Lua Cheatsheet

A comprehensive cheat sheet for the Lua scripting language, covering syntax, data structures, control flow, functions, metatables, and common APIs.

Lua Basics

Syntax and Comments

Single-line comment:	`-- This is a comment`
Multi-line comment:	`--[[ This is a multi-line comment ]]`
Variables:	`local myVariable = 10`
Assignment:	`x = 1; y = 'hello'`
Multiple assignment:	`a, b = 1, 2`

Data Types

Nil:	`nil` (absence of a value)
Boolean:	`true`, `false`
Number:	`42`, `3.14159`
String:	`'hello'`, `"world"`
Table:	Associative array (object)
Function:	First-class citizen

Operators

Arithmetic: `+`, `-`, `*`, `/`, `%`, `^` (exponentiation), `-` (unary minus)

Relational: `==`, `~=`, `<`, `>`, `<=`, `>=`

Logical: `and`, `or`, `not`

String Concatenation: `..`

Length Operator: `#` (length of a table or string)

Control Flow

Conditionals

```lua
if condition then
  -- code
elseif condition then
  -- code
else
  -- code
end
```

Example:

```lua
if x > 10 then
  print("x is greater than 10")
elseif x < 10 then
  print("x is less than 10")
else
  print("x is equal to 10")
end
```

Loops

While Loop:

```lua
while condition do
  -- code
end
```

For Loop (numeric):

```lua
for i = start, end, step do
  -- code
end
```

For Loop (generic):

```lua
for key, value in pairs(table) do
  -- code
end
```

Repeat-Until Loop:

```lua
repeat
  -- code
until condition
```

Loop Control

Break:	Exits the current loop.
Return:	Exits the current function.

Tables and Functions

Tables

Table Creation:

```lua
table = {}
```

Adding Key-Value Pairs:

```lua
table["key"] = "value"
table.key = "value"  -- Equivalent when key is a valid identifier
```

Accessing Values:

```lua
value = table["key"]
value = table.key
```

Arrays (Tables with Numeric Indices):

```lua
array = {"a", "b", "c"}
print(array[1])  -- "a" (Lua is 1-indexed)
```

Functions

Function Definition:

```
function functionName(arg1, arg2)
  -- code
  return value
end
```

Calling a Function:

```
result = functionName(value1, value2)
```

Anonymous Functions:

```
myFunc = function(x)
  return x * 2
end
```

Variable Arguments:

```
function varArgFunc(a, ...)
  local args = { ... }
  for i, v in ipairs(args) do
    print(i, v)
  end
end
```

Scope

Global:	Accessible everywhere.
Local:	Accessible only within its scope (e.g., function or block).

Metatables and Object Orientation

Metatables

Setting a Metatable:

```
mt = {}
setmetatable(table, mt)
```

Common Metamethods:

- `__index` : Table indexing fallback.
- `__newindex` : Table assignment fallback.
- `__add` , `__sub` , `__mul` , `__div` : Arithmetic operators.
- `__tostring` : String conversion.

Example:

```
mt.__index = function(table, key)
  return "Default Value"
end
```

Object Orientation

Creating a Class:

```
MyClass = {}
MyClass.__index = MyClass
```

Constructor:

```
function MyClass:new(value)
  local self = setmetatable({}, MyClass)
  self.value = value
  return self
end
```

Methods:

```
function MyClass:getValue()
  return self.value
end
```

Usage:

```
local instance = MyClass:new(10)
print(instance:getValue())
```

Common APIs

String Library

string.len(s): Returns the length of the string `s`.

string.sub(s, i, j): Extracts a substring from `s` starting at index `i` and ending at index `j`.

string.find(s, pattern, init, plain): Searches for the first occurrence of `pattern` in `s`. `init` is an optional starting index, and `plain` is a boolean to disable pattern matching.

string.gsub(s, pattern, repl, n): Replaces occurrences of `pattern` in `s` with `repl`. `n` is an optional maximum number of replacements.

string.format(formatstring, ...): Returns a formatted string using the given format string and arguments.

string.upper(s), string.lower(s): Converts the string `s` to uppercase or lowercase, respectively.

Table Library

table.insert(t, pos, value): Inserts `value` into table `t` at position `pos`. If `pos` is omitted, it defaults to `#t + 1` (appends to the end).

table.remove(t, pos): Removes the element at position `pos` from table `t`. Returns the value of the removed element.

table.sort(t, comp): Sorts the elements of table `t` in place, using the optional comparison function `comp`.

table.concat(t, sep, i, j): Concatenates the strings in table `t` from index `i` to `j`, with `sep` as a separator string.

Math Library

math.random(m, n): Returns a pseudo-random number. If called without arguments, returns a float between 0 and 1. If called with two integer arguments `m` and `n`, returns an integer between `m` and `n`.

math.abs(x): Returns the absolute value of `x`.

math.floor(x), math.ceil(x): Returns the largest integer less than or equal to `x`, or the smallest integer greater than or equal to `x`, respectively.

math.sqrt(x): Returns the square root of `x`.

math.sin(x), math.cos(x), math.tan(x): Trigonometric functions (x in radians).

math.pi: The value of pi.