Lua Cheatsheet

A concise reference for Lua syntax, data types, control structures, and standard library functions. Perfect for quick lookup and reminding yourself of the essentials.

Basics & Syntax

Basic Syntax

Comments	`--` Single-line comment `--[[ ... --]]` Multi-line comment
Variables	`local` - Scope limited to the block `variable_name = value` - Global scope if not declared local
Assignment	`=` Assignment operator
Chaining Assignment	`a, b = 10, 20` Assigns 10 to a and 20 to b.
Statements	Statements do not need to end with a semicolon (;)
Case Sensitivity	Lua is case sensitive

Data Types

- `nil` : Represents the absence of a value.
- `boolean` : `true` or `false` .
- `number` : Represents real (double-precision floating-point) numbers.
- `string` : Immutable sequence of characters.
- `table` : Associative array (hash table). The primary data structure in Lua.
- `function` : First-class functions.
- `userdata` : Raw memory blocks, used to represent new types created in C.

Operators

Arithmetic Operators	`+` , `-` , `*` , `/` , `^` (exponentiation), `%` (modulo)
Relational Operators	`==` (equal), `~=` (not equal), `<` , `>` , `<=` , `>=`
Logical Operators	`and` , `or` , `not`
Concatenation Operator	`..` (string concatenation)
Length Operator	`#` (length of a string or table)

Control Structures

Conditional Statements

`if condition then ... end`

`if condition then ... else ... end`

`if condition then ... elseif condition then ... else ... end`

Example:

```
if age >= 18 then
  print("Adult")
else
  print("Minor")
end
```

Looping Statements

`while condition do ... end`

`repeat ... until condition`

`for i = start, end, step do ... end`

`for key, value in pairs(table) do ... end`

Example (while loop):

```
i = 1
while i <= 5 do
  print(i)
  i = i + 1
end
```

Example (for loop):

```
for i = 1, 5 do
  print(i)
end
```

Example (foreach loop):

```
my_table = {"a", "b", "c"}
for index, value in ipairs(my_table) do
  print(index, value)
end
```

Break and Return

`break`	Exits the innermost loop.
`return`	Exits the function.

Tables

Table Basics

Tables are the primary data structure in Lua. They are associative arrays that can be indexed with any value (except `nil`).

Table Creation and Access

Creating a Table	`my_table = {}` (empty table) `my_table = {key1 = "value1", key2 = "value2"}` `my_table = {"value1", "value2", "value3"}` (array-like table)
Accessing Elements	`my_table.key1` or `my_table["key1"]` (using string keys) `my_table[1]` (using numerical indices - 1-based)
Adding/Modifying Elements	`my_table.new_key = "new_value"` `my_table["existing_key"] = "updated_value"`
Removing Elements	`my_table.key = nil`

Table Functions

- `table.insert(table, [pos,] value)` : Inserts a new element at a given position.
- `table.remove(table, [pos])` : Removes an element at a given position.
- `table.sort(table, [comp])` : Sorts the table elements.
- `table.concat(table, [sep, i, j])` : Concatenates elements of a table into a string.

Example (insert):

```
table.insert(my_table, 1, "first")
```

Example (remove):

```
table.remove(my_table, 2)
```

Functions

Function Definition

`function function_name(arg1, arg2, ...) ... end`
`local function function_name(arg1, arg2, ...) ... end` (local function)

Example:

```
function add(a, b)
  return a + b
end
```

Function Calls

Calling a Function	`function_name(arg1, arg2, ...)`
Calling a Method	`object:method_name(arg1, arg2, ...)` (equivalent to `object.method_name(object, arg1, arg2, ...)`)
Returning Values	Functions can return multiple values: `return value1, value2`

Variable Arguments

Use `...` to accept a variable number of arguments.
Arguments are accessed via the `arg` table (deprecated in Lua 5.2, use `...` directly).
Use `table.pack(...)` to pack the arguments into a table, which also stores the number of arguments.

Example:

```
function sum(...)
  local total = 0
  local args = {...}
  for i, v in ipairs(args) do
    total = total + v
  end
  return total
end

print(sum(1, 2, 3, 4)) -- Output: 10
```