TCL - STRINGS

The primitive data-type of Tcl is string and often we can find quotes on Tcl as string only language. These strings can contain alphanumeric character, just numbers, boolean, or even binary data. Tcl uses 16 bit unicode characters and alphanumeric characters can contain letters including non Latin characters, number or punctuation.

Boolean value can be represented as 1, yes or true for true and 0, no, or false for false.

String Representations

Unlike other languages, in Tcl, you need not include double quotes when it's only a single word. An example can be,

```
#!/usr/bin/tclsh

set myVariable hello
puts $myVariable
```

When above code is executed, it produces following result.

```
hello
```

When we want to represent multiple strings, we can use either double quotes or curly braces. It is shown below.

```
#!/usr/bin/tclsh

set myVariable "hello world"
puts $myVariable
set myVariable {hello world}
puts $myVariable
```

When above code is executed, it produces following result.

```
hello world
hello world
```

String escape sequence

A character literal can be a plain character $e.g., 'x'$, an escape sequence $e.g., '\backslash t'$, or a universal character $e.g., '\backslash u02C0'$.

There are certain characters in Tcl when they are preceded by a backslash they will have special meaning and they are used to represent like newline \n or tab \t. Here, you have a list of some of such escape sequence codes:

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell

\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Following is the example to show few escape sequence characters:

```
#!/usr/bin/tclsh

puts("Hello\tWorld\n\n");
```

When the above code is compiled and executed, it produces the following result:

```
Hello   World
```

String command

The list of subcommands for string command are listed in the following table.

SN	Methods with Description
1	**compare** string1 string2 Compares string1 and string2 lexographically. Returns 0 if equal, -1 if string1 comes before string2, else 1.
2	string1 string2 Returns the index first occurrence of string1 in string2. If not found, returns -1.
3	**index** string index Returns the character at index.
4	**last** string1 string2 Returns the index last occurrence of string1 in string2. If not found, returns -1.
5	**length** string Returns the length of string.
6	**match pattern** string Returns 1 if the string matches the pattern.
7	

range string index1 index2

Return the range of characters in string from index1 to index2.

8

tolower string

Returns the lowercase string.

9

toupper string

Returns the uppercase string.

10

trim string ?trimcharacters?

Removes trimcharacters in both ends of string. The default trimcharacters is whitespace.

11

trimleft string ?trimcharacters?

Removes trimcharacters in left beginning of string. The default trimcharacters is whitespace.

12

trimright string ?trimcharacters?

Removes trimcharacters in left end of string. The default trimcharacters is whitespace.

13

wordend findstring index

Return the index in findstring of the character after the word containing the character at index.

14

wordstart findstring index

Return the index in findstring of the first character in the word containing the character at index.

Examples of some the commonly used Tcl string sub commands are given below.

String comparison

```
#!/usr/bin/tclsh

set s1 "Hello"
set s2 "World"
set s3 "World"
puts [string compare s1 s2]
if {[string compare s2 s3] == 0} {
puts "String \'s1\' and \'s2\' are same.";
}

if {[string compare s1 s2] == -1} {
puts "String \'s1\' comes before \'s2\'.";
}
```

```
if {[string compare s2 s1] == 1} {
puts "String \'s2\' comes before \'s1\'.";
}
```

When the above code is compiled and executed, it produces the following result:

```
-1
String 's1' comes before 's2'.
String 's2' comes before 's1'.
```

Index of string

```
#!/usr/bin/tclsh

set s1 "Hello World"
set s2 "o"
puts "First occurrence of $s2 in s1"
puts [string first $s2 $s1]
puts "Character at index 0 in s1"
puts [string index $s1 0]
puts "Last occurrence of $s2 in s1"
puts [string last $s2 $s1]
puts "Word end index in s1"
puts [string wordend $s1 20]
puts "Word start index in s1"
puts [string wordstart $s1 20]
```

When the above code is compiled and executed, it produces the following result:

```
First occurrence of o in s1
4
Character at index 0 in s1
H
Last occurrence of o in s1
7
Word end index in s1
11
Word start index in s1
6
```

Length of string

```
#!/usr/bin/tclsh

set s1 "Hello World"
puts "Length of string s1"
puts [string length $s1]
```

When the above code is compiled and executed, it produces the following result:

```
Length of string s1
11
```

Handling cases

```
#!/usr/bin/tclsh

set s1 "Hello World"
puts "Uppercase string of s1"
puts [string toupper $s1]
puts "Lowercase string of s1"
puts [string tolower $s1]
```

When the above code is compiled and executed, it produces the following result:

```
Uppercase string of s1
HELLO WORLD
Lowercase string of s1
hello world
```

Trimming characters

```
#!/usr/bin/tclsh

set s1 "Hello World"
set s2 "World"
puts "Trim right $s2 in $s1"
puts [string trimright $s1 $s2]

set s2 "Hello"
puts "Trim left $s2 in $s1"
puts [string trimleft $s1 $s2]

set s1 " Hello World "
set s2 " "
puts "Trim characters s1 on both sides of s2"
puts [string trim $s1 $s2]
```

When the above code is compiled and executed, it produces the following result:

```
Trim right World in Hello World
Hello
Trim left Hello in Hello World
 World
Trim characters s1 on both sides of s2
Hello World
```

Matching strings

```
#!/usr/bin/tclsh

set s1 "test@test.com"
set s2 "*@*.com"
puts "Matching pattern s2 in s1"
puts [string match "*@*.com" $s1 ]
puts "Matching pattern tcl in s1"
puts [string match {tcl} $s1]
```

When the above code is compiled and executed, it produces the following result:

```
Matching pattern s2 in s1
1
Matching pattern tcl in s1
0
```

Append command

```
#!/usr/bin/tclsh

set s1 "Hello"
append s1 " World"
puts $s1
```

When the above code is compiled and executed, it produces the following result:

```
Hello World
```

Format command

The following table shows the list of format specifiers available in Tcl.

Specifier	Use
%s	String representation
%d	Integer representation
%f	Floating point representation
%e	Floating point representation with mantissa-exponent form
%x	Hexa decimal representation

Some simple examples are given below.

```
#!/usr/bin/tclsh

puts [format "%f" 43.5]
puts [format "%e" 43.5]
puts [format "%d %s" 4 tuts]
puts [format "%s" "Tcl Language"]
puts [format "%x" 40]
```

When the above code is compiled and executed, it produces the following result:

```
43.500000
4.350000e+01
4 tuts
Tcl Language
28
```

Scan command

scan command is used for parsing a string based to the format specifier. Some examples are shown below.

```
#!/usr/bin/tclsh

puts [scan "90" {%[0-9]} m]
puts [scan "abc" {%[a-z]} m]
puts [scan "abc" {%[A-Z]} m]
puts [scan "ABC" {%[A-Z]} m]
```

When the above code is compiled and executed, it produces the following result:

```
1
1
0
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js