

Lua クイックリファレンス

テーブル、関数、メタテーブル、コルーチン、モジュール、パターン

基本

Hello World

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

変数と代入

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

コメント

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

演算子

+	-	*	/	%	算術演算子
//	床除算 (5.3+)				
^	べき乗				
..	文字列連結				
#	長さ演算子				
==	~=	等価 / 非等価			
and	or	not	論理演算子		

型

データ型

nil	値の不在; falsy
boolean	true または false
number	倍精度浮動小数点 (5.3+ では整数も)
string	イミュータブルなバイト列
table	連想配列 (唯一の複合型)
function	第一級クロージャ
userdata	Lua のために C データをラップ
thread	コルーチンハンドル

型チェック

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

テーブル

配列スタイルのテーブル

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

辞書スタイルのテーブル

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

テーブル関数

table.insert(t, v)	配列に値を追加
table.insert(t, i, v)	位置 i に挿入
table.remove(t, i)	位置 i の要素を削除
table.sort(t [,cmp])	配列をその場でソート
table.concat(t, sep)	配列要素を文字列に結合
table.move(t,a,b,c)	a..b の要素を位置 c に移動

関数

関数定義

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

可変長引数と複数の戻り値

```
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)
```

クロージャ

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

制御フロー

条件分岐

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

ループ

```
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 と repeat

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

文字列

文字列関数

string.len(s) / #s	バイト単位の文字列長
string.sub(s, i, j)	i から j までの部分文字列
string.upper(s)	大文字に変換
string.lower(s)	小文字に変換
string.rep(s, n)	文字列を n 回繰り返す
string.reverse(s)	文字列を逆順にする
string.format(fmt, ...)	printf スタイルの書式化
string.find(s, pat)	パターンを検索し、インデックスを返す
string.gsub(s, pat, rep)	グローバル置換
string.gmatch(s, pat)	パターンマッチのイテレーター

パターン文字

.	任意の文字
%a / %A	文字 / 非文字
%d / %D	数字 / 非数字
%w / %W	英数字 / 非英数字
%s / %S	空白 / 非空白
%p	句読点
* + - ?	貪欲、貪欲、怠惰、オプション

メタテーブル

メタテーブルの設定

```
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
```

よく使うメタメソッド

__index	存在しないキーの検索 (テーブルまたは関数)
__newindex	新しいキーへの代入を横取り
__add / __sub / __mul	算術演算子
__eq / __lt / __le	比較演算子
__tostring	カスタム文字列表現
__len	カスタム # 演算子
__call	テーブルを関数として呼び出す
__concat	カスタム .. 演算子

メタテーブルを使った OOP

```
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()
```

コルーチン

コルーチンのライフサイクル

coroutine.create(f)	関数からコルーチンを作成
coroutine.resume(co, ...)	コルーチンを開始または再開
coroutine.yield(...)	実行を一時停止し値を返す
coroutine.status(co)	"running"、"suspended"、"dead"
coroutine.wrap(f)	呼び出し可能なコルーチンラッパーを作成

Lua クイックリファレンス

コルーチンの例

```
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
```

モジュール

モジュールの作成

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

モジュールの使用

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

標準ライブラリ

math	数学関数 (sin、random、huge など)
string	文字列操作とパターン
table	テーブル操作 (insert、sort など)
io	ファイル I/O 操作
os	OS 機能 (time、clock、execute)
debug	デバッグインターフェース (慎重に使用)

よく使うパターン

三項演算子のイディオム

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

安全なテーブルアクセス

```
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 と pairs の使い分け

```
-- 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
```