Einführung in das Programmieren

Reguläre Ausdrücke in Python

Christoph Draxler, Vanessa Reichel
draxler@phonetik.uni-muenchen.de
vanessa.reichel@campus.lmu.de

29. Mai 2024

Motivation

Wir kennen nun die wesentlichen Daten- und Kontrollstrukturen in Python:

- Variablen für den Zugriff auf Werte
- Funktionen als benannte Abläufe von Befehlen
- Fallunterscheidungen für verschiedene Abläufe in Programmen
- Schleifen zum wiederholten Ausführen von Befehlen
- Listen (`list` und `dictionary`) zum Speichern vieler Werte unter einem Namen
- Dateien erlauben das Speichern von Daten außerhalb von Programmen

Für das Arbeiten mit Text, ob in Dateien oder in Zeichenketten, sind *reguläre Ausdrücke* ein sehr nützliches Mittel.

Reguläre Ausdrücke: Einführung

Ein regulärer Ausdruck (engl. *regular expression*) ist ein *Suchmuster*.
Grundlage von regulären Ausdrücken ist die Kombination von *zeichenweisem Vergleich* und sog. *Metazeichen*, die quantifizierende Angaben über Zeichen oder Zeichenfolgen sind, oder für ganze Klassen von Zeichen stehen.
Technisch gesehen entsprechen reguläre Ausdrücke endlichen Automaten in der Informatik.

Einführendes Beispiel

Um reguläre Ausdrücke in Python zu verwenden, muss die Library `re` importiert werden.

```
import re

text = 'heute ist schönes Frühlingswetter'
ergebnis = re.search("^heute.*wetter$", text)
```

`re.` gibt an, dass eine Funktion aus der Library `re` verwendet werden soll, `search()` ist der Name der Funktion.
Was könnte diese Funktion prüfen oder berechnen?

Analyse des regulären Ausdrucks

`search("^heute.*wetter$", text)`

- `heute` und `wetter` sind Muster, die im Text exakt so vorkommen müssen
- in `.*` steht `.` für ein beliebiges Zeichen, und `*` für 'null oder mehr' Vorkommen
- `^` und `$` stehen für den Anfang bzw. das Ende des zu durchsuchenden Textes

Dieser reguläre Ausdruck prüft also, ob der Text mit 'heute' beginnt, dann eine beliebige Anzahl anderer Zeichen enthält und mit 'wetter' endet.

Was steht in `ergebnis`?

`ergebnis` ist ein sog. Match-Objekt, das zwei Zugriffsfunktionen und eine Eigenschaft besitzt:

`span()`	gibt ein Tupel mit der Anfangs- und Endposition des zum Muster passenden Textes zurück
`group()`	gibt den zum Muster passenden Text zurück
`string`	enthält den zu durchsuchenden Text

Wenn das Muster nicht zum Text passt, wird der Wert `None` zurückgegeben.

Beispiel

```
text = 'heute ist schönes Frühlingswetter'
ergebnis = re.search('sch.+es', text)

print(ergebnis.group())
print(ergebnis.span())
```

gibt `schönes` und das Paar `(10,17)` aus, d. h. das Wort 'schönes' geht vom 10. bis einschließlich 16. Buchstaben im Text.

Praktisches Beispiel

Die Funktion `eingabePruefen()` soll prüfen, ob eine Tastatureingabe einem Muster entspricht; wenn nicht, soll maximal n-mal eine neue Eingabe angefordert werden.

Praktisches Beispiel

Die Funktion `eingabePruefen()` soll prüfen, ob eine Tastatureingabe einem Muster entspricht; wenn nicht, soll maximal n-mal eine neue Eingabe angefordert werden.

```
def eingabePruefen (muster, n):
  passt = False
  i = 0
  while not passt and i < n:
    i = i + 1
    eingabe = input("Deine Eingabe: ")
    if re.search(muster, eingabe) != None:
      return (eingabe)
    else:
      print("Die Eingabe passt nicht zum Muster.")

  return("Keine gültige Eingabe!")
```

Metazeichen

`.`	irgendein Zeichen
`^`	Anfang des Textes
`$`	Ende des Textes
`*`	null oder mehrere Vorkommen
`+`	ein oder mehrere Vorkommen
`?`	null oder ein Vorkommen
`{n,m}`	mind. n, max. m Vorkommen
`|`	entweder – oder
`\`	schützt das nachfolgende Zeichen
`[...]`	Menge erlaubter Zeichen

9 + 1 Beispiele

Text: 'heute ist schönes Frühlingswetter'

1. Kommt 'in' im Text vor?
2. Kommt 'in' als Wort im Text vor?
3. Kommt 'heu' am Anfang des Textes vor?
4. Kommt eine Folge von 't' vor 'e' im Text vor?
5. Kommt 'h' direkt vor einem 'e' vor?
6. Kommt 'h' vor 'e' vor, und dazwischen liegt mindestens ein anderes Zeichen?
7. Kommt 'h' vor 'e' vor, und dazwischen liegt ein Paar 'tt'?
8. Kommt ein Wort mit exakt drei Buchstaben vor?
9. Beginnt ein Wort mit einem kleinen oder großen 'F'?
10. Wie sieht eine gültige Email-Adresse aus?

Entspricht die Ergebnisse Ihren Erwartungen?

Wie weit erstreckt sich ein Muster?

re.search('h.+e',text) sucht nach einem Muster, das mit einem 'h' beginnt und mindestens ein Zeichen vor einem 'e' hat. Wird nun das kürzeste Muster gefunden, ein beliebiges, oder das längste?

```
ergebnis = re.search('h.+e', text)
print(ergebnis.group())
```

Reguläre Ausdrücke finden das *längste* passende Muster, also 'heute ist schönes Frühlingswette'. Dieses Verhalten wird als *greedy*, also *gierig*, bezeichnet.

Reservierte Bezeichner

Zur einfachen Verwendung häufig benötigter Muster gibt es reservierte Bezeichner als Metazeichen:

$\backslash w$	Buchstaben, Ziffern und der Unterstrich _
$\backslash W$	kein Buchstabe, Ziffer, Unterstrich
$\backslash d$	Ziffern
$\backslash D$	keine Ziffern
$\backslash s$	Leerzeichen, Tabulator, usw.
$\backslash S$	kein Leerzeichen, Tabulator, usw.

Daneben gibt es noch weitere reservierte Bezeichner...

Weitere Funktionen

Neben `search()` gibt es noch weitere Funktionen mit regulären Ausdrücken:

`findall()` gibt *alle* Vorkommen eines Musters zurück

`split()` teilt eine Zeichenkette am Muster und gibt eine Liste von Zeichenketten zurück

`sub()` ersetzt ein Muster in einer Zeichenkette durch eine andere Zeichenkette

Reguläre Ausdrücke sind enorm praktisch, und sie sind, in teilweise leicht anderer Syntax, auch unabhängig von Python nützlich (z. B. bei Suchmaschinen, in Textdokumenten oder einem Terminal).

split()

split() teilt eine Zeichenkette am Muster, und das Muster ist standardmäßig nicht Teil des Ergebnis.

```
text = 'heute ist schönes Frühlingswetter'
seltsam = re.split('s', text)
print(seltsam)
```

ergibt ['heute i', 't ', 'chöne', ' Frühling', 'wetter']. Typischerweise verwendet man split(), um eine eingelesene Zeile einer Textdatei in Wörter oder die Felder einer Tabellenzeile zu zerlegen:

```
woerter = re.split('\s+', text)
```

`findall()`

`search()` gibt nur ein Vorkommen des Musters zurück (oder None), `findall()` alle Vorkommen in einer *Liste*.

```
text = 'astfreie flussschiffe fahren im schritttempo'
ergebnisse = re.findall("[sft]{3}", text)
for fundstelle in ergebnisse:
    print(fundstelle)
```

findet alle dreifachen Vorkommen von 's', 'f' oder 't' in beliebiger Reihenfolge und gibt sie aus: `'stf'`, `'sss'` und `'ttt'`.