# Lua Small scripting language

**START HERE**

**`lua <file.lua>`**
Run a script

**`for i = 1, 10 do ... end`**
Numeric loop

**`lua`**
Start the REPL

**`for k, v in pairs(<t>) do`**
Iterate a table

**`lua -e "<code>"`**
Run a string

**`string.format("%d", n)`**
Format text

**`local x = 10`**
Local variable

**`pcall(<f>, ...)`**
Catch an error

**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.

Lua 5.4 has separate integer and float subtypes: 3 // 2 is 1, 3 / 2 is 1.5, and / and ^ always give floats.

Calling a method with a dot instead of a colon (obj.m() vs obj:m()) leaves self nil: the classic OOP slip.

## RUN THE INTERPRETER

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

**`lua`**
Start the interactive REPL

**`lua -e "<code>"`**
Execute a string

**`lua -i <file.lua>`**
Run a script, then drop into the REPL

**`arg[0]`**
Script name; arguments are arg[1..n]

## COMPILE WITH LUAC

**`luac -o <out.luac> <file.lua>`**
Compile to bytecode

**`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

**`<a> and <b> or <c>`**
Ternary idiom (breaks if b is falsy)

**`tostring(<v>) / tonumber(<s> [, <base>])`**
Convert values

**`#<t_or_s>`**
Length of table (sequence) or string

**`<a> .. <b>`**
Concatenate strings

## OPERATORS

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

**`<a> % <b>`**
Modulo, sign follows divisor

**`<a> ^ <b>`**
Power, always a float

## 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

**`while <c> do ... end`**
While loop

**`goto <label> ... ::<label>::`**
Jump; the only way to continue

## FUNCTIONS

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

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

**`function <name>(...) local a = {...} end`**
Variadic function

**`select("#", ...)`**
Count varargs, nils included

**`select(<n>, ...)`**
Varargs from position n

## TABLES AND METATABLES

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

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

**`<t>.<key>`** **`<t>[<expr>]`**
Field access

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

**`mt.__index = <table_or_fn>`**
Fallback for missing keys, inheritance

**`mt.__call = <fn>`**
Make a table callable

**`mt.__tostring = <fn>`**
Custom tostring/print output

## TABLES AND METATABLES (cont.)

**`rawget(<t>, <k>) / rawset(<t>, <k>, <v>)`**
Bypass metamethods

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

**`function Class:method() ... end`**
Define a method with self

## STRINGS

**`#<s>`**
Length in bytes

**`<s>:sub(<i> [, <j>])`**
Substring, 1-based, negatives from end

**`<s>:upper()`** **`<s>:lower()`**
Change case

**`<s>:rep(<n> [, <sep>])`**
Repeat

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

**`<s>:match(<pat>)`**
First match or captures

**`<s>:gmatch(<pat>)`**
Iterator over all matches

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

**`string.format(<fmt>, ...)`**
printf-style formatting

## 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

**`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

## RECIPES

### Class with inheritance

```
local Animal = {}
Animal.__index = Animal
function Animal.new(name)
return setmetatable({name =
name}, Animal) end
function Animal:speak() return
self.name .. " ..." end
local Dog = setmetatable({},
{__index = Animal})
Dog.__index = Dog
function Dog.new(name) return
setmetatable(Animal.new(name),
Dog) end
```

### Split a string

```
local parts = {}
for w in (s ..
","):gmatch("(.-),") do
parts[#parts + 1] = w end
```

### Read a file safely

```
local f, err = io.open(path,
"r")
if not f then error(err) end
local text = f:read("a")
f:close()
```