Reference sheet PCL I – Version 5.1 – December 28th, 2014

Sequential data types: Lists (read/write), Tupel, Strings (read-only)

l = range(n)	list of n integers (from 0 to n-1)
l = l1 + l2	concatenation of sequences
l[i:j]	slicing from l[i] till l[j-1]
l[:k]	first 5 elements
l[-k:]	last 5 elements
l[i:j] = ['bla']	replace from l[i] till l[j-1] with ['bla'] (only with lists)
len(l)	number of elements
max(l) ; min(l)	max/min value (strings: alph. order)
l.count('the')	number of occurrences of 'the'
l.index('dog')	first index of 'dog' , or error if ‘dog’ is not in l
x in l ; x not in l	is x (not) a member of l? (evaluates to True or False)
l.append('smthg')	append 'smthg' at the end of l (only with lists)
l.insert(i,x)	insert x at position i in l (only with lists)
l.remove(x)	remove the first x in l (only with lists)
l.reverse()	reverse l (only with lists)
l.sort()	sort, first digits then chars (only with lists)

Dictionaries / Hashes (d) with key-value pairs

d = {}	create empty dictionary
d = {'a':34, 'to':23}	create dictionary
d['the']	get value of 'the'
len(d)	number of keys
d.copy()	create copy of d
d.items()	list of all items (items are 2-tuples)
d.keys()	list of all keys
d.values()	list of all values
x in d	true if there is a key x in d
del d[‘the’]	delete key-value pair from dictionary d

Regular expressions – Syntax

^	matches the starting position within the string
$	matches the ending position of the string
x{m,n}	matches the preceding expression x at least m and not more than n times
(x)	groups expression x. The string matched within the parentheses can be referenced later by \n (n from 1 to 9)
\d	[0-9]
\D	[^0-9]
\w	[A-Za-z0-9_]
\W	[^A-Za-z0-9_]
\s	[\t\r\n\v\f] ← Beware: White space is in too.
\S	[^ \t\r\n\v\f]
\b	Word boundaries

Conversion of sequential data types: strings (s), lists (l), tuples (t)

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

Remark: dictionary entry {'x':4}, left → key : right → value

Input/Output

```
import sys                                              ## necessary for command line

def main():
        filename = sys.argv[1]                          ## get the filename
        infile = open(filename, 'r')                    ## open a file for reading

        my_outfilename = filename + '.letter_a_words'   ## create a name for the output file
        outfile = open(my_outfilename, 'w')             ## open the output file for writing
        for line in infile:                             ## loop over each line in the file
                # <Do something>

        infile.close()                                  ## close the files
        outfile.close()
if __name__ == '__main__':                              ## This is the standard boilerplate
        main()                                          ## to call the function main
```

HTML

```
import nltk
from urllib import urlopen
html = urlopen(given_url).read() ## get the html-code from a given_url to read
text = nltk.clean_html(html)     ## takes an HTML string and returns only the text in it
```

Miscellaneous

from __future__ import division	necessary if you want to divide two integers and get a rational number
set(list)	creates a set of a list, but all duplicates are collapsed together
nltk.word_tokenize(given_string)	Tokenizes the given_string in a list
foobar".find("bar")	get the right index values to use for slicing the text raw of type string

Basic syntax

```
if expr: ## if conditional

   statements
elif expr:

   statements
else:
   statements
while expr: ## while loop

   statements

for w in l: ## for loop
   statements
## function with argument x
## and return value x+1
def function_name(x):

   return x+1
```

Regular expressions – module re

import re	
l = re.findall(pattern, string)	list of all matched groups
m = re.search(pattern, string)	
m.group()	returns whole match.
m.group(1)	returns the first matched group
s = re.sub(pattern, replmt, given_string)	search the pattern in the given_string, replace it and return a string
re.split(pattern, given_string)	split a given_string by the occurrences of pattern into a list

*Remark: Flag (?u) activates Unicode categories for \w and \b → pattern=ur’(?u)(*regex*)’*

List vs. list comprehension

word_list = ["foo", "bar", "lorem", "ipsum"] ## List with string literals

four_chars_words = [w for w in word_list if len(w)==4] ## list comprehension

Input processing

var = raw_input()	Save the user input in a variable as a string
integer = int(s)	Convert a string s into an integer

Import from NLTK: import nltk

NLTK – Corpora	
from nltk.corpus import corpus_name	Imports corpus corpus_name from the module nltk.corpus
corpus_name.fileids()	creates a list with all id's of the files that make up the corpus
corpus_name.raw(fileid)	creates a unicode string from the content of the file with the id fileid
corpus_name.words(fileid)	creates a „list“/list-like object from the text of the file with the id fileid. Can be converted into a normal list with list(). (List[**w**])
corpus_name.sents(fileid)	creates a two-dimensional „list“ from the text, in which every sentence is itself a list of its tokens. (List[**s**][w])
corpus_name.paras(fileid)	creates a three-dimensional „list“ from the text, which additionally groups the sentence-forming lists into paragraphs. (each paragraph is a list of sentences and each sentence is a list of words (3D-list)) (List[**p**][s][w])
Categorized or context-tagged corpora have the additional method .categories()	
corpus_name.categories()	creates a list of all categories featured in the corpus
corpus_name.categories(fileid)	creates a list of all categories that are associated with the corpus file having the id fileid
The brown corpus has been part-of-speech-tagged (pos)	
brown.tagged_words(fileid)	creates a list where a token-pos tuples (token, pos_tag) corresponds with each token of the file with the ID fileid
NLTK – Frequency distribution classes	
fdist = nltk.FreqDist(samples)	creates a dictionary-like object: each different element (=event) of the sequence samples is a key and the frequency of the element in samples is its value
fdist.N()	total number of samples
fdist.max()	event with the greatest count
fdist.keys()	events sorted in decreasing frequency
fdist.tabulate()	tabulate the frequency distribution
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()	alphabetically sorted list of conditions
cfdist[condition]	the frequency distribution for this condition
cfdist[condition][sample]	frequency for the given event for this condition
cfdist.tabulate()	tabulate the conditional frequency distribution
NLTK – Bigram generation	

bi_list = nltk.bigrams(list)	creates a generator object with bigram tuples generated from the elements of the list. Conversion into a list with list().
	Example:
	>>> bigramGen = bigrams(["Lorem", "Ipsum", "Dolor", "Sit", "Amet"])
	>>> print list(bigramGen)
	[('Lorem', 'Ipsum'), ('Ipsum', 'Dolor'), ('Dolor', 'Sit'), ('Sit', 'Amet')]