromstring("""
S -> NP VP [1.0]
VP -> V NP [0.217]
VP -> V NP PP [0.53]
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "Mary saw Bob".split()
viterbi_parser = nltk.ViterbiParser(grammar2)
for tree in viterbi_parser.parse(sent):
    print tree
```

V12.2

```python
from nltk.corpus import treebank
print treebank.sents()[314] #['The', 'offer', 'is', 'being', 'launched', ...]
print treebank.parsed_sents()[314]
#Tree('S', [Tree('NP-SBJ-45', [Tree('DT', ['The']), Tree
#('NN', ['offer'])]), Tree('VP', [Tree('VBZ', ['is']), Tree
#('VP', [Tree('VBG', ['being']), ...)
print treebank.parsed_sents()[314].productions()[4] #VP -> VBZ VP
print treebank.parsed_sents()[314].productions()[4].lhs() #VP
print treebank.parsed_sents()[314].productions()[4].rhs() #(VBZ, VP)
```

V12.3

```python
from nltk.corpus import treebank
from nltk.grammar import CFG, Nonterminal
tbank_productions = set(production for sent in
            treebank.parsed_sents() for production in
            sent.productions())
tbank_grammar = CFG(Nonterminal('S'), list(tbank_productions))
print tbank_grammar
```

V12.4

Grammar Learning

```python
import nltk
from nltk.corpus import treebank
from nltk.grammar import Nonterminal
tbank_production_list = [production for sent in
            treebank.parsed_sents() for production in
            sent.productions()]
tbank_prob_grammar = nltk.induce_pcfg(Nonterminal('S'),
                        tbank_production_list)
print tbank_prob_grammar
```

V12.5

Grammar Learning

```python
#!/usr/bin/python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import treebank
from nltk.grammar import CFG, Nonterminal
from nltk.tree import Tree
import sys
import os
def loadMap(filename):
    fileHandle = open(filename, 'r')
    resultMap = dict()
    for line in fileHandle:
        (fullTag, shortTag, desc) = line.rstrip().split(" # ")
        resultMap[fullTag] = shortTag
    fileHandle.close()
    return resultMap
def getListOfPos(rawSnt):
   global mapOfPos
   #return [mapOfPos[tag] if tag in mapOfPos else 'NOUN'
   #     for tag in zip(*rawSnt)[1]]
   ret = []
   tags = zip(*rawSnt)[1]
```

V12.6

Grammar Learning

File:'*' # . # 'added'
-NONE- # . # 'added'
`` # . # 'added'
, # . # 'added'
. # . # 'added'
CC # CONJ # 'Coordinating conjunction'
CD # NOUN # 'Cardinal number'
DT # DET # 'Determiner'
DTS # DET # 'added'
DTP # DET # 'added'
EX # X # 'Existential there'
FW # X # 'Foreign word'
IN # ADP # 'Preposition or subordinating conjunction'
JJ # ADJ # 'Adjective'
JJR # ADJ # 'Adjective, comparative'

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    print "10.1", tags
    for tag in tags:
        if tag in mapOfPos:
            terminal = mapOfPos[tag]
            ret.append(terminal)
        else:
            terminal = 'NOUN'
            ret.append(terminal)
    print "10.2", ret
    return ret
def changeLeaves(parent, root):
    """http://www.nltk.org/_modules/nltk/tree.html"""
    #if parent.label() == ROOT:
        #print "1.1:\n", "======== Sentence ========="
    for node in parent:
        if type(node) is nltk.Tree:
            #print "1.2:\n","Label:", node.label().upper()
            #print "1.3:\n","Leaves:", node.leaves(), "\n"
            if len(node.leaves()) == 1:
                #print "ho"
                # change terminal in tree
                if node.label().upper() in mapOfPos:
                    #print "hi"
                    index=parent.index(node)
                    #print parent
                    #print root
                    node.set_label(mapOfPos[node.label().upper()])
                    #print root
                    parent.remove(node)
                    #print root
                    #newnode=("label" ,node.label().upper())
                    #newnode=Tree(node.label().upper(), [node.label().upper()])
                    newnode=node.label().upper()
                    parent.insert(index, newnode)
                    #print root
                else:
                    #terminal = 'NOUN'
                    #index = parent.index(node)
                    #parent.remove(node)
                    #newnode = terminal
                    #parent.insert(index, newnode)
                    pass
                changeLeaves(node,root)  # depth first recursion
            else:
                changeLeaves(node,root) #depth first recursion
        else:
            #print "1.4:\n","Word:", node, "\n"
            pass
def changeLabels(parent, root):
    for node in parent:
        if type(node) is nltk.Tree:
            if len(node.leaves()) == 1:
                if not node.label().upper() in mapOfPos:
                    #print "11.1",node.label().upper()
                    pass
            else:
                changeLabels(node, root) # depth first recursion
        try:
            if "-" in node.label():
                #print "11.2", node.label().upper()
                node.set_label(node.label().upper()
                        [0:node.label().upper().find("-")])
                #print "11.3", node.label().upper()
        except:pass
    if "-" in parent.label():
        #print "11.4", parent.label().upper()
        parent.set_label(parent.label().upper()
                        [0:parent.label().upper().find("-")])
        #print "11.5", parent.label().upper()
"""
        I{corpus}.words(): list of str
        I{corpus}.sents(): list of (list of str)
        I{corpus}.paras(): list of (list of (list of str))
        I{corpus}.tagged_words(): list of (str,str) tuple
        I{corpus}.tagged_sents(): list of (list of (str,str))
        I{corpus}.tagged_paras(): list of (list of (list of (str,str)))
        I{corpus}.chunked_sents(): list of (Tree w/ (str,str) leaves)
        I{corpus}.parsed_sents(): list of (Tree with str leaves)
        I{corpus}.parsed_paras(): list of (list of (Tree with str leaves))
        I{corpus}.xml(): A single xml ElementTree
        I{corpus}.raw(): str (unprocessed corpus contents)
"""
if __name__ == "__main__":
    s=0
    n=15
    #treebank
    tb_sents=nltk.corpus.treebank.sents()[s:n]
    tb_raw = " ".join(map(str,tb_sents))
    tb_tagged = nltk.corpus.treebank.tagged_sents()[s:n]
    tb_parsed = nltk.corpus.treebank.parsed_sents()[s:n]
    #tb_chunked=nltk.corpus.treebank.chunked_sents()[s:n]
    tb_chunked = [nltk.ne_chunk(s, binary=True) for s in tb_tagged]
```

```
JJS # ADJ # 'Adjective, superlative'
LS # X # 'List item marker'
MD # VERB # 'Modal'
NN # NOUN # 'Noun, singular or mass'
NNS # NOUN # 'Noun, plural'
NNP # NOUN # 'Proper noun, singular'
NNPS # NOUN # 'Proper noun, plural'
PDT # DET # 'Predeterminer'
POS # X # 'Possessive ending'
PRP # PRON # 'Personal pronoun'
PRP$ # PRON # 'Possessive pronoun'
RB # ADV # 'Adverb'
RBR # ADV # 'Adverb, comparative'
RBS # ADV # 'Adverb, superlative'
RP # PRT # 'Particle'
SYM # X # 'Symbol'
TO # ADP # 'to'
UH # X # 'Interjection'
VB # VERB # 'Verb, base form'
VBD # VERB # 'Verb, past tense'
VBG # VERB # 'Verb, gerund or present participle'
VBN # VERB # 'Verb, past participle'
VBP # VERB # 'Verb, non-3rd person singular present'
VBS # VERB # 'added'
VBZ # VERB # 'Verb, 3rd person singular present'
WDT # DET # 'Wh-determiner'
WP # PRON # 'Wh-pronoun'
WP$ # PRON # 'Possessive wh-pronoun'
WRB # ADV # 'Wh-adverb' TreeTag.map
```

```
    print "1.1: treebank\nraw", tb_raw,"\nlist", tb_sents,\
        "\ntagged", tb_tagged, "\nparsed", tb_parsed, "\nchunked", tb_chunked
    #brown
    br_sents=nltk.corpus.brown.sents()[:n]
    br_raw = " ".join(map(str, br_sents))
    br_tagged=nltk.corpus.brown.tagged_sents()[:n]
    #br_parsed=nltk.corpus.brown.parsed_sents()[:n]
    #br_chunked = nltk.corpus.brown.chunked_sents()[:n]
    #br_chunked = nltk.ne_chunk(br_tagged[0], binary=True)
    br_chunked=[nltk.ne_chunk(s, binary=True) for s in br_tagged]
    # for s in br_tagged:
    #     br_chunked.append(nltk.ne_chunk(s, binary=True))
    print "1.2: brown\nraw", br_raw, "\nlist", br_sents, \
        "\ntagged", br_tagged, "\nchunked", br_chunked

    #conll
    co_sents = nltk.corpus.conll2002.sents()[:n]
    co_raw = " ".join(map(str, co_sents))
    co_tagged = nltk.corpus.conll2002.tagged_sents()[:n]
    #co_parsed = nltk.corpus.conll2002.parsed_sents()[:n]
    co_chunked = nltk.corpus.conll2002.chunked_sents()[:n]
    print "1.3: conll\nraw", co_raw, "\nlist", co_sents, \
        "\ntagged", co_tagged, "\nchunked", co_chunked

    #genesis
    ge_sents = nltk.corpus.genesis.sents()[:n]
    ge_raw = " ".join(map(str, ge_sents))
    ge_tagged = [nltk.pos_tag(s) for s in ge_sents]
    ge_chunked=[nltk.ne_chunk(s, binary=True)for s in ge_tagged]
    print "1.4: genesis\nraw", ge_raw, "\nlist", ge_sents, \
        "\ntagged", ge_tagged, "\nchunked", ge_chunked


    treebank create grammar
    for t in tb_parsed:
        print "2.1: ", t

    tb_productions = set(production
                         for sent in tb_parsed
                         for production in sent.productions())

    tb_grammar = CFG(Nonterminal('S'), list(tb_productions))
    print "2.2: ",tb_grammar
    # treebank set of pos tags
    tb_tagset=list()
    for sent in tb_tagged:
        #print "3.11: ", sent
        for word in sent:
            tag = word[1]
            #print "3.12: ", tag
            tb_tagset.append(tag)
    tb_tagset=set(tb_tagset)
    print "3.1:\n", tb_tagset
    # mapping of treebank-tag_set -> exam-tag_set
    """'ADV', 'NOUN', 'NUM', 'ADP', 'PRON', 'DET', '.',
         'PRT', 'VERB', 'X', 'CONJ', 'ADJ'"""
    mapOfPos = loadMap(os.path.join(os.path.dirname
                                    (os.path.realpath(__file__)),
                                    'TreeTag.map'))
    print "4.1:\n", mapOfPos
    # delete trees with strange tags
    for t in tb_parsed:
        changeLabels(t, t)
    # adapt leave node tags in parsed tree
    for t in tb_parsed:
        print "5.1: ",t
    ROOT = 'S'
    for t in tb_parsed:
        changeLeaves(t,t)
    for t in tb_parsed:
        print "5.2: ", t
    # new productions
    tb_productions = set(production
                         for sent in tb_parsed
                         for production in sent.productions())
    tb_grammar = CFG(Nonterminal('S'), list(tb_productions))
    print "6.1: ", tb_grammar
```

```
# NLTK can process logical expressions into subclasses of Expression object
import nltk
from nltk.sem.logic import LogicParser
lp = LogicParser()
print lp.parse('-(P & Q)') #<NegatedExpression -(P & Q)>
print lp.parse('P & Q') #<AndExpression (P & Q)>
print lp.parse('P | (R -> Q)') #<OrExpression (P | (R -> Q))>
print lp.parse('P <-> -- P') #<IffExpression (P <-> --P)>
#Assigning values:
val = nltk.Valuation([('P', True), ('Q', True), ('R', False)])
print val, val['P'] #{'Q': True, 'P': True, 'R': False}, True
#Evaluating expressions:
```

V13.1

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
dom = set([])
g = nltk.Assignment(dom)
m = nltk.Model(dom, val)
print m.evaluate('(P & Q)', g) #True
print m.evaluate('-(P & Q)', g) #False
print m.evaluate('(P | R)', g) #True
print m
```

</td><td></td></tr>
<tr><td>

```
from nltk.sem.logic import LogicParser
lp = LogicParser()
p = lp.parse('read(john)')
print p.argument #<ConstantExpression john>
print p.function #<ConstantExpression read>
```

</td><td>V13.2</td></tr>
<tr><td>

```
from nltk.corpus import wordnet as wn
print "Senses"
print wn.synsets('bank')
print "\nSenses for a given PoS-tag only"
print wn.synsets('bank', pos=wn.VERB)
print "\n"
print wn.synsets('bank')[1]
print "\nSense definitions"
print wn.synsets('bank')[1].definition()
print "\nlemmas"
print wn.synsets('bank')[1].lemmas()
print "\nlemma names"
print wn.synsets('bank')[1].lemma_names()
print "\n"
print wn.synsets('bank')[0]
print wn.synsets('bank')[0].definition()
print "\nother languages"
#print wn.langs()
print "\n"
#print wn.synsets('fromage')
print "\n"
#print wn.synsets('fromage', lang='fra')
print "\n"
dog = wn.synset('dog.n.01')
print dog.hypernyms()
print "\n"
print dog.hyponyms()
print "\n"
print dog.member_holonyms()
print "\n"
print dog.root_hypernyms()
print "\n"
good = wn.synset('good.a.01' )
print "\n"
print good.lemmas()
print "\n"
print good.lemmas()[0].antonyms()
dog = wn.synset('dog.n.01')
cat = wn.synset('cat.n.01')
print dog.path_similarity(cat)
print wn.path_similarity(dog, cat)
for s in wn.synsets('wolf'):
    print
    print s.lemma_names()
    print s.definition()
    print dog.lowest_common_hypernyms(s)
    print dog.path_similarity(s)
print dog.lowest_common_hypernyms(cat)
print dog.lowest_common_hypernyms(wn.synset('wolf.n.01'))
print dog.lowest_common_hypernyms(wn.synset('wolf.n.03'))
```

</td><td>V13.3</td></tr>
<tr><td>

```
from nltk.wsd import lesk
from nltk import word_tokenize
from nltk.corpus import wordnet as wn
sent = word_tokenize("I went to the bank to deposit money.")
word = "bank"
pos = "n"
print(lesk(sent, word, pos)) #Synset('savings_bank.n.02')
for s in wn.synsets('bank'):
    print
    print s.lemma_names()
    print s.definition()
for s in wn.synsets('deposit'):
    print
    print s.lemma_names()
    print s.definition()
for s in wn.synsets('money'):
    print
    print s.lemma_names()
    print s.definition()
```

</td><td>V13.4</td></tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
import nltk
from nltk.corpus import conll2000
from nltk.tag.hmm import HiddenMarkovModelTagger as HmmTagger
test_sents = conll2000.chunked_sents('test.txt',
                                     chunk_types=['NP'])
print test_sents[:10]
cp = nltk.RegexpParser(r"NP: {<[CDJNP].*>+}")
print cp.evaluate(test_sents)
train_sents = [ [(t, c if c[-2:] == 'NP' else 'O')
                for w, t, c in snt]
                for snt in conll2000.iob_sents('train.txt')]
print train_sents[:10]
defTagger = nltk.DefaultTagger('O')
uniTagger = nltk.UnigramTagger(train_sents, backoff=defTagger)
biTagger = nltk.BigramTagger(train_sents, backoff=uniTagger)
hmm_tagger = HmmTagger.train(train_sents)
test_sents_iob = [ [(t, c if c[-2:] == 'NP' else 'O')
                    for w, t, c in snt]
                    for snt in conll2000.iob_sents('test.txt')]
print test_sents_iob[:10]
print defTagger.evaluate(test_sents_iob)#0.43436688688604175
print uniTagger.evaluate(test_sents_iob)#0.8321126284906178
print biTagger.evaluate(test_sents_iob)#0.9341663676467484
print hmm_tagger.evaluate(test_sents_iob)#0.9360449163096017
```

</td><td>V14.1</td></tr>
<tr><td>

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
import nltk
l=[("a", "IN","Atlanta"),("b", "IN","Zurich")]
print [e1 for (e1, rel, e2) in l if rel== 'IN' and e2 == 'Atlanta']
def ie_preprocess(document):
    """To perform the first three tasks, we can define a function that simply connects
together
NLTK’s default sentence segmenter , word tokenizer , and part-of-speech tagger """
    sentences = nltk.sent_tokenize(document)
    print "1: ",sentences
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    print "2: ",sentences
    sentences = [nltk.pos_tag(sent) for sent in sentences]
    print "3: ",sentences
if __name__ == '__main__':
    document="""Next, in named entity recognition, we segment and label the entities
that might par-
ticipate in interesting relations with one another. Typically, these will be definite
noun
phrases such as the knights who say "ni", or proper names such as Monty Python. In
some tasks it is useful to also consider indefinite nouns or noun chunks, such as every
student or cats, and these do not necessarily refer to entities in the same way as
definite
NPs and proper names. Finally, in relation extraction, we search for specific patterns
between pairs of entities
that occur near one another in the text, and use those patterns to build tuples
recording
the relationships between the entities.
"""
    ie_preprocess(document)
```

</td><td>B7.2.1</td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
def ie_preprocess(document):
    """To perform the first three tasks, we can define a function that simply connects
together
NLTK’s default sentence segmenter , word tokenizer , and part-of-speech tagger """
    sentences = nltk.sent_tokenize(document)
    print "1: ",sentences
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    print "2: ",sentences
    sentences = [nltk.pos_tag(sent) for sent in sentences]
    print "3: ",sentences
    return sentences
def regex_chunk_parser(grammar, sentences):
    cp = nltk.RegexpParser(grammar)
    result = cp.parse(sentence)
    print "4: ", result
    result.draw()
if __name__ == '__main__':
    document="""the little yellow dog barked at the cat"""
    sentences = ie_preprocess(document)
    #Example 7-1. Example of a simple regular expression–based NP chunker.
    sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
     ("dog", "NN"), ("barked", "VBD"), ("at", "IN"), ("the", "DT"),
     ("cat", "NN")]
    """The rules that make up a chunk grammar use tag patterns to describe sequences of
    tagged words. A tag pattern is a sequence of part-of-speech tags delimited using
angle
    brackets"""
    """This rule says that an NP chunk should be formed whenever the
    chunker finds an optional determiner (DT)
```

</td><td>B7.2.2</td></tr>
</table>

```
    followed by any number of adjectives (JJ)
    and then a noun (NN)."""
    grammar = "NP: {<DT>?<JJ>*<NN>}"
    regex_chunk_parser(grammar, sentence)
    sentence=[("his", "PRP$"), ("Mansion", "NNP"), ("House", "NNP"),
              ("speech", "NN"), ("the", "DT"), ("price", "NN"),
              ("cutting", "VBG"), ("3", "CD"), ("%", "NN"),
              ("to", "TO"), ("4", "CD"), ("%", "NN"), ("more", "JJR"),
              ("than", "IN"), ("10", "CD"), ("%", "NN"),
              ("the", "DT,"), ("fastest", "JJS"), ("developing", "VBG"),
              ("trends", "NNS"), ("'s", "POS"), ("skill", "NN")]
    """This will chunk any sequence of tokens beginning
    with an optional determiner, followed by zero or more adjectives of any type
(including
    relative adjectives like earlier/JJR), followed by one or more nouns of any type."""
    grammar2="NP: {<DT>?<JJ.*>*<NN.*>+}"
    regex_chunk_parser(grammar2, sentence)
```

B7.2.3

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
def ie_preprocess(document):
    """To perform the first three tasks, we can define a function that simply connects
together
NLTK's default sentence segmenter , word tokenizer , and part-of-speech tagger """
    sentences = nltk.sent_tokenize(document)
    print "1: ",sentences
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    print "2: ",sentences
    sentences = [nltk.pos_tag(sent) for sent in sentences]
    print "3: ",sentences
    return sentences
def regex_chunk_parser(grammar, sentences):
    cp = nltk.RegexpParser(grammar)
    result = cp.parse(sentence)
    print "4: ", result
    result.draw()
if __name__ == '__main__':
    #Example 7-2. Simple noun phrase chunker.
    """The chunking rules are applied in turn,
    successively updating the chunk structure."""
    """The $ symbol is a special character in regular expressions, and must be
    backslash escaped in order to match the tag PP$.
    """
    grammar = r"""
                NP: {<DT|PP\$>?<JJ>*<NN>} #chunk determiner/possessive, adjectives and
nouns
                    {<NNP>+} # chunk sequences of proper nouns
                """
    sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),
    ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair", "NN")]
    regex_chunk_parser(grammar, sentence)
```

B7.2.4

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
if __name__ == '__main__':
    #grammar="CHUNK: {<V.*> <TO> <V.*>}"
    #grammar="NP: {<DT>? <JJ>* <NN>}" #Spaces dazwischen
    cp = nltk.RegexpParser(grammar)
    brown = nltk.corpus.brown
    for sent in brown.tagged_sents()[:50]:
        tree = cp.parse(sent)
        for subtree in tree.subtrees():
            if subtree.label() == 'CHUNK':
                print subtree
```

B7.2.5

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
if __name__ == '__main__':
    #Example 7-3. Simple chinker.
    grammar = r"""
            NP:{<.*>+} # Chunk everything
            }<VBD|IN>+{ # Chink sequences of VBD and IN
            """
    sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
        ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),
            ("the", "DT"), ("cat", "NN")]
    cp = nltk.RegexpParser(grammar)
    print cp.parse(sentence)
```

PCL II – Zusammenfassung Code

Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

B7.3.1

```python
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import conll2000
"""A conversion function chunk.conllstr2tree() builds a tree representation from one of
these multiline strings. Moreover, it permits us to choose any subset of the three chunk
types to use, here just for NP chunks:"""
if __name__ == '__main__':
    text = r"""
    he PRP B-NP
    accepted VBD B-VP
    the DT B-NP
    position NN I-NP
    of IN B-PP
    vice NN B-NP
    chairman NN I-NP
    of IN B-PP
    Carlyle NNP B-NP
    Group NNP I-NP
    , , O
    a DT B-NP
    merchant NN I-NP
    banking NN I-NP
    concern NN I-NP
    . . O
    """
    #nltk.chunk.conllstr2tree(text).draw()
    t=conll2000.chunked_sents('train.txt')[99]
    print t
    t=conll2000.chunked_sents('train.txt', chunk_types=['NP'])[99]
    print t
```

B7.3.2

```python
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import conll2000
#Example 7-4. Noun phrase chunking with a unigram tagger.
class UnigramChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
        """The constructor expects a list of training sentences, which will be in the form of
           chunk trees. It first converts training data to a form that’s suitable for training the tagger,
           using tree2conlltags to map each chunk tree to a list of word,tag,chunk triples. It then
           uses that converted training data to train a unigram tagger, and stores it in self.tag
           ger for later use.
           """
        self.train_data = [[(t,c) for w,t,c in nltk.chunk.tree2conlltags(sent)]
                        for sent in train_sents]
        print "4: ", nltk.chunk.tree2conlltags(train_sents[0])
        #self.tagger = nltk.UnigramTagger(self.train_data)
        self.tagger = nltk.BigramTagger(self.train_data)
    def parse(self, sentence):
        """The parse method takes a tagged sentence as its input, and begins by extracting the
           part-of-speech tags from that sentence. It then tags the part-of-speech tags with IOB
           chunk tags, using the tagger self.tagger that was trained in the constructor. Next, it
           extracts the chunk tags, and combines them with the original sentence, to yield
           conlltags. Finally, it uses conlltags2tree to convert the result back into a chunk tree.
           """
        pos_tags = [pos for (word,pos) in sentence]
        tagged_pos_tags = self.tagger.tag(pos_tags)
        chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
        conlltags = [(word, pos, chunktag) for ((word,pos),chunktag)
                     in zip(sentence, chunktags)]
        return nltk.chunk.conlltags2tree(conlltags)
if __name__ == '__main__':
    test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
    print "1: ", test_sents[:2]
    train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
    print "2: ", train_sents[:2]
    unigram_chunker = UnigramChunker(train_sents)
    print "3: ", unigram_chunker.train_data[:2]
    print unigram_chunker.evaluate(test_sents)
    """Let’s take a look at what it’s learned, by using its unigram
    tagger to assign a tag to each of
    the part-of-speech tags that appear in the corpus"""
    postags = sorted(set(pos for sent in train_sents
                     for (word, pos) in sent.leaves()))
    print "5: ",unigram_chunker.tagger.tag(postags)
    for sent in train_sents[:1]:
        print "6: ", sent.leaves()
        print "7: ", sent
        print "8: ", train_sents[:1]
```

B7.3.3

```python
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import conll2000
"""The first class is almost identical to the ConsecutivePosTagger class
from Example 6-5. The only two differences are that it calls a different feature
extractor
and that it uses a MaxentClassifier rather than a NaiveBayesClassifier . The sec-
ond class is basically a wrapper around the tagger class that turns it into a chunker.
```

```
During training, this second class maps the chunk trees in the training corpus into tag
sequences; in the parse() method, it converts the tag sequence provided by the tagger
back into a chunk tree.
"""
#Example 7-5. Noun phrase chunking with a consecutive classifier.
class ConsecutiveNPChunkTagger(nltk.TaggerI):
    def __init__(self, train_sents):
        train_set = []
        for tagged_sent in train_sents:
            untagged_sent = nltk.tag.untag(tagged_sent)
            history = []
            for i, (word, tag) in enumerate(tagged_sent):
                featureset = self.npchunk_features(untagged_sent, i, history)
                train_set.append( (featureset, tag) )
                history.append(tag)
        print "5: ", train_sents[:2]
        print "6: ", tagged_sent
        print "7: ", untagged_sent
        print "8: ", history
        print "9: ", featureset
        print "10: ", train_set[:2]
        #self.classifier = nltk.MaxentClassifier.train(
        #        train_set, algorithm='megam', trace=0)
        # self.classifier = nltk.MaxentClassifier.train(
        #                 train_set, trace=0)
        self.classifier = nltk.classify.NaiveBayesClassifier.\
                          train(train_set)
    def tags_since_dt_(self, sentence, i):
        """"past tags since last determiner (without i)"""
        tags = set()
        for word, pos in sentence[:i]:
            if pos == 'DT':
                tags = set()
            else:
                tags.add(pos) #set member add()
        print "15: ", tags
        return '+'.join(sorted(tags)) #list/set to string
    def npchunk_features(self, sentence, i, history):
        """We begin by defining a simple
           feature extractor, which just provides the part-of-speech tag of the current token.
           We can also add a feature for the previous part-of-speech tag. Adding this feature allows
           the classifier to model interactions between adjacent tags, and results in a chunker that
           is closely related to the bigram chunker.
           Next, we'll try adding a feature for the current word, since we hypothesized that word
           content should be useful for chunking.
           Finally, we can try extending the feature extractor with a variety of additional features,
           such as lookahead features , paired features , and complex contextual features .
           This last feature, called tags-since-dt, creates a string describing the set of all part-of-
            speech tags that have been encountered since the most recent determiner.
           """
        word, pos = sentence[i]
        if i == 0:
            prevword, prevpos = "<START>", "<START>"
        else:
            prevword, prevpos = sentence[i - 1]
        if i == len(sentence) - 1:
            nextword, nextpos = "<END>", "<END>"
        else:
            nextword, nextpos = sentence[i + 1]
        t_s_dt = self.tags_since_dt_(sentence, i)
        try:
            print "11: ", sentence[i - 1], prevword, prevpos
            print "12: ", sentence[i], word, pos
            print "13: ", sentence[i+1], nextword, nextpos
            print "14: ", t_s_dt
        except: pass
        return {"pos": pos,
                "word": word,
                "prevpos": prevpos,
                "nextpos": nextpos,
                "prevpos+pos": "%s+%s" % (prevpos, pos),
                "pos+nextpos": "%s+%s" % (pos, nextpos),
                "tags-since-dt": t_s_dt}
    def tag(self, sentence):
        history = []
        for i, word in enumerate(sentence):
            featureset = \
                self.npchunk_features(sentence, i,history)
            tag = self.classifier.classify(featureset)
            history.append(tag)
        zp = zip(sentence, history)
        print "16: ", sentence
        print "17: ", featureset
        print "18: ", zp
        return zp
class ConsecutiveNPChunker(nltk.ChunkParserI):
    def __init__(self, train_sents):
        tagged_sents = [[((w,t),c) for (w,t,c) in
                        nltk.chunk.tree2conlltags(sent)]
                        for sent in train_sents]
        print "3: ", train_sents[:2]
        print "4: ", tagged_sents[:2]
        self.tagger = ConsecutiveNPChunkTagger(tagged_sents)
    def parse(self, sentence):
        tagged_sents = self.tagger.tag(sentence)
        conlltags = [(w,t,c) for ((w,t),c) in tagged_sents]
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
        print "21: ", conlltags[:5]
        return nltk.chunk.conlltags2tree(conlltags)
if __name__ == '__main__':
    test_sents = conll2000.chunked_sents('test.txt',
                                         chunk_types=['NP'])
    print "1: ", len(test_sents), test_sents[:2]
    train_sents = conll2000.chunked_sents('train.txt',
                                          chunk_types=['NP'])
    print "2: ", len(train_sents), train_sents[:2]
    chunker = ConsecutiveNPChunker(train_sents[:5])
    print "done"
    # calls parse for each sentence
    print chunker.evaluate(test_sents[:2])
```

</td><td></td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
#Example 7-6. A chunker that handles NP, PP, VP, and S.
grammar = r"""
NP: {<DT|JJ|NN.*>+} # Chunk sequences of DT, JJ, NN
PP: {<IN><NP>} # Chunk prepositions followed by NP
VP: {<VB.*><NP|PP|CLAUSE>+$} # Chunk verbs and their arguments
CLAUSE: {<NP><VP>} # Chunk NP, VP
"""
sentence = [("Mary", "NN"), ("saw", "VBD"), ("the", "DT"),
            ("cat", "NN"),("sit", "VB"), ("on", "IN"),
            ("the", "DT"), ("mat", "NN")]
"""Unfortunately this result misses the VP headed by saw. It has other shortcomings,
too.
Let’s see what happens when we apply this chunker to a sentence having deeper nesting.
"""
cp = nltk.RegexpParser(grammar)
print cp.parse(sentence)
"""The solution to these problems is to get the chunker to loop over its patterns: after
trying all of them, it repeats the process. We add an optional second argument loop to
specify the number of times the set of patterns should be run:"""
cp = nltk.RegexpParser(grammar, loop=2)
print cp.parse(sentence)
```

</td><td>7.4.1</td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
def traverse(t):
    try:
        t.label()
    except AttributeError:
        print t,
    else:
        # Now we know that t.node is defined
        print '(', t.label(), #print parent node
        for child in t:
            traverse(child) #call left sub-tree, call right sub-tree
        print ')', #close parent node
if __name__ == '__main__':
    # In NLTK, we create a tree by giving a node label and a
    # list of children:
    tree1 = nltk.Tree('NP', ['Alice'])
    print "1: ", tree1
    tree2 = nltk.Tree('NP', ['the', 'rabbit'])
    print "2: ", tree2
    # We can incorporate these into successively larger trees as follows:
    tree3 = nltk.Tree('VP', ['chased', tree2])
    tree4 = nltk.Tree('S', [tree1, tree3])
    print "3: ", tree4
    # Here are some of the methods available for tree objects:
    print "4: ", tree4[1]
    print "5: ", tree4[1].label()
    print "6: ", tree4.leaves()
    print "7: ", tree4[1][1][1]
    tree4.draw()
    #Traversal
    t = nltk.Tree.fromstring('(S (NP Alice) (VP chased (NP the rabbit)))')
    traverse(t)
```

</td><td>B7.4.2</td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
"""
NE type          Examples
ORGANIZATION     Georgia-Pacific Corp., WHO
PERSON           Eddy Bonte, President Obama
LOCATION         Murray River, Mount Everest
DATE             June, 2008-06-29
TIME              two fifty a m, 1:30 p.m.
MONEY            175 million Canadian Dollars, GBP 10.40
PERCENT          twenty pct, 18.75 %
```

</td><td>B7.5.1</td></tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
FACILITY        Washington Monument, Stonehenge
GPE             South East Asia, Midlothian
"""
"""The goal of a named entity recognition (NER) system is to identify all textual men-
tions of the named entities. This can be broken down into two subtasks: identifying
the boundaries of the NE, and identifying its type. While named entity recognition is
frequently a prelude to identifying relations in Information Extraction, it can also
con-
tribute to other tasks. For example, in Question Answering (QA), we try to improve
the precision of Information Retrieval by recovering not whole pages, but just those
parts which contain an answer to the user’s question.
 Most QA systems take the documents returned by standard Information Retrieval, and then
attempt to isolate the
minimal text snippet in the document containing the answer.
Now suppose the
question was Who was the first President of the US?, and one of the documents that was
retrieved contained the following passage:
(5)The Washington Monument is the most prominent structure in Washington,
D.C. and one of the city’s early attractions. It was built in honor of George
Washington, who led the country to independence and then became its first
President.
Analysis of the question leads us to expect that an answer should be of the form X was
the first President of the US, where X is not only a noun phrase, but also refers to a
named entity of type PER. This should allow us to ignore the first sentence in the
passage.
Although it contains two occurrences of Washington, named entity recognition should
tell us that neither of them has the correct type.
"""
"""NLTK provides a classifier that has already been trained to recognize named entities,
accessed with the function nltk.ne_chunk(). If we set the parameter binary=True ,
then named entities are just tagged as NE; otherwise, the classifier adds category
labels
such as PERSON, ORGANIZATION, and GPE.
"""
if __name__ == '__main__':
    sent = nltk.corpus.treebank.tagged_sents()[13]
    print "1: ", nltk.ne_chunk(sent, binary=False)
    sentence = "Let's meet tomorrow at 9 pm";
    sentence = "I went to New York to meet John Smith"
    tokens = nltk.word_tokenize(sentence)
    pos_tags = nltk.pos_tag(tokens)
    ne_tree = nltk.ne_chunk(pos_tags, binary=False)
    print "2: ", sentence
    print "3: ", tokens
    print "4: ", pos_tags
    print "5: ", ne_tree
    ne_tree.draw()
```

B7.5.2

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
# Loads the serialized NEChunkParser object
chunker =
nltk.data.load('/home/benzro/nltk_data/chunkers/maxent_ne_chunker/english_ace_multiclass
.pickle')
# The MaxEnt classifier
maxEnt = chunker._tagger.classifier()
def maxEnt_report():
    maxEnt = chunker._tagger.classifier()
    print("These are the labels used by the NLTK\'s NEC:\n")
    print(maxEnt.labels())
    print("These are the most informative features found in the ACE corpora:\n")
    print(maxEnt.show_most_informative_features())
def ne_report(sentence, report_all=False):
    tokens = nltk.word_tokenize(sentence)
    tagged_tokens = nltk.pos_tag(tokens)
    tags = []
    for i in xrange(len(tagged_tokens)):
        featureset = chunker._tagger.feature_detector(tagged_tokens, i, tags)
        tag = chunker._tagger.choose_tag(tagged_tokens, i, tags)
        if tag != 'O' or report_all:
            print '\nExplanation on the why the word \'' + tagged_tokens[i][0] + '\' was
tagged:'
            featureset = chunker._tagger.feature_detector(tagged_tokens, i, tags)
            maxEnt.explain(featureset)
        tags.append(tag)
if __name__ == '__main__':
    maxEnt_report()
    ne_report('I am very excited about the next '
              'generation of Apple products.')
```

B7.6.1

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
import re
import nltk.sem.relextract
#Muster:
# .*        beliebige Zeichen
# \bin\b    eigenständiges Wort "in", \b ist (^\w|\w$|\W\w|\w\W)
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
# (?! foo)  Gruppe aus Negative Lookahead Asserts that what
#           immediately follows the current position in the string is not foo
# \b.+ing   eigenständiges Wort da mit "ing" endet
IN = re.compile(r'.*\bin\b(?!\b.+ing)')
for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
    for rel in nltk.sem.relextract.extract_rels('ORG', 'LOC', doc,
                corpus='ieer', pattern = IN):
        #print rel
        print nltk.sem.relextract.rtuple(rel)
```

B8.1.1

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk import CFG
if __name__ == '__main__':
    groucho_grammar = nltk.CFG.fromstring("""
        S -> NP VP
        PP -> P NP
        NP -> Det N | Det N PP | 'I'
        VP -> V NP | VP PP
        Det -> 'an' | 'my'
        N -> 'elephant' | 'pajamas'
        V -> 'shot'
        P -> 'in'
        """)
    sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
    cp = nltk.ChartParser(groucho_grammar)
    trees = cp.parse(sent)
    for tree in trees:
        print tree
```

B8.3.1

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk import CFG
"""Let's start off by looking at a simple context-free grammar (CFG). By convention,
the lefthand side of the first production is the start-symbol of the grammar, typically
S, and all well-formed trees must have this symbol as their root label. In NLTK,
context-
free grammars are defined in the nltk.grammar module."""
if __name__ == '__main__':
    grammar1 = nltk.data.load('file:mygrammar.cfg')
    grammar2 = nltk.CFG.fromstring("""
        S -> NP VP
        VP -> V NP | V NP PP
        PP -> P NP
        V -> "saw" | "ate" | "walked"
        NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
        Det -> "a" | "an" | "the" | "my"
        N -> "man" | "dog" | "cat" | "telescope" | "park"
        P -> "in" | "on" | "by" | "with"
        """)
    sent = "Mary saw Bob".split()
    rd_parser = nltk.RecursiveDescentParser(grammar1, trace=2)
    for tree in rd_parser.parse(sent):
        print tree
    for p in grammar1.productions():
        print p
```

B8.3.2

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk import CFG
"""A grammar is said to be recursive if a category occurring on the lefthand side of a
production also appears on the righthand side of a production, as illustrated in Exam-
ple 8-2. The production Nom -> Adj Nom (where Nom is the category of nominals) involves
direct recursion on the category Nom, whereas indirect recursion on S arises from the
combination of two productions, namely S -> NP VP and VP -> V S.
Beware that the RecursiveDescentParser is unable to handle left-
recursive productions of the form X -> X Y; we will return to this in Section 8.4.
"""
if __name__ == '__main__':
    grammar1 = nltk.CFG.fromstring("""
        S -> NP VP
        NP -> Det Nom | PropN
        Nom -> Adj Nom | N
        VP -> V Adj | V NP | V S | V NP PP
        PP -> P NP
        PropN -> 'Buster' | 'Chatterer' | 'Joe'
        Det -> 'the' | 'a'
        N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'
        Adj -> 'angry' | 'frightened' | 'little' | 'tall'
        V -> 'chased' | 'saw' | 'said' | 'thought' | 'was' | 'put'
        P -> 'on'
        """)
    sent = "the angry bear chased the frightened little squirrel".split()
```

<table>
<tr><td><pre>
    rd_parser = nltk.RecursiveDescentParser(grammar1, trace=1)
    for tree in rd_parser.parse(sent):
        print tree
    for p in grammar1.productions():
        print p
</pre></td><td></td></tr>
<tr><td><pre>
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk import CFG
"""In this section, we see two simple parsing algorithms, a top-down method called re-
cursive descent parsing, and a bottom-up method called shift-reduce parsing. We also
see some more sophisticated algorithms, a top-down method with bottom-up filtering
called left-corner parsing, and a dynamic programming technique called chart parsing.
"""
if __name__ == '__main__':
    grammar = nltk.CFG.fromstring("""
        S -> NP VP
        NP -> Det Nom | PropN
        Nom -> Adj Nom | N
        VP -> V Adj | V NP | V S | V NP PP
        PP -> P NP
        PropN -> 'Buster' | 'Chatterer' | 'Joe'
        Det -> 'the' | 'a'
        N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'
        Adj -> 'angry' | 'frightened' | 'little' | 'tall'
        V -> 'chased' | 'saw' | 'said' | 'thought' | 'was' | 'put'
        P -> 'on'
        """)
    print "nltk.RecursiveDescentParser"
    print "---------------------------"
    sent = "Joe saw a fish".split()
    rd_parser = nltk.RecursiveDescentParser(grammar, trace=1)
    for tree in rd_parser.parse(sent):
        print tree
    for p in grammar.productions():
        print p
    print "ShiftReduceParser"
    print "---------------------------"
    sr_parser = nltk.ShiftReduceParser(grammar,trace=1)
    sent = 'Buster was on a little log'.split()
    for tree in sr_parser.parse(sent):
        print tree
    for p in grammar.productions():
        print p
</pre></td><td>B8.4.1</td></tr>
<tr><td><pre>
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import treebank
nltk.corpus.ppattach
def filter1(tree):
    #child_nodes = [child.label() for child in tree
    #   if isinstance(child, nltk.Tree)]
    child_nodes=[]
    for child in tree:
        if isinstance(child, nltk.Tree):
            child_nodes.append(child.label())
    #return (tree.label() == 'VP') and ('S' in child_nodes)
    if (tree.label() == 'VP') and ('S' in child_nodes):
        return child_nodes
if __name__ == '__main__':
    t = treebank.parsed_sents('wsj_0001.mrg')[0]
    print "0:\n", t
    """We can use this data to help develop a grammar. For example, the program in Exam-
    ple 8-4 uses a simple filter to find verbs that take sentential complements.
Assuming
    we already have a production of the form VP -> SV S, this information enables us to
    identify particular verbs that would be included in the expansion of SV.
    """
    #"Example 8-4. Searching a treebank to find sentential complements."
    #print [subtree for tree in treebank.parsed_sents()
    #   for subtree in tree.subtrees(filter)]
    for e in treebank.parsed_sents()[13:14]:
        print "1.1:\n", e
    l=[]
    for tree in treebank.parsed_sents()[13:14]:
        subtrees = tree.subtrees(filter=filter1)
        for subtree in subtrees:
            l.append(subtree)
    for e in l:
        print "1.2:\n", e
    entries = nltk.corpus.ppattach.attachments('training')
    table = nltk.defaultdict(lambda: nltk.defaultdict(set))
    for entry in entries:
        key = entry.noun1 + '-' + entry.prep + '-' + entry.noun2
        table[key][entry.attachment].add(entry.verb)
</pre></td><td>B8.6.1</td></tr>
</table>

<table>
<tr><td>

```
    for key in sorted(table):
        if len(table[key]) > 1:
            print key, 'N:', sorted(table[key]['N']), \
                'V:', sorted(table[key]['V'])
```

</td><td></td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
def filter1(x):
    return x.label()=='NP'
def getNodes(parent):
    if parent.label() == ROOT:
        print "1.1:\n", "======== Sentence ========="
    for node in parent:
        if type(node) is nltk.Tree:
            print "1.2:\n","Label:", node.label()
            print "1.3:\n","Leaves:", node.leaves(), "\n"
            getNodes(node) #depth first recursion
        else:
            print "1.4:\n","Word:", node, "\n"
if __name__ == '__main__':
    #Bild Tree
    grammar = '''
        NP: {<DT>*(<NN.*>|<JJ.*>)*<NN.*>}
        NVN:{<NP><VB.*><NP>}
        '''
    chunker = nltk.chunk.RegexpParser(grammar)
    sent="The_Pigs are a Bristol-based punk rock " \
         "band that formed in 1977 ."
    tok=nltk.word_tokenize(sent)
    tagged=nltk.pos_tag(tok)
    tree = chunker.parse(tagged)
    print "1:\n", tree
    # Search tree V.1 / (Traverse whole tree)
    ROOT = 'S'
    getNodes(tree)
    # Search tree V.2 / (Generate all subtrees)
    for subtree in tree.subtrees(filter =  filter1):
        print "2:\n",subtree
    # Search tree V.3 / (Generate all NVN)
    for i in tree.subtrees(filter=lambda x: x.label() == 'NVN'):
        print "3:\n", i
    # Search tree V.4
    for p in tree:
        if type(p) is nltk.Tree:
            if p.label() == 'NVN':  # This climbs into your NVN tree
                for c in p:
                    if type(c) is nltk.Tree and c.label() == 'NP':
                        print "4.1:\n", c.leaves()  # This outputs your "NP"
                    else:
                        print "4.2:\n", c  # This outputs your "VB.*"
```

</td><td>B8.6.2</td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import treebank
nltk.corpus.ppattach
if __name__ == '__main__':
    """
    The PP Attachment Corpus, nltk.corpus.ppattach, is another source of information
    about the valency of particular verbs. Here we illustrate a technique for mining
this
    corpus. It finds pairs of prepositional phrases where the preposition and noun are
fixed,
    but where the choice of verb determines whether the prepositional phrase is attached
    to the VP or to the NP.
    """
    entries = nltk.corpus.ppattach.attachments('training')[:5]
    print "1:\n", entries
    table = nltk.defaultdict(lambda: nltk.defaultdict(set))
    for entry in entries:
        key = entry.noun1 + '-' + entry.prep + '-' + entry.noun2
        table[key][entry.attachment].add(entry.verb)
        print "2:\n", key
        print "3:\n", entry.attachment
        print "4:\n", entry.verb
    table_sort=sorted(table.keys())
    print "5:\n",table_sort
    for key in table_sort:
        ret=table[key]
        if len(ret) > 1:
            print "6:\n", ret, len(ret)
            print "7:\n", key, 'N:', sorted(table[key]['N']), \
                'V:', sorted(table[key]['V'])
    print"-----------------------------"
    entries = nltk.corpus.ppattach.attachments('training')
    table = nltk.defaultdict(lambda: nltk.defaultdict(set))
```

</td><td>B8.6.3</td></tr>
</table>

```
    for entry in entries:
        key = entry.noun1 + '-' + entry.prep + '-' + entry.noun2
        table[key][entry.attachment].add(entry.verb)
    for key in sorted(table):
        if len(table[key]) > 1:
            print "6:\n", len(table[key])
            print key, 'N:', sorted(table[key]['N']), 'V:', sorted(table[key]['V'])
```

B8.6.4

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import treebank
"""As we have just seen, dealing with ambiguity is a key challenge in developing broad-
coverage parsers. Chart parsers improve the efficiency of computing multiple parses of
the same sentences, but they are still overwhelmed by the sheer number of possible
parses. Weighted grammars and probabilistic parsing algorithms have provided an ef-
fective solution to these problems.
Using the Penn Treebank sample, we can examine all instances of prepositional dative
and double object constructions involving give, as shown in Example 8-5.
"""
def filter_give(t):
    """return if such a structure
        (VP
            (VBD gave)
            (NP (DT the) (NNS chefs))
            (NP (DT a) (JJ standing) (NN ovation)))
        """
    # return t.label() == 'VP' and len(t) > 2 and t[1].label() == 'NP'\
    # and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP')\
    # and ('give' in t[0].leaves() or 'gave' in t[0].leaves())
    if (t.label() == 'VP'
        and len(t) > 2      #nr of direct child nodes
        and t[1].label() == 'NP'
        and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP')
        and ('give' in t[0].leaves() or 'gave' in t[0].leaves())):
        return t
def sent(t):
    str = ' '.join(token
                   for token in t.leaves()
                        if token[0] not in '*-0')
    print "10.1", t.leaves() #(most) leaves are terminals (Worte mit Tag)
    for token in t.leaves():
        print "10.2", token
        print "10.3", token[0], token[0] not in '*-0' #einige Leaves sind keine
Terminals
    return str
def print_node(t, width):
    output = "%s %s: %s / %s: %s" %\
        (sent(t[0]),
         t[1].label(), sent(t[1]),
         t[2].label(), sent(t[2]))
    if len(output) > width:
        output = output[:width] + "..."
    print output
if __name__ == '__main__':
    #Example 8-5. Usage of give and gave in the Penn Treebank sample.
    for tree in nltk.corpus.treebank.parsed_sents()[50:150]:
        for t in tree.subtrees(filter=filter_give):
            print "1:\n", t
            print "2:\n",t.label()
            print "3:\n",len(t) #nr of direct child nodes
            print "4:\n",t[0].label(), t[0].leaves()
            print "5:\n",t[1].label()
            print "6:\n",t[2].label()
            print "---------"
            print_node(t, 72)
            print "---------"
"""We can observe a strong tendency for the shortest complement to appear first. How-
ever, this does not account for a form like give NP: federal judges / NP: a raise,
where animacy may play a role. In fact, there turns out to be a large number of
contributing factors, as surveyed by (Bresnan & Hay, 2008). Such preferences can be
represented in a weighted grammar.
A probabilistic context-free grammar (or PCFG) is a context-free grammar that as-
sociates a probability with each of its productions.
"""
```

B8.6.5

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
import nltk
from nltk.corpus import treebank
from nltk import PCFG
from nltk import CFG
"""A probabilistic context-free grammar (or PCFG) is a context-free grammar that as-
sociates a probability with each of its productions. It generates the same set of parses
for a text that the corresponding context-free grammar does, and assigns a probability
to each parse. The probability of a parse generated by a PCFG is simply the product of
the probabilities of the productions used to generate it.
The simplest way to define a PCFG is to load it from a specially formatted string con-
sisting of a sequence of weighted productions, where weights appear in brackets, as
```

```
shown in Example 8-6.
"""
if __name__ == '__main__':
    #Example 8-6. Defining a probabilistic context-free grammar (PCFG).
    #grammar = nltk.parse_pcfg("""          #alt
    #grammar = nltk.CFG.fromstring("""      #für CFG Grammatik
    grammar = nltk.PCFG.fromstring("""
    S -> NP VP [1.0]
    VP -> TV NP [0.4]
    VP -> IV [0.3]
    VP -> DatV NP NP [0.3]
    TV -> 'saw' [1.0]
    IV -> 'ate' [1.0]
    DatV -> 'gave' [1.0]
    NP -> 'telescopes' [0.8]
    NP -> 'Jack' [0.2]
    """)
    print "1.1\n", grammar
    print "1.2\n",grammar.start()
    print "1.3\n",grammar.productions()
    #The parse tree returned by parse() includes probabilities:
    viterbi_parser = nltk.ViterbiParser(grammar)
    trees = viterbi_parser.parse(['Jack', 'saw', 'telescopes'])
    """Now that parse trees are assigned probabilities, it no longer matters that there
may be
    a huge number of possible parses for a given sentence. A parser will be responsible
for
    finding the most likely parses.
    """
    for t in trees: print "2.1\n",t
```

B9.1.1

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""
 In this chapter, we will investigate the role of features in building rule-based
grammars. In contrast to feature extractors, which record features that have been au-
tomatically detected, we are now going to declare the features of words and phrases.
"""
def lex2fs(word):
    for fs in [kim, lee, chase]:
        if fs['ORTH'] == word:
            return fs
if __name__ == '__main__':
    #CAT (grammaticalcategory) and ORTH(orthography, i.e., spelling)
    kim = {'CAT': 'NP', 'ORTH': 'Kim', 'REF': 'k'}
    chase = {'CAT': 'V', 'ORTH': 'chased', 'REL': 'chase'}
    """the subject plays the role of 'agent,' whereas the object
    has the role of 'patient.' Lets add this information, using
    'sbj' (subject) and 'obj' (object) as placeholders which
    will get filled once the verb combines with its grammatical
    arguments:"""
    chase['AGT'] = 'sbj'
    chase['PAT'] = 'obj'
    """If we now process a sentence Kim chased Lee, we want to
    “bind” the verb’s agent role to the subject and the patient
    role to the object. We do this by linking to the REF feature
    of the relevant NP."""
    sent = "Kim chased Lee"
    tokens = sent.split()
    lee = {'CAT': 'NP', 'ORTH': 'Lee', 'REF': 'l'}
    subj, verb, obj = lex2fs(tokens[0]), lex2fs(tokens[1]), \
                      lex2fs(tokens[2])
    print "1: ", subj, verb, obj
    verb['AGT'] = subj['REF'] # agent of 'chase' is Kim
    verb['PAT'] = obj['REF'] # patient of 'chase' is Lee
    # check featstruct of 'chase'
    for k in ['ORTH', 'REL', 'AGT', 'PAT']:
        print "%-5s => %s" % (k, verb[k])
    """The same approach could be adopted for a different
    verb—say, surprise—though in this case, the subject
    would play the role of “source” (SRC), and the object
    plays the role of “experiencer” (EXP):
    """
    surprise = {'CAT': 'V', 'ORTH': 'surprised','REL': 'surprise',
                'SRC': 'sbj', 'EXP': 'obj'}
```

B9.1.2

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""
Table 9-1. Agreement paradigm for English regular verbs
                        Singular                        Plural
1st person          I run                           we run
2nd person          you run                     you run
3rd person      he/she/it runs              they run
Here we can see that morphological properties of the verb co-vary with
syntactic properties of the subject noun phrase. This co-variance is called agreement.
These representations indicate that the verb agrees with its subject in person
and number.
```

```
(5) the dog run-s
dog.3.SG run-3.SG
(6) the dog-s run
dog.3.PL run-3.PL
S->NP VP
NP->Det N
VP->V
Det->'this'
N->'dog'
V->'runs'
The most straightforward approach is to add new non-
terminals and productions to the grammar:
S -> NP_SG VP_SG
S -> NP_PL VP_PL
NP_SG -> Det_SG N_SG
NP_PL -> Det_PL N_PL
VP_SG -> V_SG
VP_PL -> V_PL
Det_SG -> 'this'
Det_PL -> 'these'
N_SG -> 'dog'
N_PL -> 'dogs'
V_SG -> 'runs'
V_PL -> 'run'
In place of a single production expanding S, we now have two productions.
In the next section, we will show that capturing number and person agree-
ment need not come at the cost of "blowing up" the number of productions.
Using Attributes and Constraints
We spoke informally of linguistic categories having properties, for example, that a noun
has the property of being plural. Let's make this explicit:
(9)
 N[NUM=pl]
In (9), we have introduced some new notation which says that the category N has a
(grammatical) feature called NUM (short for "number") and that the value of this feature
is pl (short for "plural"). We can add similar annotations to other categories, and use
them in lexical entries:
(10)
 Det[NUM=sg] -> 'this'
Det[NUM=pl] -> 'these'
N[NUM=sg]->'dog'
N[NUM=pl]->'dogs'
V[NUM=sg]->'runs'
V[NUM=pl]->'run'
(11)
 S -> NP[NUM=?n] VP[NUM=?n]
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
VP[NUM=?n] -> V[NUM=?n]
We are using ?n as a variable over values of NUM; it can be instantiated either to sg or
pl, within a given production. We can read the first production as saying that whatever
value NP takes for the feature NUM, VP must take the same value.
Assigning a variable value to
NUM is one way of achieving this result:
Det[NUM=?n] -> 'the' | 'some' | 'several'
But in fact we can be even more economical, and just omit any specification for NUM in
such productions.
"""
import nltk
from nltk import load_parser
if __name__ == '__main__':
    #Example 9-1. Example feature-based grammar.
    nltk.data.show_cfg('grammars/book_grammars/feat0.fcfg')
    """
    % start S
    # ###################
    # Grammar Productions
    # ###################
    # S expansion productions
    S -> NP[NUM=?n] VP[NUM=?n]
    # NP expansion productions
    NP[NUM=?n] -> N[NUM=?n]
    NP[NUM=?n] -> PropN[NUM=?n]
    NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
    NP[NUM=pl] -> N[NUM=pl]
    # VP expansion productions
    VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t, NUM=?n]
    VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP
    # ###################
    # Lexical Productions
    # ###################
    Det[NUM=sg] -> 'this' | 'every'
    Det[NUM=pl] -> 'these' | 'all'
    Det -> 'the' | 'some' | 'several'
    PropN[NUM=sg]-> 'Kim' | 'Jody'
    N[NUM=sg] -> 'dog' | 'girl' | 'car' | 'child'
    N[NUM=pl] -> 'dogs' | 'girls' | 'cars' | 'children'
    IV[TENSE=pres, NUM=sg] -> 'disappears' | 'walks'
    TV[TENSE=pres, NUM=sg] -> 'sees' | 'likes'
    IV[TENSE=pres, NUM=pl] -> 'disappear' | 'walk'
    TV[TENSE=pres, NUM=pl] -> 'see' | 'like'
    IV[TENSE=past] -> 'disappeared' | 'walked'
    TV[TENSE=past] -> 'saw' | 'liked'
    """
    tokens = 'Kim likes children'.split()
    cp = load_parser('grammars/book_grammars/feat0.fcfg',trace=2)
    #trees = cp.nbest_parse(tokens) #old
```

```
    trees = next(cp.parse(tokens))
    out  = trees[0].label()
    print "1:\n",out
    print "2:\n", trees
    for tree in trees: print "3:\n",tree
```

B9.2.1

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""Terminology
So far, we have only seen feature values like sg and pl. These simple values are usually
called atomic—that is, they can’t be decomposed into subparts. A special case of
atomic values are Boolean values, that is, values that just specify whether a property
is true or false.
Then the production V[TENSE=pres,
aux=+] -> 'can' means that can receives the value pres for TENSE and + or true for
AUX. There is a widely adopted convention that abbreviates the representation of Boo-
lean features f; instead of aux=+ or aux=-, we use +aux and -aux respectively. These are
just abbreviations, however, and the parser interprets them as though + and - are like
any other atomic value. (17) shows some representative productions:
(17)
 V[TENSE=pres, +aux] -> 'can'
V[TENSE=pres, +aux] -> 'may'
V[TENSE=pres, -aux] -> 'walks'
V[TENSE=pres, -aux] -> 'likes'
We have spoken of attaching “feature annotations” to syntactic categories. A more
radical approach represents the whole category—that is, the non-terminal symbol plus
the annotation—as a bundle of features. For example, N[NUM=sg] contains part-of-
speech information which can be represented as POS=N. An alternative notation for this
category, therefore, is [POS=N, NUM=sg].
In addition to atomic-valued features, features may take values that are themselves
feature structures. For example, we can group together agreement features (e.g., per-
son, number, and gender) as a distinguished part of a category, serving as the value of
AGR. In this case, we say that AGR has a complex value. (18) depicts the structure, in a
format known as an attribute value matrix (AVM).
(18)
 [POS = N                   ]
[                                  ]
[AGR = [PER = 3]            ]
[           [NUM = pl ]     ]
[           [GND = fem ]    ]
feature structures, like dictionaries,
assign no particular significance to the order of features.
Once we have the possibility of using features like AGR, we can refactor a grammar like
Example 9-1 so that agreement features are bundled together. A tiny grammar illus-
trating this idea is shown in (20).
(20)
 S -> NP[AGR=?n] VP[AGR=?n]
NP[AGR=?n] -> PropN[AGR=?n]
VP[TENSE=?t, AGR=?n] -> Cop[TENSE=?t, AGR=?n] Adj
Cop[TENSE=pres, AGR=[NUM=sg, PER=3]] -> 'is'
PropN[AGR=[NUM=sg, PER=3]] -> 'Kim'
Adj -> 'happy'
In this section, we will show how feature structures can be constructed and manipulated
in NLTK. We will also discuss the fundamental operation of unification, which allows
us to combine the information contained in two different feature structures.
Feature structures in NLTK are declared with the FeatStruct() constructor. Atomic
feature values can be strings or integers.
It is often helpful to view feature structures as graphs, more specifically, as directed
acyclic graphs (DAGs).
The feature names appear as labels on the directed arcs, and feature values appear as
labels on the nodes that are pointed to by the arcs.
 the value of the path ('ADDRESS') in (24) is identical to the value of the
path ('SPOUSE', 'ADDRESS'). DAGs such as (24) are said to involve structure shar-
ing or reentrancy. When two paths have the same value, they are said to be
equivalent.
"""
import nltk
from nltk import load_parser
if __name__ == '__main__':
    """A feature structure is actually just a kind of dictionary,
    and so we access its values by indexing in the usual way.
    We can use our familiar syntax to assign values to features:
    """
    fs1 = nltk.FeatStruct(TENSE='past', NUM='sg')
    print "1:\n",fs1
    fs1 = nltk.FeatStruct(PER=3, NUM='pl', GND='fem')
    print "2:\n",fs1['GND']
    print "3:\n",fs1
    """We can also define feature structures that have
    complex values, as discussed earlier.
    """
    fs2 = nltk.FeatStruct(POS='N', AGR=fs1)
    print "4:\n",fs2
    print "5:\n",fs2['AGR']
    print "6:\n",fs2['AGR']['PER']
    """An alternative method of specifying feature structures
    is to use a bracketed string consisting of feature-value
    pairs in the format feature=value"""
    fs3= nltk.FeatStruct(
        "[POS='N', AGR=[PER=3, NUM='pl', GND='fem']]")
    print "7:\n", fs3
    """Feature structures are not inherently tied to linguistic
```

<table>
<tr><td>

```
    objects; they are general-purpose structures for
    representing knowledge. For example, we could encode
    information about a person in a feature structure:
    """
    fs4=nltk.FeatStruct(name='Lee', telno='01 27 86 42 96', age=33)
    print "8:\n", fs4
    """In order to indicate reentrancy in our matrix-style
    representations, we will prefix the first occurrence of
    a shared feature structure with an integer in parentheses,
    such as (1). Any later reference to that structure will
    use the notation ->(1), as shown here.
    The bracketed integer is sometimes called a tag or a coindex.
    """
    fs5=nltk.FeatStruct("""[NAME='Lee',
                ADDRESS=(1)[NUMBER=74, STREET='rue Pascal'],
                SPOUSE=[NAME='Kim', ADDRESS->(1)]]""")
    print "8:\n", fs5
```

</td><td></td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""
It is standard to think of feature structures as providing partial information about
some object, in the sense that we can order feature structures according to how general
they are. For example, (25a) is more general (less specific) than (25b), which in turn
is
more general than (25c).
(25)
a.
 [NUMBER = 74]
b.
 [NUMBER = 74
 ]
[STREET = 'rue Pascal']
c.
 [NUMBER = 74]
[STREET = 'rue Pascal']
[CITY = 'Paris']
This ordering is called subsumption; a more general feature structure subsumes a less
general one.
"""
import nltk
from nltk import load_parser
if __name__ == '__main__':
    """Merging information from two feature structures is
    called unification and is supported by the unify() method.
    """
    fs1 = nltk.FeatStruct(NUMBER=74, STREET='rue Pascal')
    fs2 = nltk.FeatStruct(CITY='Paris')
    print "1:\n", fs1.unify(fs2)
    fs0 =nltk.FeatStruct("""[NAME=Lee,
     ADDRESS=[NUMBER=74, STREET='rue Pascal'],
     SPOUSE= [NAME=Kim,ADDRESS=[NUMBER=74,
                STREET='rue Pascal']]]""")
    print "2:\n", fs0
    fs1 = nltk.FeatStruct(
        "[SPOUSE = [ADDRESS = [CITY = Paris]]]")
    print "3:\n",fs1.unify(fs0)
    fs2 = nltk.FeatStruct("""[NAME=Lee,
    ADDRESS=(1)[NUMBER=74, STREET='rue Pascal'],
        SPOUSE=[NAME=Kim, ADDRESS->(1)]]""")
    print "4:\n", fs2
    print "5:\n", fs1.unify(fs2)
    """"As we have already seen, structure sharing can also
    be stated using variables such as ?x.
    """
    fs1 = nltk.FeatStruct(
        "[ADDRESS1=[NUMBER=74, STREET='rue Pascal']]")
    fs2 = nltk.FeatStruct("[ADDRESS1=?x, ADDRESS2=?x]")
    print "6:\n",fs2
    print "7:\n",fs2.unify(fs1)
```

</td><td>B9.2.2</td></tr>
<tr><td>

```
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""
9.3 Extending a Feature-Based Grammar
In this section, we return to feature-based grammar and explore a variety of linguistic
issues, and demonstrate the benefits of incorporating features into the grammar.
Subcategorization
In Chapter 8, we augmented our category labels to represent different kinds of verbs,
and used the labels IV and TV for intransitive and transitive verbs respectively. This
allowed us to write productions like the following:
(29)
 VP -> IV
VP -> TV NP
Although we know that IV and TV are two kinds of V, they are just atomic non-terminal
symbols in a CFG and are as distinct from each other as any other pair of symbols.
This notation doesn’t let us say anything about verbs in general; e.g., we cannot say
“All
```

</td><td>B9.3.1</td></tr>
</table>

```
lexical items of category V can be marked for tense," since walk, say, is an item of
category IV, not V. So, can we replace category labels such as TV and IV by V along with
a feature that tells us whether the verb combines with a following NP object or whether
it can occur without any complement?
A simple approach, originally developed for a grammar framework called Generalized
Phrase Structure Grammar (GPSG), tries to solve this problem by allowing lexical cat-
egories to bear a SUBCAT feature, which tells us what subcategorization class the item
belongs to. In contrast to the integer values for SUBCAT used by GPSG, the example here
adopts more mnemonic values, namely intrans, trans, and clause:
(30)
 VP[TENSE=?t, NUM=?n] -> V[SUBCAT=intrans, TENSE=?t, NUM=?n]
VP[TENSE=?t, NUM=?n] -> V[SUBCAT=trans, TENSE=?t, NUM=?n] NP
VP[TENSE=?t, NUM=?n] -> V[SUBCAT=clause, TENSE=?t, NUM=?n] SBar
V[SUBCAT=intrans, TENSE=pres, NUM=sg] -> 'disappears' | 'walks'
V[SUBCAT=trans, TENSE=pres, NUM=sg] -> 'sees' | 'likes'
V[SUBCAT=clause, TENSE=pres, NUM=sg] -> 'says' | 'claims'
V[SUBCAT=intrans, TENSE=pres, NUM=pl] -> 'disappear' | 'walk'
V[SUBCAT=trans, TENSE=pres, NUM=pl] -> 'see' | 'like'
V[SUBCAT=clause, TENSE=pres, NUM=pl] -> 'say' | 'claim'
V[SUBCAT=intrans, TENSE=past] -> 'disappeared' | 'walked'
V[SUBCAT=trans, TENSE=past] -> 'saw' | 'liked'
V[SUBCAT=clause, TENSE=past] -> 'said' | 'claimed'
When we see a lexical category like V[SUBCAT=trans], we can interpret the SUBCAT spec-
ification as a pointer to a production in which V[SUBCAT=trans] is introduced as the
head child in a VP production. By convention, there is a correspondence between the
values of SUBCAT and the productions that introduce lexical heads. On this approach,
SUBCAT can appear only on lexical categories; it makes no sense, for example, to specify
a SUBCAT value on VP. As required, walk and like both belong to the category V. Never-
theless, walk will occur only in VPs expanded by a production with the feature
SUBCAT=intrans on the righthand side, as opposed to like, which requires a
SUBCAT=trans.
In our third class of verbs in (30), we have specified a category SBar. This is a label
for
subordinate clauses, such as the complement of claim in the example You claim that
you like children. We require two further productions to analyze such sentences:
(31)
 SBar -> Comp S
Comp -> 'that'
"""
import nltk
from nltk import load_parser
if __name__ == '__main__':
    """Example 9-3. Grammar with productions for inverted
    clauses and long-distance dependencies,making use of
    slash categories.
    """
    print nltk.data.show_cfg('grammars/book_grammars/feat1.fcfg')
    tokens = 'who do you claim that you like'.split()
    cp = load_parser('grammars/book_grammars/feat1.fcfg')
    # for tree in cp.nbest_parse(tokens):
    #     print tree
    trees = next(cp.parse(tokens))
    out = trees[0].label()
    print "1:\n", out
    print "2:\n", trees
    for tree in trees: print "3:\n", tree
```

```
#!/usr/bin/python
#from nltk import FreqDist, corpus.brown #! syntax
from nltk import FreqDist #! corr
from nltk.corpus import brown #! corr
import nltk #!corr
from operator import itemgetter
###############################################
#class NGramModel: #! no object
class NGramModel(object): #! corr
   """
   a class for estimating n-gram probabilities and
   calculating probabilities of sentences (take that, Noam Chomsky!)
   """

   #startSntTok = "<s>" #! no tab
   startSntTok = "<s>" #! corr
   endSntTok = "</s>"
########################
   def __init__(self, corpus):
      """
      go through the corpus, count the n-grams and save their probabilities
      """

      corpusNgrams = []
      #for snt in corpus.sents()[0:3000]#! syntax
      for snt in corpus.sents()[0:3000]:
         corpusNgrams += self.getNgramList(snt)
      self.countsOfNgrams = FreqDist(corpusNgrams)

      ngramHistories = map(itemgetter(0), corpusNgrams)
      self.countsOfHistories = FreqDist(ngramHistories)

########################
```

Task 1: Fix me!

Take a look at the python script in the file `mostprob.py`. It is supposed to

- estimate the n-gram probabilities based on a given corpus
- apply those probabilities to the sentences of the same corpus
- display 20 unique most likely sentences from that corpus
- lower-cased tokens are used for estimations
- to save time, only the first 3000 sentences are taken

You will notice that the program does not work -- the script contains errors. Some of these will be compile-time and run-time errors and will not let the program finish. Others might be logical errors, which means that something is wrong with the output.

Your task is to fix all those errors: go through the code and make sure that the program runs and does what is expected of it.

The **hand-in** is a working script file with the errors fixed. Please add comments describing the errors that you corrected.

```
   def getNgramList(self, tokList):
      """
      take a list of tokens, return a list of bigrams
      """

      prevTok = self.startSntTok
      result = []

      for tok in tokList:
         ltok = tok.lower()

         #result.append((prevTok, tok)) #! ltok
         #prevTok = tok #! ltok
         result.append((prevTok, ltok))  # ! corr
         prevTok = ltok  # ! corr

      #return result + (prevTok, self.endSntTok) #! syntax
      #return result + [(prevTok, self.endSntTok)] #! corr Version 1
      result.append((prevTok, self.endSntTok)) #! corr Version 2
      return result  #! corr

########################
   def getNgramProb(self, ngram):
      """
      return the probability of a given n-gram
      """
      #conditional prob.
      # p(a|b)=p(a,b)/p(b)
      #return float(self.countsOfNgrams[ngram] /
      #          self.countsOfHistories[ngram[0]]) #! div
      return float(self.countsOfNgrams[ngram])/\
            self.countsOfHistories[ngram[0]] #! corr

########################
   #def getSentenceProb(sentence): #! syntax
   def getSentenceProb(self, sentence): #! corr
      #! no text
      """ # ! corr
      return the probability of a given sentence
      """
      #sntNgrams = getNgramList(sentence) #! syntax
      sntNgrams = self.getNgramList(sentence)# ! corr
      result = 1
      # p(a|b)p(b|c)=p(a,b)/p(b)p(b,c)/p(c)
      for ngram in sntNgrams:
         result *= self.getNgramProb(ngram)

      return result
################################################
if __name__ == "__main__":
   corpus = nltk.corpus.brown

   #mdl = NGramModel(nltk.corpus.brown)#! corpus
   mdl = NGramModel(corpus)#! corr

   #snts = list(corpus.sents()[0,3000])#! syntax
   snts = list(corpus.sents()[0:3000])#! corr
   #snts.sort(key=lambda x: mdl.getNgramProb(x)) #! logical
   snts.sort(key=lambda x: mdl.getSentenceProb(x))  #! corr

   prevSnt = None

   #fixed uniq implementation

   #for snt in snts[0:20]:
   #  if snt != prevSnt:
   #     print " ".join(snt) + " (" + str(mdl.getSentenceProb(snt)) + ")"
   #
   #  prevSnt = snt

   sntCount = 0
   sntIdx = 0

   while (sntIdx < len(snts) and sntCount < 20):
      snt = snts[sntIdx]

      if snt != prevSnt:
         print " ".join(snt) + " (" + str(mdl.getSentenceProb(snt)) + ")"
         sntCount += 1

      prevSnt = snt
      sntIdx += 1
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
File MakeMe1.py
-----------------
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
from MakeMe2 import NE_wrapper
import nltk
if __name__ == '__main__':
    print "from here"
    help("B7_5_3")
    help("B7_5_3.NE_wrapper")
    help("B7_5_3.NE_wrapper.make")
    #Use Cases
    #---------

    # Use case: call constructor with string
    sentence = "Let's meet tomorrow at 9 pm";
    ne = NE_wrapper(sentence)

    #Use case: call member make() with string
    sentence = "I went to New York to meet John Smith"
    ne.make(sentence)

    #Use case: call with list of tokens
    sentence = "I went to New York to meet John Smith"
    sentence=nltk.word_tokenize(sentence)
    ne = NE_wrapper(sentence)
    ne.make(sentence)

    #Error Handling
    #--------------

    #Empty List or String Input: raises ValueError exception
    ne = NE_wrapper([])
    ne.make("")

    # Wrong type: raises TypeError exception
    ne = NE_wrapper(())
    ne.make({})

    #Two strings: Just takes the first string
    ne = NE_wrapper("Hi there.","Hello man.")

    #Two lists: raises "some unknown error"
    ne = NE_wrapper([["hi","there"],["hello","man"]])

    # Uncatched logical errors
    # ------------------------
    # Two sentences in one string: takes both sentences
    tagged_sent1 = nltk.corpus.treebank.tagged_sents()[13]
    tagged_sent2 = nltk.corpus.treebank.tagged_sents()[14]
    tagged_sents = tagged_sent1 + tagged_sent1
    sentences = map(list, zip(*tagged_sents))
    #print "2: ", sentences[0]
    ne = NE_wrapper(sentences[0])

    # Two sentences in one string: takes both sentences
    ne = NE_wrapper("Hi there. Hello man.")


File MakeMe2.py
-----------------
#!", "usr", "bin", "python
#-*- coding: utf-8 -*-
"""
Chunk Parser Wrapper Module for Name Entity Recoginition
help("B7_5_3")
"""
import nltk
class NE_wrapper(object):
    """
        Generates sentence tree with POS, Chunk, NE information
        help("B7_5_3.NE_wrapper")
        """
    def __init__(self, input, output=True):
        """
                Constructor function just calls public make() method
                """
        self.make(input, output)
    def make(self, input, output=True):
        """
                Generates sentence tree and optionally prints the results
                help("B7_5_3.NE_wrapper")
                :param input: sentence as string or as list of words
                :param output: optionally prints output info into console
                :returns: chunk tree
                :raises TypeError, ValueError, UnknownErrors: raises an exception
                """
        print "~~  ~~  ~~  ~~  ~~  ~~  ~~  ~~"
        print "please wait"
        try:
            if isinstance(input, (str, unicode)):
                print "string input"
                if not input:
```

Task 2: Make me!

Your task is to create a python module with the functionality for named entity recognition in English sentences. The module must allow the user to tokenize a sentence string into a list of tokens, to tag the sentence with its part-of-speech sequence and to apply named entity recognition to the tagged sentence. Feel free to use `nltk.pos_tag`, `nltk.word_tokenize` and `nltk.ne_chunk` functions, or external taggers such as TreeTagger. It is up to you to choose, whether you want the model to contain functions or class(es), as long as

- they are properly documented: input/output parameters, functionality
- the functionality is nicely organized into functions or class methods
- your implementation is safe and follows the principles of duck typing, with handling/raising exceptions in case the input to the functions/methods is something erroneous or unexpected

Also write a standalone python script that imports your newly created module and applies the functionality to a couple of sample sentences and displays the results. It must catch the possible exceptions and display friendlier error messages.

The task is not to do good named entity recognition, but to show that you can create a usable python module, done in a nice coding style, with organization and documentation, handling exceptional situations and wrong types of input gracefully.

The **hand-ins** are the **two script files**: the module file and the sample usage file.

```
                    raise ValueError("empty string")
                self.input_ = input
                self.tok_ = self.__tok()
            elif isinstance(input, list):
                print "list input"
                if not input:
                    raise ValueError("empty list")
                self.input_ = input
                self.tok_ = input
            elif not isinstance(input, (str, unicode)) or not \
                    isinstance(input, list):
                raise TypeError("wrong type")
        except TypeError as e:
            print "you entered a", e
        except ValueError as e:
            print "you entered an", e
        except:
            print "some unknown error"
        else:
            try:
                self.tag_ = self.__tag()
                self.tree_ = self.__chunk()
                self.__show(output)
                return self.tree_
            except:
                print "some unknown error"
        finally:
            print "done"
            print "~~  ~~  ~~  ~~  ~~  ~~  ~~  ~~"
    def __tok(self):
        """
                Internal function
                """
        return nltk.word_tokenize(self.input_)
    def __tag(self):
        """
                Internal function
                """
        return nltk.pos_tag(self.tok_)
    def __chunk(self):
        """
                Internal function
                """
        return nltk.ne_chunk(self.tag_, binary=False)
    def __show(self, output):
        """
                Internal function
                """
        if output:
            print "1: self.input_\n", self.input_
            print "2: self.tok_\n", self.tok_
            print "3: tself.tag_\n", self.tag_
            print "4: self.tree_\n", self.tree_
            #self.tree.draw()
if __name__ == '__main__':
    #Use Cases
    #---------
    # Use case: call constructor with string
    sentence = "Let's meet tomorrow at 9 pm";
    ne = NE_wrapper(sentence)
    #Use case: call member make() with string
    sentence = "I went to New York to meet John Smith"
    ne.make(sentence)
    #Use case: call with list of tokens
    sentence = "I went to New York to meet John Smith"
    sentence=nltk.word_tokenize(sentence)
    ne = NE_wrapper(sentence)
    ne.make(sentence)
    #Error Handling
    #--------------
    #Empty List or String Input: raises ValueError exception
    ne = NE_wrapper([])
    ne.make("")
    # Wrong type: raises TypeError exception
    ne = NE_wrapper(())
    ne.make({})
    #Two strings: Just takes the first string
    ne = NE_wrapper("Hi there.","Hello man.")
    #Two lists: raises "some unknown error"
    ne = NE_wrapper([["hi","there"],["hello","man"]])
    # Uncatched logical errors
    # ------------------------
    # Two sentences in one string: takes both sentences
    tagged_sent1 = nltk.corpus.treebank.tagged_sents()[13]
    tagged_sent2 = nltk.corpus.treebank.tagged_sents()[14]
    tagged_sents = tagged_sent1 + tagged_sent1
    sentences = map(list, zip(*tagged_sents))
    #print "2: ", sentences[0]
    ne = NE_wrapper(sentences[0])
    # Two sentences in one string: takes both sentences
```

```
    ne = NE_wrapper("Hi there. Hello man.")
```

```
#!/usr/bin/python
from nltk.corpus import qc
from nltk import NaiveBayesClassifier, ConfusionMatrix
from nltk.classify import accuracy
from operator import itemgetter
from nltk import FreqDist
def feats(snt, genCat):
   toks = snt.lower().split(" ")
   result = dict()
   # General Category can be used as an input feature
   # (according to exam assignment)
   result['gencat'] = genCat
   result['gencat1'] = genCat
   result['gencat2'] = genCat
   result['gencat3'] = genCat
   # Use sentence length as feature (up 2%)
   result['len'] = len(toks)
   # Unigram Dimensions (word in sentence?)
   str= "color red white blue green wombat bat " \
       "dog dogs far close long cat cats tiger tigers " \
       "lion lions worm bird birds reptile reptiles " \
       "amphibian what which who how when why whom whose " \
       "where many much count animal animals "
   str="what abbreviation does stand mean how why causes " \
      "kind animal animals leg contains body color colors " \
      "film novel movie book money currency fear disease name " \
      "war world drink soft instrument play musical language " \
      "spoken languages letter vowel plant flowers product " \
      "religion brand religions god goddess sport game play " \
      "made substance elements sign symbol way ways used term " \
      "called ship enlish word letters words company team " \
      "profession occupation tiltle living city capital U.S." \
      "cities country countries mountain highest world" \
      "mountains peak rivers tallest range river find state " \
      "states number phone code information area many people " \
      "date far tall long chapter verses population number " \
      "percentage watch children average old fast speed escape " \
      "spacecraft temperatured hot baking oven diameter "
   # Unigram Dimensions in toks? True/False
   for w in str.split():
      result[w + "-present"] = w in toks
   # Unigram Dimension (last token before question mark) in toks[-2]
   result['preplast'] = toks[-2] in ("in", "from", "to", "for", "of")

   # Bigram Dimensions in (toks[0] toks[1])? True/False
   # result['whis'] = (toks[0] in ("what", "who", "where") and
   #               toks[1] in ("'s", "is", "are", "were", "was", "does"))
   # result['inwh'] = (toks[1] in ("what", "which", "whose", "whom")
   #               and toks[0] in ("in", "by", "from", "to"))
   result['b1'] = (toks[0] in ("what") and toks[1] in ("'s", "is"))
   result['b2'] = (toks[0] in ("what") and toks[1] in ("does"))
   result['b3'] = (toks[-3] in ("made") and toks[-2] in ("of"))
   result['b4'] = (toks[-3] in ("stand") and toks[-2] in ("for"))
   result['b5'] = (toks[0] in ("how") and toks[1] in ("do"))
   result['b6'] = (toks[0] in ("how") and toks[1] in ("does"))
   result['b7'] = (toks[0] in ("why") and toks[1] in ("is"))
   result['b8'] = (toks[0] in ("why") and toks[1] in ("does"))
   result['b9'] = (toks[0] in ("what") and toks[1] in ("does"))
   result['b10'] = (toks[0] in ("what") and toks[1] in ("color"))
   result['b11'] = (toks[0] in ("what") and toks[1] in ("a"))
   result['b12'] = (toks[0] in ("what") and toks[1] in ("was"))
   result['b13'] = (toks[0] in ("what") and toks[1] in ("instrument"))
   result['b14'] = (toks[0] in ("what") and toks[1] in ("language"))
   result['b15'] = (toks[0] in ("what") and toks[1] in ("letter"))
   result['b16'] = (toks[0] in ("what") and toks[1] in ("are"))
   result['b17'] = (toks[0] in ("what") and toks[1] in ("game"))
   result['b18'] = (toks[0] in ("what") and toks[1] in ("sport"))
   result['b19'] = (toks[0] in ("what") and toks[1] in ("word"))
   result['b20'] = (toks[0] in ("who") and toks[1] in ("was"))
   result['b21'] = (toks[0] in ("what") and toks[1] in ("'s","is"))
   result['b22'] = (toks[0] in ("what") and toks[1] in ("city"))
   result['b23'] = (toks[0] in ("what") and toks[1] in ("country"))
   result['b24'] = (toks[0] in ("what") and toks[1] in ("mountain"))
   result['b25'] = (toks[0] in ("where") and toks[1] in ("'s","is"))
   result['b26'] = (toks[0] in ("where") and toks[1] in ("can"))
   result['b27'] = (toks[0] in ("what") and toks[1] in ("state"))
   result['b28'] = (toks[0] in ("what") and toks[1] in ("number"))
   result['b29'] = (toks[0] in ("how") and toks[1] in ("many"))
   result['b30'] = (toks[0] in ("when") and toks[1] in ("did"))
   result['b31'] = (toks[0] in ("when") and toks[1] in ("was"))
   result['b32'] = (toks[0] in ("how") and toks[1] in ("much"))
   result['b33'] = (toks[0] in ("what") and toks[1] in ("percentage"))
   result['b34'] = (toks[0] in ("how") and toks[1] in ("long"))
   result['b35'] = (toks[0] in ("how") and toks[1] in ("fast"))
   result['b36'] = (toks[0] in ("how") and toks[1] in ("big"))
   return result
def prep(tuples):
```

Task 3: Learn me!

Take a look at the NLTK corpus of categorizing questions: `nltk.corpus.qc`. Its contents can be accessed via the method `tuples(filename)`, which returns a list of `(output, input)` tuples, where `input` is the question string and `output` is the category of that question. The parameter filename can be "train.txt" or "test.txt". For example:

```
>>> from nltk.corpus import qc
>>> print qc.tuples("train.txt")[0]
('DESC:manner', 'How did serfdom develop in and then leave Russia ?')
```

Your task is to write a script that trains a classifier on the training set (from "train.txt") to predict the category of the questions (e.g. "DESC:manner") based on the sentence and the general category (e.g. "DESC") -- i.e., the general category can be used as an input feature. Then apply it to the test set. Your script must then report the accuracy of the classifier and print out its confusion matrix.

The classifier can be anything (e.g. NLTK's Naive Bayes classifier / Decision Tree / MaxEnt / Conditional Random Field -- anything you like) and you may use whichever features you like (take a look at the corpus itself for inspiration: http://cogcomp.cs.illinois.edu/Data/QA/QC/).

NB! For just solving the task you will get a maximum of 1 point. To get the full 1.5 points the accuracy of the classifier must be higher than 0.66 >:-)

The **hand-in** is your script running the whole training/testing/result displaying process.

```
    #return [(feats(sentence, label.split(":")[0]), label)
    #     for (label, sentence) in tuples]
    result=[]
    for (label, sentence) in tuples:
       label_0 = label.split(":")[0]
       features = feats(sentence.lower(), label_0)
       result += [(features, label)]
    return result
if __name__ == "__main__":
    # http://nullege.com/
    #get data
    train = qc.tuples("train.txt")
    test = qc.tuples("test.txt")
    # get an idea of the needed features
    sorted_by_first = sorted(train, key=lambda tup: tup[0])
    for tup in sorted_by_first[:5]:
       #print "0 ", tup
       pass
    freqdists={}
    iter_first = 0
    for iter in range(1,len(sorted_by_first)):
       if sorted_by_first[iter-1][0]!=sorted_by_first[iter][0]:
          iter_last = iter-1
          subset=sorted_by_first[iter_first:iter_last]
          subset_class=map(lambda (x, y): x, subset)
          subset_strings = map(lambda (x, y): y, subset)
          subset_string=' '.join(subset_strings)
          subset_list=subset_string.split()
          #print "1 ", subset
          #print "2 ", subset_class
          #print "3 ", subset_strings
          #print "4 ", subset_string
          #print "5 ", subset_list
          fd=FreqDist(subset_list)
          #print iter_first, iter_last, subset_class[0], "\n"
          #print fd.items()
          freqdists.update({subset_class[0]:(
             fd.items(),iter_last-iter_first)})
          #freqdists.update({subset_class[0]: fd})
          #print filter(lambda k: fd[k] > 3, fd.keys())
          iter_first = iter
    #print "6 ", freqdists.items()
    stoplist=[u'?',u'the', u',']
    freqdistlist=freqdists.items()
    freqdistlist=sorted(freqdistlist, key=lambda x: x[0])
    for c, t in freqdistlist:
       l=t[0]
       l_sort=sorted(l, key=lambda x: x[1], reverse=True)
       #print "7 ",l_sort
       for w in l_sort[:15]:
          if not w[0] in stoplist:
             print "8 ", c, t[1],w
    #make features
    trainSet = prep(train)
    testSet = prep(test)
    #train classifier
    nbc = NaiveBayesClassifier.train(trainSet)
    #determine accuracy on test set
    print accuracy(nbc, testSet)
    #apply classier on test set
    goldTags = map(lambda (x, y): y, testSet)
    hypTags = map(lambda (x, y): nbc.classify(x), testSet)
    #print testSet[:3]
    #print goldTags
    #print hypTags
    #gold labels vs. classifier labels
    print ConfusionMatrix(goldTags, hypTags).\
             pretty_format(sort_by_count=True,
             show_percents=True, truncate=5)
```

File:T1.py

```
import nltk
from nltk.corpus import treebank
from nltk.grammar import Nonterminal
# enter the given correct sentences into this list
correctSentences = [
    "dog smell distracts",
    "people smell fries",
    "people smell french fries",
    "a dog sleeps",
    "the dog fries sausages",
    "the french fries smell"
    ]
# enter the given wrong sentences into this list
wrongSentences = [
    "people sleeps",
    "a dog smell",
    "a dog sleeps fries",
```

Task 4: Parse me!

Take a look at the python script `parse.py`. It contains two lists of sentences: `correctSentences` and `wrongSentences`.

Your task is to design a probabilistic context-free grammar (PCFG) and create a parser based on that grammar that would parse the correct sentences and reject (fail to parse) the wrong sentences. Use NLTK's `ViterbiParser` and its method `parse(listOfTokens);` while designing the grammar the method `draw(parseTree)` might be helpful.

NB! The output trees have to be syntactically correct in case of ambiguity -- the parser has to assign a higher probability to the correct parse (e.g. it would be wrong to parse "fruit flies like a banana" and "time flies like an arrow" with the same tree structure).

The non-terminal symbols don't have to be the classic phrase types (NP, VP, ...) or PoS-tags (N, V, Adj, ...), you can write them as you want (N / Noun / NN / ...) and they can be anything you want (e.g. PoS-tags with some morphological info like `Npl` = plural noun, or completely different non-standard categories and symbols), as long as the correct sentences are accepted (parsed) and the wrong sentences are rejected (failed to parse).

The **hand-in** is your script with a grammar, a parser, and a loop through the sentences, showing which are accepted/rejected.

```
    "a french fries smell"
    ]
# determine the pos tags
if 0:
    for snt in correctSentences:
        input_sentence = snt
        text = nltk.word_tokenize(input_sentence)
        list_of_tokens = nltk.pos_tag(text)
        print list_of_tokens
# /home/benzro/nltk_data/corpora/treebank/combined/wsj_0000.mrg
# enter the tree structure
# distinguish between plural(p) and singular(s)
if 1:
    treebankparsedsents=treebank.parsed_sents()[:6]
    print len(treebankparsedsents)
#determine the grammar productions
if 1:
    tbank_production_list = [production for sent in
                treebankparsedsents for production in
                sent.productions()]
    print tbank_production_list
    print
# determine the grammar probabilities
if 1:
    tbank_prob_grammar = nltk.induce_pcfg(Nonterminal('S'),
                tbank_production_list)
    print tbank_prob_grammar
# copy the grammar into this function
# remove the points in S-> .
# sort
# improve / remove errors
if 1:
    g = nltk.PCFG.fromstring("""
    S -> NPp VPp [0.5]
    S -> NPs VPs [0.5]
    VPp -> VBp NPp [0.666667]
    VPp -> VBp [0.333333]
    VPs -> VBs [0.75]
    VPs -> VBs NPp [0.25]
    NPs -> JJ NNs [0.333333]
    NPs -> DTs NNs [0.666667]
    NPp -> DTp JJ NNp [0.166667]
    NPp -> NNp [0.666667]
    NPp -> JJ NNp [0.166667]
    JJ -> 'french' [0.666667]
    JJ -> 'dog' [0.333333]
    DTp -> 'the' [1.0]
    DTs -> 'a' [0.5]
    DTs -> 'the' [0.5]
    VBp -> 'smell' [1.0]
    VBs -> 'fries' [0.333333]
    VBs -> 'sleeps' [0.333333]
    VBs -> 'distracts' [0.333333]
    NNp -> 'sausages' [0.166667]
    NNp -> 'fries' [0.5]
    NNp -> 'people' [0.333333]
    NNs -> 'dog' [0.666667]
    NNs -> 'smell' [0.333333]
    """)
# make viterbi parser with grammar
if 1:
    p = nltk.ViterbiParser(g)
# apply parser to all sentences, wrong and correct
if 1:
    print [(correctSentences, "correct"), (wrongSentences, "wrong")]
    for (sntList, sntGrpId) in [(correctSentences, "correct"), (wrongSentences,
"wrong")]:
       print "===========\n" + sntGrpId + "\n==========="
       for snt in sntList:
          print snt
          flag=0
          for tree in p.parse(snt.split()):
             flag=1
             print tree
          if flag==0:
             print "parse failed"
          print
```

PCL II – Zusammenfassung Code

Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

Task 2: Make me!

Your task is to create a python module with the functionality for PoS-tagging English sentences. The module must allow the user to tokenize a sentence string into a list of tokens, and to tag the sentence with its part-of-speech sequence. Feel free to use `nltk.pos_tag` and `nltk.word_tokenize` functions, or external taggers such as TreeTagger. It is up to you to choose, whether you want the model to contain functions or class(es), as long as

- they are properly documented: input/output parameters, functionality
- the functionality is nicely organized into functions or class methods
- your implementation is safe and follows the principles of duck typing, with handling/raising exceptions in case the input to the functions/methods is something erroneous or unexpected

Also write a standalone python script that imports your newly created module and applies the functionality to a couple of sample sentences and displays the results. It must catch the possible exceptions and display friendlier error messages.

The task is not to do good PoS-tagging, but to show that you can create a usable python module, done in a nice coding style, with organization and documentation, handling exceptional situations and wrong types of input gracefully.

The **hand-ins** are the two script files: the module file and the sample usage file.

```python
#!/usr/bin/python
#-*- coding: utf-8 -*-
DESC="Displays the 5 sentences that have the maximum number " \
     "of Hapax Legomena (words that appear only once) in a Document"
import sys
import nltk
import argparse#!add
from nltk.tokenize import word_tokenize #!add
def parse_command_line():
    parser = argparse.ArgumentParser(description=DESC)
    parser.add_argument('-i', '--input',
                        type=argparse.FileType('r'),
                        default=sys.stdin,
                        metavar='FILE',
                        help="Input file");
    # return parser
    args = parser.parse_args()#!add
    return args#!add
def getHapaxLegomena(rawDoc):
    #docTokList = tokenize(rawDoc)
    #docTokList = word_tokenize(rawDoc)
    docTokList = tokenizeText(rawDoc) #!corr
    #print "3: ",docTokList[:30]
    fdist = {}
    for tok in docTokList:
        if tok in fdist:
            fdist[tok] += 1
        else:
            fdist[tok] = 1

    #HapaxLego = [tok for tok in fdist if fdist[tok] != 1] #! logic
    HapaxLego = [tok for tok in fdist if fdist[tok] == 1] #!corr
    return HapaxLego
def tokenizeText(text):
    tokenizer = nltk.tokenize.TreebankWordTokenizer()#!tab/indentation
    tokenList = tokenizer.tokenize(text.lower())#!tab/indentation

    return tokenList#!tab/indentation
def main(args):
    if args.input:
        #print "2: ", args.input
        #get list of Hapax Legomena in a document
        doc=args.input.read()
        #print "2: ",doc[:100]
        docHapaxLego = getHapaxLegomena(doc)
        #print "2: ",docHapaxLego[:20]
        #change the file object position to the beginning of
        # the file
        #f.seek(offset, from_what) https://pynlp.wordpress.com/2013/10/
        # from_what=0(begin)
        #from_what=1(current pos)
        #from_what = 2(end)
        #args.input.seek(-5, 2)#54321  5 from end
        args.input.seek(0,0) #!corr
        sentHapaxInfo = {}#!corr
        for n, rawSent in enumerate(args.input):
            #print "2: ", n, rawSent
            #sentHapaxInfo = {} #!wrong position
            #sent = tokenize(rawSent)
            sent = tokenizeText(rawSent) #!corr
            count = 0
            sentHapax = []
            #updates info of tokens in the sentence that are
            # in the Hapax Legomena list
            for tok in sent:
                #if not tok in docHapaxLego:
                if tok in docHapaxLego:
                    count += 1
                    sentHapax.append(tok)
            #sentHapaxInfo[n] = []
            sentHapaxInfo[n] = {} #!corr
            sentHapaxInfo[n]["total"] = count #!tab/indentation
            sentHapaxInfo[n]["hapax"] = sentHapax
        #sort the dictionary according to the total number
        # of Hapax Legomena
        #print "2: ", sentHapaxInfo
```

Task 1: iBroke

Take a look at the python script in the file `ibroke.py`. It is supposed to:

- load text data from a file given as an argument
- tokenize the text data and get a list of Hapax Legomena (words that appear only once in the document)
- compute for each sentence the total number of Hapax Legomena that they contain
- display the 5 sentences that have the highest number of Hapax Legomena

You will notice that the program does not work -- the script contains errors. Some of these will be compile-time and run-time errors and will not let the program finish. Others might be logical errors, which means that the program finishes but something is wrong with the output.

Your task is to fix all those errors: go through the code and make sure that the program runs and does what is expected of it.

The **hand-in** is a working script file with the errors fixed. Please add comments that describe the errors that you corrected.

```
        #freqList = sorted(sentHapaxInfo.iteritems(),
        #                  key=lambda (x, y): y["total"],
        #                  reverse=False) #! reverse
        freqList = sorted(sentHapaxInfo.iteritems(),
                          key=lambda (x, y): y["total"],
                          reverse=True) #!corr
        #print "2: ", freqList
        #diplay the top 5 sentences with the highest number
        # of Hapax Legomena in a Document
        print "Sent\tTotal"
        for n, hapax_info in freqList[:5]:
            #print "%d\t%d" % (n, hapax_info["total"])
            print "%5d\t%d" % (n, hapax_info["total"])#!corr

if __name__ == '__main__':
    #shakespeare-macbeth.txt
    #Ctrl D
    args = parse_command_line()
    #print "1: ", args
    #print "1: ", args.input.read()[:100]
    #print "1: ", args.input.readline()
    main(args)
```

```
#!/usr/bin/python
import nltk
import sys
import os
#Alle S zuoberst, sonst geht es nicht.
grammarStr = """
    S -> '.'
    S -> S '.' NP VP '.'
    S -> NPp VPp '.'
    S -> NPs VPs '.'
    S -> NP VP '.'
    S -> NP NP
    S -> NP VP
    VP -> 'VERB' NP PP PP
    NP -> 'DET' 'NOUN' 'VERB' 'NOUN'
    VPs -> 'VERB' NPp
    NPp -> 'DET' 'ADJ' 'NOUN'
    ADVP -> 'ADV'
    VP -> 'VERB' NP ADVP
    NPp -> 'NOUN'
    NP -> NP PP
    PP -> 'ADP' ADVP
    NP -> 'DET' 'ADJ' 'NOUN'
    NP -> NP '.' ADJP '.'
    VP -> 'VERB' NP
    NP -> NP '.' NP '.'
    NPs -> 'ADJ' 'NOUN'
    ADVP -> NP 'ADP'
    PP -> 'ADP' NP
    VP -> 'VERB' S
    NP -> 'NOUN' 'NOUN' 'NOUN'
    VP -> 'VERB' PP
    NP -> 'PRON' 'NOUN' 'NOUN' 'NOUN'
    NP -> 'ADJ' 'NOUN'
    VP -> 'VERB' PRT ADVP
    PP -> 'ADP' S
    NP -> NP 'NOUN' 'NOUN' 'NOUN'
    ADJP -> 'ADJ' S
    VPp -> 'VERB'
    VP -> 'VERB' NP PP NP
    NP -> 'DET' 'ADJ' 'ADJ' 'NOUN'
    VP -> 'VERB' VP
    NP -> NP ADJP
    NP -> 'DET' 'NOUN' 'NOUN'
    NP -> NP '.' NP
    VP -> 'ADP' VP
    NP -> 'NOUN' 'NOUN' 'NOUN' 'NOUN'
    VPp -> 'VERB' NPp
    VP -> 'VERB' NP PP
    VP -> 'VERB' NP PP ADVP
    NP -> NP PP PP
    NPp -> 'ADJ' 'NOUN'
    NP -> 'NOUN' 'NOUN'
    NP -> ADJP 'NOUN' 'NOUN'
    ADJP -> NP 'ADJ'
    NP -> 'ADV' 'ADJ' 'NOUN'
    NP -> 'DET' 'NOUN'
    ADVP -> NP 'ADJ'
    VPs -> 'VERB'
    NP -> 'DET'
    VP -> 'VERB' NP S
    WHNP -> 'DET'
    NP -> 'NOUN'
    ADJP -> 'ADV' 'ADJ'
    ADJP -> 'ADJ' 'ADJ'
    PRT -> 'PRT'
```

Task 2: iParse

Here you will construct a context-free grammar to parse as many sentences from the Brown corpus as you can, by expanding the script `iparse.py`. The script defines a very simple grammar, and uses it to parse sentences from the Brown corpus; to make the approach simpler and more general, the parts of speech are defined as the terminal symbols of the grammar (so instead of writing rules like NP → NOUN, NOUN → "cat", NOUN → "dog", etc. you will write rules like NP → "NOUN", skipping the actual words). The sentences from the Brown corpus are filtered -- only the sentences with length between 3 and 20 words with at least one verb in them are included.

In addition to parsing the Brown sentences, the script has a list of bad sentences that are supposed to be rejected by the grammar. This ensures that the grammar does not over-generalize and include false-positives.

Your task is to design a grammar that accepts at least 1000 Brown sentences AND none of the bad sentences in order to get the full points for this task. The **hand-in** is the changed script `iparse.py`, with the updated grammar, satisfying these conditions.

```
    NPs -> 'DET' 'NOUN'
    NP -> 'PRON'
    NP -> '.'
"""
# grammarStr = """
#       S -> NPs VPs "."
#  S -> NPp VPp "."
#      S -> NP VP "."
#  NP -> "DET" "NOUN"
#  NP -> "NOUN" "NOUN"
#      NP -> "DET" "ADJ" "NOUN"
#  NPp -> "NOUN"
#      NPp -> "DET" "ADJ" "NOUN"
#      NPp -> "ADJ" "NOUN"
#      NPs -> "DET" "NOUN"
#      NPs -> "ADJ" "NOUN"
#      VP -> "VERB" VP
#      VP -> "VERB" NP PP NP
#      VPs -> "VERB" NPp
#      VPs -> "VERB"
#      VPp -> "VERB" NPp
#      VPp -> "VERB"
#      PP -> "ADP" NP
#      ADJP -> NP "ADJ"
#    """
# grammarStr = """
#    S -> NP VP | SGN
#    SGN ->  "." | S SGN
#    NP -> "NOUN" | "DET" NP | "NOUN" NP | "PRON" NP
#    VP -> "VERB" NP
#    """
# grammarStr = """
#    S -> PNP VP | VP | "VERB" PNP VP | S "." | S "!" | S "?"
#    NP -> "NOUN" | "DET" NP | "ADJ" NP | "NUM" "NOUN" | "NUM" "ADJ" NP | "ADP" NP
#    PNP -> "PRON" | NP
#    VP ->  "VERB" | "VERB" NP | "VERB" PNP | "ADV" VP | VP "ADV" | VP "PRT"
#    """
# grammarStr = """
#    S -> NP VP "."
#    NP -> "NOUN" | "DET" NP | "NOUN" NP | "PRON" NP
#    VP -> "VERB" NP
#    """
# choose grammar library
try:
   grammar = nltk.CFG.fromstring(grammarStr)
   print "0: new library"
except:
   grammar = nltk.parse_cfg(grammarStr)
   print "0: old library"
#bottom up parser (cannot handle left hand recursion)
#parser = nltk.parse.chart.BottomUpChartParser(grammar)
parser = nltk.ChartParser(grammar)
def loadMap(filename):
   fileHandle = open(filename, 'r')

   resultMap = dict()

   for line in fileHandle:
      fullTag, shortTag = line.rstrip().split()

      resultMap[fullTag] = shortTag

   fileHandle.close()

   return resultMap
def getListOfPos(rawSnt):
   global mapOfPos

   #return [mapOfPos[tag] if tag in mapOfPos else 'NOUN'
   #     for tag in zip(*rawSnt)[1]]
   ret = []
   tags = zip(*rawSnt)[1]
   #print "10.1", tags
   for tag in tags:
      if tag in mapOfPos:
         terminal = mapOfPos[tag]
         ret.append(terminal)
      else:
         terminal = 'NOUN'
         ret.append(terminal)
   #print "10.2", ret
   return ret
def parse(snt):
   global parser

   try:
      return list(parser.parse(snt))
   except:
      return False
# loads the file: 'en-brown.map'
# mapping of brown-tag_set -> exam-tag_set
mapOfPos = loadMap(os.path.join(os.path.dirname
                        (os.path.realpath(__file__)),
                        'en-brown.map'))
if __name__ == "__main__":
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
# test brown corpus
#------------------
#testlen = 0
count = 0
total = 0
print "1: ", set(mapOfPos.values()), "\n"
# go through the brown corpus tagged sentences
#print "2: total sents: ", len(nltk.corpus.brown.tagged_sents())
i=0
for rawSnt in nltk.corpus.brown.tagged_sents()[:2100]:
   print i
   i+=1
   #posSnt = getListOfPos(rawSnt)
   #test only sentences between 3 and 20 words
   #if len(posSnt) > 2 \
   if len(rawSnt) > 2 and len(rawSnt) < 21:
      # mapping of sents -> brown-tag_set -> exam-tag_set
      posSnt = getListOfPos(rawSnt)
      #print "2: ", posSnt
      #if (len(rawSnt) != len(posSnt)): testlen += 1
      # parse only sents with a verb
      if u"VERB" in posSnt:
         total += 1
         # count only those accepted by the parser
         if parse(posSnt):
            #print "3: ", posSnt
            #print "3: ", rawSnt, "\n"
            count += 1
         else:
            #print "4: ", posSnt
            #print "4: ", rawSnt, "\n"
            pass
#print "3: ", testlen
print "accepted good sentences: {0} / {1}".format(count, total)
# test bad sentences
# ------------------
count = 0
total = 0

badSents = [
   # DET NUM is a bad sequence
   'NOUN VERB DET NUM .'.split(),

   # NOUN DET NOUN is a bad noun phrase
   'NOUN DET NOUN VERB NOUN .'.split(),

   # NOUN PRON is a bad noun phrase
   'NOUN PRON NOUN VERB NOUN .'.split(),

   # ADV VERB ADV VERB is a bad verb phrase
   'NOUN ADV VERB ADV VERB NOUN .'.split(),

   # DET PRON is a bad noun phrase
   'DET PRON NOUN VERB NOUN .'.split(),

   # DET should precede some word, unlike here
   'VERB NOUN DET .'.split()]
# go through the bad sentences
for badSnt in badSents:
   # test all bad sentences
   total += 1
   # count only sentences which are accepted by the parser
   if parse(badSnt):
      print 'Bad sentence accepted:', badSnt
      count += 1

print "accepted bad sentences: {0} / {1}".format(count, total)
```

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import nltk
from nltk.corpus import nps_chat
from nltk.tokenize import wordpunct_tokenize
from nltk import NaiveBayesClassifier as nltk_classifier
from nltk.classify.util import accuracy
def post_features(postText):
    # In which we do hardcore feature engineering
    # We're trying to classify IMs into speech act types
    # This does not present an excessively fertile ground for
    # exploiting classical things like frequency
    # So we classify based on patterns and polarity items
    # Conveniently the latin alphabet comes with punctuation for
    # questions and exclamations so we use that too
    features = {}
    #list of tokens of one post
    # !!!! hier alles KLEIN schreiben ----->
    words = map(lambda w: w.lower(), wordpunct_tokenize(postText))
    # #print "5.1:\n", words
    wordset = set(words)
    # build set with class specific words
```

Task 3: iLearn

Here you will focus on the NLTK's NPS Chat Corpus, which consists of posts from instant messaging sessions. These posts have all been labeled with one of 15 dialogue act types, such as "Statement," "Emotion," "ynQuestion", and "Continuer". You will create a classifier that will predict these dialogue act types for given posts. You can find more details and some code in chapter 6, section 2.2 of the NLTK book (http://www.nltk.org/book/ch06.html).

The corpus has 15 file IDs:

```
from nltk.corpus import nps_chat
print nps_chat.fileids()
```

you will take the file with the first ID for testing and the rest for training:

```
allFileIds = sorted(nps_chat.fileids())
testFileId = allFileIds[0]
trainFileIds = allFileIds[1:]
```

Your task is to write a script that will load the corpus, split it into training/test sets as shown here, represent the train/test sets with features, train a classifier on the training set and print out its accuracy on the testing set. **NB!** To get the full points your classifier must have the accuracy of at least 0.77. The **hand-in** is the script.

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    # !!!! ---> dann auch hier alles KLEIN schreiben
    system = set(['join','join', '.action'])
    greet = set(['hey', 'hi', 'yo', 'user'])
    bye= set(['bb', 'bye', 'cya', 'l8r', 'brb'])
    emotion = set(['boo', 'lol',':P',':)', 'boo.'])
    question=set(['?'])
    accept=set(['sounds', 'good'])
    anywords = set(['any', 'anybody', 'anything',
                    'anytime', 'anywhere'])
    somewords = set(['some', 'somebody', 'something',
                 'sometime', 'somewhere'])
    whwords= set(['what', 'where', 'how'])
    exclamation = set(['!'])
    comma= set([','])
    positive = set(['yeah','yes', 'sure' 'ya', 'yep', 'ok'])
    negative = set(['no', 'nope', 'not'])
    # over all features
    features['f_initial_word'] = words[0]
    #features['f_length'] = len(words)>11
    try:
        features['f_second_word'] = words[1]
        features['f_last_word'] = words[len(words)]
    except:
        pass
    features['f_exclamation'] = True if \
        any(feat in wordset for feat in exclamation) else False
    features['f_question'] = True if \
        any(feat in wordset for feat in question) else False
    features['f_anyword'] = True if \
        any(feat in wordset for feat in anywords) else False
    features['f_someword'] = True if \
        any(feat in wordset for feat in somewords) else False
    features['f_positive'] = True if \
        any(feat in wordset for feat in positive) else False
    features['f_negative'] = True if \
        any(feat in wordset for feat in negative) else False
    features['f_bye'] = True if \
        any(feat in wordset for feat in bye) else False
    features['f_whwords'] = True if \
        any(feat in wordset for feat in whwords) else False
    features['f_comma'] = True if \
        any(feat in wordset for feat in comma) else False
    # specific features
    features['f_system'] = True if \
        any(feat in wordset for feat in system) else False
    features['f_greet'] = True if \
        any(feat in wordset for feat in greet) else False
    features['f_emotion'] = True if \
        any(feat in wordset for feat in emotion) else False
    features['f_accept'] = True if \
        any(feat in wordset for feat in accept) else False
    # #Cazim
    # wordset = set(words)
    # anywords = set(['any', 'anybody', 'anything',
    #                 'anytime', 'anywhere'])
    # somewords = set(['some', 'somebody', 'something',
    #              'sometime', 'somewhere'])
    # features['initial_word'] = words[0]
    # features['exclamation'] = True if "!" in wordset else False
    # features['question'] = True if "?" in wordset else False
    # features['bye'] = False if wordset.isdisjoint\
    #     (set(['bb', 'bye', 'cya', 'l8r'])) else True
    # features['negative'] = True if "no" in wordset else False
    # features['any'] = False if wordset.isdisjoint(anywords) else True
    # features['some'] = False if wordset.isdisjoint(somewords) else True
    # features['positive'] = True if "yeah" in wordset else False
    # """ test if not disjoint -> have common elements
    # In[111]: not set([2,3,4,5]).isdisjoint(set([2,3]))
    # Out[111]: True
    # In[112]: not set([4,5]).isdisjoint(set([2,3]))
    # Out[112]: False"""
    return features
def train_features(feats):
    classifier = nltk_classifier.train(feats)
    return classifier

def test_classifier(clf, feats):
    clf.show_most_informative_features(30)
    print('accuracy: ' + str(accuracy(clf, feats)))
if __name__ == "__main__":
    # split the corpus
    allFileIds = sorted(nps_chat.fileids())
    testFileId = allFileIds[0]
    trainFileIds = allFileIds[1:]
    # print "1:\n", testFileId, trainFileIds
    # get the contents from the two parts
    # print "2.1:\n", (nltk.corpus.nps_chat.words()[:20])
    # print "2.2:\n", (nltk.corpus.nps_chat.tagged_words()[:20])
    # print "2.3:\n", (nltk.corpus.nps_chat.tagged_posts()[:5])  #
    testPosts = nps_chat.xml_posts(testFileId)
    trainPosts = nps_chat.xml_posts(trainFileIds)
    # print "3.1:\n", testPosts[123]
```

```
    # print "3.2:\n", trainPosts[123]
    posts = nltk.corpus.nps_chat.xml_posts()
    # print "3.3:\n", len(posts)#
    # print "3.4:\n", [posts[p].text for p in range(0,5)]#
    # print "3.5:\n", [posts[p].text for p in range(0, 5)][1]#
    # print "3.6.1:\n", [posts[p].attrib['class']
    #                   for p in range(0, 5)]
    # print "3.6,2:\n", [posts[p].get('class')
    #                   for p in range(0, 5)]
    # print "3.7:\n", set([posts[p].attrib['class']
    #                   for p in range(0,len(posts))])
    # print "3.8:\n", [posts[p].attrib['user']
    #                   for p in range(0, 5)]
    # print "3.9:\n", sorted(
    #     nltk.FreqDist(p.attrib['class'] for p in posts).items(),
    #     key=lambda x: x[1], reverse=True)
    """3.9:
    [('Statement', 3185), ('System', 2632), ('Greet', 1363),
    ('Emotion', 1106), ('ynQuestion', 550), ('whQuestion', 533),
     ('Accept', 233), ('Bye', 195), ('Emphasis', 190),
     ('Continuer', 168), ('Reject', 159), ('yAnswer', 108),
     ('nAnswer', 72), ('Clarify', 38), ('Other', 35)]"""
    # n=200
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Statement':
    #         print 'Statement', p.text
    # # for p in posts[:n]:
    #     if p.attrib['class'] == 'System':
    #         print 'System', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Greet':
    #         print 'Greet', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Emotion':
    #         print 'Emotion', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'ynQuestion':
    #         print 'ynQuestion', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'whQuestion':
    #         print 'whQuestion', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Accept':
    #         print 'Accept', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Bye':
    #         print 'Bye', p.text
    # for p in posts[:n]:
    #     if p.attrib['class'] == 'Emphasis':
    #         print 'Emphasis', p.text
    # make the featuresets
    # FeaturesSet=[({features},label),({features},label), ...]
    testFeatureSets = [
        (post_features(post.text),post.get('class'))
        for post in testPosts]
    trainFeatureSets = [
        (post_features(post.text),post.get('class'))
        for post in trainPosts]
    # print "4.1:\n", testFeatureSets[:2]
    # print "4.2:\n", trainFeatureSets[:2]
    # train and test
    classifier = train_features(trainFeatureSets)
    test_classifier(classifier, testFeatureSets)
    print nltk.classify.accuracy(classifier, testFeatureSets)
```

```
from collections import defaultdict
import string
import unicodedata
# Basic levenshtein code copied from lecture materials
# Idea for transposition from wikipedia:
# http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance
def edit_distance(xs, ys):
    xs = map(lambda x: unicode(x), xs)
    ys = map(lambda x: unicode(x), ys)
    punctuation = map(lambda x: unicode(x), string.punctuation)
    # Initialize matrix for calculating distances
    matrix = defaultdict(lambda: defaultdict(int))
    for i, x in enumerate([None] + list(xs)):
        for j, y in enumerate([None] + list(ys)):
            if i > 0 and j > 0:
                # Calculate the replacement cost
                # Replacement is free if we're replacing something with itself
                if x == y:
                    replCost = 0.0 + matrix[i - 1][j - 1]
                else:
                    # Replacing punctuation with other punctuation costs 0.1
                    if x in punctuation and y in punctuation:
                        replCost = 0.1 + matrix[i - 1][j - 1]
```

Task 4: iMeasure

In the lectures and labs you have made acquaintance with the Levenshtein distance, which measures the minimum required number of edit operations (character deletion/insertion/replacement) to turn one sequence of characters into another. Your task here is to implement **generalized edit distance** with **word-level** operations and apply it in practice.

Generalized edit distance differs from the Levenshtein distance in the following way:

- any arbitrary edit operations can be specified (unlike just insertion/deletion/ substitution in the Levenshtein distance)
- each edit operation has a cost associated with it, which can be any float number (unlike the cost of 1 for all three operations in the Levenshtein distance)

Just like the Levenshtein distance, the generalized edit distance is implemented via dynamic programming and finds the minimum total cost sum for editing one string to turn it into the other.

Your task is to implement the generalized edit distance for sequences of words, with the following word-level operations and costs:

1. word insertion, with the cost depending on the word: 0.1 for punctuation and 3.0 for everything else; for example:

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
                        # Replacing a number with another number costs 4.0
                    elif x.isdigit() and y.isdigit():
                        replCost = 4.0 + matrix[i - 1][j - 1]
                    # Replacing an alphanumeric string with another costs 1.3
                    # We have to be careful not to let numbers sneak in
                    elif x.isalnum() and y.isalnum() and not y.isdigit():
                        replCost = 1.3 + matrix[i - 1][j - 1]
                    # Any other replacement operation costs 16.0
                    else:
                        replCost = 16.0 + matrix[i - 1][j - 1]
                # Inserting and deleting cost 0.1 for punctuation
                # and 3.0 for everything else
                # We check for punctuation in xs for insertion
                # and ys for deletion --
                # this is because ys is the sequence operations
                # are being "done" on
                insCost = matrix[i - 1][j] + (
                0.1 if x in punctuation else 3.0)
                delCost = matrix[i][j - 1] + (
                0.1 if y in punctuation else 3.0)
                # We take the path of least resistance
                matrix[i][j] = min(replCost, insCost, delCost)
                # Transposition remains
                # First we check whether transposition is a viable
                # operation
                # i.e. whether character i-1 in xs corresponds to
                # j in ys and the other way around
                # Transposing a token with itself costs nothing,
                # everything else costs 0.4
                transpCost = matrix[i - 2][j - 2] + (
                0 if x == y else 0.4)
                if i > 1 and j > 1 and x == ys[j - 2] and xs[
                            i - 2] == y:
                    matrix[i][j] = min(matrix[i][j], transpCost)
            # If we run out of either sequence, we've only inserts
            # and deletes left
            # If j is shorter we do inserts
            # except for obvious reasons we do not decrement j, but i
            elif i > 0:
                matrix[i][j] = (
                               0.1 if x in punctuation else 3.0) + \
                               matrix[i - 1][j]
            # Same story here but with deletes
            elif j > 0:
                matrix[i][j] = (
                               0.1 if y in punctuation else 3.0) + \
                               matrix[i][j - 1]
            # We're done
            else:
                matrix[i][j] = 0
    return matrix[i][j]
if __name__ == "__main__":
    # print edit_distance("this is a long sentence",
    #                      "and here be one hell of a lot longer sentence")
    # print edit_distance(['this', 'is', 'a', 'long', 'sentence'],
    #                      ['and', 'here', 'be', 'one', 'hell', 'of',
    #                       'a', 'lot', 'longer' 'sentence'])
    # print edit_distance("abcde", "01234567890123456789012345678901234567890"
    #                               "123456789")
    print(edit_distance("ab", "ba"))
```

- the distance between [u'one', u'two', u'three', u'four'] and [u'one', u'two', u'X', u'three', u'four'] is 3.0
- the distance between [u'one', u'two', u'three', u'four'] and [u'one', u'two', u'+', u'three', u'four'] is 0.1

2. word deletion, with the cost depending on the word: 0.1 for punctuation and 3.0 for everything else, for example:
   - the distance between [u'one', u'two', u'three', u'four'] and [u'one', u'three', u'four'] is 3.0
3. word replacement, with the cost depending on whether the two words are punctuation, numeric or others (including letter-words and words with a mixture of punctuation, numbers and letters)
   - 0.1 replacing punctuation with different punctuation
   - 4.0 for replacing numbers with different numbers
   - 1.3 for replacing a letter-word or a mixed-character word with a different one of the same kind
   - 16.0 for replacing punctuation/numbers/other words with a word of a different kind (for example replacing a number with punctuation, or replacing a letter-word with a number)
   - for example, the distance between [u'one', u'two', u'three', u'four'] and [u'one', u'new', u'three', u'four'] is 1.3
   - and the distance between [u'one', u'2', u'three', u'four'] and [u'one', u'3', u'three', u'four'] is 4.0
   - and the distance between [u'one', u'!', u'three', u'four'] and [u'one', u'?', u'three', u'four'] is 0.1
4. transposition of two adjacent words (reordering two words standing next to each other), cost 0.4; for example:
   - the distance between [u'one', u'two', u'three', u'four'] and [u'one', u'three', u'two', u'four'] is 0.4
5. General example: the minimum distance between [u'one', u'two', u'three', u'.'] and [u'two', u'one', u'?'] is 3.5 = 0.4 (transposition of 'one' and 'two') + 3.0 (deletion of 'three') + 0.1 (replacement of '.' with '?')

Using the newly implemented word-level generalized edit distance in a script or the Python interpreter, answer the following questions:

- What is the total distance between ["word", "to", "word"] and ["word", "3", "word"] and why is it not 16.0?
- What is the closest sentence in the Brown corpus (nltk.corpus.brown.sents()) to the sentence [u'Each', u'binomial', u'trial', u'of', u'an', u'experiment', u'gives', u'either', u'1', u'or', u'0', u'?']? Please also include the code that you used to find the answer.

The **hand-in** is the answer to these two questions and your script with the implementation of this generalized edit distance.

```
File: aufgabe_2_muster
# !/usr/bin/env python
# -*- coding: utf-8 -*-
import os
import json
import ConfigParser
from os import path
class ConfigReader(object):
    def __new__(cls, fpath):
        if cls == ConfigReader:
            (root, ext) = os.path.splitext(fpath)
            if ext == ".json":
                return JsonReader(fpath)
            elif ext == ".ini":
                return IniReader(fpath)
            else:
                raise ValueError('Unrecognized file format: '
                                 'must be .json or .ini')
        else:
            return object.__new__(cls, fpath)
    def _read(self, fpath):
        return self._parser(fpath)
class JsonReader(ConfigReader):
    def __init__(self, fpath):
        def load_json_from_file(fp):
            with open(fp) as f:
                return json.load(f)
        self._parser = load_json_from_file
```

U1.2
2 Fortgeschrittene Klassen
Wir möchten in Python einen Paket-Manager für die Linux-Distribution Y schreiben. Die Distribution Y ist aus dem Zusammenschluss zweier kleinerer Distributionen entstanden. Beide dieser Distributionen hatten bis anhin einen eigenen Paket-Manager, die beide ein anderes Format für Paket-Konfigurationsdateien verwendeten, weshalb die Paketdatenbanken für Y Linux ein wildes Gemisch beider Formate sind.
Glücklicherweise werden beide Formate von der Python-Standardbibliothek unterstützt: die Hälfte der Dateien ist im JSON-Format, die andere im .ini-Format.
Um die Architektur unseres Systems eleganter zu gestalten, möchten wir gerne auf die Konfigurationsdateien formatunabhängig zugreifen können. Zu diesem Zweck sollst du eine Klasse ConfigReader schreiben, die ein solches Interface implementiert.
Deine Klasse soll beim Aufruf einen Dateinamen erhalten und das Format erkennen. Um die beiden unterschiedlichen Formate zu bearbeiten, sollst du zwei Subklassen definieren: JsonReader und IniReader. Diese Subklassen sollen Dateien des jeweiligen Formats einlesen und sich nach aussen wie Dictionaries der Schlüssel-Wert-Paare in der eingelesenen Datei verhalten (definiere dazu die Methode __getitem__). Die Oberklasse ConfigReader soll die Methode __new__ so definieren, dass je nach Format der übergebenen Datei die richtige Subklasse

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
        self._cfg = self._read(fpath)
    def __getitem__(self, key):
        return self._cfg[key]
class IniReader(ConfigReader):
    def __init__(self, fpath):
        def _init_parser(parser, fpath):
            parser.read(fpath)
            return parser.get
        self._parser = ConfigParser.ConfigParser()
        self._cfg = _init_parser(self._parser, fpath)
    def __getitem__(self, key):
        return self._cfg('package', key)
```

File: test_installer

```
# -*- coding: utf-8 -*-
import aufgabe_2_muster as a2
class Installer(object):
    """
    Mockup package installer class.
    """
    def __init__(self, cfgpath):
        self.cfg = a2.ConfigReader(cfgpath)
    def install(self):
        print 'Processing package: {}'.format(self.cfg['name'])
        if self.cfg['signed'] != 'yes':
            if raw_input('Untrusted package! Do you want to continue? (Y/N) '
                    ).lower().startswith('y'):
                pass
            else:
                print 'Aborting install.'
                return None
        print 'Installing to {}...'.format(self.cfg['path'])
        #some installation functionality goes here
def test():
    for path in ['test.ini', 'test.json']:
        inst = Installer(path)
        inst.install()
if __name__ == '__main__':
    test()
```

File: Aufgabe2

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-II: Uebung 01 - Aufgabe 2, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz' : '97-923-163',
#                             'Linus Manser' : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Aufruf des Programms:
# Version 1: python Aufgabe2.py
# Version 2: run in PyCharm
# Version 3: python test_installer.py
#            (with: import Aufgabe2)
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'''
2 Fortgeschrittene Klassen
Wir möchten in Python einen Paket-Manager für die Linux-Distribution Y schreiben. Die Distri-
bution Y ist aus dem Zusammenschluss zweier kleinerer Distributionen entstanden. Beide dieser
Distributionen hatten bis anhin einen eigenen Paket-Manager, die beide ein anderes Format für
Paket-Konfigurationsdateien verwendeten, weshalb die Paketdatenbanken für Y Linux ein wildes
Gemisch beider Formate sind.
Glücklicherweise werden beide Formate von der Python-Standardbibliothek unterstützt: die Hälfte
der Dateien ist im JSON-Format, die andere im .ini-Format.
Um die Architektur unseres Systems eleganter zu gestalten, möchten wir gerne auf die Konfigurati-
onsdateien formatunabhängig zugreifen können. Zu diesem Zweck sollst du eine Klasse ConfigReader
schreiben, die ein solches Interface implementiert.
Deine Klasse soll beim Aufruf einen Dateinamen erhalten und das Format erkennen. Um die bei-
den unterschiedlichen Formate zu bearbeiten, sollst du zwei Subklassen definieren: JsonReader und
IniReader. Diese Subklassen sollen Dateien des jeweiligen Formats einlesen und sich nach aussen
wie Dictionaries der Schlüssel-Wert-Paare in der eingelesenen Datei verhalten (definiere dazu die
Methode __getitem__). Die Oberklasse ConfigReader soll die Methode __new__ so definieren, dass
je nach Format der übergebenen Datei die richtige Subklasse zurückgegeben wird. Zum Einlesen der
Dateien kannst du die Standardmodule json und ConfigParser verwenden.
```

zurückgegeben wird. Zum Einlesen der
Dateien kannst du die Standardmodule json und ConfigParser verwenden.
Im Übungsordner ist ein Skript zum Testen deines Moduls test_installer.py enthalten.

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
Im Übungsordner ist ein Skript zum Testen deines Moduls test_installer.py enthalten.
'''
## debug
# set to 1 for debugging purposes
DEBUG_FLAG=0
## import packages
# json parser
import json
# ini parser
import ConfigParser
# dict printer
from pprint import pprint
# os to split extension from file name
import os
## class definitions
# superclass ConfigReader
class ConfigReader(object):
    # overriding standard default constructor
    def __new__(cls, file_path):
        """
        Functionality:
            Overrides standard default constructor
            Called when new ConfigReader object is instantiated
            Reads file_path extension and instantiates subclass object
            Subclass object is either JsonReader or IniReader
        Input:
            file_path of data file of type .json or .ini
        Output:
            Object of type JasonReader or IniReader
        Exceptions:
        """
        # recognize extension
        (filename, file_extension) = os.path.splitext(file_path)
        # return JsonReader object
        if file_extension == ".json":
            # invoke subclass JsonReader and create new object
            if DEBUG_FLAG: print " info from 1.1: you are reading a ", \
                file_extension, " file"
            JR = JsonReader.__new__(cls, file_path)
            if DEBUG_FLAG: print " info from 1.1.1:", JR
            # JR.__init__(file_path) is done
            # implicitly by the next statement return JR
            return JR
        # return IniReader object
        elif file_extension == ".ini":
            # invoke subclass IniReader and create new object
            if DEBUG_FLAG: print " info from 1.2: you are reading a ", \
                file_extension, "file "
            IR = IniReader.__new__(cls, file_path)
            if DEBUG_FLAG: print " info from 1.2.1:", IR
            # IR.__init__(file_path) is done
            # implicitly by the next statement return IR
            return IR
        # print error message
        else:
            if DEBUG_FLAG: print " info from 1.3", filename, file_extension
            print " Please read in a .json or .ini file"
# subclass JsonReader
class JsonReader(ConfigReader):
    """
    Functionality:
        Overrides __new__
        __init__ creates instance variable self.dict_parsed_file
        Overloads [] with __getitem__: returns value of self.dict_parsed_file
    Instance Variables:
        self.dict_parsed_file of type dictionary
    """
    # overriding standard default constructor
    def __new__(cls, file_path):
        """
        Functionality:
            Overrides standard default constructor
            Calling instance is a newly instantiated ConfigReader object
            Reads file_path extension and instantiates object
        Input:
            file_path of data file of type .json
        Output:
            Object of type JasonReader
        Exceptions:
        """
        if DEBUG_FLAG: print " info from 2.1:", cls
        return object.__new__(JsonReader)
    def __init__(self, file_path):
        """
        Functionality:
            Calling instance is a newly instantiated ConfigReader object
            (called implicitly after JsonReader.__new__(cls, file_path))
            Reads file_path and initializes object
            Creates instance variable dictionary self.dict_parsed_file
        Input:
            file_path of data file of type .json
            json package as "global variable"
        Output:
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
            instance variable dict_parsed_file contains data structures
            of parsed .json file
        Exceptions:
        """
        # read file, parse file, assign content to instance variable dict_parsed_file
        # use imported json parser package
        '''
        {
        "name":"VSolve Statistical Dependency Parser",
        "path":"/usr/program_files/vsolve/vsolve",
        "size":169435,
        "signed":"yes",
        "author":"Torin Kvalm"
        }
        '''
        # json parser
        # (https://docs.python.org/2/library/json.html)
        # (http://stackoverflow.com/questions/2835559/parsing-values-from-a-json-file-
in-python)
        if DEBUG_FLAG: print " info from 2.2:", self
        with open(file_path) as data_file:
            if DEBUG_FLAG: print " info from 2.2.1:", data_file
            # define instance variable
            self.dict_parsed_file = json.load(data_file)
            if DEBUG_FLAG: print " info from 2.2.2:", self.dict_parsed_file
    def __getitem__(self, key):
        """
        Functionality:
            Invoked from main
            Overloads []
            Returns value of instance variable self.dict_parsed_file
        Input:
            dictionary key
        Output:
            dictionary value
        """
        # overloading of [];
        # invocation of myconfig["author"] calls myconfig.__getitem__("author")
        # and return "Torin Kvalm"
        return self.dict_parsed_file[key]
# subclass IniReader
class IniReader(ConfigReader):
    """
    Functionality:
        Overrides __new__
        __init__ creates instance variable self.dict_parsed_file
        Overloads [] with __getitem__: returns value of self.dict_parsed_file
        as_dict converts ConfigParser data structure into dictionary
    Instance Variables:
        self.dict_parsed_file of type dictionary
    """
    # overriding standard default constructor
    def __new__(cls, file_path):
        """
        Functionality:
            Overrides standard default constructor
            Calling instance is a newly instantiated ConfigReader object
            Reads file_path extension and instantiates object
        Input:
            file_path of data file of type .ini
        Output:
            Object of type IniReader
        Exceptions:
        """
        if DEBUG_FLAG: print " info from 3.1:", cls
        return object.__new__(IniReader)
    def __init__(self, file_path):
        """
        Functionality:
            Calling instance is a newly instantiated ConfigReader object
            (called implicitly after IniReader.__new__(cls, file_path))
            Reads file_path and initializes object
            Creates instance variable dictionary self.dict_parsed_file
        Input:
            file_path of data file of type .ini
            ConfigParser package as "global variable"
        Output:
            instance variable dict_parsed_file contains data structures
            of parsed .ini file
        Exceptions:
        """
        # read file, parse file, assign content to instance variable dict_parsed_file
        # use imported json parser package
        '''
        [package]
        name = Stree Treebank Viewer
        path = /usr/program_files/stree/stree.exe
        size = 322024
        signed = no
        author = James J. Callaghan
        '''
        # ini parser
```

```
        # (https://docs.python.org/2/library/configparser.html)
        # (http://stackoverflow.com/questions/8884188/how-to-read-and-write-ini-file-
with-python)
        if DEBUG_FLAG: print " info from 3.2:", self
        ini_parser = ConfigParser.ConfigParser()
        if DEBUG_FLAG: print " info from 3.2.1:", ini_parser
        ini_parser.read(file_path)
        # define instance variable
        self.dict_parsed_file = self.as_dict(ini_parser)
        if DEBUG_FLAG: print " info from 3.2.2:", self.dict_parsed_file
    def __getitem__(self, keys):
        """
        Functionality:
            Invoked from main
            Overloads []
            Returns value of instance variable self.dict_parsed_file
        Input:
            dictionary key
        Output:
            dictionary value
        """
        # overloading of [];
        # invocation of myconfig["author"] calls myconfig.__getitem__("author")
        # and return "Torin Kvalm"
        # (http://stackoverflow.com/questions/17478284/
        # python-is-there-a-way-to-implement-getitem-for-multidimension-array)
        package_key, key = keys
        value = self.dict_parsed_file[package_key][key]
        return value
    ## method to write into dictionary self.dict_parsed_file
    # (http://stackoverflow.com/questions/3220670/
    # read-all-the-contents-in-ini-file-into-dictionary-with-python)
    def as_dict(self, ini_parser):
        """
        Functionality:
            Invoked from __init__
            Converts ConfigParser data structure into dictionary
            Code from stackoverflow.com (not used)
        Input:
            ConfigParser (IniReader) instance
        Output:
            dictionary
        """
        d = dict(ini_parser._sections)
        for k in d:
            d[k] = dict(ini_parser._defaults, **d[k])
            d[k].pop('__name__', None)
        return d
## main procedure
#__name__ == filename when file is imported as module or package
#__name__ == __main__ when started from command line (or with run)
if __name__ == "__main__":
    print "\n######################################################"
    print 'Hi, From Program 2 : '
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    ## data files
    # json file
    file_json = "/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
                "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test.json"
    # ini file
    file_ini = "/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
               "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test.ini"
    ## read file by calling superclass ConfigReader
    # ConfigReader reads the suffix and calls either
    # JsonReader or IniReader class
    myconfig1 = ConfigReader(file_json)
    if DEBUG_FLAG: print " info from 4.1:", myconfig1
    if DEBUG_FLAG: print "info~~~~~~~~~~~"
    myconfig2 = ConfigReader(file_ini)
    if DEBUG_FLAG: print " info from 4.2:", myconfig2
    ## read data from dictionaries myconfig1, myconfig2
    print "1~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    the_author1 = myconfig1["author"]
    print " author: ", the_author1
    the_author2 = myconfig2["package", "author"]
    print " author: ", the_author2
    ## print whole dictionary myconfig
    # standard way
    print "2~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    for key, value in myconfig1.dict_parsed_file.items():
        print key, ":", value
    print "3~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    for key, value in myconfig2.dict_parsed_file.items():
        for key2, value2 in myconfig2.dict_parsed_file[key].items():
            print key2, ":", value2
    # with pprint package
    print "4~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    pprint(myconfig1.dict_parsed_file)
    pprint(myconfig2.dict_parsed_file)
    print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "Bye Bye from Programm 2! :-)"
```

```
    print "#######################################################"
```

File: Test_installer

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# PCL-II: Uebung 01 - Aufgabe 2, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz' : '97-923-163',
#                             'Linus Manser' : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# Aufruf des Programms:
# Version 1: run in PyCharm
# Version 2: python test_installer.py
#            (with: import Aufgabe2.py)
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
## debug
# set to 1 for debugging purposes
DEBUG_FLAG=0
## import packages
# import <your solution to task 2 here> as a2
import Aufgabe2
## class definitions
#
class Installer(object):
    """
    Mockup package installer class.
    """
    def __init__(self, file_path):
        """
        Functionality:
            Calling instance is the function test()
            Reads file_path and initializes object
            Creates instance variable self.cfg of type ConfigReader
        Input:
            file_path of data file of type .ini or .json
            Aufgabe2 module/package as "global variable"
        Output:
            instance variable self.CR contains a ConfigReader object
            which contains the data structures of parsed .ini or .json file
        Exceptions:
        """
        # instance variable CR
        self.CR = Aufgabe2.ConfigReader(file_path)
    def install(self):
        """
        Functionality:
            Calling instance is the function test()
            Checks the field signed of .json or .ini file data structure
            Installs signed package automatically, if unsigned asks user
            (not implemented)
        Input:
            instance variable self.CR
        Output:
            Console output, about mock installation (simulation)
        Exceptions:
            the instance variable of the json parser is a 1-dim dictionary
            he instance variable of the ini parser is a 2-dim dictionary
            used try and except instead of dimension check
        """
        try:
            #json
            cr_name=self.CR['name']
            cr_signed=self.CR['signed']
            cr_path=self.CR['path']
        except:
            #ini
            cr_name=self.CR["package",'name']
            cr_signed=self.CR["package",'signed']
            cr_path=self.CR["package",'path']
        print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
        print 'Processing package: {}'.format(cr_name)
        ## Install signed package automatically, if unsigned ask user
        # check field signed of .ini or .json file
        # signed is a field in the data structure
        if cr_signed != 'yes':
            if raw_input(
                    'Untrusted package! Do you want to continue? (Y/N) '
                    ).lower().startswith('y'):
                pass
            else:
                print 'Aborting install.'
                return None
        print 'Installing to {}...'.format(cr_path)
        print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
        #some installation functionality goes here
```

```
            # (not implemented)
            #
            #
## test case for Aufgabe2.py
def test(file_json, file_ini):
    """
    Functionality:
        Creates Installer instance
        Calls mock install method
    Input:
        Configuration files of type .json and .ini
    Output:
        Indirectly:
        Console output, about mock installation (simulation)
    Exceptions:
    """
    ## instantiate defined Installer object for each data file
    for file_path in [file_json, file_ini]:
        if DEBUG_FLAG: print " info from I.2.1:", file_path
        #new instance of Installer
        #has instance variable ConfigReader
        my_installer = Installer(file_path)
        #call mock package installern (simulation)
        my_installer.install()
## main procedure
if __name__ == '__main__':
    print "\n######################################################"
    print 'Hi, From Test_Installer : '
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    ## data files
    # json file
    file_json = "/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
                "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test.json"
    # ini file
    file_ini = "/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
               "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test.ini"
    # invoke defined test installer
    if DEBUG_FLAG: print " info from I.1.1:", __name__
    test(file_json, file_ini)
    print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "Bye Bye from Test_Installer! :-)"
    print "######################################################"
```

File: test.ini

```
[package]
name = Stree Treebank Viewer
path = /usr/program_files/stree/stree.exe
size = 322024
signed = no
author = James J. Callaghan
```

File: test.json

```
{
   "name":"VSolve Statistical Dependency Parser",
   "path":"/usr/program_files/vsolve/vsolve",
   "size":169435,
   "signed":"yes",
   "author":"Torin Kvalm"
}
```

File: test_ascii.txt

```
Das ist ein Beispieltext.
ASCII unterstuetzt nicht mal Umlaute.
```

File: test_latin1.txt

```
Das ist ein Beispieltext
�orsteinn heit ma�r. Hann var Egilsson.
TEST �����.
Je suis d�sol�.
������������
�#��[]
```

File: test_utf8.txt

```
Das ist ein Beispieltext
þorsteinn heit maðr. Hann var Egilsson.
™™™ TEST ðéóæÆ.
```

File: aufgabe3_muster

```
#!/usr/bin/env python2
#-*- coding: utf-8 -*-
import os
import argparse
import codecs
from os import path
import regex
import sys
```

U1.3
3 Input, Output, Encoding
Das UNIX-Programm sed ist ein beliebtes Tool, um Text in
einer Pipeline zu editieren. Die am häu-
figsten genutzte Funktion dieses Programms ist der Befehl
s/pattern/replace/g, um Ersetzungen
mit regulären Ausdrücken durchzuführen. Du sollt nun ein
Python-Programm schreiben, dass einen
solchen Filter implementiert.
Dein Programm soll von der Kommandozeile einen Befehl in

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```python
class SedPattern(object):
    """
    Class to handle sed-like s///g patterns.
    """
    def __init__(self, pattern):
        """
        Initializer for the SedPattern class.
        Args:
            pattern: A unicode string in the form s/re/text/g
        """
        self.pattern_string = pattern
        exprs = self._parse_pattern(pattern)
        self.search = exprs[0]
        self.replacement = exprs[1]
    def _parse_pattern(self, pattern):
        """
        Method to process a text pattern.
        Args:
            pattern: A unicode string in the form s/re/text/g
        Returns:
            A tuple of a regex object and the replacement unicode
            string.
        """
        expr_parts = pattern.split(u'/')
        if not (expr_parts[0] == u's' and expr_parts[3] == u'g'):
            raise ValueError(u'Malformed command: {}, missing s...g'.format(
                self.pattern_string))
        search = regex.compile(expr_parts[1])
        replacement = expr_parts[2]
        return search, replacement

    def sub(self, text):
        """
        Applies pattern to text.
        Args:
            text: A unicode object to replace text in.
        Returns:
            The edited text.
        """
        nl = regex.sub(self.search, self.replacement, text)
        return nl
class SubstituteStream(object):
    """
    Class to handle the program input/output in an encoding-neutral
    manner. The class assumes that the input and output streams are
    both using the same text encoding.
    """
    def __init__(self, encstr, fpath=None, opath=None):
        """
        Initializer for the SubstituteStream class. When no filepaths
        are given as arguments, sets input and output to stdin and
        stdout, respectively.
        Args:
            encstr: Name of the stream's text encoding.
            fpath: Name of the file to read from.
            opath: Name of the file to write to.
        """
        self.standard_in = True
        self.standard_out = True
        self.encoding = encstr
        if fpath is None:
            self.infile = codecs.getreader(self.encoding)(sys.stdin)
        else:
            self.infile = codecs.open(fpath, 'r', self.encoding)
            self.standard_in = False

        if opath is None:
            self.outfile = codecs.getwriter(self.encoding)(sys.stdout)
        else:
            try:
                self.outfile = codecs.open(opath, 'r+', self.encoding)
            except IOError:
                self.outfile = codecs.open(opath, 'w', self.encoding)
            self.standard_out = False
        self.text = self.infile.read()
    def __enter__(self):
        return self
    def __exit__(self, exc_type, exc_value, traceback):
        self.close()
        return False
    def apply(self, pattern):
        """
        Method to perform text substitution on the stream text.
        Pattern can be any object with a sub method.
        Args:
            pattern: A SedPattern, regex or other object
                     with a sub method.
        """
        self.text = pattern.sub(self.text)
    def write(self):
        self.outfile.write(self.text)
    def close(self):
```

der Form s/pattern/replacement/g
einlesen. Den Text soll es standardmässig von der Standardeingabe lesen und den bearbeiteten Text auf der Standardausgabe wieder ausgeben.
Das Programm sollte folgende Kommandozeilenoptionen unterstützen:
a) -e|--encoding: Das Encoding der eingelesenen Datei. Dein Programm sollte mindestens die Encodings ascii, latin-1 und utf-8 unterstützen.
b) -f|--file: Datei, aus der der Rohtext gelesen werden soll, wenn er nicht von der Standard-eingabe kommt.
c) -o|--out: Datei, in die der bearbeitete Text geschrieben werden soll, wenn er nicht auf der Kommandozeile ausgegeben werden soll.
Um Unicode-kompatible reguläre Ausdrücke zu unterstützen, solltest du ausserdem statt des Stan-dardmoduls re die Bibliothek regex verwenden. Du kannst dieses Paket von der Kommandozeile mithilfe von $ sudo pip install regex installieren. (Sollten Installationsprobleme auftreten, wende dich zuerst an die Tutoren; wenn sie nicht gelöst werden können, ist es in Ordnung, stattdessen re zu verwenden.)

```
        if not self.standard_in:
            self.infile.close()
        if not self.standard_out:
            self.outfile.close()
def _get_args():
    parser = argparse.ArgumentParser(description=u'Substitute '
        'text in a file.')
    parser.add_argument(u'pattern')
    parser.add_argument(u'-e', u'--encoding', default=u'ascii',
            help=u'Specify encoding of the input and output.')
    parser.add_argument(u'-f', u'--file', help=u'Read text from a file.')
    parser.add_argument(u'-o', u'--out', help=u'Write output to a file.')
    args = parser.parse_args()
    return args
def main():
    args = _get_args()
    try:
        decoded_pat = codecs.decode(args.pattern, args.encoding)
    except UnicodeDecodeError as ue:
        print >> sys.stderr, codecs.encode(u'Error: pattern given is not in '
        'encoding given\n{}'.format(unicode(ue)), args.encoding)
        if args.encoding == u'ascii':
            print >> sys.stderr, codecs.encode(u'You can try specifying '
            'the encoding with -e ENCODING', args.encoding)
        sys.exit(1)
    try:
        sub = SedPattern(decoded_pat)
    except ValueError as v:
        print >> sys.stderr, codecs.encode(u'Invalid pattern: '
                '{}'.format(unicode(v)), args.encoding)
        sys.exit(1)
    except regex.error as err:
        print >> sys.stderr, codecs.encode(u'Invalid pattern: '
                '{}'.format(unicode(err)), args.encoding)
        sys.exit(1)
    try:
        with SubstituteStream(args.encoding, args.file, args.out) as repltext:
            repltext.apply(sub)
            repltext.write()
    except IOError as err:
        print >> sys.stderr, codecs.encode(u'I/O Error: {}'.format(
            unicode(err)), args.encoding)
        sys.exit(1)
if __name__ == '__main__':
    main()
```

File: Aufgabe3

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# PCL-II: Uebung 01 - Aufgabe 3, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz' : '97-923-163',
#                             'Linus Manser' : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# Aufruf des Programms:
#
# #Test1: kein Text
# python Aufgabe3.py s/Regex/Python/g
# (Enter, dann 3 Mal Ctrl-D)
#
# #Test2: kein Regex
# python Aufgabe3.py
# Regex macht spass (Enter, dann 3 Mal Ctrl-D)
# couldnt replace any text!
#
# #Test3: echo
# echo "Regex macht spass" | python Aufgabe3.py s/Regex/Python/g
# Python macht spass
#
# #Test4: codecs.getreader ()(sys.stdin)
# python Aufgabe3.py s/Regex/Python/g
# Regex macht spass (Enter, dann 3 Mal Ctrl-D)
# Python macht spass
#
# #Test5: output file, option -o
# python Aufgabe3.py s/Regex/Python/g -o out.txt
# Regex macht spass (Enter, dann 3 Mal Ctrl-D)
# wrote text to file 'out.txt' ...
# Python macht spass
#
# #Test6: encoding, option -e
# python Aufgabe3.py s/Regex/Python/g -o out.txt -e latin1
# Regex macht spöss (Enter, dann 3 Mal Ctrl-D)
# wrote text to file 'out.txt' ...
# Python macht sp�ss
# $file -i out.txt (Enter)
# charset=iso-8859-1
#
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
# #Test7: input file, option -f
# python Aufgabe3.py s/Regex/Python/g -o out.txt -e latin1 -f test_latin1_.txt
# Das ist ein Beispieltext
# þorsteinn heit maðr. Hann var Egilsson.
# TEST ðéóæÆ.
# Je suis désolé.
# üÜäÄöÖÉéàÀèÈ
# ÿ#±Ç[]
# $file -i out.txt (Enter)
# charset=iso-8859-1
#
# Test8: None Ascii characters in find/replace string
# Aufgabe3.py s/Regöx/Pythön/g
# Regöx macht spass
# ASCII doesn't support characters in your input
#
# Test9: None Ascii characters in find/replace string, latin1 encoding
# Aufgabe3.py s/Regöx/Pythön/g -e latin1
# Regöx macht spass (Enter, dann 3 Mal Ctrl-D)
# Pythön macht spass
#
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# ## Python Code um Encoding einer Textdatei zu ändern.
# import codecs
# in_filename = "/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
#                  "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test_latin1.txt"
# out_filename="/media/benzro/OS/Users/benzro/Desktop/Studium Uni/" \
#                  "2)ZweitesSemester/27)PCL-2/Uebungen/Uebung01/test_latin1_.txt"
# infile = codecs.open(in_filename, 'r', encoding='utf-8')
# outfile = codecs.open(out_filename, 'w', encoding='latin1')
# for line in infile:
#      outfile.write(line)
# infile.close()
# outfile.close()
#
# file -i test_latin1_.txt
# charset=iso-8859-1
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
'''
##Aufgabe 3
#Das UNIX-Programm sed ist ein beliebtes Tool, um Text in einer Pipeline zu editieren.
Die am häufigsten
#genutzte Funktion dieses Programms ist der Befehl s/pattern/replace/g, um Ersetzungen
mit regulären Ausdrücken
#durchzuführen. Du sollt nun ein Python-Programm schreiben, dass einen solchen Filter
implementiert.
#Dein Programm soll von der Kommandozeile einen Befehl in der Form
s/pattern/replacement/g einlesen.
#Den Text soll es standardmässig von der Standardeingabe lesen und den bearbeiteten Text
auf der Standardausgabe
#wieder ausgeben.
#Das Programm sollte folgende Kommandozeilenoptionen unterstützen:
#a) -e|--encoding: Das Encoding der eingelesenen Datei. Dein Programm sollte mindestens
die
#Encodings ascii, latin-1 und utf-8 unterstützen.
#b) -f|--file: Datei, aus der der Rohtext gelesen werden soll, wenn er nicht von der
Standard-
#eingabe kommt.
#c) -o|--out: Datei, in die der bearbeitete Text geschrieben werden soll, wenn er nicht
auf der Kommandozeile ausgegeben werden soll.
'''
## debug
# set to 1 for debugging purposes
DEBUG_FLAG=0
## import packages
import sys
import regex as re
import codecs
## function definitions
# extract find/replace pattern
def get_sed_query(list):
    """
    Functionality:
        extract find/replace pattern
    Input:  list
            searches for the sed-query in the form of s/pattern/replacement/g
    Exceptions: if nothing is given -> no query defined
                if first element in the input_list isnt the query -> incorrect query
defined
    Output: extracts the pattern and replacement and returns it as a tuple (pattern,
replacement)
    """
    if DEBUG_FLAG: print " info from 1.0: get_sed_query()"
    #find/replace pattern to search for
    pattern = "s/(.*?)/(.*?)/g"
    if DEBUG_FLAG: print " info from 1.1:", pattern
    #search for find/replace argument parameter in input parameter list
    # extraction of pattern and replacement from the query
    for i in list:
        try:
            #check each list element if it
            # matches find/replace input argument
```

```
            match = re.search(pattern,i)
            pattern = match.group(1)
            replacement = match.group(2)
            #output: Tuple
            if DEBUG_FLAG: print " info from 1.2:", pattern, replacement
            return (pattern, replacement)

       # Ignores AttributeError -> if the first element in the command line isnt the
query, the search continues
        except AttributeError:
            None
# extract encoding string
def get_encoding(list):
    """
    Functionality:
        extract encoding string
    Input: takes in a list and looks for "-e" or "--encoding" to set encoding
    Output: returns the given encoding (first part after the command) as string ("utf-
8", "latin-1",...)
    """
    if DEBUG_FLAG: print " info from 2.0: get_encoding()"
    cmd = ["-e", "--encoding"]
    # read out required encoding
    # if optional command exists, do the following.
    for i in cmd:
        if i in list:
            # Take the next element in the input_list after -e or --encoding
            encoding = list[list.index(i)+1]
            if DEBUG_FLAG: print " info from 2.1:", encoding
            return encoding
    # default encoding is utf-8 (if no encoding defined)
    encoding = "ASCII"
    if DEBUG_FLAG: print " info from 2.2:", encoding
    return encoding

# extract text from file or sys.stdin
def get_text(list):
    """
    Functionality:
        extract text from file or sys.stdin
    Input: list
            checks whether a file or just the given text in the command-line is
processed
            if a file is declared, it opens it with the given encoding (default=utf-8)
    Output: returns the text inside of the file or the text behind the option-commands
    """
    if DEBUG_FLAG: print " info from 3.0: get_text()"
    text = []
    cmd = ["-f","--file"]
    ## Read input file
    # if optional command exists, open the given document (first element after -f/--
file)
    for i in cmd:
        if i in list:
            #input_filename directly after option -f
            input_filename = list[list.index(i)+1]
            if DEBUG_FLAG: print " info from 3.1:", input_filename
            #open input file for reading
            encoding=get_encoding(list)
            with codecs.open(input_filename, "r", encoding) as input_file:
                if DEBUG_FLAG: print " info from 3.2:", input_file, encoding
                try:
                    #list, one line one list element
                    for line in input_file:
                        text.append(line)
                    if DEBUG_FLAG: print " info from 3.3:", text
                    #convert to string
                    # join the list into one string -> whole document in one string
                    text = "".join(text)
                    if DEBUG_FLAG: print " info from 3.4:", text
                    return text
                except UnicodeDecodeError:
                    print "the given encoding doesn't support characters in this file.
please change the encoding..."
                    exit();
    ## Wait for user input and read text from console
    # if no -f or --file flag given, get the standard-in as input
    input_text = codecs.getreader(get_encoding(list))(sys.stdin)
    try:
        #list, one line one list element
        for i in input_text:
            text.append(i)
        if DEBUG_FLAG: print " info from 3.5:", text
        #convert to string
        text = "".join(text)
        if DEBUG_FLAG: print " info from 3.6:", text
        return text
    except UnicodeDecodeError:
        print "%s doesn't support characters in your input" % (get_encoding(list))
        exit();
# apply find/replace pattern on text
def apply_sed_query(list, text):
```

```python
    """
    Functionality:
        apply find/replace pattern on text
    Input: list, unprocessed text
    Output: processed text (replaced text)
    """
    if DEBUG_FLAG: print " info from 4.0: apply_sed_query()"
    try:
        #get find/replace command line argument
        pattern, replacement = get_sed_query(list)
        if DEBUG_FLAG: print " info from 4.1:", pattern, replacement
        #change to unicode
        pattern = pattern.decode(get_encoding(list))
        replacement = replacement.decode(get_encoding(list))
        if DEBUG_FLAG: print " info from 4.2:", pattern, replacement
        #find/replace
        out_text = re.sub(pattern,replacement,text)
        #print "replaced '%s' with '%s'\n~~~~~~~~~~~~~~~~~~~~~~~~" % (pattern,
replacement)
        if DEBUG_FLAG: print " info from 4.3:", out_text
        return out_text
    except TypeError:
        print "couldnt replace any text!"
    except UnicodeDecodeError:
        print "%s doesn't support characters in your input" % (get_encoding(list))
# write adapted text into file or console
def write_out(list, out_text):
    """
    Functionality:
        write adapted text into file or console
    Input: list, processed text
    Output: if a file is declared, write processed text to file
            else it shows the text in the bash
    """
    if DEBUG_FLAG: print " info from 5.0: write_out()"
    cmd = ["-o","--out"]
    # if optional command (cmd) exists, do the following:
    for i in cmd:
        if i in list:
            #output_filename directly after option -o
            output_filename = list[list.index(i)+1]
            if DEBUG_FLAG: print " info from 5.1:", output_filename
            #write text into output_file with requested encoding
            encoding_ = get_encoding(list)
            output_file=codecs.open(output_filename, "w", encoding=encoding_)
            output_file.write(out_text)
            if DEBUG_FLAG: print " info from 5.2:", output_file, encoding_, out_text
            print "wrote text to file '%s' ..." % output_filename
            output_file.close
            exit();
    # default behaviour: print output text directly into the bash
    if DEBUG_FLAG: print " info from 5.3:", out_text
    print out_text
## main procedure
if __name__ == "__main__":
    #print "~~~~~~~~~~~~~~~~~~~Start: Programm Aufgabe 3~~~~~~~~~~~~~~~~~~~~~~~~"
    if DEBUG_FLAG: print " info from 0.0: main()"
    #get argument parameter list
    command_list = sys.argv[1:]
    if DEBUG_FLAG: print " info from 0.1: ", command_list
    #get (sys.stdin) standard input text of input file or user input
    in_text=get_text(command_list)
    if DEBUG_FLAG: print " info from 0.2: ", in_text
    #apply find/replace on input text
    out_text = apply_sed_query(command_list, in_text)
    if DEBUG_FLAG: print " info from 0.3: ", out_text
    #write out adapted text after find/replace
    write_out(command_list, out_text)
    #print "~~~~~~~~~~~~~~~~~~~Ende: Programm Aufgabe 3~~~~~~~~~~~~~~~~~~~~~~~~"
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<books>
   <book id="1" movie="yes">
      <title>Lord of the Rings</title>
      <author nobel="no" nationality="British">J.R.R. Tolkien</author>
         <author-birthdate>January 3, 1892</author-birthdate>
         <author-deathday>September 2, 1973</author-deathday>
      <film-director oscar="3" nationality="New Zealand">Peter Jackson</film-director>
         <director-birthdate>October 31, 1961</director-birthdate>
         <director-deathday/>
   </book>
   <book id="2" movie="no">
      <title>The Martian Chronicles</title>
      <author nobel="no" nationality="American">Ray Bradbury</author>
         <author-birthdate>August 22, 1920</author-birthdate>
         <author-deathday>June 5, 2012</author-deathday>
      <film-director/>
   </book>
   <book id="3" movie="yes">
```

U2.1
1 Wohlgeformtheit
Betrachte für diese Aufgabe das beiliegende XML Dokument books.xml. Zähle alles auf, was dieses Dokument bei einem Test für Wohlgeformtheit durchfallen lässt und notiere jeweils die dazugehörige XML Syntax Regel, welche gebrochen wird. Zusätzlich sollst du das XML Dokument duplizieren und korrigieren, so dass es den Test für Wohlgeformtheit besteht.
Abzugeben: Ein Dokument mit deinen Antworten und dein korrigiertes XML-Dokument mit dem Namen books_corrected.xml.

Lösung
Bemerkung:
Die Wohlgeformtheit beschreibt ausschliesslich die Einhaltung

```xml
      <title>The Da Vinci Code</title>
      <author>Dan Brown</author>
      <film-director>Ron Howard</film-director>
   </book>
   <book id="4" movie="yes">
   <title>For Whom the Bell Tolls</title>
      <author nobel="yes" nationality="American">Ernest Hemingway</author>
         <author-birthdate>July 21, 1899</author-birthdate>
         <author-deathday>July 2, 1961</author-deathday>
      <film-director oscars="0" nationality="American">Sam Wood</film-director>
         <director-birthdate>July 10, 1884</director-birthdate>
         <director-deathday>September 22, 1949</director-deathday>
   </book>
   <book id="5" movie="no">
      <title>Rayuela</title>
      <author>Julio Cortázar</author>
      <film-director/>
   </book>
   <book id="6" movie="no">
      <title>Pedro Páramo</title>
      <author>Juan Rulfo</author>
      <film-director/>
   </book>
   <book id="7" movie="no">
      <title>Der Schwarm</title>
      <author nobel="no" nationality="German">Frank Schätzing</author>
         <author-birthdate>May 28, 1957</author-birthdate>
         <author-deathday/>
      <film-director/>
   </book>
   <book id="8" movie="no">
      <title>Die Physiker</title>
      <author>Friedrich Dürrenmatt</author>
      <film-director/>
   </book>
   <book id="9" movie="yes">
      <title>L'Étranger</title>
      <author>Albert Camus</author>
      <film-director>Luchino Visconti</film-director>
   </book>
   <book id="10" movie="no">
      <title>Cien años de soledad</title>
      <author>Gabriel García Márquez</author>
      <film-director/>
   </book>
</books>
```

der "strikten" Syntaxregeln. Sie ist
unerlässlich bei der Entscheidung über die Validität eines XML-Dokumentes. Wohlgeformte
XML Dateien, die den Regeln des dazugehörigen XML-Schemas folgen, sind valide. Das ist der
Grund, weshalb XML eine "Meta-markup" Sprache ist, denn erst dieses XML-Schema verleiht
den Tags die eigentliche Bedeutung.
Man kann die Wohlgeformtheit und Validität auf der Kommandozeile überprüfen.
Wohlgeformtheit:
xmllint books_corrected.xml --noout #Bei wohlgeformtheit passiert nichts.
+ mit Schema(falls vorhanden) -> Validität:
xmllint --schema schema.xsd books_corrected.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PLANTS>
   <PLANT zone="4">
      <COMMON>Bloodroot</COMMON>
      <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>2.44</PRICE>
      <AVAILABILITY>031599</AVAILABILITY>
   </PLANT>
   <PLANT zone="3">
      <COMMON>Columbine</COMMON>
      <BOTANICAL>Aquilegia canadensis</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>9.37</PRICE>
      <AVAILABILITY>030699</AVAILABILITY>
   </PLANT>
   <PLANT zone="4">
      <COMMON>Marsh Marigold</COMMON>
      <BOTANICAL>Caltha palustris</BOTANICAL>
      <LIGHT>Mostly Sunny</LIGHT>
      <PRICE>6.81</PRICE>
      <AVAILABILITY>051799</AVAILABILITY>
   </PLANT>
   <PLANT zone="4">
      <COMMON>Cowslip</COMMON>
      <BOTANICAL>Caltha palustris</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>9.90</PRICE>
      <AVAILABILITY>030699</AVAILABILITY>
   </PLANT>
   <PLANT zone="3">
      <COMMON>Dutchman's-Breeches</COMMON>
      <BOTANICAL>Dicentra cucullaria</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>6.44</PRICE>
      <AVAILABILITY>012099</AVAILABILITY>
   </PLANT>
   <PLANT zone="3">
      <COMMON>Ginger, Wild</COMMON>
      <BOTANICAL>Asarum canadense</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>9.03</PRICE>
      <AVAILABILITY>041899</AVAILABILITY>
```

U2.2
2 XPath
Folgend sind Anfragen auf das beiliegende XML Dokument plants.xml aufgelistet. Wandle diese
Anfragen zu XPath-Ausdrücken um.
a) Der botanische Name der zweiten Pflanze
b) Anzahl Pflanzen mit Zone ‘Annual’
c) Die Zone der Pflanzen mit ‘Shade’ als Lichtbedingung
d) Alle Namen der Pflanzen mit Zone ‘2’
e) Pflanzen, welche einen Preis grösser als 5 haben
Abzugeben: Das mit den XPath-Ausdrücken ergänzte Dokument aus Aufgabe 1.

Lösung
Untenstehend sind unsere Anfragen an das XML Dokument plants.xml
a) Der botanische Name der zweiten Pflanze (Nummerierung beginnt bei [1] nicht Null)
/PLANTS/PLANT[2]/BOTANICAL/text()
b) Anzahl Pflanzen mit Zone ‘Annual’
count(/PLANTS/PLANT[@zone = 'Annual'])
c) Die Zone der Pflanzen mit ‘Shade’ als Lichtbedingung
/PLANTS/PLANT[./LIGHT = 'Shade']/@zone
d) Alle Namen der Pflanzen mit Zone ‘2’
/PLANTS/PLANT[@zone = '2']/BOTANICAL/text()
e) Pflanzen, welche einen Preis groesser als 5 haben
/PLANTS/PLANT[./PRICE > '5']

Muster
a) /PLANTS/PLANT[2]/BOTANICAL/text()
b) count(/PLANTS/PLANT[@zone=’Annual’])
c) /PLANTS/PLANT[LIGHT=’Shade’]/@zone
d) /PLANTS/PLANT[@zone=’2’]/COMMON/text()
e) /PLANTS/PLANT[PRICE>5]

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
</PLANT>
<PLANT zone="4">
   <COMMON>Hepatica</COMMON>
   <BOTANICAL>Hepatica americana</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>4.45</PRICE>
   <AVAILABILITY>012699</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Liverleaf</COMMON>
   <BOTANICAL>Hepatica americana</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>3.99</PRICE>
   <AVAILABILITY>010299</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Jack-In-The-Pulpit</COMMON>
   <BOTANICAL>Arisaema triphyllum</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>3.23</PRICE>
   <AVAILABILITY>020199</AVAILABILITY>
</PLANT>
<PLANT zone="3">
   <COMMON>Mayapple</COMMON>
   <BOTANICAL>Podophyllum peltatum</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>2.98</PRICE>
   <AVAILABILITY>060599</AVAILABILITY>
</PLANT>
<PLANT zone="3">
   <COMMON>Phlox, Woodland</COMMON>
   <BOTANICAL>Phlox divaricata</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>2.80</PRICE>
   <AVAILABILITY>012299</AVAILABILITY>
</PLANT>
<PLANT zone="3">
   <COMMON>Phlox, Blue</COMMON>
   <BOTANICAL>Phlox divaricata</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>5.59</PRICE>
   <AVAILABILITY>021699</AVAILABILITY>
</PLANT>
<PLANT zone="7">
   <COMMON>Spring-Beauty</COMMON>
   <BOTANICAL>Claytonia Virginica</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>6.59</PRICE>
   <AVAILABILITY>020199</AVAILABILITY>
</PLANT>
<PLANT zone="5">
   <COMMON>Trillium</COMMON>
   <BOTANICAL>Trillium grandiflorum</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>3.90</PRICE>
   <AVAILABILITY>042999</AVAILABILITY>
</PLANT>
<PLANT zone="5">
   <COMMON>Wake Robin</COMMON>
   <BOTANICAL>Trillium grandiflorum</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>3.20</PRICE>
   <AVAILABILITY>022199</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Violet, Dog-Tooth</COMMON>
   <BOTANICAL>Erythronium americanum</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>9.04</PRICE>
   <AVAILABILITY>020199</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Trout Lily</COMMON>
   <BOTANICAL>Erythronium americanum</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>6.94</PRICE>
   <AVAILABILITY>032499</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Adder's-Tongue</COMMON>
   <BOTANICAL>Erythronium americanum</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>9.58</PRICE>
   <AVAILABILITY>041399</AVAILABILITY>
</PLANT>
<PLANT zone="6">
   <COMMON>Anemone</COMMON>
   <BOTANICAL>Anemone blanda</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>8.86</PRICE>
   <AVAILABILITY>122698</AVAILABILITY>
</PLANT>
```

```
<PLANT zone="6">
   <COMMON>Grecian Windflower</COMMON>
   <BOTANICAL>Anemone blanda</BOTANICAL>
   <LIGHT>Mostly Shady</LIGHT>
   <PRICE>9.16</PRICE>
   <AVAILABILITY>071099</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Bee Balm</COMMON>
   <BOTANICAL>Monarda didyma</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>4.59</PRICE>
   <AVAILABILITY>050399</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Bergamot</COMMON>
   <BOTANICAL>Monarda didyma</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>7.16</PRICE>
   <AVAILABILITY>042799</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
   <COMMON>Black-Eyed Susan</COMMON>
   <BOTANICAL>Rudbeckia hirta</BOTANICAL>
   <LIGHT>Sunny</LIGHT>
   <PRICE>9.80</PRICE>
   <AVAILABILITY>061899</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Buttercup</COMMON>
   <BOTANICAL>Ranunculus</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>2.57</PRICE>
   <AVAILABILITY>061099</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Crowfoot</COMMON>
   <BOTANICAL>Ranunculus</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>9.34</PRICE>
   <AVAILABILITY>040399</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
   <COMMON>Butterfly Weed</COMMON>
   <BOTANICAL>Asclepias tuberosa</BOTANICAL>
   <LIGHT>Sunny</LIGHT>
   <PRICE>2.78</PRICE>
   <AVAILABILITY>063099</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
   <COMMON>Cinquefoil</COMMON>
   <BOTANICAL>Potentilla</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>7.06</PRICE>
   <AVAILABILITY>052599</AVAILABILITY>
</PLANT>
<PLANT zone="5">
   <COMMON>Primrose</COMMON>
   <BOTANICAL>Oenothera</BOTANICAL>
   <LIGHT>Sunny</LIGHT>
   <PRICE>6.56</PRICE>
   <AVAILABILITY>013099</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Gentian</COMMON>
   <BOTANICAL>Gentiana</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>7.81</PRICE>
   <AVAILABILITY>051899</AVAILABILITY>
</PLANT>
<PLANT zone="4">
   <COMMON>Blue Gentian</COMMON>
   <BOTANICAL>Gentiana</BOTANICAL>
   <LIGHT>Sun or Shade</LIGHT>
   <PRICE>8.56</PRICE>
   <AVAILABILITY>050299</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
   <COMMON>Jacob's Ladder</COMMON>
   <BOTANICAL>Polemonium caeruleum</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>9.26</PRICE>
   <AVAILABILITY>022199</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
   <COMMON>Greek Valerian</COMMON>
   <BOTANICAL>Polemonium caeruleum</BOTANICAL>
   <LIGHT>Shade</LIGHT>
   <PRICE>4.36</PRICE>
   <AVAILABILITY>071499</AVAILABILITY>
</PLANT>
<PLANT zone="Annual">
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
      <COMMON>California Poppy</COMMON>
      <BOTANICAL>Eschscholzia californica</BOTANICAL>
      <LIGHT>Sun</LIGHT>
      <PRICE>7.89</PRICE>
      <AVAILABILITY>032799</AVAILABILITY>
   </PLANT>
   <PLANT zone="Annual">
      <COMMON>Shooting Star</COMMON>
      <BOTANICAL>Dodecatheon</BOTANICAL>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>8.60</PRICE>
      <AVAILABILITY>051399</AVAILABILITY>
   </PLANT>
   <PLANT zone="Annual">
      <COMMON>Snakeroot</COMMON>
      <BOTANICAL>Cimicifuga</BOTANICAL>
      <LIGHT>Shade</LIGHT>
      <PRICE>5.63</PRICE>
      <AVAILABILITY>071199</AVAILABILITY>
   </PLANT>
   <PLANT zone="2">
      <COMMON>Cardinal Flower</COMMON>
      <BOTANICAL>Lobelia cardinalis</BOTANICAL>
      <LIGHT>Shade</LIGHT>
      <PRICE>3.02</PRICE>
      <AVAILABILITY>022299</AVAILABILITY>
   </PLANT>
</PLANTS>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# PCL-I: Uebung 02 - Aufgabe 2, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz'       : '97-923-163',
#                             'Linus Manser'       : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# Aufruf des Programms:
# Version 1: python ex02_testscript.py
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
## necessary imports to parse an xml file
from lxml import etree
from StringIO import StringIO
import codecs
import sys
## parse the plants.xml file and save it as "tree"
tree = etree.parse("plants.xml")
# Frage 1
print "\n1) Der botanische Name der zweiten Pflanze?"
xpath1 = "/PLANTS/PLANT[2]/BOTANICAL/text()"
q1 = tree.xpath(xpath1)
print ">>> %s\n" % xpath1
print q1[0]
# Frage 2
print "\n2) Anzahl Pflanzen mit Zone ‘Annual’?"
xpath2 = "count(/PLANTS/PLANT[@zone = 'Annual'])"
q2 = tree.xpath(xpath2)
print ">>> %s\n" % xpath2
print q2
# Frage 3
print "\n3) Die Zone der Pflanzen mit ‘Shade’ als Lichtbedingung?"
xpath3 = "/PLANTS/PLANT[./LIGHT = 'Shade']/@zone"
q3 = tree.xpath(xpath3)
ans = [i for i in set(q3)]
print ">>> %s\n" % xpath3
for i in ans:
   print "-",i
# Frage 4
print "\n4) Alle Namen der Pflanzen mit Zone ‘2’?"
xpath4 = "/PLANTS/PLANT[@zone = '2']/BOTANICAL/text()"
q4 = tree.xpath(xpath4)
print ">>> %s\n" % xpath4
for i in q4:
    print i
# Frage 5
print "\n5) Pflanzen, welche einen Preis groesser als 5 haben?"
xpath5 = "/PLANTS/PLANT[./PRICE > '5']"
q5 = tree.xpath(xpath5)
print ">>> %s\n" % xpath5
for i in q5:
    print etree.tostring(i)
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#PCL II Uebung 02
```

U2.3
3 XML mit Python: Informationen extrahieren
Schreibe ein Python Programm, welches aus dem XML

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
#Aufgabe 3
#Musterloesung
#AutorIn: Irene
from lxml import etree as ET
#Initiating etree object form XML file
plants_file = ET.parse("plants.xml")
plants = plants_file.getroot()
zone5 = []
zone3 = []
cost = 0
shade = []
#Iterating through XML file
for plant in plants:
   if plant.get('zone') == '5':
      for common in plant.iter('COMMON'):
         zone5.append(common.text)
   elif plant.get('zone') == '3':
      name = ''
      light = ''
      for common in plant.iter('COMMON'):
         name = common.text
      for lightcond in plant.iter('LIGHT'):
         light = lightcond.text
      zone3.append((name,light))
   for price in plant.iter('PRICE'):
      cost += float(price.text)
   for lightcond in plant.iter('LIGHT'):
      if lightcond.text == 'Shade':
         for botanical in plant.iter('BOTANICAL'):
            shade.append(botanical.text)
#counting elements with xpath
count = plants.xpath('count(//PLANT)')
#printing results
print 'Pflanzen der Zone 5:'
for plant in zone5:
   print plant
print '\nPflanzen der Zone 3 und ihre Lichtbedingung: '
for plant in zone3:
   print plant[0],':',plant[1]
print '\nSummer aller Kosten:',cost
print '\nDie botanischen Namen der Pflanzen, welche Schatten brauchen: '
for plant in set(shade):
   print(plant)
print '\nAnzahl Pflanzen: ',count


#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# PCL-I: Uebung 02 - Aufgabe 3, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz'       : '97-923-163',
#                            'Linus Manser'       : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# Aufruf des Programms:
# Version 1: python ex03.py
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
## necessary imports to parse an xml file
from lxml import etree
from StringIO import StringIO
import codecs
import sys
## parse the plants.xml file and save it as "tree"
tree = etree.parse("plants.xml")
# Frage 1
print "\n1) Welche Pflanzen haben bei der Eigenschaft 'Zone' einen Wert von 5?"
xpath1 = "/PLANTS/PLANT[@zone = '5']"
q1 = tree.xpath(xpath1)
print ">>> %s\n" % xpath1
for i in q1:
   print etree.tostring(i)


# Frage 2
print "\n2) Wieviel würde es kosten, wenn man eines jeder Pflanze kaufen würde?"
xpath2 = "sum(/PLANTS/PLANT/PRICE/text())"
q2 = tree.xpath(xpath2)
print ">>> %s\n" % xpath2
print q2
# Frage 3
print "\n3) Welche Pflanzen brauchen Schatten, 'Shade'? Nenne ihre botanischen Namen."
xpath3 = "/PLANTS/PLANT[./LIGHT/text() = 'Shade']/BOTANICAL/text()"
q3 = tree.xpath(xpath3)
ans = [i for i in set(q3)]
print ">>> %s\n" % xpath3
for i in ans:
   print "-",i


# Frage 4
```

Dokument plants.xml verschiedene Informa-
tionen extrahiert. Dein Programm soll folgende Fragen über die Pflanzen im Dokument beantworten
und die Antworten im Terminal herausgeben.
- Welche Pflanzen haben bei der Eigenschaft ‘Zone’ einen Wert von 5?
- Wie viel würde es kosten, wenn man eines von jeder Pflanze kaufen würde?
- Welche Pflanzen brauchen Schatten, ‘Shade’? Nenne ihre botanischen Namen.
- Wie viele Pflanzen gibt es insgesamt? Benutze hierfür XPath.

Benutze für diese Aufgabe die lxml Bibliothek.
Abzugeben: Dein Python-Programm mit dem Namen ex03.py.

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr><td>

```
print "\n4) Wieviele Pflanzen gibt es insgesamt?"
xpath4 = "count(/PLANTS/PLANT)"
q4 = tree.xpath(xpath4)
print ">>> %s\n" % xpath4
print int(q4)
```

</td><td></td></tr>
<tr><td>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#PCL II Uebung 02
#Aufgabe 3
#Musterloesung
#AutorIn: Irene
from lxml import etree
from nltk.corpus import brown
print "Putting together XML Tree..."
#setting root with its attributes
root = etree.Element('browncorpus')
count_cat = len(brown.categories())
count_files = len(brown.fileids())
root.set("categories",str(count_cat))
root.set("files",str(count_files))
for id in brown.fileids():
   text = etree.SubElement(root,'file')
   text.set("id",id)
   text.set("cat",brown.categories(fileids=id)[0])
   w_count = len(brown.words(fileids=id))
   words = etree.SubElement(text,'words')
   words.text = str(w_count)
   sent = brown.sents(fileids=id)[-1]
   lastsent = etree.SubElement(text, 'last_sentence')
   lastsent.text = ' '.join(sent)
print "Creating XML file..."
#putting the XML "tree" together and write the actual XML file
tree = etree.ElementTree(root)
tree.write('thebrowncorpus.xml', xml_declaration=True,pretty_print=True, encoding='utf-
8')
print "XML file thebrowncorpus.xml ready."


#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# PCL-I: Uebung 02 - Aufgabe 4, FS16
# Autoren:
# c(Student, Martikelnummer) ->    {'Roland Benz'       : '97-923-163',
#                             'Linus Manser'       : '13-791-132'}
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
# Aufruf des Programms:
# Version 1: python ex04.py (user input: output_pcl_ex04)
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~
from nltk.corpus import brown
from lxml import etree
import codecs
## time module for fun
import time
def build_xml_file():
    """
    NO INPUT
    OUTPUT: Builds the XML-file with information from the brown corpus
    """
    # starting message
    print "building XML-file ..."
    # start time-measurement
    start = time.time()
    # number of texts (500) and categories (15)
    text_number = len(brown.fileids())
    cat_number = len(brown.categories())
    # creating a new root element called 'browncorpus'
    # with attributes textcount and categorycount
    root = etree.Element("browncorpus", textcount=str(text_number),
                         categorycount=str(cat_number))
    # for every fileID (individual texts) in the browncorpus...
    iter=1
    cnt=1
    for id in brown.fileids():
        # adding the SubElement 'textfile' to 'browncorpus'
        textfile = etree.SubElement(root, "textfile")
        # define attributes for each textfile element
        textfile.attrib["textID"] = str(id)
        list_category=brown.categories(fileids=[id]) #returns one element list
        textfile.attrib["category"] = str(list_category[0])
        # add SubElement 'wordcount' to 'textfile'
        wordcount = etree.SubElement(textfile, "wordcount")
        list_words=brown.words(fileids=[id])
        wordcount.text = str(len(list_words))
        # add SubElement 'lastsentence' to 'textfile'
```

</td><td>
U2.4
4 XML mit Python: Datei erstellen
Schreibe nun ein Python Programm, welches aus dem Brown Korpus folgende Informationen sammelt und in Form eines XML Dokuments speichert. Das resultierende XML Dokument soll die Informationen in der hier angegebenen Form speichern.
Falls dir ‘Brown Korpus’ nichts sagt, befolge die Anweisungen am Ende dieser Übung.
• Anzahl Kategorie und Texte im Korpus als Attribute (des Wurzelelementes browncorpus)
• Pro Text:
– Text-ID als Attribut
– Kategorie als Attribut
– Anzahl Wörter als Subelement
– Der letzte Satz als Subelement
Abzugeben: Dein Python-Programm mit dem Namen ex04.py.
</td></tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
        last_sentence = etree.SubElement(textfile, "lastsentence")
        str_last_sentence=brown.sents(fileids=[id])[-1]
        last_sentence.text = " ".join(str_last_sentence)
        # just for fun
        if iter%50==0:
            percent=text_number/50*cnt
            print percent,"% completed"
            cnt=cnt+1
        iter=iter+1
    end = time.time()
    # time difference (just for fun)
    elapsed_time = end - start
    # completion message
    print "... completed after %.3f seconds" % elapsed_time
    # returns root element (whole xml-document)
    return etree.tostring(root, xml_declaration=True,
                          encoding="utf-8", pretty_print=True)
def write_out_file(filename):
    """
    INPUT: filename
    OUTPUT: writes out the built xml-file to the file 'filename.xml'
    """
    # create or overwrite file this defined filename
    with codecs.open(filename, "w", "utf-8") as outfile:
        str_etree = build_xml_file()
        outfile.write(str_etree)
    # completion message
    print "wrote out to XML-file '%s'" % filename
if __name__ == "__main__":
    # file in which output is written (stored in same folder like ex04.py)
    # e.g. output_pcl_ex04
    filename = raw_input("please enter a filename for your XML-file, "
                         "in which the output is written: ") + ".xml"
    # opens file object (filename)
    # calls function build_xml_file() which:
    #   iterates through all files in brown corpus
    #   extracts information (id, category, nr of words, last sentence)
    #   builds xlm tree
    # writes tree to file
    write_out_file(filename)
```

```
<?xml version='1.0' encoding='utf-8'?>
<browncorpus categorycount="15" textcount="500">
  <textfile textID="ca01" category="news">
    <wordcount>2242</wordcount>
    <lastsentence>`` There wasn't a bit of trouble '' .</lastsentence>
  </textfile>
  <textfile textID="ca02" category="news">
    <wordcount>2277</wordcount>
    <lastsentence>Research projects as soon as possible on the causes and prevention of
dependency and illegitimacy .</lastsentence>
  </textfile>
  <textfile textID="ca03" category="news">
    <wordcount>2275</wordcount>
    <lastsentence>The rule was enforced by demand of Sen. Wayne Morse ( D. , Ore. ) in
connection with President Eisenhower's cabinet selections in 1953 and President
Kennedy's in 1961 .</lastsentence>
  </textfile>
  <textfile textID="ca04" category="news">
    <wordcount>2217</wordcount>
    <lastsentence>Among arrests reported by the Federal Bureau of Investigation in
1959 , about half for burglary and larceny involved persons under 18 years of
age .</lastsentence>
  </textfile>
  <textfile textID="ca05" category="news">
    <wordcount>2244</wordcount>
    <lastsentence>Local police have hesitated to prosecute them because of the heavy
court costs involved even for the simplest offense .</lastsentence>
  </textfile>
…
…
…
</browncorpus>
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# PCL2-Ü3-Aufgabe 1
# Musterlösung von Raphael Balimann (raphael.balimann@uzh.ch) - HS 2015
#
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
# Main function to showcase functionality
# One corpus for ham, one for spam, no other distinctions
# Files are expected to be merged into one respective directory,
# other layouts are easily doable with simple loops and function calls
def main():
    ham_dir =  'ham/'
    ham_corpus = buildCorpus(ham_dir)
    spam_dir = 'spam/'
```

U3.1
1 Ein Korpus erstellen
In dieser Übung werden wir eine Sammlung von Emails1 , welche im Zuge einer Untersuchung veröf-
fentlicht wurden, als Grundlage für verschiedene Zwecke benutzen.
Um diese Emails einfach zu verarbeiten soll daher ein Korpus erstellt werden, welches dann unter
Python mit NLTK einfach verwendbar sein sollte.
Lade die Daten aus OLAT (Materials/Additional_Material/Enron) herunter und überlege dir
wie du die Korpora gestalten würdest:
• Sollen getrennte Korpora für Spam und Nicht-Spam erstellt werden?
• Welche Metadaten sollten erfasst werden?
• Wie sollten die Daten in den Programmen gespeichert

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

<table>
<tr>
<td>

```
    spam_corpus = buildCorpus(spam_dir)
    if (utf8Checker(ham_corpus) & utf8Checker(spam_corpus)):
        return
    else:
        print """Something's wrong with the file encodings.
                Please refer to the comment in aufgabe_01.py to
                make sure that all files are converted to UTF-8."""
        return
# Wrapper function to build the corpus given the directory
def buildCorpus(corpus_directory):
    corpus = PlaintextCorpusReader(corpus_directory,'.*')
    return corpus
# A quick check if everything is properly encoded as UTF-8
def utf8Checker(corpus):
    for i in corpus.fileids():
        if corpus.encoding(i) != "utf8":
            return False
    return True
# Standard boilerplate to call the main() function to begin the program.
if __name__ == '__main__':
    main()
"""
If there are problems with encodings (using the data from the source webpage),
the following snippet proved helpful to convert all files to the same encoding.
Needs to be run in the directory where the files are stored.
find . -name "*.txt" -exec sh -c "iconv -f ISO-8859-1 -t UTF-8 {} > {}.utf8"  \; -exec
mv "{}".utf8 "{}" \;
Courtesy of 'UTF_or_Death' on http://stackoverflow.com/a/24836200
"""



#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 03 - Aufgabe 1, FS16
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                          'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Aufruf/Import des Programms:
# python ex01.py
# import ex01.py
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# generate corpora with nltk PlaintextCorpusReader
#==================================================
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader as PCR
import os
# generate corpus, encoding is utf-8 by default
def own_corpus_reader():
   # get current working directory
   current_dir = os.getcwd()
   corpusdir = current_dir + "/Enron/enron_subset"
   test_set_dir = current_dir + "/Enron/test_set"

   #read all .txt files in directory
   ham_spam_corpus = PCR(corpusdir, ".*\.txt")
   spam_corpus = PCR(corpusdir, ".*spam\.txt")
   ham_corpus = PCR(corpusdir, ".*ham\.txt")
   test_corpus = PCR(test_set_dir, ".*\.txt")
    #define here other variables
    #...words, nr of tokens and so on
    #or even better make a class with instance variables
    #return all variables separated by comma
   return ham_spam_corpus, ham_corpus, spam_corpus, test_corpus

#not run when file imported with: import ex01
if __name__ == "__main__":
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "corpora generated"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
```

</td>
<td>
werden?
• Wie sollten die Daten auf der Festplatte gespeichert werden?
• Wie könnte Spam markiert werden?
Implementiere deine Überlegungen als Korpus, zum Beispiel mit Hilfe des PlaintextCorpusReader
aus der NLTK-Korpus-Kollektion.
Beachte, dass die folgenden Aufgaben auf deinen erstellten Korpora aufbauen, von daher ist es
ratsam, sich zu überlegen welche Merkmale nützlich sein könnten.
Abzugeben ist ein Skript, welches ein oder mehrere Korpora erstellt und sich in den folgenden
Aufgaben einfach als Modul aufrufen lässt.
</td>
</tr>
<tr>
<td>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# PCL2-Ü3-Aufgabe 2
# Musterlösung von Raphael Balimann (raphael.balimann@uzh.ch) - HS 2015
#
import nltk
import aufgabe_01
# Main function to showcase functionality
def main():
    ham_corpus = aufgabe_01.buildCorpus('enron_data/enron1/ham/')
    spam_corpus = aufgabe_01.buildCorpus('enron_data/enron1/spam/')
    # creating a list from generators to feed to n-gram builder
    ham = [i for i in ham_corpus.words()]
    spam = [i for i in spam_corpus.words()]
    ham2 = build_ngrams(ham,2)
    spam2 = build_ngrams(spam,2)
    print "The bigram 'you may' is ",
    print('IN THE CORPUS' if check_members(ham2,["you","may"]) else 'NOT IN THE CORPUS')
```

</td>
<td>
U3.2
2 n-Gramm-Modelle
Schreibe eine Sammlung von Funktionen, welche das Korpus aus Aufgabe 1 in n-Gramme umwandelt
und auf verschiedene Eigenschaften testet:
a) Ein Test, ob ein bestimmtes n-Gramm im Korpus vorkommt.
b) Eine Funktion die eine Liste mit allen n-Grammen ausgibt, welche mit dem gewählten (n-1)-
Gramm beginnen.
c) Eine Funktion zur Berechnung der unbedingten Wahrscheinlichkeit eines n-Gramms.
d) Eine Funktion zur Berechnung der bedingten Wahrscheinlichkeit eines n-Gramms.
e) Eine Funktion welche testet, ob ein n-Gramm eine Kollokation darstellt; die Kriterien sind
selber zu wählen und im Programmcode zu begründen.
Da NLTK schon viele Werkzeuge zum Umgang mit n-Grammen enthält, sollen die Funktionen dei-
ner Sammlung auf keine externe Module zugreifen, ausser natürlich auf deine Implementation von
</td>
</tr>
</table>

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    print "and it is ",
    print('A COLLOCATION' if is_collocation(ham2,["you","may"]) else 'NOT A 
COLLOCATION')
    variety = get_finals(spam2,["viagra"])
    print "Here are some n-grams with 'viagra' in them, lean back and enjoy!"
    print variety
    viagra_click = uncondprobability(spam2,["viagra","click"])
    print """The bigram 'viagra click' occupies\
        %f percent of the spam messages.""" %(viagra_click*100)
    viagra_works = condprobability(spam2,["viagra","works"])
    print """If the word 'viagra' appears in a spam message,\
        the word 'works' will appear with a probability of\
        %f percent afterwards.""" %(viagra_works*100)
    return
# Wrapper function to build n-grams from a list provided
# Using zip to merge multiple instances of the same list into n-tuples
# Requires modifications to work as a generator
def build_ngrams(raw_token,n):
    return zip(*[raw_token[i:] for i in range(n)])
# Function to check if a list of tokens is present as an n-gram
# Returns True if found, False otherwise
# Checks if sizes of n-gram match, returns False if mismatching
def check_members(ngram_list, token_list):
    if len(token_list) != len(ngram_list[0]):
        return False
    if tuple(token_list) in ngram_list:
        return True
    return False
# Wrapper function to count n-rgams in a list of n-grams
def count_ngram(ngram_list,ngram):
    ngram = tuple(ngram)
    return ngram_list.count(ngram)
# Function to get all n-grams that start with the token found in backgram
# Returns a list of all n-grams found, so it is possible to use further
def get_finals(ngram_list,backgram):
    results = []
    backgram = tuple(backgram)
    if len(ngram_list[0]) != (len(backgram)+1):
        # returning false to signal a problem
        return False
    for ngram in ngram_list:
        if (ngram[0:len(backgram)] == backgram):
            results.append(ngram)
    return results
# Function to see if a n-gram is a collocation
# Based on an arbitrary treshold
# Returns a boolean value for further processing
def is_collocation(ngram_list,ngram):
    # arbitrary value, set low for testing values
    # 1% of all n-grams is a good value though
    treshold = 0.001
    # getting all n-1-grams
    backoffs = get_finals(ngram_list,ngram[:-1])
    if (count_ngram(ngram_list,ngram) / float(len(backoffs)) > treshold):
        return True
    else:
        return False
# Function to get the probability of a n-gram within a corpus
def uncondprobability(ngram_list, ngram):
    if len(ngram) != len(ngram_list[0]):
        # returning false to signal a problem
        return False
    probability = count_ngram(ngram_list,ngram)/float(len(ngram_list))
    return probability
# Function to get the conditional probability of a n-gram within a corpus
# n representing the level
def condprobability(ngram_list, ngram):
    if len(ngram) != len(ngram_list[0]):
        # returning false to signal a problem
        return False
    backoffs = get_finals(ngram_list,ngram[:-1])
    probability = count_ngram(ngram_list,ngram) / float(len(backoffs))
    return probability
# Standard boilerplate to call the main() function to begin the program.
if __name__ == '__main__':
    main()


#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 03 - Aufgabe 2, FS16
#
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                               'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# with import no code should be running
# (only forward declaration of classes and functions)
# instead of converting every other int to floats,
# we decided to "cheat" and imported division
from __future__ import division
import ex01
import time
```

Aufgabe 1.
Abzugeben ist ein Skript, welches die Korpora aus Aufgabe 1 einliest, die Texte in frei wählbare2
n-Gramme umwandelt und die entsprechenden Abfragefunktionen zur Verfügung stellt. Das Skript
sollte sich in den folgenden Aufgaben ohne Probleme als Modul aufrufen lassen.

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Main
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# set the flags for debug info, and data structure
debug=0
ds1=0 #data structure 1){k-gram:{word:value}}
# if ds1=1 then ds2=0 and vice versa
ds2=(ds1+1)%2 #data structure 2){k-gram:(k,{word:value})}
def main():
   print "~~~~~~~~~~~~Aufgabe 2~~~~~~~~~~~~~"
   #load ham spam corpus
   # ------------------
   ham_spam_corpus, ham_corpus, spam_corpus, test_corpus = \
      ex01.own_corpus_reader()
   nr_of_documents_loaded = len(ham_spam_corpus.fileids())
   print ham_spam_corpus
   print "nr of documents", nr_of_documents_loaded
   # build data structure containing n-grams
   # ---------------------------------------
   # 1){k-gram:{word:value}}
   # 2){k-gram:(k,{word:value})}
   # n-gram = k-gram + word
   # value: absolute count of n-gram
   start = time.time() #time measurment
   if (ds1):dict_i_to_j_gram_keys = {0: {0: 0}} #dummy element
   if (ds2):dict_i_to_j_gram_keys = {0: (0,{0: 0})}  # dummy element
   dict_i_to_j_gram_keys, corpus_words = \
      build_i_gram_to_j_gram_keys(ham_spam_corpus, 1, 3)
   dict_i_to_j_gram_keys.pop(0, None) #remove dummy element
   end = time.time() #time measurment
   elapsed_time = end - start #time measurment
   if (debug): print_list(corpus_words)
   print "nr of tokens:", len(corpus_words), \
      "\ntime to build n-grams:", elapsed_time
   build_dict_test(dict_i_to_j_gram_keys, 1)
   # Exercise 2a) - 2e)
   # ---------------------------------------
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "2a) checks whether the k-gram key exists"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   kgram="Subject :" #input
   out = k_gram_exists(dict_i_to_j_gram_keys, kgram)
   print "ngram <%s> exists: %s" %(kgram, out)
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "2b) prints the n-grams = words given k-gram"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   kgram = "Subject :" #input
   out = print_n_grams(dict_i_to_j_gram_keys, kgram)
   print "List of n-grams = words given k-gram <%s>: " %(kgram)
   print out
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "2c) uncond. probability of k-gram"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   kgram = "Subject :" #input
   #unconditional with respect to k-grams of same length k
   k = len(kgram.split())
   kgrams_cnt = len(corpus_words) + 1 - k
   print "kgrams_cnt = total_words + 1 - k: ", kgrams_cnt
   prob, kgram_cnt = unconditional_prob_of_k_gram(
      dict_i_to_j_gram_keys, kgram, kgrams_cnt)
   print "count of k-gram <%s> = %s" % (kgram, kgram_cnt)
   print "unconditional probabilty of k-gram <%s> = %s" %(kgram, prob)
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "2d) cond. probability of word given kgram"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   kgram = "Subject" #input
   word = ":" #input
   (prob, cond_word_cnt, kgram_cnt) = \
      conditional_prob_of_word_given_k_gram\
         (dict_i_to_j_gram_keys, kgram, word )
   print "count of k-gram <%s> = %s" % (kgram, kgram_cnt)
   print "conditional count of word <%s> = %s" \
        %(word, cond_word_cnt)
   print "conditional probabilty of <%s> given <%s> = %s" \
        %(kgram, word, prob)
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "2e) test collocation"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   ngram = "Subject : vastar" #input
   print ngram_is_collocation(dict_i_to_j_gram_keys, ngram)
   print "~~~~~~~~~~~~~~end~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# N-GRAM-BUILDING-PART starts here
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def build_i_gram_to_j_gram_keys(corpus, i, j):
   """
   builds a data structure of nested dictionaries  {0: {0: 0}, 0: {0: 0}, ...}
      out of a PlaintextCorpusReader object containing txt files
   outer key is a k-gram with i <= k <= j
   inner key is the word following th k-gram
   inner value is the count of occurences of n-gram=k-gram+word
   """
```

```
   # tokenize the corpus into list
   corpus_words = corpus.words()
   # data structure with i-gram to j-gram key
   if (ds1):dict_of_igram_to_jgram_keys = {0: {0: 0}}
   if (ds2):dict_of_igram_to_jgram_keys ={0: (0, {0: 0})}
   # k-gram iterator: i <= k <= j
   if (ds1):dict_of_kgrams_keys = {0: {0: 0}}
   if (ds2):dict_of_igram_to_jgram_keys ={0: (0, {0: 0})}
   print "building n-gram data-structure..."
   iter=1
   #iterate k from i to j
   for k in range(i,j+1):
      #build dict with k-gram keys
      dict_of_kgrams_keys = build_k_gram_keys(corpus_words, k)
      #update dict with additional k-gram keys
      dict_of_igram_to_jgram_keys.update(dict_of_kgrams_keys)
      percent_done=iter*int(1/(j+1-i)*100)
      print "%s percent done..." %(percent_done),
      iter+=1
   print "... done"
   #return dict
   return dict_of_igram_to_jgram_keys, corpus_words
def build_k_gram_keys(corpus_words, k):
   """
   builds a data structure of nested dictionaries  {0: {0: 0}, 0: {0: 0}, ...}
      out of a PlaintextCorpusReader object containing txt files
   outer key is a k-gram with k fixed
   inner key is the word following th k-gram
   inner value is the count of occurences of n-gram=k-gram+word
   """
   # nr of tokens in corpus
   length=len(corpus_words)
   # data structure
   if (ds1):dict_of_kgram_keys={0:{0:0}}
   if (ds2):dict_of_kgram_keys = {0: (0,{0: 0})}
   if (ds2):tuple_of_next_words_and_k=(0,{0: 0})
   dict_of_next_words={0:0}
   # iterate through all tokens in list
   for i in range(0,length-k):
      # extract the k-gram key as list from the token list
      k_gram_list = corpus_words[i:i+k] #excl. i+k
      # convert the k-gram key as list into a string
      k_gram_string =' '.join(k_gram_list)
      # the word following the k-gram key
      next_word = corpus_words[i+k] #incl. i+k
      # return the nested dictionary with key = k-gram
      if (ds1):dict_of_next_words = dict_of_kgram_keys.get(k_gram_string)
      if (ds2):
         # the nested tuple may be empty
         if dict_of_kgram_keys.get(k_gram_string) is not None:
            dict_of_next_words = dict_of_kgram_keys.get(k_gram_string)[1]
         else:
            dict_of_next_words = None
      # the nested dictionary may be empty (of type None)
      if dict_of_next_words is not None:
         # the dictionary entry with key = next word
         # gets incremented (n-gram counter)
         cnt_ngram = dict_of_next_words.get(next_word,0)
         cnt_ngram = cnt_ngram + 1
         # the nested dictionary gets updated
         dict_of_next_words.update({next_word: cnt_ngram})
         # make a nested tuple
         if (ds2):tuple_of_next_words_and_k = (k, dict_of_next_words)
         # the outer dictionary gets updated
         if (ds1):dict_of_kgram_keys.update({k_gram_string:dict_of_next_words})
         if (ds2):dict_of_kgram_keys.update(
            {k_gram_string: tuple_of_next_words_and_k})
      else:
         # the outer dictionary gets updated for a new k-gram
         if (ds1):dict_of_kgram_keys.update({k_gram_string: {next_word:1}})
         if (ds2):dict_of_kgram_keys.update(
            {k_gram_string: (k, {next_word: 1})})
   #return dictionary
   return dict_of_kgram_keys
def build_dict_test(dict, just_summary):
   """
   prints data structur sorted by keys
   or
   just a summary with length information
   Data structures:
   1) {k-gram:{word:value}}
   2) {k-gram:(k,{word:value})}
   """
   print "infos about our data structure"
   if (just_summary==0):
      print "~~~~~~~~~~~~~~~dictionary~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
      print dict
      print "~~~~~~~~~~~~~~~key {word: cnt}~~~~~~~~~~~~~~~~~~~~~~~~~"
      for key in sorted(dict.keys()):
         print  "%15s\t%15s" % (key, dict[key])
      print "~~~~~~~~~~~~~~~key word cnt~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
      for key in sorted(dict.keys()):
```

```
            if (ds1):nested_keys=sorted(dict[key].keys())
            if (ds2):nested_keys=sorted(dict[key][1].keys())
            for word in nested_keys:
               if (ds1):print "%15s\t%15s\t%15s" % (
                  key, word, dict[key][word])
               if (ds2): print "%15s\t%15s\t%15s" % (
                  key, word, dict[key][1][word])
   else:
      nr_of_keys=len(dict.keys())
      sum_up = 0
      for key in dict.keys():
         if (ds1):values = dict[key].values()
         if (ds2): values = dict[key][1].values()
         sum_up=sum_up+sum(values)
      print " nr of keys:", nr_of_keys, "\n nr of k-grams", sum_up
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
def print_list(corpus_words):
   """
   prints a list comma separated
   """
   print "~~~~~~~~~~~~~~~List~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   for word in corpus_words:
      print word,
   print
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Aufgaben a) bis e)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# a)
def k_gram_exists(dict, kgram):
   """
   function to check whether an ngram exists as a key in the main dict
   """
   if kgram in dict.keys():
      return True
   else:
      return False
# b)
def print_n_grams(dict, kgram):
   """
   function to build a list of all possible ngram-completions
      given a k-gram (n-grams = k-gram + words)
   Example:
      corpus = "The car is green. The house is big"
      complete_ngram(dict, 'is')
      >>> [u'is green', u'is big']
   """
   #checks whether the k-gram key exists
   if k_gram_exists(dict, kgram):
      # to store n-grams = k-gram + words
      ngram_list = []
      # iterate through the words of the nested dictionary
      # 1){k-gram:{word:value}}
      # 2){k-gram:(k,{word:value})}
      if (ds1):dict_nested = dict[kgram].keys()
      if (ds2):dict_nested = dict[kgram][1].keys()
      for word in dict_nested:
         # build ngram and add it to the ngram_list
         new_ngram = kgram + " " + word
         # append n-gram to list
         ngram_list.append(new_ngram)

      # returns the list of n-grams
      return ngram_list
   else:
      print "please enter a k-gram that exsists in our data"
# c)
def unconditional_prob_of_k_gram(dict, kgram, kgrams_cnt):
   """
   function to compute the unconditional probability of a k-gram
   total count of k-grams equals corpus length minus k
   """
   # checks whether the k-gram key exists
   if k_gram_exists(dict, kgram):
      # frequency of the specific ngram as a sum of all
      # frequencies of following words
      # 1){k-gram:{word:value}}
      # 2){k-gram:(k,{word:value})}
      if (ds1):kgram_cnt = sum(dict[kgram].values())
      if (ds2):kgram_cnt = sum(dict[kgram][1].values())

      # return unconditional probability of k-gram
      return (kgram_cnt / kgrams_cnt), kgram_cnt
   else:
      print "please enter a ngram that exsists in our data"
# d)
def conditional_prob_of_word_given_k_gram(dict, kgram, word):
   """
   function to compute the conditional probability of an n-gram
      given the k-gram
   """
   # checks whether the k-gram key exists
```

```
    if k_gram_exists(dict, kgram):
        # occurences of k-gram
        # 1){k-gram:{word:value}}
        # 2){k-gram:(k,{word:value})}
        if (ds1):k_gram_cnt = sum(dict[kgram].values())
        if (ds2): k_gram_cnt = sum(dict[kgram][1].values())
        # occurences of word given k-gram
        if (ds1):cond_word_cnt = dict[kgram][word]
        if (ds2): cond_word_cnt = dict[kgram][1][word]
        return (cond_word_cnt / k_gram_cnt), cond_word_cnt, k_gram_cnt
    else:
        print "please enter a ngram that exsists in our data"
# e)
def ngram_is_collocation(dict, ngram):
    """
    function applying the bigram_collocation-check to tri-grams.
    reasoning behind it: if the first two words of a trigram aren't
    a collocation, the trigram won't be a collocation either
    """
    len_ngram = len(ngram.split())
    if len_ngram < 3:

        if bigram_is_collocation(dict, ngram):
            return "'%s' is a collocation" % ngram
        else:
            return "'%s' is not a collocation" % ngram

    elif len_ngram == 3:

        if trigram_is_collocation(dict, ngram):
            return "'%s' is a collocation" % ngram
        else:
            return "'%s' is not a collocation" % ngram

    else:
        return "Little Britain: computer says nooo.\n " \
               "Monty Python: traceback"
def bigram_is_collocation(dict, bigram):
    """
    decides whether two words tend to occur together more often than alone.
    only checks the result for bigrams
    """
    # checks whether the k-gram key exists
    if k_gram_exists(dict, bigram):

        try:
            first_word = bigram.split()[-2]
            second_word = bigram.split()[-1]
        except IndexError:
            print "please enter a bigram"
            exit();
        if (ds1):cnt_bigram = dict[first_word][second_word]
        if (ds2):cnt_bigram = dict[first_word][1][second_word]

        cond_prob = conditional_prob_of_word_given_k_gram(
            dict, first_word, second_word)[0]

        ## defining the thresholds for becoming a collocation
        # the conditional probability of the first word given the
        # second should be more than 0.5 (this value can be changed)
        cond_prob_threshold = 0.5
        print "cond_prob_threshold:", cond_prob_threshold
        # the collocation should occur more than 5 times in our
        # data to count as collocation (this can be changed as well)
        absolute_frequency_threshold = 5
        print "absolute_frequency_threshold:", absolute_frequency_threshold
        if cnt_bigram > absolute_frequency_threshold and \
                    cond_prob > cond_prob_threshold:
            True
        else:
            False
    else:
        print "please enter a ngram that exsists in our data"
def trigram_is_collocation(dict, ngram):
    """
    decides whether two words tend to occur together more often than alone.
    only checks the result for bigrams
    """
    # checks whether the k-gram key exists
    if k_gram_exists(dict, ngram):
        pre_gram = " ".join(ngram.split()[:2])
        if bigram_is_collocation(dict, pre_gram):
            return bigram_is_collocation(dict, ngram)
        else:
            return False
    else:
        print "please enter a ngram that exsists in our data"
#Main
if __name__ == "__main__":
    main()
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# PCL2-Ü3-Aufgabe 3
# Musterlösung von Raphael Balimann (raphael.balimann@uzh.ch) - HS 2015
#
import nltk
import aufgabe_01
import random
# Main function to showcase functionality
def main():
    documents, spam_corpus, ham_corpus = data_builder()
    spam_words = word_builder(spam_corpus)
    ham_words = word_builder(ham_corpus)
    wordlist = spam_words + ham_words
    words = top_words(wordlist)
    featuresets = [(document_features(document,words), classification) for
(document,classification) in documents]
    train_set, test_set = featuresets[100:], featuresets[:100]
    classifier = nltk.NaiveBayesClassifier.train(train_set)
    print "Classifier accuracy: ",
    print(nltk.classify.accuracy(classifier, test_set))
    classifier.show_most_informative_features(5)
    return
# Function to build a collection of classifiable data
# Returns the corpus objects as well to save precious computing (and user) time
# Could be much nicer, maybe a good exercise for the final exam?
def data_builder():
    data = []
    spam_path = 'enron_data/full/spam/'   #full data set
    ham_path = 'enron_data/full/ham/'    #full data set
    # spam_path = 'enron_data/enron1/spam/' #testing
    # ham_path = 'enron_data/enron1/ham/' #testing
    spam = aufgabe_01.buildCorpus(spam_path)
    ham = aufgabe_01.buildCorpus(ham_path)
    spam_docs = [(list(spam.words(fileid)), 'spam') for fileid in spam.fileids()]
    ham_docs = [(list(ham.words(fileid)), 'ham') for fileid in ham.fileids()]
    data = spam_docs + ham_docs
    random.shuffle(data)
    return (data,spam,ham)
# Helper function to transform corpus obejcts into a list of words
def word_builder(corpus):
    words = []
    for word in corpus.words():
        words.append(word)
    return words
# Helper function to get the top most frequently used words
def top_words(wordlist):
    max_range = 2000
    all_words = nltk.FreqDist(w.lower() for w in wordlist)
    word_features = list(all_words)[:max_range]
    return word_features
# The infamous document features with a basic example of a feature
# Feel free to play around with the features, the sky's the limit and the accuracy may
be your guide
def document_features(document, words):
    document_words = set(document)
    features = {}
    for word in words:
        # a gain of 18% compared to the pure baseline, if there's anything to top that!
        features[u'contains({})'.format(word)] = (word in document_words)
    return features
# # Baseline empty feature to check if classifier runs as expected
# def document_features(document,words):
#
#     return {}
# Standard boilerplate to call the main() function to begin the program.
if __name__ == '__main__':
    main()


#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 03 - Aufgabe 3, FS16
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                          'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
from __future__ import division
#used corpora as data input
import ex01
import ex02
#used functionality for ML
import operator
import random
import nltk
from collections import Counter
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Main
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# set the flags for debug info
debug=0
def main():
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

U3.3
3 Klassifikation: Spam, Spam, Spam!
Nun ist es an der Zeit, die Daten einzusetzen um Emails auf Spam (unerwünschte) und Ham (erwünschte Emails) zu klassifizieren.
Verfahre analog zu Sektion 1.3 von Kapitel 6 im NLTK-Buch, wobei hier die Struktur deines Korpus (oder deiner Korpora) den Ablauf stark beeinflusst.
a) Verwende am Anfang noch keine Merkmale (Features), sondern das untenstehende Code-Schnipsel. Die resultierende Präzision des Klassifikators stellt die Baseline dar, welche dir bei der Weiterentwicklung deiner Merkmale hilft. Auf welcher Basis entscheidet der Klassifikator, ob ein Dokument Spam oder Ham ist?
b) Erstelle nun eigene Merkmale wie in Sektion 1.3 beschrieben steht. Experimentiere mit verschiedenen Merkmalen sowie deren Kombinationen, vergleiche die Präzision deines Klassifikators mit der Baseline und notiere deine Feststellungen direkt im Programmcode beim jeweiligen Merkmal.

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
print "1~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "loading corpora ...\n"
ham_spam_corpus, ham_corpus, spam_corpus, test_corpus = \
   ex01.own_corpus_reader()
# ham spam corpus
nr_of_docs_loaded = len(ham_spam_corpus.fileids())
print "nr of documents loaded: ", nr_of_docs_loaded
# ham corpus
nr_of_hams_loaded = len(ham_corpus.fileids())
print "nr of hams loaded: ", nr_of_hams_loaded
# spam corpus
nr_of_spams_loaded = len(spam_corpus.fileids())
print "nr of spams loaded: ", nr_of_spams_loaded
# test corpus
nr_of_tests_loaded = len(test_corpus.fileids())
print "nr of tests loaded: ", nr_of_tests_loaded

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "2~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "converting corpora into a list of tuples\n"
# ------------------------------------
#[([text-tokens],category=pos/neg),"
#([text-tokens],category=pos/neg),...]
# concept: used to generate the features list and train the NBC
documents_as_list_of_words_and_label = \
   [build_document_tuples(ham_spam_corpus, fileid)
   for fileid in ham_spam_corpus.fileids()]
# concept: used as completey unknown test set
# used at the end of the development process
# (ensures an unbiased test result)
test_set_as_list_of_words_and_label = \
   [build_document_tuples(test_corpus, fileid)
    for fileid in test_corpus.fileids()[500:]]
# concept: used as development test set to
# iteratively improve features
# (becomes biased after each iteration of improving
# feature extraction function)
errorsampling_set_as_list_of_words_and_label_with_id = \
   [(build_document_tuples(test_corpus, fileid), fileid)
   for fileid in test_corpus.fileids()[:500]]

# randomly shuffle the words
# --------------------------
random.shuffle(documents_as_list_of_words_and_label)
random.shuffle(test_set_as_list_of_words_and_label)
random.shuffle(errorsampling_set_as_list_of_words_and_label_with_id)
if(debug):print documents_as_list_of_words_and_label
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "3~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "making a features list containing words and n-grams ...\n"
# ------------------------------------------------
# tokenize the corpus into list
corpus_words = ham_spam_corpus.words()
ham_words = ham_corpus.words()
spam_words = spam_corpus.words()
print "nr of token in corpus: ", len(corpus_words)
print "nr of token in ham: ", len(ham_words)
print "nr of token in spam: ", len(spam_words)
# the parameter is part of a heuristic:
# has the effect of a stop list (caps the occurences of
# words from above and floors the DELTAS of occurences
# of words from below beween hams and spams)
parameter = random.randrange(
   int(nr_of_docs_loaded*0.1), int(nr_of_docs_loaded*0.2))
# build most informative n-grams
# (unigrams, bigrams, trigrams)
sorted_ngram_list, sorted_list_of_ngram_tuples, ngram_dict = \
   most_informative_n_grams_dict(
   corpus_words, ham_words, spam_words, parameter)
print "nr of different words extracted: ", len(sorted_ngram_list)
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "list of most informative words in the corpus:\n" \
     "(only a subset printed)\n", sorted_ngram_list[:100]
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "list of most informative words in the corpus:\n" \
     "inclusive DELTA = difference of occurence in hams and spams" \
     "(only a subset printed)\n", sorted_list_of_ngram_tuples[:100]
# limit the feature dimensions
# parameter is a heuristic to limit the number of dimensions
parameter = min(nr_of_docs_loaded, 700)
word_features_list = sorted_ngram_list[:parameter]
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "4~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "calling feature extractor function ...\n"
# call feature extractor function that checks
# whether each of the words in the word_features_list
# is present in the given documents"
# ---------------------------------------------------
#[({text-tokens of document 1:True/False},category=ham/spam),
#({text-tokens of document 2:True/False},category=ham/spam),...]
#this is used in step 5 as training_set and development_test_set
#(notice: the same documents were used to make the features list
```

```
# in step 3)
documents_as_features_and_label = \
   [(document_features(list_of_words, word_features_list),
     doc_label)
     for (list_of_words, doc_label) in
    documents_as_list_of_words_and_label]
#this is used in step 6 as test_set
#(completely unknown documents to NBC)
test_set_as_features_and_label = \
   [(document_features(list_of_words, word_features_list),
     doc_label)
    for (list_of_words, doc_label) in
    test_set_as_list_of_words_and_label]
# task 3a)
# empty feature set:
# document_features_baseline returns an empty dictionary
# (imput arguments not needed)
documents_as_empty_features_and_label = \
   [(document_features_baseline(list_of_words, word_features_list),
     doc_label)
    for (list_of_words, doc_label) in
    documents_as_list_of_words_and_label]
if (debug):print documents_as_features_and_label
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "5~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "training classifier ...\n"
# ------------------
# cut the set into 90% training and 10% development test set
cut = (int)(nr_of_docs_loaded*0.9)
print "divide nr of documents loaded into\ntraining examples: " \
     "%s \ntest examples: %s" \
     %(cut, nr_of_docs_loaded-cut)
training_set = documents_as_features_and_label[:cut]
development_test_set = documents_as_features_and_label[cut:]
# NBC training
classifier = nltk.NaiveBayesClassifier.train(training_set)
# task 3a)
# train NBC with empty feature set
classifier_baseline = nltk.NaiveBayesClassifier.train\
   (documents_as_empty_features_and_label)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "6~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
# check accuracy
# since the development_test_set and the generation of
# the features list in step 3 used the same documents
# the accuracy is biased (higher 93%)
print "accuracy with development_test_set:"
print nltk.classify.accuracy(
   classifier, development_test_set)
# since the documents in the test set are completely unknown
# to the NBC the accuracy is unbiased (lower 85%)
print "accuracy with test_set:"
print nltk.classify.accuracy(
   classifier, test_set_as_features_and_label)
# task 3a)
# accuracy of NBC trained with empty feature set
# (always says spam if more spam documents are used to train
# the NBC and vice versa.)
print "accuracy with empty feature set (baseline):"
print nltk.classify.accuracy(
   classifier_baseline, test_set_as_features_and_label)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "7~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
# check most informative features
# result of a principal component analysis
print "True means Ham\nFalse means Spam"
classifier.show_most_informative_features(5)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "8~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
# Application of the trained NBC
# on the iteratively used set
# (becomes biased after each iteration of improving feature
# extraction function)
print "Error-list to help specify new features \n" \
     "(dev_test-section in the nltkbook):\n"

# contains the wrong classifications
errors = []
# iterate through the set by extracting ((mail,label), id)
for ((mail,label), id) in \
      errorsampling_set_as_list_of_words_and_label_with_id:
   # appy the trained NCB on each document in the set
   guess = classifier.classify(
      document_features(mail, word_features_list))
   # checks whether the guess of NBC was correct
   if guess != label:
      #renaming of the output (0/1 -> spam/ham)
      if guess == 0:
         guess = "spam"
      else:
         guess = "ham"
```

```
        if label == 0:
           label = "spam"
        else:
           label = "ham"
        errors.append( (label, guess, id) )
   # output of all wrong classifications
   for (label, guess, id) in sorted(errors):
      print "correct=%-5s guess=%-5s mail_ID=%s" % (label, guess, id)
   # output of summary informations
   print "\nnumber of mails: %i \nnumber of errors: %i" \
        "\nerror-rate: %f"  % \
   (len(errorsampling_set_as_list_of_words_and_label_with_id),
    len(errors),
    (len(errors)/len(errorsampling_set_as_list_of_words_and_label_with_id)))

   print "\nAntwort zur Frage 3a):\nGegeben man nimmt die Baseline-funktion," \
   "welche keine features deklariert, waehlt der Klassifikator den im Trainingsset am
haeufigsten vorkommende Tag " \
   "fuer alle Mails.\n"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "End of Script"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
def most_informative_n_grams_dict(corpus_words, ham_words, spam_words,
                                    parameter):
   """
   returns a dict of most informative n-grams
   parameter servers as a frequency cap
   """
   # dict to store and return n-grams
   ngram_dict={}
   #calculate n-grams
   #unigram_dict = nltk.FreqDist(w.lower() for w in corpus_words)
   print "building n-grams...",
   corpus_unigram_dict = Counter(corpus_words)
   corpus_bigram_dict = build_n_gram_dict(corpus_words, 2)
   corpus_trigram_dict = build_n_gram_dict(corpus_words, 3)
   print "40% done...",
   ham_unigram_dict = Counter(ham_words)
   ham_bigram_dict = build_n_gram_dict(ham_words, 2)
   ham_trigram_dict = build_n_gram_dict(ham_words, 3)
   print "70% done...",
   spam_unigram_dict = Counter(spam_words)
   spam_bigram_dict = build_n_gram_dict(spam_words, 2)
   spam_trigram_dict = build_n_gram_dict(spam_words, 3)
   print "99% done..."
   # only those n-grams with a certain DELTA frequency
   # unigrams
   for key in corpus_unigram_dict:
      # remove very frequent words
      corpus_cnt = corpus_unigram_dict.get(key,0)
      if corpus_cnt < parameter:
         ham_cnt = ham_unigram_dict.get(key, 0)
         spam_cnt = spam_unigram_dict.get(key, 0)
         # delta should be resonably high
         DELTA = abs(ham_cnt - spam_cnt)
         if DELTA > parameter*0.3:
            ngram_dict.update({key: DELTA})
   # bigrams
   for key in corpus_bigram_dict:
      # remove very frequent words
      corpus_cnt = corpus_bigram_dict.get(key,0)
      if corpus_cnt < parameter:
         ham_cnt = ham_bigram_dict.get(key, 0)
         spam_cnt = spam_bigram_dict.get(key, 0)
         # delta should be resonably high
         DELTA = abs(ham_cnt - spam_cnt)
         if DELTA > parameter*0.2:
            ngram_dict.update({key:DELTA})
   # trigrams
   for key in corpus_trigram_dict:
      # remove very frequent words
      corpus_cnt = corpus_trigram_dict.get(key,0)
      if corpus_cnt < parameter:
         ham_cnt = ham_trigram_dict.get(key, 0)
         spam_cnt = spam_trigram_dict.get(key, 0)
         # delta should be resonably high
         DELTA = abs(ham_cnt - spam_cnt)
         if DELTA > parameter*0.1:
            ngram_dict.update({key:DELTA})
   # reverse sorted list of n-gram tuples (n-gram, DELTA)
   sorted_list_of_ngram_tuples = []
   for key in sorted(ngram_dict.keys(), key=ngram_dict.get, reverse=True):
      ngram_tuple=(key, ngram_dict[key])
      sorted_list_of_ngram_tuples.append(ngram_tuple)
   # reverse sorted list of n-grams
   sorted_ngram_list = sorted(ngram_dict.keys(), key=ngram_dict.get, reverse=True)
   print "... done"
   # return
   return sorted_ngram_list, sorted_list_of_ngram_tuples, ngram_dict
def build_n_gram_dict(corpus_words, n):
   """
   returns a dict of n-grams
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    """
    # nr of tokens in corpus
    length=len(corpus_words)
    # data structure
    dict_of_ngrams={}
    # iterate through all tokens in corpus list
    for i in range(0,length-n):
        # extract the n-gram as list from the token list
        n_gram_list = corpus_words[i:i+n] #excl. i+n
        # convert the k-gram as list into a string
        n_gram_string =' '.join(n_gram_list)
        n_gram_string = n_gram_string.lower()
        # update the dict of n-grams
        # the dictionary entry with key = next word
        # gets incremented (n-gram counter)
        cnt_ngram = dict_of_ngrams.get(n_gram_string, 0)
        cnt_ngram = cnt_ngram + 1
        dict_of_ngrams.update({n_gram_string:cnt_ngram})
    #return dictionary
    return dict_of_ngrams
def build_document_tuples(corpus, fileid):
    """
    input: FileID
    output:    builds a list of tokenized emails in a tuple together with the spam/ham
label
    ham: True
    spam: False
    """
    #tokenize document with given fileid
    list_of_words = corpus.words(fileid)
    #check whether the fileid contains the text <ham.txt>
    if ".ham.txt" in fileid:
        return (list_of_words, True)
    else:
        return (list_of_words, False)
#feature extractor functions
def document_features(list_of_words, word_features_list):
     """
     A feature extractor for document classification, whose features indicate whether or
not
     individual words are present in a given document.
     """
     #convert list of words to set
     set_of_words = set(list_of_words)
    #calculate k-gram
     #dictionary to store the document features
     dict_of_features = {}
    #set flag True/False depending on whether a word
    #in the word_features_list is present in the document
     for word in word_features_list:
         #features[key]=value, key=word, value=True/False
         dict_of_features['contains(%s)' % word] = (word in set_of_words)
     #returns a dictionary
     return dict_of_features

# empty dict to comply with the requirements of the classifier
def document_features_baseline(list_of_words, word_features_list):
    return {}

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import collections
import operator
import StringIO
import sys
from collections import defaultdict
from operator import itemgetter
class IntDict(dict):
    def __missing__(self, key):
        self[key] = int()
        return self[key]
class BigramTagger(object):
    def __init__(self):
        self.freqs = defaultdict(IntDict)
    def train(self, training_data):
        for sent in training_data:
            last_tag = None
            for token, tag in sent:
                context = (last_tag, token)
                self.freqs[context][tag] += 1
                last_tag = tag
    def tag(self, word, last_tag):
        context = (last_tag, word)
        try:
            best_tag = max(self.freqs[context].items(), key=itemgetter(1))[0]
            return (word, best_tag)
        except Exception:
```

U4.1
1 POS-Tagging
In dieser Aufgabe geht es darum, selbst einen POS-Tagger zu implementieren. Der Tagger soll ein statistischer Bigramm-Tagger sein. Als Trainingskorpus kannst du getaggte Sätze aus einer Kategorie deiner Wahl des Brown Corpus verwenden.
a) Teile dein Korpus in zwei Teile zu je 90 und 10%. Der grössere Teil soll dein Trainingskorpus sein, der kleinere dein Testkorpus.
b) Implementiere deinen Tagger. Du musst deinen Tagger selber schreiben; es ist nicht erlaubt, die fertigen Tagger von NLTK zu verwenden. Du darfst externen Code benutzen, aber du solltest zeigen, dass du verstanden hast, wie der Tagger genau funktioniert. Wenn der Tagger ein Bigramm noch nicht gesehen hat, sollte er es als unbekannt taggen. Teste deinen Tagger auf dem Satz “This is a sentence that we want to tag.” Welches Problem tritt auf?
c) Schreibe eine Funktion evaluate(tagger, tagged_sents), die die Genauigkeit eines Taggers berechnet. Evaluiere deinen Tagger auf deinem Testkorpus. Wie genau ist er?
d) Erstelle eine Konfusionsmatrix für deinen Tagger. Bei welchen Tags macht er besonders viele Fehler?

```python
                return (word, None)
    def tag_sentence(self, sent):
        tagged = []
        last_tag = None
        for word in sent:
            tagged_word = self.tag(word, last_tag)
            tagged.append(tagged_word)
            last_tag = tagged_word[1]
        return tagged
def evaluate(tagger, corpus):
    total = 0
    correct = 0
    count = 0
    for sent in corpus:
        words = [t[0] for t in sent]
        tagged_words = tagger.tag_sentence(words)
        count += 1
        for my_tag, gold_tag in zip([t[1] for t in sent],
                [t[1] for t in tagged_words]):
            total += 1
            if my_tag == gold_tag:
                correct += 1
    return float(correct) / float(total)
def generate_confusion_matrix(tagger, corpus, tagset):
    cmatrix = defaultdict(IntDict)
    cmtable = StringIO.StringIO()
    tag_rows = u''.join([u'{:>8}' for t in tagset])
    for sent in corpus:
        words = [t[0] for t in sent]
        tagged_words = tagger.tag_sentence(words)
        for my_tag, gold_tag in zip([t[1] for t in sent],
                [t[1] for t in tagged_words]):
            cmatrix[gold_tag][my_tag] += 1
    cmtable.write(u'        ')
    cmtable.write(tag_rows.format(*tagset))
    cmtable.write(u'\n')
    for tag in tagset:
        cmtable.write(u'{:>8}'.format(tag))
        cmtable.write(tag_rows.format(*[cmatrix[tag][tag2] for tag2 in tagset]))
        cmtable.write(u'\n')
    cmt_str = cmtable.getvalue()
    cmtable.close()
    return cmt_str


#!/usr/bin/env python
# -*- coding: utf-8 -*-
import aufgabe1
import collections
import operator
import StringIO
import sys
from aufgabe1 import IntDict
from collections import defaultdict
from operator import itemgetter
class DefaultTagger(object):
    def __init__(self, dtag):
        self._tag = dtag
    def tag(self, word):
        return (word, self._tag)
class UnigramTagger(object):
    def __init__(self, backoff_tagger=None):
        self.freqs = defaultdict(IntDict)
        self.backoff = backoff_tagger
    def train(self, training_data):
        for sent in training_data:
            for word,tag in sent:
                self.freqs[word][tag] += 1
    def tag(self, word):
        try:
            best_tag = max(self.freqs[word].items(), key=itemgetter(1))[0]
            return (word, best_tag)
        except Exception:
            if self.backoff is not None:
                return self.backoff.tag(word)
            else:
                return (word, None)
    def tag_sentence(self, sentence):
        return list(map(self.tag, sentence))
class BigramTagger(object):
    def __init__(self, backoff_tagger=None):
        self.freqs = defaultdict(IntDict)
        self.backoff = backoff_tagger
    def train(self, training_data):
        for sent in training_data:
            last_tag = None
            for token, tag in sent:
                context = (last_tag, token)
                self.freqs[context][tag] += 1
                last_tag = tag
    def tag(self, word, last_tag):
        context = (last_tag, word)
```

U4.2
2 Backoffs
In dieser Aufgabe sollst du den Tagger aus der letzten Aufgabe erweitern, sodass er unbekannte
Kontexte mit Backoffs auflöst.
Schreibe dazu zwei weitere Tagger: einen Unigramm-Tagger und einen Default-Tagger. Ändere dei-
nen Tagger so, dass er Wörter in unbekannten Kontexten mit dem Unigramm-Tagger taggt. Der
Unigramm-Tagger sollte wiederum bei unbekannten Wörtern auf den Default-Tagger zurückgreifen.
a) Trainiere und evaluiere den neuen Tagger auf den gleichen Korpora wie in der letzten Aufgabe.
Gibt es einen Unterschied in der Genauigkeit?
b) Evaluiere deine Tagger auf einer anderen Kategorie des Brown-Korpus. Wie verändert sich ihre
Genauigkeit? Wie sieht es aus, wenn du sie über dem conll2000-Korpus evaluierst?

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
        try:
            best_tag = max(self.freqs[context].items(), key=itemgetter(1))[0]
            return (word, best_tag)
        except Exception:
            if self.backoff is not None:
                return self.backoff.tag(word)
            else:
                return (word, None)
    def tag_sentence(self, sent):
        tagged = []
        last_tag = None
        for word in sent:
            tagged_word = self.tag(word, last_tag)
            tagged.append(tagged_word)
            last_tag = tagged_word[1]
        return tagged


#!/usr/bin/env python
# -*- coding:utf-8 -*-
import aufgabe1 as a1
import aufgabe2 as a2
import nltk
from nltk.corpus import brown
from nltk.corpus import conll2000
from nltk.tag import mapping
def test_tagging(tagger):

    test_bg = tagger.tag_sentence([u'this', u'is', u'a', u'sentence',
        u'that', u'we', u'want', u'to', u'tag', u'.'])
    print u' '.join((u'{}/{}'.format(word, tag) for word, tag in test_bg))
    print u'\n'
def print_accuracy(tagger, test):
    score = a1.evaluate(tagger, test)
    print u'Tagger accuracy: {}'.format(score)
    print u'\n'
def print_confusion_matrix(tagger, test, tagset):
    cmatr = a1.generate_confusion_matrix(tagger, test, tagset)
    print u'Tagger Confusion Matrix (Rows: Target, Columns: Actual)'
    print cmatr
    print u'\n'
def main():
    # 1. a)
    bts = brown.tagged_sents(categories=u'news', tagset=u'universal')
    brown_size = int(len(bts) * 0.9)
    brown_training = bts[:brown_size]
    brown_test = bts[brown_size:]
    tagset = list(mapping._UNIVERSAL_TAGS)
    simple_tagger = a1.BigramTagger()
    simple_tagger.train(brown_training)
    #1. b)
    test_tagging(simple_tagger)
    #1. c)
    print u'Simple bigram tagger'
    print_accuracy(simple_tagger, brown_test)
    #1. d)
    print_confusion_matrix(simple_tagger, brown_test, tagset)
    #2. a)
    default_tagger = a2.DefaultTagger(u'NN')
    unigram_tagger = a2.UnigramTagger(backoff_tagger=default_tagger)
    unigram_tagger.train(brown_training)
    bigram_tagger = a2.BigramTagger(backoff_tagger=unigram_tagger)
    bigram_tagger.train(brown_training)
    print u'Bigram tagger with backoffs'
    print_accuracy(bigram_tagger, brown_test)
    #2. b)
    other_cat = brown.tagged_sents(categories='romance', tagset='universal')
    print u'Simple bigram tagger, other genre'
    print_accuracy(simple_tagger, other_cat)
    print u'Backoff tagger, other genre'
    print_accuracy(bigram_tagger, other_cat)
    conll_sents = conll2000.tagged_sents(tagset=u'universal')
    print u'Simple bigram tagger, other corpus'
    print_accuracy(simple_tagger, conll_sents)
    print u'Backoff tagger, other corpus'
    print_accuracy(bigram_tagger, conll_sents)
if __name__ == '__main__':
    main()
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 04 - Aufgabe 1, FS16
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                               'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Hinweis für Cazim, Irene, Raffael:
# Wir geben beide Versionen ab. Es reicht eine anzuschauen.
# Linus Version benützt defaultdict(). Diese Datenstruktur kann
# wie eine Matrix in Matlab/R verwendet werden. Entsprechend sind
```

U4.1/ U4.2
Roland
Linus

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
# seine Funktionsdefinitionen einiges eleganter codiert.
# Rolands Version ist sehr ausführlich dokumentiert. Alle Fragen
# der Aufgabestellung im Main beantwortet und auf Console geprintet.
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Reflexion/Feedback
# a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen
# zusammen.
# Ich habe gelernt einen Trainingsalgorithmus für einen Classifier
# (Tagger) selber zu programmieren, damit einen Text zu taggen
# und die Ergebnisse mit Accuracy und Confusionsmatrix zu
# evaluieren. Zweite Erkenntnis: defaultdict() verwenden.
# b) Wie viel Zeit hast du in diese Übungen investiert?
# Roland 15 Stunden
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
from __future__ import division
#used corpora as data input
from nltk.corpus import brown
#used functionality for ML
import nltk
import random
import operator
import math
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Main
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# set the flags for debug info
debug_train = 0
debug_test = 0
debug_eval = 0
def main():
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "1~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "load corpus ... \n split it into" \
          " 90 % training set & 10 % test set\n"
    # 1a) Teile dein Korpus in zwei Teile zu je 90 und 10%
    # ---------------------------------------------------------
    # load list of sents with tuples of word and tag
    list_of_tagged_sents = brown.tagged_sents(
        categories = "news", tagset = "universal")
    print "list of tagged sents: \n" \
          "(only a subset printed)\n", list_of_tagged_sents[:2]
    # cast ConcatenatedCorpusView object into a list
    list_of_tagged_sents = list(list_of_tagged_sents)
    # load list of tokens (not needed)
    list_of_tokens = brown.words(
        categories = "news")
    print "\nlist of tagged tokens: \n", \
        "(not nedded & only a subset printed)\n", list_of_tokens[:5]
    # info about loaded corpus size
    nr_of_sents_loaded = len(list_of_tagged_sents)
    print "\nnr of sents loaded: ", nr_of_sents_loaded
    nr_of_tokens_loaded = sum(
        [len(x) for x in list_of_tagged_sents])
    print "nr of tokens loaded (from sents): ", nr_of_tokens_loaded
    nr_of_tokens_loaded = len(list_of_tokens)
    print "nr of tokens loaded (from words): ", nr_of_tokens_loaded
    # partition the loaded list of tagged sents
    cut = (int)(nr_of_sents_loaded * 0.9)
    training_set = list_of_tagged_sents[:cut]
    test_set = list_of_tagged_sents[cut:]
    # random shuffle the training_set (not needed)
    # (only works on list not on ConcatenatedCorpusView)
    random.shuffle(training_set)
    print "split loaded list of tagged sents into: \n" \
          " nr of training sents: %s \n nr or test sents: %s" \
          % (cut, nr_of_sents_loaded - cut)
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "2~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "train classifier (tagger) with training set\n"
    # 1b) Implementiere deinen Tagger.
    # Du darfst externen Code benutzen, aber du solltest zeigen,
    # dass du verstanden hast, wie der Tagger genau funktioniert.
    # Wenn der Tagger ein Bigramm noch nicht gesehen hat,
    # sollte er es als unbekannt taggen.
    #
    # Datenstruktur 1: bigrams_with_all_tags_and_cnt
    # {(tag_N-1, token_N):{tag_N_1:cnt_1, tag_N_2:cnt_2, ...},
    #  (tag_N, token_N+1):{tag_N+1_1:cnt_1, tag_N+1_2:cnt_2, ...},
    #  ...}
    # Datenstruktur 2: bigrams_with_most_likely_tag
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # ---------------------------------------------------------
    bigrams_with_all_tags_and_cnt, bigrams_with_most_likely_tag = \
        train_bigram_tagger(training_set)
    print "tagger_bigram ((tag_N-1, token_N),(most likely tag N)):\n" \
          "(only a subset printed)\n", \
        bigrams_with_most_likely_tag.items()[:20]
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "3~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "pos-tag the test set with the bigram tagger\n"
    # 1b) Teste deinen Tagger auf
```

```
    # dem Satz “This is a sentence that we want to tag.”
    # Welches Problem tritt auf?
    # Anwort 1b): ..... (siehe console output)
    #
    # data structure: test_set_untagged
    # list of sents containing list of tokens
    # [[token_0, token_1, ...]_sent1,
    #  [token_0, token_1, ...]_sent2, ...]
    #
    # data structure: bigrams_with_most_likey_tag:
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    #
    # data structure: test_set_tagged
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  ...}
    # ---------------------------------------------------
    # untag the test set
    test_set_untagged = untag_sents_tagged(test_set)
    # tag the test set with tagger
    test_set_tagged = bigram_tagger(
        test_set_untagged, bigrams_with_most_likely_tag)
    print "test set untagged:\n" \
          "(only a subset printed)\n", test_set_untagged[:2]
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "test set tagged with bigram tagger:\n" \
          "(only a subset printed)\n", test_set_tagged[:2]
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "test set tagged with brown.tagged_sents:\n" \
          "(only a subset printed)\n", test_set[:2]
    # tag the sentence "This is a sentence that we want to tag."
    test_sentence = [[(u'This',u'DET'), (u'is',u'VERB'),
        (u'a',u'DET'), (u'sentence',u'NOUN'),
        (u'that', u'ADP'), (u'we', u'PRON'),
        (u'want', u'VERB'), (u'to',u'PRT'),
        (u'tag',u'NOUN'), (u'.',u'.')]]
    test_sentence_untagged = [[u'This', u'is', u'a', u'sentence',
                u'that', u'we', u'want', u'to', u'tag', u'.']]
    test_sentence_tagged = bigram_tagger(
        test_sentence_untagged, bigrams_with_most_likely_tag)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "test sentence untagged:\n " \
          "(only a subset printed)\n",\
                test_sentence_untagged[:2]
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "test sentence tagged with bigram tagger:\n " \
          "(only a subset printed)\n",\
                test_sentence_tagged[:2]
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "Antwort 1b)\n Problem:\n The tagger has no key containing " \
          "UNKNOWN as pos-tag. Therefore the first time " \
          "a specific bigram is not found in the tagger " \
          "dictionary, the pos-tag UNKNOWN propagates " \
          "through the sentence."
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "4~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "evaluate test set (pre-labels vs. own-labels)\n"
    # 1c) Schreibe eine Funktion evaluate(tagger, tagged_sents),
    # die die Genauigkeit eines Taggers berechnet.
    # Evaluiere deinen Tagger auf deinem Testkorpus.
    # Wie genau ist er?
    # Anwort 1c): ..... (siehe console output)
    # ---------------------------------------------------
    # evaluate accuracy of tagger
    accuracy = evaluate(
        test_set, bigrams_with_most_likely_tag)
    print "Antwort 1c)\n accuracy of tagger: ", accuracy
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "5~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "print the confusion matrices\n " \
          " on the brown corpus 10% test sentences\n " \
          " on the 1 test sentence"
    # 1d) Erstelle eine Konfusionsmatrix für deinen Tagger.
    # Bei welchen Tags macht er besonders viele Fehler?
    # Anwort 1d): ..... (siehe console output)
    # ---------------------------------------------------
    confusion_matrix(test_set, test_set_tagged)
    confusion_matrix(test_sentence, test_sentence_tagged)
    print "Antwort 1d)\n Interpretation of the confusion matrix:\n" \
          " The optimal solution has a diagonal matrix." \
          " Not diagonal entries show the number of wrongly" \
          " tagged tokens.\"" \
          " The ratio unknown:known bigrams is about 6:1" \
          " for most english tokens, 9:1 for nouns, and inf:1 " \
          " for foreign tokens with pos-tax X, since none has been" \
          " tagged correctly."
    # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "6~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "bigram-unigram-default-backoff\n"
    # 2) Schreibe zwei weitere Tagger:
```

```
# einen Unigramm-Tagger und einen Default-Tagger.
# Ändere deinen Tagger so, dass er Wörter in unbekannten
# Kontexten mit dem Unigramm-Tagger taggt.
# Der Unigramm-Tagger sollte wiederum bei unbekannten Wörtern
# auf den Default-Tagger zurückgreifen.
# --------------------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "7~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "train classifier (tagger) with training set\n"
# 2a) Trainiere den neuen Tagger auf den
# gleichen Korpora wie in der letzten Aufgabe.
# --------------------------------------------------
# train unigram tagger
unigrams_with_all_tags_and_cnt, unigrams_with_most_likely_tag = \
    train_unigram_tagger(training_set)
# train default tagger
default_with_most_likely_tag = \
    train_default_tagger(training_set)
print "tagger_unigram ((token_N),(most likely tag N)):\n " \
      "(only a subset printed)\n",\
        unigrams_with_most_likely_tag.items()[:20]
print "\ntagger_default ((most likely tag N)):\n", \
        default_with_most_likely_tag
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "8~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "pos-tag the test set with the bigram tagger\n"
# 2) Teste deinen Tagger auf
# dem Satz “This is a sentence that we want to tag.”
# --------------------------------------------------
# tag the test set with tagger
test_set_tagged = bigram_tagger_with_backoff(
    test_set_untagged, bigrams_with_most_likely_tag,
    unigrams_with_most_likely_tag, default_with_most_likely_tag)
print "test set untagged:\n" \
      "(only a subset printed)\n", test_set_untagged[:2]
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "test set tagged with bigram tagger with backoff:\n" \
      "(only a subset printed)\n", test_set_tagged[:2]
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "test set tagged with brown.tagged_sents:\n" \
      "(only a subset printed)\n", test_set[:2]
# tag the sentence "This is a sentence that we want to tag."
test_sentence_untagged = [[u'This', u'is', u'a', u'sentence',
                           u'that', u'we', u'want', u'to',
                           u'tag', u'.']]
test_sentence_tagged = bigram_tagger_with_backoff(
    test_sentence_untagged, bigrams_with_most_likely_tag,
    unigrams_with_most_likely_tag,
    default_with_most_likely_tag)
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "test sentence untagged:\n" \
      "(only a subset printed)\n",\
        test_sentence_untagged[:2]
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "test sentence tagged with bigram tagger:\n" \
      "(only a subset printed)\n",\
        test_sentence_tagged[:2]
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "Problem from 1b):\n -> Solved.\n" \
      "The backoff functionality, checks the bigrams first,\n" \
      "if unknown, it checks the unigrams,\n" \
      "if unknown, it takes the pre calculated default post tag"
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "9~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "evaluate test set (pre-labels vs. own-labels)\n"
# a) Evaluiere den neuen Tagger auf den
# gleichen Korpora wie in der letzten Aufgabe.
# Gibt es einen Unterschied in der Genauigkeit?
# Anwort 2a): ..... (siehe console output)
# --------------------------------------------------
# evaluate accuracy of tagger
accuracy = evaluate_with_backoff(
    test_set, bigrams_with_most_likely_tag,
    unigrams_with_most_likely_tag,
    default_with_most_likely_tag)
print "Antwort 2a) accuracy of tagger: ", accuracy
print " Without vs. with backoff functionality:\n" \
      " From under 20% to over 90% accuracy."
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
print "10~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
print "confusion matrix\n"
# 2a) Erstelle eine Konfusionsmatrix für deinen Tagger.
# Bei welchen Tags macht er besonders viele Fehler?
# Anwort 2a): ..... (siehe console output)
# --------------------------------------------------
#print confusion matrix
confusion_matrix(test_set, test_set_tagged)
# print confusion matrix
confusion_matrix(test_sentence, test_sentence_tagged)
print "Antwort 2a)\n Interpretation of the confusion matrix:\n" \
      " The optimal solution has a diagonal matrix." \
      " Not diagonal entries show the number of wrongly" \
```

```
           " tagged tokens.\"" \
           " ADJ, VERB, NUM, make use of the default tagger in " \
           " about 0 to 30 % of cases. X in nearly all of the cases." \
           " ADV-ADJ, ADV-ADP, ADP-PRT show a high error ratio."
  # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    print "11~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "evaluation of bigram tagger with backoff on other " \
          "corpora and categories\n"
    # 2b) Evaluiere deine Tagger auf einer anderen Kategorie des
    # Brown-Korpus.
    #
    # Wie verändert sich ihre Genauigkeit?
    # Wie sieht es aus, wenn du sie über dem conll2000-Korpus evaluierst?"
    # Anwort 2b): ..... (siehe console output)
    # --------------------------------------------------
    #Brown corpus categories
    print "brown corpus categories\n", brown.categories()
    print "\n1~~~load tagged sents of lore category"
    # load list of sents with tuples of word and tag
    list_of_tagged_sents = brown.tagged_sents(
        categories="lore", tagset="universal")
    # cast ConcatenatedCorpusView object into a list
    list_of_tagged_sents = list(list_of_tagged_sents)
    print "3~~~untag tagged sents"
    # untag the test set
    test_set = list_of_tagged_sents
    test_set_untagged = untag_sents_tagged(test_set)
    print "8~~~load tagged sents"
    # tag the test set with tagger
    test_set_tagged = bigram_tagger_with_backoff(
        test_set_untagged, bigrams_with_most_likely_tag,
        unigrams_with_most_likely_tag,
        default_with_most_likely_tag)
    print "10~~~print confusion matrix"
    # print confusion matrix
    confusion_matrix(test_set, test_set_tagged)
    print "9~~~evaluate accuracy of tagger trained with" \
          " category news and tagged with category lore"
    # evaluate accuracy of tagger
    accuracy = evaluate_with_backoff(
        test_set, bigrams_with_most_likely_tag,
        unigrams_with_most_likely_tag,
        default_with_most_likely_tag)
    print "\nAntwort 2b)\n accuracy of tagger: ", accuracy
    print "The question can be answered hypothetically without" \
        " testing the hypothesis for the conll2000-Korpus:\n" \
        " The accuracy decreases for one simple reason." \
        " Machine learning is about generalization from seen" \
        " examples. Unseen dimensions which exist in new text " \
        " categories or corpora in the form of frequent new" \
        " bigrams/unigrams/default are not part of the trained " \
        " classifier (tagger). The worst that can happen, is that" \
        " the default pos-tag is no longer true for the new texts."
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Function definitions
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
def train_bigram_tagger(training_set):
    """
        return a dictionary with:
            key = bigram consisting of  tuple (last tag, this token)
            value = most likely tag
        most likely tag = max ({tag1:cnt1, tag2:cnt2, ...}, cnt_i)
        """
    # Input data structure
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Output data structure 1:
    # {(tag_N-1, token_N):{tag_N_1:cnt_1, tag_N_2:cnt_2, ...},
    #  (tag_N, token_N+1):{tag_N+1_1:cnt_1, tag_N+1_2:cnt_2, ...},
    #  ...}
    # Output data structure 2:
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # Intermediate data structure: dict_new_tag_cnt
    # {tag_N-1:cnt, tag_N:cnt, tag_N+1:cnt, ...}
    # output data structure 1
    dict_last_tag_new_token__dict_new_tag__cnt = {}
    # output data structure 2
    dict_last_tag_new_token__most_likey_new_tag = {}
    # nested dictionary
    dict_new_tag__cnt = {}
    if (debug_train): training_set = training_set[:2]
    # iterate through sentences in training set
    for list_sent in training_set:
        # first tag in sentence
        tag_0 = "NONE"
        #iterate through words in sentence
        for i, tuple_token_tag in enumerate(list_sent):
            # extract last tag i-1
            if not i == 0:
```

```
                tag_i_minus_1 = list_sent[i - 1][1]
            else:
                tag_i_minus_1 = tag_0
            # extract token i
            token_i = tuple_token_tag[0]
            # extract tag i
            tag_i = tuple_token_tag[1]
            # extract counter
            dict_new_tag__cnt = \
                dict_last_tag_new_token__dict_new_tag__cnt.\
                    get((tag_i_minus_1, token_i),{})
            cnt_i = dict_new_tag__cnt.get(tag_i,0)
            # increment counter
            cnt_i = cnt_i+1
            # update nested dictionary
            dict_new_tag__cnt.update(
                {tag_i:cnt_i})
            # update data structure 1
            dict_last_tag_new_token__dict_new_tag__cnt.update(
                {(tag_i_minus_1, token_i):dict_new_tag__cnt})
            if (debug_train): print "%s: (%s,%s), %s "\
                              %(i, tag_i_minus_1, token_i, tag_i)
    # generate output data structure 2
    for last_tag_new_token, dict_new_tag__cnt \
            in dict_last_tag_new_token__dict_new_tag__cnt.items():
        # extract most likely tag
        most_likey_new_tag = max(
            dict_new_tag__cnt.iteritems(),
            key=operator.itemgetter(1))[0]
        # update data structure 2
        dict_last_tag_new_token__most_likey_new_tag.update(
                {last_tag_new_token:most_likey_new_tag})
    if (debug_train):
        print dict_last_tag_new_token__dict_new_tag__cnt, "\n-----\n", \
            dict_last_tag_new_token__most_likey_new_tag
    # return the data structure
    return dict_last_tag_new_token__dict_new_tag__cnt, \
           dict_last_tag_new_token__most_likey_new_tag
def train_unigram_tagger(training_set):
    """
        unigram tagger
        """
    # Input data structure
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Output data structure 1:
    # {(token_N):{tag_N_1:cnt_1, tag_N_2:cnt_2, ...},
    #  (token_N+1):{tag_N+1_1:cnt_1, tag_N+1_2:cnt_2, ...},
    #  ...}
    # Output data structure 2:
    # {(token_N):tag_N_most_likely,
    #  (token_N+1):tag_N_most_likely,
    #  ...}
    # Intermediate data structure: dict_new_tag__cnt
    #{tag_N_1: cnt_1, tag_N_2: cnt_2, ...}
    # output data structure 1
    dict_new_token__dict_new_tag__cnt = {}
    # output data structure 2
    dict_new_token__most_likey_new_tag = {}
    # nested dictionary
    dict_new_tag__cnt = {}
    if (debug_train): training_set = training_set[:2]
    # iterate through sentences in training set
    for list_sent in training_set:
        #iterate through words in sentence
        for i, tuple_token_tag in enumerate(list_sent):
            # extract token i
            token_i = tuple_token_tag[0]
            # extract tag i
            tag_i = tuple_token_tag[1]
            # extract counter
            dict_new_tag__cnt = \
                dict_new_token__dict_new_tag__cnt.\
                    get((token_i),{})
            cnt_i = dict_new_tag__cnt.get(tag_i,0)
            # increment counter
            cnt_i = cnt_i + 1
            # update nested dictionary
            dict_new_tag__cnt.update(
                {tag_i:cnt_i})
            # update data structure 1
            dict_new_token__dict_new_tag__cnt.update(
                {(token_i):dict_new_tag__cnt})
            if (debug_train): print "%s: (%s,%s) "\
                              %(i, token_i, tag_i)
    # generate output data structure 2
    for new_token, dict_new_tag__cnt \
            in dict_new_token__dict_new_tag__cnt.items():
        # extract most likely tag
        most_likey_new_tag = max(
            dict_new_tag__cnt.iteritems(),
```

```
            key=operator.itemgetter(1))[0]
        # update data structure 2
        dict_new_token__most_likey_new_tag.update(
                {new_token:most_likey_new_tag})
    if (debug_train):
        print dict_new_token__dict_new_tag__cnt, "\n-----\n", \
            dict_new_token__most_likey_new_tag, "\n-----\n"
    # return the data structure
    return dict_new_token__dict_new_tag__cnt, \
           dict_new_token__most_likey_new_tag
def train_default_tagger(training_set):
    """
        default tagger
        """
    # Input data structure
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Intermediate data structure: dict_tag_cnt
    # {tag_N-1:cnt, tag_N:cnt, tag_N+1:cnt, ...}
    # intermediate data structure
    dict_tag_cnt = {}
    if (debug_train): training_set = training_set[:2]
    # count true and false tagger tagged tags
    for j, sent_j in enumerate(training_set):
        # iterate through tokens in sentence
        for i, (token_i, tag_i) in enumerate(sent_j):
            # extract and increment counter of tag i
            cnt_i = dict_tag_cnt.get(tag_i, 0)
            cnt_i = cnt_i + 1
            # uptdate dictionary
            dict_tag_cnt.update({tag_i:cnt_i})
    if (debug_train):print dict_tag_cnt
    # determine most frequent tag
    most_likely_tag = max(
        dict_tag_cnt.iteritems(),
        key=operator.itemgetter(1))[0]
    if (debug_train): print most_likely_tag
    #return most frequent tag
    return most_likely_tag
def untag_sents_tagged(test_set):
    """
        returns list of tokens:
        """
    # Input data structure
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Output data structure
    # list of sents containing list of tokens
    # [[token_0, token_1, ...]_sent1,
    #  [token_0, token_1, ...]_sent2, ...]
    # output list
    list_of_untagged_tokens = []
    if (debug_test): test_set = test_set[:2]
    # extract the token from the input data structure
    for j, sent_j in enumerate(test_set):
        # append a new empty list to the output list
        list_of_untagged_tokens.append([])
        # iterate through tokens in sentence
        for token_i, tag_i in sent_j:
            if (debug_test): print "%s: (%s,%s)" \
                                % (j, token_i, tag_i)
            # apppend new token to output list
            list_of_untagged_tokens[j].append(token_i)
    if (debug_test): print list_of_untagged_tokens
    # return list of untagged tokens
    return list_of_untagged_tokens
def bigram_tagger(
        list_of_sents_with_tokens, bigrams_with_most_likey_tag):
    """
        tags the input list of sents containing list of tokens
        with bigram data structure:
        """
    # Input data structure list_of_sents_with_tokens
    # list of sents containing list of tokens
    # [[token_0, token_1, ...]_sent1,
    #  [token_0, token_1, ...]_sent2, ...]
    # Input data structure bigrams_with_most_likey_tag:
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # Output data structure
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  ...}
    # output list
    list_tokens_tags=[]
    # first tag in sentence
    tag_0 = "NONE"
    if (debug_test):
        list_of_sents_with_tokens = list_of_sents_with_tokens[:2]
```

```
    # extract the token from the input data structure
    for j, sents_j in enumerate(list_of_sents_with_tokens):
        # append a new empty list to the output list
        list_tokens_tags.append([])
        # iterate through tokens in sentence
        for i, token_i in enumerate(sents_j):
            # last tag i-1
            if i == 0:
                tag_i_minus_1 = tag_0
            else:
                tag_i_minus_1 = tag_i
            # build bigram
            bigram_i = (tag_i_minus_1, token_i)
            # extract new tag or
            # return unknown if bigram unknown
            tag_i = bigrams_with_most_likey_tag.get(
                bigram_i, "UNKNOWN")
            # update output data structure
            list_tokens_tags[j].append(
                (token_i, tag_i))
    if (debug_test): print list_tokens_tags
    # return list of tagged tokens
    return list_tokens_tags
def bigram_tagger_with_backoff(list_of_sents_with_tokens,
            bigrams_with_most_likely_tag,
            unigrams_with_most_likely_tag,
            default_with_most_likely_tag):
    """
        tags the input list of sents containing list of tokens
        with a backoff algorithm staring with a bigram data structure:
        """
    # Input data structure: list_of_sents_with_tokens
    # list of sents containing list of tokens
    # [[token_0, token_1, ...]_sent1,
    #  [token_0, token_1, ...]_sent2, ...]
    # Input data structure: bigrams_with_most_likey_tag:
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # Input data structure: unigrams_with_most_likey_tag
    # {(token_N):tag_N_most_likely,
    #  (token_N+1):tag_N_most_likely,
    #  ...}
    # Input data structure:default_with_most_likey_tag
    # (tag_most_likely)
    # Output data structure:
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    # [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  ...}
    # output list
    list_tokens_tags = []
    # first tag in sentence
    tag_0 = "NONE"
    if (debug_test):
        list_of_sents_with_tokens = list_of_sents_with_tokens[:2]
    # extract the token from the input data structure
    for j, sents_j in enumerate(list_of_sents_with_tokens):
        # append a new empty list to the output list
        list_tokens_tags.append([])
        # iterate through tokens in sentence
        for i, token_i in enumerate(sents_j):
            # last tag i-1
            if i == 0:
                tag_i_minus_1 = tag_0
            else:
                tag_i_minus_1 = tag_i
            # build bigram
            bigram_i = (tag_i_minus_1, token_i)
            # extract new tag or
            # return unknown if bigram unknown
            tag_i = bigrams_with_most_likely_tag.get(
                bigram_i, "UNKNOWN")
            # backoff to unigram
            # extract new tag or
            # return unknown if unigram unknown
            if tag_i == "UNKNOWN":
                tag_i = unigrams_with_most_likely_tag.get(
                    token_i, "UNKNOWN")
            # backoff to default
            if tag_i == "UNKNOWN":
                tag_i = default_with_most_likely_tag
            # update output data structure
            list_tokens_tags[j].append(
                (token_i, tag_i))
    if (debug_test): print list_tokens_tags
    # return list of tagged tokens
    return list_tokens_tags
def evaluate(pretagged_sents, bigrams_with_most_likey_tag):
    """
        evaluates the tagger
        returns proportion of true tagger tagged tags
        """
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    # Input data structure: pretagged_sents
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Input data structure: bigrams_with_most_likey_tag
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # Intermediate data structure: sents_tagger_tagged
    # [[(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  ...]
    # untag the test set
    sents_untagged = untag_sents_tagged(pretagged_sents)
    # tag the test set with tagger
    sents_tagger_tagged = bigram_tagger(
        sents_untagged, bigrams_with_most_likey_tag)
    # counters
    cnt_true_tags = 0
    cnt_false_tags = 0
    cnt_tokens = 0
    if (debug_eval):pretagged_sents = pretagged_sents[:2]
    # count true and false tagger tagged tags
    for j, sent_j in enumerate(pretagged_sents):
        # iterate through tokens in sentence
        for i, (token_i, pretagged_tag_i) in enumerate(sent_j):
            # increment token counter
            cnt_tokens = cnt_tokens + 1
            #extract tagger_tagged_tag_i of tagger tagged tags
            tagger_token_i = sents_tagger_tagged[j][i][0]
            tagger_tagged_tag_i = sents_tagger_tagged[j][i][1]
            # compare tags and increment tags counter
            if pretagged_tag_i == tagger_tagged_tag_i:
                cnt_true_tags = cnt_true_tags + 1
            else:
                cnt_false_tags = cnt_false_tags + 1
            if (debug_eval):
                print "%s: (%s,%s), (%s,%s)" % (j,
                    token_i, pretagged_tag_i,
                    tagger_token_i, tagger_tagged_tag_i)
    # calculate accuracy
    proportion_of_correctly_tagged_tags = (
        cnt_true_tags/cnt_tokens)
    # return accuracy
    return proportion_of_correctly_tagged_tags
def evaluate_with_backoff(pretagged_sents,
                          bigrams_with_most_likey_tag,
                          unigram_with_most_likey_tag,
                          default_with_most_likey_tag):
    """
        evaluates the tagger
        returns proportion of true tagger tagged tags
        """
    # Input data structure: pretagged_sents
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Input data structure: bigrams_with_most_likey_tag:
    # {(tag_N-1, token_N):tag_N_most_likely,
    #  (tag_N, token_N+1):tag_N_most_likely,
    #  ...}
    # Input data structure: unigrams_with_most_likey_tag
    # {(token_N):tag_N_most_likely,
    #  (token_N+1):tag_N_most_likely,
    #  ...}
    # Input data structure:default_with_most_likey_tag
    # (tag_most_likely)
    # Intermediate data structure: sents_tagger_tagged
    # [[(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  [(token_N, tag_N_most_likely),(token_N+1, tag_N+1_most_likely),...],
    #  ...]
    # untag the test set
    sents_untagged = untag_sents_tagged(pretagged_sents)
    # tag the test set with tagger
    sents_tagger_tagged = bigram_tagger_with_backoff(
        sents_untagged, bigrams_with_most_likey_tag,
        unigram_with_most_likey_tag, default_with_most_likey_tag)
    # counters
    cnt_true_tags = 0
    cnt_false_tags = 0
    cnt_tokens = 0
    if (debug_eval):pretagged_sents = pretagged_sents[:2]
    # count true and false tagger tagged tags
    for j, sent_j in enumerate(pretagged_sents):
        # iterate through tokens in sentence
        for i, (token_i, pretagged_tag_i) in enumerate(sent_j):
            # increment token counter
            cnt_tokens = cnt_tokens + 1
            #extract tagger_tagged_tag_i of tagger tagged tags
            tagger_token_i = sents_tagger_tagged[j][i][0]
            tagger_tagged_tag_i = sents_tagger_tagged[j][i][1]
            # compare tags and increment tags counter
```

```
            if pretagged_tag_i == tagger_tagged_tag_i:
                cnt_true_tags = cnt_true_tags + 1
            else:
                cnt_false_tags = cnt_false_tags + 1
            if (debug_eval):
                print "%s: (%s,%s), (%s,%s)" % (j,
                    token_i, pretagged_tag_i,
                    tagger_token_i, tagger_tagged_tag_i)
    # calculate accuracy
    proportion_of_correctly_tagged_tags = (
        cnt_true_tags/cnt_tokens)
    # return accuracy
    return proportion_of_correctly_tagged_tags
def confusion_matrix(pretagged_sents, sents_tagger_tagged):
    """
        prints confusion matrix out to console
        all numbers in the matrix sum up to number of tokens in text
        optimal result: only diagonal filled with numbers
        """
    # Input data structure: pretagged_sents AND tagger_tagged_sents
    # list of sents containing list of tuples (token, tag)
    # [[(token_0,tag_0), (token_1,tag_1), ...]_sent1,
    #  [(token_0,tag_0), (token_1,tag_1), ...]_sent2, ...]
    # Output data structure: confusion_matrix
    # {(pretagged_tag, tagger_tagged_tag):cnt,
    #  (pretagged_tag, tagger_tagged_tag):cnt,
    #  ... }
    # counters
    confusion_matrix = {}
    cnt_tokens = 0
    if (debug_eval): pretagged_sents = pretagged_sents[:2]
    # count true and false tagger tagged tags
    for j, sent_j in enumerate(pretagged_sents):
        # iterate through tokens in sentence
        for i, (token_i, pretagged_tag_i) in enumerate(sent_j):
            # extract tagger_tagged_tag_i of tagger tagged tags
            tagger_token_i = sents_tagger_tagged[j][i][0]
            tagger_tagged_tag_i = sents_tagger_tagged[j][i][1]
            # increment token counter
            cnt_tokens = cnt_tokens + 1
            # extract and increment counter of tag i
            cnt_i = confusion_matrix.get(
                (pretagged_tag_i, tagger_tagged_tag_i), 0)
            cnt_i = cnt_i + 1
            # compare tags and increment tags counter
            confusion_matrix.update(
                {(pretagged_tag_i, tagger_tagged_tag_i):cnt_i})
            if (debug_eval):
                print "%s: (%s,%s), (%s,%s), (%s)" % (j,
                        token_i, pretagged_tag_i,
                        tagger_token_i, tagger_tagged_tag_i,
                        cnt_i)
    # cast confusion matrix into a list of tuples
    # [(('a', 'a'), 1), (('a', 'b'), 1), ...]
    confusion_matrix_sorted = sorted(
        confusion_matrix.iteritems(), key=operator.itemgetter(0))
    # extract axes
    set_down=set()
    set_right=set()
    for (down, right), c in confusion_matrix_sorted:
        set_down.add(down)
        set_right.add(right)
    # sort axes
    list_down = sorted(set_down)
    delta = list(set_right - set_down)
    list_right = list_down + delta
    #print title
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    print "confusion matrix"
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    # iterate through right axis
    print "%-6s" % (""),
    for right in list_right:
        #print column names
        print "%-6s" % (right[0:5]),
    print
    # iterate through down axis
    for down in list_down:
        #print line names
        print "\n"
        print "%-6s" % (down[0:5]),
        # iterate through right axis
        for right in list_right:
            #print counter
            cnt = confusion_matrix.get((down,right),0)
            print "%-6s" %(cnt),
    print "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    #return confusion matrix
    return confusion_matrix
if __name__ == "__main__":
```

```
    main()



#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 04 - Aufgabe 1, FS16
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                               'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Cazim, Irene, Raffael:
# Wir geben beide Versionen ab. Es reicht eine anzuschauen.
# Linus Version benützt defaultdict(). Diese Datenstruktur kann
# wie eine Matrix in Matlab/R verwendet werden. Entsprechend sind
# seine Funktionsdefinitionen einiges eleganter codiert.
# Rolands Version ist sehr ausführlich dokumentiert. Alle Fragen
# der Aufgabestellung im Main beantwortet und auf Console geprintet.
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Fuer die vollständig gelöste d.h. Übung inkl. beantworteten Fragen
# bitte Rolands Lösung anschauen.
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Reflexion/Feedback
# a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen
# zusammen.
# (Ich kann mich Roland nur anschliessen):
# Ich habe gelernt einen Trainingsalgorithmus für einen Classifier
# (Tagger) selber zu programmieren, damit einen Text zu taggen
# und die Ergebnisse mit Accuracy und Confusionsmatrix zu
# evaluieren. Zweite Erkenntnis: Die Verwendung von 'defaultdicts'
# erleichtert einem das Leben.
# b) Wie viel Zeit hast du in diese Übungen investiert?
# Linus ~12 Stunden
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
from __future__ import division #instead of casting to floats and potentially forgetting
to do so
from nltk.corpus import brown
from collections import defaultdict
import operator
def dict_builder(sent_list):
    """
    function to build a uni- and bigram-dictionary
    input: sentence as a list containing single words
        >>> [('Es','PRON'), ('ist','VERB'),...]
    output: dictionary containing all occured tag/word-combination
        with the corresponding POS-tag-frequencies for
        the word - for uni- and bigrams.
    """
    # initiating dictionnaries (nested dictionnaries)
    unigram_dict = defaultdict(lambda: defaultdict(int))
    bigram_dict = defaultdict(lambda: defaultdict(int))
    for sent in sent_list:

        for i in range(0,len(sent)):
            current_word = sent[i][0]

            prev_tag = sent[i-1][1]

            current_word_tag = sent[i][1]
            tup = (prev_tag, current_word)
            # updating the dictionaries (uni and bigram)
            # bigram
            bigram_dict[tup][current_word_tag] += 1
            # unigram
            unigram_dict[current_word][current_word_tag] += 1

    # returns a tuple with the two dictionaries (uni- and bigram)
    return (bigram_dict, unigram_dict)
def the_glorious_zebra_butt():
    """
    oh, glorious zebra butt, tell us your secret!
    """
    pos_tag = "NOUN" # returns 'NOUN' pos-tag as a last resort ('NOUN' is the most common
pos_tag)
    return pos_tag
def unigram_tagger(word, dictionary, backoff_tagger=the_glorious_zebra_butt):
    """
    backoff-tagger in case the pos_tag-word-combination doesn't
    exist as key in the dictionary
    returns the most probable pos_tag for a given word
    if the unigram is oov, ask the_glorious_zebra_butt for guidance (backoff-"tagger")
    """
    try:
        pos_tag = max(dictionary[word].iteritems(), \
            key=operator.itemgetter(1))[0]
    except ValueError:
        pos_tag = backoff_tagger()
    return pos_tag
def bigram_tagger(sent_list, dictionary, backoff_tagger=None, backoff_dictionary=None):
    """
    actual POS-tagger-function
    input: sentence as list or string
    output: tagged sentence as list with tuples ('word','POS-tag')
```

```
    """
    output_list = []
    for sent in sent_list:
        # prev_tag = None <- would be better... nltk fills in a '.' for no-context?!
        prev_tag = '.'
        sub_list = []
        for i in range(0,len(sent)):
            current_word = sent[i]
            try:
                # save the most frequent tag for this postag/word-combination as 'pos_tag'
                pos_tag = max(dictionary[(prev_tag, current_word)].iteritems(), \
                    key=operator.itemgetter(1))[0]
            # backoff
            except ValueError:
                if backoff_tagger == None:
                    pos_tag = "UNKNOWN"
                else:
                    pos_tag = backoff_tagger(current_word, backoff_dictionary)
            sub_list.append((current_word, pos_tag))
            prev_tag = pos_tag
        output_list.append(sub_list)
    return output_list
def evaluate(tagged_sents, tagger, dictionary, backoff_tagger=None,
backoff_dictionary=None):
    """
    evaluation function:
    input: tagged sentences, tagger, dictionary, backoff_tagger, backoff_dictionary
    output: tuple with tagger, eval_dict, #total_pos_tags, #correct_pos_tags, precision
        eval_dict:
        dictionary with each occuring POS-tag as key and the corresponding
        frequencies of proposed_tags for it
        >>>    {"NOUN": {"NOUN":100, "VERB":2,...}}
                     v
                   correct
    """
    eval_dict = defaultdict(lambda: defaultdict(int))
    # build sentence list without tags to let the tagger do its work
    # this newly tagged list will be compared to the gold standard, which is
    # the same list with the pre-annotated tags
    test_sents = []
    for sent in tagged_sents:
        test_sent = []
        for tup in sent:
            word = tup[0]
            test_sent.append(word)
        test_sents.append(test_sent)

    # depending on whether a backoff_tagger is given, call function differently
    if backoff_tagger == None and backoff_dictionary == None:
        result = tagger(test_sents, dictionary)
    else:
        result = tagger(test_sents, dictionary, backoff_tagger, backoff_dictionary)
    gold = tagged_sents
    total_pos_tags = 0
    for i in range(0,len(gold)):
        for x in range(0,len(gold[i])):
            proposed_tag = result[i][x][1]
            correct_tag = gold[i][x][1]
            if proposed_tag == correct_tag:
                eval_dict[correct_tag][correct_tag] += 1
            else:
                eval_dict[correct_tag][proposed_tag] += 1
            # increment total_pos_tags count by 1
            total_pos_tags += 1
    correct_pos_tags = 0
    # computing the total number of correct pos_tags
    for key in eval_dict:
        correct_pos_tags += eval_dict[key][key]
    precision = correct_pos_tags / total_pos_tags
    evaluation = (tagger, eval_dict, total_pos_tags, correct_pos_tags, precision)
    return evaluation
def display_evaluation(evaluation):
    """
    input: evaluation-function (tuple)
    prints out the evaluation like so:
        total_pos_tags   : #
        correct_pos_tags : #
        precision        : %
    """
    (tagger, eval_dict, total_pos_tags, correct_pos_tags, precision) = evaluation
    # displaying precision (printed)
    return "%s\n%-17s: %i\n%-17s: %i\n%-17s: %f" % (tagger, "total_pos_tags",
total_pos_tags, "correct_pos_tags", correct_pos_tags,\
        "precision", precision)
def display_confusionmatrix(evaluation):
    """
    input: evaluation
    prints out a confusionmatrix
    """
    (tagger, eval_dict, total_pos_tags, correct_pos_tags, precision) = evaluation
    # key_list with keys for the first coloumn
    col_key_list = []
```

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
   for key in eval_dict:
      col_key_list.append(key)
   # sorting the list
   col_key_list.sort()
   # to integrate the "UNKNOWN" tag into the key_list, I created a new list, which takes
in the
   # additional pos_tag, which normally isn't in the universal tagset.
   row_key_list = []
   for key in col_key_list:
      row_key_list.append(key)
   row_key_list.append("UNKNOWN")
   ##start of formatting the matrix
   # header (first row)
   print "\n","%-5s%s" % ("", "|"),
   for key in row_key_list:
      print '%-7s' % (key),
   # horizontal dividing line
   print "\n", (len(row_key_list)+1)*8* "-"

   # actual content rows
   for key in col_key_list:
      row = []
      for i in range(0, len(row_key_list)):
         row.append(eval_dict[key][row_key_list[i]])

      output_row = ['%-8i' % (x) for x in row]
      # each content line
      print '%-5s%s' % (key, "|"), "".join(output_row)
def main():
   print "~~~~~~~~~~~~~beginning of script~~~~~~~~~~~~~~~~"
   # loading the corpus
   print "LOADING CORPUS:\n"
   tagged_sents = brown.tagged_sents(categories="belles_lettres", tagset="universal")
   print "splitting corpus in ratio 90(training)/10(test)...",
   slice_index = int(len(tagged_sents)*0.9)
   training_corpus = tagged_sents[:slice_index]
   test_corpus = tagged_sents[slice_index:]
   print "done"
   print "training corpus: %i sentences (%i words)" % \
      (len(training_corpus), sum([len(x) for x in training_corpus]))
   print "test corpus: %i sentences (%i words)" % \
      (len(test_corpus), sum([len(x) for x in test_corpus]))
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "TRAINING:\n"
   print "building bigram and unigram dictionaries...",
   (bigram_dict, unigram_dict) = dict_builder(training_corpus)
   print "done"
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "TAGGING:\n"
   print "Tag the sentence 'This is a sentence we want to tag' without backoff:"
   print bigram_tagger([["This", "is", "a", "sentence", "we", "want", "to", "tag"]\
        ,[]], bigram_dict)
   print "\nTag the sentence 'This is a sentence we want to tag' with backoff:"
   print bigram_tagger([["This", "is", "a", "sentence", "we", "want", "to", "tag"],[]]\
        , bigram_dict, unigram_tagger, unigram_dict)
   print "\nAnswer to the question:"
   print "The problem here is, that once the 'UNKNOWN'-tag appeares, it 'corrupts' all"
   print "following (pos_tag,word)-checks. This has the consequence that every following"
   print "word is labelled as 'UNKNWON'."
   print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   print "EVALUATION with CONFUSIONMATRIX:\n"
   print "without backoff:"
   evaluation_without_backoff = evaluate(test_corpus, bigram_tagger, bigram_dict)
   print display_evaluation(evaluation_without_backoff)
   display_confusionmatrix(evaluation_without_backoff)
   print "\nwith backoff:"
   evaluation_with_backoff = evaluate(test_corpus, bigram_tagger, bigram_dict,
unigram_tagger, unigram_dict)
   print display_evaluation(evaluation_with_backoff)
   display_confusionmatrix(evaluation_with_backoff)
   print "\n~~~~~~~~~~~~~~~~~end of script~~~~~~~~~~~~~~~~~~"
if __name__ == '__main__':
   main()
```

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
#PCL 2, Uebung 05
#Aufgabe 2
#Irene Ma
from nltk.corpus import brown
import re, string, nltk
from collections import defaultdict
rePunct = re.compile("^[" + string.punctuation + "]*$")
reNum = re.compile("^[0-9]*$")
matrix = defaultdict(lambda: defaultdict(float))
def single_cost(token):
   """
      Helper function to calculate the cost of deletion/insertion
      Args:
         token: a token
```

U5.2
2 Levensthein-Distanz
Aus der Vorlesung kennst du die Berechnung der Levenshtein-Distanz auf buchstäblicher Ebene.
In dieser Übung beschäftigen wir uns mit der Levenshtein-Distanz auf wörtlicher Ebene. Mit der Levenshtein-Distanz auf wörtlicher Ebene kann die Anzahl Operationen, um einen Satz A in einen Satz B umzuwandeln, berechnet werden. Folgend sind die Operationen und ihre Kosten definiert, welche du für diese Übung anwenden sollst:
1. Eine Interpunktion einfügen: 0.1
Alles andere einfügen: 3.0
2. Eine Interpunktion entfernen: 0.1
Alles andere entfernen: 3.0
3. Element x mit Element y ersetzen:

```
    """
    global rePunct
    return 0.1 if rePunct.match(token) else 3.0
def compare_cost(x,y):
    """
        Helper function to calculate the cost of replacement
        Args:
            x,y: tokens to compare
    """
    global rePunct,reNum
    if x == y:
        return 0.0
    #if x and y are not the same, determine their type and calculate cost
    xClass = "punct" if rePunct.match(x) else "num" if reNum.match(x) else "else"
    yClass = "punct" if rePunct.match(y) else "num" if reNum.match(y) else "else"

    if xClass == "punct" and yClass == "punct":
        return 0.1
    elif xClass == "num" and yClass == "num":
        return 4.0
    elif xClass == "else" and yClass == "else":
        return 1.3
    elif xClass != yClass:
        return 16.0
    else:
        raise Exception("Hmm " + x + ", " + y)
def gen_edit_dist(a,b):
    """
        Function that calculates the general edit distance of two sentences
        via dynamic programming, top-down (recursive + memoization)
        Args:
            a,b: list of tokens (originally sentences)
    """
    global matrix
    #if a and b are not empty sentences, continue
    if(len(a) != 0 or len(b) != 0):
        A = " ".join(a)
        B = " ".join(b)
        #if the costs for matrix[A][B] haven't been calculated yet, calculate the minimum cost
        if(not matrix[A][B]):
            ops = []
            if(len(a) != 0 and len(b) != 0):
                #insertion cost
                ops.append(gen_edit_dist(a,b[:-1]) + single_cost(b[-1]))
                #deletion cost
                ops.append(gen_edit_dist(a[:-1],b) + single_cost(a[-1]))
                #replacement cost
                ops.append(gen_edit_dist(a[:-1],b[:-1]) + compare_cost(a[-1],b[-1]))
                #transposing cost
                if(len(a)>1 and len(b)>1 and a[-2] == b[-1] and a[-1] == b[-2]):
                    ops.append(gen_edit_dist(a[:-2],b[:-2])+0.4)
                matrix[A][B] = min(ops)
            elif(not len(a)): #B is longer than A
                insCost = gen_edit_dist(a,b[:-1]) + single_cost(b[-1])
                matrix[A][B] = insCost
            elif(not len(b)): #B is shorter than A
                delCost = gen_edit_dist(a[:-1],b) + single_cost(a[-1])
                matrix[A][B] = delCost
    else:
        #if both are empty we are at the start
        A=""
        B=""
        matrix[A][B] = 0.0
    return matrix[A][B]
def main():
    global matrix
    A = raw_input("Sentence A: ")
    B = raw_input("Sentence B (enter brown for d)): ")
    if B != "brown":
        a = nltk.wordpunct_tokenize(A);
        b = nltk.wordpunct_tokenize(B);
        res = gen_edit_dist(a,b)

        print "The General Edit Distance of",A,"and",B, "is",res
    else:
        a = nltk.wordpunct_tokenize(A);
        sents = brown.sents()

        listofres = [(gen_edit_dist(a,s),s) for s in sents]
        res = min(listofres, key=lambda x: x[0])
        print "The brown sentence closest to '",A,"' is '"," ".join(res[1]),"' with
general edit distance",res[0]
if __name__ == "__main__":
    main()



#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 04 - Aufgabe 1, FS16
#
# Autoren:
```

- x und y sind beides Interpunktionen: 0.1
- x und y sind beides Zahlen: 4.0
- x und y sind beides Wörter (‘abc3’ gilt als ein Wort): 1.3
- x und y sind nicht vom gleichen Typ: 16.0

Beispiel:

- Satz A: Vladimir Levenshtein übernahm dies im Jahre 1960.
- Satz B: Vladimir Iosifovich Levenshtein entwickelte dies im Jahre 1965.
- 3.0 (‘Iosifovich’ einfügen) + 1.3 (‘übernahm’ mit ‘entwickelte’ ersetzen)

+ 4.0 (‘1960’ mit ‘1965’ ersetzen) = 8.3

a) Folgend sind Satz A und Satz B. Berechne die Kosten der Operationen (= die Distanz), um
Satz A in Satz B umzuwandeln und notiere jeden Schritt wie oben veranschaulicht. (A soll am
Schluss so aussehen wie B):

- Satz A: Computerlinguistik 2 ist spannend.
- Satz B: Computerlinguistik macht Spass und ist spannend!
- Wieso ist die Distanz 22.1 nicht die optimale Lösung?

Erkläre in höchstens drei Sätzen.

b) Implementiere nun die Funktion lev_word(A,B), welche die Levenshtein-Distanz auf Wort-
Ebene von Satz A nach Satz B anhand der oben definierten Kosten berechnet. Überprüfe dein
Skript mit den Sätzen von a).

c) Jetzt sollst du die Funktion gen_edit_dist(A,B) implementieren, welche eine erweiterte Form
der Levenshtein-Distanz ist. Zusätzlich zu den drei Operationen (einfügen, entfernen, erset-
zen) soll nun untenstehende Operation mit ihren Kosten möglich sein. Anstelle der klassischen
Levenshtein-Distanz kann nun die generelle Bearbeitungsdistanz (‘general edit distance’) be-
rechnet werden.

- Zwei benachbarte Wörter transponieren: 0.4

Beispiel:

- Satz A: In Rätsel immer Yoda spricht.
- Satz B: Yoda spricht immer in Rätsel.

Generelle Bearbeitungsdistanz: 5.2

d) I wish you loved me.

Verwende nun die generelle Bearbeitungsdistanz, um herauszufinden, welcher Satz im Brown
Korpus (nltk.corpus.brown.sents()) diesem am ähnlichsten ist und was die generelle Bear-
beitungsdistanz ist (die Berechnungen können eine Weile dauern).

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                          'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Reflexion/Feedback
# a) Fasse deine Erkenntnisse und Lernfortschritte in zwei Sätzen
# zusammen.
# Die Arbeiten von Richard Bellman gehören mit zum Schwierigsten,
# was ich an der ETH hätte verstehen sollen. Ich habe diese Übung
# zwar mit viel Aufwand mit einem bottom-up-approach lösen können,
# war jedoch nicht in der Lage einen rekursiven Algorithmus mit
# Memoization zu implementieren. Mir war auch lange nicht klar,
# ob meine Implementation richtig funktioniert, da sie eine
# Subtitution mit Kosten 16 durch ein löschen und einfügen ersetzt.
# In der Matrix also nicht diagonal, sondern erst nach unten dann
# nach rechts geht.
# b) Wie viel Zeit hast du in diese Übungen investiert?
# Roland 20 Stunden
# Linus 10 Stunden
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#import
from __future__ import division
from nltk.corpus import brown
import re, string, nltk
from collections import defaultdict
#global variables
rePunct = re.compile("^[" + string.punctuation + "]*$")
reNum = re.compile("^[0-9]*$")
reWord = re.compile("^[a-zA-ZäöüÄÖÜ0-9_]*$")
matrix = defaultdict(lambda: defaultdict(float))
#debug flag
debug = 0
print_matrix = 1
def main():
   # levenshtein distance for lists of tokens
   #-----------------------------------------
    #PCL Solutions
    #-------------
    print "~~~~~~~~~Beispiel)~~~~~~~~~~~~~"
    l1 = "Vladimir Levenshtein uebernahm dies im Jahre 1960 .".split()
    l2 = "Vladimir Iosifovich Levenshtein entwickelte dies im Jahre 1965 .".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "ld=3+1.3+4=8.3, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    print "~~~~~~~~~2a und 2b)~~~~~~~~~~~~~"
    l1 = "Computerlinguistik 2 ist spannend .".split()
    l2 = "Computerlinguistik macht Spass und ist spannend !".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "ld= nicht 22.1, ->%s" % (ld)
    print "\nWeshalb ist 22.1 nicht die Optimale Lösung?\n" \
        "Unsere Version geht wie folgt vor:\n" \
        "delete: 2 (Kosten 1 * 3)\n" \
        "insert: macht Spass und (Kosten 3 * 3.0)\n" \
        "substitute: . und ! (Kosten 1 * 0.1)\n" \
        "Totale Kosten von 12.1 nicht 22.1\n" \
        "Die substitution (go diagonal) mit Kosten 16\n" \
        "wird immer umgangen\n" \
        "und durch ein delete (go down) und \n" \
        "ein insert (go right) ersetzt."
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    print "~~~~~~~~~2c)~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    l1 = "spricht Yoda in immer Raetseln .".split()
    l2 = "Yoda spricht immer in Raetseln !".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "ld=0.9, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "das ist !".split()
    l2 = "ist das !".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "ld=0.4, ->%s" % (ld)
    print "~~~~~~~~~2d)~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    #load brown corpus sents
    print "Brown categories:\n %s\n" %brown.categories()
    brown_sents=brown.sents(categories = "romance")
    #find minimal distance between l1 and l2
    l1 = "I wish you loved me .".split()
    ld_min=1000 #Startwert
    #iterate through sent in brown corpus
    print"\nUm die beste Lösung zu finden:\n" \
       "Setze den print_matrix flag auf 0 " \
       "und iteriere über die gesamte Kategorie " \
       "romance (entferne auf nachfolgender Zeile [0:2])\n"
    for sent in brown_sents[0:2]:
        #calculate distance
        l2 = sent
        ld = levenshtein_on_tokens(l1, l2)
        #print out best match so far
        print l1, "\n", l2
        print "ld= , ->%s" % (ld)
        if ld<ld_min:
```

```
            ld_min=ld
            l2_min=l2
        print "ld_min= ", ld_min
        print "l2_min= ", l2_min
        print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    # Additional:
    # Test cases
    print"\nZusätzliche Testfälle:\n"
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    # ----------------------------------------
    l1 = "a b c d . a a".split()
    l2 = "a b c d . a c".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "1) ld= 1.3, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b c d . a 2".split()
    l2 = "a b c d . a 4".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "2) ld= 4.0, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b .".split()
    l2 = "a b !".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "3) ld= 0.1, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b c 4".split()
    l2 = "a b c .".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "4) ld= nicht 16.0, 3.1, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b c d".split()
    l2 = "a e".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "5) ld= 7.3, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b c d .".split()
    l2 = "a e i o".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "6) ld= 4.0, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "a b c d !".split()
    l2 = "a b c d .".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "7) ld= 0.1, ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "!".split()
    l2 = "b".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "8) ld= nicht 16, 3.1 , ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "! 3 4".split()
    l2 = "b ? a".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "9) ld=12.1 , ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    l1 = "l e v e n s h t e i n".split()
    l2 = "m e i l e n s t e i n".split()
    ld = levenshtein_on_tokens(l1, l2)
    print l1, "\n", l2
    print "10) ld=7.8 , ->%s" % (ld)
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
    # levenshtein distance for strings of characters
    # -----------------------------------------------
    print"\nZusätzlich:"
    print "levenshtein distance for strings of chars\n" \
        "Dieser Algorithmus war das Grundgerüst der\n" \
        "gesamten Aufgabe"
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    s1 = "kitten"
    s2 = "sitting"
    ld = levenshtein_on_characters(s1, s2)
    print "\nld= ", ld
    print s1, "\n", s2
#The first version is a Dynamic Programming algorithm,
#with the added optimization that only the last two rows of
#the dynamic programming matrix are needed for the computation:
def levenshtein_on_tokens(l1, l2):
    """
    function to return a levenshtein-distance (float)
    This task was solved BOTTM-UP (starting at the first
        token an continue up to the whole sentence)
    input:  two sentences
    output: levenshtein distance (float)
```

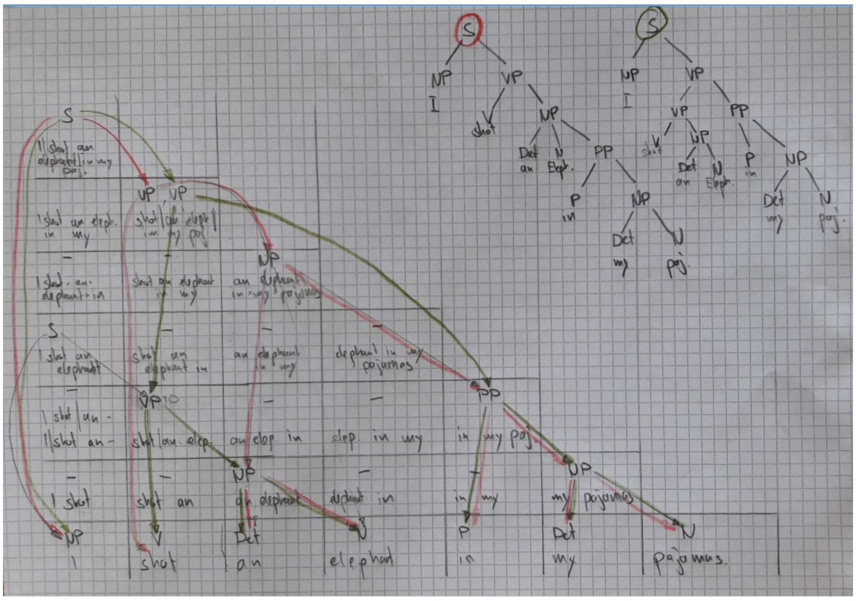
```
    """
    # l1 must be the longer list
    if len(l1) < len(l2):
        return levenshtein_on_tokens(l2, l1)
    # now len(l1) >= len(l2)
    if len(l2) == 0:
        return len(l1)
    pprevious_row=[]

    # first row
    previous_row=[0]
    for j, c2 in enumerate(l2):
        previous_row.append(previous_row[j]+costs_insertion(c2))
    if (print_matrix):
        print ("%5s")%(""),
        for j, c2 in enumerate(l2):
            print ("%5s" % (c2[:4])),
        print
        for j, c2 in enumerate(l2):
            print ("%5s" % (previous_row[j])),
        print ("%5s" % (previous_row[j+1])),
        print
    # iterate through the longer list (go down)
    for i, c1 in enumerate(l1):
        if (debug): print "i=%s, c1=%s" % (i+1, c1)
        # first element of row i+1
        current_row = [previous_row[0]+costs_deletion(c1)]
        if (debug): print " current_row", current_row
        # iterate through the smaller list (go right)
        for j, c2 in enumerate(l2):
            if (debug): print "    j=%s, c2=%s" % (j+1, c2)
            """"
            compare current element (i+1,j+1) with:
            element above (i,j+1): equivalent to previous_row[j + 1]
            element left (i+1,j): equivalent to current_row[j]
            element left above (i,j): previous_row[j]
            """
            # element above: move down -> delete label from l1
            cost_deletion = costs_deletion(c1)#1
            deletions = previous_row[j + 1] + cost_deletion
            deletions = round(deletions, 2)
            # element left: move right -> insert label of l2 to l1
            cost_insertion = costs_insertion(c2)#1
            insertions = current_row[j] + cost_insertion
            insertions = round(insertions, 2)
            # element left above -> substitute or keep
            cost_substitution = (c1 != c2)*costs_substitution(c1,c2)#(c1 != c2)
            substitutions = previous_row[j] + cost_substitution
            substitutions = round(substitutions, 2)
            ## implementation of transposing costs
            transposing = False
            try:
                transposing_list1 = [l1[i], l1[i-1]]
                transposing_list2 = [l2[j], l2[j-1]]
                ##  by jumping over one row and accessing the pprev-row directly, the
matrix
                #   doesn't get filled in correctly, because the computing-steps in
                #   between don't get ignored. This makes it impossible to backtrack the
path nicely
                #   (i.e. without jumping over a matrix cell)
                transposing = pprevious_row[j-1] + costs_transposing(transposing_list1,\
                     transposing_list2)
            except IndexError:
                None
            except TypeError:
                None
            # in case a tranposition is possible, consider it as a possible minimal
value as well
            if transposing:
                current_row.append(min(insertions, deletions, substitutions,
transposing))

            current_row.append(min(insertions, deletions, substitutions))
            if (debug): print "    cost_insertion, cost_deletion, cost_substitution", \
                cost_insertion, cost_deletion, cost_substitution
        if (print_matrix):
            for j, c2 in enumerate(l2):
                print ("%5s" %(current_row[j])),
            print ("%5s %5s" % (current_row[j+1],c1)),
            print
        # previous row gets updated
        if (previous_row): pprevious_row = previous_row
        previous_row = current_row
    #return last element of last row, which is the minimal levenshtein distance
    return previous_row[-1]
def costs_transposing(x,y):
    """
    input:  two lists with len = 2
    output: returns the cost of 0.4 if the items in the list
            satisfy the condition to be able to be transposed
    Ex: Das Haus ist grün !
        Das Haus grün ist !
```

```
        extracted lists as input for this function:
        ["ist", "grün"], ["grün", "ist"]
        -> returns 0.4
    """
    try:
        if x[1] == y[0] and x[0] == y[1]:
            return 0.4
    # if the lists aren't long enough (at the start/end of the sentence)
    except IndexError:
        None
def costs_insertion(x):
    """
    cost-function for insertions of a token
    input:  a single token
    output: depending on the type of token (punctuation or not)
            returns the cost:
                > punctuation:      0.1
                > everything else:  3.0
    """
    # checks whether the token is a punctuation
    match_obj = re.match(rePunct, x)
    if (match_obj):
        return 0.1
    else:
        return 3.0
def costs_deletion(x):
    """
    cost-function for deletions of a token
    input:  a single token
    output: depending on the type of token (punctuation or not)
            returns the cost:
                > punctuation:      0.1
                > everything else:  3.0
    """
    # checks whether the token is a punctuation
    match_obj = re.match(rePunct, x)
    if (match_obj):
        return 0.1
    else:
        return 3.0
def costs_substitution(x, y):
    """
    cost-function for substitutions of tokens
    input:  two tokens
    output: returns the cost according to the combination
            of tokens:
                > x,y are both punctuations: 0.1
                > x,y are both numbers:      4.0
                > x,y are both words:        1.3
                (as soon as the string contains a alphabetical
                    character it counts as word)
                > x,y are different types:   16.0
    """
    # checking the types of the tokens x and y
    match_obj_x_punct = re.match(rePunct, x)
    match_obj_y_punct = re.match(rePunct, y)
    match_obj_x_num = re.match(reNum, x)
    match_obj_y_num = re.match(reNum, y)
    match_obj_x_word = re.match(reWord, x)
    match_obj_y_word = re.match(reWord, y)
    # both are punctuations
    if (match_obj_x_punct and match_obj_y_punct):
        return 0.1
    # both are numbers
    elif (match_obj_x_num and match_obj_y_num):
        return 4.0
    # both are words
    elif (match_obj_x_word and match_obj_y_word
          and not match_obj_x_num and not match_obj_y_num):
        return 1.3
    # they are from a different type
    else:
        return 16.0
#DP
def levenshtein_on_characters(s1, s2):
    """The Levenshtein algorithm (also called Edit-Distance) calculates the least
    number of edit operations that are necessary to modify one string to obtain
    another string. The most common way of calculating this is by the dynamic
    programming approach. A matrix is initialized measuring in the (m,n)-cell
    the Levenshtein distance between the m-character prefix of one with the
    n-prefix of the other word. The matrix can be filled from the upper left
    to the lower right corner. Each jump horizontally or vertically corresponds
    to an insert or a delete, respectively. The cost is normally set to 1 for each
    of the operations. The diagonal jump can cost either one, if the two characters
    in the row and column do not match or 0, if they do. Each cell always minimizes
    the cost locally. This way the number in the lower right corner is the
    Levenshtein distance between both words. Here is an example that features
    the comparison of "meilenstein" and "levenshtein":"""
    if len(s1) < len(s2):
        return levenshtein_on_characters(s2, s1)
    # len(s1) >= len(s2)
    if len(s2) == 0:
```

PCL II – Zusammenfassung Code

Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```python
        return len(s1)
    previous_row = range(len(s2) + 1)
    if (debug): print "previous_row", previous_row
    for i, c1 in enumerate(s1):
        if (debug): print "i=%s, c1=%s" %(i, c1)
        current_row = [i + 1]
        if (debug): print " current_row", current_row
        for j, c2 in enumerate(s2):
            if (debug): print "    j=%s, c2=%s" % (j, c2)
            # j+1 instead of j since previous_row and current_row
            # are one character longer than s2
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            if (debug): print "    insertions, deletions, substitutions", \
                insertions, deletions, substitutions
            current_row.append(
                min(insertions, deletions, substitutions))
        if (debug): print " current_row", current_row
        previous_row = current_row
    return previous_row[-1]
if __name__ == "__main__":
   main()
```



U6.1

1 Manuelles CYK-Parsing

Gegeben ist die folgende kontextfreie Grammatik:

S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas'
V -> 'shot'
P -> 'in'

Parse den Satz I shot an elephant in my pajamas mithilfe des CYK-Algorithmus. Welche Probleme ergeben sich?

Abzugeben sind die Tabelle und die entstandenen Bäume.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# PCL2-Ü6-Aufgabe 2
# Musterlösung von Raphael Balimann (raphael.balimann@uzh.ch) - HS 2015
#
import nltk
import sys
def main():
    """ Showcasing the functionality with the sentences given"""
    # How can this be made more robust? Hint: getopt
    grammar_path = sys.argv[1]
    sentence_path = sys.argv[2]
    output_path = sys.argv[3]
    forest = parse_file(grammar_path, sentence_path)
    # for item in forest:
    #     for i in item:
    #         print pretty_print_qtree(i)
    generate_qtree_document(forest, output_path)
    return
def create_grammar(raw_grammar):
    """ A wrapper that returns a CFG in CNF"""
    return nltk.CFG.fromstring(raw_grammar)
def parse_sentence(grammar, sentence):
    """ A wrapper that accepts:
        - a CFG in CNF
        - a sentence as a list of token
        Returns a tree object
    """
    cfg_grammar = create_grammar(grammar)
    current_parser = nltk.ChartParser(cfg_grammar)
    tree = current_parser.parse(sentence)
    return tree
def parse_file(grammar_path, sentence_path):
```

U6.2

2 Kontextfreie Grammatiken in Python

Für diese Aufgabe können verschiedene Parser aus NLTK verwendet werden. Wähle einen Parser aus und implementiere die folgenden Funktionalitäten für die Grammatik aus Aufgabe 1:

a) Parsen eines einzelnen Satzes als String
b) Parsen von Sätzen und Grammatiken aus Plain-Text-Dateien
c) Eine schöne Darstellung der Hierarchien auf der Kommandozeile
d) Ausgabe von kompilierbarem *TeX-Quellcode aller möglichen Bäume eines Satzes

```
    """ A wrapper that accepts two file paths:
        - a CFG as plain text
        - a plain text file with sentences
        Returns a list of tree objects
    """
    with open(grammar_path) as grammar_file:
        raw_grammar = []
        for line in grammar_file:
            raw_grammar.append(line)
    with open(sentence_path) as sentence_file:
        raw_sentences = []
        for line in sentence_file:
            raw_sentences.append(line)
    sentences = [nltk.word_tokenize(raw_sent) for raw_sent in raw_sentences]
    tree_list = []
    for sentence in sentences:
        tree_list.append(parse_sentence(raw_grammar, sentence))
    return tree_list
def pretty_print_qtree(tree):
    """ A function to convert tree objects
        into the notation used by the LaTeX qtree package
        Accepts a single tree item,
        returns a string that can be used further
    """
    return tree.pformat_latex_qtree()
def pretty_print_stree(tree):
    """ A function to convert tree objects
        into a notation that can be displayed on the standard output
        Accepts a single tree item,
        prints directly to the standard output
    """
    tree.pretty_print()
    return
def generate_qtree_document(trees, filepath):
    """ A function to create a full (Xe)LaTeX document containing trees
        Accepts a list of tree objects
        Writes a full source file to the the specified path
    """
    # frontmatter for a full tex file, should be usable with many diferent (La)TeX
varieties
    frontmatter = r"""
        \documentclass[12pt]{article}
        \usepackage{tikz}
        \usepackage{tikz-qtree}
        \title{Programmiertechniken in der Computerlinguistik II: Assignment 6}
        \author{Raphael Balimann - 11-739-679}
        \date{Abgabe: 2016/06/02}
        \begin{document}
        \maketitle
    """
    # centering
    start_center = r'\begin{center}'
    end_center = r'\end{center}'
    # endmatter for document
    endmatter = r"""
    \end{document}
    """
    with open(filepath, 'w') as output:
        output.write(frontmatter)
        for forest in trees:
            for tree in forest:
                print "Writing tree to file:"
                pretty_print_stree(tree)
                output.write(start_center)
                output.write(pretty_print_qtree(tree))
                output.write(end_center)
        output.write(endmatter)
    return
if __name__ == '__main__':
    main()
```

File: cfg.txt

```
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas'
V -> 'shot'
P -> 'in'
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# PCL2-Ü6-Aufgabe 3
# Musterlösung von Raphael Balimann (raphael.balimann@uzh.ch) - HS 2015
#
import nltk
import sys
def main():
```

U6.3
3 Probabilistische kontextfreie Grammatiken in Python
In den bisherigen Aufgaben wurden Ambiguitäten, welche aus der Grammatik entstanden, unverän-
dert weitergegeben. In dieser Aufgabe soll dieses Problem anhand einer probabilistischen kontext-
freien Grammatik (PCFG) reduziert werden.
Ergänze die bestehende Grammatik aus Aufgabe 1 mit Wahrscheinlichkeiten und implementiere die

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    """ Showcasing the functionality with the sentences given"""
    # How can this be made more robust? Hint: getopt
    grammar_path = sys.argv[1]
    sentence_path = sys.argv[2]
    output_path = sys.argv[3]
    forest = parse_file(grammar_path, sentence_path)
    # for item in forest:
    #     for i in item:
    #         print pretty_print_qtree(i)
    generate_qtree_document(forest, output_path)
    return
def create_grammar(raw_grammar):
    """ A wrapper that returns a PCFG in CNF"""
    return nltk.PCFG.fromstring(raw_grammar)
def parse_sentence(grammar, sentence):
    """ A wrapper that accepts:
        - a CFG in CNF
        - a sentence as a list of token
        Returns a tree object
    """
    pcfg_grammar = create_grammar(grammar)
    current_parser = nltk.pchart.InsideChartParser(pcfg_grammar)
    tree = current_parser.parse(sentence)
    return tree
def parse_file(grammar_path, sentence_path):
    """ A wrapper that accepts two file paths:
        - a PCFG as plain text
        - a plain text file with sentences
        Returns a list of tree objects
    """
    with open(grammar_path) as grammar_file:
        raw_grammar = []
        for line in grammar_file:
            raw_grammar.append(line)
    with open(sentence_path) as sentence_file:
        raw_sentences = []
        for line in sentence_file:
            raw_sentences.append(line)
    sentences = [nltk.word_tokenize(raw_sent) for raw_sent in raw_sentences]
    tree_list = []
    for sentence in sentences:
        tree_list.append(parse_sentence(raw_grammar, sentence))
    return tree_list
def pretty_print_qtree(tree):
    """ A function to convert tree objects
        into the notation used by the LaTeX qtree package
        Accepts a single tree item,
        returns a string that can be used further
    """
    return tree.pformat_latex_qtree()
def pretty_print_stree(tree):
    """ A function to convert tree objects
        into a notation that can be displayed on the standard output
        Accepts a single tree item,
        prints directly to the standard output
    """
    tree.pretty_print()
    return
def generate_qtree_document(trees, filepath):
    """ A function to create a full (Xe)LaTeX document containing trees
        Accepts a list of tree objects
        Writes a full source file to the the specified path
    """
    # frontmatter for a full tex file, should be usable with many diferent (La)TeX
varieties
    frontmatter = r"""
        \documentclass[12pt]{article}
        \usepackage{tikz}
        \usepackage{tikz-qtree}
        \title{Programmiertechniken in der Computerlinguistik II: Assignment 6}
        \author{Raphael Balimann - 11-739-679}
        \date{Abgabe: 2016/06/02}
        \begin{document}
        \maketitle
    """
    # centering
    start_center = r'\begin{center}'
    end_center = r'\end{center}'
    # endmatter for document
    endmatter = r"""
    \end{document}
    """
    with open(filepath, 'w') as output:
        output.write(frontmatter)
        for forest in trees:
            for tree in forest:
                print "Writing tree to file:"
                pretty_print_stree(tree)
                output.write(start_center)
                output.write(pretty_print_qtree(tree))
                output.write(end_center)
        output.write(endmatter)
```

Funktionalitäten der vorherigen Aufgabe.
• Welche Schwierigkeiten ergeben sich bei PCFG?
• Welche Wahrscheinlichkeitsverteilung lässt immer noch Ambiguitäten zu?
Hinweise zu L A TEX
• Der von den Skripten ausgegebene *TeX-Quellcode sollte (idealerweise) kompilierbar sein.
• Für die Darstellung von Syntaxbäumen bietet sich die qtree-Syntax an, welche von verschie-
denen Seiten her unterstützt wird.
• Ein Beispiel für *TeX-Quellcode mit einem Syntaxbaum ist mit der Übung mitgeliefert.

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
    return
if __name__ == '__main__':
    main()


File: pcfg.txt
S -> NP VP  [1.0]
PP -> P NP [1.0]
NP -> Det N [0.4] | Det N PP [0.4] | 'I' [0.2]
VP -> V NP [0.6] | VP PP [0.4]
Det -> 'an' [0.4] | 'my' [0.6]
N -> 'elephant' [0.5] | 'pajamas' [0.5]
V -> 'shot' [1.0]
P -> 'in' [1.0]
```

U6.2 U6.3
Linus

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# PCL-I: Uebung 06 - FS16
#
# Autoren:
# c(Student, Martikelnummer) -> {'Roland Benz' : '97-923-163',
#                          'Linus Manser' : '13-791-132'}
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Aufgabe 1:
# siehe 'Aufgabe01.jpg'
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Aufgabe 3:
# (Wir haben uns entschieden, den PCFG-Parser direkt in die Aufgabe 2
# zu integrieren. Je nach Grammatik (mit oder ohne Wahrscheinlichkeiten)
# waehlt das Script den funktionierenden Parser aus.)
#
# Bei der PCFG ergeben sich unter anderem diese Schwierigkeiten:
# - Lexikalische Informationen werden bei der Beschaffung der Wahrschein-
#   lichkeiten nicht miteinbezogen. Dies würde viele Ambiguitäten aus
#   dem Weg schaffen.
# - Die Summe der Wahrscheinlichkeiten für jede Art von Regel muss 1.0
#   ergeben. Dies führt bei inkompletten Grammatiken zu teils falschen
#   Wahrscheinlichkeitsverteilungen bzw. man schreibt einem Konstituenten
#   einen zu hohen Wert zu.
#
# Gegeben: NP -> Det N [0.4] | Det N PP [0.3] | 'I' [0.3]
# ... haben wir gesehen, dass bei 'VP -> V NP [0.25] | VP PP [0.75]'
# der Wendepunkt für die beiden verschiedenen Baumvarianten ist.
# Theoretisch sollten nur diese Regeln für die verschiedenen Baum-
# varianten ausschlaggebend sein. Jedoch haben wir es noch nicht
# geschafft, den Parser so weit zu bringen, dass er sich gar nicht
# entscheiden kann und beide Varianten ausgibt.
#
# komplette Ausgabe (fyi)
# S -> NP VP [1.0]
# PP -> P NP [1.0]
# NP -> Det N [0.4] | Det N PP [0.3]| 'I' [0.3]
# VP -> V NP [0.25] | VP PP [0.75]
# Det -> 'an' [0.5] | 'my' [0.5]
# N -> 'elephant' [0.6] | 'pajamas' [0.4]
# V -> 'shot' [1.0]
# P -> 'in' [1.0]
# # Wendepunkt bei:
# VP -> V NP [0.25] | VP PP [0.75]
# VP PP mehr als 0.75:
#         S
#  ┌──────┴───────────────────────┐
#  │                              VP
#  │              ┌───────────────┴────────────────┐
#  │              VP                               PP
#  │      ┌───────┴──────┐                  ┌──────┴──────┐
#  │      │              NP                 │             NP
#  │      │       ┌──────┴────────┐         │      ┌──────┴───────┐
#  NP     V      Det              N         P     Det             N
#  │      │       │               │         │      │              │
#  I     shot     an           elephant     in     my           pajamas

#  VP PP weniger als 0.75:
#         S
#  ┌──────┴────────────────┐
#  │                       VP
#  │      ┌────────────────┴─────────┐
#  │      │                          NP
#  │      │       ┌────────┬─────────┴──────┐
#  │      │       │        │                PP
#  │      │       │        │         ┌──────┴──────┐
#  │      │       │        │         │             NP
#  │      │       │        │         │      ┌──────┴────────┐
#  NP     V      Det       N         P     Det              N
#  │      │       │        │         │      │               │
#  I     shot     an   elephant     in     my          pajamas
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Reflexion/Feedback
# a) Ich habe ein neues Modul von nltk kennengelernt, welches mir
#    erlaubt mit kontextfreien Grammatiken (mit oder ohne Wahr-
#    scheinlichkeiten) Syntaxbäume zu generieren und sie auf
#    verschiedene Arten auszugeben. Zusätzlich konnte ich mit
#    der ersten auf Aufgabe von Hand den CYK-Algorithmus "visualisieren"
```

```python
#    und ihn somit besser kennenlernen.
#
# b) 8 Stunden
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#imports
import nltk
import argparse
import sys
import os
#debug flag - set to 1 if in debug mode
DEBUG = 0
def build_tree_variations(parser, sent):
    """
    function that compiles all possible or the most probable tree(s)
    for a sentence, depending on which parser you use
    input:  parser (ChartParser, ViterbiParser, ...)
            a sentence which needs to be parsed
    output: ChartParser -> all possible trees
            ViterbiParser -> most probable tree
            as a tuple together with the corresponding sentence
    """
    # list of all possible trees generated from one sentence
    tree_variations = []
    if DEBUG: print "\ninformation for sent:", sent

    for tree in parser.parse(sent):
        if DEBUG: print "type of tree: ", type(tree)
        tree_variations.append(tree)
    if DEBUG: print "tree variations: ", tree_variations
    # returns the trees together with the original sentence as a tuple
    # the sentence is passed on, to be able to be accessed later on for the display
    return (sent, tree_variations)
def write_out_to_tex(trees, out_file, grammar):
    """
    all-in-one function to convert all trees from all sentences into a .tex-file
    combines static content (prolog, epilog, subsections, ...) with the syntax trees
    and used grammar
    input:  all trees from all sentences
            output file (given as an argument)
            the grammar used for parsing the trees
    output: TeX-file written to the output file (pref: *.tex)
    The TeX-file is compilable with the 'pdflatex' command
    """
    br = "\n" #linebreak
    ## casting the grammar to a string object (was CFG/PCFG object)
    #  it will later be inserted into the static content
    grammar = str(grammar)
    ## hard-coded text as static content for the .tex-file
    #  including:
    #    -the prolog (usedpackages, author, date, ...)
    #    -the grammar (static, since only one grammar is used per file)
    #    -the "epilog" a.k.a end-of-document declaration
    #  the variable content i.e. non-static content
    if DEBUG: print "grammar for texfile:", grammar
    prolog = r"\documentclass[12pt]{article}" + br + \
    r"\usepackage{tikz}" + br + \
    r"\usepackage{tikz-qtree}" + br + \
    r"\title{Programiertechniken in der Computerlinguistik II: Assignment 6}" + br + \
    r"\author{Linus Manser (13-791-132), Roland Benz (97-923-163)}" + br + \
    r"\date{Abgabe: 2016/06/08}" + br + br + \
    r"\begin{document}" + br + br + \
    r"\maketitle" + br + \
    r"\section{Grammar}" + br + \
    r"\begin{verbatim}" + br + \
    grammar + br + \
    r"\end{verbatim}" + br + \
    r"\section{Syntax trees}" + br
    content = ""
    # non-static content starts here
    for tree_variations in trees:
        # extracting the data from the tuple
        (sent, tree_variations) = tree_variations
        sent = " ".join(sent)
        content += r"\subsection{'" + sent + r"'}" +\
            br + r"\begin{enumerate}" + br

        for tree in tree_variations:
            # http://www.nltk.org/howto/tree.html
            # compiling latex content which goes in-between (actual trees)
            content += r"\item\begin{center}" + "\n%s\n" % tree.pformat_latex_qtree() +\
                 br + r"\end{center}" + br

        content += r"\end{enumerate}" + br
        # non-static content ends here

    # marking the end of the latex-file
    epilog = r"\end{document}"
    # concatenating the whole content (static and non-static)
    whole_content = prolog + content + epilog
    # write the whole content to the given output file
    out_file.write(whole_content)
    if DEBUG: print "DONE: outfile written (see below)\n", whole_content
def main():
```

```
    """
    main function of the second and third part of PCL2 exercise 6
    call of the script via the command line. Example call:

        $ python aufgabe02.py -g grammar.txt -s sentences.txt -o out.tex
    required arguments:
     -g / --grammar: txt-file containing either a CFG or PCFG
    optional arguments:
     -s / --sents:   txt-file containing sentences (one per line)
     -o / --out:     tex-file where the trees should get written to
    Unless you set an output file, the parsed sentences are only displayed
    on the command line. Otherwise, the trees are written qtree-conform to
    the declared tex-file.
    Assuming you have LaTeX installed, you can create a pdf file with your
    trees by typing the following command in your command line:
        $ pdflatex outfile.tex

    If no sentences(-s) given, you can write your sentences directly on the command
    line. To finish the input-mode, press 'ctrl+D'.
    """
    # setting up the arguments with argparse
    argparser = argparse.ArgumentParser()
    argparser.add_argument('-o','--out', type=argparse.FileType('w'),\
        metavar='FILE', help='output file')
    argparser.add_argument('-g','--grammar', type=argparse.FileType('r'),\
        metavar='FILE', help='grammar file')
    argparser.add_argument('-s','--sents', type=argparse.FileType('r'), \
        default=sys.stdin, metavar='FILE', help='sentence file')
    args = argparser.parse_args()

    # try to form a string from the data of the grammar file
    try:
        grammar_string = "".join(args.grammar)
    # if no grammar is given, exit the script
    # (assuming that nobody wants to write the same grammar over and over again)
    except TypeError:
        print "try:\t$ python aufgabe02.py -g [grammar.txt] -s [sentence.txt] -o "\
        "[outfile.tex]\n\t(where '-s' and '-o' are optional arguments)"
        exit()
    # parsing grammar from string
    try:
        grammar = nltk.CFG.fromstring(grammar_string)
        parser = nltk.ChartParser(grammar)
    except ValueError:
        # Part of Ex03
        # if the grammar contains probabilites, take the PCFG-method
        # (and use the ViterbiParser)
        grammar = nltk.PCFG.fromstring(grammar_string)
        parser = nltk.ViterbiParser(grammar)

    if DEBUG: print "parser used:", type(parser)
    # collecting input form the given file (or: stdin by default)
    sent_file = args.sents
    # assigning the output file to a new variable
    out_file = args.out
    # list containing all possible trees for every sentence
    all_trees_from_all_sentences = []
    # parse all sentences inside the sentence file
    for sent in sent_file:
        if DEBUG: print "\nparsing sentence: ", sent, type(sent)
        sent = sent.split()
        try:
            # parsing the sentence with the given parser
            # and appending it to the bigger list (all_trees_from_all_sentences)
            all_trees_from_all_sentences.append(build_tree_variations(parser, sent))
            if DEBUG: print "parsing of sentence: ", sent, "DONE"
        except ValueError as e:
            print "\nERROR:", sent, "couldn't be parsed\nReason: %s" % e
    # printing the trees onto the command line with pretty print
    for tree_variations in all_trees_from_all_sentences:
        # extracting the data from the given tuple
        (sent, tree_variations) = tree_variations
        sent = " ".join(sent)
        print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
        print "sentence: '%s'" % sent
        v_enum = 1
        for tree in tree_variations:
            print "version %i" % v_enum
            # http://www.nltk.org/howto/tree.html
            tree.pretty_print(unicodelines=True, nodedist=4)
            v_enum += 1
    print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    # compiling the .tex file
    if out_file:
        write_out_to_tex(all_trees_from_all_sentences, out_file, grammar)
    else:
        print "(no output file selected. write '-o outfile.tex' as an argument when " \
            "running the script if you want to generate a .tex file)"
if __name__ == "__main__":
   main()
```

File: grammar.txt

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

```
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas' | 'elk' | 'shorts'
V -> 'shot'
P -> 'in'

File: prob_grammar.txt
S -> NP VP [1.0]
PP -> P NP [1.0]
NP -> Det N [0.4] | Det N PP [0.3] | 'I' [0.3]
VP -> V NP [0.24] | VP PP [0.76]
Det -> 'an' [0.5] | 'my' [0.5]
N -> 'elephant' [0.3] | 'pajamas' [0.2] | 'elk' [0.3] | 'shorts' [0.2]
V -> 'shot' [1.0]
P -> 'in' [1.0]

File: prob_out.tex
\documentclass[12pt]{article}
\usepackage{tikz}
\usepackage{tikz-qtree}
\title{Programiertechniken in der Computerlinguistik II: Assignment 6}
\author{Linus Manser (13-791-132), Roland Benz (97-923-163)}
\date{Abgabe: 2016/06/08}
\begin{document}
\maketitle
\section{Grammar}
\begin{verbatim}
Grammar with 15 productions (start state = S)
    S -> NP VP [1.0]
    PP -> P NP [1.0]
    NP -> Det N [0.4]
    NP -> Det N PP [0.3]
    NP -> 'I' [0.3]
    VP -> V NP [0.249]
    VP -> VP PP [0.751]
    Det -> 'an' [0.5]
    Det -> 'my' [0.5]
    N -> 'elephant' [0.3]
    N -> 'pajamas' [0.2]
    N -> 'elk' [0.3]
    N -> 'shorts' [0.2]
    V -> 'shot' [1.0]
    P -> 'in' [1.0]
\end{verbatim}
\section{Syntax trees}
\subsection{'I shot an elephant in my pajamas'}
\begin{enumerate}
\item\begin{center}
\Tree [.S
        [.NP I ]
        [.VP
          [.VP [.V shot ] [.NP [.Det an ] [.N elephant ] ] ]
          [.PP [.P in ] [.NP [.Det my ] [.N pajamas ] ] ] ] ]
\end{center}
\end{enumerate}
\subsection{'I shot an elk in my shorts'}
\begin{enumerate}
\item\begin{center}
\Tree [.S
        [.NP I ]
        [.VP
          [.VP [.V shot ] [.NP [.Det an ] [.N elk ] ] ]
          [.PP [.P in ] [.NP [.Det my ] [.N shorts ] ] ] ] ]
\end{center}
\end{enumerate}
\end{document}

File:sentence.txt
I shot an elephant in my pajamas
I shot an elk in my shorts

File: out.pdf
```

1 Grammar

```
Grammar with 15 productions (start state = S)
    S -> NP VP
    PP -> P NP
    NP -> Det N
    NP -> Det N PP
    NP -> 'I'
    VP -> V NP
    VP -> VP PP
    Det -> 'an'
    Det -> 'my'
    N -> 'elephant'
    N -> 'pajamas'
    N -> 'elk'
    N -> 'shorts'
    V -> 'shot'
    P -> 'in'
```

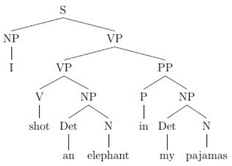
PCL II – Zusammenfassung Code

Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

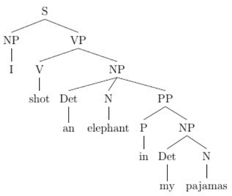
2 Syntax trees

2.1 'I shot an elephant in my pajamas'

1.



2.



File: prob_out.pdf

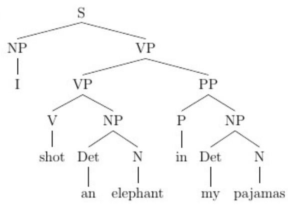
1 Grammar

```
Grammar with 15 productions (start state = S)
    S -> NP VP [1.0]
    PP -> P NP [1.0]
    NP -> Det N [0.4]
    NP -> Det N PP [0.3]
    NP -> 'I' [0.3]
    VP -> V NP [0.249]
    VP -> VP PP [0.751]
    Det -> 'an' [0.5]
    Det -> 'my' [0.5]
    N -> 'elephant' [0.3]
    N -> 'pajamas' [0.2]
    N -> 'elk' [0.3]
    N -> 'shorts' [0.2]
    V -> 'shot' [1.0]
    P -> 'in' [1.0]
```

2 Syntax trees

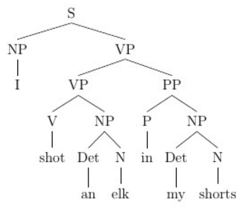
2.1 'I shot an elephant in my pajamas'

1.



2.2 'I shot an elk in my shorts'

1.



PCL II – Zusammenfassung Code

Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)

PCL II – Zusammenfassung Code
Linus Manser (lmanser, 13-791-132) und Roland Benz (rolben, 97-923-163)