

Database-baseret Web-publicering, efterår 2001

Foredælsning 6, tirsdag den 2. oktober 2001.

Streng-matching og indhentning af data fra fremmede web-sites

- Udtagning af tilfældigt element fra en liste
- To typer af form-variable
- Terningekast, frekvensliste og vandrette søjler
- Regulære udtryk (mønstre)
- Tcl-kommandoen regexp
- Bill Gates Personal Wealth Clock
- Tcl-kommandoen regexp
- Eksempler på brug af regexp-kommandoen

Eksemplet typesetting fra sidste uge (typesetting4.html)

I sidste uge lærte vi, at flere værdier kan overføres til serveren under samme navn.

Betragt følgende HTML formular:

```
<form action="typesetting4.tcl" method=post>
Den tekst du skriver her
<input type=text name=tekst>
bliver formateret således:<p>
<blockquote>
Kursiv: <input type=checkbox name=font value="i"><p>
Fed: <input type=checkbox name=font value="b"><p>
understregning: <input type=checkbox name=font value="u"><p>
</blockquote>
Du kan vælge mellem sort og gul baggrund:
Grøn <input type=radio name=farve value="Green">
Gul: <input type=radio name=farve value="Yellow" checked><p>
<input type=reset value="Forfra">
<input type=submit value="Formatter tekst">
</form>
```

Variablen font vil antage nul, en eller flere værdier:

Udtagning af et tilfældigt element fra en liste

I sidste uge lærte vi om tilfældige tal.

For at lave en procedure som returnerer et tilfældigt udvalgt element fra en given liste har vi brug for at:

- kende antallet af elementer i listen (llength)
 - kunne udtage et element givet et indeks ind i listen (lindex).
- ```
proc udtag_fra_liste { l } {
return [lindex $l [randomrange [llength $l] 1]]
}
```

Eksempel: En web-service, som viser en tilfældigt valgt quote (quotes.tcl).

```
proc home_page { title body } { ... }
proc mk_quote { } {
set quotes [list "The world will end in 5 minutes. Please log out..."]
lappend quotes "WARNING: Keyboard Not Attached. Press F10 to Continue."
...
lappend quotes "Cannot find REALITY.SYS...Universe Halted."
return "
<table bgcolor=white>
<tr><th>Category: Error Messages</th></tr>
<tr><td><i>[udtag_fra_liste $quotes]</i>-Anon.<td></tr>
</table><p>
Press the <i>reload</i>-button to get a new quote."]
ns_return 200 text/html [home_page "Quotes" [mk_quote]]
```

### To typer af form-variable

Vi bliver altså nødt til at skelne mellem

form-variable der kun kan antage en værdi

og

form-variable der kan antage nul, en eller flere værdier.

Nul, en eller flere værdier repræsenteres bedst som en liste!

Vi har to nye procedurer som henter form-variable ind i vores Tcl program.

- set\_form\_var form\_var checker om form\_var eksisterer, og i givet fald opretter en variabel med navn form\_var. Hvis form\_var findes med flere værdier, udvælges en værdi tilfældigt.
- set\_form\_var\_as\_list form\_var checker om form\_var eksisterer, og i givet fald opretter en variabel med navn form\_var. Variablen indeholder en liste med alle værdier, som form\_var er tildelt i HTML formularen.

### Eksemplet Typesetting fortsat (typesetting4.tcl)

```
proc home_page { title body } { ... }
proc lrev { l } { ... }
set_form_var tekst
set_form_var farve
set_form_var_as_list font

if { [info exists font] &&
 [info exists tekst] &&
 [info exists farve] } {
 set prefix " "
 set postfix " "
 foreach e $font { append prefix "<$e>" }
 foreach e [lrev $font] { append postfix "</$e>" }

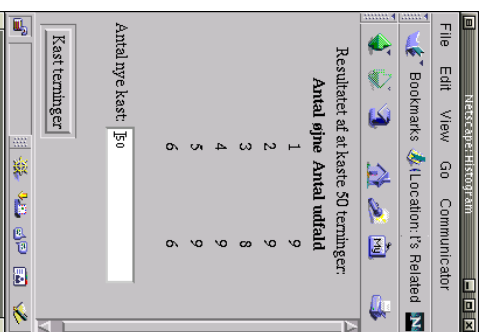
 ns_return 200 text/html [home_page "Typesetting Service" "
 <table bgcolor=$farve>
 <tr><td>
 Teksten $tekst ser således ud med formatering $prefix $tekst $postfix<p>
 </td></tr>
 </table>"]
 } else {
 ns_return 200 text/html [home_page "Typesetting Service" "Du mangler
 at indtaste en eller flere værdier."]
 }
```

### Frekvensliste over terningekast

Antag, at vi kan kaste en terning 6000 gange.

Hvorledes kan vi optælle antallet af 1'ere, 2'ere osv?

Index	0	1	2	3	4	5	6
Indhold							



### Optælling af søjlediagram (histogram) ved hjælp af lister

Vi kan lave en metode, som kaster en terning:

```
proc kast_terning { } {
 return [expr [randomRange 6] + 1]
}
```

Hvordan fordeles udfaldene sig ved 6000 kast med en terning, dvs. hvor tilfældig er tilfældigstalsgeneratoren?

Brug en liste lrev til at indeholde antallet af forekomster af 1, 2, 3, 4, 5, 6.

Element lindex 1 er antallet af enere, lindex 2 er antallet af toere, osv.

Til at begynde med er alle antal 0: set lrev [list 0 0 0 0 0 0]

Når terningen viser 4, så forøg lindex 4 med 1. Dvs., udfør

```
set lrev [lreplace $lrev 4 4 [expr [lindex $lrev 4] + 1]]
```

I almindelighed, hvis variablen r indeholder antal øjne som terningen viser, så udfør

```
set lrev [lreplace $lrev $r $r [expr [lindex $lrev $r] + 1]]
```

hvor

- lindex \$lrev \$r læser det nuværende antal
- expr [lindex \$lrev \$r] + 1 lægger 1 til det nuværende antal
- set lrev [lreplace \$lrev \$r \$r [expr [lindex \$lrev \$r] + 1]] opdaterer listen med det nye antal

### Opbygning af histogram (histogram.tcl)

```
proc gen_histogram { antal } {
 set lrev [list 0 0 0 0 0 0]
 for {set c 0} {$c < $antal} {incr c} {
 set r [kast_terning]
 set lrev [lreplace $lrev $r $r [expr [lindex $lrev $r] + 1]]
 }
 set res "Resultatet af at kaste $antal terninger:
 <table align=center>
 <tr><th>Antal øjne</th><th>Antal udfald</th></tr>\n"
 for {set c 1} {$c <= 6} {incr c} {
 append res "<tr><td align=right>${c}</td>
 <td align=right>[lindex $lrev $c]</td></tr>\n"
 }
 append res "</table>\n"
 return $res
}
set_form_var antal
if { [info exists antal] } {
 set antal 1000
}
ns_return 200 text/html [home_page "Histogram" "
[gen_histogram $antal]<p>
<form method=post action=histogram.tcl>
Antal nye kast: <input type=text value=\"$antal\" name=antal><p>
<input type=submit value=\"Kast terninger\">"]
```

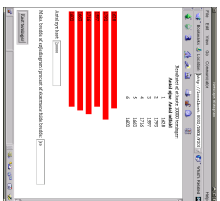
## Histogram som et vandret søjlediagram (histogram2.tcl)

I HTML kan vi ved hjælp af tabeller vise vandrette søjler – dvs. ingen billeder er nødvendige.

Det ser bedst ud, når søjlerne skaleres således at den længste søjle får en fast længde, f.eks. 90% af browserens vindue.

For at skalere de resterende søjler skal vi finde et største tal i listen.

```
proc max_elem { l } {
 set max 0
 foreach e $l {
 if {$e > $max} {
 set max $e
 }
 }
 return $max
}
```



Derefter finder vi længden af de enkelte søjler ved  $w_i = \frac{f_i * 90}{max}$ , hvor  $w_i$  er længden af søjle  $i$  og  $f_i$  er værdien i listen  $f_{req}$  ved indeks  $i$ .

Index $i$	0	1	2	3	4	5	6
Indhold $f_i$	0	37	52	35	45	58	53
Søjlelængde $w_i$	0	57	80	54	70	90	82

## Regulære udtryk (mønstre)

Spørg til at beskrive strenge der har en bestemt form tilfælles. Bruges til:

- at spørge om et givet mønster forekommer i en tegnsekvens
- at udskifte tegnsekvenser der matcher et mønster med andre tegnsekvenser

To vigtige brug af mønstre:

- Sikring af at data fra brugere har den rigtige form
  - hvis et tal forventes i en HTML-form må web-programmet ikke godtage andre tegn end tal cifre
  - mønstre kan ikke bruges til alle krav, hvornår er en dato gyldig?
- Indhentning af data fra fremmede web-sites
  - dollarkursen
  - dagens nyheder fra Reuters
  - vejret
  - ...
  - XML

## Histogram som et vandret søjlediagram, fortsat (histogram2.tcl)

```
proc gen_histogram { antal max_w } {
 set freq [list 0 0 0 0 0 0 0 1]
 for {set c 0} {$c < $antal} {incr c} {
 set r [expr [kast_terning]]
 set freq [lreplace $freq $r $r [expr [lindex $freq $r] + 1]]
 }

 set max [max_elem $freq]
 for {set c 1} {$c <= 6} {incr c} {
 set w [expr [lindex $freq $c] * $max_w / $max]
 append res "<table width=$w%>
 <tr><td bgcolor=red width=$w%>[[lindex $freq $c]</td></tr>
 </table>\n"
 }
 return $res
}

ns_return 200 text/html [home_page "Histogram" "
]
[gen_histogram $antal $max_w]<p>
<form method=post action=histogram2.tcl>
Antal nye kast: <input type=text value=\"$antal\" name=antal><p>
Maks. bredde af søjlediagram i procent af skærmens fulde bredde:
<input type=text value=\"$max_w\" name=max_w><p>
<input type=submit value=\"$kast_terning\">"]
```

## Syntaks for regulære udtryk, del 1

Et mønster  $m$  kan blandt andet tage følgende former:

- $c$  matches af ethvert tegn
- $m_1m_2$  matches af tegnet  $c$ ; tegnet, angives som  $\backslash c$
- $m^*$  matches af 0 eller flere efterfølgende forekomster af tegnsekvenser som matcher mønsteret  $m$ . F.eks. strengene 'abbbba' og 'aa' matcher mønsteret 'ab\*a'
- $(m)$  matches af strenge som matcher  $m$ . F.eks. strengen 'cababc' matcher mønsteret '(ab)\*c'
- $m+$  matches af eller flere efterfølgende forekomster af tegnsekvenser som matcher mønsteret  $m$ . F.eks. mønsteret 'caab' matches af strengen 'caaab' men ikke af strengen 'cb'
- $m?$  matches af 0 eller 1 forekomst af tegnsekvenser som matcher mønsteret  $m$ . F.eks. strengene 'abc' og 'ab' matcher begge mønsteret 'abc?'

## Syntaks for regulære udtryk, del 2

Et mønster  $m$  kan blandt andet tage følgende former:

- $m_1 \mid m_2$  matches af tegnsekvenser som matcher enten  $m_1$  eller  $m_2$ . Føks. mønstret '(gris | ko)' matches af tegnsekvensen 'gris' og tegnsekvensen 'ko'
- [...] matches af tegn i klassen. Føks. [abc1-4]\* matches af tegnsekvenser bestående af tegnene a, b, c, 1, 2, 3, 4
- [^...] matches af alle andre tegn end dem i klassen. Føks. [^abc1-4]\* matches af tegnsekvenser bestående af alle tegn bortset fra a, b, c, 1, 2, 3, 4. Hatten ^ betyder altså, alt andet end de tegn der følger

## Eksempler på mønstre

- [A-Za-zÆØÅæøå] : matcher alle tegn i det danske alfabet
  - [0-9][0-9] : matcher to cifrede tal der kan starte med 0
  - (ko|gris) : matcher de to strenge ko og gris
  - ((a|b)a)\* : matcher aa, ba, aaaa, baaa, ...
  - (0|1)+ : matcher binære tal, dvs 0, 1, 01, 11, 011101010, ...
  - ... : matcher to vilkårlige tegn.
  - ([1-9][0-9]+)/([1-9][0-9]+) : matcher heltalsbrøker, føks. 1/8, 32/5645, 45/6, ...
- Matches strengen 012/54 af mønstret?
- <html>\* : </html> : matcher en HTML side.
  - www\.(it-c|itu)\.dk | (it\ .edu) : matcher www.itu.dk, www.it-c.dk og www.it.edu.
  - http://hug.it.edu:8034/oevelse2/(.\*)\.?cl : matcher alle Tol filer på hug i katalog oevelse2 for den bruger der anvender portnummer 8034.

## Tcl-kommandoen regexp

Kan bruges til at matche en streng mod et mønster:

```
regexp pattern string ?match ?arg1 ... ?argN
```

Kommandoen returnerer 1 hvis der findes en delstreng i strengen *string* som matcher mønstret *pattern*. Ellers returneres 0.

Hvis *pattern* startes med '^' må ingen tegn forekomme før delstrengen i *string*.

Tilsvarende, hvis *pattern* sluttes med '\$' må ingen tegn forekomme efter delstrengen i *string*.

Eksempel:

```
% regexp {[0-9]+} "aa99AA"
1
% regexp {^[0-9]+$} "aa99AA"
0
% regexp {[0-9]+} "99AA"
1
% regexp {[0-9]+$} "99AA"
0
% regexp {^[0-9]+} "aa99"
0
```

Hvis et variabelnavn gives for *?match* bindes variabelen til delstrengen som matcher mønstret.

Hvis variabelnavne gives for *arg1 ... argN* bindes variablerne til delstrenge matchende delmønstre givet i parenteser i *pattern*.

## Eksempler på brug af regexp-kommandoen

Eksempel: Telefonnummer

```
% set s "Name: Hans Hansen Tlf: 66 66 66 66"
Name: Hans Hansen Tlf: 66 66 66 66
% regexp {Name: ([a-zA-Z]+) Tlf: ([0-9]+)} $s match name tlf
1
% set match
Name: Hans Hansen Tlf: 66 66 66 66
% set name
Hans Hansen
% set tlf
66 66 66 66
%
```

Eksempel: Email

```
proc valid_email { e } {
 set pattern {^[a-zA-Z][0-9a-zA-Z\._]*@[0-9a-zA-Z\._]+$}
 return [regexp $pattern $e]
}
% valid_email name@compagny.com
1
% valid_email name@compagny@com
0
```

Opgave: Givet mønstret ovenfor. Hvorledes må en mailadresse se ud?

## Flere eksempler på brug af regexp-kommandoen

Eksempel: URL (Http)

```
% set pattern {^http://[a-zA-Z\\._]+(:[0-9]+)*(/[0-9a-zA-Z\\\\\\._-]+)*}$
~http://[a-zA-Z\\._]+(:[0-9]+)*(/[0-9a-zA-Z\\\\\\._-]+)*$
% regexp $pattern http://www.it.edu
1
% regexp $pattern http://bug.itu.dk:8077/oevelses5/
1
% regexp $pattern attp://bug.itu.dk:8077/oevelses5/
0
```

Eksempel: Dansk Cpr-nummer

```
% set pattern {[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}]
[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}[0-9]{0-9}]
% regexp $pattern "234543-1143"
1
% regexp $pattern "23543-1143"
0
% regexp $pattern "2354831143"
0
% regexp $pattern "13-54-83-1143"
0
```

Dør er ikke noget semantisk check af cpr-nummeret – kun et syntaktisk check.

## Bill Gates Personal Wealth Clock, fortsat

På url `http://www.census.gov/cgi-bin/popclock` finder vi befolkningsantallet i USA. HTML koden indeholder bl.a.:

```
<center><h1> 283,655,630</h1>
```

Følgende regulære udtryk matcher ud de tre tal adskilt af kommaer:

```
regexp {<h1>[0-9]*([0-9]+),([0-9]+),([0-9]+).*</h1>} \
$population_html match millions thousands units
```

Variablen `millions` sættes lig 283, `thousands` lig 655 og `units` lig 630.

Befolkningsantallet beregnes således

```
set population [expr ($millions * 1000000) + ($thousands * 1000) + $units]
```

For at beregne Gates rigdom estimerer vi hans beholdning af Microsoft aktier. Dit bidrag beregnes herefter nemt idet vi kender befolkningsstallet:

```
set gates_shares_pre_split [expr double(141159990)]
set gates_shares [expr $gates_shares_pre_split * 2]
set gates_wealth [expr $gates_shares * $msft_stock_price]
set personal_share [expr $gates_wealth / $population]
```

## Bill Gates Personal Wealth Clock

Vi har brug for kursen på Microsoft aktier, hvilket vi får fra følgende url:

```
http://qs.secapi.com/cgi-bin/qs?ticks=MSFT.
```

HTML siden indeholder bl.a. strengen:

```
Last Traded at</td><td align=right>55 13/16
```

Vi er interesseret i kursen 55 13/16, som vi matcher ud med følgende regexp kommando:

```
regexp {Last Traded at</td><td align=right>([A-z]*)} \
$quote_html match raw_quote
```

Variablen `raw_quote` vil indeholde 55 13/16.

For at regne med kursen skal 55 13/16 omformes til et kommatal. Dette gøres igen med regulære udtryk:

```
regexp {(.*)(.*)} "55 13/16" match whole fraction
regexp {(.*)(.*)} {$fraction match num denom
set extra [expr double($num) / $denom]
set kurs [expr $whole + $extra]
```

Først binder vi `whole` til 55 og `fraction` til 13/16.

Derefter binder vi 13 til variablen `num` og 16 til variablen `denom`.

Givet `whole`, `num` og `denom` beregner vi brøken, dvs. kursen, som et kommatal: `kurs = 55.8125`.

## Opgaver til regulære udtryk og regexp-kommandoen

1. Opskriv regexp kommando som for strengen "DDMMYY-DDDD" binder DD til variablen `dag`, MM til `maaned`, YY til `aar` og DDDD til `k`.

```
set pattern _____
regexp $pattern "280370-3213" _____
```

2. Opskriv regexp kommando som for strengen "SE49584832-AB34" binder 49584832 til variablen `nr1` og AB34 til `nr2`.

```
set pattern _____
regexp $pattern "SE49584832-AB34" _____
```

3. Opskriv regexp kommando som for strengen "ISBN 1-55860-534-7" binder 1 til variablen `nr1`, 55860 til `nr2`, 534 til `nr3` og 7 til `nr4`.

```
set pattern _____
regexp $pattern "ISBN 1-55860-534-7" _____
```

## Tcl-kommandoen `regsub`

Bruges til at udskifte tegnsekvenser som matcher et mønster med andre tegnsekvenser.

```
regsub exp string subSpec varName
```

Kommandoen matcher det regulære udtryk *exp* mod *string* og kopierer *string* til variabelen *varName*.

Under kopieringen erstattes de dele af *string* der matcher *exp* med *subSpec*.

Eksempel: udskiftning af `<` med `&lt;t;`:

```
% set html "<html><body>Hej</body></html>"
<html><body>Hej</body></html>
% regsub -all < $html {\&t;} res
4
% set res
&t;html&t;body&Hej&t;/body&t;/html>
%
```

Eksempel: udskiftning af lokalnummer 853 med 824

```
% set tlf "38 16 88 53"
38 16 88 53
% regsub "8 53" $tlf "8 24" res
1
% set res
38 16 88 24
%
```

## Flere eksempler på brug af `regsub`-kommandoen

Templates:

```
% set template "Hi name,\nHappy birthday!\nCheers, Niels"
% regsub name $template "George Bush" message
1
% set message
Hi George Bush,
Happy birthday!
Cheers, Niels
```

Udskiftning af email-adresser:

```
% set text "Min email adresse er nh@itu.dk, Frank's er bjergoe@itu.dk
og lise's er lise@itu.dk."
Min email adresse er nh@itu.dk, Frank's er bjergoe@itu.dk
og lise's er lise@itu.dk.
% regsub -all "@it-c.dk" $text "@it-c.dk" text
3
% set text
Min email adresse er nh@it-c.dk, Frank's er bjergoe@it-c.dk
og lise's er lise@it-c.dk.
% regsub -all "@it-c.dk" $text "@it.edu" text
3
% set text
Min email adresse er nh@it.edu, Frank's er bjergoe@it.edu
og lise's er lise@it.edu.
```

Husk, at tilføjelsen `-all` gør at alle match fanges!

## Tcl-kommandoen `regsub`, fortsat

```
regsub exp string subSpec varName
```

Enkeltte tegn i *subSpec* har speciel betydning for et given match:

- `&` og `\` står for den del af *string* som matcher *exp*

Eksempel: givet lokalnummer N NN konstruer et fuldt nummer: 38 16 8N NN:

```
% set tlf "8 34, 8 55, 8 67"
8 34, 8 55, 8 67
% regsub -all {[0-9] [0-9]} $tlf "38 16 8&" res
3
% set res
38 16 88 34, 38 16 88 55, 38 16 88 67
% regsub -all {[0-9] [0-9]} $tlf {38 16 8\0} res
3
% set res
38 16 88 34, 38 16 88 55, 38 16 88 67
```

- N står for den del af *string* som matcher det N'de regulære udtryk omgivet af parenteser i *exp* - læst fra venstre mod højre.

Eksempel: givet lokalnummer M NN konstruer et fuldt nummer 38 16 89 NN, hvor M er udskiftet med 9:

```
% set tlf "8 34, 8 55, 8 67"
8 34, 8 55, 8 67
% regsub -all {[0-9]} {[0-9]} $tlf {38 16 89 \2} res
3
% set res
38 16 89 34, 38 16 89 55, 38 16 89 67
```

## Opgaver til regulære udtryk og `regsub`-kommandoen

1. Omform strengen `abe`, `abe` til `abekat`, `abekat`:

```
% set text "abe abe"
abe abe
% regsub -all {_____} $text {_____} res
2
% set res
abekat abekat
```

2. Omform strengen "nh, lise, bjergoe" til `nh@it.edu`, `lise@it.edu`, `bjergoe@it.edu`:

```
% set text "nh, lise, bjergoe"
nh, lise, bjergoe
% regsub -all {([_____])} $text {_____} res
3
% set res
nh@it.edu, lise@it.edu, bjergoe@it.edu
```

3. Omform strengen "hej <i>Jonas</i>" til "hej <u>Jonas</u>":

```
% set text "hej <i>Jonas</i>"
% regsub -all {([_____])} $text {_____} res
2
% set res
hej <u>Jonas</u>
```

Vink: Benyt `/?` til at matche både `<i>` og `</i>`.