Reguljära Uttryck

Pontus Haglund
Institutionen för datavetenskap (IDA)
Anpassat från material ursprungligen av: Eric Elfving

3 oktober 2018

LINKÖPINGS UNIVERSITET

Översikt

- Introduktion
- Syntax
- Användning i Python

Introduktion

- Reguljära uttryck används för att matcha ett mönster (pattern) mot en text.
- Kallas ofta regex eller regexp (från regular expression)
- Skapades på 50-talet och finns tillgängligt i de flesta språk.
- Ett väldigt bra verktyg som kan användas till mycket.

Syntax

Ett tecken matchar sig själv

Mönster:	`a`	`han`	`kul`
Söksträng:	`a anna`	`han hon hanna`	`kul kulle ful`

Syntax

Med hakparenteser ([ ]) skapas en teckenklass

En teckenklass matchar ett av tecknen innanför hakarna

Mönster: `a[ab]c`
Sökträng: `abc aac acabca`

Syntax

Teckenklasser

Man kan även sätta ett intervall inom hakparenteser

Mönster: `a[a-c]c`
Sökträng: `abc aac acc`

Syntax

Med cirkumflex (`^`) tar du inversen av en teckenklass

Man matchar då alla tecken utom den givna teckenklassen

Mönster: `a[^ab]c`
Sökträng: `adc aqc aaca`

Syntax

Med lodstreck (|) anger du ett logiskt ELLER

Mönster:	a | c	aa | ab
Sökträng:	abc	acaab abc

Syntax

Vanliga parenteser skapar en "grupp

Grupper kan användas för att ändra operatorers räckvidd och prioritering

Mönster: `a(aa|ab)c`

Söksträng: `aabc aaac acaabca`

Syntax

Upprepningar

Följande operatorer skrivs direkt efter det som ska upprepas.

- `?` Matchar 0 eller 1
- `*` Matchar 0 eller flera
- `+` Matchar minst 1

Mönster:	`colou?r`	`a[nl]+a`	`S(oe?|ö)der`
Söksträng:	`color colour`	`anna alla`	`Söder Soder`

Syntax

Upprepningar

- Med klammerparenteser `{}` upprepar man saker ett visst antal gånger.
- `x{n, m}` matchar `x` upprepat `n` till `m` gånger.
- Precis som i Python kan man utelämna `m`

Mönster:	`[0-9]{2,3}`	`[a-zA-z]{5}`	`[a-zA-z]{4,}`
Söksträng:	`17a 5262`	`Linkoping`	`Linköpings Unive`

Syntax

Speciella teckensekvenser

Teckensekvens	Motsvarighet
\w	[A-Za-z0-9_]
\W	[^A-Za-z0-9_]
\s	Vita tecken
\S	Inte vita tecken
\d	[0-9]
\D	[^0-9]

Mönster: `\d+`

Sökträng: `123a`

Syntax

Bakåtreferenser

Alla grupper tilldelas ett nummer (indexeras från 0). Med `\N` refererar man tillbaka till grupp nummer `N`.

Mönster: `(\d)a\0`
Sökträng: `0a0 1a6 4a4`

Syntax

Några speciella tecken

Följande tecken har speciell betydelse:

Tecken	Matchar
`.`	Valfritt tecken (utom newline)
`^`	Början av rad
`$`	Slut på rad eller fil

Mönster:	`^ka..e$`	`^ka..e$`
Sökträng:	`kalle`	`kalle svensson`

Syntax

Escapesekvenser

För att matcha någon av de speciella tecken (metatecken) som nämnts tidigare (`{}[]()^$.|*+?\`) används ett backslash:

Mönster:	`\d\.\d`	`\\a`	`\{\w+\}`
Sökträng:	`1.2 4a4`	`\a`	`{hej}`

Användning i Python

- Python har, med stöd av sitt stora bibliotek, så klart stöd för regex.
- Importera bara modulen `re`!

`http://docs.python.org/3/library/re.html`

Användning i Python

- re har tre vanligt använda funktioner (och så klart fler):
  - `re.match(pat,str)` Söker efter mönstret `pat` i början av strängen `str`
  - `re.search(pat,str)` Söker efter första förekomsten av mönstret `pat` i strängen `str`
  - `re.findall(pat,str)` Hittar alla förekomster av mönstret `pat` i strängen `str`
- De två första ger ett `"match object"` vid matchning, annars `None`. `findall` ger en lista av matchande strängar.

Användning i Python


```
import re
val = input(Mata in ett heltal: ')
while re.match('^[0–9]+$', val) is None:
  val = input('Felaktig inmatning, nytt heltal: ')

val = int(val)
```

Användning i Python

Det finns ett problem med reguljära uttryck... backslash! Backslash används ju redan för escapesekvenser i Python. Tänk er att vi ska söka efter texten `\begin`. Då måste vi skriva såhär:

```
import re
match = re.search('\\\\begin', str)
```

Vi vill söka efter ett \, då behövs ett extra enligt regex-syntax samt ett extra för varje backslash för att inte Python ska tolka det som en escapesekvens!

Användning i Python

Detta problem löses lättast med "råsträngar (raw string)

```
import re
match = re.search(r'\\begin', str)
```

Användning i Python

Match objects

Grupper i mönstret kan tas fram i efterhand med funktionen `group`:

```
import re
match = re.search(r'(\d+) (\w+)', str)
if match is not None:
  print(match.group(0)) # Skriver ut hela matchningen
  print(match.group(1)) # Skriver ut första gruppen
  print(match.group(2))
```

Om en grupp var valbar (hade ett frågetecken) kan den ha värdet `None`.

Användning i Python

findall

```
>>> str="""En sträng som ibland har heltal i sig: 12345
... Den kan vara lite 2313 olika lång. Denna sträng
... är 3 rader lång!"""

>>> lst = re.findall(r'\d+', str)
['12345', '2313', '3']

>>> [int(x) for x in lst]
[12345, 2313, 3]

>>> sum(map(int, lst))
14664
```

Användning i Python

Namngivna grupper

I Python kan man namnge sina grupper:

```
>>> import re
>>> text="'Namn: Anna
... Ålder: 32"'
>>> for m in re.finditer(r'(?P<key>\w+): (?P<val>\w+)', str):
...   print(m.group('key'), m.group('val'), sep=' => ')
...
Namn => Anna
Ålder => 32
```

Läs vidare

`http://en.wikipedia.org/wiki/Regular_expression` Wikipedia

`http://www.regular-expressions.info/` En samling med tutorials, verktyg och guider

`https://docs.python.org/3/howto/regex.html` Mer om regex i Python

`http://www.regexr.com/` En bra testare av dina uttryck

Uppgifter

1. Ett LiU-ID skrivs på formatet FEN där F är de tre första bokstäverna i förnamnet, E de två första i efternamnet och N är tal med två eller tre siffror (beroende på om det är en anställd eller student). Om förnamnet eller efternamnet endast är två tecken långt blir den delen endast två tecken. Skriv ett mönster för att fånga ett LiU-ID.
2. Ett datum i ISO-format skrivs YYYY-MM-DD. Skriv ett mönster för att fånga detta.
3. Utöka ditt regexp för att kontrollera datumets giltighet (full kontroll går inte då vi saknar logik men t.ex. är en månad som inleds med 9 felaktig).
4. Skriv ett Python-program som (med hjälp av reguljära uttryck) hittar och skriver ut alla html-element i en given fil (på hårddisken eller url, välj själv).

www.liu.se

LINKÖPINGS UNIVERSITET