REFERENCIA RÁPIDA DE LUA

Tablas, funciones, metatables, corrutinas, módulos, patrones

Fundamentos

Hola Mundo

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

Variables y asignación

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

Comentarios

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

Operadores

`+ - * / %`	Operadores aritméticos
`//`	División entera (5.3+)
`^`	Exponenciación
`..`	Concatenación de cadenas
`#`	Operador de longitud
`== ~=`	Igual / no igual
`and or not`	Operadores lógicos

Tipos

Tipos de datos

`nil`	Ausencia de valor; falsy
`boolean`	true o false
`number`	Flotante de doble precisión (o entero en 5.3+)
`string`	Secuencia de bytes inmutable
`table`	Array asociativo (único tipo compuesto)
`function`	Closure de primera clase
`userdata`	Datos C envueltos para Lua
`thread`	Manejador de corrutina

Verificación de tipo

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

Tablas

Tablas 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
```

Tablas estilo diccionario

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

Funciones de tabla

`table.insert(t, v)`	Agregar valor al array
`table.insert(t, i, v)`	Insertar en la posición i
`table.remove(t, i)`	Eliminar elemento en la posición i
`table.sort(t [,cmp])`	Ordenar array en el lugar
`table.concat(t, sep)`	Unir elementos en cadena
`table.move(t,a,b,c)`	Mover elementos de a..b a la posición c

Funciones

Definición de función

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

Variádicas y retornos múltiples

```
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
```

Flujo de control

Condicionales

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

Bucles

```
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 y Repeat

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

Cadenas

Funciones de cadena

`string.len(s) / #s`	Longitud en bytes
`string.sub(s, i, j)`	Subcadena de i a j
`string.upper(s)`	Convertir a mayúsculas
`string.lower(s)`	Convertir a minúsculas
`string.rep(s, n)`	Repetir cadena n veces
`string.reverse(s)`	Invertir cadena
`string.format(fmt, ...)`	Formato estilo printf
`string.find(s, pat)`	Buscar patrón, retornar índices
`string.gsub(s, pat, rep)`	Sustitución global
`string.gmatch(s, pat)`	Iterador sobre coincidencias del patrón

Caracteres de patrón

`.`	Cualquier carácter
`%a / %A`	Letras / no letras
`%d / %D`	Dígitos / no dígitos
`%w / %W`	Alfanumérico / no alfanumérico
`%s / %S`	Espacio en blanco / no espacio
`%p`	Puntuación
`* + - ?`	Codicioso, codicioso, perezoso, opcional

Metatables

Establecer 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 comunes

`__index`	Buscar claves faltantes (tabla o función)
`__newindex`	Interceptar asignación de nueva clave
`__add / __sub / __mul`	Operadores aritméticos
`__eq / __lt / __le`	Operadores de comparación
`__tostring`	Representación de cadena personalizada
`__len`	Operador # personalizado
`__call`	Llamar tabla como función
`__concat`	Operador .. personalizado

OOP con 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()
```

Corrutinas

Ciclo de vida de corrutina

`coroutine.create(f)`	Crear corrutina desde función
`coroutine.resume(co, ...)`	Iniciar o continuar corrutina
`coroutine.yield(...)`	Suspender ejecución, retornar valores
`coroutine.status(co)`	"running", "suspended", "dead"
`coroutine.wrap(f)`	Crear envoltura de corrutina invocable

Ejemplo de corrutina

```
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

Crear un módulo

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

Usar módulos

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

Bibliotecas estándar

`math`	Funciones matemáticas (sin, random, huge, etc.)
`string`	Manipulación de cadenas y patrones
`table`	Manipulación de tablas (insert, sort, etc.)
`io`	Operaciones de E/S de archivos
`os`	Utilidades del SO (time, clock, execute)
`debug`	Interfaz de depuración (usar con precaución)

Patrones comunes

Idioma ternario

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

Acceso seguro a tabla

```
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
```

Iterar con 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
```