

Introduktion til Tcl:

- Kommandoer
- Grundbegreber: værdier, variable, erklæringer, tildeling, udtryk
- Datamatens lager ved udvikling af et program
- Gruppering af argumenter med "{ og }
- Sammensætning af strenge
- if-kommandoen
- while-kommandoen
- procedurer
- løbeseddel 1

Et lidt større Tcl program (body mass .index. tcl)

```
# Bruger indtaster en vægt i kg og en højde i cm.
set vægt 70.0;
set højde 180.0;
puts "Vægt er angivet til $vægt kg.";
puts "Højde er angivet til $højde cm.";
set bmi [expr $vægt / (($højde / 100.0)*($højde / 100.0))];
puts "Dit BMI er $bmi";

if {$bmi < 20.0} {
  puts "Dit BMI er for lav.";
} elseif {$bmi < 25.0} {
  puts "Dit BMI er normalt.";
} elseif {$bmi < 30.0} {
  puts "Dit BMI er for høj.";
} else {
  puts "Dit BMI er alt for høj.";
}
```

Kommandoer

Et Tcl-program består af en række kommandoer på formen:

```
cmd arg1 arg2 ... argN
```

- cmd er et kommandonavn (også kaldet procedurenavn)
- arg1, arg2, ..., argN er argumenter til kommandoen

Et simpelt Tcl program (hello.tcl)

```
puts "Det virker sandelig!"
```

Programmet ligger i filen hello.tcl.

puts er en kommando, som udskriver på skærmen.

Kommandoer adskilles af ny-lin-tegn og/eller semikolon:

```
% puts "Hello"; puts "World"
Hello
World
```

Grundbegreber: værdi

En værdi kan f.eks. være

- et heltal: 120000
- et kommat: 70.0
- en tegnstring: "Højde er angivet til"

Tcl opfatter som udgangspunkt alle værdier som tegnstringe.

Hvis du prøver at regne med to værdier, så checker Tcl, at værdierne enten er et heltal eller kommat.

```
% puts "2+2";           % puts "[expr "2.4 + 2"]"
2+2                     4.4
% puts "[expr "2+2"]";  % puts "[expr "2 + a"]";
4                        syntax error in expression "2 + a"
% set res "2+2";       % set res "2+a";
2+2                     2+a
% puts "[expr $res]";  % puts "[expr $res]"
4                        syntax error in expression "2+a"
```

Grundbegreber: variable, erklæringer og tildeling

Variabler bruges til at gemme værdier.

En variabel svarer til et sted i datamatens lager.

Du navngiver selv dine variable.

Kommandoen set kan bruges til at *erklære* (oprette) en variabel (to argumenter):

```
% set msg "Soon I will be a Web-programmer"
Soon I will be a Web-programmer
```

Kommandoen set kan også bruges til at få fat i indholdet af en variabel (et argument):

```
% set msg
Soon I will be a Web-programmer
```

Man kan også *tildele* en allerede oprettet variabel en ny værdi:

```
% set msg "Soon I will be an excellent Web-programmer"
Soon I will be an excellent Web-programmer
% set msg
Soon I will be an excellent Web-programmer
```

Hvilke variable og tildelinger findes i programmet body_mass_index.tcl?

Variabler - fortsat

Lad os bruge variablen vaegt til at skrive en fornuftig tekst-streng:

```
% puts "Vægt er angivet til [set vaegt] kg." ;
Vægt er angivet til 70.0 kg.
```

Dollar-notation:

Ovenstående kan også skrives som:

```
% puts "Vægt er angivet til $vaegt kg." ;
Vægt er angivet til 70.0 kg.
```

Indlejrede (eng. *nested*) kommandoer

Vi har allerede set flere eksempler på *indlejrede kommandoer*, f.eks. [set vaegt] ovenfor.

Indlejrede kommandoer skrives i kantede parenteser (..).

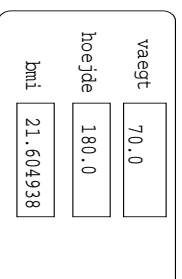
Resultatet af en indlejret kommando kan anvendes i et argument til en anden kommando, f.eks. puts ovenfor.

```
% puts "Vægt efter at have tabt 10 kilo: [expr $vaegt - 10]"
Vægt efter at have tabt 10 kilo: 60
```

Datamatens lager ved atvikling af body_mass_index.tcl

```
set vaegt 70.0 ;
set hoejde 180.0 ;
set bmi [expr $vaegt / (( $hoejde / 100.0 ) * ( $hoejde / 100.0 ) ) ] ;
```

Lager



Grundbegreber: Udtryk og kommandoer

Tilstanden består af datamatens lagerindhold, uddata, osv.

Et *udtryk* beregner en værdi, f.eks. et tal.

Kommandoen expr (expression) bruges til at regne med tal:

```
% expr 4 * 7
28
% expr 4.0 * 7
28.0
```

Der er mange operationer til rådighed: *, /, -, +, ...

Kommandoen set ændrer tilstanden ved f.eks. at opdatere datamatens lager.

Kommandoen puts ændrer tilstanden ved at udskrive på skærm.

Valutaberegning i Unixbank:

```
beloeb.dollar = beloeb.kroner - gebyr
                dollarkurs
% set dollarkurs 7.34
7.34
% set beloeb_kroner 999.0
999.0
% set beloeb_dollar [expr ($beloeb_kroner - 20.0) / $dollarkurs]
133.378746594

Pas på!!! Der er forskel på division med heltal:

% expr 5/9
0

og kommata:

% expr 5.0/9
0.555555555556
```

Backslash's i double-quote grupperinger

Eftersom double-quote (") anvendes som grupperings tegn - hvorledes skriver man så " inde i en double-quoted tegnstring?

```
% set "Byen "København" er ikke så stor"
wrong # args: should be "set varName ?newValue?"
```

Ved brug af backslash (\) kan alle tegn skrives i double-quote grupperinger:

```
% puts "Byen \"København\" er ikke så stor"
Byen "København" er ikke så stor
% puts {Byen "København" er ikke så stor}
Byen "København" er ikke så stor"
```

Et andet eksempel:

```
% set laptop_price 2000
2000
% puts "En ny laptop koster \\$laptop_price"
En ny laptop koster $2000
% puts {En ny laptop koster $$laptop_price}
En ny laptop koster $$laptop_price
```

Hvorledes skriver man en backslash (\) i en double-quoted tegnstring?

Gruppering af argumenter

Der er to måder at gruppere argumenter til kommandoer på:

- % set by København
% puts "Hovedstaden i Danmark er \$by"
Hovedstaden i Danmark er København
- % puts {Hovedstaden i Danmark er \$by}
Hovedstaden i Danmark er \$by

Double-quote grupperinger fortolkes inden strengen sendes som argument til kommandoen.

Brace-grupperinger sendes ufortolket som argument til kommandoen.

Man kan, som oftest, ikke udelade grupperinger

```
% puts Hvad er hovedstaden i København
wrong # args: should be "puts ?-nonewline? ?channelId? string"
```

Sammensætning af strenge

For at generere HTML kode, så er det essentielt, at vi kan manipulere med strenge.

Kommandoen append bruges til at sammensætte strenge:

```
append varname arg1 arg2 ... argN
```

Variablen varname sættes til sammensætningen af

- variabelens gamle værdi og
 - argumenterne arg1, arg2, ..., argN
- ```
% set message "Hello"
Hello
% append message " " "World"
Hello World
```

Hvis variablen ikke allerede er oprettet, så opretter append variablen:

```
% set ny_var
can't read "ny_var": no such variable
% append ny_var "En " " " "variabel oprettes"
En variabel oprettes
% set ny_var
En variabel oprettes
```

## Gruppering og append

Er eksplicit gruppering altid nødvendig ved append?

```
% append ord Her følger nogle flere ord
Herfølernogleflereord
% set ord
Herfølernogleflereord
```

Det virker, men mellemrum forsvinder.

Pointen er, at append tager et vilkårligt antal argumenter!

Hvert ord opfattes som et argument.

Vil vi have mellemrum, så må vi angive dem med grupperingstegn:

```
% append ord.igen Her " " følger " " nogle " " flere " " ord
Her følger nogle flere ord
% append ord.igen? Her { } følger { } nogle { } flere { } ord
Her følger nogle flere ord
```

## Programmer skal kunne vælge – If-sætninger

If-sætninger bruges til at udføre forskellig kode på baggrund af en test:

```
if {$saldo >= $udtraek} {
 set saldo [expr $saldo - $udtraek]
 puts "Saldo efter fratræk af $udtraek kr. er $saldo"
} else {
 puts "Saldoen $saldo er desværre mindre end det du vil hæve $udtraek"
}
```

Et andet eksempel (kroner.tcl):

```
set kroner_i_lommen 1
if { $kroner_i_lommen == 1 } {
 puts "Jeg har 1 krone i lommen."
} else {
 puts "Jeg har $kroner_i_lommen kroner i lommen."
}
```

Vores Web-services skal helst generere grammatisk korrekte tekster.

Hvordan skriver man ofte ovenstående – den lette og ikke ligeså gode løsning?

## Eksempel: Hæve på min bankkonto (saldo.tcl)

```
set saldo 1000;
set udtraek 100;
set saldo [expr $saldo - $udtraek]
puts "Saldo efter fratræk af $udtraek kr. er $saldo"
set udtraek 550;
set saldo [expr $saldo - $udtraek]
puts "Saldo efter fratræk af $udtraek kr. er $saldo"
```

Hvordan ser lageret ud?

Vilker programmet stadig, hvis jeg hæver yderligere kr. 500?

## Else-grener kan udelades (saldo2.tcl)

```
set saldo 1000;
set udtraek 100;
if {$saldo >= $udtraek} {
 set saldo [expr $saldo - $udtraek]
 puts "Saldo efter fratræk af $udtraek kr. er $saldo"
}
set udtraek 950;
if {$saldo >= $udtraek} {
 set saldo [expr $saldo - $udtraek]
 puts "Saldo efter fratræk af $udtraek kr. er $saldo"
}
```

Hvad udskriver programmet?

### if - elseif - else konstruktionen

Man kan lave mere end to valg med if-kommandoen.

#### Beregning af brevporto (brevporto.tcl)

```
set vaegt 45
puts "Du indtastede en vægt på $vaegt gram"
if {$vaegt < 0} {
 puts "Du kan ikke sende breve med en negativ vægt!"
} elseif {$vaegt <= 20} {
 puts "Taksten er 4,00 kr."
} elseif {$vaegt <= 50} {
 puts "Taksten er 5,25 kr."
} elseif {$vaegt <= 100} {
 puts "Taksten er 5,75 kr."
} elseif {$vaegt <= 250} {
 puts "Taksten er 9,75 kr."
} elseif {$vaegt <= 500} {
 puts "Taksten er 17,00 kr."
} elseif {$vaegt <= 1000} {
 puts "Taksten er 21,00 kr."
} elseif {$vaegt <= 1500} {
 puts "Taksten er 28,00 kr."
} elseif {$vaegt <= 2000} {
 puts "Taksten er 30,00 kr."
} else {
 puts "Du kan ikke sende et så tungt brev!"
}
```

### While-løkker (while1.tcl)

Hvorledes kan vi udskrive "I Love web-programming" 200 gange?

#### Dårlig løsning:

```
puts "I Love web-programming"
...198 gange...
puts "I Love web-programming"
```

#### God løsning: While-løkker bruges til repetition:

```
set counter 200
while { $counter >= 1 } {
 puts "I Love web-programming"
 set counter [expr $counter - 1]
}
```

Kommandoen puts udløber 200 gange, hvor counter = 200, 199, ..., 1.

I hvilken rækkefølge udløres kommandoerne?

Er det vigtigt, at variablen counter nedtælles i hvert gennemløb af løkken?

### Kommandoen if

Generelt format:

```
if {test1} {
 kommandoer1
} elseif {test2} {
 kommandoer2
} else {
 kommandoer3
}
```

Virkning:

1. beregn værdien af test1.
2. hvis 1 så udlør kommandoer1, ellers
3. beregn værdien af test2
4. hvis 1 så udlør kommandoer2, ellers
5. udlør kommandoer3

Man kan have et vilkårligt antal elseif sekvenser, se f.eks. brevporto.tcl.

Der er mange former for tests: <, >, <=, !=, ==, ...

### While-løkker - fortsat (while2.tcl)

Lad os prøve at addere alle talene fra 0 til 10:

```
set res 0
set c 10
while { $c >= 0 } {
 set res [expr $res + $c]
 set c [expr $c - 1]
}
puts "c = $c"
puts "res = $res"
```

| c  | res |
|----|-----|
| 10 | 0   |
| 9  | 10  |
| 8  | 27  |
| 7  | 44  |
| 6  | 61  |
| 5  | 78  |
| 4  | 95  |
| 3  | 112 |
| 2  | 129 |
| 1  | 146 |
| 0  | 163 |

## Syntaks for while-løkker

Generelt format:

```
while {test} {
 kommandoer
}
```

Den virker sådan:

(1) udregn test

(2) hvis 1 (dvs. sand), så udfør *kommandoer*, og fortsæt ved (1)

Dvs. `while`-løkken kører så længe betingelsen (*test*) er sand.

Indmaden i `while` kaldes *løkketroppen*.

Oftest anvendt `while`-konstruktion, se f.eks. `while2.tcl`:

```
initialisering
while {test} {
 kommandoer
 optælling
}
```

## Eksempler på while-løkker (while3.tcl)

Standard-løkke, hvor i gennemløber \_\_\_\_\_:

```
set i 10
while {$i >= 1} {
 incr i -1
}
```

puts "Løkkeeksempel 1: \$i"

Standard-løkke, hvor i gennemløber \_\_\_\_\_:

```
set i 1
while {$i <= 10} {
 incr i 2
}
puts "Løkkeeksempel 2: $i"
```

Standard-løkke, hvor i gennemløber \_\_\_\_\_:

```
set i 1
while {$i <= 100} {
 set i [expr $i * 2]
}
puts "Løkkeeksempel 3: $i"
```

Hvilken værdi indeholder i efter hver løkke?

## Optælling med incr

Optælling af en variabel kan ske med `set` og `expr`, f.eks.

```
% set i 10
10
% set i [expr $i - 1]
9
```

En kortere notation for at optælle en variabel sker med kommandoen `incr`.

Første argument er den variabel som skal optælles – den optælles med 1 hvis der kun er et argument:

```
% set i
9
% incr i
10
```

Et eventuelt andet argument angiver hvormeglet variabelen skal optælles (eller nedtælles):

```
% set i
10
% incr i -1
9
% incr i 5
14
```

Hvilken værdi indeholder variabelen `i`?

## Opgaver (while4.tcl)

Skriv en `while`-løkke der udskriver 64, 32, 16, 8, 4, 2, 1:

```
set i ____
while {____ >= ____} {
 puts "$i"
 set i [expr ____ / ____]
}
```

Skriv en `while`-løkke der udskriver 2, 4, 6, 8, ..., 100:

```
set i ____
while {____ <= ____} {
 puts "$i"
 incr i ____
}
```

Skriv en `while`-løkke der udskriver 100, 110, 120, ..., 200:

```
set i ____
while {$i <= ____} {
 puts "$i"
 incr i ____
}
```

Det er muligt at lave sine egne kommandoer (proc1.tcl):

```
proc italic { arg } {
 return "<i>${arg}</i>"
}
```

```
puts "I [italic really] like coffee"
```

Proc kommandoen tager tre argumenter:

- procedurenavn
- argumentliste
- procedure krop

Man navngiver selv sine procedurer (dvs. kommandoen):

Man bestemmer selv hvilke argumenter kommandoen forventer.

Man bestemmer selv hvilke andre kommandoer, som skal stå i procedure kroppen.

Intuitivt: Kommandoen `italic` navngiver koden, her `return` ... således at den kode udføres hver gang man refererer til den, via navnet `italic`.

**Husk:** Der er forskel på små og store bogstaver (eng. case sensitive!)

### Procedurer - fortsat (proc2.tcl)

En procedure returnerer en værdi, angivet ved `return`.

Hvad udskriver programmet:

```
proc f {} {
 puts "I"
 return 1
}
proc g {} {
 puts "E"
 return [f]
}
proc main {} {
 puts "H"
 set res [expr [g] + [f]]
 puts "O"
 puts $res
}
main
```

### Tcl-programmer i filer

Kommandoen `source` kan bruges til at hente Tcl programmer ind i tclsh

```
% source file.tcl
```

### Kommentarer

Kommentarer i Tcl-filer skal starte med et #-tegn som første tegn på en linje:

```
% # dette er en kommentar
```

eller efter et semikolon som afslutter kommandoen inden kommentaren:

```
% set year 2001; # Current year is 2001
2001
```

### Eksempel: HTML-utilities (html\_utilities.tcl)

```
proc urllink { url name } {
 return "${name}"
}
proc italic { arg } {
 return "<i>${arg}</i>"
}
proc htmlpage { title body } {
 return "<html>
<head><title>${title}</title></head>
<body>
$body
</body>
</html>"
}
```

### Eksempel: HTML-utilities - fortsat (html\_utilities.tcl)

Filen html\_utilities.tcl hentes ind i tc1sh forløkkeren:

```
% source html_utilities.tcl
% set w2_url "http://www.it.edu/courses/W2/E2001/"
http://www.it-c.dk/courses/W2/E2001
% htmlpage "Kursets hjemmeside" "Se [urlink $w2_url "DWEB"]"
<html>
<head><title>Kursets hjemmeside</title></head>
<body>
Se DWEB
</body>
</html>
%
```

Tegnstringen som htmlpage returnerer kan vises i en browser:

Med ns\_return kan vi lave vores første dynamiske web-side (dyn\_web\_page.tcl).

```
set page [htmlpage "Kursets hjemmeside" "Se [urlink $w2_url "DWEB"]"]
ns_return 200 text/html $page
```

### Øvelse 1 – Kommentarservice

Dine to hjemmesider, f.eks.:

1. www.it-c.dk/courses/W2/E2001/Lb/1b1.html
2. www.it-c.dk/courses/W2/E2001/Lb/1b2.html

Dine kommentarlink bliver (som indsættes i den side du vil kommentere)

```
http://greenspun.com/dwebE2001/Lb/1b1.html
og
```

```
http://greenspun.com/com/dwebE2001/Lb/1b2.html
```

På Loquacious er der i databasen registreret følgende:

```
dwebE2001 = www.it-c.dk/courses/W2
```

Informationen dwebE2001/Lb/1b1.html og dwebE2001/Lb/1b2.html identificerer nu henholdsvis 1 og 2 ovenfor.

Jeg kan kommentere alle sider under www.it-c.dk/courses/W2/E2001/