

# Natural Language Processing

## Reguläre Ausdrücke

Vitor Fontanella

Hochschule Hannover, Abteilung Information und Kommunikation

Folien von Christian Wartena NLP Vorlesung

# Suchen

- Wir können leicht in Zeichenketten Suchen
- Mit dem folgende Programm, können wir die Zeichenkette *Kraft* in der Datei `BrückenkursPhysik.txt` suchen

```
import codecs

buch = codecs.open('BrückenkursPhysik.txt', 'r', 'utf8')

for zeile in buch:
    if "Kraft" in zeile:
        print(zeile)

buch.close()
```

1  
2  
3  
4  
5  
6  
7  
8  
9

- Probleme:
  - 1 Wir möchten vielleicht auch den Plural, *Kräfte* finden
  - 2 Wir möchten auch *Schwerkraft* oder *Reibungskraft*
  - 3 Wir möchten ein Wort wie *Kraftwerk* dagegen nicht finden,

# Suchen

- Wir können leicht in Zeichenketten Suchen
- Mit dem folgende Programm, können wir die Zeichenkette *Kraft* in der Datei `BrückenkursPhysik.txt` suchen

```
import codecs

buch = codecs.open('BrückenkursPhysik.txt', 'r', 'utf8')

for zeile in buch:
    if "Kraft" in zeile:
        print(zeile)

buch.close()
```

1  
2  
3  
4  
5  
6  
7  
8  
9

- Probleme:
  - 1 Wir möchten vielleicht auch den Plural, *Kräfte* finden
  - 2 Wir möchten auch *Schwerkraft* oder *Reibungskraft*
  - 3 Wir möchten ein Wort wie *Kraftwerk* dagegen nicht finden,

# Suche mit regulären Ausdrücken

- Mit einem regulären Ausdruck ist das leicht zu erreichen:

```
import re

buch = codecs.open('BrückenkursPhysik.txt', 'r', 'utf8')
for zeile in buch:
    if re.search(r'\b\S*[Kk]r(a|ä)fte?\b', zeile):
        print(zeile)
buch.close()
```

1  
2  
3  
4  
5  
6  
7

- Vor dem String schreiben wir in Python ein `r`. Das `r` steht für *raw* und heißt, das Python nicht versuchen soll eine Folge wie `\b` zu interpretieren, sondern alles wie es da steht an die Funktion *search* übergibt.
- Wirkt vielleicht unübersichtlich, ist aber ganz einfach!

# Perl-Syntax für reguläre Ausdrücke

**Tabelle:** Alternativen und Wiederholung in der Perl-Syntax für regulären Ausdrücke

Symbol	Bedeutung
$\alpha \mid \beta$	$\alpha$ oder $\beta$
$\alpha^*$	$\alpha$ beliebig oft (Kleene-Abschluss)
$\alpha^+$	$\alpha$ beliebig oft, aber mindestens ein Mal
$\alpha?$	$\alpha$ oder nichts ( $\alpha$ is optional)
$\alpha\{n\}$	$\alpha$ $n$ -mal wiederholt
$\alpha\{n, m\}$	$\alpha$ mindestens $n$ Mal und maximal $m$ Mal
$\alpha\{n, \}$	$\alpha$ mindestens $n$ Mal

# Perl-Syntax für reguläre Ausdrücke

Tabelle: Sonderzeichen und Zeichenklassen

Symbol	Bedeutung
.	Ein beliebiges Zeichen, auSSer Zeilenumbruch
\.	.
\t	Tab-Zeichen
\n	Zeilenumbruch (Zeilenvorschub)
\r	Wagenrücklauf
\d	Ziffer (Digit)
\D	Beliebiges Zeichen, auSSer eine Zahl
\w	Ziffer oder Buchstabe ('Wort-Zeichen')
\W	Beliebiges Zeichen, auSSer ein Wort-Zeichen
\s	Leerzeichen oder Tab
\S	Beliebiges Zeichen, auSSer Leerzeichen und Tab
[ $\alpha$ ]	Ein beliebiges Zeichen aus $\alpha$ , z.B. [aoueIAOUEI]
[ $x - y$ ]	Ein beliebiges Zeichen aus der Reihe $x$ bis $y$
[ $^{\alpha}$ ]	Ein beliebiges Zeichen, auSSer ein Zeichen aus $\alpha$

# Perl-Syntax für reguläre Ausdrücke

**Tabelle:** Symbole für Grenzen in der Perl-Syntax für regulären Ausdrücke

Symbol	Bedeutung
<code>^</code>	Zeilenanfang
<code>\$</code>	Zeilenende
<code>\b</code>	Wortgrenze (Wortende oder Wortanfang)

# Was wurde gefunden?

- Wenn wir z.B. nach `ge\w+t` suchen, wissen wir, dass etwas gefunden wurden, aber nicht was.
- Die Ergebnisse werden von der Funktion `search` zurückgegeben.
- Mit `group()` bekommen wir ein Tuple. Das erste Element des Tuples, ist der String der mit dem Suchmuster übereinstimmt.

```
import re
```

```
text = "In einer Gesellschaft, die durch Zentralregierungen  
und Globalisierung geprägt ist, können diese Sprachen sich  
nur schwer behaupten und jährlich sterben schätzungsweise  
einige dutzende Sprachen aus."
```

```
fund = re.search(r'ge\S+t',text)
```

```
print(fund.group(0))
```



# Wo wurde gefunden?

- Mit den Methoden `start()` und `end()` des Match-Objektes finden wir die Position des Matches im Text:

```
import re

text = "In einer Gesellschaft, die durch
Zentralregierungen und Globalisierung geprägt
ist, können diese Sprachen sich nur schwer
behaupten und jährlich sterben schätzungsweise
einige dutzende Sprachen aus."

fund = re.search(r'ge\S+t',text)

print(fund.start() , "--" , fund.end())
```

# Gruppen

- Wir können in einem RE beliebig viele Klammerpaare hinzufügen, um
  - Reichweite von Operatoren festzulegen
  - Gruppen zu definieren
- Alle Klammerpaare bilden Gruppen, die von links nach rechts nummeriert werden
- Die Nummerierung fängt mit 1 an!
- Die Teile der gefundenen Zeichenkette, die einer Gruppe entsprechen können mit der Nummer abgefragt werden:



# Gruppen

```
import re

text = "In einer Gesellschaft, die durch
Zentralregierungen und Globalisierung geprägt
ist, können diese Sprachen sich nur schwer
behaupten und jährlich sterben schätzungsweise
einige dutzende Sprachen aus."

fund = re.search(r'ge(\S+)t',text)
verb = fund.group(1) + "en"

print(verb, "(Stamm gefunden auf:",
      fund.start(1), "--",fund.end(1),")")
```

# Search

- Mit `search()` finden wir das **erste** Vorkommen des Suchmusters im String.

# Findall

- Mit `findall()` finden wir alle Matches
- Das Ergebnis ist eine Liste
  - von Strings, wenn keine Gruppen genutzt wurden.
  - von Listen von Strings, wenn Gruppen genutzt wurden.

```
import re

text = "In einer Gesellschaft, die..."
fundliste = re.findall(r'[Gg]e\S+t', text)
for fund in fundliste:
    print(fund)
```

1  
2  
3  
4  
5  
6

```
import re

text = "In einer Gesellschaft, die..."
fundliste = re.findall(r'([Gg]e)(\S+)(t)', text)
for fund in fundliste:
    print(fund[0], "-", fund[1], "-", fund[2])
```

1  
2  
3  
4  
5  
6

# Split

- Mit `split()` teilen wir einen Text auf in eine Liste von Strings. Geteilt wird da, wo das Suchmuster gefunden wird.

```
import re
text = "In einer Gesellschaft , die durch Zentralregierungen
       und Globalisierung ..."
tokens = re.split(r'[ ,.\r\n]+', text)
print(tokens)
```

1  
2  
3  
4  
5  
6  
7

# Ersetzen

- Mit `sub(Muster,Ersetzung,String)` ersetzen wir jedes Vorkommen vom Muster im String durch Ersetzung.

```
import re
text = "Christian.Wartena@hs-hannover.de und rosa.tsegaye-
      aga@hs-hannover.de"
hidden = re.sub('@','(at)',text)
print(hidden)
```

1  
2  
3  
4  
5  
6  
7

# Gruppen Ersetzen

- Die Gruppen, die im Suchmuster definiert werden, können in der Ersetzung mit \1 usw. wieder aufgegriffen werden.
- Beispiel:

```
import re

IstNamen = []

for name in open("Namen.txt"):
    name1 = re.sub(r'(.*)_(^[^\\n\\r]*)\\s*', r'\2, \1', name)
    IstNamen.append(name1)

IstNamen.sort()

for name in IstNamen:
    print(name)
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12