# Lua Cheatsheet

## Tooling & Running Code Run

- Run file: `lua main.lua`
- REPL: `lua` (then type expressions)
- LuaJIT: `luajit main.lua` (if using LuaJIT)

### Packages

- LuaRocks (common): `luarocks install <pkg>`
- Require: `local m = require("modname")`

---

## Basics: Values, Variables, Types Hello

```
print("Hello, Lua")
```

### Types

- `nil`, `boolean`, `number`, `string`, `table`, `function`, `userdata`, `thread`
- `type(x)` returns type name

### Variables

- Local: `local x = 1`
- Global (avoid): `x = 1`
- Multiple assign: `local a,b = 1,2`
- Unassigned becomes `nil`

### Truthiness

- Only `false` and `nil` are falsey
- `0` and `""` are truthy

---

## Operators Arithmetic

- `+ - * /` integer division (Lua 5.3+): `//`
- Mod: `%` Power: `^`

### Comparison

- `== ~= < <= > >=`
- Tables compare by reference (identity), not deep-equal

### Logic

- `and`, `or`, `not`
- Return operands (not strict booleans): `a or b`

### Concatenation

- `..` (strings)

---

## Control Flow If / elseif / else `if x > 0 then`

```
print("pos") elseif x < 0 then print("neg") else print("zero") end
```

### While / repeat

- `while cond do ... end`
- `repeat ... until cond` (runs at least once)

### For loops

- Numeric: `for i=1,10,2 do ... end` (step optional)
- Generic: `for k,v in pairs(t) do ... end`

### Break

- `break` exits nearest loop

---

## Functions & Closures Definitions

- `local function add(a,b) return a+b end`
- Sugar: `function add(a,b) return a+b end` (global)

### Anonymous functions

- `local f = function(x) return x*2 end`

### Multiple returns

- `return a, b, c`
- Assignment: `local x,y = f()`
- Only last expr expands in lists: `g(f())` vs `g((f()))`

### Varargs

- `function f(...) local args={...} end`

---

## Tables (The One Data Structure) Literals

- Array-ish: `local xs = {10, 20, 30}`
- Map-ish: `local m = {a=1, b=2}`
- Mixed: `{[1]="x", key="y"}`

### Indexing

- `t[k]` field sugar for string keys: `t.key == t["key"]`
- Set: `t[k] = v` delete: `t[k] = nil`

### Length

- `#t` (works reliably for sequences; undefined for "holes")

### Iteration

- `pairs(t)` for all keys
- `ipairs(t)` for array part 1..n (Lua 5.1/5.2; 5.3+ still exists but less emphasized)

### Standard table funcs

- `table.insert`, `table.remove`
- `table.sort(xs)`
- `table.concat(xs, ",")`

---

## Strings Basics

- Immutable
- Concatenate: `"a" .. "b"`
- Length: `#s`

### Patterns (not full regex)

- Match: `string.match(s, "patt")`
- Find: `string.find(s, "patt")`
- Replace: `string.gsub(s, "patt", "repl")`

### Format

- `string.format("x=%d", x)`

## Modules & Requires **Module pattern** `local M = {} function M.hello() return "hi" end return M`

### Require

- `local M = require("mymod")`
- Cached in `package.loaded`
- Search paths: `package.path`, `package.cpath`

## Metatables (Power Feature) **Basics**

- Attach behavior: `setmetatable(t, mt)`
- Read: `getmetatable(t)`

### Common metamethods

- `__index`: fallback lookup (inheritance/prototypes)
- `__newindex`: intercept writes
- `__call`: make table callable
- `__tostring`: stringify
- Arithmetic: `__add`, etc.

**Prototype OO sketch** `local Vec = {} Vec.__index = Vec function Vec.new(x,y) return setmetatable({x=x,y=y}, Vec) end function Vec:len() return math.sqrt(self.x*self.x + self.y*self.y) end`

## OOP-ish Patterns **Colon sugar**

- Define: `function T:m(x) ... end`
- Call: `obj:m(1)` equals `obj.m(obj, 1)`

### Class-like

- Constructor returns table with metatable to methods table
- Use `__index` for method lookup

## Errors & Protected Calls **Error**

- Throw: `error("bad")`
- Assert: `assert(x, "msg")`

### pcall / xpcall

- `local ok, res = pcall(f, arg)`
- `xpcall(f, debug.traceback)` for stack traces

### Common pattern

- Return `nil, err` instead of raising for expected failures

## Coroutines (Lua Threads) **Basics**

- Create: `co = coroutine.create(f)`
- Resume: `coroutine.resume(co, ...)`
- Yield: `coroutine.yield(v)`
- Status: `coroutine.status(co)`

### Typical use

- Cooperative multitasking; game loops, async-ish flows

## Testing & Style **Formatting / lint**

- Stylua (formatter), Luacheck (linter) (external)
- Lua language server for diagnostics (LSP)

### Unit tests

- Busted (common), LuaUnit

## Common Idioms (Quick Recipes) **Default values**

- `x = x or 10` (works because `nil/false` are falsey)

### Safe nested access

- `local v = t and t.a and t.a.b`

### Copy table (shallow)

- `local c = {}; for k,v in pairs(t) do c[k]=v end`

### Avoid globals

- Use `local`; consider `local _ENV = ...` patterns in modules

## Mini Reference

- Only falsey: `false`, `nil`
- `..` concat; `#` length; `~=` not equal
- Tables are references; copying needs manual work
- Metatables enable operator overloading / prototypes