CHEATSHEET FOR

Lua

Comments

```
-- comment
--[[ Multiline
     comment ]]
```

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    <-->     print([[a multi-line
 message]]                        message]])
f{x=10, y=20}           <-->     f({x=10, y=20})
type{}                  <-->     type({})
```

Tables / arrays

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

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

-- Remember, arrays are also tables
array = { "a", "b", "c", "d" }
```

```
print(array[2])       -- "b" (one-indexed)
print(#array)         -- 4 (length)
```

Loops

```
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
```

Conditionals

```
if condition then
  print("yes")
elsif condition then
  print("maybe")
else
  print("no")
end
```

Variables

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

Functions

```lua
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", ...)
  --> print("Doing 'write' to", "Shirley", "Abed")
end

doAction('write', "Shirley", "Abed")
```

Lookups

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

-- 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
```

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)
```

Classes

```
Account = {}

function Account:new(balance)
  local t = setmetatable({}, { __index = Account })

  -- Your constructor stuff
  t.balance = (balance or 0)
  return t
end

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
```

Constants

```
nil
false
true
```

Operators (and their metatable names)

```
-- 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 10     --> 10
false and nil  --> false
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
-- __concat(left, right)
"hello, "..name

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

API: Global functions (ref)

```
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" | "userdat

-- 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)
```

API: Strings

```
'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

```
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)
```

API: Math (ref)

```
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
```

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()
```

Reference

http://www.lua.org/pil/13.html http://lua-users.org/wiki/ObjectOrientedProgramming