

Use "range," not "for"?

Source: <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2004-02/0642.html>

From: David McClamrock (*mcclamrock_at_locl.net*)

Date: 02/13/04

Date: Thu, 12 Feb 2004 19:16:33 -0500

Thanks to everyone who suggested ways to get a proc to recognize all variables in the global scope—here's why I wanted to know. (Maybe someone has done this better already—if so, don't hesitate to let me know!)

I've found that a simple "foreach" loop fairly often won't do the jobs I want done, and I need to use "for"—ugh! Well, I recently read at least part of a book about Python. It didn't look like an improvement over Tcl for the most part, but it has at least a few good features, one of which is the "range" expression. I thought a "range" procedure in Tcl might be a good substitute for the clumsy "for." So, here's my first effort. Instead of writing this:

```
for {set i 1} {$i <= 10} {incr i} {puts "$i. \"for\" is ugly!"}
```

you can write this:

```
range i 1 to 10 {puts "$i. \"range\" is beautiful!"}
```

Or, if you have a list called "lum," instead of writing this:

```
for {set i 0} {$i < [llength $lum]} {incr i} {puts [lindex $lum $i]}
```

you can write this:

```
range i 0 no [llength $lum] {puts [lindex $lum $i]}
```

To go backward, skipping every other number, instead of this:

```
for {set i 10} {$i >= 0} {incr i -2} {puts $i}
```

you can write this:

```
range i 10 to 0 -2 {puts $i}
```

The usage is pretty obvious: "range var start cutoff end ?incr? body." In other words, (1) the word "range"; (2) a variable to change through the range; (3) the beginning of the range; (4) "to" if the range is inclusive or "no" if it excludes the last number; (5) the end of the range; (6) a

comp.lang.tcl: Use "range," not "for"?

number for the increment if it isn't 1 (for an ascending range) or -1 (for a descending range); and (7) a body of code to execute on each iteration of the loop.

The procedure is below. Please let me know if (1) somebody already did this better, (2) you want to suggest any improvements or bugfixes, (3) you don't think it's worth the effort of copying this procedure to get rid of all the braces, repetition, and other minor difficulties in the "for" command, or (4) (insert additional options here). Thanks in advance!

David McClamrock

Procedure to do a "for" loop without ugly, awkward "for" code:

```
uplevel #0 {
proc range {var star cutoff fin args} {
    global codex $var
    if {[catch {expr $var+1}] == 0} {
        error "Use a letter, not a number, for the first \"arg\"!"
    }
    if {[catch {expr $star+$fin}] > 0} {
        error "Range must contain two numbers!"
    }
    if {$star == $fin} {
        error "Range cannot begin and end with the same number!"
    }
    switch $cutoff {
        "to" {set inclu 1}
        "no" {set inclu 0}
        default {
            error "Use \"to\" for an inclusive range,\
                or \"no\" for a noninclusive range!"
        }
    }
    set argleng [llength $args]
    set codex [lindex $args end]
    if {$argleng < 1 || $argleng > 2} {
        error "Number of additional \"args\" must be one or two!"
    } elseif {$argleng == 2} {
        set incro [lindex $args 0]
        if {[catch {expr $incro+1}] == 1} {
            error "First of two additional \"args\" must\
                be a number!"
        }
    }
    set starless [expr $star < $fin]
    switch "$starless $argleng $inclu" {
        "1 1 1" {
            set sign "<="
            set incro 1
        }
    }
}
```

comp.lang.tcl: Use "range," not "for"?

```
}
"1 1 0" {
    set sign "<"
    set incro 1
}
"0 1 1" {
    set sign ">="
    set incro -1
}
"0 1 0" {
    set sign ">"
    set incro -1
}
"1 2 1" {
    set sign "<="
}
"1 2 0" {
    set sign "<"
}
"0 2 1" {
    set sign ">="
}
"0 2 0" {
    set sign ">"
}
default {
    error "Number of additional \"args\" must\
        be one or two!"
}
}
if {$starless == 1 && $incro <= 0} {
    error "Range increases--increment must be positive!"
} elseif {$starless == 0 && $incro >= 0} {
    error "Range decreases--increment must be negative!"
}
for "set $var $star" "$$var $sign $fin" "incr $var $incro" {
    uplevel #0 {
        eval $codex
    }
}
}
}
```