Linguagens de Programação

Lua 5.1

Prof. Bruno Silvestre
brunoos@inf.ufg.br

O que é Lua

- Uma linguagem dinâmica, com alguma semelhança com Perl, Python, Ruby, JavaScript...
- Linguagem de descrição de dados (anterior a XML)
- Linguagem de script, com facilidade de comunicação com outras linguagens e desenvolvimento multi-linguagem

De Onde Vem

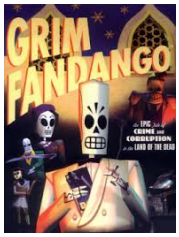
- Desenvolvida na PUC-Rio, desde 1993, por Roberto Ierusalimschy, Luiz Henrique de Figueireido e Waldemar Celes
- Início modesto, para uso interno, expansão lenta e gradual nos últimos seguintes
- Única linguagem criada em um país em desenvolvimento a ganhar projeção mundial

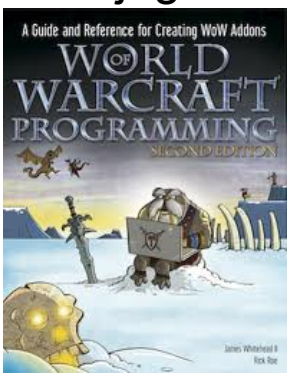
Por que Usar Lua

- Portabilidade
- Simplicidade
- “Acoplabilidade” (scripting)
- Eficiência
- Pequeno tamanho
  - Interpretador → 200Kb
  - Core + Bibliotecas padrão → 329Kb
  - Java → ??? Mb (zip)

Usos de Lua

- Usada em todo o tipo de aplicação
- Mas, encontrou um nicho especial em jogos





middleware Ginga
TV Interativa se faz com Ginga!

WIRESHARK

NetBSD The NetBSD Project
"Of course it runs NetBSD"

Informações

- Site do projeto
  - http://www.lua.org
- Manual de Lua 5.1
  - Consulta rápida
  - http://www.lua.org/manual/5.1/
- PIL – Programming in Lua (1ª Edição)
  - Grátis (em inglês)
  - Cobre Lua 5.0, mas grande parte funciona com Lua 5.1
  - http://www.lua.org/pil/

Informações

- Programando em Lua (3ª Edição)
  - Cobre Lua 5.3

Software

- Usaremos a versão Lua 5.1
- Lua 5.0 vs Lua 5.1 vs Lua 5.2 vs Lua 5.3
  - No primeiro momento, as versões 5.x podem parecer semelhantes
  - Mas existem diferenças consideráveis

Software

- Distribuições Linux e BSD veem com pacotes para Lua 5.1
  - No Debian/Ubuntu
    - lua5.1 → interpretador
    - liblua5.1-dev → bibliotecas para desenvolvimento com C
- Binários podem ser baixados
  - http://luabinaries.sourceforge.net/
  - “bin” → interpretador
  - “dll” ou “lib” → bibliotecas para desenvolvimento com C

Primeiros Passos

- Famoso “Hello World”
- Criar um arquivo *hello.lua*

```
print("Hello, world!")
```

- No terminal

```
lua  hello.lua
```

Primeiros Passos

- Modo interativo
  - Executar apenas o interpretador (lua) no terminal
  - Ele permite que você execute comandos rápidos

Trecho de Código

```
function fatorial(n)
  if n == 0 then
    return 1
  else
    return n * fatorial(n-1)
  end
end

print("enter a number: ")
v = io.read("*n")
print(fatorial(v))
```

Sintaxe

- Linguagem *case-sensitive*
- Identificadores parecidos com C
  - Nome de variáveis e funções
  - Letras, números e "_"

Sintaxe

- Poucas palavras reservadas
- Influência de Pascal

```
and       break     do
else      elseif    end
false     for       function
if        in        local
nil       not       or
repeat    return    then
true      until     while
```

Comentários

- Comentário de linha (como o "//" de C)

```
-- Exibe o estado interno
print(status)  -- estado do programa
```

- Comentários longos (como "/* */" de C)

```
--[[
  Exibe o estado interno.
  Pode mostrar vários valores.
]]
print(status)
```

Entrada e Saída

- A função `print()` exibe o conteúdo na tela seguido de um "\n"
- Ela pode receber mais de um parâmetros, os valores são exibidos separados por tabulação ("\t")

```
print(1, "Hello", true)

print("Valor da soma", 3+4)
```

Entrada e Saída

- A função `io.read()` lê da entrada padrão
- O parâmetro da função vai indicar o que deve ser lido, por exemplo:
  - “*l” : lê uma linha (retorna uma string)
  - “*n” : lê um número (retorna um número)
- Há outras opções, ver o manual se necessário

Entrada e Saída

- A função `io.read()` lê da entrada padrão
- O parâmetro da função vai indicar o que deve ser lido, por exemplo:
  - “*l” : lê uma linha (retorna uma string)

```
str = io.read("*l")

print(str)
```

Conversões

- `tostring( v )` : converte um valor *v* de Lua (número, booleano, tabela, etc.) em string
- `tonumber( s )` : converte uma string *s* para número

```
l = io.read("*l")

n = tonumber(l)

print(n+1) -- sucessor
```

Variáveis

Variáveis Globais

- Válidas no programa como um todo
- Não é necessário declará-las
  - Criadas implicitamente
- Qual a saída do programa?

```
a = 10
print(a)

print(b)
```

Variáveis Globais

- Válidas no programa como um todo
- Não é necessário declará-las
  - Criadas implicitamente
- Variáveis não inicializada não geram erro

```
a = 10
print(a)

print(b)
```

Variáveis Locais

- Variáveis locais são visíveis apenas no escopo que é definida
  - Escopo: arquivo, função, bloco (if, while, etc.)
- Dê preferência a variáveis locais em vez de globais

Variáveis Locais

- Variáveis são definidas como locais quando a palavra-chave “local” é usada na sua declaração ou criação

```
i = 2              -- variável global

local j            -- variável local
j = 2 + i          -- muda o valor

local t = {}       -- variável local
local b = true     -- variável local
```

Variáveis Locais

```
$ lua prog.lua
```

prog.lua

```
a = 10
local b = 20

print("1",a,b,c,d)

-- Executa aux.lua
dofile("aux.lua")

print("3",a,b,c,d)
```

aux.lua

```
local d
c = 30
d = 40

print("2",a,b,c,d)
```

Variáveis Locais

x é visível no arquivo

z é visível no "if"

i é visível no "for"

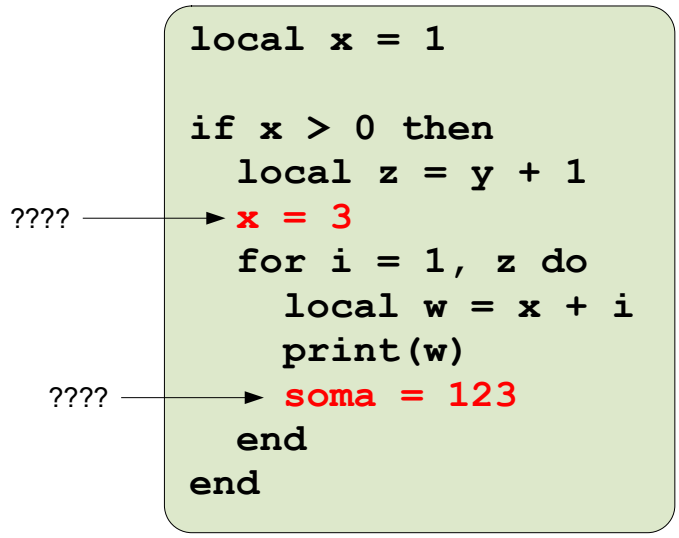
w é visível no "for"

```
local x = 1

if x > 0 then
  local z = y + 1

  for i = 1, z do
    local w = x + i
      print(w)
  end
end
```

Variáveis Locais



Tipos e Valores

Tipos e Valores

- Tipos de Lua
  - nil
  - boolean
  - number
  - string
  - table
  - function
  - userdata
  - thread
- Todos esses valores podem ser armazenados em variáveis, parâmetros ou retornados em funções

Tipos e Valores

- Lua é uma linguagem dinâmica
- Não é necessário definir o tipo para a variável
  - Os valores têm tipo, não as variáveis

```
a = "Hello"
print(a)      --> "Hello"

a = 10.5
print(a)      --> 10.5

a = true
print(a)      --> true
```

Tipos e Valores

- Por ser dinâmica, é possível descobrir o tipo do valor ou variável
- Lua possui a função *type* que retorna uma string com o nome do tipo

```
print(type("Hello"))      --> ???

print(type(10.5))         --> ???

print(type(nil))          --> ???

print(type(type))         --> ???

print(type(type(print))) --> ???
```


Tipos e Valores

- Podemos perguntar qual é o tipo do valor que uma variável está armazenando

```
a = 10.5
print(type(a))   --> ?

a = true
print(type(a))   --> ?

a = print
a(type(a))       --> ?

a = {}
print(type(a))   --> ?
```

nil

Tipos e Valores: nil

- **nil** representa ausência de valor
  - Variáveis não inicializadas possuem *nil*
  - Funciona como *null* em C ou Java
- Até a versão 4 de Lua, era a única forma de representar ***falso***
  - E ainda mantém esse comportamento
- Pode-se atribuir *nil* a uma variável para remover seu valor

```
soma = nil
```

boolean

Tipos e Valores: boolean

- Representado por ***true*** e ***false***
- Introduzido na versão 5.0 de Lua
  - ***nil*** **era usado tanto para falso como nulo**
  - Isso causava ambiguidade em alguns casos

Lua 4

```
-- 'debug' é opcional
debug = nil

-- 'debug' desativado
debug = nil
```

Lua 5

```
-- 'debug' é opcional
debug = nil

-- 'debug' desativado
debug = false
```

Tipos e Valores: boolean

- Em Lua 5.x, **nil** e **false** significam *falso*
- Qualquer outro valor significa *verdadeiro*
  - Semântica parecida com da linguagem C?

```
if 5.5   then print("verdadeiro") end
if 0     then print("verdadeiro") end
if print then print("verdadeiro") end

if false then print("não exibe") end

if nil   then print("não exibe") end
```

Operadores Booleanos

- Operadores
  - not → negação
  - and → E lógico
  - or → OU lógico
- Exemplos

```
print(count > 0  and  count < 20)

access = login == "admin"  or  login == "root"

access = not(login == "admin"  or  login == "root")
```

number

Tipos e Valores: number

- O tipo *number* representa números
- Sintaxe

```
a = 3
b = 5.67

c = 0x12      -- 0x12 hex   = 18

i = 2e3       --  2 x 10^3  = 2000
j = 1E-3      --  1 x 10^-3 = 0.001

m = 2.0E4     -- 2.0 x 10^4  = 20000.0
n = 3.5e-2    -- 3.5 x 10^-2 = 0.035
```

Tipos e Valores: number

- O tipo número em Lua 5.1 é implementado geralmente com o tipo *double* da linguagem C
  - Representa inteiros até 2^{53} sem perda de precisão
  - 1 / 7 → 0.142857 (não é divisão inteira)
  - 20 - 12.7 - 7.3 = 8.8817841970013e-16

Operadores

- Operadores aritméticos e relacionais
- Além disso temos as funções do módulo “math”

+	-	*	/
%			
^			
-			

>	<
>=	<=
==	~=

Módulo *math*

- O módulo *math* possui um conjunto de funções matemáticas

  http://www.lua.org/manual/5.1/manual.html#5.6
- Alguma função não conhecida?

string

Tipos e Valores: string

- Em Lua 5.1, string é uma sequência de caracteres
  - Cada caractere é de 8bits
  - Sem suporte à UNICODE ou UTF-8 nativamente
- Imutável
  - Concatenação ou substring criam novas strings
- Coletada pela GC
- Programas Lua que manipulam strings grandes são bem comuns
  - Não tenham receio de usar string

Tipos e Valores: string

- Delimitadores: " ou '

  "Uma String" ou 'Uma String'
- String longas:
  - [[ ]]
  - [=[ ]=]
  - [==[ ]==]
  - ...

```
"a = b[1]"
'a = b[1]'

[[É uma string]]
[=[É uma string]=]

[[Este 'A' é um "A"]]
```

Tipos e Valores: string

- String longas

```
local txt = [[
  Um vetor pode ser acessado
  utilizando o operador [].
  Como exemplo, m[v[2]].
]]

print(txt)
```

ERRO!

```
local txt = [==[
  Um vetor pode ser acessado
  utilizando o operador [].
  Como exemplo, m[v[2]].
]==]

print(txt)
```

String

- Caracteres de escape

  \a \b \f \n \r \t \\ \" \'

- Operador de Concatenação: ..

```
str = "Hello" .. '   ' .. [[world]]
```

- Operador de tamanho da string: #

```
#"Hello"
#str
#[[bla bla bla]]
#''
```

String

- Comparação: > >= < <= == ~=

```
if "uva" < "banana"  then  print("menor") end
```

- Não tenham receio de usar string

Estilo linguagem C

```
local access

if access == 1 then
  -- codigo aqui
end
if access == 2 then
  -- codigo aqui
end
if access == 3 then
  -- codigo aqui
end
```

Estilo linguagem Lua

```
local access

if access == "admin" then
  -- codigo aqui
end
if access == "user" then
  -- codigo aqui
end
if access == "guest" then
  -- codigo aqui
end
```

Módulo *string*

FUNÇÃO	DESCRIÇÃO
string.byte	
string.char	
string.find	
string.format	
string.len	
string.lower	
string.rep	
string.reverse	
string.sub	
string.upper	

Módulo *string*

- Usando a função print() e string.format() podemos fazer o mesmo que o printf() de C

```
print( string.format("%s : %d", str, num) )
```

Atividade

3) Alguns editores de texto possuem a possibilidade de formatar string como “Palavra Iniciando Com Maiúsculo” (PICM)

- Dada uma string com o nome completo de uma pessoa, faça um pequeno programa que imprima o nome da pessoa formatado como PICM
- Assuma nomes com 3 palavras

  nome = “pAuLo AnToNiO OliVEirA”

  nome = “ANA dos sANtos”