Natural Language Processing

Session 2: Basic string processing and regular expressions

Sascha Göbel
goebel@hertie-school.org

September 17, 2025

Basic string processing

Strings

- Record textual information
- Collections of one-character strings
- Sequences that maintain a left-to-right order
- In NL elements of these sequences are not independent

Basic string processing

Strings

- Record textual information
- Collections of one-character strings
- Sequences that maintain a left-to-right order
- In NL elements of these sequences are not independent

In Python

- Enclosed in single or double quotes
- Immutable (cannot be changed in place)
- Modification, extraction, splitting operations via built-in methods and functions

Regular Expressions (regex)

→ A set of characters to match and find substrings in a text
→ A tool for describing text patterns

Why regular expressions?

- Pattern-based/rule-based approach to NLP
- Generalized modification, extraction, splitting operations
- Useful in the absence of large amounts of data
- Useful to encode domain knowledge

In Python

- via `re` library
- using `re.match()`, `re.search()`, `re.sub()`, `re.findall()`, `re.split()`

Method	Purpose
`re.compile(`*`pattern`*`)`	Compile regex pattern
`re.match(`*`pattern, string`*`)`	Find first match from beginning of the string
`re.search(`*`pattern, string`*`)`	Find first match throughout the string
`re.findall(`*`pattern, string`*`)`	Find all matches throughout the string
`re.sub(`*`pattern, repl, string`*`)`	Replace all matches throughout the string
`re.split(`*`pattern, string`*`)`	Split string by all pattern occurrences

Regex patterns and grammar

Anchors

- Match a position before, after, or between characters

Pattern	Match
^	beginning of string
$	end of string
\b	word boundary

Character classes

- Match only one out of several characters

Pattern	Match
[A-Z]	upper case letters
[a-z]	lower case letters
[0-9]	digits
[^A-Z]	characters that are not upper case letters

Aliases

- Shorthand character classes

Pattern	Match
\d	any digit, e.g., [0-9]
\D	any non digit, e.g., [^0-9]
\w	any alphanumeric., e.g., [A-Za-z0-9_]
\W	any non-alphanumeric., e.g., [^A-Za-z0-9_]
\s	any whitespace., e.g., [\t\r\n\f]
\S	any non-whitespace., e.g., [[^ \t\r\n\f]

Quantifiers

- Repetition operators

Pattern	Match
?	zero or one instance of the previous character
*	zero or more occurences of the previous character
+	one or more occurences of the previous character
{n}	exactly n occurrences of the previous character
{n,m}	from n to m occurrences of the previous character
{n,}; {,m}	at least n; at least m occurrences of the previous character

Grouping and capturing

- Group parts of a regex together and potentially backreference

Pattern	Match
(*pattern*)	group *pattern* and capture, automatically numbered
\1	Recall/reference the captured group by number
(?:*pattern*)	group *pattern* without capturing

Lookaround

- Match conditional on context

Pattern	Match
match(?=*context*)	positive lookahead
match(?!*context*)	negative lookahead
(?<=*context*)*match*	positive lookbehind
(?<!*context*)*match*	negative lookbehind

Special characters

- Characters with special meaning, escaped with \ for literal use

Characters	Meaning
.	wildcard, matches (almost) any character
?,*,+, { }	quantifiers
|	disjunction
()	groupings
[]	character classes
\	escape sequences, aliases, backreferences, non-printable characters

Precedence

- Order of evaluation

Order	Operator
1.	\ (escaped characters)
2.	[] (character classes)
3.	() (groups)
4.	?,*,+,{} (quantifiers)
5.	literal characters, ^, $, \b
6.	| (alternation)