Tcl Programming Cheatsheet

A quick reference guide to the Tcl scripting language, covering syntax, commands, and common usage patterns.

Tcl Basics

Syntax Fundamentals

Command Structure	`commandName arg1 arg2 ... argN` Commands are space-separated.
Variable Assignment	`set variableName value` Assigns a value to a variable.
Variable Substitution	`$variableName` Substitutes the value of a variable.
Command Substitution	`[commandName arg1 arg2]` Executes a command and substitutes the result.
Comments	`# This is a comment`
Quoting	`"` - Prevents word splitting and allows variable substitution. `{}` - Prevents word splitting and inhibits all substitutions.

Basic Commands

`puts`

Prints a string to the standard output.

```
puts "Hello, World!"
```

`set`

Assigns a value to a variable.

```
set name "John"
puts "Hello, $name!"
```

`expr`

Evaluates an expression.

```
set result [expr 2 + 2]
puts $result
```

`if`

Conditional execution.

```
if { $x > 10 } {
  puts "x is greater than 10"
}
```

`for`

Looping construct.

```
for {set i 0} {$i < 5} {incr i} {
  puts "Iteration $i"
}
```

`proc`

Defines a procedure.

```
proc greet {name} {
  puts "Hello, $name!"
}
greet "Alice"
```

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`

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

`foreach`

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

`break`

Exits the current loop.

```
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

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

Regular Expressions

`regexp`	Matches a regular expression against a string. `regexp {pattern} string [matchVar] [submatchVar1] [submatchVar2] ...`

Example: Matching	`if { [regexp {\d+} "abc123def" ] } {`
	`  puts "Match found"`
	`}`

Example: Capturing	`regexp {(\d+)} "abc123def" match number`
	`puts "Match: $match, Number: $number"`

`regsub`	Substitutes regular expression matches. `regsub {pattern} string replacement varName`

Example: Substitution	`regsub {\s+} "Hello  World" " " result`
	`puts $result ;# Output: Hello World`