Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozentin: Esther Seyffarth

Fortgeschrittene Zeichenkettenverarbeitung
mit regulären Ausdrücken

Motivation

- Wir haben bereits einige String-Operationen kennengelernt, z.B. `string.startswith()`, `string.endswith()` oder `string.replace(old, new)`.
- Allerdings ist es mit diesen Operationen nur möglich, konkrete Strings zu vergleichen (bzw. zu ersetzen).
- Reguläre Ausdrücke erlauben es uns, komplexe Muster für Strings zu definieren und für die weitere Verarbeitung zu nutzen.

Beispielaufgabe

- Beispielaufgabe: Schreiben Sie ein Programm, das alle Zahlen in einer Datei zensiert, also jede Zahl durch ein X ersetzt!
- Vorgehen ohne reguläre Ausrücke: Ersetze 1 durch X, ersetze 2 durch X, ersetze 3 durch X...
- Vorgehen mit regulären Ausrücken: Ersetze jedes Vorkommen eines Zeichens aus der Gruppe "Zahlen" durch X!

```
1 >>> import re
2 >>> zahlen = re.compile("[0-9]")
3 >>> secret = "Sozialversicherungsnr.: 968127490567"
4 >>> print(re.sub(zahlen, "X", secret))
5 Sozialversicherungsnr.: XXXXXXXXXXXX
```

Einsatzgebiete für reguläre Ausdrücke

- Mit regulären Ausdrücken können wir prüfen, ob ein String (z.B. Benutzereingabe) einem gewünschten Muster entspricht:
  - Ist ein als String übergebenes Datum korrekt formatiert?
  - Wurde eine valide Email-Adresse eingegeben?
  - Ermitteln des Betonungsmusters in transkribierten Wörtern:
    `IRREGULAR IH0 R EH1 G Y AH0 L ER0`
- Oder wir verändern Strings entsprechend unseren Wünschen:
  - Beim Verarbeiten von Webseiten: Entferne alle HTML-Tags!
  - Tokenisierung von Texten mit Abtrennung aller Satzzeichen von adjazenten Wörtern

Übersicht über grundlegende RegEx-Elemente

- `A`: Findet alle Vorkommen von `A` im String.
- `(A|B)`: Findet alle Vorkommen von `A` oder `B` im String.
- `+`: Voriges Element kommt mindestens einmal vor
- `*`: Voriges Element kommt 0 mal oder beliebig oft vor
- `?`: Voriges Element kommt 0 oder 1 mal vor
- `{3, 5}`: Voriges Element kommt 3 bis 5 mal vor
- `.`: Platzhalter für beliebiges Zeichen
- `\.`: Punkt
- `(...)`: Gruppierung mehrerer Elemente
- `[...]`: Definiert Mengen von Zeichen: Findet alle Zeichen, die hier angegeben werden. Mit `[^...]` kann eine Menge von Zeichen ausgeschlossen werden.
- `^`: Steht für den Anfang der Zeichenkette
- `$`: Steht für das Ende der Zeichenkette
- `\s`: Findet alle Whitespace-Zeichen (Leerzeichen, Tab, Zeilenumbruch)

Einfache reguläre Ausdrücke

- Formulieren Sie für jede Aufgabe einen regulären Ausdruck! Sie können Ihren Ausdruck z.B. mit `http://regexpal.com/` testen.
  - Personennamen nach dem Muster "Vorname Nachname" (erweitert: Personennamen nach diesem Muster oder nach dem Muster "Frau/Herr Nachname")
  - URLs, die `http://www.phil-fak.uni-duesseldorf.de/` untergeordnet sind (beliebig viele Unterordnungsebenen)
  - Datumsangaben nach dem Format TT.MM.JJJJ - Achtung! Nur mögliche Daten erlauben!

Reguläre Ausdrücke in Python

- Um mit regulären Ausdrücken arbeiten zu können, müssen wir das entsprechende Modul zunächst importieren.
- Mit `re.compile("...")` erzeugen wir ein RegEx-Objekt.
- `re.match(pattern, string)` und `re.search(pattern, string)` prüfen, ob ein Muster in einem String enthalten ist. Haben Sie eine Idee, was der Unterschied zwischen den beiden Funktionen ist?

```
1  import re
2  vowels = re.compile("[AEIOUaeiou]+")
3  word = input("Please enter a word: ")
4  if not re.search(vowels, word):
5      print("Invalid input!")
```

Übungsaufgabe zu regulären Ausdrücken

- Schreiben Sie ein Programm, das den Nutzer ein Wort eingeben lässt und ihn informiert, ob die Eingabe eine valide lateinische Substantivform ist oder nicht.
- Für die lateinischen Substantivformen können Sie in der Aufgabenstellung zu HA1 nachsehen.
- Definieren Sie einen regulären Ausdruck "wortform", der alle erlaubten Wörter erkennt!

Verarbeitung von regulären Ausdrücken: group()

- Mithilfe der Gruppierung in regulären Ausdrücken können wir auf die Substrings, die gefunden wurden, später zugreifen.
- Dazu erzeugen wir zuerst ein Match-Objekt, das die Verbindung zwischen dem Pattern und dem String darstellt, und ermitteln mit `group()` die Teilausdrücke, die den Gruppen im regulären Ausdruck entsprechen.

```
1  import re
2  satz = "Das ist ein Satz!"
3  match = re.search('[A-Za-z\s]+([.,!?;:])',satz)
4  if match:
5      satzzeichen = match.group(1)
6      print(satzzeichen)
```

- Achtung: `match.group(0)` liefert die gesamte Zeichenkette zurück, die das Pattern erfüllt.

Verarbeitung von regulären Ausdrücken: re.sub()

- In der Beispielaufgabe mit den zensierten Zahlen wurde bereits `re.sub()` verwendet.
- Auch bei `re.sub()` können wir auf die Teilstrings zugreifen.
- Was ist die Aufgabe dieses Codes?

```
1 import re
2 wort = "Singer-Songwriter"
3 kompositum = re.compile("([A-Za-z]+)(\-|\s)([A-Za-z]+)")
4 if re.search(kompositum, wort):
5     tokenized = re.sub(kompositum, "\\1 \\3", wort)
6     print(tokenized)
```

- Achtung: Die Gruppen müssen hier von zwei \ präfigiert werden.

- Das Lesen von regulären Ausdrücken ist nicht gerade einfach...
- Strukturieren Sie die Ausdrücke möglichst übersichtlich! Gruppieren Sie Elemente, die eine Einheit bilden!
- Hausaufgabe: Lösen Sie mindestens 3 Aufgaben auf `http://regexcrossword.com/`.
- Nicht besprochene Elemente von regulären Ausdrücken können Sie in der Doku nachlesen: `https://docs.python.org/3.4/library/re.html`.