Referenzkarte PCL I – Version 5.1 – 28. Dezember 2014

Sequenzielle Datentypen: Listen (lesen/schreiben), Tupel, Strings (nur lesen)

l = range(n)	Liste von n Zahlen (von 0 bis n-1)
l = l1 + l2	Verknüpfung von Sequenzen
l[i:j]	Teilsequenz von l[i] bis l[j-1]
l[:k]	die ersten k Elemente der Sequenz
l[-k:]	die letzten k Elemente der Sequenz
l[i:j] = ['bla']	ersetzen von l[i] bis l[j-1] mit der Liste ['bla'] (nur für Listen)
len(l)	Anzahl Elemente
max(l) ; min(l)	max./min. Wert (strings: alph. geordnet)
l.count('the')	Anzahl auftreten von 'the' in der Liste l
l.index('dog')	erster Index von 'dog' , oder Fehler falls ‘dog’ nicht in l
x in l ; x not in l	ist x (nicht) in l enthalten? (Resultat True oder False)
l.append('bla')	anfügen von 'bla' am Ende von l (nur für Listen)
l.insert(i,x)	füge x an der Position i in l ein (nur für Listen)
l.remove(x)	entferne das erste x in l (nur für Listen)
l.reverse()	Reihenfolge von l umkehren (nur für Listen)
l.sort()	sortieren, zuerst Zahlen, dann Buchstaben (nur für Listen)

Dictionaries / Hashes (d) mit Schlüssel-Wert-Paaren

d = {}	leeres Dictionary erzeugen
d = {'a':34, 'to':23}	Dictionary mit zwei Einträgen erzeugen
d['the']	gibt den Wert von 'the' zurück (in diesem Fall 34)
len(d)	Anzahl Schlüssel
d.copy()	erzeugt eine Kopie von d
d.items()	Liste aller Einträge (Einträge sind 2-Tupel)
d.keys()	Liste aller Schlüsseln
d.values()	Liste aller Werte
x in d	True falls der Schlüssel x in d enthalten ist
del d[‘the’]	entfernt Schlüssel ‘the’ und zugehörigen Wert aus d

Reguläre Ausdrücke – Syntax

^	Startposition innerhalb des Strings
$	Endposition eines Strings
x{m,n}	x kommt mindestens m mal vor, aber nicht mehr als n mal
(x)	gruppiert den Ausdruck x. Kann über \i aufgerufen werden (i von 1 bis 9)
\d	[0-9]
\D	[^0-9]
\w	[A-Za-z0-9_]
\W	[^A-Za-z0-9_]
\s	[\t\r\n\v\f] ← *Vorsicht: inklusive Leerzeichen!*
\S	[^ \t\r\n\v\f]
\b	Wortgrenzen

Konversion sequenzielle Datentypen: strings (s), lists (l), tuples (t)

l = list(t) ; l = list(s) ; t = tuple(l) ; t = tuple(s)

Input/Output

```
import sys                                              ## Systemmodul, u.a. mit Kommandozeil.-Funktionen

def main():
        filename = sys.argv[1]                          ## Dateiname auslesen aus Kommandozeile
        infile = open(filename, 'r')                    ## die Datei zum Schreiben öffnen

        my_outfilename = filename + '.letter_a_words'   ## Name für Ausgabedatei
        outfile = open(my_outfilename, 'w')             ## Datei zum Schreiben öffnen
        for line in infile:                             ## Schleife für jede Zeile in der Datei
                # <Do something>

        infile.close()                                  ## Schliessen der Dateien
        outfile.close()
if __name__ == '__main__':                              ## Standard main-Boilerplate mit Funktion-Aufruf
        main()
```

HTML

```
import nltk
from urllib import urlopen
html = urlopen(given_url).read() ## öffne den Quellcode des Dokuments mit der Adresse given_url zum Lesen
text = nltk.clean_html(html)     ## nimmt den Quelltext und extrahiert den Text daraus
```

Anderes

from __future__ import division	nötig, falls man zwei Integer teilt und eine rationale Zahl erhalten will
set(list)	erzeugt ein Set einer Liste, in dem Duplikate zusammengefasst werden
l = nltk.word_tokenize(given_string)	tokenisiert given_string in eine Liste
foobar".find("bar")	sucht das erste Vorkommen von "bar" in "foobar" und liefert den ersten Index der Sequenz zurück

Reguläre Ausdrücke – Modul re

import re

l = re.findall(pattern, string)	Liste mit allen gefunden Gruppen
m = re.search(pattern, string)	
m.group()	gibt alle Übereinstimmungen zurück
m.group(1)	gibt nur die erste Übereinstimmung zurück
s = re.sub(pattern, replacement, given_string)	sucht pattern in given_string und ersetzt es durch replacement
re.split(pattern, given_string)	trennt given_string nach pattern in eine Liste auf

*Hinweis: Flag (?u) aktiviert Unicode Kategorie für \w and \b → pattern=ur'(?u)(*regex*)'*

Liste vs. Listenkomprehension

```
word_list = ["foo", "bar", "lorem", "ipsum"] ## Liste mit Stringliteralen
four_chars_words = [w for w in word_list if len(w)==4] ## Listenkomprehension
```

Tastatureingaben

var = raw_input()	Speichert die Eingabe in einer String-Variable
integer = int(s)	Konvertiert einen String in einen Integer

Grundlegende Syntax

```
if expr: ## if-Abfrage
   statements
elif expr:
   statements
else:
   statements
while expr: ## while Schleife
   statements

for w in l: ## for Schleife
   statements
## Funktion mit Argument x
## und Rückgabewert x+1
def function_name(x):
   return x+1
```

Import von NLTK: import nltk

NLTK – Korpora	
from nltk.corpus import corpus_name	importieren des Korpus corpus_name aus dem Modul nltk.corpus
corpus_name.fileids()	erzeugt eine Liste mit allen Datei-ID's des Korpus
corpus_name.raw(fileid)	erzeugt einen unicode-String aus dem Inhalt der Datei mit der ID fileid
corpus_name.words(fileid)	erzeugt eine „Liste“, bzw. ein listenähnliches Objekt aus dem Text der Datei mit der ID fileid. Kann mit list() in eine normale Liste umgewandelt werden. (List[**w**])
corpus_name.sents(fileid)	erzeugt eine zweidimensionale „Liste“ aus dem Text, in der jeder Satz eine Element ist, mit seinen Wörtern wiederum als Liste. (List[**s**][w])
corpus_name.paras(fileid)	erzeugt eine dreidimensionale „Liste“ aus dem Text, in der die satzbildenden Listen zusätzlich nach Absätzen zusammengefasst sind (List[**p**][s][w])
Kategorisierte oder kontext-getaggte Korpora haben zusätzlich die Methode .categories():	
corpus_name.categories()	erzeugt eine Liste mit allen Kategorien des Korpus.
corpus_name.categories(fileid)	erzeugt eine Liste mit den Kategorien, die der Korpus-Datei mit der ID fileid zugeordnet sind.
Der Brown-Korpus ist mit Part-of-Speech-Tags (POS) getaggt:	
brown.tagged_words(fileid)	erzeugt eine Liste von Wort-POS-Tupel (word, pos_tag) für jedes Wort
NLTK – Frequency distribution classes	
fdist = nltk.FreqDist(samples)	erzeugt ein dictionary-ähnliches Objekt: jedes Element (=event) der samples-Sequenz ist ein Schlüssel und die Frequenz des Elements in sample ist der Wert
fdist.N()	Anzahl Elemente in samples
fdist.max()	Event mit grösstem Zähler
fdist.keys()	Events sortiert nach ihrer Frequenz
fdist.tabulate()	tabellarische Anordnung der Frequenz Anordnung
cfdist = nltk.ConditionalFreqDist(pairs)	creates frequency distributions of events conditioned on a condition: pairs is a list of tupels of the form (condition, event)
cfdist.conditions()	Alphabetisch sortierte Liste der Bedingungen (conditions)
cfdist[condition]	Dictionary mit Häufigkeiten von allen events gegeben eine Bedingung (condition)
cfdist[condition][sample]	Häufigkeit eines bestimmten events gegeben eine Bedingung (condition)
cfdist.tabulate()	tabellarische Ausgabe von Bedingungen und Events
NLTK – Bigramme erzeugen	
bi_list = nltk.bigrams(list)	erzeugt ein Generatorobjekt mit Bigramm-Tupeln aus den Elementen der Liste list. Umwandlung in eine Liste mit list(). Beispiel: **>>>** bigramGen = bigrams(["Lorem", "Ipsum", "Dolor", "Sit", "Amet"]) **>>>** print list(bigramGen) [('Lorem', 'Ipsum'), ('Ipsum', 'Dolor'), ('Dolor', 'Sit'), ('Sit', 'Amet')]