IWGS1_WS1920_Zettel7

July 7, 2020

1 Informatische Werkzeuge in den Geistes- und Sozialwissenschaften I

1.1 Hausaufgabe 7 (Reguläre Ausdrücke, 100 Punkte)

Erschienen: 30.11.2019
Abgabe bis: 08.12.2019

Bitte laden Sie Ihre Notebooks bis 23:59 Uhr am Abgabetag in Ihrer Übungsgruppe bei StudOn hoch.

Wenn Ihnen einige der hier verwendeten Konzepte unbekannt sind oder Sie nicht wissen, wie Sie fortfahren sollen, können Sie die Vorlesungsunterlagen zu Rate ziehen oder jederzeit Fragen im Forum stellen, im Tutorium nachfragen, oder Ihrem Tutor eine Mail schreiben.

In dieser Übung werden wir uns weiter mit regulären Ausdrücken (= Regular Expression = RegEx) auseinander setzen. Um diese in Python überhaupt benutzen zu können ist es wichtig, dass Sie daran die `re`-Bibliothek mit folgendem Ausdruck zu importieren:

```
import re
```

In diesem Notebook ist das in der nächsten Zelle schon für Sie getan, führen Sie diese also auf jeden Fall aus, bevor Sie beginnen. Außerdem stellen wir Ihnen eine Funktion `validMatch(r,s)` zur Verfügung. Mit dieser können Sie einfach und unkompliziert überprüfen, ob ein String `s` *komplett* von einem RegEx `r` gematcht wird. Machen Sie sich auch mit dieser Funktion und den dazugehörigen Test vertraut.

[2]:
```
import re

# validMatch gibt nur "True" zurück, falls
# der _gesamte_ String auf den RegEx passt.
def validMatch(regex, string):
    if re.fullmatch(regex,string) is None:
        print("Kein Match!")
    else:
        print("Match!")

validMatch("e+se+l", "eeeseeel")   # Passt.
validMatch("e+se+l", "weeeseeel")  # Passt nicht, String startet anders.
```

```
print("-----")
validMatch("mat.*", "matterhorn") # Passt.
validMatch("mat.", "matterhorn")  # Passt nicht, String geht weiter.
validMatch("mat.", "cats")        # Passt nicht, String startet anders.
```

```
Match!
Kein Match!
-----
Match!
Kein Match!
Kein Match!
```

1.1.1 Aufgabe 7.1 (RegEx für einfache Postadressen, 15 Punkte)

Der größte Vorteil von regulären Ausdrücken ist, dass sich damit sehr leicht große Mengen Text auf bestimmte Muster durchsuchen lassen können. Dieser Umstand kann besonders in Textwissenschaften zum Tragen kommen. Eines der bekanntesten Beispiele im Internet für dieses Phänomen ist jedoch folgender **xkcd** Webcomic:

Wir werden in dieser Aufgabe versuchen, ein ähnliches Problem zu lösen (allerdings werden wir als Programmiersprache Python statt Perl verwenden). Wir werden nach Ausdrücken suchen, die aus einer (fünfstelligen) Postleitzahl und einem einfachen deutschen Ortsnamen bestehen.

Aufgabe: Weisen Sie der Variable `address` einen regulären Ausdruck zu, der auf Strings passt, die aus genau fünf Ziffern (PLZ) gefolgt von einem Leerzeichen, gefolgt von einem Ortsnahmen bestehen. Ein Ortsname ist für unsere Zwecke ein Wort mit genau einem großen Buchstaben am Anfang, gefolgt von mindestens einem kleinen Buchstaben. Der Einfachheit halber können Sie für diese Aufgabe annehmen, dass Ortsnamen niemals länger als 20 Zeichen sind (Gegenbeispiel), keine Interpunktion, Sonderzeichen oder Leerzeichen enthalten (Gegenbeispiel). Die Buchstaben `Ä`, `Ö`, `Ü` und `ß` kommen jedoch vor, die ersten drei potentiell auch am Anfang des Wortes.

Testen Sie Ihren RegEx mit den Tests in der übernächsten Codezelle.

[4]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
address = ""
```

[6]:
```
# Tests zur Überprüfung Ihres RegEx.

validMatch(address,"91058 Erlangen")          # Sollte übereinstimmen.
validMatch(address,"33649 Bielefeld")         # Sollte übereinstimmen.
validMatch(address,"85716 Unterschleißheim") # Sollte übereinstimmen
print("-----")
validMatch(address,"IWGS macht Spaß!")        # Sollte _nicht_ übereinstimmen.
validMatch(address,"33649 BIELEFELD")         # Sollte _nicht_ übereinstimmen.
validMatch(address,"Erlangen 91058")          # Sollte _nicht_ übereinstimmen.
validMatch(address,"123456789 Entenhausen")   # Sollte _nicht_ übereinstimmen.
```

```
Kein Match!
Kein Match!
Kein Match!
-----
Kein Match!
Kein Match!
Kein Match!
Kein Match!
```

1.1.2 Aufgabe 7.2 (RegEx für einfache Email-Adressen, 15 Punkte)

Nachdem nun Ädgar Ägyptologe allen seinen Kolleg*innen im boomenden Wirtschaftszweig der angewandten Ägyptologie von Ihren Fähigkeiten erzählt hat, wird Ihr E-Mail-Postfach in den letzten Tagen geradezu überschwämmt von Anfragen nach kostenloser Arbeit oder Arbeit für "Reichweite". Sie beschließen, eine strengere SPAM-Filter-Regel einzuführen, um sich diesen Andrang zu ersparen. Allerdings wollen Sie keine für Ihr Studium wichtigen E-Mails aus Versehen aussortieren. Sie beschließen also, dass Sie in Zukunft nur noch Emails von anderen Uni-Accounts annehmen werden.

Aufgabe: Schreiben Sie einen regulären Ausdruck (und weisen Sie ihn der Variable `email` zu), der nur auf Email-Addressen passt, die nach dem Muster `vorname.nachname@fau.de` oder `vorname.vorname.nachname@fau.de` aufgebaut sind. Vor- und Nachnamen bestehen aus Buchstaben.

Sie können der Einfachheit halber für diese Zwecke das Kürzel `\w` (identisch zum Ausdruck `[0-9a-zA-Z_]`) benutzen, auch wenn in tatsächlichen Namen selten Zahlen oder Unterstriche vorkommen. Sonderzeichen wie `[ÄäÖöÜüßéè...]` werden von `\w` nicht getroffen, diese sind allerdings in Email-Addressen unüblich, da diese klassischerweise auf den ASCII-Zeichensatz beschränkt sind.

Testen Sie Ihren RegEx mit den Tests in der übernächsten Codezelle.

[4]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle
↪auszuführen)
```

[5]:
```
# Tests zur Überprüfung Ihres RegEx.

#validMatch(email, "philipp.kurth@fau.de")             # Sollte übereinstimmen.
#validMatch(email, "jonas.betzendahl@fau.de")          # Sollte übereinstimmen.
#validMatch(email, "michael.erfunden.kohlhase@fau.de") # Sollte übereinstimmen.
print("-----")
#validMatch(email, "philipp.kurth@fau#de")                          # Sollte
↪_nicht_ übereinstimmen.
#validMatch(email, "jonas.betzendahl@gmail.com")                    # Sollte
↪_nicht_ übereinstimmen.
#validMatch(email, "michael.mehr.als.zwei.vornamen.kohlhase@fau.de") # Sollte
↪_nicht_ übereinstimmen.
```

```
-----
```

1.1.3 Aufgabe 7.3 (RegEx Gruppen, 40 Punkte)

In der nächsten Codezelle finden Sie in der Variable `gizmodo_heat` einen adaptierten Auszug aus "Brutal Heat Wave Breaks Temperature Records Across Europe", einem Artikel über die Hitzewelle im Sommer diesen Jahres. Darin kommen mehrere Angaben zu Temperaturen vor, welche allerdings alle in Grad Fahrenheit ("degrees Fahrenheit") gegeben sind, da der Artikel für ein US-Amerikanisches Publikum geschrieben wurde. Im Rahmen dieser Aufgabe werden Sie diese Temperaturangaben für besseres Verständnis in Europa in Grad Celsius umrechnen.

Zur Erinnerung, die Formel zur Umrechnung ist wie folgt: $[°C] = ([°F] - 32) \cdot \frac{5}{9}$.

Dabei ist wichtig zu bemerken, dass im Text auch Zahlen vorkommen, die keine Temperaturangaben darstellen (z.B. "2015" oder "1947"). Hier können *Gruppen* in RegExes hilfreich werden. Eine *Gruppe* in einem regulären Ausdruck ist ein Teil des Ausdrucks, der in runde Klammern gestellt wird. Wenn nun dieser RegEx für Suche im Text benutzt wird, wird nach wie vor auf den gesamten regulären Ausdruck abgeglichen, aber wenn am Ende ein Ergebnis produziert wird, werden nur die Teile des Ausdrucks, die in Gruppen standen, zurück gegeben.

Unser Beispiel dafür wird der RegEx `(.)at` sein. Dieser findet alle drei Zeichen langen Teilstrings, die auf `"at"` enden (`.` ist der RegEx für ein beliebiges Zeichen). Da wir aber eine Gruppe um das erste Zeichen gesetzt haben, wird nur dieses in der Ergebnisliste zu finden sein.

Beispiel:

```
# Ohne Gruppe.
re.findall(".at", "The cat with the hat on the mat wants a pat.") == ['cat','hat','mat','pat']

# Mit Gruppe.
re.findall("(.)at", "The cat with the hat on the mat wants a pat.") == ['c','h','m','p']
```

Basierend auf der Art und Weise, wie die meisten Programmiersprachen, darunter auch Python, mit Kommazahlen umgehen, kann es zu geringen Ungenauigkeiten kommen. Wir werden hier nicht genauer auf die technischen Details eingehen, aber wir wollen gerne lernen, Zahlen wie `1.5000000000000002` in den Ausgaben unserer Programme zu vermeiden. Eine Möglichkeit dafür ist die Funktion `round(z,n)`, die eine Zahl `z` auf `n` Nachkommastellen rundet.

Beispiel:

```
# 1.5000000000000002
print(3.2-1.7)

#1.5
print(round(3.2-1.7,2))
```

Aufgabe: Schreiben Sie ein Python-Programm, das den gegebenen Text mit einem regulären Ausdruck nach allen Temperaturangaben durchsucht. Achten Sie darauf, dass Ihr Ausdruck tatsächlich nur Temperaturen in Grad Fahrenheit trifft und keine Datumsangaben. Benutzen Sie dafür eine *Gruppe* in Ihrem Ausdruck. Konvertieren Sie alle Treffer in Zahlen und rechnen Sie diese in Grad Celsius um und geben Sie die Ergebnisse (auf zwei Nachkommastellen gerundet) aus.

[6]:
```
# Text adapted from "Brutal Heat Wave Breaks Temperature Records Across Europe"
```

```
# https://earther.gizmodo.com/
 ↪brutal-heat-wave-breaks-temperature-records-across-euro-1836695482
gizmodo_heat = """Germany saw its temperature break Wednesday as Geilenkirchen,␣
 ↪a town in the northwestern corner
                  part of the country near the Netherlands border, reached 104.
 ↪9 degrees Fahrenheit. The previous
                  record from July 5, 2015, was 104.5 degrees Fahrenheit,␣
 ↪according to the DWD German weather
                  service.

                  Next door in the Netherlands, record temperatures broke, too.␣
 ↪On Thursday, the municipality of
                  Gilze en Rijen reached 104.7 degrees Fahrenheit, shooting␣
 ↪past the previous record of
                  104 degrees Fahrenheit, the Royal Netherlands Meteorological␣
 ↪Institute reports. That previous
                  record was set just a day earlier.

                  In France, temperatures have been breaking nonstop throughout␣
 ↪cities Thursday, according to the
                  French meteorological service. Paris saw its mercury reach␣
 ↪108.6 degrees Fahrenheit; its
                  previous record was 104.7 degrees Fahrenheit set in 1947.␣
 ↪Lille in northern France near the
                  Belgium border broke its record - 99.7 degrees Fahrenheit␣
 ↪from just last year - at
                  104.9 degrees Fahrenheit. Officials are concerned the heat␣
 ↪could cause the collapse of the
                  fragile Notre Dame Cathedral, which almost burned to the␣
 ↪ground in April."""
```

[7]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
 ↪auszuführen)
```

1.1.4 Aufgabe 7.4 (RegEx Kreuzworträtsel, 30 Punkte)

Nachdem Sie Ihrer guten Freundin, Beatrice Beispiel, letzte Woche bereits mit ihren Problemen mit regulären Ausdrücken geholfen haben, kommt sie diese Woche erneut auf sie zurück. Dieses Mal nicht mit einer Bitte um Hilfe, sondern mit einer lustigen Freizeitbeschäftigung, die sie zwei zusammen genießen können, so sagt sie. Sie stutzen kurz, weil Sie sich innerlich schon auf mehr Arbeit vorbereitet hatten, aber lassen sich schnell von Freizeit überzeugen.

Beatrice stellt Ihnen RegEx-Kreuzworträtsel vor. Diese funktionieren ähnlich wie normale Kreuzworträtsel, mit dem Unterschied, dass die Hinweise keine Umschreibungen des Wortes sind, sondern reguläre Ausdrücke, die auf die Einträge in der jeweiligen Zeile oder Spalte matchen (Wenn diese Regeln unklar erscheinen, können Sie hier nachlesen).

Ein paar einfache Beispiele hat Beatrice schon selbst ausgefüllt:

Beispiel 1:

	A+
A|B	A

Beispiel 2:

	B*
[ABC]	B
A|B	B

Beispiel 3:

	A?	**B***	**C?E?D?**
[CAB]+	A	B	C

Sie beschleicht das nagende Gefühl, dass Beatrice diese Kreuzworträtsel doch nicht nur zur Unterhaltung mitgebracht hat. Statt aber die direkte Frage zu stellen, ob sie zufällig in naher Zukunft für einen ihrer Kurse ausgefüllte RegEx-Kreuzworträtsel abgeben muss, geben Sie sich der Aufgabe hin. Immerhin sieht es durchaus spaßig aus.

Aufgabe: Füllen Sie die folgenden RegEx-Kreuzworträtseltabellen so aus, dass alle an den Rändern gegebenen RegExe erfüllt werden. In jeder Zelle wird nur ein Zeichen erwartet.

Rätsel 1 (Beatles):

	[^SPEAK]+	**EP|IP|IF**
HE|LL|O+	L	O
[PLEASE]+	K	Q

Rätsel 2 (Ghost):

	[COBRA]+	**(AB|O|OR)+**
BO|QR|LL		
[^ABRC]+		

Rätsel 3 (Symbolism):

	.?.+	**.+**
[*]+		

	.?.+ .+
/+	

[ ]: