TCL Advanced

- By Prince Thapa

Topics

- Data Types
- Working with strings
- Glob style matching
- Command line arguments
- Procedures
- File Input/Output

Data Types

Simple Tcl Objects

```
set myVariable "some string"
puts $myVariable
set myVariable 18
puts $myVariable
puts [expr $myVariable + 6 + 9]
```

List

Array

List

List
The list is the basic Tcl data structure. A list is simply an ordered collection of stuff; numbers, words, strings, or other lists.
Lists can be created in several ways:

by setting a variable to be a list of values

```
set int_list {1 2 3 4 5}
set char_list "a b c"
```

with the **split** command

```
#split the string separated by “ “ (whitespace)
set stringlist [split "hello this is tcl" “ ”]
#split the string separated by “:” (colon)
set stringlist [split “abc:def:fgh:ijk” “:”]
```

with the **list** command.

```
set lst [list "item 1" "item 2" "item 3"]
```

The items in list can be iterated through using the **foreach** command:

```
foreach item $my_list {
        $item
}
```

Or by using **for** loop

```
for {set i 0} {$i < [llength $my_list] } {incr i} {
    puts "item at index $i is : [lindex $my_list $i]"
}
```

`lindex my_list index` → Returns the index'th item from the my_list.
`llength my_list` → Returns the number of elements in a my_list.

Array

Array
An array is a Tcl variable with a string-valued index. You can think of the index as a key, and the array as a collection of
related data items identified by different keys. The index, or key, can be any string value.

Syntax

```
set arr(index) value
set Days(1) "Monday"
set Days(2) "Tuesday"
set Days(3) "Wednesday"

for {set i 1} {$i <= [array size Days]} {incr i} {
  puts "Days($i) : $Days($i)"
}
```

```
array set Employee {
      "Name" "Shravan"
      "Age"  23
      "Id"   100
      "Dept" "CAE"
}

puts "size of array Employee : [array size
Employee]"

set keys [array names Employee]
foreach key $keys {
  puts "Employee($key) : $Employee($key)"
}
```

Working with strings

String manipulation

```
set name "Prince Thapa“

# find the length of the string
string length $name

# comparing strings
# string compare ?-nocase? ?-length len? str1 str2

string compare -nocase "prince thapa" $name
string compare -nocase -length 4 "prin" $name

# string equal ?-nocase? str1 str2
string equal "prince thapa" $name
```

String matching

Matches single character

 # string match pattern str

```
string match a* alpha ; # matches returns 1
string match *e* boxer ; #matches
string match {[ab]*x} tcellox ; #doesn't match returns 0
string match {[ab]*x} box; #matches
```

Regular expression

 # regexp pattern str

```
set pat {[x-z1-5]t*.pc}
set filename "xyz123452341test.pc”
puts [regexp $pat $filename] ; # matches
set filename1 "zxy1236test1.pc“
puts [regexp $pat $filename1] ; # doesn't match
```

Glob style matching

The glob command

```
# find all the files ending with ‘.tcl’
set files [glob -type f *.tcl]

# find all the files starting with 'ar' and ending with ‘.tcl’
set files1 [glob -type f ar*.tcl]

# find all the subdirectories
set subdirs [glob -type d *]

# find all the subdirectories whose name starts with ‘a’
set subdirs_startingWith_a [glob -type d a*]
```

Command line arguments

tclsh args.tcl arg1 arg2 … argn

argc : no. of arguments passed
argv : list of arguments passed

Accessing arguments from **argv**

```
for {set i 0} {$i < [llength $argv]}  {incr i}  {
    puts "arg $i is [lindex $argv $i]"
}
```

Or

```
foreach item $argv {
  puts $item
}
```

Procedures

Procedures are nothing but code blocks with series of commands that provide a specific reusable functionality.
It is used to avoid same code being repeated in multiple locations.
Procedures are equivalent to the functions used in many programming languages and are made available in Tcl with the help of **proc** command.

Syntax:

```
proc procedureName {arguments} {
   body
}
```

```
set intList {1 2 5 6 12 15 19}

#procedure declaration
proc printList { my_list } {
  foreach item $my_list {
    puts "$item"
  }
}

printList $intList; #calling procedure printList
```

```
set x 3
set y 9

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

set sum [add $x $y]
set sum1 [add 16 20]
```

File Input/Output

Tcl provides several methods to read from and write to files on disk.
The simplest methods to access a file are via **gets** and **puts**.
When there is a lot of data to be read, however, it is sometimes more efficient to use the read command to load an entire file or a large chunk of it, and then parse the file into lines with the split command.

open *fileName ?access? ?permissions?*
Opens a file and returns a token to be used when accessing the file via gets, puts, close, etc.

•*fileName* is the name of the file to open.
•*access* is the file access mode

close fileID
Closes a file previously opened with open, and flushes any remaining output.

Mode	Meaning
r	Open the file for reading. The file must already exist.
r+	Open the file for reading and writing. The file must already exist.
w	Open the file for writing. Create the file if it doesn't exist, or set the length to zero if it does exist.
w+	Open the file for reading and writing. Create the file if it doesn't exist, or set the length to zero if it does exist.
a	Open the file for writing. The file must already exist. Set the current location to the end of the file.
a+	Open the file for reading and writing. Create the file if it does not exist. Set the current location to the end of the file.

File Input/Output

```
#  open file.txt
set fid [open “file.txt” r]

# create and write to a new file “out_file.txt”
set out_fid [open "out_file.txt" w+]

# close “file.txt”
close $fid

# close “out_file.txt”
Close $out_fid
```

```
# reading file line by line
eof  -> end of file

while { [eof $fid] != 1} {
        set line [gets $fid]
        puts $line
}
```

Source

http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/A2-Tcl/Tcl-6-array+list.html
https://www.tcl.tk/man/tcl8.5/tutorial/Tcl8.html
https://wiki.tcl-lang.org/page/Tcl+Tutorial+Lesson+0

https://www.tutorialspoint.com/execute_tcl_online.php