Lua Small scripting language

Table your data → `setmetatable` → **Metatable** __index, __add... → `missing key / operator` → **Metamethod** function or table → `result` → **Value** returned

CONCEPTS

Tables are the only data structure: arrays, maps, objects and modules are all tables. Arrays start at index 1.

Variables are global unless declared local; typos create new globals silently. Only nil and false are falsy: 0 and "" are true.

RUN THE INTERPRETER

`lua <file.lua> [<args>]`
Run a script with arguments

`lua`
Start the interactive REPL

COMPILE WITH LUAC

`luac -p <file.lua>`
Check syntax only

TYPES AND VARIABLES

`local <name> = <value>`
Declare a local variable

`type(<v>)`
Name of the value's type

OPERATORS

`<a> // <b>`
Floor division

CONTROL FLOW

`if <c> then ... elseif <c> then ... else ... end`
Conditional

`for i = <a>, <b> [, <step>] do ... end`
Numeric loop, end inclusive

`for i, v in ipairs(<t>) do ... end`
Array loop, stops at first nil

`for k, v in pairs(<t>) do ... end`
All keys, unspecified order

FUNCTIONS

`function <name>(<args>) ... end`
Define a function

`local function <name>(<args>) ... end`
Local, can recurse

TABLES AND METATABLES

`local t = {1, 2, 3}`
Array constructor

TABLES AND METATABLES (cont.)

`local t = {name = "x", ["k k"] = 1}`
Record constructor

`setmetatable(<t>, <mt>)`
Attach a metatable, returns t

`obj:method(<args>)`
Sugar for obj.method(obj, args)

STRINGS

`#<s>`
Length in bytes

`<s>:find(<pat> [, <init> [, true]])`
Start and end index; true = plain text

`<s>:gsub(<pat>, <repl> [, <n>])`
Replace; returns string and count

PATTERNS

`.  %a  %d  %l  %u  %s  %w  %p  %x`
Any, letter, digit, lower, upper, space, alnum, punct, hex

`[abc]  [a-z]  [^0-9]`
Set, range, negated set

FORMAT SPECIFIERS

`%d  %i`
Integer

TABLE LIBRARY

`table.insert(<t>, <v>)`
Append

`table.insert(<t>, <pos>, <v>)`
Insert, shifting up

`table.remove(<t> [, <pos>])`
Remove last or at pos, returns it

`table.concat(<t> [, <sep>])`
Join strings

MATH

`math.floor(<x>)`
`math.ceil(<x>)`
Round down / up

ERRORS

`error(<msg> [, <level>])`
Raise an error; level 2 blames the caller

`assert(<v> [, <msg>])`
Raise if v is falsy, else return args

ERRORS (cont.)

`pcall(<f>, ...)`
Protected call: true, results or false, err

MODULES

`local m = require("<name>")`
Load a module once, cached

`return M`
End a module file with its table

COROUTINES

`coroutine.create(<f>)`
New suspended coroutine

OS AND I/O

`print(<values>)`
Print tab-separated with newline

`io.open(<path>, "r")`
Open a file; nil, err on failure

LUAROCKS

`luarocks install <rock>`
Install a package