Encoding Text
Regular Expressions
Basic Text Processing

What will you learn

- Encoding text in computer
  - ASCII, UTF-8, etc
- Regular expressions
  - E.g.: [ab]xxx(.*)yyy
- Tokenization
  - How to split text into *words*?
- Sentence segmentation
  - How to split text into *sentences*?

Encoding text in computer

Writing systems

- Different types of writing systems
  - Alphabetic (English, Estonian, Russian)
  - Syllabic: symbols represent *syllables* (Japanese *kana*, Cherokee)
  - Logographic: single symbols represent complete words (Japanese kanji, Chinese, Ancient Egyptian), but also symbols such as $, %, €

ア 阿	イ 伊	ウ 宇	エ 江	オ 於
カ 加	キ 機	ク 久	ケ 介	コ 己
サ 散	シ 之	ス 須	セ 世	ソ 曽
タ 多	チ 千	ツ 川	テ 天	ト 止
ナ 奈	ニ 仁	ヌ 奴	ネ 祢	ノ 乃
ハ 八	ヒ 比	フ 不	ヘ 部	ホ 保
マ 末	ミ 三	ム 牟	メ 女	モ 毛
ヤ 也		ユ 由		ヨ 與
ラ 良	リ 利	ル 流	レ 礼	ロ 呂
ワ 和	ヰ 井		ヱ 恵	ヲ 乎
ン 尓				

日

Alphabetic symbols

- Alphabets (phonemic alphabets)
  - Represent all sounds (consonants and vowels)
- Abjads (consonant alphabets)
  - Represent consonants only (in some cases also some vowels)
  - Examples: Arabic, Aramaic, Hebrew
  - Arabic alphabet is used with many other languages, e.g. Kurdish, Pashto, Kazakh (in some areas)

العربية

Alphabet example

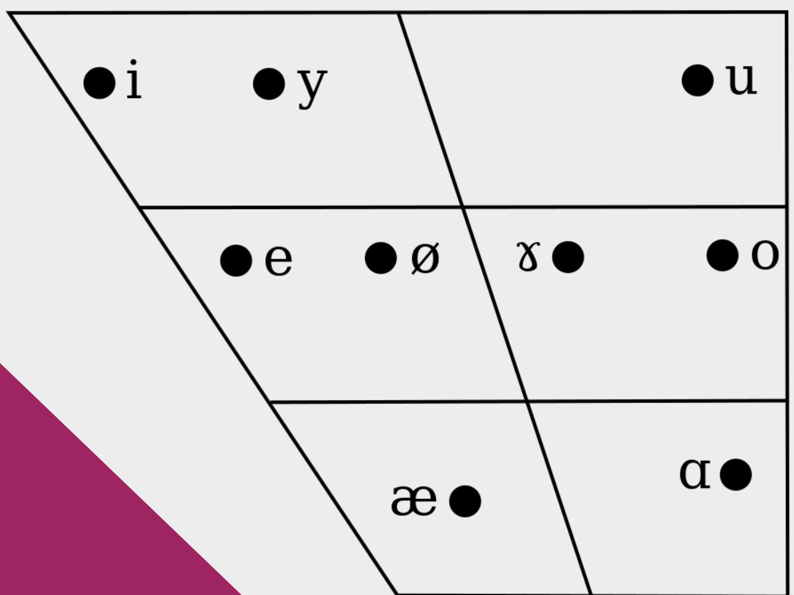
- Fraser alphabet, used to write *Lisu*, a Tibeto-Burman language spoken by about 700 000 people in South-East Asia

ꓐ	[b]	ꓑ	[p]	ꓒ	$[p^h]$	ꓓ	[d]	ꓔ	[t]	ꓕ	$[t^h]$
ꓖ	[g]	ꓗ	[k]	ꓘ	$[k^h]$	ꓙ	[dʑ]	ꓚ	[tɕ]	ꓛ	$[tɕ^h]$
ꓜ	[dz]	ꓝ	[ts]	ꓞ	$[ts^h]$	ꓟ	[m]	ꓠ	[n]	ꓡ	[l]
ꓢ	[s]	ꓣ	[ʒ]	ꓤ	[z]	ꓥ	[ŋ]	ꓦ	[h]	ꓧ	[x]
ꓨ	[ɦ]	ꓩ	[f]	ꓪ	[w]	ꓫ	[ɕ]	ꓬ	[ʑ]	ꓭ	[ɣɑ]

ꓮ	[ɑ]	ꓯ	[ɛ]	ꓰ	[e]	ꓱ	[ø]	ꓲ	[i]
ꓳ	[o]	ꓴ	[u]	ꓵ	[y]	ꓶ	[ɯ]	ꓷ	[ə]

International Phonetic Alphabet (IPA)

- Special alphabet for representing speech sounds (phonemes)
- See http://web.uvic.ca/ling/resources/ipa/charts/IPAlab/IPAlab.htm
- Estonian vowels and consonants:



		Labial	Alveolar		Postalveolar	Dorsal	Glottal
			plain	palatalized			
Nasal		m	n	nʲ			
Plosive	short	p	t	tʲ		k	
	geminated	pː	tː	tʲː		kː	
Fricative	voiceless short	f	s	sʲ	ʃ		h
	voiced short	v					
	geminated	fː	sː	sʲː	ʃː		
Approximant			l	lʲ		j	
Trill			r				

Encoding written language in computer

- Using a single byte (8 bits), one can represent 256 different characters
- Earliest character encoding: ASCII
  - Uses only 7 bits = 128 possible characters
  - English letters, numbers, punctuation marks
  - First 31 codes reserved for control characters (backspace, line feed, etc)

0 NUL	10 DLE	20	30 0	40 @	50 P	60 `	70 p
1 SOH	11 DC1	21 !	31 1	41 A	51 Q	61 a	71 q
2 STX	12 DC2	22 "	32 2	42 B	52 R	62 b	72 r
3 ETX	13 DC3	23 #	33 3	43 C	53 S	63 c	73 s
4 EOT	14 DC4	24 $	34 4	44 D	54 T	64 d	74 t
5 ENQ	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u
6 ACK	16 SYN	26 &	36 6	46 F	56 V	66 f	76 v
7 BEL	17 ETB	27 '	37 7	47 G	57 W	67 g	77 w
8 BS	18 CAN	28 (	38 8	48 H	58 X	68 h	78 x
9 HT	19 EM	29)	39 9	49 I	59 Y	69 i	79 y
A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z
B VT	1B ESC	2B +	3B ;	4B K	5B [	6B k	7B {
C FF	1C FS	2C ,	3C <	4C L	5C \	6C l	7C |
D CR	1D GS	2D -	3D =	4D M	5D]	6D m	7D }
E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~
F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL

Different coding systems

- ASCII contains only English letters
- What about other languages and characters, such as *õ, š, ш, 见*?
- ASCII was extended to 8-bit, to include extra characters
- However, this gives only 128 extra symbols
- Result: different encodings for different languages:
  - ISO-8859-1: includes extra letters needed for French, German, Spanish
  - ISO-8859-7: Greek alphabet
  - ISO-8859-8: Hebrew alphabet
  - ISO-8859-13: Baltic languages (including Estonian)

Problem with character sets

- Conflicts
  - Two different encodings can use the same number for different characters (e.g. ***š*** in ISO-8859-13 is encodes as 240, which in ISO-8859-1 corresponds to **ð**)
  - Sometimes, the same character corresponds to different codes in different character sets
- Also, a text file using a language-specific encoding (e.g. ISO-8859-13) cannot mix in letters from other encodings (e.g. Greek or Chinese)
- Also, when opening a text file using a ISO-8859-X encoding, you have to ***know*** the encoding, otherwise the content will be garbled
  - That’s why you sometimes see something like “*Öösel sõin ma ðokolaadi*” when opening text files

Unicode

- Unicode tries to fix this mess by having a single representation for every possible character in all world languages
- The latest version of Unicode (10.0) contains 136 755 characters covering 139 scripts
- Unicode uses 32 bits, meaning we can store 2^{32} = over 4 billion different characters

UTF-8

- The most straightforward way to store Unicode text is to use UTF-32, where each character is represented using 4 bytes
- However, this uses up a lot of memory, especially for English text: 4 times more than ASCII
- Therefore, 90% of texts on the web is stored using **UTF-8**:
  - Characters are represented using one or more bytes
  - Highest bit of the first byte is used a flag:
    - Highest bit 0: single character (0xxxxxxx)
    - Highest bit 1: part of a multibyte character
110xxxxx + 10xxxxxx
1110xxxx + 10xxxxxx + 10xxxxxx
  - Nice consequence: ASCII text is valid UTF-8 text
  - Estonian text in UTF-8 takes only slightly more space than ISO-8859-13 (only *öäüõš* use 2 bytes)
  - The higher is the Unicode ID of the character, the more bytes it uses in UTF-8

ISO-8859-13 *vs* UTF-8 example

- “käru”
- ISO-8859-13:
  `01101011 (k)  11100100 (ä)`
  `01110010 (r)  01110101 (u)`
- UTF-8:
  `01101011 (k)  11000011 10100100 (ä)`
  `01110010 (r)  01110101 (u)`

Regular expressions

Regular expressions

- Most important tool for describing *text patterns*
- Can be used to defining string patterns, e.g. that match:
  - Dollar amounts: $199, $25, $24.99
  - Western style person names: Donald Trump, Toomas Hendrik Ilves, George W. Bush (but be careful with names)
  - Hashtags: #yolo, #nlp, #regexp
- And for transforming:
  - $199 → 199 dollars
  - Donald Trump → D. Trump, Toomas Hendrik Ilves → T. H. Ilves, George W. Bush → G. W. Bush
- Highly useful in practical NLP, e.g. for data cleaning and transforming between simple formats

Basic Regular Expression Patterns

- Simplest regular expression is a sequence of simple characters, e.g. /woodchucks/ (note that '/' characters are not part of the regex, they simply denote that the part between /../ is a regex)
- Regular expressions are case-sensitive

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/!/	"You've left the burglar behind again!" said Nori

Sets

- Square braces denote a set of characters from which one has to match:

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

- Sets can also include a character range (e.g. [A-Z], be careful with *öäüõ*)

RE	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Negation and disjunction

- Caret (^) specifies what a single symbol cannot be
- Can be used for excluding single symbols or character sets

RE	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	"Oyfn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"I have no exquisite reason for't"
/[^\.]/	not a period	"our resident Djinn"
/[e^]/	either 'e' or '^'	"look up ^ now"
/a^b/	the pattern 'a^b'	"look up a^b now"

- Pipe symbol | denotes disjunction operator ("or"):
  - /cat|dog/ matches <u>cat</u> or <u>dog</u> but not cadog

Quantors

- Optional preceding symbol or expression: **?**
  - /Tallinn?/ matches Tallinn, Tallin
- Zero or more preceding symbols or expressions: *
  - /Tallinn*/ maches Tallin, Tallinn, Tallinnnnn
  - /a[bc]*/ matches a, acbb, but not adc
- One or more preceding symbols or expressions: **+**
  - /ba+!/ matches ba!, baa! but not b!
- Exactly *n* repetitions of the previous symbol/expression: **{n}**
  - /ba{4}/ matches baaa
- From *n* to *m* repetitions: **{n,m}**:
  - /ba{2,4}/ matches baa, baaa, baaaa

Wildcard

- Dot (“.”) is a **wildcard** that matches **any** single character
  - /beg.in/ matches *begin*, *began*, *beg n*, *beg9n*
  - Can be combined with quantors:
    - /.*/ matches string of any length
    - /aa.*bb/ matches *aabb*, *aaabb*, *aaccggrrttbb*

Anchors

- Anchors are special characters that *achor* regexes to special places in string:
  - ^ - start of string
  - $ end of string
  - \b word boundary
- Examples:
  - /^dog/ matches *dog* at the beginning of string, but not in a sentence like “I have a dog”
  - /dog$/ matches a *dog* at the end of a string
  - /\bthe\b/ matches the but not other
  - /\b99\b/ matches *99* in “I have 99 dollars” but not in “This costs $99” or “This happened in 1999”

Operator precedence

- Order of operator precedence:

Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ^my end$
Disjunction	|

- Examples:
  - **/the*/** matches the, theeee, but not thethe
  - **/(the)+/** matches the, thethe, thethethe
- When in doubt, use parenteses
  - **/cit(y)|(ies)/** matches city, cities

Predefined sets

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

Special symbols

RE	Match	First Patterns Matched
`\*`	an asterisk "*"	"K*A*P*L*A*N"
`\.`	a period "."	"Dr. Livingston, I presume"
`\?`	a question mark	"Why don't they come and lend a hand?"
`\n`	a newline	
`\t`	a tab	

Practical examples

- Find words ending with “a”: **/\S*a\b/**
- Find e-mail addresses that use .ee domain:
 /\b[a-z._]+@([a-z_]+\.)+ee\b/
- Find dates in format 03/11/2017:
/\b\d{2}\/\d{2}\/\d{4}\b/ (careful: also finds 91/88/0000)

Regexp exercise

Anchors

Anchros

^
Start of string, or start of line in multi-line pattern

\A
Start of string

$
End of string, or end of line in multi-line pattern

\Z
End of string

\b
Word boundary

\B
Not word boundary

\<
Start of word

\>
End of word

String

Replacement

$n
nth non-pa-ssive group

$2
"-xyz-" in /^(abc-(xy-z))$/

$1
"-xyz-" in /^(?:a-bc)-(xyz)$/

$`
Before matched string

Quantifiers

Quantifiers

*
0 or more

+
1 or more

?
0 or 1

{3}
Exactly 3

{3,}
3 or more

{3,5}
3, 4 or 5

Modifiers

Modifiers

g
Global match

i
Case-i-nse-nsitive

m
Multiple lines

s
Treat string as single line

x
Allow comments and white space in pattern

e
Evaluate replac-ement

U

Character

Character Classes

\c
Control character

\s
White space

\S
Not white space

\d
Digit

\D
Not digit

\w
Word

\W
Not word

\x
Hexade-cimal digit

\O
Octal digit

Special

\n
New line

\r
Carriage return

\t
Tab

\v
Vertical tab

\f
Form feed

\xxx
Octal character xxx

Examples

Metacharacter

^abc
abc, abcdefg, abc123, ...

abc$
abc, endsinabc, 123abc, ...

a.c
abc, aac, acc, adc, aec, ...

bill|ted
ted, bill

ab{2}c
abbc

a[bB]c
abc, aBc

(abc){2}
abcabc

ab*c
ac, abc, abbc, abbbc, ...

ab+c
abc, abbc, abbbc, ...

ab?c
ac, abc

a\sc
a c

Sample

([A-Za-z0-9-]+)
Letters, numbers and hyphens

(\d{1,2}\/\d{1,2}\/\d{4})
Date (e.g. 21/3/2006)

([^\s]+(?=\.(jpg|gif|png))\.\2)
jpg, gif or png image

(^[1-9]{1}$|^[1-4]{1}[0-

POSIX

POSIX

[:upper:]
Upper case letters

[:lower:]
Lower case letters

[:alpha:]
All letters

[:alnum:]
Digits and letters

[:digit:]
Digits

[:xdigit:]
Hexade-cimal digits

[:punct:]
Punctu-ation

[:blank:]
Space and tab

[:space:]
Blank characters

[:cntrl:]
Control characters

[:graph:]
Printed characters

[:print:]
Printed characters and spaces

[:word:]
Digits, letters and underscore

Support Us

Groups

Groups and Ranges

.
Any character except new line (\n)

(a|b)
a or b

(...)
Group

(?:...)
Passive (non-c-apt-uring) group

[abc]
Range (a or b or c)

[^abc]
Not a or b or c

[a-q]
Letter from a to q

[A-Q]
Upper case letter from A to Q

[0-7]
Digit from 0 to 7

\n
nth group/-sub-pattern

Assertions

Assertions

?=
Lookahead assertion

?!
Negative lookahead

?<=

Substitutions

- Regular expressions are often used for **substitution**
- In many Unix tools (*perl, vim, sed*), substitution can be invoked using:
  s/regexp/substitution/
- For example:
  `perl -npe 's/colour/color/g' f1.txt > f2.txt`
  replaces all instances of *colour* in *f1.txt with color* and saves the result to *f2.txt*
- Mastering regular expression substitutions makes many routine and awkward data cleaning and transformation tasks very easy

Substitutions using references

- Using references makes regular expression substitutions really powerful
- Numeric references (\1, \2, …) ***refer*** to parts in parentheses (and) in the search expression:
  - **s/(\d+)/<\1>/** surrounds all number sequences with <>, e.g.
    `foo 44 bar → foo <44> bar`
  - **s/(\w+) (\w+)/\2 \1/** switches the first two words of the line, e.g.:
    `John Smith 56998874 → Smith John 56998874`

Practical substitution examples

- Very hacky HTML markup remover:
  s/<[^>]*>/ /g
  a nice day → a nice day
- Add a +372 prefix to all (probable) Estonian phone numbers:
  s/\b(\d{6,7})\b/+372\1/g
  6888999 → +3726888999
- Replace names like *John Smith* with *J. Smith*:
  s/([A-Z])[a-z]* ([A-Z][a-z]+)]/\1. \2/
  (note that this fails with non-English characters and any word that looks different)

Basic text processing

- Basic NLP pre-processing usually consists of:
  - Tokenization
  - Text normalization
  - Sentence segmentation

Tokenization: what are words?

- How many words (**tokens**) in the following sentence: *Tom's bike is red, my bikes are red.*
  - 8 words if not counting punctuation
  - 10 if counting punctuation
  - Whether to treat punctuation as a word depends on the task
  - But how many different words (word **types**)?
    - 7 (with punctuation)
  - But are *bike* and *bikes* different words?
    - Thy have the same **lemma** (*bike)* but are different **wordforms**

How many different words are there?

- English:

Corpus	Tokens = N	Types = $\|V\|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

- Estonian
  - Number of word types increases more rapidly because of inflections (*koer, koera, koeraga, …*) and compound words (*hundikoer, koerakutsikas, …*)

How to split text into words?

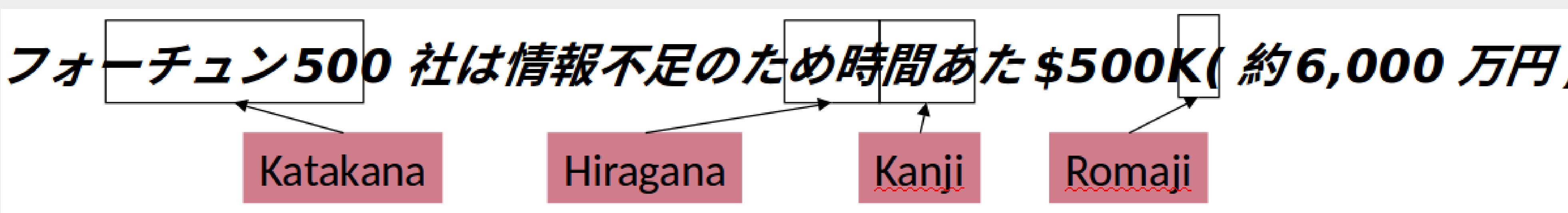
- Also called tokenization
- Split at whitespace?
  - What about punctuation? We usually want *The bike is red.* → *The bike is red .*
  - However, word-internal punctuation is usually kept: *John’s bike is red.* → *John’s bike is red .* *AT&T* → *AT&T, m.p.h* → *m.p.h*
  - Also, word-ending punctuation is not always a separate token:
    - “Mr. Big”
    - In Estonian “See juhtus 1976. aastal”

Tokenization: language issues

- French:
  - *l'ensemble* → l'ensemble or le *enseble*
- *English:*
  - *doesn't* → *doesn't* or *does n't*
- Finnish:
  - *Honkaharjun sairaalan osasto 3:lla* → *Honkaharjun sairaalan osasto 3:lla*

Tokenization: language issues

- Chinese and Japanese no spaces between words:
  莎拉波娃现在居住在美国东南部的佛罗里达。
  莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled



- Dates/amounts in multiple formats

Tokenization algorithm

- Text tokenization is language-specific
- Often done using rules, for example:
  - Split at whitespace
  - For every resulting token:
    - If it looks like an URL or e-mail, don't do anything
    - Separate token ending , ! ? : ; …
    - Separate quotes at word beginning and ends
    - Separate . at token ends, unless:
      - The token before . is numeric (e.g. *44.*)
      - The token is in a list of known abbreviations (e.g. *m.h.*) or is a single uppercase letter (e.g. *M.*)
      - The token after the current token starts with a lowercase letter
- Tokenization algorithm needs to be robust – it often needs to deal with typos, grammar mistakes.
- E.g., the algorithms above fails if there is no space after period in the sentence end:
  Koer sööb konti.Kass sööb kala. → [Koer] [sööb] [konti.Kass] [sööb] [kala] [.]
- Therefore, complicated heuristic rules are often added

Practical tokenization

- Pretty good tokenization (and many other NLP tools) for Estonian is implemented in the **EstNLTK** python package:

```
>>> from estnltk import Text
>>> text = Text('M. Õun elab 3. korrusel.')
>>> print (" ".join([w['text'] for w in text['words']]))
M. Õun elab 3. korrusel .
```

- For English, **spaCy** toolkit is a good choice:

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
>>> doc = nlp("Apple isn't interested in buying U.K. startup for $1 billion.")
>>> print(" ".join([w.text for w in doc]))
Apple is n't interested in buying U.K. startup for $ 1 billion .
```

Text normalization

- During text normalization, we standardize the words:
  - Depends on the task
  - Can include of the following steps:
    - Fixing most common orthographic problems (e.g. *š* is often written as *s*~ or sh in Estonian text)
      Masha ja Karu → Maša ja Karu
    - Normalizing punctuation marks to common UNICODE codepoints:
      John`s → John's
      "great" → ''great''
    - Recasing text written in all-caps
      THIS IS GREAT → this is great
    - Converting abbreviations to common form:
      U.S. → US
    - True-casing sentence-beginning words:
      Mari kasvab metsas → mari kasvab metsas
      Mari on tore tüdruk → Mari on tore tüdruk
    - Sometimes, we **lemmatize words** (useful for information retrieval)
      mari kasvab metsas → mari kasvama mets
      tõstsin pöidla üles → tõstma pöial üles
    - Expanding numbers, units (e.g., needed for speech synthesis):
      $45 → fourty five dollars
      kaalun 80 kg → kaalun 80 kilogrammi
  - Usually implemented using a mixture of rule-based and machine-learning methods
  - Often complicated, boring and time-consuming to implement, but very important for overall success

Sentence segmentation

- Sentence segmentation divides running text into sentence:
  I *see a dog. The dog eats a bone.* →
  [I see a dog .], [The dog eats a bone.]
- Usually quite simple, if tokenization has been done before (just split the text on separate .!?):
  I *see a dog . the dog eats a bone .* →
  [I see a dog .], [the dog eats a bone .]

Questions?