

IN1140: Introduksjon til språkteknologi

Forelesning #4

Samia Touileb

Universitetet i Oslo

07. september 2020



► Regulære uttrykk



Regular expressions¹ (called **REs**, or **regexes**, or **regex** patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the `re` module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as “Does this string match the pattern?”, or “Is there a match for the pattern anywhere in this string?”. You can also use **REs** to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given **RE**, and write the **RE** in a certain way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching engine's internals.

¹<https://docs.python.org/3/howto/regex.html>

- ▶ Regulære uttrykk:
 - ▶ Standard notasjon for å karakterisere tekst sekvenser
 - ▶ Blir brukt for å spesifisere tekst strenger i all slags type tekst processing og informasjonsutvinning
- ▶ Regulære uttrykk kan også bli implementert ved bruk av endelige tilstandsmaskiner (Finite-State Automata, FSA)
 - ▶ Veldig viktig i NLP
 - ▶ Har mange variasjoner: finite-state transducers, hidden Markov models, N-gram grammars ...
 - ▶ Vi skal ikke se på FSA

- ▶ Tokenisering
- ▶ Tekstprosessering (finne, erstatte)
 - ▶ Finne alle telefonnumre i en tekst, feks:
 - ▶ *Du kan også ringe Kundeservice på **815 22 040***
 - ▶ *Du kan også ringe Kundeservice på **815 22 041***
 - ▶ *Du kan også ringe Kundeservice på **815 22 501***
 - ▶ Finne flere tilgrensende instanser av samme ord i en tekst
- ▶ Bestemme språket i en setning/tekst: spansk eller polsk?
 - ▶ *Sytuacja na Bliskim Wschodzie jest napieta, szczególnie po wczorajszym ataku*

- ▶ Validere felter i en database (datoer, e-postadresser, URL'er)
- ▶ Søk i et korpus etter lingvistiske mønstre
 - ▶ samle statistikk
- ▶ Tildele ordklasse til disse, selv om de ikke fins i ordboken. F.eks.:
 - ▶ *conurbation, cadence, disproportionality, Thatcherization*

- ▶ Utstrakt bruk innenfor:
 - ▶ informasjonsshenting ("information extraction"), f.eks. navn på personer og firmaer
 - ▶ automatisk morfologisk analyse

- ▶ Et regulært uttrykk er en beskrivelse av en mengde strenger
- ▶ Finnes en rekke UNIX-verktøy (grep, sed), editorer (emacs) og programmeringsspråk (perl, python, java) som har funksjonalitet for regulære uttrykk
- ▶ Som alle formalismer har ikke regulære uttrykk noe språklig (lingvistisk) innhold, men kan snarere brukes til å referere til lingvistiske enheter

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

- ▶ Riktig dato?
- ▶ Hva er formatet?
 - ▶ tall: 0-9
 - ▶ mellomrom: \s
 - ▶ ord (sekvens av bokstaver): [A-Za-z]
 - ▶ mellomrom: \s
 - ▶ tall: 1900-2099

regex $\rightarrow [0 - 9] + \backslash s[A - Za - z] + \backslash s((19|20)[1 - 9]+)$

Fanger den kun riktige strenger?

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

regex \rightarrow $[[0 - 9] + \backslash s[A - Za - z] + \backslash s((19|20)[1 - 9] +)$

Fanger følgende:

- ▶ 6 januar 1992
- ▶ 3342 november 1988
- ▶ 9 fredag 1999
- ▶ 1 mars 2018

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

- ▶ Hva er formatet?
 - ▶ tall: 0-9
 - ▶ mellomrom: \s
 - ▶ ord (sekvens av bokstaver): (januar|februar|mars|april|mai|juni|juli|august|september|oktober|november|desember)
 - ▶ mellomrom: \s
 - ▶ tall: ((19|20)[0-9]+)

regex \rightarrow $[[0 - 9] + \backslash s$
(januar|februar|mars|april|mai|juni|juli|august|september|
oktober|november|desember)\s ((19|20)[0 - 9]+)

Fanger den kun riktige strenger?

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

*regex \rightarrow $[[0 - 9] + \backslash s$
(januar|februar|mars|april|mai|juni|juli|august|september|
oktober|november|desember)\backslash s ((19|20)[0 - 9] +)*

Den fanger:

- ▶ 6 januar 1992
- ▶ 25 juni 2005
- ▶ 3342 november 1988
- ▶ 1 mars 2018
- ▶ 30 juli 190221

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

► Hva er formatet?

- tall: 1-9, 1 etterfulgt av 0-9, 2 etterfulgt av 0-9, 3 etterfulgt av 0 eller 1
- mellomrom: \s
- ord (sekvens av bokstaver): (januar|februar|mars|april|mai|juni|juli|august|september|oktober|november|desember)
- mellomrom: \s
- tall: (19|20) etterfulgt av maks 2 tall 0-9

regex \rightarrow $[[([1-9]|([12][0-9]|3[01]))\s(januar|februar|mars|april|mai|juni|juli|august|september|oktober|november|desember)\s((19|20)\d{2})]$

Fanger den kun riktige strenger?

liste = ['6 januar 1992', '25 juni 2005', '3342 november 1988', '9 fredag 1999', '1 mars 2018', '30 juli 190221']

*regex \rightarrow $[([1-9] | [12][0-9] | 3[01]) \backslash s$
 $(januar|februar|mars|april|mai|juni|juli|august|september|$
 $oktober|november|desember) \backslash s ((19|20) \backslash d \{2\})$*

Den fanger:

- ▶ 6 januar 1992
- ▶ 25 juni 2005
- ▶ 1 mars 2018

To slags tegn (“characters”):

- ▶ **Bokstaver**

- ▶ ethvert teksttegn er et RU og refererer til seg selv

- ▶ **Meta-tegn**

- ▶ spesialtegn som lar deg kombinere RU på forskjellige måter

- ▶ Eksempel:

- ▶ /a/ refererer til a
 - ▶ /a*/ refererer til ϵ (null) eller a eller aa eller aaa eller ...

- ▶ MERK: // (i feks /a/) er **ikke** en del av et RU. Brukes som notasjon for å gjøre det tydelig hva er RU og hva ikke er RU

Regulære uttrykk består av:

- ▶ Strenger bestående av tegn: `/b/`, `/IN1140/`, `/informatikk/`
- ▶ Disjunksjon:
 - ▶ vanlig disjunksjon: `/spise|ete/`, `/penge(r|ne)/`
 - ▶ tegnklasser: `/[Dd]en/`, `/m[ae]nn/`, `/bec[oa]me/`
 - ▶ rekker ("ranges"): `[A-Z]`, `[a-z]`, `[0-9]`
- ▶ Negasjon:
 - ▶ Bruk av `^`
 - ▶ `[^b]`
 - ▶ `[^A-Z0-9]`
 - ▶ `a\b` ser etter sekvensen "`a^b`" i en streng
 - ▶ feks: "se etter **a^b** nå"
 - ▶ `e\^` ser etter sekvensen "`e^`" i en streng
 - ▶ feks: "se etter **e^** nå"

Regulære uttrykk består av (forts.):

- ▶ Tellere

- ▶ opsjonalitet (0 eller 1): ?
 - ▶ /woodchucks?/ (fanger begge "woodchucks" og "woodchuck")
 - ▶ /colou?r/ (fanger begge "colour" og "color")
- ▶ hvilket som helst antall (0 eller flere): Kleene *
 - ▶ /baaa*!/ (fanger "b!", "ba!", "baa!", "baaa!", "baaaa!", "baaaaaaaaaaaaa!")
 - ▶ /[0-9][0-9]*/
- ▶ Minst en: +
 - ▶ /baaa*!/ (fanger "ba!", "baa!", "baaa!", "baaaa!", "baaaaaaaaaaaaa!")
 - ▶ /[0-9]+ kroner/

Regulære uttrykk består av (forts.):

- ▶ “wildcard” for et hvilket som helst tegn: .
 - ▶ /beg.n/ (alt som finnes mellom *beg* og *n*)
 - ▶ “begin”
 - ▶ “beg’n”
 - ▶ “begun”
 - ▶ brukes ofte sammen med stjerne: “hva som helst”:
 - ▶ /beltedy.*r*.beltedy.*r*/
 - ▶ “**beltedy**r” er en familie av gomlere som er i utgangspunktet en ren søramerikansk gruppe, og Sør-Amerika pluss de sørlige delene av Nord-Amerika er de eneste stedene hvor **beltedy**r finnes vilt i dag”

Regulære uttrykk består av (forts.):

Ankere: spesielle tegn som forankrer det regulære uttrykket til spesifikt sted i strengen/teksten

- ▶ `^` - begynnelsen av linjen
 - ▶ `/^Den/` matcher “Den” bare i begynnelsen av en linje
 - ▶ `^` har tre mulige bruk:
 - ▶ Matche begynnelsen av en linje
 - ▶ Indikerer negasjon innefor firkantede parenteser (`[]`) altså tegnklasser
 - ▶ Bare karakteren `^`
- ▶ `$` - slutten av linjen
 - ▶ `/der\.$/` matcher “der.” på slutten av en linje
- ▶ `/^Det hvite huset\.$/ ?`
 - ▶ Matcher kun en linje som inneholder frasen “Det hvite huset.”

Regulære uttrykk består av (forts.):

- ▶ Ankere `\b` og `\B`
- ▶ `\b` matcher en “word boundary” (altså ordgrense)
 - ▶ `/\bthe\b/` matcher ordet “the” men ikke “other”
- ▶ `\B` matcher en “non-boundary”

Disjunksjon, gruppering, og presedens:

- ▶ Ser etter “katter” og “hunder” i en tekst:
 - ▶ Kan ikke bruk “[” og “]” for å søke for “katter **eller** hunder”
 - ▶ Må bruke **disjunksjon**: `/katter|hunder/`
- ▶ Ser etter “guppy” og “guppies”:
 - ▶ Kan vi si `/guppy|ies/`?
 - ▶ Nei! Vil matche strengene “guppy” og “ies”
 - ▶ Sekvensen guppy har **presedens** over disjunksjonen “|”
 - ▶ Må derfor bruke “(” og “)” , (**gruppering**)
 - ▶ `/gupp(y|ies)/`

Kleene (cleany) * operator:

- ▶ Vi vil fange repetisjoner av en instans, feks: Kolone 1 Kolone 2 Kolone 3.
- ▶ Hva vil `/Kolone[0-9]+*/` fange?
 - ▶ “Kolone 1” etterfulgt av uendelig mange mellomrom (feks. “Kolone 1...”)
- ▶ Må bruke “()”:
 - ▶ `/(Kolone[0-9]+)*`
- ▶ Her har “()” presedens over “*”

Kleene (cleany) * operator:

- ▶ Vi har $/[a-z]^*/$
- ▶ Setningen “once upon a time”
- ▶ Hva vil $/[a-z]^*/$ fange?
 - ▶ ingenting? “o”? “on”? “onc”? “once”? “once_□”? “once_□u”? ...
 - ▶ “once”
- ▶ RU matcher alltid lengste strengen, de er kalt **greedy**

Presedens av RU operatorer:

1	Parentes	()
2	Tellere (counters)	+ - ? {}
3	Sekvenser og ankere	the ^my end\$
4	Disjunksjon	

- Nyttige forkortelser:

RU	=	Eksempel
<code>\d</code>	<code>[0-9]</code>	Lag av <u>5</u>
<code>\D</code>	<code>[^0-9]</code>	<u>B</u> lå himmel
<code>\w</code>	<code>[A-Za-z0-9_]</code>	<u>D</u> ag
<code>\W</code>	<code>[^A-Za-z0-9_]</code>	<u>!</u> !!!!
<code>\s</code>	<code>[\r\t\n]</code>	
<code>\S</code>	<code>[^\s]</code>	<u>P</u> å Fløyen

- Nyttige forkortelser:

RU	Matcher
*	null eller flere
+	en eller flere
?	null eller en
.	wildcard, matcher hva som helst
{ <i>n</i> }	<i>n</i> forekomster
{ <i>n</i> , <i>m</i> }	fra <i>n</i> til <i>m</i> forekomster
{ <i>n</i> ,}	minst <i>n</i> forekomster

- Det som trenger “backslash”:

RU	Eksempel
*	I_*N*1*1*4*0
\.	Dr. Nesehorn
\?	Dr. Nesehorn?
\n	ny linje
\t	tab

- Men også /\[/, og /\[/ ...

Vi vil lage en app som hjelper en bruker med å kjøpe en PC på nettet. Brukeren vil ha "hvilken som helst maskin med minst 6 GHz og 500 GB av minne for under \$1000". Hvordan kan vi skrive et regulært uttrykk (RU) som kan fange dette?

Først, lage et RU for å fange dollar-tegn etterfulgt av en streng av tall:

- ▶ `/\${0-9}{1,4}/`
- ▶ `/\${0-9}{1,4}\.[0-9][0-9]/`
- ▶ `/\b\${0-9}{1,4}(\.[0-9][0-9])?\b/`

Til slutt, tenke på GB:

- ▶ `/\b[0-9]{3,}\s*(GB|[Gg]igabytes?)\b/`
 - ▶ Fanger denne minst 500GB regelen?

- ▶ **Et eksempel: Finne alle forekomster av *og* i en tekst**
 - ▶ `/og/`
 - ▶ Problem: *Og det ble kveld, og det ble morgen ...*
 - ▶ `/[Oo]g/`
 - ▶ Problem: *Togets rutetider*
 - ▶ `/\s[Oo]g\s/` eller `/\b[Oo]g\b/`
- ▶ Minimere antall falske positive (uriktig matchet, feks *Toget*) og falske negative (uriktig utelukket, feks *Og*)

- ▶ Substitusjon (perl format):
 - ▶ `s/color/colour/`
 - ▶ `s/versjon\s2\.4/versjon\s2\.5/`
- ▶ Substitusjon (python format):
 - ▶ `re.sub("color", "colour", text)`
 - ▶ `re.sub("versjon 2.4", "versjon 2.5", text)`

- ELIZA (Weizenbaum 1966):

User:	<i>Men are all alike</i>
ELIZA:	IN WHAT WAY?
User:	<i>They're always bugging us about something</i>
ELIZA:	CAN YOU THINK OF AN EXAMPLE?
User:	<i>Well, my boyfriend made me come here</i>
ELIZA:	YOUR BOYFRIEND MADE YOU COME HERE.
User:	<i>He says I am depressed much of the time</i>
ELIZA:	I'M SORRY TO HEAR THAT YOU ARE DEPRESSED.

```
s/. * I am (depressed|sad) . *  
/ I am sorry to hear that you are \1/  
s/. * (all|always) . *  
/ Can you think of an example /
```




- ▶ Beskriv strengene som aksepteres av følgende regulære uttrykk
 - ▶ `/ab+a/`
 - ▶ `/(ab)*/`
 - ▶ `/([~aeiou][aeiou])/`
 - ▶ `/\sdis[a-z]+\s/`