

### Forelæsning 3

#### Videre med Tcl og dynamiske web-sider

- For-løkker
- Aritmetiske operatorer
- Procedurer — igen
- Teknologier for sites som er programmer
- Tcl på web-serveren
- Overførsel af data fra HTML dokumenter til Tcl-programmer
- Eksempel: Valutahandel
- Håndtering af fejl i web-scripts

#### For-løkker — fortsat

En For-løkke har denne form:

```
for {initialisering} {test} {optælling} {  
    kommandoer  
}
```

Den virker sådan:

- (1) Udtør *initialisering*.
- (2) Udregn *test*; hvis 0, (dvs. falsk) fortsæt efter løkken.
- (3) Udtør løkke-kroppen *kommandoer*.
- (4) Udtør *optælling*.
- (5) Fortsæt med punkt 2.

Kommandoerne under For kaldes *løkke-kroppen*.

#### For-løkker — en kortere notation for while-løkker (For1.tcl)

Med For-løkker kan man det samme som med while-løkker.

Hverken mere eller mindre!

while1.tcl:

```
set counter 200  
while { $counter >= 1 } {  
    puts "I love web-programming"  
    incr counter -1  
}
```

omskrevet til en for-løkke:

```
for {set counter 200} { $counter >= 1 } {incr counter -1} {  
    puts "I love web-programming"  
}
```

I hvilken rækkefølge udføres kommandoerne?

I vælger selv om I vil anvende while- eller for-løkker.

while-løkker er oftest mere intuitive for begyndere.

#### Eksempler (For2.tcl)

Standard-løkke, hvor programmet udskriver ...

```
for {set i 10} { $i >= 1 } {incr i -1} {  
    puts " $i "  
}
```

Standard-løkke, hvor programmet udskriver ...

```
for {set i 1} { $i <= 10 } {incr i 2} {  
    puts " $i "  
}
```

Standard-løkke, hvor programmet udskriver ...

```
for {set i 1} { $i <= 100 } {set i [expr $i * 2]} {  
    puts " $i "  
}
```

### Opgaver (For 3. tc1)

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

```
for { set i ___ } { ___ >= ___ } { set i [expr ___ / ___] } {  
  puts "$i"  
}
```

Skriv en løkke, hvor der udskrives 2, 4, 6, 8, ..., 100:

```
for { set i ___ } { ___ <= ___ } { incr i ___ } {  
  puts "$i"  
}
```

Skriv en løkke, hvor der udskrives 100, 110, 120, ..., 200:

```
for { set i ___ } { ___ <= ___ } { incr i ___ } {  
  puts "$i"  
}
```

### Indlejede løkker — lad os lave en multiplikationstabel (mul. tc1)

```
for { set i 1 } { $i < 10 } { incr i } {  
  set line ""  
  for { set j 1 } { $j < 10 } { incr j } {  
    append line " [expr $i * $j]"  
  }  
  puts $line  
}
```

Den ydre løkke kører 9 gange,  $i = 1, 2, \dots, 9$ .

For hvert  $i$  kører den indre løkke med  $j = 1, 2, \dots, 9$ .

Hvor mange gange udløres kommandoen `append` ?

### En For-løkke kan altid omskrives til en while-løkke

```
for {initialisering} {test} {optælling} {  
  kommandoer  
}
```

omskrevet til while:

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

### Aritmetiske operatører og deres udregningsrækkefølge (præcedens)

Operator	Betydning	Eksempler	
*	Multiplikation	1.5 * 60	24 * 60
/	Division	134.0 / 60	134 / 60
%	Rest	—	134 % 60
+	Addition	1.1 + 60	14 + 60
-	Subtraktion	1.1 - 60	134 - 60

Operatører med høj præcedens binder stærkere end operatører med lav præcedens, og udregnes først.

Operatørerne `*`, `/` og `%` har højere præcedens end `+` og `-` og udregnes altså før `+` og `-`.

Når operatørerne har samme præcedens (binder lige stærkt) regnes fra venstre mod højre.

Eksempel: Hvilken værdi og type har følgende udtryk?

Aritmetisk udtryk	Resultatværdi
1.5 * 60	
150.0 / 60	
150 / 60	
134 % 60	
1.1 + 60	
1.1 - 60	

Vi kan lave vilkårligt komplicerede udtryk, f.eks.:

$$22 - 34 + 43 \% 34 * 23 + 122 / 43.22 * 23 + 43 \text{ der giver } 302.92364646$$

Uden præcedensregler kan udtryk være tvetydige: vil  $2+4*4$  give 24 eller 18?

Efter som \* har højere præcedens end +, udregnes  $4*4$  først, og derefter lægges 2 til.

Hvis vi vil lægge sammen først, skal vi benytte parenteser:  $(2+4) * 4$  giver 24.

Efter som \* og / har samme præcedens, regnes fra venstre mod højre:  $2*5/2$  giver 5

Vi kan dividere først ved at sætte en parentes:  $2 * (5/2)$  giver 4

Eksempel

Lad x have værdien 2 og y have værdien 4.

Logisk udtryk	Resultatværdi
$1 == 1$	
$\$x != \$y$	
$\$x < 3 + \$y$	
$(\$x + \$y > 3) == 0$	
$0 != \$x < 3$	
$\$x == \$y == 0$	

Tallet 1 svarer til sand og tallet 0 svarer til falsk.

### Sammenligningsoperatorer og deres præcedens

Operator	Betydning	Eksempel
<	Mindre end	$\$x < 60$
<=	Mindre end eller lig med	$\$x <= 60$
>	Større end	$\$x > 60$
>=	Større end eller lig med	$\$x >= 60$
==	Lig med	$\$x == 60$
!=	Forskellig fra	$\$x != 60$

De aritmetiske operatorer \*, /, %, +, - binder alle stærkere end sammenligningsoperatorerne <, <=, >, >=.

Sammenligningsoperatorerne <, <=, >, >= binder alle stærkere end == og !=.

### Logiske operatorer og deres præcedens

Operator	Betydning	Eksempel
!	Ikke (Negation)	$!(\$x == 60)$
&&	Og (Konjunktion)	$0 <= \$x \ \&\& \ \$x < 60$
	Eller (Disjunktion)	$\$x < 0 \    \ \$x >= 60$

Operatoren ! binder stærkere end && som binder stærkere end ||.

! binder også stærkere end sammenligningsoperatorerne og de aritmetiske operatorer.

&& og || binder svagere end sammenligningsoperatorerne og de aritmetiske operatorer.

Der udregnes fra venstre mod højre.

Hvis *udtryk1* er falsk i *udtryk1 && udtryk2* så udregnes *udtryk2* ikke.

Hvis *udtryk1* er sandt i *udtryk1 || udtryk2* så udregnes *udtryk2* ikke.

### Eksempel: Hvad er resultatet af følgende logiske udtryk?

Lad  $x = 2$  og  $y = 4$ .

Boolsk udtryk	Resultatværdi
$i = 0$	
$i = 1$	
$i = 1 == 0$	
$i (1 == 0)$	
$1 \&\& 0$	
$0     1$	
$\$x + \$y > 3 \&\& \$x < \$y$	
$\$x + \$y == 3    \$x < 4$	

Husk, at 1 svarer til sand og 0 til falsk.

### Returværdier fra procedurekald

Hvis en procedure ikke afsluttes med en `return`-kommando returneres resultatet af den i proceduren sidste udlørte kommando.

Eksempel:

```
proc add { a b } {  
  return [expr $a + $b]  
}
```

kan altså også skrives:

```
proc add { a b } {  
  expr $a + $b  
}
```

Det anbefales altid at skrive `return`, således at den værdi der returneres angives eksplicit – programmet er da nemmere at læse.

### Procedurer igen (`multi.tcl`)

Proceduren `multi` tager to parametre, en streng `word` og et tal `n`

```
proc multi {word n} {  
  set res ""  
  while {$n >= 1} {  
    append res $word  
    set n [expr $n - 1]  
  }  
  return $res  
}
```

Nedenfor kaldes `multi` med argumenterne "All work and no play makes Jack a dull boy." og 348:

```
puts [multi "All work and no play makes Jack a dull boy." 348]
```

### Forskellen på brug af `puts` og `return` (`gang_med_to.tcl`)

Eksempel:

```
proc gang_med_to { tal } {  
  return [expr 2 * $tal]  
}  
set a [gang_med_to 230]  
puts "Variablen 'a' er sat til værdien '$a' "
```

Udskiftes `return` med `puts` i proceduren så bliver variablen `a` sat lig den tomme streng ("") fordi `gang_med_to` så returnerer den tomme streng:

```
proc gang_med_to { tal } {  
  puts [expr 2 * $tal]  
}
```

Procedure-kroppen udløres først når proceduren kaldes!

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

```
proc urlLink { url name } {
    return "<a href=\" $url \">$name</a>"
}

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

proc htmlPage { title body } {
    return "<html>
<head><title>$title</title></head>
<body>
    $body
</body>
</html>"
}
```

#### Teknologier for Sites som er Programmer

- Beslut om du bygger dynamiske dokumenter eller programmer med et web-interface
- Vælg et programmeringssprog (C, Tcl, Java, ML, ...)
- Vælg en programinvokeringmekanisme (CGI-scripts, webserver-fortolkning)
- Vælg en web-server som understøtter ovenstående tre valg

#### Programmer med et web-interface

- Programmer som genererer HTML kode — site generation
- Server-fortolkede programmer som genererer HTML kode (f.eks. Tcl på AOLserver) — dynamic sites

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

Flen html\_utilities.tcl hentes ind i tclsh fortolkeren:

```
% source html_utilities.tcl
% set w2_url "http://www.it.edu/courses/W2/"
http://www.it-c.dk/courses/W2/
% htmlPage "Kursets hjemmeside" "Se [urlLink $w2_url "DWEB"]"
<html>
<head><title>Kursets hjemmeside</title></head>
<body>
Se <a href="http://www.it-c.dk/courses/W2/">DWEB</a>
</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 [urlLink $w2_url "DWEB"]"]
ns_return 200 text/html $page
```

#### HTML dokumenter med dynamisk indhold

AOLserver Dynamic Pages (adp-sider):

```
<html>
...
<% tcl-kommandoer %>
...
</html>
```

Microsoft's Active Server Pages (asp-sider):

```
<html>
...
<% VBScript eller JavaScript %>
...
</html>
```

PHP3:

```
<html>
...
<? PHP3 kode ?>
...
</html>
```

## Vælg et Programmeringssprog

### C

- ÷ usikkert og utypet
  - ÷ langsom udviklingscyklus (oversættelse)
- ### Tcl (Perl, PHP, ...)
- + lille og simpelt
  - + godt til at håndtere strenge
  - + hurtig udviklingscyklus (ingen oversættelse)
  - ÷ dynamisk typet; fejl fanges først når programmet kører
- ### Java
- ÷ stort og komplekst
  - + statisk typet; mange fejl fanges under oversættelse

## Vælg en Web-Server

### Microsoft's Internet Information Server (IIS)

- + understøtter .asp sider

### Apache

- + nem at udvide med moduler (til f.eks. fortolkning af Tcl-kode)

### AOLserver

- + indbygget Tcl-fortolker
- + god API (Application Programmer Interface)

## Vælg en Programinvokeringsmekanisme

### CGI-scripts

- + alle programmeringssprog kan i princippet bruges
- ÷ stor starttid for hvert program (skaleres dårligt)

### Programmer fortolket i serveren

- ÷ kun understøttede sprog kan bruges
- + lav opstarttid for hvert program

## Tcl på Web-serveren

- En side med endelsen **.tcl** bliver fortolket af web-serveren når en klient forespørger siden.
- Tcl-programmer på web-serveren må **ikke** gøre brug af **puts** kommandoen.
- I stedet bruges kommandoen **ns\_return** til at sende en HTML-side tilbage til en browser:  
`ns_return 200 text/html " . . . "`
- En side med endelsen **.html** bliver sendt uændret tilbage til klienten der forespørger siden.

### Et simpelt web-server program (simple.tcl)

Følgende program returnerer en simpel HTML-side til browseren:

```
set name "Martin Elsmann"
set email "mael@it-c.dk"
ns_return 200 text/html "<html>
<head>
<title>Home page for $name</title>
<body>
Home page for $name <p> <hr>
<a href="\mailto:$email\">$email</a>
</body>
</html>"
```

Prøv `http://hug.it.edu:8002/F2002/lec3/simple.tcl`

### Overførsel af data fra HTML dokument til Tcl program

```
my_first.html:
<html>
</body>
<form method=post action=my_first.tcl>
<input type=text name=login>
</form>
</body>
</html>

my_first.tcl:
set_form_variables
# Nu er variabelen login tilgængelig
ns_return 200 text/html "<html>
<body>
Du skrev $login i tekstboksen på den forrige HTML side.
</body>
</html>"
```

Kommandoen `set_form_variables` erklærer en variabel svarende til hver `input-felt` i HTML formen.

Prøv `http://hug.it.edu:8002/F2002/lec3/my_first.html`

### Valutahandel (valutahandel.html)

For at bygge en valutahandelservice kræves kun to filer.

```
<html>
<head>
<title>Valutahandel</title>
</head>
</body>
<form method=post action=valutahandel.tcl>
Hvor mange kroner vil du veksle til dollars? <p>
<input type=text size=10 name=kroner>
<input type=submit value="Beregn" >
</form>
</body>
</html>
```

Prøv `http://hug.it.edu:8002/F2002/lec3/valutahandel.html`

### Valutahandel — fortsat (valutahandel.tcl)

```
# set the form-variable 'kroner'
set_form_variables

# compute dollar amount
set dollarkurs 8.34
set dollars [expr ($kroner - 20.0) / $dollarkurs]

ns_return 200 text/html "<html>
<head>
<title>Valutahandel - resultat</title>
</head>
<body bgcolor=white>
<center>
<h1>Valutahandel - resultat</h1>
For $kroner kroner modtager du $$dollars <p>
<a href="\valutahandel.html\">Tilbage</a>
</center>
</body>
</html>"
```

## Valutahandel — fortsat

Hvilke problemer er der med valutahandel-serviceen?

- Hvad sker der når brugeren indtaster et beløb på mindre end kr. 20?
- Hvad sker der hvis brugeren ikke indtaster et tal?

## Web-server fejlhåndtering

- Ved "Server Error" ser du en fejlmeddelelse i din browser:  
`http://hug.it.edu:8002/F2002/lec3/fej1.tcl.`
- "Uendelig løkke" på web-serveren:  

```
set i 10
while { $i >= 0 } {
  set i 1
}
```

Genstart web-serveren ved brug af "Web Server Services":

```
http://hug.itu.dk:8077/webserver.html
```