LUA RIFERIMENTO RAPIDO

Tabelle, funzioni, metatable, coroutine, moduli, pattern

Basi

Hello World

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

Variabili e Assegnazione

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

Commenti

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

Operatori

`+ - * / %`	Operatori aritmetici
`//`	Divisione intera (5.3+)
`^`	Esponenziazione
`..`	Concatenazione di stringhe
`#`	Operatore lunghezza
`== ~=`	Uguale / diverso
`and or not`	Operatori logici

Tipi

Tipi di Dato

`nil`	Assenza di valore; falsy
`boolean`	true o false
`number`	Float a doppia precisione (o intero in 5.3+)
`string`	Sequenza di byte immutabile
`table`	Array associativo (unico tipo composto)
`function`	Closure di prima classe
`userdata`	Dati C integrati in Lua
`thread`	Handle di coroutine

Controllo del Tipo

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

Tabelle

Tabelle stile 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
```

Tabelle stile Dizionario

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

Funzioni per Tabelle

`table.insert(t, v)`	Aggiunge un valore all'array
`table.insert(t, i, v)`	Inserisce nella posizione i
`table.remove(t, i)`	Rimuove l'elemento nella posizione i
`table.sort(t [,cmp])`	Ordina l'array in-place
`table.concat(t, sep)`	Unisce gli elementi dell'array in stringa
`table.move(t,a,b,c)`	Sposta gli elementi da a..b alla posizione c

Funzioni

Definizione di Funzione

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

Vararg e Ritorni Multipli

```
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)
```

Closure

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

Controllo del Flusso

Condizionali

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

Cicli

```
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
```

Stringhe

Funzioni per Stringhe

`string.len(s) / #s`	Lunghezza della stringa in byte
`string.sub(s, i, j)`	Sottostringa da i a j
`string.upper(s)`	Converti in maiuscolo
`string.lower(s)`	Converti in minuscolo
`string.rep(s, n)`	Ripeti la stringa n volte
`string.reverse(s)`	Inverti la stringa
`string.format(fmt, ...)`	Formattazione stile printf
`string.find(s, pat)`	Trova pattern, restituisce indici
`string.gsub(s, pat, rep)`	Sostituzione globale
`string.gmatch(s, pat)`	Iteratore sulle corrispondenze del pattern

Caratteri Pattern

`.`	Qualsiasi carattere
`%a / %A`	Lettere / non-lettere
`%d / %D`	Cifre / non-cifre
`%w / %W`	Alfanumerico / non-alfanumerico
`%s / %S`	Spazio bianco / non-spazio
`%p`	Punteggiatura
`* + - ?`	Greedy, greedy, lazy, opzionale

Metatable

Impostazione delle Metatable

```
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
```

Metamethod Comuni

`__index`	Ricerca chiavi mancanti (tabella o funzione)
`__newindex`	Intercetta l'assegnazione di nuove chiavi
`__add / __sub / __mul`	Operatori aritmetici
`__eq / __lt / __le`	Operatori di confronto
`__tostring`	Rappresentazione testuale personalizzata
`__len`	Operatore # personalizzato
`__call`	Chiama la tabella come funzione
`__concat`	Operatore .. personalizzato

OOP con Metatable

```
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()
```

Coroutine

Ciclo di Vita delle Coroutine

`coroutine.create(f)`	Crea coroutine dalla funzione
`coroutine.resume(co, ...)`	Avvia o riprende la coroutine
`coroutine.yield(...)`	Sospende l'esecuzione, restituisce valori
`coroutine.status(co)`	"running", "suspended", "dead"
`coroutine.wrap(f)`	Crea un wrapper chiamabile per coroutine

Esempio di 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
```

Moduli

Creazione di un Modulo

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

Uso dei Moduli

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

Librerie Standard

`math`	Funzioni matematiche (sin, random, huge, ecc.)
`string`	Manipolazione di stringhe e pattern
`table`	Manipolazione di tabelle (insert, sort, ecc.)
`io`	Operazioni di I/O su file
`os`	Funzioni OS (time, clock, execute)
`debug`	Interfaccia di debug (usare con parsimonia)

Pattern Comuni

Idioma Ternario

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

Accesso Sicuro alle Tabelle

```
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
```

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
```