An Introduction to Regular Expressions in Python

Fabienne Braune[1]

[1]LMU Munich

May 29, 2017

Outline

Introductory Exercise (10 min.)

Consider the following variants of the German verb "sagen":

- sagen, sagt, sagte, gesagt, zugesagt

Write a function **matchSag()** that takes a string as argument and returns:

- true if the string contains one of the variants
- false if the string does not contain a variant

Er sagt mir immer guten Tag $\rightarrow$ true

Das hat er nie gesagt $\rightarrow$ true

Er hat wieder was interessantes berichtet $\rightarrow$ false

Introductory Exercise

How many lines of code does your program have?
Did you use any special module to implement your code?

Regular Expressions Basics

Regular Expressions

Regular Expressions (REGEX) describe patterns of text

- \d stands for digit
  → regex for 0162-787-334-229
  → \d\d\d\d-\d\d\d -\d\d\d-\d\d\d

- \w stands for word
  → regex for toy238
  → \w\w\w\w\w\w
  → \w\w\w\d\d\d
  → toy\d\d\d

Creating Regex Objects

Regex functions are in **re** module

→ **import re** at beggining of script

→ **re** class implements regex manipulation

Creating regex with **re**

- use **re** object to call functions that manipulate regex
- **re.compile()** creates a pattern to match

  → argument is string value describing regex

  → returns pattern object corresponding to regex

  → myphone=**re.compile(r'\d\d\d-\d\d\d-\d\d\d-\d\d\d')**
- **r** raw string no need to escape backslashes

Searching for Regex Objects

Searching for regex with **re**

myRegex.search() searches *myRegex* in given string

→ argument is string containing regex

- myphone=**re.compile(r'\d\d\d-\d\d\d-\d\d\d-\d\d\d')**

→ returns Match Object or **None**

- match if pattern is matched in string
  → match.**group()** returns matched text

myMatch=**myphone.search**('Meine Telefonnummer lautet 0162-787-334-229')

None=**myphone.search**('Meine Telefonnummer lautet 0162-WWSR-hh-2'))

Recap: Search for Phone Numbers in Strings

1. **import re**
2. myphone=**re.compile(r'\d\d\d-\d\d\d-\d\d\d-\d\d\d')**
3. match=**myphone.search**('Meine Telefonnummer lautet 0162-787-334-229'))
4. print(match.**group()**)
5. → 0162-787-334-229

Interactive Session

Let's search for birth dates in strings!

Grouping

Using groups subpatterns of regex can be printed

→ Create groups by inserting parentheses inside regex

- myphone=**re.compile(r'(\d\d\d\d)-(\d\d\d)-(\d\d\d)-\d\d\d')**
- match=**myphone.search**('Meine Telefonnummer lautet 0162-787-334-229'))

  print(match.**group(1)**) → 0162

  print(match.**group(2)**) → 787

  print(match.**group(3)**) → 334

  print(match.**group(0)**) → 0162-787-334-229

Matching Multiple Groups

Sometimes we want to match multiple groups

→ Match multiple groups with |

- mySagen=re.compile(r'sag\w\w | \w\wsag\w')
- mySagen.search('Er sagt ja, Sie sagen nein') → sagen
- mySagen.search('Sie haben nein gesagt') → gesagt

Match object contains first occurrence that matches regex

- mySagen.search('Sie sagen nein, er hat ja gesagt') → sagen
  → **findall()** returns list with all regex

Matching Multiple Groups

Convenient to match groups with same prefix

→ Use | and parentheses

- mySagen=re.compile(r'sag(te|en|t|ten)')
- mySagen.search('Er sagt ja, Sie sagen nein') → sagt
- mySagen.search('Sie sagten nein') → sagten
- mySagen.search('Sie haben nein gesagt') → sagt
- mySagen.findall('Er sagte ja, sie sagt nein und die anderen sagten vielleicht') → sagte, sagt, sagten

Optional Matching

Convenient to match groups optionally

- ? matches 0 or 1 occurrences
- * matches 0 to n occurrences
- + matches 1 to n occurrences

- mySagen=re.compile(r'(ge)?sag(te|en|t|ten)')
- mySagen.search('Er sagt ja, Sie sagen nein') → sagt
- mySagen.search('Sie haben nein gesagt') → gesagt

- mySagen=re.compile(r'(ge)*sag(te|en|t|ten)')
- mySagen.search('Sie haben nein gegegegesagt') → gegegegesagt

Optional Matching

Also works with single characters

- ? matches 0 or 1 occurrences
- * matches 0 to n occurrences
- + matches 1 to n occurrences

- mySagen=re.compile(r'sag(te?|en|ten)')
- mySagen.search('Er sagt ja, Sie sagen nein') → sagt
- mySagen.search('Er sagte ja, Sie sagten nein') → sagte

- mySagen=re.compile(r'sag(te*|en|ten)')
- mySagen.search('Er sagteeeeee nein') → sagteeeeee

Matching Repetitions

Useful to define number of matches

{1,4} matches 1 to 4 repetitions

- mySagen=re.compile(r'(ge){1,4}sag(te|en|t|ten)')
- mySagen.search('Sie haben nein gesagt') → gesagt
- mySagen.search('Sie haben nein gegegegesagt') → gegegegesagt
- mySagen.search('Sie haben nein gegegegegegegegesagt') → no match

Introductory Exercise (5 min.)

Consider the following inflexions of the German verb sagen:

- sagen, sagt, sagte, gesagt, zugesagt, gesagten, zugesagten, abgesagten

Using the regex module write a function **matchSag()** that takes a string as argument and returns:

- true if the string contains one of the variants
- false if the string does not contain a variant

Er sagt mir immer guten Tag $\rightarrow$ true

Das hat er nie gesagt $\rightarrow$ true

Er hat wieder was interessantes berichtet $\rightarrow$ false

Interactive Session (10 min.)

Let's try out your solutions!

Greedy and Non-Greedy Matching

Python regex greedy by default

- mySagen=re.compile(r'(ge){1,4}sag(te|en|t|ten)')
- mySagen.search('Sie haben nein gesagt') → gesagt
- mySagen.search('Sie haben nein gegegegesagt') → gegegegesagt
  → could also match gesagt but takes longest

Enable non-greedy mode with ?

- mySagen=re.compile(r'(ge){1,4}?sag(te|en|t|ten)')
- mySagen.search('Sie haben nein gesagt') → gesagt
- mySagen.search('Sie haben nein gegegegesagt') → gesagt
  → could also match gegegegesagt but takes shortest

Matching Beginning of String (only)

Use **match()** instead of **search()**

- mySagen=re.compile(r'(ge){1,4}sag(te|en|t|ten)')
- mySagen.match('Sie haben nein gesagt') → no match
- mySagen.match('gesagt hat er nichts') → gesagt
- mySagen.search('gegegegesagten') → gegegegesagten

Returning all Matched Strings

Match object returned by search() contains first matched string

What if all matched strings should be returned?

→ Use findall()

- mySagen=re.compile(r'(ge){1,4}sag(te|en|t|ten)')
- mySagen.search('gesagt oder nicht gegesagt') → gesagt
- mySagen.findall('gesagt oder nicht gegesagt') → gesagt,gegesagt
  → when multiple groups in regex, returns a list of tuples

Some Character Classes

\d	digit from 0 to 9
\D	not digit from 0 to 9
\w	word: letter, digit or underscore
\W	not word: letter, digit or underscore
\s	space: space, tab or newline
\S	not space: space, tab or newline

mystery=re.compile(r'\d+\s\w+')

What does mystery match? Give examples.

Creating Character Classes

The pre-defined character classes may be too broad

→ Define custom character classes with [ ]

- [aeiouAEIOU] any vowel
- [a-zA-Z0-9] all lowercased letters
- [0-5] ?

→ Negative classes are defined with ^

- What does [^aeiouAEIOU] match?
- What does [^0-5] match?

More Special Symbols

- ^ : match beginning of string

  beg=re.compile(r'^sag(te|en|t|ten)*')

  beg.search('sag hallo') → sag

- $: match end of string

  beg=re.compile(r'$sag(te|en|t|ten)*')

  beg.search('hallo sagen') → sagen

- . : match everything

  beg=re.compile(r'$sag(.)*')

  beg.search('hallo sagen') → sagen

  beg.search('hallo sagen sagen sagen') → sagen sagen sagen (greedy)

Regex symbols recap (10 min.)

What do the following symbols do?

- ?
- *
- +
- {n} {n,} {,m} {n,m}
- *? +? {n,m}?
- ^
- $
- .
- [abc] [^ abc]

Advanced Operations on Regex and Match Objects

String Modifications

Useful to modify string containing regex.

beg=re.compile(r'^sag(te|en|t|ten)*')

beg.search('sag hallo') → sag

beg.sub('schrei', 'sag hallo') → 'schrei hallo'

re.sub(r'^sag(te|en|t|ten)*','schrei', 'sag hallo') → 'schrei hallo'

- re.escape(p) : escape pattern
- re.purge() : empty regex cache

Advanced Grouping

Print list containing all groups with groups()

→ Returns list containing all matched groups

Non-capturing groups

- Match a group but don't save its content
  - → use ?: at beginning of group
  - → myMatch = re.match("(?:Asimov)+", "Isaac Asimov")
  - → myMatch.groups() is empty

Advanced Grouping

Named groups

- Match a group and save its name
  - → use ?P<abc> at beginning of group
  - → myMatch = re.match("(?P<abc>abc)+", "abc")
  - → myMatch.group('abc') returns abc

Positive and negative Lookahead

- → use ?= at beginning of group
- → myMatch = re.match("Isaac (?=Asimov)+", "Isaac Asimov")
- → myMatch.groups() is Isaac
- → only matches Isaac Asimov

- Negative lookahead with ?! at beginning of group

Advanced Grouping

Print dictionary containing all groups with their names

- → use groupdict()
- → myMatch = re.match("(?P<lemma>sag)(?P<suff>(te|ten))", "sagten")
- → myMatch.groupdict() returns lemma: sag, suffix: ten

Return span of matched group

- → use span(group)
- → myMatch = re.match("(?P<lemma>sag)(?P<suff>(te|ten))", "sagten")
- → myMatch.span(1) returns 0,2
- → myMatch.start(1) returns 0
- → myMatch.end(1) returns 2

Compilation Flags

re.compile(regex,flags=0)

Possibility to pass compilation flags

Flag	**Short**	**Description**
re.IGNORECASE	re.I	Case insensitive matching
re.MULTILINE	re.M	^ $ also match at beginning of newline
re.DOTALL	re.S	. matches any symbol (also newline)
re.VERBOSE	re.X	allows to insert comments and newlines
re.UNICODE	re.U	char sequences dependent on unicode

Exercise Session

Digit Manipulation

Create a function that:

1. Recognizes a date in YYYY-MM-DD format
2. Modifies YYYY-MM-DD into DD MM YYYY

Digit Manipulation: Solution

```
import re

def rewriteDate(s):
    res=re.search(r'(\d{4})-(\d{2})-(\d{2})')
    print res.group(3) + " " + res.group(2) + " " + res.group(1)
```

Digit and Word Manipulation

Given this list:

l = ["555-8396 Neu, Allison",
"Burns, C. Montgomery",
"555-5299 Putz, Lionel",
"555-7334 Simpson, Homer Jay"]

Write a function that transforms the list into:

Allison Neu 555-8396
C. Montgomery Burns
Lionel Putz 555-5299
Homer Jay Simpson 555-7334

Digit and Word Manipulation: Solution

```
import re

def rewriteList(l):
    for i in l:
        res = re.search(r'([0-9-]*)\s*([A-Za-z]+),)\s+(.*)', i)
        print res.group(3) + " " + res.group(2) + " " + res.group(1)
```

Word Manipulation

Given a list of words write regexes to:

1. Find all words that include four consecutive vowels
2. Find every word with 5 repeat letters
3. Find all words beginning with "sag"
4. Find all words containing the segment "sag"
5. Remove repeat letters

Make a function that creates an acronym from a phrase.

Interactive Session

Let's try out your solutions!