UNIVERSITÄT
ZU KÖLN

SPRACHVERARBEITUNG: ÜBUNG

SoSe 2024

Janis Pagel

Institut für Digital Humanities

2024-04-25

01

REGULAR EXPRESSIONS (REGEX)

Introduction

- Formal expressions to describe a (finite or infinite) set of strings
- Implemented in many (if not all) programming languages
  - Careful: Syntax of regex is often different between different programming languages
- There are many things to say about regex from a theoretcial point of view (formal languages, finite state automata, Chomsky hierarchy, computability, etc.), but this exercise will focus on the practical aspects and the implementation in Python
- Regex can in practice be used to search/query for strings or replace/delete (sub-)strings

Special Symbols in Python's RegEx I

- Most symbols just match a symbol in a string
  - the regex `a` matches a single character "a"
  - the regex `hello` matches the string "hello"
- The plus sign + modifies the previous symbol in a regex and means *matched at least once and repeated as often as wanted*
  - the regex `a+` matches the strings "a", "aa", "aaa", "aaaa", ...
  - the regex `ba+` matches the strings "ba", "baa", "baaa", "baaaa", ...
  - the regex `a+b+` matches the strings "ab", "aab", "aaab", "aaaab", "abb", "abbb", "abbbb", "aabb", ...
- The asterisk * modifies the previous symbol in a regex and means *repeated as often as wanted or empty string*
  - the regex `a*` matches the strings "", "a", "aa", "aaa", ...
  - the regex `aa*` matches the strings "a", "aa", "aaa", "aaaa", ...
  - the regex `a*b*` matches the strings "", "a", "b", "ab", ...
- The question mark ? makes the previous symbol optional
  - the regex `a?` matches "" or "a"
  - the regex `ba?` matches "b" or "ba"
- If you want to use a literal question mark, asterisk, plus sign, etc. you have to escape it with a backslash \
  - the regex `a?b\?` matches "b?" and "ab?"

Special Symbols in Python's RegEx II

- Curley brackets {} can be used to indicate the number of times the previous symbol should be repeated
  - the regex `a{2}` matches "aa"
  - the regex `a{2,4}` matches "aa", "aaa" and "aaaa"
  - the regex `a{3,}` matches at least three, "aaa", "aaaa", "aaaaa", ...
  - the regex `a{,3}` matches at most three, "a", "aa" and "aaa"
- Square brackets [] are used to indicate choices and specially defined ranges
  - the regex `a[bc]d` matches "abd" and "acd"
  - the regex `[0-9]` matches "0", "1", ..., "8", "9"
  - the regex `[a-d]` matches "a", "b", "c" and "d"
- The dot . matches any symbol (except newline)
  - the regex `a.c` matches the strings "abc", "azc", "a$c", "aéc", "a॔c", ...
- \w matches any alphanumerical character plus underscore
- \W matches any non-alphanumerical character
- \d matches any digit (mostly [0-9])
- \D matches any non-digit
- \s matches any whitespace (mainly space, newline and tab)
- \S matches any non-whitespace

Special Symbols in Python's RegEx III

- The caret `^` matches the beginning of a string and the dollar sign `$` matches the end of a string
  - the regex `^house$` matches "house", but not "brickhouse" or "house warming", etc.
- Round brackets with `?:` after the opening bracket `(?:)` can be used to group symbols together and apply operators on the whole group
  - the regex `fast(?:er)?` matches "fast" and "faster"
  - the regex `(?:abba)+` matches "abba", "abbaabba", "abbaabbaabba", ...
- The pipe symbol | indicates an alternative (or)
  - the regex `(?:(?:laugh|look|shout)ed)|went` finds "laughed", "looked", "shouted" and "went"
- The caret inside squared brackets means negation
  - the regex `hell[^o]` matches "hella", "hellb", "hellc", ..., but not "hello"
- There are many more options, check out `https://docs.python.org/3/library/re.html`

Regex Functions in Python

```python
import re
string = "something something that I want to match"
if re.search("that", string): # re.search returns true if the regex matches the (sub)string at least once
    print(True)
if re.match("something", string): # re.match returns true if the regex matches the beginning of the string
    print(True)
if not re.match("that", string):
    print(True)

print(re.findall("something", string)) # re.findall returns a list with all matches
print(re.findall("\w+", string))
print(re.findall("\s", string))

print(re.findall(" \w+ ", string)) # Matches are not overlapping

print(re.findall("<.+>", "<tag1> -- <tag2>")) # the * and + operators are "greedy" by default
print(re.findall("<.+?>", "<tag1> -- <tag2>")) # a question mark behind * and + makes them non-greedy

> True
> True
> True

> ['something', 'something']
> ['something', 'something', 'that', 'I', 'want', 'to', 'match']
> [' ', ' ', ' ', ' ', ' ', ' ']

> [' something ', ' I ', ' to ']

> ['<tag1> -- <tag2>']
> ['<tag1>', '<tag2>']
```

02

EXERCISE 04


UNIVERSITY
OF COLOGNE
Janis Pagel
Institut für Digital Humanities
eMail janis.pagel@uni-koeln.de
Website https://janispagel.de
Foto: Gregor Hübl