# Tcl Language Cheat Sheet

by AHA via cheatography.com/22239/cs/18129/

## List operations

| | |
|---|---|
| lappend **<list>** $ele | $ele is added to list as a single index |

set A1 [concat $list1 "ABC" $list2 $var2]; *#concat will add lists as a separate indexes, if list 1 contains 3 elements it will add 3 index values; ○ If you want to maintain list structure use lappend, if you want simple list with all the elements separate use concat*

| | |
|---|---|
| llength <$list> | length of a list |

lindex <$list> <#index_no> ?<#2nd index_no in case of nested list>?

lrange <$list> <#1st_index> <#2nd_index>; #Returns elements from 1st index to 2nd index

linsert <$var> <#index_position> <ele1> <ele2> <ele3> .... ; *#Output: will have list added at index position*

**linsert <$list> <#index_position> <ele1> <ele2> <ele3> .... ;** *# Output will have separate elements, first index will be WHOLE original list*

lreplace <$list> <#start_index> <#end_index> <item1> <item2> <item3>...; *# replace items between start and end indexes with item1, item2 .....*

lsearch ?-<all| dictionary| decreasing| glob| exact| regexp| not| sorted| integer| real| #start_index>? <$list> <pattern>

lsort ?-<ascii| dictionary| integer| real| increasing| decreasing| #index_sublist| unique>? <$list> *# index option =3 will sort the main list based on 4th element of each sublist*

| | |
|---|---|
| split <$list/$var> ?<chars>? | **String to list** |
| join <$list/$var> ?<chars>? | **List to string** |

*○ set random [list a b c {bad} { had } 123 1a2b "1.22" 15.43 $had $tag {A B C D E F G}]; #Here each element is a separate element, each list with grouping is a single element*
*○ list $a $b $c; #Here list contains 3 elements only, even though a b or c contains list inside it*

## String operations

**string match ?-nocase? <pattern> <string>** *# if pattern matches string output is 1*

**string equal ?-nocase|-length int? <string1> <string2>** # Returns 1 if string1 and string2 are identical, or 0 when not.

**string compare ?-nocase|-length #N? <string1> <string2>** #check strings lexicographically, if string1> string2 output is 1

**string map {<from> <to>} <string>** #string map {H 000} $a 17 #replace H in Hello with 000

**string replace <string> <#start_index> <#end_index> ?<replacement_string>?**

## String operations (cont)

**subst ?-nobackslashes? ?-nocommands? ?-novariables? string** *#set a 44 ;subst {xyz {$a}}; output is xyz{44}*

**string range <string> <#start_index> <#end_index>**

**puts [format "Today is %s %d %f" $day $month $year]**

string length <string>

string first <char or pattern to search> <string> ?<start index>?

string index <string> <index #| end|end-n>

string bytelength <string>

string last <char or pattern to search> <string> ?<end index>?

string is <class> <string> #class==alpha|ascii|boolan|control|digit|double|false|graph|integer|list|lower|space|punct|true|upper|lower

string repeat <string> <#count>

string toupper/tolower/totitle <string> ?<#start_index>? ?<#end_index>?

string trimleft/trimright/trim <string> ?<chars>?

append <string_var> $x, $y # "ZZZ"

string reverse <string>

string wordend|wordstart <string> <#index>

## Arrays

```
array set array1 [list {123} {Abigail Aardvark} \
{234} {Bob Baboon} \ {345} {Cathy Coyote} \ {456}
{Daniel Dog} ]
```

set fruit(Apple) 143

array size <arrayname> ; #

array names <arrayname> ?<pattern>?;# gives all the keys as iteratable list

array get <arrayname>; # returns list where each odd member is key and even is value

array exists <arrayname>

```
foreach key [array names array1] { puts "Key is
$key and value is $array1($key)"; }
```

parray <arrayname>; #

array startsearch <arrayName>; #

## Arrays (cont)

array nextelement <arrayName> <searchID> ;#

array anymore <arrayName> <searchID>

array donesearch <arrayName> <searchID>

## Dictionaries

**dict set <DICT NAME><Key1> <value1>** *#create a dict with DICT_NAME key and value*

**dict set <DICT NAME><Key1> <nested Key1_1> <value1>** *#create a dict with DICT_NAME key and value*

**set <DICT NAME> [dict create 1 "SK" 2 "KK" 3 "ZK"]***#1 is key and SK in value*

**dict unset names <$DICT_NAME>** ; #removes key/value pair

dict replace <$DICT NAME> <key> <new_value> ;#replace the value corresponding to key

dict keys <$DICT_NAME> ; #provides all the keys as list

dict values <$DICT_NAME>; #provides all values as list

dict get $<DICT_NAME> <key> ;#returns value on that key:ZK

dict get $DICT_NAME ;#returns all key and value pair as a list

```
dict for {key value} $DICT { puts "Key:$key Value:-$value";}
```

```
foreach keys [dict keys $DICT] { puts "Key:$keys Values [dict get $DICT $keys]"; }
```

dict append <DICT_NAME> 4 LA;#adds one key to dictionary

dict lappend <DICT_NAME> 4 LA SF PO;#adds values as a single value to key 4

set filtered [dict filter $<DICT_NAME> key|value|script 1] ;#filter by name value or script

set filtered [dict filter $<DICT_NAME> script {key value} {Expr {$key < 3 }}] ;# returns 1 SK 2 KK

dict exists $names <key> ;# Checks for key 3 in names dict, returns 1 is key exists

dict incr $names one<key> 4<increment by> ;#increments value by integer <value>

dict info $names ; # provides info on dict

set merged [dict merge $test1 $test2] ; #merges two dicts

set new [dict remove $test <key1> <key2> ....] ; # removes key/value pair based on specif ied keys

#### Dictionary modification

dict size $names ; #get the number of key/value pairs

## Dictionaries (cont)

```
# Define a simple dict: {a b c d} ; set dd [dict create a b c d] ; dict update dd a aVar c cVar { set aVar 2; set cVar [string toupper $cVar]; } # displays {a 2 c D} set dd
```

## Dictionary Examples

```
dict for {key value} $names {
        puts "Key is: $key and Value is: $value"
        }
foreach key [dict keys $name_number_dict] value [dict values $name_number_dict] {
             puts "$key == $value"
             }
#dict update
        % set didi {key1 value1 key2 value2}
        key1 value1 key2 value2
        % dict update didi key1 varKey1 key2 varKey2 {
            append varKey1 new
            append varKey2 new
            unset varKey1; #deletes key value pair
          }
        value2new
        % set didi
        key1 value1new key2 value2new
#dict with : you can convert key names to variables directly
        % set pers_detail [dict create forenames Joe surname Schmoe street {147 Short Street} \
            city Springfield phone 555-1234]
        dict with pers_detail {
            puts " Name: $forenames $surname"
            puts " Address: $street, $city"
            puts " Telephone: $phone"
        }
**##Inventory system
KEY FIRST LAST TITLE
```

## Dictionary Examples (cont)

```
1 Clif Flynt Tcl/Tk For Real Programmers
2 Clif Flynt Tcl/Tk: A Developer's Guide 2'nd
edition
3 Brent Welch Practical Programming in Tcl/Tk
4 Michael McClennan Effective Tcl/Tk Programming
5 Don Libes Exploring Expect
dict set books 1 [list first Cliff last Flint
title "Tcl/TK a dev guide" year 2009]
dict set books 2 first Brent
dict set books 2 last Welch
dict set books 2 title "Practical Programming in
Tcl/Tk"
dict set books 2 year 1867
foreach {sr first last title year} {
     3 Michael McClennan {Effective Tcl/Tk Progra-
mming} 2001
     4 Don Libes {Exploring Expect} 2008 } {
     dict set books $sr [list first $first last
$last title $title year $year]
     }
#set first firsttest
dict for {sr infor} $books {
dict with infor {
puts "$sr $first $last $title $year"
}
}**
```

## Looping

while {<test exp>} { <body> }

for {set i 10} {$i>=0} {incr i-1} { <body> }

foreach ele $list { <body> }

foreach {ele1 ele2} $list { <body> }; # 2 elements are taken from list2 for each iteration

foreach ele1 $list1 ele2 $list2 { <body> }; # l1 iterates over list1 and l2 iterates over list2

## Switch

```
• switch ?-option (exact, glob, regexp, nocase) --
? $string \
$pattern_1 {body} \
$pattern_2 {body} \
.
.
default {body};

• switch ?-option (exact, glob, regexp, nocase) --
? $string \
{A body}
{B body}
.
.
{default body};
Here pattern substitutions cannot occur
```

## procs

```
• Proc with optional arguments
proc random_num {min {max 100}} {
<body>
}
• Proc with variable number of arguments
proc random_num {args} {
puts $args; llength $args; lindex $args 0;
<body>
}
```

## regex

**regexp ?** <*about*| *expanded*| *indices*| *line*| *linestop*| *lineanchor*| *nocase*| *all*| *inline*| *start*| *#index_start*>*?* **{<regex_to_match>} <$string> match submatch1 submatch2 .....**

## regex (cont)

**regsub ?** *<all| expanded| line| linestop| lineanchor| nocase| inline| #index_start>?* **{<regex_to_match>} <$string> {<regex_to_replace>}** *?<storage_variable>?*

Regsub
○ \1: 1st grouping match
○ \2: 2nd grouping match
○ &: whole expression/inside {} match
○ command returns replaced string dictated by regex to replace {}
○ if storage_variable specified, return 1 or 0 depending on <regex to match> or not
○ `regsub {(\w+).(\w+)} "report.txt" {\1.tcl} ext`
`puts $ext (output will be report.tcl)`

## File IO

set FILEHANDLE [ open <filename> ?access_mode (r| r+| w| w+| a | a+)? ?permissions(755, 744, 777)?

close $FILEHANDLE

if {[catch {set FILEHANDLE [ open <filename> r+] } ?<error_var>?]} { puts $error_var; <body> }

*r+ mode does not create a file if it does not exists, where a+/w+ will create a file if it does not exist*

## Reading a file proc (memory)

```
proc read_file_less_memory {file_in} {
        if {[file exists $file_in]} {
                set fh [open $file_in r];
                 while {[gets $fh line] >= 0} {
                         lappend file_data $line;
                }
                close $fh;
                 return $file_data;
        } else {
                puts "File does not exist."
}
```

## Reading a file proc

```
proc read_file {f} {
                puts "Reading :\t $f";
                 if {[catch {set fh [open $f r]} err] } {
                   puts "File does not exist $f:$err";
                  } else {
                        set fh [open $f r];
                         set fdata [split [read $fh] "\n"]
                        close $fh;
                        return $fdata;
                  }
          }
```

## MISC

| User input | gets stdin <VAR NAME TO ASSIGN VALUE> |
|---|---|
| Random number | set i [expr {int(rand() * 6)}] |
| global | change the scope of the variables to global from inside proc |
| Upvar | Used to pass by value to a proc, while instantiating proc do not provide values with a $sign, leave out $sign ; ` |
| argc | number of command line arguments |
| argv0 | Script name |
| argv | all command line arguments |
| set A1 [lindex $argv 0] | first argument |

## Error Handling

```
Error Handling:
proc Div {a b} {
   if {$b == 0} {
       error "Error generated by error" "Info String for error" 401
```

## Error Handling (cont)

```
       error <Error Message> <Error Info> <Error
code>
   } else {
      return [expr $a/$b]
   }
}
if {[catch {puts "Result = [Div 10 0]"} errmsg]} {
   puts "ErrorMsg: $errmsg"
   puts "ErrorCode: $errorCode"
   puts "ErrorInfo:\n$errorInfo\n"
}
```

## Upper/Global

**Global/By value:**

```
proc by_global {A B C} {
        global j1;
        global j2;
        global j3;
        set j1 $A; set j2 $B; set j3 $C;
        puts "In $j1 $j2 $j3"
}
by_global 100 200 300
```

**Upvar/By reference:**

```
proc by_reference {P Q R} {
        upvar $P p;
        upvar $Q q;
        upvar $R r;
        set p [expr $p*2];
        set q [expr $q*2];
        set r [expr $r*2];
        puts "Inside upvar: $p $q $r";
}
by_reference j1 j2 j3;
```

## To Do

- Multi- Dimensional Arrays
- How to set command line arguments for a scripts?
- **packages vs module vs namespace vs proc**
- namespace
- parse_proc_arguments
- seek/tell
- upvar
- uplevel
- lset
- string map
- How to set up Help for a script?
- $argv and $argc
- global
- Scan

## Current Time & Date

```
set systemTime [clock seconds]
set current_time [clock format $systemTime -format
%H:%M:%S]
set ct [clock format $systemTime -format %H_%M]
set current_date [clock format $systemTime -format
%D]
set cdt [clock format $systemTime -format
%m_%d_%y]
```