Programando em Lua

Judismar Arpini Junior
Linguagens de Programação

Agenda

- Introdução
- Valores em Lua
- Condicional if – then – else
- Loops
- Funções
- Tabelas
- Metatabelas
- Lua em Python
- Conclusão

Introdução

- LP multi-paradigma, como Python

- Será ensinado o básico de programação estruturada: loops, condicionais etc

- Também será ensinado com detalhes dois conceitos centrais de Lua: tabelas e metatabelas

Valores em Lua

- Diferente de Python, nem todo elemento atribuído à uma variável é objeto!!

- Ex: x = 23

  Um espaço é alocado para a abstração x, armazenando o valor 23

- Outros exemplos:

  x = 5

  l = false

Valores em Lua

- Apenas valores do tipo table, function, thread e userdata são objetos

- As variáveis não contem os valores em si, mas sim referências à esses valores

- Ex: function f()

```
        return {3, 5}
    end
```

Retorno de referência para tabela!

Condicional if – then – else

```
if x > 10 then
    y = 0
else
    y = y + 1
end
```

- Qualquer valor diferente de false e nil são tomados como verdadeiro
- Atribuição não é expressão (não tem retorno)!

Loops

- O while:

```
i = 0
while i < 10 do
    i = i + 1
end
```

Loops

- O repeat:

```
i = 0
repeat
      i = i + 1
until i > 10
```

Loops

- Os possíveis for:

```
for i = 1, 10 do
    print(i)
end
```

```
for i = 1, 10, 2 do
    print(i)
end
```

Loops

- O for em tabelas:

```
t = {x = 1, y = -3, delta = 900}
for chave, valor in pairs(t) do
    print(chave, 'é chave de ', valor)
end
```

Funções

```
function f(x)
    print(x)
    return x
end
```

```
function f(x, y)
    return x, y
end
```

- Tipo primário! Pode ser passada como parâmetro, retornada, atribuir a variáveis

Tabelas

- Estrutura de dados mais importante de Lua
- Pode ser usada como vetor, como registro (ou struct), como um conjunto de funções
- Pode conter valores de tipos misturados

Tabelas

- Usando table como vetor:

```
vet = {2, 5, 10, 11}
print(x[1])
print(x[4])
table.insert(vet, -32)
table.remove(vet, 2)
```

Índices de 1 a n

Tabelas

- Usando table como registro (estrutura):

```
t = {x = 3, y = 5}
t.z = 11
print(t.x) -> print(t[“x”])
```

Metatabelas

- São tabelas usadas para facilitar a manipulação das outras tabelas, com as quais são associadas
- Exemplo para facilitar entendimento: metadados
- O Dropbox tem uma pasta .metadata, que possui dados que não são do usuário do software, mas sim para manipular os dados dele

Metatabelas

- O comportamento de uma tabela é modificado quando é implementado os metamétodos da metatabela associada
- Os metamétodos devem ser implementados e atribuídos a chaves especiais da metatabela
- Exemplo:

```
__add = function(t1, t2)
        return t1.valor + t2.valor
end
```

Metatabelas

- setmetatable(tabela, metatabela)

- getmetatable(tabela)

Metatabelas

```
v = {3, 5, 1} --tabela-vetor de índices 1, 2 e 3
meta_tabela = {
        __newindex = function(t, key, value) print('Erro!'); os.exit() end
        ,
        __tostring = function(t) --converte em string
                string = "["
                for i = 1, #t do
                        string = string .. t[i]
                        if i < #t then
                                string = string .. ', '
                        end
                end
                string = string .. ']'
                return string
        end
        ,
        __add = function(t, t2)
                if #t ~= #t2 then
                        print('Erro!')
                        return nil
                end
                result = setmetatable({}, meta_tabela)
                for i = 1, #t do
                        table.insert(result, t[i] + t2[i])
                end
                return result
        end
```

Metatabelas

```
,
__sub = function(t, t2)
        if #t ~= #t2 then
                print('Erro!')
                return nil
        end
        result = setmetatable({}, meta_tabela)
        for i = 1, #t do
                table.insert(result, t[i] - t2[i])
        end
        return result
end
,
__mul = function(c, t)
        result = setmetatable({}, meta_tabela)
        for i = 1, #t do
                table.insert(result, c*t[i])
        end
        return result
end
,
```

Metatabelas

```
}

function main()
        setmetatable(v, meta_tabela)

        table.insert(v, 21)

        print(v)
        print(v + {1, -1, 0 , 3})
        print(v - {3, 5, 2, 21})
        print(3*v)
        print(2*v + {-2, 1, 12, 10})
end

main()
```

Metatabelas

```
Aluno = {}
Aluno.__index = Aluno --iniciando a classe: será uma metatabela
Aluno.__newindex = function(a, key, value) print('Erro!'); os.exit() end

Aluno.new = function(nome, nota)
        return setmetatable({nome = nome or "", nota = nota or 0}, Aluno)
end

Aluno.calculaMedia = function(a, prova1, prova2, trabalho)
        a.nota = (prova1 + prova2 + trabalho)/3
end

Aluno.aprovado = function(a)
        return a.nota > 7
end

Aluno.__tostring = function(a)
        return "Nome: " .. a.nome .. "\nNota: " .. a.nota
end

function main()
        a = Aluno.new('Florisvaldo Firmino');
        print(a)

        a:calculaMedia(10, 8.5, 9)

        print(a)
        print(a:aprovado())
end

main()
```

Lua em Python

- Lua pode ser embutida em várias linguagens de programação, inclusive Python

- Para embutir o código de Lua em Python, usaremos LUPA. Será enviado junto com a apresentação e os códigos um script que faz toda a instalação

- Executar:

  sudo sh lupa_install.sh

  Após o término, Python poderá incluir código Lua

  Obs: instale python-dev antes pra evitar problemas

Lua em Python

- Para executar Lua, o código deve ser escrito como String

```
from lupa import LuaRuntime
lua = LuaRuntime()
x = 3; x = str(x)
y = 5; y = str(y)
codigo_lua = '''
--codigo de lua em string
function soma(a, b)
        return a + b
end
'''
#adicionar a chamada da funcao na string
codigo_lua += 'print(soma(' + x + ', '+ y + '))'
lua.execute(codigo_lua)
```

Lua em Python

- Também será necessário o uso do método 'eval'
- O eval (evaluate) avalia uma expressão de Lua e manda o retorno para Python
- Usar o eval para passar as tabelas de Lua para Python, inclusive objetos de classes

Lua em Python

- Exemplos de uso de eval:

  ```
  cinco = lua.eval('2 + 3')
  vet = lua.eval('{3, 5, 1}')
  vet[1]; vet[2]; vet[3] #acessando como em Lua!

  ponto = lua.eval('{x = 3.1, y = 5.7}')
  ponto.x; ponto.y
  ```

- Lembrete: expressões sempre tem retorno!

Lua em Python

- Um último exemplo:

```
lua.execute('a = Aluno.new(“Ana Paula”, 9.5)')
ana = lua.eval('a')
ana.nome #Ana Paula
ana.nota #9.5
```

- Lembrete: atribuição não é expressão!

Lua em Python

- Recomendação: escreva o código em um arquivo .lua e depois passe para Python (use a palavra reservada 'require' de Lua para importar scripts com definições)

- Para executar o código Lua no terminal:

  lua codigo.lua

- Ao passar para Python, não é necessário executar tudo em uma só String. Execute em substrings para facilitar (como no exemplo anterior)

Lua em Python

- Passando dados de Python para Lua é um pouco mais complicado
- Facilitador: usar função de Callback!
- Lua recebe um ponteiro para uma função de Python que inicia o que quiser ser iniciado

Lua em Python

```
from lupa import LuaRuntime

def preparaFuncaoDeLuaComCallback(ambienteLua, arquivo):
        with open(arquivo) as entrada:
                luaCode = entrada.read()
                return ambienteLua.eval(luaCode)

class Astronauta:
        nome = ""
        pronto = True
        destino = ""
        def __init__(self, nome, destino):
                self.nome = nome
                self.destino = destino

def paraUniverso(nome, destino):
        return Astronauta(nome, destino)

ambienteLua = LuaRuntime()
funcaoLua = preparaFuncaoDeLuaComCallback
(ambienteLua, "codigo_lua.lua")
funcaoLua(paraUniverso)
```

Lua em Python

```
function(callback)
  astronauta = callback("Varejão", "Lua");
  astronauta2 = callback("Judis", "Saturno");
  print(astronauta.nome .. ": Olá, "..astronauta.destino.."!");
  print(astronauta2.nome .. ": Olá, "..astronauta2.destino.."!");
end
```

- Pode-se criar uma pequena lista em Python contendo todos os dados que vai passar para “o outro lado” e enviar pela função de Callback!!

Conclusão

- Refazer o primeiro trabalho em Lua, reaproveitando o código de Python em que é feita a entrada de dados
- O processamento da entrada será escrito em Lua embutido em Python, a partir dos dados lidos.
- Feito o processamento, devolver os dados para Python, fazendo então a saída de dados

Conclusão

- Mandem dúvidas para solid.judis@gmail.com

- Documentação de Lupa:

  http://pypi.python.org/pypi/lupa/0.20

- Há mais alguns metamétodos que não foram abordados, como __metatable. É facil encontrar no Google todos eles
- Obrigado!