Lua
Quick Guide
&
Pitfalls

iantian@cloudia.cn

Outline

- 运行方式
- 语法
- 控制语句
- 函数
- Table
- MetaTable & MetaMethod
- “面向对象”
- 模块
- 常见坑
- Lua风格

Quick Guide

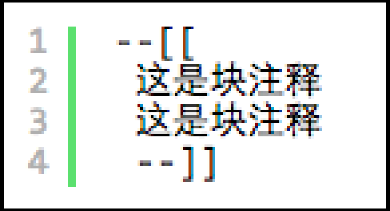
运行

- Lua命令行
- Lua文件运行

```
> lua file.lua
> Chmod +x file.lua
```

语法

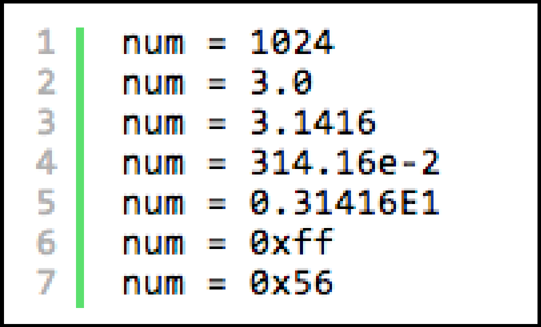
- 注释
  - 行内注释
  - 块注释



```
--[[
 这是块注释
 这是块注释
 --]]
```

语法

- 变量
  - 数字只有double型，64位
  - 数字

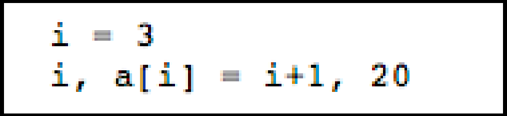


```
num = 1024
num = 3.0
num = 3.1416
num = 314.16e-2
num = 0.31416E1
num = 0xff
num = 0x56
```

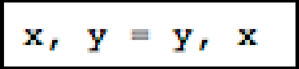
  - 没声明过的变量是nil
  - 没有local全是全局变量

赋值表达式

- 多赋值



```
i = 3
i, a[i] = i+1, 20
```



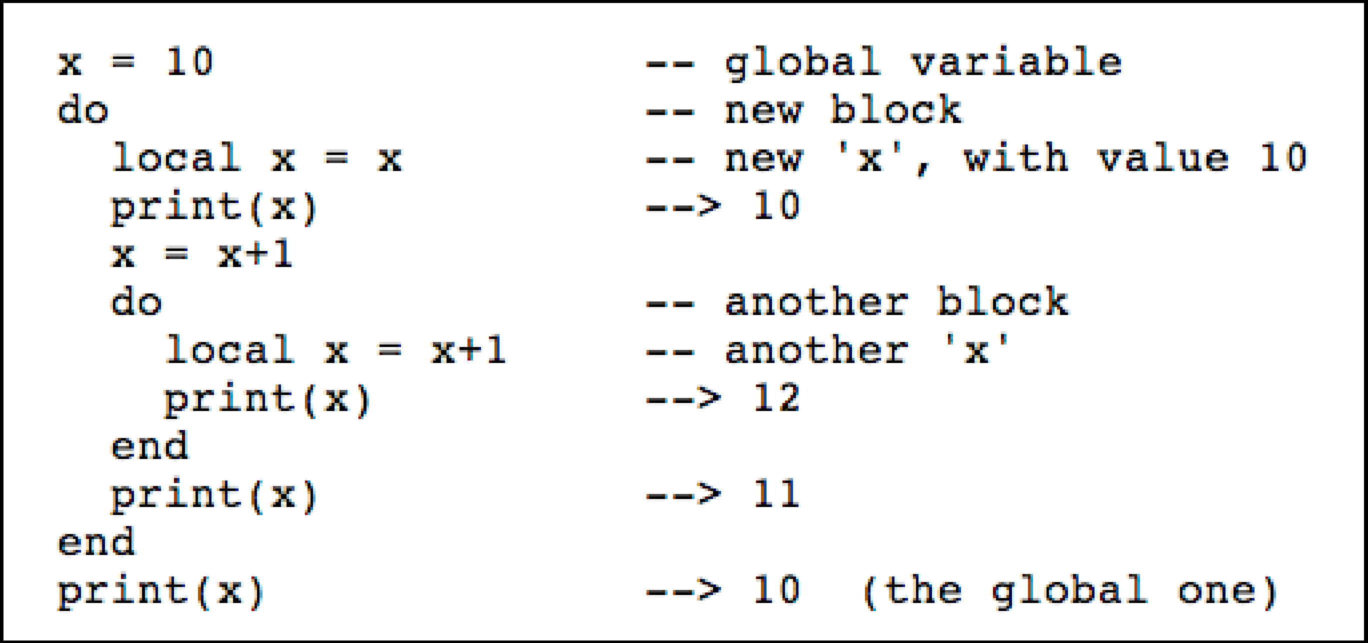
```
x, y = y, x
```

控制语句

- while do end
- for do end
- repeat until
- do end
- elseif
- no i++
- no +=
- ~=
- ..
- and, or, not

作用域

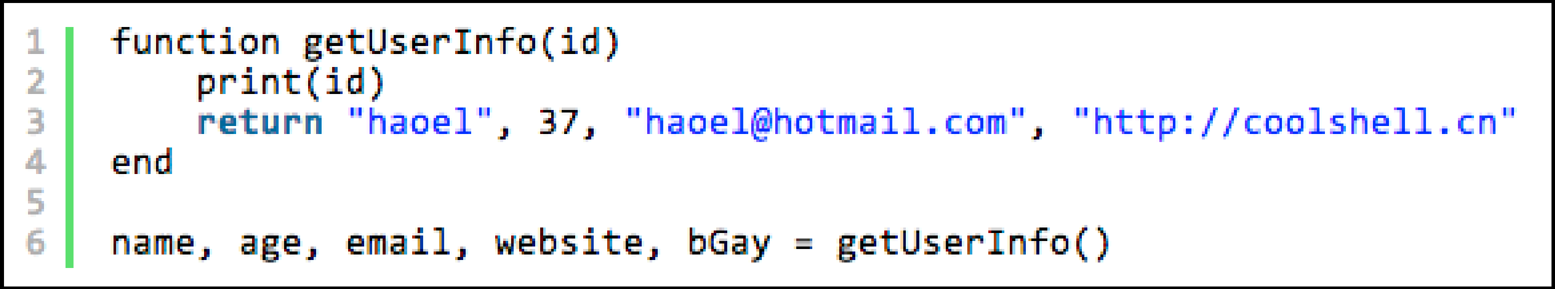
- 词法作用域（lexical scope）
  - end



```
x = 10                     -- global variable
do                         -- new block
  local x = x              -- new 'x', with value 10
  print(x)                 --> 10
  x = x+1
  do                       -- another block
    local x = x+1          -- another 'x'
    print(x)               --> 12
  end
  print(x)                 --> 11
end
print(x)                   --> 10  (the global one)
```

函数

- 多返回值

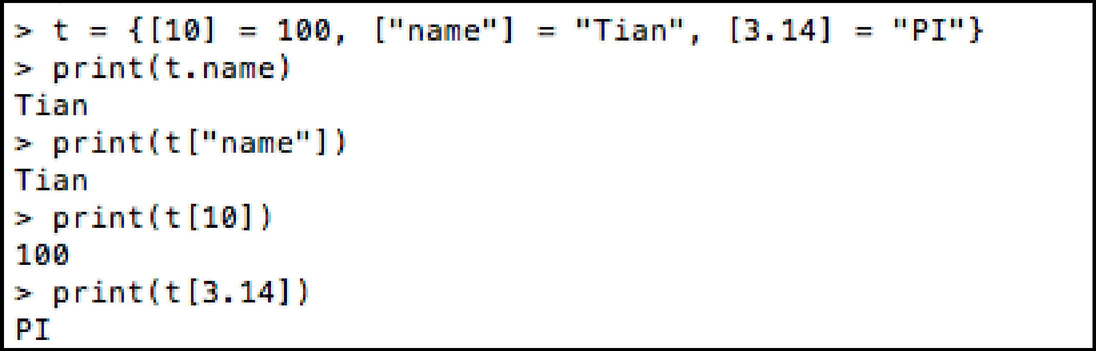


```lua
function getUserInfo(id)
    print(id)
    return "haoel", 37, "haoel@hotmail.com", "http://coolshell.cn"
end

name, age, email, website, bGay = getUserInfo()
```

- 闭包

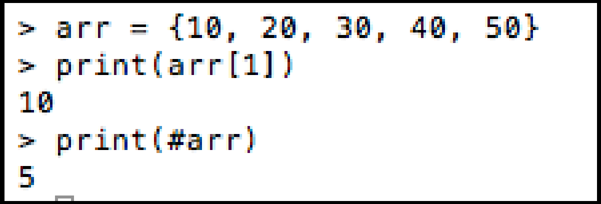
Table

- 字典



```
> t = {[10] = 100, ["name"] = "Tian", [3.14] = "PI"}
> print(t.name)
Tian
> print(t["name"])
Tian
> print(t[10])
100
> print(t[3.14])
PI
```

- 数组
  - 下标从1开始



```
> arr = {10, 20, 30, 40, 50}
> print(arr[1])
10
> print(#arr)
5
```

Table

- 遍历
  - pairs
  - ipairs
- 全局Table
  - _G

MetaTable

- Every value in Lua can have a *metatable*
- We call
  - Keys in a metatable: *events*
  - Values in a metatable: *metamethod*

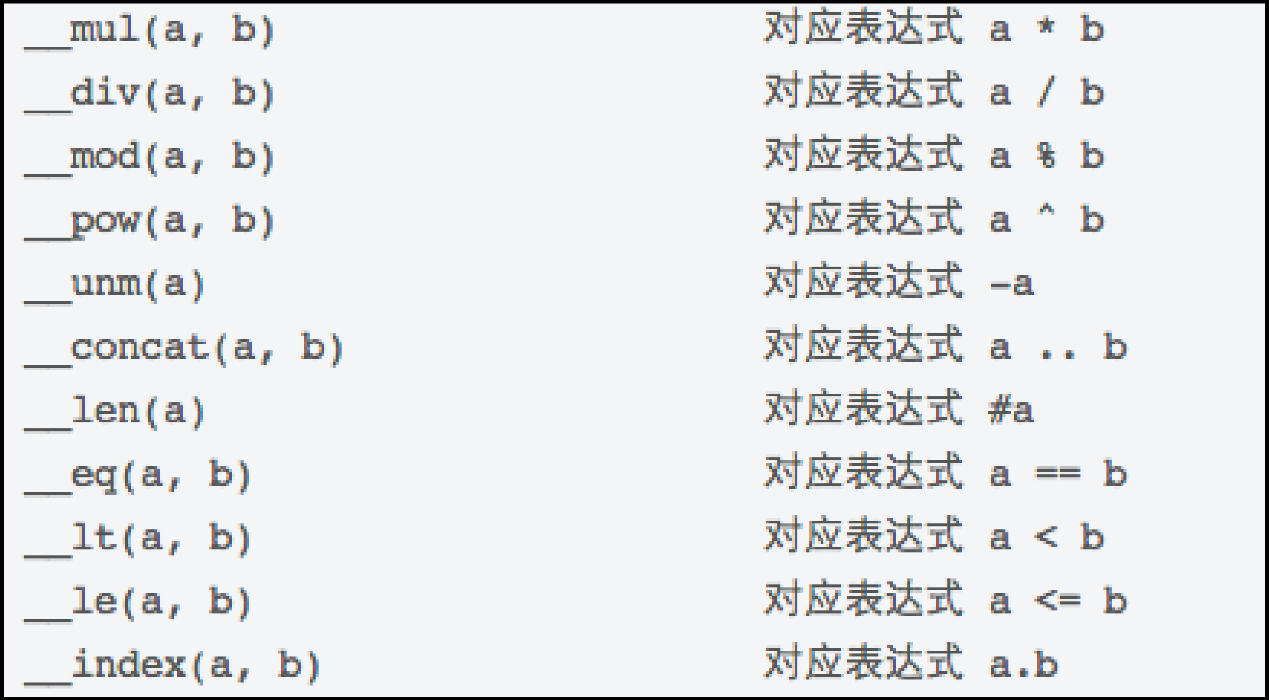
```lua
fraction_a = {numerator=2, denominator=3}
fraction_b = {numerator=4, denominator=7}
```

```lua
fraction_op={}
function fraction_op.__add(f1, f2)
    ret = {}
    ret.numerator = f1.numerator * f2.denominator + f2.numerator * f1.denomir
    ret.denominator = f1.denominator * f2.denominator
    return ret
end
```

```lua
setmetatable(fraction_a, fraction_op)
setmetatable(fraction_b, fraction_op)
```

```lua
fraction_s = fraction_a + fraction_b
```

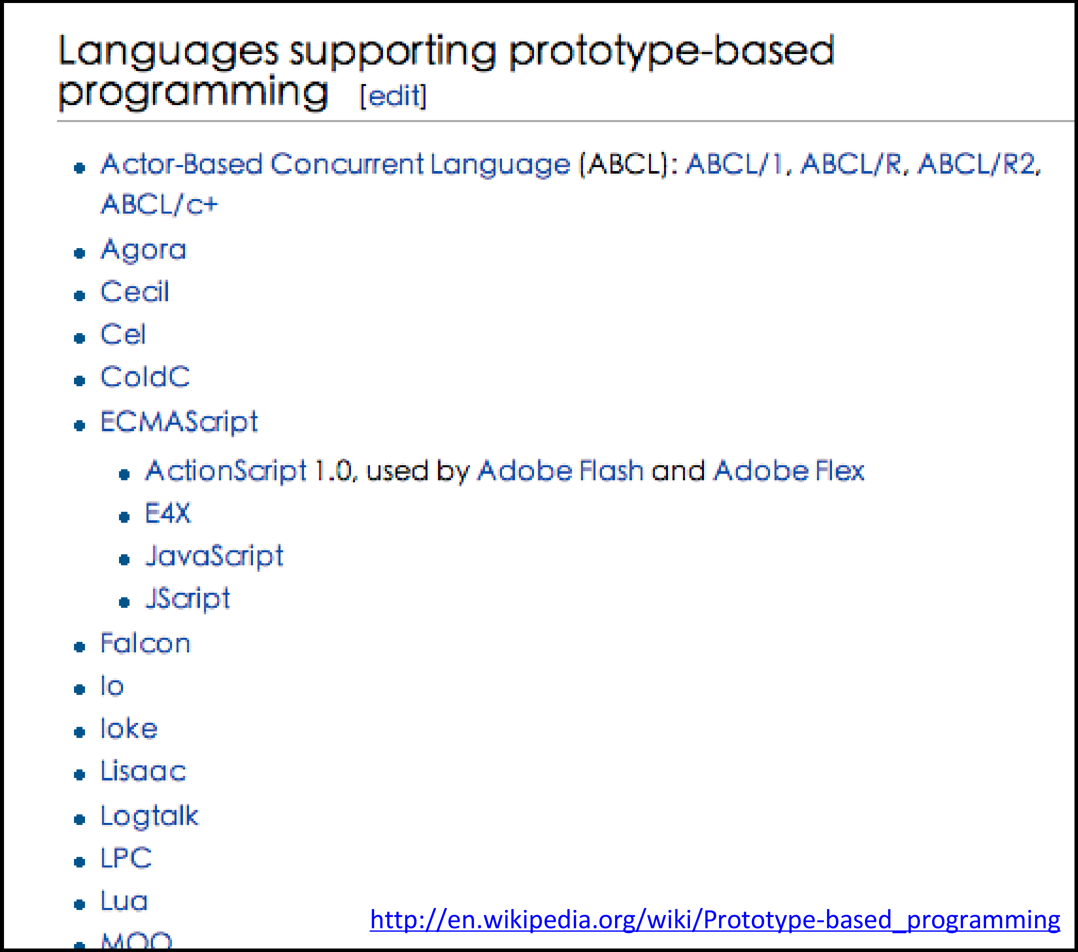
MetaTable



`__mul(a, b)`	对应表达式 a * b
`__div(a, b)`	对应表达式 a / b
`__mod(a, b)`	对应表达式 a % b
`__pow(a, b)`	对应表达式 a ^ b
`__unm(a)`	对应表达式 -a
`__concat(a, b)`	对应表达式 a .. b
`__len(a)`	对应表达式 #a
`__eq(a, b)`	对应表达式 a == b
`__lt(a, b)`	对应表达式 a < b
`__le(a, b)`	对应表达式 a <= b
`__index(a, b)`	对应表达式 a.b

“面向对象“

- 原型式编程语言

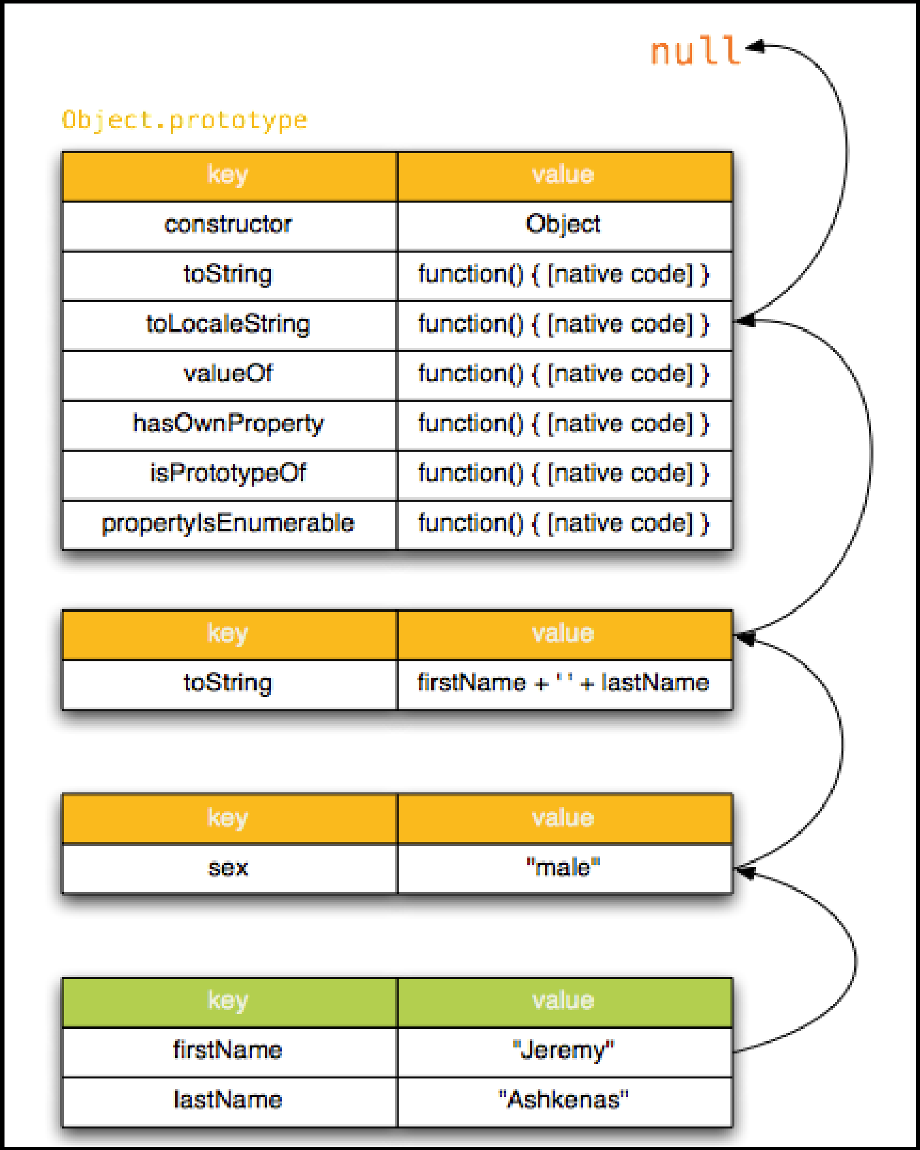


Languages supporting prototype-based programming [edit]

- Actor-Based Concurrent Language (ABCL): ABCL/1, ABCL/R, ABCL/R2, ABCL/c+
- Agora
- Cecil
- Cel
- ColdC
- ECMAScript
  - ActionScript 1.0, used by Adobe Flash and Adobe Flex
  - E4X
  - JavaScript
  - JScript
- Falcon
- Io
- Ioke
- Lisaac
- Logtalk
- LPC
- Lua
- MOO

http://en.wikipedia.org/wiki/Prototype-based_programming

原型式编程



key	value
constructor	Object
toString	function() { [native code] }
toLocaleString	function() { [native code] }
valueOf	function() { [native code] }
hasOwnProperty	function() { [native code] }
isPrototypeOf	function() { [native code] }
propertyIsEnumerable	function() { [native code] }

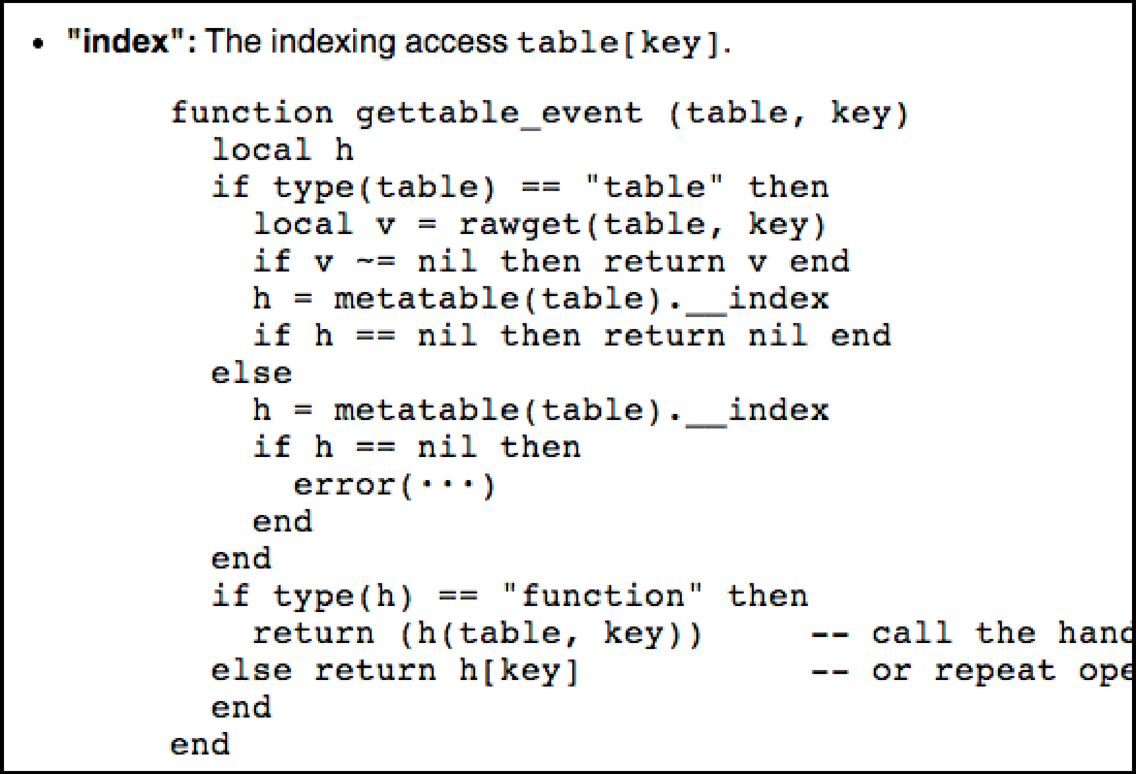
key	value
toString	firstName + ' ' + lastName

key	value
sex	"male"

key	value
firstName	"Jeremy"
lastName	"Ashkenas"

“面向对象“

- Lua对原型式编程的支持方式
  - __index

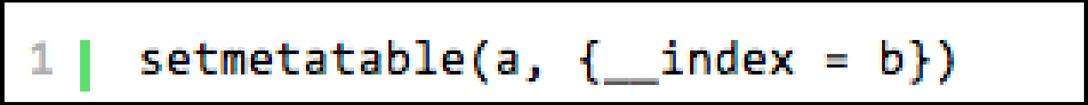


- **"index"**: The indexing access `table[key]`.

```
function gettable_event (table, key)
  local h
  if type(table) == "table" then
    local v = rawget(table, key)
    if v ~= nil then return v end
    h = metatable(table).__index
    if h == nil then return nil end
  else
    h = metatable(table).__index
    if h == nil then
      error(···)
    end
  end
  if type(h) == "function" then
    return (h(table, key))     -- call the hanc
  else return h[key]           -- or repeat ope
  end
end
```

“面向对象“

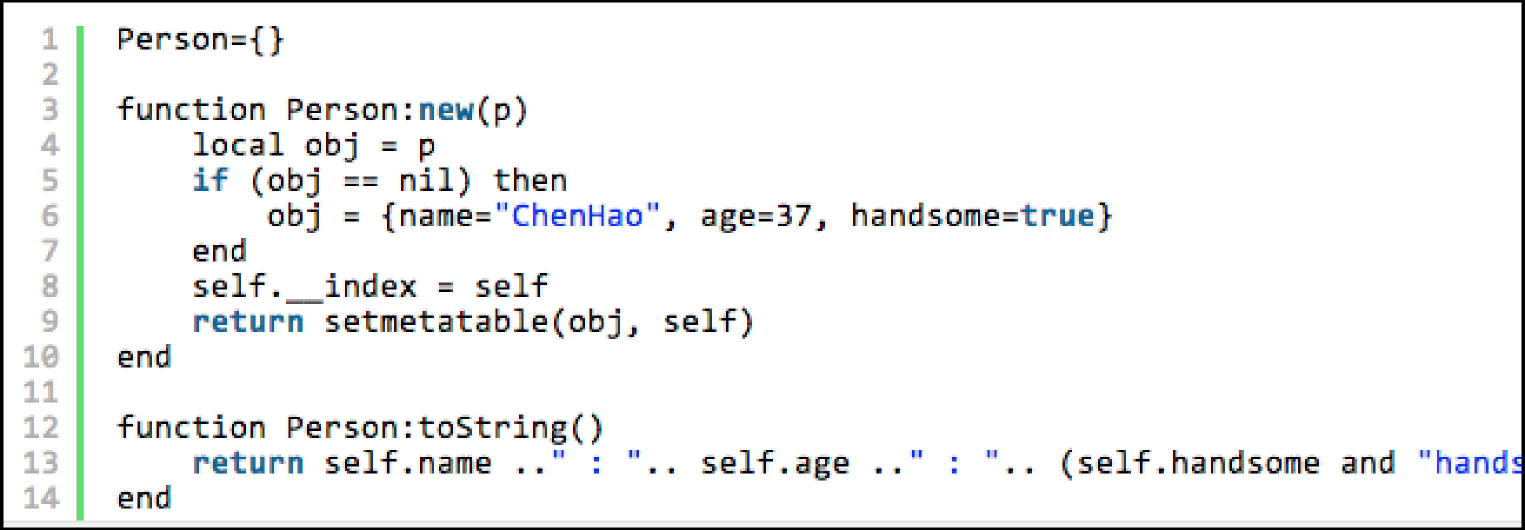
- Lua对原型式编程的支持方式



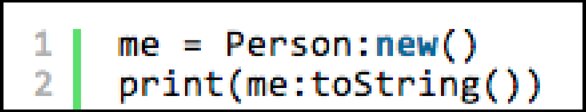
```lua
setmetatable(a, {__index = b})
```

“面向对象“

- 创建“对象”



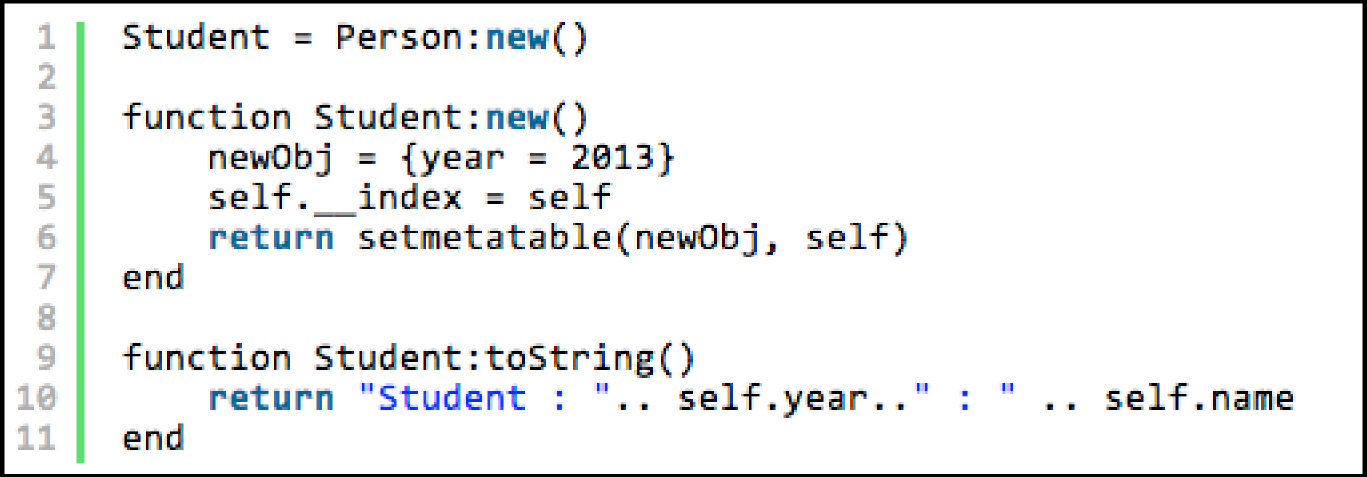
```lua
Person={}

function Person:new(p)
    local obj = p
    if (obj == nil) then
        obj = {name="ChenHao", age=37, handsome=true}
    end
    self.__index = self
    return setmetatable(obj, self)
end

function Person:toString()
    return self.name .." : ".. self.age .." : ".. (self.handsome and "hands
end
```



```lua
me = Person:new()
print(me:toString())
```

“面向对象“

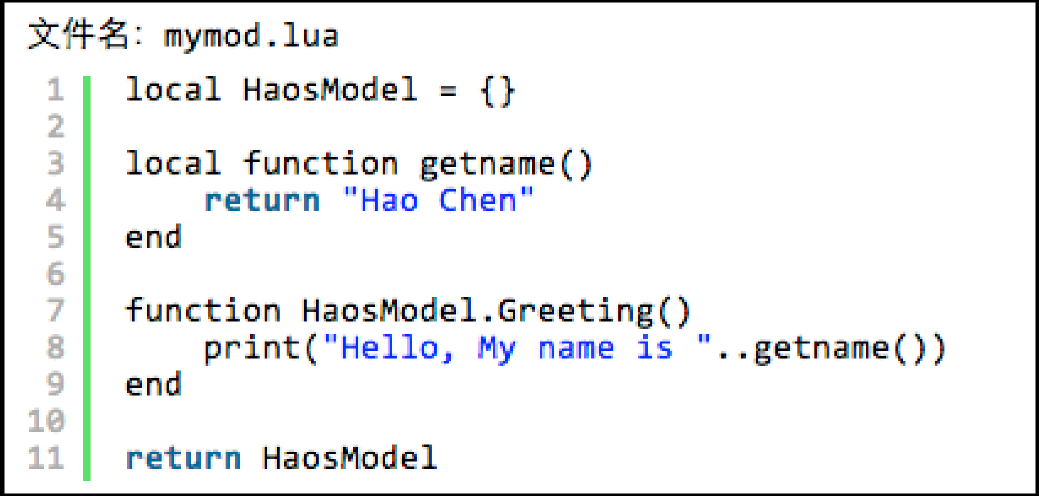
- “继承”



```lua
Student = Person:new()

function Student:new()
    newObj = {year = 2013}
    self.__index = self
    return setmetatable(newObj, self)
end

function Student:toString()
    return "Student : ".. self.year.." : " .. self.name
end
```

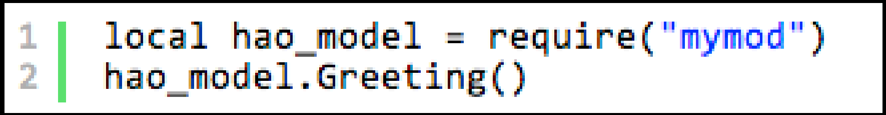
模块

- require(“module_name”)
  – 载入并执行
- 定义模块



文件名: mymod.lua

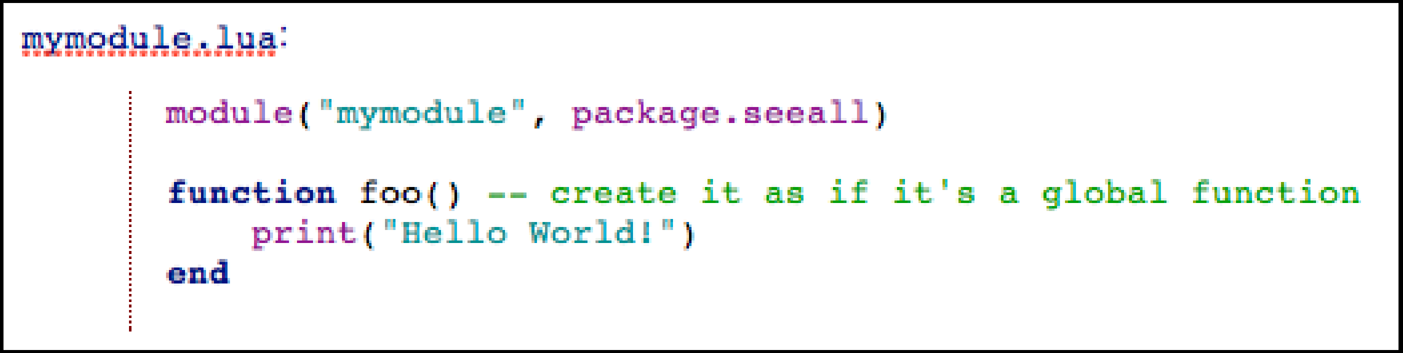
```lua
local HaosModel = {}

local function getname()
    return "Hao Chen"
end

function HaosModel.Greeting()
    print("Hello, My name is "..getname())
end

return HaosModel
```



```lua
local hao_model = require("mymod")
hao_model.Greeting()
```

模块

- 定义模块
  - 官方不建议的方式



mymodule.lua:

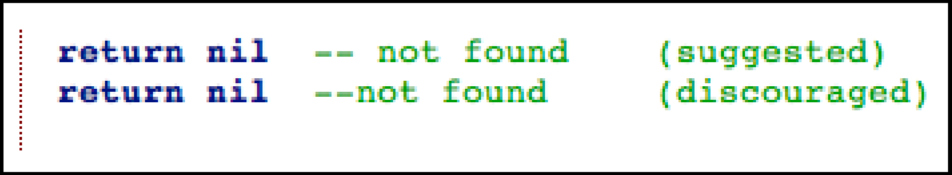
```
module("mymodule", package.seeall)

function foo() -- create it as if it's a global function
    print("Hello World!")
end
```

Lua编程习惯

I will follow the rules
I will follow the rules
I will follow the rules
I will follow the rules
I will fol

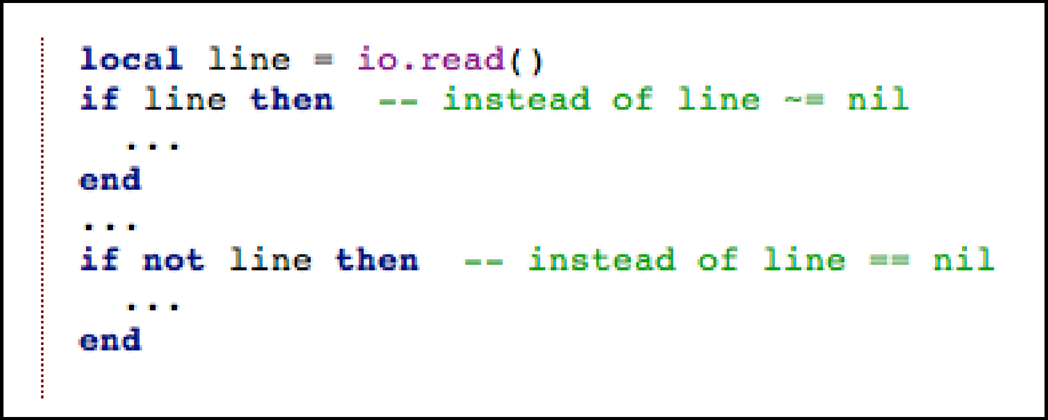
Lua编程习惯

- 注释



```
return nil   -- not found     (suggested)
return nil   --not found      (discouraged)
```

- nil判断



```
local line = io.read()
if line then   -- instead of line ~= nil
  ...
end
...
if not line then   -- instead of line == nil
  ...
end
```

Lua编程习惯

- or和and妙用

```lua
local function test(x)
  x = x or "idunno"
    -- rather than if x == false or x == nil then x = "idunno" end
  print(x == "yes" and "YES!" or x)
    -- rather than if x == "yes" then print("YES!") else print(x) end
end
```

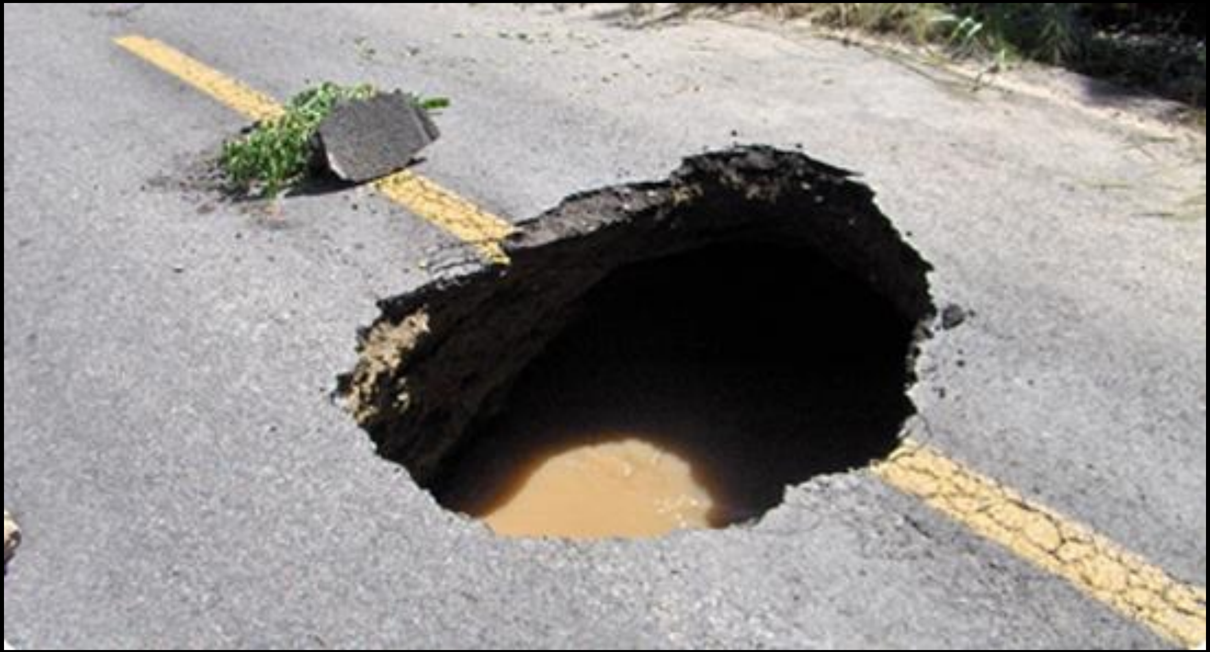
- 复制小型table

```lua
u = {unpack(t)}
```

- 判断table是否空

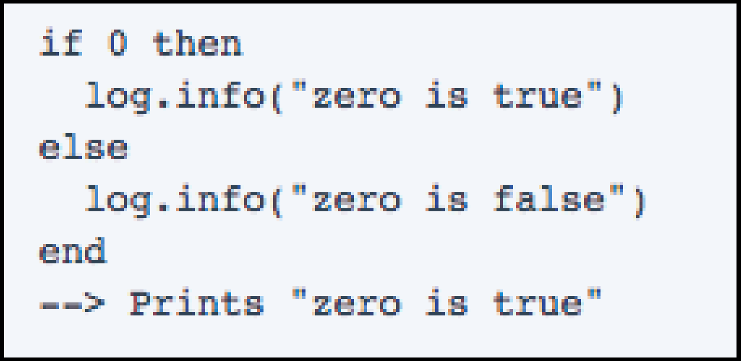
```lua
if next(t) == nil then ...
```

坑



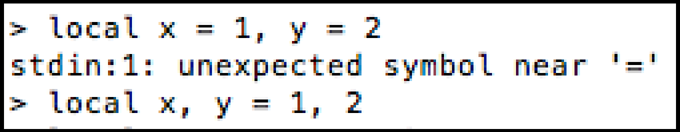
坑

- if
  - 除了nil和false都是true



```
if 0 then
  log.info("zero is true")
else
  log.info("zero is false")
end
--> Prints "zero is true"
```

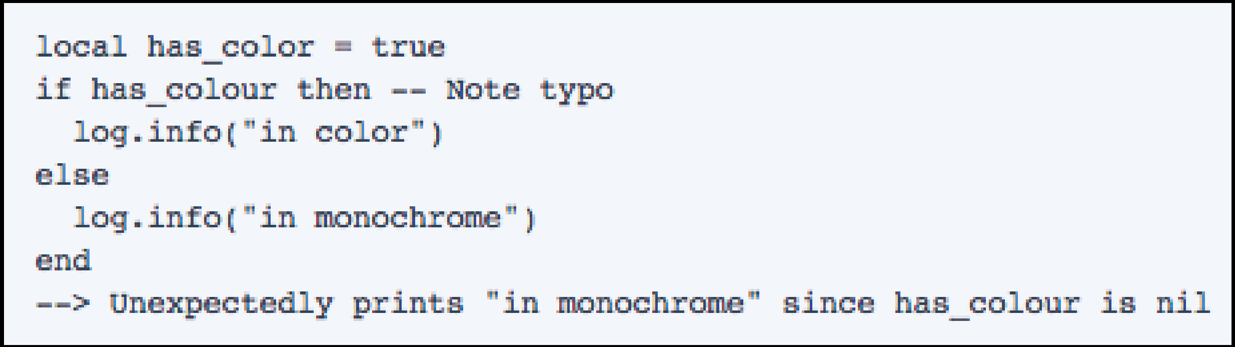
- 声明和赋值



```
> local x = 1, y = 2
stdin:1: unexpected symbol near '='
> local x, y = 1, 2
```

坑

- 变量和函数先定义后使用
- 默认是局部变量
  - use local
- 未定义的变量是全局变量
  - 注意拼写



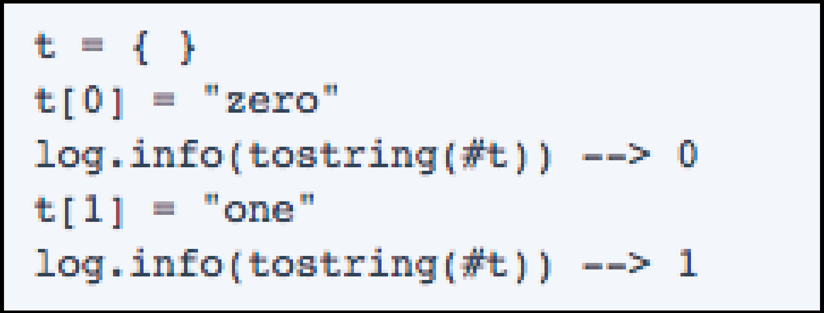
```
local has_color = true
if has_colour then -- Note typo
  log.info("in color")
else
  log.info("in monochrome")
end
--> Unexpectedly prints "in monochrome" since has_colour is nil
```

坑

- 自动类型转换
  - type(10 + “20”)
    - number
  - type(10 .. “20”)
    - string
  - type(“10” + “20”)
    - number 30
  - 10 == “10”
    - false

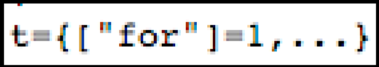
坑

- Table
  - 数组
    - 下标从1开始



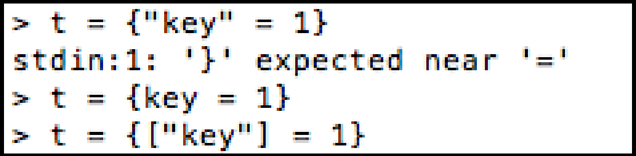
```
t = { }
t[0] = "zero"
log.info(tostring(#t)) --> 0
t[1] = "one"
log.info(tostring(#t)) --> 1
```

  - 字典
    - 关键字作为key的情况

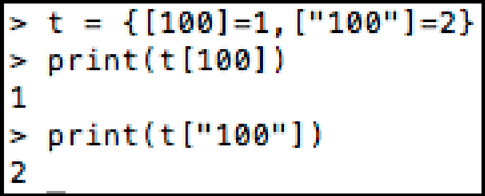


```
t={["for"]=1,...}
```

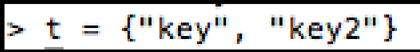
    - 别和数组混淆



```
> t = {"key" = 1}
stdin:1: '}' expected near '='
> t = {key = 1}
> t = {["key"] = 1}
```



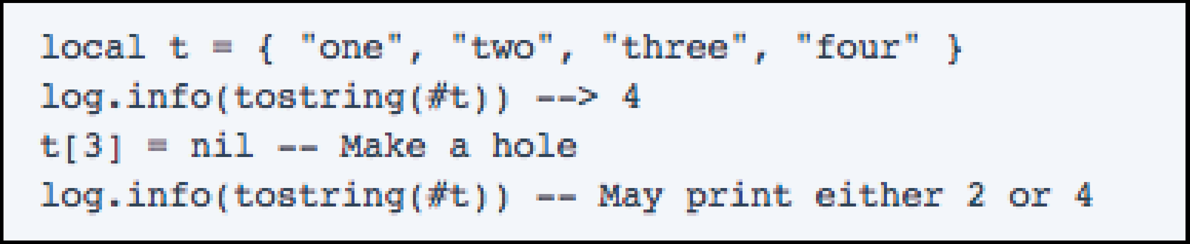
```
> t = {[100]=1,["100"]=2}
> print(t[100])
1
> print(t["100"])
2
```



```
> t = {"key", "key2"}
```

坑

- Holes in arrays



```
local t = { "one", "two", "three", "four" }
log.info(tostring(#t)) --> 4
t[3] = nil -- Make a hole
log.info(tostring(#t)) -- May print either 2 or 4
```

http://www.lua.org/manual/5.1/manual.html#2.5.5

坑

- 函数参数有table
  - 传的是引用
- 函数调用
  - obj:method() eq obj.method(obj)
  - string.find(str, “hello”) eq str:find(“hello”)

参考

- Lua参考手册
  - http://www.lua.org/manual/5.1/manual.html
  - http://www.codingnow.com/2000/download/lua_manual.html
- Lua简明教程
  - http://coolshell.cn/articles/10739.html
- Modules Tutorial
  - http://lua-users.org/wiki/ModulesTutorial
- Patterns Tutorial
  - http://lua-users.org/wiki/PatternsTutorial
- Lua Gotchas
  - http://www.luafaq.org/gotchas.html