RÉFÉRENCE RAPIDE LUA

Tables, fonctions, métatables, coroutines, modules, motifs

Bases

Hello World

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

Variables et affectation

```lua
local name = "Lua"      -- variable locale
x = 10                  -- globale (à éviter)
local a, b = 1, 2       -- affectation multiple
a, b = b, a             -- échanger des valeurs
```

Commentaires

```lua
-- commentaire sur une ligne
--[[ commentaire
     multi-lignes ]]
```

Opérateurs

`+ - * / %`	Opérateurs arithmétiques
`//`	Division entière par défaut (5.3+)
`^`	Exponentiation
`..`	Concaténation de chaînes
`#`	Opérateur de longueur
`== ~=`	Égal / différent
`and or not`	Opérateurs logiques

Types

Types de données

`nil`	Absence de valeur ; falsy
`boolean`	true ou false
`number`	Flottant double précision (ou entier en 5.3+)
`string`	Séquence d'octets immuable
`table`	Tableau associatif (seul type composé)
`function`	Fermeture de première classe
`userdata`	Données C encapsulées pour Lua
`thread`	Handle de coroutine

Vérification de type

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

Tables

Tables style tableau

```lua
local fruits = {"apple", "banana", "cherry"}
print(fruits[1])          -- "apple" (indexé à 1)
table.insert(fruits, "date")
table.remove(fruits, 2)   -- supprimer "banana"
print(#fruits)            -- longueur
```

Tables style dictionnaire

```lua
local user = {name = "Alice", age = 30}
user.email = "a@b.com"    -- ajouter un champ
user["name"] = "Bob"      -- accès par crochets
user.age = nil            -- supprimer un champ
```

Fonctions de table

`table.insert(t, v)`	Ajouter une valeur au tableau
`table.insert(t, i, v)`	Insérer à la position i
`table.remove(t, i)`	Supprimer l'élément à la position i
`table.sort(t [,cmp])`	Trier le tableau en place
`table.concat(t, sep)`	Joindre les éléments du tableau en chaîne
`table.move(t,a,b,c)`	Déplacer les éléments de a..b vers la position c

Fonctions

Définition de fonction

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

Variadique et retours multiples

```lua
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)
```

Fermetures

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

Flux de contrôle

Conditionnelles

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

Boucles

```lua
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 et Repeat

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

Chaînes de caractères

Fonctions de chaînes

`string.len(s) / #s`	Longueur de chaîne en octets
`string.sub(s, i, j)`	Sous-chaîne de i à j
`string.upper(s)`	Convertir en majuscules
`string.lower(s)`	Convertir en minuscules
`string.rep(s, n)`	Répéter la chaîne n fois
`string.reverse(s)`	Inverser la chaîne
`string.format(fmt, ...)`	Formatage style printf
`string.find(s, pat)`	Trouver le motif, retourner les indices
`string.gsub(s, pat, rep)`	Substitution globale
`string.gmatch(s, pat)`	Itérateur sur les correspondances du motif

Caractères de motif

`.`	N'importe quel caractère
`%a / %A`	Lettres / non-lettres
`%d / %D`	Chiffres / non-chiffres
`%w / %W`	Alphanumérique / non-alphanumérique
`%s / %S`	Espace blanc / non-espace blanc
`%p`	Ponctuation
`* + - ?`	Gourmand, gourmand, paresseux, optionnel

Métatables

Définir des métatables

```lua
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
```

Métaméthodes courantes

`__index`	Chercher les clés manquantes (table ou fonction)
`__newindex`	Intercepter l'affectation d'une nouvelle clé
`__add / __sub / __mul`	Opérateurs arithmétiques
`__eq / __lt / __le`	Opérateurs de comparaison
`__tostring`	Représentation en chaîne personnalisée
`__len`	Opérateur # personnalisé
`__call`	Appeler la table comme une fonction
`__concat`	Opérateur .. personnalisé

POO avec les métatables

```lua
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

Cycle de vie d'une coroutine

`coroutine.create(f)`	Créer une coroutine depuis une fonction
`coroutine.resume(co, ...)`	Démarrer ou continuer la coroutine
`coroutine.yield(...)`	Suspendre l'exécution, retourner des valeurs
`coroutine.status(co)`	"running", "suspended", "dead"
`coroutine.wrap(f)`	Créer un wrapper de coroutine appelable

Exemple de coroutine

```lua
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
```

Modules

Créer un module

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

Utiliser des modules

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

Bibliothèques standard

`math`	Fonctions mathématiques (sin, random, huge, etc.)
`string`	Manipulation de chaînes et motifs
`table`	Manipulation de tables (insert, sort, etc.)
`io`	Opérations d'E/S sur fichiers
`os`	Facilités OS (time, clock, execute)
`debug`	Interface de débogage (à utiliser avec parcimonie)

Modèles courants

Idiome ternaire

```lua
-- Lua n'a pas d'opérateur ternaire ; utiliser and/or
local val = condition and "yes" or "no"
-- Attention : échoue si "yes" est false/nil
```

Accès sécurisé aux tables

```lua
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")  -- accès imbriqué sécurisé
```

Itérer avec ipairs vs pairs

```lua
-- ipairs : partie tableau, s'arrête au premier nil
for i, v in ipairs(arr) do print(i, v) end
-- pairs : toutes les clés (non ordonnées)
for k, v in pairs(tbl) do print(k, v) end
```