Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: 02-01 (Strings)

Einführung in die computerlinguistische Programmierung mit Python

09-01: Reguläre Ausdrücke schreiben 

In der Computerlinguistik haben wir viel mit Strings, Wörtern, Texten zu tun. Diese Woche beschäftigen wir uns mit einer Möglichkeit, Strings in Bezug auf selbstdefinierte Muster zu untersuchen: **Reguläre Ausdrücke** (englisch *Regular Expressions*, *regex*).

Was ist ein regulärer Ausdruck?

Reguläre Ausdrücke sind eine Art Beschreibungssprache, mit der wir **Bedingungen für den Aufbau von Strings (Zeichenketten)** ganz genau ausformulieren können. Das erlaubt uns, Strings zusammenzufassen oder zu unterscheiden, indem wir prüfen, **ob ein String zu unserem selbstdefinierten Muster passt oder nicht**.

Heute programmieren wir nicht in Python, sondern schreiben nur reguläre Ausdrücke. Auf https://www.regexpal.com/ kann man reguläre Ausdrücke entwerfen und direkt für **Beispieleingaben** testen, auf welche Strings der Ausdruck zutrifft und auf welche nicht. Alle hier folgenden Beispiele könnt ihr auf regexpal selbst ausprobieren und nachvollziehen.

Hier im Skript wird eine Auswahl der wichtigsten Regeln für reguläre Ausdrücke beschrieben. Man findet aber auch online jede Menge Ressourcen und "Cheat Sheets". Die **Python-Doku** selbst ist recht umfangreich in der Beschreibung von regulären Ausdrücken: https://docs.python.org/3.8/library/re.html#regular-expression-syntax

Um trivial anzufangen: Jedes Wort kann gleichzeitig auch ein regulärer Ausdruck sein. Hier zum Beispiel ein regulärer Ausdruck, der alle Wörter beschreibt, die mit einem M beginnen, gefolgt von einem a, gefolgt von einem n, gefolgt von einem g, gefolgt von einem o:

```
Mango
```

Syntax für reguläre Ausdrücke: "oder"

Wir können einen regulären Ausdruck schreiben, der die beiden Wörter `"Mango"` und `"Manga"` **zusammenfasst**. Der Ausdruck soll also die ersten 4 Buchstaben fest vorgeben und danach entweder ein `"o"` oder ein `"a"` erlauben. Der Ausdruck dafür sieht so aus:

```
Mang(o|a)
```

Dabei spielt die Reihenfolge der Elemente in den Klammern keine Rolle. Wir könnten stattdessen auch das hier schreiben:

```
Mang(a|o)
```

In beiden Fällen würden die zwei Wörter **gematcht**, die uns interessieren, nämlich `"Manga"` und `"Mango"`.

Wir können auch längere Sequenzen durch die Pipe (`|`) verknüpfen. So können wir die drei Wörter `"Manga"`, `"Mango"` und `"Manko"` beschreiben:

```
Man(ga|(g|k)o)
```

In diesem Ausdruck folgt nach den ersten drei Buchstaben entweder die Sequenz `"ga"`, oder eine Sequenz, die zum Teilausdruck `(g|k)o` passt. Der Teilausdruck beschreibt die Sequenzen `"go"` und `"ko"`. So werden durch den Gesamtausdruck alle drei Wörter abgedeckt.

Syntax für reguläre Ausdrücke: Start und Ende

Leider stimmen die Beispiele oben noch nicht so ganz, denn wir haben nicht explizit angegeben, dass die Wörter mit dem Start des regulären Ausdrucks beginnen und mit dem Ende enden müssen. So würde das Wort `"Mangold"` (eine gewöhnungsbedürftige Gemüsesorte) ebenfalls von unserem Ausdruck gematcht werden, weil `Mangold` keine der beschriebenen Bedingungen verletzt - es sind ja genau die Sequenzen enthalten, die enthalten sein müssen.

Wir können das Zeichen `^` verwenden, um den **Beginn eines Strings** zu markieren, und das Zeichen `$`, um das **Ende zu markieren**. So könnten wir ausdrücken, dass der reguläre Ausdruck Wörter nur beschreibt, wenn nach dem letzten Element des Ausdrucks keine weiteren Zeichen kommen:

```
^Mang(a|o)$
```

Syntax für reguläre Ausdrücke: Mengen

Statt einzelne Zeichen so wie oben zu "verodern", können wir im regulären Ausdruck eine **Menge von erlaubten Zeichen** angeben. Für das `"Manga"` / `"Mango"` -Beispiel ist das schön übersichtlich:

```
Mang[ao]
```

Dieser Ausdruck beschreibt alle Strings, in denen auf die vier Buchstaben am Anfang, die fest vorgegeben sind, ein Element aus der Menge `[ao]` folgt.

Mengen können auch **umfangreicher** sein, so wie im folgenden Beispiel, das **fünf englische Wörter auf einmal** beschreibt:

```
b[aeiou]d
```

Die beschriebenen Wörter sind `bad`, `bed`, `bid`, `bod`, und `bud`. Pro String kann nur **ein Zeichen aus der Menge** ausgewählt werden, deshalb wäre z.B. `"bead"` nicht durch diesen regulären Ausdruck abgedeckt.

Mengen können etwas abgekürzt werden, wenn man **Buchstaben oder Zahlen innerhalb eines bestimmten sortierten Bereichs** matchen möchte:

```
Grundschulklasse [1-4][a-c]
```

Auch Mengen sind **case sensitive**. Der folgende Ausdruck beschreibt fünf verschiedene Hepatitis-Arten:

```
Hepatitis [A–E]
```

Wenn im Impfpass allerdings von "Hepatitis b" die Rede ist, kann der reguläre Ausdruck diesen String nicht matchen, weil das Zeichen b nicht in der Menge `[A–E]` enthalten ist.

Achtung: Wir können auch die Menge `[a–z]` verwenden, um beispielsweise einen **beliebigen Kleinbuchstaben** zu erlauben. Umlaute sind in dieser Menge aber nicht enthalten, da sie im Alphabet nicht zwischen `a` und `z` stehen! Wir können zum Glück Mengen um zusätzliche Zeichen erweitern, dann sieht der Ausdruck so aus:

```
[a–zäöü]
```

Oder wenn auch Großbuchstaben erlaubt sein sollen:

```
[A–ZÄÖÜa–zäöü]
```

Syntax für reguläre Ausdrücke: Besondere Zeichen

Einige Arten von Zeichen, wie z.B. die Zahlen von 0 bis 9, bilden eine eigene Klasse von Symbolen. Für sie gibt es vordefinierte Sonderzeichen, mit denen reguläre Ausdrücke sich kompakter schreiben lassen:

Zeichen	Matcht auf...
`\d`	...alle Zahlen von 0 bis 9.
`\D`	...sämtliche Zeichen außer den Zahlen von 0 bis 9.
`\s`	...sämtliche Arten von Whitespace, unter anderem Leerzeichen, Tabs und Zeilenumbrüche.
`\S`	...sämtliche Zeichen außer Whitespace-Symbole.
`\w`	...alle alphanumerischen Zeichen; gleichbedeutend mit der Menge `[A–Za–z0–9_]`.
`\W`	...sämtliche Zeichen außer alphanumerischen Zeichen.
`^`	...den Anfang des Strings. Wird verwendet, wenn der reguläre Ausdruck nur am Anfang der Zeichenkette gematcht werden soll.
`$`	...das Ende des Strings. Wird verwendet, wenn der reguläre Ausdruck nur am Ende der Zeichenkette gematcht werden soll.
`.`	...genau ein beliebiges Zeichen. Achtung, kein Punkt!
`\.`	...den Punkt (`.`)

Syntax für reguläre Ausdrücke: Funktionale und nicht-funktionale Klammern

Die Syntax von regulären Ausdrücken belegt runde und eckige Klammern mit einer Spezialfunktion, nämlich dem Erzeugen von Gruppen und Mengen. Die folgende Tabelle zeigt, wie diese Funktionen einzusetzen sind und wie ein regulärer Ausdruck die Klammersymbole selbst als Schriftzeichen matchen kann.

Ausdruck	Funktion
`A`	Findet alle Vorkommen des Zeichens `A` im String.
`(A | B)`	Findet alle Stellen im String, die dem Zeichen `A` oder dem Zeichen `B` entsprechen.
`(...)`	Definiert eine Gruppe. Innerhalb von Gruppen werden Zeichenabfolgen in der vorgegebenen Reihenfolge gematcht. Die runden Klammern sind Teil der Syntax von regulären Ausdrücken und kommen nicht im gematchten String vor.
`\(` und `\)`	Matcht auf öffnende bzw. schließende runde Klammern im String.
`[...]`	Definiert eine Menge. Die Reihenfolge der Zeichen spielt keine Rolle. Jedes Vorkommen von einem der Zeichen in der Menge führt zu einem Match. Groß- und Kleinschreibung wird unterschieden!
`[A–Z]`	Definiert die Menge aller Großbuchstaben von A bis Z (beide inklusive).
`[e–h]`	Definiert die Menge aller Kleinbuchstaben von e bis h (beide inklusive).
`[^...]`	Definiert eine auszuschließende Menge. Ein Match findet an jeder Stelle des Strings statt, an der *keins* der Zeichen in der Menge steht.
`\[` und `\]`	Matcht auf öffnende bzw. schließende eckige Klammern im String.

Syntax für reguläre Ausdrücke: Quantifizierer für Teilausdrücke

Soll ein Teil eines regulären Ausdrucks optional sein oder mehrfach nacheinander vorkommen können, werden die folgenden Zeichen verwendet. Sie beziehen sich immer auf den direkt vor ihnen stehenden Teilausdruck, also auf das vorangegangene Zeichen, die vorangegangene Gruppe oder die vorangegangene Menge.

Ausdruck	Funktion
`+`	Voriges Element kommt **entweder einmal oder mehrmals** direkt nacheinander im String vor.
`*`	Voriges Element kommt **entweder nullmal oder mehrmals** direkt nacheinander im String vor.
`?`	Voriges Element kommt **entweder nullmal oder einmal** im String vor, ist also optional.
`{3}`	Voriges Element kommt **genau 3 mal** nacheinander im String vor.
`{3,5}`	Voriges Element kommt **zwischen 3 und 5 mal** nacheinander im String vor.
`{3,}`	Voriges Element kommt **mindestens 3 mal** nacheinander String vor.
`{,5}`	Voriges Element kommt **höchstens 5 mal** nacheinander im String vor.

Finden wir je einen String, der auf die folgenden regulären Ausdrücke matcht?

1. `ha{3,5}`
2. `\.\.\.`
3. `...`

Und finden wir jeweils einen regulären Ausdruck, der die folgenden Arten von Strings beschreibt?

1. Titel von "Drei ???"-Episoden
2. Matrikelnummern an der HHU
3. Die Vornamen aller Dozierenden und Tutor_innen im Pythonkurs

Zusammenfassung

- Reguläre Ausdrücke erlauben uns, Muster zu definieren und zu prüfen, ob ein String dem angegebenen Muster entspricht oder nicht.
- Einzelne Sonderzeichen und Spezialfunktionen von regulären Ausdrücken können kombiniert werden, um einen Gesamtausdruck zu entwerfen, der (falls nötig) hochkomplexe Bedingungen für Strings beschreibt.
- Auf regexpal.com können wir reguläre Ausdrücke entwerfen und per Hand testen.

Weitere Themen dieser Woche:

- 09-02: Reguläre Ausdrücke in Python matchen
- 09-03: Reguläre Ausdrücke und Gruppen