

# 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

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

## Tipi

### Tipi di Dato

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

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

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

## Caratteri Pattern

<b>.</b>	Qualsiasi carattere
<b>%a / %A</b>	Lettere / non-lettere
<b>%d / %D</b>	Cifre / non-cifre
<b>%w / %W</b>	Alfanumerico / non-alfanumerico
<b>%s / %S</b>	Spazio bianco / non-spazio
<b>%p</b>	Punteggiatura
<b>* + - ?</b>	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

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

<b>coroutine.create(f)</b>	Crea coroutine dalla funzione
<b>coroutine.resume(co, ...)</b>	Avvia o riprende la coroutine
<b>coroutine.yield(...)</b>	Sospende l'esecuzione, restituisce valori
<b>coroutine.status(co)</b>	"running", "suspended", "dead"
<b>coroutine.wrap(f)</b>	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
```

# Lua Riferimento Rapido

---

## Uso dei Moduli

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

## Librerie Standard

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