



Tcl Basics

Syntax Fundamentals

Command Structure	<code>commandName arg1 arg2 ... argN</code>
	Commands are space-separated.
Variable Assignment	<code>set variableName value</code>
	Assigns a value to a variable.
Variable Substitution	<code>\$variableName</code>
	Substitutes the value of a variable.
Command Substitution	<code>[commandName arg1 arg2]</code>
	Executes a command and substitutes the result.
Comments	<code># This is a comment</code>
Quoting	<ul style="list-style-type: none"> <code>"</code> - Prevents word splitting and allows variable substitution. <code>{}</code> - Prevents word splitting and inhibits all substitutions.

Basic Commands

<code>puts</code>	Prints a string to the standard output. <code>puts "Hello, World!"</code>
<code>set</code>	Assigns a value to a variable. <code>set name "John"</code> <code>puts "Hello, \$name!"</code>
<code>expr</code>	Evaluates an expression. <code>set result [expr 2 + 2]</code> <code>puts \$result</code>
<code>if</code>	Conditional execution. <code>if { \$x > 10 } {</code> <code> puts "x is greater than 10"</code> <code>}</code>
<code>for</code>	Looping construct. <code>for {set i 0} {\$i < 5} {incr i}</code> { <code> puts "Iteration \$i"</code> }
<code>proc</code>	Defines a procedure. <code>proc greet {name} {</code> <code> puts "Hello, \$name!"</code> } <code>greet "Alice"</code>

Control Flow and Procedures

Conditional Statements

```

if-elseif-else

if {condition1} {
    # Code to execute if condition1 is
    true
} elseif {condition2} {
    # Code to execute if condition2 is
    true
} else {
    # Code to execute if all conditions
    are false
}

```

switch

```

switch $variable {
    value1 { code_block1 }
    value2 { code_block2 }
    default { default_code_block }
}

```

Looping Constructs

```

while e
    while {condition} {
        # Code to execute while the
        condition is true
    }

```

```

foreach ch
    foreach variable list {
        # Code to execute for each
        element in the list
    }

```

```

break k
    while {1} {
        if {condition} { break }
    }

```

```

continue
    Skips the current iteration and
    continues with the next.

    foreach i {1 2 3} {
        if { $i == 2 } { continue }
        puts $i
    }

```

Procedures (Functions)

Procedure Definition

```
proc procedureName {arg1 arg2 ...} {  
    # Procedure body  
    return value  
}
```

Calling a Procedure

```
procedureName value1 value2
```

Example

```
proc add {a b} {  
    return [expr $a + $b]  
}  
  
set sum [add 5 3]  
puts $sum ;# Output: 8
```

Variable Scope - By default, variables are local to the procedure. Use `global` to access global variables.

```
set globalVar 10  
proc modifyGlobal {} {  
    global globalVar  
    set globalVar [expr $globalVar + 5]  
}  
modifyGlobal  
puts $globalVar ;# Output: 15
```

String Manipulation and Lists

String Operations

`string length` Returns the length of a string.

`string length "Hello"`
;# Output: 5

`string index` Returns the character at a specific index.

`string index "Hello" 1`
;# Output: e

`string range` Extracts a substring.

`string range "Hello" 1 3`
;# Output: ell

`string compare` Compares two strings.

`string compare "apple" "banana"`
;# Output: -1 (apple < banana)

`string tolower` Converts a string to lowercase.

`string tolower "HELLO"`
;# Output: hello

`string toupper` Converts a string to uppercase.

`string toupper "hello"`
;# Output: HELLO

List Manipulation

`list` Creates a list.

`list 1 2 3`
;# Output: 1 2 3

`lindex` Returns an element from a list by index.

`lindex {1 2 3} 1`
;# Output: 2

`llength` Returns the length of a list.

`llength {1 2 3}`
;# Output: 3

`lappend` Appends elements to a list.

`set myList {1 2}`
`lappend myList 3 4`
`puts $myList ;# Output: 1 2 3 4`

`linsert` Inserts elements into a list at a given index.

`linsert {1 2 3} 1 a b`
;# Output: 1 a b 2 3

`lreplace` Replaces elements in a list.

`lreplace {1 2 3} 1 1 a b`
;# Output: 1 a b 3

File I/O and Regular Expressions

File Input/Output

<code>open</code>	Opens a file. <code>set file [open "myfile.txt" r]</code>
<code>read</code>	Reads data from a file. <code>set data [read \$file]</code>
<code>puts</code> (to file)	Writes data to a file. <code>puts \$file "Hello, file!"</code>
<code>close</code>	Closes a file. <code>close \$file</code>
<code>gets</code>	Reads a line from a file. <code>set line [gets \$file lineVar]</code>

Regular Expressions

<code>regexp</code>	Matches a regular expression against a string. <code>regexp {pattern} string [matchVar] [submatchVar1] [submatchVar2] ...</code>
Example: Matching	<code>if { [regexp {\d+} "abc123def"] } { puts "Match found"</code> }
Example: Capturing	<code>regexp {(\d+)} "abc123def" match number</code> <code>puts "Match: \$match, Number: \$number"</code>
<code>regsub</code>	Substitutes regular expression matches. <code>regsub {pattern} string replacement varName</code>
Example: Substitution	<code>regsub {\s+} "Hello World" " " result</code> <code>puts \$result ;# Output: Hello World</code>