TCL - DATA TYPES

The primitive data-type of Tcl is string and often we can find quotes on Tcl as string only language. These primitive data-types in turn create composite data-types for list and associative array. In Tcl, data-types can represent not only simple Tcl objects, but also can represent complex objects like handles, graphic objects*mostlywidgets*, and I/O channels. Let's look in detail about each of the above.

Simple Tcl Objects

In Tcl, whether it is an integer number, boolean, floating point number, or a string. When you want to use a variable, you can directly assign value to it, there is no step of declaration in Tcl. There can be internal representations for these different types of objects. It can transform one data-type to an other when required. The syntax for assigning value to variable is as follows.

```
#!/usr/bin/tclsh

set myVariable 18
puts $myVariable
```

When above code is executed, it produces following result.

```
18
```

The above statement will create a variable name myVariable and stores it as a string even though, we have not used double quotations. Now, if we try to make an arithmetic on the variable, it is automatically turned to an integer. A simple example is shown below.

```
#!/usr/bin/tclsh

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

When above code is executed, it produces following result.

```
33
```

One important thing to note is that, these variables don't have any default values and must be assigned value before they are used.

If we try to print using puts, the number is transformed to proper string. Having two representations, internal and external, help Tcl to create complex data structures easily compared to other languages. Also, Tcl is more efficient due to its dynamic object nature.

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
```

Lists

List is nothing but a group of elements. A group of words either using double quotes or curly braces can be used to represent a simple list. A simple list is shown below.

```
#!/usr/bin/tclsh

set myVariable {red green blue}
puts [lindex $myVariable 2]
set myVariable "red green blue"
puts [lindex $myVariable 1]
```

When above code is executed, it produces following result.

```
blue
green
```

Associative Array

Associative arrays have an index *key* that is not necessarily an integer. It is generally a string that acts like key value pairs. A simple example is shown below.

```
#!/usr/bin/tclsh

set  marks(english) 80
puts $marks(english)
set  marks(mathematics) 90
puts $marks(mathematics)
```

When above code is executed, it produces following result.

```
80
90
```

Handles

Tcl handles are commonly used to represent files and graphics objects. These can include handles to network requests and also other channels like serial port communication, sockets or I/O devices. The following is an example where a file handle is created.

```
set myfile [open "filename" r]
```

You will see more detail on files in the **Tcl file I/O** chapter.