

Programovací jazyk TCL

Pavel Tišnovský
tisnik@centrum.cz

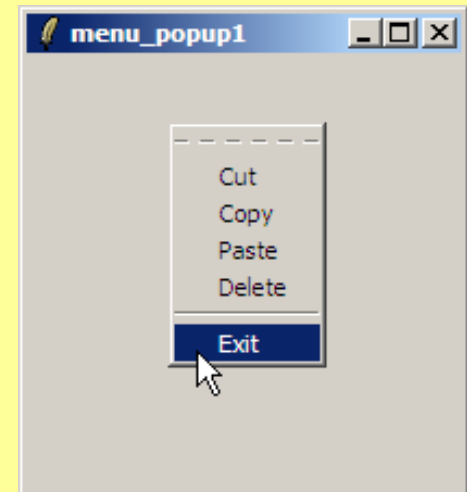
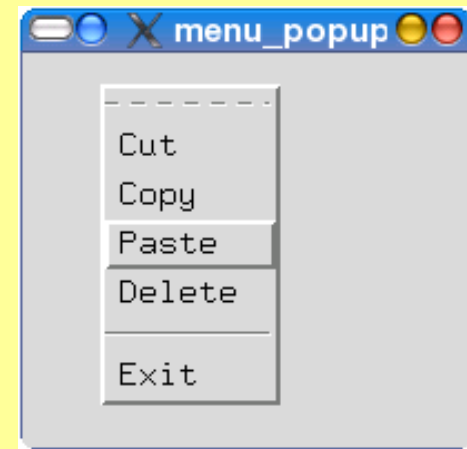
Obsah kurzu

1. Základní informace o Tcl/Tk
2. Instalace a spuštění interpretru Tcl
3. Syntaxe a sémantika jazyka Tcl
4. Textové substituce
5. Matematické a logické operace
6. Tvorba uživatelských funkcí
7. Řídicí konstrukce (větvení a cykly)
8. Seznamy, asociativní pole a řetězce
9. Regulární výrazy
10. Příkaz switch
11. Práce se soubory
12. Odkazy na Internetu

Základní informace o Tcl/Tk

Tcl/Tk

- **Tcl** (Tool Command Language)
 - John Ousterhout
 - interpretovaný programovací jazyk
 - podpora prototypových aplikací
 - zabudování interpreteru i do dalších aplikací
- **Tk** (ToolKit)
 - John Ousterhout
 - knihovna pro práci s grafickým uživatelským rozhraním
 - widgety a kontejnery
 - přenositelné na další platformy



Základní vlastnosti Tcl

- Ideově vychází z Lispu:
 - přehlednější zápis algoritmů
 - poněkud složitější syntaxe
 - pravidla pro textové substituce
 - samotný jazyk obsahuje pouze velmi malé množství syntaktických prvků
- Vykonávání programů napsaných v Tcl
 - Původně interpreter
 - Dnes existují i překladače

Přednosti jazyka Tcl

- Jednoduchost tvorby aplikací a prototypů
- Dobrá návaznost na další programovací jazyky
 - embedded Tcl
 - Tcl jako „lepidlo“ dalších skriptů a utilit
- Portabilita
 - Unixové systémy
 - Microsoft Windows
 - Mac OS X

Zápory Tcl

- Nižší rychlost provádění skriptů
 - ovšem interpretry se postupně zlepšují
- Nepřítomnost plného IDE+WYSIWYG pro Tk
- Nedostatečná podpora OOP
 - „je to kýžená vlastnost nikoli chyba“
- V současnosti není o Tcl takový zájem jako v minulosti
 - menší podpora ze strany OS
 - „konkurenční“ programovací jazyky
 - Python, Ruby, Perl (částečně i BASH, AWK...)

Instalace a spuštění interpretru Tcl

Instalace Tcl

- Oficiální adresa:
 - <http://www.tcl.tk/>
- Microsoft Windows:
 - <http://www.activestate.com/>
- Apple:
 - <http://www.tcl.tk/software/mac/>
- Linuxové distribuce:
 - většinou existuje jako předpřipravený balíček (balíčky)
 - možnost ruční kompilace ze zdrojových kódů

Instalace Tcl – Fedora, RHEL...

```
~/ $ sudo yum install tcl tkinter
```

```
Loaded plugins: refresh-packagekit
```

```
Setting up Install Process
```

```
Resolving Dependencies
```

```
--> Running transaction check
```

```
---> Package tcl.i386 1:8.5.3-1.fc10 set to be updated
```

```
---> Package tkinter.i386 0:2.5.2-1.fc10 set to be updated
```

```
--> Processing Dependency: libtk8.5.so for package:
```

```
tkinter-2.5.2-1.fc10.i386
```

```
--> Processing Dependency: libTix.so for package: tkinter-2.5.2-1.fc10.i386
```

```
--> Running transaction check
```

```
---> Package tix.i386 1:8.4.3-1.fc10 set to be updated
```

```
---> Package tk.i386 1:8.5.3-5.fc10 set to be updated
```

```
--> Finished Dependency Resolution
```

```
...
```

```
Installing      : 1:tcl-8.5.3-1.fc10.i386                1/4
```

```
Installing      : 1:tk-8.5.3-5.fc10.i386                 2/4
```

```
Installing      : 1:tix-8.4.3-1.fc10.i386                3/4
```

```
Installing      : tkinter-2.5.2-1.fc10.i386              4/4
```

```
...
```

```
Complete!
```

Spuštění interpretru Tcl

- tclsh
 - řádkový interpret
- wish
 - grafický shell, ideální při tvorbě GUI
- tkcon
 - vylepšená konzole Tcl
 - většinou je nutné ji doinstalovat zvlášť (Linux)
 - `sudo yum install tkcon`

tclsh (konzole v Linuxu)



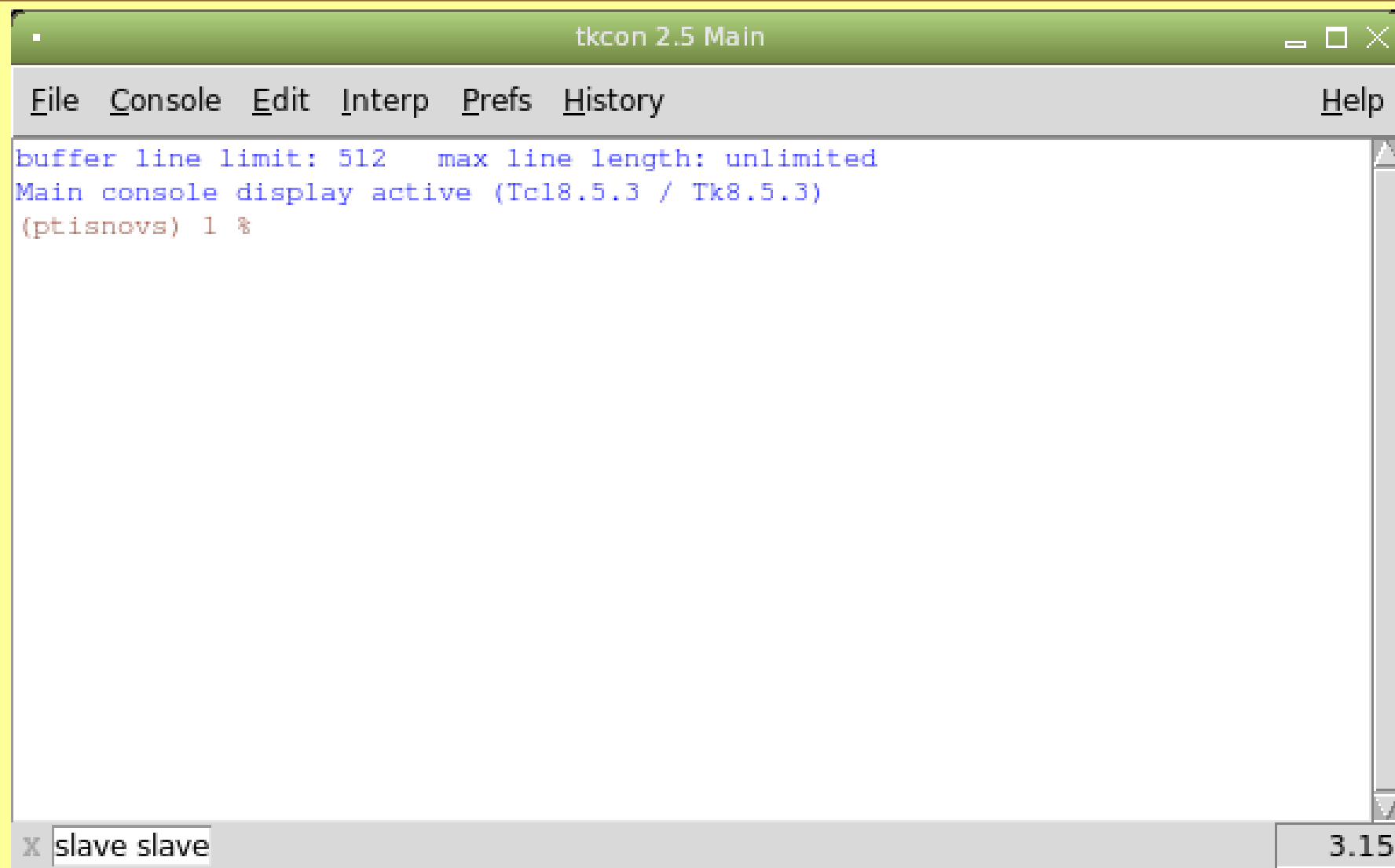
A screenshot of a Linux terminal window with a green title bar. The title bar contains the text 'dhcp-lab-190:~' and standard window control icons (minimize, maximize, close). The terminal has a black background with white text. The prompt 'dhcp-lab-190:~/ \$' is followed by the command 'tclsh'. The prompt changes to a '%' symbol. The user enters '% set a 10', and the terminal outputs '10'. Then the user enters '% set b 20', and the terminal outputs '20'. Next, the user enters '% expr \$a + \$b', and the terminal outputs '30'. Finally, the user enters '%', and the terminal shows a red cursor block. A mouse cursor is visible at the bottom center of the terminal window.

```
dhcp-lab-190:~/ $ tclsh
% set a 10
10
% set b 20
20
% expr $a + $b
30
% █
```

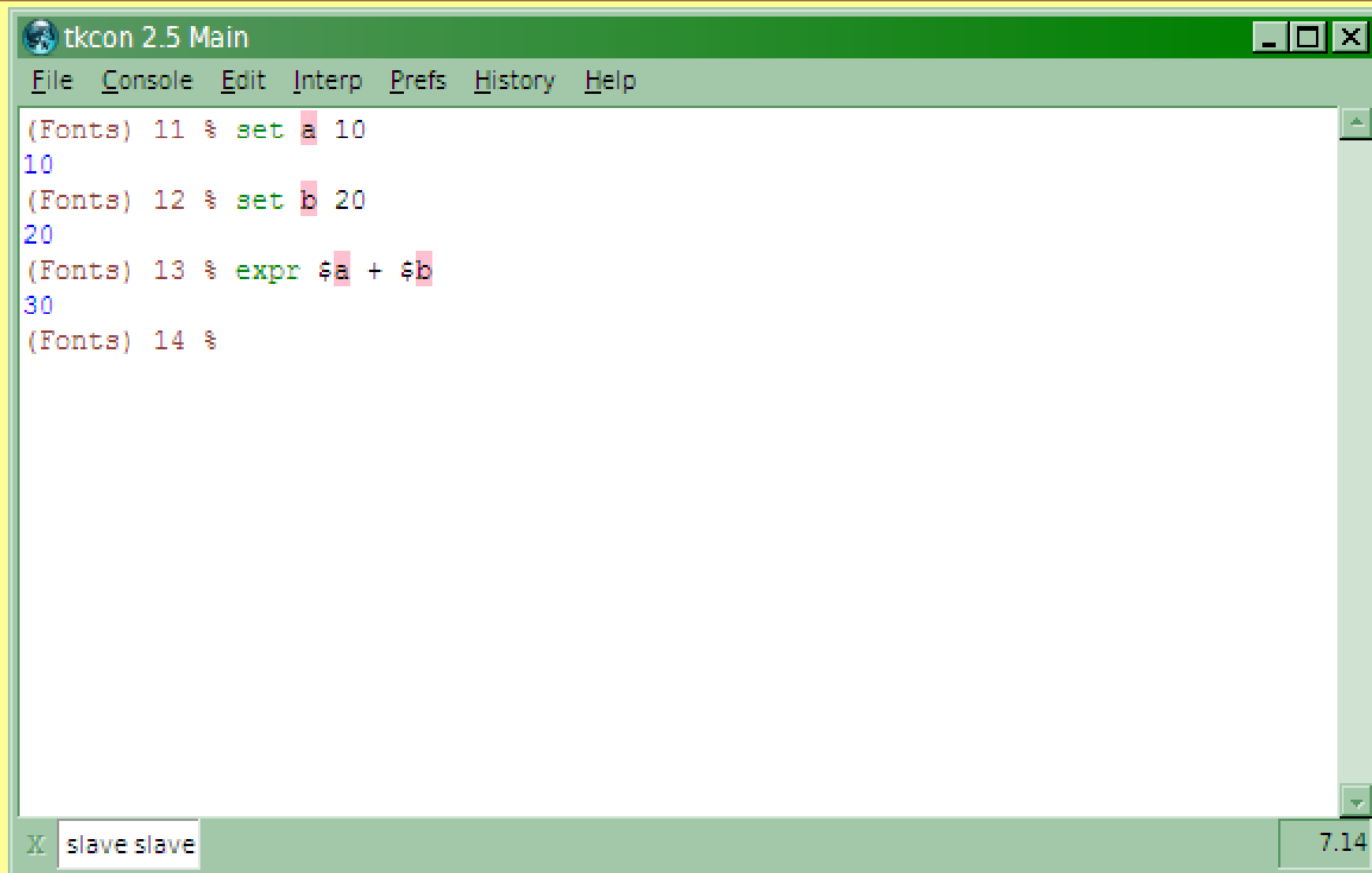
tclsh (konzole v MS Windows)

```
% set a 10
10
% set b 20
20
% expr $a + $b
30
% expr $a * $b
200
% _
```

tkcon (GUI v Linuxu)



tkcon (GUI v MS Windows)




The screenshot shows a window titled "tkcon 2.5 Main" with a menu bar containing "File", "Console", "Edit", "Interp", "Prefs", "History", and "Help". The main area is a text field containing the following Tcl commands and their outputs:

```
(Fonts) 11 % set a 10
10
(Font) 12 % set b 20
20
(Font) 13 % expr $a + $b
30
(Font) 14 %
```

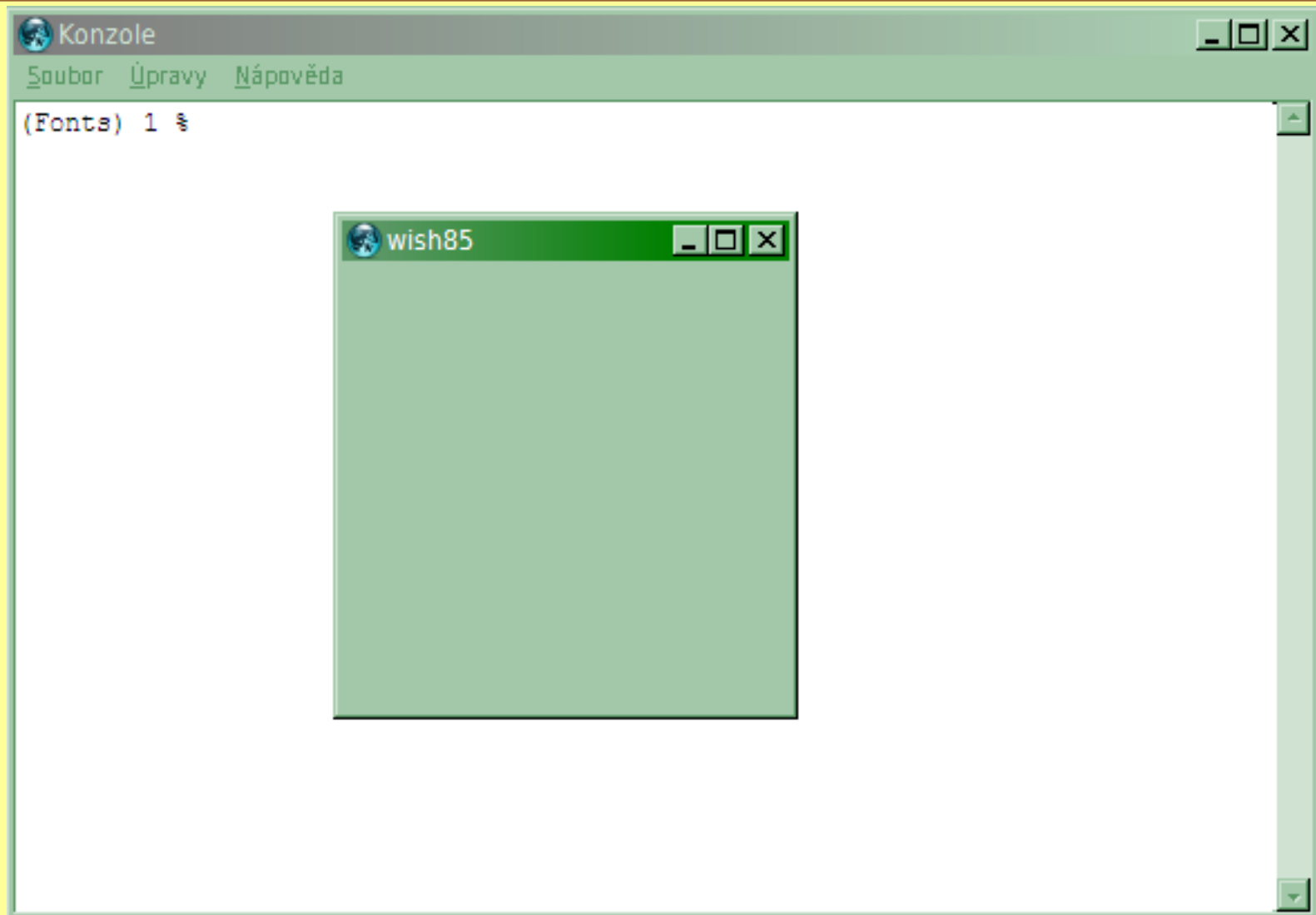
At the bottom left, there is a tab labeled "X slave slave". At the bottom right, the version number "7.14" is displayed.

wish (Linux)

```
dhcp-lab-190:~  
dhcp-lab-190:~/ $ tclsh  
% set a 10  
10  
% set b 20  
20  
% expr $a + $b  
30  
% exit  
dhcp-lab-190:~/ $ wish  
% button .b1 -text "Pokus"  
.b1  
% grid .b1 -row 0  
% button .b2 -text "Quit"  
.b2  
% grid .b2 -row 1  
% 
```

A small graphical window titled "Pokus" is displayed over the terminal. It has a green title bar with standard window controls (minimize, maximize, close). The window contains a single button labeled "Quit".

wish (MS Windows)



Syntaxe a sémantika jazyka Tcl

Skladba skriptu v Tcl

- Skript (program) se v Tcl skládá ze:
 - ze sekvence příkazů
 - příkazy jsou odděleny:
 - novým řádkem
 - středníkem (bodkočiarka)
- Příkaz v Tcl se skládá:
 - z jednoho nebo více slov oddělených mezerou
 - první slovo značí jméno příkazu
 - ostatní slova jsou argumenty příkazu
- Příkaz vrací **řetězec** jako svůj výstup

Barevné zvýraznění na následujících slajdech

- hnědá
 - poznámka
- modrá
 - zvýrazněný příkaz/výraz
- černá
 - běžný příkaz/výraz
- zelená
 - výstup či vrácená hodnota
- červená
 - chyba vypsaná interpretrem

Zápis příkazů (1)

`název_příkazu arg_1 arg_2... arg_n`

```
# vypíše řetězec "ahoj"  
puts "ahoj"
```

```
# vypíše řetězec "ahoj"  
puts ahoj
```

```
# vypíše řetězec "ahoj"  
puts {ahoj}
```

Zápis příkazů (2)

```
příkaz1 arg1..argn; příkaz2 ....
```

```
# toto je moje poznámka  
puts ahoj; # vypíše řetězec "ahoj"
```

```
puts hello; puts world!  
hello  
world!
```

Zápis příkazů (3)

vše je považováno za řetězec

```
set answer 42
```

```
set os Linux
```

```
set hello "Hello world"
```

```
set answer
```

```
set hello
```

```
set os
```

```
set xxx ; # chyba
```

```
can't read "xxx": no such variable
```

Proměnné

- Jméno proměnné
 - písmena
 - čísla
 - podtržítko (podčiarnik)
- Nastavení hodnoty proměnné
 - `set název_proměnné hodnota`
- Zrušení proměnné
 - `unset název_proměnné`
- Výpis hodnoty proměnné na std. výstup
 - `puts $název_proměnné`
- Přístup k hodnotě proměnné – znak `$`

Práce s proměnnými (1)

```
set answer 42
set os Linux
set hello "Hello world"
unset answer
unset hello
unset os
unset xxx
can't unset "xxx": no such variable

unset $answer
can't unset "42": no such variable
```

Práce s proměnnými (2)

název proměnné vs. hodnota proměnné

```
set answer 42  
set hello "Hello world"
```

řetězce nemusí být v uvozovkách!

```
puts answer  
puts hello  
puts $answer  
puts $hello
```

Upozornění – rozdíl mezi C a Tcl

Programovací jazyk C:

```
x= 10;  
y = x + 1;
```

Programovací jazyk Tcl:

```
set x 4  
set y x+10    (!!!)  
puts $x  
4  
puts $y  
x+10          (!!!)
```

Upozornění – „numerické“ hodnoty

```
set a 10;      puts $a; incr a; puts $a  
10  
11
```

```
set a 0xff;    puts $a; incr a; puts $a  
0xff  
256
```

```
set a 077;     puts $a; incr a; puts $a  
077  
64
```

Textové substituce

Substituce proměnných (1)

```
set answer 42
```

```
# substituce (též „interpolace“)  
puts "Odpověď je $answer"
```

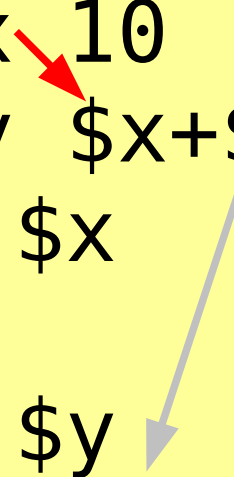
```
# výpis řetězce bez substituce  
puts {Odpověď je $answer}
```

```
# zabráníme interpretaci dolaru  
puts "Odpověď je \ $answer"
```

Substituce proměnných (2)

vzhledem k tomu, že se řetězce
nemusí zapisovat do uvozovek,
může docházet k zápisu, který se
vyhodnotí na řetězec aniž by se
hlásila chyba:

```
set x 10  
set y $x+$x+$x  
puts $x  
10  
puts $y  
10+10+10
```



Substituce příkazů (1)

sémanticky odpovídá bloku příkazů

příkaz1 argument argument

příkaz1 [příkaz2 argument
argument ...] argument ...

příkaz1 [příkaz2 argument [příkaz3
argument ...] argument ...] argument
...

Substituce příkazů (2)

```
set x 10
# zde se vyvolá příkaz uvnitř závorek
# a poté se použije jeho výsledek
# (expr bude vysvětleno dále)
set y [expr $x + 1]

# ověříme funkčnost
puts $x
10
puts $y
11
```

Substituce příkazů (3)

```
# oba typy substitucí lze samozřejmě  
# vzájemně kombinovat  
puts "$x + 1 se rovná: [expr $x + 1]"  
10 + 1 se rovná: 11
```

```
# ovšem zde jsou substituce zakázány  
puts {$x + 1 se rovná: [expr $x + 1]}  
$x + 1 se rovná: [expr $x + 1]
```

Textové substituce - shrnutí

- **\$** substituce proměnných
 - náhrada jména proměnné její hodnotou
- **[]** vyhodnocení příkazu
 - příkaz v závorkách se vyhodnotí nejdříve
- **""** zpracování mezer
 - potlačuje zpracování mezer jako argumentů příkazů
- **{ }** zákaz substitucí
 - stejné jako uvozovky s tím rozdílem, že se všechny substituce uvnitř závorek zakazují
- **** escape
 - ruší zvláštní význam následujícího znaku

Matematické a logické operace

Matematické operace

- Jazyk Tcl neobsahuje ve své syntaxi žádnou podporu pro zápis matematických výrazů!
- Funkce **expr**
 - podpora různých operátorů a funkcí
 - funkce se zapisují se závorkami okolo parametrů
 - vrací řetězec
 - jedná se o vypočtenou hodnotu výrazu
 - podpora ternárního C-čkového operátoru
 - při použití proměnných nutno použít znak dolaru \$

Příklady využívající funkci expr

```
expr 1 + 2
```

```
expr 10.0 / 5
```

```
expr 1 << 10
```

```
expr $a + $b
```

```
set a [expr 1+2]
```

```
set b 20
```

```
set c "$a + $b se rovná [expr $a + $b]"
```

```
3 + 20 se rovna 23
```

```
# zde nedojde k substituci
```

```
set c {$a + $b se rovná [expr $a + $b]}
```

```
$a + $b se rovna [expr $a + $b]
```

Operátory

Unární

+ unární plus
- unární mínus

Binární

+ součet
- rozdíl
* součin
/ podíl
% zbytek po dělení

Ternární

?:

Použití operátorů

```
puts [expr -1]
```

-1

```
puts [expr +1]
```

1

```
puts [expr 1+2]
```

3

```
puts [expr 1+2*3]
```

7

```
puts [expr (1+2)*3]
```

9

```
puts [expr 1<2 ? 1234 : 5678]
```

1234

Matematické funkce

acos	floor	sin
asin	fmod	sinh
atan	hypot	sqrt
atan2	log	tan
ceil	log10	tanh
cos	pow	

Použití matematických funkcí

```
puts [expr floor(1.0/2)]
```

```
0.0
```

```
puts [expr ceil(1.0/2)]
```

```
1.0
```

```
puts [expr log10(1000)]
```

```
3.0
```

```
puts [expr pow(2,10)]
```

```
1024.0
```

```
puts [expr atan2(1,0)*180/3.1415926]
```

```
90.00000153523452
```

Logické a bitové operace

<<	bitový posuv doleva
>>	bitový posuv doprava
~	negace bit po bitu
!	logická negace
&	bitový operátor AND
	bitový operátor OR
^	bitový operátor XOR
&&	logický součin
	logický součet

Použití logických a bitových operací

```
puts [expr true || false]
```

1

```
puts [expr true && false]
```

0

```
puts [expr !true]
```

0

```
puts [expr !false]
```

1

```
puts [expr 1 && 0]
```

0

```
puts [expr 1<<10]
```

1024

Relační operátory

< menší než
> větší než
<= menší nebo rovno
>= větší nebo rovno
== rovnost
!= nerovnost

Použití relačních operátorů

```
puts [expr 1<2]
```

```
1
```

```
# ovšem pozor!:
```

```
puts 1<2
```

```
1<2
```

```
set a 10; set b 20
```

```
puts [expr $a<$b]
```

```
1
```

```
puts [expr $a!=$b]
```

```
1
```

```
puts [expr $a==$b]
```

```
0
```

Pravdivostní hodnoty

```
puts [expr !! "yes"]
```

1

```
puts [expr !! "true"]
```

1

```
puts [expr !! "false"]
```

0

```
puts [expr !! "no"]
```

0

```
puts [expr !! "neco_jineho"]
```

can't use non-numeric string as operand of
"!"

```
puts [expr !! ""]
```

can't use empty string as operand of "!"

Tvorba uživatelských funkcí

Funkce v Tcl skriptech

- Vestavěné funkce
- Funkce vytvořené pomocí **proc**
- Externí funkce (C, ...)
- Použití **proc**
 - jméno_procedurey
 - argumenty
 - zákaz vyhodnocování - {}
 - tělo procedurey/funkce
 - zákaz vyhodnocování - {}
 - návratová hodnota – příkaz **return**

Vytvoření uživatelské funkce

```
proc hello {} {puts "Hello world"}
```

```
hello
```

```
Hello world
```

```
hello parametr
```

```
wrong # args: should be "hello"
```

otevírací složená závorka musí být na
stejném řádku, kde začíná definice funkce

```
proc hello2 {}
```

```
wrong # args: should be "proc name args  
body"
```

Funkce s návratovou hodnotou

```
proc dup {x} {return $x$x}
```

```
dup 10
```

```
1010
```

```
dup "hello "
```

```
hello hello
```

```
proc plus {a b} {
```

```
    return [expr $a+$b]
```

```
}
```

```
plus 10 20
```

```
30
```

Implicitní hodnota parametrů

```
proc decr {x {y 1}} {  
    expr $x-$y  
}
```

```
puts [decr 10]  
9
```

```
puts [decr 10 2]  
8
```

Variabilní počet parametrů

```
proc sum args {  
    set s 0  
    # args může být ve skutečnosti seznam!  
    foreach i $args {  
        incr s $i  
    }  
    return $s  
}
```

```
puts [sum 1]
```

1

```
puts [sum 1 2 3]
```

6

V Tcl je téměř vše považováno za řetězec

```
proc log_funkce {operator} {  
    puts [expr 0 $operator 0]  
    puts [expr 0 $operator 1]  
    puts [expr 1 $operator 0]  
    puts [expr 1 $operator 1]  
}
```

```
log_funkce &&
```

```
0
```

```
0
```

```
0
```

```
1
```

V Tcl je téměř vše považováno za řetězec (pokr.)

```
log_funkce ||
```

```
0
```

```
1
```

```
1
```

```
1
```

```
log_funkce ^
```

```
0
```

```
1
```

```
1
```

```
0
```

```
log_funkce +
```

```
0 1 1 2
```

Řídicí konstrukce

Řídicí konstrukce


- Větvení běhu programu
 - if-then
 - if-then-else
 - if-then-elseif-...-else
 - switch (probereme později)
- Cykly (programové smyčky)
 - while
 - for
 - foreach (probereme později)
 - + přidružené příkazy
 - break
 - continue

Větvení běhu programu

Příkaz if s jednou větví

```
# v podmínce není zapotřebí volat expr  
if {1<2} then {puts "1 je mensi 2"}  
1 je mensi 2
```

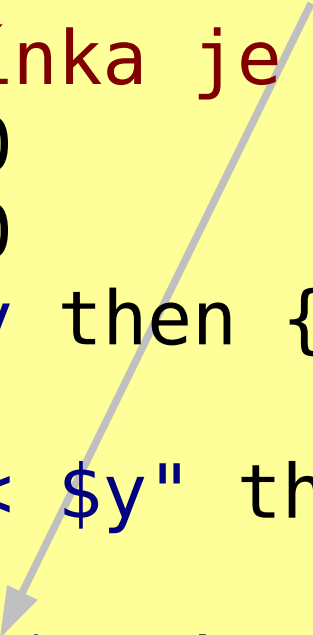
```
# slovo then není povinné  
if {1<2} {puts "1 je mensi 2"}  
1 je mensi 2
```



```
set x 1  
set y 2  
if {$x<$y} {puts "x je mensi nez y"}  
x je mensi nez y
```

Podmínka v „if“

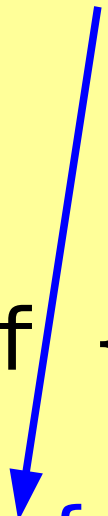
```
# složené závorky jsou nepovinné,  
# ale pozor na přebytečné mezery  
# (podmínka je prvním parametrem if)  
set x 10  
set y 20  
if $x<$y then {puts "mensi"}  
mensi  
if "$x < $y" then {puts "mensi"}  
mensi  
if $x < $y then {puts "mensi"}  
wrong # args: extra words after "else"  
clause in "if" command
```



Příkaz if s větví else

```
if {výraz} {  
    tělo_podmínky  
} else {  
    tělo_druhé_větve  
}
```

```
if {výraz} {  
    tělo_podmínky  
} {  
    tělo_druhé_větve  
}
```

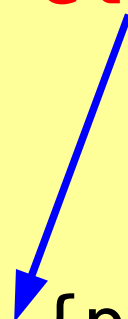


Příkaz if s větví else

```
if {$x<$y} {puts "x je mensi nez y"} else  
    {puts "x je vetsi nez y"}  
x je mensi nez y
```

```
set y 0  
if {$x<$y} {puts "x je mensi nez y"} else  
    {puts "x je vetsi nez y"}  
x je vetsi nez y
```

```
if {$x<$y} {puts "x je mensi nez y"} {puts  
    "x je vetsi nez y"}  
x je vetsi nez y
```



Příkaz if s více větve

```
set x 10
set y 20
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} else {puts "shodne"}
mensi
set y 0
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} else {puts "shodne"}
vetsi
set y 10
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} else {puts "shodne"}
shodne
```

Příkaz if s více větvemi (bez else)

```
set x 10
set y 20
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} {puts "shodne"}
```

mensi

```
set y 0
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} {puts "shodne"}
```

vetsi

```
set y 10
if {$x<$y} {puts "mensi"} elseif {$x>$y}
    {puts "vetsi"} {puts "shodne"}
```

shodne

Cykly (programové smyčky)

Programová smyčka „while“ (1)

```
set i 1
while {$i<10} {
    puts $i
    set i [expr $i+1]
}
```

```
set i 1
while {$i<10} {
    puts $i
    incr i
}
```

Programová smyčka „while“ (2)

```
# změna kroku
set i 2
while {$i<=20} {
    puts $i
    incr i 2
}
```

```
# zpětné počítání:
set i 10
while {$i} {
    puts $i
    incr i -1; # pozor, neexistuje decr!
}
```

Programová smyčka „while“ (3)

```
# výpočet faktoriálu
proc fact {n} {
    set result 1.0
    if {$n<0} {return 1}
    while {$n} {
        set result [expr $result*$n]
        incr n -1
    }
    return $result
}
```

```
fact 10
3628800.0
```

Programová smyčka „while“ (4)

```
# výpočet mocniny
proc power {base p} {
    set result 1
    while {$p} {
        set result [expr $result * $base]
        incr p -1
    }
    return $result
}
```

```
power 2 24
16777216
```

Programová smyčka „for“ (1)

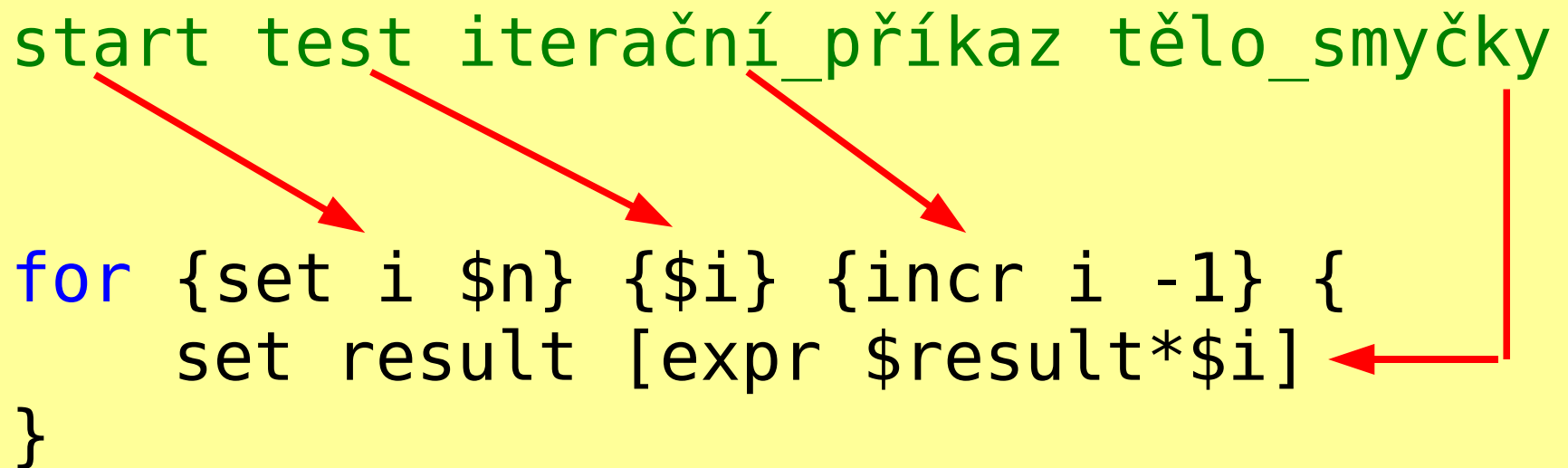
for start test iterační_příkaz tělo_smyčky

V reálných programech bývají všechny
čtyři části smyčky for zapsány ve
složených závorkách, aby se zamezilo
předčasnému vyhodnocení příkazů. Výpočet
faktoriálu:

```
proc fact2 {n} {  
    set result 1.0  
    for {set i $n} {$i} {incr i -1} {  
        set result [expr $result*$i]  
    }  
}
```

Programová smyčka „for“ (2)

for start test iterační_příkaz tělo_smyčky



```
for {set i $n} {$i} {incr i -1} {  
    set result [expr $result*$i]  
}
```

Řízení běhu programu ve smyčce

- break
 - ukončení programové smyčky
- continue
 - ukončení právě probíhající iterace

Seznamy, asociativní pole a řetězce

Seznamy

```
set seznam1 { 1 2 3 4 5 6 }  
set seznam2 { jedna dve tri ctyri pet sest }  
set seznam3 { 1 {2 3} {4 5} {6 7} {8} }
```

```
puts $seznam1  
1 2 3 4 5 6
```

```
puts $seznam2  
jedna dve tri ctyri pet sest
```

```
puts $seznam3  
1 {2 3} {4 5} {6 7} {8}
```

Seznamy a smyčka foreach

```
foreach i $seznam1 {puts $i}
```

```
foreach i {jedna dve tri ctyri} {  
    puts $i  
}
```

```
foreach {i j} {1 2 3 4 5 6} {  
    puts "i= $i"  
    puts "j= $j"  
}
```

```
foreach i {1 2 3} j {4 5 6} {  
    puts "i= $i"  
    puts "j= $j"  
}
```

Funkce pro práci se seznamy (1)

- **list**
 - vytvoření seznamu z argumentů, které jsou tomuto příkazu zadány
- **concat**
 - spojení dvou a více seznamů
- **llength**
 - získání počtu prvků v seznamu
- **split**
 - rozložení řetězce na seznam buď podle bílých znaků nebo podle specifikovaného oddělovače

Funkce pro práci se seznamy (2)

- **join**
 - vytvoření řetězce spojením prvků seznamu, mezi něž se může volitelně vložit oddělovač
- **lappend**
 - přidání jednoho či více prvků do seznamu
- **linsert**
 - vložení jednoho či více prvků na danou pozici (index)
- **lindex**
 - získání prvku ze seznamu na dané pozici (indexu)

Funkce pro práci se seznamy (3)

- **lreplace**
 - nahrazení prvků v seznamu
- **lrange**
 - vyjmutí více prvků ze seznamu (souvislá oblast)
- **lsearch**
 - hledání prvků v seznamu (možné i podle regulárních výrazů!)
- **lsort**
 - setřídění prvků v seznamu podle zadáných kritérií

Použití funkcí pro práci se seznamy (1)

```
set retezec "prvni,druha,treti,ctvrta"  
puts $retezec  
prvni,druha,treti,ctvrta
```

```
set seznam [split $retezec ","]
```

```
puts $seznam  
prvni druha tretí ctvrta
```

```
puts [llength $seznam]  
4
```

```
puts [join $seznam "- - -"]  
prvni- - -druha- - -tretí- - -ctvrta
```

Použití funkcí pro práci se seznamy (2)

```
set seznam {první druha treti ctvrta}
```

```
put [llength $seznam]
```

```
4
```

```
puts [lsearch $seznam "treti"]
```

```
2
```

```
puts [lsearch $seznam "*t*"]
```

```
2
```

```
puts [lsearch -all $seznam "*t*"]
```

```
2 3
```


Rozdíl mezi lappend a linsert

```
set seznam {první druha treti ctvrta}  
puts $seznam  
prvni druha treti ctvrta
```

```
# přidání prvku do stávajícího seznamu  
lappend seznam pata  
puts $seznam  
prvni druha treti ctvrta pata
```

```
# vytvoření nového seznamu s přidáním prvkem  
set seznam [linsert $seznam 0 nulta]  
puts $seznam  
nulta prvni druha treti ctvrta paty
```

Otočení prvků v seznamu

```
set a {prvni druhy treti ctvrty}
```

```
set b ""
```

```
set i [expr [llength $a] - 1]
```

```
while {$i >= 0} {  
    lappend b [lindex $a $i]  
    incr i -1  
}
```

```
puts $a
```

```
prvni druhy treti ctvrty
```

```
puts $b
```

```
ctvrty treti druhy prvni
```

Asociativní pole (1)

```
# celočíselné indexy jsou ve  
# skutečnosti řetězci
```

```
set pole(0) 100  
set pole(1) 150  
set pole(9) "foo bar baz"
```

```
puts $pole(0) 100  
puts $pole(1) 150  
puts $pole(9) foo bar baz
```

```
# pokus o čtení neinicializovaného prvku  
puts $pole(2)  
can't read "pole(2)": no such element in  
array
```

Asociativní pole (2)

```
set slovník(pocitac) computer  
set slovník(mys) mouse  
set slovník(skrok) jump
```

```
puts $slovník(pocitac)  
computer
```

```
puts $slovník(nezname_slovo)  
can't read "slovník(xxx)": no such  
element in array
```

Funkce pro práci s poli (1)

- array get pole
 - vrací všechny hodnoty klíčů i prvků pole
- array get pole vzor
 - vrací všechny hodnoty klíčů i prvků pole, kde klíče odpovídají zadanému vzoru
- array names pole
 - vrací všechny klíče (indexy) pole
- array names pole vzor
 - vrací všechny klíče pole, které odpovídají zadanému vzoru

Funkce pro práci s poli (2)

- array set pole seznam
 - vytváří pole ze seznamu (indexuje se automaticky)
- array exists pole
 - provede ověření (predikát), zda existuje pole o zadaném názvu - vrátí řetězec 0 nebo 1
- array size pole
 - vrátí počet prvků v poli - vhodné pro počítané smyčky

Využití funkcí pro práci s poli

```
set slovník(pocitac) computer  
set slovník(mys) mouse  
set slovník(skrok) jump
```

```
puts [array size slovník]  
3
```

```
puts [array names slovník]  
skrok mys pocitac
```

Vyhledávání v poli

```
puts [array get slovník]
skok jump mys mouse pocitac computer
```

```
puts [array get slovník "m*"]
mys mouse
```

```
puts [array get slovník "*o*"]
skok jump pocitac computer
```


Procházení polem (1)

```
set slovník(pocitac) computer
set slovník(mys) mouse
set slovník(skrok) jump

foreach name [array names slovník] {
    puts $name
}
skrok
mys
pocitac
```

Procházení polem (2)

```
set slovník(pocitac) computer
set slovník(mys) mouse
set slovník(skok) jump

foreach name [array names slovník] {
    puts $name:$slovník($name)
}
skok:jump
mys:mouse
pocitac:computer
```

Procházení polem (3)

```
set slovník(pocitac) computer
set slovník(mys) mouse
set slovník(skok) jump

set seznam [array get slovník]
foreach {key value} $seznam {
    puts "pole($key)=\t\t$value"
}

pole(skok)=      jump
pole(mys)=       mouse
pole(pocitac)=   computer
```

Procházení polem (4)

```
set slovník(pocitac) computer
set slovník(mys) mouse
set slovník(skok) jump

foreach {key value} [array get
    slovník] {
    puts "pole($key)=\t\t$value"
}

pole(skok)=      jump
pole(mys)=       mouse
pole(pocitac)=   computer
```

Řetězce (1)

- **append**
 - pomocí této funkce je možné přidat znaky na konec řetězce
- **subst**
 - tento příkaz vnucuje substituci proměnných a příkazů
- **string**
 - funkce pro manipulaci s řetězcem s mnoha dalšími volbami
- **regexp**
 - vyhledání shody v řetězci podle zadaného regulárního výrazu (zde se používá kompletní repertoár značek v regulárním výrazu)

Řetězce (2)

- **regsub**
 - vyhledání shody v řetězci podle zadaného regulárního výrazu a zápis nového řetězce do zadané proměnné
- **format**
 - formátování řetězce ve stylu C-čkovské funkce `sprintf()`
- **scan**
 - čtení hodnot z řetězce ve stylu C-čkovské funkce `sscanf()`
- **binary**
 - formátování a získávání informací z takzvaných binárních řetězců

Využití funkce format

```
format "%s %10.3f" "vysledek=" [expr  
    1.0/2]  
vysledek=          0.500
```

```
set i 0  
while {$i<10} {puts [format "%2d -> %  
    +5.3f" $i [expr cos($i)]]; incr i}  
0 -> +1.000  
1 -> +0.540  
2 -> -0.416  
3 -> -0.990
```

Příkaz string (1)

- **string length**
 - tato funkce vrátí, jak již samotný název napovídá, délku řetězce
- **string index**
 - pomocí této funkce lze získat znak na určité pozici, přičemž první znak má, podobně jako v C-čku, index nulový
- **string first**
 - vyhledávání podřetězce v řetězci směrem od začátku (odpovídá C-čkovské funkci strstr())

Příkaz string (2)

- **string last**
 - vyhledávání podřetězce v řetězci směrem od konce
- **string compare**
 - lexikografické porovnání dvou řetězců (odpovídá C-čkovské funkci strcmp(), výstupní hodnoty jsou -1, 0 a 1)
- **string match regexp retezec**
 - porovnání řetězce se vzorem, přičemž je možné použít zjednodušených regulárních výrazů (podobně jako v shellu, výstupní hodnoty jsou 0 nebo 1)

Příkaz string (3)

- **string range**
 - tato funkce vrátí podřetězec zvolený dvěma indexy. Místo horního indexu lze použít i slovo end (=expr [string length] -1)
- **string tolower**
 - tato funkce vrátí řetězec odpovídající původnímu řetězci, přičemž se provádí převod na malá písmena - minusky (funguje i pro CZ/SK znaky)
- **string toupper**
 - obdoba předchozí funkce s tím, že se zde provádí převod na kapitálky (opět podporuje CZ/SK)

Příkaz string (4)

- **string trim**
 - tato funkce vrátí podřetězec, ze kterého jsou odstraněny vybrané počáteční a koncové znaky
- **string trimleft**
 - obdoba předchozí funkce s tím rozdílem, že se odstraňují pouze počáteční znaky
- **string trimright**
 - odstranění pouze koncových znaků
- **string wordstart**
 - tato funkce vrátí index prvního znaku slova, které obsahuje zadanou pozici

Regulární výrazy

Regulární výrazy

- V Tcl existují dva typy regulárních výrazů:
 - glob matching
 - výrazy pro příkazy regexp a regsub
- Mohou se používat i v rozhodovací konstrukci switch

Glob matching

- *
- libovolně dlouhá sekvence znaků
- ?
- libovolný znak (pouze jeden)
- [...]
- libovolný znak ze zadané množiny (lze použít i interval - viz příkazy shellu)
- \<znak>
- odstranění speciálního významu výše zmíněných zástupných znaků
- další znaky další znaky se s řetězcem porovnávají bez dalšího zpracování

regexp a regsub (1)

- `.`
 - libovolný znak
- `*`
 - nula či více výskytů předchozí položky
- `+`
 - jeden či více výskytů předchozí položky
- `|`
 - volba mezi dvěma výrazy (or)
- `[...]`
 - libovolný znak ze zadané množiny (lze použít i interval - viz příkazy shellu)

regexp a regsub (2)

- `^`
 - začátek řetězce (uvnitř závorek funguje jako negace)
- `$`
 - konec řetězce
- `\<znak>`
 - odstranění speciálního významu zástupného znaku
- další znaky další znaky se s řetězcem porovnávají bez dalšího zpracování

regexp a regsub (3)

```
set sample "Where there is a will,  
There is a way."
```

```
set result [regexp {[a-z]+}  
$sample match]
```

```
puts "Result: $result match:  
$match"
```

```
Result: 1 match: here
```

Rozhodovací konstrukce switch

Rozhodovací konstrukce switch

- Jednotlivé větve se nemusí ukončovat příkazem break
- Příkazu je možné předat několik voleb
- -exact
 - exaktní porovnávání řetězce se vzorem (implicitní nastavení)
- -glob
 - porovnávání na základě jednodušší formy regulárního výrazu (jako v shellu)
- -regexp
 - porovnávání na základě rozšířené formy regulárního výrazu (jako ve Vimu)

Příkaz switch (1)

```
# příkaz switch na jednom řádku  
# s exaktním porovnáváním  
set foo "abc"  
switch abc a {expr 1} b {expr 1} $foo  
    {expr 2} default {expr 3}
```

```
# víceřádková forma  
switch $promenna {  
    a {expr 1}  
    b {expr 1}  
    default {expr 3}  
}
```

Příkaz switch (2)

```
# příkaz switch rozepsaný  
# na více řádků s nastaveným  
# shellovským prohledáváním
```

```
switch -glob aaab {  
    a*b      {expr 1}  
    b        {expr 1}  
    a*       {expr 2}  
    default  {expr 3}  
}
```

Příkaz switch (3)

```
# spojení dvou větví  
# (obdoba C-čkovského stylu  
# psaní více větví):
```

```
switch -glob aaab {  
    a*b      -  
    b        {expr 1}  
    a*       {expr 2}  
    default  {expr 3}  
}
```

Práce se soubory

Otevření souboru

- **open** jméno_souboru přístup práva
- jméno_souboru
 - absolutní nebo relativní cesta
- přístup
 - čtení (read - „r“)
 - zápis (write - „w“)
 - přidání na konec (append - „a“)
- práva
 - výchozí práva – viz chmod
 - default 0666

Uzavření souboru, čtení, zápis

Uzavření souboru:

`close` kanál

Čtení dat:

`gets` kanál proměnná

`read` kanál

`read` kanál počet_znaků

Zápis dat:

`puts` kanál řetězec

`puts -nonewline` kanál řetězec

Zápis dat do souboru

```
set fout [open "data.txt" "w"]
while {$i<10} {
    puts $fout $i
    incr i
}
close $fout
```

```
--- shell ---
```

```
cat data.txt
```

```
0
```

```
1
```

```
2
```

```
3...
```

Kopie souboru po řádcích

```
proc copyFile { inFileName outFileName } {  
    set inFile [open $inFileName "r"]  
    set outFile [open $outFileName "w"]  
    while { [gets $inFile temp] != -1 } {  
        puts $outFile $temp  
    }  
    close $inFile  
    close $outFile  
}
```

Systémové funkce pro práci se soubory (1)

- file size soubor
 - vrací délku souboru v bytech
- file atime soubor
 - vrací poslední čas přístupu k souboru
- file mtime soubor
 - vrací čas poslední úpravy souboru
- file dirname soubor
 - vrací část jména souboru reprezentujícího adresář (na Unixech jde o rozdělení řetězce před posledním lomítkem)

Systémové funkce pro práci se soubory (2)

- file extension
 - soubor vrací příponu souboru (na Unixech se jedná o rozdělení řetězce před poslední tečkou)
- file rootname soubor
 - vrací jméno souboru bez přípony
- file exists soubor
 - vrací hodnotu 1, pokud soubor existuje
- file isdirectory soubor
 - vrací hodnotu 1, pokud je názvem adresář (v newspeaku složka :-)

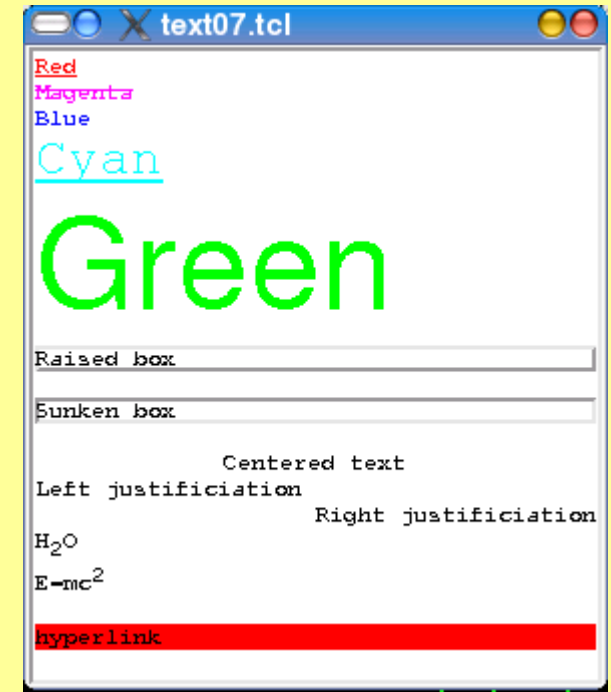
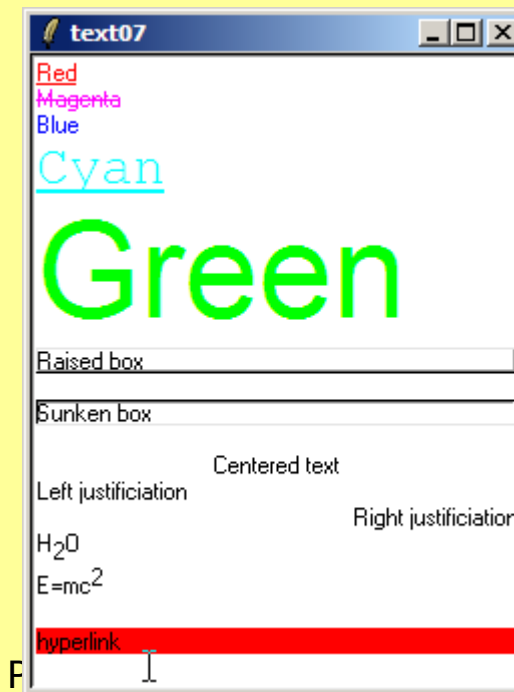
Systémové funkce pro práci se soubory (3)

- file isfile soubor
 - vrací hodnotu 1, pokud je názvem běžný soubor (ne adresář ani roura)
- file readable soubor
 - vrací hodnotu 1, pokud lze soubor číst (test práv uživatele)
- file writable soubor
 - vrací hodnotu 1, pokud lze do souboru zapisovat (test práv uživatele)
- file executable soubor
 - vrací hodnotu 1, pokud je soubor spustitelný (pro daného uživatele)

Tk

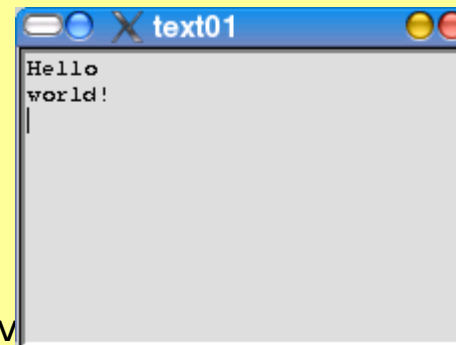
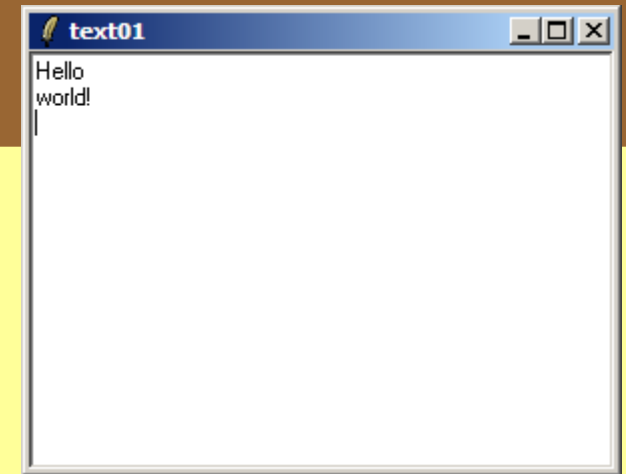
Tcl/Tk

- Tk primárně určeno pro použití v Tcl
 - Nejlepší integrace Tcl+Tk
 - Proto také celosvětově používané Tcl/Tk
- Existence wrapperů i pro další jazyky
 - Tkinter
 - Perl/Tk
 - Ruby/Tk
 - apod.
 - Již není tak elegantní jako v Tcl



Tk

- Knihovna pro tvorbu GUI
- John Ousterhout
- Platformově nezávislé
 - v některých případech Look&Feel OS
- Mnohem méně LOC:
 - Tcl/Tk:Motif~1:20 (podobně WinAPI)
 - Tcl/Tk:Xlib~1:100
- Podpora i z dalších jazyků, nejenom Tcl
 - Tk Interface: tkinter



Základy Tk

- Widgety
- Kontejnery
- Sada widgetů:
 - label
 - button
 - checkbox
 - scale
 - entry
 - spinbox
 - ...

Základy Tk

- Sada kontejnerů:
 - frame
 - toplevel (samostatné okno)
 - canvas
- Vlastnosti
 - background
 - foreground
 - font
 - text
 - textvariable
 - ...

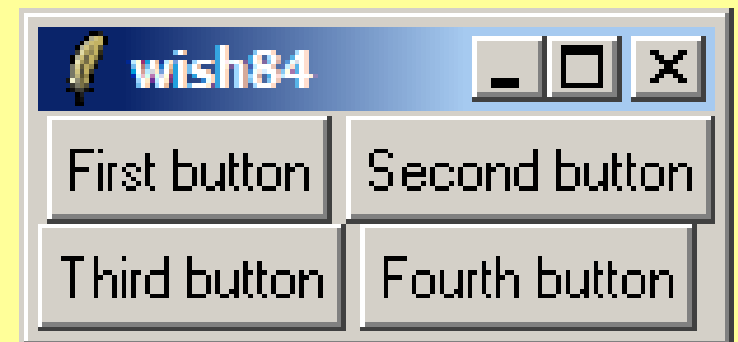
Událости

```
#!/usr/bin/wish  
label .hello -text "Hello world!"  
button .close -text "Close window" -command {exit}  
pack .hello  
pack .close
```



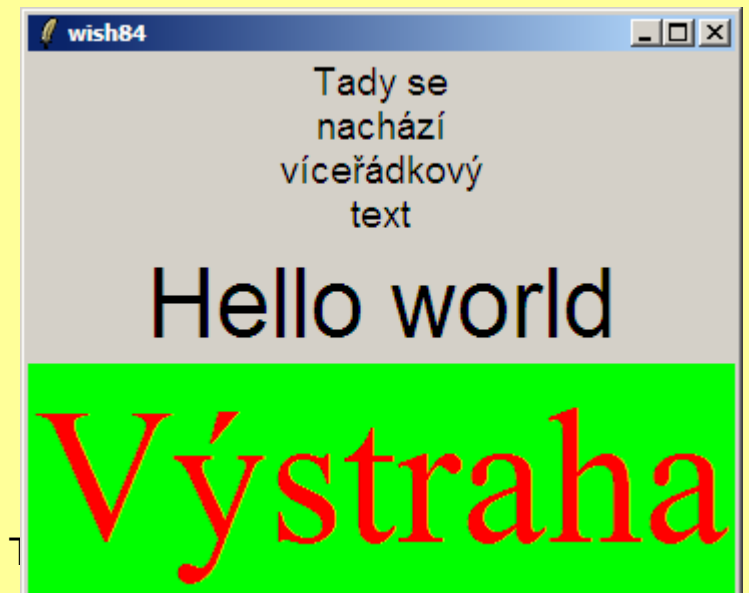
Správce geometrie

```
#!/usr/bin/wish
button .prvni -text "First button" -command
    {exit}
button .druhy -text "Second button" -command
    {exit}
button .treti -text "Third button" -command
    {exit}
button .ctvrty -text "Fourth button" -command
    {exit}
grid .prvni -column 1 -row 1
grid .druhy -column 2 -row 1
grid .treti -column 1 -row 2
grid .ctvrty -column 2 -row 2
```



Nastavení vlastností

```
#!/usr/bin/wish
label .label3 -text "Tady
    se\nnachází\nvíceřádkový\ntext" -font
    "Helvetica +14"
label .label4 -text "Hello world" -font
    "Lucida +36"
label .label5 -text "Výstraha" -background
    green -foreground red -font "Times +72"
pack .label3
pack .label4
pack .label5
```



Vstupní políčka

```
#!/usr/bin/wish
```

```
# vytvoření popisků
```

```
label .name -text "Name"
```

```
label .password -text "Password"
```

```
label .role -text "Role"
```

```
# textová vstupní pole
```

```
entry .nameEntry -textvariable name
```

```
entry .passwordEntry -textvariable password
```

```
entry .roleEntry -textvariable role
```

```
# použití manažera geometrie
```

```
grid .name -row 0 -column 0 -sticky w
```

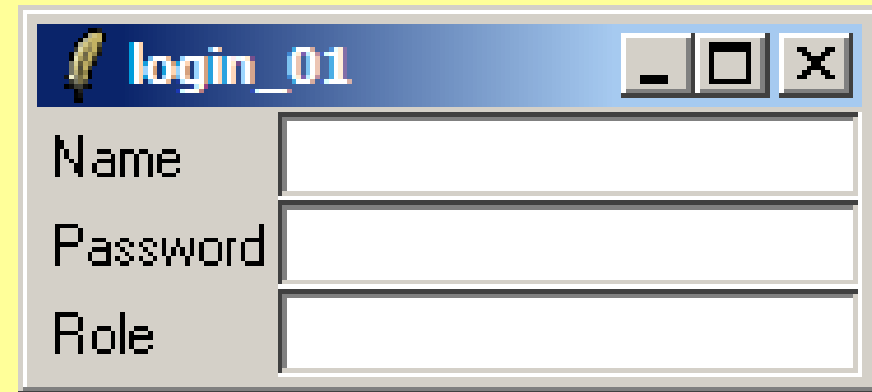
```
grid .password -row 1 -column 0 -sticky w
```

```
grid .role -row 2 -column 0 -sticky w
```

```
grid .nameEntry -row 0 -column 1 -sticky w
```

```
grid .passwordEntry -row 1 -column 1 -sticky w
```

```
grid .roleEntry -row 2 -column 1 -sticky w
```



The screenshot shows a graphical user interface window titled "login_01". It contains three vertically stacked input fields. The first field is labeled "Name", the second "Password", and the third "Role". Each field is a simple text entry box with a light gray border and a white background. The window has a standard title bar with minimize, maximize, and close buttons.

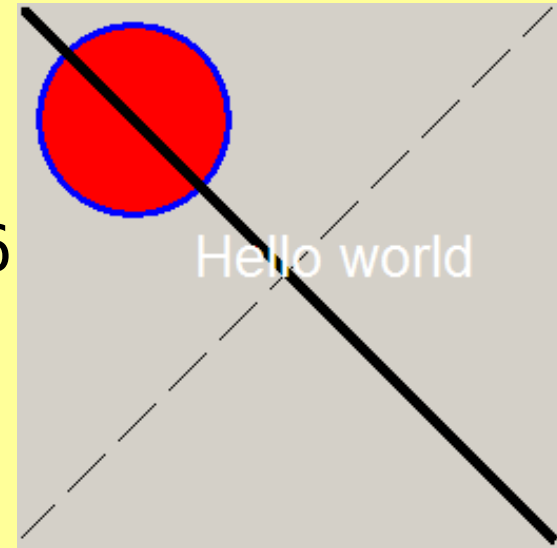
Canvas

```
#!/usr/bin/wish

# vytvoříme plátno
canvas .platno -width 256 -height 256

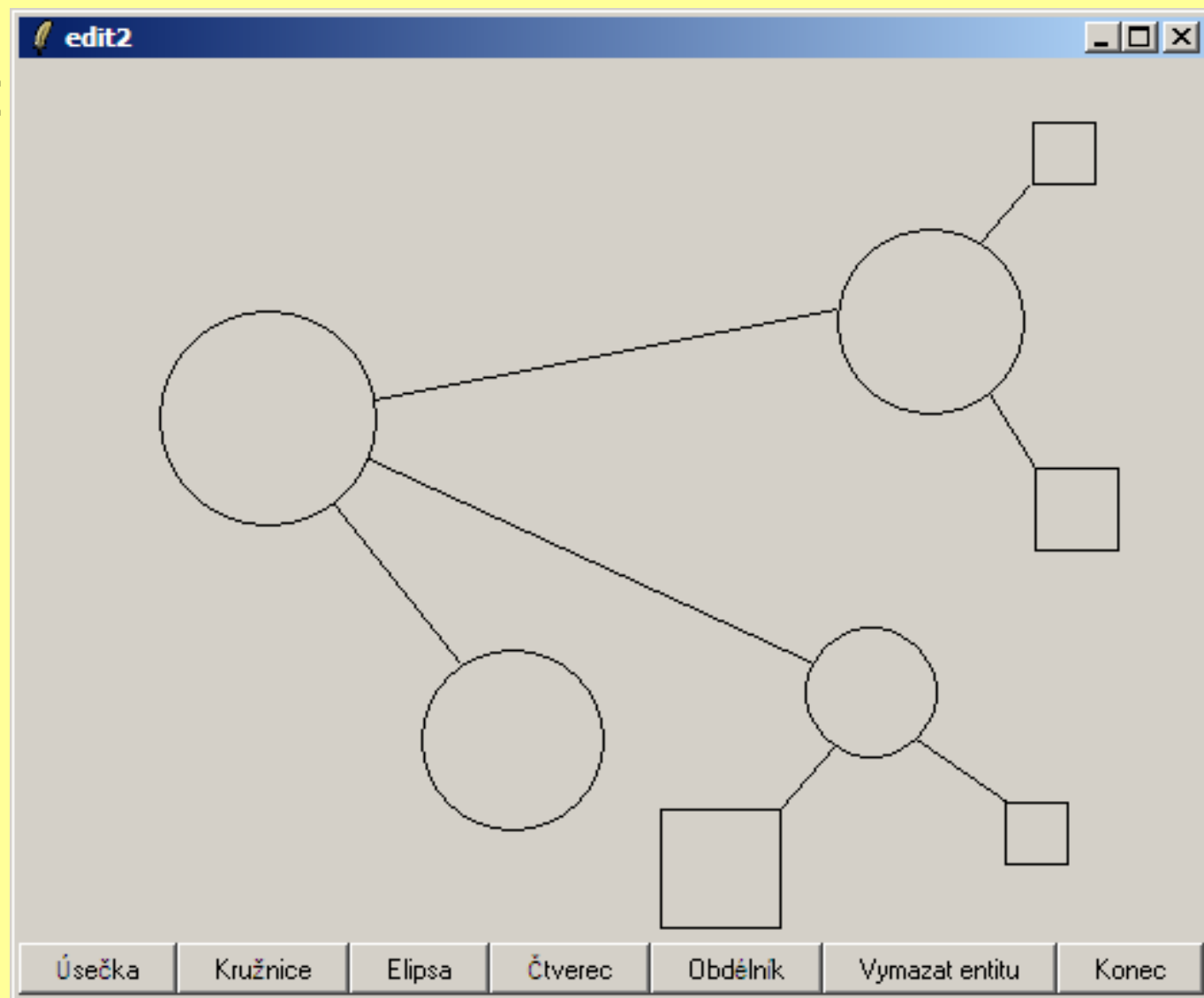
# vložíme plátno do okna
grid .platno -column 1 -row 1

# na plátnu vytvoříme několik objektů
# se změněnými vlastnostmi
.platno create oval 10 10 100 100 -fill red
      -outline blue -width 3
.platno create line 0 0 255 255 -width 5
.platno create line 0 255 255 0 -dash 123
.platno create text 150 120 -text "Hello
world" -fill white -font Helvetica 20
```



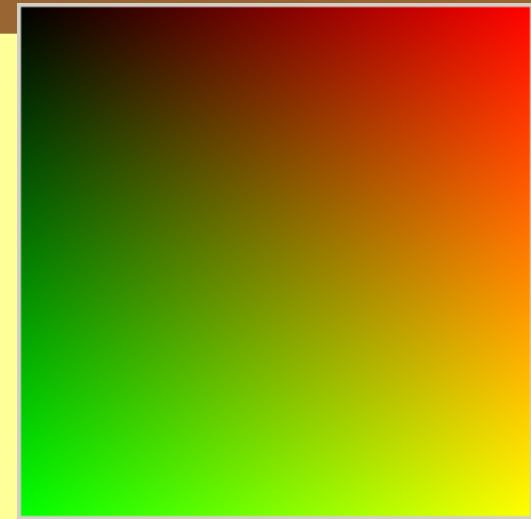
Grafický editor vytvořený pomocí canvasu

- plně funkční
- přesně 100 LOC
- i s mnoha komentáři

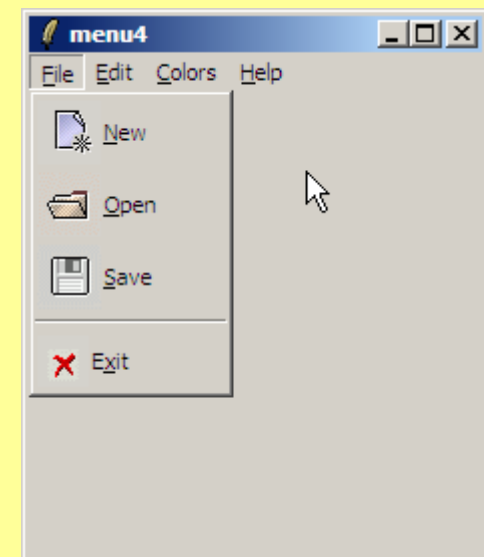
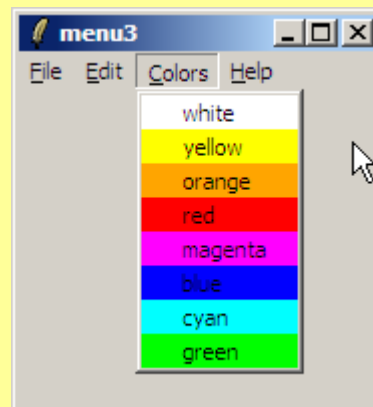
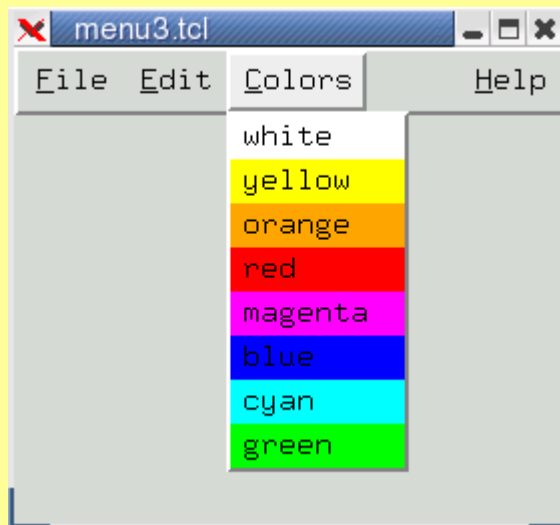
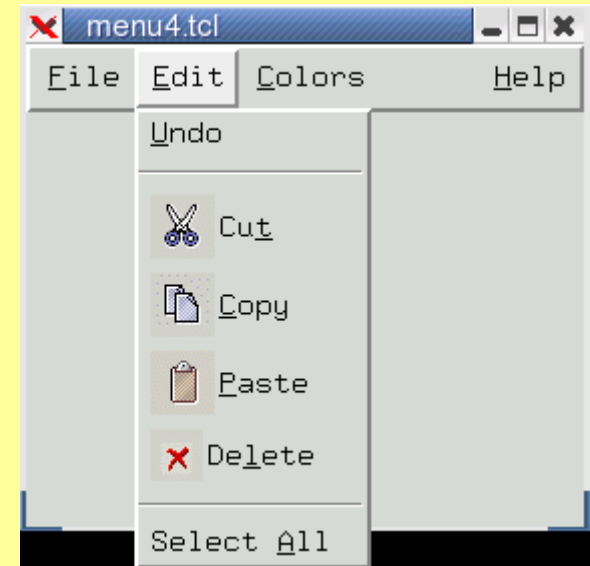
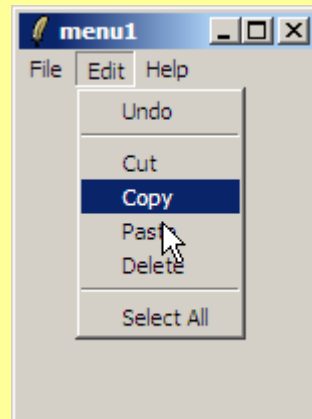
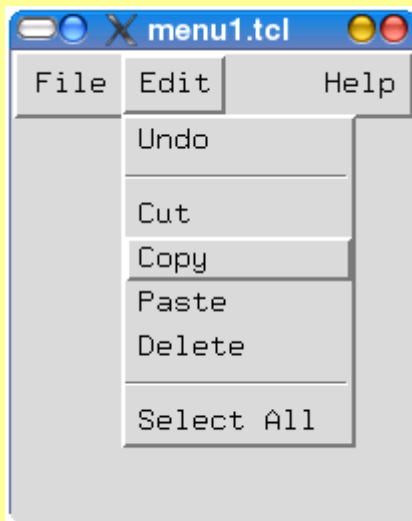


Image

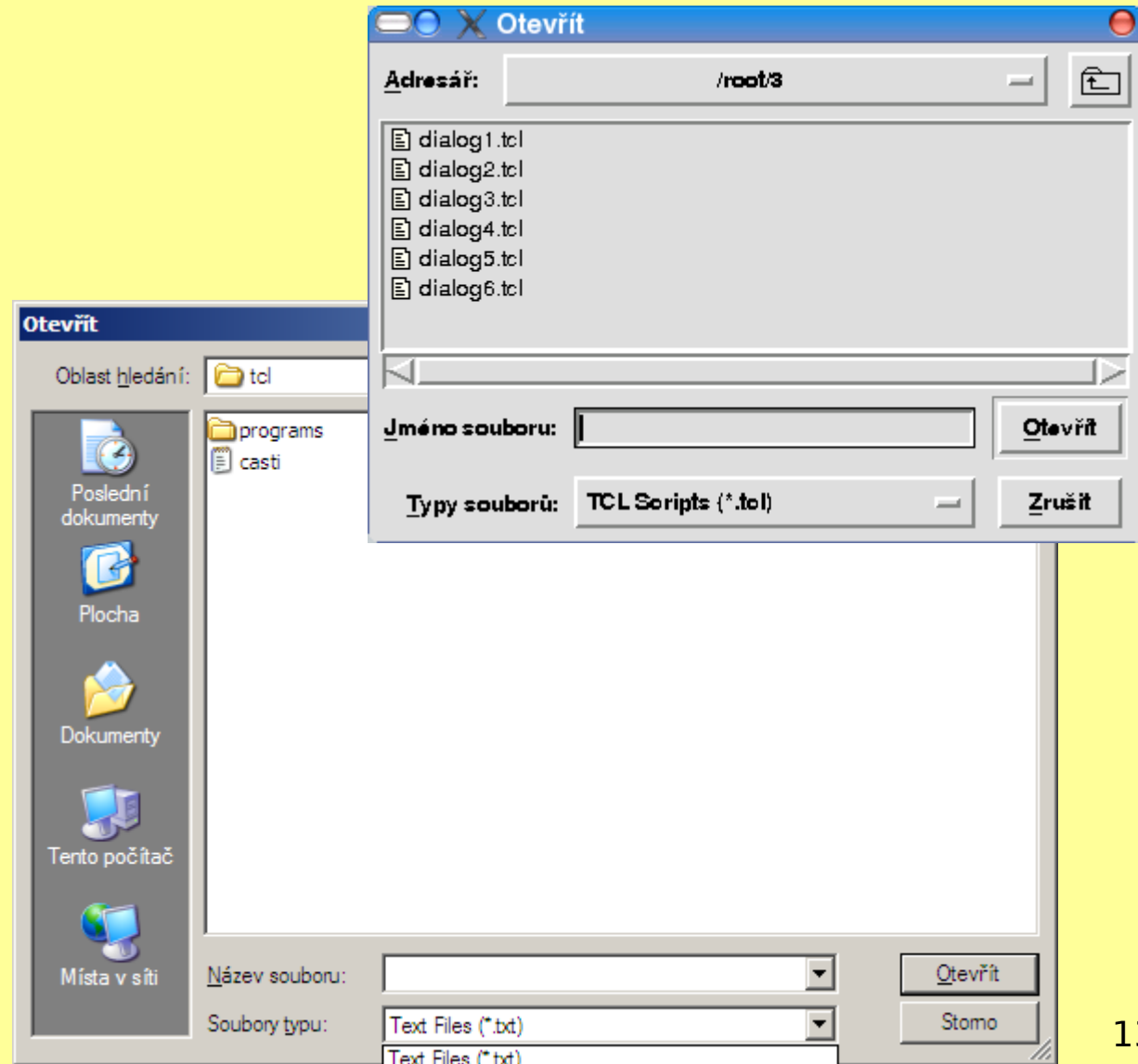
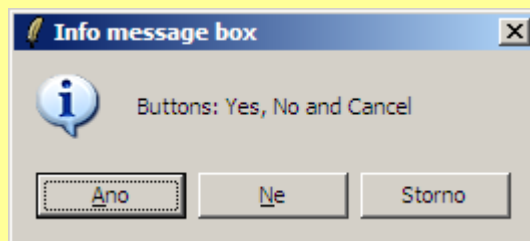
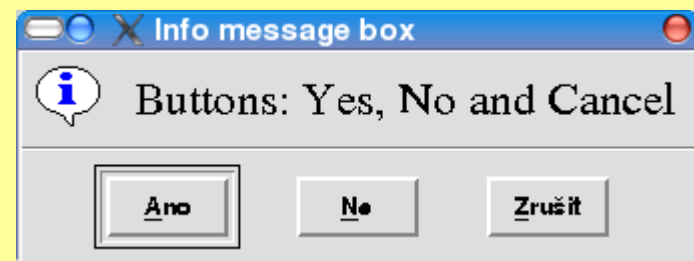
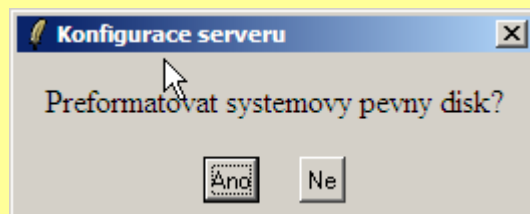
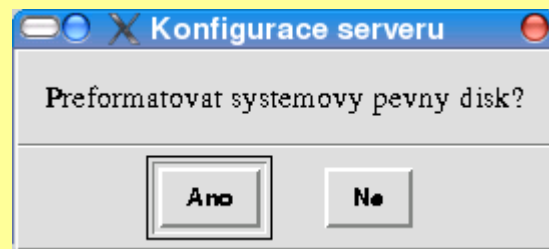
```
#!/usr/bin/wish
# rozměry obrázku
set width 256
set height 256
# vytvoření plátna
canvas .canvas -width $width -height $height
pack .canvas -fill both
# vytvoření obrázku
set image [image create photo -width $width -height $height]
$image blank
# vložení obrázku na plátno
.canvas create image 0 0 -image $image -anchor nw
# naplnění pixelů v obrázku
for {set row 0} {$row<$height} {incr row} {
    for {set col 0} {$col<$width} {incr col} {
        # barva obrázku v HTML stylu
        set colour [format "#%02X%02X%02X" $col $row 0]
        # zapsat barvu pixelu
        $image put $colour -to $col $row
    }
}
```



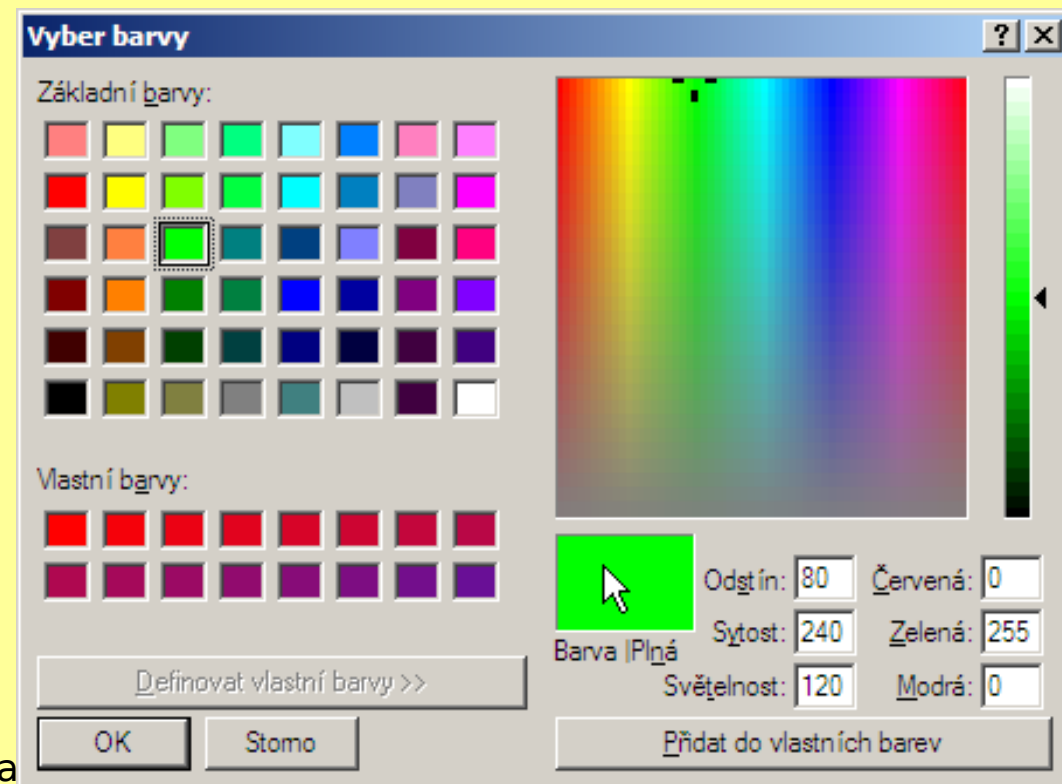
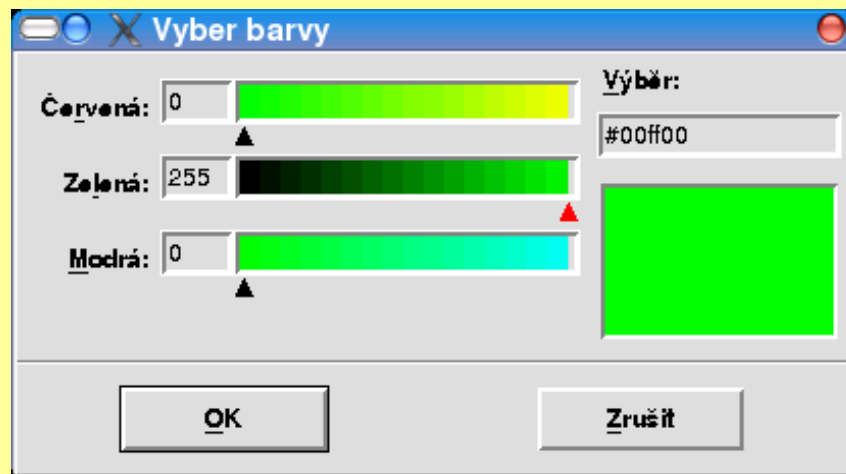
Menu



Dialogy: tk_dialog a další



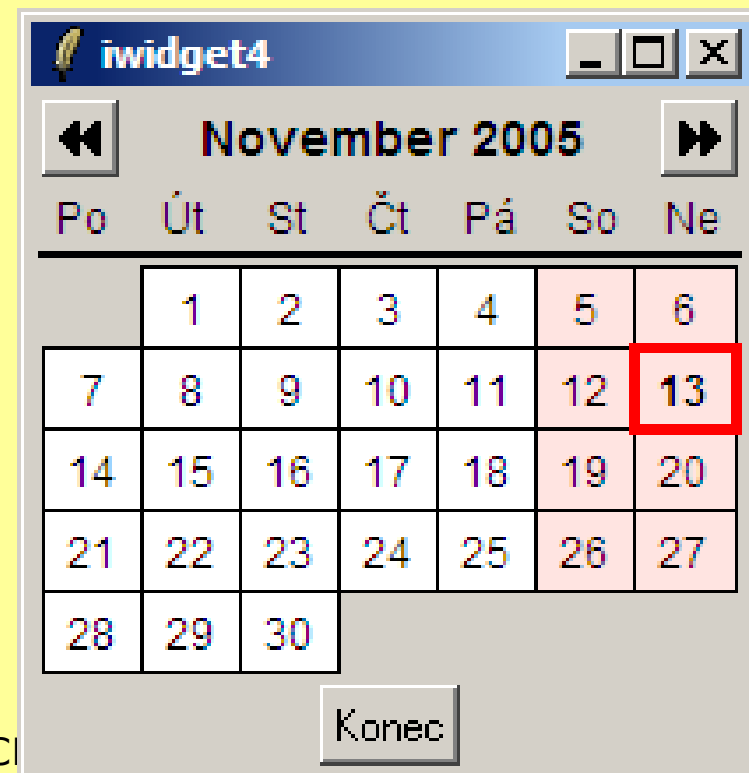
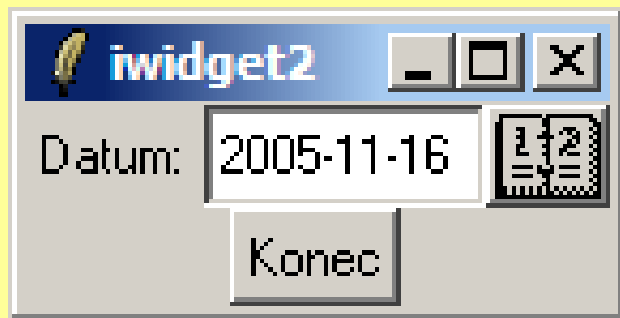
Dialogy: tk_dialog a další



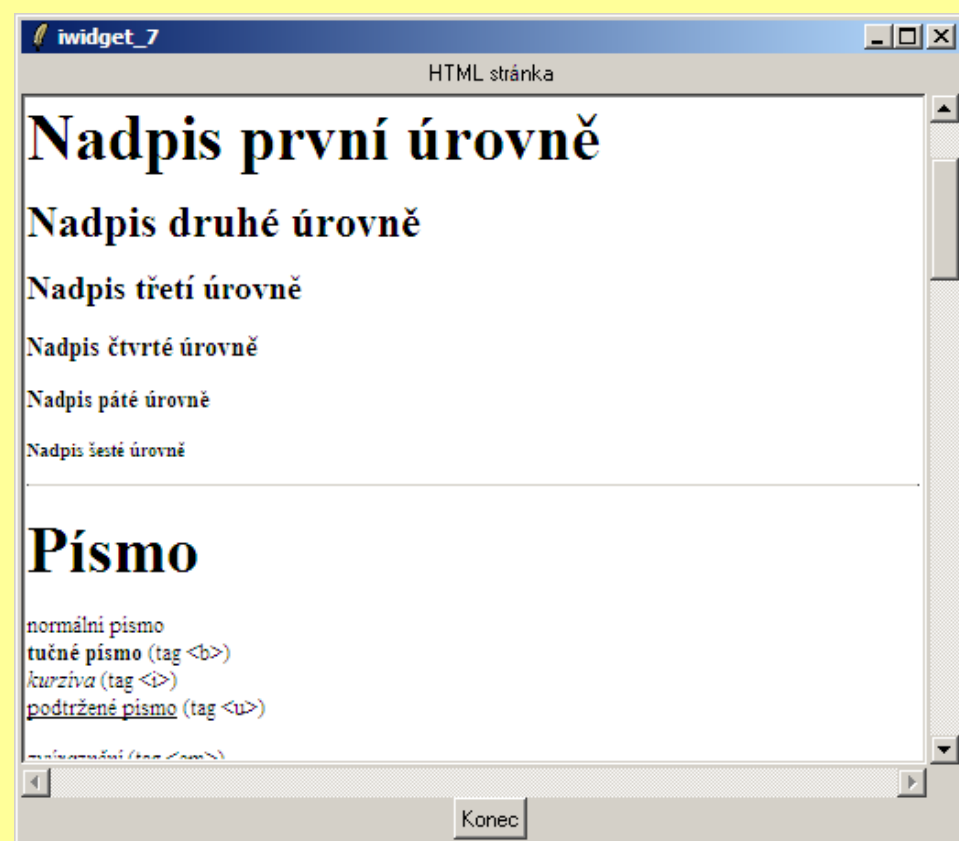
Rozšiřující toolkity

IWidgets

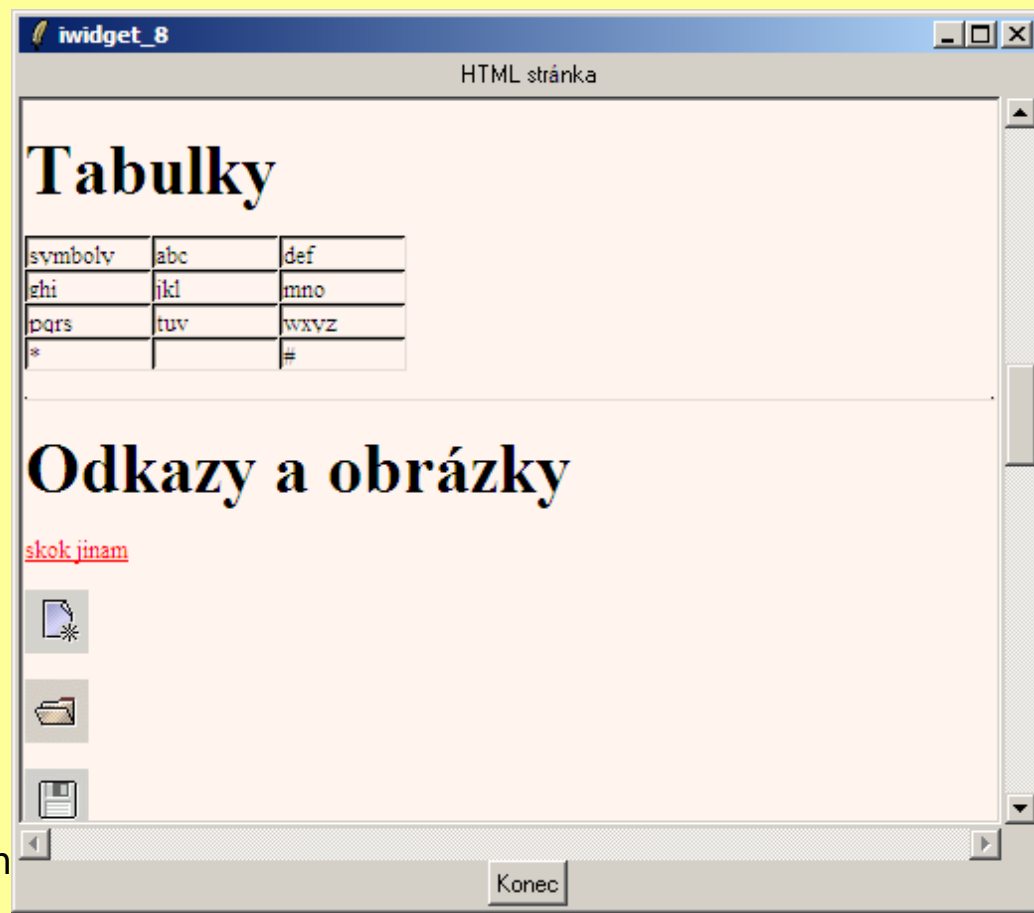
- IWidgets
 - modální i nemoďální dialogy
 - kalendář
 - hodiny
 - zadání časových údajů
 - plocha s HTML stránkou



IWidgets

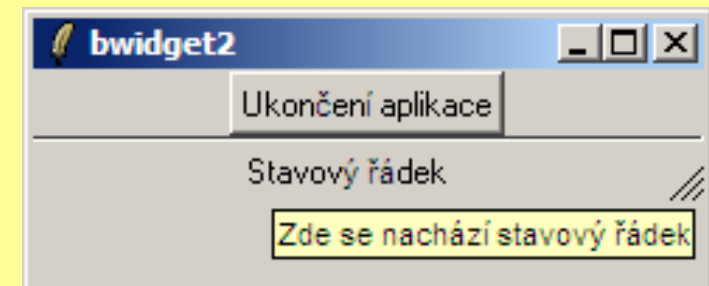
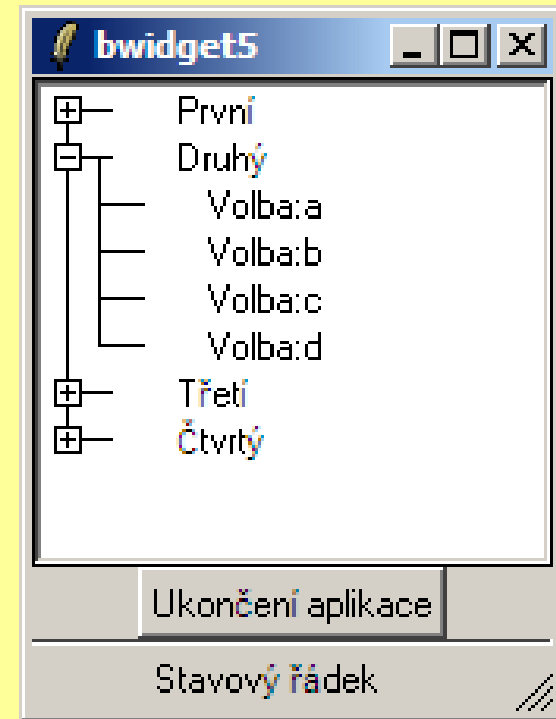


Program



BWidgets

- Nové widgety
- Rozšířená funkce widgetů
- Složené widgety a kontejnery
 - zobrazení stromové struktury
- Dialogové boxy
 - modalita/nemodalita dle OS



Odkazy na Internetu

Odkazy na Internetu (1)

- Oficiální adresa:
 - <http://www.tcl.tk/>
- Instalace pro Microsoft Windows:
 - <http://www.activestate.com/>
- Instalace pro Mac OS:
 - <http://www.tcl.tk/software/mac/>
- Stránka na Wikipedii o Tcl (EN):
 - <http://en.wikipedia.org/wiki/Tcl>
- Why Tcl is better than Perl
 - <http://www.tcl.tk/advocacy/perl.html>
- Seriál na Rootu (česky):
 - <http://www.root.cz/serialy/programovaci-jazyk-tcl/>

Odkazy na Internetu (2)

- TkCon 2.5 (Jeff Hobbs)
 - <http://sf.net/projects/tkcon>
- TclX 8.4 (Karl Lehenbauer, Mark Diekhans, Jeff Hobbs)
 - <http://sf.net/projects/tclx>
- Expect 5.44.1 (Don Libes)
 - <http://sf.net/projects/expect>
- IncrTcl/Tk 3.4 (Michael McLennan)
 - <http://sf.net/projects/incrTcl>
- TkTable 2.10 (Jeff Hobbs)
 - <http://sf.net/projects/tktable>

Odkazy na Internetu (3)

- Vuwidgets 2.3 (Jeff Hobbs)
 - <http://sf.net/projects/tktable>
- Tile 0.7.8 (Joe English)
 - <http://tktable.sourceforge.net/tile/>
- BWidgets 1.9 (Jeff Hobbs, Damon Courtney)
 - <http://sf.net/projects/tcllib>
- IWidgets 4.0.2 (Michael McLennan)
 - <http://sf.net/projects/incrTcl>
- Tcllib 1.12
 - <http://sf.net/projects/tcllib>

Odkazy na Internetu (4)

- Tklib 0.4.1
 - <http://sf.net/projects/tcllib>
- TkImg 1.3 (Jan Nijtmans)
 - <http://sf.net/projects/tkimg>
- TclXML 2.6 (Steve Ball)
 - <http://sf.net/projects/tclxml>
- TclDOM 2.6 (Steve Ball)
 - <http://sf.net/projects/tclxml>
- TclSOAP 1.6.8 (Pat Thoyts)
 - <http://sf.net/projects/tclsoap>
- Snack 2.2.10 (Kare Sjolander)
 - <http://www.speech.kth.se/snack/>

Odkazy na Internetu (5)

- TkTreectrl 2.2.9 (Tim Baker)
 - <http://sf.net/projects/tktreectrl>
- TkHTML 2.0 (Richard Hipp)
 - <http://tkhtml.tcl.tk>
- Tcom 3.9 (Chin T. Huang)
 - <http://www.vex.net/~cthuang/tcom/>
- XOTcl 1.6.5 (Uwe Zdun, Gustaf Neumann)
 - <http://www.xotcl.org/>
- OraTcl 4.3 (Tom Poindexter, Todd Helfter)
 - <http://sf.net/projects/oratcl>

Odkazy na Internetu (6)

- SQLite 3.6.22 (Richard Hipp)
 - <http://www.sqlite.org>
- tDOM 0.8.2 (Jochen Loewer, Rolf Ade)
 - <http://www.tdom.org/>
- TLS 1.6 (Dan Razell, Matt Newman, Jeff Hobbs)
 - <http://sf.net/projects/tls>
- Trofs 0.4.4 (Don Porter)
 - <http://math.nist.gov/~DPorter/tcltk/trofs/>