COMPUTERWORLD

Regulære udtryk, 4. del, Kogebogsudtryk

Artiken gennemgår regex'er til nogle almindelige opgaver: dato, klokkeslet, email adresse, URL, HTML-tags og -attributter, hel linje med bestemte ord og password. Desuden introduceres if-then-else matchning.

Skrevet den **04. Feb 2009** af **nielle** I kategorien **Programmering / Reg.Exp.** | ★★★☆☆

Indledning

Dette er den 4. og foreløbig sidste artikel i min miniserie om regulære udtryk. Den var fra starten af tænkt til at skulle være en simpel liste over nogle specielt brugbare "dagligdags" regulære udtryk. Den endte med at være noget lidt mere end det. Den endte i øvrigt også med at være noget længere end planlagt.

Artiklen er ikke nogen facitliste. Andre kan sagtens have en anden mening om, hvordan en given regexp bør se ud. Under alle omstændigheder afhænger den slags altid af sammenhængen det skal bruges i.

Der er også forskel på hvordan en regexp bør se ud, hvis man ønsker at udtrække data af en streng - i forhold til hvordan den skal se ud hvis den skal bruges til at validere et input. I det første tilfælde behøver den ikke at være så omstændelig idet man forudsætter at data allerede er i orden, og man blot ønsker at undgå at få fat i "falske" match. I det andet tilfælde ønsker man netop at sikre sig at input faktisk er ok.

Som altid vil kodeeksemplerne være i PHP. Hvis man ønsker at bruge regexp i andre sprog vil jeg henvise til den 3. artikel som er tænkt som en slags regexp'ernes Rosetta-sten. :^)

v. 1.0: 23/12/2007 - Første version.
v. 1.1: 09/01/2008 - Tilføjelse til mønsteret for matchning af en dato.
v. 1.1b: 09/01/2008 - Rettelse i dato-mønsteret: Der er da ikke 32 dage i en måned...

Match en lovlig dato

Datoer på formen dd-mm-yyyy kan matches med dette mønster:

```
$pattern = "(0[1-9]|[12]\d|3[01])-(0[1-9]|1[012])-(19|20)\d{2}";
```

Det tager dog ikke højde for at nogen måneder kun er 28, 29 eller 30 dage. Det er endvidere baseret på at året af formen 19xx eller 20xx, men denne del kan selvfølgelig tilpasses.

Eller, hvis datoer også skal kunne matches selvom der ikke nødvendigvis er foranstillet med '0' - som f.eks. 9-1-2008:

```
$pattern = "(0?[1-9]|[12]\d|3[01])-(0?[1-9]|1[012])-(19|20)\d{2}";
```

Match et klokkeslæt

Klokkeslæt på formen hh:mm:ss kan matches med:

```
$pattern = "\d{2}:\d{2}:\d{2}";
```

Er der millisekunder med, kan det matches med:

```
$pattern = "\d{2}:\d{2}:\d{2}:\d{3}";
```

Disse mønstre tillader godt nok ulovlige klokkeslæt som f.eks. 99:99:99 og hvis man vil være helt sikker på at undgå at matche på disse kan man gøre noget i stil med det der er gjort ovenfor for datoer.

Match en email adresse

En typisk anvendelse af regulære udtryk er validering af input fra brugeren. F.eks. validering af en indtastet email adresse fra en form på en hjemmeside:

```
<?
$name = trim($_POST['name']);
$eMail = trim($_POST['eMail']);
?>

Jeg vil gerne tilmeldes jeres nyhedsbrev:
<form method="post">
Navn: <input type="text" name="name" value="<?= $name?>"><br>
EMail: <input type="text" name="eMail" value="<?= $eMail?>"><br>
<input type="submit" value="Send">
</form>

<?php
function RegisterForNewsletterConfirmationMail($name, $eMail) {
    // Gem i databasen som mulig modtager af newsletter.
    // Send en godkendelses email.
    // ...

    echo "Tak for din interesse.
        Du vil modtage en email med et link som du skal klikke
        på før at din tilmelding til vores nyhedsbrev aktiveres.";
}

if (!empty($name) && !empty($eMail)) {
    $namePattern = "^.{10,}$";  // Mindst 10 tegn i navnet
    $eMailPattern = "^[^@]+@(?:[\w-]+\.)+[a-z]{2,}$";

    if (!preg_match("/$eMailPattern/i", $eMail)) {
```

```
        echo "Det ligner ikke en email adresse!";
    } else if (!preg_match("/$namePattern/", $name)) {
        echo "Navnet er for kort!";
    } else {
        // Alt ok
        RegisterForNewsletterConfirmationMail($name, $eMail);
    }
} else if (!empty($name) || !empty($eMail)) {
    echo "Du mangler at udfylde mindst et af felterne.";
}
?>
```

Email regexp'en:

^[^@]+@(?:[\w-]+\.)+[a-z]{2,}$

består af følgende dele:

^ ... [^@]+ ... @ ... (?:[\w-]+\.)+ ... [a-z]{2,} ... $

(o) ^ og $: matcher hhv. "starten af strengen" og "slutningen af strengen". De er med for at sikre at der ikke er andet end email-kandidaten i strengen.
(o) [^@]+ : matcher det der står foran email-adressens @-tegn. Faktisk er der ikke nogen restriktioner på hvad der må stå der - det er overladt til den modtagende mail server at bestemme om det er en rigtig eller forkert modtager.
(o) @ : matcher "et @ tegn".
(o) (?:[\w-]+\.)+ : matcher domæne-delen af email adressen på nær TLD'et. I "nielle@www.eksperten.dk" matcher den "www.eksperten." og i "nielle@eksperten.dk" matcher den "eksperten.". Læg specielt mærke til at punktummet er inkluderet i det matchede
(o) [a-z]{2,} : matcher Top Level Domænet. De længste TLD'er som pt. er i brug er ".museum" og ".travel" så delmønsteret kunne også have været den mere strikse [a-z]{2,6}. Men hvem ved ... måske kommer der nogle endnu længere i fremtiden, og så ville det være lidt trist at skulle rette i sit script af den grund.

Husk i øvrigt på, at selv om en regexp kan validere at noget ligner en email adresse så kan den hverken verificere:

(o) at den overhovedet findes
(o) at der er nogen som faktisk læser de emails der sendes til den
(o) at den tilhører den bruger som har indtastet den

For at tage højde for den slags bør man have et system som sender en email med et aktiveringslink til den angivne adresse. Dette falder dog lidt uden for denne artikel.

Generelt om validering af input fra brugerne

Når man validere på en form kan man vælge at validere clientside med JavaScript eller serverside med PHP. Og man kan selvfølgelig vælge at gøre begge dele.

Man må endelig ikke basere sin sikkerhed på clientside validering (nu snakker jeg altså web programmering). Folk med hacker-tendenser har rigeligt af muligheder for at slå diverse valideringer fra eller at sammensætte specielt input med det specifikke formål at kompromittere et system. Validering på

clientside har sin berettigelse, men det er først og fremmest som en **hjælp** til brugerne snarere end som en sikkerhedsmekanisme.

Hvis input skal valideres, så skal det **også valideres på serverside**. Men sikkerheden stopper ikke nødvendigvis der - man bør f.eks. også være opmærksom på angrebstyper som SQL injektion og cross-side scripting.

Match en URL

Der er visse ligheder mellem hvordan man behandler en web adresse i forhold til en email adresse. Dette er selvfølgelig ikke så underligt eftersom at delleddet efter '@' tegnet jo er en web adresse.

Imidlertid er der alligevel nogle ekstra problemstillinger.

(o) Man ønsker måske ikke at kunne komme til at matche på sådan noget som "Folk glemmer måske mellemrummet efter et punktum.De fortsætter bare lige efter...".
(o) En web adresse kan indeholde en del mere end blot selve domænet, f.eks. "http://www.eksperten.dk/artikler/1162".

Mit bud på en løsning:

```
<?
$pattern = '
((?:https?|ftp)://)?            # scheme
([\w-]+\.)+                     # host - domain og subdomain\'s
([a-z]{2,})                     # host - tld
(:\d+)?                         # port
(/~?[^\s/?&]*)*                 # path
([?&][^\s=?&\#]+=[^\s=?&\#]*)*  # query - after the question mark ?
(\#[^\s]+)?                     # fragment
';
$phpPattern = "{" . $pattern . "}x";

$text = "xxx www.eksperten.dk yyy";
$text = preg_replace($phpPattern, "<a href='$0'>$0</a>", $text);
echo $text . "<br>";

$text = "xxx http://dk.php.net/manual/en/reference.pcre.pattern.modifiers.php
yyy";
$text = preg_replace($phpPattern, "<a href='$0'>$0</a>", $text);
echo $text . "<br>";

$text = "xxx
https://subdomain1.subdomaim2.domain.tld:80/path1/path2.php?query1=7&query2=7&
query3=13#fragment yyy";
$text = preg_replace($phpPattern, "<a href='$0'>$0</a>", $text);
echo $text . "<br>";
?>
```

Et URL består potentielt af så mange dele at det hurtigt bliver svært at holde styr på hvilket delmønster af

regexp'en som holder styr på hvad. Ved at tilføje x-modifieren til regexp'en får man mulighed for at skrive *kommentarer*. Disse starter med et '#', og da '#' dermed er blevet et specialtegn (på samme måde som '.', '^', osv.) skal det dermed også escapes hvis det blot skal matche et '#' - som i fragment-leddet ovenfor.

Samtidigt betyder x-modifieren at eventuelle whitespace-tegn i regexp'en ignoreres, og det er årsagen til at jeg kan tillade mig at opstille de enkelte kommentarer pænt over hinanden. Hvis man har brug for at matche whitespace-tegn kan man altid falde tilbage på at matche dem med \s.

Somme tider kan man ikke få det som man vil; den viste regexp vil f.eks. fortolke punkt 1 fra ovenfor, "... et punktum.De fortsætter ..." som et URL. Tit ser man at folk forsøger at lave en regexp som bare skal kunne klare alt. Resultatet er ofte at de i stedet får lavet et monster udtryk som introducerer flere fejl end det eliminere. Desuden bliver sådanne udtryk som regel temmelig skrøbelige, forstået på den måde at man ikke kan lave en eneste lille bitte rettelse uden at den slet ikke fungere mere. Det er vigtigt at man er klar over at regexp ikke er et vidundermiddel, og at man nogen gange er bedre tjent med at dele en given opgave over flere udtryk, eller måske endda finde en helt anden løsning.

Matchning på HTML tags

Regulære udtryk kan bruges til at saniterer potentielt farligt input fra brugerne - men husk at gøre det servereside, ellers er det ikke meget værd som beskyttelse!

Specielt som webudvikler skal man kende lidt til hacking, i hvert fald til det niveau hvor man er klar over hvilke typer angreb der findes, og hvordan man så beskytter sig i mod dem. Et af disse angreb er *cross-site scripting*, ofte forkortet som XSS for ikke at forveksle det med cascading style sheets. I al sin enkelthed går det ud på at en hacker-type forsøger at poste tekst med skadelig JavaScript (eller anden type clientside script) til dit site i håb om at du så bare poster det råt og ubehandlet på din tagwall. Hvis det sker for dig en dag så er du uheldigvis i fint selskab med et par rigtigt store websites.

```
<?
$tilTagwall = 'html kode her ...
<script type="text/javascript">
{Noget potentielt farligt JavaScript her}
</script>
... og mere html kode her ...
<SCRIPT type="text/javascript">
{Lidt mere JavaScript her}
</script>
... og resten af html\'en her.';

$pattern = "<script[^>]*>.*?</script>";  // Læg mærke til tricket med [^>]*
$phpPattern = "#$pattern#is";

$tilTagwall = preg_replace($phpPattern, "", $tilTagwall);

echo $tilTagwall;
?>
```

Resultat:

html kode her ... ... og mere html kode her ... ... og resten af html'en her.

Nu har PHP faktisk allerede en ganske udmærket funktion, i form af *strip_tags()*, til netop den slags problemstillinger. Imidlertid klare den ikke lige det viste eksempel, hvor at man ville få dette:

html kode her ... {Noget potentielt farligt JavaScript her} ... og mere html kode her ... {Lidt mere JavaScript her} ... og resten af html'en her.

(selve <script> taggene er dog forsvundet - så det postede er ikke skadelig mere)

Efter min mening gør denne "mangel" nu heller ikke så meget, for jeg ville aldrig drømme om at bruge ret mange kræfter at smukkesere poster med indhold i stil med det ovenstående. **Sådan nogen ville jeg simpelthen smide helt væk i stedet for!** Hvis folk forsøger at sabotere mit site, står de alligevel til at blive bannet - måske endda anmeldt.

Som sagt har PHP allerede strip_tags() og eksemplerne i denne sektion er mere tænkt som en demonstration (eller til brug i andre sprog).

Match attributter - brug af if-then-else regexp'en

HTML-tags har ofte attributter, og attributterne har værdier; Desværre er folk ikke altid så omhyggelig med om der skal et '-tegn eller et "-tegn eller ingenting om værdien:

Dette giver nogle problemer når man f.eks. ønsker at få fat på værdien af src-atributten i det ovenstående. Det er nu nok til at klare, for det er vel bare et eller andet med at matche alt efter 'src' indtil at man enten møder et mellemrum eller en '/' eller en '>'. Så kan man altid trimme for eventuelle ' eller "-tegn bagefter.

Hmm, rent faktisk er den med '/'-tegnet et problem, for netop det tegn indgår muligvis også i filstien.

Endnu værre bliver det dog med alt-attributten. Der kan man ikke tillade sig at antage at matche til og med første mellemrum, for det er jo også tilladt med mellemrum i selve teksten:

Nu er jeg ikke så ond at jeg fortæller om alle problemerne for så at fortælle at der ikke er en løsning. Løsningen er noget i stil med at se på det første tegn efter '=' og så gøre noget forskelligt alt efter om det er et '-tegn eller et "-tegn eller ingen af delene:

```
<?
$pattern = '<img [^>]*?alt=(["\'])?(.*?)(?(1)\1|[ />])';  // '-tegnet skal
escapes i en '-streng
$phpPattern = "#$pattern#i";

$tekst = "xxx <img aaa='111' alt='a b c' bbb='222'> yyy";
preg_match($phpPattern, $tekst, $matches);
echo "alt = [" . $matches[2] . "], omklamrings-tegn = [" . $matches[1] .
"]<br>";
```

```
$tekst = 'xxx <img aaa="111" alt="a b c" bbb="222"> yyy';
preg_match($phpPattern, $tekst, $matches);
echo "alt = [" . $matches[2] . "], omklamrings-tegn = [" . $matches[1] .
"]<br>";

$tekst = 'xxx <img aaa="111" alt=a b c bbb="222"> yyy';
preg_match($phpPattern, $tekst, $matches);
echo "alt = [" . $matches[2] . "], omklamrings-tegn = [" . $matches[1] .
"]<br>";
?>
```

Resultat:

alt = [a b c], omklamrings-tegn = [']
alt = [a b c], omklamrings-tegn = ["]
alt = [a], omklamrings-tegn = []

Forklaring:

Den kødfulde del af regexp'en er denne her

alt=(["\'])?(.*?)(?(1)\1|[/>])

som kan deles i disse 4 delmønstre:

alt= ... (["\'])? ... (.*?) ... (?(1)\1|[/>])

(o) alt= : matcher teksten "alt=" og det er denne del, som sikre at det netop er alt-attributtens værdi vi ender med.

(o) ["'] : matcher "et '-tegn eller et "-tegn".
(o) (["']) : Dette bliver fanget i den første gruppe, og kan derefter tilbagerefereres som \1, hvilket jeg benytter om lidt.
(o) (["'])? : Det afsluttende ?-tegn er yderlig en detalje - den sikre at der også kan matches i tilfældet hvor der hverken er et '- eller et "-tegn på den plads.

(o) .*? : Lazy-matcher det der kommer efter gruppe 1. Hvis der ikke er noget som tvinger denne til at matche noget, ville .*? faktisk slet ikke matche andet end den tomme streng. Tvangen kommer fra det 4. delmønster som jeg kommer til lige om et sekund.
(o) (.*?) : Det matchede huskes som 2. gruppe, og det er derfor at vi efterfølgende kan udskrive det som værdien af $matches[2].

Det 4. delmønster bruger en regex-konstruktion som jeg ikke har fortalt om før nu - *if-then-else*:

(o) (?(n) then | else) : "hvis gruppe *n* matchede noget, matcher *if-then-else* det der står på pladsen 'then', og ellers matcher den det der står på pladsen 'else'". Pladserne 'then' og 'else' kan være regulære udtryk.

I det konkrete tilfælde, (?(1)\1|[/>]), er det:

(o) n = 1 : Hvis det lykkes for gruppe 1 at matche noget er det 'then' delen der benyttes, og ellers er det 'else' delen.
(o) then = \1 : Hvis gruppe 1 matcher noget, vil "\1" være en *backreference* til den. Hvis der derfor

matches et '- eller et "-tegn vil 4. delmønster tvinge den totale regex til at lede efter dennes tvilling i den modsatte ende af værdien.
(o) else = [/>] : Hvis det derimod ikke lykkes for gruppe 1 at matche, vil 4. delmønster i stedet tvinge regex'en til at fortsætte med at matche indtil der mødes et mellemrum eller et '>'-tegn.

Matche en linje, som indeholder et bestemt ord

Vi har et stykke tekst, bestående af flere linjer, og vi ønsker at få fat i netop de linjer hvor et bestemt ord indgår. Et første gæt kunne være at forsøge at matche på "\bord\b":

```
<?
$tekst =
"Linje med et bestemt ord i.
Endnu en linje, hvor ordet ikke er i.
Det der ord igen.";

$pattern = "\bord\b";  // Delmønstre: \b ... ord ... \b
$phpPattern = "/$pattern/im";  // m-modifieren er vigtig her!

preg_match_all($phpPattern, $tekst, $matches);

echo "[pre]";
print_r($matches);
echo "</pre >";
?>
```

Resultat:

```
Array
(
    [0] => Array
    (
        [0] => ord
        [1] => ord
    )
)
```

Vi får godt nok fat i de to gange "ord", men så heller ikke ret meget mere... vi får ikke resten af linjernes indhold. En mulig løsning er at ændre vores mønster til:

```
$pattern = "^.*?\bord\b.*$";
```

og det giver det ønskede resultat:

```
Array
```

```
(
    [0] => Array
    (
        [0] => Linje med et bestemt ord i.
        [1] => Det der ord igen.
    )
)
```

Løsningen virker, men den skalerer ikke specielt godt til flere ord...

Match en linje, som indeholder to bestemte ord

Vi har et stykke tekst, bestående af flere linjer, og vi ønsker at få fat i netop de linjer hvor to bestemte ord indgår.

Hvis vi ved/forlanger at ord1 kommer før ord2:

```
$pattern = "^.*\bord1\b.*?\bord2\b.*$";
```

Eller... hvis vi intet ved om, eller ligefrem er ligeglad med, rækkefølgen af dem:

```
$pattern = "^.*?(\bord1\b.*?\bord2\b|\bord2\b.*?\bord1\b).*$";
```

Måske ikke særligt kønt, men det klare jobbet. Det bliver dog rigtigt hurtigt endnu værre når vi forsøger at udvide løsningerne til 3, 4, 5, ... ord.

Positive lookahead to the rescue!

Lad os prøve en anden fremgangsmåde; test på et enkelt ord (igen):

$pattern = "^(?=.*?\bord\b).*$";

Parentesen (?=...) kaldes for en *positive lookahead* (artikel nr. 2). Den går basalt ud på at:

(o) xxx(?=yyy) : matcher "det der står på pladsen xxx, hvis og kun, hvis det efterfølges af det der står på pladsen yyy".

Både xxx og yyy kan være regulære udtryk i egen ret, og det er de i det ovenstående; i mønsteret "^(?=.*?\bord\b)" er xxx lige med mønsteret '^' og yyy lig med ".*?\bord\b":

(o) ^ : matcher "starten af strengen".
(o) .*?\bord\b : lazy-matcher "et vilkårligt antal tegn, efterfulgt af starten af et ord (\b), efterfulgt af teksten/ordet 'ord', efterfulgt af slutningen af et ord (\b)".
(o) ^(?=.*?\bord\b) : matcher derfor "starten af strengen, hvis og kun, hvis den er efterfulgt af et stykke tekst som indeholder ordet 'ord' (uden at dette er en del af et større ord)".

Læg her specielt mærke til formuleringen:

... matcher "starten af strengen ..." ...

Det er nemlig kun **starten** der matches af det ovenstående! For at få fat i indholdet af selve linjen skal den derfor udvides en smule til:

(o) ^(?=.*?\bord\b).*$: matcher "starten af strengen, hvis og kun, hvis ordet 'ord' indgår, efterfulgt af et vilkårligt antal tegn, efterfulgt af enden af strengen".

Vi har altså stadig et regulært udtryk, som matcher en linje, hvis og kun, hvis den indeholder 'ord'. Det er det samme som en side eller to længere oppe... Så hvad har vi fået egentlig ud af det?

Vi har fået ud det ud af det at vi nu har et udtryk som kan "stables" på en skalerbar måde:

(o) ^(?=.*?\bord1\b).*$: matcher "en linje, hvis og kun, hvis den indeholder ordet 'ord1'".
(o) ^(?=.*?\bord1\b) (?=.*?\bord2\b).*$: matcher "en linje, hvis og kun, hvis den indeholder ordene 'ord1' og 'ord2'".
(o) ^(?=.*?\bord1\b) (?=.*?\bord2\b) (?=.*?\bord3\b).*$: matcher "en linje, hvis og kun, hvis den indeholder ordene 'ord1' og 'ord2' og 'ord3'".
(o) osv.

1, 2, mange ... - match et password

Vi startede altså med den grundlæggende regex, "\bord\b", og den blev til pakket ind til "^(?=.*?\bord\b).*$". Den generelle opskrift ser sådan ud:

Hvis man vil matche alle de linjer, som indeholder noget, som matcher "det regulære udtryk 'regex'", skal man bruge regex'en "^(?=.?regex).*$".*

Eksempel:

Vi ønsker at et password skal opfylde visse betingelser for at være sikker:

Det skal være mindst otte tegn langt, indeholde minimum to store bogstaver, minimum to små bogstaver, mindst et tal og mindst et af tegnene '@', '#', '£', '¤', '$', '%' og '&'.

Hver af disse krak kan udtrykkes med en regex:

(o) regex1 = .{8,} : matcher "8 eller flere vilkårlige tegn".
(o) regex2 = [A-Z].*?[A-Z] : matcher "mindst 2 store bogstaver". 'Æ', 'Ø' og 'Å' er dog ikke dækket af løsningen.
(o) regex3 = [a-z].*?[a-z] : matcher "mindst 2 små bogstaver". 'æ', 'ø' og 'å' undtaget.
(o) regex4 = \d : matcher "mindst 1 ciffer".
(o) regex5 = [@#£¤$%&] : matcher "et vilkårligt af tegnene '@', '#', '£', '¤', '$', '%' eller '&'".

Med opskriften fra før klares dette med denne stablede regex:

$pattern = '^(?=.*?regex1)(?=.*?regex2)(?=.*?regex3)(?=.*regex4)(?=.*regex5).*$';

Koden:

```
```

```
<form method="post">
<input type="password" name="glPassword"><br>
<input type="password" name="nyPassword1"><br>
<input type="password" name="nyPassword2"><br>
<input type="submit" value="Nyt password">
</form>

<?
$glPassword = trim($_POST['glPassword']);
$nyPassword1 = trim($_POST['nyPassword1']); $nyPassword2 =
trim($_POST['nyPassword2']);

if (!empty($glPassword))
{
    // 1) Tjek at $glPassword er det gamle password
    // 2) Tjek at $nyPassword1 == $nyPassword2
    // ... og så ...

    $pattern = '^(?=.{8,})(?=.*?[A-Z].*?[A-Z])(?=.*?[a-z].*?[a-
z])(?=.*?\d)(?=.*?[@#£¤$%&]).*$';
    $phpPattern = "/$pattern/";

    if (preg_match($phpPattern, $nyPassword1)) {
        echo "OK";
        // Opdater brugeren til at bruge det nye password
    } else {
        echo "Ikke OK";
    }
} else {
    echo "...";
}
?>
```

Der er snydt lidt på vægten ved regex1 og det oprindelige delmønster er simplificeret fra (?=.*?.{8,}) til (?=.{8,}) men ellers går den det samme.

Kommentar af cf560 d. 13. Jan 2008 | 1

takker

Kommentar af jih d. 28. Aug 2008 | 2

nice.. :-)

Kommentar af horizon d. 20. Aug 2008 | 3

En lille ting jeg har undret mig over mht. validering af emails og urls i php og som jeg sjældent ser kommenteret nogen steder er hvorfor man ikke "bare" bruger funktionen filter_var($string, FILTER_VALIDATE_EMAIL) eller FILTER_VALIDATE_URL er dette ikke hurtigere at bruge eller kontrollerer disse ikke lige så grundigt? (og ja, er med på reg.exp. også bruges i mange andre progammeringssprog, undrer mig bare i php sammenhænge)