

Programmierkurs Python

Michaela Regneri

2009-12-03

(Folien basieren auf dem gemeinsamen Kurs mit Stefan Thater)



Übersicht

- Reguläre Ausdrücke
- (etwas) mehr zu URLs

Reguläre Ausdrücke

- Reguläre Ausdrücke sind Suchmuster (Pattern) für Zeichenketten (Strings)
- „Matchen“ eines Reguläre Ausdrucks:
Passt das Muster auf einen (Teil-) String?
- Weitere Operationen:
 - Ersetzen von Teilstrings in einem String
 - Zerlegen von Strings
 - Extraktion von Teilstrings

3

Reguläre Ausdrücke in Python

- Reguläre Ausdrücke sind nicht „fest eingebaut“ – um sie zu verwenden, muss das re-Modul importiert werden:

```
import re
```

- Keine eigene Syntax: Reguläre Ausdrücke werden als String-Literale aufgeschrieben
- Die einfachsten regulären Ausdrücke sind Strings - die matchen sich selbst
 - 'a' matcht *a*
 - 'hallo' matcht *hallo*

4

Reguläre Ausdrücke

- Im einfachsten Fall besteht ein regulärer Ausdruck aus einem normalen String.
- Matchen von regulären Ausdrücken.
 - `re.match("fool", "foolisch")` ⇒ Match-Objekt
 - "fool" – der reguläre Ausdruck
 - "foolisch" – der String, auf den wir den regulären Ausdruck anwenden.
- Das Resultat ist ein Match-Objekt ... oder None, wenn das Muster nicht passt. Über Match-Objekte kann man u.a. Teilstrings extrahieren (später mehr dazu).

5

Reguläre Ausdrücke in Python

- Der Standard-Weg, um in Python mit Regulären Ausdrücken zu arbeiten:
 - kompiliere den Ausdruck zu einem Pattern-Objekt
 - matche das Pattern mit einem Text (suche es im Text oder teste, ob der text dem Pattern entspricht)
 - Wenn der Abgleich erfolgreich war, kommt ein „Match-Objekt“ zurück, das Informationen über das Matching enthält
- Den Kompilier-Schritt muss man nicht explizit hinschreiben, es kann aber praktischer und effizienter sein (später mehr)

6

Reguläre Ausdrücke in Python

Testet, ob text1
vollständig pattern
entspricht, sonst wird
matcher zu None

Sucht das erste
Auftreten von pattern
in text2

(Start- und End-)
Position der
gematchten Sequenz
in text2

```
import re
text1 = 'ab'
pattern = 'ab'
matcher = re.match(pattern, text1)
if matcher != None:
    print("Match.")
text2 = 'aaabababababbb'
matcher = re.search(pattern, text2)
print(matcher.span())
```

Match.
(2, 4)

Konsolen-Ausgabe

7

match & search

- `re.match(pattern, string)` ist erfolgreich, wenn pattern auf den Anfang von string matcht.
- `re.search(pattern, string)` ist erfolgreich, wenn pattern auf einen beliebigen Teilstring von string matcht
- beide geben Objekte vom Typ `MatchObject` zurück; das `MatchObject` von `re.search` entspricht dem ersten gefundenen Match

8

Funktionszeichen (Meta-Characters)

- Die meisten Zeichen in einem regulären Ausdruck matchen auf sich selbst
 - Groß- und Kleinschreibung wird unterschieden
 - Folgende Funktionszeichen haben eine besondere Bedeutung: `. ^ $ * + ? { [] \ | ()`
- Maskierungszeichen (Escape Character): Wenn man ein Funktionszeichen als normales Zeichen („wörtlich“) verwenden möchte: Backslash als Präfix

9

„Raw String“ Notation

- Der Backslash ist Maskierungszeichen sowohl für Strings als auch für reguläre Ausdrücke
- Da reguläre Ausdrücke als Strings kodiert werden, führt dies schnell zu einer „Backslash-Plage“
 - Zeichenkette in einer Datei: `\section`
 - Als Python-String: `"\\section"`
 - Als regulärer Ausdruck: `"\\\\section"`
- Alternative String-Notation: `r"raw strings"`
 - Der Backslash wird einfach als Zeichen betrachtet
 - Zum Beispiel: `r"\section"`

10

Zeichenklassen (Character classes)

- Mit [und] kann man Zeichenklassen spezifizieren; sie matchen auf jedes Zeichen in der Klasse
 - [abc] match 'a', 'b' oder 'c'
 - [a-z] matcht alle Kleinbuchstaben (keine Umlaute)
- Funktionszeichen (modulo Backslash) verlieren in Zeichenklassen ihre Sonderbedeutung:
 - [ab\$] matcht 'a', 'b' oder '\$'
- Komplementieren: [^abc] matcht alle Zeichen, die nicht in der Klasse [abc] sind

11

Zeichenklassen (Character classes)

- Einige vordefinierte Zeichenklassen:
 - \d = [0-9] \D = [^0-9]
 - \s = [\t\n\r\f\v] \S = [^ \t\n\r\f\v]
 - \w =(Buchstaben +\d +_) \W = (der Rest)
- Der Punkt . matcht normalerweise alles bis auf das Zeichen für den Zeilenumbruch (Newline, \n)

12

Quantoren

- Mit dem *-Operator matcht man den vorhergehenden Ausdruck beliebig oft
 - `[ab]*` matcht `'ac'`, `'aabaac'`, `'c'`, etc.
- Weitere Quantoren:
 - `a+` (mindestens einmal)
 - `a?` (höchstens einmal)
 - `a{m,n}` (mindestens m, höchstens n mal)
- Der Quantor bezieht sich auf den unmittelbar vorhergehenden Ausdruck:
 - `ab*c` matcht `'abbc'`, aber nicht `'ababc'`
 - `(ab)*c` matcht `'ababc'`, aber nicht `'abbc'`

13

Weitere Operatoren

- `a|b` matcht a oder b
 - `(ver|ent|be)sorg(en|t|te)` – versorgen, versorgte, besorgen, besorgt, ...
- `^`, `$` matchen den Zeilenanfang bzw. das Zeilenende
- `\A`, `\Z` matchen den Anfang bzw. das Ende des Strings
 - `\A[abc]*\Z` matcht alle Strings, die aus beliebigen Kombinationen von `'a'`, `'b'` und `'c'` bestehen
- `\b` matcht Wortgrenzen:
 - `class\b.*` matcht `'class next Friday'`
 - `class\b.*` matcht nicht `'classified'`

14

Übung – Wer matcht was?

- | | |
|-----------|------------------------|
| (1) abc | (1) $ab+c?$ |
| (2) ac | (2) $a?b^*c$ |
| (3) abbb | (3) $b+c^*$ |
| (4) bbc | (4) $^b+c^*\$$ |
| (5) aabcd | (5) $a.+b?c$ |
| (6) b | (6) $b\{2,\}c?$ |
| | (7) $^a\{1,2\}b+.?d^*$ |

15

Was matcht $a(ab)^*a$?

- (1) abababa
- (2) aaba
- (3) aabbbaa
- (4) aba
- (5) aabababa

16

Gruppierungen

- Mit Klammern kann man reguläre Ausdrücke gruppieren
- Ausserdem kann man auf die Gruppe später über das Match-Objekt auf den gematchten Teilstring zugreifen:

```
mo = re.search(r'"(\d+)"', 'price="123"')
```

```
mo.groups() ⇒ ('123',)
```

```
mo.group(1) ⇒ '123'
```

`group(0) == group()`
– der vollständige Treffer-String

- Named Groups:

```
mo = re.search(r'"(?:P<price>\d+)"', 'price="123"')
```

```
mo.groupdict() ⇒ { 'price': '123' }
```

17

Match-Objekte: Weitere Methoden

- `MatchObject.start([group])`
Position im String, an der die Gruppe `group` beginnt.
- `MatchObject.end([group])`
Position im String, an der die Gruppe `group` endet.
- `MatchObject.span([group])`
Paar aus Beginn und Ende der Gruppe (Positionen)

18

„Greedy“ quantifiers („gierige Quantoren“)

- Vorsicht beim Verwenden von Quantoren in Gruppen:
Quantoren matchen den langstmöglichen Teilstring!

```
mo = re.search('\"(.*)\"', 'price="123" art="ball"')  
mo.group(1) ⇒ '123" art="ball'
```

- Mit einem nachgestellten ? ändert man das:

```
mo = re.search('\"(.*)?\"', 'price="123" art="ball"')  
mo.group(1) ⇒ '123'
```

19

Non-Capturing Groups & Lookaheads

- Wenn man nur den regulären Ausdruck klammern möchte:

```
mo = re.match('(?:abc)+', 'abcabc')  
mo.groups ⇒ ()
```

- Positiver Lookahead: matcht den regulären Ausdruck, ohne Teile der Eingabe zu konsumieren

```
mo = re.match('(.*)(?=(abc))(.*)', 'abcabc')  
mo.groups ⇒ ('abc', 'abc', 'abc')
```

- Negativer Lookahead: (?!a), a darf nicht matchen

20

Weitere Funktionen

- `re.findall(pattern, string)` – suchte alle nicht-überlappenden matchenden Teilstrings in `string` (Rückgabe: Liste von Strings / Tupeln)
- `re.finditer(pattern, string)`: wie `findall`; gibt iterierbares Objekt zurück, ein Sammelobjekt mit Match-Objekten
- `re.split(pattern, string)` – Zerlegt `string` in Teilstrings an Stellen, an denen `pattern` matcht
- `re.sub(pattern, repl, string)` – ersetzt alle matchenden Teilstrings in `string` durch `repl`

21

Optionen beim Matchen

- Man kann das Verhalten von `re.match` etc. durch Optionen modifizieren:
 - `re.IGNORECASE` – Groß/Kleinschreibung ignorieren
 - `re.MULTILINE` – `^`, `$` matchen den Anfang bzw. das Ende jeder Zeile im String.
 - `re.DOTALL` – `.` matcht auch den Zeilenumbruch (`\n`)
- Für mehrere Flags: kombination mit „|“ (bit-weises oder)

```
flag = re.DOTALL | re.IGNORECASE
mo = re.match('a.b', 'aA\nb', flag)
print(mo.group())
```

A	Konsolen-Ausgabe
b	

22

Pattern kompilieren

- Wenn man das Pattern vorher kompilieren will, verändert sich die Benutzung von Match / Search:
 - `re.compile(str)` erzeugt ein Objekt vom Typ `re.RegexObject`
 - `re.RegexObject` implementiert alle Such-/Matchfunktionen aus `re` (`match`, `search`, `findall`,...) mit gleichem Rückgabotyp
- Flags werden ggf. schon beim Kompilieren mitgegeben:
 - `re.compile(str, flag)` produziert ein `RegexObject`, das nur mit den Bedingungen von `flag` sucht / matcht

23

Pattern kompilieren

- Wenn man das gleiche Pattern mehrmals braucht, ist es viel effizienter, es einmal zu kompilieren und dann mit dem `RegexObject` weiter zu arbeiten
- außerdem sind `search` und `match` hier flexibler:
 - `RegexObject.match(str, [start, [end]])` matcht den String von Position `start` bis `end` (wenn `end` fehlt, bis zu Ende)
 - `RegexObject.search(str, [start, [end]])` `search` und `RegexObject.findall(str, [start, [end]])` funktionieren analog

24

Pattern kompilieren

```
import re
pattern = re.compile('a.?b', re.DOTALL | re.IGNORECASE)
matcher1 = pattern.match('A\nb')
matcher2 = pattern.search('aaba')
matcher3 = pattern.match('aaba', 1, 4)
matcher4 = pattern.search('abA?BaB', 3)
```

`matcher.group()` ⇒ ?

25

Mehr zu regulären Ausdrücken

- <http://docs.python.org/3.1/library/re.html>
- <http://docs.python.org/howto/regex.html>

26

Reguläre Ausdrücke & Unicode

- Unicode ist Standard in regulären Ausdrücken
- `\w` z.b. matcht alle Buchstaben aus egal-welchem in Unicode repräsentierten Alphabet
- man kann explizit angeben, kein Unicode benutzen zu wollen, mit `re.ASCII` (und ggf. `re.LOCALE`)
 - `re.findall('(\w+)', 'Hällo, WörlD!', re.ASCII)`

```
> re.findall('(\w+)', 'Hällo, WörlD!')
['Hällo', 'WörlD']

> re.findall('(\w+)', 'Hällo, WörlD!', re.ASCII)
['H', 'llo', 'W', 'rld']
```

27

UTF-8 – Achtung!

- Wenn man mit UTF8-kodierten Bytestrings arbeitet, muss man beachten, dass Zeichen mit Code-Point ≥ 128 durch eine variable Anzahl von Bytes repräsentiert werden
 - `len("Hällo") \Rightarrow 5`
 - `len("Hällo".encode("utf-8")) \Rightarrow 6`
 - `"Hällo"[:2] \Rightarrow "Hä"`
 - `"Hällo"[:2].encode("utf-8") \Rightarrow b'H\xc3\xa4'`
- Beim Verwenden regulärer Ausdrücke sollte generell mit Unicode-Strings gearbeitet werden

28

(Etwas) mehr zu URLs

- Nicht alle URLs erlauben jeden automatischen Zugriff
- Manchmal braucht man kompliziertere Operationen als nur öffnen (Formulare,...)
- dafür gibt es in `urllib.request` (u.a.) die Klassen `URLopener` und `Request`
 - `Request` kann detaillierte Informationen beim Öffnen der Webseite mitschicken (wie den Namen des Programms, das die Webseite gerade öffnet)
 - `URLopener` führt den komplexen Request aus

29

(Etwas) mehr zu URLs

- ein Beispiel:

```
import urllib.request as url
```

```
req = url.Request('http://www.google.com/search?q=hallo')
```

```
browser = 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)'
```

```
req.add_header('User-Agent', browser)
```

Anfrage, die so tut, als käme sie von Mozilla

```
opener = url.build_opener()
```

Der eigentliche „Browser“, der die Anfrage ausführt

```
for line in opener.open(req):
```

```
    print(line)
```

Achtung - Byte-Strings!

30

Einfache „Web-Linguistik“ in Python

- automatisches Ermitteln von Google-Trefferzahlen
- Finde die URL, mit der man auf Suchergebnisse zugreift
- Finde einen regulären Ausdruck, der genau die Trefferanzahl matcht

Results 1 - 10 of about 133,000,000 for hallo. (0.11 seconds)

Results 1 - 10 of about 133,000,000 for hallo. (0.11 seconds)

31

Einfache „Web-Linguistik“ in Python

```
# -*- coding:utf-8 -*-
import urllib.request as url
import re

req = url.Request('http://www.google.com/search?q=hallo')
browser = 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)'
req.add_header('User-Agent', browser)

opener = url.build_opener()
pattern = re.compile('of about..b\\W((\\d+\\W)*(\\d+)).+b')

for line in opener.open(req):
    matcher = pattern.search(str(line))
    if matcher != None:
        print(matcher.group(1))
```

Pseudo-Mozilla-Browser

reg. Ausdruck für den String, der die Trefferzahl enthält

Byte-Strings! :-)

32

Übung 1: Bigramme

MARLEY was dead: to begin with. There is no doubt whatever about that. The register of his burial was signed by the clergyman, the clerk, the undertaker and the chief mourner. Scrooge signed it: and Scrooge's name was good upon 'Change, for anything he chose to put his hand to. Old Marley was as dead as a door-nail.

Mind! I don't mean to say that I know, of my own knowledge, what there is particularly dead about a door-nail. I might have been inclined, myself, to regard a coffin-nail as the deadest piece of ironmongery in the trade. But the wisdom of our ancestors is in the simile; and my unhallowed hands shall not disturb it, or the Country's done for. You will therefore permit me to repeat, emphatically, that Marley was as dead as a door-nail.

```
door nail: 3
a door: 3
was as : 2
in the : 2
dead as : 2
...
```

33

Übung 2: Web Crawler



Little Prince - Link collection
Der kleine Prinz - Linksammlung
Le Petit Prince - collection des liens



Webseiten über den Kleinen Prinzen
Little Prince Websites

- Official Website - The 'official' Little Prince website.
- Julians Page - Julian, one of my fellow collectors, presents his own site. Very good work!
- Jaume Arbonès - Beautiful and comprehensive page!
- Michaels Site - Very detailed site with lots of background information.
- Little Prince-Online versions - Large collection of front covers.
- Patricks Page - Extensive list of existing translations (including references, addresses and ISBN). Needs your assistance!
- Ralfs Page - Ralf is collecting the wonderful "fox message" from chapter XXI.
- Slaweks Page - Slawek is another collector providing a beautiful homepage.
- Stefanos Page - Stefano is a successful collector from Italy, have a look!

34

Übung 2: Web Crawler

```
<TD VALIGN="MIDDLE" height="18">  
<a href="http://www.lepetitprince.com/"><b>Official Website</b></a> - The  
'official' Little Prince website.</TD> </TR>
```

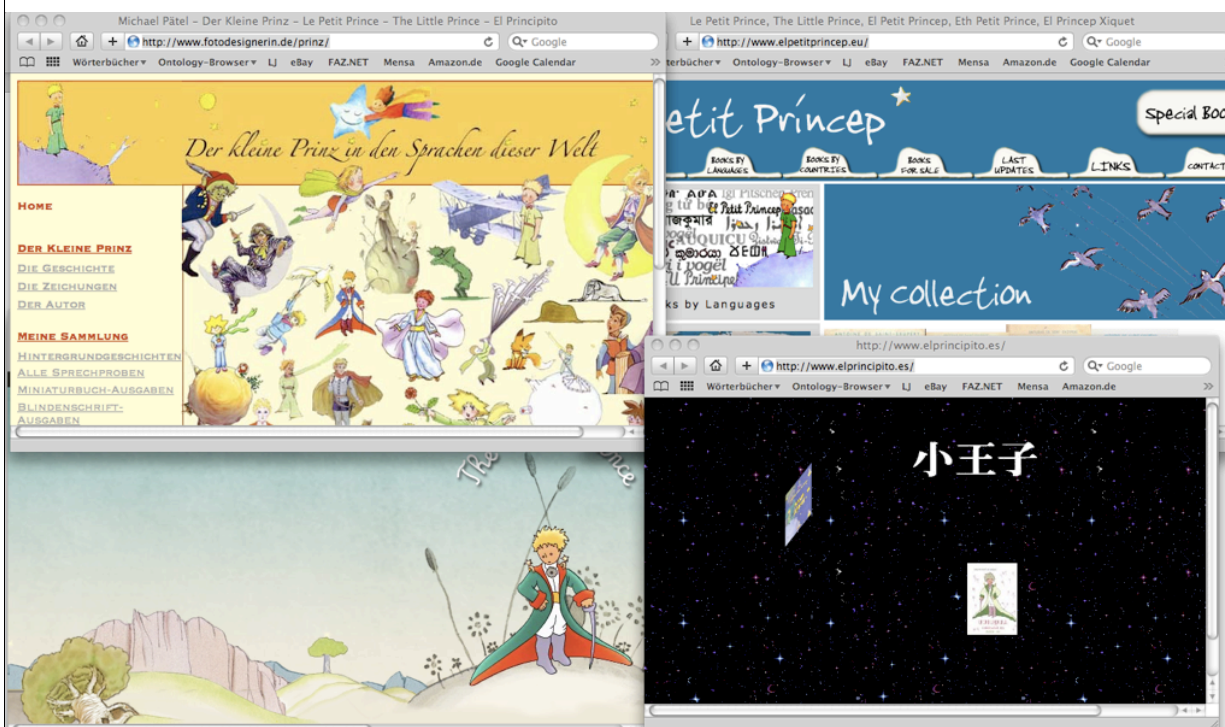
```
<TR><TD VALIGN="MIDDLE" height="18">  
<a href="http://www.elprincipito.es/"><b>Julians Page</b></a> - Julian,  
one of my fellow collectors, presents his own site. Very good work!</  
TD></TR>
```

```
<TR><TD VALIGN="MIDDLE" height="18">  
<a href="http://www.elpetitprincep.eu/"><b>Jaume Arbonès</b></a> -  
Beautiful and comprehensive page!</TD></TR>
```

```
<TR> <TD VALIGN="MIDDLE" height="18">  
<a href="http://www.fotodesignerin.de/prinz/"><b>Michaels Site</b></a> -  
Very detailed site with lots of background information.</TD></TR>
```

35

Übung 2: Web Crawler



36