Reguläre Ausdrücke

Dominik Pataky

18. Dezember 2017

Python-Kurs 2017

Gliederung

1. Grundlagen
2. Matching Regeln
    - Sonderzeichen
    - Zusammengesetzte Regex
    - Spezielle Sequenzen
3. Methoden
4. requlare expression Objekt
5. Match Objekt

Grundlagen

Grundlagen

Das re Modul der Python Standard Library ist die Python Implementierung von regulären Ausdrücken.

Reguläre Ausdrücke werden verwendet um die Struktur von Text/Sprachen zu beschreiben.

Grundlagen

Ein erstellter regulärer Ausdruck kann verwendet werden um:

- die Struktur eines Textes zu überprüfen
- bestimmte Teile eines Textes zu extraktieren

Grundlagen

Die Anwendung eines regulären Ausdruckes nennt man Matching.

Matcht der reguläre Ausdruck einem String, bedeutet dies, dass der String die Struktur hat, die der reguläre Ausdruck beschreibt.

Reguläre Ausdrücke werden oft auch ’regex’ genannt (kurz für regular expression).

Matching Regeln

Matching Regeln

Jeder Buchstabe und jede Zahl matcht immer einmal sich selbst.

Das bedeutet:

- Der String `'a'` matcht der regex `'a'`.
- Der String `'abc'` matcht der regex `'abc'`, nicht aber regex `'a'`.
- `'4'` matcht `'4'`, nicht aber `'5'`, `'a'` oder `'41'` usw.

Sonderzeichen

`.` (Punkt) matcht einem beliebingen Schriftzeichen, außer `'\n'` (newline)

Für die regex `'.'` gilt:

- `'a'` matcht
- `'b'` matcht
- `'4'` matcht
- `'ab'` matcht nicht, denn `'.'` ist nur ein Zeichen und `'ab'` sind zwei.

Sonderzeichen

`[]` matcht jedem der in den Klammern stehenden Zeichen, jedoch nur einmal (wie bei `.`).

Für die regex `'[abg]'` gilt also:

- `'a'` matcht
- `'b'` matcht
- `'g'` matcht
- `'ab'` matcht nicht (zwei zeichen matchen nicht einem).

Sonderzeichen

`( )` erstellt eine Gruppe. Alles was in den Klammern steht, muss genau so vor kommen.

Für die regex `'(abc)'` gilt:

- `'a'` matcht nicht
- `'ab'` matcht nicht
- `'abc'` matcht

Sonderzeichen

`\` escaped ein Sonderzeichen.

Alle hier aufgeführten Sonderzeichen können nicht in einem Pattern vorkommen, als das, was sie eigentlich bedeuten, dafür müssen sie extra markiert werden.

- `'\\'` als Pattern matcht auf den String `'\'`
- `'\.'` als Pattern matcht auf den String `'.'`

Sonderzeichen

`^` matcht ab dem Anfang eines Strings oder ab jedem `\n`

Für die regex `'^a'` ergibt sich also:

- `'a'` matcht
- `'ba'` matcht nicht, da der falsche Character am Anfang steht.
- `'aba'` matcht

Sonderzeichen

`$` matcht dem Ende eines Strings (oder dem Zeilenende)

Für die regex `'a$'` folgt daraus:

- `'a'` matcht
- `'ba'` matcht
- `'bab'` matcht nicht, da der falsche Character am Ende steht
- `'aba'` matcht

Sonderzeichen

| ist ein **ODER**. Entweder die regex davor oder danach muss matchen.

Für die regex ‘’a|b‘‘ gilt:

- `'a'` matcht
- `'b'` matcht
- `'ab'` matcht nicht, da `'a|b'` mit [ab] gleichzusetzen ist

Zusammengesetzte Regex

Regular Expressions setzen sich aus kleineren Regular Expressions zusammen.
So kann man z.B. auch festlegen, wie häufig ein Zeichen auftreten soll.

Zusammengesetzte Regex

`*` - Die vorangestellte Regex muss 0 - n Mal vorkommen.

Für die regex `'a*'` gilt:

- `''` matcht
- `'a'` matcht
- `'aa'` matcht
- `'aaaab'` matcht nicht, da ein anderes Zeichen als ’a’ vorkommt

Zusammengesetzte Regex

`+` - Die vorangestellte Regex muss 1 - n Mal vorkommen.

Für die regex `'a+'` gilt:

- `''` matcht nicht, da es kein mal vorkommt
- `'a'` matcht
- `'aa'` matcht
- `'ab'` matcht nicht, da ein anderes Zeichen als ’a’ vorkommt

Zusammengesetzte Regex

`?` - Die vorangestellte regex muss 0 - 1 Mal vorkommen.

Für die regex `'a?'` gilt:

- `''` matcht
- `'a'` matcht
- `'aa'` matcht nicht, da das Zeichen öfter, als nur ein mal vorkommt

Zusammengesetzte Regex

`{m}` - Die vorangestellte regex muss genau m Mal vorkommen.

Für die regex `'y{3}'` gilt:

- `'yyy'` matcht
- `'y'` matcht nicht, da es zu wenige Zeichen sind
- `'yyyy'` matcht nicht, da es mehr Zeichen sind

Zusammengesetzte Regex

`{m,n}` - Die vorangestellte regex muss m - n Mal vorkommen.

Für die regex `'y{2,5}'` gilt:

- `'yyy'` matcht
- `'y'` matcht nicht, da es zu wenige Zeichen sind
- `'yyyy'` matcht
- `'yyyyyy'` matcht nicht, da es zu viele Zeichen sind

Spezielle Sequenzen

Desweiteren gibt es noch spezielle Sequenzen, wie z. B.

- `\d` für Unicode Ziffern, äquivalent für `[0-9]`
- `\D` ist das Gegenteil, alles was keine Unicode Ziffern sind
- `\s` für alle Whitespace Zeichen, das entspricht `'[ \t\n\r\f\v]'`
- `\S` entspricht wieder dem Gegenteil
- `\w` für alle Unicode Zeichen `'[a-zA-Z0-9_]'`
- `\W` für alle Nicht-Unicode Zeichen
- `[^...]` entspricht allem, was nicht in den Klammern steht

Methoden

Methoden

```
compile(pattern, flags=0)
```

Wandelt einen String in ein regular expression Objekt um.

```
search(pattern, string, flags=0)
```

Sucht in ‘string‘ nach dem Pattern ‘pattern‘.

```
match(pattern, string, flags=0)
```

Sucht am Begin des Strings nach dem Pattern.

Methoden

```
fullmatch(pattern, string, flags=0)
```

Der komplette String und das Pattern müssen übereinstimmen.

```
findall(pattern, string, flags=0)
```

Gibt eine Liste von Strings mit allen passenden Übereinstimmungen zurück.

```
finditer(pattern, string, flags=0)
```

Gibt einen Iterator, welcher ‘match‘ Objekte beinhaltet zurück

Die restlichen Funktionen können in den Docs gefunden werden.

requlare expression Objekt

requlare expression Objekt

Ein solches Objekt hat im Großen und Ganzen die selben Methoden, jedoch ohne zusätzliches Pattern, da das Objekt an sich bereits ein Pattern enthält.

Match Objekt

Match Objekt

```
start([group])
```

Gibt die Startposition des Patterns im String zurück.

```
end([group])
```

Gibt die Endposition des Patterns im String zurück.

```
span([group])
```

Gibt ein Tuple zurück `(m.start([group]), m.end([group]))`