

# Regulære Udtryk i PHP

Jonas Holbech og Martin Elsmann  
IT Universitetet i København\*

27. februar 2008

## Resumé

I denne note gives en introduktion til brugen af regulære udtryk, også kaldet mønstre, i PHP. Notens beskriver hvordan regulære udtryk bruges til at klassificere strenge og derved kan bruges til at sikre at brugerinput følger en specificeret form og til at identificere—og derved gøre brug af—en specificeret del af en streng.

Konkret introduceres læseren til PHP-funktionen `ereg`.

## 1 Introduktion

Regulære udtryk bruges til at klassificere tekststrenge. Uden brug af regulære udtryk kan vi skrive PHP-kode som undersøger om to strenge er identiske (består af de samme tegn). Følgende PHP-kode undersøger for eksempel om en variabel `$email` indeholder strengen `holbech@itu.dk`, strengen `mael@itu.dk` eller en helt tredje streng:

```
if ( $email == 'holbech@itu.dk' ) {  
    // gør noget  
} elseif ( $email == 'mael@itu.dk' ) {  
    // gør noget andet  
} else {  
    // gør noget tredje  
}
```

Denne løsning virker fint så længe vi arbejder med en begrænset, foruddefineret, mængde data. Hvis vi derimod arbejder med input fra brugere, må vi ofte understøtte en større mængde af inputmuligheder, såsom alle strenge som tager form af et cpr-nummer, alle strenge som tager form af en emailadresse, og så videre. Som vi skal se i det følgende giver regulære udtryk, også kaldet mønstre, mulighed for, meget præcist at specificere hvilken form for data der skal accepteres af programmet.

---

\*Adresse: Rued Langgaards Vej 7, 2300 København S, Danmark. Email: `holbech@itu.dk` og `mael@itu.dk`.

## 1.1 Hvorfor bruge regulære udtryk

Regulære udtryk er nyttige til en lang række anvendelser i forbindelse med udvikling af webapplikationer:

1. **Validering af brugerinput.** Det er som hovedregel nødvendigt at validere input fra brugere for at undgå ikke-autoriseret eksekvering af databaseforespørgsler på serveren. Herudover giver validering af brugerinput mulighed for at hjælpe brugere med præcise fejlmeddelelser. Hvis for eksempel en webapplikation beder en bruger om en emailadresse til at sende en adgangskode til, kan applikationen, ved brug af regulære udtryk, undersøge om brugerens indtastning tager form af en emailadresse. På samme måde kan en webapplikation sikre at postnumre tager form af postnumre, at cpr-numre tager form af cpr-numre, og så videre.

Bemærk dog at regulære udtryk ofte kun kan benyttes til at undersøge om data er velformet og ikke om data konkret giver mening. Eksempelvis giver regulære udtryk ikke mulighed for at undersøge om en email-adresse konkret eksisterer, at en person med et givet cpr-nummer eksisterer eller om et postnummer eksisterer. På samme vis er det vanskeligt med regulære udtryk præcist at undersøge om en angivet dato eksisterer (f.eks. er det vanskeligt at specificere at datoen 2007-02-29 ikke er en eksisterende dato, men at datoen 2004-02-29 er eksisterende. Det er dog muligt at bruge regulære udtryk til at undersøge om data følger ISO-formatet YYYY-MM-DD.

2. **Anvendelse af data fra fremmede websider.** En webapplikations anvendelse af data fra fremmede websider (også kaldet webscraping) er ofte let at implementere ved brug af regulære udtryk til præcist at specificere hvilke dele af en web-side applikationen vil gøre brug af. Det er således muligt med regulære udtryk at finde, og gøre brug af, data der allerede er publiceret på Internettet. Eksempler på sådan data inkluderer valutakurser, vejrprognoser, nyheder og aktiekurser.
3. **Søgning efter mønstre i datamængder.** Regulære udtryk kan med fordel benyttes til at søge efter data i begrænsede datamængder, såsom strenge af begrænset størrelse. Regulære udtryk kan også bruges til at søge efter relevante rækker i en databasetabel; eksempelvis understøtter MySQL brug af regulære udtryk i WHERE-betingelser.
4. **Tekstsubstitution i strenge.** Regulære udtryk kan også bruges til udskiftning af præcist specificerede delstrenge i en streng (søg-og-erstat). En sådan anvendelse af regulære udtryk kan for eksempel bruges til at generere specifikke email-beskeder ud fra en template, hvori modtagerens navn er refereret til som NAVN.

## 2 Mønstre og Matching

I det følgende benytter vi os ofte af begrebet “mønster” (engelsk: pattern) i stedet for det noget længere “regulære udtryk”. I dette afsnit introducerer vi et sprog for mønstre. Sproget defineres på baggrund af en relation, kaldet *matching*, der definerer klassen af strenge

---

$p$	Definition
.	matcher alle karakterer
$c$	matcher karakteren $c$
$\backslash c$	matcher den escapede karakter $c$ , hvor $c$ er en af <code> , *, +, ?, (, ), [, ], \$, ., \, t, n, v, f, r</code>
$p_1 p_2$	matcher en streng $s$ hvis $p_1$ matcher et prefix af $s$ og $p_2$ matcher resten af $s$ (ex: strengen <code>abc</code> matches af mønsteret <code>a.c</code> )
$p^*$	matcher 0, 1 eller flere instanser af mønsteret $p$ (ex: strengene <code>abbbbba</code> og <code>aa</code> matches af mønsteret <code>ab*a</code> )
$(p)$	matcher strenge der matcher $p$ (ex: strengen <code>cababcc</code> matches af mønsteret <code>c(ab)*cc</code> )
$p^+$	matcher 1 eller flere instanser af mønsteret $p$ (ex: mønsteret <code>ca+b</code> matcher strengen <code>caaab</code> , men ikke strengen <code>cb</code> )
$p_1   p_2$	matcher strenge der matcher enten $p_1$ eller $p_2$ (ex: mønsteret <code>(pig cow)</code> matcher strengene <code>pig</code> og <code>cow</code> )
$[class]$	matcher en karakter i $class$ ; se nedenfor for en definition af karakterklasser $class$ . Mønsteret <code>[abc1-4]*</code> matcher sekvenser af karaktererne <code>a, b, c, 1, 2, 3, 4</code> ; rækkefølgen er uden betydning.
$[^class]$	matcher en karakter som ikke er i $class$ . Mønsteret <code>[^abc1-4]*</code> matcher sekvenser af vilkårlige karakterer som ikke er blandt <code>a, b, c, 1, 2, 3, 4</code> .
$\$$	matcher den tomme streng.
$p?$	matcher 0 eller 1 instanser af mønsteret $p$ (ex: strengene <code>aa</code> og <code>aba</code> matcher mønsteret <code>ab?a</code> , men strengen <code>abba</code> matcher ikke mønsteret <code>ab?a</code> ).

Figur 1: Syntaksen for regulære udtryk (mønstre). Bogstavet  $p$  bruges til at identificere et vilkårligt mønster. Ordet  $class$  bruges til at identificere en vilkårlig klasse; se Figur 2.

---

der specificeres ved det pågældende regulære udtryk. Ud fra definitionen af matching kan man således spørge om et mønster  $p$  matcher en streng  $s$ . Syntaksen for mønstre er givet i Figur 1.

Den simpleste form for regulært udtryk giver mulighed for at undersøge om en streng er identisk med en anden streng. For eksempel matcher det regulære udtryk `tele` strengen `tele`. Mere fleksibilitet kan opnås ved brug af wildcardet `.`. Således matcher det regulære udtryk `.at` strengene `kat` og `hat`.

## 2.1 Karakterklasser

Karakterklasser giver mulighed for at specificere at en af flere mulige karakterer (tegn) skal matches. En karakterklasse  $class$  specificerer således en mængde af ASCII-karakterer,

---

<i>class</i>	<b>Definition</b>
<i>c</i>	klasse indeholdende den specifikke karakter <i>c</i>
$\backslash c$	klasse indeholdende den escapede karakter <i>c</i> , hvor <i>c</i> er en af  , *, +, ?, (, ), [, ], \$, ., \, t, n, v, f, r.
<i>c</i> <sub>1</sub> - <i>c</i> <sub>2</sub>	klasse indeholdende ASCII-karakterer i intervallet $[[c_1]]$ til $[[c_2]]$ (defineret som karakterernes ASCII værdi)
<i>class</i> <sub>1</sub> <i>class</i> <sub>2</sub>	klasse bestående af karakterer i <i>class</i> <sub>1</sub> og i <i>class</i> <sub>2</sub>

---

Figur 2: Syntaksen for karakterklasser. Vi bruger *class* til at identificere en vilkårlig klasse.

---

som defineret i i Figur 2.

Karakterklasser defineres ved brug af de specielle tegn [ og ]. Det regulære udtryk [abcdefg] matcher således alle strenge, der indeholder et af de angivne bogstaver og består af præcist et tegn. Samme effekt kan opnås ved at gøre brug af muligheden for at specificere karakterintervaller i karakterklasser. Således matcher det regulære udtryk [a-z] alle strenge af længde 1, som indeholder et af bogstaverne fra a til z. For at få de danske bogstaver med skal vi angive dem separat. Udtrykket [a-zæøå] matcher således alle små bogstaver i det danske alfabet. Vil vi inkludere de store bogstaver, angiver vi dem i samme klasse: [A-ZÆØÅa-zæøå]. Samme teknik kan benyttes til at specificere klasser bestående af cifre. For eksempel specificerer det regulære udtryk [1-4] cifrene 1, 2, 3 og 4.

Ved brug af tegnet ^ først i en karakterklasse defineres klassen som klassen af de tegn der ikke er specificeret ved den efterfølgende angivne klasse. Således matcher det regulære udtryk [^a-z] strengene æ, ø og å, men ikke g.

## 2.2 Repetition og deludtryk

De specielle tegn \*, + og ? bruges til at specificere et antal gentagne matches.

- \* Betyder at udtrykket må matches nul eller flere gange.
- + Betyder at udtrykket må matches en eller flere gange.
- ? Betyder at udtrykket må matches nul eller én gang.

For nærmere at specificere antallet af tilladte matches kan et antal eller et interval angives efter et regulært udtryk. Således specificerer det regulære udtryk a{3} at bogstavet a skal matches tre gange. Således matcher det regulære udtryk a{3} strengen aaa. På samme vis specificerer det regulære udtryk a{2,4} at bogstavet a skal matches mellem 2 og 4 gange og det regulære udtryk a{2,} at bogstavet a skal matches mindst 2 gange.

Det er også muligt at gruppere deludtryk med parenteser Således matcher det regulære udtryk a(bc)\*d strengene ad, abcd, abcbcd, og så videre.

## 2.3 Grene

Regulære udtryk giver også mulighed for at specificere alternativer ved brug af operatoren `|`. Således matcher det regulære udtryk `a|b` enten bogstavet `a` eller bogstavet `b`. Som et større eksempel kan vi betragte det regulære udtryk

```
(0|[1-9][0-9]*)
```

Dette regulære udtryk specificerer at en matchende streng enten skal være cifferet `0` eller et ciffer `1-9` efterfulgt af `0` eller flere forekomster af cifre `0-9`. Det regulære udtryk specificerer således mængden af naturlige tal, inklusiv tallet `0`.

## 3 Funktionen ereg

Den indbyggede PHP-funktion `ereg` kan benyttes til at afgøre om et mønster `m` matcher en delstreng i en streng `s`. Et kald `ereg(m,s)` returnerer PHP-værdien `true` hvis mønsteret `m` matcher en delstreng i `s`. Ellers returnerer funktionen PHP-værdien `false`. Hvis mønsteret `m` startes med `^` må ingen tegn forekomme før delstrengen i `s`. Tilsvarende, hvis mønsteret `m` slutes med `$` må ingen tegn forekomme efter delstrengen i `s`.

Her følger en række eksempler på brug af funktionen `ereg`:

Funktionskald		Resultat
<code>ereg('[0-9]+', "aa38AA")</code>	→	1
<code>ereg('^ [0-9]+', "aa99")</code>	→	0
<code>ereg('^ [0-9]+', "77AA")</code>	→	1
<code>ereg('^ [0-9]+\$', "aa87AA")</code>	→	0

Til check af brugerindtastninger vil vi oftest benytte os af tegnene `^` og `$` først og sidst i mønsteret. Bemærk også at vi til opskrivningen af mønstre benytter os af muligheden for at indkapsle strenge med `'...'` for at undgå den specielle betydning af tegnet `$` i strenge på formen `"..."`

### 3.1 Specielle karakterer

Hvis vi ønsker at benytte specielle karakterer uden for karakterklasser, er det nødvendigt at escape dem med backslash (`\`). Dette gælder blandt andet karaktererne `.` (wildcardet), `\` (backslash) og `$` (dollartegn). Vil vi således matche den noget specielle streng `.\$` skal vi escape således:

```
\\.\\\$
```

Figur 3 giver en oversigt over en del af de specielle karakterer og deres betydning uden for karakterklasser når regulære udtryk bruges i PHP-funktionen `ereg`.

---

Tegn	Betydning
\	Escapetegn
^	Match fra starten af strengen
\$	Match til slutningen af strengen
.	Match alle tegn (bortset fra den lidt specielle "newline", \n)
	Eller-mønster
(	Start deludtryk
)	Afslut deludtryk
*	Nul eller flere gange repetition
+	En eller flere gange repetition
?	Nul eller en gang

Figur 3: En del af de specielle karakterer og deres betydning, når de er brugt uden for karakterklasser.

---

## 3.2 Eksempler

Mulig brug af regulære udtryk illustreres bedst med en række eksempler:

- `[A-Za-zÆØÅæøå]` : matcher alle karakterer i det danske alfabet.
- `[1-9][1-9]` : matcher tal bestående af to cifre, hvor ingen af cifrene må være 0.
- `(cow|pig)s?` : matcher de fire strenge `cow`, `cows`, `pig` og `pigs`.
- `((a|b)a)*` : matcher `aa`, `ba`, `aaaa`, `baaa`, ...
- `(0|1)+` : matcher de binære tal (ex: 0, 1, 01, 11, 011101010, ...).
- `..` : matcher to vilkårlige karakterer.
- `([1-9][0-9]*)/([1-9][0-9]*)` : matcher positive brøker (ex: 1/8, 32/5645 og 45/6). Bemærk at mønsteret ikke matcher brøkerne 012/54 og 1/0.
- `<html>.*</html>` : matcher HTML-sider.
- `www\.(((it-c|itu)\.dk)|(it\.edu))` : matcher de tre webadresser `www.itu.dk`, `www.it-c.dk` og `www.it.edu`.
- `http://hug.it.edu:8034/ps2/(.*)\.php` : matcher alle URL'er med endelse `.php` på maskinen `hug.it.edu` i biblioteket `ps2` for servicen der kører på port 8034.