Tutorial Lua

Complementos sobre linguagens de programação

Grupo 2

Bárbara Nóbrega Galiza – 105937
João Miguel Dias Andrade – 107969
Tomás de Oliveira Victal - 109018

01

Introdução

O que é Lua?

Origem

Criada no Brasil em 1993

Multiparadigma

Acomoda diversos paradigmas

Simples

Apenas 21 palavras reservadas!

Linguagem Scripting

Utilizada para extender outro software

Embutível

Suportada por diversas linguagens

Multi Plataforma

Interpretador de byte-code feito em C

Paradigmas

Imperativo

Suporta a execução sequencial de comandos, loops condicionais e manipulação de variaveis

Funcional

Lua tem alguns aspetos de programação funcional como funções de primeira classe, closures e funções de ordem superior

Orientado a objetos

Lua não tem classes
mas permite obter os mesmos resultados usando tables e metatables

Multiplataforma

Tal como Java, o código lua é compilado para byte code e é de seguida interpertado pela virtual machine de Lua.

Esta virtual machine é escrita em C o que faz com que seja muito rápida e múito leve.

Velocidade e Simplicidade

Velocidade

O pacote de lua que contém o código fonte e a documentação ocupa apenas 1.4M descompactado.
O interpretador contém apenas 30000 linhas de C.
Estes fatores fazem com que Lua seja considerada uma das linguagem de scripting mais rápidas.

Simplicidade

Sendo que Lua é para ser usada como uma linguagem de script esta tenta ser o mais simples possivel.
Ao mesmo tempo tenta não comprometer funcionalidade. Através do uso de tables e metatables esta permite a implementação de classes e herança

Package Manager

Lua Rocks Package Manager

A standard library de Lua é bastante minimalista para manter o seu tamanho pequeno.

Módulos e Pacotes podem ser instalados usando o Lua Rocks Package Manager.

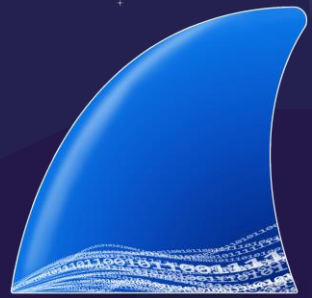
Onde é usada?

O Neovim é um fork do editor de texto Vim que tem suporte embutido à linguagem Lua para scripting e configuração.

Neovim

Os utilizadores podem desenvolver jogos, controlar personagens, definir interações e gerenciar praticamente todos os aspectos do jogo.

Roblox



O wireshark permite aos utilizadores usar lua para automatizar diferentes tarefas e customizar a visualização e análise dos dados.

Wireshark

Que linguagens suportam Lua?







Principais Conclusões

- Caso queiram adicionar funcionalidade á vossa aplicação, ou da a possibilidade de os utilizadores o fazerem Lua é uma das melhores escolhas
- Linguagem de script mais utilizada em video-jogos
- Rápida e Embutível
- Simples e fácil de aprender


02
Sintaxe Básica

Tipos

Existem 8 tipos basicos em Lua:

- *nil*
- *boolean*
- *number*
- *string*
- *function*
- *userdata*
- *thread*
- *table*

Prints, comentários e strings

```lua
print("Hello lua!")
io.write("Lua is easy\n")

--[[
    Comentário multilinha
]]

a = 5
b = 10
print("a: " .. a .. " b: " .. b .. " a + b: " .. a+b) -- string concatenation

str = "lua"
print("Size of str: " .. #str)  -- lenght operator
```

```
Hello lua!
Lua is easy
a: 5 b: 10 a + b: 15
Size of str: 3
```

If-else e *nil*

```lua
-- If-else statements
exp = 10
degree = true
skills = 40
if degree and exp >= 100 then
    job = true
elseif degree and skills > 30 then
    job = true
else
    job = false
end
print("Hired: " .. tostring(job)) -- tostring() converte para string

-- Nil
var = nil

print(var)

if var then
    print("var is not nil")
else
    var = 10
end

print(var)
```

```
Hired: true
nil
10
```

Loops

```lua
i = 1
while (i <= 10) do
    io.write(i)
    i = i + 1 -- i++ e i += 1 não existem em Lua
    -- continue não existe em Lua
    if i == 8 then break end
end

print()

-- Repeat-until corre o código pelo menos uma vez (do-while)
repeat
    io.write("Adivinhe um número entre 0 e 10: ")
    guess = io.read() -- Lê do terminal
until tonumber(guess) == 5 -- tonumber() converte para número

print("Acertou!")

-- For estilo "C": inicialização, condição, incremento
for i = 1, 10, 1 do
    io.write(i .. "-")
end

print()
```

```
1234567
Adivinhe um número entre 0 e 10: 3
Adivinhe um número entre 0 e 10: 7
Adivinhe um número entre 0 e 10: 5
Acertou!
1-2-3-4-5-6-7-8-9-10-
```

Tabelas (I)

```lua
-- Criar uma tabela
aTable = {1, 2, 3, 4, 5}

-- Acessar valor pelo índice (OBS: índices começam em 1)
io.write("First Item : ", aTable[1], "\n")

-- Número de itens na tabela
io.write("Number of Items : ", #aTable, "\n")

-- Inserir item na tabela: (tabela, índice, item)
table.insert(aTable, 1, 0)

-- Combina uma tabela como uma string e separa com o separador fornecido
print(table.concat(aTable, ", "))

-- Remover item no índice: (tabela, índice)
table.remove(aTable, 1)
print(table.concat(aTable, ", ")) -- OBS: Só serve para tabelas com índices numéricos sequenciais (sem nils)

-- Ordenar itens em ordem decrescente
table.sort(aTable, function(a,b) return a>b end)
print(table.concat(aTable, ", "))
```

```
First Item : 1
Number of Items : 5
0, 1, 2, 3, 4, 5
1, 2, 3, 4, 5
5, 4, 3, 2, 1
```

Tabelas (II)

```lua
-- Criar uma tabela multidimensional
aMultiTable = {}

for i = 0, 9 do -- índices podem ser o que nós definirmos (índice = key)
  aMultiTable[i] = {}
  for j = 0, 9 do
    aMultiTable[i][j] = tostring(i) .. "," .. tostring(j)
  end
end

print(aMultiTable[1][5]) -- 1,5

-- For estilo "foreach" (itera sobre uma tabela)
dias = {"Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado", "Domingo"}

for i, dia in pairs(dias) do
    print(i .. " - " .. dia)
end

-- Array associativo (aka Dicionário, Map, etc)
local teams = {
    ["teamA"] = 12,
    ["teamB"] = 15
}

print(teams["teamA"]) -- 12

for key,value in pairs(teams) do
  print(key .. " : " .. value)
end
```

```
1,5
1 - Segunda
2 - Terça
3 - Quarta
4 - Quinta
5 - Sexta
6 - Sábado
7 - Domingo
12
teamA : 12
teamB : 15
```

```lua
function Set (list)
  local set = {}
  for _, l in ipairs(list) do set[l] = true end
  return set
end

reserved = Set{"while", "end", "function", "local", }
```

Exercício: Tabelas

No repositório: /exercises/table.lua

Funções

```lua
function getSum(num1, num2)
    return num1 + num2
end

print(string.format("5 + 2 = %d", getSum(5,2)))

-- A variável "i" só existe dentro da função
function func()
    local funcVar = 5
end

print(funcVar)
```

```
5 + 2 = 7
nil
```

```lua
-- Variadic Function: recebe um número variável de argumentos
function getSumMore(...)
    local sum = 0

    for k, v in pairs{...} do
        sum = sum + v
    end
    return sum
end

io.write("Sum : ", getSumMore(1,2,3,4,5,6), "\n")

-- As funções podem ser tratadas como variáveis
-- Funções anónimas (ou lambda)
doubleIt = function(x) return x * 2 end
print("Dobro de 4: " .. doubleIt(4))
```

```
Sum : 21
Dobro de 4: 8
```

Funções

```lua
function split(str)
    local words = {} -- Table vazia
    local i = 1

    for word in string.gmatch(str, "[^%s]+") do  -- split a partir de espaços. gmatch retorna um iterador
        words[i] = word
        i = i + 1
    end

    -- Funções podem retornar mais de um valor
    return words, i
end

-- Recebe múltiplos valores
words, count = split("Complementos sobre linguagens de programação")

for j = 1, count do
    print(string.format("%d : %s", j, words[j]))
end
```

```
1 : Complementos
2 : sobre
3 : linguagens
4 : de
5 : programação
6 : nil
```

Coroutines

```lua
-- Coroutines são como threads, com a diferença de que não podem ser executadas em paralelo
-- Uma coroutine possui os estados de running, suspended, dead ou normal

-- Usar "create" para criar uma que execute alguma ação
co = coroutine.create(function()
    for i = 1, 10, 1 do
    print(i)
    print(coroutine.status(co))
    if i == 5 then coroutine.yield() end
    end
end)

-- Status inicial: suspended
print(coroutine.status(co))

-- Executar: resume(). Muda o status para running
coroutine.resume(co)

-- Yeld faz ela suspender a execução e mudar o status para suspended
print(coroutine.status(co))

co2 = coroutine.create(function()
    for i = 101, 110, 1 do
        print(i)
    end
end)

coroutine.resume(co2)
coroutine.resume(co)

-- Após a execução, ela possui o status de dead
print(coroutine.status(co))
```

```
suspended
1
running
2
running
3
running
4
running
5
running
suspended
101
102
103
104
105
106
107
108
109
110
6
running
7
running
8
running
9
running
10
running
dead
```

File I/O

```lua
-- Modos de acesso:
-- r: Somente leitura (padrão)
-- w: Sobrescrever ou criar um novo ficheiro
-- a: Append ou criar um novo ficheiro
-- r+: Ler e escrever em um ficheiro existente
-- w+: Sobrescrever, ler, ou criar um ficheiro
-- a+: Append, ler, ou criar um ficheiro

-- Criar um novo ficheiro para leitura e escrita
file = io.open("file.txt", "w+")

-- Escrever texto no ficheiro
file:write("Fly me to the moon\n")
file:write("And let me play along the stars\n")

-- Voltar para o início do ficheiro
file:seek("set", 0)

-- Ler do ficheiro
print(file:read("a")) -- "a": ler o ficheiro todo

-- Fechar o ficheiro
file:close()

file = io.open("file.txt", "a+")

file:write("Let me see what spring is like on Jupiter and Mars\n")

file:seek("set", 0)

print(file:read("*a"))

file:close()
```

```
Fly me to the moon
And let me play along the stars

Fly me to the moon
And let me play along the stars
Let me see what spring is like on Jupiter and Mars
```

Módulos

```lua
local convert = {} -- cria o módulo
function convert.ftToCm(feet)
  return feet * 30.48
end
return convert
```

```lua
-- Require: obter acesso às funções no módulo
moduleConvert = require("convert")

-- Executar a função no módulo
print(string.format("%.3f cm", moduleConvert.ftToCm(12)))
```

```
:~/EI/CSLP/Lua_tutorial/examples (main)$ lua module.lua
365.760 cm
```

Exercício: Módulos + funções

No repositório: /exercises/module.lua e /exercises/distance.lua

POO: tabelas, metatabelas e funções

```lua
-- Classes podem ser implementadas a partir de tables, metatables e funções

Enemy = {}

function Enemy:new(lives)
    setmetatable(self, Enemy)  -- Sintaxe: setmetatable(table, metatable) -- Cria o objeto
    self.lives = lives or 3 -- valor por defeito
    self.full_hp = 100
    self.hp = self.full_hp
    return self
end

function Enemy:receive_damage(damage) -- ou Enemy.receive_damage(self, damage)
    self.hp = self.hp - damage
    if self.hp <= 0 then
        self.lives = self.lives - 1
        self.hp = self.full_hp
    end
end

function Enemy:tostring()
    return "[Enemy] Lives: " .. self.lives .. ", HP: " .. self.hp
end
```

POO (II)

```lua
e1 = Enemy:new(nil) -- para usar valores por defeito, usar nil

print(e1.lives)  -- 3
print(e1.hp)  -- 100

print(e1:tostring())

e1:receive_damage(50)

print(e1:tostring()) -- lives 3, hp 50

e1:receive_damage(50)

print(e1:tostring()) -- lives 2, hp 100
```

```
3
100
[Enemy] Lives: 3, HP: 100
[Enemy] Lives: 3, HP: 50
[Enemy] Lives: 2, HP: 100
```

POO (III): Herança

```lua
-- Herança

Zombie = Enemy:new()

function Zombie:new(lives, speed)
    setmetatable(self, Zombie)
    self.lives = lives or 2
    self.full_hp = 150
    self.hp = self.full_hp
    self.speed = speed or 20
    return self
end

-- Override
function Zombie:tostring()
    return "[Zombie] Lives: " .. self.lives .. ", HP: " .. self.hp .. ", Speed: " .. self.speed
end

z1 = Zombie:new(2, 20)

print(z1:tostring()) -- lives 2, hp 100, speed 20

z1:receive_damage(50)

print(z1:tostring()) -- lives 2, hp 50, speed 20
```

```
[Zombie] Lives: 2, HP: 150, Speed: 20
[Zombie] Lives: 2, HP: 100, Speed: 20
```


03
Integração com C

Obrigado!

Questões?

Referências: consultar README.md do repositório no github