Permissions

Unix/Linux permissions are shown with "ls -l" or "ls -al" (Option a für alle, auch versteckte Dateien):

```
-rwxr-x---  1 tom      acct 7538 Dec  9 08:48 execfile
```

The very first part of the permissions tells you whether what you are seeing is an ordinary file, a directory, or something else. Another way to find out what you are looking at is to use the "file" command. For example, "file *"...

This file is owned by tom, and has the group ownership of "acct". The owner (tom) has full permissions- he can read this file (display it, read it into an editor, copy it somewhere else), write it (replace its contents with something else), and execute it.
Anyone in the group "acct" can read and execute it, but can't write it.
Anyone else can't do anything.

The x or "execute" permission needs some explanation for those from the Windows world: Unix programs don't use extensions like "bat" or "exe" to be executable- instead, the "x" permissions settings determines this. Of course, the file still has to be a legitimate program- but a legitimate program will not run until it is marked as "x".

Erzeuge leere Datei oder überschreibe bestehende:$ >result.txtA

Directories

Permissions on directories can be a little confusing. Read permission ("r") means that you can list the directory- for example, use "ls" on it. But that's all it means: if you only have read permission, you can't cd to that directory, you can't copy files into that directory, and you can't even do an "ls -l" on it. Note that it doesn't help if files or directories under that would be accessible by you: you won't be able to get to them. Write permission ("w") allows you to create new files and to remove files. Normally, write permission is all you need to remove a file- you might not have any permissions on the file itself, but if you have write permission on the directory and the text bit is not set (see below) you will be able to remove the file whether or not you have any permissions on the file itself, and no matter who owns it. However, you will need at least "x" permission also, because "x" is what lets you "search" a directory. You might also think of it as "use its contents". You need "x" permissions on the directory to read a file in that directory, to copy it elsewhere and (as noted above) to delete it.

The "id" command (most Unix/Linux) will show you what groups you belong to:
uid=1000(benzro) gid=1000(benzro) groups=1000(benzro),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),115(lpadmin),125(sambashare)

The command "ps auxww" provides complete information about the processes, including all parameters.

The locate command is often the simplest and quickest way to find the locations of files and directories on Linux. The command locate file1 dir1 would list the absolute paths of all files named *file1 (or containing file1)* and all directories named *dir1* (or containing dir1) for which the user had access permission. The specificity of locate can be increased by using it together with *wildcards* or other *regular expressions*. The command locate "*.png" | less uses the star wildcard to display all files on the system that have the *.png file extension.*

benzro@benzro-eMachines-E625:~$ grep --help

Usage: grep [OPTION]... PATTERN [FILE]...

Search for PATTERN in each FILE or standard input.
PATTERN is, by default, a basic regular expression (BRE).

Regexp selection and interpretation:
-E, --extended-regexp PATTERN is an extended regular expression (ERE)
-F, --fixed-strings PATTERN is a set of newline-separated fixed strings
-G, --basic-regexp PATTERN is a basic regular expression (BRE)
-P, --perl-regexp PATTERN is a Perl regular expression
-e, --regexp=PATTERN use PATTERN for matching
-f, --file=FILE obtain PATTERN from FILE

-i, --ignore-case ignore case distinctions
-w, --word-regexp force PATTERN to match only whole words
-x, --line-regexp force PATTERN to match only whole lines
-z, --null-data a data line ends in 0 byte, not newline

Miscellaneous:
-s, --no-messages suppress error messages
-v, --invert-match select non-matching lines
-V, --version display version information and exit
--help display this help text and exit

Context control:
-B, --before-context=NUM print NUM lines of leading context
-A, --after-context=NUM print NUM lines of trailing context
-C, --context=NUM print NUM lines of output context
-NUM same as --context=NUM
--color[=WHEN],
--colour[=WHEN] use markers to highlight the matching strings;
WHEN is 'always', 'never', or 'auto'
-U, --binary do not strip CR characters at EOL (MSDOS/Windows)
-u, --unix-byte-offsets report offsets as if CRs were not there
(MSDOS/Windows)

Output control:
-m, --max-count=NUM stop after NUM matches
-b, --byte-offset print the byte offset with output lines
-n, --line-number print line number with output lines
--line-buffered flush output on every line
-H, --with-filename print the file name for each match
-h, --no-filename suppress the file name prefix on output
--label=LABEL use LABEL as the standard input file name prefix
-o, --only-matching show only the part of a line matching PATTERN
-q, --quiet, --silent suppress all normal output
--binary-files=TYPE assume that binary files are TYPE;
TYPE is 'binary', 'text', or 'without-match'
-a, --text equivalent to --binary-files=text
-I equivalent to --binary-files=without-match
-d, --directories=ACTION how to handle directories;
ACTION is 'read', 'recurse', or 'skip'
-D, --devices=ACTION how to handle devices, FIFOs and sockets;
ACTION is 'read' or 'skip'
-r, --recursive like --directories=recurse
-R, --dereference-recursive likewise, but follow all symlinks
--include=FILE_PATTERN search only files that match FILE_PATTERN
--exclude=FILE_PATTERN skip files and directories matching FILE_PATTERN
--exclude-from=FILE skip files matching any file pattern from FILE
--exclude-dir=PATTERN directories that match PATTERN will be skipped.
-L, --files-without-match print only names of FILEs containing no match
-l, --files-with-matches print only names of FILEs containing matches
-c, --count print only a count of matching lines per FILE
-T, --initial-tab make tabs line up (if needed)
-Z, --null print 0 byte after FILE name

grep-Terminalanzeigen:
::::::::::::::::::::::::::::::::
-o (=nur Treffer ausgeben)
--colour (=Treffer in Zeile einfärben)
-n (=Zeilennummer ausgeben)
-v (=inverse match)
-h (=keine Dateiangaben)

grep anhalten / wieder / beenden:
::::::::::::::::::::::::::::::::
--> Ctrl+s / Ctrl+q / Ctrl+z oder Ctrl+c

REGULAR EXPRESSIONS

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using

Repetition

A regular expression may be followed by one of several repetition operators:
? The preceding item is optional and matched at most once.

various operators to combine smaller expressions.

grep understands three different versions of regular expression syntax: “basic” (BRE), “extended” (ERE) and “perl” (PRCE). In GNU grep, there is no difference in available functionality between basic and extended syntaxes. In other implementations, basic regular expressions are less powerful. The following description applies to extended regular expressions; differences for basic regular expressions are summarized afterwards. Perl regular expressions give additional functionality, and are documented in pcresyntax(3) and pcrepattern(3), but only work if pcre is available in the system.

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any meta-character with special meaning may be quoted by preceding it with a backslash.

The period . matches any single character.

Character Classes and Bracket Expressions

A bracket expression is a list of characters enclosed by [and]. It matches any single character in that list; if the first character of the list is the caret ^ then it matches any character not in the list. For example, the regular expression [0123456789] matches any single digit.

Within a bracket expression, a range expression consists of two characters separated by a hyphen. It matches any single character that sorts between the two characters, inclusive, using the locale's collating sequence and character set. For example, in the default C locale, **[a-d] is equivalent to [abcd].**

Many locales sort characters in dictionary order, and in these locales [a-d] is typically not equivalent to [abcd]; it might be equivalent to [aBbCcDd], for example. To obtain the traditional interpretation of bracket expressions, you can use the C locale by setting the LC_ALL environment variable to the value C.

Finally, certain named classes of characters are predefined within bracket expressions, as follows. Their names are self explanatory, and they are [:alnum:], [:alpha:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], and [:xdigit:]. For example, [[:alnum:]] means the character class of numbers and letters in the current locale. In the C locale and ASCII character set encoding, this is the same as **[0-9A-Za-z]**.

(Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket expression.) **Most meta-characters lose their special meaning inside bracket expressions. To include a literal] place it first in the list. Similarly, to include a literal ^ place it anywhere but first. Finally, to include a literal - place it last.**

Anchoring

The caret ^ and the dollar sign $ are meta-characters that respectively match the empty string at the beginning and end of a line.

The Backslash Character and Special Expressions

The symbols \< and \> respectively match the empty string at the beginning and end of a word. The symbol \b matches the empty string at the edge of a word, and \B matches the empty string provided it's not at the edge of a word. The symbol \w is a synonym for [_[:alnum:]] and \W is a synonym for [^_[:alnum:]].

*** The preceding item will be matched zero or more times.**
+ The preceding item will be matched one or more times.
{n} The preceding item is matched exactly n times.
{n,} The preceding item is matched n or more times.
{,m} The preceding item is matched at most m times. This is a GNU extension.
{n,m} The preceding item is matched at least n times, but not more than m times.

Concatenation

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated expressions.

Alternation

Two regular expressions may be joined by the **infix operator |;** the resulting regular expression matches **any string matching either alternate expression.**

Precedence

Repetition takes precedence over concatenation, which in turn takes precedence over alternation. A whole expression may be enclosed in parentheses to override these precedence rules and form a subexpression.

Back References and Subexpressions

The back-reference \n, where n is a single digit, matches the substring previously matched by the nth parenthesized subexpression of the regular expression.

Basic vs Extended Regular Expressions

In basic regular expressions the **meta-characters** ?, +, {, |, (, and) lose their special meaning; instead **use the backslashed versions \?, \+, \{, \|, \(, and \)**.

Traditional egrep did not support the { meta-character, and some egrep implementations support \{ instead, so portable scripts should avoid { in grep -E patterns and should use [{] to match a literal {.

GNU grep -E attempts to support traditional usage by assuming that { is not special if it would be the start of an invalid interval specification. For example, the command grep -E '{1' searches for the two-character string {1 instead of reporting a syntax error in the regular expression. POSIX allows this behavior as an extension, but portable scripts should avoid it.

Durchsuchen von Datei-namen und Ordner-namen im gesamten Filesystem:

```
benzro@benzro-eMachines-E625:~$ locate -i calendar | wc
   562    567   32221
benzro@benzro-eMachines-E625:~$ locate -i calendar | grep -n  doc
35:/usr/local/texlive/2015/texmf-dist/doc/generic/pgf/text-en/pgfmanual-en-library-calendar.tex
36:/usr/local/texlive/2015/texmf-dist/doc/generic/pgf/text-en/pgfmanual-en-pgfcalendar.tex
37:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar
38:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/Changes
39:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/README
40:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/pst-calendar-docDE.ltx
41:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/pst-calendar-docDE.pdf
42:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/pst-calendar-docDE.tex
43:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/pst-calendar-docFR.pdf
44:/usr/local/texlive/2015/texmf-dist/doc/latex/pst-calendar/pst-calendar-docFR.tex
45:/usr/local/texlive/2015/texmf-dist/doc/metapost/bbcard/README.calendar
144:/usr/share/doc/libakonadi-calendar4
145:/usr/share/doc/bsdmainutils/calendarJudaic.py.gz
146:/usr/share/doc/libakonadi-calendar4/changelog.Debian.gz
147:/usr/share/doc/libakonadi-calendar4/copyright
148:/usr/share/doc/texlive-doc/generic/pgf/text-en/pgfmanual-en-library-calendar.tex.gz
149:/usr/share/doc/texlive-doc/generic/pgf/text-en/pgfmanual-en-pgfcalendar.tex.gz
150:/usr/share/doc/texlive-doc/latex/pst-calendar
151:/usr/share/doc/texlive-doc/latex/pst-calendar/README
152:/usr/share/doc/texlive-doc/latex/pst-calendar/pst-calendar-docDE.ltx.gz
153:/usr/share/doc/texlive-doc/latex/pst-calendar/pst-calendar-docDE.pdf
154:/usr/share/doc/texlive-doc/latex/pst-calendar/pst-calendar-docDE.tex.gz
155:/usr/share/doc/texlive-doc/latex/pst-calendar/pst-calendar-docFR.pdf
156:/usr/share/doc/texlive-doc/latex/pst-calendar/pst-calendar-docFR.tex.gz
benzro@benzro-eMachines-E625:~$ locate -i calendar | grep -n  doc | wc
    24      24    1734
benzro@benzro-eMachines-E625:~$ locate -i calendar | grep -n -c doc
24
```

Durchsuchen der Datei-inhalte im lokalen Verzeichnis:

Suche in allen Dateien (*.*) nach dem Inhalt „Dokument“

```
benzro@benzro-eMachines-E625:~$ grep "Document" *.*
callPCL.py:chdir("/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Übungen/PCL_uebung"+intUebung+"_python/")
```

Gib nur den Dateinamen des Treffers aus (Option -l):

```
benzro@benzro-eMachines-E625:~$ grep -l  "Document" *.*
callPCL.py
```

Gib den Treffer in der Datei ohne den Dateinamen aus (Option -h):

```
benzro@benzro-eMachines-E625:~$ grep -h "Document" *.*
chdir("/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Übungen/PCL_uebung"+intUebung+"_python/")
```

Gib den gesamten Inhalt (Pattern leer matched alles) der versteckten Dateien aus (.*)

```
benzro@benzro-eMachines-E625:~$ grep .* | less
```

Ins richtige Verszeichnis wechseln auf Lubuntu Partition:

```
pwd
--> /home/benzro
cd .. (2 mal)
cd media/benzro/OS/Users/benzro/Documents/LubuntuShared/'8) PCL-1'/Übungen/PCL_uebung1_grep

benzro@benzro-eMachines-E625:~$ wc --help
Usage: wc [OPTION]... [FILE]...
  or:  wc [OPTION]... --files0-from=F
Print newline, word, and byte counts for each FILE, and a total line if
more than one FILE is specified.  With no FILE, or when FILE is -,
read standard input.  A word is a non-zero-length sequence of characters
delimited by white space.
```

Dateien Analyse alle Jahrgänge 1930-1939:

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
-->Total Zeilen=Anzahl Artikel + Anzahl Abschnittsanfänge + Anzahl POS-Tags
2741140=923+142101+35+5+24+1+2598051
-->Total an Zeilen:
grep -P '^.' SAC-Jahrbuch_193* | wc
2741140 8222497 137232508
-->Anzahl Artikel:
grep -P -o '^<article' SAC-Jahrbuch_193* | wc
```

Bereinigter vertikalisierter Korpus erzeugen:

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
-->(Entferne 923 Artikelzeilen, 142101 Abschnittszeilen und 75 fehlerhafte POS-tags)
Option -h (=gibt keine Dateiangaben aus)
grep -P -h '\t(([\w\$\.\$,\$\(]+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* > Dump2598051_SAC-Jahrbuch_193x
wc Dump2598051_SAC-Jahrbuch_193x
--> 2598051  7794153 40554002
Die Datei Dump2598051_SAC-Jahrbuch_193x wird für die Aufgaben verwendet
```

923 923 39689
-->Anzahl Abschnittsanfänge :
grep -P -o '^<s lang=' SAC-Jahrbuch_193* | wc
142101 284202 6110343
-->Anzahl erlaubte und unerlaubte POS-Tags :
'\t([\w\W]+)\t' (=\W könnte auch ein \t sein, funktioniert nur, weil jede Zeile nur zwei \t hat und keinen dritten.)
grep -P -h '\t([\w\W]+)\t' SAC-Jahrbuch_193* | wc
grep -P -h '\t(.+)\t' SAC-Jahrbuch_193* | wc
2598116 7794348 128890391
-->Anzahl POS-Tag mit unerlaubtem Zeichen) alleinstehend (Nicht in TAG-Listen):
'\t(\x29)\t' (=Hexadezimaler ASCII Code für ")")
grep -P '\t(\x29)\t' SAC-Jahrbuch_193* | wc
35 105 1428
-->Anzahl Pos-Tag mit unerlbauten Zeichen ", &, ', ` alleinstehend (Nicht in TAG-Listen):
'\t((\x22+)|(\x26+)|(\x27+)|(\x60+))\t' (=Hexadezimaler ASCII Code für die Zeichen ", &, ', `)
grep -P '\t((\x22+)|(\x26+)|(\x27+)|(\x60+))\t' SAC-Jahrbuch_193* | wc
5 15 205
-->Anzahl Pos-Tag mit unerlbautem Zeichen ":" alleinstehend (Nicht in TAG-Listen):
grep -P -h '\t(:)\t' SAC-Jahrbuch_193* | wc -l
24
-->Anzahl Pos-Tag mit unerlbautem Zeichen "IN/that"(Nicht in TAG-Listen):
grep -P -h '\t(IN/that)\t' SAC-Jahrbuch_193* | wc -l
1
-->Anzahl erlaubte Pos-Tags
(\w+:\w+(:\w+)?) (=Italienische Tags mit einem oder zwei Doppelpunkten)
grep -P -h '\t(([\w\$\.\$,\$\()]+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* | wc -l
2598051

In wie vielen Wörtern kommt der Buchstabe "r" (klein oder gross) vor?
Wörter stehen in der ersten Spalte, --> Suche im Bereich: ^()\t
Erste Annäherung:
grep -P -i --colour '^([\wäöü]*[r][\wäöü]*)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->645098
Exakte Lösung:
grep -P --colour '^([^\t]*[rR]+[^\t]*)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
grep -P -i --colour '^([^\t]*[r]+[^\t]*)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->708311
In 708311 Wörtern.
Regex: **[^\t] der Hut am Anfang negiert den Klasseninhalt, also kein Tabulator**
Option -i und [r] hat denselben Effekt wie [Rr]

In wie vielen Lemmata kommt die Endung "-er" vor?
Lemmata stehen in der letzten Spalte, --> Suche im Bereich \t()$
Anzahl Zeilen mit "er" in der letzten Spalte:
grep -P -i --colour '\t([^\t]*er[^\t]*)$' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->284272
Anzahl Zeilen mit "er" am Ende der Zeile
grep -P -i --colour '\t([^\t]*er)$' Dump2598051_SAC-Jahrbuch_193x | wc -l
grep -P -i --colour 'er$' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->147174
In 147174 Lemmata

Ausgabe von Zeilen einer Datei

sed [OPTION]... {script-only-if-no-other-script} [input-file]...
Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more effi- cient.
To print a specific line (line 5) from a file
sed -n 5p <filename>
Print all the lines between 10 and 20 of a file
sed -n '10,20p' <filename>

Vergleichen von zwei Dateien:

This tells you the lines of `file1` which do *not* (`-v`) appear in `file2`
grep -nFvf testdump1 testdump2

This tells you the lines which differ in `file1` and `file2`
diff -y testdump1 testdump2
Die Zeilen, die mit einer spitzen, öffnenden Klammer „<" beginnen, sind nur in der ersten Datei vorhanden, diejenigen, die mit einer spitzen, schließenden Klammer „>" beginnen, sind nur in der zweiten Datei vorhanden. Zeilen, die in beiden Dateien gleich sind, werden nicht ausgegeben. Die einzelnen Blöcke werden durch so genannte *change commands* („Änderungsbefehle") getrennt, die angeben, welche Aktion (Zeilen hinzufügen - *a*, ändern - *c* oder entfernen - *d*) in welchen Zeilen ausgeführt werden soll.

grep -P -h '\t([\w$.,(]+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* | wc
2598052 7794157 40554052

Die mehrfachen \ und $ können weggelassen werden:
grep -P -h '\t([\w\$\.\$,\$\(]+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* > testdump1
2598052 7794157 40554052

Gruppen-Klammern in Klasse haben keinen Einfluss. Einziger Unterschied zum testdump1 sind 35 zusätzliche Klammern „)":
grep -P -h '\t([(\w)(\$\.)(\$,)(\$\()]+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* >testdump2
2598087 7794262 40554290

210 alleinstehende „ ," „ (" „ PP$ " sind hier nicht enthalten
grep -P -h '\t((\w)+|(\$\.)+|(\$,)+|(\$\()+|(\w+:\w+(:\w+)?))\t' SAC-Jahrbuch_193* > testdump3
2597842 7793527 40552666

Anzeige der Zeilen von testdump2 nicht in testdump1:
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Übungen/PCL_uebung1_grep$ **grep -nFvf testdump1 testdump2**
707333:)))
877019:)))
usw.

benzro@benzro-eMachines-E625:~$**sed -n 707332,707336p testdump2**
permitting VVG permit
)))
. SENT .
» (»
2. ADJA @ord@

Anzeige der Zeilen von testdump1 nicht in testdump2:
benzro@benzro-eMachines-E625:~$ **grep -nFvf testdump2 testdump1**
leere Ausgabe

benzro@benzro-eMachines-E625:~$ **sed -n 707332,707336p testdump1**
permitting VVG permit
. SENT .
» (»
2. ADJA @ord@

Anzeige unterschiedlichen Zeilen: a
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Übungen/PCL_uebung1_grep$ **diff testdump1 testdump2**
707332a707333
>)))
877017a877019
>)))
usw.

benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Übungen/PCL_uebung1_grep$ **diff testdump2 testdump1**
707333d707332
<)))
877019d877017
<)))
usw.

Welches ist das häufigste deutsche Relativpronomen?
POS-Tags stehen in der mittleren Spalte, -->Suche im Bereich \t()\t
Anzahl der deutschen Relativpronomen:
grep -P -i --colour '\t(PRELS|PRELAT)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->16992
Worte sortieren und aggregieren mit sort|uniq -c|sort -n
grep -P -i --colour '\t(PRELS|PRELAT)\t' Dump2598051_SAC-Jahrbuch_193x | sort | uniq -c | sort -n
1 Das PRELS d
1 Dem PRELS d
1 deren PRELS d
2 derer PRELS d

Welche Jahreszahlen kommen im Text vor?
Jahreszahlen stehen in der ersten Spalte, --> Suche im Bereich: ^[^\t]*()[^\t]*\t
grep -P -i '^[^\t]*([0,1,2][1-9]{3})[^\t]*\t' Dump2598051_SAC-Jahrbuch_193x | grep -P -i -o '\t(\w*\W*)*\t' | sort | uniq -c | sort -n
grep -P -i '^[^\t]*([0,1,2][1-9]{3})[^\t]*\t' Dump2598051_SAC-Jahrbuch_193x | grep -P -i -o '\t(.)*\t' | sort | uniq -c | sort -n
1 ADJD
1 A_ord
2 NE
6 V
8 A_qual
11 PRO_card

```
  3 Die       PRELS     d
  9 Der       PRELS     d
 40 dessen    PRELS     d
 50 Was       PRELS     was
 78 welchen   PRELS     welch
111 welchem             PRELS     welch
129 welches  PRELS     welch
282 deren    PRELAT    d
313 welcher  PRELS     welch
314 dessen   PRELAT    d
346 denen    PRELS     d
686 welche   PRELS     welch
749 was      PRELS     was
1054 dem     PRELS     d
1267 den     PRELS     d
1751 das     PRELS     d
3857 der     PRELS     d
-->5948 die  PRELS     d
```

"Was" am Satzanfang mit 50 Treffern und "die" allgemein am häufigsten mit 5948 Treffern

Pipe: **sort | uniq -c | sort -n: Erst wird alphabetisch sortiert, dann gleiche Zeilen aggregiert und die Anzahl hinzugefügt (-c), zuletzt wird numerisch sortiert (-n)**

```
  20        ADJA
  20        NN
  24        CD
  44        N_C
 102        NUM
 103        D_card
 104        A_card
 303        N_P
4319        N_card
4832        CARD
```

Dieser Befehl zeigt die Jahreszahlen an:
Wörter und Pos-Tags stehen in der ersten und zweien Spalte-->Suche im Bereich: ^[^\t]*()[^\t]*\t()\t
grep -P -i '^([0,1,2][1-9]{3})(-[0,1,2][1-9]{3})?\t(((CD)|(NUM)|(N_Card)|(CARD))\t' Dump2598051_SAC-Jahrbuch_193x
Zusätzlich kommen noch römische Jahreszahlen mit POS-Tag A_ord N_C vor. Z.B. XIIe siècle.

Regex: **(-[0,1,2][1-9]{3})? Für Zahlen wie z.B. 1934-1938**

Pipe: **grep -P -i -o '\t(.)*\t' | sort | uniq -c | sort -n entnimmt die gesammte mittlere Spalte sortiert alphabetisch, aggregiert, fügt Anzahl hinzu und sortiert numerisch**

Wieviele Wörter im Text enthalten zwei Vokale (inkl. Umlaute) nacheinander?
Wörter stehen in der ersten Spalte, --> Suche im Bereich: ^[^\t]*()[^\t]*\t
genau zwei:
grep -P -i '^[^\t]*([^\taeiouäöüéàâûô][aeiouäöüéàâûô]{2}[^\taeiouäöüéàâûô])[^\t]*\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->377900
mindestens zwei:
grep -P -i '^[^\t]*([^\taeiouäöüéàâûô][aeiouäöüéàâûô]{2,}[^\taeiouäöüéàâûô])[^\t]*\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->394660
Je nachdem, wie man die Frage interpretiert sind es 377900 oder 394660 Wörter

Welche Wörter gibt es, die genau fünf Buchstaben lang sind, wo sowohl der erste und der letzte Buchstabe ein Vokal sind?
Wörter stehen in der ersten Spalte, --> Suche im Bereich: ^[^\t]*()[^\t]*\t
grep -P -i '^([aeiouäöüéàâûô]([\w-,()\+*$!\]&?%\[£.,:;<>=]{3})[aeiouäöüéàâûô])\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->8404
Es sind 8404 Worte, die teilweise mehrfach auftreten. Eine Liste mit 645 Wörtern erhält man mit folgendem Befehl:
grep -P -i '^([aeiouäöüéàâûô]([\w-,()\+*$!\]&?%\[£.,:;<>=]{3})[aeiouäöüéàâûô])\t' Dump2598051_SAC-Jahrbuch_193x | sort | uniq -c

Wie oft ist "soit" eine Verbform? Wie oft eine Konjunktion?
Wörter und Pos-Tags stehen in der ersten und zweien Spalte-->Suche im Bereich: ^[^\t]*()[^\t]*\t()\t
grep -P -i '^(soit)\t([V])\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->147
grep -P -i '^(soit)\t([^V]+)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->252
cat Dump2598051_SAC-Jahrbuch_193x | grep -Pic '^soit\tC_C'
-->173

Wieviele Wörter gibt es, die mit einem Vokal beginnen und auf "r", "s", "t" oder "w" enden?
Wörter stehen in der ersten Spalte, --> Suche im Bereich: ^[^\t]*()[^\t]*\t
grep -P -i '^[aeiouäöüéàâûô](\w*)[rstw]\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->173748
Bemerkung:
(\w)* deckt die Sonderzeichen nicht ab. (\W)* beinhaltet den \t und kann nicht mit \w in Serie geschaltet werden.
Es wäre möglich, dass noch mehr Worte vorkommen, die Sonderzeichen enthalten [\w-,()&?%/\+*$!\]äöüéàâûô\[£.,:;<>=]*
grep -P -i '^[aeiouäöüéàâûô]([\w-,()&?%/\+*$!\]äöüéàâûô\[£.,:;<>=])*[rstw]\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->183603
Es sind mindestens 183603 und ev. noch einige mehr, die ein weiteres Sonderzeichen enthalten

Welche Verkleinerungsform ist häufiger, "-chen" oder "-lein"?
Wörter und Pos-Tags stehen in der ersten und zweien Spalte-->Suche im Bereich: ^[^\t]*()[^\t]*\t()\t
Verkleinerungsformen kommen bei Nomen vor. Die POS-Tag für deutsche Nomen sind:
NN
NE
Mit dem folgenden Befehl erhält man eine sehr gute Liste mit Wörtern, in denen -lein die Funktion einer Verkleinerung hat
grep -P -i '^[^\t]{3,}(lein)\t(NN|NE)\t' Dump2598051_SAC-Jahrbuch_193x | sort | uniq -c
grep -P -i '^[^\t]{3,}(lein)\t(NN|NE)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->541
Mit -chen ist es etwas schwieriger, da -chen auch bei längeren Wörtern eine andere Funktion als die Verkleinerung einnehmen kann.
grep -P -i '^[^\t]{3,}(chen)\t(NN|NE)\t' Dump2598051_SAC-Jahrbuch_193x | sort | uniq -c
grep -P -i '^[^\t]{3,}(chen)\t(NN|NE)\t' Dump2598051_SAC-Jahrbuch_193x | wc -l
-->3420
Die 3420 müssen noch nach unten korrigiert werden, die 541 sind ziemlich exakt. Es ist sehr wahrscheinlich, dass
-chen in der Funktion einer Verkleinerung mit Faktor 3 bis 5 häufiger auftritt.

Aus Musterlösung:
cat SAC-Jahrbuch_193*_mul_columns.txt | grep -Pc 'NN\t[\wäüö]+lein$
-->296'
cat SAC-Jahrbuch_193*_mul_columns.txt | grep -Pc 'NN\t[\wäüö]+chen$
-->1749'

Wie lautet der Befehl, der die Anzahl der Adjektive für jedes Jahrbuch (in deinem Jahrzehnt) einzeln ausgibt?
POS-Tags stehen in der mittleren Spalte, -->Suche im Bereich \t()\t
Pos-Tags für Adjektive:
deutsch: ADJA und ADJD
französchisch: A
italienisch: ADJ
engisch: JJ und JJR und JJS

grep -P -i -o '\t(ADJA|ADJD|A|ADJ|JJ|JJR|JJS)\t' SAC-Jahrbuch_193*
-->Listet alle Adjektive mit Dateiangabe auf

```
SAC-Jahrbuch_1930_mul_columns.txt:      ADJD
SAC-Jahrbuch_1930_mul_columns.txt:      ADJA
SAC-Jahrbuch_1930_mul_columns.txt:      ADJA
SAC-Jahrbuch_1930_mul_columns.txt:      ADJA
SAC-Jahrbuch_1930_mul_columns.txt:      ADJA
SAC-Jahrbuch_1930_mul_columns.txt:      ADJD
```

grep -P -i -o '^[\w\-\.\:]*\t'
-->Entfernt den POS-Tag, dh. Wählt die erste Spalte mit SAC-Jahrbuch_...:
| sort | uniq -c | sort -n
-->Aggregiert und sortiert

Die Anzahl der Adjektive über die vier Sprachen D,E,F,I aggregiert und für jedes Jahr gruppiert erhält man mit:
grep -P -i -o '\t(ADJA|ADJD|A|ADJ|JJ|JJR|JJS)\t' SAC-Jahrbuch_193* | grep -P -i -o '^[\w\-\.\:]*\t' | sort | uniq -c | sort -n

```
14245 SAC-Jahrbuch_1931_mul_columns.txt:
14435 SAC-Jahrbuch_1936_mul_columns.txt:
14629 SAC-Jahrbuch_1937_mul_columns.txt:
14983 SAC-Jahrbuch_1933_mul_columns.txt:
14999 SAC-Jahrbuch_1930_mul_columns.txt:
15009 SAC-Jahrbuch_1935_mul_columns.txt:
15222 SAC-Jahrbuch_1934_mul_columns.txt:
15288 SAC-Jahrbuch_1939_mul_columns.txt:
15583 SAC-Jahrbuch_1932_mul_columns.txt:
17043 SAC-Jahrbuch_1938_mul_columns.txt:
```

Der tr command findet und ersetzt

Usage: tr [OPTION]... SET1 [SET2]
Translate, squeeze, and/or delete characters from standard input,
writing to standard output.

```
 -c, -C, --complement    use the complement of SET1
 -d, --delete            delete characters in SET1, do not translate
 -s, --squeeze-repeats   replace each input sequence of a repeated character
                         that is listed in SET1 with a single occurrence
                         of that character
 -t, --truncate-set1     first truncate SET1 to length of SET2
    --help     display this help and exit
    --version  output version information and exit
```

Wie viele Types (unterschiedliche Wörter) enthält der Text?
-->Text vertikalisieren mit tr -s '\' '\n' < inputfile > outputfile
tr -s '\' '\n' < SecondVariety.txt > DumpVerticalSecondVariety.txt
-->Interpunktion entfernen
tr -d '.,:;?!-"' < DumpVerticalSecondVariety.txt > DumpVerticalSecondVariety_.txt
-->Anzahl unterschiedliche Wörter
sort DumpVerticalSecondVariety_.txt | wc -l
3828
Es gibt 3828 unterschiedliche "Wörter"

Wie viele Wörter enthält der Text?
benzro@benzro-eMachines-E625:~$ **wc SecondVariety.txt**
2973 18500 111571 SecondVariety.txt

benzro@benzro-eMachines-E625:~$ wc DumpVerticalSecondVariety_.txt
19427 18500 104900 DumpVerticalSecondVariety_.txt

Es gibt 18500 Wörter auf 19427 Zeilen. Und somit gegen 1000 Leerzeilen.

Was sind die 5 häufigsten Wörter?
-->Aggregieren mit sort -f und uniq -i für case insensitive
sort -f DumpVerticalSecondVariety_.txt | uniq -i -c | sort -n
330 to
340 and
361 a
375 of
927
1069 the
Die fünf häufigsten Wörter case insensitve sind: the, of, a, and, to

Aus Musterlösung
tr ' ' '\n' < mein_text.txt | sort | uniq -c | wc | sort -nr | head -5

Der sort command sortiert Strings und Dateien
Usage: sort [OPTION]... [FILE]...
Ordering options:
-f, --ignore-case fold lower case to upper case characters
-g, --general-numeric-sort compare according to general numerical value
-i, --ignore-nonprinting consider only printable characters
-M, --month-sort compare (unknown) < 'JAN' < ... < 'DEC'
-h, --human-numeric-sort compare human readable numbers (e.g., 2K 1G)
-n, --numeric-sort compare according to string numerical value
-m, --merge merge already sorted files; do not sort
-o, --output=FILE write result to FILE instead of standard output
-r, --reverse reverse the result of comparisons
-u, --unique with -c, check for strict ordering;
without -c, output only the first of an equal run

Der uniq command aggregiert exakt gleiche Zeilen
Usage: uniq [OPTION]... [INPUT [OUTPUT]]
Filter adjacent matching lines from INPUT (or standard input),
writing to OUTPUT (or standard output).

With no options, matching lines are merged to the first occurrence.
-c, --count prefix lines by the number of occurrences
-i, --ignore-case ignore differences in case when comparing

sort –u Datei_X.txt
ODER
sort Datei_X.txt | uniq

Welche Wörter folgen auf "nicht"? Gib die fünf häufigsten Wörter an.
Aufs Englische not folgen:
grep -P --colour -o '(not|Not)[ ](\w)+[ ]' SecondVariety.txt | grep -P -o '[ ](\w)+' | sort -f| uniq -c -i| sort -n
grep: **'(not|Not)[ ](\w)+[ ]' liefert not Wort**
'[ ](\w)+' liefert Wort

grep: $ funktioniert bei diesem Text nicht als Anker am Ende der Zeile.

Aus Musterlösung
grep -Pio '\bnicht\b [\wäüö]+' mein_text.txt| sort | uniq -c | sort -rn | head -5

Wie oft kommt die Wortfolge "Wort1 und Wort2" vor, wo beide Wörter auf "-en" enden (z.B. "die grünen und blauen ...")?
Im Englischen -y and -y:
PCL-1/Übungen/PCL_uebung1$ **grep -P --colour -o '(\w)+y[ ](and)[ ](\w)+y[ ]' SecondVariety.txt**
slowly and methodically
Es gibt einen Treffer slowly and methodically

Syntax

`\`	Escapes the character immediately following it
`.`	Matches any single character except a newline (unless `/s` is used)
`^`	Matches at the beginning of the string (or line, if /m is used)
`$`	Matches at the end of the string (or line, if /m is used)
`*`	Matches the preceding element 0 or more times
`+`	Matches the preceding element 1 or more times
`?`	Matches the preceding element 0 or 1 times
`{...}`	Specifies a range of occurrences for the element preceding it
`[...]`	Matches any one of the characters contained within the brackets
`(...)`	Groups subexpressions for capturing to $1, $2...
`(?:...)`	Groups subexpressions without capturing (cluster)
`|`	Matches either the subexpression preceding or following it
`\1, \2 ...`	The text from the Nth group

Escape sequences

These work as in normal strings.

`\a`	Alarm (beep)
`\e`	Escape
`\f`	Formfeed
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\038`	Any octal ASCII value
`\x7f`	Any hexadecimal ASCII value
`\x{263a}`	A wide hexadecimal value
`\cx`	Control-x
`\N{name}`	A named character

Character classes

`[amy]`	Match 'a', 'm' or 'y'
`[f-j]`	Dash specifies *range*
`[f-j-]`	Dash escaped or at start or end means 'dash'
`[^f-j]`	Caret indicates "match any character *except* these"

`\d`	A digit	`[0-9]`
`\D`	A nondigit	`[^0-9]`
`\w`	A word character	`[a-zA-Z0-9_]`
`\W`	A non-word character	`[^a-zA-Z0-9_]`
`\s`	A whitespace character	`[ \t\n\r\f]`
`\S`	A non-whitespace character	`[^ \t\n\r\f]`

`\C` Match a byte (with Unicode, '`.`' matches a charac-

Anchors

All are zero-width assertions.

`^`	Match string start (or line, if `/m` is used)
`$`	Match string end (or line, if /m is used) or before newline
`\b`	Match word boundary (between `\w` and `\W`)
`\B`	Match except at word boundary (between `\w` and `\w` or `\W` and `\W`)
`\A`	Match string start (regardless of `/m`)
`\Z`	Match string end (before optional newline)
`\z`	Match absolute string end
`\G`	Match where previous `m//g` left off

Quantifiers

Quantifiers are greedy by default – match the **longest** leftmost.

Maximal	Minimal	Allowed range
`{n,m}`	`{n,m}?`	Must occur at least n times but no more than m times
`{n,}`	`{n,}?`	Must occur at least n times
`{n}`	`{n}?`	Must occur exactly n times
`*`	`*?`	0 or more times (same as `{0,}`)
`+`	`+?`	1 or more times (same as `{1,}`)
`?`	`??`	0 or 1 time (same as `{0,1}`)

Extended constructs

`(?#text)`	A comment
`(?imxs-imsx:...)`	Enable/disable option (as per `m//` modifiers)
`(?=...)`	Zero-width positive lookahead assertion
`(?!...)`	Zero-width negative lookahead assertion
`(?<=...)`	Zero-width positive lookbehind assertion
`(?<!...)`	Zero-width negative lookbehind assertion
`(?>...)`	Grab what we can, prohibit backtracking

Metacharacters:

The following 12 characters need to be escaped with a backslash - "\" - because by default, they mean something special.

```
\ | ( ) [ { ^ $ * + ? .
```

`.`	Match any one character (except `\n`)
`|`	Alternation
`( )`	Group and capture
`[ ]`	Define character class
`\`	Modify the meaning of the next char.

Anchors:

`^`	Match at the beginning of a string (or line)
`$`	Match at the end of a string (or line)
`\b`	Match at a 'word' boundary
`\B`	Match at not a 'word' boundary

These are also known as *zero width assertions*.

Quantifiers:

These quantifiers apply to the preceding *atom*.

`*`	Match 0 or more times
`+`	Match 1 or more times
`?`	Match 0 or 1 times
`{N}`	Match exactly ***N*** times
`{N,}`	Match at least ***N*** times
`{N,M}`	Match at least ***N*** but not more than ***M*** times

By default, quantifiers are "*greedy*". They attempt to match as many characters as possible. In order to make them match as few characters as possible, follow them with a question mark "`?`".

Character class metacharacters:

`^`	If the first character of a class, negates that class
`-`	Unless first or last character of a class, used for a range

Character class shortcuts:

`\d`	`[0-9]`	A digit
`\D`	`[^0-9]`	A non-digit
`\s`	`[ \t\n\r\f]`	A whitespace char.
`\S`	`[^ \t\n\r\f]`	A non-whitespace char.
`\w`	`[a-zA-Z0-9_]`	A 'word' char.
`\W`	`[^a-zA-Z0-9_]`	A 'non-word' char.

These shortcuts can be used either on their own, or within a character class.

Modifiers:

These modifiers apply to the entire pattern

`/i`	Ignore case
`/g`	Match globally (all)
`/m`	Let `^` and `$` match next to embedded `\n`
`/s`	Let `.` match `\n`
`/x`	Ignore most whitespace and allow comments
`/e`	Evaluate right hand side of `s///` as an expression

recursive grep search

```
grep -ril 'null' /home/al/sarah /var/www  # search multiple dirs
```

Searching for a string in multiple files

Our next `grep` command example searches for all occurrences of the text string `joe` within all files of the current directory:

```
grep 'joe' *
```

search all files in the current directory that end in the file extension *.txt*, like this:

```
grep 'joe' *.txt
```

One way to find all the sub-directories in the current directory is to mix the Linux `ls` and `grep` commands together in a pipe, like this:

```
ls -al | grep '^d'
```

grep gzip files

```
zgrep foo myfile.gz   # all lines containing the pattern 'foo'
```

Power file searching with find and grep

This is a special way of mixing the Linux find and grep commands together to search every file in every subdirectory of my current location. It searches for the string "foo" in every file below the current directory, in a case-insensitive manner.

```
find . -type f -exec grep -il 'foo' {} \;
```

- **-"." means "look in the current directory"**
- **`-type f` means "look in files only"**
- **`-exec grep -il foo` means "search for the string 'foo' in a case-insensitive manner, and return the matching line and filename when a match is found**
- **`{} \;` is a little bizarre syntax that you need to add to the end of your find command whenever you add the `-exec` option**. I try to think of it as a placeholder for the filenames the find command finds.

1. Schritt

Wo sind wir?

pwd

Wie kommen wir in das Verzeichnis, in dem wir arbeiten wollen?

cd Verzeichnis_X ODER **cd ..**

Wie sehen wir, welche Dateien im Verzeichnis vorliegen?

ls –l

Wie können wir in eine Datei hineinschauen?

more Datei_X.txt (quit with 'q'), [less]

2. Schritt

Wie können wir die Wörter in einer Textdatei zählen?

wc Datei_X.txt

Wie können wir die Anzahl unterschiedlicher Wörter in einem Text zählen?

1. Text vertikalisieren
2. Wörter sortieren
3. Duplikate entfernen
4. Wörter zählen

Wie können wir einen Text vertikalisieren?

1. Suche Leerschlag und ersetze mit \n im Editor oder
2. **tr ' ' '\n' < Datei_X.txt**

3. Schritt

Wie können wir die Zeilen einer Datei sortieren?

sort Datei_X.txt

Wie können wir eine Datei sortieren und identische Zeilen entfernen?

sort –u Datei_X.txt ODER

sort Datei_X.txt | uniq

4. Schritt

Wie berechnen wir die Häufigkeit für jedes Wort in einer vertikalisierten Textdatei ?

sort Datei_X.txt | uniq –c

Wie sortieren wir nach der Häufigkeit?

sort –n (numerische vs. alphabet. Sortierung)

Wie sortieren wir rückwärts (= häufigste zuerst)?

sort –rn

5. Schritt

Wie können wir grosse Textmengen durchsuchen?

grep 'Suchwort' Dateimuster

Wieviele Sätze kommen im Jahrbuch 1900 vor?

grep '<s ' SAC_Jahrbuch_1900.xml | wc

Wieviele Sätze kommen in allen Jahrbüchern des 19. Jhd. vor?

grep '<s ' SAC_Jahrbuch_18*.xml | wc

grep –c '<s ' SAC_Jahrbuch_18*.xml

6. Schritt

Wie oft kommt das **Wort** 'Berg' vor?

grep 'Berg\t' *18*.xml | wc

Wie oft kommt das **Lemma** 'Berg' vor?

grep '\tBerg$' *18*.xml | wc

Wieviele Substantive kommen vor?

grep '\tNN\t' *18*.xml | wc

Wieviele deutsche Adjektive kommen vor?

grep '\tADJA\t' *18*.xml | wc

tr = transliterate

Vertikalisieren eines Textes: Ersetze jeden Leerschlag mit einem Zeilenumbruch

tr ' ' '\n' < Test1.txt

Konvertiere alle Grossbuchstaben in Kleinbuchstaben

tr '[A-Z]' '[a-z]' < Test1.txt

Unix-Kommandos

Umlenken der Ausgabe mit Überschreiben: '>'

Umlenken der Ausgabe mit Anfügen: '>>'

Umlenken der Eingabe: '<'

Verbinden von zwei Kommandos: Pipe '|'

Wildcards

= Stellvertretersymbole

- Wildcard * == beliebige Anzahl Zeichen
- Wildcard ? == ein beliebiges Zeichen

Wildcard **[ ]** == definiert eine Menge von Zeichen, von denen eines zutreffen muss.

Wildcard **-** == ein Strich in einer Klammer [ ] definiert einen Bereich von – bis.

Zeichen-Kodierung

Latin-1 = iso-8859-1 (Umlaute als 1 byte) **Alt!** ☹

UTF-8 (Umlaute als 2 bytes) **Gut!** ☺

Unix-Kommando zur Bestimmung der Kodierung

file –i filename

Unix-Kommando zur Konvertierung einer Kodierung in eine andere (**f**rom – **t**o)

iconv –f –t filename

grep

grep und Zählen (count) der Ergebnisse

`grep -c pattern file` zählt per Datei

`grep pattern file | wc -l` zählt die gesamte Ausgabe

grep

grep für ein Muster xx und Anzeige des Kontexts

`grep -B 2 xx file` Zeige zwei Zeilen des vorangehenden Kontexts (Before)

`grep -A 4 xx file` Zeige vier Zeilen des folgenden Kontexts (After)

`grep -C 3 xx file` Zeige drei Zeilen des vorangehenden und folgenden Kontexts (Context)

grep

Gegeben ein Text aus dem Text+Berg-Korpus mit Part-of-Speech Tags und Lemmas im 3-Spalten-Format.

grep VV	findet alle Verben im Deutschen
grep ADJ	findet alle Adjektive im Deutschen
grep ADV	findet alle Adverbien
grep befreien	findet Lemma und Wortform
grep '^befreien'	findet Wortform
grep '^be.*en\t'	findet alle Wortformen mit *be...en*
grep '\tbe.*en$'	findet alle Lemmas mit *be...en*

grep

grep –o '^be.*en\t' gibt statt der Zeile nur den Treffer aus → erlaubt Sortieren und Zählen der Hits

grep --colour '^be.*en\t' markiert den Treffer in der Zeile

grep und Erweiterungen

grep bietet die Grundfunktionalität der Suche mit regulären Ausdrücken.

Erweiterungen von grep

egrep

grep –E ist wie **egrep**

grep –P erlaubt reguläre Ausdrücke wie in Perl. Sehr mächtig! Empfohlen!

grep

Befehl	Erklärung
grep '\tbe..en$'	findet alle Lemmas mit 2 Zeichen zwischen ***be*** und ***en***
grep '\tbe.[aeiou].en$'	findet alle Lemmas mit Vokal zwischen 2 Buchstaben
grep '\tbe.[äöü].en$'	findet alle Lemmas mit Umlaut zwischen 2 Buchstaben
grep '^[bg]e..en\t'	findet alle Wortformen mit *be..en* und *ge..en*

Wichtig!

Ein regulärer Ausdruck sucht immer den längsten möglichen String (= *greedy pattern matching*).

Unterscheide:

grep –o '^be.*en\t' findet alle Wortformen mit *be...en*

und

grep –o '^be.*en' findet alle **Zeilen** mit *be...en*

grep

grep aa

grep -i "aa" Case-insensitive! findet "aa" "AA", "aA", "Aa"

grep [aeiou][aeiou] findet alle Vokal-Paare!

grep [aeiou][aeiou][aeiou] findet alle Vokal-Tripel!

grep [aeiou][aeiou][aeiou][aeiou] findet alle Vokal-Quadrupel! Gibt es wirklich!

grep '[aeiou]\{4\}' mit Anf-Zeichen!

```
benzro@benzro-eMachines-E625:~$ grep -P '[aeiou]{4}' greptest.txt
weeaaioann
ienaeeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dsijeioö eieii
djijioeij ifojfiojf
benzro@benzro-eMachines-E625:~$ grep -P '[aeiou]\{4\}' greptest.txt
benzro@benzro-eMachines-E625:~$ grep -P [aeiou]\{4\} greptest.txt
weeaaioann
ienaeeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dsijeioö eieii
djijioeij ifojfiojf
benzro@benzro-eMachines-E625:~$ grep -P [aeiou]{4} greptest.txt
weeaaioann
ienaeeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dsijeioö eieii
djijioeij ifojfiojf
```

grep

grep '^[1234567890]' findet Wörter, die mit einer Ziffer beginnen

grep '^[1234567890][a-z]' findet Wörter, die mit einer Ziffer + einem Buchstabe beginnen (Beachte grosse und kleine Buchstaben und Umlaute)

grep -E '^\d[a-z]' Alternative Notation!

Wichtige Symbole

\n	neue Zeile
\t	Tabulator
\d	Ziffern 0-9 ("digits")
\w	Buchstaben [A-Za-z], Unterstrich und Ziffern
\s	Leerschlag, Tabulator, neue Zeile

grep -P

grep –P '(ADJ|ADV)' mit Alternativen

grep –P 'th?al' findet *thal* und *tal*

grep –P 't(h|)al' ist äquivalent zu

grep –P 'th?al' !!!

grep –P '^schnee*' ist äquivalent zu

grep –P '^schne+' !!!

```
benzro@benzro-eMachines-E625:~$ grep -P ee* greptest.txt
weeaaioann
ienaeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dijdse
dsijeioö eieii
dfsiadjjasije
djijioeij ifojfiojf
diojeojjfndoiaj
benzro@benzro-eMachines-E625:~$ grep -P e+ greptest.txt
weeaaioann
ienaeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dijdse
dsijeioö eieii
dfsiadjjasije
djijioeij ifojfiojf
diojeojjfndoiaj
```

grep -P

grep –P ‘([aeiou])\1’ findet alle Vokalpaare

Beachte die **Wiederaufnahme** des ersten Treffers (durch runde Klammern markiert) mit \1

grep –P ’^(\w)\1’ findet alle Buchstaben- oder Ziffernpaare am Wortanfang.

grep –P ’(\d)\1(\d)\2’ findet zwei gleiche Ziffernpaare.

findet Buchstabentrippelgrep -P "(\

```
benzro@benzro-eMachines-E625:~$ grep -P '(\w)\1' greptest.txt
weeaaioann
ienaeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
dieiaööööeaieiied
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
dijödljsioejiiie
dsijeioö eieii
dfsiadjjasije
diojeojjfndoiaj
benzro@benzro-eMachines-E625:~$ grep -P '(\w)\1.\1' greptest.txt
ienaeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
```

```
benzro@benzro-eMachines-E625:~$ grep -P '(\w)\1(\w)\2' greptest.txt
weeaaioann
ienaeeaeanenienlödieeeeeakldjije dikeieeeeeediek dieiiiieis
ieeiadijeiiiiiiii dijkeijjffnneiiiiidd
eieifejfioieiiioeioiod ieowiejiofoajjejejjjeiieieeeeeeee
```

Reguläre Ausdrücke - Eine rekursive Definition

Jedes Zeichen (auch ein leeres) ist ein regulärer Ausdruck.

Beispiel: “a”

Die **Sequenz** (Folge) von regulären Ausdrücken r_1 und r_2 ist ein regulärer Ausdruck.

Beispiel: “r_1r_2”

Die **Alternative** von regulären Ausdrücken r_1 und r_2 ist ein regulärer Ausdruck.

Beispiel: “$r_1|r_2$”

Ein regulärer Ausdruck r mit **Optionalität, ein- oder mehrmaliger Wiederholung** ist ein regulärer Ausdruck.

Beispiel: “ r* ”

Repetition takes precedence over concatenation, which in turn tak

Reguläre Ausdrücke

a+ == aa* (a einmal oder mehrmals)

a? == a|_ (a oder nichts)

a{3} == aaa

a{2,3} == aa|aaa

[ab] == a|b

\d == 0|1|2|3|4|5|6|7|8|9

Grenzen Regulärer Ausdrücke

aaabbb ist ein regulärer Ausdruck

a*b* ist ein regulärer Ausdruck

Wir können aber nicht allgemein formulieren, dass genauso viele **a** wie **b** erkannt werden sollen. Salopp: Reguläre Ausdrücke haben keine Zähler!

Formal:

a^nb^n mit $n \in N$ ist **keine** reguläre Sprache!

a^nba^n mit $n \in N$ ist **keine** reguläre Sprache!

Aber

a^nb^n mit $n \in N, n < 10$ ist eine reguläre Sprache!

Reguläre Sprachen

= Sprachen, die mit regulären Ausdrücken definiert werden können.

Formal:

a^nb^n mit $n \in N$ ist **keine** reguläre Sprache!

Aber

a^nb^n mit $n \in N, n < 10$ ist eine reguläre Sprache!

a^nb^m mit $n,m \in N$ ist eine reguläre Sprache

$a^nb^mc^m$ mit $n,m \in N, m < 5$ ist eine reg. Sprache

$a^mb^5c^m$ mit $m \in N$ ist **keine** reguläre Sprache!

Reguläre Ausdrücke vs. Wildcards

sind nicht das selbe!

- Wildcard * == beliebige Anzahl Zeichen
- RegEx * == null oder mehr Wiederholungen eines reg. Ausdrucks

- Wildcard ? == ein beliebiges Zeichen
- RegEx ? == null oder 1 Instanz eines reg. Ausdrucks

First steps in the Python interpreter

1. Start the Python interpreter with 'python'
2. Test Python commands
3. Quit the Python interpreter with 'Ctrl+d'

Write programs in Python

1. Write your Python program in an editor.
2. Save the file to e.g. 'my_program_1.py'
3. Execute the program with 'python my_program_1.py'

Note: Interrupt an 'endless' program with 'Ctrl+c'.

First Python commands

	Examples
Numbers and Arithmetic	2 + 4 21 / 3 * 4**7
Variables and Assignments	counter = 21 counter = 21 / 7 * 5 counter += 2 x = y = z = 15
Strings - concatenation - length function	word1 = 'der' word2 = word1 + ' Computer' x = len(word2)
Strings with Indexes - from the beginning to position 4 - two characters from the end	letter = word[0] part = word[0:5] suffix = word[-2:]

```
>>> word="Alibaba"
>>> len(word)
7
Element 2 bis aber nicht inklusive 5
>>> istr=word[2:5]
>>> istr
'iba'
Element 0 bis aber nicht inklusive 5
>>> istr=word[:5]
>>> istr
'Aliba'
Element 0 bis aber nicht inklusive 7
>>> istr=word[:7]
>>> istr
'Alibaba'
Element StringLänge-2 bis aber nicht inklusive Stringlänge
>>> isuff=word[-2:]
>>> isuff
'ba'
>>> istr=word[len(word)-2:]
>>> istr
'ba'
```

Loops and conditions in Python

	Examples

while Loops and Indentation	while counter < 10:
	  print "Hello"
	  counter += 1

Comparisons	counter < 50 counter == 13 word == 'Linguist'

if – elif – else Conditions	if x < 10:
	  print "Hello"
	else:
	  print "Hi"

Comments	# This is a comment

Variables

When a variable is used for the first time, it must get a **value**.

The first value assignment determines the variable's **type** (integer, floating point number, string, Boolean).

The variable's type restricts the possible **operations**. Examples:

- Integer: +, -, *, /
- String: +, len()

Indentation in Python

Indentation in Python is semantically significant.

A block of commands in e.g. loops or conditions must be indented.

Note: The output of a Python program (often) changes with indentation. ☺ Be careful!

	Examples
for loop	for my_item in my_list for my_num in range(0, 30) for my_num in range(0, 30, 3) for my_num in range(30)
lists	[2, 4, 6, 8, 10] ['python', 'is', 'great'] ['27.', 'Oktober', 2011]
list assignment and concatenation	my_list1 = [3, 6, 9, 12] my_list2 = [2, 4, 8, 16] my_long_list = my_list1 + my_list2
accessing and changing items in a list	my_item = my_list[2] my_part = my_long_list[:5] my_long_list[3] = 64

Range von 0 bis aber nicht inklusive 5
```
>>> ilist1=range(0, 5)
>>> ilist1
[0, 1, 2, 3, 4]
```
Zweierschritte, Start bei 0, bis aber nicht inklusive 5
```
>>> ilist2=range(0, 5, 2)
>>> ilist2
[0, 2, 4]
```
Liste: + ist nicht dasselbe wie append (siehe unten)
```
>>> ilist=ilist1+ilist2
>>> ilist
[0, 1, 2, 3, 4, 0, 2, 4]
```
Set, Menge
```
>>> set(ilist)
set([0, 1, 2, 3, 4])
```

Vorsicht mit Methoden wie append, nicht dasselbe wie +
```
>>> ilist=ilist1.append(ilist2)
>>> ilist
>>> type(ilist)
<type 'NoneType'>
>>> ilist1
[0, 1, 2, 3, 4, [0, 2, 4]]
```

	Examples
sorting a list - reverse order - according to length of items	my_list = sorted(my_list) my_list = sorted(my_list, reverse=True) my_list = sorted(my_list, key=len)
inserting an item at the end	my_list.append(42)
inserting an item at the beginning - or at any other place	my_list.insert(0, 'hello') my_list.insert(3, 'world')
deleting a list deleting an item from a list	del my_list del my_list[3]
converting a string to a list - by splitting on the blank symbol	my_text = 'this is magic' my_list = my_text.split(' ')

>>> dir()
```
['__builtins__', '__doc__', '__name__', '__package__', 'ilist', 'ilist1', 'ilist2', 'istr', 'isuff', 'word']
```
>>> locals()
```
{'isuff': 'ba', 'word': 'Alibaba', 'ilist2': [0, 2, 4], '__builtins__': <module '__builtin__' (built-in)>, '__package__': None, 'istr': 'ba', 'ilist': None, '__name__': '__main__', 'ilist1': [0, 1, 2, 3, 4, [0, 2, 4]], '__doc__': None}
```
>>> globals()
```
{'isuff': 'ba', 'word': 'Alibaba', 'ilist2': [0, 2, 4], '__builtins__': <module '__builtin__' (built-in)>, '__package__': None, 'istr': 'ba', 'ilist': None, '__name__': '__main__', 'ilist1': [0, 1, 2, 3, 4, [0, 2, 4]], '__doc__': None}
```

```
# initialize the variable
#text = 'programmieren macht spass'
text = '01234567'

# while the position is smaller than the length of the text
for position in range(3, len(text)+1):
    # print the trigram
    print position, ": ",text[position-3:position]
print "position", len(text)+1, "wird im range nicht angenommen.\n" \
            +"(Bis aber nicht inklusive)"
```

```
3 :  012
4 :  123
5 :  234
6 :  345
7 :  456
8 :  567
position 9 wird im range nicht angenommen.
(Bis aber nicht inklusive)
```

```
# initialize the list variables with lists of words
lexicon = ['computer', 'linguistik', 'fleisch', 'fleischer', \
           'erzeugnis', 'zeugnis']

testwords = ['computerlinguistik', 'computerexperte', \
           'fleischerzeugnis']

# for each word in the list  testwords
for word in testwords:
    # split the word in all possible ways
    # the first element is at least 3 characters long
    for position in range(3, len(word)):
        ## assign the split parts to two variables
        part1 = word[0:position]
        part2 = word[position:]
        ## print the current split for inspection
        print part1, part2

        ## solution with two nested if conditions
#       if part1 in lexicon:
#           if part2 in lexicon:
#               print 'Found compound ', part1, part2, ':-)'

        ## solution with a complex condition connected with 'and'
        if (part1 in lexicon) and (part2 in lexicon):
            print 'Found compound ', part1, part2, ':-)'
```

```
computere xperte
computerex perte
computerexp erte
computerexpe rte
computerexper te
computerexpert e
fle ischerzeugnis
flei scherzeugnis
fleis cherzeugnis
fleisc herzeugnis
fleisch erzeugnis
Found compound  fleisch erzeugnis :-)
fleische rzeugnis
fleischer zeugnis
Found compound  fleischer zeugnis :-)
fleischerz eugnis
fleischerze ugnis
fleischerzeu gnis
fleischerzeug nis
fleischerzeugn is
fleischerzeugni s
```

```python
import sys  ## necessary for command line arguments
import codecs
## define a variable for the length of the longest word
max_len = 0
## ... and for the word itself
max_word = ''
## get the filename (to be searched) from the command line
#filename = sys.argv[1]
filename="/media/benzro/OS/Users/benzro/Documents/LubuntuShared/\
8) PCL-1/Beispielsprogramme/person_name_and_gender.txt"
## open the file for 'reading'
infile = codecs.open(filename, 'r', 'utf-8')
## for each line in this file do
for line in infile:
    my_line_list = line.split(' ')
    print "Space delimiter", my_line_list
    my_line_list = line.split()
    print "Default delimiter set", my_line_list
    for word in my_line_list:
        ## if the current word is longer than max_len
        if len(word) > max_len:
            ## set max_len to the length of the current word
            max_len = len(word)
            ## store the current word in max_word
            max_word = word
            print max_len, max_word
## close the file
infile.close()
## print the length of the longest word and the word itself
print max_len, max_word
```

```
Space delimiter [u'Tom\tm\n']
Default delimiter set [u'Tom', u'm']
3 Tom
Space delimiter [u'Huck\tm\n']
Default delimiter set [u'Huck', u'm']
4 Huck
Space delimiter [u'Becky\tf\n']
Default delimiter set [u'Becky', u'f']
5 Becky
Space delimiter [u'Sid\tm\n']
Default delimiter set [u'Sid', u'm']
Space delimiter [u'Sidney\tm\n']
Default delimiter set [u'Sidney', u'm']
6 Sidney
Space delimiter [u'Jim\tm\n']
Default delimiter set [u'Jim', u'm']
Space delimiter [u'Ben\tm\n']
Default delimiter set [u'Ben', u'm']
Space delimiter [u'Mary\tf\n']
Default delimiter set [u'Mary', u'f']
Space delimiter [u'Peter\tm\n']
Default delimiter set [u'Peter', u'm']
6 Sidney
```

Tom	m
Huck	m
Becky	f
Sid	m
Sidney	m
Jim	m
Ben	m
Mary	f
Peter	m

Finde Median einer Liste mit gerader Länge

```
>>> ilist= [18, 21, 16, 24, 19, 16, 23, 26]
>>> ilist_sorted=sorted(ilist)
>>> ilist_sorted
[16, 16, 18, 19, 21, 23, 24, 26]
>>> ilen=len(ilist)/2
>>> ilen
4
>>> ilist_sorted[ilen]
21
```

Finde Median einer Liste mit ungerader Länge

```
>>> ilist= [18, 21, 16, 24, 19, 16, 23, 26, 14]
>>> ilist_sorted=sorted(ilist)
>>> ilist_sorted
[14, 16, 16, 18, 19, 21, 23, 24, 26]
>>> ilen=len(ilist)/2
>>> ilen
4
>>> ilist_sorted[ilen]
19
```

Overview: File handling

	Examples
Command line arguments	import sys my_filename = sys.argv[1] my_arguments = sys.argv
Open file for reading Open file for writing - my_infile / my_outfile are file objects	my_infile = open(my_filename, 'r') my_outfile = open(my_filename, 'w')
Close file	my_infile.close()
Loop over each line in a file	for my_line in my_infile

```python
import codecs
## open the file for 'reading'
infile = codecs.open(filename, 'r', 'utf-8')
```

```python
print("I just printed %s pages to the printer %s" % (num, printer))
```

However, the current standard for this sort of string formatting is to use the *format* method of strings with numbered or named positions within the string:

```python
print("I just printed {0} pages to the printer {1}".format(num, printer))
print("I just printed {num} pages to the printer {printer}".format(num=num, printer=printer))
```

```
>>> print("""Dear %(recipient)s,
...
... I wish you to leave Sunnydale and never return.
...
... Not Quite Love,
... %(sender)s
... """ % {'sender': 'Buffy the Vampire Slayer', 'recipient': 'Spike'})
Dear Spike,

I wish you to leave Sunnydale and never return.

Not Quite Love,
Buffy the Vampire Slayer

>>>
```

Hashes (= Python Dictionaries)

	Examples
Define a hash (= dictionary) - with two entries	my_hash = {} my_hash = {'uni':'noun', 'gute':'adj'}
Enter a key – value pair	my_hash['mit'] = 'praep' my_hash.update({"mit":"praep"})
Check if a key is in the hash - and get the value	my_hash.has_key(my_word) my_word in my_hash my_hash.get(my_word)
Loop over a hash	for my_word in my_hash.keys() for my_word in my_hash

>>> idict{'a': '2', 'b': 5}Äquivalente Zugriffe==========>:

```
>>> a_tuple = 1, 2, 3, "four"
>>> a_tuple
(1, 2, 3, 'four')
>>> a_tuple = (1, 2, 3, "four")
>>> a_tuple
(1, 2, 3, 'four')
>>> a_tuple = (1, 2, 3, "four", a_tuple)
>>> a_tuple
(1, 2, 3, 'four', (1, 2, 3, 'four'))
>>> a_tuple = (1, 2, 3, "four", a_tuple)
>>> a_tuple
(1, 2, 3, 'four', (1, 2, 3, 'four', (1, 2, 3, 'four')))
>>> a_tuple[4][4][3]
'four'
>>>
```

	Examples
get all **keys** into a list	my_list = my_hash.keys()
get all **values** into a list	my_list = my_hash.values()
get the **number** of keys (= entries)	my_length = len(my_hash)
get all key-value **tuples** into a list	my_list = my_hash.items()

Hashes (= Python Dictionaries): Typical Usages

	Examples
word + PoS tag	my_hash = {'der':'ART', 'in':'APPR', 'Auto':'NN'}
item + frequency	my_hash = {'der':4, 'in':12, 'Auto':2} my_hash = {'ART':7, 'NN':12, 'APPR':8}
word + translation	my_hash = {'der':'the', 'in':'in', 'Auto':'car', 'Haus':'house'}
string + dummy value (for quick access)	my_hash = {'der':1, 'in':1, 'Auto':1}

List

my_list = []

stores a sequence of values $x_1, x_2, x_3 \ldots$

preserves order

quick access via number key to value, but **slow** access to a particular value

Hash

my_hash = {}

stores relations $x_1 \rightarrow y_1$ $x_2 \rightarrow y_2$

random order !

quick access via key to value

Regular Expressions
Search and Substitution

	Examples
Regular expression matching	import re my_match = re.search('abc', my_text) my_match = re.search('^\waa..', my_text) print my_match.group()
Get hit from within a match	my_match = re.search('ab(\w+)', my_text) print my_match.group(1)
Get multiple matches in a list	my_matches = re.findall('a\w+', my_text)
Find and substitute	my_new_text = re.sub('t', 'th', my_text)

Immer prüfen. Wenn none gibts son

Textstring

```
>>> my_text
'dama abc jalj daapa saama jobab jobabc gamaala aslkjlabkd jamaboielkdfjabkiie abumumabizekluehzabj'
```

Alle Matches mit findall()

```
>>> my_matches = re.findall('a\w+', my_text)
>>> my_matches
['ama', 'abc', 'alj', 'aapa', 'aama', 'ab', 'abc', 'amaala', 'aslkjlabkd', 'amaboielkdfjabkiie', 'abumumabizekluehzabj']
```

Erster Match mit search()

```
>>> my_match = re.search('abc', my_text)
>>> my_match.group()
'abc'
```

Erster Match mit search()

```
>>> my_match2 = re.search('\w*aa..', my_text)
>>> my_match2
<_sre.SRE_Match object at 0xb7477db0>
>>> my_match2.group()
'daapa'
```

Erster Match mit search()

```
>>> my_match3 = re.search('ab(\w+)', my_text)
>>> my_match3.group()
'abc'
>>> my_match3.group(1)
'c'
```

Finden und Ersetzen (Find and Substitute)

```
>>> my_new_text = re.sub('abc','xyziak' , my_text)
>>> my_new_text
'dama xyziak jalj daapa saama jobab jobxyziak gamaala aslkjlabkd jamaboielkdfjabkiie abumumabizekluehzabj'
```

Miscellaneous

	Examples
Defining functions - the main function	def my_function(): def main():
Global variables (defined outside a function)	
Remove **leading** (left) "whitespace" characters	my_line.lstrip()
Remove **trailing** (right) "whitespace" characters	my_line.rstrip()

Python Tuples (are like Lists but immutable)

	Examples
Define a tuple - with three elements	my_tuple = () my_tuple = ('Junge', 'NN', 'ADJA')
Accessing an element	my_tuple[2]
Looping over the elements of a tuple	for x in my_tuple:
Tuples can be sorted	(2, 4, 100) < (2, 5, 1)

```
>>> itupel=2,5,4,7,1>>> sorted
'tuple' object does not support item assignment>>>
```

Using tuples for sorting values from a hash

```
my_hash = {'x':3, 'z':10, 'y':4}
my_list = []
for key, value in my_hash.items():
        ## create a list of tuples
        my_list.append((value,key))

print sorted(my_list)
```

Dictionary mit key value Pair

```
>>> my_hash={'y': 4, 'x': 3, 'z': 10}
>>> my_list=[]
```

Speichere Key Value Paare in einer List aus Tupeln

```
>>> for key, value in my_hash.items():
…         my_list.append((value,key))
...
>>> my_list
[(4, 'y'), (3, 'x'), (10, 'z')]
```

Eine Liste aus Tupeln kann sortiert werden

```
>>> print sorted(my_list)
[(3, 'x'), (4, 'y'), (10, 'z')]
```

Python Functions

We can define and call functions with arguments.

```
def my_print_function(freq):
        for x in range(0, freq):
                print x, 'Hello world'

print 'And now we write some lines'
my_print_function(40)
```

```
## open a file for reading
infile = open(filename, 'r')
## create a name for the first output file
outfilename_1 = filename + '.a_words.txt'
## open the first file for writing
outfile_a = open(outfilename_1, 'w')
## create a name for the second output file
outfilename_2 = filename + '.b_words.txt'
## open the second file for writing
outfile_b = open(outfilename_2, 'w')
## for each line in the input file
for line in infile:
    # split the line into a list of strings
    line_list = line.split()
    # for each word in the line
    for word in line_list:
        # remove a comma at the end of the word
        if word[-1] == ',' or word[-1] == '.':
            word = word[0:-1]
        # check if the word starts with 'a'
        if word[0] == 'a' and len(word)>10:
            print word
            # write output to the first file
            outfile_a.write(word)
            outfile_a.write('\n')
        # check if the word starts with 'b'
        elif word[0] == 'b' and len(word)>10:
            print word
            # write output to the second file
            outfile_b.write(word)
            outfile_b.write('\n')
## close the files
infile.close()
```

```
>>> istr="asdine"
>>> istr[-1]
'e'
>>> istr[-1:]
'e'
>>> istr[-5:]
'sdine'
>>> istr[-5]
's'
```

Liste aus drei Tupeln

```
>>> my_list
[(4, 'y'), (3, 'x'), (10, 'z')]
```

Die letzten zwei Tupel in einer Liste

```
>>> my_list[-2:]
[(3, 'x'), (10, 'z')]
```

Das zweitletzte Tupel

```
>>> my_list[-2]
(3, 'x')
```

Das erste Tupel der zwei letzten Tupel

```
>>> my_list[-2:][1]
(10, 'z')
```

...und davon das erste Element

```
>>> my_list[-2:][1][1]
'z'
```

Das letzte Tupel der letzten zwei Tupel

```
>>> my_list[-2:][-1]
(10, 'z')
```

...als Liste

```
>>> my_list[-2:][-1:]
[(10, 'z')]
```

...und Zugriff auf das zweitletzte Element des Ersten Tupels

```
>>> my_list[-2:][-1:][0][-2]
10
```

```python
#!/usr/bin/python

# program to find compound words in a given text
# - assumes the words from the text itself to be the lexicon
# by Martin Volk

import sys  ## necessary for command line arguments
def main():
    ## define a list for the lexicon
    lexicon = []
    ## define a type counter
    type_counter = 0
    ## get the filename
    #filename = sys.argv[1]
    filename="/media/benzro/OS/Users/benzro/Documents/LubuntuShared/"\
            +"8) PCL-1/Beispielsprogramme/SecondVariety.txt"
    ## open the input file for reading
    infile = open(filename, 'r')
    ## for each line in the input file
    for line in infile:
        # split the line into a list of strings
        line_list = line.split()
        # for each word in the line
        for word in line_list:
            # if the word is not in the lexicon, then add it
            if word not in lexicon:
                lexicon.append(word)
                # count the words in the lexicon
                type_counter += 1
    ## close the file
    infile.close()
    ##########################################################
    ## define a compound counter
    compound_counter = 0
    ## check for each word in the lexicon
    ## if it is composed of two other words from the lexicon
    for word in lexicon:
        # split the word in all possible ways
        # the first element must be at least 3 characters long
        # and the last element must be at least 3 characters long
        for position in range(3, len(word)-2):
            ## check if both parts are in the lexicon
            if word[0:position] in lexicon and word[position:] in lexicon:
                print 'Found compound ', word[0:position], word[position:]
                compound_counter += 1
    print
    print 'I collected', type_counter, 'different words for the lexicon.'
    print 'I found', compound_counter, 'compounds ;-)'

## This is the standard boilerplate to call the function main
if __name__ == '__main__':
    main()
```

```python
## assumes the person names to be in the format: name\tgender
##
## by Martin Volk

import sys  ## necessary for command line arguments
## define a hash as a global variable
my_person_hash = {}
## define a function that reads the person names from file <person_name_file>
## and stores them in the global hash
def read_and_store_person_names():
    ## get the filename of the person name file
    #filename = sys.argv[1]
    filename="/media/benzro/OS/Users/benzro/Documents/LubuntuShared/"\
           +"8) PCL-1/Beispielsprogramme/person_name_and_gender.txt"
    ## open the person name file for reading
    infile = open(filename, 'r')
    ## for each line in the file
    for line in infile:
        # strip away the \n symbol
        clean_line = line.rstrip()
        # split the line into a list of strings
        line_list = clean_line.split('\t')
        name = line_list[0]
        gender = line_list[1]
        # save the name --> gender pair in a hash
        my_person_hash[name] = gender
        # add the genitive form of the name
        # example: Tom --> Toms
        name = name + 's'
        # save the genitive name --> gender pair in the hash
        my_person_hash[name] = gender
##########################################
def main():
    # initialize the counters
    person_counter = 0
    gender_counter = {}
    # read the person names into a hash
    read_and_store_person_names()
    ## get the filename of the text file
    #filename = sys.argv[2]
    filename="/media/benzro/OS/Users/benzro/Documents/LubuntuShared/"\
           +"8) PCL-1/Beispielsprogramme/SecondVariety.txt"
    ## open the file for reading
    infile = open(filename, 'r')
    ## for each line in the file
    cnt=0
    for line in infile:
        cnt+=1
        # split the line into a list of strings
        line_list = line.split()
        # for each word in the line
        for word in line_list:
            # if the word is in the person name hash
            if word in my_person_hash:
                gender = my_person_hash[word]
                # print a line of success and joy :-)
                print cnt,'Found name ', word, ' gender ', gender
                # count the number of found name occurrences
                person_counter += 1
                # count the gender occurrences; get(key[, default])
                gender_counter[gender] = gender_counter.get(gender, 0) + 1
    ## close the file
    infile.close()
```

default value fallskey

```python
# program to do word-by-word translation based on a bilingual dictionary
# by Martin Volk

# define a hash (= dictionary)
my_lexicon = {'Tante':'aunt',
    'Polly':'Polly',
    'sah':'saw',
    'Mann':'man',
    'mit':'with',
    'Fernglas':'telescope',
    'im':'in the',
    'Garten':'garden',
    'dem':'the',
    'der':'the',
    'den':'the' }
test_sentence = 'Tante Polly sah den Mann mit dem Fernglas'
test_sentence = 'der Mann sah die Tante Polly mit dem Fernglas'
# test_sentence = 'Tante Polly sah den Mann im Garten'
test_sentence = 'Tante Polly sah den Mann im Garten mit dem Fernglas'
print 'Input sentence:', test_sentence
# convert the sentence into a list
test_list = test_sentence.split()
print 'Output sentence:',
# work through the list and translate each word
for word in test_list:
    ## if the word is in the lexicon as key
    if word in my_lexicon:
        print my_lexicon[word],
    else:
        print '\n >>>', word, ' is unknown!!! :-('
print
print '---------------------'
```

Encoding einer Datei bestimmen mit file command

```
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/8) PCL-1/Vorlesung/vl07-vorspann$ file **-*
AE-l1-encoded-as-l1.txt:       ISO-8859 text
AE-l1-encoded-as-utf8.txt:     UTF-8 Unicode text
AE-l1-encoded-as-utf8.txt~:    UTF-8 Unicode text
ae-l1_.txt:                    ISO-8859 text
ae-l1.txt:                     ISO-8859 text
AE-utf-8-encoded-as-l1.txt:    very short file (no magic)
AE-utf-8-encoded-as-utf8.txt:  very short file (no magic)
```

Von L1 nach utf-8 konvertieren
$ **iconv -f iso-8859-1 -t utf8 ae-l1.txt > ae-utf8.txt**
Von utf-8 nach L1 konvertieren
$ **iconv -f utf8 -t iso-8859-1 ae-utf8.txt > ae-l1_test.txt**
Von utf-8 nach ascii konvertieren
es wird eine Fehlermeldung angezeigt, da das Zeichen ä nicht konvertiert werden kann
$ **iconv -f utf8 -t ascii ae-utf8.txt > ae-ascii.txt**
iconv: illegal input sequence at position 1

$ **file ***

ae-utf8.txt:	UTF-8 Unicode text
ae-ascii.txt:	very short file (no magic)
ae-l1_test.txt:	ISO-8859 text

Historically a \n was used to move the carriage down, while the \r was used to move the carriage back to the left side of the page.

Two different characters.

\n is used as an end-of-line terminator in Unix text files

\r is used as an end-of-line terminator in Mac text files

\r\n (ie both) are used to terminate lines in Windows and DOS text files.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

import codecs

#Datei ae-l1.txt Western(ISO-8859-15)
#a@ mit utf-8; a\E4 mit Gedit

#http://unicode-table.com/de/#latin-1-supplement
#http://www.utf8-zeichentabelle.de/
#https://de.wikipedia.org/wiki/ISO_8859-1
print "\nHexadezimale Representation des Speicherinhalts für:"
print "ä: ISO8859-1=E4; Unicode=U+00E4(228); utf-8=c3 a4(11000011 10
print "Ä: ISO8859-1=C4; Unicode=U+00C4(196); utf-8=c3 84(11000011 10

print "ü: ISO8859-1=FC; Unicode=U+00FC(252); utf-8=c3 bc"
print "Ü: ISO8859-1=DC; Unicode=U+00DC(220); utf-8=c3 9c"
print "ö: ISO8859-1=F6; Unicode=U+00F6(246); utf-8=c3 b6"
print "Ö: ISO8859-1=D6; Unicode=U+00D6(214); utf-8=c3 96"


#Encode-Decode
#.------------
#Decode: Typ String --> Typ Unicode
#Encode: Typ Unicode --> Typ String
```

```
# Decode from l1 encoded file into unicode strings
f = open("./ae-l1.txt", "r")
f1 = codecs.open("./ae-l1.txt", "r", "l1")
f2 = codecs.open("./ae-l1.txt", "r", "utf-8")

# Encode unicode strings into UTF-8 or l1 encoded file
g = open("./AE-encoded.txt", "w")
g1 = codecs.open("./AE-l1-encoded-as-l1.txt", "w", "l1")
g2 = codecs.open("./AE-l1-encoded-as-utf8.txt", "w", "utf-8")
g3 = codecs.open("./AE-utf-8-encoded-as-l1.txt", "w", "l1")
g4 = codecs.open("./AE-utf-8-encoded-as-utf8.txt", "w", "utf-8")

print "\nDatei ae-l1.txt hat Western(ISO-8859-15)= l1 Kodierung\
 und enthält die Zeichen a\E4 (Gedit), also aä; a@ mit utf-u8 \n"

print 'f ist vom Typ String und übernimmt a\E4. E4 wird jedoch' \
      'nicht erkannt von der upper() Methode und ignoriert'
print 'g schreibt A\E4 in Datei AE-encoded.txt' \
      ' (ISO-8859 text) gemäss file *\n'\
      ' (Es erscheint beim Öffnen: A@ mit utf-8; A\E4 mit Gedit)'
for line in f: #f thought input was utf-8
    g.write(line.upper())
    print "f Type:", type(line) #output as l1
    print "f Canonical repr(line.upper()):", "==>",repr(line.upper()
    print "f Canonical repr(line):", "==>",repr(line), "<=="
    print "f Printed line.upper():","==>",line.upper(), "<=="
    print "f Printed line:","==>",line, "<==\n"
```

```
print 'f1 ist von Typ unicode. f1 wusste, dass Input l1 kodiert ' \
      'ist, E4 wird von upper()Methode erkannt und auf C4 geändert'
print 'g1 schreibt A\C4 in Datei AE-l1-encoded-as-l1.txt' \
      ' (ISO-8859 text) gemäss file *\n' \
      ' (Es erscheint beim Öffnen: A@ mit UTF-8; A\C4 mit Gedit)'
print 'g2 schreibt AÄ in Datei AE-l1-encoded-as-utf8.txt' \
      ' (UTF-8 Unicode text) gemäss file *\n' \
      ' (Es erscheint beim Öffnen: AÄ mit UTF-8; AÄ mit Gedit)'
for line in f1: #f1 knew input was l1
    g1.write(line.upper())
    g2.write(line.upper()) #output as utf-8
    print "f1 Type:", type(line) #output as l1
    print "f Canonical repr(line.upper()):", "==>",repr(line.upper()
    print "f Canonical repr(line):", "==>",repr(line), "<=="
    print "f Printed line.upper():","==>",line.upper(), "<=="
    print "f Printed line:","==>",line, "<==\n"
```

```
print 'f2 ist von Typ unicode. f2 dachte Input sei utf-8 und ' \
      'ignorierte das unbekannte ä'
print 'g3 schreibt A in Datei AE-utf-8-encoded-as-l1.txt' \
      ' (very short file (no magic)) gemäss file *\n' \
      ' (Es erscheint beim Öffnen: A mit UTF-8; A mit Gedit)'
print 'g4 schreibt A in Datei AE-utf-8-encoded-as-utf8.txt' \
      ' (very short file (no magic)) gemäss file *\n' \
      ' (Es erscheint beim Öffnen: A mit UTF-8; A mit Gedit)'
for line in f2: #f2 thought input was utf-8
    g3.write(line.upper())
    g4.write(line.upper()) #output as utf-8
    print "f2 Type:", type(line) #output as l1
    print "f Canonical repr(line.upper()):", "==>",repr(line.upper()
    print "f Canonical repr(line):", "==>",repr(line), "<=="
    print "f Printed line.upper():","==>",line.upper(), "<=="
    print "f Printed line:","==>",line, "<==\n"

print type(f), type(f1), type(f2),

f1.close()
g1.close()
f2.close()
g2.close()
```

```
Hexadezimale Representation des Speicherinhalts für:
ä: ISO8859-1=E4; Unicode=U+00E4(228); utf-8=c3 a4(11000011
 10100100)
Ä: ISO8859-1=C4; Unicode=U+00C4(196); utf-8=c3 84(11000011
 10000100)
ü: ISO8859-1=FC; Unicode=U+00FC(252); utf-8=c3 bc
Ü: ISO8859-1=DC; Unicode=U+00DC(220); utf-8=c3 9c
ö: ISO8859-1=F6; Unicode=U+00F6(246); utf-8=c3 b6
Ö: ISO8859-1=D6; Unicode=U+00D6(214); utf-8=c3 96

Datei ae-l1.txt hat Western(ISO-8859-15)= l1 Kodierung und
 enthält die Zeichen a\E4 (Gedit), also aä; a@ mit utf-u8

f ist vom Typ String und übernimmt a\E4. E4 wird jedochnicht
 erkannt von der upper() Methode und ignoriert
g schreibt A\E4 in Datei AE-encoded.txt (ISO-8859 text)
 gemäss file *
 (Es erscheint beim Öffnen: A@ mit utf-8; A\E4 mit Gedit)
f Type: <type 'str'>
f Canonical repr(line.upper()): ==> 'A\xe4\n' <==
f Canonical repr(line): ==> 'a\xe4\n' <==
f Printed line.upper(): ==> A@
<==
f Printed line: ==> a@
<==
```

```
f1 ist von Typ unicode. f1 wusste, dass Input l1 kodiert ist,
  E4 wird von upper()Methode erkannt und auf C4 geändert
g1 schreibt A\C4 in Datei AE-l1-encoded-as-l1.txt (ISO-8859
  text) gemäss file *
  (Es erscheint beim Öffnen: A mit UTF-8; A\C4 mit Gedit)
g2 schreibt AÄ in Datei AE-l1-encoded-as-utf8.txt (UTF-8
  Unicode text) gemäss file *
  (Es erscheint beim Öffnen: AÄ mit UTF-8; AÄ mit Gedit)
f1 Type: <type 'unicode'>
f Canonical repr(line.upper()): ==> u'A\xc4\n' <==
f Canonical repr(line): ==> u'a\xe4\n' <==
f Printed line.upper(): ==> AÄ
<==
f Printed line: ==> aä
<==

f2 ist von Typ unicode. f2 dachte Input sei utf-8 und
  ignorierte das unbekannte ä
g3 schreibt A in Datei AE-utf-8-encoded-as-l1.txt (very short
  file (no magic)) gemäss file *
  (Es erscheint beim Öffnen: A mit UTF-8; A mit Gedit)
g4 schreibt A in Datei AE-utf-8-encoded-as-utf8.txt (very
  short file (no magic)) gemäss file *
  (Es erscheint beim Öffnen: A mit UTF-8; A mit Gedit)
f2 Type: <type 'unicode'>
f Canonical repr(line.upper()): ==> u'A' <==
f Canonical repr(line): ==> u'a' <==
f Printed line.upper(): ==> A <==
f Printed line: ==> a <==

<type 'file'> <type 'instance'> <type 'instance'>
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

#ohne die coding Zeile gibt es eine Fehlermeldung, wenn diese
#Datei einen Umlaut enthält (wie in enthält)
#(line 4 SyntaxError: Non-ASCII character '\xc3' in file)
#Roland Benz

#Die unicode, L1 und utf-8 Tabellen enthalten die Ascii Tabelle
#Die unicode Tabelle enthält die L1 Tabelle (oder zumindest Teile d
#Die unicode und utf-8 Tabellen ähneln zwischen den Zeichen 128-191
#utf-8 hat in diesem Bereich lediglich eine c2 oder c3 davor.

def tell_me_about(s): return (type(s), s)

#A plain string
#---------------
v = "\xC4pple" # type string
u = u'\xc4pple' # type unicode

print "0:", tell_me_about(v)
#(<type 'str'>, '\xc4pple')

print "1:",tell_me_about(u)
#(<type 'unicode'>, u'\xc4pple')
```

```
#Versuch den String als utf-8 interpretiert(encoded) darzustellen
print "2:", repr(v), str(v)
#'\xc4pple' pple
#aber im Interpreter:
#>>> str('\xC4pple')
#'\xc4pple'
#>>> print str('\xC4pple')
#pple

print "3:", repr(u)
#u'\xc4pple'

#Versuch aus einem Typ unicode einen String darzustellen
#scheitert daran, dass ascii ein Problem hat.
#print "3:",str(u)
# UnicodeEncodeError: 'ascii' codec can't encode character
# u'\xc4' in position 0: ordinal not in range(128)

#Versuch den String als utf-8 interpretiert(encoded) darzustellen
print "4:", v
#pple

#Der Typ unicode wird korrekt verarbeitet
print "5:", u
#Äpple
```

```
#Decoding a iso8859-1 string - convert plain string to unicode
#---------------
#decode string to unicode with L1 table
uv = v.decode("iso-8859-1") #benützt L1 Tabelle
print "6:", repr(uv)
#u'\xc4pple'      #c4 ist Ä in unicode (und L1)

print "7:", tell_me_about(uv)
#(<type 'unicode'>, u'\xc4pple')

#Der Typ unicode wird korrekt verarbeitet
print "8:",uv,  "\xC4pple".decode("iso-8859-1")
#Äpple Äpple

print "9:", v.decode('iso-8859-1') == u'\xc4pple'
#True

#Versuch den String mit utf-8 Tabelle zu decoden schlägt fehl
#uv1 = v.decode("utf-8")
#print "9:", repr(uv)
# return codecs.utf_8_decode(input, errors, True)
# UnicodeDecodeError: 'utf8' codec can't decode byte 0xc4
# in position 0: invalid continuation byte

#decode String (in utf-8) to unicode with (useless?) L1 table
#Es wird ein Typ unicode erzeugt mit utf-8 Kodierung
uv2 = "Äpple".decode("iso-8859-1")
print "10:", repr(uv2), repr("Äpple")
#u'\xc3\x84pple'    #c3 84 ist utf-8 für Ä
```

```
print "11:", tell_me_about(uv2)
#(<type 'unicode'>, u'\xc3\x84pple')

#decode String (in utf-8) to unicode with utf-8 table
#Es wird ein Typ unicode erzeugt mit unicode (und L1) Kodierung
uv3 = "Äpple".decode("utf-8")
print "12:", repr(uv3)
#u'\xc4pple'        #c4 ist Ä in unicode (und L1)

print "13:", tell_me_about(uv3)
#(<type 'unicode'>, u'\xc4pple')


#A little more illustration – with "Ä"
#---------------
#Typ unicode ist gleich Typ unicode
print "14:", u"Ä" == u"\xc4", repr(u"Ä"), repr(u"\xc4")
#True u'\xc4' u'\xc4'

#Typ String in utf-8 ist nicht gleich Typ unicode
print "15:", "Ä" == u"\xc4", repr("Ä"), repr(u"\xc4")
#False '\xc3\x84' u'\xc4'

#Typ String in utf-8 ist gleich Typ unicode im Ascii Bereich
print "16:", "a" == u"a", repr("a"), repr(u"a")

#Typ String (in utf-8) in Typ unicode dekodiert ist gleich Typ unic
print "17:", "Ä".decode('utf8') == u"\xc4", \
    repr("Ä".decode('utf-8')), repr(u"\xc4")
#True u'\xc4' u'\xc4'
```

```
#Typ String (in utf-8) ist ungleich Typ String (in L1, unicode)
print "18:", "Ä" == "\xc4"
#False

#Encode-Decode
#.------------
#Decode: Typ String --> Typ Unicode
#Encode: Typ Unicode --> Typ String

#Encoding to UTF
#---------------
#v = "\xC4pple" # type string
# convert Type String (in iso-8859-1) to Type unicode
# and to Type String (in utf-8)
u8 = v.decode("iso-8859-1").encode("utf-8")
print "19:",repr(u8), str(u8), u8
#'\xc3\x84pple' Äpple Äpple

print "20:",tell_me_about(u8)
#(<type 'str'>, '\xc3\x84pple')

#Convert to String in utf-16
u16 = v.decode('iso-8859-1').encode('utf-16')
print "21:",tell_me_about(u16), str(u16), u16
#(<type 'str'>, '\xff\xfe\xc4\x00p\x00p\x00l\x00e\x00')
#  ppl e  ppl e

#Convert to Type unicode
print "22:",tell_me_about(u8.decode('utf8'))
#(<type 'unicode'>, u'\xc4pple')
```

```python
#Convert to Type unicode
print "23:",tell_me_about(u16.decode('utf16'))
#(<type 'unicode'>, u'\xc4pple')

#Relationship between unicode and UTF and latin1
#----------------
#Typ String in utf-8
print "24:", u8
#Äpple

#Typ unicode
print "25:",u8.decode('utf-8') # printing unicode
#Äpple

#Typ String in utf-16, print statement interprets u16 as utf-8
print "26:",u16     # printing 'bytes' of u16
#ÿþÄpple

#Typ unicode
print "27:",u16.decode('utf16')
#Äpple              # printing unicode

#Typ String in L1 vs Typ String in utf-8
print "28:",v == u8
#False              # v is a iso8859-1 string; u8 is a utf-8 string

#Typ unicode vs Typ String in utf-8
print "29:",v.decode('iso8859-1') == u8
#False              # v.decode(...) returns unicode
```

```python
#Strings decode to the same representation in Typ unicode
print "30:",u8.decode('utf-8') == v.decode('latin1') == u16.decode(
#True              # all decode to the same unicode memory represen
                   # (latin1 is iso-8859-1)

#Unicode Exceptions
#----------------
#Versuch Typ String in utf-8 direkt nach Typ String in l1 zu kodier
print "31:", u8.encode('iso8859-1')
#Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
#UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in positi
#  ordinal not in range(128)

print "32:", u16.encode('iso8859-1')
#Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
#UnicodeDecodeError: 'ascii' codec can't decode byte 0xff in positi
#  ordinal not in range(128)

v.encode('iso8859-1')
#Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
#UnicodeDecodeError: 'ascii' codec can't decode byte 0xc4 in positi
#  ordinal not in range(128)
```

```
0: (<type 'str'>, '\xc4pple')
1: (<type 'unicode'>, u'\xc4pple')
2: '\xc4pple' ÿþpple
3: u'\xc4pple'
4: ÿþpple
5: Äpple
6: u'\xc4pple'
7: (<type 'unicode'>, u'\xc4pple')
8: Äpple Äpple
9: True
10: u'\xc3\x84pple' '\xc3\x84pple'
11: (<type 'unicode'>, u'\xc3\x84pple')
12: u'\xc4pple'
13: (<type 'unicode'>, u'\xc4pple')
14: True u'\xc4' u'\xc4'
15: False '\xc3\x84' u'\xc4'
16: True 'a' u'a'
17: True u'\xc4' u'\xc4'
18: False
19: '\xc3\x84pple' Äpple Äpple
20: (<type 'str'>, '\xc3\x84pple')
21: (<type 'str'>, '\xff\xfe\xc4\x00p\x00p\x00l\x00e\x00')
 ÿþ p p l e  ÿþ p p l e
22: (<type 'unicode'>, u'\xc4pple')
23: (<type 'unicode'>, u'\xc4pple')
24: Äpple
25: Äpple
26: ÿþ p p l e
27: Äpple
28: False
29: False
30: True
31:
```

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

# Representation of strings in UTF-8 encoded files

print "Length of  'a':", len('a'), "Canonical:", repr('a')
print "Length of  'ä':", len('ä'), "Canonical:", repr('ä')
print "Length of u'a':", len(u'a'), "Canonical:", repr(u'a')
print "Length of u'ä':", len(u'ä'), "Canonical:", repr(u'ä')
```

```
 /str_representation_utf8.py"
Length of  'a': 1 Canonical: 'a'
Length of  'ä': 2 Canonical: '\xc3\xa4'
Length of u'a': 1 Canonical: u'a'
Length of u'ä': 1 Canonical: u'\xe4'

Process finished with exit code 0
```

```python
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

# simon.clematide@uzh.ch
# PCL I

# Representation of in latin-1 encoded files

print "length of  'a': ", len('a'), " Canonical", repr('a')
print "length of  'ä': ", len('ä'), " Canonical", repr('ä')
print "length of u'a': ", len(u'a'), " Canonical", repr(u'a')
print "length of u'ä': ", len(u'ä'), " Canonical", repr(u'ä')
```

```
 /str_representation_l1.py"
length of  'a':  1  Canonical 'a'
length of  'ä':  1  Canonical '\xe4'
length of u'a':  1  Canonical u'a'
length of u'ä':  1  Canonical u'\xe4'

Process finished with exit code 0
```

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

# Data type unicode

# Einzeilige (!) Zeichenkette
# mit Escape-Sequenzen
s1 = u"\u20aca\n\xe4"
print "Canonical s1: ",repr(s1)
print "Printed   s1: ",s1

# Rohe Sequenz ur"..."
# \uNNNN werden aufgelöst!
s2 = ur'\u20aca\n\xe4'
print "Canonical s2: ",repr(s2)
print "Printed   s2: ",s2

# Longstring
s3 = u"""\u20aca
ä"""
print "Canonical s3: ",repr(s3)
print "Printed   s3: ",s3

# Roher Longstring
s4 = ur"""€a
\u00e4"""
print "Canonical s4: ",repr(s4)
print "Printed   s4: ",s4

print
print "Type of s1:", type(s1)
print "Type of s2:", type(s2)
print "Type of s3:", type(s3)
print "Type of s4:", type(s4)

print
print "Test: s1 == s2 Result:", s1 == s2
print "Test: s1 == s3 Result:", s1 == s3
print "Test: s2 == s4 Result:", s2 == s4
print "Test: s3 == s4 Result:", s3 == s4
```

```
 /unicode_literals.py"
Canonical s1:  u'\u20aca\n\xe4'
Printed   s1:  €a
ä
Canonical s2:  u'\u20aca\\n\\xe4'
Printed   s2:  €a\n\xe4
Canonical s3:  u'\u20aca\n\xe4'
Printed   s3:  €a
ä
Canonical s4:  u'\u20aca\n\xe4'
Printed   s4:  €a
ä

Type of s1: <type 'unicode'>
Type of s2: <type 'unicode'>
Type of s3: <type 'unicode'>
Type of s4: <type 'unicode'>

Test: s1 == s2 Result: False
Test: s1 == s3 Result: True
Test: s2 == s4 Result: False
Test: s3 == s4 Result: True
```

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

# Data type str

# Einzeilige (!) Zeichenkette
# mit Escape-Sequenzen
s1 = "a\n\x61"
print "Canonical s1: ",repr(s1)
print "Printed   s1: ",s1

# Rohe Sequenz r"..."
# ohne Escapes
s2 = r"a\n\x61"
print "Canonical s2: ",repr(s2)
print "Printed   s2: ",s2

# Longstring
s3 = """a
a"""
print "Canonical s3: ",repr(s3)
print "Printed   s3: ",s3

# Rohe Longstring
s4 = r"""a
a"""
print "Canonical s4: ",repr(s4)
print "Printed   s4: ",s4

print
print "Type of s1:", type(s1)
print "Type of s2:", type(s2)
print "Type of s3:", type(s3)
print "Type of s4:", type(s4)

print
print "Test: s1 == s2 Result:", s1 == s2
print "Test: s1 == s3 Result:", s1 == s3
print "Test: s2 == s4 Result:", s2 == s4
print "Test: s1 == s4 Result:", s1 == s4
```

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl07-vorspann/str_literals
 .py"
Canonical s1:  'a\na'
Printed   s1:  a
a
Canonical s2:  'a\\n\\x61'
Printed   s2:  a\n\x61
Canonical s3:  'a\na'
Printed   s3:  a
a
Canonical s4:  'a\na'
Printed   s4:  a
a

Type of s1: <type 'str'>
Type of s2: <type 'str'>
Type of s3: <type 'str'>
Type of s4: <type 'str'>

Test: s1 == s2 Result: False
Test: s1 == s3 Result: True
Test: s2 == s4 Result: False
Test: s1 == s4 Result: True

Process finished with exit code 0
```

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import re

text = u"Viele Köche verderben den Brei."
pattern_ur = ur"(\w+)"
pattern_u = u"(\w+)"
pattern_r = r"(\w+)"
pattern = "(\w+)"
print "\n",pattern_ur, pattern_u, pattern_r, pattern,"\n"
print repr(pattern_ur), repr(pattern_u), repr(pattern_r),\
    repr(pattern),"\n"
# Alle Matches finden
m_ur = re.findall(pattern_ur, text)
m_u = re.findall(pattern_u, text)
m_r = re.findall(pattern_r, text)
m = re.findall(pattern, text)
for g in m_ur:
    print g
print ""
for g in m_u:
    print g
print ""
for g in m_r:
    print g
print ""
for g in m:
    print g
```

```
 /re_findall_flag.py"

(\w+) (\w+) (\w+) (\w+)

u'(\\w+)' u'(\\w+)' '(\\w+)' '(\\w+)'

Viele
K
che
verderben
den
Brei

Viele
K
che
verderben
den
Brei

Viele
K
che
verderben
den
Brei

Viele
K
che
verderben
den
Brei
```

ö ist nicht in \w

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

import re

text = u"Viele Köche verderben den Brei."
pattern = ur"(\w+)"

# Das Flag (?u) aktiviert Unicode-Kategorien fuer \w und \b
pattern = ur"(?u)(\w+)"

# Resultat ist eine Liste
m = re.findall(pattern, text)

for s in m:
    print s
```

```
 /re_findall_flag_u.py"
Viele
Köche
verderben
den
Brei
```

ur"(?u)(\w+)"(?u) nun ist \w nicht

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import re

text = "Hässliche Köche verdürben das Gebräu" #utf8
text2 = u"Hässliche Köche verdürben das Gebräu" #unicode

pattern = r"([aeioäöü]+)" #utf-8
pattern2 = ur"([aeioäöü]+)" #unicode

print "\n",repr(text),repr(pattern),repr(text2),repr(pattern2)

# Im Ersetzungstext können gematchte Gruppen eingefügt werden.
# \N (N ist die N-te gruppierende Klammer im Pattern)
replacement = r"[\1]"
replacement2 = ur"[\1]"

print "\n",repr(replacement),repr(replacement2),"\n"

#pattern und text müssen beide Typ String oder beide Typ unicode
#sein, um die Umlaute zu finden
print re.sub(pattern, replacement, text) #alle
print re.sub(pattern, replacement, text2) #ohne äöü
print re.sub(pattern, replacement2, text) #alle
print re.sub(pattern2, replacement, text) #ohne äöü
print re.sub(pattern, replacement2, text2) #ohne äöü
print re.sub(pattern2, replacement2, text) #ohne äöü
print re.sub(pattern2, replacement, text2) #alle
print re.sub(pattern2, replacement2, text2) #alle
```

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl07-vorspann/re_sub.py"

'H\xc3\xa4ssliche K\xc3\xb6che verd\xc3\xbcrben das
 Gebr\xc3\xa4u' '([aeio\xc3\xa4\xc3\xb6\xc3\xbc]+)'
 u'H\xe4ssliche K\xf6che verd\xfcrben das Gebr\xe4u'
 u'([aeio\xe4\xf6\xfc]+)'

'[\\1]' u'[\\1]'

H[ä]ssl[i]ch[e] K[ö]ch[e] v[e]rd[ü]rb[e]n d[a]s G[e]br[ä]u
Hässl[i]ch[e] Köch[e] v[e]rdürb[e]n d[a]s G[e]bräu
H[ä]ssl[i]ch[e] K[ö]ch[e] v[e]rd[ü]rb[e]n d[a]s G[e]br[ä]u
Hässl[i]ch[e] Köch[e] v[e]rdürb[e]n d[a]s G[e]bräu
Hässl[i]ch[e] Köch[e] v[e]rdürb[e]n d[a]s G[e]bräu
Hässl[i]ch[e] Köch[e] v[e]rdürb[e]n d[a]s G[e]bräu
H[ä]ssl[i]ch[e] K[ö]ch[e] v[e]rd[ü]rb[e]n d[a]s G[e]br[ä]u
H[ä]ssl[i]ch[e] K[ö]ch[e] v[e]rd[ü]rb[e]n d[a]s G[e]br[ä]u
```

text und patternmüssen dieselbe Kodierung

Pattern immer mitr"" für text ""oderur"" für te

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I


import re

#Vorsicht: Innerhalb einer Klasse [.] hat der Punkt die
#Bedeutung eines Punktes und nicht jedes Zeichens ausser \n
text = 'my email is simon.clematide@uzh.ch and yours?'
pattern = r'([\w.]+)@([\w.]+)'

m = re.search(pattern, text)

# get whole match
print "Whole match:", m.group()

# get the first matched group
print "First group:", m.group(1)

# get the second matched group
print "Second group:", m.group(2)

# What is m?
print m
```

```
 /re_search_group.py"
Whole match: simon.clematide@uzh.ch
First group: simon.clematide
Second group: uzh.ch
<_sre.SRE_Match object at 0xb74a2f08>
```

I need to extract all letters *after* the `+` sign or at the beginning of a string like this:

```
formula = "X+BC+DAF"
```

I tried so, and I do not want to see the `+` sign in the result. I wish see only `['X', 'B', 'D']`.

```
>>> re.findall("^[A-Z]|[+][A-Z]", formula)
['X', '+B', '+D']
```

When I grouped with parenthesis, I got this strange result:

```
re.findall("^([A-Z])|[+]([A-Z])", formula)
[('X', ''), ('', 'B'), ('', 'D')]
```

Why it created tuples when I try to group ? How to write the regexp directly such that it returns `['X', 'B', 'D']` ?

If there are any capturing groups in the regular expression then `re.findall` returns only the values captured by the groups. If there are no groups the entire matched string is returned.

`re.findall(pattern, string, flags=0)`

Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found. **If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group.** Empty matches are included in the result unless they touch the beginning of another match.

How to write the regexp directly such that it returns ['X', 'B', 'D'] ?

Instead of using a capturing group you can use a non-capturing group:

```
>>> re.findall(r"(?:^|\+)([A-Z])", formula)
['X', 'B', 'D']
```

Or for this specific case you could try a simpler solution using a word boundary:

```
>>> re.findall(r"\b[A-Z]", formula)
['X', 'B', 'D']
```

Or a solution using `str.split` that doesn't use regular expressions:

```
>>> [s[0] for s in formula.split('+')]
['X', 'B', 'D']
```

```python
# -*- coding: utf-8 -*-

# simon.clematide@uzh.ch
# PCL I

import re
#Achtung beim Bilden von Gruppen (). Gibt es capturing groups ()
#dann werden nur die Werte der capturing groups zurückgegeben.
#Sollen alle Matches zurückgegeben werden müssen alle Gruppen
#non capturing groups (?: ) sein.
text = "That U.S.A. poster-print costs $12.40... per m^2."
pattern = r'''(?x)          # set flag (?x) to allow verbose regexes
      (?:[A-Z]\.)+          # abbreviations, e.g. U.S.A.
    | \$?\d+(?:[.,]\d+)*%?  # currency/percentages, $12.40, 82%
    | \w+(?:-\w+)*          # words with optional internal hyphens
    | \.\.\.                # ellipsis
    | [.,;?]+               # punctuation
    | \S+                   # catch-all for non-layout characters
    '''
m = re.findall(pattern,text)
print m
```

```
 /re_findall_tokenizer.py"
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...',
 'per', 'm', '^2.']
```

Interne Hilfe

```
>>> help(len)
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or collection.
```

Erklärung der Erklärung

- Die Signatur (*signature*) beschreibt die Typen der Funktionsargumente und des resultierenden Funktionswerts
- Bedeutung wird informell erklärt.
- Mehr Info in der Online-Doku
  `https://docs.python.org/2/library/functions.html?#len`.

Hilfe Modus: Methoden/Funktionen

>>> **help(dict)**

Help on class dict in module __builtin__:

```
Hilfe Modus: help()
>>>help()
modules
help>__builtin__
help>q
>>>
```

```
Hilfe Modus: Topics/Module
>>>help("topics")
>>>help("DICTIONARIES")
Mapping Types --- "dict"
>>>help("modules")
>>>help("__builtin__")
>>>
```

```
>>> " sep ".join(["1","2","3","4"])

'1 sep 2 sep 3 sep 4'
```

```
>>> dict([("a",1),("2","b")])
{'a': 1, '2': 'b'}
>>> dict([("a",1),("b","2")])
{'a': 1, 'b': '2'}
>>> {'a': 1, 'b': '2'}
{'a': 1, 'b': '2'}
>>> dict(a=1, b="2")
{'a': 1, 'b': '2'}
>>> idict=dict()
>>> idict["a"]=1
>>> idict.update({"b":"2"})
{'a': 1, 'b': '2'}
```

Interne Hilfe

```
>>> help(re.split)
Help on function split in module re:

split(pattern, string, maxsplit=0, flags=0)
    Split the source string by the occurrences of the pattern,
    returning a list containing the resulting substrings.
```

Erklärung der Erklärung

- *Name=Wert* zeigt Standardwert optionaler Argumente.
- Ergebnis- und Argumenttypen werden im Fliesstext informell erklärt.
- Mehr Info in der Online-Doku `http://docs.python.org/2/`.

```
Erst modul importieren:
>>>import re
>>>help(re.split)

Oder via modul help
>>>help("re")
```

Python ist eine objektorientierte Programmiersprache. ▶

Alle Daten (Werte, Datenstrukturen) in Python sind Objekte.

Python ist eine dynamisch getypte Programmiersprache. ▶

Alle Objekte haben einen Typ.

- Warum "dynamisch"? Der Typ einer Variablen wird nicht statisch im Quelltext deklariert, er wird dynamisch zur Laufzeit bestimmt.

Alle Objekte haben einen Typ

Eingebaute Funktion `type()`

Sie bestimmt den Typ von jedem Objekt.

Typen bestimmen

```
>>> type(1)
>>> x = 1+3*42
>>> type(x)
>>> type("ABBA")
>>> type("AB"+'BA')
>>> type("A" == "a")
>>> type(['a','b'])
>>> type(['a','b'][0])
>>> type({})
>>> type(re.search('X','aaa'))
>>> type(None)
>>> type(re.search('a','aaa'))
```

```
>>> import re
print(type(1))
x=1+3*42
print("1",type(x))
print("2",type("ABBA"))
print("3",type("AB"+'BA'))
print("4",type("A"=="a"))
print("5",type(['a','b']))
print("6",type(['a','b'][0]))
print("7",type({}))
print("8",type(re.search('X','aaa')))
print("9",type(None))
print("10",type(re.search('a','aaa')))
<type 'int'>
('1', <type 'int'>)
('2', <type 'str'>)
('3', <type 'str'>)
('4', <type 'bool'>)
('5', <type 'list'>)
('6', <type 'str'>)
('7', <type 'dict'>)
('8', <type 'NoneType'>)
('9', <type 'NoneType'>)
('10', <type '_sre.SRE_Match'>)
```

Libre Office Finden und Ersetzen mit REGEX

\n will match a newline (Shift-Enter)
$ on its own will match a paragraph mark (Enter)
[ ] will match a space

\<([^]+)[ ]+\1 finds duplicate words separated by spaces

Die eingebaute Funktion `repr()`

Erzeugt eine kanonische Zeichenkette aus jedem Objekt.

```
>>> repr("a")
"'a'"
>>> repr(['a',"b"])
"['a', 'b']"
```

Read-Eval-Print-Loop (REPL)▸2

Im interaktiven Gebrauch wird der zuletzt eingegebene Ausdruck (*expression*) evaluiert und das Resultat als kanonischer String mit `print` ausgegeben.

```
>>> ['a',"b"]
"['a', 'b']"
>>> print repr(['a',"b"])
 ['a', 'b']
```

```
>>> repr("a")
"'a'"
>>> repr("a")
"'a'"
>>> repr('a')
"'a'"
>>> repr(['a',"b"])
"['a', 'b']"
>>> ['a',"b"]
['a', 'b']
>>> print repr(['a',"b"])
['a', 'b']
```

Objekte haben Methoden.

Methoden sind Funktionen, die von einem Objekt aus aufgerufen werden.

Methoden aufrufen

Methoden-Aufrufe (*invocation*) evaluieren zu einem Objekt.
Mit entsprechenden Argumenten!

```
>>> "ABBA".count("B")
2
>>> "ABBA".lower()
'abba'
```

Methoden sind Objekte

Methoden-Namen evaluieren zu einem abstrakten Objekt.

```
>>> "ABBA".count
<built-in method count of
 str object at 0x10f743cc0>
```

Objekte haben benannte Attribute.

Punktnotation: OBJECT.ATTRIBUT
Attribute sind Objekte.

Methoden = aufrufbare Attribute

Eine Methode ist ein aufrufbares Attribut eines Objekts.
Eine Funktion ist ein aufrufbares Objekt eines Moduls.

Dokumentationsstrings von Funktionen als Attribut `__doc__`

```
>>> len.__doc__
'len(object) -> integer\n\nReturn the number of items of a sequence or coll
>>> help(len)
Help on built-in function len in module __builtin__:
len(...)
    len(object) -> integer
    Return the number of items of a sequence or collection.
```

Zeichenkodierungen und Zeichensätze

Einschränkungen von ASCII

Die weitverbreiteste Zeichenkodierung mit 128 Kodes (7-Bit) unterstützt keine nicht-englischen Buchstaben.

Verschiedene Erweiterungen mit 8-Bit Kodierungen

Zeichenkodierungen mit 256 Kodes (1 Byte) für verschiedene Alphabete oder Betriebssysteme:
ISO-8859-1, ISO-8859-9, Mac-Roman, Windows-1252

Universale Lösung Unicode

Unicode weist jedem Zeichen einen eindeutigen Zahlen-Kode und eine Kategorie zu.
Unicode 6.0 definiert 109'449 graphische Zeichen

Speicher- und Transportformat UTF-8

Persistente Speicherung und Datenübertragung mit Unicode

UTF (Abk. für *Unicode Transformation Format*) beschreibt Methoden, einen Unicode-Wert auf eine Folge von Bytes abzubilden.

Beispiele für UTF-8-Kodierungen

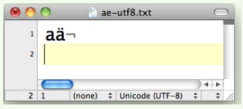
Zeichen	Unicode	Unicode binär	UTF-8 binär	UTF-8 hexadezimal
Buchstabe *y*	U+0079	00000000 01111001	01111001	0x79
Buchstabe *ä*	U+00E4	00000000 11100100	11000011 10100100	0xC3 0xA4
Eingetragene Marke ®	U+00AE	00000000 10101110	11000010 10101110	0xC2 0xAE
Eurozeichen €	U+20AC	00100000 10101100	11100010 10000010 10101100	0xE2 0x82 0xAC

Quelle: http://de.wikipedia.org/wiki/UTF-8

Textdatei als Bytefolge

Die Repräsentation der Zeichen mit Kodes > 127 sind unterschiedlich.

Datei in UTF-8-Kodierung

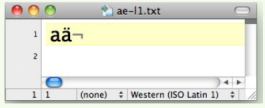


ä = 2 Bytes = C3 A4

```
$ hexdump  ae-utf8.txt
0000000 61 c3 a4 0a
0000004
```

Datei mit 4 Bytes

Datei in Latin-1-Kodierung



ä = 1 Byte = E4

```
$ hexdump  ae-l1.txt
0000000 61 e4 0a
0000003
```

Datei mit 3 Bytes

```
$ grep . AE-l1-encoded-as-l1.txt
A
$ sed -n 1,2p AE-l1-encoded-as-l1.txt
A
$ hexdump -x -c -C AE-l1-encoded-as-l1.txt
0000000    c441    000a
0000000   A   \n
00000000 41 c4 0a                                |A..|
00000003

$ grep . AE-l1-encoded-as-utf8.txt
AÄ
$ sed -n 1,2p AE-l1-encoded-as-utf8.txt
AÄ
$ hexdump -x -c -C AE-l1-encoded-as-utf8.txt
0000000    c341    0a84
0000000   A  204   \n
00000000 41 c3 84 0a                             |A...|

$ echo aä | hexdump
0000000 c361 0aa4
$ echo aä | hexdump -C
00000000 61 c3 a4 0a                             |a...|
echo aä | xxd -b
0000000: 01100001 11000011 10100100 00001010     a...
```

```
$ echo abc | hexdump -e '/1 "% _ax) "' -e '/1 "%02X" "\n"'
0) 61
1) 62
2) 63
3) 0A
```

```
U+0061 a 61
U+00E4 ä c3 a4
U+0041 A 41
U+00C4 Ä c3 84

$ echo aäAÄ | hexdump -x -c -C
0000000    c361    41a4    84c3    000a
0000000   a   A  204   \n
00000000 61 c3 a4 41 c3 84 0a                    |a..A...|
```

```
U+0031 1 31
U+0032 2 32
U+0038 8 38
$ echo 128 | xxd -b
0000000: 00110001 00110010 00111000 00001010                  128.
$ echo 128 | xxd
0000000: 3132 3830a                                           128.
$ echo 128 | hexdump -C
00000000  31 32 38 0a                                        |128.|
```

Zeichenkode berechnen aus Zeichen

```
>>> ord("A")
65
>>> ord('a')
97
>>> ord('\n')
10
>>> ord("\t")
9
>>> ord('\x20') # Hexadezimal
32
>>> ord("\'")
39
```

Zeichen berechnen aus Zeichenkode

```
>>> chr(65)
'A'
>>> chr(97)
'a'
>>> chr(10)
'\n'
>>> chr(9)
'\t'
>>> chr(32)
' '
>>> chr(39)
"'"
```

Buchstabe A

```
>>> ord("A")
65
>>> chr(65)
'A'
>>> hex(65)
'0x41'
>>> chr(0x41)
'A'
```

Buchstabe ä als Type unicode

```
>>> ord(u"ä")
228
>>> unichr(228)
u'\xe4'
>>> u'\xe4'
u'\xe4'
>>> print u'\xe4'
ä
>>> u'\xe4'.encode("utf-8")
'\xc3\xa4'
>>> print '\xc3\xa4'
ä
```

Buchstabe ä als Type String

```
>>> ord("ä")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: ord() expected a character, but string of length 2 found
>>> chr(228)
'\xe4'
>>> chr(0xE4)
'\xe4'
>>> print chr(0xE4)
>>> print '\xe4'
ä
```

Zahlen-Konversion in Terminal

```
$ echo $((0xE4))
228

$ echo 'ibase=16; E4'|bc
228

$ echo 'ibase=10; obase=16;228'|bc
E4
```

Zahlenkonversion in Python

```
>>> int('010110', 2)
22
>>> hex(22)
'0x16'
>>> hex(int('010110', 2))
'0x16'
>>> int('0x16', 16)
22
```

Datentyp bestimmen und testen

```
>>> type('ABBA')
<type 'str'>

>>> type("ABBA") == str
True
```

String-Literale notieren ▸3

```
# Einzeilige (!) Zeichenkette
# mit Escape-Sequenzen
s1 = "a\n\x61"
# Rohe Sequenz r"..."
# ohne Escapes
s2 = r"a\n\x61"
```

Unicode Zeichenkodes

```
>>> ord(u'€')
8364
>>> unichr(8364)
u'\u20ac'
```

Datentyp bestimmen und testen

```
>>> type(u'A')
<type 'unicode'>
>>> type(u'ab') == unicode
True
>>> isinstance('ab',unicode)
False
```

```
>>> isinstance(u"ab",unicode)Tr
```

String-Literale notieren ▸4

```
# Einzeilige (!) Zeichenkette
# mit Escape-Sequenzen
s1 = u"\u20aca\n\xe4"

# Rohe Sequenz ur"..."
# \uNNNN werden aufgelöst!
s2 = ur'\u20aca\n\xe4'

# Longstring
s3 = u"""\u20aca

# Roher Longstring
s4 = ur"""€a
\u00e4"""
```

Kodierung der Python-Quellkodes deklarieren

Kodierungskommentar für UTF-8-kodierte Quelltexte

Deklariere Kodierung immer mit Kodierungskommentar, wenn Nicht-ASCII-Zeichen vorkommen!

Datei in UTF-8-Kodierung

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

print "Length of 'a':", len('a'), "Canonical:", repr('a')
print "Length of 'ä':", len('ä'), "Canonical:", repr('ä')
print "Length of u'a':", len(u'a'), "Canonical:", repr(u'a')
print "Length of u'ä':", len(u'ä'), "Canonical:", repr(u'ä')
```

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

iso-8859-1 ist in Python 2 Standard.

Enkodieren und Dekodieren von Zeichenketten

Explizites Dekodieren von UTF-8-Repräsentation

```
>>> text = 'B\xc3\xa4h' # UTF-8-Repräsentation von Bäh
>>> unicodetext = text.decode('utf-8')
```

Es entsteht ein Unicode-Objekt!

Explizites Enkodieren von Unicode-Zeichen als UTF-8-Bytefolge

```
>>> unicode_text = u'Bäh'
>>> utf8_text = unicode_text.encode('utf-8')
```

Es entsteht eine Byte-String!

Das Modul codecs

codecs: Kodieren und Dekodieren

Funktionen für Lesen und Schreiben von Unicode-Strings

Einlesen von Latin-1 und schreiben von UTF-8

```
import codecs

# Decode from l1 encoded file into unicode strings
f = codecs.open("./ae-l1.txt", "r", "l1")

# Encode unicode strings into UTF-8 encoded file
g = codecs.open("./AE-l1-encoded-as-utf8.txt", "w", "utf-8")
for line in f:
    g.write(line.upper())
```

Beim Einlesen entstehen Zeichenketten vom Typ `unicode`.

Funktion `re.sub()`: Ersetzen mit regulären Ausdrücken

Globales Ersetzen mit Rückreferenz ▸8

```
import re

text = u"Hässliche Köche verdürben das Gebräu"

pattern = ur"([aeioäöü]+)"

# Im Ersetzungstext können gematchte Gruppen eingefügt werden.
# \N (N ist die N-te gruppierende Klammer im Pattern)
replacement = ur"[\1]"

print re.sub(pattern, replacement, text)
```

`replacement` ist eine Zeichenkette, kein regulärer Ausdruck! Falls nichts gematcht wird, bleibt die Zeichenkette unverändert!

Warum *raw strings* für Reguläre Ausdrücke in Python?

`r'REGEX'` oder `ur'REGEX'`

- Generell empfohlen in `http://docs.python.org/2/library/re.html`.
- Für viele Escape-Sequenzen macht es zwar keinen Unterschied, weil Python-Strings und Reguläre Ausdrücke letztlich dieselben Zeichen bedeuten: `\a,\f,\n,\r,\t,\v`
- Andere Escape-Sequenzen existieren nur in der Regulären Notation und Python lässt den Backslash stehen: `\A,\B,\w,\W,\d,\D,\s,\S`
- Aber andere Reguläre Notationen würden beim Einlesen von Nicht-Raw-Strings missinterpretiert: Um einen einzelnen Backslash zu matchen, müssten wir schreiben: `re.match("\\\\", "\\")`
- \b: Bell-Zeichen (ASCII-Code 8) im String; aber Grenze zwischen Wortzeichen und Nicht-Wortzeichen in Regex.
  `re.sub("\\bthe\\b","THE", "Other uses of the")`
- Numerische Rückreferenzen `\1`

Gruppierung mit/ohne Rückreferenzen

Runde Klammern ergeben referenzierbare Gruppierungen

```
>>> text = 'Blick-Leser, A-Post-Fans, andere Bindestrich-Komposita'
>>> re.sub(r'(\w+-)+(\w+)', r'\2', text)
```

Nicht alle Gruppen müssen referenzierbar sein! (Effizienzgründe!)

Nichtreferenzierbare Gruppierung: `(?:REGEX)`

```
>>> text = 'Blick-Leser, A-Post-Fans, andere Bindestrich-Komposita'
>>> re.sub(r'(?:\w+-)+(\w+)', r'\1', text)
```

```
>>> import re
>>> text = 'Blick-Leser, A-Post-Fans, andere Bindestrich-Komposita'
>>> re.sub(r'(\w+-)+(\w+)', r'\2', text)
'Leser, Fans, andere Komposita'
>>> re.sub(r'(?:\w+-)+(\w+)', r'\1', text)
'Blick-, Post-, andere Bindestrich-'.
>>> re.sub(r'(?:\w+-)+(?:\w+)', "jummy", text)
'jummy, jummy, andere jummy'
```

Funktion `re.findall()`: Globale Suche

Alle nicht-überlappenden Matches extrahieren

```
import re

text = u"Viele Köche verderben den Brei."
pattern = ur"(\w+)"

# Alle Matches finden
m = re.findall(pattern, text)

for g in m:
    print g
```

```
>>> import re
text = u"Viele Köche verderben den Brei."
pattern = ur"(\w+)"
# Alle Matches finden
m = re.findall(pattern, text)
for g in m:
    print g

Viele
K
che
verderben
den
Brei
```

Pattern und Text müssen immer vom gleichen String-Typ sein!

Regex-Alternative ist nicht kommutativ!

Die Reihenfolge in einer Regex-Alternative ist nicht beliebig!

```
import re
print re.findall(r'a|aa',"Saal")
print re.findall(r'aa|a',"Saal")
```

```
>>> import re
print re.findall(r'a|aa',"Saal")
print re.findall(r'aa|a',"Saal")
['a', 'a']
['aa']
```

Gruppierung mit/ohne Rückreferenzen: `re.findall()`

`re.findall()` und gruppierende Klammern

Unterschiedliche Funktionalität, falls gruppierende Klammern im regulären Ausdruck sind oder nicht!

- **Ohne**: Liste der Matches
- **Mit**: Liste von Tupeln, wobei jedes Tupel-Element den gematchten Inhalt der entsprechenden gruppierenden Klammer enthält.

```
>>> re.findall(r'a(h)|a(a)', "kahler Saal")
[('h', ''), ('', 'a')]
```

Was bedeutet das?

```
    kahler Saal
      |      |
    ('h',    '')  # 1. möglicher Match
    ('',     'a')  # 2. möglicher Match
```

```
>>> re.findall(r'a(h)|a(a)', "kahler Saal aataah brama")

[('h', ''), ('', 'a'), ('', 'a'), ('', 'a')]
>>> re.findall(r'a(h)|a(a)', "kahler Saal ahtaah brama")

[('h', ''), ('', 'a'), ('h', ''), ('', 'a')]
>>> re.findall(r'a(h).+a(a)', "kahler Saal ahtaah brama")

[('h', 'a')]
>>> re.findall(r'a(?:h).+a(?:a)', "kahler Saal ahtaah brama")

['ahler Saal ahtaa']
>>> re.findall(r'a(?:h)|a(?:a)', "kahler Saal ahtaah brama")

['ah', 'aa', 'ah', 'aa']
```

Letzes Beispiel: letztes aa wird gefunden, aber überschneidendes ah nicht.

Unicode Flag: Was ist ein Wortzeichen?

Unicode-Kategorien aktivieren

```
import re

text = u"Viele Köche verderben den Brei."
pattern = ur"(\w+)"

# Das Flag (?u) aktiviert Unicode-Kategorien fuer \w und \b
pattern = ur"(?u)(\w+)"

# Resultat ist eine Liste
m = re.findall(pattern, text)

for s in m:
    print s
```

```
>>> import re
text = u"Viele Köche verderben den Brei."
pattern = ur"(\w+)"
# Das Flag (?u) aktiviert Unicode-Kategorien fuer \w und \b
pattern = ur"(?u)(\w+)"
# Resultat ist eine Liste
m = re.findall(pattern, text)
for s in m:
    print s

Viele
Köche
verderben
den
Brei
```

Das Unicode-Flag `(?u)` zählt nicht als Gruppe wie alle `(?...)`.

Lesbare und kommentierte reguläre Ausdrücke

Was matcht dieser Ausdruck?

```
(?:[A-Z]\.)+|\w+(?:-\w+)*|\$?\d+[.\d]*%?|\.\.\.|[.,;?]+|\S+
```

Lesbare und kommentierbare Ausdrücke dank Flag `(?x)` ▸11

```
import re
text = "That U.S.A. poster-print costs $12.40..."
pattern = r'''(?x)          # set flag (?x) to allow verbose regexps
     (?:[A-Z]\.)+           # abbreviations, e.g. U.S.A.
   | \$?\d+(?:[.,]\d+)*%?   # currency/percentages, $12.40, 82%
   | \w+(?:-\w+)*           # words with optional internal hyphens
   | \.\.\.                 # ellipsis
   | [.,;?]+                # punctuation
   | \S+                    # catch-all for non-layout characters
   '''
m = re.findall(pattern,text)
print m
```

Single and double quoted strings in Python are identical. The only difference is that single-quoted strings can contain unescaped double quote characters, and vice versa. For example:

```
'a "quoted" word'
"another 'quoted' word"
```

Module vs Package==========A Python module is simply a Python sour

Keywords [edit]

Python has the following keywords or *reserved words*; they cannot be used as identifiers.[3][4]

- `and`
- `as`
- `assert`
- `break`
- `class`
- `continue`
- `def`
- `del`
- `elif`
- `else`
- `except`
- `exec` [note 1]
- `False` [note 2]
- `finally`
- `for`
- `from`
- `global`
- `if`
- `import`
- `in`
- `is`
- `lambda`
- `None`
- `nonlocal` [note 2]
- `not`
- `or`
- `pass`
- `print` [note 1]
- `raise`
- `return`
- `True` [note 2]
- `try`
- `while`
- `with`
- `yield`

Notes

1. ^ a b Starting from Python 3, `exec` and `print` are functions, so they are not keywords anymore.
2. ^ a b c Starting from Python 3, keywords `True`, `False` and `nonlocal` were introduced.

`foo` function in Python:

```
def foo(x):
    if x == 0:
        bar()
        baz()
    else:
        qux(x)
        foo(x - 1)
```

```
To illustrate, the following examples all return a dictionary equal
to "{"one": 1, "two": 2, "three": 3}":

   >>> a = dict(one=1, two=2, three=3)
   >>> b = {'one': 1, 'two': 2, 'three': 3}
   >>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
   >>> d = dict([('two', 2), ('one', 1), ('three', 3)])
   >>> e = dict({'three': 3, 'one': 1, 'two': 2})
   >>> a == b == c == d == e
   True
```

Anweisung (engl. Statement)(Programmierung)

Als **Anweisung** (engl. *statement*) bezeichnet man in der Informatik ein zentrales Element vieler imperativer Programmiersprachen. Die Programme derartiger Sprachen setzen sich primär aus einer oder mehreren Anweisungen zusammen. Eine Anweisung stellt eine in der Syntax einer Programmiersprache formulierte einzelne Vorschrift dar, die im Rahmen der Abarbeitung des Programms auszuführen ist. Wie eine Anweisung syntaktisch auszusehen hat, wird durch die jeweilige Programmiersprache bzw. deren Spezifikation festgelegt und ist innerhalb eines Programms nicht änderbar. Anweisungen sind üblicherweise Zuweisungen, Kontrollanweisungen (wie Sprünge, Schleifen und bedingte Anweisungen) und Prozeduraufrufe. Abhängig von der Programmiersprache sind teilweise auch Zusicherungen, Deklarationen, Klassen- und Funktionsdefinitionen Anweisungen.

Im Gegensatz zu Ausdrücken haben Anweisungen nicht immer einen Wert. Einige Anweisungen können jedoch auch Ausdrücke sein, so z. B. Zuweisungen, Inkrementoperatoren oder Funktionsaufrufe.

In computer programming, a **statement** is the smallest standalone element of an imperative programming language that expresses some action to be carried out. It is an instruction written in a high-level language that commands the computer to perform a specified action.[1] A program written in such a language is formed by a sequence of one or more statements. A statement may have internal components (e.g., expressions).

Many languages (e.g. C) make a distinction between statements and definitions, with a statement only containing executable code and a definition declaring an identifier, while an expression evaluates to a value only.[2] A distinction can also be made between simple and compound statements; the latter may contain statements as components.

Assembler-Anweisung	`MOV AX,BX`
Initialisierung	`TYPE SALARY = INTEGER`
Deklaration	`VAR A:INTEGER`
Zuweisung	`A := A + 1`
Block	`begin WRITE('Number? '); READLN(NUMBER); end`
Bedingte Anweisung	`if A > 3 then WRITELN(A) else WRITELN("NOT YET") end`
Switch-Case-Anweisung	`switch (c) { case 'a': alert(); break; case 'q': quit(); break; }`
While-do-Schleife	`while NOT EOF DO begin READLN end`
Do-while-Schleife	`do {computation(&i); } while (i < 10);`
For-Schleife	`for A:=1 to 10 do WRITELN(A) end`
Unterprogrammaufruf	`CLEARSCREEN()`
Return-Anweisung	`return 5;`
Goto-Anweisung	`goto 1`
Assertion	`assert(ptr != NULL);`

Expression (computer science)

An **expression** in a programming language is a combination of one or more explicit values, constants, variables, operators, and functions that the programming language interprets (according to its particular rules of precedence and of association) and computes to produce ("to return", in a stateful environment) another value. This process, as for mathematical expressions, is called evaluation. The returned value can be of various types, such as numerical, string, and logical.

In many programming languages a function, and hence an expression containing a function, may have side effects. An expression with side effects does not normally have the property of referential transparency. In many languages (e.g. C++), expressions may be ended with a semicolon (`;`) to turn the expression into an expression statement. This asks the implementation to evaluate the expression for its side-effects only, and to disregard the result of the expression.

Statement/Anweisung (Instruction/Command/Befehl)

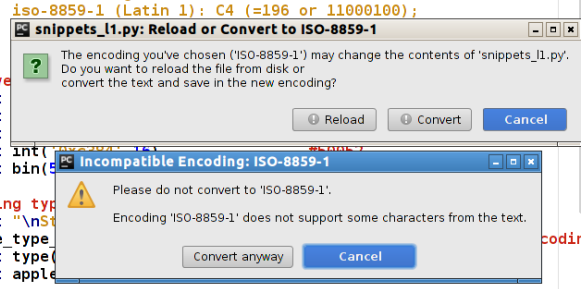
- Nur Seiteneffekte, retourniert keinen Wert
- Kann Ausdrücke enthalten
- Definition: Von Variablen, von Funktionen (inkl. Deklaration des Funktionsprototypen/der Signatur)
- Funktionsaufruf

Expression/Ausdruck:

- Evaluiert und tetourniert einen Wert
- Hat immer einen Wert

Packages:Files named __init__.py are used to mark directories on disk as Python package directories. If you have the filesmydir/spam/__init__

```
#!/usr/bin/python

#Encoding for letter Ä
#http://www.utf8-zeichentabelle.de/
print """\nEncodings for letter Ä:
      iso-8859-1 (Latin 1): C4 (=196 or 11000100);
```




```
#String type str with utf-8 encoding
```

```
#!/usr/bin/python
# -*- coding: utf-8 -*-


# import sys
# reload(sys)
# sys.setdefaultencoding('utf-8')


import sys, locale, os
print(sys.stdout.encoding)
print(sys.stdout.isatty())
print(locale.getpreferredencoding())
print(sys.getfilesystemencoding())
print(os.environ["PYTHONIOENCODING"])
print(chr(246), unichr(9786), unichr(9787)),"\n"

print """File is encoded with utf-8
      File Coding needs to specified (see 2. line)
      so that python is able to read letter Ä"""

#Encoding for letter Ä
#http://www.utf8-zeichentabelle.de/
print """\nEncodings for letter Ä:
      iso-8859-1 (Latin 1): C4 (=196 or 11000100);
      unicode             : U+00C4 (=196 or 11000100);
      utf-8               : c3 84 (=50052 or 11000011 10000100)"""
```

```
#Conversion between hex, decimal and binary
print "\nConversion between hex, decimal and binary"
print int('0xc4',16)                    #196
print bin(196)                          #0b11000100
print int('0xc384',16)                  #50052
print bin(50052)                        #0b 11000011 10000100

#String type str with iso-8859-1 encoding
print "\nString type str with iso8859-1 encoding"
apple_type_str_l1 = "\xC4pple"          #Äpple with iso-8859-1 encoding
print type(apple_type_str_l1)           #<type 'str'>
print apple_type_str_l1                 #Äpple

#String type str with utf-8 encoding
print "\nString type str with utf-8 encoding"
apple_type_str_utf8 = "\xC3\x84pple"    #Äpple with utf-u encoding
print type(apple_type_str_utf8)         #<type 'str'>
print apple_type_str_utf8               #Äpple

#String type unicode
print "\nString type unicode"
apple_type_unicode = u"\xC4pple"        #Äpple with unicode encoding
print type(apple_type_unicode)          #<type 'unicode'>
print apple_type_unicode                #Äpple
```

```
#String type unicode
print "\nString type unicode"
apple_type_unicode = u"\xC4pple"        #Äpple with unicode encoding
print type(apple_type_unicode)          #<type 'unicode'>
print apple_type_unicode                #Äpple

#Conversion between iso8859-1, unicode and utf-8
print "\nConversion between iso8859-1, unicode and utf-8"
apple_type_unicode = apple_type_str_l1.decode('iso-8859-1')
print type(apple_type_unicode)          #<type 'unicode'>
print apple_type_unicode                #Äpple with unicode encoding
apple_type_str_utf8 = apple_type_unicode.encode('utf-8')
print type(apple_type_str_utf8)         #<type 'str'>
print apple_type_str_utf8               #Äpple with utf-u encoding

print "\nConversion between utf-8, unicode and iso8859-1"
apple_type_unicode = apple_type_str_utf8.decode('utf-8')
print type(apple_type_unicode)          #<type 'unicode'>
print apple_type_unicode                #Äpple with unicode encoding
apple_type_str_l1 = apple_type_unicode.encode('iso-8859-1')
print type(apple_type_str_l1)           #<type 'str'>
print apple_type_str_l1                 #Äpple with iso-8859-1 encoding
```

```
def describe_number(n):
    if n > 1000000:
        return "LARGE"
    elif n > 1000:
        return "Medium"
    else:
        return "small"

    print "Never printed!"
```

```
from nltk.book import *

words = set(text1)

longwords = [w for w in words if len(w)>15]

print longwords
```

```
V = [x**2 for x in range(10)]


print V
```

```
# simon.clematide@uzh.ch
# PCL I
# Datei: def_block.py
def wc1(textfile):
    c = 0
    for line in textfile:
        for word in line.split():
            c += 1
    return c #count words of whole file

def wc2(textfile):
    c = 0
    for line in textfile:
        for word in line.split():
            c += 1
        return c #count words of first line

def wc3(textfile):
    c = 0
    for line in textfile:
        for word in line.split():
            c += 1
            return c #no count, return 1

#ruft sich selber auf
print "wc1:", wc1(open('def_block.py','r'))
print "wc2:", wc2(open('def_block.py','r'))
print "wc3:", wc3(open('def_block.py','r'))
```

```python
def foo(a):
    """ Return foo of a """
    result = 0
    for item in a:
        result += item
    return result

c = foo([5,10,23])
print c
```

```python
s = 'Python is great!'
def f():
    s = "But that's strange."
    print s

print s
f()
print s

# Output
# Python is great!
# But that's strange.
# Python is great!
```

```python
s = 'Python is great!'
def f():
    global s  #das sollte man nicht tun
    s = "But that's strange."
    print s

print s
f()
print s

#Output
# Python is great!
# But that's strange.
# But that's strange.
```

```python
# Importiere Modul book aus Package nltk
import nltk.book

# Objekte und Funktionen aus nltk.book können nur in
# vollqualifizierter Punktnotation bezeichnet werden.
print "Zweites Wort aus text1:", nltk.book.text1[1]

# Objekte und Funktionen können nicht direkt bezeichnet werden:
print text1[1] #Traceback (most recent call last):
```

```python
# Lade Modul book aus Package nltk und
# importiere alle Objekte und Funktionen ins aktuelle Modul
from nltk.book import *

# Objekte und Funktionen aus nltk.book können ohne
# Modulpräfixe bezeichnet werden.
print "Zweites Token aus text1:", text1[1]

# Die vollqualifizierter Punktnotation geht dann nicht
print "Zweites Wort aus text1:", \
    nltk.book.text1[1] #Traceback (most recent call last):
```

Locate mit Regex kombiniert:
$ locate DateiNamenMuster
Beispiele:
$ **locate -i --regex nltk\.\+corpora | less**
$ **locate -i --regex nltk\.\+corpora\.+.txt | less**

Locate mit Regex und Grep kombiniert
$ locate DateiNamenMuster | xargs grep DateiInhaltMuster
Beispiel:
Durchsucht die mit locate gefundenen Dateien nach dem Muster „Moby Dick“
$ **locate -i --regex nltk\.\+corpora\.+.txt | xargs grep -l "Moby Dick"**
/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt

In NANO Shell Befehle ausführen. Resultate werden im Shell Modus angezeigt.

!grep -n --color "\bman\b" "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt"
!grep -c "\bman\b" "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt"
!grep "\bman\b" "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | wc

```
22559:Starbuck.  I am old;--shake hands with me, man."
22564:a brave man that weeps; how great the agony of the persuasion then!"
22612:main-truck--"Ha! he soars away with it!--Where's the old man now?
22739:inboard again; the third man helplessly dropping astern, but still
22818:whole aspect, and spite of all that mortal man could do, the solid
!done  (press RETURN)
    499    6040   33307
!done  (press RETURN)
```

xargs - build and execute command lines from standard input
xargs reads items from the standard input, delimited by blanks or newlines, and executes the command (default is /bin/echo) one or more times with any initial-arguments followed by items read from standard input. **xargs [options] [command [initial-arguments]]**

Examples:
Durchsucht die mit locate gefundenen Dateien nach dem Muster „Moby Dick“
$ **locate -i --regex nltk\.\+corpora\.+.txt | xargs grep -l "Moby Dick"**

Generates a compact listing of all the users on the system.
cut -d: -f1 < /etc/passwd | sort | xargs echo

Find files named thefoo in or below the directory /tmp and delete them, processing filenames in such a way that file or directory names containing spaces or newlines are correctly handled.
find /tmp -name thefoo -type f -print0 | xargs -0 /bin/rm -f

-0 Input items are terminated by a null character instead of by whitespace, and the quotes and backslash are not special (every character is taken literally). Disables the end of file string, which is treated like any other argument. Useful when input items might contain white space, quote marks, or backslashes. The GNU find -print0 option produces input suitable for this mode.

Anzahl Zeichen
$ **cat "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | wc**
22924 212030 1242990
Anzahl Token (vereinfacht)
$ **grep "\b.*\b" "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | wc**
19650 212029 1236439
Anzahl Leerzeichen
$ **grep -v "\b.*\b" "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | wc**
3274 1 6551

Text vertikalisieren mit tr -s,
grep anwenden um Sätzzeichen und Leerzeichen zu entfernen,
alphabetisch sortieren und aggregieren mit uniq,
numerisch sortieren
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | grep -P -o "\b.*\b" | sort | uniq -c | sort -n

```
2866 that
3889 in
4477 a
4496 to
5958 and
6507 of
13624 the
```

ohne grep : 12177 vs 13624 the vs (13721 bei NLTK)
Begründungsversuche mit nachfolgenden commands
tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | sort | uniq -c | sort -n
2478 that
3273
3531 in
3979 a
4117 to
5298 and
5953 of
12177 the
mit grep aber ohne Option -o : 12177 vs 13624 the (vs 13721 bei NLTK)
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | grep -P "\b.*\b" | sort | uniq -c | sort -n
2478 that
3531 in
3979 a
4117 to
5298 and
5953 of
12177 the

Nach der vereinfachten Vertikalisierung gibt es noch einige Komposita, welche „the“ enthalten
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | grep -P -o "\b.*\b" | sort | uniq -c | grep --color ".*the." | sort -n
1 aesthetically
1 aesthetics
1 again--there
1 again!--there
1 ago!"--then
1 Ahab—there's

In nur drei Fällen folgt ein nichtwortzeichen
tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | grep -P -o "\b.*\b" | sort | uniq -c | sort -n | grep -P ".*the\W.*"
1 Jonas-in-the-Whale
1 the"--but
2 Season-on-the-Line

tr gefolgt von sort und uniq -c liefert zwei verschiedene Zeilen mit z.B. the
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | sort | uniq -c | sort -n
522 of
583 and
1427 the
2478 that
3273
3979 a
4117 to
5298 and
5953 of
12177 the

$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | sort | uniq -c | grep --color ".*the.*" | sort -n
320 other
391 there
522 they
529 their
1427 the
12177 the

Des Rätsels Lösung:
In diesen Fällen stand the nicht am Ende der Zeile
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | sort | uniq -c | grep -P --color " the$" | sort -n
12177 the
In diesen Fällen stand the am Ende der Zeile und der Carrige Return \r ist noch vorhanden!
$ tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" | sort | uniq -c | grep -P --color " the\r$" | sort -n
1427 the

Mit Echo lässt sich das Piping gut testen
```
$ echo 'Morgen bringen wir Cäsar um' | tr [a-zA-Z] [n-za-mN-ZA-M]
Zbetra oevatra jve Päfne hz
$ echo 'Zbetra oevatra jve Päfne hz' | tr [a-zA-Z] [n-za-mN-ZA-M]
Morgen bringen wir Cäsar um
```

Tr muss mehrfach angewendet werden. Es gibt zwei Varianten
Wie bis anhin tr -s '\ ' '\n' und zusätzlich tr -d '\r'
$ **tr -s '\ ' '\n' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" |tr -d '\r' | sort | uniq -c | grep -P --color " the$" | sort -n**
13604 the
Erst den.. Carriage Return in Space umwandeln und dann Space und Spaces in Newline
Das funktioniert da **tr -s '\ ' '\n'** mehrere Spaces als einen betrachtet
$ **tr -s '\r ' '\ ' < "/home/benzro/nltk_data/corpora/gutenberg/melville-moby_dick.txt" |tr -s '\ ' '\n' | sort | uniq -c | grep -P --color " the$" | sort -n**
13604 the
.

```
from nltk.book import *
print "from 0 :"
texts()
#Texte sind Sequenzen von Tokens
# (Textanzeige)
print "from 1 :"
print text1[0:10]
print "from 2 :"
print text2[0:10]
#Konkordanzen erstellen
#KWIC (Keyword in Context)
# (Stichwort mit etwas Text davor -B6 oder danach -A6 )
print "from 3 :"
text1.concordance("man") #Keyword=man
print "from 4 :"
text2.concordance("man")
print "from 5 :"
text1.concordance("woman") #Keyword=woman
print "from 6 :"
text2.concordance("woman")
#Ähnlichkeit Similarity
# (andere Wörter, welche in ähnlichen Kontexten auftauchen)
print "from 7 :"
text1.similar("woman")
print "from 8 :"
text2.similar("woman")
print "from 9 :"
text1.similar("love")
print "from 10 :"
text2.similar("love")
```

NLTK_DemoTour.py"
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
...gekürzt
from 0 :
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
...gekürzt
from 1 :
[u'[', u'Moby', u'Dick', u'by', u'Herman', u'Melville', u'1851', u']', u'ETYMOLOGY', u'.']
from 2 :
[u'[', u'Sense', u'and', u'Sensibility', u'by', u'Jane', u'Austen', u'1811', u']', u'CHAPTER']
from 3 :
Displaying 25 of 527 matches:
Civitas) which is but an artificial man ." -- OPENING SENTENCE OF HOBBES ' S
y of that sort that was killed by any man , such is his fierceness and swiftnes
...gekürzt
from 4 :
Displaying 25 of 121 matches:
ate owner of this estate was a single man , who lived to a very advanced age ,
The son , a steady respectable young man , was amply provided for by the fortu
...gekürzt
from 5 :
Displaying 10 of 10 matches:
nclude that , like the dyspeptic old woman , he must have " broken his digester
flections by the sight of a freckled woman with yellow hair and a yellow gown ,
...gekürzt
from 6 :
Displaying 25 of 68 matches:
ties . Had he married a more amiable woman , he might have been made still more
t was so much the greater , and to a woman in Mrs . Dashwood ' s situation , wi
...gekürzt
from 7 :
man king shark serpent camel job father hare bull fleece fellow
surveyor as cobbler monks laugh kings ship times fiddler
from 8 :
man men word year friend letter moment way part gentleman situation
person living creature style day little thing kind state
from 9 :
man sea it ship captain me boats them bone matter sharks nature life
air hand way head water blubber place
from 10 :
affection town heart see mother sister time me word elinor life it
marianne dear family him bed do regard her

```
# Häufigkeit in einem Korpus
# (word count $wc oder grep -c)
print "from 11 :"
print text1.count('love')
print "from 12 :"
print text2.count('love')
# Häufigkeitsdistributionen
# Alle Häufigkeiten aller verschiedenen Tokens berechnen
#
print "from 13_1:"
fdist1 = FreqDist(text1)
print type(fdist1)
vocabulary1= sorted(fdist1, key=fdist1.get,reverse=True)
for w in vocabulary1[:50]:
    print w, "\t\t", fdist1[w]
print "from 13_2:"
fdist2 = FreqDist(text2)
print type(fdist2)
vocabulary2= sorted(fdist2, key=fdist2.get,reverse=True)
for w in vocabulary2[:20]:
    print w, "\t\t", fdist2[w]
print "from 14_1 :"
fdist1.plot(50,cumulative=False)
print "from 14_2 :"
fdist2.plot(20,cumulative=False)
```

```
from 11 :
24
from 12 :
77
from 13_1:
<class 'nltk.probability.FreqDist'>
,                       18713
the                     13721
.                       6862
of                      6536
and                     6024
a                       4569
...gekürzt

from 13_2:
<class 'nltk.probability.FreqDist'>
,                       9397
to                      4063
.                       3975
the                     3861
of                      3565
and                     3350
…gekürzt

from 14_1 :
from 14_2 :
```

```
# Statistische Kollokationen
# Welche Wörter kommen unerwartet häufig nebeneinander vor?
print "from 15 :"
text1.collocations()
print "from 16 :"
text2.collocations()
# Dispersions Plots
# Wo tauchen auf einer Zeitachse Wörter wie oft auf?
# Amerikanische Ansprachen
print "from 17 :"
text4.dispersion_plot(["freedom","war"])
print "from 18 :"
text4.dispersion_plot(["economy","war"])
```

```
from 15 :
Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab; sperm
whale; Right Whale; Captain Peleg; New Bedford; Cape Horn; cried Ahab;
years ago; lower jaw; never mind; Father Mapple; cried Stubb; chief
mate; white whale; ivory leg; one hand
from 16 :
Colonel Brandon; Sir John; Lady Middleton; Miss Dashwood; every thing;
thousand pounds; dare say; Miss Steeles; said Elinor; Miss Steele;
every body; John Dashwood; great deal; Harley Street; Berkeley Street;
Miss Dashwoods; young man; Combe Magna; every day; next morning
from 17 :
from 18 :
```

```
# Textgenerierung
# Statistische Generierung von Texten aus Trigramm-Statistiken von Wört
# Typischer Wortverbindungen aus einem Korpus.
# (In NLTK 3 momentan kaputt.) Politische Rede?
# Sich inspirieren lassen von präsidialen
# Ansprachen der vergangenen Präsidenten der USA.
print "from 19 :"
#text4.generate()
"""Fellow - Countrymen : At this second gathering , our commerce and
share them . Instead of undertaking particular recommendations on this
occasion , feeling the emotions which the Executive , in seeking
practical plans for mediation , conciliation , and evil is real , and
defense ; if a preference , upon a party , and we have piled deficit
upon deficit , mortgaging our future is this which gives inestimable
value to those who came here believed they could to merit it can
deliver ; from inflated rhetoric that postures instead of others , and
by our own"""
# Listenkomprehension
print "from 20 :"
V = set(text1)
long_words = [w for w in V if len(w)>15]
print long_words
```

```
from 19 :
from 20 :
[u'hermaphroditical', u'subterraneousness', u'uninterpenetratingly', u'irresistibleness',
u'responsibilities', u'comprehensiveness', u'uncompromisedness', u'superstitiousness',
u'uncomfortableness', u'supernaturalness', u'circumnavigating', u'characteristically',
u'cannibalistically', u'circumnavigations', u'indispensableness', u'preternaturalness',
u'apprehensiveness', u'CIRCUMNAVIGATION', u'simultaneousness', u'undiscriminating',
u'Physiognomically', u'physiognomically', u'circumnavigation', u'indiscriminately']

Process finished with exit code 0
```

```
import nltk
text = """At eight o'clock on Thursday morning
Arthur didn't feel very good."""

#Tokenizer
tokens = nltk.word_tokenize(text)
print "\nTokenizer\n",tokens

#Tagger
tagged = nltk.pos_tag(tokens)
print "\nTagger\n",tagged

#Chunker
entities = nltk.chunk.ne_chunk(tagged)
print "\nChunker\n",entities

#Stemmer
stemmer = nltk.stem.PorterStemmer()
print "\nStemmer"
for tok in nltk.word_tokenize('He believes in stemming.'):
    print tok,"\t", stemmer.stem(tok)

#Lemmatizer
print "\nLemmatizer"
lemmatizer = nltk.stem.WordNetLemmatizer()
for tok in nltk.word_tokenize('He believes in stemming.'):
    print tok,"\t", lemmatizer.lemmatize(tok)
```

Tokenizer (make „words“)
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']

Tagger (add pos-tags)
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'NN'), ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'), ('Arthur', 'NNP'), ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'), ('very', 'RB'), ('good', 'JJ'), ('.', '.')]

Chunker (add constituents, noun groups, verb groups etc)
Shallow parsing (also **chunking**, "light parsing") is an analysis of a sentence which identifies the constituents (noun groups, verbs, verb groups, etc.), but does not specify their internal structure, nor their role in the main sentence.

```
(S
  At/IN
  eight/CD
  o'clock/NN
  on/IN
  Thursday/NNP
  morning/NN
  (PERSON Arthur/NNP)
  did/VBD
  n't/RB
  feel/VB
  very/RB
  good/JJ
  ./.)
```

Stemmer (token as input and adds a truncated form called stem)

He	He
believes	believ
in	in
stemming	stem
.	.

Lemmatizer (token as input and adds the dictionary or base form called lemma)

He	He
believes	belief
in	in
stemming	stemming
.	.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import nltk
text = """At eight o'clock on Thursday morning
Arthur didn't feel very good."""
print "\nRaw Text\n",text
text_ = 'He believes in stemming.'
print "\nRaw Text_\n",text_

#Tokenizer
tokens = nltk.word_tokenize(text)
print "\nTokenizer\n",tokens
itokens=text_.split()
print "\niSimpleSplitTokenizer\n",itokens

#Text
itext=nltk.Text(itokens)
print "\niText\n ", itext #to print the first line
for i in itext:
    print type(i), i
print "  Text Länge und Typ\n ", len(itext), type(itext)

#Tagger
tagged = nltk.pos_tag(tokens)
print "\nTagger\n",tagged
itagged = nltk.pos_tag(itokens)
print "\niTagger\n",itagged
```

```
/usr/bin/python2.7 "/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Vorlesung/vl08/NLTK_Einf.py"

Raw Text
At eight o'clock on Thursday morning
Arthur didn't feel very good.

Raw Text_
He believes in stemming.

Tokenizer
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']

iSimpleSplitTokenizer
['He', 'believes', 'in', 'stemming.']

iText
  <Text: He believes in stemming....>
<type 'str'> He
<type 'str'> believes
<type 'str'> in
<type 'str'> stemming.
  Text Länge und Typ
  4 <class 'nltk.text.Text'>

Tagger
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'NN'), ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'), ('Arthur', 'NNP'), ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'), ('very', 'RB'), ('good', 'JJ'), ('.', '.')]

iTagger
[('He', 'PRP'), ('believes', 'VBZ'), ('in', 'IN'), ('stemming.', 'NN')]
```

```
#Chunker
entities = nltk.chunk.ne_chunk(tagged)
print "\nChunker\n",entities
ientities = nltk.chunk.ne_chunk(itagged)
print "\niChunker\n",ientities

#Stemmer
stemmer = nltk.stem.PorterStemmer()
print "\nStemmer"
for tok in nltk.word_tokenize('He believes in stemming.'):
    print tok,"\t", stemmer.stem(tok)
print "\niStemmer"
for tok in itext:
    print tok,"\t", stemmer.stem(tok)

#Lemmatizer
print "\nLemmatizer"
lemmatizer = nltk.stem.WordNetLemmatizer()
for tok in nltk.word_tokenize('He believes in stemming.'):
    print tok,"\t", lemmatizer.lemmatize(tok)
print "\niLemmatizer"
for tok in itext:
    print tok,"\t", lemmatizer.lemmatize(tok)
```

```
Chunker
(S
  At/IN
  eight/CD
  o'clock/NN
  on/IN
  Thursday/NNP
  morning/NN
  (PERSON Arthur/NNP)
  did/VBD
  n't/RB
  feel/VB
  very/RB
  good/JJ
  ./.)

iChunker
(S He/PRP believes/VBZ in/IN stemming./NN)

Stemmer
He      He
believes        believ
in      in
stemming        stem
.       .

iStemmer
He      He
believes        believ
in      in
stemming.       stemming.

Lemmatizer
He      He
believes        belief
in      in
stemming        stemming
.       .

iLemmatizer
He      He
believes        belief
in      in
stemming.       stemming.
```

Short and dense: http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

> The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.
>
> However, the two words differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

From the NLTK docs:

> Lemmatization and stemming are special cases of normalization. They identify a canonical representative for a set of related word forms.

Lemmatisation is closely related to **stemming**. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

For instance:

1. The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
2. The word "walk" is the base form for word "walking", and hence this is matched in both stemming and lemmatisation.
3. The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatisation can in principle select the appropriate lemma depending on the context.

```
tuples2 = [('a',1),('b',2),('c',3),('d',4)]
#tuples2= "klklk"   #funktioniert nicht da kein Tupel

l1 = [(c2,c1) for (c1,c2) in tuples2]

print l1   #[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

```
        # Einfaches Statement
        print "Something to print"

        # If-Statement mit komplexen Ausdrücken drin
        if len("A "+"String") > 5:
            print "A "+"String".lower()

        # Prinzip:
        # Statements sind keine Expressions

        # Frage:
        # Aber warum kann ich interaktiv schreiben?

        "A Statement"

        # oder

        3*4+8
```

```
#Demo von Mengen- und Dictionarykomprehension
#Alphanumeric is a combination of alphabetic and numeric
# characters, and is used to describe the collection of
# Latin letters and Arabic digits or a text constructed
# from this collection.
mc = {x.lower() for x in "Das Alphabet. 12" if x.isalnum()}
print mc #set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p',
            # 's', '2', 't']) der Punkt fällt weg

ms = set()
for x in "Das Alphabet. 12":
    if x.isalnum():
        ms.add(x.lower())
print ms #set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p',
           #  's', '2', 't']) Der Punkt fällt weg

text = "abrakadabra. 12"
dc = {c:text.count(c) for c in set(text)}
print dc #{'a': 5, ' ': 1, 'b': 2, 'd': 1, 'k': 1, '.': 1,
        #  '1': 1, 'r': 2, '2': 1}

# Wie würde man das als For-Schleife implementieren?
ds = dict()
for c in set(text):
    ds.update({c:text.count(c)})
print ds #{'a': 5, ' ': 1, 'b': 2, '1': 1, 'd': 1, '2': 1,
        #  'k': 1, 'r': 2, '.': 1}
```

```
/usr/bin/python2.7 "/media/benzro/OS/Users/benzro/Documents
  /LubuntuShared/8) PCL-1/Vorlesung/vl09/other_comprehensions.py"
set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p', 's', '2', 't'])
set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p', 's', '2', 't'])
{'a': 5, ' ': 1, 'b': 2, 'd': 1, 'k': 1, '.': 1, '1': 1, 'r':
  2, '2': 1}
{'a': 5, ' ': 1, 'b': 2, '1': 1, 'd': 1, '2': 1, 'k': 1, 'r':
  2, '.': 1}

Process finished with exit code 0
```

```
from nltk.corpus import brown

# Korpus als Liste von 2-Tupeln (Wort, POS-Tag)
brown_tagged_words = brown.tagged_words()

# Das balancierte Korpus umfasst Texte aus 15 Kategorien
print "brown.categories ", brown.categories(),"\n"

# Es werden alle 500 Einzeldateien verarbeitet!
cnt=0
for f in brown.fileids()[1:10]:
    cnt+=1
    print cnt, ": ", f
    print brown.abspath(f)

# brown
print 60*'*',\
    '\nKorpus als Liste von Paaren von Woertern und POS-Tags:
print brown_tagged_words,"\n", \
        len(brown_tagged_words), "\n" #1'161'192 Zeichen
print brown_tagged_words[0:20],"\n"
```

```
brown.categories  [u'adventure', u'belles_lettres', u'editorial',
 u'fiction', u'government', u'hobbies', u'humor', u'learned',
 u'lore', u'mystery', u'news', u'religion', u'reviews', u'romance',
 u'science_fiction']

1 :  ca02
/home/benzro/nltk_data/corpora/brown/ca02
2 :  ca03
/home/benzro/nltk_data/corpora/brown/ca03
3 :  ca04
/home/benzro/nltk_data/corpora/brown/ca04
4 :  ca05
/home/benzro/nltk_data/corpora/brown/ca05
5 :  ca06
/home/benzro/nltk_data/corpora/brown/ca06
6 :  ca07
/home/benzro/nltk_data/corpora/brown/ca07
7 :  ca08
/home/benzro/nltk_data/corpora/brown/ca08
8 :  ca09
/home/benzro/nltk_data/corpora/brown/ca09
9 :  ca10
/home/benzro/nltk_data/corpora/brown/ca10
```

```
# Wo liegen die Dateien dieses Korpus?
#Python
print "brown.root path: ", brown.root

#Unix
#$ locate -i --regex nltk\.\+corpora\.+brown
#$ ls -l "/home/benzro/nltk_data/corpora/brown/README"
# -rw-rw-r-- 1 benzro benzro 387 Nov 13 14:46
#$ ls -l "/home/benzro/nltk_data/corpora/brown/CONTENTS"
# -rw-rw-r-- 1 benzro benzro 145896 Nov 13 14:46
#$ less "/home/benzro/nltk_data/corpora/brown/CONTENTS"
#$ less "/home/benzro/nltk_data/corpora/brown/README"
```

```
Korpus als Liste von Paaren von Woertern und POS-Tags:
[(u'The', u'AT'), (u'Fulton', u'NP-TL'), ...]
1161192

[(u'The', u'AT'), (u'Fulton', u'NP-TL'), (u'County', u'NN-TL'),
 (u'Grand', u'JJ-TL'), (u'Jury', u'NN-TL'), (u'said', u'VBD'),
 (u'Friday', u'NR'), (u'an', u'AT'), (u'investigation', u'NN'),
 (u'of', u'IN'), (u"Atlanta's", u'NP$'), (u'recent', u'JJ'),
 (u'primary', u'NN'), (u'election', u'NN'), (u'produced', u'VBD'),
 (u'``', u'``'), (u'no', u'AT'), (u'evidence', u'NN'), (u"''",
 u"''"), (u'that', u'CS')]

brown.root path /home/benzro/nltk_data/corpora/brown
```

```
from nltk.corpus import gutenberg

# oder absoluter Pfad einer Textdatei
filename = 'austen-emma.txt'

# Korpus als eine lange Zeichenkette
emma_chars = gutenberg.raw(filename)
print 60*'*', '\nKorpus als eine lange Zeichenkette:'
print emma_chars[-324:] #die letzten 324 Zeichen

# Korpus als Liste von Wörtern
# (Wort ist Zeichenkette)
emma_words = gutenberg.words(filename)
print 60*'*', '\nKorpus als Liste von Woertern:'
print emma_words[-30:] #die letzten 30 Wörter

# Korpus als Liste von Sätzen
# (Satz ist Liste von Wörten)
emma_sents = gutenberg.sents(filename)
print 60*'*', '\nKorpus als Liste von Saetzen:'
print emma_sents[-5:] #die letzten 5 Sätze

# Korpus als Liste von Paragraphen
# (Paragraph ist Liste von Sätzen)
emma_paras = gutenberg.paras(filename)
print 60*'*', '\nKorpus als Liste von Paragraphen:'
print emma_paras[-2:] #die letzten 2 Paragraphen
```

```
************************************************************
Korpus als eine lange Zeichenkette:
white satin, very few
lace veils; a most pitiful business!--Selina would stare when she
heard of it."--But, in spite of these deficiencies, the wishes,
the hopes, the confidence, the predictions of the small band
of true friends who witnessed the ceremony, were fully answered
in the perfect happiness of the union.
FINIS
************************************************************
Korpus als Liste von Woertern:
[u',', u'the', u'confidence', u',', u'the', u'predictions', u'of', u'the', u'small', u'band', u'of', u'true',
u'friends', u'who', u'witnessed', u'the', u'ceremony', u',', u'were', u'fully', u'answered', u'in', u'the',
u'perfect', u'happiness', u'of', u'the', u'union', u'.', u'FINIS']
************************************************************
Korpus als Liste von Saetzen:
[gekürzt..., [u'The', u'wedding', u'was', u'very', u'much', u'like', u'other', u'weddings', u',', u'where',
u'the', u'parties', u'have', u'no', u'taste', u'for', u'finery', u'or', u'parade', u';', u'and', u'Mrs', u'.',
u'Elton', u',', u'from', u'the', u'particulars', u'detailed', u'by', u'her', u'husband', u',', u'thought', u'it',
u'all', u'extremely', u'shabby', u',', u'and', u'very', u'inferior', u'to', u'her', u'own', u'.--"', u'Very',
u'little', u'white', u'satin', u',', u'very', u'few', u'lace', u'veils', u';', u'a', u'most', u'pitiful', u'business',
u'!--', u'Selina', u'would', u'stare', u'when', u'she', u'heard', u'of', u'it', u'."'], [u'--', u'But', u',', u'in',
u'spite', u'of', u'these', u'deficiencies', u',', u'the', u'wishes', u',', u'the', u'hopes', u',', u'the',
u'confidence', u',', u'the', u'predictions', u'of', u'the', u'small', u'band', u'of', u'true', u'friends',
u'who', u'witnessed', u'the', u'ceremony', u',', u'were', u'fully', u'answered', u'in', u'the', u'perfect',
u'happiness', u'of', u'the', u'union', u'.'], [u'FINIS']]
************************************************************
Korpus als Liste von Paragraphen:
[[[u'The', u'wedding', u'was', u'very', u'much', u'like', u'other', u'weddings', u',', u'where', u'the',
u'parties', u'have', u'no', u'taste', u'for', u'finery', u'or', u'parade', u';', u'and', u'Mrs', u'.', u'Elton', u',',
u'from', u'the', u'particulars', u'detailed', u'by', u'her', u'husband', u',', u'thought', u'it', u'all',
u'extremely', u'shabby', u',', u'and', u'very', u'inferior', u'to', u'her', u'own', u'.--"', u'Very', u'little',
u'white', u'satin', u',', u'very', u'few', u'lace', u'veils', u';', u'a', u'most', u'pitiful', u'business', u'!--',
u'Selina', u'would', u'stare', u'when', u'she', u'heard', u'of', u'it', u'."'], [u'--', u'But', u',', u'in', u'spite',
u'of', u'these', u'deficiencies', u',', u'the', u'wishes', u',', u'the', u'hopes', u',', u'the', u'confidence', u',',
u'the', u'predictions', u'of', u'the', u'small', u'band', u'of', u'true', u'friends', u'who', u'witnessed',
u'the', u'ceremony', u',', u'were', u'fully', u'answered', u'in', u'the', u'perfect', u'happiness', u'of',
u'the', u'union', u'.']], [[u'FINIS']]]
```

```
# Wieviele Paragraphen?
number_of_paras=len(gutenberg.paras(filename))
print "\n number of paras: ",number_of_paras,"\n"

# Wieviele Sätze?
number_of_sents=len(gutenberg.sents(filename))
print "\n number of sents: ",number_of_sents,"\n"

# Wieviele Sätze pro Paragraph im Schnitt?
emma_paras_sents_mean=number_of_sents/number_of_paras
print "\n number sents per para: ",emma_paras_sents_mean,"\n"

# Wieviele Wörter pro Satz im Schnitt?
number_of_words=len(gutenberg.words(filename))
emma_sents_words_mean=number_of_words/number_of_sents
print "\n number of words per sent: ",emma_sents_words_mean,"\

# Wie kann man die durchschnittliche Wortlänge berechnen?
number_of_chars=len(gutenberg.raw(filename))
emma_words_chars_mean=number_of_chars/number_of_words
print "\n number of chars per word: ",emma_words_chars_mean,"\

# Wo liegen die Dateien dieses Korpus?
print "\n gutenberg.root directory: ", gutenberg.root,"\n"
```

```
number of paras:  2371


number of sents:  7752


number sents per para:  3


number of words per sent:  24


number of chars per word:  4


gutenberg.root directory:  /home/benzro/nltk_data/corpora/gutenberg
```

NLTK (Natural Language Toolkit)

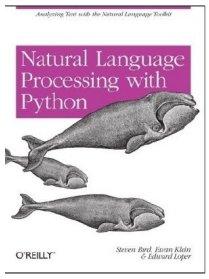
NLTK-Framework `http://www.nltk.org`

- Sammlung von Open-Source-Python-Modulen für die Sprachverarbeitung (Natural Language Processing, NLP)
- Ressourcen: frei verfügbare Korpora, Treebanks, Lexika ...
- Applikationen: Tokenizer, Stemmer, Tagger, Chunker, Parser, Semantik, Alignierung...
  (oft eher Toy-Implementationen für Lehrzwecke; andere Toolkits sind industrial-strength `http://spacy.io`)
- Module für Evaluation, Klassifikation, Clustering, Maschinelles Lernen (Schnittstellen zu State-of-the-Art-Bibliotheken)
- API (Application Programming Interface) für WordNet und Lexika

Installationsanleitungen für Win, Mac, Linux

`http://www.nltk.org/install.html`

Bird et. al (2009): Natural Language Processing in Python[1]



- Praktische Einführung in NLP mit Hilfe von NLTK 3
- Anwendungsorientiert, keine vertiefte Einführung in Python-Konzepte
- Kursbuch für PCL I (und auch II)
- 2. Edition (nur online!) benutzt Syntax von Python 3 (kompatibel mit futurisiertem Python 2)
- 1. Edition (NLTK 2!) als PDF abgelegt im Materialordner
- Weiteres NLTK-basiertes Buch mit vielen NLP-Rezepten: [Perkins 2010]

Modulübersicht

Language processing task	NLTK modules	Functionality
Accessing corpora	nltk.corpus	standardized interfaces to corpora and lexicons
String processing	nltk.tokenize, nltk.stem	tokenizers, sentence tokenizers, stemmers
Collocation discovery	nltk.collocations	t-test, chi-squared, point-wise mutual information
Part-of-speech tagging	nltk.tag	n-gram, backoff, Brill, HMM, TnT
Classification	nltk.classify, nltk.cluster	decision tree, maximum entropy, naive Bayes, EM, k-means
Chunking	nltk.chunk	regular expression, n-gram, named-entity
Parsing	nltk.parse	chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	nltk.sem, nltk.inference	lambda calculus, first-order logic, model checking
Evaluation metrics	nltk.metrics	precision, recall, agreement coefficients
Probability and estimation	nltk.probability	frequency distributions, smoothed probability distributions
Applications	nltk.app, nltk.chat	graphical concordancer, parsers, WordNet browser, chatbots
Linguistic fieldwork	nltk.toolbox	manipulate data in SIL Toolbox format

Verzeichnisstruktur vom NLTK 3

```
$ tree -F /Library/Python/2.7/site-packages/nltk/
/Library/Python/2.7/site-packages/nltk
|-- VERSION
|-- __init__.py
|-- __init__.pyc
|-- app
|   |-- __init__.py
|   |-- __init__.pyc
|   |-- chartparser_app.py
|   |-- chartparser_app.pyc
|   |-- chunkparser_app.py
|   |-- chunkparser_app.pyc
...
|-- util.py
...
24 directories, 648 files
```

- Module: Dateien mit Python-Quellkode: `util.py`
- Maschinenunabhängig kompilierter Bytekode: `chunkparser_app.pyc`
- Packages: Verzeichnisse wie `nltk` oder `app` mit `__init__.py`

Module importieren

Anweisung: `import Module` ▸1

```
# Importiere Modul book aus Package nltk
import nltk.book

# Objekte und Funktionen aus nltk.book können nur in
# vollqualifizierter Punktnotation bezeichnet werden.
print "Zweites Wort aus text1:", nltk.book.text1[1]

# Objekte und Funktionen können nicht direkt bezeichnet werden:
print text1[1]
```

Alle Objekte und Funktionen aus Modulen importieren

Anweisung: `from Module import *` ▸2

```
# Lade Modul book aus Package nltk und
# importiere alle Objekte und Funktionen ins aktuelle Modul
from nltk.book import *

# Objekte und Funktionen aus nltk.book können ohne
# Modulpräfixe bezeichnet werden.
print "Zweites Token aus text1:", text1[1]

# Die vollqualifizierter Punktnotation geht dann nicht
print "Zweites Wort aus text1:", nltk.book.text1[1]
```

Python 2 futurisieren

Wichtige Direktiven zur Kompatibilität von Python 2 mit Python 3

```
from __future__ import print_function, unicode_literals, division
```

Python 2: print-Statement

```
>>> print u'sähe:', count
```

Python 3: print-Funktion ▶

```
>>> print('sähe:', count)
```

Python 2: Ganzzahldivision

```
>>> print 1/3
0
```

Python 3: Division

```
>>> print(1/3)
0.3333333333333333
```

Eine Tour durch Kapitel 1

- Repräsentation von Text-Korpora als Objekt vom Typ Text (im Wesentlichen als Liste von String-Token)
- KWIC (*Keyword in context*): Konkordanzen erstellen und anzeigen
- Vorkommensähnlichkeit (*similarity*): Welche unterschiedlichen Wörter erscheinen häufig in ähnlichen Kontexten?
- Häufigkeitsverteilungen (*frequency distribution*) berechnen: Wie oft kommt welche Wortform vor?
- Statistische Kollokationen (*collocations*): Welche Wortpaare kommen viel häufiger zusammen vor als zufällig zu erwarten wäre?
- Dispersion-Plot (Korpuslinguistik): An welchen Stellen in einem Korpus kommt ein Wort vor? [Baker et al. 2006]

Visualisierungen mit Plotting benötigen separate Diagramm-Bibliothek matplotlib (Download via `matplotlib.org`).

Listenkomprehension (list comprehension)

Mathematische Mengenkomprehension ▶

Die Menge aller Quadratzahlen aus der Grundmenge der Zahlen von 0 bis 9.

$$\{x^2 \mid x \in \{0,1,2,3,4,5,6,7,8,9\}\}$$

In Python mit Listen als Mengen:

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Syntax

```
[ x**2 for x in range(10) ]
```

Bildungsvorschrift | Grundmenge, deren Werte x durchläuft

Listenkomprehension mit Bedingungen ▶

Syntaxschema (aus [Bird et al. 2009, 19])

$\{w \mid w \in V \,\&\, P(w)\}$ — Die Liste aller Elemente w aus V, für die die Eigenschaft $P(w)$ wahr ist.

```
[w for w in V if p(w)]
```

Filtern von Vokabularlisten ▸3

```
from nltk.book import *

words = set(text1)

longwords = [w for w in words if len(w)>15]
```

`set(text1)` erzeugt Menge aller Listenelemente aus `text1`.

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

6. Now you're ready to load your own corpus, using the following code:

```
>>>
>>>
>>> f=open('ccc.txt','rU')
>>> text=f.read()
>>> text1=text.split()
>>> abstracts=nltk.Text(text1)
>>>
>>>
```

Basically, these lines simply split all the words in your file into a list form that the NLTK can access and read, so that you can run analyses on your corpus using the NLTK tools. Above, 'ccc.txt' is the plain text file which was saved in my Python folder. 'Abstracts' is just what I label the file while working with the NLTK. You can name it whatever you want.

Package nltk
>>> **from nltk import ***
>>> **help**
<module 'nltk.help' from '/usr/local/lib/python2.7/dist-packages/nltk/help.pyc'>

Für das NLTK BUCH gibt es eine book collection welche mit dem Befehl
>>> nltk.download() runtergeladen werden kann
Modul book aus Package nltk
>>> **from nltk.book import ***
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9

Modul corpus aus Package nltk
Corpus Gutenberg
Pfad des Ordners gutenberg im Ordner corpora
>>> **from nltk.corpus import gutenberg**
>>> **print gutenberg.root**
/home/benzro/nltk_data/corpora/gutenberg
Dateien des Ordners gutenberg
>>> **gutenberg.fileids()**
[u'austen-emma.txt', u'austen-persuasion.txt', u'austen-sense.txt', u'bible-kjv.txt', u'blake-poems.txt', u'bryant-stories.txt', u'burgess-busterbrown.txt', u'carroll-alice.txt', u'chesterton-ball.txt', u'chesterton-brown.txt', u'chesterton-thursday.txt', u'edgeworth-parents.txt', u'melville-moby_dick.txt', u'milton-paradise.txt', u'shakespeare-caesar.txt', u'shakespeare-hamlet.txt', u'shakespeare-macbeth.txt', u'whitman-leaves.txt']

Corpus Brown
>>> **from nltk.corpus import brown**
>>> **print brown.root**
/home/benzro/nltk_data/corpora/brown
>>> **print brown.fileids()**
[u'ca01', u'ca02', u'ca03', u'ca04', u'ca05', u'ca06', u'ca07', u'ca08', u'ca09'...]

Keine Ordner sondernTe

Funktionen definieren und aufrufen

Definition der einstelligen Funktion `foo()`

```
def foo(a):
    """ Return foo of a """
    result = 0
    for item in a:
        result += item
    return result
```

`return`-Statement ergibt Rückgabewert, d.h. den Funktionswert. Die Parameter der Funktion (Platzhalter für Argumente) stehen in Klammern. Die erste Zeichenkette im Rumpf ist der Doc-String.

Funktionsaufruf (call)

```
c = foo([5,10,23])
```

Mehrere `return`-Anweisungen

Effekt der return-Anweisung

```
def describe_number(n):
    if n > 1000000:
        return "LARGE"
    elif n > 1000:
        return "Medium"
    else:
        return "small"
    print "Never printed!"
```

- Verarbeitung der `return`-Anweisung beendet die Ausführung der Funktion.
- Beliebige Objekte können als Funktionswert zurückgegeben werden, auch Listen.

Wann und wozu sind Funktionen gut?

```
def foo(a):
    """ Return foo of a """
    result = 0
    for item in a:
        result += item
    return result
```

Heilmittel gegen Spaghettikode

- Abstraktion: Eine Funktion kann einige Zeilen Kode bezeichnen, welche oft gebraucht werden.
- Schnittstelle: Die Parameter einer Funktion machen den Kode an ganz bestimmten Stellen variabel (= Parametrisierung).
- Klarheit: Eine gute Funktion hat eine klar beschreibbare (=spezifizierbare) Funktionalität.

Skopus (Erreichbarkeit) von Variablennamen

Modulweit erreichbare globale Variablen (*globals*)

- (Variablen-)Namen, die in einem Modul zugewiesen werden, sind danach im ganzen Modul erreichbar.
- Modul `foo` ist Python-Quellkode aus Datei `foo.py`.

Funktionsweit erreichbare lokale Variablen (*locals*)

- Parameter `a` und `b` einer Funktion `foo(a,b)`, sind nur innerhalb der Funktion `foo()` erreichbar.
- (Variablen)-Namen, die in einer Funktion definiert werden, sind nur innerhalb der Funktion erreichbar.

Introspektion

Die eingebauten Funktionen `globals()` und `locals()` geben die zum Aufrufzeitpunkt definierten Namen aus.

```
# Listenbildung mit iterativen Statements
sl = list()
for c in "St. Moritz-Str. 23":
    if c.isalnum():
        sl.append(c.lower())
print sl

# Listenbildung mit Listenkomprehensionsausdruck
el = [c.lower() for c in "St. Moritz-Str. 23" if c.isalnum()]
print el
```

```
PCL-1/Vorlesung/vl09/iter_statement_expression.py"
['s', 't', 'm', 'o', 'r', 'i', 't', 'z', 's', 't', 'r', '2', '3']
['s', 't', 'm', 'o', 'r', 'i', 't', 'z', 's', 't', 'r', '2', '3']

Process finished with exit code 0
```

```
# Als Statement, Anweisung
sl = []
for c in "St. Moritz-Str. 23":
    if c.isalnum():
        sl.append(c)
    else:
        sl.append(' ')
print sl

# Als Expression, Ausdruck
el = [ c if c.isalnum() else ' ' for c in "St. Moritz-Str. 23" ]
print el
```

```
PCL-1/Vorlesung/vl09/if_then_else_statement_vs_expression.py"
['S', 't', ' ', ' ', 'M', 'o', 'r', 'i', 't', 'z', ' ', 'S', 't', 'r', ' ', ' ', '2', '3']
['S', 't', ' ', ' ', 'M', 'o', 'r', 'i', 't', 'z', ' ', 'S', 't', 'r', ' ', ' ', '2', '3']

Process finished with exit code 0
```

```
# Demonstration von Wichtigem
from nltk.book import *

#Set
words = set(text1)
print "\n",len(words),"\n"
#print words[:20] #set hat kein Attribut __getitem__

#String
print ', '.join(words)[:20],"\n " #ersten 20 chars

#List
print list(words)[:20],"\n" #ersten 20 list elements

#List
longwords = [w for w in words if len(w)>15]
print longwords

#List
l = []
l.append(3)           # ein Element anhängen
l.extend((4,'x',5))   # eine ganze Sequenz anhängen
print l
del l[3]              # ein Element löschen
l[2] = 2              # eine Element austauschen
l.sort(reverse=True) # in-place umgekehrt sortieren
print l
```

```
19317

funereal, unscientif

[u'funereal', u'unscientific', u'divinely', u'foul', u'four', u'gag', u'prefix', u'woods', u'clotted',
u'Duck', u'hanging', u'plaudits', u'woody', u'Until', u'marching', u'disobeying', u'canes', u'granting',
u'advantage', u'Westers']

[u'hermaphroditical', u'subterraneousness', u'uninterpenetratingly', u'irresistibleness',
u'responsibilities', u'comprehensiveness', u'uncompromisedness', u'superstitiousness',
u'uncomfortableness', u'supernaturalness', u'circumnavigating', u'characteristically',
u'cannibalistically', u'circumnavigations', u'indispensableness', u'preternaturalness',
u'apprehensiveness', u'CIRCUMNAVIGATION', u'simultaneousness', u'undiscriminating',
u'Physiognomically', u'physiognomically', u'circumnavigation', u'indiscriminately']

[3, 4, 'x', 5]
[4, 3, 2]
```

```python
import re

teststr = '''This
is a
Test.'''

# Traditionelle Funktionsdefinition
def sf(s):
    # \s is any white space character
    # and equivalent to [ \t \n \r \f]
    return re.sub(r'\s+','',s)
print sf(teststr)

def isf(s):
    return re.sub(r'[ \t \n \r \f]+','',s)
print isf(teststr)

# Funktionsdefinition mit Lambda-Ausdruck
ef = lambda s: re.sub(r'\s+','',s)
print ef(teststr)

# Was passiert, wenn kein return Statement existiert?
def sf2(s):
    re.sub(r'\s+','',s)
print sf2(teststr)

# Was passiert, wenn ich schreibe:
# SyntaxError: invalid syntax -> return
# ef = lambda s: return re.sub(r'\s+','',s)
# print sf(teststr)
```

```
 PCL-1/Vorlesung/vl09/functions_statement_vs_expression.py"
ThisisaTest.
ThisisaTest.
ThisisaTest.
None

Process finished with exit code 0
```

```python
import nltk
from nltk.corpus import gutenberg

#Load text
emma_words = gutenberg.words('austen-emma.txt')
print "\n", type(emma_words), "\n", len(emma_words)

#Calculate frequency distribution of words
emma_fd = nltk.FreqDist(emma_words)
print "\n", type(emma_fd),"\n", emma_fd

#Print frequency distribution of words
emma_fd_sorted=sorted(emma_fd, key=emma_fd.get, reverse=True )
print "\n", type(emma_fd_sorted),"\n", emma_fd_sorted[:20], "\n"
for w in emma_fd_sorted[:5]:
    print w, "\t\t", emma_fd[w]

#Tabuliere die häufigsten 20 Wörter
print "\n", type(emma_fd.tabulate)
emma_fd.tabulate(20)

#Plot frequency distribution of words
emma_fd.plot(10, cumulative=False)

# Finde alle Wörter für die gilt:
# - mehr als 10 Buchstaben und
# - kommen mindestens 7 mal vor
# - printe nur die ersten 20
wl = sorted([w for w in emma_fd.keys()
             if len(w)>10 and emma_fd[w]> 7])
print "\n", wl[:20], "\n"
```

List

Dict

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl09/freqdist_emma.py"

<class 'nltk.corpus.reader.util.StreamBackedCorpusView'>
192427

<class 'nltk.probability.FreqDist'>
<FreqDist with 7811 samples and 192427 outcomes>

<type 'list'>
[u',', u'.', u'to', u'the', u'and', u'of', u'I', u'a', u'was',
 u'her', u';', u'it', u'in', u'not', u'"', u'be', u'she',
 u'that', u'you', u'had']

,       11454
.       6928
to      5183
the         4844
and         4672

<type 'instancemethod'>
  ,    .   to  the  and   of    I    a  was  her    ;   it
 in  not    "   be  she that  you  had
11454 6928 5183 4844 4672 4279 3178 3004 2385 2381 2199 2128
 2118 2101 2004 1970 1778 1730 1677 1606

[u'accomplished', u'acknowledge', u'acknowledged',
 u'acquaintance', u'affectionate', u'approbation',
 u'arrangement', u'circumstance', u'circumstances',
 u'comfortable', u'comfortably', u'communicated',
 u'communication', u'compliments', u'concealment',
 u'consciousness', u'consequence', u'considerable',
 u'consideration', u'considering']
```

```python
import nltk
from nltk.corpus import brown

#Conditional Frequency Distribution
cfd = nltk.ConditionalFreqDist([
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre)])
print "\n",type(cfd),"\n",cfd,"\n"

#Wie oft kommen die modals (samples)
# conditional in den genres (conditions) vor
genres = ['news', 'religion', 'hobbies',
          'science_fiction', 'romance', 'humor']
modals = ['can', 'could', 'may', 'might', 'must', 'will']

cfd.tabulate(conditions=genres, samples=modals)
```

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl09/CondFreqDist_brown.py"

<class 'nltk.probability.ConditionalFreqDist'>
<ConditionalFreqDist with 15 conditions>

                 can could  may might must will
          news    93   86   66   38   50  389
      religion    82   59   78   12   54   71
       hobbies   268   58  131   22   83  264
science_fiction   16   49    4   12    8   16
       romance    74  193   11   51   45   43
         humor    16   30    8    8    9   13

Process finished with exit code 0
```

Example	Description
`fdist = FreqDist(samples)`	Create a frequency distribution containing the given samples
`fdist.inc(sample)`	Increment the count for this sample
`fdist['monstrous']`	Count of the number of times a given sample occurred
`fdist.freq('monstrous')`	Frequency of a given sample
`fdist.N()`	Total number of samples
`fdist.keys()`	The samples sorted in order of decreasing frequency
`for sample in fdist:`	Iterate over the samples, in order of decreasing frequency
`fdist.max()`	Sample with the greatest count
`fdist.tabulate()`	Tabulate the frequency distribution
`fdist.plot()`	Graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	Cumulative plot of the frequency distribution
`fdist1 < fdist2`	Test if samples in `fdist1` occur less frequently than in `fdist2`

Assignment always copies the value of an expression, but a value is not always what you might expect it to be. In particular, the "value" of a structured object such as a list is actually just a *reference* to the object. In the following example, ❶ assigns the reference of `foo` to the new variable `bar`. Now when we modify something inside `foo` on line ❷, we can see that the contents of `bar` have also been changed.

```
>>> foo = ['Monty', 'Python']
>>> bar = foo ❶
>>> foo[1] = 'Bodkin' ❷
>>> bar
['Monty', 'Bodkin']
```

The line `bar = foo` ❶ does not copy the contents of the variable, only its "object reference." To understand what is going on here, we need to know how lists are stored in the computer's memory. In Figure 4-1, we see that a list `foo` is a reference to an object stored at location 3133 (which is itself a series of pointers to other locations holding strings). When we assign `bar = foo`, it is just the object reference 3133 that gets copied.

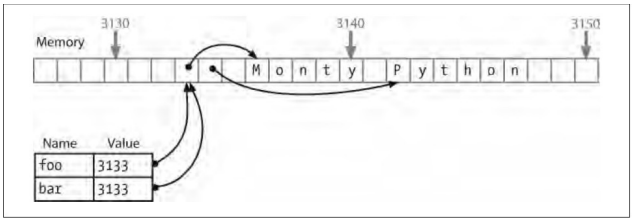


Figure 4-1. List assignment and computer memory: Two list objects `foo` *and* `bar` *reference the same location in the computer's memory; updating* `foo` *will also modify* `bar`*, and vice versa.*

Let's experiment some more, by creating a variable `empty` holding the empty list, then using it three times on the next line.

```
>>> empty = []
>>> nested = [empty, empty, empty]
>>> nested
[[], [], []]
>>> nested[1].append('Python')
>>> nested
[['Python'], ['Python'], ['Python']]
```

Observe that changing one of the items inside our nested list of lists changed them all. This is because each of the three elements is actually just a reference to one and the same list in memory.

```
>>> nested = [[]] * 3
>>> nested
[[], [], []]
>>> nested[1].append('Python')
>>> nested
[['Python'], ['Python'], ['Python']]
>>> nested[1] = ['Monty']
>>> nested
[['Python'], ['Monty'], ['Python']]
```

We began with a list containing three references to a single empty list object. Then we modified that object by appending 'Python' to it, resulting in a list containing three references to a single list object ['Python']. **Next, we overwrote one of those references with a reference to a new object ['Monty'].** This last step modified one of the three object references inside the nested list. However, the ['Python'] object wasn't changed, and is still referenced from two places in our nested list of lists. **It is crucial to appreciate this difference between modifying an object via an object reference (.append) and overwriting an object reference (=[]).**

```
>>> nested = [[]] * 3
>>> nested[1].append('Pythons')
>>> nested[1].append('Pythons')
>>> nested[1].append('Pythons')
>>> nested
[['Pythons', 'Pythons', 'Pythons'], ['Pythons', 'Pythons', 'Pythons'],
  ['Pythons', 'Pythons', 'Pythons']]
>>> nested[1] = ['Monty']
>>> nested[1].append('Pythons')
>>> nested
[['Pythons', 'Pythons', 'Pythons'], ['Monty', 'Pythons'], ['Pythons',
  'Pythons', 'Pythons']]
>>> nested[0].append('Montys')
>>> nested[0].append('Montys')
>>> nested[0].append('Montys')
>>> nested
[['Pythons', 'Pythons', 'Pythons', 'Montys', 'Montys', 'Montys'], ['Monty',
  'Pythons'], ['Pythons', 'Pythons', 'Pythons', 'Montys', 'Montys',
  'Montys']]
```

Important: To copy the items from a list `foo` to a new list `bar`, you can write `bar = foo[:]`. This copies the object references inside the list. To copy a structure without copying any object references, use `copy.deepcopy()`.

```
>>> ilist[1, 2, 3]>>> ilist2=ilist>>> ilist3=ilist[:]>>> ilist[:
                    from copy import deepcopylst1 = ['a','b',['ab','ba']]l
```

Python provides two ways to check that a pair of items are the same. The `is` operator tests for object identity. We can use it to verify our earlier observations about objects. First, we create a list containing several copies of the same object, and demonstrate that they are not only identical according to ==, but also that they are one and the same object:

```
>>> size = 5
>>> python = ['Python']
>>> snake_nest = [python] * size
>>> snake_nest[0] == snake_nest[1] == snake_nest[2] == snake_nest[3] == snake_nest[4]
True
>>> snake_nest[0] is snake_nest[1] is snake_nest[2] is snake_nest[3] is snake_nest[4]
True
```

```
>>> snake_nest = [python] * 5
>>> snake_nest
[['Python'], ['Python'], ['Python'], ['Python'], ['Python']]
>>> python = ['Python']
>>> snake_nest = [python] * 5
>>> snake_nest
[['Python'], ['Python'], ['Python'], ['Python'], ['Python']]
>>> snake_nest[3]=['Cobra']
>>> snake_nest
[['Python'], ['Python'], ['Python'], ['Cobra'], ['Python']]
>>> snake_nest[1].append('Boa')
>>> snake_nest
[['Python', 'Boa'], ['Python', 'Boa'], ['Python', 'Boa'], ['Cobra'],
  ['Python', 'Boa']]
>>> [id(snake) for snake in snake_nest]
[2942339148L, 2942339148L, 2942339148L, 2941265932L, 2942339148L]
```

Conditionals

In the condition part of an `if` statement, a non-empty string or list is evaluated as true, while an empty string or list evaluates as false.

```
>>> mixed = ['cat', '', ['dog'], []]
>>> for element in mixed:
...     if element:
...         print element
...
cat
['dog']
```

That is, we *don't* need to say `if len(element) > 0:` in the condition.

```
>>> sent = ['No', 'good', 'fish', 'goes', 'anywhere', 'without', 'a',
 'porpoise', '.']
>>> all(len(w) > 4 for w in sent)
False
>>> any(len(w) > 4 for w in sent)
True
>>> [(len(w) > 4 for w in sent)]
[<generator object <genexpr> at 0xb653e43c>]
>>> [len(w) > 4 for w in sent]
[False, False, False, False, True, True, False, True, False]
```

Caution!

Tuples are constructed using the comma operator. Parentheses are a more general feature of Python syntax, designed for grouping. A tuple containing the single element `'snark'` is defined by adding a trailing comma, like this: `'snark',`. The empty tuple is a special case, and is defined using empty parentheses `()`.

```
>>> first_list = [1,2,3,4]
my_set=set(first_list)
my_list = list(my_set)
>>> first_list
[1, 2, 3, 4]
>>> my_set
set([1, 2, 3, 4])
>>> my_list
[1, 2, 3, 4]
```

Table 4-1. Various ways to iterate over sequences

Python expression	Comment
`for item in s`	Iterate over the items of s
`for item in sorted(s)`	Iterate over the items of s in order
`for item in set(s)`	Iterate over unique elements of s
`for item in reversed(s)`	Iterate over elements of s in reverse
`for item in set(s).difference(t)`	Iterate over elements of s not in t
`for item in random.shuffle(s)`	Iterate over elements of s in random order

The sequence functions illustrated in Table 4-1 can be combined in various ways; for example, to get unique elements of `s` sorted in reverse, use `reversed(sorted(set(s)))`.

We can convert between these sequence types. For example, `tuple(s)` converts any kind of sequence into a tuple, and `list(s)` converts any kind of sequence into a list. We can convert a list of strings to a single string using the `join()` function, e.g., `':'.join(words)`.

Some other objects, such as a `FreqDist`, can be converted into a sequence (using `list()`) and support iteration:

```
>>> raw = 'Red lorry, yellow lorry, red lorry, yellow lorry.'
>>> text = nltk.word_tokenize(raw)
>>> fdist = nltk.FreqDist(text)
>>> list(fdist)
['lorry', ',', 'yellow', '.', 'Red', 'red']
>>> for key in fdist:
...     print fdist[key],
...
4 3 2 1 1 1
```

In the next example, we use tuples to re-arrange the contents of our list. (We can omit the parentheses because the comma has higher precedence than assignment.)

```
>>> words = ['I', 'turned', 'off', 'the', 'spectroroute']
>>> words[2], words[3], words[4] = words[3], words[4], words[2]
>>> words
['I', 'turned', 'the', 'spectroroute', 'off']
```

There are also functions that modify the structure of a sequence, which can be handy for language processing. Thus, zip() takes the items of two or more sequences and "zips" them together into a single list of pairs. Given a sequence s, enumerate(s) returns pairs consisting of an index and the item at that index.

```
>>> words = ['I', 'turned', 'off', 'the', 'spectroroute']
>>> tags = ['noun', 'verb', 'prep', 'det', 'noun']
>>> zip(words, tags)
[('I', 'noun'), ('turned', 'verb'), ('off', 'prep'),
('the', 'det'), ('spectroroute', 'noun')]
>>> list(enumerate(words))
[(0, 'I'), (1, 'turned'), (2, 'off'), (3, 'the'), (4, 'spectroroute')]
```

```
>>> text = nltk.corpus.nps_chat.words()
>>> cut = int(0.9 * len(text)) ❶
>>> training_data, test_data = text[:cut], text[cut:] ❷
>>> text == training_data + test_data ❸
True
>>> len(training_data) / len(test_data) ❹
9
```

```
>>> words = 'I turned off the spectroroute'.split()
>>> wordlens = [(len(word), word) for word in words]
>>> wordlens
[(1, 'I'), (6, 'turned'), (3, 'off'), (3, 'the'), (12,
 'spectroroute')]
>>> wordlens.sort()
>>> wordlens
[(1, 'I'), (3, 'off'), (3, 'the'), (6, 'turned'), (12,
 'spectroroute')]
>>> ' '.join(w for (_, w) in wordlens)
'I off the turned spectroroute'
>>> ' '.join({w:k} for (k, w) in wordlens)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: sequence item 0: expected string, dict found
>>> ' '.join(w*k for (k, w) in wordlens)
'I offoffoff thethethe turnedturnedturnedturnedturned
 spectroroutespectroroutespectroroutespectroroutespectroroutespect
```

```
>>> lexicon = [
...     ('the', 'det', ['Di:', 'D@']),
...     ('off', 'prep', ['Qf', 'O:f'])
... ]
```

Here, a lexicon is represented as a list because it is a collection of objects of a single type—lexical entries—of no predetermined length. An individual entry is represented as a tuple because it is a collection of objects with different interpretations, such as the orthographic form, the part-of-speech, and the pronunciations (represented in the SAMPA computer-readable phonetic alphabet; see *http://www.phon.ucl.ac.uk/home/sampa/*). Note that these pronunciations are stored using a list. (Why?)

```
>>> max([w.lower() for w in nltk.word_tokenize(text)]) ❶
'word'
>>> max(w.lower() for w in nltk.word_tokenize(text)) ❷
'word'
```

The second line uses a **generator expression**. This is more than a notational convenience: in many language processing situations, generator expressions will be more efficient. In ❶, storage for the list object must be allocated before the value of `max()` is computed. If the text is very large, this could be slow. In ❷, the data is streamed to the calling function. Since the calling function simply has to find the maximum value—the word that comes latest in lexicographic sort order—it can process the stream of data without having to store anything more than the maximum value seen so far.

There is a common pattern to all of these examples: `[w for w in text if condition]`, where *condition* is a Python "test" that yields either true or false. In the cases shown in the previous code example, the condition is always a numerical comparison. However, we can also test various properties of words, using the functions listed in Table 1-4.

Table 1-4. Some word comparison operators

Function	Meaning
`s.startswith(t)`	Test if s starts with t
`s.endswith(t)`	Test if s ends with t
`t in s`	Test if t is contained inside s
`s.islower()`	Test if all cased characters in s are lowercase
`s.isupper()`	Test if all cased characters in s are uppercase
`s.isalpha()`	Test if all characters in s are alphabetic
`s.isalnum()`	Test if all characters in s are alphanumeric
`s.isdigit()`	Test if all characters in s are digits
`s.istitle()`	Test if s is titlecased (all words in s have initial capitals)

Finally, let's combine the idioms we've been exploring. First, we create a list of *cie* and *cei* words, then we loop over each item and print it. Notice the comma at the end of the print statement, which tells Python to produce its output on a single line.

```
>>> tricky = sorted([w for w in set(text2) if 'cie' in w or 'cei' in w])
>>> for word in tricky:
...     print word,
ancient ceiling conceit conceited conceive conscience
conscientious conscientiously deceitful deceive ...
>>>
```

Gutenberg-Projekte: Elektronische Edition älterer Texte

Definition (Korpus (sächlich, im Plural Korpora))

Ein Korpus ist eine Sammlung von Texten.

Sammlung vorwiegend englischsprachiger Bücher

Sammlung von über 50'000 frei verfügbaren Büchern, deren Copyright abgelaufen ist in den USA.
`http://www.gutenberg.org`

Sammlung deutschsprachiger Bücher

Sammlung von über 7'000 frei verfügbaren Büchern, deren Copyright abgelaufen ist in Deutschland. D.h. 70 Jahre nach dem Tod des Autors oder Übersetzers.
`http://gutenberg.spiegel.de`

Das Package `nltk.corpus`

Enthält Packages und Module, mit denen Korpora in verschiedensten Formaten eingelesen werden können.

Das Objekt `nltk.corpus.gutenberg`

Stellt eine Auswahl von 18 englischsprachigen Gutenberg-Texten (ASCII) als Teil der NLTK-Korpusdaten zum Einlesen zur Verfügung.

Funktionen des Objekts `nltk.corpus.gutenberg`

Repräsentationen für reine Text-Korpora (ASCII oder iso-latin-1) ▸1

```
from nltk.corpus import gutenberg

filename = 'austen-emma.txt' # oder absoluter Pfad einer Textdatei

# Korpus als eine lange Zeichenkette
emma_chars = gutenberg.raw(filename)

# Korpus als Liste von Wörtern (Wort ist Zeichenkette)
emma_words = gutenberg.words(filename)

# Korpus als Liste von Sätzen (Satz ist Liste von Wörten)
emma_sents = gutenberg.sents(filename)

# Korpus als Liste von Paragraphen (Paragraph ist Liste von Sätzen)
emma_paras = gutenberg.paras(filename)
```

Methoden des Objekts `nltk.corpus.brown`[2]

Zusätzlich zu den Funktionen von Textkorpora, gibts Listen mit Paaren aus einem Token und seinem POS-Tag.

Repräsentationen für getaggte Korpora ▸2

```
from nltk.corpus import brown

# Korpus als Liste von 2-Tupeln (Wort, POS-Tag)
brown_tagged_words = brown.tagged_words()
```

Eigenheiten des Brownkorpus: Unterschiedliche Textsorten

```
# Das balancierte Korpus umfasst Texte aus 15 Kategorien
brown.categories()
```

Arten von Korpora: Korpus-Typologie

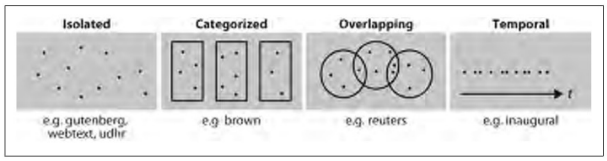


Figure 2-3. Common structures for text corpora: The simplest kind of corpus is a collection of isolated texts with no particular organization; some corpora are structured into categories, such as genre (Brown Corpus); some categorizations overlap, such as topic categories (Reuters Corpus); other corpora represent language use over time (Inaugural Address Corpus).

Quelle: [Bird et al. 2009, 50]

- Die Texte in einem Korpus (Textsammlung, d.h. mehrere Texte) können in unterschiedlicher Ordnung zueinander stehen.
- Ein Korpus kann balanciert (repräsentativ zusammengestellt) oder opportunistisch (nimm, was du kannst!) sein.

Word Tally

the	𝍸 𝍸 𝍸 ||||
been	𝍸 𝍸 |
message	||||
persevere	|
nation	𝍸 |||

- Deskriptive Statistiken: Fundamentale Funktionalität in korpuslinguistischen Auswertungen
- Buchstaben, Wörter, Eigennamen, Sätze, Paragraphen zählen
- Minima, Maxima und Mittelwerte ermitteln
- Verteilung der Häufigkeiten darstellen (Tabelle, Plots)
- Letztlich: Verteilungen vergleichen

Häufigkeitsverteilungen als allgemeine Datenstruktur

- Allgemein: Häufigkeit der Items einer variierenden Grösse (eine statistische Variable) auszählen
- NLTK-Klasse `nltk.FreqDist` ist eine Datenstruktur zum einfachen Erstellen von Häufigkeitsverteilungen (*frequency distribution*)

(Abstrakte) Datenstrukturen

"In der Informatik und Softwaretechnik ist eine Datenstruktur ein Objekt zur Speicherung und Organisation von Daten. Es handelt sich um eine Struktur, weil die Daten in einer bestimmten Art und Weise angeordnet und verknüpft werden, um den Zugriff auf sie und ihre Verwaltung effizient zu ermöglichen. Datenstrukturen sind nicht nur durch die enthaltenen Daten charakterisiert, sondern vor allem durch die Operationen auf diesen Daten, die Zugriff und Verwaltung ermöglichen und realisieren."
(`http://de.wikipedia.org/wiki/Datenstruktur`)

www.nltk.org/api/nltk.html#nltk.probability.FreqDist.N

```
>>> from nltk.tokenize import word_tokenize
>>> from nltk.probability import FreqDist
>>> sent = 'This is an example sentence'
>>> fdist = FreqDist()
>>> for word in word_tokenize(sent):
...     fdist[word.lower()] += 1
```

An equivalent way to do this is with the initializer:

```
>>> fdist = FreqDist(word.lower() for word in word_tokenize(sent))
```

Formally, a frequency distribution can be defined as a function mapping from each sample to the number of times that sample occurred as an outcome. Frequency distributions are generally constructed by running a number of experiments, and incrementing the count for a sample every time it is an outcome of an experiment. For example, the following code will produce a frequency distribution that encodes how often each word occurs in a text:

Berechnen der häufigsten längsten Wörter

```
import nltk
from nltk.corpus import gutenberg

emma_words = gutenberg.words('austen-emma.txt')

emma_fd = nltk.FreqDist(emma_words)

# Finde alle Wörter für die gilt:
# - mehr als 10 Buchstaben und
# - kommen mindestens 7 mal vor

wl = sorted([w for w in emma_fd.keys()
             if len(w)>10 and emma_fd[w]> 7])
```

Bivariate (bedingte) Häufigkeitsverteilungen

Condition: News

the	𝍸 𝍸 𝍸 ||||
cute	
Monday	𝍸 ||||
could	|
will	𝍸 |||

Condition: Romance

the	𝍸 𝍸 |||
cute	|||
Monday	|
could	𝍸 𝍸 𝍸
will	||||

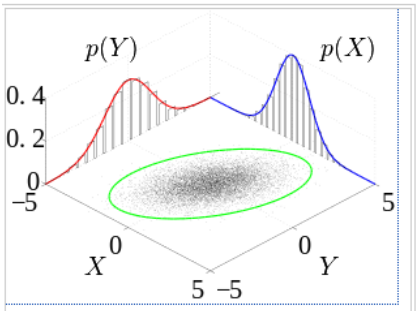
Gemeinsame Häufigkeit der Items von 2 variierenden Grössen (zweier statistischer Variable) auszählen

- Sprechweise: Die eine Variable heisst in NLTK Bedingung (*condition*), die andere Ereignis (*event*, *sample*)
- Eine bedingte Häufigkeitsverteilung besteht pro Bedingung aus einer einfachen Häufigkeitsverteilung.
- NLTK-Klasse `nltk.ConditionalFreqDist` umfasst geeignete Methoden für Frequenzdistributionen von Paaren (=2er-Tupel): (*condition*,*sample*)
- Beispiel: Mit den 15 Kategorien im Brownkorpus ergeben sich 15 Bedingungen mit insgesamt 1'161'192 Events (Wörtern).

Joint probability distribution

In the study of probability, given at least two random variables X, Y, ..., that are defined on a probability space, the **joint probability distribution** for X, Y, ... is a probability distribution that gives the probability that each of X, Y, ... falls in any particular range or discrete set of values specified for that variable. In the case of only two random variables, this is called a **bivariate distribution**, but the concept generalizes to any number of random variables, giving a **multivariate distribution**.

The joint probability distribution can be expressed either in terms of a joint cumulative distribution function or in terms of a joint probability density function (in the case of continuous variables) or joint probability mass function (in the case of discrete variables). These in turn can be used to find two other types of distributions: the marginal distribution giving the probabilities for any one of the variables with no reference to any specific ranges of values for the other variables, and the conditional probability distribution giving the probabilities for any subset of the variables conditional on particular values of the remaining variables.



Many sample observations (black) are shown from a joint probability distribution. The marginal densities are shown as well.

Bedingte Häufigkeiten berechnen

Modalverben in Abhängigkeit von Textkategorien

```
import nltk
from nltk.corpus import brown

cfd = nltk.ConditionalFreqDist([
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre)])

genres = ['news', 'religion', 'hobbies',
        'science_fiction', 'romance', 'humor']

modals = ['can', 'could', 'may', 'might', 'must', 'will']

cfd.tabulate(conditions=genres, samples=modals)
```

Funktionen der Klasse `nltk.ConditionalFreqDist`

Table 2-4. NLTK's conditional frequency distributions: Commonly used methods and idioms for defining, accessing, and visualizing a conditional frequency distribution of counters

Example	Description
`cfdist = ConditionalFreqDist(pairs)`	Create a conditional frequency distribution from a list of pairs
`cfdist.conditions()`	Alphabetically sorted list of conditions
`cfdist[condition]`	The frequency distribution for this condition
`cfdist[condition][sample]`	Frequency for the given sample for this condition
`cfdist.tabulate()`	Tabulate the conditional frequency distribution
`cfdist.tabulate(samples, conditions)`	Tabulation limited to the specified samples and conditions
`cfdist.plot()`	Graphical plot of the conditional frequency distribution
`cfdist.plot(samples, conditions)`	Graphical plot limited to the specified samples and conditions
`cfdist1 < cfdist2`	Test if samples in `cfdist1` occur less frequently than in `cfdist2`

Sequenz-Datentypen: list, str, unicode, tuple

Definition (Sequenz = Endliche Folge von Objekten)

- Zugriff auf Elemente einer Sequenz mittels ganzzahligem Index: `s[i]`
- Zugriff auf Abschnitte (slice) mittels Angabe von Start- und exklusiver Endposition: `s[start:end]`
- Bestimmen der Anzahl Element mittels `len(s)`

Typen von Sequenzen und ihre Notation

		Länge		
Type	***Mutable***	**0**	**1**	**2**
list	Ja	`[]`	`[1]`	`[1,'n']`
str	Nein	`''`	`'1'`	`'ab'`
tuple	Nein	`()`	`(1,)`	`(1,'n')`

Listen: Veränderliche (*mutable*) Sequenzen ▶

Typische Modifikationen für Listen

```
l = []                # Zuweisung ist keine Listenmodifikation!
l.append(1)           # ein Element anhängen
l.extend((4,'x',5))   # eine ganze Sequenz anhängen
del l[3]              # ein Element löschen
l[2] = 3              # eine Element austauschen
l.sort(reverse=True) # in-place rückwärts sortieren
print l
```

- Nur bei Listen, d.h. veränderlichen Sequenzen, können Elemente (oder Abschnitte (*slices*)) gelöscht, ersetzt oder ergänzt werden.
- : Listen-Methoden, welche in-place-Modifikationen durchführen, liefern als Rückgabewert `None` zurück und nicht die Liste!

Set aus Liste mit unveränderlichen Elementen generieren->ok
```
>>> iset=set(["m",1,("b","m")])
>>> iset
set([1, 'm', ('b', 'm')])
```
Set aus Tupel (enthält selbst nur unveränderliche Elemente) generieren->ok
```
>>> iset=set(("m",1,("b","m")))
>>> iset
set([1, 'm', ('b', 'm')])
```
Set ohne umschliessende Klammern um Elemente generieren->error
```
>>> iset=set("m",1,("b","m"))
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: set expected at most 1 arguments, got 3
```
Set mit Element Liste oder anderen veränderlichen Datentypen->error
```
>>> iset=set(("m",1,["b","m"]))
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unhashable type: 'list'
```
Set Index abfragen->error
```
>>> iset[3]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Syntaktischer Zucker für Methoden von Sequenzen

Python bietet für wichtige Methoden von Sequenzen Spezialnotation an. Ob die Spezialnotation funktioniert, hängt nur davon ab, ob mein Objekt die entsprechende Methode kann!

Enthalten (*Membership*)

```
>>> 3 in [1,2,3]
True
>>> [1,2,3].__contains__(3)
True
```

i-tes Element

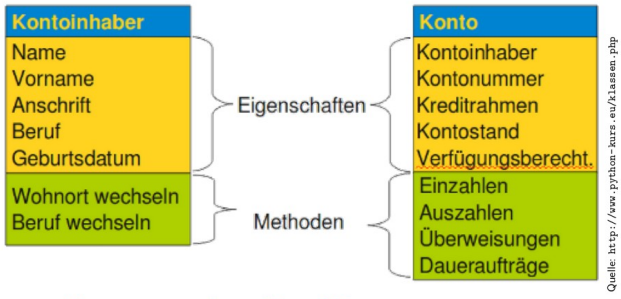
```
>>> ('a','c')[1]
'c'
>>> ('a','c').__getitem__(1)
'c'
```

Abschnitt (*Slicing*)

```
>>> "ABBA"[1:3]
'BB'
>>> "ABBA".__getslice__(1,3)
'BB'
```

```
>>> help(str.__getitem__)
Help on wrapper_descriptor:
__getitem__(...)
    x.__getitem__(y) <==> x[y]
```

Objektorientierte Modellierung



Mittels Klassen können eigene (Daten-)Typen geschaffen werden.

Typen/Klassen und Objekte

Gattungen und Individuen in der Welt

Gattung	Individuum
Mensch	Elvis Presley
Hauptstadt	Paris

Typen/Klassen und Objekte/Klassen-Instanzen in Python

Typ/Klasse	Objekt/Instanz
int	3
str	'abc'
list	[1,2,3]

Wichtig: Objekte sind Instanzen eines Typs oder einer Klasse.

Typ-Aufrufe als Objekt-Konstruktoren

Konstruiere Objekte von einem bestimmten Typ, indem du den Typ wie eine Funktion aufrufst!

Viele Konstruktor-Funktionen erlauben Argumente.
Erklärungen gibt `help(type)`.

Default-Objekte

```
>>> unicode()
u''
>>> int()
0
>>> list()
[]
>>> dict()
{}
>>> set()
set([])
```

Konstruktoren mit Parametern

```
>>> str(123)
'123'
>>> int('10110',2)
22
>>> set([3,3,2,2,'a',1.1,'a'])
set(['a', 2, 3, 1.1])
>>> list(set([2,1,'a']))
['a', 1, 2]
>>> dict(a='DET',do='VB')
{'a': 'DET', 'do': 'VB'}
```

Objekte konstruieren mittels Klassen/Typ-Konstruktor

Konstruktoren (*constructors*) vs Literale (*literals, displays*)

- Nur die Objekte der wichtigsten eingebauten Datentypen können als Literale oder mit Spezialnotation konstruiert werden.
- Konstrukturen der Form `TYPE()` sind immer möglich!

Typ/Klasse	Objekt/Instanz herstellen
nltk.probability.FreqDist	`nltk.FreqDist("abrakadabra")`
collections.Counter	`collections.Counter('abrakadabra')`

Hinweis: Die Klasse `nltk.probability.FreqDist` erweitert die eingebaute Python-Klasse `collections.Counter` ▶!
Klassenbezeichner sind wichtig! Keine Objekte ohne Typen/Klassen!

8.3.1. `Counter` objects

A counter tool is provided to support convenient and rapid tallies. For example:

```
>>> # Tally occurrences of words in a list
>>> cnt = Counter()
>>> for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
...     cnt[word] += 1
>>> cnt
Counter({'blue': 3, 'red': 2, 'green': 1})

>>> # Find the ten most common words in Hamlet
>>> import re
>>> words = re.findall(r'\w+', open('hamlet.txt').read().lower())
>>> Counter(words).most_common(10)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669), ('i', 631),
 ('you', 554),  ('a', 546), ('my', 514), ('hamlet', 471), ('in', 451)]
```

Common patterns for working with `Counter` objects:

```
sum(c.values())                 # total of all counts
c.clear()                       # reset all counts
list(c)                         # list unique elements
set(c)                          # convert to a set
dict(c)                         # convert to a regular dictionary
c.items()                       # convert to a list of (elem, cnt) pairs
Counter(dict(list_of_pairs))    # convert from a list of (elem, cnt) pairs
c.most_common()[:-n-1:-1]       # n least common elements
c += Counter()                  # remove zero and negative counts
```

Ein Blick hinter die Kulisse: Methodenaufrufe

Schein: Objekt ruft seine Methode auf

OBJECT.METHOD(ARG1)

Sein: Klasse/Type ruft Methode mit Objekt als 1. Argument auf

CLASS.METHOD(OBJECT, ARG1)

Schein	Sein
`"A Test".lower()`	`str.lower("A Test")`
`"ABBA".count('A')`	`str.count("ABBA",'A')`

Das macht VIEL Sinn!

Die Addition definiert man auch auf der Klasse der Ganzzahlen, nicht für jede Zahl einzeln!
Objekte rufen Methoden auf, welche auf Klassenebene definiert sind!

Unterschied zwischen *Statements* und *Expressions*

Anweisungen (*statements*) ▸5

werden vom Python-Interpreter ausgeführt und evaluieren zu keinem Wert.

`print` Statement ▸6

```
print "Something to print"
```

Ausdrücke (*expressions*) ▸7

werden zu einem Wert (Objekt) evaluiert und enthalten keine Statements.

Boole'sche und andere Ausdrücke innerhalb von Statements

```
# If-Statement mit komplexen Ausdrücken drin
if len("A "+"String") > 5:
    print "A "+"String".lower()
```

Listenbildung via Anweisungen und Ausdruck

Listenbildung mit iterativen Statements

```
sl = list()
for c in "St. Moritz-Str. 23":
    if c.isalnum():
        sl.append(c.lower())
```

Listenbildung mit einem Ausdruck: Listenkomprehension

```
el = [c.lower() for c in "St. Moritz-Str. 23" if c.isalnum()]
```

If-then-else als Anweisung und If-Else als Ausdruck

Listenbildung mit iterativen Statements

```
sl = []
for c in "St. Moritz-Str. 23":
    if c.isalnum():
        sl.append(c)
    else:
        sl.append(' ')
```

Default-if-else Ausdruck

```
el = [ c if c.isalnum() else ' ' for c in "St. Moritz-Str. 23" ]
```

: Abweichende Reihenfolge von if-then-else-Bestandteilen, da typischerweise der Then-Ausdruck der Standardwert ist.

Funktionsdefinition via Anweisungen und Ausdruck

Funktionsdefinition mit iterativem Statement ▸8

```
def sf(s):
    return re.sub(r'\s+','',s)
```

Funktionsdefinition via Lambda-Ausdruck

```
ef = lambda s: re.sub(r'\s+','',s)
```

Lambda-Ausdrücke (Lambda expression))

Mathematische Notation zur Definition von anonymen Funktionen:

- Funktionsdefinition (Rechenvorschrift): $(\lambda x : x + 1)$
- Funktionsevaluation: $(\lambda x : x + 1)(3) = 4$
- Lambda bindet/abstrahiert die Funktionsparameter im Funktionsrumpf
- Kurz: Parametrisierte Ausdrücke

Komprehension von Mengen und Dictionaries

Mengenkomprehension und iterative Lösung

```
mc = {x.lower() for x in "Das Alphabet" if x.isalnum()}

ms = set()
for x in "Das Alphabet":
    if x.isalnum():
        ms.add(x.lower())
```

Komprehension von Dictionaries

```
text = "abrakadabra"
dc = {c:text.count(c) for c in set(text)}
```

Wie würde man das iterativ programmieren?

Tupel vs Set

```
#Tupel CAN contain mutable objects:
>>> itupel = ([1, 2, 3], [3, 2, 1])
>>> itupel
([1, 2, 3], [3, 2, 1])
>>> itupel=(("m",1,["b","m"]))
>>> itupel
('m', 1, ['b', 'm'])
#Set CANNOT contain mutable objects:
>>> iset=set(("m",1,["b","m"]))
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unhashable type: 'list'
# Tupel ARE indexable
>>> itupel[2][1]
'm'
# Set ARE NOT indexable
>>> iset[3]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'set' object does not support indexing
#Tupel CAN change mutable objects:
>>> itupel[2][1]="aha"
>>> itupel
('m', 1, ['b', 'aha'])
#Tupel CANNOT change immutable objects:
>>> itupel[1][1]="aha"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'int' object does not support item assignment
>>> del(itupel[1][1])
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'int' object does not support item deletion
```

```
#List
lc = [x.lower() for x in "Das Alphabet. 12" if x.isalnum()]
print lc
lc = list(x.lower() for x in "Das Alphabet. 12" if x.isalnum())
print lc
#Sets
mc = {x.lower() for x in "Das Alphabet. 12" if x.isalnum()}
print mc
mc = set(x.lower() for x in "Das Alphabet. 12" if x.isalnum())
print mc
#Dictionary
text = "abrakadabra. 12"
dc = {c:text.count(c) for c in set(text)}
print dc
dc = dict((c,text.count(c)) for c in set(text))
print dc
```

```
#List
['d', 'a', 's', 'a', 'l', 'p', 'h', 'a', 'b', 'e', 't', '1', '2']
['d', 'a', 's', 'a', 'l', 'p', 'h', 'a', 'b', 'e', 't', '1', '2']
#Sets
set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p', 's', '2', 't'])
set(['a', 'b', 'e', 'd', 'h', 'l', '1', 'p', 's', '2', 't'])
#Dictionary
{'a': 5, ' ': 1, 'b': 2, 'd': 1, 'k': 1, '.': 1, '1': 1, 'r': 2, '2': 1}
{'a': 5, ' ': 1, 'b': 2, 'd': 1, 'k': 1, '.': 1, '1': 1, 'r': 2, '2': 1}
```

Sequence Types -- str, unicode, list, tuple, buffer, xrange
An iterable which supports efficient element access using integer indices via the __getitem__() special method and defines a len() method that returns the length of the sequence. Some built-in sequence types are list, str, tuple, and unicode. Note that dict also supports __getitem__() and __len__(), but is considered a mapping rather than a sequence because the lookups use arbitrary immutable keys rather than integers.

Mapping Types -- dict
A *mapping* object maps immutable values (keys) to arbitrary objects (values). Mappings (dict) are mutable objects. There is currently only one standard mapping type, the *dictionary*. A dictionary's keys are almost arbitrary values. Only values containing lists, dictionaries or other mutable types (that are compared by value rather than by object identity) may not be used as keys.

Iterable Types – str, unicode, list, tuple, dict, file …
Examples of iterables include all sequence types (such as list, str, and tuple) and some non-sequence types like dict and file and objects of any classes you define with an __iter__() or __getitem__() method. Iterables can be used in a for loop and in many other places where a sequence is needed (zip(), map(), ...).

Wortlisten als Lexika

Definition (Wortlisten)

Die einfachste Form von Lexika sind Wortlisten.
Als Rohtext-Datei typischerweise 1 Wort pro Zeile und sortiert.

Stoppwortlisten (*stopwords*) in NLTK ▸1

```
stopwords_en = nltk.corpus.stopwords.words('english')

print len(stopwords_en), stopwords_en[::20]
# >>> 127 ['i', 'herself', 'was', 'because', 'from', 'any', 't']
```

Hinweis: Spezialsyntax `[::20]` gibt jedes 20. Element zurück.

```
import nltk

stopwords_en = nltk.corpus.stopwords.words('english')
print len(stopwords_en), stopwords_en[::20]  # 127
    # [u'i', u'herself', u'was', u'because', u'from',
    # u'any', u't']

### Using German stopword list this way suffers from
### encoding problems
stopwords_de = nltk.corpus.stopwords.words('german')
print len(stopwords_de), sorted(stopwords_de)[::20] # 231
    # [u'aber', u'auch', u'dem', u'dieser', u'er', u'hin',
    # u'jeder', u'manchem', u'nur', u'solches', u'warst',
    # u'wollen']
```

Anteil von echten Inhaltswörtern bestimmen

Wie kann man die Interpunktionstoken eliminieren? ▸3

```
import re

def delete_punctuation(s):
    """ Return string with all punctuation symbols of iso-latin 1 deleted.
    p = r'[!"#%&\x27`()*,-./:;?@[\]_{}\xa1\xab\xb7\xbb\xbf]'
    return re.sub(p,'',s)

def content_word_percentage(text):
    """ Return the percentage of content words in a list of English tokens.
    content_words = [w for w in text
                     if delete_punctuation(w) != ''
                     and w.lower() not in stopwords_en_set]
```

```
from __future__ import division
import nltk

#Brown words
brown_words=nltk.corpus.brown.words()
print "\nfrom 0.0: ", len(brown_words), type(brown_words)
print brown_words[:20]

#List: Inffizienterer Lookup für die Stoppwörter mit Listen
stopwords_en = nltk.corpus.stopwords.words('english')
print "\nfrom 0.1: ", len(stopwords_en), stopwords_en[::20]

# Was berechnet foo()?
def foo(text):
    """ Hier fehlt Dokumentation... """
    bar = [w for w in text
           if w.lower() not in stopwords_en]
    print "\nfrom 1.1: ", len(bar), len(text)
    return len(bar)/len(text)*100.
print "\nfrom 1.2: ", foo(brown_words)

# Set: Effizienterer Lookup für die Stoppwörter mit Mengen
stopwords_en_set = {w for w in stopwords_en}
print "\nfrom 2: ", len(stopwords_en_set), stopwords_en_set

# Optimierte Funktion
def non_stopwords_percentage(text):
    """ Return the percentage of non-stopwords
    in a list of English tokens. """
    non_stopwords = [w for w in text
                     if w.lower() not in stopwords_en_set]
    print "\nfrom 3.1: ", len(non_stopwords), len(text)
    return len(non_stopwords)/len(text)*100.
print "\nfrom 3.2: ", non_stopwords_percentage\
                        (brown_words)
print "\nfrom 3.3: "
help(non_stopwords_percentage)

# Wie misst man die Effizienz?
#============================
import cProfile

# Vorberechnen der Liste mit 1 Mio. Token
brown_words_list = list(nltk.corpus.brown.words())
print "\nfrom 4: ", len(brown_words_list), type(brown_words_list)
print brown_words_list[:20]

# Ineffizientere Verarbeitung mit Stoppwortlisten
print "\nfrom 5.1: "
cProfile.run("foo(brown_words)")
print "\nfrom 5.2: "
cProfile.run("foo(brown_words_list)")

# Effizientere Verarbeitung mit Stoppwortmenge
# (oder Dictionary mit Dummy-Werten)
print "\nfrom 6.1: "
cProfile.run("non_stopwords_percentage(brown_words)")
print "\nfrom 6.2: "
cProfile.run("non_stopwords_percentage(brown_words_list)")
```

foo_fraction_en.py"

Die Ausgabe sieht aus wie eine Liste ist aber ein ineffizientes Objekt
from 0.0: 1161192 <class 'nltk.corpus.reader.util.ConcatenatedCorpusView'>
[u'The', u'Fulton', u'County', u'Grand', u'Jury', u'said', u'Friday', u'an', u'investigation', u'of', u"Atlanta's", u'recent', u'primary', u'election', u'produced', u'``', u'no', u'evidence', u"''", u'that']

from 0.1: 127 <type 'list'>
[u'i', u'herself', u'was', u'because', u'from', u'any', u't']

Neue Divisionsregeln importieren sonst berechnet die Funktion immer 0
from 1.2:
from 1.1: 688946 1161192
59.3309289075

Mit set Datenstruktur geht's schneller
from 2: 127 set([u'all', u'just', u'being', u'over', u'both', u'through', u'yourselves', u'its', u'before', u'herself', u'had', u'should', u'to', u'only', u'under', u'ours', u'has', u'do', u'them', u'his', u'very', gekürzt...])

Neue Divisionsregeln importieren sonst berechnet die Funktion immer 0
from 3.2:
from 3.1: 688946 1161192
59.3309289075

from 3.3:
Help on function non_stopwords_percentage in module __main__:

non_stopwords_percentage(text)
Return the percentage of non-stopwords
in a list of English tokens.

Mit List Datenstruktur geht's schneller
from 4: 1161192 <type 'list'>
[u'The', u'Fulton', u'County', u'Grand', u'Jury', u'said', u'Friday', u'an', u'investigation', u'of', u"Atlanta's", u'recent', u'primary', u'election', u'produced', u'``', u'no', u'evidence', u"''", u'that']

Effizienzmessungen

Inhaltswörter: class 'nltk.corpus.reader.util.ConcatenatedCorpusView'
Stopwörter: List
from 5.1:
from 1.1: 688946 1161192
10488536 function calls (10488532 primitive calls) in 48.990 seconds

Inhaltswörter: List
Stopwörter: List
from 5.2:
from 1.1: 688946 1161192
1161199 function calls in 10.615 seconds

Inhaltswörter: class 'nltk.corpus.reader.util.ConcatenatedCorpusView'
Stopwörter: Set
from 6.1:
from 3.1: 688946 1161192
10488536 function calls (10488532 primitive calls) in 41.013 seconds

Inhaltswörter: List
Stopwörter: Set
from 6.2:
from 3.1: 688946 1161192
1161199 function calls in 4.211 seconds

```
# Wie kann man die Interpunktionstoken ausfiltern?
#=================================================
import re

def delete_punctuation(s):
    """ Return string with all punctuation symbols
    of iso-latin 1 deleted. """
    p = r'[!"#%&\x27`()*,-./:;?@[\]_{}\xa1\xab\xb7\xbb\xbf]'
    return re.sub(p,'',s)

def content_word_percentage(text):
    """ Return the percentage of content words in
    a list of English tokens. """
```

Die Inhaltswörter machen nicht 59% sonder nur 46% aus
Es gibt nicht mehr 688946 sondern nur noch 541074 Inhaltswörter

from 7.2:
from 7.1: 541074 1161192
46.5964284976

Delete_Punctuation benötigt um die 20 zusätzliche Sekunden
from 8:

from 7.1: 541074 1161192
5658095 function calls in 26.922 seconds

```
    content_words = [w for w in text
                     if delete_punctuation(w) != ''
                     and w.lower() not in stopwords_en_set]
    print "\nfrom 7.1: ", len(content_words), len(text)
    return len(content_words)/len(text)*100.

print "\nfrom 7.2: ", \
    content_word_percentage(brown_words_list)

print "\nfrom 8: "
cProfile.run("content_word_percentage(brown_words_list)")
```

Tokens welche die Funktion verändert. Alle Satzzeichen, welche vorher als Inhaltswort mitgezählt wurden, sind nun aus der Berechnung entfernt worden.

.
,
creativity-oriented creativityoriented
,
,
,
;
;
people-oriented peopleoriented
M.D. MD
father's fathers
;

CMU (Carnegie Mellon University) Pronouncing Dictionary

```
File Format: Each line consists of an uppercased word,
a counter (for alternative pronunciations), and a transcription.
Vowels are marked for stress (1=primary, 2=secondary, 0=no stress).
E.g.: NATURAL 1 N AE1 CH ER0 AH0 L

The dictionary contains 127069 entries.  Of these, 119400 words are assigned
a unique pronunciation, 6830 words have two pronunciations, and 839 words have
three or more pronunciations.  Many of these are fast-speech variants.

Phonemes: There are 39 phonemes, as shown below:

    Phoneme Example Translation    Phoneme Example Translation
    ------- ------- -----------    ------- ------- -----------
    AA      odd     AA D           AE      at      AE T
    AH      hut     HH AH T        AO      ought   AO T
...
$ grep -w  RESEARCH  /Users/siclemat/nltk_data/corpora/cmudict/cmudict
RESEARCH 1 R IY0 S ER1 CH
RESEARCH 2 R IY1 S ER0 CH
```

Wie soll man solche Information in Python als Daten repräsentieren?

CMU (Carnegie Mellon University) Pronouncing Dictionary

Strukturierte Lexikoneinträge

CMU besteht aus Paaren von Lemma und Listen von phonetischen Kodes.

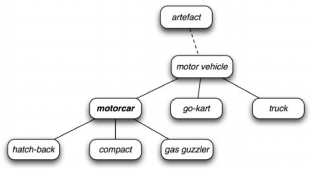
Filtern von Lexikoneinträgen ▸4

```
import nltk

entries = nltk.corpus.cmudict.entries()

print entries[71607]
# ('love', ['L', 'AH1', 'V'])]

# Finde alle Wörter auf -n, welche als -M ausgeprochen werden.
print [ word for (word,pron) in entries
          if  pron[-1] == 'M'
          and word[-1] == 'n' ]
```

WordNet: Ein Netz von Bedeutungsbeziehungen[1]

Wie lässt sich die Bedeutung eines Worts angeben?

- Klassische Charakterisierung: Umschreibung, Definition
- Relationale lexikalische Semantik = Bedeutungsbeziehungen
- Angabe von Synonymen, Hypernymen, Hyponymen, Antonymen usw., welche ein Netz (Hierarchie) von verknüpften Bedeutungen ergeben



Quelle: http://www.nltk.org/images/wordnet-hierarchy.png

WordNet: Komplexe lexikalische Datenstruktur[2]

Speziell zugeschnittene Datenstruktur benötigt

- Zugriff auf Bedeutungen (*synsets*) und Wörter (*lemmas*)
- Navigation im Wortnetz entlang der semantischen Relationen (Oberbegriffe, Unterbegriffe, Gegenbegriffe)
- Berechnen von semantischer Verwandtschaft (Nähe, Bezüge) im Netz

WordNet in NLTK ▸5

```
import nltk
from nltk.corpus import wordnet as wn
# Welche Bedeutungen hat das Wort "car"?
print wn.synsets('car')
# Definition einer Bedeutung
print wn.synset('car.n.01').definition()
# Alle hyponymen Bedeutungen eines Lemmas berechnen
print wn.synset('car.n.01').hyponyms()
```

```
import nltk

# Mit "as" kann man Kurznamen einführen.
from nltk.corpus import wordnet as wn


# <h2>Synsets (unterschiedliche Bedeutungen) eines
# Lemmas berechnen</h2>
# Welche Bedeutungen hat das Wort "car"?
print "\nfrom 1: ", wn.synsets('car')
print "\nfrom 2", wn.synsets('motorcar')

# Definition einer Bedeutung
print "\nfrom 3: ", wn.synset('car.n.01').definition()

# <h2>Alle synonymen Lemmas eines Synsets berechnen</h2>
print "\nfrom 4: ", wn.synset('car.n.01').lemma_names()
print "\nfrom 5: ", wn.synset('car.n.01').lemmas()

# Was ist eine Lemma-Datenstruktur?
print "\nfrom 6: ", type(wn.synset('car.n.01').lemmas()[0])
print "\nfrom 7: "
help(nltk.corpus.reader.wordnet.Lemma)

# Alle hyponymen Bedeutungen eines Lemmas berechnen
print "\nfrom 8: ", wn.synset('car.n.01').hyponyms()
```

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl10/nltk_wordnet.py"

from 1:  [Synset('car.n.01'), Synset('car.n.02'), Synset('car.n
 .03'), Synset('car.n.04'), Synset('cable_car.n.01')]

from 2 [Synset('car.n.01')]

from 3:  a motor vehicle with four wheels; usually propelled by
 an internal combustion engine

from 4:  [u'car', u'auto', u'automobile', u'machine', u'motorcar']

from 5:  [Lemma('car.n.01.car'), Lemma('car.n.01.auto'),
 Lemma('car.n.01.automobile'), Lemma('car.n.01.machine'),
 Lemma('car.n.01.motorcar')]

from 6:  <class 'nltk.corpus.reader.wordnet.Lemma'>

from 7:
Help on class Lemma in module nltk.corpus.reader.wordnet:

class Lemma(_WordNetObject)
 |  The lexical entry for a single morphological form of a
 |  sense-disambiguated word.
 |
```

from 8: [Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bus.n.04'), Synset('cab.n.03'), Synset('compact.n.03'), Synset('convertible.n.01'), Synset('coupe.n.01'), Synset('cruiser.n.01'), Synset('electric.n.01'), Synset('gas_guzzler.n.01'), Synset('hardtop.n.01'), Synset('hatchback.n.01'), Synset('horseless_carriage.n.01'), Synset('hot_rod.n.01'), Synset('jeep.n.01'), Synset('limousine.n.01'), Synset('loaner.n.02'), Synset('minicar.n.01'), Synset('minivan.n.01'), Synset('model_t.n.01'), Synset('pace_car.n.01'), Synset('racer.n.02'), Synset('roadster.n.01'), Synset('sedan.n.01'), Synset('sport_utility.n.01'), Synset('sports_car.n.01'), Synset('stanley_steamer.n.01'), Synset('stock_car.n.01'), Synset('subcompact.n.01'), Synset('touring_car.n.01'), Synset('used-car.n.01')]

```
import nltk

entries = nltk.corpus.cmudict.entries()

print entries[71607] # ('love', ['L', 'AH1', 'V'])]

# Finde alle Wörter auf -n, welche als -M ausgeprochen werden.
print [ word for (word,pron) in entries
            if  pron[-1] == 'M'
            and word[-1] == 'n' ]
```

```
(u'love', [u'L', u'AH1', u'V'])

[u'autumn', u'column', u'condemn', u'damn', u'goddamn', u'hymn', u'solemn']
```

```
global_list = [('der',1200),('die',1000),('das',900)]
print "\nfrom 0: ", global_list

def del_first(l):
    print "\nfrom 1.1: ", 'global_list is parameter l:',\
        global_list is l
    del l[0]
    print "\nfrom 1.2: ", 'global_list is parameter l:',\
        global_list is l

del_first(global_list)
print "\nfrom 2: ", global_list

del_first(global_list)
print "\nfrom 3: ", global_list
```

```
from 0:  [('der', 1200), ('die', 1000), ('das', 900)]

from 1.1:  global_list is parameter l: True

from 1.2:  global_list is parameter l: True

from 2:  [('die', 1000), ('das', 900)]

from 1.1:  global_list is parameter l: True

from 1.2:  global_list is parameter l: True

from 3:  [('das', 900)]

Process finished with exit code 0
```

Elemente hinzuzählen

Inkrementieren

In [18]:
```
fdist.update('a')
fdist
```

Out[18]:
```
Counter({'a': 5, 'b': 2, 'd': 1, 'k': 1, 'r': 2})
```

Wieviele Elemente gibt es insgesamt?

Die Methode N():

In [19]:
```
fdist.N()
```

Out[19]:
```
11
```

Was ist die *absolute Häufigkeit* eines Elements?

In [20]:
```
fdist['r']
```

Out[20]:
```
2
```

Was ist die *relative Häufigkeit* eines Elements?

In [21]:
```
fdist.freq('r')
```

Out[21]:
```
0.18181818181818182
```

Welches Element kommt am häufigsten vor?

In [22]:
```
fdist.max()
```

Out[22]:
```
'a'
```

Distributionen addieren

In [28]:
```
fdist2 = nltk.FreqDist('Hokuspokus')
fdist + fdist2
```

Out[28]:
```
Counter({'H': 1,
         'a': 5,
         'b': 2,
         'd': 1,
         'k': 3,
         'o': 2,
         'p': 1,
         'r': 2,
         's': 2,
         'u': 2})
```

Iteriere durch alle Elemente nach Häufigkeit

In [*]:
```
for item in fdist:
    print item, fdist[item]
```

In [*]:
```
for item,freq in fdist.most_common():
    print item, freq
```

In [26]:
```
for item in sorted(fdist,key=fdist.get,reverse=True):
    print item, fdist[item]
```

```
a 5
r 2
b 2
k 1
d 1
```

```
Zahlen runden
round(0.05,1) ==> 0.1  up
round(0.15,1) ==> 0.1
round(0.25,1) ==> 0.3  up
round(0.35,1) ==> 0.3
round(0.45,1) ==> 0.5  up
round(0.55,1) ==> 0.6
round(0.65,1) ==> 0.7  up
round(0.75,1) ==> 0.8  up
round(0.85,1) ==> 0.8
round(0.95,1) ==> 0.9

round(1.05,1) ==> 1.1  up
round(1.15,1) ==> 1.1
round(1.25,1) ==> 1.3  up
round(1.35,1) ==> 1.4  up
round(1.45,1) ==> 1.4
round(1.55,1) ==> 1.6  up
round(1.65,1) ==> 1.6
round(1.75,1) ==> 1.8  up
round(1.85,1) ==> 1.9  up
round(1.95,1) ==> 1.9

'Runden' durch die Formatierung
'%.1f' % 0.05 ==> 0.1  up
'%.1f' % 0.15 ==> 0.1
'%.1f' % 0.25 ==> 0.2
'%.1f' % 0.35 ==> 0.3
'%.1f' % 0.45 ==> 0.5  up
'%.1f' % 0.55 ==> 0.6  up
'%.1f' % 0.65 ==> 0.7  up
'%.1f' % 0.75 ==> 0.8  up
'%.1f' % 0.85 ==> 0.8
```

```
# Demonstration von Funktionen mit yield
#=======================================
def quadriere(iterierbar):
    for i in iterierbar:
        yield i*i

quadrat = quadriere([10,11])
print "from 1: ", quadrat
print "from 2: ",type(quadrat)

# next() liefert nächstes Element zurück.
print "from 3: ",quadrat.next()
print "from 4: ",quadrat.next()

#Falls keines mehr erhältlich ist, entsteht eine Ausnahme.
# In Iterationskontexten führt diese Ausnahme dazu,
# dass die Iteration beendet wird.
# Die Ausnahme wird "entsprechend" behandelt.
#print "from 5: ",quadrat.next()

# Generatoren sind erschöpft nach einem Durchgang
print "from 7: ",sum(quadrat)

#Aggregationsfunktion auf Generator anwenden
print "from 6: ",sum(quadriere([10,11]))
```

Output
from 1: <generator object quadriere at 0xb7253edc>
from 2: <type 'generator'>
from 3: 100
from 4: 121
from 7: 0
from 6: 221

```
# Demonstration von Generatorausdrücken
quadrat = (i*i for i in [10,11])
print quadrat
print type(quadrat)

# generator hat keine len()
# print len(quadrat)
# next() liefert nächstes Element zurück.
print quadrat.next()
print quadrat.next()

# Durchiterieren
quadrat = (i*i for i in xrange(10,13))
for q in quadrat:
  print q
```

```
 /LubuntuShared/8) PCL-1/Vorlesung/vl11/generators_next.py"
<generator object <genexpr> at 0xb7218edc>
<type 'generator'>
100
121
100
121
144
```

```
while True:
    x = raw_input('Please type in a number: ')
    try:
        float(x)
        print "from 1: "
        break  # nur hier kann Schleife beendet werden
        print "from 2: "
    except ValueError:
        print "from 3: "
        pass
        print "from 4: "
    finally:
        print "from 5: "
print "The number was:", x
```

```
# Please type in a number: null
# from 3:
# from 4:
# from 5:
# Please type in a number: eins
# from 3:
# from 4:
# from 5:
# Please type in a number: 7,78
# from 3:
# from 4:
# from 5:
# Please type in a number: 90.25
# from 1:
# from 5:
# The number was: 90.25
```

```
import sys

#Den Inhalt dieser Datei
# ohne Leerzeilen printen
filename = "with_open.py"

with open(filename,'r') as f:
  for l in f:
    if l.rstrip() != '':
       sys.stdout.write(l)
```

```
# Modul zur Zeitmessung von Python-Statements
#============================================
import timeit
setup = 'import random'
# Effekt von xrange vs. range bei random.sample
# Konstruiere 2 Timer-Objekte
tr = timeit.Timer('random.sample( range(1000000),100)', setup)
tx = timeit.Timer('random.sample(xrange(1000000),100)', setup)
# Führe Timings je einmal durch und speichere Anzahl Sekunden
trsecs = tr.timeit(1)
txsecs = tx.timeit(1)
print "Aufgabe: Sample 100 Zahlen aus dem Bereich 0 bis 999999."
print "Zeit mit xrange:", txsecs, "Sekunden"
print "Zeit mit  range:", trsecs, "Sekunden"
print "xrange ist etwa", trsecs/txsecs, "Mal schneller!"
```

Output
Aufgabe: Sample 100 Zahlen aus dem Bereich 0 bis 999999.
Zeit mit xrange: 0.000220060348511 Sekunden
Zeit mit range: 0.133285045624 Sekunden
xrange ist etwa 605.674972914 Mal schneller!

```
# Rechenzeit und Speichereffizienz messen
#========================================
# HOWTO und Erklärungen
# (1) Programm via Terminal im Hintergrund aufstarten (&):
#     $ python timeit_listcomprehension.py &
#     $ python timeit_generator.py &
#     Das Programm gibt jeweils die Prozess-ID aus, unter der es
läuft.
#     Damit können die Programme auseinander gehalten werden.
#     Im Folgenden steht PID1 bzw. PID2 für die jeweilge
nummerische ID.
```

Im Terminal:

mit „&“ wird der Prozess im Hintergrund gestartet
$ python timeit_listcomprehension.py &
[1] 3720
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Vorlesung/vl11$ process id: 3720
python timeit_generator.py &
[2] 3722
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8) PCL-1/Vorlesung/vl11$ process id: 3722

```
# (2) Speicherverbrauch messen im Terminal mit ps:
#     $ ps -vp PID1
#     $ ps -vp PID2
#     Die entscheidende Information steckt im Attribut RSS
(resident set size),
#     das den physikalisch belegten Speicher in Kilobytes (im
Gegensatz zum
#     virtuellen Speicher) misst.
# (3) Speicherverbrauch messen mit Task Manager (WIN) oder
Aktivitätsanzeige (MACOS)
#     Am besten Filtern auf python, danach den Prozess über seine
Prozess-ID identifizieren.

timeit_listcomprehension.py
import nltk, timeit, time, os
print 'process id:', os.getpid()
words = nltk.corpus.brown.words()
def test_listcomprehension():
    return set([w.lower() for w in words])
# Initialisiere Timer-Objekt
tl = timeit.Timer(test_listcomprehension)
# Timing von Listenkomprehension
print 'Timed list comprehension (seconds):', tl.timeit(1)
# Speicherverbrauch muss von ausserhalb gemessen werden.
# Es gibt keinen einfachen Weg, das innerhalb von Python zu
machen.
time.sleep(600) # 600 Sekunden

timeit_generator.py
import nltk, timeit, time, os
print 'process id:', os.getpid()
words = nltk.corpus.brown.words()
def test_generator():
    return set(w.lower() for w in words)
# Initialisiere Timer-Objekt
tg = timeit.Timer(test_generator)
# Timing von Generatorausdruck
print 'Timed generator (seconds):', tg.timeit(1)
# Speicherverbrauch muss von ausserhalb gemessen werden.
# Es gibt keinen einfachen Weg, das innerhalb von Python zu
machen.
time.sleep(600) # 600 Sekunden
```

```
benzro@benzro-eMachines E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8)
PCL-1/Vorlesung/vl11$
Timed list comprehension (seconds): 18.6455891132
Timed generator (seconds): 17.7329790592
ps -vp 3720
  PID TTY      STAT   TIME  MAJFL  TRS   DRS   RSS %MEM COMMAND
 3720 pts/1    Rl     0:08      0 3385 152566 84860  2.1 python timeit_li
ps-vp 3722
  PID TTY      STAT   TIME  MAJFL  TRS   DRS   RSS %MEM COMMAND
 3722 pts/1    Sl     0:10      0 3385 108694 41244  1.0 python timeit_ge
benzro@benzro-eMachines-E625:/media/benzro/OS/Users/benzro/Documents/LubuntuShared/8)
PCL-1/Vorlesung/vl11$
```

Im Task Manager:

Command	User	CPU%	RSS	VM-Size	PID	State	Prio	PPID
python	benzro	0%	85.3 MB	154.3 MB	3693	S	0	1
python	benzro	0%	40.3 MB	109.5 MB	3696	S	0	1
bash	benzro	0%	4.9 MB	7.5 MB	3703	S	0	10579
python2.7	benzro	0%	85.1 MB	154.3 MB	3718	S	0	10579
python	benzro	0%	85.0 MB	154.3 MB	3720	S	0	3703
python	benzro	0%	40.3 MB	109.5 MB	3722	S	0	3703

Zusammenfassung:

Die Listenkomprehension (ID3720) braucht 154 MB RAM und 18.6 Sekunden
Der Generator (ID3722) braucht 109.5 MB RAM und 17.3 Sekunden

```
Listenkomprehension: mit eckigen Klammern
return set([w.lower() for w in words])

Testgenerator: ohne eckige Klammern
return set(w.lower() for w in words)
```

Instanzen (isinstance(Object,Class))

Relation zwischen einem Objekt und seiner Klasse (Typ)!

```
>>> isinstance([1,2,3], list)
True
>>> isinstance([1,2,3], dict)
```

Unterklassen (issubclass(lowerclass, upperclass))

Relation zwischen 2 Klassen/Typen!

```
>>> issubclass(nltk.FreqDist, dict)
True
>>> issubclass(nltk.FreqDist, collections.Counter)
True
>>> issubclass(collections.Counter, dict)
True
```

```
>>> help(issubclass)
Help on built-in function issubclass in module __builtin__:

issubclass(...)
    issubclass(C, B) -> bool

    Return whether class C is a subclass (i.e., a derived class) of class B.
```

- Klassen spezifizieren und implementieren die Eigenschaften (=Attribute) und Funktionen (=Methoden) von Objekten.
- Klassen abstrahieren gemeinsame Eigenschaften und Funktionalitäten.
- Klassen sind in Unter-/Oberklassen (*superclass/subclass*) organisiert (Vererbung).
- Vererbung heisst, dass Eigenschaften/Methoden einer Oberklasse defaultmässig auch in der Unterklasse zur Verfügung stehen.
- Die Methoden können in der Unterklasse aber auch umdefiniert werden (Flexibilität).
- Jede selbstdefinierte Klasse muss eine Oberklasse haben.
- Eine selbstdefinierte Klasse kann auch mehrere Oberklassen haben, d.h. Mehrfachvererbung ist möglich.

Die Eigentümlichkeiten der obersten Klasse `object`

Die Oberklasse aller Klassen in Python heisst `object`.

Die Klasse `object` ist trotz ihres Namens eine Klasse!

```
>>> help(object)
Help on class object in module __builtin__:
class object
 |  The most base type
```

Objekte (Instanzen) der Klasse `object`

```
>>> o = object()
>>> type(o)
<type 'object'>
```

>>> iobj=object()>>> ty

Gibt es eine Oberklasse der Klasse `object`?

```
>>> issubclass(object, object)
True
```

Klassen definieren: Case-insensitive Strings

Motivation: Konsistenter Umgang mit Zeichenketten, wo Gross-/Kleinschreibung keine Rolle spielt.

Definition der Klasse, der Konstruktorfunktion und einer Methode

```
class Istr(object):              # Unterklasse von object
    """Case-insensitive string class"""
    def __init__(self, s):       # Konstruktor-Funktion
        self._is = s.lower()     # self ist Instanzparameter
                                 # _is ist Instanzvariable
    def endswith(self, s):       # Methode endswith(s)
        return self._is.endswith(s.lower())
```

Instantiierung eines Objekts und Methodenaufruf

```
s = Istr('ABC') # Konstruktion eines Objekt der Klasse Istr
s.endswith('c') # Methoden-Aufruf
```

Zusammenhang von Definition und Verwendung

Klassendefinition

```
class Istr(object):
```

Konstruktordefinition

```
def __init__(self,s):
 self._is = s.lower()
```

Methodendefinition

```
def find(self,s):
 ls = s.lower()
 return self._is.find(ls)
```

Objektinstantiierung

```
s = Istr('ABC')
```

Methodenaufruf

```
s.find('bC')
```

Instanzvariablen

Jede Objekt-Instanz kann Attribute mit individuellen Werten in sich tragen.

Normalerweise werden diese Instanzvariablen beim Konstruieren des Objekts mit einem Wert belegt.

Instanzvariablen

```
>>> s = Istr('ABC')
>>> print s._is
abc
>>> s2 = Istr('XYZ')
>>> print s2._is
xyz
```

Grade der Öffentlichkeit: Namenskonvention

- Öffentliche Instanzvariablen ohne Unterstrich am Anfang: Überall frei benutzbar!
- Private Instanzvariablen beginnen mit Unterstrich: Sollen nur innerhalb der Klassendefinition verwendet werden! Grund: Datenabstraktion: Interne Implementation kann ändern, ohne das Klassenbenutzung sich ändert muss

Instanzvariablen

Jede Objekt-Instanz kann Attribute mit individuellen Werten in sich tragen.

Normalerweise werden diese Instanzvariablen beim Konstruieren des Objekts mit einem Wert belegt.

Instanzvariablen

```
>>> s = Istr('ABC')
>>> print s._is
abc
>>> s2 = Istr('XYZ')
>>> print s2._is
xyz
```

Grade der Öffentlichkeit: Namenskonvention

- Öffentliche Instanzvariablen ohne Unterstrich am Anfang: Überall frei benutzbar!
- Private Instanzvariablen beginnen mit Unterstrich: Sollen nur innerhalb der Klassendefinition verwendet werden!
Grund: Datenabstraktion: Interne Implementation kann ändern, ohne das Klassenbenutzung sich ändern muss

Fazit Objektorientierte Programmierung (OOP)

Kernkonzepte nach http://en.wikipedia.org/wiki/Object-oriented_programming

- Datenkapselung I: Bündeln von Datenstrukturen und zugehöriger Funktionalität unter einer Adresse (=Objekt)
- Datenkapselung (Abstraktion) II: Klare Schnittstelle, welche Attribute und Methoden für öffentliche und welche für private (objektinterne) Zwecke nutzbar sind
- Klassenzugehörigkeit: Objekte sind Instanzen einer Klasse
- Vererbung: Unterklassen können Attribute/Methoden von ihren Oberklassen erben
- Dynamische Bindung: Welche Methode (d.h. Methode von welcher (Ober-)klasse) ein Objekt benutzt, wird erst beim Aufruf der Methode festgelegt anhand der *method resolution order*.
- Selbst-Parameter (*self*): Platzhalter für das Instanzobjekt in der Definition einer Klasse

>>> class iclass(object):... def __init__(self,s):

>>> class iclass3(type):... def __init__(self,s):...

Namen, Zuweisung, Bindung und Objekte

Zuweisung (*Assignment*)

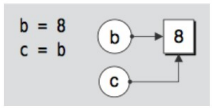
```
a = 5*8
```

Was passiert beim Verarbeiten der Zuweisungsanweisung?

1. Evaluiere (*evaluate*) RHS-Ausdruck `5*8` zu einem Ganzzahl-Objekt. (RHS=*right-hand side*)
2. Binde (*binding*) das evaluierte Ganzzahl-Objekt an den Namen a.

Namen referieren auf Objekte

1. Rechteck = Objekte
2. Kreis = Referenz auf Objekt



Quelle: [Summerfield 2008, 14]

Was passiert beim Statement `c = b`?

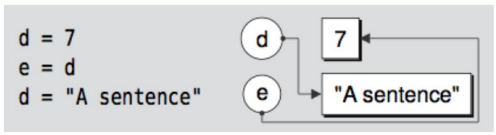
Zuweisungsverhalten bei einfachen Datentypen

b erhält die Referenz aufs Objekt 8
```
>>> b=8
```
c erhält die Referenz aufs Objekt 8
```
>>> c=b
```
b erhält die Referenz aufs Objekt 7
```
>>> b=7
```
c behält die Referenz aufs Objekt 8
```
>>> c
8
```

versus Zuweisungsverhalten von Referenzdatentypen

Mehrfaches Binden eines Namens (*rebinding*)

Was passiert, wenn derselbe Namen mehrfach zugewiesen wird?



Unreferenzierte Objekte und Müllsammlung

```
s = "Ein String"              # s macht "Ein String" zugänglich
                              # im nachfolgenden Programm.
s += " wird zusammengesetzt!" # Nach dieser Anweisung ist
                              # "Ein String" nicht mehr
                              # zugänglich via Name s.
print s                       # s referenziert ein neues Objekt.
Ein String wird zusammengesetzt!
```

- Unbenannte Objekte sind ausserhalb des Ausdrucks, in dem sie vorkommen, nicht mehr zugänglich und benutzbar: Sie sind Datenmüll (*garbage*).
- Nicht mehr zugängliche Objekte können periodisch gesammelt und entsorgt werden (*garbage collection*). Dadurch wird Speicherplatz frei.

Python weiss für jedes Objekt, wie viele Referenzen (Namen) darauf existieren. Das Modul `gc` ist eine Schnittstelle zur *garbage collection*.

Unreferenzierte Objekte und Müllsammlung

```
s = "Ein String"              # s macht "Ein String" zugänglich
                              # im nachfolgenden Programm.
s += " wird zusammengesetzt!" # Nach dieser Anweisung ist
                              # "Ein String" nicht mehr
                              # zugänglich via Name s.
print s                       # s referenziert ein neues Objekt.
Ein String wird zusammengesetzt!
```

- Unbenannte Objekte sind ausserhalb des Ausdrucks, in dem sie vorkommen, nicht mehr zugänglich und benutzbar: Sie sind Datenmüll (*garbage*).
- Nicht mehr zugängliche Objekte können periodisch gesammelt und entsorgt werden (*garbage collection*). Dadurch wird Speicherplatz frei.

Python weiss für jedes Objekt, wie viele Referenzen (Namen) darauf existieren. Das Modul `gc` ist eine Schnittstelle zur *garbage collection*.

Identität eines Objekts (`id()`) und *mutable data*

Identität bei veränderlichen Datenstrukturen

Verschiedene Namen können auf dasselbe Objekt referenzieren. Die eingebaute Funktion `id()` identifiziert jedes Objekt über eine Ganzzahl (entspricht ungefähr seiner Speicheradresse). Python garantiert, dass 2 verschiedene Objekte gleichzeitig nie dieselbe ID haben.

(Re-)Binding einer Variable

```
>>> l = ['a']
>>> id(l)
4300400112
>>> l = ['a']
>>> id(l)
4299634664
```

Weshalb?

Veränderliche Datenstrukturen

```
>>> l = ['a']
>>> id(l)
4300400112
>>> l[0] = 'b'
>>> id(l)
4300400112
```

Weshalb?

Binding in *for*-Konstrukten

Identität vs. Wertgleichheit (*equality*, Äquivalenz)

- `o1 == o2` testet, ob 2 Objekte/Variablen denselben Wert haben
- `o1 is o2` testet, ob 2 Objekte/Variablen dieselbe Identität haben, d.h. identisch sind, d.h. `id(o1) == id(o2)`

Was wird hier ausgegeben? ▸2

```
>>> l = [('der',1200),('die',1000),('das',900)]
>>> for i,e in enumerate(l):
        print e is l[i]
True
True
True
```

In for-Konstrukten werden bestehende Objekte an neue Namen gebunden!

Kopieren von Listen

Eine spannende Verbindung: Binding und Listen ▸4

Wie spielen Zuweisung von Listen-Namen und Veränderbarkeit zusammen?

Kopieren oder Binding?

```
l1 = [('der',1200),('die',1000)]
# Binding
l2 = l1
# Kopieren via Slicing
l3 = l1[:]
# Welche Listen werden modifiziert?
l1[0] = ('der',1201)
```

Kopieren via allgemeinem Modul zum Kopieren von Objekten

```
import copy
l4 = copy.copy(l1)
```

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
l1 = [('der',1200),('die',1000)]
l2=[]
for i,e in enumerate(l1):
    print "\nfrom 1: ", i #iterator
    print "\nfrom 2: ",e #element
    #l2[i]=(i,e) #funktioniert nicht, index out of range
    l2.append((i,e))
    print "\nfrom 3: ",l1[i]
    print "\nfrom 4: ",e is l1[i]
print "\nfrom 5: ",enumerate(l1)
print "\nfrom 6: ",l2
help(enumerate)
# class enumerate(object)
#  |  enumerate(iterable[, start]) ->
#         iterator for index, value of iterable|
#  |  enumerate is useful for obtaining an indexed list:
#  |      (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
```

```
/LubuntuShared/8) PCL-1/Vorlesung/vl11/snippets.py"

from 1:  0

from 2:  ('der', 1200)

from 3:  ('der', 1200)

from 4:  True

from 1:  1

from 2:  ('die', 1000)

from 3:  ('die', 1000)

from 4:  True

from 5:  <enumerate object at 0xb71dce3c>

from 6:  [(0, ('der', 1200)), (1, ('die', 1000))]
```

```python
global_list = [('der',1200),('die',1000),('das',900)]
print "from 1: ", global_list
def del_first(l):
    print "from 2: ",
    print 'global_list is parameter l:', global_list is l
    del l[0]
del_first(global_list)
print "from 3: ",global_list
```

```
from 1:  [('der', 1200), ('die', 1000), ('das', 900)]
from 2:  global_list is parameter l: True
from 3:  [('die', 1000), ('das', 900)]
```

```python
l1 = [('der',1200),('die',1000)]
print "from 1: ", l1
l2 = l1
print "from 2: ", l2
# Kopieren via Slicing
l3 = l1[:]
print "from 3: ", l3
# Kopieren via copy modul
import copy
l4 = copy.copy(l1)
print "from 4: ", l4
# Welche Listen werden modifiziert?
l1[0] = ('der',999999999)
print "from 5.1: ", l1
print "from 5.2: ", l2
print "from 5.3: ", l3
print "from 5.4: ", l4
print "\n",40*"=","\n"
```

```
from 1:  [('der', 1200), ('die', 1000)]
from 2:  [('der', 1200), ('die', 1000)]
from 3:  [('der', 1200), ('die', 1000)]
from 4:  [('der', 1200), ('die', 1000)]
from 5.1:  [('der', 999999999), ('die', 1000)]
from 5.2:  [('der', 999999999), ('die', 1000)]
from 5.3:  [('der', 1200), ('die', 1000)]
from 5.4:  [('der', 1200), ('die', 1000)]

========================================
```

Sortieren und maximieren

Ordnung erzeugen bei Dictionaries

- `min()`, `max()`, `in`, `sorted()` etc. operieren über Schlüsseln.
- `dict.values()` ist Liste aller Werte.
- Höchster Schlüssel: `max(d)`
- Höchster Wert: `max(d.values())`
- Schlüssel mit höchstem Wert: `max(d,key=d.get)`
- Nach Schlüssel sortieren: `sorted(d)`
- Nach Werten sortieren: `sorted(d,key=d.get)`
- Umgekehrt nach Werten sortieren: `sorted(d, key=d.get, reverse=True)`

Dict mit (Key:Value) pairs

```
>>> d={"neun":9,"fünf":5, "a":2,"z":99,"gg":"h"}
```

Grösster Key

```
>>> max(d.keys())
'z'
>>> max(d)
'z'
```

Grösster Value

```
>>> max(d.values())
'h'
```

Key des grössten Values

```
>>> max(d.keys(), key=d.get)
'gg'
>>> max(d,key=d.get)
'gg'
```

Value des grössten Keys

```
>>> d[max(d.keys())]
99
```

Undefinierbar??

```
>>> max(d.values(),key=d.get)
2
```

Motivation

key word in context

Ziel: KWIC in Python

Eine Klasse programmieren, welche eine Konkordanz über einem gestemmten Index anzeigt.

Beispiel-Output

```
>>> text.concordance('die', width=30)
 BLACK KNIGHT : Then you shall die . ARTHUR : I command you
Camelot . He was not afraid to die , O brave Sir Robin .
 Concorde , you shall not have died in vain ! CONCORDE : Uh
              2 : Oh , he ' s died ! FATHER : And I want
 that ? MAYNARD : He must have died while carving it . LAUNCE
     ARTHUR : Look , if he was dying , he wouldn ' t bother
```

Beispiel: Konkordanzprogramm über gestemmten Wörtern

KWIC als Klasse: Datenstrukturen und Funktionalitäten

- Text: Folge von Wörtern
- Index: Abbildung von (gestemmtem) Wort zu allen Vorkommenspositionen im Text
- Stemmer: Stemming von Wortformen
- KWIC-Anzeige: Formatierung der Treffer im KWIC-Stil

Benötigte Kompetenzen

- Wie lassen sich (einfache) Klassen definieren?
- Definition eigener Regex-Stemmer oder Benutzung von NLTK-Stemmern
- Formatierung von zentriertem textuellem Output mit Format-Ausdrücken

```
# Ziehe eine zufällige Auswahl von Vorkommen
# von Token aus einem Korpus
#=========================================
import nltk, random
corpus = nltk.corpus.nps_chat.words()
#xrange_=0,1,2,3...len(corpus)
xrange_=xrange(len(corpus))
print "\nfrom 1: ", xrange_, len(corpus)
#ziehe Anzahl (anz) Zufallszahlen aus xrange_
anz=20
randsamp=random.sample(xrange_,anz)
print "\nfrom 2: ", randsamp
#printe die Zufallswörter
for i in randsamp:
    print "\nfrom 3: ", corpus[i]
# as a reusable function with a generator return value
def sample_corpus1(text,size):
    return (text[i] for i in
            random.sample(xrange(len(text)),size))
# as a reusable function with a list return value
def sample_corpus2(text,size):
    return [text[i] for i in
            random.sample(xrange(len(text)),size)]
print "\nfrom 4: ", sample_corpus2(corpus,20)
```

random_sample_xrange.py"

from 1: xrange(45010) 45010

from 2: [18928, 29965, 26596, 33993, 21320, 11471, 35897, 33148, 3189, 2246, 22251, 43138, 38653, 20100, 4934, 3503, 13063, 1062, 18675, 38180]

from 3: adds
from 3: o
from 3: one
gekürzt…

from 4: [u'Pat', u'thats', u'..', u'U121', u'.....', u'....', u'for', u':)', u',', u'pfft', u'she', u'didnt', u'the', u'missed', u'.', u'welome', u'U35', u'song', u'welcome', u'i']

Process finished with exit code 0

```
# Example from NLTK book p. 107-108
import nltk, re
class RegexStemmer(object):
    #ObjektInitialisierung
    def __init__(self, r=
            r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'):
        #Eigener Regex zur Wortstammbildung
        self._r = r
        print "\nfrom 0.0: ", type(self._r), self._r
    #Hier wird der WortStamm bestimmt
    def stem(self,word):
        m = re.match(self._r, word)
        #Stamm ^(.*?)
        try:
            return m.group(1)
        except:
            return "" #falls keine Gruppe 1 besteht
class IndexedText(object):
    #ObjektInitialisierung
    def __init__(self, stemmer, text):
        #Text ist Wortliste
        self._text = text
        print "\nfrom 0.1: ", type(self._text),self._text[:10]
        #Stemmer zur Stammbildung
        self._stemmer = stemmer
        print "\nfrom 0.2: ", type(self._stemmer),self._stemmer
        #Textindex mit WortStamm, dict, {WortStamm, AlleIndices}
        self._index = nltk.Index((self._stem(word), i)
                                 for (i, word) in enumerate(text))
        print "\nfrom 0.3: ", type(self._index)
        print "\nfrom 0.3: is dict? ",issubclass(nltk.Index,dict)
    #Stamm in Kleinbuchstaben
    def _stem(self, word):
        return self._stemmer.stem(word).lower()
    #Konkordanzprint
    def concordance(self, word, width=20):
        #Aufruf der Methode stem
        key = self._stem(word)      # stemmed keyword
        #width: chars vor und nach word
        wc = width/4                # words of context
        #Iteriere über AlleIndices des key=Wortstamm
        for i in self._index[key]:
            #linker Kontext ohne word
            lcontext = ' '.join(self._text[i-wc:i])
```

stemmed_kwic.py"

from 1.1:
from 1.2:
from 0.1: <class 'nltk.corpus.reader.util.StreamBackedCorpusView'> [u'SCENE', u'1', u':', u'[', u'wind', u']', u'[', u'clop', u'clop', u'clop']
from 0.2: <class 'nltk.stem.porter.PorterStemmer'> <PorterStemmer>
from 0.3: <class 'nltk.util.Index'>
from 0.3: is dict? True

from 1.3:
KWIC mit Porter-Stemmer
Defeat at the castle seems to have utterl
afraid our life must seem very dull and q
PIGLET : Well , what seems to be the trou
. You know , it seemed a bit daft me
, who , when he seemed about to reco
thur and his knights seemed hopeless , wh

==========

from 2.1:
from 0.0: <type 'str'> ^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$

from 2.test: seem

from 2.2:
from 0.1: <class 'nltk.corpus.reader.util.StreamBackedCorpusView'> [u'SCENE', u'1', u':', u'[', u'wind', u']', u'[', u'clop', u'clop', u'clop']
from 0.2: <class '__main__.RegexStemmer'> <__main__.RegexStemmer object at 0xb1b8b5ac>
from 0.3: <class 'nltk.util.Index'>
from 0.3: is dict? True

from 2.3:
KWIC mit simplem Regex-Stemmer
Defeat at the castle seems to have utterl
afraid our life must seem very dull and q
PIGLET : Well , what seems to be the trou
. You know , it seemed a bit daft me
, who , when he seemed about to reco
thur and his knights seemed hopeless , wh

==========

from 3.0:
GHT : Then you shall die . ARTHUR : I
He was not afraid to die , O brave Sir

```
            #word (index i) und rechter Kontext
            rcontext = ' '.join(self._text[i:i+wc])
            ldisplay = '%*s'  % (width, lcontext[-width:])
            rdisplay = '%-*s' % (width, rcontext[:width])
            print ldisplay, rdisplay
#Text laden
text = nltk.corpus.webtext.words('grail.txt')
#Neues nltk stemmer objekt
#=========================
print "\nfrom 1.1:"
porter_stemmer = nltk.PorterStemmer()
#Neues Text Objekt, mit nltk Stemmer und Index
print "\nfrom 1.2:"
porter_index = IndexedText(porter_stemmer, text)
#Konkordanz von gestemmtem Wort seem
print "\nfrom 1.3: ","\nKWIC mit Porter-Stemmer\n"
porter_index.concordance('seem')
print "\n",40*"="  ,"\n"
#Neues Stemmer Objekt mit eigener Regex
#======================================
print "\nfrom 2.1:"
regex_stemmer = RegexStemmer()
#Test: Stemmer anwenden auf seeming
print "\nfrom 2.test: ", regex_stemmer.stem('seeming')
#Neues Text Objekt mit eigenem Stemmer und Index
print "\nfrom 2.2:"
regex_index = IndexedText(regex_stemmer, text)
#Konkordanz von gestemmten Wort seem
print "\nfrom 2.3: ","\nKWIC mit simplem Regex-Stemmer\n"
regex_index.concordance('seem')
print "\n",40*"="  ,"\n"
## Wie gut ist unser simpler Stemmer?
#===================================
print "\nfrom 3.0:"
porter_index.concordance('dies')
print "\nfrom 3.1:"
regex_index.concordance('dies')
print "\n",40*"="  ,"\n"
print "\nfrom 3.test: ",regex_stemmer.stem('dies')
print "\nfrom 3.test: ",regex_stemmer.stem('dying')
print "\nfrom 3.test: ",regex_stemmer.stem('is')
print "\n",40*"="  ,"\n"
## Wie gut ist unser better Stemmer?
#==================================
regex = r'^(.{2,}?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
print "\nfrom 3.2: ", regex
better_regex_index = IndexedText(RegexStemmer(r=regex), text)
print "\nfrom 3.3: ","\nKWIC mit better Regex-Stemmer\n"
better_regex_index.concordance('dies')
print "\n",40*"="  ,"\n"
```

```
, you shall not have died in vain ! CONCO
       Oh , he ' s died ! FATHER : And
YNARD : He must have died while carving i
   Look , if he was dying , he wouldn '

from 3.1:
      # 1 : Where ' d you get the coconu
      : No , they ' d have to have it
  , simple ! They ' d just use a strand
very nice . And how d ' you get that
    here . Oh ! How d ' you do ?
     at me , they ' d put me away !
      of that . I ' d rather -- FATHER :
   ?! HERBERT : I ' d rather ... [ music
   Lucky , so you ' d better get used to
 ARTHUR : Then we ' d best leave them he
   Three . And we ' d better not risk an
  ' aaggggh '. He ' d just say it !

========================================


from 3.test:  d
from 3.test:  dy
from 3.test:  i

========================================



from 3.2:  ^(.{2,}?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$
from 0.0:  <type 'str'> ^(.{2,}?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$
from 0.1:  <class 'nltk.corpus.reader.util.StreamBackedCorpusView'> [u'SCENE', u'1', u':', u'[',
u'wind', u']', u'[', u'clop', u'clop', u'clop']
from 0.2:  <class '__main__.RegexStemmer'> <__main__.RegexStemmer object at 0xb1b96eac>
from 0.3:  <class 'nltk.util.Index'>
from 0.3: is dict?  True

from 3.3:
KWIC mit better Regex-Stemmer
, you shall not have died in vain ! CONCO
       Oh , he ' s died ! FATHER : And
  carry on on foot . Dis - mount ! TIM
YNARD : He must have died while carving i

========================================

Process finished with exit code 0
```

Formatierung mit Hilfe von Format-Ausdrücken

- Flexiblere Kontrolle für Ausgabe von Zahlen und Strings ist erwünscht
- Formatierungsausdruck: `'STRINGTEMPLATE WITH FORMATS' % TUPLE`
- Ein Formatierungsausdruck (*string formatting expression*) trennt Layout (Platzhalter `%d`, `%f` für Zahlen, `%s` für Strings) von den variablen Daten (Tupel)
- Anzahl Nachkommastellen (`'%.2f'`), Padding mit Leerzeichen (`'% 4.2f'`), linksbündig (`'%-7s'`), rechtsbündig(`'%7s'`)

```
>>> 'a string:%s and an integer:% 4d' % ('abc',3)
'a string:abc and an integer:   3'

>>> 'Padding a string:%-6s and a float:% 8.2f' % ('abc',3.175)
'Padding a string:abc    and a float:    3.17'
```

Formatierungsausdrücke

Überraschung ▸1

```
>>> '%.1f' % 0.05
'0.1'
>>> '%.1f' % 0.15
'0.1'
>>> round(0.05,1)
0.1
>>> round(0.15,1)
0.1
```

Schulregel mit aufzurundendem .5 verzerrt systematisch (*bias*) ▸2

Prozentzeichen schützen mit %

```
>>> '%.1f%%' % 0.15
'0.1%'
```

Variables Padding mit *

```
>>> width = 8
>>> '%*s' % (width, 'abc')
'     abc'
>>> '%-*s' % (width, 'abc')
'abc     '
```

Regex-Stemmer-Klasse definieren

Klasse mit optionalem Konstruktor-Argument ▸3

```
class RegexStemmer(object):
    def __init__(self, r=r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'):
        self._r = r

    def stem(self,word):
        m = re.match(self._r, word)
        return m.group(1)
```

Initialisierung und Verwendung

```
regex_stemmer = RegexStemmer()
regex_stemmer.stem('seeming')
```

Textindex als Klasse `IndexedText` definieren

```
class IndexedText(object):
    def __init__(self, stemmer, text):
        self._text = text
        self._stemmer = stemmer
        self._index = nltk.Index((self._stem(word), i)
                                 for (i, word) in enumerate(text))
```

`enumerate(s)` generiert Paare `(Position,Element)` aus Sequenz

```
l = ['wenn', 'fliegen', 'hinter', 'fliegen', 'fliegen']
>>> list(enumerate(l))
[(0, 'wenn'), (1, 'fliegen'), (2, 'hinter'), (3, 'fliegen'), (4, 'fliegen')]
```

`nltk.Index(Pairs)` erzeugt invertierten Index aus `(Element,Position)`

```
>>> index = nltk.Index((w,i) for (i,w) in enumerate(l))
>>> index['fliegen']
[1, 3, 4]
```

Private und öffentliche Methode von `IndexedText`

Unterstrich markiert Privatheit: Nur für Benutzung **in** der Klasse

```
def _stem(self, word):
    return self._stemmer.stem(word).lower()
```

Öffentliche Methode für Formatierung

```
def concordance(self, word, width=40):
    key = self._stem(word)    # stemmed keyword
    wc = width/4              # words of context
    for i in self._index[key]:
        lcontext = ' '.join(self._text[i-wc:i])
        rcontext = ' '.join(self._text[i:i+wc])
        ldisplay = '%*s' % (width, lcontext[-width:])
        rdisplay = '%-*s' % (width, rcontext[:width])
        print ldisplay, rdisplay
```

Noch privater sind Attribute der Form `o.__NAME`.

Generatorausdrücke (*generator expressions*)

Listenkomprehension: Prinzip "Liste aller Dinge, die . . . "

Baue die Liste aller kleingeschriebenen Wörter aus dem Brown-Korpus und erzeuge danach aus der Liste eine Menge!

```
set([w.lower() for w in nltk.corpus.brown.words()])
```

Generatorausdrücke: Prinzip "Der Nächste, bitte!"

Nimm ein kleingeschriebenes Wort nach dem andern und mache es zum Element der Menge!

```
set(w.lower() for w in nltk.corpus.brown.words())
```

Listenkomprehension vs. Generatorausdrücke

Generatorausdrücke statt Listenkomprehension

Im NLTK-Buch wird aus Effizienzgründen `set(w.lower() for w in text)` statt `set([w.lower() for w in text])` notiert.

- Listenkomprehension erzeugt im Arbeitsspeicher immer eine Liste aller Elemente.
- Generatorausdrücke sind speichereffizient. Sie übergeben ihre Element auf Verlangen einzeln der auswertenden Funktion (intern `g.next()`).
- Generatorausdrücke unterstützten darum Längenmethode `len()` nicht.
- Generatorausdrücke unterstützten kein Slicing: `l[:10]`.
- Mit `list(generator)` wird jeder Generator zur Liste.
- Speichereffizienz ist bei allen Funktionen optimiert, welche Daten vom Typ *iterable* verarbeiten: `max()`, `sum()`, `set()` usw.
- Generatoren sind nach 1 Durchgang erschöpft, d.h. aufgebraucht!

Generatorausdrücke und die Iteratorfunktion `next()`▸4

```
>>> quadrat = (i*i for i in [10,11])
>>> quadrat
<generator object <genexpr> at 0x16a6f80>
>>> type(quadrat)
<type 'generator'>
>>> quadrat.next()
100
>>> quadrat.next()
121
>>> quadrat.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Die Ausnahme (*exception*) `StopIteration` erscheint, wenn der Generator erschöpft ist. Konsumenten von Generatoren müssen Ausnahme korrekt behandeln.

Generatorfunktionen mit `yield`

```
def quadriere(iterierbar):
    for i in iterierbar:
        yield i*i

quadrat = quadriere([10,11])
>>> quadrat
<generator object quadriere at 0x103945500>
>>> type(quadrat)
<type 'generator'>
>>> quadrat.next()
100
>>> sum(quadrat)
121
>>> sum(quadriere([10,11]))
221
```

Rechenzeit und Speicherverbrauch messen

Programm mit Generatorausdrücken

```
import nltk, timeit, time, os
words = nltk.corpus.brown.words()

def test_generator():
    return set(w.lower() for w in words)

# Initialisiere Timer-Objekt
tg = timeit.Timer(test_generator)

# Timing von Generatorausdruck
print 'Timed generator (seconds):', tg.timeit(1)
```

Der Speicherverbrauch muss extern gemessen werden.

Rechenzeit und Speicherverbrauch messen

Programm mit Listencomprehension

```
import nltk, timeit, time, os
words = nltk.corpus.brown.words()

def test_listcomprehension():
    return set([w.lower() for w in words])

# Initialisiere Timer-Objekt
tl = timeit.Timer(test_listcomprehension)

# Timing von Listenkomprehension
print 'Timed list comprehension (seconds):', tl.timeit(1)
```

Der Speicherverbrauch muss extern gemessen werden.

Effizienz in Rechenzeit

Zufällige Auswahl von Elementen aus einem Bereich

```
# Modul zur Zeitmessung von Python-Statements
import timeit
setup = 'import random'

# Konstruiere 2 Timer-Objekte
tr = timeit.Timer('random.sample( range(1000000),100)', setup)
tx = timeit.Timer('random.sample(xrange(1000000),100)', setup)

# Führe Timings je einmal durch und speichere Anzahl Sekunden
trsecs = tr.timeit(1)
txsecs = tx.timeit(1)

print "Aufgabe: Sample 100 Zahlen aus dem Bereich 0 bis 999999."
print "Zeit mit xrange:", txsecs, "Sekunden"
print "Zeit mit range:", trsecs, "Sekunden"
print "xrange ist etwa", trsecs/txsecs, "Mal schneller!"
```

Zufälliges Auswählen von Wörtern

Das Ziehen einer zufälligen Stichprobe (*sample*) aus einem Korpus. ▸9

```
import nltk, random

corpus = nltk.corpus.nps_chat.words()

# for demonstration
for i in random.sample(xrange(len(corpus)),20):
    print corpus[i]
# as a reusable function with a generator return value
def sample_corpus1(text,size):
    return (text[i] for i in random.sample(xrange(len(text)),size))

# as a reusable function with a list return value
def sample_corpus2(text,size):
    return [text[i] for i in random.sample(xrange(len(text)),size)]
```

Häufige Exceptions

Ausnahmen können in jeder Stufe der Programmausführung auftreten!

SyntaxError

```
print 1 2
```

KeyError

```
a = {}
a["test"]
```

NameError

```
print a
```

RuntimeError

```
def x(): return x()
x()
```

ZeroDivisionError

```
1 / 0
```

TypeError

```
sum(["1", "2", "3"])
```

IndexError

```
a = [1, 2, 3]
a[3]
```

Wie gehe ich mit Fehlern um?

```
x = raw_input()
```

Robuste Programmierung

- Wir wollen x in eine Zahl umwandeln, bei ungültiger Eingabe eine neue Eingabe verlangen.
- `float(x)` führt zu Programmterminierung
- `x.isdigit()` akzeptiert nur Teilmenge aller Zahlen
- Verkettung von Regeln möglich, aber umständlich

Ausnahmen (Exceptions)

- Ausnahmen können im Programm abgefangen werden, anstatt dass sie zur Terminierung führen.
- Oft eleganter, als Ausnahmen zu vermeiden.

Ausnahmen auffangen: *try*-Konstrukt

Syntax-Schema

```
try:
   block1
except E:
   block2
```

Syntax-Schema mit finally

```
try:
   block1
except E:
   block2
finally:
   block3
```

Erklärung

- Führe `block1` aus.
- Wenn währenddessen eine Ausnahme vom Typ `E` auftritt, führe `block2` aus
- Führe `block3` auf jeden Fall am Schluss aus.

Ausnahmen sind ebenfalls Objekte und haben infolgedessen einen Typ!

Ausnahmen ignorieren ▸10

```
while True:
  x = raw_input('Please type in a number: ')
  try:
    float(x)
    break
  except ValueError:
    pass
```

Leere Blöcke

- Blöcke müssen immer mindestens eine Anweisung enthalten
- pass für leere Blöcke (*no operation*, no(o)p)

Philosophien der Fehlerbehandlung: LBYL vs EAFP

LBYL

```
if w in freqs:
  freqs[w] += 1
else:
  freqs[w] = 1
```

Look before you leap.

EAFP

```
try:
  freqs[w] += 1
except KeyError:
  freqs[w] = 1
```

It's easier to ask for forgiveness than for permission.

Fehler auffangen: Wie spezifisch?

```
try:
  ...
  (ganz viel Code)
  ...
except:
  pass
```

Welche Exceptions soll man abfangen?

- Zu allgemeine except-Klauseln erschweren das Bemerken und Finden von Programmierfehlern.
- Setze try/except-Klauseln gezielt ein.
- Bestimme den Ausnahmentyp, der abgefangen werden soll.

with-Konstrukt für Datei-Handling ▸11

- Das Betriebssystem erlaubt nicht, dass Hunderte von Dateien von einem Prozess geöffnet sind.
- Bei Prozessen, welche vielen Dateien lesen/schreiben, müssen die Dateien geschlossen werden.
- Das with-Konstrukt mit Datei-Objekten macht dies automatisch (was auch immer für Ausnahmesituationen beim Dateiverarbeiten entstehen).

```
filename = "with_open.py"

with open(filename,'r') as f:
  for l in f:
    if l.rstrip() != '':
      sys.stdout.write(l)
```

with-Konstrukt für Datei-Handling ▸11

- Das Betriebssystem erlaubt nicht, dass Hunderte von Dateien von einem Prozess geöffnet sind.
- Bei Prozessen, welche vielen Dateien lesen/schreiben, müssen die Dateien geschlossen werden.
- Das with-Konstrukt mit Datei-Objekten macht dies automatisch (was auch immer für Ausnahmesituationen beim Dateiverarbeiten entstehen).

```
filename = "with_open.py"

with open(filename,'r') as f:
  for l in f:
    if l.rstrip() != '':
      sys.stdout.write(l)
```