Lua

Cheatsheet

P J

1 Lua Cheatsheet

1.1 Comments

```
-- comment
--[[ multiline
     comment
]]
--[==[ multiline comment
       in different level
]==]
```

1.2 Invoking functions

```
print()
print("Hi")
```

You can omit parentheses **if** the argument is one string or table literal:

```
print "Hello World"      -- print("Hello World")
dofile 'a.lua'           -- dofile ('a.lua')
print [[a multi-line
 message]]               --[==[ print([[a multi-line
                                    message]]) ]==]
f{x=10, y=20}            -- f({x=10, y=20})
type{}                   -- type({})
```

1.3 Tables / arrays

Remember, arrays are also tables:

```lua
t = {}
t = { a = 1, b = 2 }
t.a = function() ... end

t = { ["hello"] = 200 }
t.hello

array = { "a", "b", "c", "d" }
print(array[2])       -- "b" (one-indexed)
print(#array)         -- 4 (length)
```

1.4 Loops

```lua
while condition do
end

for i = 1,5 do
end

for i = start,finish,delta do
end

for k,v in pairs(tab) do
end

repeat
until condition

-- Breaking out:
while x do
  if condition then break end
end
```

1.5 Conditionals

No `switch .. case` here :.)

```lua
if condition then
  print("yes")
elseif condition then
  print("maybe")
else
```

```
  print("no")
end
```

1.6 Variables

Do `local` most of the time:

```
local x = 2
two, four = 2, 4
```

1.7 Functions

```
function myFunction()
  return 1
end

function myFunctionWithArgs(a, b)
  -- ...
end

myFunction()

anonymousFunctions(function()
  -- ...
end)

-- Not exported in the module
local function myPrivateFunction()
end

-- Splats
function doAction(action, ...)
  print("Doing '"..action.."' to", ...)
end

doAction('write', "Shirley", "Abed")  -- Doing 'write' to, Shirley, Abed)
```

1.8 Lookups

```
mytable = { x = 2, y = function() .. end }

```

```lua
-- The same:
mytable.x
mytable['x']

-- Syntactic sugar, these are equivalent:
mytable.y(mytable)
mytable:y()

mytable.y(mytable, a, b)
mytable:y(a, b)

function X:y(z) .. end
function X.y(self, z) .. end
```

1.9 Metatables

```lua
mt = {}

-- A metatable is simply a table with functions in it.
mt.__tostring = function() return "lol" end
mt.__add      = function(b) ... end       -- a + b
mt.__mul      = function(b) ... end       -- a * b
mt.__index    = function(k) ... end       -- Lookups (a[k] or a.k)
mt.__newindex = function(k, v) ... end    -- Setters (a[k] = v)

-- Metatables allow you to override behavior of another table.
mytable = {}
setmetatable(mytable, mt)

print(myobject)
```

1.10 Classes

```lua
Account = {}
function Account:new(balance)
  local t = setmetatable({}, { __index = Account })
  -- Your constructor stuff
  t.balance = (balance or 0)
  return t
end

```

```lua
function Account:withdraw(amount)
  print("Withdrawing "..amount.."...")
  self.balance = self.balance - amount
  self:report()
end

function Account:report()
  print("Your current balance is: "..self.balance)
end

a = Account:new(9000)
a:withdraw(200)    -- method call
```

1.11 Constants

```lua
nil
false
true
```

1.12 Operators (and their metatable names)

1.12.1 Arithmetic

Operation	Syntax	Description	Example
Arithmetic negation	`-a`	Changes the sign of a and returns the value	`-3.14159`
Addition	`a + b`	Returns the sum of `a` and `b`	`5.2 + 3.6`
Subtraction	`a - b`	Subtracts b from a and returns the result	`5.2 + 3.6`
Multiplication	`a * b`	Returns the product of a and b	`3.2 * 1.5`
Exponentiation	`a ^ b`	Returns a to the power b, or the exponentiation of a by b	`5 ^ 2`
Division	`a / b`	Divides a by b and returns the result	`6.4 / 2`
Modulus operation	`a % b`	Returns the remainder of the division of a by b	`5 % 3`

1.12.2 Boolean

Operation	Syntax	Description
Boolean negation	`not a`	If a is false or nil, returns true. Otherwise, returns false.

Operation	Syntax	Description
Logical conjunction	`a and b`	Returns the first argument if it is false or nil. Otherwise, returns the second argument.
Logical disjunction	`a or b`	Returns the first argument if it is neither false nor nil. Otherwise, returns the second argument.

1.12.3 Metatable

```
-- Relational (binary)
-- __eq  __lt  __gt  __le  __ge
   ==    <     >     <=    >=
~=   -- Not equal, just like !=

-- Arithmetic (binary)
-- __add  __sub  __muv  __div  __mod  __pow
   +      -      *      /      %      ^

-- Arithmetic (unary)
-- __unm (unary minus)
   -

-- Logic (and/or)
nil and false  -- nil
false and nil  -- false
0 and 20       -- 20
10 and 20      -- 20

-- Length
-- __len(array)
#array

-- Indexing
-- __index(table, key)
t[key]
t.key

-- __newindex(table, key, value)
t[key]=value

-- String concat
```

```lua
-- __concat(left, right)
"hello, "..name

-- Call
-- __call(func, ...)
```

1.13 API: Global functions

```lua
dofile("hello.lua")
loadfile("hello.lua")

assert(x)    -- x or (raise an error)
assert(x, "failed")

type(var)   --[[  "nil" | "number"   | "string" | "boolean"
                "table" | "function" | "thread" | "userdata" ]]

-- Does /not/ invoke meta methods (__index and __newindex)
rawset(t, index, value)    -- Like t[index] = value
rawget(t, index)           -- Like t[index]

_G  -- Global context
setfenv(1, {})  -- 1: current function, 2: caller, and so on -- {}: the new _G

pairs(t)     -- iterable list of {key, value}
ipairs(t)    -- iterable list of {index, value}

tonumber("34")
tonumber("8f", 16)
```

1.14 API: Strings

```lua
'string'..'concatenation'

s = "Hello"
s:upper()
s:lower()
s:len()    -- Just like #s

s:find()
s:gfind()
```

```
s:match()
s:gmatch()

s:sub()
s:gsub()

s:rep()
s:char()
s:dump()
s:reverse()
s:byte()
s:format()```

<!--}}}-->
## API: Tables <!--{{{-->

```lua
table.foreach(t, function(row) ... end)
table.setn
table.insert(t, 21)          -- append (--> t[#t+1] = 21)
table.insert(t, 4, 99)
table.getn
table.concat
table.sort
table.remove(t, 4)
```

1.15 API: Math

```
math.abs     math.acos    math.asin       math.atan    math.atan2
math.ceil    math.cos     math.cosh       math.deg     math.exp
math.floor   math.fmod    math.frexp      math.ldexp   math.log
math.log10   math.max     math.min        math.modf    math.pow
math.rad     math.random  math.randomseed math.sin     math.sinh
math.sqrt    math.tan     math.tanh

math.sqrt(144)
math
```
```

1.16 API: Misc

```
io.output(io.open("file.txt", "w"))
io.write(x)
io.close()

for line in io.lines("file.txt")

file = assert(io.open("file.txt", "r"))
file:read()
file:lines()
file:close()
```