Computerlinguistik II: Übung

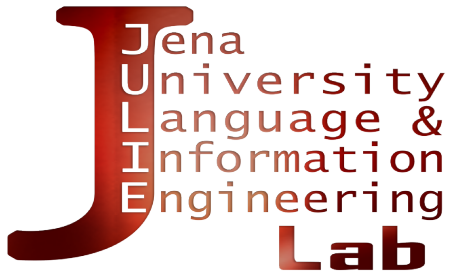
Reguläre Ausdrücke

Sven Büchel

Jena Language & Information Engineering (JULIE) Lab
Friedrich-Schiller-Universität Jena, Germany

https://julielab.de/

Sommersemester 2019



Grundlagen Regulärer Ausdrücke

- Reguläre Ausdrücke (engl. *regular expressions* (regex)) sind **abstrakte Ausdrücke** zur Beschreibung von Zeichenfolgen
- Ein Regex beschreibt ein Muster (**Pattern**), auf das eine Zeichenfolge passen kann, oder auch nicht (**Matching**)
- Regex können daher für eine verbesserte Suche eingesetzt werden, z.B.:
  - `Herrn? M[ae][iy]er` findet *Herr Meier, Herrn Meier, Herr Maier,...*
  - `\w+\.\w+@uni-jena.de` findet alle FSU-Email-Adressen
  - `\d{4,6}\.\d{4,8}` findet Telefonnummern

Allgemeines zur Syntax

- Die meisten Zeichen ändern ihre Bedeutung nicht, stehen also für sich selber. Darunter
  - Buchstaben (`a, b, c, ...`)
  - Ziffern (`0, 1, 2,...,9`)
  - Das Leerzeichen ␣
- Andere Zeichen kommen in Regex Sonderbedeutungen zu, insbesondere Klammern und manche Satzzeichen. Diesen muss ein Backslash vorangestellt werden, um trotzdem ihre *normale* Bedeutung darzustellen. Z.B:
  - `\.`
  - `\(`
  - `\?`
  - `\\`

Disjunktion und Negation

- Darstellung unterschiedlicher möglicher Zeichen oder Zeichengruppen (**Disjunktion**)
  - Einzelne Zeichen mit eckigen Klammern: `M[ae][iy]er`
  - Eine Spanne (**Range**) von Zeichen mit eckigen Klammer und Minus: `[0-9], [A-Za-z]`
  - Disjunktion beliebiger Zeichenfolgen mit dem Pipe-Operator: `(Herrn|Frau)`
- **Negation** von Zeichen durch Zirkumflex und eckigen Klammern
  - `[^a]`, jedes beliebige Symbol außer “a”
  - `[^0-9]`, keine Zahlen

Zeichenklassen

- `\d`, Ziffern, entspricht `[0-9]`
- `\D`, keine Ziffern
- `\w`, alphanumierische Zeichen (`[a-zA-Z0-9]`)
- `\W`, nicht-alphanumerische Zeichen
- `\s`, Whitespace-Zeichen (Space `\r`, Tab `\t`, Newline `\n`)
- `\S`, Nicht-Whitespace-Zeichen (Zahlen, Buchstaben, Satzzeichen)
- `\b`, Wortgrenzen-Zeichen (boundary), z.B. Whitespace und Interpunktion
- Darüber hinaus gibt viele weitere, insbesondere für UTF-8

Quantoren

- Quantoren (Quantifier) geben an, wie häufig das *vorausgehende* Zeichen wiederholt werden darf
- `?`, null oder ein Mal
- `+`, 1 bis *n*
- `*` , 0 bis *n*
- `{n}`, genau *n* Mal
- `{n,m}`, zwischen und *n* und *m* Mal

Wildcard

- Der Punkt `.` dient als *Wildcard* und matcht jedes beliebige Zeichen.
- Z.B. `w.rf` matcht “wirf” und “warf”, aber auch “wurf”, “werf”, “w4rf”,...
- Gerade die Kombination von Wildcard und Quantifiern (v.a. `+` und `*`) kann zu unerwarteten Effekten führen:
  - Regex: `dies.*\b`
  - *soll* matchen: dieser, dieses, dies,...
  - Text: *Sie findet diesen Sommer besonders schön.*
  - Matcht: *diesen Sommer besonders schön.*

Greediness

- Üblicherweise matchen Regex mit Quantoren die *längsmögliche* Zeichenfolge (**greedy**). Z.B.:
  - Regex: `a+h`
  - Text: *Aaaaaaah!*
  - Match: *aaaaaah* (und nicht *ah*)!
- Das Fragezeichen hinter dem Quantifer ändert dessen Verhalten, so dass jetzt die *kürzestmögliche* Zeichenfolge gematcht wird (**non-greedy, lazy**).
- Beispiel `dies.*?\b`

Escaping

- Um Zeichen mit Sonderbedeutungen darzustellen müssen diese **escaped** werden (Backslash voranstellen)
  - `\(\)`
  - `\{\}`
  - `\[\]`
  - `\.`
  - `\+`
  - `\?`
  - `\|`
  - `\\`

Cheat-Sheet Reguläre Ausdrücke

Disjunktion:

- `[Bb]`
- `[A-Z], [A-Za-z]`
- `(o|ou)`

Negation:

- `[^a], [^0-9]`

Quantoren:

- `?` 0 oder 1
- `+` 1 bis n
- `*` 0 bis n
- `{n}` genau n
- `{n,m}` n bis m

Zeichenklassen:

- `\d` Ziffern, `\D` keine Ziffern
- `\w` Alphanumerische
- `\s` Whitespace
- `\b` Wortgrenze

Wildcard:

- `.` matcht jedes Zeichen

Non-Greediness:

- `?`, z.B. `a*?`

Übung

Auf welche Art von Zeichenkette passen jeweils die folgenden Regex?

- `diese?`
- `g\wb`
- `(1\d|20)\d{2}`
- `\w+\.\w+@[\w\-]+\.\w{2,3}`

Umsetzung in Python

```
1  import re  # Import-Befehl. Ermoeglich Zugriff
       auf Pythons Bibliothek fuer Regulaere
       Ausdruecke.
2
3  text = "Ein schoener Bespielsatz mit vielen
       Buchstaben gefolgt von dem Punkt."
4
5  regex = r"[A-Za-z]+"  # Der Regulaere Ausdruck.
       Das r-Praefix wird nachfolgend erklaert.
6
7  re.findall(regex, text)  # Gibt Liste von
       Matches zurueck.
```

Probleme mit Backslashes

```
1  # Bei der Definition von Strings verwendet
       Python den Backslash fuer bestimmte
       Sonderzeichen
2  print("a\na")   # Ausgabe a <Zeilenumbruch> a
3
4  # Um einen Backslash zu schreiben, muss dieser
       daher escaped werden
5  print("\\")   # Ausgabe: \
6
7  # Gleichzeitig verwenden Regulaere Ausdruecke
       Backslashes zum Escapen IHRER Sonderzeichen
8  regex_fure_backslash = "\\\\"
9
10 # Praefix r deaktivert Python-Sonderrolle von
       Backslash. Regex bitte daher so schreiben.
11 print(r"\ ")   # Ausgabe: \
```

Sonderrolle von ()-Gruppen bei findall()

```
1  text = "Ein schoener Bespielsatz mit vielen
       Buchstaben gefolgt von dem Punkt."
2
3  # ()-Gruppen werden bevorzugt zurueckgegeben
4  re.findall(r"(Ein|dem) [a-zA-Z]+", text)  # ['
       Ein', 'dem']
5
6  # Bei mehreren Gruppen werden Tupel
       zurueckgegeben
7  re.findall(r"(mit) (vielen) [a-zA-Z]+", text)  #
       [('mit', 'vielen')]
8
9  # ?: am Anfang macht Gruppen "non-capturing"
10 re.findall(r"(?:Ein|dem) [a-zA-Z]+", text)  #['
       Ein schoener', 'dem Punkt']
```

Sonderrolle von ()-Gruppen bei findall()

Alternativen:

- Klammern um den gesamten Regex schreiben
- finditer()

Zum Nach- und Weiterlesen

https://docs.python.org/3/howto/regex.html

Übungen Reguläre Ausdrücke

Schreiben Sie jeweils einen Regex der (möglichst genau) folgendes erfasst.

1. Handynummern mit Ländervorwahl (*+49 157 8557354*, “Vorwahl” des Anbieters abgetrennt)
2. Erwähnungen von einem Herrn Friedrich Mayer, wobei auch die Kurzformen Fritz oder Fritzchen für den Vornamen verwendet werden könnten und Sie sich auch bei der Orthografie des Nachnamens unsicher sind. Sofern eine Anrede (z.B. “Herr”) vorhanden ist, soll diese ebenfalls erfasst werden.
3. Email-Adressen der FSU
4. Beliebige Email-Adressen