Referência Rápida de Lua

Tabelas, funções, metatables, coroutines, módulos, padrões

Básico

Hello World

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

Variáveis e Atribuição

```
local name = "Lua"      -- local variable
x = 10                  -- global (avoid)
local a, b = 1, 2      -- multiple assignment
a, b = b, a            -- swap values
```

Comentários

```
-- single line comment
--[[ multi-line
     comment ]]
```

Operadores

`+ - * / %`	Operadores aritméticos
`//`	Divisão inteira (5.3+)
`^`	Exponenciação
`..`	Concatenação de strings
`#`	Operador de comprimento
`== ~=`	Igual / diferente
`and or not`	Operadores lógicos

Tipos

Tipos de Dados

`nil`	Ausência de valor; falso
`boolean`	true ou false
`number`	Ponto flutuante de dupla precisão (ou inteiro no 5.3+)
`string`	Sequência de bytes imutável
`table`	Array associativo (único tipo composto)
`function`	Closure de primeira classe
`userdata`	Dados C encapsulados para Lua
`thread`	Identificador de coroutine

Verificação de Tipo

```
print(type(42))        -- "number"
print(type("hi"))      -- "string"
print(type(nil))       -- "nil"
print(type({}))        -- "table"
```

Tabelas

Tabelas Estilo Array

```
local fruits = {"apple", "banana", "cherry"}
print(fruits[1])          -- "apple" (1-indexed)
table.insert(fruits, "date")
table.remove(fruits, 2)   -- remove "banana"
print(#fruits)            -- length
```

Tabelas Estilo Dicionário

```
local user = {name = "Alice", age = 30}
user.email = "a@b.com"    -- add field
user["name"] = "Bob"      -- bracket access
user.age = nil            -- remove field
```

Funções de Tabela

`table.insert(t, v)`	Adicionar valor ao final do array
`table.insert(t, i, v)`	Inserir na posição i
`table.remove(t, i)`	Remover elemento na posição i
`table.sort(t [,cmp])`	Ordenar array no local
`table.concat(t, sep)`	Juntar elementos do array em string
`table.move(t,a,b,c)`	Mover elementos de a..b para posição c

Funções

Definição de Função

```
local function add(a, b)
  return a + b
end
local mul = function(a, b) return a * b end
print(add(2, 3))  -- 5
```

Varargs e Múltiplos Retornos

```
local function sum(...)
  local s = 0
  for _, v in ipairs({...}) do s = s + v end
  return s
end
local function swap(a, b) return b, a end
local x, y = swap(1, 2)
```

Closures

```
local function counter()
  local n = 0
  return function()
    n = n + 1; return n
  end
end
local c = counter()
print(c(), c())  -- 1  2
```

Fluxo de Controle

Condicionais

```
if x > 0 then
  print("positive")
elseif x == 0 then
  print("zero")
else
  print("negative")
end
```

Laços

```
for i = 1, 10 do print(i) end
for i = 10, 1, -1 do print(i) end
for k, v in pairs(tbl) do print(k, v) end
for i, v in ipairs(arr) do print(i, v) end
```

While e Repeat

```
while x > 0 do x = x - 1 end
repeat
  x = x + 1
until x >= 10
```

Strings

Funções de String

`string.len(s) / #s`	Comprimento da string em bytes
`string.sub(s, i, j)`	Substring de i a j
`string.upper(s)`	Converter para maiúsculas
`string.lower(s)`	Converter para minúsculas
`string.rep(s, n)`	Repetir string n vezes
`string.reverse(s)`	Inverter string
`string.format(fmt, ...)`	Formatação estilo printf
`string.find(s, pat)`	Encontrar padrão, retornar índices
`string.gsub(s, pat, rep)`	Substituição global
`string.gmatch(s, pat)`	Iterador sobre correspondências do padrão

Caracteres de Padrão

`.`	Qualquer caractere
`%a / %A`	Letras / não-letras
`%d / %D`	Dígitos / não-dígitos
`%w / %W`	Alfanumérico / não-alfanumérico
`%s / %S`	Espaço em branco / não-espaço
`%p`	Pontuação
`* + - ?`	Guloso, guloso, preguiçoso, opcional

Metatables

Definindo Metatables

```
local mt = {}
mt.__add = function(a, b)
  return {val = a.val + b.val}
end
local a = setmetatable({val=1}, mt)
local b = setmetatable({val=2}, mt)
local c = a + b  -- c.val == 3
```

Metamétodos Comuns

`__index`	Busca de chaves ausentes (tabela ou função)
`__newindex`	Interceptar atribuição de nova chave
`__add / __sub / __mul`	Operadores aritméticos
`__eq / __lt / __le`	Operadores de comparação
`__tostring`	Representação em string personalizada
`__len`	Operador # personalizado
`__call`	Chamar tabela como função
`__concat`	Operador .. personalizado

POO com Metatables

```
local Dog = {}; Dog.__index = Dog
function Dog.new(name)
  return setmetatable({name=name}, Dog)
end
function Dog:bark() print(self.name.." says Woof") end
local d = Dog.new("Rex"); d:bark()
```

Coroutines

Ciclo de Vida da Coroutine

`coroutine.create(f)`	Criar coroutine a partir de função
`coroutine.resume(co, ...)`	Iniciar ou continuar coroutine
`coroutine.yield(...)`	Suspender execução, retornar valores
`coroutine.status(co)`	"running", "suspended", "dead"
`coroutine.wrap(f)`	Criar wrapper de coroutine chamável

Exemplo de Coroutine

```
local function gen(max)
  for i = 1, max do coroutine.yield(i) end
end
local co = coroutine.wrap(gen)
print(co(5))  -- 1
print(co())   -- 2
```

Módulos

Criando um Módulo

```
-- mylib.lua
local M = {}
function M.greet(name)
  return "Hello, " .. name
end
return M
```

Referência Rápida de Lua

Usando Módulos

```
local mylib = require("mylib")
print(mylib.greet("World"))
```

Bibliotecas Padrão

`math`	Funções matemáticas (sin, random, huge, etc.)
`string`	Manipulação de strings e padrões
`table`	Manipulação de tabelas (insert, sort, etc.)
`io`	Operações de E/S de arquivos
`os`	Recursos do SO (time, clock, execute)
`debug`	Interface de depuração (use com moderação)

Padrões Comuns

Idioma Ternário

```
-- Lua has no ternary; use and/or idiom
local val = condition and "yes" or "no"
-- Caution: fails if "yes" is false/nil
```

Acesso Seguro a Tabelas

```
local function get(t, ...)
  for _, k in ipairs({...}) do
    if type(t) ~= "table" then return nil end
    t = t[k]
  end
  return t
end
get(config, "db", "host")  -- safe nested access
```

Iterando com ipairs vs pairs

```
-- ipairs: array part, stops at first nil
for i, v in ipairs(arr) do print(i, v) end
-- pairs: all keys (unordered)
for k, v in pairs(tbl) do print(k, v) end
```